UNIVERSIDADE TUIUTI DO PARANÁ

ANDRÉ VINICIUS AGIO
GUILHERME ALVES DOS SANTOS
LUCAS VENANCIO THIELE
OCTÁVIO KONZEN

ESTUDO DIRIGIDO DE DESENVOLVIMENTO MOBILE

CURITIBA
2025

ANDRÉ VINICIUS AGIO
GUILHERME ALVES DOS SANTOS
LUCAS VENANCIO THIELE
OCTÁVIO KONZEN

ESTUDO DIRIGIDO DE DESENVOLVIMENTO MOBILE

Trabalho apresentado ao curso de Análise e Desenvolvimento de Sistemas, da Universidade Tuiuti do Paraná, como requisito avaliativo do 2º bimestre da disciplina de Desenvolvimento Mobile.

Professor: Chauã C.Q.B da Silva.

CURITIBA
2025

RESUMO

A crescente dependência de aplicativos móveis para tarefas cotidianas, desde transações bancárias a interações sociais, elevou drasticamente a superfície de ataque para agentes maliciosos. Este trabalho analisa os principais desafios de segurança inerentes ao desenvolvimento de aplicações para as plataformas Android e iOS. O objetivo é fornecer uma análise aprofundada das vulnerabilidades mais críticas, como armazenamento inseguro de dados, comunicação desprotegida e falhas de autenticação, com base nas diretrizes do projeto OWASP Mobile Top 10. Para cada vulnerabilidade, são discutidos seus impactos potenciais e os métodos de exploração. Em contrapartida, o estudo apresenta um conjunto de boas práticas e soluções técnicas preventivas, abordando criptografia de dados em trânsito e em repouso, implementação de autenticação multifator, gerenciamento seguro de sessões e o princípio do menor privilégio na gestão de permissões. A pesquisa examina casos reais de falhas de segurança em aplicativos de grande visibilidade, detalhando as consequências e as lições aprendidas. Adicionalmente, são exploradas ferramentas e metodologias para testes de segurança, como SAST, DAST e testes de invasão (pentest). Como resultado prático, é proposto um checklist de segurança para desenvolvedores, visando integrar a segurança em todo o ciclo de vida do desenvolvimento de software (SDLC). Conclui-se que a adoção de uma mentalidade de Security by Design é fundamental para mitigar riscos e proteger tanto os usuários quanto a reputação das organizações.

Palavras-chave: Segurança Mobile. Desenvolvimento Seguro. Vulnerabilidades. Android. iOS. OWASP.

SUMÁRIO

1 INTRODUÇÃO..... 4
2 PRINCIPAIS VULNERABILIDADES EM APLICATIVOS MÓVEIS..... 5
2.1 M1: USO INADEQUADO DA PLATAFORMA..... 5
2.2 M2: ARMAZENAMENTO INSEGURO DE DADOS..... 6
2.3 M3: COMUNICAÇÃO INSEGURA..... 7
2.4 M4: AUTENTICAÇÃO INSEGURA..... 7
2.5 M5: CRIPTOGRAFIA INSUFICIENTE..... 8
2.6 M6: AUTORIZAÇÃO INSEGURA..... 9
2.7 M9: ENGENHARIA REVERSA..... 10
3 BOAS PRÁTICAS DE SEGURANÇA NO DESENVOLVIMENTO MOBILE..... 10
3.1 CRIPTOGRAFIA DE DADOS: UMA DEFESA EM CAMADAS..... 10
3.2 AUTENTICAÇÃO ROBUSTA E CONTROLE DE SESSÕES..... 12
3.3 GESTÃO DE PERMISSÕES E O RESPEITO À PRIVACIDADE DO USUÁRIO..... 13
4 CASOS REAIS DE FALHAS DE SEGURANÇA EM APLICATIVOS POPULARES..... 13
4.1 BRITISH AIRWAYS: VAZAMENTO DE DADOS VIA SCRIPT MALICIOSO..... 13
4.2 GRINDR: EXPOSIÇÃO DA LOCALIZAÇÃO DE USUÁRIOS POR API INSEGURA..... 14
4.3 TIKTOK: VULNERABILIDADES NA CADEIA DE CDN..... 15
5 FERRAMENTAS E TÉCNICAS DE ANÁLISE DE SEGURANÇA MOBILE..... 15
5.1 ANÁLISE ESTÁTICA (SAST) E ANÁLISE DE COMPOSIÇÃO DE SOFTWARE (SCA). 15
5.2 ANÁLISE DINÂMICA (DAST) E TESTES INTERATIVOS (IAST)..... 16
5.3 TESTES DE INVASÃO (PENTEST) EM AMBIENTES ANDROID/IOS..... 16
6 CHECKLIST DE SEGURANÇA PARA O CICLO DE DESENVOLVIMENTO SEGURO..... 17
7 CONCLUSÃO..... 19
REFERÊNCIAS..... 20

1 INTRODUÇÃO

A era da hiperconectividade é impulsionada por smartphones, que gerenciam um volume colossal de dados sensíveis e se tornaram alvos centrais para o cibercrime. A sofisticação dos ataques expõe aplicativos a vulnerabilidades com consequências devastadoras, incluindo prejuízos financeiros, danos à reputação de marcas e pesadas sanções sob legislações como a LGPD e GDPR.

Nesse cenário, a segurança deve ser um pilar no desenvolvimento de sistemas, uma prática conhecida como Security by Design. Contudo, a pressão por agilidade e a falta de conhecimento técnico ainda geram inúmeros aplicativos vulneráveis.

Este trabalho busca preencher essa lacuna, oferecendo um estudo aprofundado sobre os desafios em segurança mobile. Baseado no projeto OWASP Mobile Top 10, o objetivo é capacitar desenvolvedores a identificar e mitigar riscos, servindo como um guia prático para a construção de aplicações mais seguras e resilientes.

2 PRINCIPAIS VULNERABILIDADES EM APLICATIVOS MÓVEIS

A Fundação OWASP, através do seu projeto Mobile Security, cataloga as falhas de segurança mais críticas e prevalentes em aplicações móveis. A compreensão aprofundada dessas categorias é o primeiro passo para a construção de defesas eficazes.

2.1 M1: USO INADEQUADO DA PLATAFORMA

Esta vulnerabilidade surge do desconhecimento ou da má utilização das funcionalidades de segurança, convenções e APIs fornecidas pelos sistemas operacionais Android e iOS.

- **Descrição Técnica:** No Android, um exemplo clássico é a declaração de um componente (Activity, Service, Broadcast Receiver) como exported="true" no AndroidManifest.xml sem a devida proteção por permissões. Isso permite que qualquer outro aplicativo no dispositivo invoque este componente, potencialmente explorando suas funcionalidades internas ou passando-lhe dados maliciosos. Outro exemplo é o uso indevido de Intents pendentes (PendingIntent) que podem ser sequestrados por um app malicioso para executar ações com os privilégios do app vítima.

- **No iOS:** Uma falha comum é a configuração incorreta da proteção de dados em arquivos (NS File Protection). Se um desenvolvedor não especifica um nível de proteção adequado (ex: NS File Protection Complete), os arquivos que contêm dados sensíveis podem ficar acessíveis mesmo quando o dispositivo está bloqueado. O uso inadequado de esquemas de URL (URL Schemes) também pode permitir que um app malicioso invoque funcionalidades do app vítima e extraia dados.

- **Impacto e Exploração:** O impacto varia desde o vazamento de informações até a execução remota de código no contexto do aplicativo vulnerável. Um atacante pode desenvolver um app de fachada que escaneia o dispositivo em busca de componentes exportados vulneráveis e os explora para roubar tokens de sessão, dados de usuário ou redirecionar o fluxo da aplicação para fins de phishing.

2.2 M2: ARMAZENAMENTO INSEGURO DE DADOS

Uma das falhas mais comuns e perigosas, consiste em armazenar informações sensíveis sem criptografia ou em locais facilmente acessíveis.

- **Descrição Técnica:** Inclui salvar senhas, chaves de API, tokens de autenticação ou dados pessoais em arquivos de texto plano, bancos de dados SQLite não criptografados, SharedPreferences (Android) ou UserDefaults (iOS). Estes locais de armazenamento, embora convenientes, não oferecem proteção intrínseca.

- **Cenário de Ataque:** Um atacante com acesso físico ao dispositivo, ou que tenha infectado o aparelho com um malware que obteve privilégios de root (Android) ou jailbreak (iOS), pode navegar livremente pelo sistema de arquivos. Com o comando adb shell (Android) ou um cliente SSH (iOS com jailbreak), ele pode acessar o diretório de dados do aplicativo (ex: /data/data/com.empresa.app/) e extrair os arquivos.
    - Exemplo de comando para extrair um banco de dados:
      Bash
      adb exec-out run-as com.empresa.app cat databases/usuario.db > usuario.db

  Uma vez extraído, o banco de dados pode ser aberto com qualquer visualizador de SQLite, revelando os dados em texto claro.

- **Impacto:** Comprometimento total das contas de usuário, roubo de identidade e

acesso a todos os dados armazenados pelo aplicativo.

2.3 M3: COMUNICAÇÃO INSEGURA

Refere-se à falha em proteger adequadamente os dados enquanto estão sendo transmitidos pela rede.

- **Descrição Técnica:** A causa mais óbvia é o uso de protocolos não criptografados como o HTTP. No entanto, mesmo ao usar HTTPS, a implementação pode ser falha. Erros comuns incluem: não validar a cadeia de certificados do servidor, aceitar qualquer certificado (confiando em certificados auto assinados), ou o uso de versões de protocolo TLS/SSL fracas e cifras criptográficas obsoletas.

- **Cenário de Ataque (MitM):** Um atacante conectado à mesma rede Wi-Fi pública que a vítima (em um café, aeroporto) pode usar ferramentas como **Ettercap** ou **bettercap** para realizar um ataque de envenenamento de ARP (ARP spoofing). Isso o posiciona como um intermediário (*Man-in-the-Middle*) entre o dispositivo da vítima e o roteador. Todo o tráfego do dispositivo passa pelo seu computador. Com um proxy como o **Burp Suite** ou **OWASP ZAP**, ele pode interceptar, ler e até modificar as requisições HTTPS se a validação do certificado no app for fraca, pois ele pode apresentar seu próprio certificado falso, que o app aceitará erroneamente.

- **Impacto:** Roubo de credenciais de login, tokens de sessão, dados de cartão de crédito e qualquer outra informação trocada entre o app e o servidor.

2.4 M4: AUTENTICAÇÃO INSEGURA

Cobre todas as fraquezas no processo de verificação da identidade do usuário.

- **Descrição Técnica:** Vai além de senhas fracas. Inclui a capacidade de um

atacante executar ações em nome do usuário sem se autenticar, falhas no mecanismo de "lembrar-me" que armazenam credenciais de forma insegura, e a não imposição de uma nova autenticação para funcionalidades críticas (como a troca de senha ou transações financeiras). A autenticação biométrica pode ser facilmente contornada (por exemplo, se a verificação for apenas um booleano true/false retornado pela UI, que um atacante pode forçar usando ferramentas de instrumentação).

- **Impacto:** Acesso não autorizado a contas (*Account Takeover*), fraude e roubo de dados.

- **Exploração:** Ataques de força bruta contra endpoints de login que não possuem limitação de tentativas (*rate limiting*), ou a exploração de endpoints da API que foram deixados desprotegidos e não verificam o token de sessão do usuário antes de executar uma ação.

2.5 M5: CRIPTOGRAFIA INSUFICIENTE

Ocorre quando a criptografia é aplicada, mas de forma frágil ou incorreta.

- **Descrição Técnica:** Envolve o uso de algoritmos criptográficos conhecidamente quebrados ou fracos (ex: DES, RC4, MD5, SHA1). Um erro crítico é o *hardcoding* de chaves criptográficas diretamente no código-fonte do aplicativo.
- **Cenário de Ataque:** Um atacante pode usar ferramentas de engenharia reversa para descompilar o app. Ao analisar o código, ele pode encontrar a chave hardcoded.
    - Exemplo em código Java (Android):

      Java

      ```
      // PÉSSIMA PRÁTICA! NÃO FAÇA ISSO!
      String chaveSecreta = "minhaChaveSuperSecreta123";
      Cipher c = Cipher.getInstance("AES");
      SecretKeySpec keySpec = new SecretKeySpec(chaveSecreta.getBytes(), "AES");
      c.init(Cipher.DECRYPT_MODE, keySpec);
      ```

Com a posse da chave, o atacante pode decifrar todos os dados que foram criptografados com ela, sejam eles arquivos locais ou dados interceptados na rede.

- **Impacto:** Anulação completa dos benefícios da criptografia, resultando na exposição de todos os dados supostamente protegidos.

2.6 M6: AUTORIZAÇÃO INSEGURA

Diferente da autenticação (que confirma quem você é), a autorização determina o que você pode fazer.

- **Descrição Técnica:** Falhas de autorização ocorrem quando um usuário autenticado consegue acessar recursos ou executar ações que não deveria. Um exemplo clássico é a Referência Insegura e Direta a Objetos (IDOR - *Insecure Direct Object Reference*). Isso acontece quando a API utiliza um identificador fornecido pelo cliente (ex: id_usuario=123) para acessar um recurso, sem verificar se o usuário autenticado de fato tem permissão para acessar o recurso daquele ID.

- **Cenário de Ataque (IDOR):** Um usuário legítimo faz uma requisição para ver seus próprios dados: GET /api/v1/user/123/profile. O atacante, também um usuário legítimo com ID 456, intercepta essa requisição e a modifica para GET /api/v1/user/123/profile, trocando seu próprio ID pelo da vítima. Se o backend não verificar que o usuário da sessão (ID 456) não é o dono do perfil solicitado (ID 123), ele retornará os dados do perfil da vítima.
- **Impacto:** Acesso, modificação ou exclusão de dados de outros usuários, escalonamento de privilégios.

2.7 M9: ENGENHARIA REVERSA

É o processo de desconstruir um aplicativo compilado para entender seu funcionamento interno, sua lógica de negócio e seus segredos.

- **Descrição Técnica:** Atacantes usam descompiladores (como **Jadx** para Android) e desmontadores (como **Hopper** ou **Ghidra** para iOS) para converter o código de máquina do app de volta para um formato legível por humanos (Java/Kotlin ou Objective-C/Swift).

- **Objetivos do Atacante:**
    1. **Encontrar segredos:** Chaves de API, senhas, chaves criptográficas hardcoded.
    2. **Entender a lógica da API:** Mapear todos os endpoints da API, mesmo os não documentados.
    3. **Desativar defesas:** Localizar e desabilitar código de detecção de root/jailbreak ou de *certificate pinning*.
    4. **Roubo de Propriedade Intelectual:** Copiar algoritmos proprietários.

- **Impacto:** Facilita a exploração de todas as outras vulnerabilidades, além de permitir a criação de versões maliciosas e repackaging do aplicativo.

3 BOAS PRÁTICAS DE SEGURANÇA NO DESENVOLVIMENTO MOBILE

Mitigar as vulnerabilidades descritas exige uma abordagem de defesa em profundidade (*defense-in-depth*), onde múltiplas camadas de segurança trabalham em conjunto.

3.1 CRIPTOGRAFIA DE DADOS: UMA DEFESA EM CAMADAS

- **Dados em Trânsito (Comunicação):**

- **TLS por padrão:** Toda a comunicação de rede deve, sem exceção, usar TLS (HTTPS), versão 1.2 ou superior.
- **Certificate Pinning:** Para aplicações de alta sensibilidade (bancos, saúde), implementar *certificate pinning* é crucial. Isso envolve embutir no aplicativo a identidade (o certificado público ou sua hash) do servidor esperado. Durante o handshake TLS, o app compara o certificado apresentado pelo servidor com o que ele tem armazenado. Se não corresponderem, a conexão é abortada, frustrando ataques MitM mesmo que o atacante consiga um certificado falso de uma Autoridade Certificadora (CA) confiável.
  - **Implementação no Android:** Pode ser feito via NetworkSecurityConfig.xml:

    XML

```
<network-security-config>
    <domain-config>
                                                    <domain
includeSubdomains="true">api.suaempresa.com</domain>
        <pin-set expiration="2026-01-01">
            <pin digest="SHA-256">PIN_DO_CERTIFICADO_AQUI</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

- **Dados em Repouso (Armazenamento):**
  - **Gestão de Chaves:** A segurança da criptografia depende da segurança da chave. As chaves nunca devem ser hardcoded.
    - **Android Keystore System:** Este sistema permite que o app gere e armazene chaves criptográficas em um contêiner seguro, que pode ser apoiado por hardware de segurança como o *Trusted Execution Environment* (TEE). O sistema operacional gerencia o acesso à chave,

impedindo que ela seja extraída, mesmo em um dispositivo com root. A biblioteca **Jetpack Security** abstrai essa complexidade.

- **iOS Keychain Services:** O Keychain é um banco de dados criptografado e seguro para armazenar pequenos segredos. Os itens no Keychain são protegidos pelo sistema e podem ser configurados para exigir autenticação do usuário (Face ID/Touch ID) para acesso, sendo gerenciados pelo *Secure Enclave*.

- **Criptografia de Arquivos e Banco de Dados:** Para dados maiores, use APIs de criptografia (como a já mencionada Jetpack Security ou a API **SQLCipher** para criptografar bancos de dados SQLite inteiros) com chaves gerenciadas pelo Keystore/Keychain.

3.2 AUTENTICAÇÃO ROBUSTA E CONTROLE DE SESSÕES

- **Autenticação Multifator (MFA):** Deve ser um padrão, não uma opção. A combinação de algo que o usuário sabe (senha), algo que ele tem (celular com app autenticador) e algo que ele é (biometria) oferece uma defesa robusta contra o roubo de credenciais.
- **Gerenciamento de Tokens de Sessão:** A abordagem moderna utiliza um par de tokens:
  - **Access Token:** Um JWT de vida curta (ex: 15 minutos) que é enviado a cada requisição para a API. Por ser de curta duração, o risco em caso de vazamento é limitado no tempo.
  - **Refresh Token:** Um token de vida longa (ex: dias ou semanas) que é armazenado de forma segura no dispositivo (usando Keychain/Keystore) e é usado exclusivamente para obter um novo *access token* quando o antigo expira. Ele só é enviado para um endpoint específico de renovação.
- **Biometria Segura:** Sempre utilize as APIs nativas (BiometricPrompt no Android, LocalAuthentication no iOS). Elas realizam a autenticação dentro do ambiente

seguro do SO, e o app recebe apenas uma notificação de sucesso ou falha, sem nunca ter acesso aos dados biométricos em si.

3.3 GESTÃO DE PERMISSÕES E O RESPEITO À PRIVACIDADE DO USUÁRIO

- **Princípio do Menor Privilégio e Acesso *Just-in-Time*:** Solicite o mínimo de permissões necessárias. Se uma permissão só é necessária para uma funcionalidade específica, solicite-a contextualmente, no momento em que o usuário tenta usar essa funcionalidade, explicando claramente o motivo. Isso aumenta a confiança e a probabilidade de o usuário conceder o acesso.
- **Conexão com LGPD/GDPR:** A LGPD, em seu Art. 6º, estabelece o princípio da necessidade, que limita o tratamento de dados ao mínimo necessário para a realização de suas finalidades. Solicitar permissões excessivas viola diretamente este princípio e pode acarretar sanções. A transparência na solicitação de permissões atende ao princípio do livre acesso e da transparência.
- **Análise de Permissões Perigosas:** No Android, permissões como READ_SMS, PROCESS_OUTGOING_CALLS, e ACCESS_FINE_LOCATION são consideradas perigosas por darem acesso a dados privados sensíveis. O uso delas é rigorosamente fiscalizado pela Google Play Store e deve ser evitado a todo custo, a menos que seja a funcionalidade principal e indispensável do app.

4 CASOS REAIS DE FALHAS DE SEGURANÇA EM APLICATIVOS POPULARES

4.1 BRITISH AIRWAYS: VAZAMENTO DE DADOS VIA SCRIPT MALICIOSO

- **Análise Técnica Detalhada:** A vulnerabilidade explorada foi uma falha de Cross-Site Scripting (XSS) nos servidores da British Airways. Os atacantes conseguiram modificar um arquivo JavaScript da biblioteca *Modernizr* no site, injetando 22 linhas de código malicioso. Como o aplicativo móvel carregava a página de pagamento através de um WebView, ele herdou a vulnerabilidade do

site. O script malicioso criava um *listener* para os eventos de mouseup e touchend no botão de submissão do formulário. Quando o usuário clicava para pagar, o script serializava os dados do formulário (nome, endereço, número do cartão, data de validade e CVV) e os enviava em formato JSON para um servidor controlado pelos atacantes (baways.com), antes mesmo que a transação legítima fosse processada.

- **Lição Aprendida:** A segurança de um app mobile é indissociável da segurança de seus endpoints de backend e de suas dependências web. É crucial implementar **Sub Resource Integrity (SRI)** e **Content Security Policy (CSP)** para garantir que apenas scripts conhecidos e íntegros sejam executados.

4.2 GRINDR: EXPOSIÇÃO DA LOCALIZAÇÃO DE USUÁRIOS POR API INSEGURA

- **Análise Técnica Detalhada:** A falha não estava no aplicativo em si, mas na API que ele consumia. A API retornava uma lista de perfis próximos, e para cada perfil, havia um campo "distance" com alta precisão. Pesquisadores criaram um script que consultava a API do Grindr a partir de três localizações geográficas falsas. Ao registrar a distância da vítima a cada um desses três pontos, eles podiam usar a **trilateração** (um processo matemático similar ao usado pelo GPS) para triangular a posição exata da vítima no mapa com uma margem de erro de poucos metros.

- **Lição Aprendida:** As APIs devem ser projetadas com a privacidade em mente. Retornar dados excessivamente precisos pode criar riscos. A solução envolve a ofuscação de dados (retornar distâncias aproximadas, "a menos de 1km") e a implementação de *rate limiting* robusto na API para dificultar a automação de consultas em massa.

4.3 TIKTOK: VULNERABILIDADES NA CADEIA DE CDN

- **Descrição do Problema:** Em 2020, pesquisadores da Check Point Research descobriram uma série de vulnerabilidades no TikTok. Uma delas permitia que um atacante enviasse uma mensagem SMS falsa para um usuário, contendo um link malicioso. Ao clicar no link, o atacante poderia executar código e obter controle sobre a conta do usuário, permitindo-lhe apagar vídeos, fazer upload de vídeos não autorizados e tornar vídeos privados em públicos. A falha principal residia no fato de que parte da infraestrutura do TikTok, especificamente sua rede de distribuição de conteúdo (CDN) para vídeos, ainda utilizava HTTP, permitindo a um atacante em uma posição de MitM redirecionar o app para um servidor malicioso.

- **Lição Aprendida:** Toda a cadeia de comunicação, incluindo serviços de terceiros como CDNs, deve ser segura. A política "HTTPS Everywhere" é fundamental. Além disso, a validação de links e redirecionamentos deve ser rigorosa para evitar que os usuários sejam levados a sites maliciosos.

5 FERRAMENTAS E TÉCNICAS DE ANÁLISE DE SEGURANÇA MOBILE

5.1 ANÁLISE ESTÁTICA (SAST) E ANÁLISE DE COMPOSIÇÃO DE SOFTWARE (SCA)

- **SAST (Static Application Security Testing):** Analisa o código-fonte em busca de padrões que indicam vulnerabilidades.
    - **Ferramentas: MobSF (Mobile Security Framework)** é uma ferramenta open-source poderosa que automatiza a análise estática e dinâmica. **Checkmarx** e **Veracode** são soluções comerciais líderes que se integram a pipelines de CI/CD para análise contínua.

- **SCA (Software Composition Analysis):** Foca em identificar as bibliotecas e

dependências de terceiros no projeto e as compara com bancos de dados de vulnerabilidades conhecidas (CVEs).

- **Ferramentas: OWASP Dependency-Check**, **Snyk**, **GitHub Dependabot**. Essencial, pois uma vulnerabilidade em uma biblioteca popular pode afetar milhares de aplicativos.

5.2 ANÁLISE DINÂMICA (DAST) E TESTES INTERATIVOS (IAST)

- **DAST (Dynamic Application Security Testing):** Testa o aplicativo em execução, simulando ataques.
  - **Configuração de Proxy:** O tráfego do dispositivo móvel (físico ou emulador) é redirecionado através de um proxy de interceptação como o **Burp Suite** ou o **OWASP ZAP**. Isso permite ao analista inspecionar, modificar e reenviar requisições entre o app e o servidor para testar falhas de IDOR, injeções de SQL, etc.

- **IAST (Interactive Application Security Testing):** Uma abordagem híbrida que combina SAST e DAST. Agentes de instrumentação são usados para monitorar o fluxo de dados dentro do aplicativo enquanto ele é testado dinamicamente, permitindo identificar com precisão a linha de código vulnerável que foi explorada.

5.3 TESTES DE INVASÃO (PENTEST) EM AMBIENTES ANDROID/IOS

O pentest é o teste mais aprofundado, simulando as ações de um atacante real.

- **Metodologia:**
  1. **Reconhecimento:** Coleta de informações sobre o app, a empresa e a infraestrutura de backend.
  2. **Análise Estática e Engenharia Reversa:** Descompilação do app (.apk/.ipa) para encontrar segredos e entender a lógica.
  3. **Análise Dinâmica:** Configuração do proxy, testes de API, análise de

comunicação inter-processos (IPC).

4. **Instrumentação em Tempo de Execução:** Utilização de ferramentas como o **Frida** ("o Greasemonkey dos aplicativos nativos"). Frida permite injetar snippets de JavaScript no processo do aplicativo em execução para "hookar" funções. Um analista pode usar Frida para:
    - **Bypass de Certificate Pinning:** Encontrar a função que realiza a verificação do pino e forçá-la a sempre retornar true.
    - **Bypass de Detecção de Root/Jailbreak:** Interceptar chamadas que verificam a existência de arquivos como /system/bin/su e manipulá-las para que o app pense que está em um ambiente seguro.
    - **Monitorar Criptografia:** Visualizar dados antes de serem criptografados e após serem decifrados, para validar a implementação.

6 CHECKLIST DE SEGURANÇA PARA O CICLO DE DESENVOLVIMENTO SEGURO

Este checklist expandido deve ser usado como um guia prático em cada fase do SDLC.

Fase 0: Treinamento e Conscientização

- [ ] Equipe de desenvolvimento treinada em práticas de desenvolvimento seguro (ex: OWASP Top 10).
- [ ] A segurança é definida como uma responsabilidade de todos, não apenas de uma equipe.

Fase 1: Requisitos e Design (Threat Modeling)

- [ ] Realizar modelagem de ameaças usando uma metodologia como STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).
- [ ] Definir requisitos de segurança para dados em trânsito (TLS 1.3, Certificate Pinning) e em repouso (AES-256-GCM com chaves no Keystore/Keychain).
- [ ] Mapear todas as permissões necessárias e justificar cada uma delas (Princípio

do Menor Privilégio).

Fase 2: Desenvolvimento (Codificação Segura)

- [ ] **Validação de Entrada:** Validar, sanitizar e codificar todas as entradas de fontes externas (usuário, APIs, WebViews).
- [ ] **Gerenciamento de Erros:** Capturar exceções e exibir mensagens de erro genéricas para o usuário, registrando os detalhes completos apenas nos logs do servidor.
- [ ] **Prevenção de Engenharia Reversa:** Habilitar ofuscação e minificação de código (ProGuard/R8 no Android).
- [ ] **Controles Anti-Tampering:** Implementar verificações de integridade do código em tempo de execução para detectar se o app foi modificado.
- [ ] **Detecção de Ambiente Inseguro:** Implementar verificações de root (Android) e jailbreak (iOS) e definir uma política de resposta (ex: encerrar o app ou limitar funcionalidades).
- [ ] **Segurança de WebView:** Desabilitar a execução de JavaScript (setJavaScriptEnabled(false)) se não for necessário. Limitar os domínios que podem ser carregados.

Fase 3: Testes de Segurança

- [ ] Integrar ferramentas SAST e SCA no pipeline de CI/CD para feedback rápido.
- [ ] Realizar varreduras DAST em ambientes de teste.
- [ ] Conduzir um pentest completo por uma equipe interna ou externa especializada antes de cada lançamento majoritário.
- [ ] Verificar se todos os endpoints da API possuem controles de autorização adequados.
- [ ] Garantir que o app se comporta de forma segura em caso de perda de conectividade ou interrupções.

Fase 4: Lançamento e Resposta a Incidentes

- [ ] Publicar o manifesto de privacidade do app nas lojas (App Store, Play Store).
- [ ] Monitorar continuamente os logs em busca de padrões de ataque.
- [ ] Manter um plano de resposta a incidentes atualizado e testado.
- [ ] Ter um canal claro para que pesquisadores de segurança possam reportar vulnerabilidades de forma responsável (*Vulnerability Disclosure Program*).

7 CONCLUSÃO

A jornada pela segurança de aplicativos móveis é complexa, dinâmica e sem um destino final. Este trabalho buscou desmistificar essa jornada, demonstrando que, embora os desafios sejam significativos, as soluções são, em grande parte, acessíveis e implementáveis através da disciplina, do conhecimento técnico e da adoção de uma cultura de segurança proativa.

Para o futuro profissional de Análise e Desenvolvimento de Sistemas, as conclusões deste estudo são inequívocas. A proficiência técnica deve andar de mãos dadas com uma sólida compreensão dos princípios de segurança. Em um cenário onde novas ameaças e tecnologias, como a Internet das Coisas (IoT) e a computação em nuvem, se integram cada vez mais aos dispositivos móveis, a superfície de ataque só tende a aumentar. Portanto, a capacidade de construir aplicações seguras por design não é mais um diferencial, mas sim um requisito fundamental para a relevância e a responsabilidade profissional no século XXI. A segurança não é um recurso a ser adicionado, mas a fundação sobre a qual aplicações confiáveis são construídas.

REFERÊNCIAS

GOOGLE. **Security best practices**. Android Developers. Disponível em: *https://developer.android.com/topic/security/best-practices*. Acesso em: 10 jun. 2025.

GOOGLE. **Network security configuration**. Android Developers. Disponível em: *https://developer.android.com/training/articles/network-security-config*. Acesso em: 10 jun. 2025.

OWASP FOUNDATION. **OWASP Mobile Top 10**. Disponível em: *https://owasp.org/www-project-mobile-top-10/*. Acesso em: 10 jun. 2025.

OWASP FOUNDATION. **OWASP Mobile Application Security Verification Standard (MASVS)**. Disponível em: *https://owasp.org/www-project-mobile-app-security/*. Acesso em: 10 jun. 2025.

Fontes extras:

1. *https://abxtelecom.com.br/armazenamento-de-dados/armazenamento-de-dados/*
2. *https://www.scielo.br/j/abc/a/85K4S7dwsXSVCHVLrLZD4jJ/?lang=en*