

Hackeando aplicativos iOS

um guia de teste detalhado

Preparado por:
Dinesh Shetty,
Gerente Sênior - Segurança da Informação
@Din3zh



Índice

1. Configuração do laboratório de pentest do iOS.....	5
1.1 Obter um dispositivo iOS.....	5
1.2 Como fazer jailbreak em um dispositivo iOS	7
1.3 Instalação do software e dos utilitários necessários	10
2. Aquisição de binários do iOS	13
3. Geração de binário do iOS (arquivo .IPA) a partir do código-fonte do Xcode	15
3.1 Método I - Com uma conta de desenvolvedor paga válida.....	15
3.2 Método II - Sem uma conta de desenvolvedor paga válida	18
4. Instalação de binários do iOS em dispositivos físicos	23
4.1 Método I - Usando o iTunes.....	23
4.2 Método II - Usando o Cydia Impactor	27
4.3 Método III - Usando o iOS App Signer.....	27
4.4 Método IV - Instalação do arquivo .app	27
4.5 Método V - Instalação do binário modificado.....	28
4.6 Método VI - Usando o utilitário Installipa	29
4.7 Método VII - Usando o utilitário de configuração do iPhone	29
4.8 Método VIII - Usando o iFunBox	29
5. Guia do pacote binário do iOS	30
5.1 Entendendo a estrutura do pacote binário do iOS.....	30
5.2 Compreensão das arquiteturas compatíveis com o aplicativo fornecido	31
5.3 Entendendo a arquitetura disponível nos dispositivos de teste	32
5.4 Conversão de binários de aplicativos de binário FAT para binário de arquitetura específica	34
5.5 Conversão de executáveis pré-iOS 9 em um executável iOS 9.....	34
5.6 Conversão de aplicativos de 32 bits em aplicativos de 64 bits no Xcode	35
6. Compilação do código-fonte fornecido pelo cliente para pentesting no iOS mais recente usando o Xcode	36
6.1 Faça o download do código-fonte	36
6.2 Iniciar o espaço de trabalho	36
6.3 Configuração de aplicativos.....	37
7. Cartilha do modelo de segurança do iOS	41
7.1 Recursos de segurança.....	41
8. Explorando o sistema de arquivos do iOS	42

8.1 Leitura de dados usando o iExplorer	42
8.2 Leitura de dados usando o iFunBox	42
8.3 Leitura dos dados da caixa de areia do aplicativo iOS > 8.3 usando o método de backup	44
8.3.1 <i>Fazendo backup do iDevice</i>	44
8.3.2 <i>Usando o iBackupBot</i>	45
8.3.3 <i>Usando o iExplorer</i>	45
8.4 Leitura de dados de aplicativos usando o OpenSSH	47
8.5 Leitura de dados do aplicativo usando SSH via USB.....	48
8.6 Leitura de dados de aplicativos no dispositivo iOS	49
8.6.1 <i>FileExplorer/iFile</i>	49
8.6.2 <i>Uso de terminais móveis</i>	50
9. Criptografia de dados de aplicativos	50
9.1 Entendendo a API de proteção de dados da Apple	50
9.2 Validar as classes de proteção de dados que estão sendo usadas.....	51
9.3 Armazenamento local inseguro de dados	52
9.3.1 <i>Arquivos de PropertyList</i>	52
9.3.2 <i>Classe NSUserDefaults</i>	53
9.3.3 <i>Chaveiro</i>	54
9.3.4 <i>Bancos de dados CoreData e SQLite</i>	57
9.4 Criptografia quebrada	58
10. Análise binária	61
10.1 Análise binária - Verificação de mitigações de exploração - Executável independente de posição (PIE e ASLR)	61
10.2 Análise binária - Verificação de mitigações de explorações - Contagem automática de referências (ARC)	62
10.3 Análise binária - Verificação de mitigações de exploração - Protetores de pilha	64
10.4 Análise binária - Listar todas as bibliotecas usadas no binário do iOS.....	65
10.5 Engenharia reversa simples de binários do iOS usando class-dump-z.....	68
11. Descriptografia de aplicativos iOS (binários da AppStore)	72
11.1 Método manual	72
11.1.1 <i>Usando o GDB</i>	72
11.1.2 <i>Usando o LLDB</i>	75
11.2 Método automatizado	79
11.2.1 <i>Usando o dump descriptografado</i>	79
11.2.2 <i>Usando a embreagem</i>	81
12. Depuração de aplicativos iOS - Manipulação do tempo de execução	85
12.1 Cycript em um dispositivo com jailbreak	85
12.1.1 <i>Uso do Cycript para invocar métodos internos</i>	85
12.1.2 <i>Uso do Cycript para substituir métodos internos</i>	90
12.2 Depuração de aplicativos iOS usando LLDB	94
13. Engenharia reversa usando o funil	100
14. Engenharia reversa usando o IDA PRO	112

15. MITM no iOS	113
15.1 Tráfego HTTP MITM	114
15.2 MITM Tráfego SSL/TLS	116
15.3 Tráfego MITM não HTTP/SSL/TLS	118
15.4 MITM usando VPN	118
15.5 MITM quando o aplicativo iOS é acessível somente via VPN	119
15.6 MITM contornando a fixação de certificados	120
15.7 MITM por sequestro de DNS.....	123
15.8 MITM usando gateway de rede	123
15.9 Monitoramento das atividades do sistema de arquivos do iOS.....	124
16. Vazamento no canal lateral	127
16.1 Mecanismo de cache de captura de tela padrão do iOS.....	127
16.2 iOS UIPasteboard Caching.....	130
16.3 Armazenamento de cookies do iOS	132
16.4 Armazenamento de cache de teclado do iOS	134
16.5 Registro de dispositivos iOS	137

1. Configuração do laboratório de pentest do iOS

A configuração de um dispositivo é uma das primeiras prioridades antes de iniciar um projeto programado. Se estiver configurando um dispositivo iOS pela primeira vez, é provável que algo se quebre (mesmo que o dispositivo tenha sido usado anteriormente), portanto, é melhor testar o dispositivo alguns dias antes do início do pentest para garantir que as ferramentas nele contidas ainda funcionem.

1.1 Obter um dispositivo iOS

Uma fonte confiável de dispositivos iOS é o eBay (<https://www.ebay.com/>). As atualizações do iOS e a compatibilidade de hardware podem ser um problema com os produtos da Apple, portanto, sempre tente comprar um dos dispositivos mais novos. No momento da publicação deste guia, o iPhone mais recente no mercado é o Apple iPhone 7/7+ e o telefone mais antigo recomendado é o Apple iPhone 5s. Um iPad Mini também é uma boa opção. Se for preferível usar um novo dispositivo iOS, mas os casos de teste relacionados ao uso da operadora de rede não forem uma preocupação, considere um iPod Touch de 6^a geração. Eles são relativamente baratos em comparação com outros dispositivos novos que executam as versões mais recentes do iOS. Para obter melhores resultados, escolha uma versão do iOS superior a 9.0+.

OBSERVAÇÃO: ao tentar comprar um dispositivo no eBay, use a funcionalidade "Auction" (Leilão) em conjunto com o filtro "Time: ending soonest" (Tempo: terminando mais cedo).

www.ebay.com/sch/i.html?_odkw=iphone&_sop=1&LH_Auction=1&osacat=0&_from=R40&_trksid=p2045573.m570.l1313.TR0.TRC0.H0.Xiph

Hi Dinesh! | Daily Deals | Gift Cards | Help & Contact | The Right Gifts, Right Here | Sell | My eBay

ebay Shop by category

iphone 5s

Related: iphone 5s case | iphone 5c | iphone 5s unlocked | iphone 4s | iphone 5s gold | samsung galaxy s4 | iphone 5s verizon | iphone 5s 16gb... Inclu

All Listings Auction Buy It Now

Sort: Time: ending soonest View:

2,648 results for iphone 5s

FEATURED
Put a smile on their face
Save up to 40% on iPhones, MacBooks, and more gifts from Apple
[SHOP NOW](#)

Apple iPhone 5s - 16GB - Silver verizon (Factory Unlocked) Smartphone
\$157.60
13 bids
Free shipping
[See more like this](#)

Dispositivos desbloqueados com pelo menos 32 GB de memória são preferíveis, pois oferecem espaço suficiente para atualizar o dispositivo e instalar todas as ferramentas. Lembre-se de que nem todas as versões do iOS podem ser desbloqueadas, portanto, escolha um dispositivo que tenha um jailbreak público disponível (consulte a seção Jailbreak neste guia para determinar se a versão do iOS de um dispositivo pode ser desbloqueada). Se a descrição do produto indicar a versão do iOS em execução

no dispositivo que você está considerando, envie uma mensagem ao vendedor para confirmar a versão do iOS. Para enviar uma mensagem ao vendedor, abra a página do produto, vá até o final da descrição e clique no link, conforme mostrado abaixo.

Outer Width	10.3 in.
Weight	3.95 oz
Miscellaneous	
Release Date	9/10/2013

iPhone and all features works great. Has tiny nick on top, which I was lucky to get around receiver (in photo) Was kept in authentic Otterbox Defender case. Will throw in otterbox case if you would like (please specify in correspondence).

Questions and answers about this item

No questions or answers have been posted about this item.

[Ask a question](#)

1.2 Como fazer jailbreak em um dispositivo iOS

Jailbreaking é o processo de obter acesso root a todo o dispositivo. A melhor abordagem para testar a segurança de um aplicativo é examiná-lo em um dispositivo com jailbreak. O jailbreak de um dispositivo iOS permite:

- Remoção das limitações de segurança (e outras) do sistema operacional impostas pela Apple
- Fornecimento de acesso root ao sistema operacional
- Permitir a instalação de importantes ferramentas de software de teste
- Fornecimento de acesso ao tempo de execução do Objective-C

Os aplicativos iOS armazenam dados na sandbox do aplicativo, que não é acessível ao público (mas está disponível para o root e o próprio aplicativo). Sem acesso root, não é possível acessar a sandbox do aplicativo, ver quais dados estão sendo armazenados e como eles são armazenados. Além disso, a maioria dos arquivos de nível de sistema é de propriedade do root.

O processo para fazer o jailbreak de várias versões do iOS pode ser bem diferente. As instruções para fazer o jailbreak de dispositivos iOS podem ser encontradas por meio de uma simples pesquisa no Google. No entanto, esteja ciente de que os links do Google podem não ser legítimos, mesmo que incluam nomes iguais aos de ferramentas genuínas de jailbreak.

Exemplo:

<https://www.google.com/#q=jailbreak+ios+10.2>

iOS 10.2 Jailbreak - Download Pangu

www.downloadpangu.org › How to › iOS 10 ▾
iOS 10.2 Jailbreak is now possible using the latest Pangu Jailbreak tool that was created by Pangu and PP team. This tool provides iOS 10.2 Jailbreak for ...

iOS 10.0.2 Jailbreak - Download Pangu

www.downloadpangu.org › How to › iOS 10 ▾
Are you looking for iOS 10.0.2 Jailbreak from Pangu team. ... the bugs will be ironed out, maximum amount of iDevices will be sold & possibly iOS 10.2 for a solid ...

Jailbreak iOS 10.2 - Pangu 9

pangu8.com/jailbreak/10.2/ ▾
Unofficial Pangu Jailbreak tool and new Jailbreak method release for iOS 10.2. It is the future of Jailbreaking.

iOS 10/10.0.1/10.0.2/10.0.3 Jailbreak - Pangu 9

pangu8.com/10.html ▾
Pangu Jailbreak is available for iOS 10/10.0.1/10.0.2 versions and iOS 10.0.3 for ... Please refer the iOS 10.1 and 10.2 Jailbreak pages for more details about the ...

iOS 10.2 Jailbreak - iOS 9 Cydia

www.ios9cydia.com › How to ▾
iOS 10.2 Jailbreak is now possible using the latest Pangu Installer released by Pangu Team 24 hours after iOS 10.2 release. This tool allows iOS 10.2 Jailbreak.

iOS 10.2 Jailbreak - TAIG9

taig9.com/beta3.2/ ▾
iOS 10.2 Jailbreak is going to be the hottest topic in cool December, because Apple is planning to release iOS 10.2 with a set of cool new features. The new ...

iOS 10 - iOS 10.0.3 Jailbreak - TaiG9

taig9.com/beta3/ ▾
TaiG beta v3.0.0 can be used to jailbreak all the released versions of iOS 10 including iOS 10.0, iOS 10.0.1, iOS 10.0.2 ... TAIG9 BETA 3.2; iOS 10.2 Jailbreak.

O exemplo acima mostra que muitos dos resultados incluem "pangu" e "taig" (ferramentas legítimas de jailbreak), mas nenhum dos links para o iOS 10.2 é genuíno.

Sites recomendados:

- <https://www.theiphonewiki.com/wiki/Jailbreak> Um site confiável para verificar se o Jailbreak para um dispositivo iOS está disponível e qual software usar
- <https://www.redmondpie.com/> Inclui guias passo a passo com links para o software real
- <https://www.reddit.com/r/jailbreak/> Bom recurso para acompanhar os eventos de jailbreak atualizados em todo o mundo (observação: use com cautela e verifique novamente as informações encontradas neste site)

Use o guia abaixo para fazer o jailbreak de um dispositivo iOS 10.2:

<http://www.redmondpie.com/jailbreak-ios-10-for-iphone-ipad-ipod-touch-latest-status-update/>

The screenshot shows a web browser displaying a Redmond Pie article titled "Jailbreak iOS 10 / 10.2 / 10.1.1 On iPhone 7, Plus, 6s, iPad Pro Using Yalu [Updated]". The article is dated March 10th, 2017, by Paul Morris. The content discusses the jailbreak process for various iOS devices, mentioning supported and unsupported models. It provides links for download links, IPA files, and tutorials for different iOS versions.

Como este é um site legítimo, esses links podem ser usados para baixar o IPA ou o código-fonte adequado para o aplicativo de jailbreak. Este site também inclui guias úteis de orientação.

Uma rápida pesquisa no Redmond Pie confirmará se há etapas de jailbreak para várias versões do IOS, quais são elas e como implementá-las.

OBSERVAÇÃO: nunca use a opção "redefinir todo o conteúdo e as configurações" em um dispositivo iOS com jailbreak, pois ele SEMPRE ficará preso em um loop de reinicialização. Quando isso acontecer, o dispositivo precisará ser restaurado (provavelmente para a versão mais recente). Se ocorrer um loop de reinicialização, tente as etapas mencionadas nos links abaixo para corrigir o problema:

- <https://www.qdtricks.net/how-to-fix-iphone-stuck-on-apple-logo/>

- <https://support.apple.com/en-in/HT201263>
- <http://www.ikream.com/2016/02/how-to-fix-apple-iphone-6-boot-loop-blod-and-other-power-related-issues-troubleshooting-guide-23912> <http://www.iphonehacks.com/2016/08/fix-boot-loop-jailbreak-ios-9-3-3-iphone-ipad-ipod-touch.html>

1.3 Instalação do software e dos utilitários necessários

Depois de fazer o jailbreak de um dispositivo iOS, os seguintes utilitários precisarão ser instalados. A maioria das ferramentas, se não todas, pode ser instalada a partir do Cydia. O Cydia é um invólucro de GUI para o apt e, depois que o apt é instalado, o restante pode ser instalado por meio da linha de comando. O Cydia é preferível devido à facilidade de uso.

As etapas de instalação de muitas dessas ferramentas são abordadas em outra parte deste guia.

- OpenSSH
 - Um utilitário para fornecer aos usuários a capacidade de se conectar remotamente ao sistema de arquivos do iOS. O utilitário OpenSSH está quebrado no jailbreak do iOS 10.2 lançado por Luca, mas há um serviço SSH DropBear padrão em execução no dispositivo para garantir que o acesso SSH não seja perdido.
 - Conecte-se ao DropBear usando as mesmas etapas mencionadas no Método 8 (Leitura de dados do aplicativo usando SSH por USB)
 - IMPORTANTE: altere a senha do OpenSSH assim que o OpenSSH for instalado.
- Ferramentas recomendadas pelo BigBoss
 - Uma coleção de todas as ferramentas CLI recomendadas para hackers, como wget, tar, vim etc., que não vêm pré-instaladas com o repositório do Cydia.
- Substrato de Cydia
 - Um requisito importante para muitos dos ajustes e ferramentas incluídos neste guia. Necessário para modificar o software durante o tempo de execução no dispositivo sem acesso ao código-fonte. Ferramentas como o Cycript precisam do Cydia Substrate instalado.
 - Tenha cuidado ao instalar patches de terceiros no iOS mais recente. Descobriu-se que as correções feitas por Ijapija00 para o iOS 10 e 10.1.1 causavam falhas nos dispositivos.
- APT 0.6 transitional (comando apt-get)
 - Ferramentas de empacotamento para iOS
- Class-dump-z, class-dump, classdump-dyld
 - Uma ferramenta de engenharia reversa para iOS que ajuda a despejar as declarações de classes, categorias e protocolos.
- Cycript
 - Um utilitário que fornece um mecanismo para modificar aplicativos durante o tempo de execução usando uma combinação de Objective-C++ e sintaxe JavaScript.
- Console do instalador do IPA

- Um utilitário de linha de comando para instalar aplicativos de terceiros em um dispositivo iOS com jailbreak.
- AppSync
 - Um ajuste do iOS que permite a instalação de um pacote IPA modificado e falsamente assinado no dispositivo iOS.
 - Certifique-se de que o Jailbreak seja compatível com essa ferramenta ou o dispositivo poderá acabar em um loop de reinicialização.
 - O AppSync está temporariamente interrompido no jailbreak do iOS 10.2, portanto, a instalação não é recomendada.
- Clutch do repositório iphonecake (com.iphonecake.clutch2)
 - Um utilitário que permite aos usuários fazer o dump de binários iOS descriptografados de um dispositivo com jailbreak.
- GDB do repositório cydia.radare.org
 - O depurador GNU para iOS com jailbreak em arm64.
- MTerminal
 - Um terminal no dispositivo para executar comandos no dispositivo iOS sem a necessidade de um laptop separado.
- Filemon
 - Um software de monitoramento do sistema de arquivos do iOS em tempo real.
 - Pode ser baixado em www.newosxbook.com
- Introspy-iOS
 - Uma ferramenta para ajudar os pesquisadores de segurança a traçar o perfil dos aplicativos iOS usando uma abordagem de caixa preta
 - Pode ser baixado em <https://github.com/iSECPartners/Introspy-iOS>
- Chave de bloqueio SSL 2
 - Uma ferramenta para ajudar a contornar a validação e a fixação de SSL em aplicativos iOS
 - Pode ser baixado em <https://github.com/nabla-c0d3/ssl-kill-switch2>

Em um laptop, será necessário instalar o software abaixo:

- Funil
 - Uma ferramenta de engenharia reversa barata, mas útil, para ajudar a desmontar, descompilar e depurar aplicativos iOS.
- IDA Pro
 - Uma ferramenta cara, mas avançada, para ajudar na engenharia reversa do iOS.
- Suíte para arrotos
 - Um proxy de interceptação para realizar MITM em aplicativos iOS.
- idb
 - Uma ferramenta para auxiliar muitos dos casos de teste de aplicativos iOS comumente vistos.
- ArquivoDP
 - Uma ferramenta para ajudar a extrair a classe de proteção de dados de arquivos no dispositivo iOS.
 - Pode ser baixado em <http://www.securitylearn.net/wp-content/uploads/tools/iOS/FileDP.zip>

- Dispositivo libimobilizado
 - Uma excelente biblioteca de protocolo multiplataforma para acessar dispositivos iOS.
 - Pode ser baixado em <https://github.com/libimobiledevice/>

2. Aquisição de binários do iOS

Os clientes nem sempre fornecerão um arquivo .IPA para um pentest. Abaixo estão algumas maneiras alternativas de adquirir binários do iOS para análise.

1. Abra a iTunes App Store no Mac. Baixe o aplicativo da App Store usando o aplicativo Mac Native. Selecione "Apps" e escolha o nome do aplicativo na "Biblioteca". Clique com o botão direito do mouse e selecione "Mostrar no Finder" para obter o caminho do IPA. Normalmente é /Users/<username>/Music/iTunes/iTunes Media/Mobile Applications/
2. Quando o dispositivo é sincronizado com o iTunes, o arquivo .IPA é enviado para a pasta do iTunes. Extraia o arquivo .IPA da pasta do iTunes. (Funciona em dispositivos sem jailbreak)
3. Use uma ferramenta como o iMazing. Inicie o iMazing e conecte o dispositivo iOS ao laptop. Clique em Apps. Selecione o binário do aplicativo a ser extraído. Clique em Manage Apps (Gerenciar aplicativos) na parte inferior da tela. Clique em Extract App (Extrair aplicativo) e escolha um local para armazenar o aplicativo no computador. (Funciona bem em aplicativos anteriores à versão 9.0. As versões posteriores à 9.0 não funcionam bem)
4. Use uma ferramenta como o iFunBox. Inicie o iFunBox e conecte o dispositivo iOS. Clique na guia iFunBox Class e, na seção "Connected Devices" (Dispositivos conectados), selecione o dispositivo iOS. Clique em Aplicativos do usuário. Selecione o aplicativo a ser extraído. Clique com o botão direito do mouse e selecione "Backup to .ipa Package". Salve o aplicativo em qualquer local. (Funciona somente até o iOS 8.3 ou em um dispositivo com jailbreak)
5. Use o iTools. Conecte o dispositivo. Clique em Apps. Selecione o aplicativo. Clique com o botão direito do mouse e selecione arquivar para obter o binário do aplicativo. (Funciona somente até o iOS 8.3 ou em um dispositivo com jailbreak)
6. Com acesso ao código-fonte, é possível compilar o binário do aplicativo diretamente. Isso é útil quando se trabalha com dispositivos antigos com jailbreak, pois permite compilar o aplicativo para ser executado no dispositivo antigo e realizar o teste.
7. Faça o download do aplicativo na App Store. O problema de usar esses binários para testes é que eles são criptografados para sua proteção e para o gerenciamento de direitos digitais (DRM). As técnicas para quebrar o FairPlayDRM e realizar a análise dos binários criptografados da App Store serão discutidas mais adiante neste guia.
8. Use a opção "Transferir compras do dispositivo" no iTunes.

9. Às vezes, o cliente fornecerá acesso ao aplicativo por meio do TestFlight (<https://developer.apple.com/testflight/>), onde você poderá fazer login diretamente na conta e baixar o arquivo IPA.

Se todos os métodos acima falharem, o que é improvável, peça ao cliente o arquivo .IPA. Certifique-se sempre de obter o certificado de provisão móvel junto com o binário do aplicativo.

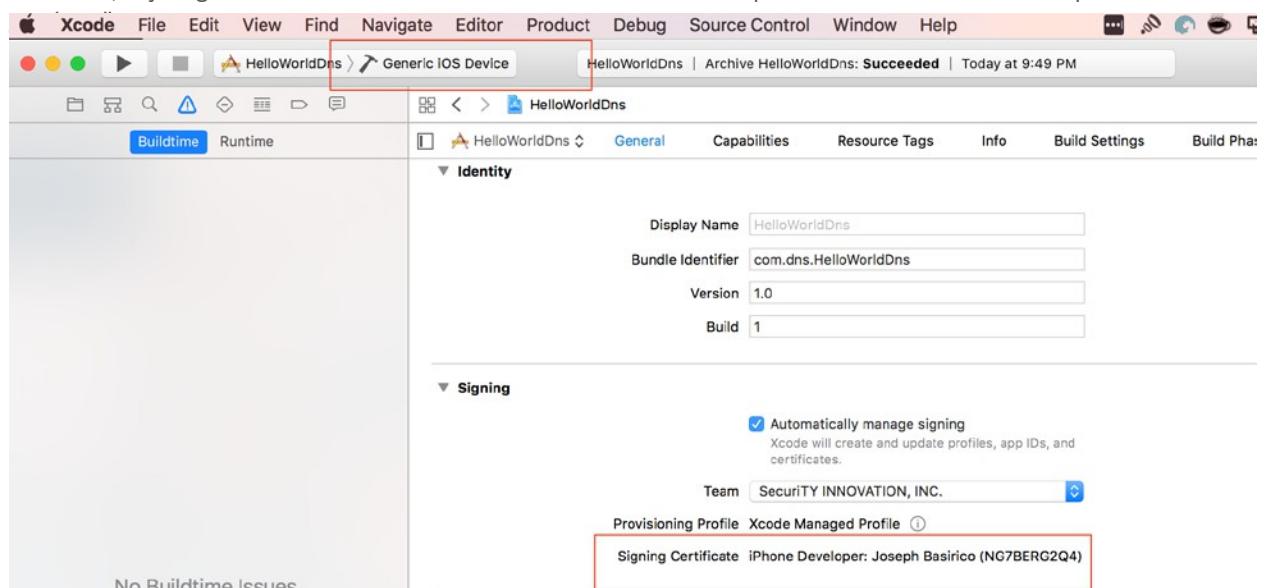
3. Geração de binário do iOS (arquivo .IPA) a partir do código-fonte do Xcode:

O teste de um aplicativo iOS requer acesso ao arquivo IPA. Abaixo estão duas maneiras de gerar arquivos IPA:

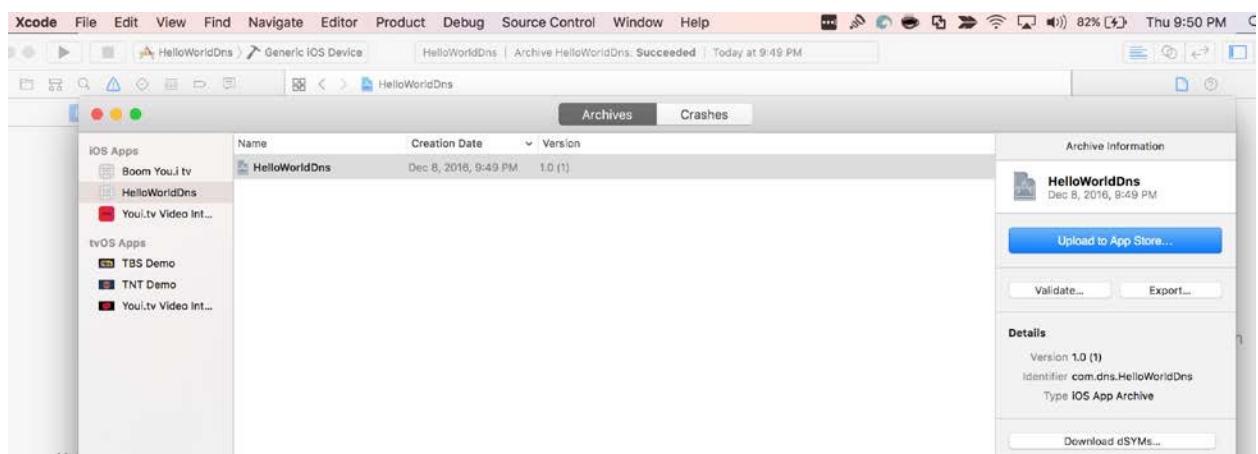
3.1 Método I - Com uma conta de desenvolvedor paga válida.

Certifique-se de que o dispositivo iOS esteja registrado na conta de desenvolvedor usando as etapas mencionadas aqui:https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#/apple_ref/doc/uid/TP40012582-CH30-SW10

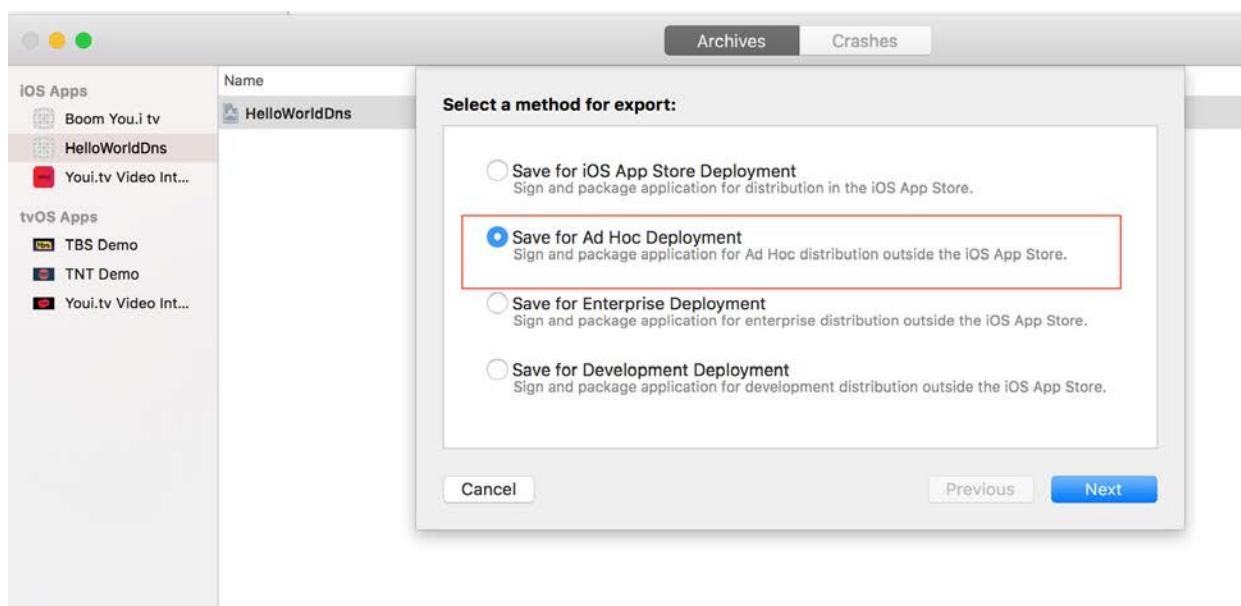
1. No Xcode, faça login na conta de desenvolvedor correta. Defina o dispositivo de destino como "Dispositivo iOS"



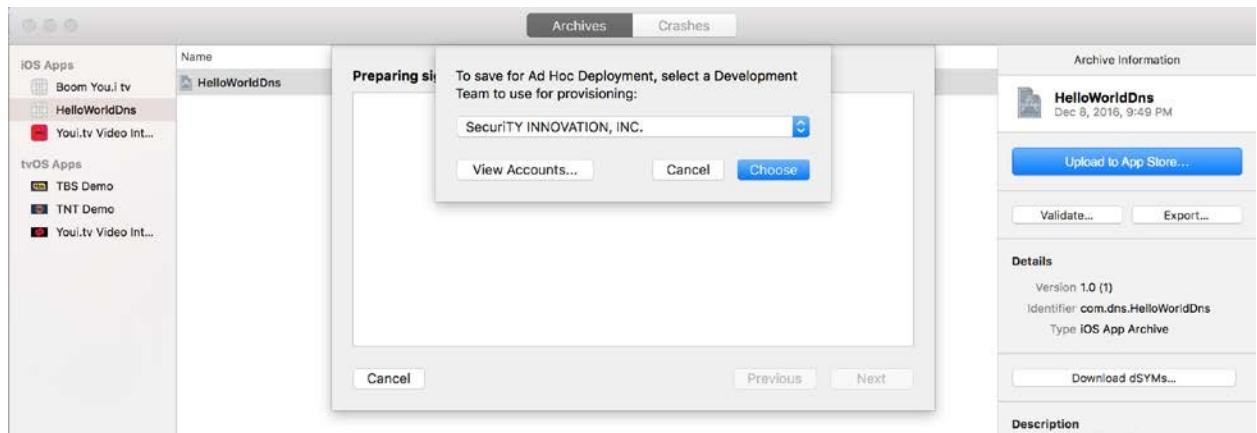
2. Conecte o dispositivo iOS a um laptop.
3. Vá para o menu Product (Produto) na parte superior e selecione Archive (Arquivar). Isso arquivará a compilação atual e fornecerá uma lista de arquivos no Organizer.



4. Vá para Janela -> Organizador. Pressione Exportar e selecione a distribuição ad-hoc.



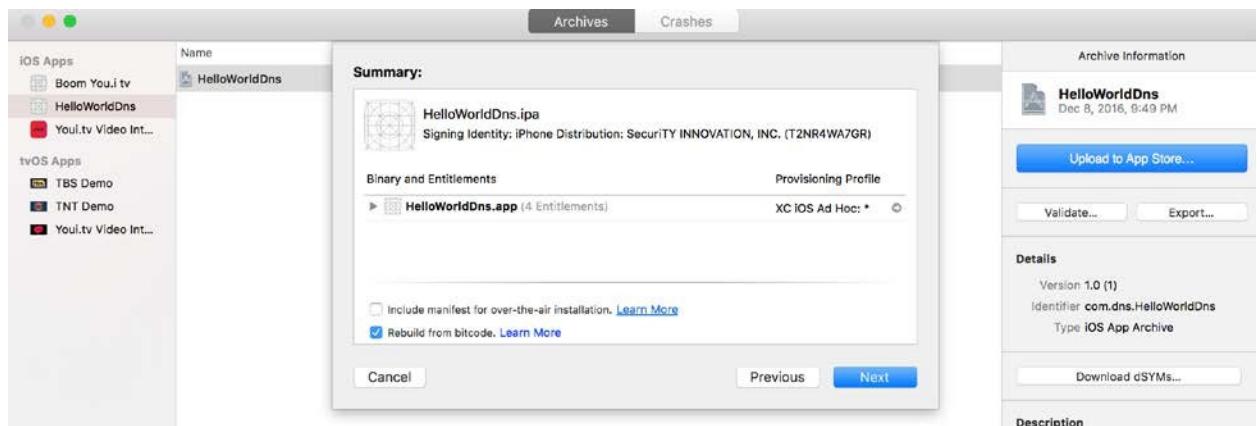
5. SELECIONE A EQUIPE DE DESENVOLVIMENTO ADEQUADA PARA O PROVISIONAMENTO.



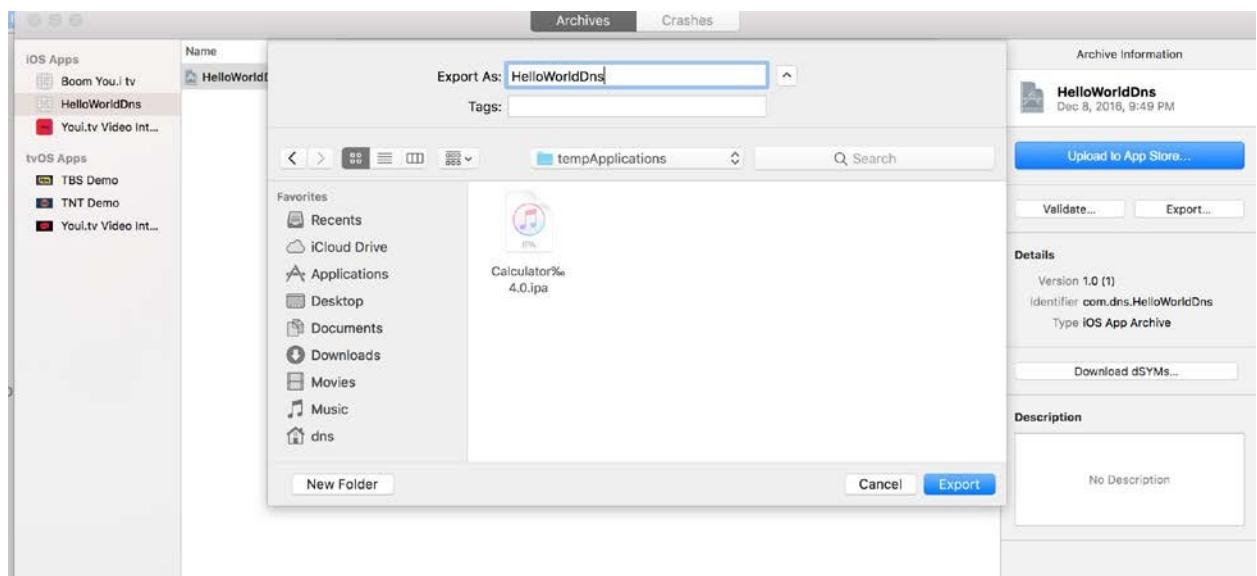
6. Selecione a opção de suporte a dispositivos mais adequada (geralmente essa é a opção padrão).



7.



8. Salve o arquivo .IPA em qualquer local conhecido do laptop para uso posterior pela equipe de testes de segurança.



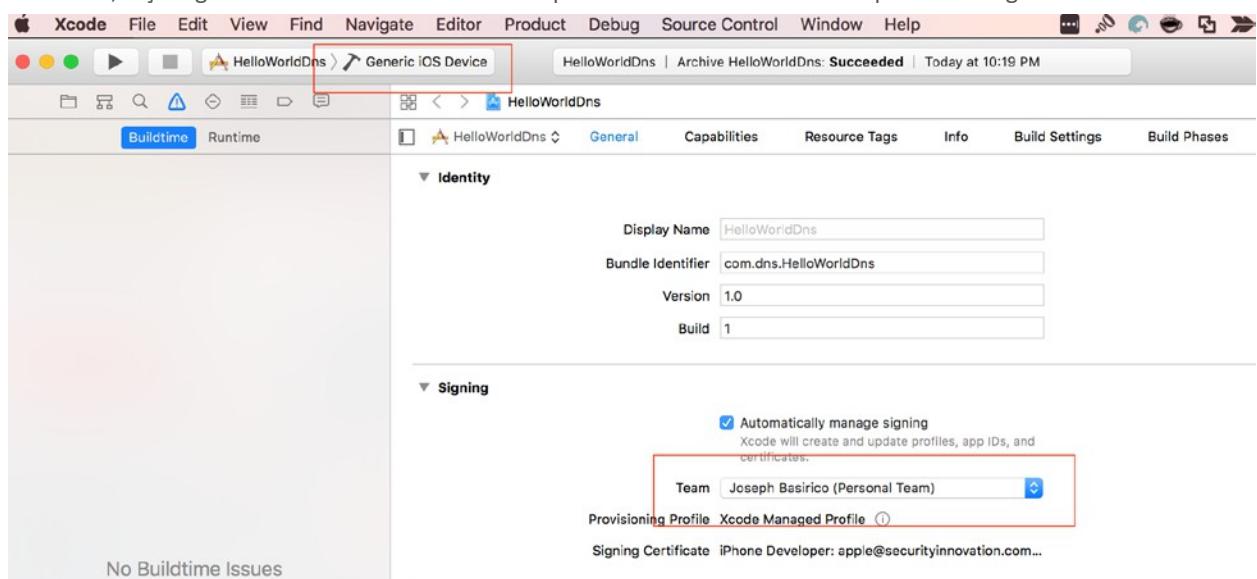
3.2 Método II - Sem uma conta de desenvolvedor paga válida

As etapas abaixo podem ser seguidas para gerar um arquivo .IPA quando não houver uma conta de desenvolvedor válida e um certificado de equipe pessoal estiver sendo usado.

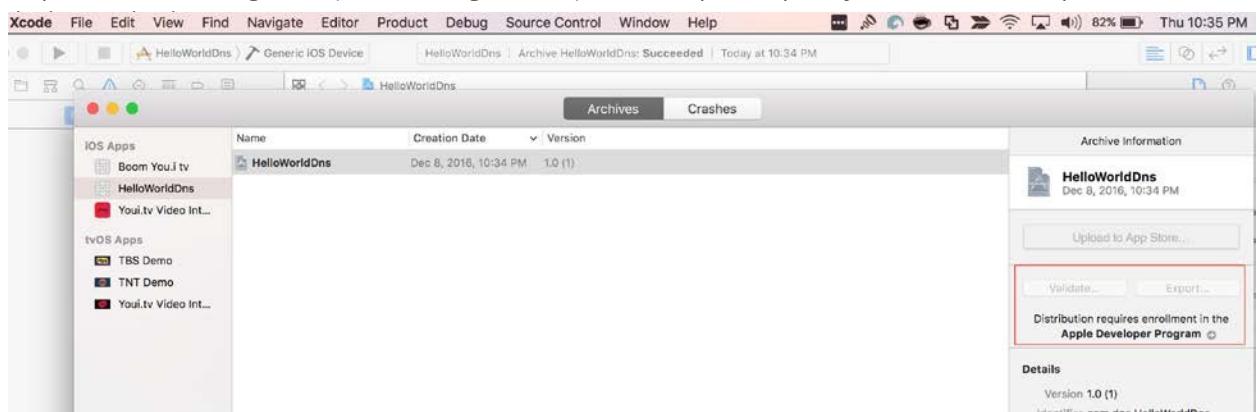
Certifique-se de que o dispositivo iOS esteja registrado na conta de desenvolvedor usando as etapas mencionadas aqui:

https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#/apple_ref/doc/uid/TP40012582-CH30-SW10

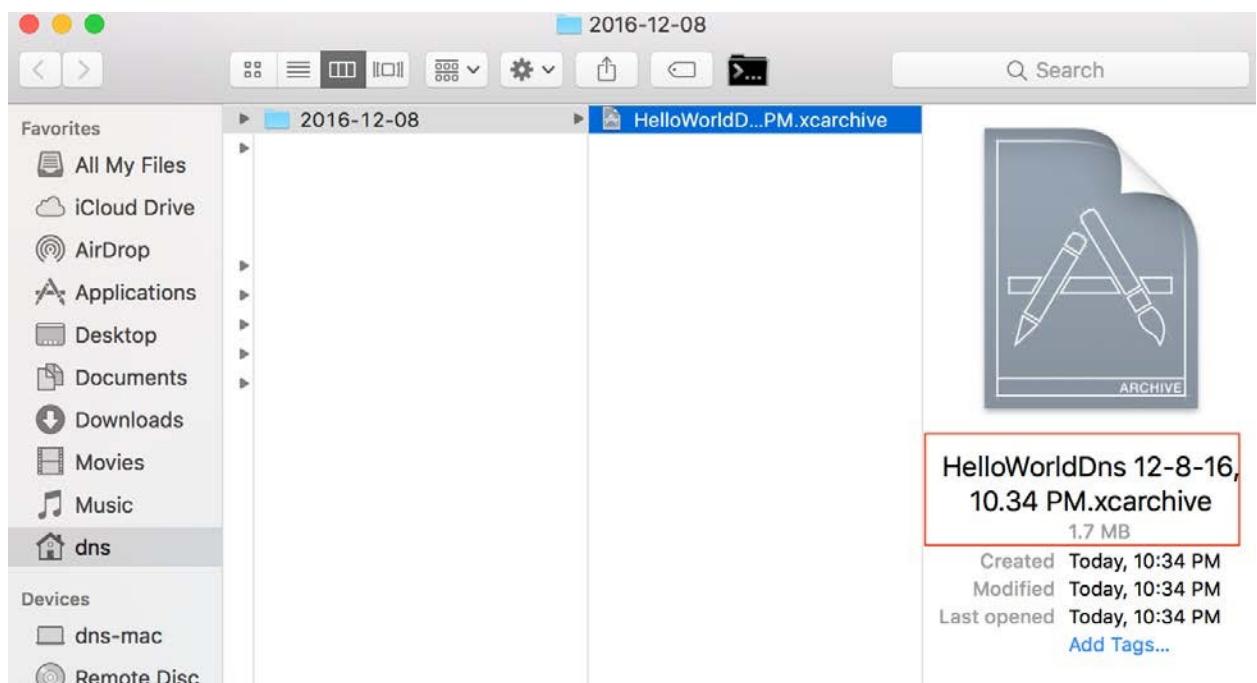
- No Xcode, faça login na conta correta. Defina o dispositivo de destino como "Dispositivo iOS genérico".



-
-
- Vá para o menu Product (Produto), na parte superior. Clique em Archive (Arquivar). Isso arquivará a compilação atual e fornecerá uma lista de arquivos no Organizer.
- Vá para Window -> Organizer (Janela -> Organizador). Observe que a exportação não funcionará, pois não há conta



- Clique com o botão direito do mouse no nome do arquivo e selecione "Mostrar no Finder".



6. Abra o terminal nesse local e digite o comando abaixo:
 - o `xcodebuild -exportArchive -exportFormat ipa -archivePath <path-to-application.xcarchive -exportPath ~/somepath/ipatobegenerated.ipa`

```

adding: Payload/HelloWorldDns.app/HelloWorldDns      (in=209520) (out=51399) (deflated 75%)
adding: Payload/HelloWorldDns.app/Info.plist   (in=1133) (out=738) (deflated 35%)
adding: Payload/HelloWorldDns.app/PkgInfo        (in=8) (out=8) (stored 0%)
total bytes=229221, compressed=62118 -> 73% savings
]
Results at '/var/folders/rg/y3w76kfn1hd_dnhm2pvh46_80000gn/T/CC3DFCA7-FAFC-4614-83A8-E7C91DAB3BAF-24347-0001E4509AF86870>HelloWorldDns.ipa'
Moving exported product to '/Users/dns/Library/Developer/Xcode/Archives/2016-12-08/dnsipa.ipa'
** EXPORT SUCCEEDED **

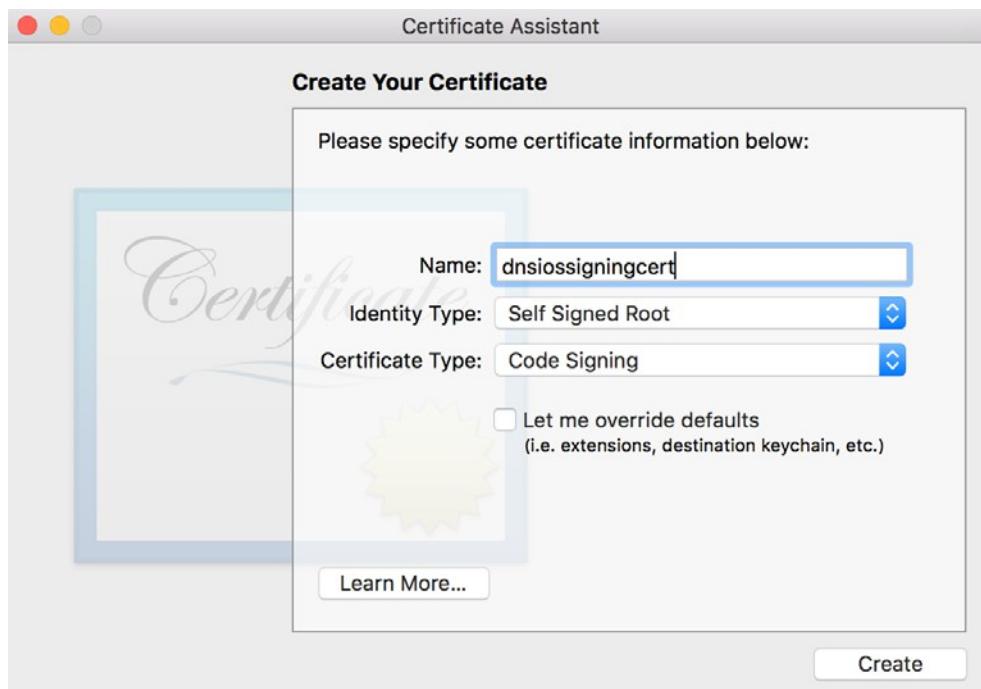
⇒ 2016-12-08 ls -al
total 152
drwxr-xr-x  5 dns  staff   170 Dec  8 22:42 .
drwxr-xr-x  4 dns  staff   136 Dec  8 22:35 ..
-rw-r--r--@  1 dns  staff  6148 Dec  8 22:35 .DS_Store
drwxr-xr-x  7 dns  staff   238 Dec  8 22:34 HelloWorldDns 12-8-16, 10.34 PM.xarchive
-rw-r--r--  1 dns  staff  66176 Dec  8 22:42 dnsipa.ipa
⇒ 2016-12-08

```

Esse arquivo ipa gerado pode ser usado para análise binária, mas, para instalá-lo em um dispositivo real, o aplicativo precisará ser assinado novamente.

Isso pode ser feito usando as etapas abaixo ou usando ferramentas como o Cydia Impactor, conforme explicado na "seção 4":

1. Verifique a assinatura atual usada para assinar o aplicativo usando o comando abaixo:
 - `codesign -v -d HelloWorldDns.app`
2. Crie uma assinatura autoassinada usando o Assistente de certificado no Keychain Access.
 - Escolha Acesso ao Keychain > Assistente de certificado > Criar um certificado.
 - Digite um nome para o certificado.
 - Defina o tipo de identidade como "Self Signing Root" e o tipo de certificado como "Code Sign".



- o Clique em Create (Criar).
 - o No Keychain Access, procure o certificado criado e copie-o para um local conhecido no laptop.
3. Modifique a assinatura do aplicativo usando o codesign.
- `codesign -v -fs "<abovecreatedcertificatename>" HelloWorldDns.app/`

```

➔ Payload cp dnsiossigningcert.cer dnsiossigningcert
➔ Payload codesign -v -fs "dnsiossigningcert" HelloWorldDns.app/
HelloWorldDns.app/: replacing existing signature
HelloWorldDns.app/: signed app bundle with Mach-O universal (armv7 arm64) [com.dns.HelloWorldDns]
➔ Payload

```

4. Renuncie ao aplicativo usando o ldid no binário dentro da pasta .app
 - o *ldid -s <nome do aplicativo>*
5. Escolha uma das etapas a seguir:
 - o Crie uma nova pasta chamada Payload. Mova a pasta .app para dentro dela e comprima a pasta Payload como Payload.zip. Renomeie Payload.zip para <applicationname>.ipa. O aplicativo pode então ser instalado usando as etapas mencionadas no "Módulo 4" (Usando o utilitário installipa)".
 - OU
 - o Copie o arquivo .app para o diretório /Applications (Aplicativos) no dispositivo. O aplicativo pode então ser instalado usando as etapas mencionadas no "Módulo 4" (Usando o .app).

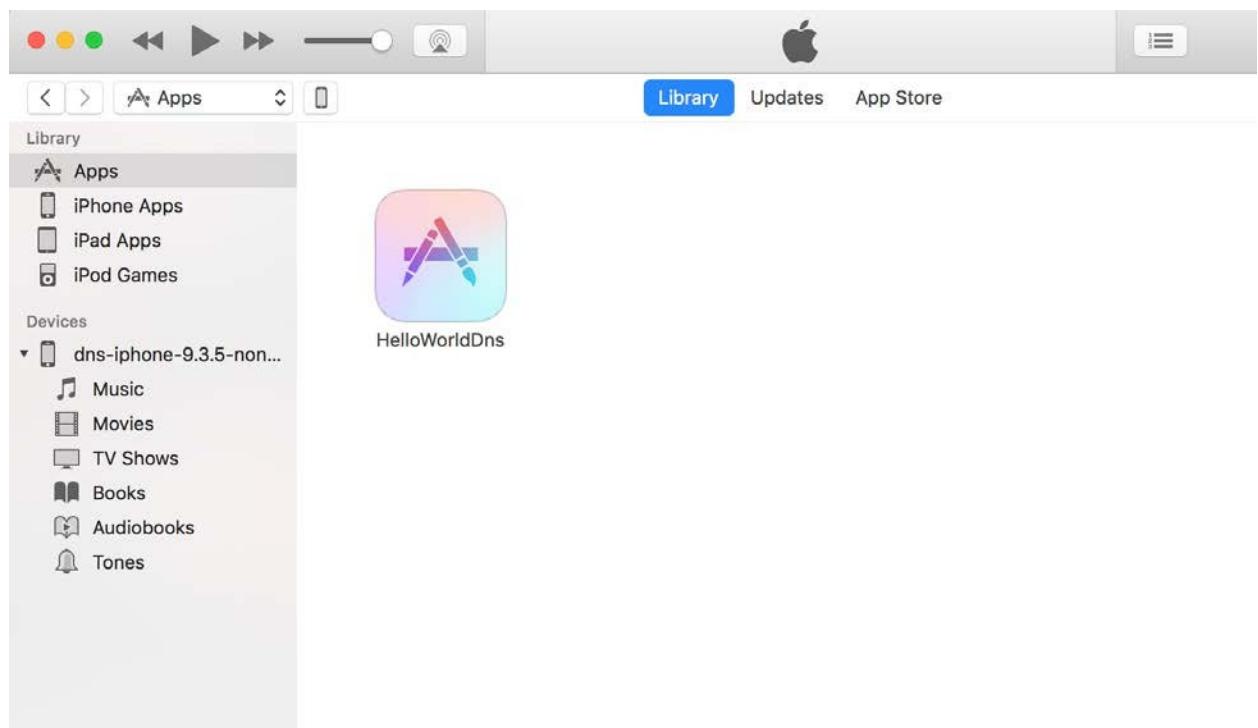
4. Instalação de binários do iOS em dispositivos físicos

Se o cliente fornecer binários do iOS, veja abaixo alguns dos métodos para instalá-los em um dispositivo físico.

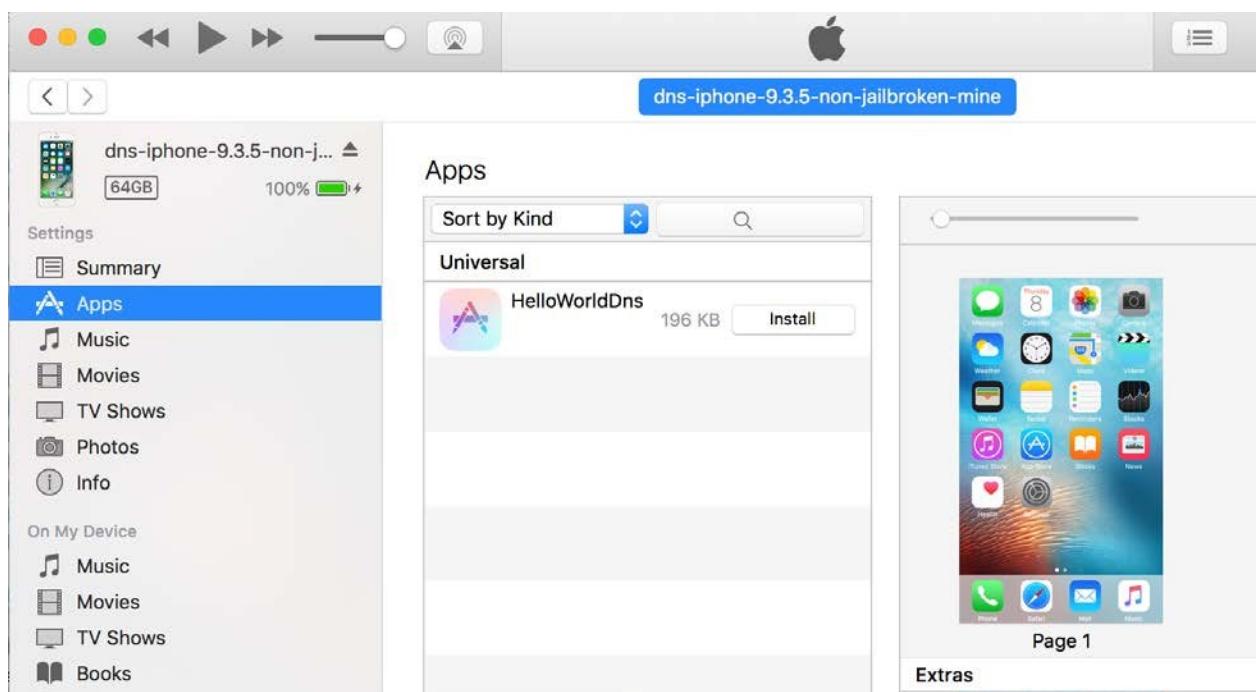
4.1 Método I - Usando o iTunes

As etapas abaixo podem ser usadas para instalar o aplicativo em um dispositivo depois que o acesso for concedido ao arquivo .IPA ou .app. Dependendo das circunstâncias, pode haver necessidade de um arquivo de provisão móvel separado, que é o certificado provisório para distribuição ad hoc do arquivo binário.

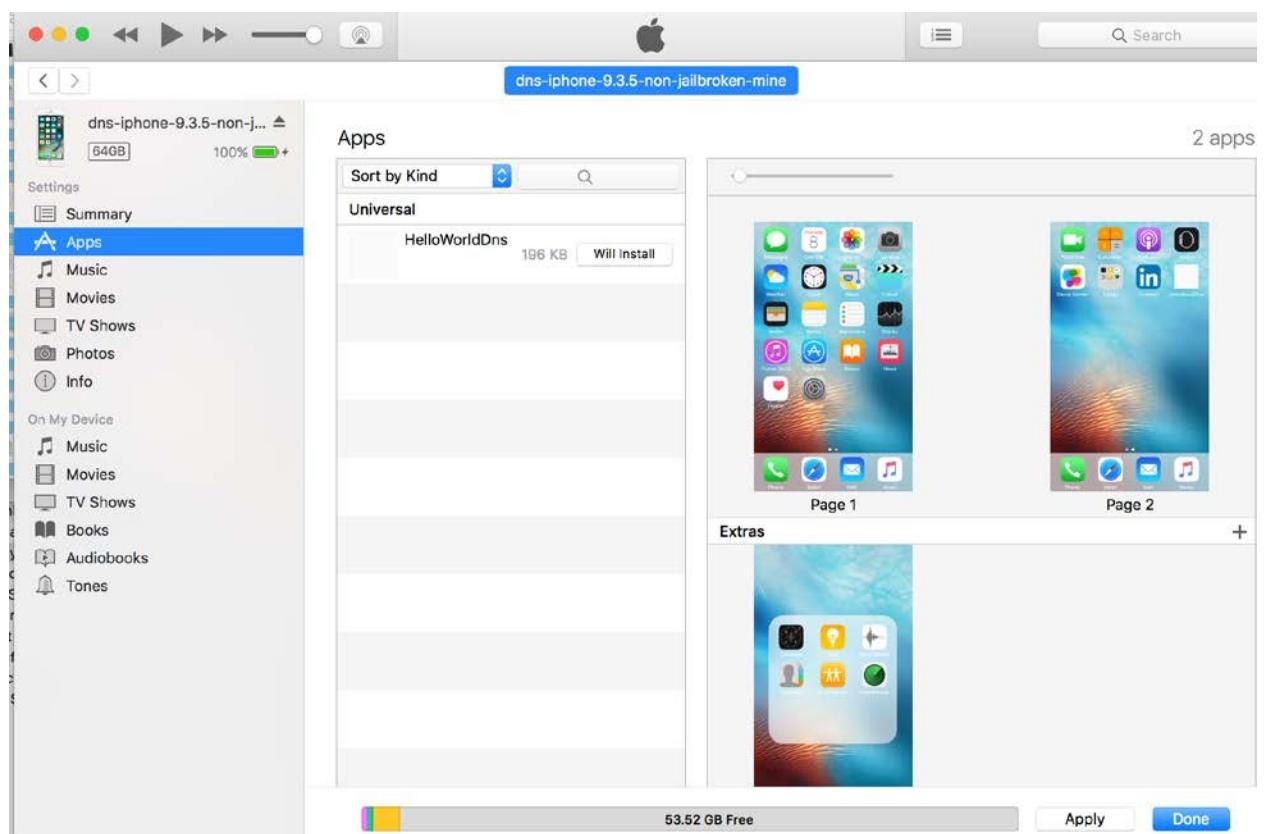
1. Iniciar o iTunes
2. Arraste e solte o arquivo .app/.ipa e o certificado provisório na guia "Apps" do iTunes na Biblioteca (não nos Apps do dispositivo). Se a guia "Apps" não estiver presente, siga as etapas abaixo.
 - a. No Mac: iTunes -> menu suspenso Música -> Clique no menu Editar -> verifique se Apps está selecionado ou não. Se não estiver, clique em Music Dropdown -> Clique em Edit Menu -> Enable Apps.
 - b. No Windows: iTunes -> Editar -> verifique se os aplicativos estão selecionados ou não.



3. Conecte o dispositivo iOS ao laptop e o nome do dispositivo iOS aparecerá na barra lateral. Clique nele e selecione "Apps" no menu do dispositivo iTunes. (Os aplicativos no dispositivo)



4. Selecione o aplicativo iOS a ser instalado, clique em Install (Instalar), Apply for the application to Sync (Aplicar para sincronização do aplicativo) e Install on the device (Instalar no dispositivo).



4.2 Método II - Usando o Cydia Impactor

Em um dispositivo sem jailbreak, o Cydia Impactor pode ser usado para instalar binários autoassinados do iOS e instalá-los no dispositivo.

1. Faça o download da ferramenta em cydiaimpactor.com
2. Faça o download do arquivo .deb ou .IPA
3. Instale a versão mais recente do iTunes
4. Conecte o dispositivo iOS ao laptop
5. Inicie o Impactor e arraste e solte o binário do iOS no menu suspenso com o nome do dispositivo
6. Faça login usando uma conta de desenvolvedor da Apple. Selecione o certificado Agent/iOS Distribution na lista.
 - o Uma conta gratuita também pode ser usada, mas o certificado expirará após 7 dias. Além disso, observe que um certificado de desenvolvedor iOS existente será revogado para dar lugar a esse novo certificado de dispositivo
7. Clique em OK no aviso para desenvolvedores da Apple
8. Em Settings > General > Profile & Device Management, localize o perfil usado para assinar o aplicativo e *confie* nele

4.3 Método III - Usando o iOS App Signer

Em um dispositivo sem jailbreak, o iOS App Signer pode ser usado para instalar binários autoassinados do iOS e instalá-los no dispositivo. O binário pode ser um arquivo .IPA ou um arquivo .deb.

Esse é um aplicativo relativamente simples. As etapas podem ser encontradas aqui:

- <http://dantheman827.github.io/ios-app-signer/>

4.4 Método IV - Instalação do arquivo .app

Em um dispositivo com jailbreak:

1. `scp -r HelloWorldApp.app/ root@10.0.1.24:/Applications/`

2. `cd /Aplicativos/HelloWorldApp.app/`
3. `chmod +x HelloWorldApp`
4. `uicache`

4.5 Método V - Instalação do binário modificado

Há várias maneiras de modificar/patchar um binário (veja mais informações posteriormente neste guia). Devido à assinatura de código, esses aplicativos não funcionarão como estão no dispositivo iOS. Siga as etapas abaixo para garantir que o binário modificado funcione.

1. Baixe o arquivo .app do dispositivo (use ipa -> ipa descriptografado -> arquivo app se estiver usando binário criptografado).
2. Extraia o conteúdo do arquivo .app e procure o binário do aplicativo nele contido.
3. Faça o patch do arquivo binário usando qualquer técnica.
4. Crie uma assinatura autoassinada usando o Assistente de certificado no Keychain Access.
 - o Escolha Acesso ao Keychain > Assistente de certificado > Criar um certificado.
 - o Digite um nome para o certificado.
 - o Defina o tipo de identidade como "Self Signing Root" e o tipo de certificado como "Code Sign".
 - o Clique em Create (Criar).
 - o No Keychain Access, procure o certificado criado e copie-o para um local conhecido no laptop.
5. Modifique a assinatura do aplicativo usando o codesign.
 - `codesign -v -fs "<abovcreatedcertificatename>" HelloWorldDns.app/`
6. Renuncie ao aplicativo usando o ldid no binário dentro da pasta .app
 - o `ldid -s <nome do aplicativo>`
7. Copie o arquivo .app modificado que ainda não foi convertido em um arquivo válido para o dispositivo usando o comando abaixo:
 - o `scp -r HelloWorldApp.app/ root@10.0.1.24:/Applications/`
8. Navegue até o diretório e execute os comandos abaixo para limpar o cache do dispositivo iOS.
 - o `cd /Aplicativos/HelloWorldApp.app/`
 - o `chmod +x HelloWorldApp`
 - o `uicache`
9. O aplicativo agora aparece no dispositivo iOS e pode ser usado sem problemas.

4.6 Método VI - Usando o utilitário Installipa

Em um dispositivo com jailbreak, o Installipa Utility pode ser usado para instalar binários autoassinados do iOS como usuário "mobile" ou "root". O Installipa pode ser baixado do Cydia.

Copie o arquivo .IPA no dispositivo usando o ssh. Com o AppSync instalado no dispositivo usando o Cydia, entre no dispositivo por ssh como usuário "mobile" e use o comando abaixo para instalar o aplicativo como usuário "mobile":

- *instalar HelloWorldApp.ipa*

4.7 Método VII - Usando o utilitário de configuração do iPhone

Para instalar o arquivo binário .IPA, use o utilitário de configuração do iPhone (agora renomeado como Apple Configurator e disponível para download na Mac AppStore) aqui:

- https://www.theiphonewiki.com/wiki/IPhone_Configuration.Utility

O AppSync deve ser instalado em um iPhone via Cydia para que a maioria dos arquivos .IPA seja instalada no dispositivo. O Appsnc Unified é um software que permite a instalação de arquivos IPA assinados falsos no dispositivo.

Se o AppSync não estiver instalado, adicione <http://cydia.angelxwind.net> como um repositório e procure o AppSync Unified.

Depois que a ferramenta estiver instalada em um MAC e em um iPhone, inicie a ferramenta no MAC e adicione o arquivo IPA para instalar no Apple Configurator.

4.8 Método VIII - Usando o iFunBox

O iFunBox também pode ser usado para instalar um arquivo .IPA. Isso requer um dispositivo com jailbreak. Siga as etapas mencionadas aqui:

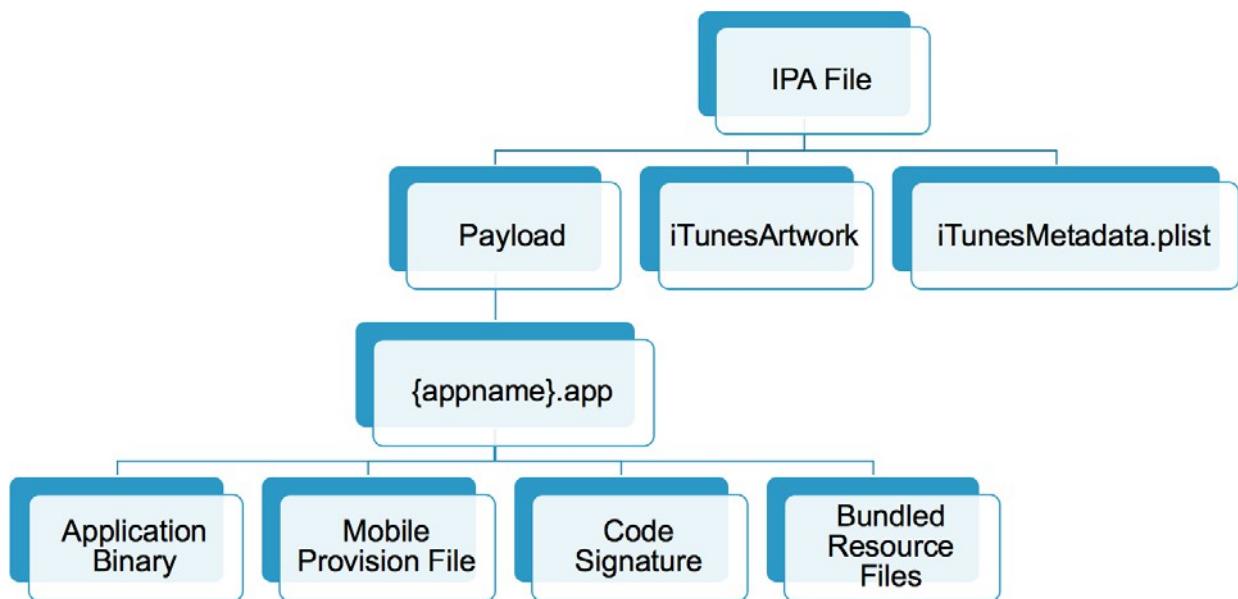
- <http://iosgeeksblog.blogspot.in/2013/01/how-to-install-ipa-files-directly-on-iphone-with-ifunbox.html>

5. Guia do pacote binário do iOS

Os aplicativos iOS têm um formato de arquivo binário conhecido como IPA, que são basicamente arquivos ZIP. Os arquivos .IPA incluem um binário para a arquitetura ARM e só podem ser instalados em um dispositivo iOS. Não há maneiras conhecidas de instalar o arquivo .IPA em um simulador de iOS.

Os arquivos .IPA podem ser descompactados usando um utilitário de descompactação.

5.1 Entendendo a estrutura do pacote binário do iOS



iTunesArtwork: Uma imagem PNG de 512 x 512 pixels. Ela contém o ícone dos aplicativos que aparece no iTunes e no aplicativo App Store no dispositivo iOS.

iTunesMetadata.plist: Um arquivo xml de lista de propriedades que contém informações do desenvolvedor, como nome do desenvolvedor, ID, informações de direitos autorais, nome do aplicativo, informações de lançamento, etc.

Carga útil: A pasta que contém os dados do aplicativo.

Binário do aplicativo: O arquivo executável que contém o código do aplicativo. O nome desse arquivo é sempre o mesmo que o nome real do aplicativo, sem a extensão .app. Durante o pentest, a análise binária completa é realizada nesse binário do aplicativo.

Arquivo de provisionamento móvel: por padrão, os aplicativos no iOS só podem ser instalados por meio da AppStore. Em casos especiais, quando o aplicativo deve ser testado na versão beta, são gerados e usados certificados de provisão móvel. Esse é o arquivo que é incluído no binário quando a distribuição ad hoc do arquivo deve ser feita. Um perfil de provisão é um documento que lista os certificados digitais, os dispositivos e as IDs dos aplicativos autorizados a operar em um dispositivo. Ele é usado especificamente para estágios beta (geralmente denominado *Ad_Hoc_Distribution_Profile.mobileprovision*).

Para obter mais informações, consulte: <http://www.wikihow.com/Install-Ad-hoc-iPhone-OS-Apps>

Assinatura de código: A finalidade da assinatura de código é garantir que a integridade do arquivo .app seja mantida desde quando o aplicativo foi lançado. Qualquer tipo de edição ou exclusão (mesmo que as imagens tenham o mesmo nome) invalidará a assinatura. Todas as alterações feitas no arquivo .app exigem que o pacote inteiro seja assinado novamente.

Arquivos de recursos incluídos no pacote: Imagens, vídeos, sons, HTML, arquivos de lista de propriedades, etc., que são necessários para que o aplicativo seja instalado no dispositivo móvel.

Para obter mais informações sobre a estrutura do pacote de aplicativos iOS, acesse aqui

- https://developer.apple.com/library/content/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#/apple_ref/doc/uid/10000123i-CH101-SW1.

5.2 Compreensão das arquiteturas compatíveis com o aplicativo fornecido

O Lipo é um utilitário para Mac que pode ser usado para visualizar todas as arquiteturas nas quais o aplicativo fornecido pode ser instalado. Talvez seja necessário instalar o Lipo em um Mac ao executá-lo pela primeira vez. A sintaxe para usar o lipo é mostrada abaixo:

```
lipo -info <applicationbinary>
```

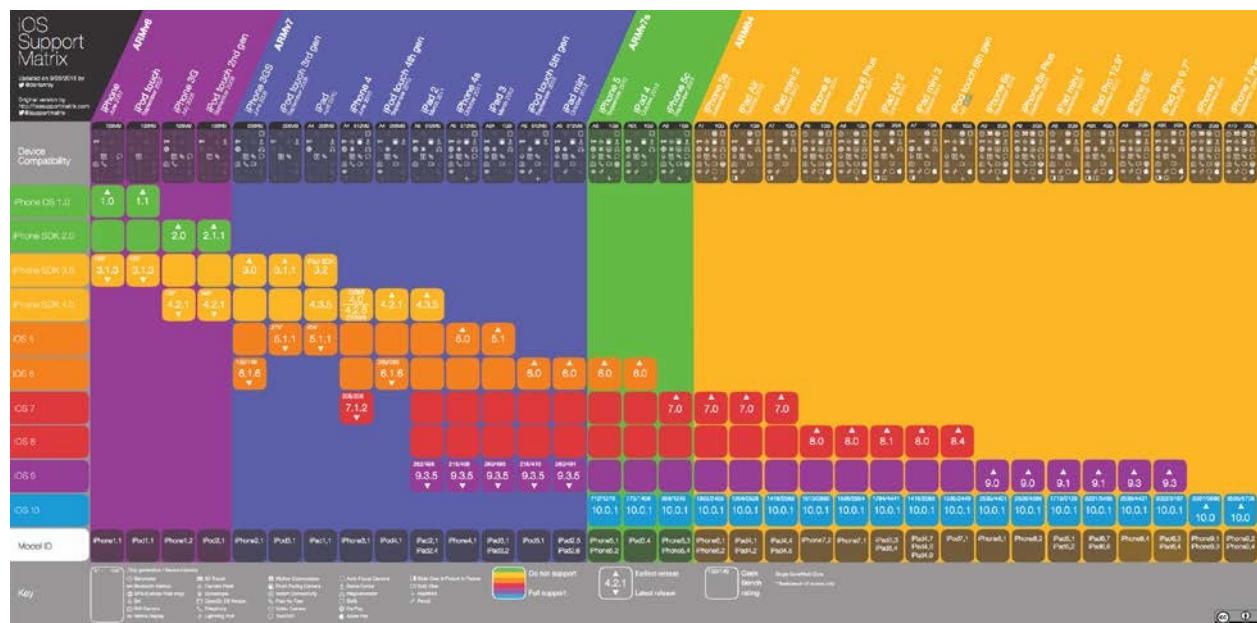
Lembre-se de executar o *lipo* no binário *dentro* da pasta .app, não no arquivo IPA.



```
● ○ ● HelloWorldDns — dns@dns-mac — ..oWorldDns.app — -zsh — 80x24
Last login: Sat Dec 10 16:31:53 on ttys005
→ HelloWorldDns.app ls
Base.lproj                               _CodeSignature
HelloWorldDns                           archived-expanded entitlements.xcent
Info.plist                                embedded.mobileprovision
PkgInfo
→ HelloWorldDns.app lipo -info HelloWorldDns
Architectures in the fat file: HelloWorldDns are: armv7 arm64
→ HelloWorldDns.app
```

5.3 Entendendo a arquitetura disponível nos dispositivos de teste

A imagem abaixo ilustra a matriz de suporte do iOS. Uma versão mais detalhada pode ser encontrada aqui:
<http://iossupportmatrix.com/>



Fonte da imagem: http://dorianroy.com/blog/wp-content/uploads/2016/09/iOS_Support_Matrix_v4.2.pdf

Uma versão somente de texto dessa matriz de suporte pode ser encontrada aqui:

<https://www.innerfence.com/howto/apple-ios-devices-dates-versions-instruction-sets>

Os identificadores de dispositivos iOS podem ser encontrados aqui: <https://www.theiphonewiki.com/wiki/Models>

A versão máxima do iOS para cada dispositivo iOS pode ser encontrada aqui:

<http://www.everyi.com/by-capability/maximum-supported-ios-version-for-ipod-iphone-ipad.html>

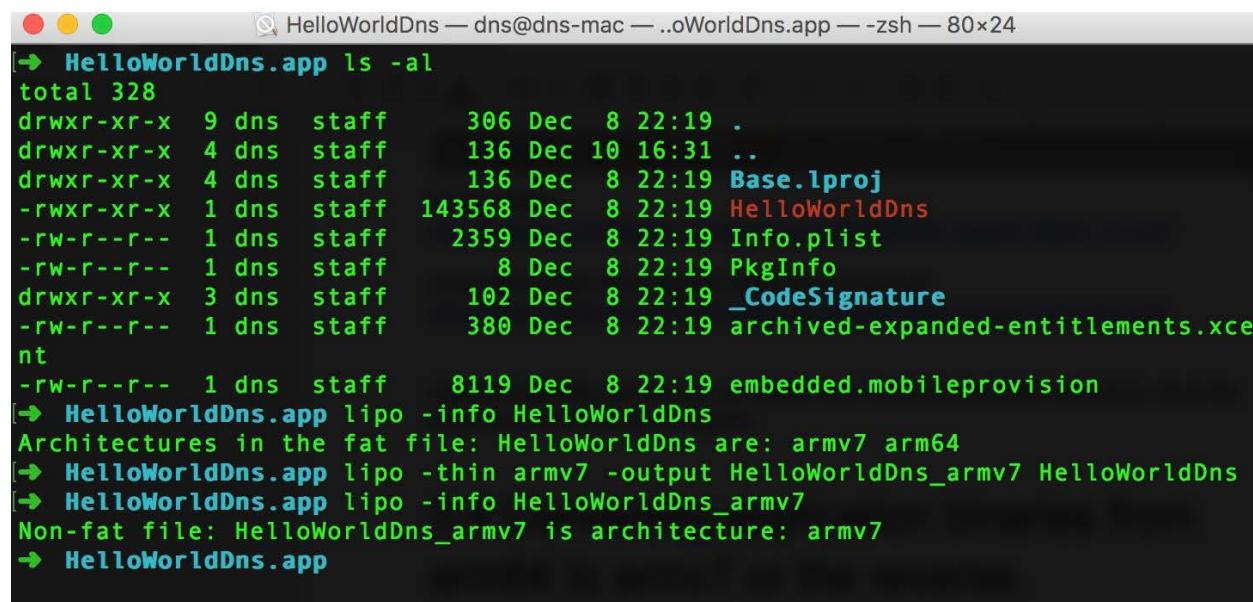
Os usuários com acesso jailbroken podem executar `uname -a` no terminal SSH local do dispositivo para verificar a arquitetura que o dispositivo suporta.

5.4 Conversão de binários de aplicativos de binário FAT para binário de arquitetura específica

Muitas das ferramentas disponíveis não são compatíveis com binários AArch64. Para analisar esses binários de 64 bits, remova uma arquitetura específica do binário fat.

O Lipo pode ser usado para remover uma arquitetura específica (64 bits - `arm64`) do binário fornecido. Use o comando abaixo:

```
lipo -thin armv7 -output <newarmv7binaryname> <binaryname>
```



```
HelloWorldDns — dns@dns-mac — ..oWorldDns.app — -zsh — 80x24
→ HelloWorldDns.app ls -al
total 328
drwxr-xr-x  9 dns  staff      306 Dec  8 22:19 .
drwxr-xr-x  4 dns  staff      136 Dec 10 16:31 ..
drwxr-xr-x  4 dns  staff      136 Dec  8 22:19 Base.lproj
-rw-r--r--  1 dns  staff  143568 Dec  8 22:19 HelloWorldDns
-rw-r--r--  1 dns  staff    2359 Dec  8 22:19 Info.plist
-rw-r--r--  1 dns  staff      8 Dec  8 22:19 PkgInfo
drwxr-xr-x  3 dns  staff     102 Dec  8 22:19 _CodeSignature
-rw-r--r--  1 dns  staff    380 Dec  8 22:19 archived-expanded entitlements.xcent
-rw-r--r--  1 dns  staff   8119 Dec  8 22:19 embedded.mobileprovision
→ HelloWorldDns.app lipo -info HelloWorldDns
Architectures in the fat file: HelloWorldDns are: armv7 arm64
→ HelloWorldDns.app lipo -thin armv7 -output HelloWorldDns_armv7 HelloWorldDns
→ HelloWorldDns.app lipo -info HelloWorldDns_armv7
Non-fat file: HelloWorldDns_armv7 is architecture: armv7
→ HelloWorldDns.app
```

O aplicativo pode então ser reempacotado em um arquivo .IPA e instalado no dispositivo usando as etapas mencionadas no "Módulo 4 - Método VI" (Usando o utilitário `installipa`).

5.5 Conversão de executáveis pré-iOS 9 em um executável iOS 9

Use essa ferramenta para converter executáveis pré-iOS 9 em um executável iOS 9:

- <https://github.com/Starwarsfan2099/iOS-9-Executable-Converter>

5.6 Conversão de aplicativos de 32 bits em aplicativos de 64 bits no Xcode

Esse método pode ser usado para executar aplicativos de 32 bits em um dispositivo de 64 bits, mesmo com apenas um binário de 32 bits. Com acesso ao código-fonte, siga as etapas abaixo:

1. Abra o código-fonte do aplicativo no Xcode
2. Atualize as configurações do projeto para suportar a versão mais recente do iOS disponível
3. Em Build Settings, vá para a seção Architectures e defina Architectures como "Standard architectures (arm64)"
4. Corrija todos os avisos do compilador que foram anulados usando as etapas mencionadas aqui:<http://www.chupamobile.com/blog/2015/01/19/convert-app-64-bit-requirement/> e https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaTouch_64BitGuide/ConvertingYourAppto64-Bit/ConvertingYourAppto64-Bit.html
5. Execute o código do projeto atualizado em um simulador de iOS de 64 bits para garantir que o aplicativo funcione.

6. Compilação do código-fonte fornecido pelo cliente para pentesting no iOS mais recente usando o Xcode

Para testes de penetração assistidos por código, a melhor abordagem é obter o código-fonte do cliente e configurá-lo para ser executado localmente. Isso oferece recursos adicionais de depuração que podem ser muito úteis a longo prazo. Durante a chamada inicial com um cliente, solicite um "*pacote de código pronto para ser construído com todas as dependências*". Isso evitárá a falta de bibliotecas e outros problemas de dependência que podem surgir se a equipe de desenvolvimento apenas copiar uma pasta.

Mesmo com um código-fonte pronto para compilação, pode haver problemas para colocar o aplicativo em funcionamento.

Suponha que um cliente tenha fornecido o código-fonte do DVIA (aplicativo Damn Vulnerable iOS) para um teste de penetração assistido por código.

As etapas abaixo pressupõem que o Xcode 8.x está sendo usado (o Xcode mais recente disponível no momento da publicação deste guia). O dispositivo de destino para teste pode ser o iOS 9.x ou superior.

6.1 Faça o download do código-fonte

Faça o download do código-fonte do DVIA em <https://github.com/prateek147/DVIA> usando este comando:

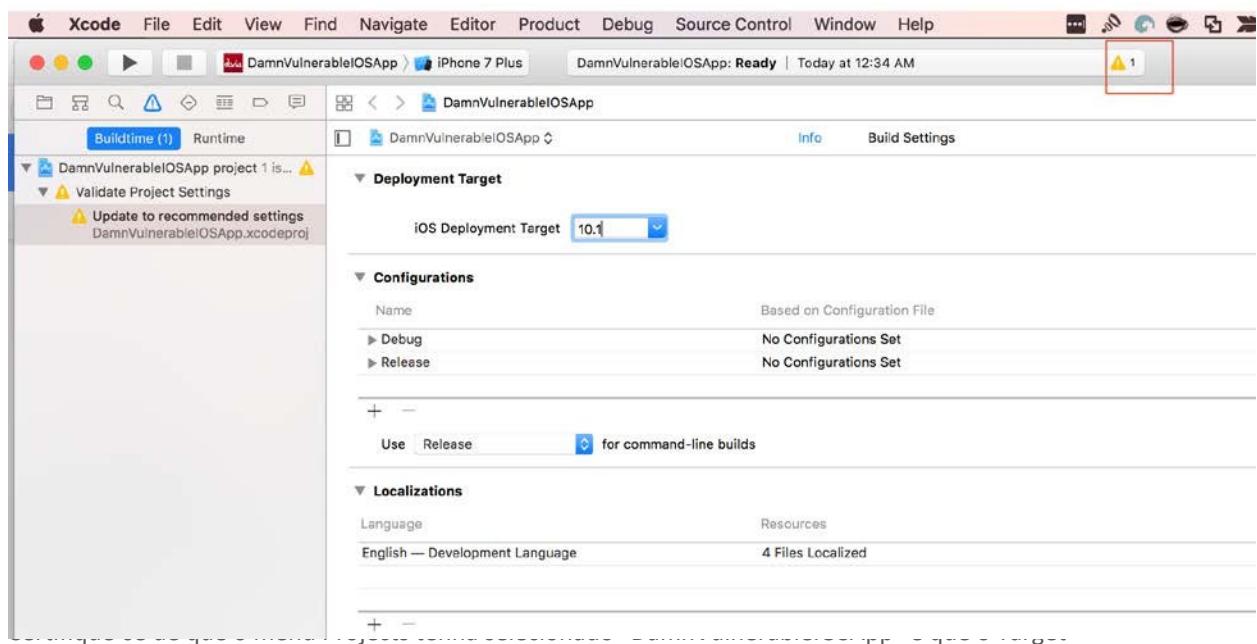
```
git clone https://github.com/prateek147/DVIA.git
```

6.2 Iniciar o espaço de trabalho

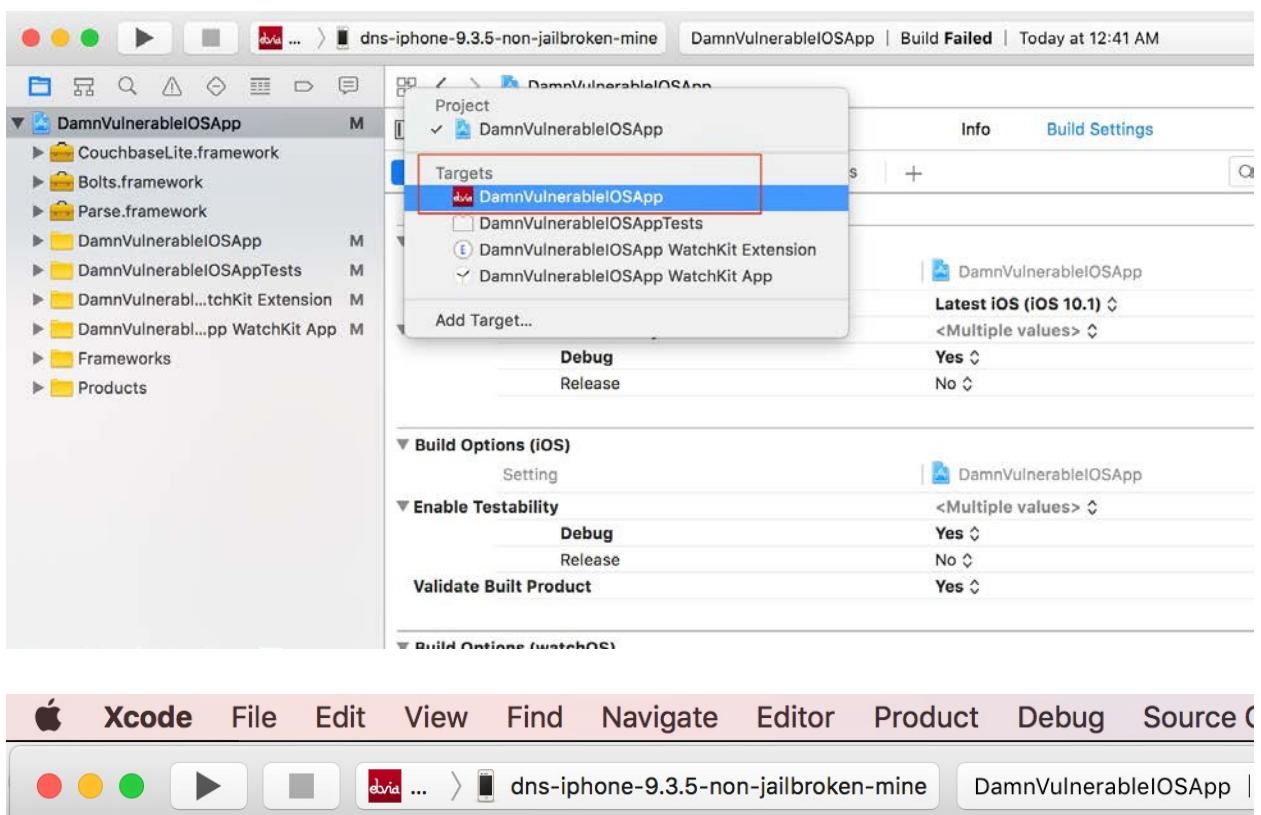
A partir do código-fonte baixado, abra o arquivo. xcworkspace usando o Xcode. Permita que o processo de "Indexação" seja concluído.

6.3 Configuração de aplicativos

1. Altere o destino da implantação do iOS 8.2 para a versão do iOS do dispositivo e clique no sinal de aviso amarelo, conforme mostrado aqui:



2. Esteja definido como o dispositivo físico que está conectado ao laptop.



3. Clique em General (Geral) e defina o Bundle Identifier (Identificador do pacote) como um valor exclusivo que ainda não esteja registrado em nenhuma das outras contas de desenvolvedor.
4. Na seção Assinatura, ative "Gerenciar automaticamente a assinatura".
A ativação da opção "Gerenciar assinatura automaticamente" redefine as configurações de compilação de assinatura.
5. Altere o nome da equipe para a conta de desenvolvedor.
6. Repita as etapas 4 e 5 para todos os outros alvos, como extensões do WatchKit, etc.

7. Se o aplicativo Apple Watch não for testado, desative o grupo de aplicativos AppleWatch. Navegue até o menu Recursos e, em Grupos de aplicativos, desmarque "group.dvia.applewatch". Adicione um novo grupo "group.dns.dvia".
8. Repita a etapa 7 para todos os outros alvos, como DamnVulnerableiOSApp, etc.
9. No destino do aplicativo WatchKit, remova os binários do WatchKit em General -> Embedded Binaries (Geral -> Binários incorporados).
10. Nas extensões do Watchkit, renomeie *com.highaltitudehacks.dvia* para *com.dns.dvia* com base no nome do grupo
11. No código-fonte, procure referências do identificador do pacote original "com.highaltitudehacks.dvia" usando o comando abaixo
ack -i com.highaltitudehacks.dvia

```
→ tempApplications ack -i com.highaltitudehacks.dvia
DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit App/Info.plist
31:      <string>com.highaltitudehacks.dvia</string>

DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit Extension/Info.plist
30:              <string>com.highaltitudehacks.dvia.watchkitapp</string>

DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp.xcodeproj/project.pbxproj
2364:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia-dns";
2398:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia-dns";
2479:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2505:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2533:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2556:                      PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
→ tempApplications
```

Referência atualizada com "com.highaltitudehacks.dvia-dns"

Em nosso caso, abra "DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit App/Info.plist" e substitua "com.highaltitudehacks.dvia" por "com.highaltitudehacks.dvia-dns"

Abra "DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit Extension/Info.plist" e substitua o conteúdo pelo nome do grupo - *com.dns.dvia.watchkitapp*

12. Nas configurações de compilação de todos os alvos, defina *ativar código de bits = não*
13. Se houver links físicos no caminho de pesquisa da estrutura, remova-os.
14. (Se estiver usando o POD) Faça um "pod init"
15. (Se estiver usando o POD) Adicione todos os pods necessários no Podfile e faça uma "instalação de pod"

16. (Se estiver usando o POD) Remova todos os itens que foram adicionados no Podfile das estruturas gerais -> vinculadas
17. Remova *-ObjC* de "Other Linker Flags" e adicione *\$(inherited)*
18. Mantenha apenas *\$(inherited)* em todo o "caminho de pesquisa"
19. Reinicie o espaço de trabalho no Xcode e não o arquivo de projeto
20. Um problema de conexão de rede significa que o aplicativo não tem o ATS desativado. Para desativá-lo, defina o código abaixo em DamnVulnerableIOSApp-Info.plist antes do último dict.

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key> <true/>
</dict>
```

21. Certifique-se de que o dispositivo iOS esteja registrado em uma conta de desenvolvedor usando as etapas mencionadas aqui:
https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#/apple_ref/doc/uid/TP40012582-CH30-SW10

7. Cartilha do modelo de segurança do iOS

Veja a seguir alguns recursos importantes sobre o modelo de segurança do iOS.

7.1 Recursos de segurança

1. Os aplicativos precisam ser assinados com um certificado de desenvolvedor Apple pago.
2. Os binários do aplicativo são criptografados usando FairPlayDRM, semelhante ao que é usado no iTunes Music.
3. Os aplicativos são protegidos por assinatura de código.
4. Aplicativos corrigidos não podem ser instalados em dispositivos sem jailbreak.
5. Todo aplicativo iOS é executado em sua própria sandbox. Após o iOS 8.3+, esses dados de sandbox não podem ser acessados sem a necessidade de fazer o jailbreak do dispositivo iOS.
6. Nenhum aplicativo pode acessar dados pertencentes a outro aplicativo. Os manipuladores de protocolo, como esquemas de URL, são a única forma de comunicação entre aplicativos a ser usada para a passagem de mensagens entre aplicativos. Os dados também podem ser armazenados em chaveiros.
7. Sempre que novos arquivos são criados no dispositivo iOS, são atribuídas a eles classes de proteção de dados, conforme especificado pelos desenvolvedores. Isso ajuda a impor restrições de acesso a esses arquivos.
8. Os aplicativos precisam solicitar especificamente a permissão do usuário para acessar recursos como câmera, mapas, contatos etc.
9. Os dispositivos iOS 5s+ têm um componente de hardware seguro chamado Secure Enclave. Ele é uma versão altamente otimizada do TrustZone da ARM e impede que o processador principal acesse diretamente dados confidenciais. Mais detalhes sobre o Secure Enclave podem ser encontrados aqui:
 - <https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf>

Uma descrição detalhada dos recursos de segurança disponíveis no iOS pode ser encontrada aqui:

- https://www.apple.com/business/docs/iOS_Security_Guide.pdf
- <https://support.apple.com/en-us/HT207143>
- <https://www.apple.com/support/security/>

8. Explorando o sistema de arquivos do iOS

O sistema de arquivos do iOS pode ser acessado tanto em telefones com jailbreak quanto em telefones sem jailbreak. A quantidade de acesso varia. A partir do iOS 8.3+, a sandbox do aplicativo só pode ser acessada em um telefone com jailbreak. A classificação de risco para essa vulnerabilidade deve ser explicada aos clientes para que eles possam tomar uma decisão informada.

Se houver dificuldade para ler os diretórios de arquivos (exceto os diretórios de mídia e de aplicativos no dispositivo iOS) mesmo após o jailbreak, verifique se o AFC2 (Apple File Conduit2) está instalado no Cydia (<https://cydia.saurik.com/package/com.saurik.afc2d/>).

8.1 Leitura de dados usando o iExplorer

O iExplorer é uma das ferramentas mais simples para visualizar o uso da estrutura de arquivos do iOS, mesmo sem um dispositivo com jailbreak. Antes do iOS 8.3, a sandbox do aplicativo e seu conteúdo eram diretamente visíveis com o iExplorer. A partir do iOS

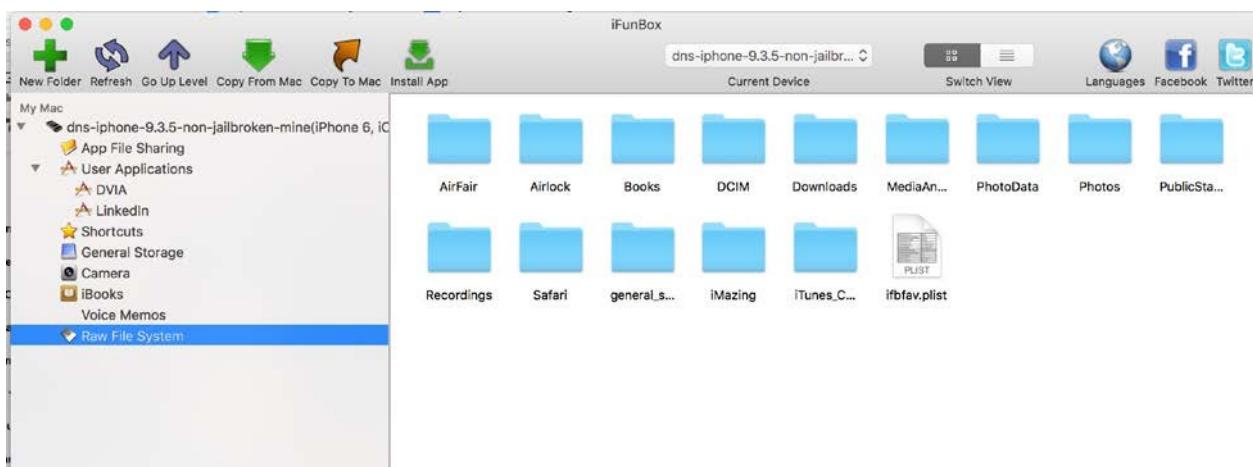
8.3+, a sandbox de aplicativos e os diretórios do dispositivo raiz são acessíveis somente após o jailbreak.

O utilitário iExplorer instalado pode ser baixado aqui: <https://macroplant.com/downloads> Conectar o dispositivo a um laptop e iniciar o iExplorer deve permitir a leitura do sistema de arquivos do dispositivo.

8.2 Leitura de dados usando o iFunBox

O iFunBox é outro aplicativo que pode ser usado para acessar o sistema de arquivos do iOS. Antes do iOS 8.3, a sandbox do aplicativo e seu conteúdo eram diretamente visíveis usando o iFunBox.

Os seguintes diretórios podem ser acessados pelo iFunBox.



A partir do iOS 8.3+, a sandbox de aplicativos e os diretórios raiz do dispositivo só podem ser acessados após o jailbreak.



O iFunBox também pode ser usado para instalar aplicativos iOS no dispositivo usando o recurso "Install App". Aplicativos crackeados ou aplicativos sem um certificado de fornecimento exigem um dispositivo com jailbreak.

Aplicativos legítimos podem ser instalados diretamente usando o método do iTunes mencionado em "4.1 Método 1 - Uso do iTunes".

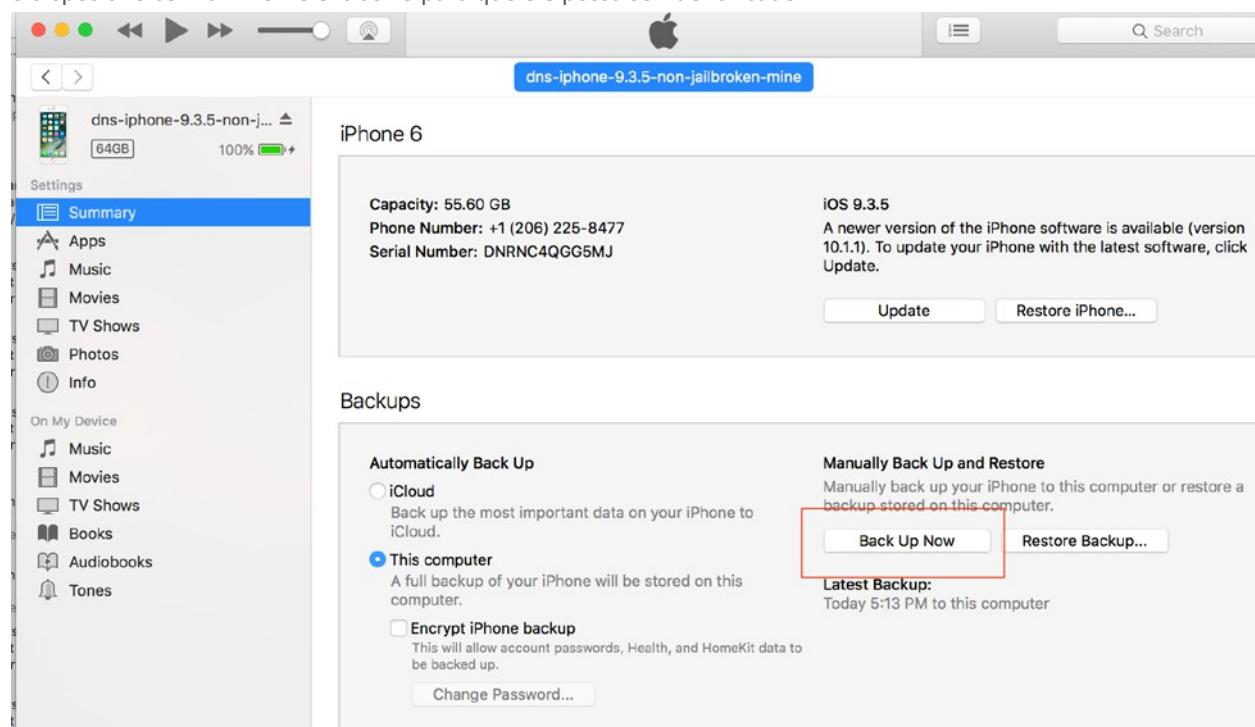
8.3 Leitura dos dados da caixa de areia do aplicativo iOS > 8.3 usando o método de backup

Os dados da sandbox do aplicativo para iOS > 8.3 não são permitidos pela Apple, mas há uma solução alternativa que pode ser usada para acessar esses dados. Antes de prosseguir, faça um backup do dispositivo e dos dados do aplicativo.

8.3.1 Como fazer backup do iDevice

Método 1: iTunes

O backup do dispositivo pode ser feito usando o iTunes, conforme mostrado abaixo. Não se esqueça de renomear o dispositivo com um nome exclusivo para que ele possa ser identificado.



Método 2: Libimobiledevice

Como alternativa, o backup pode ser realizado usando o utilitário *idevicebackup2* que pode ser instalado a partir da biblioteca libimobiledevice. A instalação é feita da seguinte forma - *brew install libimobiledevice*.

O uid do utilitário pode ser encontrado usando "*idevice_id -l*"

Pode ser necessário executar `sudo chmod -R 777 /var/db/lockdown` antes de fazer o backup dos

dados. Use o comando abaixo para fazer o backup do conteúdo:

```
idevicebackup2 backup --full --source <deviceuid> --udid <deviceuid> ~/Documents
```

Método 3: ferramentas de terceiros

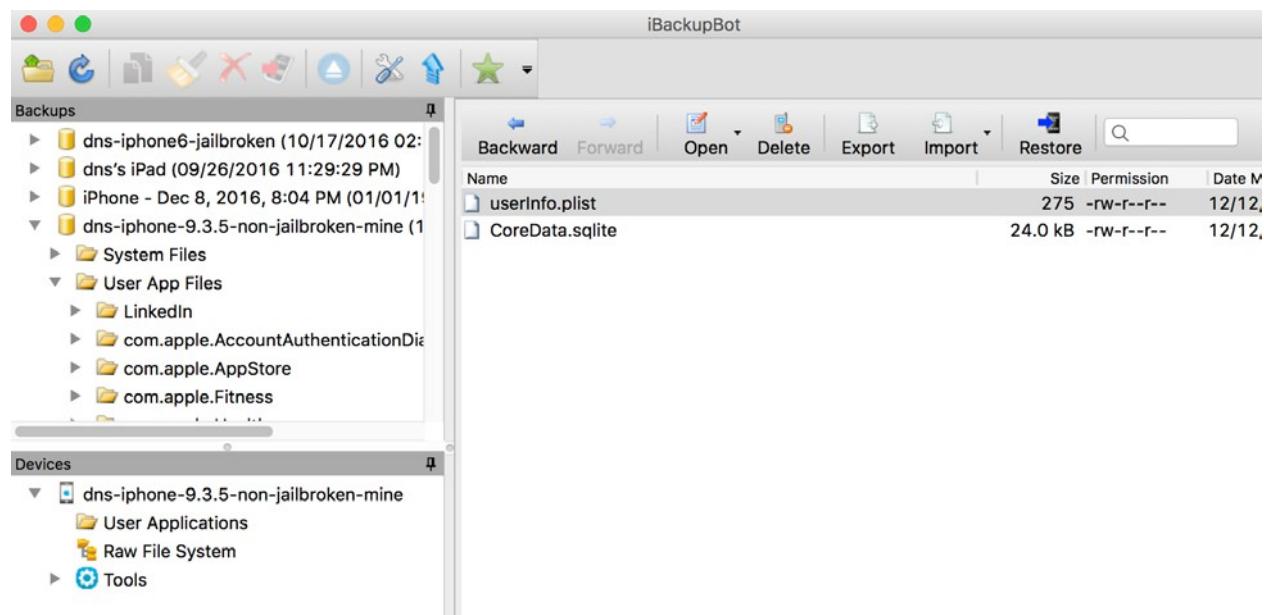
O backup do dispositivo também pode ser feito usando o conteúdo do iCloud. Ferramentas como o iLoot (<https://github.com/hackappcom/iloot>) podem ser usadas. Para que o backup funcione, são necessárias credenciais de login legítimas vinculadas ao dispositivo de destino.

Use o comando abaixo para fazer um backup do conteúdo do iCloud:

```
python iloot.py iCloudemailaddress iCloudpassword
```

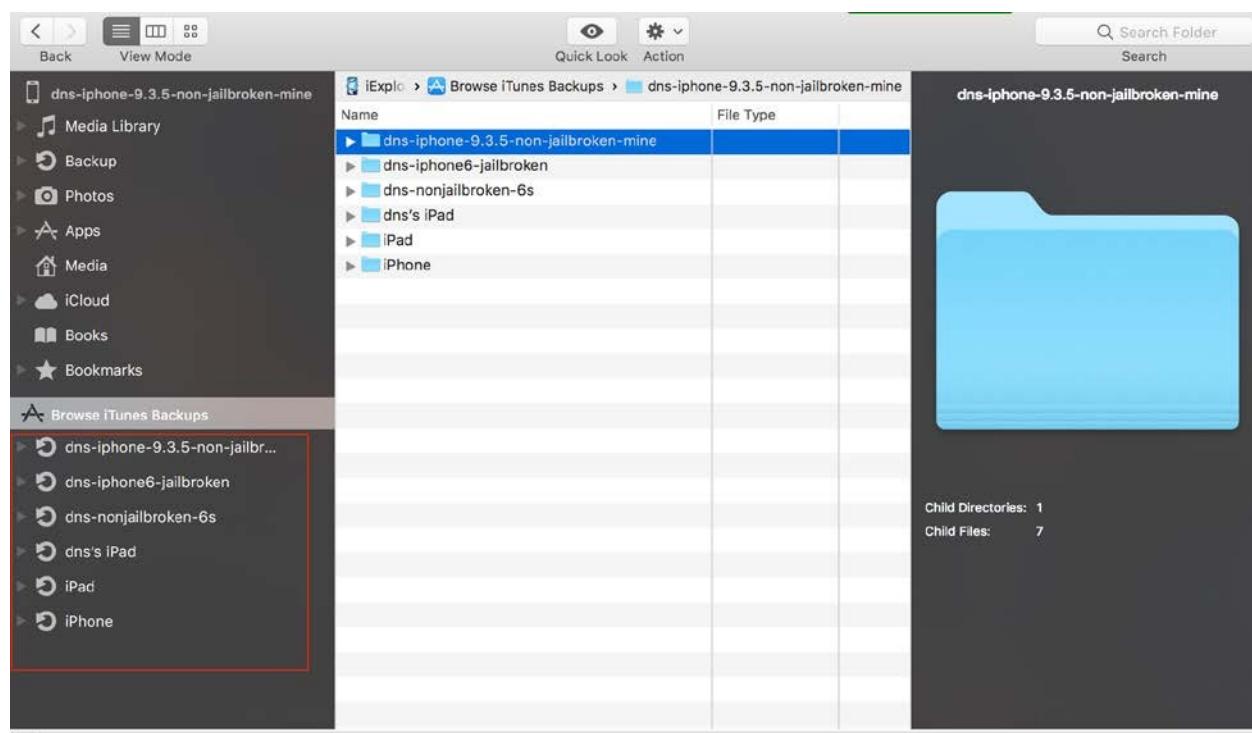
Após a conclusão do backup, ferramentas como o iExplorer ou o iBackupBot podem ser usadas para visualizar os dados da área restrita do aplicativo.

8.3.2 Usando o iBackupBot

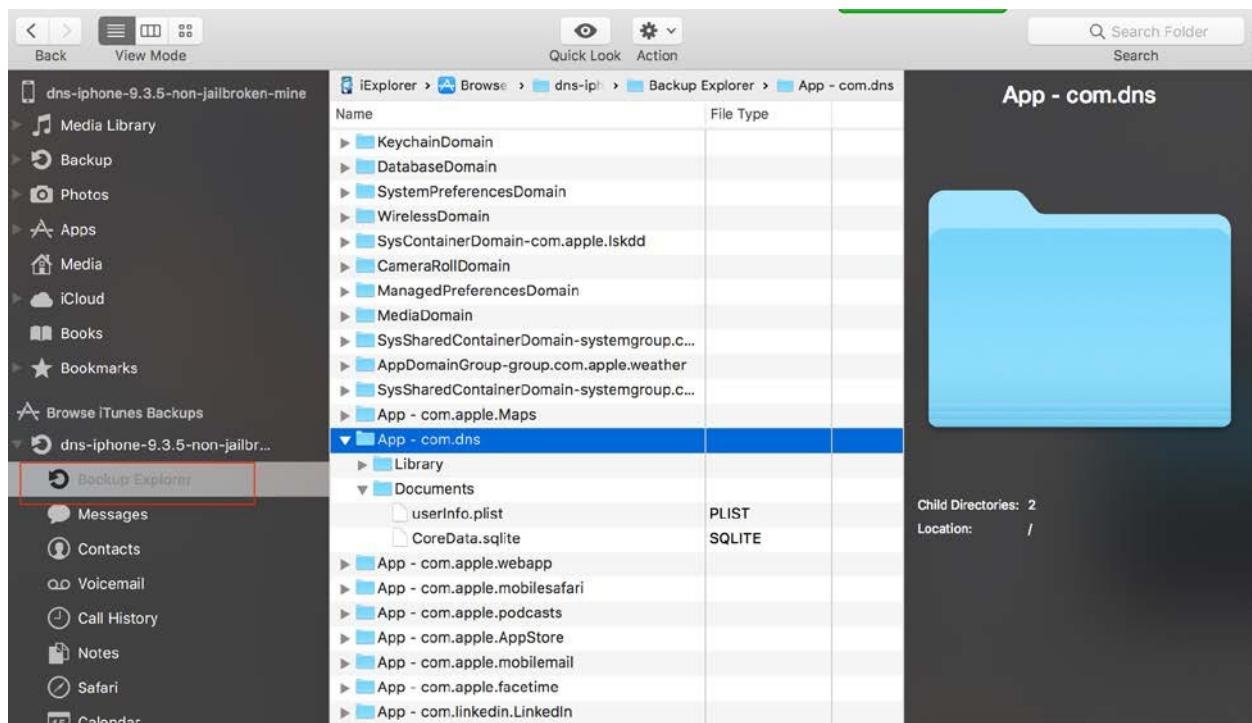


8.3.3 Usando o iExplorer

Inicie o iExplorer, localize os nomes dos dispositivos com backup e clique em Backup Explorer para visualizar o conteúdo, conforme mostrado abaixo:



Observe que o backup da sandbox do aplicativo é feito e os dados do aplicativo podem ser visualizados na pasta Backup Explorer.



8.4 Leitura de dados de aplicativos usando o OpenSSH

O OpenSSH é um dos primeiros aplicativos a ser instalado em um dispositivo com jailbreak para acessar o conteúdo do sistema de arquivos do iOS. O OpenSSH pode ser instalado a partir do Cydia.

Para instalar o OpenSSH, navegue até o Cydia, procure o OpenSSH e clique em Install (Instalar).

As credenciais padrão para o servidor SSH hospedado pelo OpenSSH localmente no dispositivo são "**alpine**" (alpine era, na verdade, o codinome do iOS 1.0).

Depois de instalar o OpenSSH, escolha um utilitário para acessar o sistema de arquivos do iOS. Para obter uma GUI do sistema de arquivos, use o FileZilla no Mac ou no Linux. No Windows, use o WinSCP para se conectar ao dispositivo iOS.

Certifique-se de alterar a senha do OpenSSH após o primeiro login (**ssh root@IPAD_IP_ADDRESS**) no SSH usando o simples "passwd" por meio de uma CLI.

Para obter mais informações, acesse: <http://lifehacker.com/5760626/how-to-install-and-set-up-ssh-on-your-jailbroken-ios-device>

O acesso à raiz permite o acesso a todo o sistema de arquivos do iOS. Examine o sistema de arquivos para entender os vários diretórios e locais importantes onde o aplicativo pode armazenar dados. Esse local pode ser a área restrita do aplicativo ou o cache do sistema operacional iOS.

OBSERVAÇÃO: o OpenSSH não precisa ser instalado no jailbreak mais recente do iOS 10.2 porque o jailbreak vem com uma instância do Dropbear pronta para uso.

8.5 Leitura de dados do aplicativo usando SSH por USB

Às vezes, especialmente durante conferências, não é possível acessar vários dispositivos diretamente em uma rede. Nesses casos, a única maneira de acessar um dispositivo com jailbreak é por meio de USB em vez de WiFi.

Uma ferramenta que funciona bem (mesmo no iOS versão 10.2) é:
http://www.hackthatphone.com/5x/open_ssh_usb.shtml.

Instale o OpenSSH no dispositivo. Com o dispositivo iOS desbloqueado conectado a um laptop, execute o comando abaixo para criar um túnel SSH para acessar o sistema de arquivos do iOS:

```
python tcprelay.py -t 22:3333 &
```



```

python-client — dns@dns-mbp — ..python-client — -zsh ▶ Python — 80×2
[→ python-client python tcprelay.py -t 22:3333 &
 [1] 44035
 → python-client Forwarding local port 3333 to remote port 22
 → python-client █

```

Instale o SSH no dispositivo via USB, porta 3333 (`ssh root@localhost -p 3333`) com credenciais como *raiz/alpina*.

8.6 Leitura de dados do aplicativo no dispositivo iOS

No primeiro dia de teste, é comum percorrer o aplicativo para se familiarizar com ele. Talvez não seja necessário conectar o dispositivo iOS a um laptop. Em vez disso, pode ser útil ter ferramentas simples para acessar os arquivos na área restrita do aplicativo diretamente no dispositivo com jailbreak.

Esta seção mostra algumas das ferramentas que podem ser usadas.

8.6.1 FileExplorer/iFile

O iFile pode ser instalado em um dispositivo com ou sem jailbreak para acessar dados locais diretamente em um dispositivo. A sandbox do aplicativo não pode ser acessada sem ser em um dispositivo sem jailbreak.

O iFile normalmente é instalado por meio do Cydia, mas pode ser instalado em um dispositivo sem jailbreak usando as etapas mencionadas em "4.2 Método II - Usando o Cydia Impactor" e o arquivo IPA aqui:
<https://drive.google.com/open?id=0B0b4lUTjHfRKX3VrdW9GV2NUb2c>

O BillyE tem um FileExplorer aqui: <https://github.com/Billy-Ellis/iOS-File-Explorer> Essa é uma boa alternativa para telefones sem jailbreak, mas observe que os diretórios acessíveis incluem apenas arquivos que são acessíveis publicamente.

8.6.2 Uso de terminais móveis

Use programas de terminal em dispositivos iOS para ler dados ou acessar o shell do iOS. Os dois terminais que funcionam melhor são o NewTerm e o MobileTerminal.

9. Criptografia de dados de aplicativos

Os desenvolvedores podem armazenar dados confidenciais em um dispositivo iOS de várias maneiras. Os dados podem ser armazenados na sandbox do aplicativo ou no iOS Keychain. Esta seção abordará a API de proteção de dados da Apple e as maneiras pelas quais os dados podem ser armazenados em um dispositivo Apple.

9.1 Noções básicas sobre a API de proteção de dados da Apple

Se os dados confidenciais forem armazenados na sandbox do aplicativo, eles poderão ser protegidos usando a API de proteção de dados da Apple. A API de proteção de dados da Apple (DPAPI) especifica quando a chave de descriptografia deve estar disponível. A DPAPI usa uma combinação da senha do dispositivo do usuário e o UID do hardware para criptografar cada arquivo.

A proteção de dados é gerenciada arquivo por arquivo. Sempre que um arquivo é criado em um dispositivo iOS, a Apple usa uma chave específica de arquivo exclusivo de 256 bits e a fornece ao mecanismo de hardware Apple AES integrado. O mecanismo de hardware criptografa o arquivo usando o modo AES-CBC por padrão. Nos dispositivos A8, a criptografia é realizada usando o AES-XTS.

Essa chave específica do arquivo é criptografada com a "chave de classe", dependendo de como e quando o arquivo deve ser acessado e armazenado nos metadados do arquivo que, por sua vez, são criptografados com a chave do sistema de arquivos. A chave de classe é uma chave simplesmente aleatória e é aplicada aos arquivos com base no nível da DPAPI. Consulte a página 15 de <http://esec-lab.sogeti.com/static/publications/11-hitbamsterdam-iphonedataprotection.pdf> para obter uma compreensão detalhada da ID da chave de classe.

As classes de proteção de dados determinam quando a classe é acessível. Veja abaixo as classes de proteção de dados disponíveis no momento:

a) *Proteção completa (NSFileProtectionComplete):*

Esse é o nível de proteção mais seguro que pode ser usado, a menos que seja necessário um acesso contínuo de leitura/gravação ao arquivo em segundo plano ou se o dispositivo estiver bloqueado. A chave de classe é protegida com uma chave derivada da senha do usuário e do UID do dispositivo. Se o dispositivo estiver bloqueado, dependendo da configuração ""exigir senha"" no dispositivo, a chave de classe descriptografada será logo descartada. Os dados protegidos por esse atributo não podem ser acessados até que o usuário digite a senha novamente ou desbloqueie o dispositivo usando o Touch ID.

b) *Protegido a menos que seja aberto (NSFileProtectionCompleteUnlessOpen)*

Esse nível de proteção garante que os arquivos que estão abertos ainda possam ser acessados, mesmo que o usuário bloquee o dispositivo. Outros arquivos com o mesmo nível de proteção não podem ser acessados, a menos que já tenham sido abertos quando o dispositivo foi bloqueado. Os arquivos podem ser criados enquanto o dispositivo estiver bloqueado, mas, uma vez fechados, não poderão ser abertos novamente até que o dispositivo seja desbloqueado.

c) *Protegido até a primeira autenticação do usuário (NSFileProtectionCompleteUntilFirstUserAuthentication):*

O nível de proteção padrão para aplicativos de terceiros. Essa configuração é aplicada automaticamente se outro atributo de proteção não for especificado. Essa classe de proteção é semelhante à Complete Protection, mas o arquivo fica disponível para os usuários depois que eles desbloqueiam o dispositivo pela primeira vez. O arquivo é armazenado em um formato criptografado em um disco e não pode ser acessado até que o dispositivo seja inicializado e até que o primeiro dispositivo seja desbloqueado (semelhante à criptografia de volume total em laptops).

d) *Sem proteção (NSFileProtectionNone):*

Somente a chave de classe com o UID é protegida. O arquivo pode ser lido e gravado a qualquer momento.

Para obter um código-fonte detalhado, juntamente com uma explicação dessas classes de proteção de dados, consulte as seções "Protection Levels" (Níveis de proteção) e "Checking for Protected Data Availability" (Verificação da disponibilidade de dados protegidos) em "iOS Application Security" (Segurança de aplicativos iOS), de David Thiel (<https://www.nostarch.com/iossecurity>).

9.2 Validar as classes de proteção de dados que estão sendo usadas

O FileDp é usado para localizar a classe de proteção de dados do arquivo (<http://www.securitylearn.net/wp-content/uploads/tools/iOS/FileDP.zip>).

Abaixo estão as etapas para usar o FileDp:

1. Envie o FileDp para o dispositivo iOS usando SSH
2. Torne o FileDp executável usando chmod +x
3. Use o comando abaixo para visualizar a constante de acessibilidade de proteção de dados para o arquivo ou diretório na área restrita do aplicativo:
`./FileDP -<f/d> <caminho do arquivo/caminho do diretório>`

9.3 Armazenamento local inseguro de dados

Abaixo estão algumas das maneiras pelas quais os dados podem ser armazenados em um dispositivo:

- Arquivos PropertyList
- Classe NSUserDefaults
- Chaveiro
- Bancos de dados CoreData e SQLite

9.3.1 Arquivos PropertyList

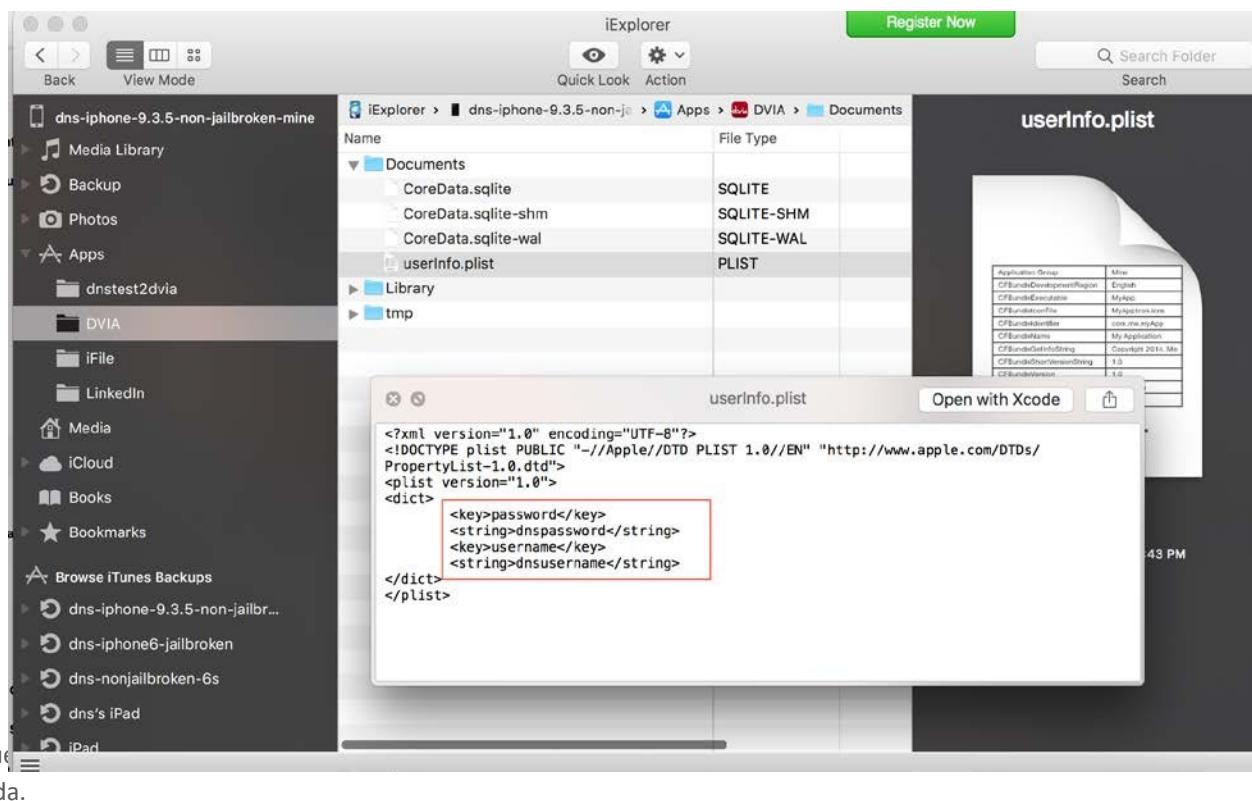
Os arquivos Plist são uma das formas mais padrão de armazenar dados em um dispositivo iOS na forma de pares de valores-chave. Os arquivos plist são basicamente apenas arquivos XML que podem ser lidos pelo Xcode. É muito comum, durante os testes de penetração, observar que os desenvolvedores armazenam dados confidenciais em arquivos plist. Geralmente, os dados confidenciais incluem credenciais, informações de cartão de crédito, chaves de API, informações financeiras, PII etc. Os arquivos plist não são criptografados por padrão e não devem ser usados para armazenar informações confidenciais em texto não criptografado.

Aplicativo usado como exemplo: Abordagem de teste de caixa preta

de aplicativo IOS vulnerável:

1. Inicie o aplicativo e navegue até a seção Armazenamento de dados inseguro.
2. Clique em Plist.
3. Digite o nome de usuário e a senha.
4. Clique em Save in Plist file (Salvar no arquivo Plist).
5. Conecte o dispositivo ao laptop.
6. É possível ler o conteúdo da sandbox do iOS usando qualquer uma das ferramentas e métodos mencionados em "8 Exploring iOS File System" (8 Explorando o sistema de arquivos do iOS).
Este exemplo usa o iExplorer.

7. Navegue até o DVIA no iExplorer. Na pasta Documentos, clique com o botão direito do mouse no arquivo userInfo.plist e selecione "Quick Look".



9.3.2 Classe NSUserDefaults

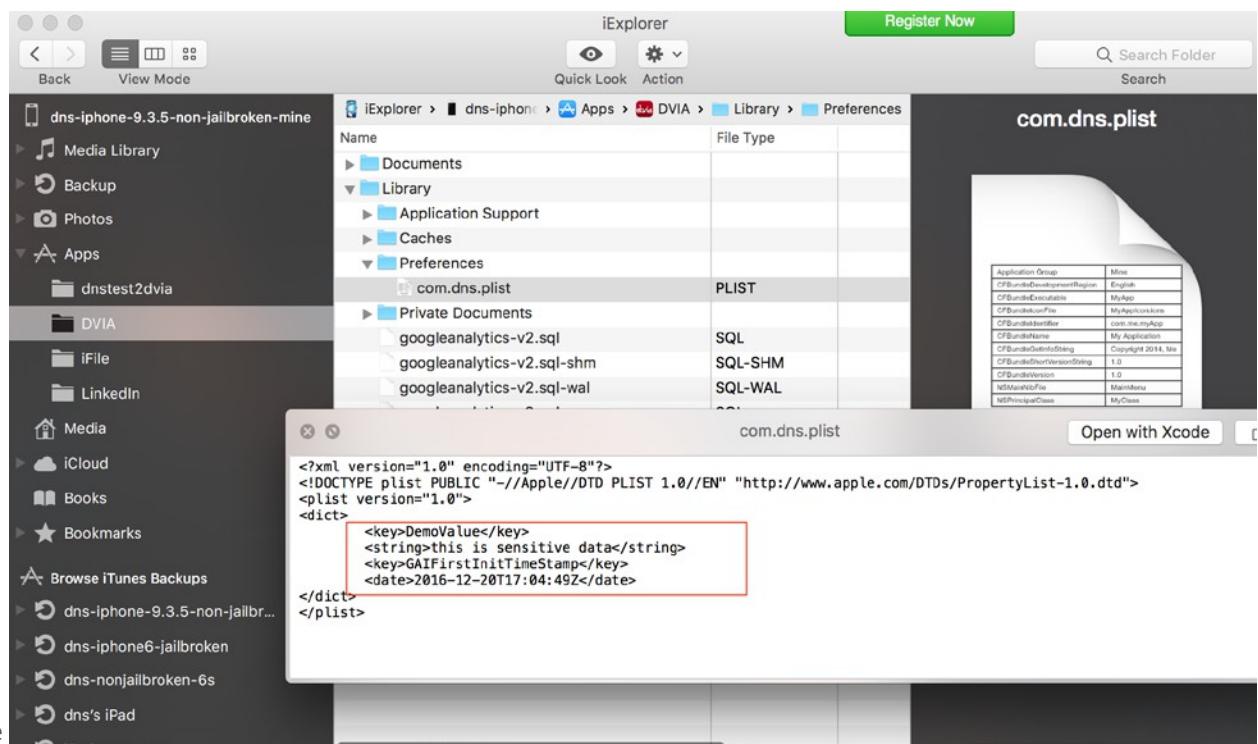
A classe NSUserDefaults é mais uma maneira de os dados no dispositivo iOS persistirem mesmo após a reinicialização. As informações armazenadas na classe NSUserDefaults são armazenadas em um arquivo plist de texto simples em: <Application Directory>/Library/Preferences/<Bundle Identifier>.plist. Durante os pentests, os desenvolvedores podem presumir que os dados serão criptografados e optar por armazenar dados confidenciais aqui.

Aplicativo usado como exemplo: Abordagem de teste de caixa preta

do aplicativo IOS vulnerável:

1. Inicie o aplicativo e navegue até a seção Armazenamento de dados inseguro.
2. Clique em NSUserDefaults.

3. Digite o texto a ser armazenado no campo de texto.
4. Clique em Save (Salvar) em NSUserDefaults.
5. Conecte o dispositivo a um laptop.
6. É possível ler o conteúdo da área restrita do iOS usando qualquer uma das ferramentas e métodos mencionados em "8 Explorando o sistema de arquivos do iOS". Este exemplo usa o iExplorer.
7. Navegue até o DVIA no iExplorer. Na pasta Library > Preferences (Biblioteca > Preferências), clique com o botão direito do mouse no arquivo plist e selecione "Quick Look" (Visualização rápida).



Observe pentest, certifique-se de que nenhuma informação confidencial seja armazenada em arquivos plist sem a criptografia adequada

9.3.3 Chaveiro

O iOS Keychain é um dos melhores locais para armazenar dados confidenciais, como chaves e tokens de login. Os itens do Keychain podem ser compartilhados somente entre aplicativos do mesmo desenvolvedor. Os itens do keychain são criptografados com identificadores de hardware do dispositivo usando AES-GCM. Os dados do keychain são protegidos usando uma estrutura de classe semelhante à usada na API de proteção de dados. As classes também têm comportamento semelhante ao da API de proteção de dados, mas usam chaves distintas e nomes diferentes. Veja a captura de tela abaixo do Guia de Segurança da Apple:
[\(https://www.apple.com/business/docs/iOS_Security_Guide.pdf\)](https://www.apple.com/business/docs/iOS_Security_Guide.pdf)

Availability	File Data Protection	Keychain Data Protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnlessOpen	N/A
After first unlock	NSFileProtectionCompleteUntilFirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessible-WhenPasscodeSetThisDeviceOnly

As classes de proteção do chaveiro determinam quando a classe está acessível. Veja abaixo as classes de proteção de dados atuais:

- Proteção completa (kSecAttrAccessibleWhenUnlocked):* O valor padrão para itens do keychain adicionados sem definir explicitamente uma constante de acessibilidade. Os desenvolvedores usam esse nível de proteção quando o aplicativo precisa acessar os dados do keychain somente quando o aplicativo está em primeiro plano. Quando usado, os dados do item do chaveiro podem ser acessados somente quando o dispositivo estiver desbloqueado. Os itens de dados do keychain com esse atributo migram para um novo dispositivo ao usar o backup criptografado.
- Protected Until First User Authentication (kSecAttrAccessibleAfterFirstUnlock) (Protegido até a primeira autenticação do usuário):* Semelhante ao Complete Protection, mas os itens do keychain ficam disponíveis para os usuários depois que eles desbloqueiam o dispositivo pela primeira vez. Os itens do keychain são armazenados em um formato criptografado em um disco e não podem ser acessados até que o dispositivo seja inicializado e até que o primeiro dispositivo seja desbloqueado.
- Protegido quando o código de acesso está ativado (kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly):* Os desenvolvedores usam esse nível de proteção quando o aplicativo precisa acessar os dados do keychain somente quando o aplicativo está em primeiro plano e precisa de segurança adicional. Quando usado, os dados do item do chaveiro podem ser acessados somente quando o dispositivo estiver desbloqueado e uma senha estiver ativada no dispositivo. Os dados não podem ser armazenados no chaveiro do dispositivo quando o código PIN não estiver ativado.

definido no dispositivo. Os itens de dados do chaveiro com esse atributo nunca migram para um novo dispositivo. Se o código PIN for desativado, os dados do item do chaveiro serão excluídos.

- d. *Sem proteção (kSecAttrAccessibleAlways):* Quando esse nível de proteção é usado, os dados no item do chaveiro estão sempre acessíveis, mesmo quando o dispositivo está bloqueado.

Embora o keychain seja uma forma segura de armazenar dados, em um dispositivo com jailbreak, um invasor ainda pode obter acesso a essas informações. Durante um pentest, procure informações confidenciais, como senhas armazenadas no keychain. As senhas de texto simples nunca devem ser armazenadas em chaveiros. Como atenuante, configure um mecanismo de tratamento de sessão por meio de cookies para evitar a necessidade de armazenar credenciais nos keychains do iOS.

Aplicativo usado como exemplo: Abordagem de teste de caixa preta

de aplicativo iOS vulnerável:

1. Inicie o aplicativo e navegue até a seção Armazenamento de dados inseguro.
2. Clique em Keychains.
3. Digite o texto a ser armazenado no campo de texto.
4. Clique em Salvar no Keychain.
5. SSH no dispositivo iOS com as credenciais root/alpine.
6. Faça o download da ferramenta keychain dumper usando o comando abaixo:
 - o `wget http://github.com/ptoomey3/Keychain-Dumper/archive/master.zip --no-check-certificate`
7. Descompacte a pasta zip e navegue até a pasta Keychain-dumper-master.
8. Torne o arquivo `keychain_dumper` executável usando o comando abaixo:
 - o `chmod +x keychain_dumper`
9. Execute o comando abaixo e observe que as informações que deveriam ser críticas e confidenciais foram encontradas armazenadas em texto simples.
 - o `./keychain_dumper`

Se não estiverem protegidos, os dados do chaveiro também poderão ser encontrados no backup do dispositivo iOS.

Para obter o código-fonte detalhado e uma explicação sobre o uso desses esquemas de proteção do Keychain, consulte o Capítulo 13: seção "Using the Keychain" em "iOS Application Security", de David Thiel (<https://www.nostarch.com/iossecurity>).

9.3.4 Bancos de dados CoreData e SQLite

O CoreData é a estrutura que gerencia a camada entre a interface do usuário e os dados armazenados em um banco de dados. A principal vantagem do CoreData sobre os bancos de dados SQLite é a velocidade e a facilidade de uso. O uso do CoreData cria arquivos sqlite no dispositivo iOS.

A principal diferença entre o uso do SQLite e do CoreData é que as tabelas são prefixadas com Z no CoreData. Os arquivos SQLite são armazenados na pasta Documents na área restrita do aplicativo.

Aplicativo usado como exemplo: Abordagem de teste de caixa preta

de aplicativo IOS vulnerável:

1. Inicie o aplicativo e navegue até a seção Armazenamento de dados inseguro.
2. Clique em Core Data.
3. Insira os dados em todos os campos.
4. Clique em Salvar nos dados principais.
5. Conecte o dispositivo a um laptop.
6. É possível ler o conteúdo da sandbox do iOS usando qualquer uma das ferramentas e métodos mencionados em "8 Exploring iOS File System" (8 Explorando o sistema de arquivos do iOS). Este exemplo usa o iExplorer.
7. Navegue até o DVIA no iExplorer. Na pasta Documents (Documentos), clique com o botão direito do mouse no arquivo CoreData.sqlite e exporte-o para o laptop.
8. O arquivo sqlite pode ser lido usando o SQLite Browser (<http://sqlitebrowser.org/>), o SQLite Manager (<https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>) ou a CLI do sqlite3. Ao usar o sqlite3 no dispositivo iOS, use os comandos abaixo para visualizar o conteúdo do banco de dados sqlite3:
 - o `sqlite3 CoreData.sqlite`
 - o `.tabelas`
 - o `select * from ZUSER`

Observe que as credenciais estão sendo armazenadas em texto simples no dispositivo iOS.

Etapas semelhantes devem ser usadas para testar as vulnerabilidades de armazenamento do SQLite, exceto que as tabelas não terão o prefixo "Z".

9.4 Criptografia quebrada

CommonCrypto é a estrutura que o iOS usa para operações criptográficas. O CCCrypt é a principal função de criptografia e descriptografia da estrutura para criptografia simétrica.

A assinatura do método para CCCrypt:

```
CCCrypt(CCOperation op, CCAgorithm alg, CCOptions options, const void *key, size_t keyLength,const void *iv,
const void *dataIn, size_t dataInLength, void *dataOut, size_t dataOutAvailable, size_t
*dataOutMoved);
```

Os tipos de argumentos passados:

```
CCCryptorStatus CCCrypt(
    CCOperation op,           //operação: kCCEncrypt ou CCAgorithm alg,
    // algoritmo: kCCAlgorithmAES128...
    CCOptions options,        //operação:... const void *key, // chave
    size_t keyLength,          // comprimento da chave
    const void *iv,            //vetor de inicialização(opcional)
    const void *dataIn,         // dados de entrada
    size_t dataInLength,       //comprimento dos
    dados de entrada void *dataOut,   //
    buffer de dados de saída
    size_t dataOutAvailable, // comprimento dos dados de saída disponíveis
    size_t *dataOutMoved) // comprimento real dos dados de saída gerados
```

Alguns dos casos de teste comuns a serem testados ao analisar a segurança criptográfica usada no aplicativo estão descritos na tabela abaixo.

ID do teste	Título do teste	Descrição do teste
1	Geração segura de números aleatórios	Confirme que a geração aleatória segura é executada lendo os bytes do arquivo de dispositivo /dev/random. Isso pode ser feito usando o comando SecRandomCopyBytes

		função. Consulte https://developer.apple.com/reference/security/16585_65-randomization_services para obter detalhes sobre o serviço de randomização no iOS.
2	Vetor de inicialização aleatório	Confirme se o IV usado no processo de criptografia e hashing é gerado usando um gerador pseudoaleatório seguro (usando /dev/random, conforme mencionado em 1) e se é suficientemente aleatório e exclusivo todas as vezes.
3	Vetor de inicialização codificado	Confirme se o IV no processo de criptografia e hashing não está codificado na fonte do aplicativo.
4	Tamanho da chave	Confirme se o tamanho da chave usada no processo de criptografia e hashing do sistema de criptografia é suficientemente grande.
5	Sal aleatório	Confirme se o salt no processo de criptografia e hashing é gerado usando um gerador pseudoaleatório seguro (usando /dev/random, conforme mencionado em 1) e se é suficientemente aleatório e exclusivo todas as vezes.
6	Sal com código rígido	Confirme se o sal no processo de criptografia e hashing não está codificado na fonte do aplicativo.
7	Semente criptográfica robusta para gerador de números aleatórios	Certifique-se de que o valor da semente tenha entropia suficiente e não dependa de fontes fracas de entropia. Além disso, garanta a redefinição periódica do valor da semente.
9	Força criptográfica	Confirme que um algoritmo criptograficamente forte é usado para criptografar dados confidenciais.
10	Verificações de integridade - Criptografia	Confirme se a integridade e a autenticidade da função de criptografia são mantidas por meio do ENCRYPT-then-MAC ou do modo de criptografia autenticada.
11	Hashing de senhas	Confirme se as senhas são armazenadas usando uma função de hash de senha segura.
12	Número de iterações	Confirme se o número de iterações da função de hash/criptografia em uso é suficientemente grande.
13	Verificações de integridade - Hashing	Confirme que a integridade/autenticidade da função de hashing é mantida por meio de ENCRYPT-then-

		MAC.
14	Modo de cifra fraca	Confirme se o modo de cifra ECB fraco não está em uso.
15	Mensagens de erro	Confirme se as mensagens de erro não revelam informações confidenciais sobre o sistema de criptografia ou o computador que hospeda o aplicativo.
16	Criptografia baseada em senha	Confirme se os algoritmos de criptografia baseados em senhas fracas e antigas não estão em uso no momento.
17	Criptografia personalizada	Confirme se o aplicativo não usa um algoritmo de criptografia personalizado.

Certifique-se de que a chave usada no CCCrypt ou em qualquer uma das funções relacionadas nunca seja codificada. Ela deve ser gerada dinamicamente no dispositivo e pode ser armazenada no keychain.

Se for sugerido adicionar funcionalidade de criptografia, o RNCryptor (<https://github.com/rnapier/RNCryptor>) é sua melhor aposta. Basicamente, ele funciona como um wrapper sobre o CommonCrypto e permite a criptografia usando AES e uma chave fornecida pelo usuário. Na maioria dos casos, a chave é gerada dinamicamente no dispositivo e pode ser armazenada no chaveiro.

Use a versão mais recente do RNCryptor, pois as versões mais antigas têm vulnerabilidades conhecidas: (<http://robnapier.net/rncryptor-hmac-vulnerability>).

10. Análise binária

Análise binária - verificação de mitigações de exploração - dificultando a exploração de estouro de buffer

Os estouros de buffer ocorrem quando a entrada habilmente construída pelo invasor sobrescreve a memória, resultando na execução do código do invasor.

Há três tecnologias que são usadas para evitar estouros de buffer

- Randomização de layout de espaço de endereço (ASLR)
- Contagem automática de referência (ARC)
- Protetores de pilha

Se uma ou mais dessas opções não forem usadas ou se houver strings manipuladas incorretamente ou funções de string perigosas, o aplicativo poderá ficar vulnerável a uma exploração de estouro de buffer.

Embora isso não afete diretamente os testes de segurança, é muito mais difícil escrever estouros de buffer para códigos compilados com essas opções.

Os engenheiros de teste precisam saber se essas tecnologias estão implementadas nos aplicativos que estão sendo testados. Esta seção aborda os detalhes necessários para entender melhor essas tecnologias.

10.1 Análise binária - Verificação de mitigações de exploração - Executável independente de posição (PIE e ASLR)

- As vulnerabilidades de corrupção de memória normalmente dependem de saber em que parte do espaço de endereço do processo o código ou os dados podem ser substituídos
 - O código não se importa com sua localização na memória, portanto, é movido para um local aleatório. Isso torna os ataques de Programação Orientada a Retorno (ROP) muito mais difíceis de serem executados de forma confiável.
(<https://access.redhat.com/blogs/766093/posts/1975793>)
 - A ASLR (Address Space Layout Randomization, randomização do layout do espaço de endereço) randomiza o local em que o código e os dados são mapeados no espaço de endereço dos processos
- Os binários executáveis são feitos inteiramente de código independente de posição
 - O ASLR permite a criação de executáveis independentes de posição
 - Todos os aplicativos incorporados são compilados com o PIE por padrão após o iOS 5.0.

Aplicativo usado para o exemplo: LinkedIn da AppStore

Abordagem de teste de caixa branca:

1. Verifique se "Generate Position-Dependent Code" (Gerar código dependente da posição) está definido como "YES" (Sim) na configuração de compilação do projeto XCode
2. Você também pode procurar os sinalizadores *-fPIC* e *-pie* definidos para o compilador

Abordagem de teste de caixa preta:

1. Use o otool para verificar se o PIE está ativado no binário do aplicativo. Execute o comando abaixo e procure o sinalizador PIE no cabeçalho do mach

- otool -hv <nome do aplicativo>

```
● ○ ● LinkedIn — dns@dns-mbp — ..../LinkedIn.app — -zsh — 80x24
[→ LinkedIn.app otool -hv LinkedIn
Mach header
    magic cputype cpusubtype  caps      filetype ncmds sizeofcmds   flags
    MH_MAGIC      ARM        V7  0x00      EXECUTE  106       8284   NOUNDEFS DYL
DLINK TWOLEVEL PIE
Mach header
    magic cputype cpusubtype  caps      filetype ncmds sizeofcmds   flags
MH_MAGIC_64    ARM64      ALL  0x00      EXECUTE  106       8912   NOUNDEFS DYL
DLINK TWOLEVEL PIE
→ LinkedIn.app
```

10.2 Análise binária - Verificação de mitigações de explorações - Contagem automática de referências (ARC)

- A contagem automática de referências (ARC) elimina a responsabilidade do gerenciamento de memória
 - O compilador gerencia a memória, reduzindo a probabilidade de introduzir vulnerabilidades de corrupção de memória no aplicativo
- O ARC avalia os requisitos de tempo de vida dos objetos e insere automaticamente as chamadas de gerenciamento de memória apropriadas durante a pré-compilação
 - Os desenvolvedores não precisam mais lembrar quando usar objetos de memória de retenção, liberação e liberação automática

- O compilador determina quando o tempo de vida de um objeto expirou e desalocará automaticamente os objetos para o desenvolvedor
- Protege contra muitas vulnerabilidades de corrupção de memória e, especialmente, contra falhas de uso após a liberação e de liberação dupla de objetos

Aplicativo usado para o exemplo: LinkedIn da AppStore

Abordagem de teste de caixa branca:

1. Verifique se "Objective-C Automatic Reference Counting" está definido como "YES" na configuração de compilação do projeto XCode
2. Você também pode procurar os sinalizadores de compilador `-fobjc-arc` ou `-fno-objc-arc`

objc-arc que estão sendo usados na abordagem de teste de caixa preta:

1. Use o otool para verificar se o ARC está ativado no binário do aplicativo. Execute o comando abaixo para verificar as referências do ARC:

- `otool -lv <appbinary> | grep objc_`

```
[→ [→ LinkedIn.app otool -lv LinkedIn | grep objc_
→ 0x0004cb6c 406 _objc_autoreleaseReturnValue
→ 0x0004cb7c 407 _objc_constructInstance
→ 0x0004cb8c 409 _objc_getClass
→ 0x0004cb9c 410 _objc_getMetaClass
→ 0x0004cbac 411 _objc_getProtocol
→ 0x0004cbbc 412 _objc_getRequiredClass
→ 0x0004cbcc 413 _objc_initializeClassPair
→ 0x0004cbdc 414 _objc_lookUpClass
→ 0x0004cbec 415 _objc_msgSend
→ 0x0004cbfc 416 _objc_msgSendSuper2
→ 0x0004cc0c 417 _objc_msgSend_stret
→ 0x0004cc1c 419 _objc_registerClassPair
→ 0x0004cc2c 420 _objc_release
→ 0x0004cc3c 421 _objc_retain
→ 0x0004cc4c 422 _objc_retainAutoreleasedReturnValue
→ 0x0005818c 420 _objc_release
→ 0x000581f8 404 _objc_allocateClassPair
→ 0x000581fc 405 _objc_autoreleasePoolPush
→ 0x00058200 408 _objc_copyClassNamesForImage
→ 0x00058204 409 _objc_getClass
→ 0x00058208 410 _objc_getMetaClass
→ 0x0005820c 411 _objc_getProtocol
O Otool n
```

```
_objc_retain
_OBJC_RELEASE
_OBJC_STORESTRONG
_OBJC_RELEASERETURNVALUE
_OBJC_AUTORELEASERETURNVALUE
_OBJC_RETAINAUTORELEASERETURNVALUE
```

10.3 Análise binária - Verificação de mitigações de exploração - Stack Protectors

- Proteção contra vulnerabilidades de corrupção de memória que tentam sobreescriver a pilha, como estouro de buffer baseado em pilha.
- Obtido com a colocação de um valor conhecido ou "canário de pilha" antes das variáveis locais na pilha para proteger o ponto de base salvo, o ponteiro de instrução e os argumentos de função salvos.
- Quando a função retorna, o valor canário é verificado para confirmar que a pilha não foi sobreescrita
- A proteção de pilha é ativada por padrão no aplicativo iOS

mais recente usado para Exemplo: LinkedIn da AppStore

Abordagem de teste de caixa branca:

- Procure os sinalizadores de compilador `-fstack-protector-all`

que estão sendo usados Abordagem de teste de caixa preta:

- Use o otool para verificar se os protetores de pilha estão ativados no binário do aplicativo. Execute o comando abaixo para examinar as referências de pilha:
 - o `otool -lv <appbinary> | grep "stack"`



```
[→ LinkedIn.app otool -Iv LinkedIn | grep "stack"
0x0004ccdc 332 __stack_chk_fail
0x000581ec 333 __stack_chk_guard
0x00058498 332 __stack_chk_fail
0x0000000100040cf4 304 __stack_chk_fail
0x000000010004c150 305 __stack_chk_guard
0x000000010004c718 304 __stack_chk_fail
→ LinkedIn.app]
```

A presença desses símbolos indica proteção da pilha

- __falha_de_chk_pilha
- __proteção_de_pilha_chk

10.4 Análise binária - Lista todas as bibliotecas usadas no binário do iOS

Ao realizar um teste de penetração em um aplicativo iOS, verifique a postura de segurança do aplicativo, bem como as bibliotecas usadas no aplicativo, usando o otool ou uma ferramenta mais avançada chamada Jtool.

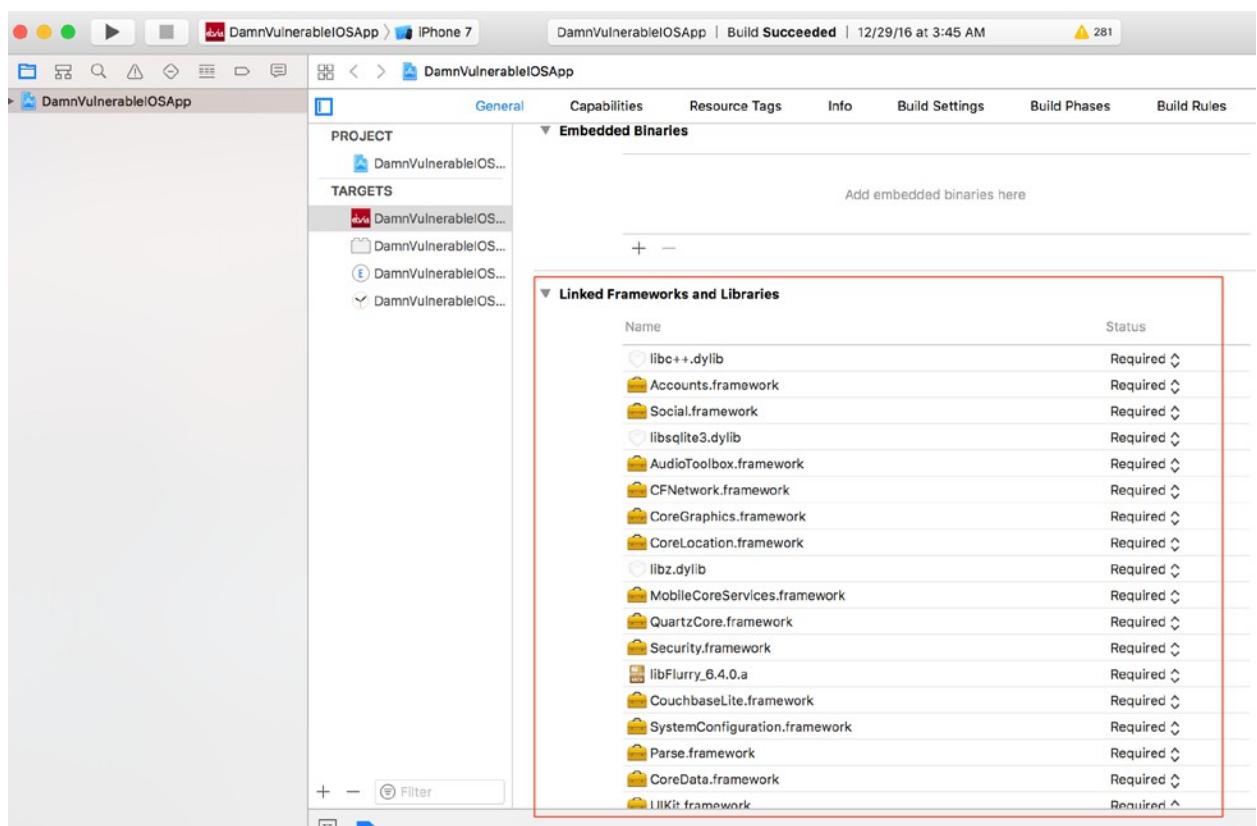
As bibliotecas de vulnerabilidades no aplicativo podem permitir vazamentos de informações do aplicativo que não deveriam estar disponíveis para os invasores.

Faça o download da versão mais recente do Jtool em <http://newosxbook.com/tools/jtool.html>.

Aplicativo usado como exemplo: Abordagem de teste de caixa branca

de aplicativo iOS vulnerável:

1. Abra o projeto Xcode e visualize as propriedades "General" do projeto.
2. Role a tela para baixo para ver a seção "Linked Framework and Libraries". Ela lista as estruturas usadas no aplicativo.



vulnerabilidades abertas publicamente conhecidas nessas bibliotecas deixam o aplicativo vulnerável.

Aplicativo usado para Exemplo: Abordagem de teste de

caixa preta do aplicativo do LinkedIn:

Extraia o arquivo .app ou o arquivo .IPA usando as etapas mencionadas em "2. Aquisição de binários do iOS". Descompacte o arquivo e execute o comando abaixo no arquivo binário:

- `./jtool -L <binário do aplicativo> -arch arm64`

```
LinkedIn — dns@dns-mac — ../LinkedIn.app — -zsh — 80x44
[→ LinkedIn.app ./jtool -L LinkedIn
Fat binary, big-endian, 2 architectures: armv7, arm64
Specify one of these architectures with -arch switch, or export the ARCH environment variable
[→ LinkedIn.app ./jtool -L LinkedIn -arch arm64
@rpath/AHEasing.framework/AHEasing
@rpath/ArtDeco.framework/ArtDeco
@rpath/CocoaLumberjack.framework/CocoaLumberjack
@rpath/ConsistencyManager.framework/ConsistencyManager
@rpath/CrNet.framework/CrNet
@rpath/EKGClient.framework/EKGClient
@rpath/FLAnimatedImage.framework/FLAnimatedImage
@rpath/FMDB.framework/FMDB
@rpath/FortHallRootViewController.framework/FortHallRootViewController
@rpath/Hakawai.framework/Hakawai
@rpath/I18n.framework/I18n
@rpath/LIAnnotationKit.framework/LIAnnotationKit
@rpath/LIAuthLibrary.framework/LIAuthLibrary
@rpath/LICookies.framework/LICookies
@rpath/LICrosslink.framework/LICrosslink
@rpath/LICrosspromo.framework/LICrosspromo
@rpath/LIDelightUI.framework/LIDelightUI
@rpath/LIMemberSettings.framework/LIMemberSettings
@rpath/LIPayments.framework/LIPayments
@rpath/LIPerformance.framework/LIPerformance
@rpath/LIRealTime.framework/LIRealTime
@rpath/LISemaphoreLib.framework/LISemaphoreLib
@rpath/LIToolbox.framework/LIToolbox
@rpath/LITrackingLib.framework/LITrackingLib
@rpath/LIVideo.framework/LIVideo
@rpath/LayoutKit.framework/LayoutKit
@rpath/LixLib.framework/LixLib
@rpath/Networking.framework/Networking
@rpath/Operations.framework/Operations
@rpath/PIXImage.framework/PIXImage
@rpath/Rate.framework/Rate
@rpath/RestLiClientLib.framework/RestLiClientLib
@rpath/RestLiObjectData.framework/RestLiObjectData
```

Quaisque

10.5 Engenharia reversa simples de binários do iOS usando class-dump-z

Ao realizar um teste de penetração em um aplicativo iOS, é importante ler o código do aplicativo fornecido e entender as classes de back-end e as informações ocultas. Isso permite a exploração do aplicativo para obter acesso a informações confidenciais ou para redirecionar o fluxo do aplicativo de forma mal-intencionada. A engenharia reversa de um aplicativo iOS é completamente diferente da de um apk Android.

O código-fonte original completo não pode ser revivido de um aplicativo iOS existente. Somente as declarações de classes, categorias e protocolos podem ser descompiladas de um determinado binário do iOS. Ferramentas avançadas, como IDA Pro ou Hopper, podem ser usadas para examinar o pseudocódigo.

Usando o *class-dump-z* do repositório cydia.radare.org no Cydia como exemplo, observe que o utilitário padrão de despejo de classe fornecido com o Cydia não é compatível com arquivos Mach-O de 64 bits. A versão Mac OSX do class dump para engenharia reversa do aplicativo também pode ser usada.

Aplicativo usado para o exemplo: Aplicativo padrão de ações

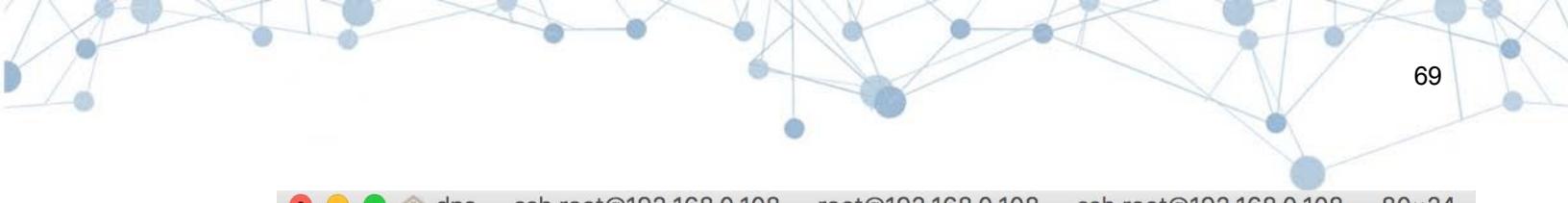
Abaixo estão as etapas para executar a descompilação dos aplicativos iOS usando o *class-dump-z*:

1. SSH no dispositivo iOS usando credenciais como root:alpine.

```
dns ~ ssh root@192.168.0.108
The authenticity of host '192.168.0.108 (192.168.0.108)' can't be established.
RSA key fingerprint is SHA256:8mLkKfoLMK0IZH13QgKufU+vLWVaObPIlogF8T8U114.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.108' (RSA) to the list of known hosts.
root@192.168.0.108's password:
dns-iphone6-jailbroken:~ root# whoami
root
dns-iphone6-jailbroken:~ root#
```

- 2.

"App".



```
[dns-iphone6-jailbroken:~ root# ps -ax | grep "App"
 90 ??      0:00.09 /System/Library/CoreServices/AppleIDAuthAgent
 237 ??     0:00.80 /System/Library/PrivateFrameworks/ApplePushService.framework
ework/apsd
 407 ??     0:00.03 /System/Library/PrivateFrameworks/AppSupport.framework/
Support/cplogd
 412 ??     0:00.80 /Applications/MobileMail.app/MobileMail
 435 ??     0:00.18 /private/var/db/stash/_rlCHnR/Applications/ServerDocum
ents.app/PlugIns/ServerFileProvider.appex/ServerFileProvider
 605 ??     0:06.03 /Applications/Preferences.app/Preferences
 710 ??     0:00.15 /System/Library/CoreServices/CacheDeleteAppContainerCac
hes
 714 ??     0:00.17 /Applications/MobileSafari.app/webbookmarksd
 727 ??     0:00.23 /private/var/db/stash/_rlCHnR/Applications/MobileCal.a
pp/PlugIns/CalendarWidget.appex/CalendarWidget
 729 ??     0:00.42 /private/var/db/stash/_rlCHnR/Applications/Stocks.app/
PlugIns/StocksWidget.appex/StocksWidget
 745 ??     0:13.00 /var/mobile/Containers/Bundle/Application/64D7C0EC-7D28
-4BE1-B2A9-6C527CA2C27A/Twitter.app/Twitter
 772 ??     0:01.20 /Applications/Stocks.app/Stocks
 777 ttys000  0:00.00 grep App
dns-iphone6-jailbroken:~ root#]
```

Conforme mostrado no diagrama acima, o aplicativo está sendo executado no local "
/Applications/Stocks.app/Stocks".

- Navegue até "/Applications/Stocks.app/" por meio do shell. Use o class-dump-z para fazer a engenharia reversa desse aplicativo. Trata-se de um utilitário de linha de comando para examinar as informações de tempo de execução do Objective-C armazenadas em arquivos Mach-O. Ele gera declarações para as classes, categorias e protocolos. Faça isso usando o comando abaixo:

- `class-dump-z Stocks > /tmp/Stockreversed.txt`.

O class-dump-z -H /var/mobile/<app-binary-to-be-reversed> -o
/var/mobile/<outputdirectory>/ também pode ser usado para obter os cabeçalhos em arquivos separados.

Em dispositivos arm64, pode ocorrer o seguinte erro:

```
[dns-iphone6-jailbroken:/Applications/Stocks.app root# class-dump-z Stocks > /tmp
/Stockreversed.txt
dyld: Library not loaded: /usr/lib/libpcre.0.dylib
Referenced from: /usr/bin/class-dump-z
Reason: image not found
Trace/BPT trap: 5
dns-iphone6-jailbroken:/Applications/Stocks.app root# ]
```

Se for o caso, instale o "pcre" pelo Cydia para corrigir.

Observação: Mesmo depois de executar o class-dump-z corretamente, pode ocorrer um erro "null", conforme mostrado na captura de tela a seguir.



```

dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80x24
[dns-iphone6-jailbroken:/Applications/Stocks.app root# class-dump-z Stocks > /tmp
/Stockreversed.txt
[dns-iphone6-jailbroken:/Applications/Stocks.app root# cat /tmp/Stockreversed.txt]

/**
 * This header is generated by class-dump-z 0.2a.
 * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 *
 * Source: (null)
 */

dns-iphone6-jailbroken:/Applications/Stocks.app root#

```

Se for o caso, instale o "classdump-dyld" em vez do class-dump-z padrão (`classdump-dyld Stocks > /tmp/Stockreversed.txt`). Outra opção é tentar `classdump-dyld -o /tmp/dump Stocks`.

4. A captura de tela abaixo mostra o conteúdo do arquivo "Stockreversed.txt". Ele é legível e contém informações valiosas.

```

dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80x3
-(void)setScrollViewSubviews:(NSMutableArray *)arg1 ;
-(void)_applyLayout:(id)arg1 ;
-(void)restoreSavedState;
-(void)setSavedPageForInfiniteScrollView:(long long)arg1 ;
-(void)_setUpScrollViewsWithLayout:(id)arg1 ;
-(long long)savedPageForInfiniteScrollView;
-(UIView *)grayView;
-(UIView *)statusViewDivider;
-(UIView *)primaryHorizontalDivider;
-(UIView *)secondaryHorizontalDivider;
-(UIView *)secondaryVerticalDivider;
-(UIView *)blackView;
-(void)setCurrentLayout:(StocksLayout *)arg1 ;
-(void)reorderScrollViewSubviews:(id)arg1 ;
-(void)positionScrollViewSubviews;
-(void)_viewDidLayoutSubviews;
-(void)_prepareForTransitionToSize:(CGSize)arg1 ;
-(void)_animateTransitionToSize:(CGSize)arg1 duration:(double)arg2 ;
-(void)_completeTransitionToSize:(CGSize)arg1 ;
-(long long)visibleDetailViewType;
-(void)setVisibleDetailViewType:(long long)arg1 ;
-(void)updateChartViews;
-(void)_flashNewsViewScrollIndicatorsIfNeeded;
-(id)_stockWithOffset:(long long)arg1 ;
-(void)setLayoutCache:(NSMutableDictionary *)arg1 ;
-(UIView *)secondaryGrayView;
-(void)setBlackView:(UIView *)arg1 ;
-(void)setChartViews:(NSMutableArray *)arg1 ;
-(StockInfoView *)infoView;
-(void)setInfoView:(StockInfoView *)arg1 .

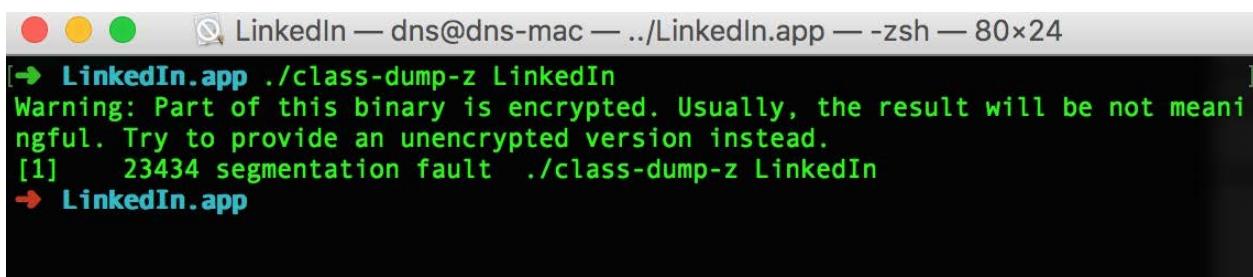
```

As declarações das classes e dos protocolos permitem a depuração do aplicativo usando o GDB. Como alternativa, é possível se conectar às funções presentes no aplicativo por meio do Cycript e tentar alterar seu comportamento. Esse tópico será explicado em mais detalhes posteriormente neste guia.

Observação: A versão Mac OSX do despejo de classe para engenharia reversa do aplicativo também pode ser usada.

11. Descriptografia de aplicativos iOS (binários da AppStore)

Às vezes, é necessário testar os aplicativos que estão ativos nas App Stores. Se a extração do arquivo .IPA da App Store (usando os métodos mencionados na seção 2, Aquisição de binários do iOS) e a tentativa de descompilar o aplicativo por meio de ferramentas como class-dump-z falharem, é provável que isso se deva ao esquema FairPlay DRM da Apple para proteção contra pirataria.



```
LinkedIn — dns@dns-mac — ..//LinkedIn.app — -zsh — 80x24
[→ LinkedIn.app ./class-dump-z LinkedIn
Warning: Part of this binary is encrypted. Usually, the result will be not meaningful. Try to provide an unencrypted version instead.
[1] 23434 segmentation fault ./class-dump-z LinkedIn
→ LinkedIn.app
```

Os aplicativos que normalmente não são criptografados incluem:

- Aplicativos instalados por padrão no dispositivo iOS (localizados em /Applications/)
- Aplicativos autodistribuídos
- Aplicativos com carregamento lateral

Para esses aplicativos, não há necessidade de fazer nada para descriptografiá-los. A análise binária pode ser realizada neles "como estão".

11.1 Método manual

Esse é o método de descriptografia mais complicado e mais demorado.

11.1.1 Usando o GDB

A opção mais fácil é usar um dispositivo com jailbreak que execute o GDB corretamente e sem erros. Se essa não for uma opção, considere as sugestões da próxima seção e veja o uso do LLDB. Use o GDB do repositório cydia.radare.org. Se houver problemas ao usar o GDB, tente usar o lipo.

Aplicativo usado para o exemplo: Aplicativo do Facebook da AppStore

Abaixo estão as etapas para executar a descriptografia dos binários do iOS manualmente:

1. Inicie o aplicativo iOS em um dispositivo e localize o segmento criptografado por meio do otool usando a seguinte sintaxe:
 - o `otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4`

```
sparrow:Facebook.app dns$ otool -l Facebook | grep LC_ENCRYPTION_INFO -A 4
    cmd LC_ENCRYPTION_INFO
    cmdsize 20
    cryptoff 16384
    cryptsize 31598352
    cryptid 1
sparrow:Facebook.app dns$
```

O cryptid=1 indica que o aplicativo está criptografado.

2. Localize o segmento criptografado usando o comando abaixo:
 - o `otool -l <nome_do_aplicativo> | grep LC_ENCRYPTION_INFO -A 4`

```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Facebook.app root# otool -l Facebook | grep LC_ENCRYPTION_INFO -A 4
    cmd LC_ENCRYPTION_INFO
    cmdsize 20
    cryptoff 16384
    cryptsize 37208064
    cryptid 1
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Facebook.app root#
```

O campo cryptoff fornece o início dos dados criptografados (16384 bytes [0x4000] no arquivo). O campo cryptsize é o tamanho do segmento criptografado (37208064, [0x237C000]).

3. O comando abaixo fornece o vmsize (o tamanho completo do segmento)
 - o `otool -arch all -Vl <nome_do_aplicativo>`

```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Facebook.app root# otool -arch all -Vl Facebook
Facebook:
Load command 0
    cmd LC_SEGMENT
    cmdsize 56
    segname __PAGEZERO
    vmaddr 0x00000000
    vmsize 0x00004000
    fileoff 0
    filesize 0
```

Calcule os endereços inicial e final:

Endereço inicial = hex(cryptoff) + endereço base = 0x4000 + 0x4000 = 0x8000

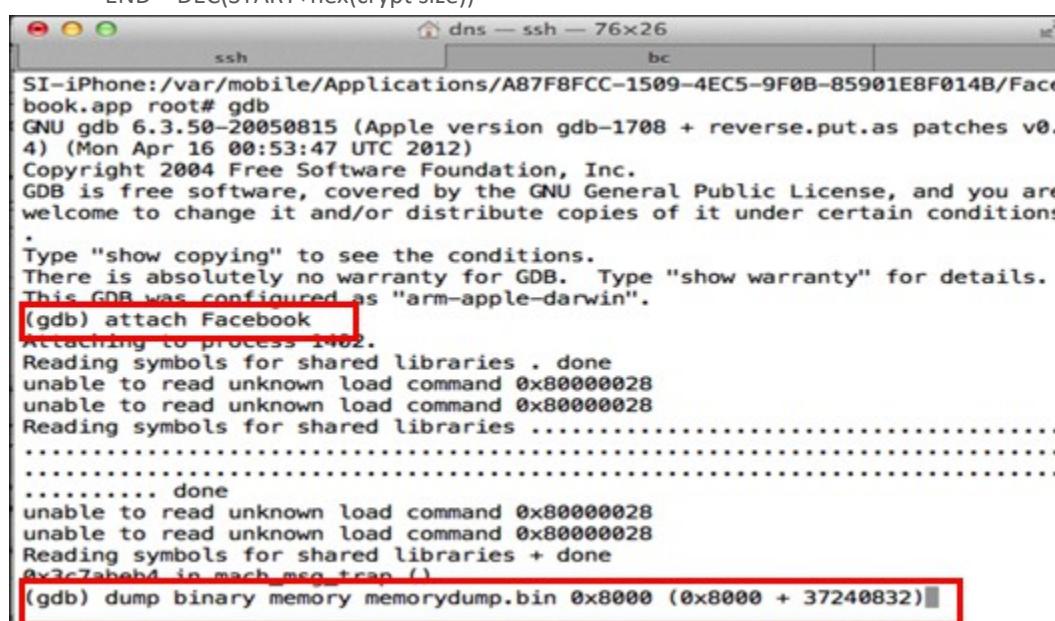
Endereço final = endereço inicial + cryptsize = 0x8000 + 0x237C000 = 0x2384000 (37240832)

Observação: o endereço base é o mesmo que vmsize ou pode ser encontrado usando "info sharedlibrary".

4. Definir um ponto de interrupção usando o GDB
 - o `gdb attach <nome_do_aplicativo>`
5. Despeje o segmento descriptografado da memória e salve-o em um arquivo que será usado para corrigir o binário criptografado
 - o `despejar memória binária memorydump.bin <start_addr> <end_addr>`

Observação: START = hex(base)+hex(cryptoff)

END = DEC(START+hex(crypt size))



```

SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# gdb
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.
4) (Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions
.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
(gdb) attach Facebook
Attaching to process 1402.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
unable to read unknown load command 0x80000028
Reading symbols for shared libraries .....  

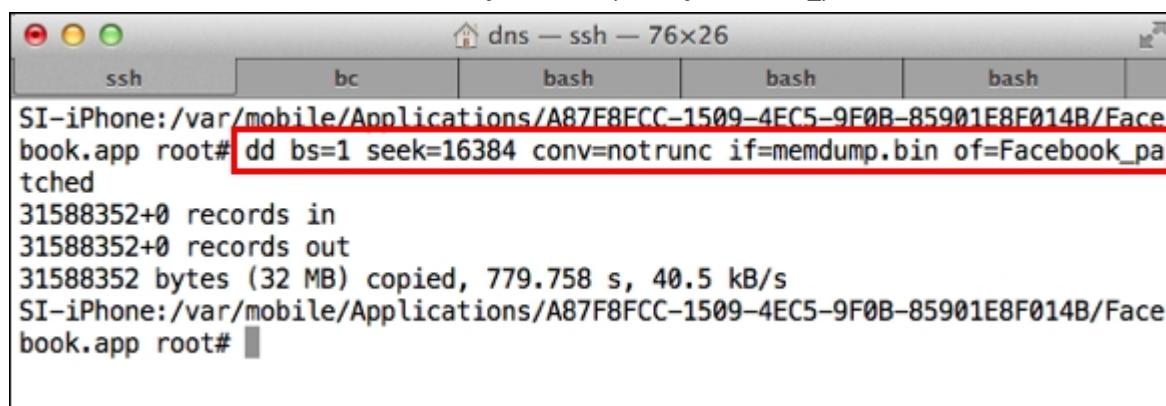
.....  

..... done
unable to read unknown load command 0x80000028
unable to read unknown load command 0x80000028
Reading symbols for shared libraries + done
a=2-7shbh4 in mach_mach_trap()
(gdb) dump binary memory memorydump.bin 0x8000 (0x8000 + 37240832)

```

6. Substitua os dados criptografados por dados descriptografados da memória. Copie os dados descriptografados para o binário usando o comando abaixo:

- o `dd bs=1 seek=16384 conv=notrunc if=memdump.bin of=Facebook_patched`



```

SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# dd bs=1 seek=16384 conv=notrunc if=memdump.bin of=Facebook_pa
tched
31588352+0 records in
31588352+0 records out
31588352 bytes (32 MB) copied, 779.758 s, 40.5 kB/s
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root#

```

16384 é o valor cryptoff encontrado na consulta otool. O

binário não será mais criptografado no dispositivo.

7. Para converter o binário em um binário descriptografado, é necessário corrigir o cryptid para desativar o comando de carregamento de criptografia. Localizar o deslocamento do cryptid usando o MachOView.



Usar um editor hexadecimal para definir o campo cryptid como 0x0.

8. Verifique Cryptid para mostrar que a carga de criptografia está desativada (=0)

```
Desktop — bash — 76x26
ssh          bc      bash      bash
sparrow:Desktop dns$ otool -l Facebook_patched | grep crypt
    cryptoff 16384
    cryptsize 37208064
    cryptid  0
sparrow:Desktop dns$
```

OBSERVAÇÃO: o endereço base também pode ser encontrado usando "info shared library" no GDB (detalhes mencionados no Mobile Application Hacker's Handbook).

11.1.2 Usando o LLDB

Desde o Xcode 5, o LLDB tem sido o padrão para a depuração do iOS. Ele foi criado em estreita coordenação com os compiladores LLVM para substituir o GDB.

Aplicativo usado para o exemplo: Aplicativo LinkedIn da AppStore

Abaixo estão as etapas para executar a descriptografia dos binários do iOS manualmente:

- Inicie o aplicativo iOS no dispositivo e localize o segmento criptografado por meio do otool usando a seguinte sintaxe:

- `otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4`

```
● ○ ● LinkedIn — dns@DellVortex — ..file/LinkedIn — -zsh — 80x24
[→ LinkedIn otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4

    cmd LC_ENCRYPTION_INFO
    cmdsize 20
    cryptoff 16384
    cryptsize 311296
    cryptid 1
    --
    cmd LC_ENCRYPTION_INFO_64
    cmdsize 24
    cryptoff 16384
    cryptsize 294912
    cryptid 1
→ LinkedIn
```

- O campo `cryptid=1` indica que o aplicativo está criptografado. O campo `cryptoff` fornece o início dos dados criptografados e o campo `cryptsize` é o tamanho do segmento criptografado.
- Na etapa anterior, observe que há duas entradas diferentes para `LC_ENCRYPTION_INFO`. Isso indica que o aplicativo é um aplicativo de várias arquiteturas. Escolha e descriptografe um aplicativo de cada vez. Use o comando abaixo para visualizar os detalhes da arquitetura:

- `otool -fh LinkedIn`

```
LinkedIn — dns@DellVortex — ..bile/LinkedIn — -zsh — 80x26
[→ LinkedIn otool -fh LinkedIn
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 12
    cpusubtype 9
    capabilities 0x0
    offset 16384
    size 514432
    align 2^14 (16384)
architecture 1
    cputype 16777228
    cpusubtype 0
    capabilities 0x0
    offset 540672
    size 513088
    align 2^14 (16384)
Mach header
    magic cputype cpusubtype  caps      filetype ncmds sizeofcmds   flags
    0xfeedface      12          9 0x00          2    103        8080 0x00200085
Mach header
    magic cputype cpusubtype  caps      filetype ncmds sizeofcmds   flags
    0xfeedfacf 16777228          0 0x00          2    103        8704 0x00200085
→ LinkedIn
```

3. A arquitetura específica pode ser escolhida usando o atributo `-arch`.

- o `otool -arch armv7 -l LinkedIn | grep crypt`

```
[→ LinkedIn otool -arch armv7 -l LinkedIn | grep crypt
cryptoff 16384
cryptsize 311296
cryptid 1
→ LinkedIn ]
```

(16384) e o campo `cryptsize` é o tamanho do segmento criptografado (311296).

4. Em um dispositivo iOS, inicie um servidor de depuração e conecte o aplicativo LinkedIn seguindo as etapas mencionadas na seção intitulada "Depuração de aplicativo iOS usando LLDB".

```
((lldb) platform select remote-ios
Platform: remote-ios
Connected: no
SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
((lldb) process connect connect://10.10.0.120:6666

(error: Process 2400 is currently being debugged, kill the process before connecting.
Process 2400 stopped
* thread #1: tid = 0x314b5, 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-thread', stop reason =
signal SIGSTOP
    frame #0: 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
-> 0x199104c30 <+8>: ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
0x199104c34 <+0>: movn  x16, #0x1f
0x199104c38 <+4>: svc    #0x80
0x199104c3c <+8>: ret
((lldb) ]
```

5. Imagem executável na memória.

Observação - Se o aplicativo for compilado com ASLR (PIE) ativado, esse deslocamento de imagem será diferente sempre que o aplicativo for iniciado.

6. Despeje o segmento descriptografado da memória e salve-o em um arquivo que será usado para corrigir o binário criptografado usando o comando abaixo:

- o `(lldb) leitura de memória --force --outfile LinkedIn_memdump.bin --binary --count <cryptsize> <deslocamento de imagem>+<cryptoff>`

7. Substitua os dados criptografados por dados descriptografados da memória. Copie os dados descriptografados para o binário usando o comando abaixo:

- o `dd bs=1 seek=<cryptoff> conv=notrunc if=LinkedIn_memdump.bin of=LinkedIn_patched`

Where seekvalue= <offset from "otool -fh" + cryptoff from "otool -arch armv7 -l"> Esse binário de saída não é mais criptografado no dispositivo.

8. Corrija o cryptid para desativar o comando de carregamento de criptografia. Isso pode ser facilmente modificado por meio do MachOView. Faça o download em:
<https://sourceforge.net/projects/machoview/>. Abra o binário corrigido do LinkedIn no MachOView. Localize "cryptid". Na UI, clique duas vezes em "Data" em "Crypt ID" para cryptid=1 e defina-o como zero. Salve o binário.
9. Para verificar se a alteração da cripta foi refletida, visualize o valor usando o comando abaixo:
 - o `otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4`
10. Usando o binário descriptografado, é possível revertê-lo por meio de ferramentas como class-dump-z

OBSERVAÇÃO: guias recomendados sobre a descriptografia manual de aplicativos da AppStore podem ser encontrados aqui:

- <http://codedigging.com/blog/2016-03-01-decrypting-apps-from-appstore/>
- <http://codedigging.com/blog/2016-04-27-debugging-ios-binaries-with-lldb/> Ou

consulte o Capítulo 6 do livro "iOS Application Security", de David Thiel (<https://www.nostarch.com/iossecurity>)

11.2 Método automatizado

Como o método manual consome muito tempo, considere uma das seguintes ferramentas automatizadas para ajudar a descriptografar os binários do iOS para análise binária.

11.2.1 Usando dump descriptografado

O Dump decrypted funciona injetando um construtor no aplicativo por meio de um vinculador dinâmico. Esse construtor extrai o segmento descriptografado da mesma forma que o método manual.

O link para a ferramenta está aqui: <https://github.com/stefanesser/dumpdecrypted>. Use "make" para criar o .dylib necessário.

Aplicativo usado para o exemplo: Aplicativo LinkedIn da AppStore

Para usar a ferramenta, faça upload do dumpdecrypted.dylib para a pasta Documentos do iOS em Dispositivo /var/mobile/Containers/Data/Application/<Linkedin-GUID>/Documents. O Linkedin-GUID pode ser encontrado executando 'ps aux|grep -i <appname>' e observando o caminho na última coluna da tela.

Mude a pasta de trabalho para esse diretório e execute o seguinte comando na área restrita do aplicativo:

DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib
 /var/mobile/Containers/Bundle/Application/LinkedIn.app/LinkedIn

```
LinkedIn — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root...
[dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D]
63040865/Documents root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/C
ontainers/Bundle/Application/57FD44E6-5642-4531-9B2F-AF66E5F7644E/LinkedIn.app/L
inkedIn
mach-o decryption dumper

DISCLAIMER: This tool is only meant for security research purposes, not for appl
ication crackers.

[+] detected 64bit ARM binary in memory.
[+] offset to cryptid found: @0x1000a8a78(from 0x1000a8000) = a78
[+] Found encrypted data at address 00004000 of length 294912 bytes - type 1.
[+] Opening /private/var/mobile/Containers/Bundle/Application/57FD44E6-5642-4531
-9B2F-AF66E5F7644E/LinkedIn.app/LinkedIn for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a plain MACH-O image
[+] Opening LinkedIn.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Setting the LC_ENCRYPTION_INFO->cryptid to 0 at offset a78
[+] Closing original file
[+] Closing dump file
dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D
63040865/Documents root# █
```

Isso gera uma cópia descriptografada do binário "LinkedIn.decrypted" no diretório de trabalho atual. Agora, execute o comando abaixo para ter certeza de que o binário está descriptografado, observando o valor de cryptid.

- `otool -l LinkedIn.decrypted | grep LC_ENCRYPTION_INFO -A 4`



```
LinkedIn — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root...
[dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D
63040865/Documents root# otool -l LinkedIn.decrypted | grep LC_ENCRYPTION_INFO -
A 4
    cmd LC_ENCRYPTION_INFO_64
    cmdsize 24
    cryptoff 16384
    cryptsize 294912
    cryptid 0
dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D
63040865/Documents root#
```

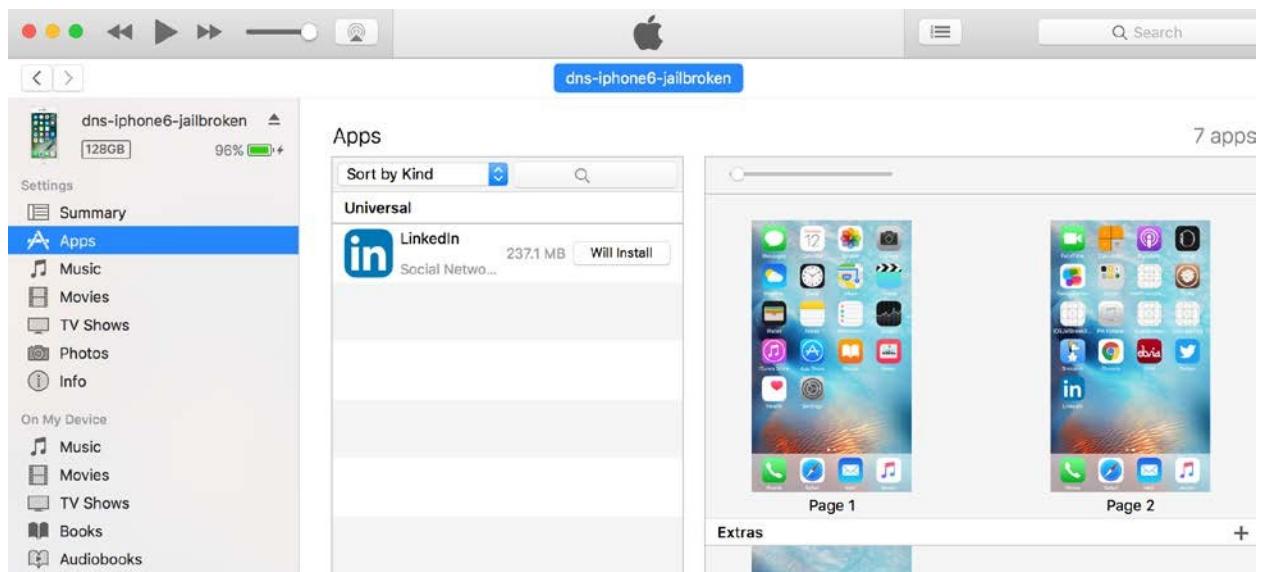
Cryptid de 0 indica que o aplicativo foi descriptografado.

11.2.2 Usando a embreagem

Essa é a maneira mais fácil de descriptografar binários criptografados do iOS. Faça o download da versão mais recente do Clutch em <https://github.com/KJCracks/Clutch/releases> e mova-o para a pasta /bin/ no dispositivo iOS ou instale o Clutch a partir do repositório do cydia <http://cydia.iphonecake.com>.

Aplicativo usado para o exemplo: Aplicativo LinkedIn da AppStore

1. Faça o download do aplicativo LinkedIn usando a AppStore no MacBook. Instale e sincronize o aplicativo do LinkedIn com o dispositivo iOS.



2. SSH no dispositivo iOS.
3. Use o comando abaixo para listar todos os aplicativos instalados no dispositivo iOS
 - *Embreagem -i*

```

dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.1...
[dns-iphone6-jailbroken:~ root# Clutch -i
Installed apps:
 1: Chrome - web browser by Google <com.google.chrome.ios>
 2: Twitter <com.atebits.Tweetie2>
 3: LinkedIn <com.linkedin.LinkedIn>
dns-iphone6-jailbroken:~ root#

```

4.
 - *Clutch -d <app-id do comando anterior>*
- Para descriptografar o LinkedIn, digite
- *Embreagem -d 3*



```

dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 110×24
DUMP | Framework64Dumper <arm64> <VoyagerPublishing> Success! Child exited with status 0
Dumping <VoyagerFeedShell> arm64
Dumping <VoyagerShell> arm64
Dumping <VoyagerPeople> arm64
Dumping <VoyagerFeed> arm64
DUMP | Framework64Dumper <arm64> <VoyagerCore> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerSearch> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerMe> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerGrowth> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerPeople> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerEntities> Success! Child exited with status 0
Dumping <VoyagerMessaging> arm64
DUMP | Framework64Dumper <arm64> <VoyagerProfileEdit> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerFeedShell> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerFeed> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerProfileView> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerShell> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerDataModel> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerMessaging> Success! Child exited with status 0
DONE: /private/var/mobile/Documents/Dumped/com.linkedin.LinkedIn-iOS8.0-(Clutch-2.0).ipa
Finished dumping com.linkedin.LinkedIn in 104.5 seconds
dns-iphone6-jailbroken:/private/var/mobile/Documents/Dumped root#
dns-iphone6-jailbroken:/private/var/mobile/Documents/Dumped root# █

```

O aplicativo descriptografado pode ser encontrado na forma de um arquivo IPA no mesmo dispositivo em /private/var/mobile/Documents/Dumped/

5. Descompacte o aplicativo descriptografado e execute o comando abaixo para certificar-se de que o binário foi descriptografado, observando o valor de cryptid.
 - o `otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4`
6. Agora é possível executar o class-dump-z no binário.

Observação:

- A mensagem "Segmentation fault: 11" (Falha de segmentação: 11), que é emitida com o uso de "ulimit -n 2048" pela Clutch, indica que o número de manipuladores de arquivos abertos permitidos por processo é aumentado para resolver o problema.

- Use "Clutch -f" para limpar o cache do Clutch.
- Às vezes, o class-dump-z não produz nenhum resultado no dispositivo. Se isso acontecer, execute o Clutch, extraia o arquivo ipa do dispositivo e execute a versão para Mac do class-dump nele.
- A mensagem de erro, "Killed: 9", significa que há um erro de assinatura. ldid -S <binary> deve corrigi-lo.

12. Depuração de aplicativos iOS - Manipulação do tempo de execução

A análise de tempo de execução (dinâmica) é a capacidade de manipular aplicativos enquanto eles estão em execução. Isso é feito ativando a depuração e a funcionalidade de rastreamento de tempo de execução. Com a funcionalidade de depuração e rastreamento ativada, um invasor pode manipular como o aplicativo se comporta durante o tempo de execução.

A Manipulação do tempo de execução permite que o invasor:

- Executar funcionalidade oculta que não deve ser acessível
- Descobrir criptografia fraca/ausente
- Contornar restrições do lado do cliente
- Desbloqueie recursos adicionais e conteúdo premium
- Despejar conteúdo protegido por direitos autorais

12.1 Cycript em um dispositivo com jailbreak

O Cycript é a ferramenta mais comumente usada para realizar depuração ou manipulação de tempo de execução em aplicativos iOS. Um guia detalhado sobre como usar o Cycript pode ser encontrado aqui:
http://iphonedevwiki.net/index.php/Cycript_Tricks.

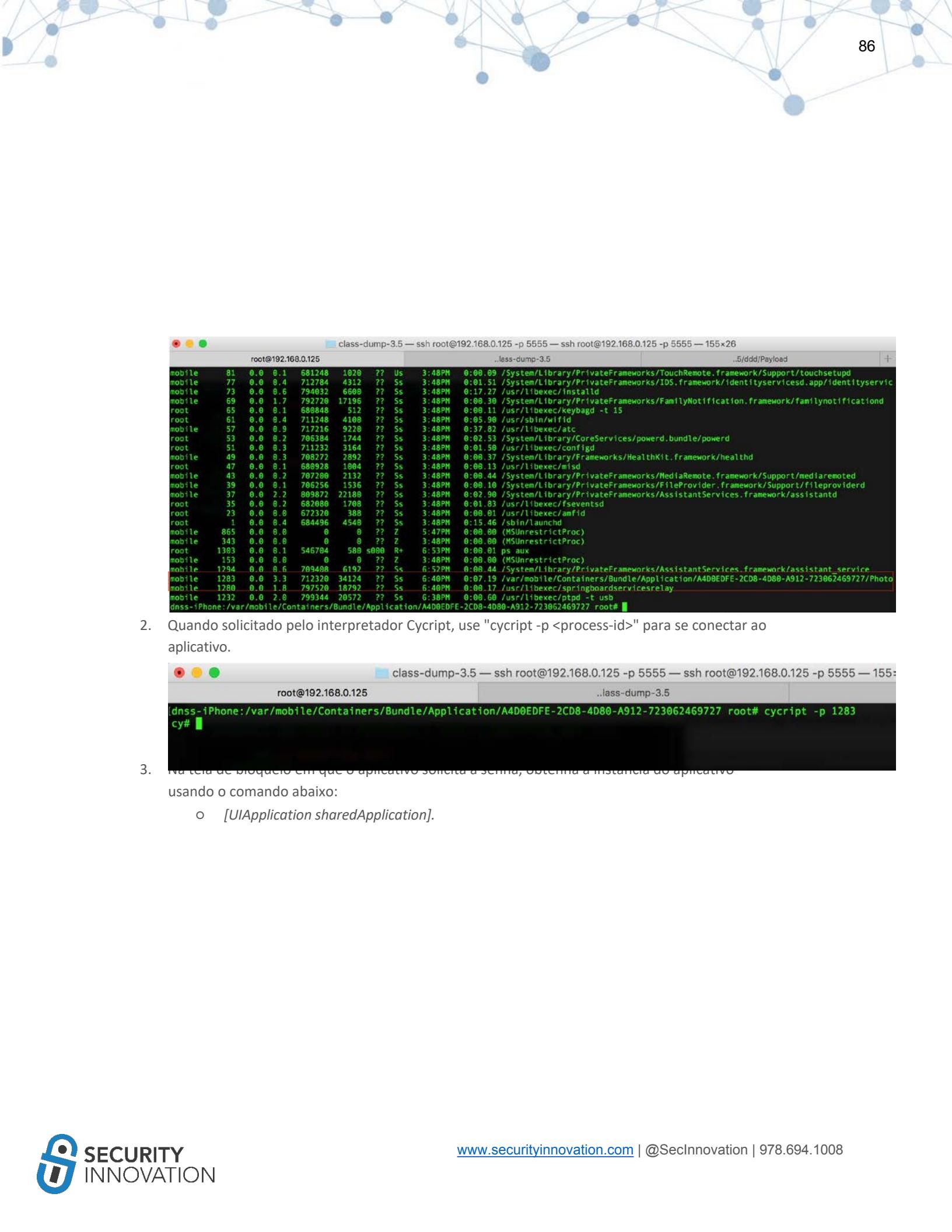
12.1.1 Uso do Cycript para invocar métodos internos

Aplicativo usado para o exemplo: Aplicativo Photo Vault versão 2.5/3.1 de:

- <https://drive.google.com/open?id=0B0b4IUTjHfRKWTRIMW1WUy14bkE>
- <https://drive.google.com/open?id=0B0b4IUTjHfRKN1I3Mk1hSDNBU0k>

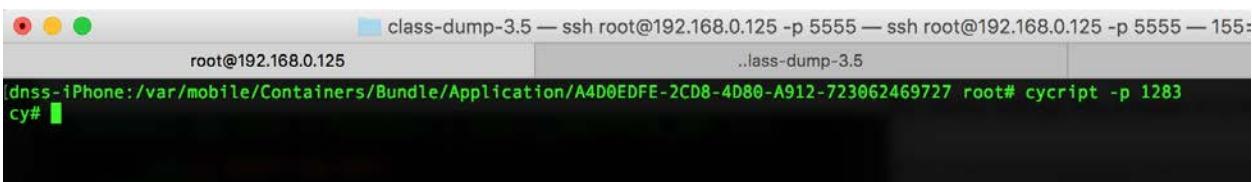
As etapas a seguir referem-se à versão 2.5.

1. Inicie o aplicativo Photo Vault no dispositivo. Quando for solicitado o PIN, defina-o como "9876". Faça o SSH no dispositivo iOS e obtenha o ID do processo do aplicativo usando o comando "ps aux".



```
class-dump-3.5 — ssh root@192.168.0.125 -p 5555 — ssh root@192.168.0.125 -p 5555 — 155x26
root@192.168.0.125 ..lass-dump-3.5 ..5/ddd/Payload
mobile 81 0.0 0.1 681248 1020 ?? Us 3:48PM 0:00.09 /System/Library/PrivateFrameworks/TouchRemote.framework/Support/touchsetupd
mobile 77 0.0 0.4 712784 4312 ?? Ss 3:48PM 0:01.51 /System/Library/PrivateFrameworks/IDS.framework/identityservicesd.app/identityservic
mobile 73 0.0 0.6 794032 6509 ?? Ss 3:48PM 0:17.27 /usr/libexec/Installd
mobile 69 0.0 1.7 792720 17196 ?? Ss 3:48PM 0:00.30 /System/Library/PrivateFrameworks/FamilyNotification.framework/familynotificationd
root 65 0.0 0.1 688848 512 ?? Ss 3:48PM 0:00.11 /usr/libexec/Keybagd -t 15
root 61 0.0 0.4 711248 4108 ?? Ss 3:48PM 0:05.90 /usr/sbin/wifid
mobile 57 0.0 0.9 717216 9228 ?? Ss 3:48PM 0:37.82 /usr/libexec/atc
root 53 0.0 0.2 706384 1744 ?? Ss 3:48PM 0:02.53 /System/Library/CoreServices/powerd.bundle/powerd
root 51 0.0 0.3 711232 3164 ?? Ss 3:48PM 0:01.50 /usr/libexec/configd
mobile 49 0.0 0.3 708272 2892 ?? Ss 3:48PM 0:00.37 /System/Library/Frameworks/HealthKit.framework/healthd
root 47 0.0 0.1 688928 1004 ?? Ss 3:48PM 0:00.13 /usr/libexec/misd
mobile 43 0.0 0.2 707200 2132 ?? Ss 3:48PM 0:00.44 /System/Library/PrivateFrameworks/MediaRemote.framework/Support/mediaremoted
mobile 39 0.0 0.1 706256 1536 ?? Ss 3:48PM 0:00.10 /System/Library/PrivateFrameworks/FileProvider.framework/Support/fileproviderd
mobile 37 0.0 2.2 809872 22188 ?? Ss 3:48PM 0:02.90 /System/Library/PrivateFrameworks/AssistantServices.framework/assistantd
root 35 0.0 0.2 682680 1708 ?? Ss 3:48PM 0:01.83 /usr/libexec/fsevents
root 23 0.0 0.8 672320 388 ?? Ss 3:48PM 0:00.81 /usr/libexec/amfid
root 1 0.0 0.4 684496 4549 ?? Ss 3:48PM 0:15.46 /sbin/launchd
mobile 865 0.0 0.8 0 0 ?? Z 5:47PM 0:00.00 (MSUnrestrictProc)
mobile 343 0.0 0.8 0 0 ?? Z 3:48PM 0:00.00 (MSUnrestrictProc)
root 1303 0.0 0.1 546704 580 s000 R+ 6:53PM 0:00.01 ps aux
mobile 153 0.0 0.8 0 0 ?? Z 3:48PM 0:00.00 (MSUnrestrictProc)
mobile 1794 0.0 0.6 709408 6192 ?? Ss 6:52PM 0:00.44 /System/Library/PrivateFrameworks/AssistantServices.framework/assistant_service
mobile 1283 0.0 3.3 712320 34124 ?? Ss 6:40PM 0:07.19 /var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727/Photo
mobile 1280 0.0 1.8 797520 18792 ?? Ss 6:40PM 0:00.17 /usr/libexec/springboardservicesrelay
dnss-iPhone:/var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727 root#
```

- Quando solicitado pelo interpretador Cycript, use "cycript -p <process-id>" para se conectar ao aplicativo.



```
root@192.168.0.125 ..lass-dump-3.5
dnss-iPhone:/var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727 root# cycript -p 1283
cy#
```

- Na tela de bloqueio em que o aplicativo solicita a senha, obtenha a instância do aplicativo usando o comando abaixo:

- o [UIApplication sharedApplication].



class-dump-3.5 — ssh root@192.168.0.125

```
Q~ unlock
[cy# [UIApplication sharedApplication]
#"<UIApplication: 0x33a880>"
```

- Obtenha a classe delegada para o aplicativo usando o comando abaixo:

- `[UIApplication sharedApplication].delegate`

class-dump-3.5 — ssh root@192.168.0.125

```
Q~ unlock
[cy# [UIApplication sharedApplication].delegate
#"<AppDelegate: 0x33ac10>"
```

-

```
function printMethods(className, isa) { var
count = new new Type("I");
var classObj = (isa != undefined) ? objc_getClass(className).constructor : objc_getClass(className);
var methods = class_copyMethodList(classObj, count);
```

```

var methodsArray = []; for(var i
= 0; i < *count; i++) { var
method = methods[i];
methodsArray.push({selector:method_getName(method), implementation:method_getImplementation(method)});
}
free(methods); return
methodsArray;
}

```

Em seguida, digite `printMethods("AppDelegate")` para imprimir a lista completa de métodos possíveis para essa tela.

```

class-dump-3.5 — ssh root@192.168.0.125 -p 5555 — ssh root@192.168.0.125 -p 5555 — 155x34
root@192.168.0.125 ..class-dump-3.5 ..5/ddd/Payload + 
Q~ unlock
(cy# [UIApplication sharedApplication].delegate
#<AppDelegate: 0x33ac19>
cy# function printMethods(className, isa) {
var count = new new Type("I");
var classObj = (isa != undefined) ? objc_getClass(className).constructor : objc_getClass(className);
var methods = class_copyMethodList(classObj, count);
var methodsArray = [];
for(var i = 0; i < *count; i++) {
var method = methods[i];
methodsArray.push({selector:method_getName(method), implementation:method_getImplementation(method)});
}
free(methods);
return methodsArray;
}
cy# printMethods("AppDelegate")
{[selector:@selector(emailPin).implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(lockController:didFinish:),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(pinLockControllerDidFinishUnlocking),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(pinLockControllerDidCancel),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(pinLockController:didFinishSelectingNewPin),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(pinLockControllerDidFinishRemovingPin),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(setTabBarController),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(documentsDirectory),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(setActionBar),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(addAlbumToOpenedAlbums),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(promptForEmail),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(pinManagement),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(checkInstaLockForAlbum),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(navigator:shouldDoOpenURL:),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(lockControllerDidCancel),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(runOnce),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(application:handleOpenURL:),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(applicationWillResignActive),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(applicationDidEnterForeground),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(applicationDidFinishLaunching),implementation:&(extern "C" id "(id, SEL, ...)).{selector:@selector(alertView:clickedButtonAtIndex:).
cy# 
OBSERVAÇÃO: essa lista de funções também pode ser encontrada usando class-dump-z. Focare na saída do class-dump por AppDelegate e procure por métodos abaixo da @interface AppDelegate na saída.

```

6. Na saída da etapa anterior, observe um método chamado "pinLockControllerDidFinishUnlocking". A saída do Class-dump-z revelará que essa função não recebe nenhum argumento e pode ser chamada diretamente.

Use o comando abaixo para chamar a função diretamente:

- `[UIApp.delegate pinLockControllerDidFinishUnlocking]`

Observe que a tela de bloqueio é ignorada.

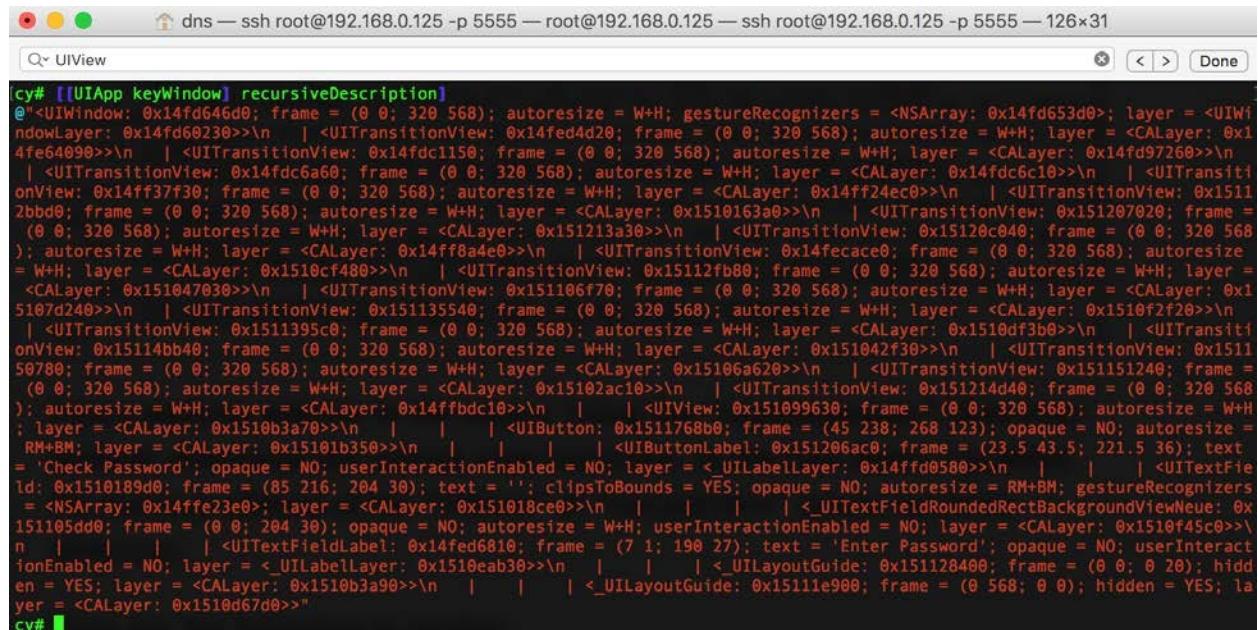
Uma abordagem semelhante pode ser usada para contornar a detecção de jailbreak em aplicativos iOS.

Pode ser difícil encontrar os detalhes do ViewController atual (tela atual) em que o usuário está. Use as instruções abaixo para localizar o nome do ViewController atual no dispositivo iOS.

Aplicativo usado para o exemplo: Aplicativo CycriptDemoDNS no link mencionado abaixo:

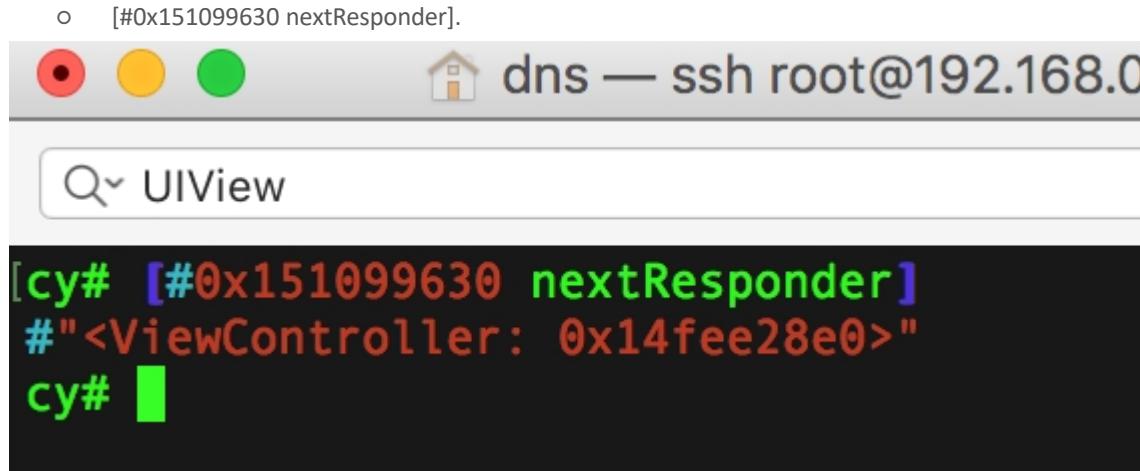
- <https://drive.google.com/open?id=0B0b4lUTjHfRKXy1pU29oZmdSUjg>

- Inicie o aplicativo e navegue até o ViewController apropriado.
- Digite o comando abaixo para obter detalhes sobre keyWindow (a janela atual que aceita eventos de toque do usuário):
 - `[[UIApp keyWindow] recursiveDescription]`



```
[cy# [[UIApp keyWindow] recursiveDescription]
@<UIWindow: 0x14fd646d0; frame = (0 0; 320 568); autoresize = W+H; gestureRecognizers = <NSArray: 0x14fd653d0>; layer = <UIWindowLayer: 0x14fd60230>>\n  | <UITransitionView: 0x14fed4d20; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14fe64090>>\n    | <UITransitionView: 0x14fdc1150; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14fd97260>>\n      | <UITransitionView: 0x14ff37f30; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14ff24ec0>>\n        | <UITransitionView: 0x15112bb0; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510163a0>>\n          | <UITransitionView: 0x151207020; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x151213a30>>\n            | <UITransitionView: 0x15120c040; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14ff8a4e0>>\n              | <UITransitionView: 0x14fecace0; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x151047030>>\n                | <UITransitionView: 0x15112fb80; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x15107d240>>\n                  | <UITransitionView: 0x151135540; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510f2f20>>\n                    | <UITransitionView: 0x151143950; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x151042f30>>\n                      | <UITransitionView: 0x15114bb40; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x15106a620>>\n                        | <UITransitionView: 0x151151240; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x15102ac10>>\n                          | <UITransitionView: 0x151214d40; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510b3a70>>\n                            | <UIView: 0x151099630; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510b3b350>>\n                              | <UIButton: 0x1511768b0; frame = (45 238; 268 123); opaque = NO; autoresize = RM+BM; layer = <CALayer: 0x15101b350>>\n                                | <UILabelLabel: 0x151206ac0; frame = (23.5 43.5; 221.5 36); text = 'Check Password'; opaque = NO; userInteractionEnabled = NO; layer = <UILabelLayer: 0x14ffd0580>>\n                                  | <UITextFileId: 0x1510189d0; frame = (85 216; 204 30); text = ''; clipsToBounds = YES; opaque = NO; autoresizingMask = RM+BM; gestureRecognizers = <NSArray: 0x14ffe23e0>; layer = <CALayer: 0x151018ce0>>\n                                    | <_UITextFieldRoundedRectBackgroundViewNeue: 0x151105dd0; frame = (0 0; 204 30); opaque = NO; autoresize = W+H; userInteractionEnabled = NO; layer = <CALayer: 0x1510f45c0>>\n                                      | <UITextLabel: 0x14fed6810; frame = (7 1; 190 27); text = 'Enter Password'; opaque = NO; userInteractionEnabled = NO; layer = <UILabelLayer: 0x1510eb30>>\n                                        | <UILayoutGuide: 0x151128400; frame = (0 0; 0 20); hidden = YES; layer = <CALayer: 0x1510b3a90>>\n                                          | <UILayoutGuide: 0x15111e900; frame = (0 568; 0 0); hidden = YES; layer = <CALayer: 0x1510d67d0>>\n
```

- `[[#0x151099630 nextResponder].`



```
[cy# [#0x151099630 nextResponder]
#"<ViewController: 0x14fee28e0>
cy#
```

O nome do ViewController atual é indicado.

- Para usar esse detalhe para ignorar a tela, obtenha os nomes das funções usando printMethod (detalhado no módulo anterior) e execute o comando abaixo para chamar a função de login diretamente, ignorando todas as verificações disponíveis.

- [#0x14fee28e0 doSuccess].
 - Onde 0x14fee28e0 é o endereço do ViewController e doSuccess é a função a ser chamada.

```

dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126x31
cy# [#0x151099630 nextResponder]
# <ViewController: 0x14fee28e0>
cy# printMethods("ViewController")
{[selector:@selector(isLoginSuccessful),implementation:&{extern "C" id "-[ViewController isLoginSuccessful]:(id, SEL, ...)},{selector:@selector(doSuccess),implementation:&{extern "C" id "-[ViewController doSuccess]:(id, SEL, ...)},{selector:@selector(doFail),implementation:&{extern "C" id "-[ViewController doFail]:(id, SEL, ...)},{selector:@selector(checkPasswordButtonPressed),implementation:&{extern "C" id "-[ViewController checkPasswordButtonPressed:]:(id, SEL, ...)},{selector:@selector(didReceiveMemoryWarning),implementation:&{extern "C" id "-[ViewController didReceiveMemoryWarning]:(id, SEL, ...)},{selector:@selector(viewDidLoad),implementation:&{extern "C" id "-[ViewController viewDidLoad]:(id, SEL, ...)}}]
cy# [#0x14fee28e0 doSuccess]
cy#

```

12.1.2 Uso do Cycript para substituir métodos internos

Aplicativo usado para o exemplo: Aplicativo CycriptDemoDNS no link mencionado abaixo:

- <https://drive.google.com/open?id=OB0b4IUTjHfRKXy1pU29oZmdSUJg>

1. Inicie o aplicativo CycriptDemoDNS no dispositivo. Entre no SSH do dispositivo iOS e obtenha o ID do processo do aplicativo usando o comando "ps aux".

```

dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126x24
mobile 73 0.0 0.6 794032 6660 ?? Ss Tue03PM 0:23.68 /usr/libexec/installd
mobile 69 0.0 1.7 792720 17056 ?? Ss Tue03PM 0:00.32 /System/Library/PrivateFrameworks/FamilyNotification.framework
root 65 0.0 0.1 680848 912 ?? Ss Tue03PM 0:00.19 /usr/libexec/keybagd -t 15
root 61 0.0 0.4 711248 4504 ?? Ss Tue03PM 0:34.62 /usr/sbin/wifid
mobile 57 0.0 0.8 717216 8448 ?? Ss Tue03PM 0:50.56 /usr/libexec/atc
root 53 0.0 0.2 706400 1888 ?? Ss Tue03PM 0:08.47 /System/Library/CoreServices/powerd.bundle/powerd
root 51 0.0 0.3 711232 3336 ?? Ss Tue03PM 0:06.04 /usr/libexec/configd
mobile 49 0.0 0.3 708272 3116 ?? Ss Tue03PM 0:01.22 /System/Library/Frameworks/HealthKit.framework/healthd
root 47 0.0 0.1 680928 1004 ?? Ss Tue03PM 0:00.15 /usr/libexec/misd
mobile 43 0.0 0.2 707200 2012 ?? Ss Tue03PM 0:00.54 /System/Library/PrivateFrameworks/MediaRemote.framework
mobile 41 0.0 0.4 710800 4052 ?? Ss Tue03PM 0:15.35 /usr/libexec/routined
mobile 39 0.0 0.1 706256 1492 ?? Ss Tue03PM 0:00.19 /System/Library/PrivateFrameworks/FileProvider.framework
mobile 37 0.0 2.2 811024 22152 ?? Ss Tue03PM 0:04.79 /System/Library/PrivateFrameworks/AssistantServices.framework
root 35 0.0 0.2 682080 1592 ?? Ss Tue03PM 0:04.83 /usr/libexec/fsevents
root 23 0.0 0.0 672320 384 ?? Ss Tue03PM 0:00.01 /usr/libexec/amfid
root 1 0.0 0.4 684496 4396 ?? Ss Tue03PM 0:41.76 /sbin/launchd
mobile 865 0.0 0.0 0 0 ?? Z Tue05PM 0:00.00 (MSUnrestrictProc)
mobile 343 0.0 0.0 0 0 ?? Z Tue03PM 0:00.00 (MSUnrestrictProc)
root 1685 0.0 0.1 546704 580 s000 R+ 9:40PM 0:00.01 ps aux
mobile 153 0.0 0.0 0 0 ?? Z Tue03PM 0:00.00 (MSUnrestrictProc)
mobile 1683 0.0 0.6 709408 6144 ?? Ss 9:40PM 0:00.50 /System/Library/PrivateFrameworks/AssistantServices.framework
mobile 1681 0.0 2.2 802768 22864 ?? Ss 9:40PM 0:00.31 /var/mobile/Containers/Bundle/Application/A573E1A9-CF68
mobile 1673 0.0 0.3 680816 3104 ?? Ss 9:39PM 0:00.14 /System/Library/PrivateFrameworks/SyncedDefaults.framework
dns:iPhone:~ root#

```

2. Substitua o método interno doSuccess no ViewController com o endereço obtido no passo 1.

```
[dns — ssh root@192.168.0.125 -p 1681
cy#]
```

3. Localize a keyWindow (a janela atual que aceita eventos de toque do usuário) usando o comando abaixo:

- o `UIApp.keyWindow`

```
[cy# UIApp.keyWindow
#<UIWindow: 0x14fd646d0; frame = (0 0; 320 568); autoresize = W+H; gestureRecognizers = <NSArray: 0x14fd653d0>; layer = <UIWindowLayer: 0x14fd60230>>
cy#]
```

4. O comando abaixo fornece o rootViewController para a keyWindow.

```
[cy# UIApp.keyWindow.rootViewController
#"<UIViewController: 0x14fd629f0>
cy#]
```

5. Use a função abaixo para obter os métodos da classe delegada encontrada na etapa anterior.

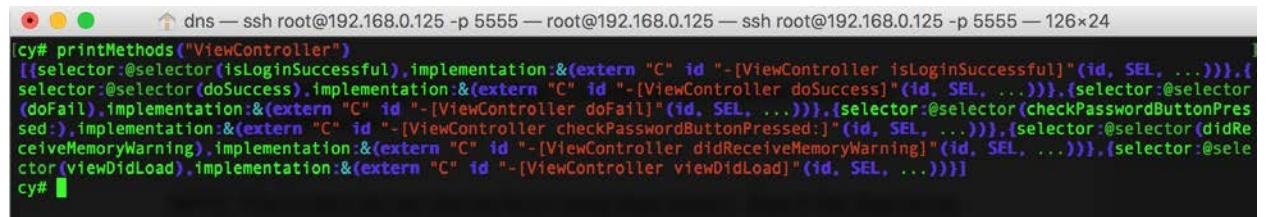
```
function printMethods(className, isa) { var
  count = new new Type("I");
  var classObj = (isa != undefined) ? objc_getClass(className).constructor : objc_getClass(className);
  var methods = class_copyMethodList(classObj, count); var
  methodsArray = [];
  for(var i = 0; i < *count; i++) {
    var method = methods[i];
    methodsArray.push({selector:method_getName(method), implementation:method_getImplementation(method)});
  }
}
```

```

        free(methods); return
        methodsArray;
    }
}

```

Digite `printMethods("ViewController")` para imprimir a lista completa de métodos possíveis para essa tela.



```

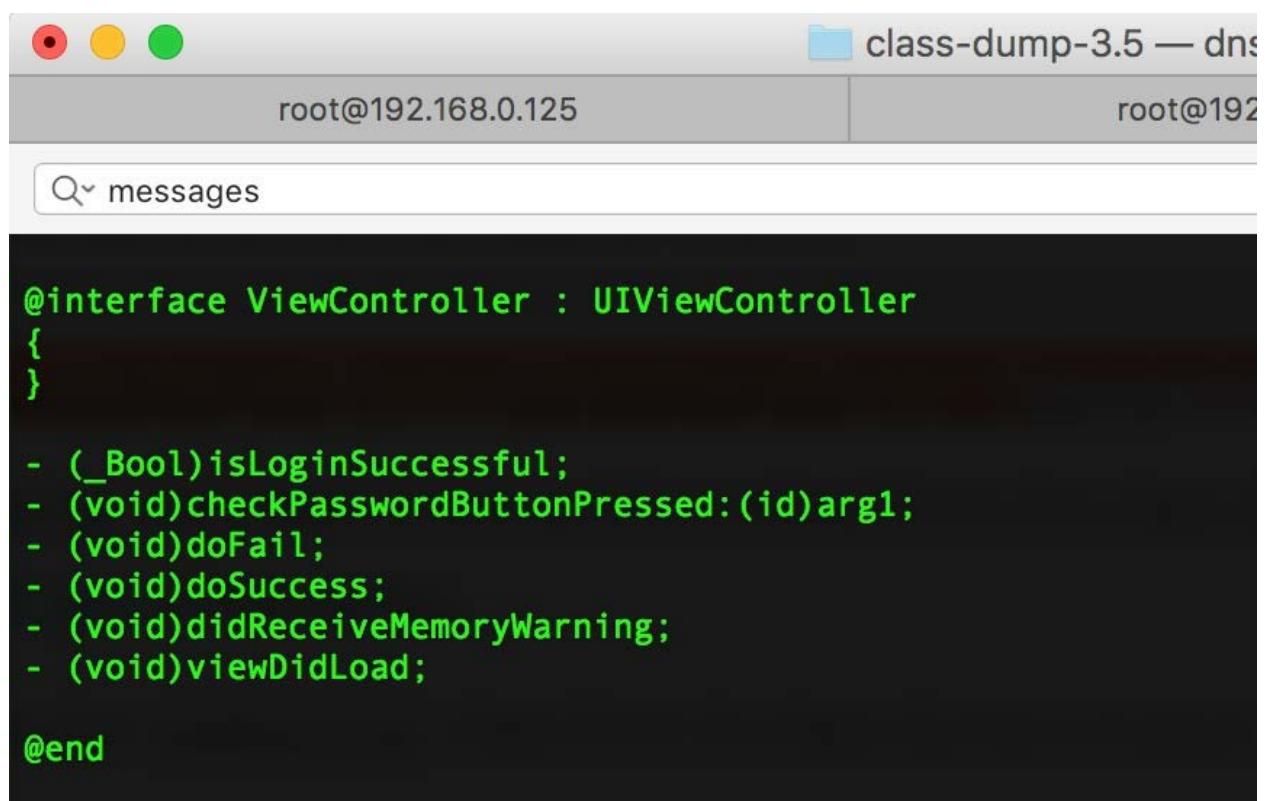
dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126x24
[cy# printMethods("ViewController")]
  [{selector:@selector(isLoginSuccessful), implementation:&{extern "C" id "-[ViewController isLoginSuccessful]"(id, SEL, ...)}}], {selector:@selector(doSuccess), implementation:&{extern "C" id "-[ViewController doSuccess]"(id, SEL, ...)}}], {selector:@selector(doFail), implementation:&{extern "C" id "-[ViewController doFail]"(id, SEL, ...)}}], {selector:@selector(checkPasswordButtonPressed), implementation:&{extern "C" id "-[ViewController checkPasswordButtonPressed:]"(id, SEL, ...)}}], {selector:@selector(didReceiveMemoryWarning), implementation:&{extern "C" id "-[ViewController didReceiveMemoryWarning]"(id, SEL, ...)}}], {selector:@selector(viewDidLoad), implementation:&{extern "C" id "-[ViewController viewDidLoad]"(id, SEL, ...)}}]
cy#

```

OBSERVAÇÃO: essa lista de funções também pode ser encontrada usando class-dump-z. Pesquise o resultado do class-dump para AppDelegate e procure a @interface nele.

OBSERVAÇÃO: "UIApp.keyWindow.rootViewController.visibleViewController" pode ser usado com frequência para exibir o view controller atual.

6. O Class-dump revela que o `isLoginSuccessful` retorna um valor BOOL que determina se o login deve ser bem-sucedido ou não.



```

class-dump-3.5 — dns
root@192.168.0.125                                     root@192.168.0.125
messages

@interface ViewController : UIViewController
{
}

- (_Bool)isLoginSuccessful;
- (void)checkPasswordButtonPressed:(id)arg1;
- (void)doFail;
- (void)doSuccess;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;

@end

```

OU

- `[UIApp.keyWindow.rootViewController isLoginSuccessful].`

```
[cy# UIApp.keyWindow.rootViewController
#"<ViewController: 0x14fd629f0>"[cy# UIApp.keyWindow.rootViewController.isLoginSuccessful()
false
cy# ]
```

8. Use o comando abaixo para modificar a função para que ela sempre retorne TRUE, independentemente dos valores ou da operação executada.

- `ViewController.prototype.isLoginSuccessful = function() { return true;};`

```
cy# ViewController.prototype.isLoginSuccessful = function() { return true;};[cy#
cy# UIApp.keyWindow.rootViewController.isLoginSuccessful()
true
cy#
cy# ]
```

9. Clique no botão "Check Password" (Verificar senha) na tela do dispositivo e observe que o login foi bem-sucedido, embora nenhuma senha tenha sido fornecida.

Uma abordagem semelhante pode ser usada para contornar a detecção de jailbreak em aplicativos iOS.

12.2 Depuração de aplicativos iOS usando LLDB

Desde a introdução do iOS 8, o GDB pode não ser uma solução viável para depuração, pois o suporte ao GDB só está disponível para arm7. Nesses casos, considere o substituto do GDB da Apple, o LLDB (saiba mais sobre o LLDB nos vídeos da WWDC da Apple). Para obter mais informações, consulte os links abaixo:

- <http://asciwwdc.com/2016/sessions/417>
- <https://developer.apple.com/videos/play/wwdc2015/402/>
- <https://developer.apple.com/videos/play/wwdc2013/413/>

Um conjunto de comandos semelhante ao GDB torna o LLDB fácil de usar para aqueles que estão familiarizados com o GDB. Além disso, o LLDB tem a reputação de ser mais rápido do que o GDB.

Para depurar um aplicativo iOS, inicie o utilitário do servidor de depuração em execução no dispositivo iOS. O servidor de depuração é o utilitário que o Xcode usa para depurar aplicativos no dispositivo iOS.

Por padrão, o *servidor de depuração* pode ser encontrado no Mac na imagem do disco de desenvolvedor do Xcode. Navegue até o seguinte

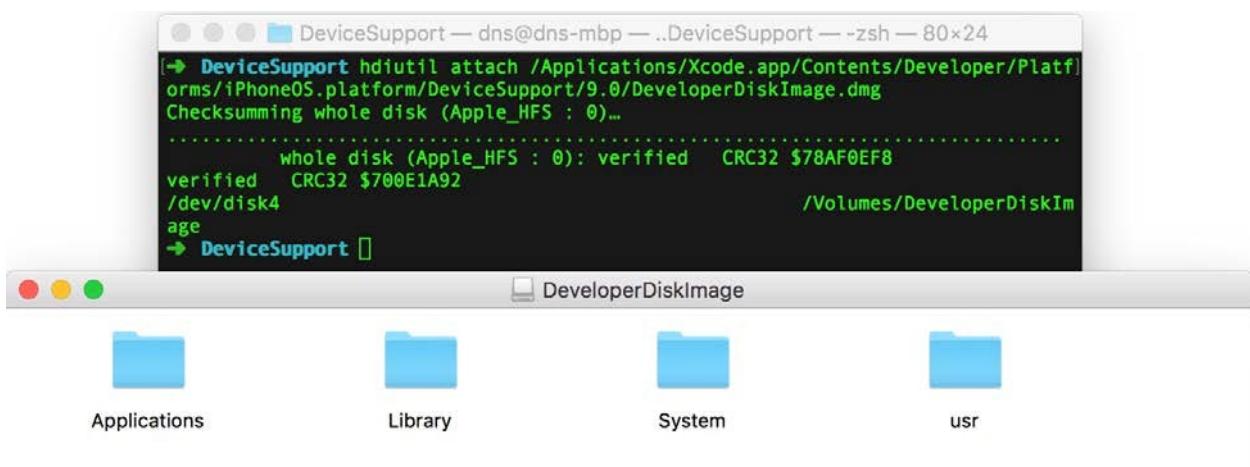
local:/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/ para visualizar todas as versões do iOS disponíveis para você.

```
● ○ ● DeviceSupport — dns@dns-mbp — ..DeviceSupport — -zsh — 80x24
[→ DeviceSupport pwd
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport
[→ DeviceSupport ls
10.0      10.2 (14C89) 8.1      8.3      9.0      9.2
10.1      8.0        8.2      8.4      9.1      9.3
→ DeviceSupport ]
```

Escolha a versão que está sendo executada no dispositivo iOS e monte a imagem de disco de desenvolvedor do Xcode relacionada em seu Mac para extrair o servidor de depuração. Nesse caso, a versão do iOS em execução é o iOS 9.0.*.

Use o comando abaixo para montar a imagem de disco do desenvolvedor:

- `hdiutil attach /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/9.0/DeveloperDiskImage.dmg`



Copie o binário do debugserver para o seu dispositivo em um local conhecido usando o comando abaixo:

- `cp /Volumes/DeveloperDiskImage/usr/bin/debugserver /Users/dns/Desktop/mobile/lldb_guide`

Por padrão, esse binário do servidor de depuração só pode depurar aplicativos que são assinados por um perfil de provisionamento específico. Isso se deve à falta de autorização para permitir `task_for_pid()`. Para evitar isso, crie um arquivo de autorização com o sinalizador `task_for_pid` definido como `true` e use-o para assinar o binário do servidor de depuração.

Veja o conteúdo do arquivo de direitos `entitlements.plist` abaixo:

```
<?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>com.apple.springboard.debugapplications</key> <true/>
  <key>run-unsigned-code</key> <true/>
  <key>get-task-allow</key> <true/>
  <key>task_for_pid-allow</key> <true/>
```

```
</dict>
</plist>
```

Renuncie ao binário do servidor de depuração usando o comando abaixo:

- `codesign -s - --entitlements entitlements.plist -f debugserver`

```
lldb_guide — dns@dns-mbp — -zsh — 80x24
..DeviceSupport ..le/lldb_guide
[→ lldb_guide pwd
/Users/dns/Desktop/mobile/lldb_guide
[→ lldb_guide cp /Volumes/DeveloperDiskImage/usr/bin/debugserver /Users/dns/Desktop/mobile/lldb_guide
[→ lldb_guide ls
debugserver
[→ lldb_guide vim entitlements.plist
[→ lldb_guide codesign -s - --entitlements entitlements.plist -f debugserver
debugserver: replacing existing signature
→ lldb_guide
```

Copie o servidor de depuração assinado para o dispositivo iOS.

```
lldb_guide — dns@dns-mbp — -zsh — 80x24
..DeviceSupport ..le/lldb_guide
[→ lldb_guide scp -P 5555 debugserver root@10.10.0.120:/usr/bin/
The authenticity of host '[10.10.0.120]:5555 ([10.10.0.120]:5555)' can't be established.
RSA key fingerprint is SHA256:WBDYcpnkRPlzssVN/4Sx1UL0BhDYUfWNTsZdIMvEWnw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.10.0.120]:5555' (RSA) to the list of known hosts
.
[root@10.10.0.120's password:
debugserver
→ lldb_guide ]
```

Para iniciar a depuração, faça o SSH no dispositivo iOS e conecte o servidor de depuração ao aplicativo iOS em execução a ser depurado.

- `debugserver *:6666 -a LinkedIn`



```
lldb_guide — ssh root@10.10.0.120 -p 5555 — ssh root@10.10.0.120
..DeviceSupport ..le/lldb_guide
dnss-iPhone:~ root# debugserver *:6666 -a LinkedIn
debugserver-@(#)PROGRAM:debugserver PROJECT:debugserver-340.3.51.1
for arm64.
Attaching to process LinkedIn...
Listening to port 6666 for a connection from *...
```

OBSERVAÇÃO: se a ferramenta não funcionar, remova o binário fat para algo diferente de arm64, pois muitas das ferramentas não funcionam em arm64.

No Mac, enquanto estiver conectado à mesma rede que seu dispositivo iOS, digite o comando abaixo para se conectar à instância do servidor de depuração:

```
lldb
(lldb) platform select remote-ios
(lldb) process connect connect://<iphone-ip>:6666
```

A conexão pode demorar um pouco. Quando conectado, a saída será semelhante à captura de tela abaixo:

```
lldb_guide — lldb — lldb — 135x24
..DeviceSupport ..le/lldb_guide root@10.10.0.120 ssh roo... lldb
(lldb) platform select remote-ios
Platform: remote-ios
Connected: no
SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
(lldb) process connect connect://10.10.0.120:6666

(lldb)
error: Process 2400 is currently being debugged, kill the process before connecting.
Process 2400 stopped
* thread #1: tid = 0x314b5, 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
  frame #0: 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
-> 0x199104c30 <+8>: ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
  0x199104c34 <+0>: movn  x16, #0x1f
  0x199104c38 <+4>: svc    #0x80
  0x199104c3c <+8>: ret

(lldb)
```

Agora você pode depurar o aplicativo.

Veja a seguir alguns dos comandos básicos do LLDB.

"po" pode ser usado para imprimir as instâncias de objeto e os delegados por meio dos comandos abaixo:

- *po [UIApplication sharedApplication]*
- *po [[UIApplication sharedApplication] delegate]*

```

[~] lldb_guide lldb
[(lldb) platform select remote-ios
Platform: remote-ios
Connected: no
SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
[(lldb) process connect connect://192.168.0.125:6666
Process 845 stopped
* thread #1: tid = 0x3ebf, 0x0000000198fd8c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-thread
  frame #0: 0x0000000198fd8c30 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
-> 0x198fd8c30 <+8>; ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
  0x198fd8c34 <+0>; movn x16, #0x1f
  0x198fd8c38 <+4>; svc #0x80
  0x198fd8c3c <+8>; ret
[(lldb) po [UIApplication sharedApplication]
<UIApplication: 0x145e97e70>

[(lldb) po [[UIApplication sharedApplication] delegate]
<LinkedInAppDelegate: 0x145d572e0>

(lldb)

```

Para despejar todos os símbolos do executável principal e de todas as bibliotecas compartilhadas, use o comando "image dump symtab".

Para despejar todas as seções do executável principal e todas as bibliotecas compartilhadas, use o comando "image dump sections".

Para listar o executável principal e todas as bibliotecas compartilhadas dependentes, juntamente com sua localização na memória, use "image list".

Para despejar as informações armazenadas para um endereço bruto no executável ou em qualquer biblioteca compartilhada, use "image lookup --address 0x00011e4".

Para procurar funções que correspondam a uma expressão regular em um binário, use os comandos abaixo:

image lookup -r -n <FUNC_REGEX> → localiza símbolos de depuração

image lookup -r -s <FUNC_REGEX> → localiza símbolos de não

depuração OBSERVAÇÃO: forneça uma lista de binários como argumentos para limitar a pesquisa.

É possível definir pontos de interrupção em qualquer um dos métodos do aplicativo iOS. Para fazer isso, localize os nomes dos métodos. Isso pode ser obtido por meio de um class-dump-z do Hopper.

Descompile o binário usando o Clutch, extraia-o do dispositivo iOS e execute-o no Hopper para ver alguns dos nomes dos métodos. Para extrair o binário do dispositivo, use:

- `scp -P 5555 root@192.168.0.125:"/tmp/LinkedIn_Patches" ./`

O ponto de interrupção pode ser definido usando uma sintaxe da forma mostrada abaixo:

```
breakpoint set --name "-[NSString stringWithFormat:]" b -
```

```
[NSString stringWithFormat:]
```

```
b -[AppDelegate pinLockControllerDidFinishUnlocking:]
```

Use "target stop-hook add" para adicionar qualquer comando ou script a ser executado sempre que um ponto de interrupção for atingido.

Use "process continue" para continuar depois que o ponto de interrupção for definido.

Consulte <http://lldb.llvm.org/lldb-gdb.html> para obter um mapa de comandos GDB para LLDB, juntamente com um conjunto de comandos detalhado para LLDB.

OBSERVAÇÃO: para usar o LLDB para um binário local do iOS em um Mac, a maioria das etapas permanece a mesma. As únicas alterações serão na inicialização do lldb:

```
(lldb) platform select remote-ios
```

```
(lldb) target create --arch arm64 /Users/dns/Desktop/mobile/lldb_guide/newiOSBinaryWithoutPIE
```

13. Engenharia reversa usando o funil

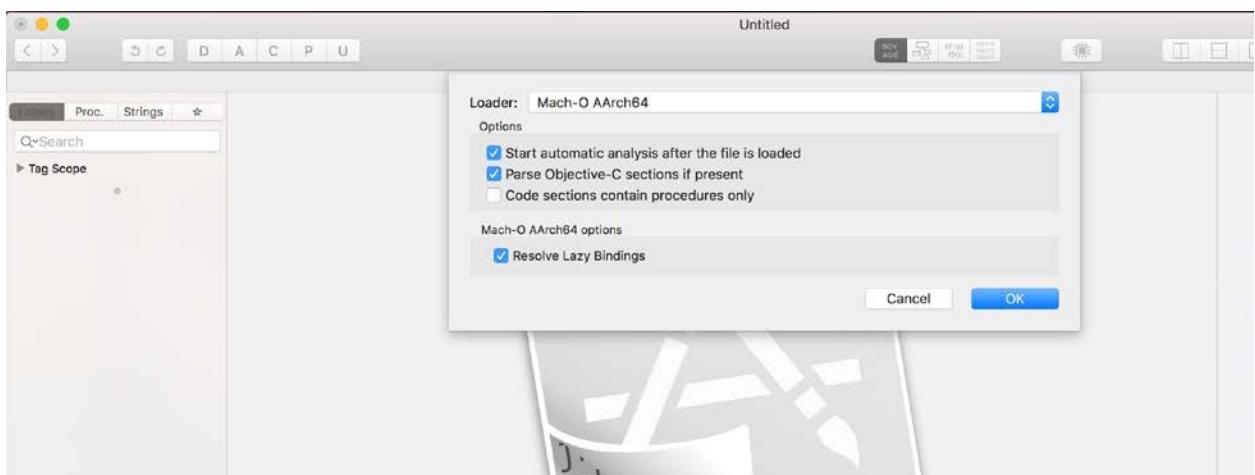
O Hopper é o melhor (e geralmente o mais barato) desmontador para engenharia reversa de aplicativos iOS. A versão de demonstração permite a engenharia reversa, mas é limitada no tempo e não permite o despejo dos executáveis recém-modificados. O Hopper pode ser usado para reverter aplicativos iOS de 32 bits e 64 bits.

Aplicativo usado para o exemplo: Aplicativo JailbreakDetectionDemoDNS no link mencionado abaixo:

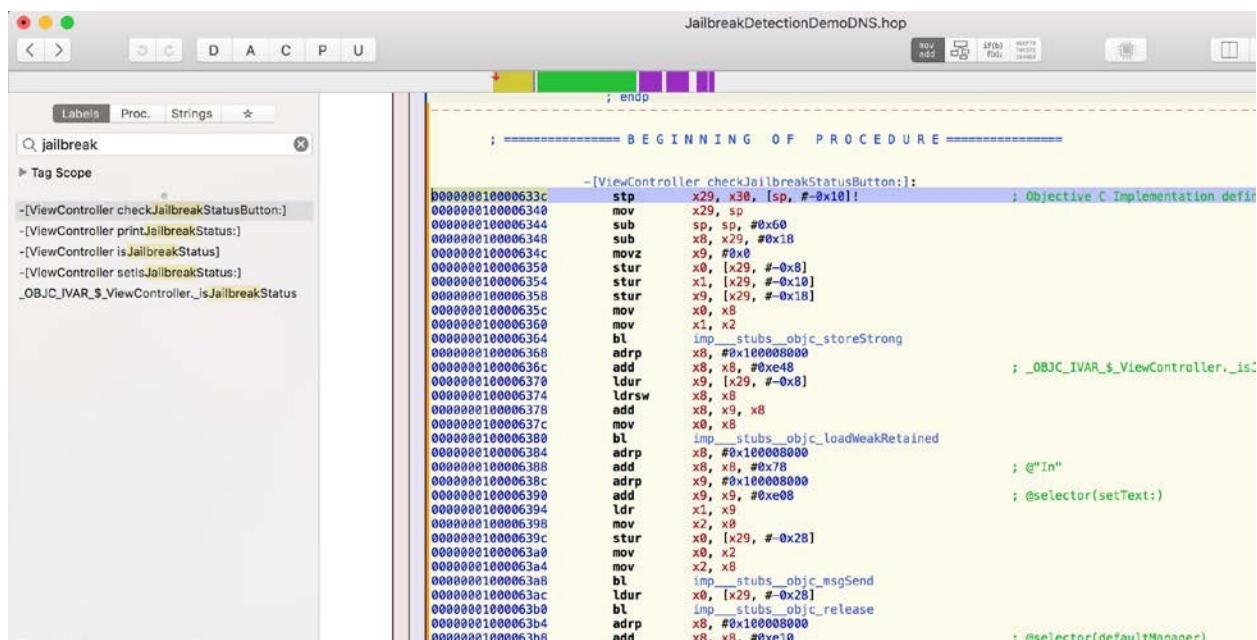
- <https://drive.google.com/open?id=0B0b4lUTjHfRKNTdCVzlvN2ZReVk>

Instale o aplicativo JailbreakDetectionDemoDNS usando o Xcode no dispositivo iOS. (As etapas serão diferentes se o arquivo .IPA for usado em vez da instalação via Xcode).

1. Extraia o arquivo ipa do JailbreakDetectionDemoDNS e abra o binário do JailbreakDetectionDemoDNS no Hopper.



2. Quando a análise automática estiver concluída, observe os rótulos presentes no binário. Para ignorar a detecção de jailbreak, pesquise a palavra-chave "jailbreak".



The screenshot shows the Hopper Disassembler interface with the file 'JailbreakDetectionDemoDNS.hop' open. The assembly code for the 'checkJailbreakStatusButton' method is displayed. The code is annotated with Objective-C implementation details, such as selector names like '@selector(setText:)'. The assembly instructions include stp, mov, sub, add, ldr, and bl (branch) instructions, along with various memory addresses and register references.

```

;----- BEGINNING OF PROCEDURE -----
-[ViewController checkJailbreakStatusButton];
000000010000633c    stp    x29, x30, [sp, #-0x10]! ; Objective C Implementation defi
0000000100006340    mov    x29, sp
0000000100006344    sub    sp, sp, #0x60
0000000100006348    sub    x8, x29, #0x18
000000010000634c    movz   x9, #0x0
0000000100006350    stur   x0, [x29, #-0x8]
0000000100006354    stur   x1, [x29, #-0x10]
0000000100006358    stur   x9, [x29, #-0x18]
000000010000635c    mov    x0, x8
0000000100006360    mov    x1, x2
0000000100006364    bl    imp_stubs_objc_storeStrong
0000000100006368    adrp   x8, #0x100008000
000000010000636c    add    x8, x8, #0xe48
0000000100006370    ldrur x9, [x29, #-0x8] ; _OBJC_IVAR_$_ViewController._isJ
0000000100006374    ldrsw  x8, x8
0000000100006378    add    x8, x9, x8
000000010000637c    mov    x0, x8
0000000100006380    bl    imp_stubs_objc_loadWeakRetained
0000000100006384    adrp   x8, #0x100008000
0000000100006388    add    x8, x8, #0x78
000000010000638c    adrp   x9, #0x100008000
0000000100006390    add    x9, x9, #0xe08 ; @In"
0000000100006394    ldr    x1, x9 ; @selector(setText:)
0000000100006398    mov    x2, x0
000000010000639c    stur   x0, [x29, #-0x28]
00000001000063a0    mov    x0, x2
00000001000063a4    mov    x2, x8
00000001000063a8    bl    imp_stubs_objc_msgSend
00000001000063ac    ldrur x0, [x29, #-0x28]
00000001000063b0    bl    imp_stubs_objc_release
00000001000063b4    adrp   x8, #0x100008000 ; @selector(defaultManager)
00000001000063b8    add    x8, x8, #0xe10

```

Dê uma olhada em "checkJailbreakStatusButton". Veja o código desmontado à direita.

3. Um recurso interessante do Hopper é a opção de ver o pseudocódigo de uma função. Para ver o pseudocódigo do aplicativo, clique no botão mostrado na captura de tela a seguir.

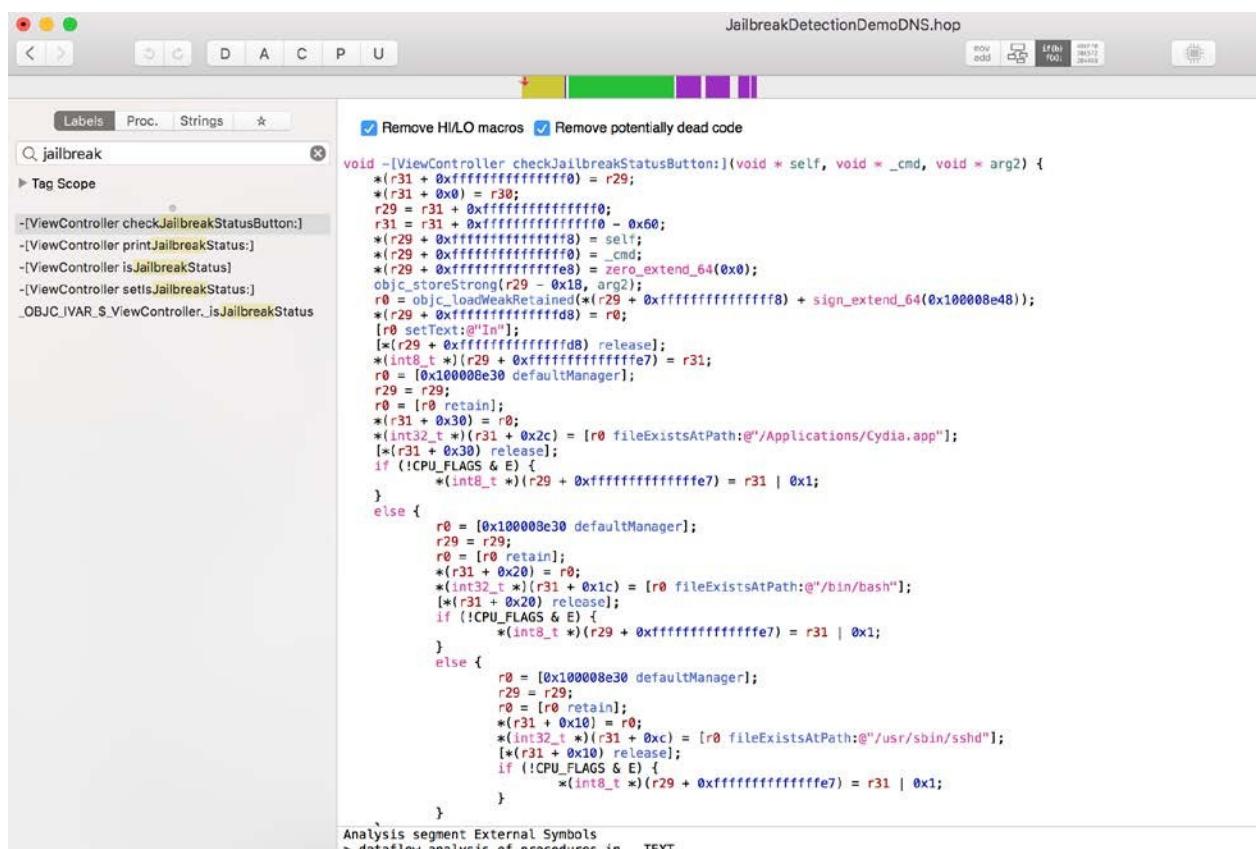


JailbreakDetectionDemoDNS.hop

```

Labels Proc. Strings ★
Q Jailbreak
▶ Tag Scope
-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus]
_OBJC_IVAR_$_ViewController._isJailbreakStatus
; ====== B E G I N N I N G O F P R O C E D U R E ======
-[ViewController checkJailbreakStatusButton:]:
000000010000633c    stp    x29, x0, [sp, #-0x10]! ; Objective C Implementation def
0000000100006340    mov    x29, sp
0000000100006344    sub    sp, sp, #0x60
0000000100006348    sub    x8, x29, #0x18
000000010000634c    movz   x9, #0x0
0000000100006350    stur   x0, [x29, #-0x8]
0000000100006354    stur   x1, [x29, #-0x10]
0000000100006358    stur   x9, [x29, #-0x18]
000000010000635c    mov    x0, x8
0000000100006360    mov    x1, x2
0000000100006364    bl     imp_stubs_objc_storeStrong
0000000100006368    addp   x8, #0x1000000000
000000010000636c    add    x8, x8, #0xe48
0000000100006370    ldrur  x9, [x29, #-0x8]
0000000100006374    ldrsw  x8, x8
0000000100006378    add    x8, x9, x8
000000010000637c    mov    x0, x8
0000000100006380    bl     imp_stubs_objc_loadWeakRetained
0000000100006384    addp   x8, #0x1000000000
0000000100006388    add    x8, x8, #0x78
000000010000638c    addp   x9, #0x1000000000
0000000100006390    add    x9, x9, #0xe08
0000000100006394    ldr    x1, x9
0000000100006398    mov    x2, x0
000000010000639c    stur   x0, [x29, #-0x28]
00000001000063a0    mov    x0, x2
00000001000063a4    mov    x2, x8
00000001000063a8    bl     imp_stubs_objc_msgSend
00000001000063ac    ldrur  x0, [x29, #-0x28]
00000001000063b0    bl     imp_stubs_objc_release

```



The screenshot shows a debugger interface with the title "JailbreakDetectionDemoDNS.hop". The assembly code is displayed in a scrollable window, with several lines highlighted in yellow. The code is written in a pseudo-assembly language with some Objective-C syntax. It includes annotations such as "Remove Hi/Lo macros" and "Remove potentially dead code". The assembly instructions involve registers r29, r31, and r0, along with memory operations like zero_extend_64 and sign_extend_64. The code appears to be checking for jailbreak status by comparing file paths like "/Applications/Cydia.app" and "/bin/bash" against "/usr/sbin/sshd".

```

void -[ViewController checkJailbreakStatusButton:](void * self, void * _cmd, void * arg2) {
    *(r31 + 0xfffffffffffff0) = r29;
    *(r31 + 0x0) = r30;
    r29 = r31 + 0xfffffffffffff0;
    r31 = r31 + 0xfffffffffffff0 - 0x0;
    *(r29 + 0xfffffffffffff8) = self;
    *(r29 + 0xfffffffffffff0) = _cmd;
    *(r29 + 0xfffffffffffffe8) = zero_extend_64(0x0);
    objc_storeStrong(r29 - 0x18, arg2);
    r0 = objc_loadWeakRetained(*(r29 + 0xfffffffffffff8) + sign_extend_64(0x100008e48));
    *(r29 + 0xfffffffffffffd8) = r0;
    [r0 setText:@"In"];
    [(int8_t *)r29 + 0xffffffffffffd8] release;
    *(int8_t *)r29 + 0xfffffffffffffe7 = r31;
    r0 = [0x100008e30 defaultManager];
    r29 = r29;
    r0 = [r0 retain];
    *(r31 + 0x30) = r0;
    *(int32_t *)r31 + 0x2c = [r0 fileExistsAtPath:@"/Applications/Cydia.app"];
    [(r31 + 0x30) release];
    if (!CPU_FLAGS & E) {
        *(int8_t *)r29 + 0xfffffffffffffe7 = r31 | 0x1;
    } else {
        r0 = [0x100008e30 defaultManager];
        r29 = r29;
        r0 = [r0 retain];
        *(r31 + 0x20) = r0;
        *(int32_t *)r31 + 0x1c = [r0 fileExistsAtPath:@"/bin/bash"];
        [(r31 + 0x20) release];
        if (!CPU_FLAGS & E) {
            *(int8_t *)r29 + 0xfffffffffffffe7 = r31 | 0x1;
        } else {
            r0 = [0x100008e30 defaultManager];
            r29 = r29;
            r0 = [r0 retain];
            *(r31 + 0x10) = r0;
            *(int32_t *)r31 + 0x1c = [r0 fileExistsAtPath:@"/usr/sbin/sshd"];
            [(r31 + 0x10) release];
            if (!CPU_FLAGS & E) {
                *(int8_t *)r29 + 0xfffffffffffffe7 = r31 | 0x1;
            }
        }
    }
}

```

Analysis segment External Symbols
~ dataflow analysis of procedures in TEXT

O pseudocódigo facilita a compreensão do que o código faz.

4. A captura de tela abaixo mostra como o aplicativo procura a presença de binários como Cydia, bash e sshd no sistema de arquivos.

JailbreakDetectionDemoDNS.hop

Labels Proc. Strings *

Q jailbreak

▶ Tag Scope

-[ViewController checkJailbreakStatusButton:]

-[ViewController printJailbreakStatus:]

-[ViewController isJailbreakStatus]

-[ViewController setisJailbreakStatus:]

_OBJC_IVAR_\$_ViewController._isJailbreakStatus

```
r29 = r31 + 0xfffffffffffff0;
r31 = r31 + 0xfffffffffffff0 - 0x60;
*(r29 + 0xfffffffffffff0) = self;
*(r29 + 0xfffffffffffff0) = _cmd;
*(r29 + 0xfffffffffffff0) = zero_extend_64(0x0);
objc_storeStrong(r29 - 0x18, arg2);
r0 = objc_loadweakRetained(*(r29 + 0xfffffffffffff0) + sign_extend_64(0x10000e48));
*(r29 + 0xfffffffffffff0) = r0;
[r0 setText:@"In"];
[(r29 + 0xfffffffffffffd8) release];
*int8_t *(r29 + 0xfffffffffffffe7) = r31;
r0 = [0x10000e30 defaultManager];
r29 = r29;
r0 = [r0 retain];
*(r31 + 0x30) = r0;
*(int32_t *)(r31 + 0x2c) = [r0 fileExistsAtPath:@"Applications/Cydia.app"];
[(r31 + 0x30) release];
if (!CPU_FLAGS & E) {
    *int8_t *(r29 + 0xfffffffffffffe7) = r31 | 0x1;
}
else {
    r0 = [0x10000e30 defaultManager];
    r29 = r29;
    r0 = [r0 retain];
    *(r31 + 0x20) = r0;
    *(int32_t *)(r31 + 0x1c) = [r0 fileExistsAtPath:@"bin/bash"];
    [(r31 + 0x20) release];
    if (!CPU_FLAGS & E) {
        *int8_t *(r29 + 0xfffffffffffffe7) = r31 | 0x1;
    }
    else {
        r0 = [0x10000e30 defaultManager];
        r29 = r29;
        r0 = [r0 retain];
        *(r31 + 0x10) = r0;
        *(int32_t *)(r31 + 0xc) = [r0 fileExistsAtPath:@"usr/sbin/sshd"];
        [(r31 + 0x10) release];
        if (!CPU_FLAGS & E) {
            *int8_t *(r29 + 0xfffffffffffffe7) = r31 | 0x1;
        }
    }
}
[*r29 + 0xfffffffffffff8] printJailbreakStatus:*(int8_t *)(r29 + 0xfffffffffffffe7) & 0x1;
objc_storeStrong(r29 - 0x18, zero_extend_64(0x0));
r31 = r29;
return;
```

Analysis segment External Symbols

> dataflow analysis of procedures in __TEXT

> dataflow analysis of procedures in __DATA

> dataflow analysis of procedures in __LINKEDIT

> dataflow analysis of procedures in External Symbols

definirá c

5. Juntamente com o pseudocódigo, é sempre importante examinar o gráfico de fluxo de controle (CFG) que o Hopper fornece para entender o código e a ramificação. Para obter o CFG, clique no botão mostrado na captura de tela a seguir.

```

Labels Proc. Strings ☆
Q. jailbreak
▶ Tag Scope

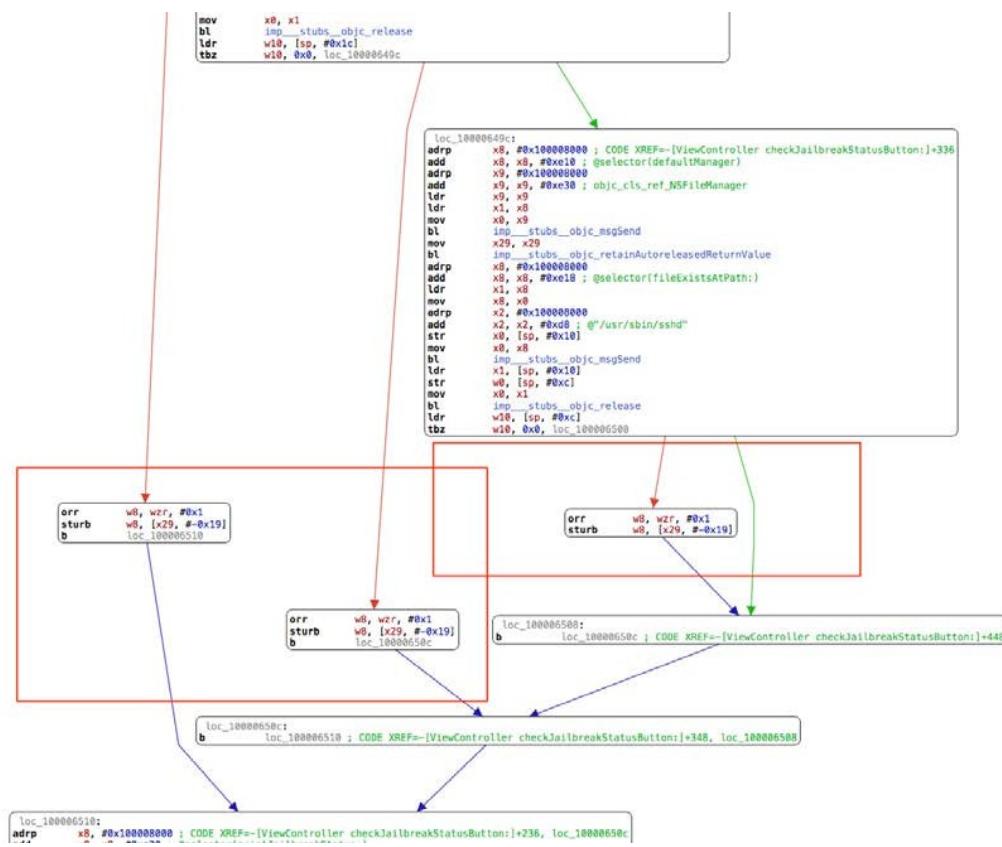
-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

----- BEGINNING OF PROCEDURE -----
; Objective-C Implementation defined at 0x10000010 : (instance method), DATA-MOD=0x10000010
mov    x0, x0, lsr #4<x0>[1]
add    x1, #0, #0x80
sub    x0, x0, #0x80
movz   x5, #0x80
str    x5, [x2, #0x80]
stur   x5, [x2, #0x80]
str    x5, [x2, #0x80]
mov    x0, x0
mov    x1, x2
bl    lmc_startObjc_loadHeapRetained
adrp   x5, #0x10000000
add    x5, x0, #0x8040 ; _OBJC_IVAR_$_ViewController._isJailbreakStatus
ldr    x5, [x5, #0x80]
ldr    x5, [x5, #0x80]
add    x5, x0, #0x80
mov    x0, x0
str    x5, [x2, #0x80]
mov    x0, x0
mov    x1, x0
bl    lmc_startObjc_mugSend
ldr    x5, [x0, #0x80]
bl    lmc_startObjc_release
addp   x5, #0x8010 ; _objc_SelectorInfoFileAtPathManager
adrp   x5, #0x10000000
add    x5, x0, #0x8040 ; _objc_ivar_ref_NSPFileManager
str    x5, [x2, #0x8010]
stur   x5, [x2, #0x8010]
ldr    x5, [x5, #0x80]
add    x5, x0, #0x80
str    x5, [x2, #0x8010]
mov    x0, x0
mov    x1, x0
bl    lmc_startObjc_mugSend
ldr    x5, [x0, #0x80]
addp   x5, #0x8010 ; _objc_SelectorInfoFileAtPathReturnValue
adrp   x5, #0x10000000
add    x5, x0, #0x8030 ; _objc_SelectorInfoFileAtPathPath
ldr    x5, [x5, #0x80]
add    x5, x0, #0x80
str    x5, [x2, #0x8030]
mov    x0, x0
mov    x1, x0
bl    lmc_startObjc_mugSend
ldr    x5, [x0, #0x80]
addp   x5, #0x8010 ; _objc_SelectorInfoFileAtPathPath
adrp   x5, #0x10000000
add    x5, x0, #0x8030 ; _objc_SelectorInfoFileAtPathPath
ldr    x5, [x5, #0x80]
add    x5, x0, #0x80
str    x5, [x2, #0x8030]
mov    x0, x0

```

Analysis comment External Symbolic

6. Use esse CFG e mapeie-o para o pseudocódigo apropriado.



Observe o CFG e os três nós que podem afetar as condições de ramificação. Para ignorar a detecção de jailbreak, use qualquer um dos métodos abaixo:

- Converta o "tbz" no CFG para "tbnz"
- Converter "tbz" em "endereço jmp"
- Converter "tbz" em "mov w10, 0x0"
- NOP de todas as operações que definem o booleano de status do jailbreak como verdadeiro.

7. Elimine todas as operações que definem o booleano de status do jailbreak como verdadeiro. Faça isso selecionando a instrução e escolhendo NOP em Modificar -> região NOP.

```

Leaving CFG mode: No procedure at this address.

0000001000063c4    sturb   w2r, [x29, #-0x19]
0000001000063c8    ldr     x9, x9
0000001000063cc    ldr     x1, x8
0000001000063d0    mov     x0, x9
0000001000063d4    bl      imp_stubs_objc_msgSend
0000001000063d8    mov     x29, x29
0000001000063dc    bl      imp_stubs_objc_retainAutoreleasedReturnValue
0000001000063e0    adrp    x8, #0x100008000
0000001000063e4    add     x8, #0xe18
0000001000063e8    ldr     x1, x8
0000001000063ec    mov     x2, x0
0000001000063f0    adrp    x8, #0x100008000
0000001000063f4    add     x8, x8, #0x98
0000001000063f8    str     x0, [sp, #0x30]
0000001000063fc    mov     x0, x2
000000100006400    mov     x2, x8
000000100006404    bl      imp_stubs_objc_msgSend
000000100006408    ldr     x1, [sp, #0x30]
00000010000640c    str     w0, [sp, #0x2c]
000000100006410    mov     x0, x1
000000100006414    bl      imp_stubs_objc_release
000000100006418    ldr     w10, [sp, #0x2c]
000000100006420    tbz    w10, 0x0, -[ViewController checkJailbreakStatusButton:]+240
000000100006428    nop
000000100006424    sturb   w8, [x29, #-0x19]
000000100006428    b      -[ViewController checkJailbreakStatusButton:] +468
00000010000642c    adrp    x8, #0x100008000
000000100006430    add     x8, #0xe18
000000100006434    adrp    x9, #0x100008000
000000100006438    add     x9, x9, #0xe30
00000010000643c    ldr     x9, x9
000000100006440    ldr     x1, x8
000000100006444    mov     x0, x9
000000100006448    bl      imp_stubs_objc_msgSend
00000010000644c    mov     x29, x29
000000100006450    bl      imp_stubs_objc_retainAutoreleasedReturnValue
000000100006454    adrp    x8, #0x100008000
000000100006458    add     x8, x8, #0xe18
00000010000645c    ldr     x1, x8
000000100006460    mov     x8, x0
000000100006464    adrp    x2, #0x100008000
000000100006468    add     v2, v2, #0xvhR

```

```

0000000100006418      w10, [sp, #0x2c]
000000010000641c      tbz   w10, 0x0, -[ViewController checkJailbreakStatusButton:] +240
0000000100006420      nop
0000000100006424      nop
0000000100006428      b    -[ViewController checkJailbreakStatusButton:] +468
000000010000642c      adrp  x8, #0x100008000 ; CODE XREF=-[ViewController chec
0000000100006430      add   x8, x8, #0xe10 ; @selector(defaultManager)
0000000100006434      adrp  x9, #0x100008000
0000000100006438      add   x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
000000010000643c      ldr   x9, x9
0000000100006440      ldr   x1, x8
0000000100006444      mov   x0, x9
0000000100006448      bl    imp_stubs_objc_msgSend
000000010000644c      mov   x29, x29
0000000100006450      bl    imp_stubs_objc_retainAutoreleasedReturnValue
0000000100006454      adrp  x8, #0x100008000
0000000100006458      add   x8, x8, #0xe18 ; @selector(fileExistsAtPath:)
000000010000645c      ldr   x1, x8
0000000100006460      mov   x8, x0
0000000100006464      adrp  x2, #0x100008000
0000000100006468      add   x2, x2, #0xb8 ; @"/bin/bash"
000000010000646c      str   x0, [sp, #0x20]
0000000100006470      mov   x0, x8
0000000100006474      bl    imp_stubs_objc_msgSend
0000000100006478      ldr   x1, [sp, #0x20]
000000010000647c      str   w0, [sp, #0x1c]
0000000100006480      mov   x0, x1
0000000100006484      bl    imp_stubs_objc_release
0000000100006488      ldr   w10, [sp, #0x1c]
000000010000648c      tbz   w10, 0x0, -[ViewController checkJailbreakStatusButton:] +352
0000000100006490      nop
0000000100006494      nop
0000000100006498      b    -[ViewController checkJailbreakStatusButton:] +464
000000010000649c      adrp  x8, #0x100008000 ; CODE XREF=-[ViewController chec
00000001000064a0      add   x8, x8, #0xe10 ; @selector(defaultManager)
00000001000064a4      adrp  x9, #0x100008000
00000001000064a8      add   x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
00000001000064ac      ldr   x9, x9
00000001000064b0      ldr   x1, x8
00000001000064b4      mov   x0, x9
00000001000064b8      bl    imp_stubs_objc_msgSend
00000001000064bc      mov   x29, x29
00000001000064c0      bl    imp_stubs_objc_retainAutoreleasedReturnValue
00000001000064c4      adrp  x8, #0x100008000

```



```

000000010000648c    tbz    w10, 0x0, -[ViewController checkJailbreakStatusButton:] +352
0000000100006490    nop
0000000100006494    nop
0000000100006498    b      -[ViewController checkJailbreakStatusButton:] +464
000000010000649c    adrp   x8, #0x100008000 ; CODE XREF=-[ViewController checkJailbreakStatusButton:] +464
00000001000064a0    add    x8, x8, #0xe10 ; @selector(defaultManager)
00000001000064a4    adrp   x9, #0x100008000 ; @selector(defaultManager)
00000001000064a8    add    x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
00000001000064ac    ldr    x9, x9
00000001000064b0    ldr    x1, x8
00000001000064b4    mov    x0, x9
00000001000064b8    bl     imp_stubs_objc_msgSend
00000001000064bc    mov    x29, x29
00000001000064c0    bl     imp_stubs_objc_retainAutoreleasedReturnValue
00000001000064c4    adrp   x8, #0x100008000 ; @selector(fileExistsAtPath:)
00000001000064c8    add    x8, x8, #0xe18 ; @selector(fileExistsAtPath:)
00000001000064cc    ldr    x1, x8
00000001000064d0    mov    x8, x0
00000001000064d4    adrp   x2, #0x100008000 ; @"/usr/sbin/sshd"
00000001000064d8    add    x2, x2, #0xd8
00000001000064dc    str    x0, [sp, #0x10]
00000001000064e0    mov    x0, x8
00000001000064e4    bl     imp_stubs_objc_msgSend
00000001000064e8    ldr    x1, [sp, #0x10]
00000001000064ec    str    w0, [sp, #0xc]
00000001000064f0    mov    x0, x1
00000001000064f4    bl     imp_stubs_objc_release
00000001000064f8    ldr    w10, [sp, #0xc]
00000001000064fc    tbz   w10, 0x0, -[ViewController checkJailbreakStatusButton:] +460
0000000100006500    nop
0000000100006504    nop
0000000100006508    b      -[ViewController checkJailbreakStatusButton:] +464 ; CODE XREF=-[ViewController checkJailbreakStatusButton:] +464
000000010000650c    b      -[ViewController checkJailbreakStatusButton:] +468 ; CODE XREF=-[ViewController checkJailbreakStatusButton:] +468
0000000100006510    adro   x8, #0x100008000 ; CODE XREF=-[ViewController checkJailbreakStatusButton:] +468

```

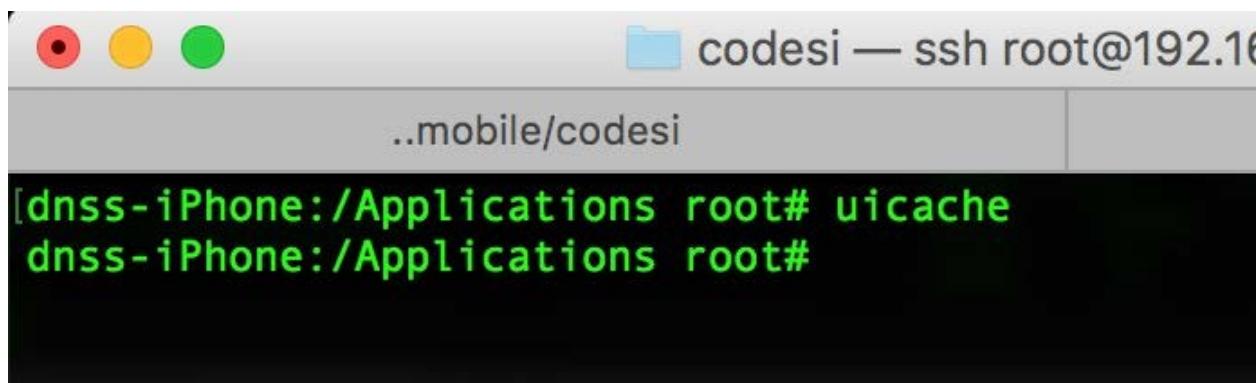
8. Selecione File -> Produce New Executable (Arquivo -> Produzir novo executável) para salvar o binário modificado.



- Instale o binário modificado de volta no dispositivo iOS usando as etapas mencionadas em "3.2 Método II
- Sem uma conta de desenvolvedor válida" e "Módulo 4" (método .app modificado).

```
codesi — dns@dns-mbp — ..mobile/codesi — -zsh — 80x24
→ JailbreakDetectionDemoDNS.app cp ~/Desktop/JailbreakDetectionDemoDNS .
→ JailbreakDetectionDemoDNS.app cd ..
→ codesi codesign -v -fs dnsioscodesigncert1 JailbreakDetectionDemoDNS.app
JailbreakDetectionDemoDNS.app: replacing existing signature
JailbreakDetectionDemoDNS.app: signed app bundle with Mach-O thin (arm64) [com.dns.JailbreakDetectionDemoDNS]
→ codesi cd JailbreakDetectionDemoDNS.app
→ JailbreakDetectionDemoDNS.app ldid -s JailbreakDetectionDemoDNS
→ JailbreakDetectionDemoDNS.app cd ..
→ codesi scp -P 5555 -r JailbreakDetectionDemoDNS.app root@192.168.0.125:/Applications/
root@192.168.0.125's password:
CodeResources
01J-lp-oVM-view-Ze5-6b-2t3.nib
Info.plist
UIViewController-01J-lp-oVM.nib
BYZ-38-t0r-view-8bC-Xf-vdC.nib
Info.plist
UIViewController-BYZ-38-t0r.nib
embedded.mobileprovision
Info.plist
JailbreakDetectionDemoDNS
PkgInfo
→ codesi
```

	100%	4803	163.5KB/s	00:00
CodeResources	100%	1792	254.7KB/s	00:00
01J-lp-oVM-view-Ze5-6b-2t3.nib	100%	258	55.5KB/s	00:00
Info.plist	100%	832	152.0KB/s	00:00
UIViewController-01J-lp-oVM.nib	100%	3552	442.2KB/s	00:00
BYZ-38-t0r-view-8bC-Xf-vdC.nib	100%	258	53.2KB/s	00:00
Info.plist	100%	964	211.3KB/s	00:00
UIViewController-BYZ-38-t0r.nib	100%	12KB	1.2MB/s	00:00
embedded.mobileprovision	100%	1144	234.9KB/s	00:00
Info.plist	100%	64KB	2.2MB/s	00:00
JailbreakDetectionDemoDNS	100%	8	1.9KB/s	00:00
PkgInfo				



The screenshot shows a terminal window titled 'codesi — ssh root@192.168.1.111'. The window contains the command 'uicache' being run in the directory '..mobile/codesi'. The output shows the command being entered and then completed.

```
[dnss-iPhone:/Applications root# uicache  
dnss-iPhone:/Applications root#
```

10. Depois que o aplicativo for instalado, inicie-o e clique no botão para verificar o status do jailbreak. Observe que o aplicativo agora afirma que o dispositivo não está desbloqueado, o que indica que a aplicação de patches foi bem-sucedida.

Observação: Se você perceber que o aplicativo trava ao ser aberto, em vez de usar a abordagem .app para instalar o aplicativo, use o método .IPA e instale o IPA no dispositivo usando o Cydia Impactor

14. Engenharia reversa usando o IDA PRO

A Hex-rays tem um excelente documento sobre como fazer engenharia reversa usando o IDA PRO. Encontre-o no link abaixo:

https://www.hex-rays.com/products/ida/support/tutorials/ios_debugger_tutorial.pdf

15. MITM no iOS

A intenção básica de um proxy de interceptação é determinar se um canal de transmissão é seguro ou não. Diferentemente dos aplicativos da Web tradicionais, o protocolo usado por um aplicativo móvel pode variar. Além disso, há canais adicionais a serem considerados, como 4G, GPRS, SMS, USSD, Bluetooth, NFC etc. durante o teste.

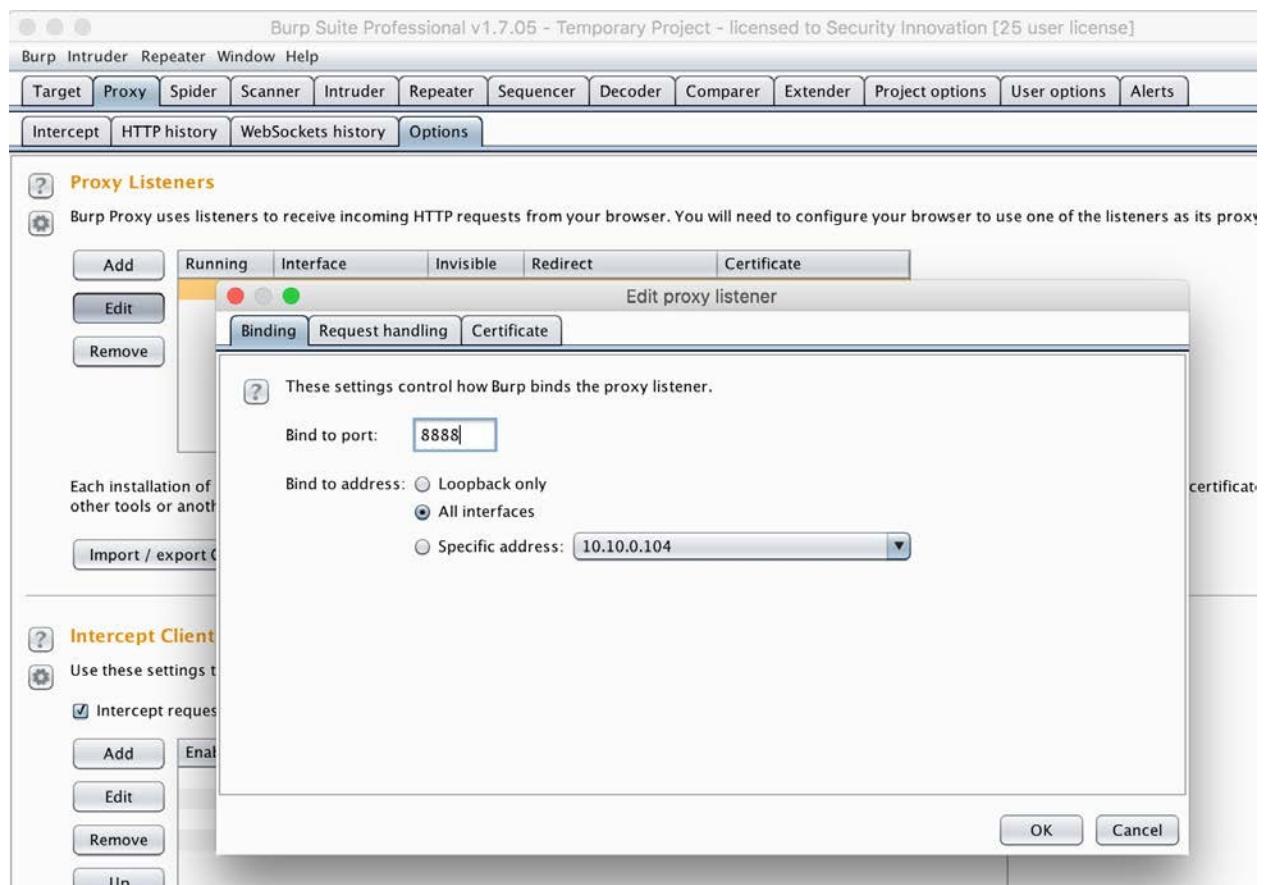
A configuração mais simples para interceptar o tráfego do iOS é mostrada abaixo:



O objetivo é garantir que todos os dados que saem do dispositivo iOS passem pelo laptop. Esse laptop pode ser usado para detectar o tráfego ou para a manipulação de dados.

15.1 Tráfego HTTP MITM

- Inicie o Burp Suite e configure-o para escutar o tráfego em todas as interfaces. Defina o número da porta como um número aleatório.



-
-
-
- Selecione "Manual" na seção HTTP PROXY.
- Defina o servidor para usar o laptop como proxy e a porta na qual o Burp Suite está sendo executado.
- Observe que o tráfego é interceptado no Burp Suite sem nenhum problema.

Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 us...]

Burp Intruder Repeater Window Help

Sequencer Decoder Comparer Extender Project options User options Alerts

Target **Proxy** Spider Scanner Intruder Repeater

Intercept HTTP history WebSockets history Options

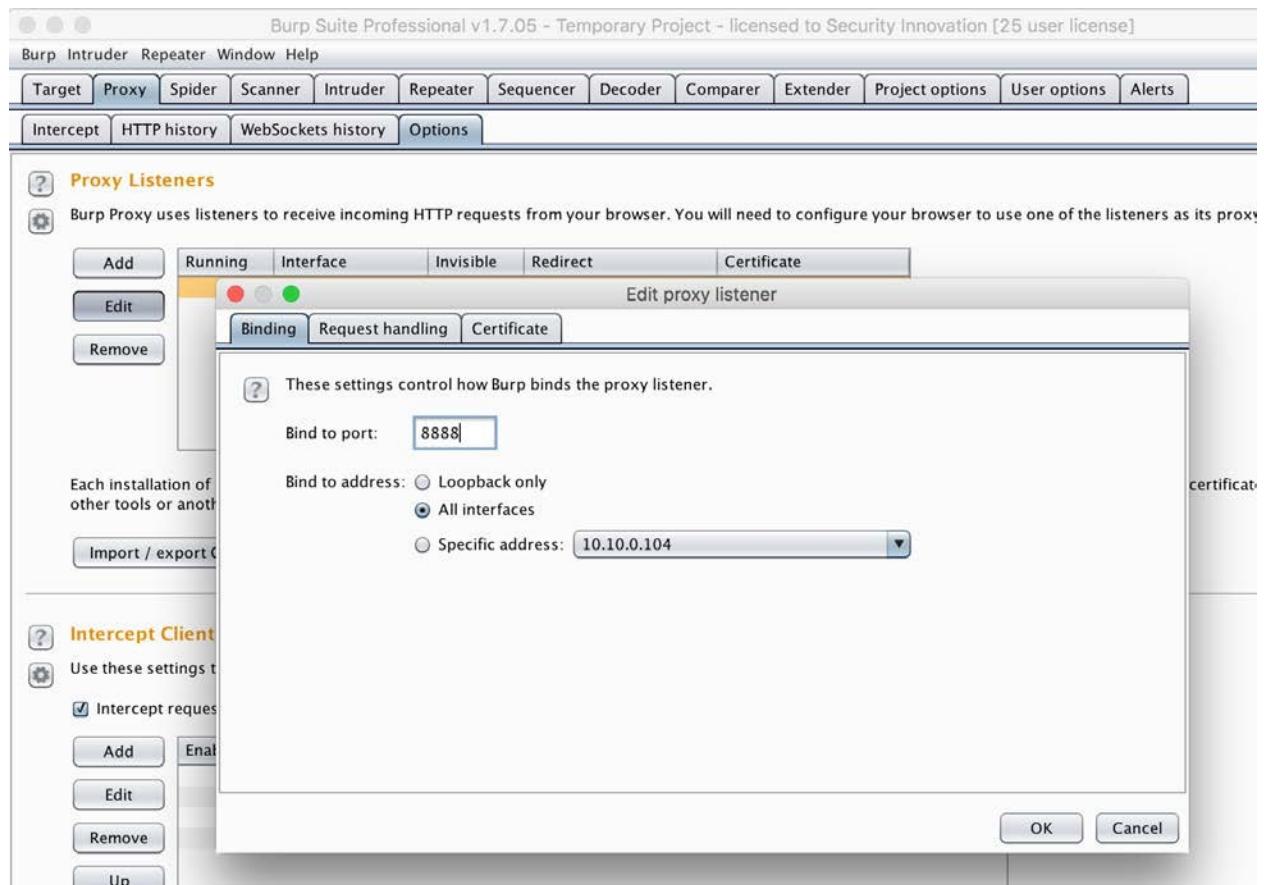
Request to http://bing.com:80 [204.79.197.200]
Forward Drop Intercept is on Action Comment this item

Raw Params Headers Hex

```
GET / HTTP/1.1
Host: bing.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: _EDGE_V=1; MUID=01F554B59D7669622D865C4E9C6668F3; SRCHD=AF=NOFORM;
SRCBUSR=DOB=20160418
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_0_2 like Mac OS X)
AppleWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13A452 Safari/601.1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: close
```

15.2 Tráfego MITM SSL/TLS

1. Inicie o Burp Suite e configure-o para escutar o tráfego em todas as interfaces. Defina o número da porta como um número aleatório.



- 2.
- 3.
4. Selecione "Manual" na seção HTTP PROXY.
5. Defina o servidor para usar seu laptop como proxy e a porta na qual o Burp Suite está sendo executado.
6. No navegador móvel, navegue até <http://burp>
7. Clique em CA Certificate (Certificado CA)
8. Quando solicitado, instale o certificado de CA do PortSwigger.
9. Inicie o aplicativo do LinkedIn e observe que o tráfego TLS é interceptado no Burp Suite sem nenhum problema

● ○ ● Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 us...]

Burp Intruder Repeater Window Help

Sequencer	Decoder	Comparer	Extender	Project options	User options	Alerts
Target	Proxy	Spider	Scanner	Intruder	Repeater	

Intercept	HTTP history	WebSockets history	Options
-----------	--------------	--------------------	---------

Request to https://www.linkedin.com:443 [108.174.10.10]

Forward Drop Intercept is on Action Comment this item

Raw Params Headers Hex

```

POST /uas/authenticate HTTP/1.1
Host: www.linkedin.com
Connection: close
Content-Length: 417
X-LI-Track:
{"orientation": "P", "osName": "iOS", "clientVersion": "9.7.1983.5", "timezoneOffset": "-5", "osVersion": "9.0.2", "appId": "com.linkedin.LinkedIn", "locale": "en_US", "deviceType": "iphone", "deviceId": "75CA1DDB-100E-4BCF-846A-F8A46A30F653", "clientMinorVersion": "910.48", "language": "en", "model": "iphone6_1", "carrier": "AT&T", "clientTimestamp": "1485466071297"}
Content-Type: application/x-www-form-urlencoded
X-Li-User-Agent: LIAuthLibrary:11.0.14 com.linkedin.LinkedIn:9.7.1983.5 iPhone:9.0.2
Csrf-Token: ajax:3716046476349964270
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_0_2 like Mac OS X) AppleWebKit/601.1 (KHTML, like Gecko) Mobile/13A452 Safari/601.1.46
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en
Cookie: bcookie="v=2&edb68541-d665-4ac5-8548-54670a928c4b";
bscookie="v=1&2017012101192765316945-a84c-4ad8-8a59-faada2efbd29AQEDEL8Zwji5xETI23J6BLU-itI0FPOJ"; JSESSIONID="ajax:3716046476349964270"; lang=v=2&lang=en-us;
lidc="b=VGST03:g=264:u=1:i=1485466034:t=1485552434:s=AQEg65cWPSwfKg78BHRjDFnWEEnFP1xxK"
session_key=dnstest%40gmail%2Ecom&session_password=randompass&bscookie=v%3D1%262017012101192765316945-a84c-4ad8-8a59-faada2efbd29AQEDEL8Zwji5xETI23J6BLU-itI0FPOJ&lidc=b%3DVGST

```

15.3 Tráfego MITM não HTTP/SSL/TLS

Para tráfego não HTTP/(s), use o Mallory. Você pode consultar o guia abaixo sobre como instalar e usar o Mallory: <https://www.pentestpartners.com/blog/advanced-traffic-interception-for-mobile-apps-using-mallory-and-burp/>

15.4 MITM usando VPN

Em um ambiente de gerenciamento de dispositivos móveis (MDM), muitas vezes há o desafio de mostrar a um cliente como interceptar o tráfego sem se conectar ao WiFi no dispositivo iOS. Isso pode ser feito por meio da VPN do iOS.

1. No laptop Linux que atuará como servidor VPN, instale o pacote pptpd usando o comando abaixo:
 - o sudo apt-get install pptpd
2. Edite o arquivo /etc/pptpd.conf para limitar o intervalo de ip dos endereços alocados aos clientes VPN (aqui, o intervalo de lan é 10.10.1.0/24). Defina o cliente VPN para usar 10.10.10.1 como gateway
 - o localip 10.10.10.1
 - o remoteip 10.10.10.100-254
3. Edite o arquivo /etc/ppp/pptpd-options e configure o servidor DNS
 - o ms-dns 8.8.8.8
 - o ms-dns 8.8.4.4
4. Edite o arquivo de configuração /etc/ppp/chap-secrets e defina as credenciais
 - o #Endereços IP secretos do servidor cliente
 - o dinesh pptpd admin *
 - Aqui, admin/admin é a credencial e pptpd é o tipo de serviço
5. Configure o encaminhamento de IP usando o comando abaixo
 - o sysctl -w net.ipv4.ip_forward=1
6. Configure o IP masquerading usando o comando abaixo
 - o iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
7. Reinicie o serviço VPN usando o comando abaixo:
 - o /etc/init.d/pptpd restart ou service pptpd restart
8. No dispositivo iOS, localize a opção de configurações de VPN. Edite as configurações de PPTP para refletir as informações abaixo:

- Descrição - SomeVPNName
 - Servidor - 10.10.1.106 (IP do servidor VPN) (IP remoto)
 - Conta - dinesh
 - Senha - admin
 - Nível de criptografia - Auto
 - Enviar todo o tráfego - ON
9. Defina PROXY como Manual e insira as configurações de proxy para a máquina e a porta do Burp.
Observe que o tráfego agora é capturado no Burp Suite.

15.5 MITM quando o aplicativo iOS é acessível somente via VPN

Andreas Kurtz escreveu um [excelente](#) artigo explicando como fazer o pentest de aplicativos iOS quando o aplicativo é acessível somente por meio de uma conexão VPN. Isso pode acontecer quando o backend só pode ser acessado em um ambiente corporativo interno e o acesso à LAN local é restrito quando uma conexão VPN é estabelecida. Abaixo estão as etapas sugeridas por Kurtz.

1. Inicie o Burp Suite no laptop
 2. No Burp Suite, Options -> Connections -> Upstream Proxy Servers, configure o servidor como 127.0.0.1 e a porta como 7777
 3. No dispositivo iOS, instale o "3proxy" do Cydia
 4. Faça o SSH no dispositivo iOS e configure o arquivo de configuração do 3proxy em /var/root/3proxy.cfg com a configuração abaixo:
 - log /var/root/3proxy.log D
 - Formato de registro "%d-%m-%Y %H:%M:%S %U %C:%c %R:%r %O %I %T"
 - proxy -p7777 -n
 5. Inicie o 3proxy no dispositivo iOS usando o comando abaixo:
 - 3proxy /var/root/3proxy.cfg &
 6. No dispositivo iOS, nas configurações de VPN, defina as configurações de proxy como Manual e defina a configuração conforme mostrado abaixo:
 - Servidor: 127.0.0.1
 - Porta: 8080 (número da porta do Burp)
- OBSERVAÇÃO: se, devido à solução MDM, não for possível inserir o IP como 127.0.0.1, edite o arquivo /etc/hosts no dispositivo iOS e defina um alias para localhost.
7. No laptop, como a conexão com endereços locais é bloqueada quando a VPN é iniciada, faça o SSH no dispositivo usando SSH sobre USB, conforme explicado no Módulo 8 (leitura de dados do aplicativo usando SSH sobre USB).
 8. No dispositivo, execute o comando abaixo:
 - ssh -p 2222 -L 7777:127.0.0.1:7777 -R 8080:127.0.0.1:8080 root@127.0.0.1

Observação: consulte um dos links abaixo para obter mais detalhes:

- <https://andreas-kurtz.de/2013/07/ios-proxy-fight/>
- <http://techie-anand.blogspot.com/2015/02/how-to-intercept-traffic-from-devices.html>

15.6 MITM contornando a fixação de certificados

A fixação de certificados é a maneira de limitar um aplicativo móvel, permitindo que apenas o certificado do servidor seja confiável, em vez de depender do próprio armazenamento de certificados confiáveis do celular.

Em vez de confiar no armazenamento confiável do dispositivo, alguns desenvolvedores configuram o aplicativo para confiar apenas no certificado SSL do servidor. Dessa forma, ao se conectar a um servidor SSL/TLS específico, não há necessidade de um terceiro compartilhar a identidade do servidor. Além disso, o comprometimento de qualquer CA no armazenamento confiável do dispositivo não é um problema, pois a conexão não depende mais dele. No entanto, a fixação de certificados ainda pode ser contornada por meio de ferramentas como o SSL Kill Switch ou o iOS TrustMe.

A versão mais recente do SSL Kill Switch é a v2 e pode ser encontrada aqui: <https://github.com/nabla-c0d3/ssl-kill-switch2>.

Antes de usar o SSL Kill Switch 2, instale as seguintes dependências usando o Cydia:

- Empacotador Debian
- Substrato Cydia
- PreferenceLoader

Extraia o binário de instalação mais recente do site <https://github.com/nabla-c0d3/ssl-kill-switch2/releases> para o dispositivo usando o comando abaixo:

- `wget https://github.com/nabla-c0d3/ssl-kill-switch2/releases/download/0.10/com.nabla-c0d3.SSLKillSwitch2_0.10.deb --no-check-certificate`

```
bin — ssh root@192.168.0.125 — ssh root@192.168.0.125 —
root@192.168.0.125
dnss-iPhone:/tmp root# dpkg -i com.nabla-c0d3.SSLKillSwitch2_0.10.deb
Selecting previously deselected package com.nabla-c0d3.sslkillswitch2.
(Reading database ... 7103 files and directories currently installed.)
Unpacking com.nabla-c0d3.sslkillswitch2 (from com.nabla-c0d3.SSLKillSwitch2_0.10.deb) ...
Setting up com.nabla-c0d3.sslkillswitch2 (0.10-2) ...
dnss-iPhone:/tmp root#
```

Reinicie o dispositivo

- `killall -HUP SpringBoard`

No menu Settings (Configurações), veja uma opção adicional "SSL Kill Switch 2" que permite desativar a validação de certificados e ignorar a fixação de certificados.

Aplicativo usado para o exemplo: Aplicativo Twitter da AppStore

1. Configure o dispositivo iOS para fazer proxy do tráfego via Burp Suite usando as etapas mencionadas no Módulo 15 (Tráfego MITM SSL/TLS)
2. Inicie o aplicativo do Twitter no dispositivo iOS e tente entrar no aplicativo. Observe que você está impedido de fazer login no aplicativo e recebe uma mensagem de erro "TLS trust verification failed". Isso indica que a fixação de certificados está ativada no aplicativo do Twitter. Observe também que as solicitações não chegam ao Burp Suite.
3. Ative a opção "Disable Certificate Validation" (Desativar validação de certificado) usando o SSL Kill Switch 2 no menu Settings (Configurações).
4. Reinicie o aplicativo Twitter e agora tente fazer login no aplicativo. Observe que a fixação do certificado foi contornada e o tráfego agora chega ao Burp Suite.

● ○ ● Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 us...]

Burp Intruder Repeater Window Help

Sequencer Decoder Comparer Extender Project options User options Alerts

Target **Proxy** Spider Scanner Intruder Repeater

Intercept HTTP history WebSockets history Options

Request to https://api.twitter.com:443 [104.244.42.194]

Forward Drop Intercept is on Action Comment this item

Raw Params Headers Hex

```
POST /auth/1/xauth_password.json HTTP/1.1
Host: api.twitter.com
X-Guest-Token: 824850307507122176
X-Twitter-Client-DeviceID: 10331E7D-92B7-4A98-9CE0-EBAF4C7C47B6
X-Twitter-Client-Version: 6.70
Authorization: Bearer
AAAAAAAAAAAAAAAAAAj4AQAAAAAPraK64zCZ9CSzdLesbE7LB%2Bw4uE%3DVJQREvQNCZJNiz3rH071oXlk
V0Qkzdsgu6WgcazdMUaGoUGm
X-Client-UUID: DB1F1E93-3F3D-4358-8B6B-2FC244C7115E
X-Twitter-Client-Language: en
X-B3-TraceId: 00a3726dfdc8678e
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 139
User-Agent: Twitter-iPhone/6.70 iOS/9.0.2 (Apple;iPhone6,1;;;;;1)
Connection: close
X-Twitter-Client-Limit-Ad-Tracking: 1
X-Twitter-API-Version: 5
X-Twitter-Client: Twitter-iPhone

send_error_codes=1&x_auth_identifier=blah%40blah.com&x_auth_login_verification=true&x_auth_password=securepassword&x_auth_supports_1fa=true
```

É possível fazer login no aplicativo mesmo com o Burp Suite interceptando suas solicitações.

Um guia detalhado sobre a fixação de SSL pode ser encontrado aqui:

- http://media.blackhat.com/bh-us-12/Turbo/Diquet/BH_US_12_Diquet_Osborne_Mobile_Certificate_Pinning_Slides.pdf

Outra ferramenta a ser usada para ignorar a fixação de certificados no iOS é o TrustMe, que pode ser baixado aqui:

• <https://github.com/intrepidusgroup/trustme> As instruções para o TrustMe são as mesmas que para o SSL Kill Switch 2.

Siga o guia abaixo para aprender a contornar a fixação de ssl aberto em aplicativos iOS:

- <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2015/january/bypassing-openssl-certificate-pinning-in-ios-apps/>

15.7 MITM por sequestro de DNS

Use o DNSChef para tentar fazer MITM no tráfego do iOS usando o sequestro de DNS. Os detalhes podem ser encontrados aqui:

- <http://thesprawl.org/research/ios-data-interception/#dns-hijacking>

15.7 MITM usando o gateway de rede

A maioria das técnicas de MITM foi abordada neste guia. No entanto, se você ainda achar que a interceptação de tráfego falha, siga o guia no link abaixo para entender como fazer MITM usando regras adicionais da tabela IP:

- <http://thesprawl.org/research/ios-data-interception/#capturing-on-the-network-gateway>

15.8 Monitoramento das atividades do sistema de arquivos do iOS

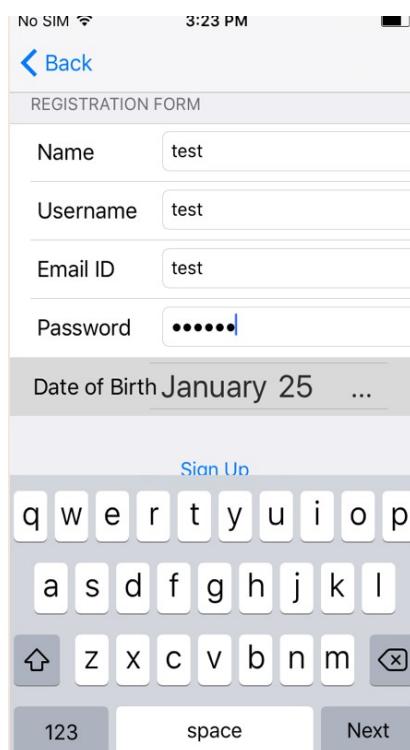
Pode haver ocasiões em que um aplicativo esteja gravando em arquivos no sistema de arquivos sem o seu conhecimento. Nessa situação, uma ferramenta como o Filemon pode ser útil.

O Filemon é um utilitário que rastreia as atividades do sistema de arquivos no IOS. Ele mostra as alterações feitas junto com as operações realizadas no sistema de arquivos do aplicativo durante o tempo de execução.

Aplicativo usado para o exemplo: Aplicativo DemoLogin do site
<https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU>

Para instalar o Filemon em um dispositivo com jailbreak

- SSH para o dispositivo iOS a partir de um PC
 - Faça o download do arquivo mon em <http://newosxbook.com/tools/filemon.tgz>
 - wget http://newosxbook.com/tools/filemon.tgz
 - Descompacte o arquivo filemon.tgz
-
1. Execute o Filemon usando a opção -f [string], que filtrará os caminhos que contêm a string fornecida. Nesse caso, o GUID do aplicativo é E3A9DACE-C895-4B28-9A38-0312A90EA06F. Execute o comando abaixo:
 - a. ./filemon -c -f E3A9DACE-C895-4B28-9A38-0312A90EA06F"
 2. Inicie o aplicativo "DemoLogin" no dispositivo iOS
 3. Navegue até a tela "Register" (Registrar) e preencha o formulário de registro. Em seguida, clique em "Sign Up" (Registrar-se).



4. O Filemon registra e mostra quais arquivos foram acessados e quais operações foram realizadas durante o tempo de execução:

```
dnss-iPhone:~ root#
dnss-iPhone:~ root# ./filemon -c -f E3A9DACE-C895-4B28-9A38-0312A90EA06F
Adding File filter 0: E3A9DACE-C895-4B28-9A38-0312A90EA06F
  181 cprefsd Created      /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Library/Preferences/shrth.DemoLogin.plist.l10QDQj
  181 cprefsd Chowned     /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Library/Preferences/shrth.DemoLogin.plist.l10QDQj
  181 cprefsd Modified    /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Library/Preferences/shrth.DemoLogin.plist.l10QDQj
  181 cprefsd Renamed     /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Library/Preferences/shrth.DemoLogin.plist.l10QDQj  /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Library/Preferences/shrth.DemoLogin.plist
```

A abertura dos arquivos indicou que os detalhes do registro estavam armazenados em texto simples no sistema de arquivos do iOS.

16. Vazamento no canal lateral

O vazamento de canal lateral ocorre quando os dados estão sendo armazenados em cache no dispositivo, seja pelo aplicativo ou pelo próprio iOS.

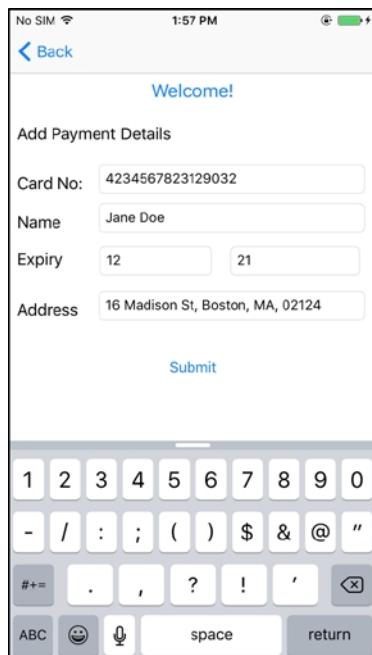
16.1 Mecanismo de cache de captura de tela padrão do iOS

O dispositivo iOS faz uma captura de tela da página atual do aplicativo quando a tecla HOME é pressionada. Isso é usado para gerar uma animação quando o aplicativo se retrai para o plano de fundo e se expande de volta para a tela quando o usuário abre o aplicativo novamente. A imagem é armazenada em cache no armazenamento local e pode ser facilmente acessada por um invasor que tenha acesso físico ao dispositivo. Se informações confidenciais, como detalhes de pagamento, SSN ou outras PII, estiverem sendo exibidas na página durante o plano de fundo, elas poderão ser roubadas por um invasor ao recuperar o arquivo de captura de tela do dispositivo.

Aplicativo usado para o exemplo: Aplicativo DemoLogin de
<https://drive.google.com/open?id=0B0b4IUTjHfRKRW9uQ0hKVzM2dEU>

Abordagem de teste de caixa preta:

1. Inicie o aplicativo "DemoLogin", registre um novo usuário e faça login no aplicativo.
2. Insira as informações de pagamento e pressione o botão INÍCIO no dispositivo.



3. Faça FTP para o dispositivo iOS usando um cliente como o FileZilla para obter a captura de tela da pasta Snapshots. Isso pode ser encontrado aqui:

- */private/var/mobile/Containers/Data/Application/<DemoLogin-GUID>/Library/Caches/Snapshots/*

Como o GUID no nome da pasta muda a cada instalação e/ou durante o tempo de execução, é necessário procurar manualmente em cada pasta para localizar a pasta do aplicativo "DemoLogin". Algumas outras técnicas que podem ser empregadas são:

4. No FileZilla, navegue até "/private/var/mobile/Containers/Data/Application/" e classifique por Last modified (Última modificação). Como o DemoLogin foi o último aplicativo usado, a pasta modificada recentemente é onde se encontram os dados do aplicativo, como o arquivo de captura de tela.

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
06EBB07F-13E4-41F9-AA51-E6942EDC8CC2		Directory	01/27/2017 13:55:32	drwxr-xr-x	mobile mobile
7974EE08-CB1F-46D3-833B-26832EFB5428		Directory	01/26/2017 23:11:15	drwxr-xr-x	mobile mobile
A5AB5D1-6072-4488-A4BD-28AE1CB3FE86		Directory	01/26/2017 22:57:48	drwxr-xr-x	mobile mobile
D1DBC81-D311-4F1A-BBCD-F6893C7961CC		Directory	01/26/2017 22:57:47	drwxr-xr-x	mobile mobile
4CFAD9DE-AF01-42FF-8EDD-B4A2891BCAF1		Directory	01/12/2017 20:22:42	drwxr-xr-x	mobile mobile
5C050C03-2877-4D20-99AF-12460B97BEF6		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
58D19DBC-8F3B-47D4-BE97-E20E49F4CBDA		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
28BFF43A-A420-4804-9D7C-491FB9591AD		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
CFD671FB-8172-437F-B62B-ED163DE1EA41		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
320DB08A-CB18-4E85-9775-AAAD15CA74F8		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
791B12AB-6B36-458E-9D6B-53ED156FBA89		Directory	01/12/2017 20:22:39	drwxr-xr-x	mobile mobile
AB26A705-41F3-496F-857A-1E04825895EC		Directory	05/22/2016 23:14:09	drwxr-xr-x	mobile mobile
82DD99E5-179C-44B5-BE4B-1B2F04ADD8C		Directory	05/22/2016 23:00:31	drwxr-xr-x	mobile mobile
CBD703D8-5722-40CC-9CAD-235B42F7		Directory	05/22/2016 21:17:56	drwxr-xr-x	mobile mobile
56E6CEEA-6459-44DC-BF4F-48556799A461		Directory	05/22/2016 17:39:50	drwxr-xr-x	mobile mobile
DD38D8BE-4A2E-49FE-9E71-A5D2F1986380		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
D2A407E6-F206-4719-AF8C-46DE85B39963		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC818A8B-B8DC-4619-81B8-6C4EA3B0D76C		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC1B6266-D668-48FE-9807-F8A270D1245		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
B7292798-F20C-46CC-A01C-E782ED79080B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
9304A9E3-8214-419A-A66F-9FC9A374BA41		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
903FEF77-1E27-4157-86FC-BFFA72D5F1F1		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
OC6DAC99-23E1-4E63-9DB7-4EFA4D36F8B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
FDA841DB-60FD-491D-A91E-C96526B904B6		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile
D9C1FECC-2921-45AC-8BAA-A0A01F341E25		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile

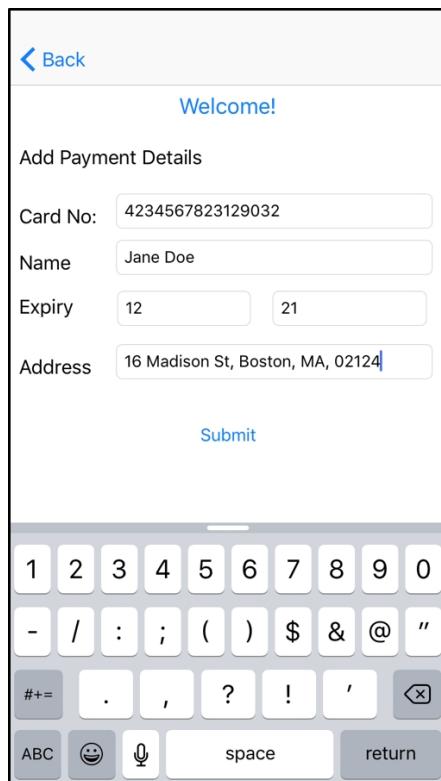
Selected 1 directory.

5. Como alternativa, entre por SSH no dispositivo iOS, navegue até "/private/var/mobile/Containers/Data/Application/" e execute o comando abaixo para listar todos os arquivos plist. Se os arquivos plist foram criados, podemos usar o nome do arquivo plist para identificar a pasta do aplicativo.

- o `find -type f -name '*plist' | grep 'Preferences' 2>/dev/null`

```
dns-iphone6-jailbroken:/private/var/mobile/Containers/Data/Application root# find -type f -name '*plist' | grep 'Preferences' 2>/dev/null
./06EBB07F-13E4-41F9-AA51-E6942EDC8CC2/Library/Preferences/shrth.DemoLogin.plist
./2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9/Library/Preferences/com.apple.stocks.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.homesharing.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.medialibrary.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.springboard.plist
./4FD5A90A-41F9-4B9D-B3EE-C7807731395E/Library/Preferences/com.apple.mobilemail.plist
./6C1A80AD-E37E-4438-91D1-9C0E9E68507A/Library/Preferences/UIKitInputContextIdentifiers.plist
./6C1A80AD-E37E-4438-91D1-9C0E9E68507A/Library/Preferences/com.apple.mobilesafari.plist
./791B12AB-6B36-458E-9D6B-53ED156FBA89/Library/Preferences/com.google.chrome.ios.plist
./D1DBC81-D311-4F1A-BBCD-F6893C7961CC/Library/Preferences/com.atebits.Tweetie2.plist
./D9C1FECC-2921-45AC-8BAA-A0A01F341E25/Library/Preferences/com.apple.medialibrary.plist
./F6BB83E3A-7795-4A6F-BB13-2D71E7A0E43/Library/Preferences/com.appleMaps.plist
dns-iphone6-jailbroken:/private/var/mobile/Containers/Data/Application root#
```

6. Navegue até o local abaixo e faça o download das capturas de tela.
 - `/private/var/mobile/Containers/Data/Application/06EBB07F-13E4-41F9-AA51-E6942EDC8CC2/Library/Caches/Snapshots/shrth.DemoLogin`
7. A captura de tela contém os detalhes do pagamento que foram armazenados em cache no dispositivo.



16.2 Cache do UIPasteboard do iOS

Quando um texto é copiado em um dispositivo iOS, os dados vão para o buffer e podem ser usados em diferentes áreas do aplicativo, bem como por outros aplicativos no dispositivo. Se algum dado confidencial estiver sendo copiado, esses dados no pasteboard poderão ser vazados para outros aplicativos de terceiros. Um aplicativo de terceiros em execução no dispositivo poderia registrar o conteúdo do pasteboard. Um exemplo de código é mostrado abaixo:

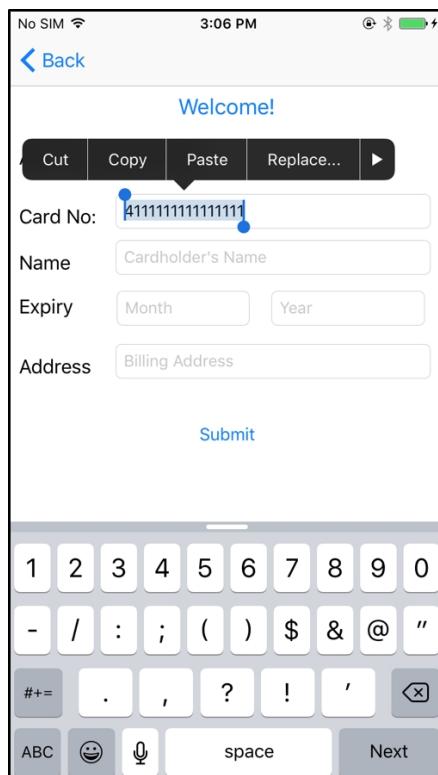
- `NSLog(@"Print the contents of pasteboard: %@", [UIPasteboard generalPasteboard] string);`

Durante um pentest de caixa preta, procure campos sensíveis no aplicativo que permitam o recurso de copiar e colar e veja se eles estão sendo armazenados em cache pelo iOS.

Aplicativo usado para o exemplo: Aplicativo DemoLogin de
<https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU>

Abordagem de teste de caixa preta:

1. Abra o aplicativo "DemoLogin", registre um novo usuário e faça login no aplicativo.
2. Insira qualquer dado no campo de texto e copie o conteúdo para a área de transferência.



3. Entre com o SSH no dispositivo iOS e obtenha o ID do processo do aplicativo usando o comando "ps aux".
4. Use "cyclicrypt -p <process-id>" para se conectar ao aplicativo. Você será solicitado a usar o interpretador Cyclicrypt.

```
mobile  38  0.0  0.1  684144 1384 ?? Ss  11Jan17  0:00.30 /System/Library/PrivateFrameworks/AddressBookUI.framework/Versions/A/Resources/AddressBookUI
root   36  0.0  0.6  739616 5820 ?? Ss  11Jan17  1:28.63 /usr/libexec/fseventsd
root   1  0.0  0.4  685520 4064 ?? Ss  11Jan17  1:44.98 /sbin/launchd
mobile 7682  0.0  0.0    0    0 ?? Z   10:50PM  0:00.00 (MSUnrestrictProc)
mobile 7408  0.0  0.0    0    0 ?? Z   8:00PM  0:00.00 (MSUnrestrictProc)
root   8212  0.0  0.1  537488  568+01 R+  3:07PM  0:00.01 ps aux
mobile 160  0.0  0.0    0    0 ?? Z   11Jan17  0:00.00 (MSUnrestrictProc)
mobile 8204  0.0  1.2  741056 11920 ?? Ss  3:05PM  0:01.58 /var/mobile/Containers/Bundle/Application/7E3E0CFD-CBA3-4B33-83C0-31FF5E5B90C0/DemoLogin.app/De
mobile 8183  0.0  0.4  713280 4412 ?? Ss  3:01PM  0:00.34 /usr/libexec/ptpd -t usb
root   8882  0.0  0.1  548192  812 ?? Ss  2:05PM  0:00.11 /usr/libexec/sftp-server
root   8880  0.0  0.2  548896 1668 ?? Ss  2:05PM  0:00.41 sshd: root@natty
```

5. Digite o comando abaixo para listar o conteúdo da área de transferência do dispositivo. A captura de tela abaixo destaca o item copiado.

- o [UIPasteboard generalPasteboard].items

```

dns-iphone6-jailbroken:/ root# cycript -p 8204
cy# [[UIPasteboard generalPasteboard].items
@]{ {"com.apple.flat-rtfd": "#<72746664 00000000 03000000 02000000 07000000 5458542e 72746601 0000002e 09010000 2
b000000 01000000 01010000 7b5c7274 66315c61 6e73695c 616e7369 63706731 3235320a 7b5c666f 6e747462 6c5c6630 5c66
7377 6973735c 66636861 72736574 30204865 6c766574 6963613b 7d0a7b5c 636f6c6f 7274626c 3b5c7265 64323535 5c67726
5 656e3235 355c626c 75653235 353b7d0a 5c706172 645c7478 3536305c 74783131 32305c74 78313638 305c7478 32323430 5
c747832 3830305c 74783333 36305c74 78333932 305c7478 34343830 5c747835 3034305c 74783536 30305c74 78363136 305c
7478 36373230 5c706172 6469726e 61747572 616c5c70 61727469 67687465 6e666163 746f7230 0a0a5c66 305c6673 3234205
c 63663020 34313131 31313131 31313131 7d010000 00230000 00010000 00010000 00545854 7e272466 10000000 2
ba88b58 b6010000 00000000 00000000 ", "public.utf8-plain-text" "4111111111111111", "Apple Web Archive pasteboard
type": "#<3c21444f 43545950 45206874 6d6c2050 55424c49 4320222d 2f2f5733 432f2t44 54442048 544d4c20 342e3031 2f2
f454e 22202268 7474703a 2f2f7777 772e7733 2e6f7267 2f54522f 68746d6c 342f7374 72696374 2e647464 223e0a3c 68746d
6c 3e0a3c68 6561643e 0a3c6d65 74612068 7474702d 65717569 763d2243 6f6e7465 6e742d54 79706522 20636f6e 74656e74
3d227465 78742f68 746d6c3b 20636861 72736574 3d555446 2d38223e 0a3c6d65 74612068 7474702d 65717569 763d2243 6f6
e7465 6e742d53 74796c65 2d547970 65222063 6f6e7465 6e743d22 74657874 2f637373 223e0a3c 7469746c 653e3c2f 746974
6c 653e0a3c 6d657461 206e616d 653d2247 656e6572 61746f72 2220636f 6e74656e 743d2243 6f636f61 2048544d 4c205772
69746572 223e0a3c 7374796c 65207479 70653d22 74657874 2f637373 223e0a70 2e703120 7b6d6172 67696e3a 20302e30 707
82030 2e307078 20302e30 70782030 2e307078 7d0a7370 616e2e73 31207b66 6f6e742d 66616d69 6c793a20 2748656c 766574
69 6361273b 20666f6c 742d7765 69676874 3a206e6f 726d616c 3b20666f 6e742d73 74796c65 3a206e6f 726d616c 3b20666f
6e742d73 697a653a 2031322e 30307074 7d0a3c2f 7374796c 653e0a3c 2f686561 643e0a3c 626f6479 3e0a3c70 20636c61 737
33d22 7031223e 3c737061 6e20636c 6173733d 22733122 3e343131 31313131 31313131 313c2f73 70616e3e 3c2f70
3e 0a3c2f62 6f64793e 0a3c2f68 746d6c3e 0a>"]}
cy#

```

16.3 Armazenamento de cookies do iOS

O dispositivo iOS permite que o aplicativo armazene cookies que podem ser criados pelo sistema de carregamento de URL ou pela solicitação HTTP pela visualização da Web. Os cookies são armazenados no armazenamento do dispositivo e um invasor com acesso físico ao dispositivo pode roubá-los. Os cookies são armazenados em formato binário e podem ser analisados usando o script *BinaryCookieReader.py* em <http://securitylearn.net/wp-content/uploads/tools/iOS/BinaryCookieReader.py>

Aplicativo usado para o exemplo: Aplicativo DemoLogin de
<https://drive.google.com/open?id=OB0b4lUTjHfRKRW9uQ0hKVzM2dEU>

Abordagem de teste de caixa preta:

1. Inicie o aplicativo "DemoLogin" e faça login com credenciais válidas. Preencha as informações de pagamento e toque no botão enviar.
2. Faça FTP no dispositivo iOS usando um cliente como o FileZilla para obter o arquivo de cookie. Isso pode ser encontrado em:
`/private/var/mobile/Containers/Data/Application/<DemoLogin-GUID>/Library/Cookies` Como o GUID no nome da pasta muda a cada instalação e/ou durante o tempo de execução, teríamos que procurar manualmente em cada pasta para encontrar a pasta do aplicativo "DemoLogin". Algumas outras técnicas que podem ser empregadas são mostradas abaixo.
3. No Filezilla, navegue até "`/private/var/mobile/Containers/Data/Application/`" e classifique por Última modificação.

4.

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
311C2C31-6E53-47AD-96C2-8231DF83CD96		Directory	01/27/2017 14:55:27	drwxr-xr-x	mobile mobile
7974EE08-CB1F-46D3-833B-26832EFB5428		Directory	01/26/2017 23:11:15	drwxr-xr-x	mobile mobile
AB5AB5D1-6072-4488-A4BD-28AE1CB3FE86		Directory	01/26/2017 22:57:48	drwxr-xr-x	mobile mobile
D1DBCA81-D311-4F1A-BBCD-F6893C7961CC		Directory	01/26/2017 22:57:47	drwxr-xr-x	mobile mobile
4CFAD9DE-AF01-42FF-8EDD-BA42891BCAF1		Directory	01/12/2017 20:22:42	drwxr-xr-x	mobile mobile
5C050C03-2877-4D20-99AF-12460B97BEF6		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
58D19DBC-8F3B-47D4-BE97-E20E49F4CBDA		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
28BFF43A-A420-4804-9D7C-491FB9591AD		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
CFD671FB-8172-437F-B62B-ED163DE1EA41		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
320DB08A-CB18-4E85-9775-AAAD15CA74F8		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
791B12AB-6B36-458E-9D6B-53ED156FBA89		Directory	01/12/2017 20:22:39	drwxr-xr-x	mobile mobile
AB26A705-41F3-496F-857A-1E04825895EC		Directory	05/22/2016 23:14:09	drwxr-xr-x	mobile mobile
82DD99E5-179C-44B5-BE4B-1B2F04ADD8C		Directory	05/22/2016 23:00:31	drwxr-xr-x	mobile mobile
CBD703D8-5722-40CC-9CAD-495E235B42F7		Directory	05/22/2016 21:17:56	drwxr-xr-x	mobile mobile
56E6CEEA-6459-44DC-BF4F-48556799A461		Directory	05/22/2016 17:39:50	drwxr-xr-x	mobile mobile
DD38D8BE-4A2E-49FE-9E71-A5D2F1986380		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
D2A407E6-F206-4719-AF8C-46DE85B39963		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC818A8B-88DC-4619-81B8-6C4EA3BD076C		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC1B6266-D66B-48FE-9807-F84A270D1245		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
B7292798-F20C-46CC-A01C-E782ED79080B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
9304A9E3-8214-419A-A66F-9FC9A374BA41		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
903FEF77-1E27-4157-86FC-BFFA72D5F1F1		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
FDA841DB-60FD-491D-A91E-C96526B904B6		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile
D9C1FECC-2921-45AC-8BAA-A0A01F341E25		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile

43 directoires

Remote site: /private/var/mobile/Containers/Data/Application/311C2C31-6E53-47AD-96C2-8231DF83CD96/Library/Cookies

0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B			
1CDCDC08-3438-4B45-AB11-8B70DEFCE926			
27F7CA6E-9657-49CC-9229-C8A26F15CAE8			
28BF43A-A420-4804-9D7C-491FB9591AD			
2AEF1028-9029-476A-A479-65E8718C055E			
2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9			
311C2C31-6E53-47AD-96C2-8231DF83CD96			
Documents			
Library			
Caches			
Cookies			
Cookies.binarycookies	296 binaryco... 01/27/2017 15:49:55	-rw-r--r--	mobile mobile

- Os cookies binários podem ser analisados usando o script *BinaryCookieReader.py*. Execute o comando conforme mostrado na captura de tela.

Faça o download do arquivo *BinaryCookieReader.py* no link mencionado abaixo:

<https://gist.github.com/sh1n0b1/4bb8b737370bfe5f5ab8>

```
/nuttertools/testground — ssh root@10.10.0.167 /nuttertools/testground —
|haxorhead:testground sharathunni$ python BinaryCookieReader.py Cookies.binarycookies
*****
# BinaryCookieReader: developed by Satishb3: http://www.securitylearn.net #
*****
Cookie : password=d1password; domain=localhost; path=; expires=Mon, 27 Jul 2048;
Cookie : username=demologinuser; domain=localhost; path=; expires=Mon, 27 Jul 2048;
haxorhead:testground sharathunni$
```

Os cookies armazenados pelo aplicativo contêm informações confidenciais, como o nome de usuário e a senha.

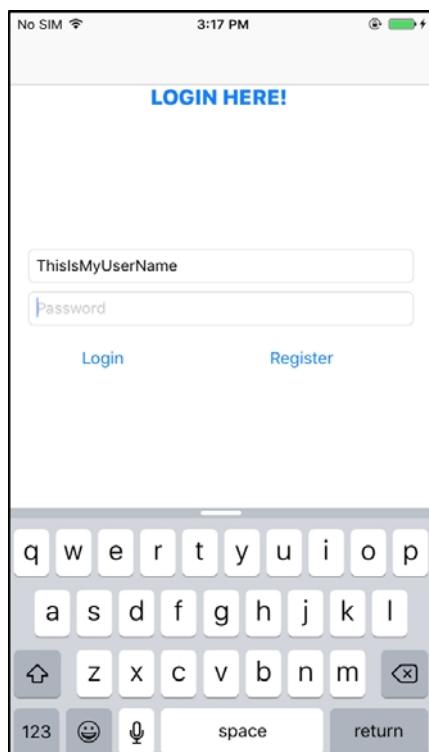
16.4 Armazenamento do cache do teclado do iOS

O iOS armazena em cache os dados que estão sendo digitados e que são utilizados pelo recurso de correção automática. Os dados são armazenados em cache em texto claro na ordem em que são digitados pelo iOS e podem ser encontrados no arquivo "/var/mobile/Library/Keyboard/dynamic-text.dat".

Aplicativo usado para o exemplo: Aplicativo DemoLogin de
<https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU>

Abordagem de teste de caixa preta:

1. Inicie o aplicativo "DemoLogin" e digite qualquer valor no campo Username (Nome de usuário).



2. Faça login no SSH do dispositivo iOS e navegue até a pasta abaixo:
 - o /var/mobile/Library/Keyboard
3. Execute o comando strings no arquivo "dynamic-text.dat" e observe que o texto digitado está sendo armazenado em cache pelo dispositivo.

```
[dns-iphone6-jailbroken:/var/mobile/Library/Keyboard root# strings dynamic-text.dat
DynamicDictionary-5
admin
Boston
Cryptokey
demoiser
demo
developer
dude
gags
hdhdj
hdhd
hello
https
Jane
John
Madison
medium
ndjdjdjd
owasp
PASSWORD
password
shah
Sharath
shar
sjdjjd
superpassword
tester
test
ThisIsMyUserName
unni
user
```

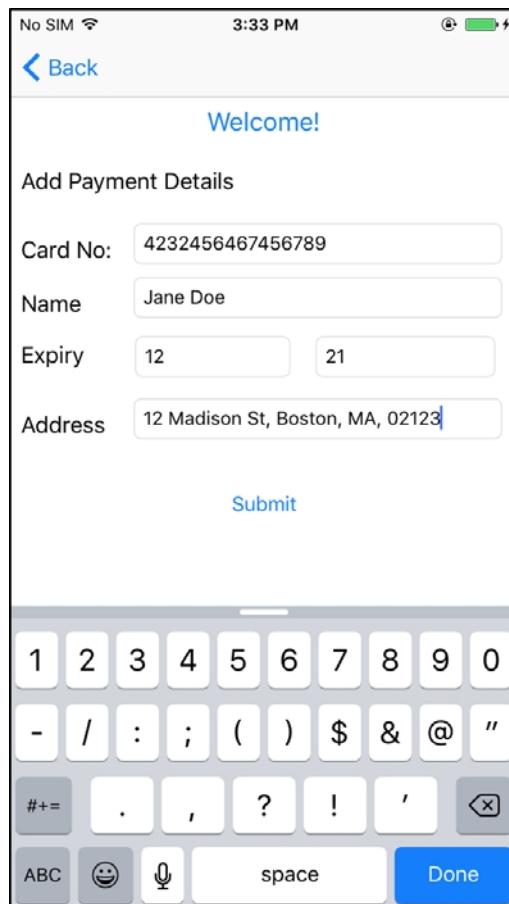
16.5 Registro de dispositivos iOS

Os desenvolvedores geralmente usam o NSLog para fins de depuração e diagnóstico. Às vezes, dados confidenciais são registrados, e esses dados são capturados nos logs do dispositivo. Como os registros de erros não são limitados pela área restrita do aplicativo, eles podem ser lidos por outro aplicativo no dispositivo. Isso pode permitir que um aplicativo mal-intencionado em execução no dispositivo roube dados confidenciais registrados pelo aplicativo. É possível ler os registros de erros criados pelo aplicativo usando o aplicativo Console ou o Xcode. Durante o pentest, é importante navegar por todas as áreas do aplicativo para verificar se algum dado confidencial está sendo registrado.

Aplicativo usado para o exemplo: Aplicativo DemoLogin de
<https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU>

Abordagem de teste BlackBox:

1. Inicie o aplicativo "DemoLogin", registre um novo usuário e faça login no aplicativo.
2. Insira os detalhes do pagamento e toque no botão Submit (Enviar).



3. Inicie o aplicativo "Console" do iOS no Mac OS e selecione o dispositivo apropriado no painel esquerdo.
4. A captura de tela abaixo mostra os dados do usuário sendo registrados pelo aplicativo.

The screenshot shows the macOS Console application window. The title bar says "Console (314 messages)". The interface includes tabs for "All Messages" and "Errors and Faults", and buttons for "Now", "Activities", "Clear", "Reload", and "Info". The main pane lists log entries. A red box highlights the "DemoLogin" entry at the bottom, which contains the message "Card information 4232456467456789 successfully accepted". Other entries in the log relate to SpringBoard processes and accessibility UI servers.

Type	Time	Process	Message
Devices	15:32:18.000000	SpringBoard	CGContextRestoreModelState: invalid context 0x0. If you want to see the backtrace, please set CG_CONTEXT_SHOW_BACKTRACE environmental variable.
Reports	14:32:18.000000	SpringBoard	LICreateIconForImage passed NULL CGImageRef image
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:34.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	DemoLogin	Registration in progress
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	syslogd	ASL Sender Statistics
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	DemoLogin	Dismiss button tapped!
Reports	14:32:38.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:42.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:43.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:44.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:48.000000	DemoLogin	Login Success
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:51.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:32:52.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:33:21.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:33:30.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..
Reports	14:33:42.000000	DemoLogin	Card information 4232456467456789 successfully accepted
Reports	18:59:59.000000	SpringBoard	[<_UIKeyboardArbiterHandle: 0x139419aa8; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {}]; level 0.00000..

Direitos autorais © 2017 Security Innovation, Inc. Todos os direitos reservados. Todas as marcas registradas, marcas de serviço, nomes comerciais, nomes de produtos e logotipos que aparecem no site são de propriedade de seus respectivos proprietários.