Introduction to iOS Penetration Testing

Confidence 20/05/2016 | Slawomir Kosowski

About Me

- Working as Mobile and Web pentester
- Focused on iOS
- Previously worked on wide range of security projects
- Educational background in Telecommunications

https://www.linkedin.com/in/skosowski

Introduction

How many iOS developers do we have here?

How many of you are actually writing in Swift?

Any pentesters?

Android folks?

Agenda

What we will cover:

- Introduction to iOS
- Basics of Objective-C runtime
- Setting up testing environment
- Fundamentals of app testing
  - Focus on black-box testing

What we will **not** cover:

- Jailbreak development
- Swift
- White box testing / code review
- Webapp pentesting

Introduction to iOS

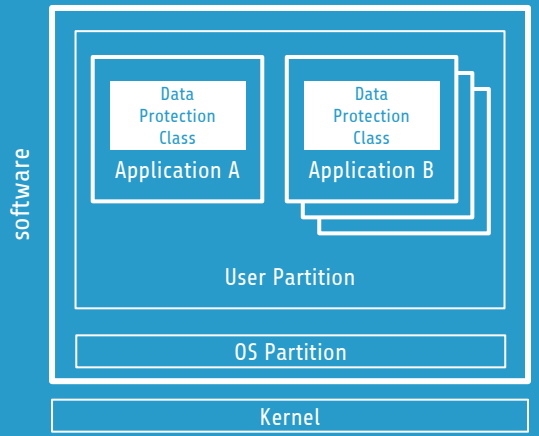
- Mobile operating system from Apple
- Based on XNU kernel from Darwin OS (Unix)
- iOS is closed-source with some exceptions*
- Applications written in Objective-C (and Swift)
- Cocoa Touch - main API for iOS handling user interaction

* https://opensource.apple.com

Introduction to iOS security model

Apple controls both hardware and software to provide end-to-end security, with following key features:

- Secure Boot Chain
- Secure Enclave (and Touch ID)
- Encryption and Data Protection
- Trusted Code Execution
- Network Security



Source: https://www.apple.com/business/docs/iOS_Security_Guide.pdf

Keychain

- Stores sensitive data such as passwords, certificates, tokens, etc.
- Is implemented as SQLite database
- Application can access only items in its `keychain-access-group`
- Can be arbitrarily read on a jailbroken device using `keychain-dumper`

Application Sandbox

- iOS Sandbox derives from TrustedBSD MAC framework
- Each third-party application runs as `mobile` user
- Only a few system daemons/apps run as `root`
- Application can access only its own files and data
- IPCs are very limited

Objective-C

- Superset of C, adding object-oriented functionality
  - This means you can include C code in your apps
- Based on Smalltalk language, supporting message passing, dynamic typing and infix notation
- Uses *interface* and *implementation* file
  - Think about .h and .cpp files in C++

Objective-C

Objective-C is using infix notation with arguments listed after colon:

```
[Object method:argument]

[NSString stringWithString:@”Confidence2016”]
```

Class vs Instance Methods

- Class method can be called on its own
- Instance method must use instance of an object

```
@interface MyClass : NSObject

+ (void)aClassMethod;
- (void)anInstanceMethod;

@end

[MyClass aClassMethod];

MyClass *object = [[MyClass alloc] init];
[object anInstanceMethod];
```

Objective-C – Message Passing

When you pass method, a special function `objc_msgSend()` is called:

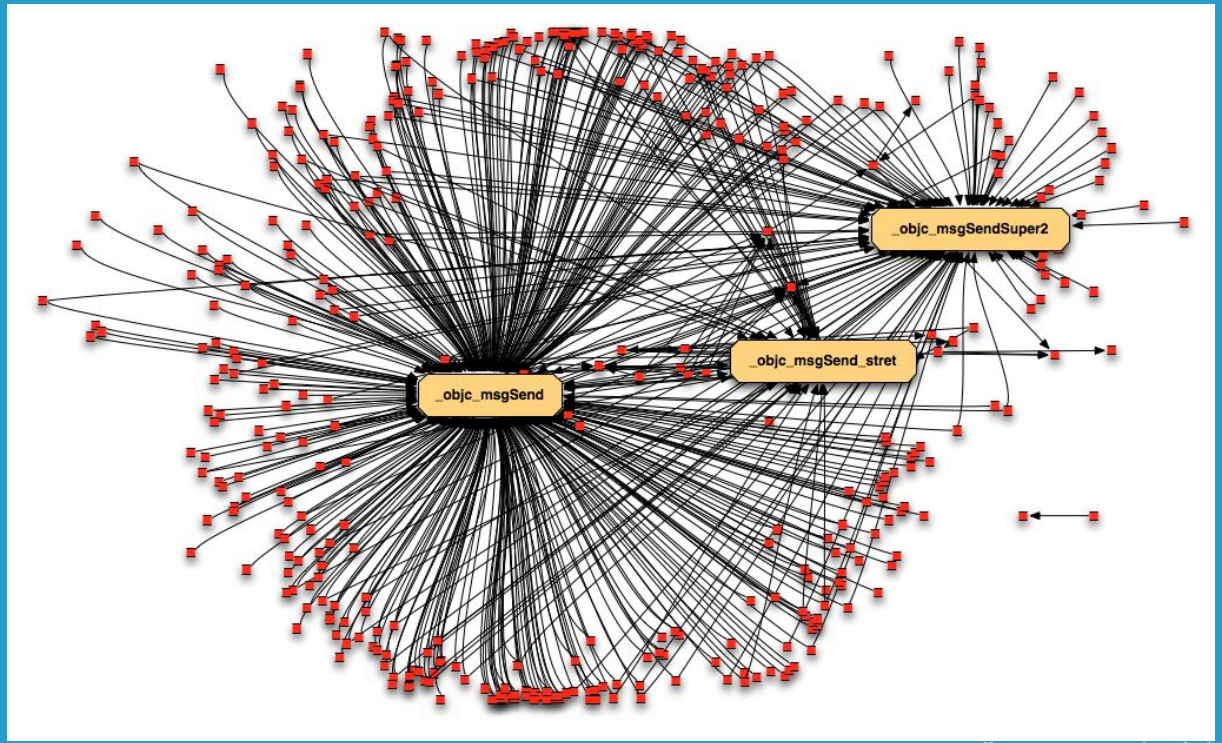
```
SaySomething *saySomething = [ [ SaySomething alloc ] init ];
  [ saySomething say: @"Hello, world!" ];
  [ saySomething release ];
```

Which gets translated into C calls:

```
objc_msgSend(
   objc_msgSend(
       objc_msgSend(
           objc_msgSend(
               objc_getClass("SaySomething"), NSSelectorFromString(@"alloc")),
               NSSelectorFromString(@"init")),
        NSSelectorFromString(@"say:"), @"Hello, world!"),
   NSSelectorFromString(@"release:"));
```

Source: Hacking and Securing iOS Applications – J. Zdziarski

Objective-C Call Graph



Source:http://blog.zynamics.com/2010/04/27/objective-c-reversing-i

Objective-C Runtime

- Objective-C runtime is written in C and assembly
- Very interesting subject on its own!
- Calls are cached so that subsequent messages are dispatched quicker
- Decision on which method will be called is resolved dynamically
- This is called Method Swizzling
- It will help us during black-box testing and runtime manipulation

Read more:

https://mikeash.com/pyblog/friday-qa-2009-03-20-objective-c-messaging.html

http://cocoasamurai.blogspot.com/2010/01/understanding-objective-c-runtime.html

http://www.friday.com/bbum/2009/12/18/objc_msgsend-part-1-the-road-map/

Introduction to Application Analysis

Static Binary Analysis

- IDA Pro
- Hopper (demo closing after 30 minutes)
- class-dump
- otool
- strings

Runtime Manipulation

1. Cycript
    a. injects into process and enables to manipulate the runtime with interactive console
    b. supports mixed Objective-C and Javascript syntax
2. Frida
    a. injects Javascript V8 engine into process runtime
    b. can inject a hook into starting process

Runtime Manipulation – Cont'd

3. Debugger
    a. Apple moved from GCC and GDB to LLVM and LLDB
    b. GDB is fully supported until iOS7
    c. iOS8 and onwards uses LLDB
    d. Some key features are still missing in LLDB
        i. `info mach-regions`
        ii. Symbols from stripped ObjC Mach-O binary are not loaded in LLDB

Setting up Pentesting Lab

- Bare minimum is one iDevice, e.g. iPad running iOS 8.x
  - Recommended at least two or more iDevices
- You will need to Jailbreak it
- Ideally grab another pair of iPads/iPhones running older iOS for any legacy apps
- OS X and XCode is very useful, but not mandatory
- Alternatively, grab your favourite Linux

Setting up Pentesting Lab – Cont'd

- Beware: if you fail to JB your device correctly you can restore it **and upgrade** with iTunes
- Semi-restore might be handy if your JB fails: https://semi-restore.com/
- **No possibility to downgrade iOS version**

Jailbreaking

- Get appropriate jailbreak (Pangu or TaiG) - straightforward for iOS < 9.2
- This will install Cydia - the alternative application store as well as couple of useful services
- From Cydia install `aptitude` and `openssh`
- Install additional packages with `aptitude`

Jailbreaking - cont'd

- SSH to your iDevice
  - Two users are `root` and `mobile`
  - Default password: `alpine`
- Install additional packages with `aptitude`

```
inetutils                    odcctools
syslogd                      cycript
less                         sqlite3
com.autopear.installipa      adv-cmds
class-dump                   bigbosshackertools
com.ericasadun.utilities
```

Install Frida

- Check install guide: www.frida.re/docs/ios
- Basically add https://build.frida.re to Cydia repo
- Then install Frida with Cydia

```
~ $ sudo pip install frida
~ $ frida-trace -i "recv*" Twitter
recvfrom: Auto-generated handler: …/recvfrom.js
Started tracing 21 functions.
1442 ms     recvfrom()
# Live-edit recvfrom.js and watch the magic!
5374 ms     recvfrom(socket=67, buffer=0x252a618,
length=65536, flags=0, address=0xb0420bd8,
address_len=16)
```

IPA file and Binary

- IPA file is simply a ZIP archive
  - Think of APK but for iOS
  - Contains all relevant files like binary itself, graphics, certificates, default data, etc.
- For static analysis, the Mach-O Binary is interesting
- Usually it contains two architectures ARM7(s) and ARM64
- You probably want to stick to 32-bit as long as you can...

Read more about Mach-O File Format:

https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachORuntime/index.html

Important File Location

You can find system applications in `/Applications`

For all the rest use `installipa`:

```
iOS8-jailbreak:~ root# installipa -l
me.scan.qrcodereader
iOS8-jailbreak:~ root# installipa -i me.scan.qrcodereader
Bundle: /private/var/mobile/Containers/Bundle/Application/09D08A0A-0BC5-423C-8CC3-FF9499E0B19C
Application: /private/var/mobile/Containers/Bundle/Application/09D08A0A-0BC5-423C-8CC3-FF9499E0B19C/QR Reader.
app
Data: /private/var/mobile/Containers/Data/Application/297EEF1B-9CC5-463C-97F7-FB062C864E56
```

Usual Test Approach

1. Obtain IPA file; do binary checks
2. Bypass jailbreak detection (if present)
3. Bypass certificate pinning (if present)
4. Inspect HTTP(S) traffic – usual web app test
5. Abuse application logic by runtime manipulation
6. Memory forensics
7. Check for local data storage (caches, binary cookies, plists, databases)
8. Check for client-specific bugs, e.g. SQLi, XSS
9. Other checks like: logging to ASL with NSLog, application screenshots, no app backgrounding

Binary Encryption

- Each app in Apple AppStore uses FairPlay DRM, hence is encrypted
  - You must decrypt it before doing static analysis
- Easiest way to do it is to use `Clutch`
- Alternatively you can use `lldb` and dump process memory once encrypted
  - Broadly documented in the Internet
- If you are doing a pentest, you will get most likely unencrypted IPA file

Binary Security Features

- **ARC** - Automatic Reference Counting - memory management feature
  - adds `retain` and `release` messages when required
- **Stack Canary** - helps preventing buffer overflow attacks
- **PIE** - Position Independent Executable - enables full ASLR for binary

All of above are currently set by default in XCode.

Binary Checks – PIE

```
$ unzip DamnVulnerableiOSApp.ipa

$ cd Payload/DamnVulnerableIOSApp.app

$ otool -hv DamnVulnerableIOSApp

DamnVulnerableIOSApp (architecture armv7):
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
   MH_MAGIC     ARM         V7  0x00     EXECUTE    38       4292   NOUNDEFS DYLDLINK TWOLEVEL
WEAK_DEFINES BINDS_TO_WEAK PIE

DamnVulnerableIOSApp (architecture arm64):
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
MH_MAGIC_64   ARM64        ALL  0x00     EXECUTE    38       4856   NOUNDEFS DYLDLINK TWOLEVEL
WEAK_DEFINES BINDS_TO_WEAK PIE
```

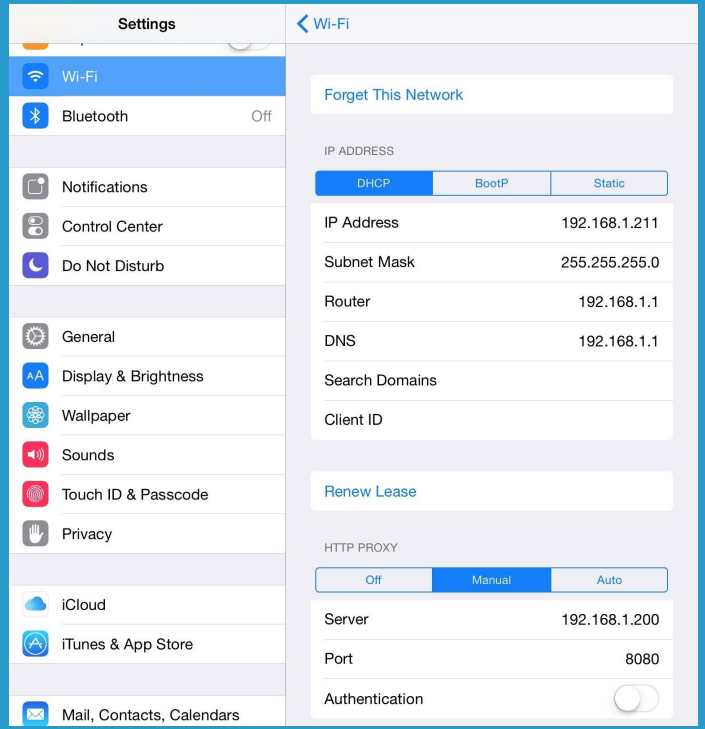
Binary Checks – SSP

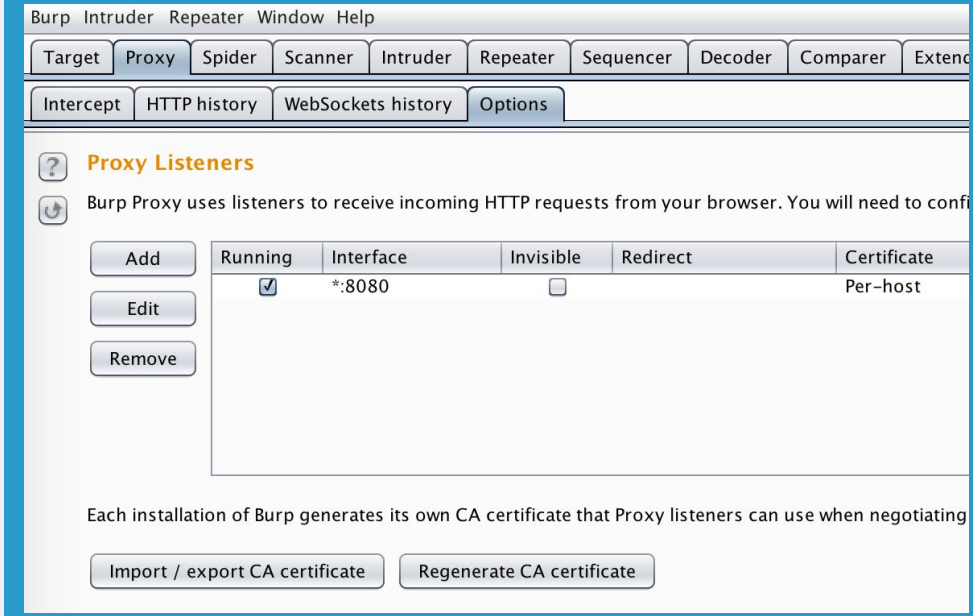
```
$ otool -Iv DamnVulnerableIOSApp  | grep stack

0x0046040c 83177 ___stack_chk_fail
0x0046100c 83521 _sigaltstack
0x004fc010 83178 ___stack_chk_guard
0x004fe5c8 83177 ___stack_chk_fail
0x004fe8c8 83521 _sigaltstack
0x00000001004b3fd8 83077 ___stack_chk_fail
0x00000001004b4890 83414 _sigaltstack
0x0000000100590cf0 83078 ___stack_chk_guard
0x00000001005937f8 83077 ___stack_chk_fail
0x0000000100593dc8 83414 _sigaltstack
```

Binary Checks – ARC

```
$ otool -Iv DamnVulnerableIOSApp  | grep release

0x0045b7dc 83156 ___cxa_guard_release
0x0045fd5c 83414 _objc_autorelease
0x0045fd6c 83415 _objc_autoreleasePoolPop
0x0045fd7c 83416 _objc_autoreleasePoolPush
0x0045fd8c 83417 _objc_autoreleaseReturnValue
0x0045ff0c 83441 _objc_release
[SNIP]
```

Setting up Burp





Jailbreak Detection - Common Methods

- Check existence of additional files, e.g.: `/bin/bash`
- Check API calls like:
  - `fork()` - forbidden on non-JB devices
  - `system(NULL)` returns 0 on non-JB and 1 on JB devices
- Check if `cydia://` URL scheme is registered

Read more:

https://www.trustwave.com/Resources/SpiderLabs-Blog/Jailbreak-Detection-Methods/

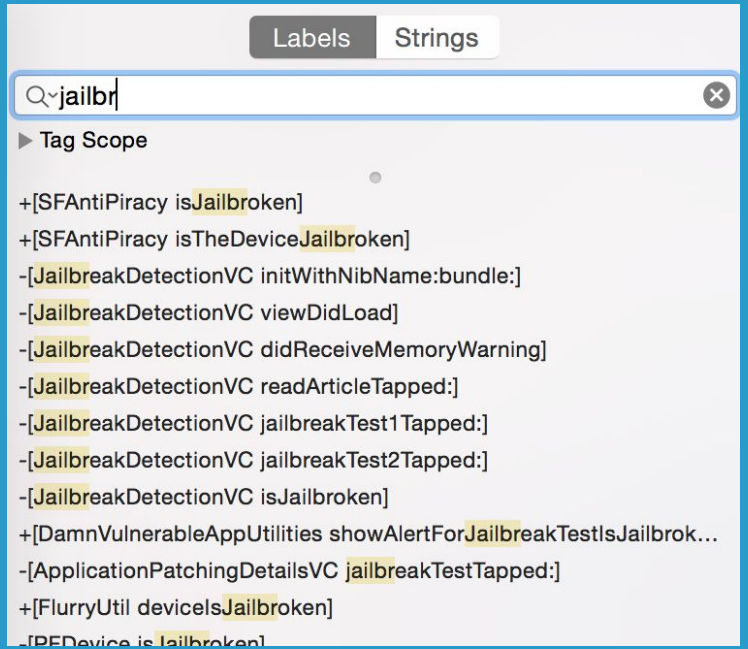
Bypassing JB detection

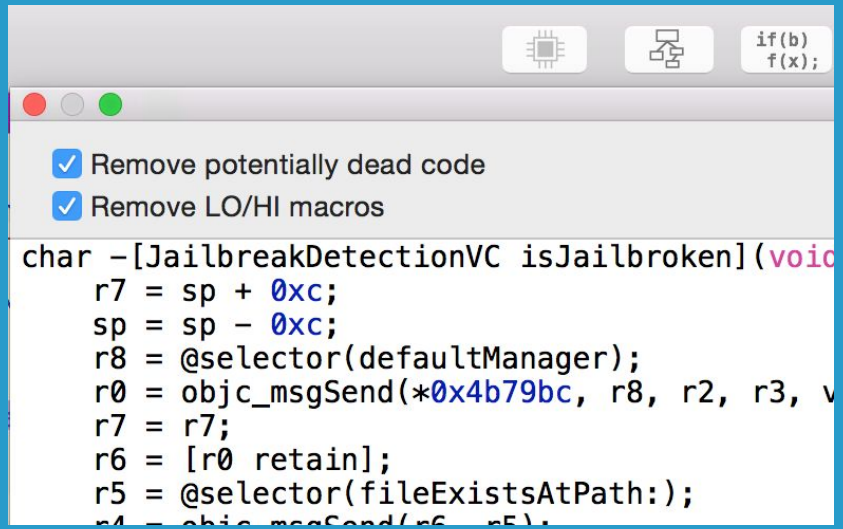
1. The easy way: xcon
2. More challenging:
    a. Debugger/Binary patching
    b. Frida
    c. Cycript

Getting Info with Class-dump

```
iOS8-jailbreak:~ root# lipo -thin armv7 DamnVulnerableIOSApp -output DVIA32

iOS8-jailbreak:~ root# class-dump DVIA32

@interface FlurryUtil : .
/DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp/YapDatabase/Extensions/Views/Internal/
{
}
+ (BOOL)appIsCracked;
+ (BOOL)deviceIsJailbroken;
```

Hopper - Disassembling





Cycript

```
iOS8-jailbreak:~ root# cycript -p 12345
cy# [SFAntiPiracy isTheDeviceJailbroken]
true
cy# a=choose(JailbreakDetectionVC)
[]
cy# a=choose(JailbreakDetectionVC)
[#"<JailbreakDetectionVC: 0x14ee15620>"]
cy# [a[0] isJailbroken]
True
```

Worth reading: http://iphonedevwiki.net/index.php/Cycript_Tricks

Cycript

```
cy# [a[0] isJailbroken]
true
cy# JailbreakDetectionVC.prototype.
isJailbroken=function(){return false}
cy# [a[0] isJailbroken]
false
```

Frida – Method Tracing

1. Install Frida on your workstation and iDevice
2. Connect iDevice to USB
3. Use `frida-trace`

```
$ frida-trace -U -f /Applications/DamnVulnerableIOSApp.app/DamnVulnerableIOSApp  -m "-
[JailbreakDetectionVC isJailbroken]:"
```

4. This creates JS hook with onEnter and onLeave callback functions:

```
    onLeave: function (log, retval, state) {
      console.log("Function [JailbreakDetectionVC isJailbroken] originally returned:"+ retval);
      retval.replace(0);
      console.log("Changing the return value to:"+retval);
    }
```

Frida - Method Tracing - Output

```
$ frida-trace -U -f /Applications/DamnVulnerableIOSApp.app/DamnVulnerableIOSApp  -m "-
[JailbreakDetectionVC isJailbroken]:"

Instrumenting functions...                                          `...
-[JailbreakDetectionVC isJailbroken]: Loaded handler at ".
/__handlers__/__JailbreakDetectionVC_isJailbroken_.js"
Started tracing 1 function. Press Ctrl+C to stop.
Function [JailbreakDetectionVC isJailbroken] originally returned:0x1
Changing the return value to:0x0
           /* TID 0x303 */
  6890 ms  -[JailbreakDetectionVC isJailbroken]
Function [JailbreakDetectionVC isJailbroken] originally returned:0x1
Changing the return value to:0x0
 22475 ms  -[JailbreakDetectionVC isJailbroken]
```

Testing for Certificate Pinning

Gradually relax requirements for server certificate, and check if traffic is successfully proxied through Burp on each stage:

1. Set Burp in proxy settings, make sure that SSL Killswitch is disabled and that Burp Profile is ***not*** installed ⟶ no certificate validation
2. Install Burp Profile (certificate) ⟶ no certificate pinning
3. Enable SSL Killswitch ⟶ certificate pinned
4. Bypass certificate pinning manually

Bypassing Certificate Pinning

- Killswitch: https://github.com/iSECPartners/ios-ssl-kill-switch
- Bypassing OpenSSL cert pinning with cycript: https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2015/january/bypassing-openssl-certificate-pinning-in-ios-apps/

Other tips:

- Certificate is often bundled in the application- look for .der or .pem
- Class-dump binary looking for strings like X509 or Cert
- Look for the following methods in the binary: NSURLSession, CFStream, AFNetworking

Source: iOS Application Security - D. Thiel

Investigating Local Storage

1. Check app Data directory /private/var/mobile/Containers/Data/Application/<app Bundle>
   a. .db - using SQLite - check with `sqlite3`
   b. plists
2. NSUserDefaults
   a. /User/Library/Preferences/
   b. /<app>/Library/Preferences/
3. Keychain protection class
   a. fileDP tool*

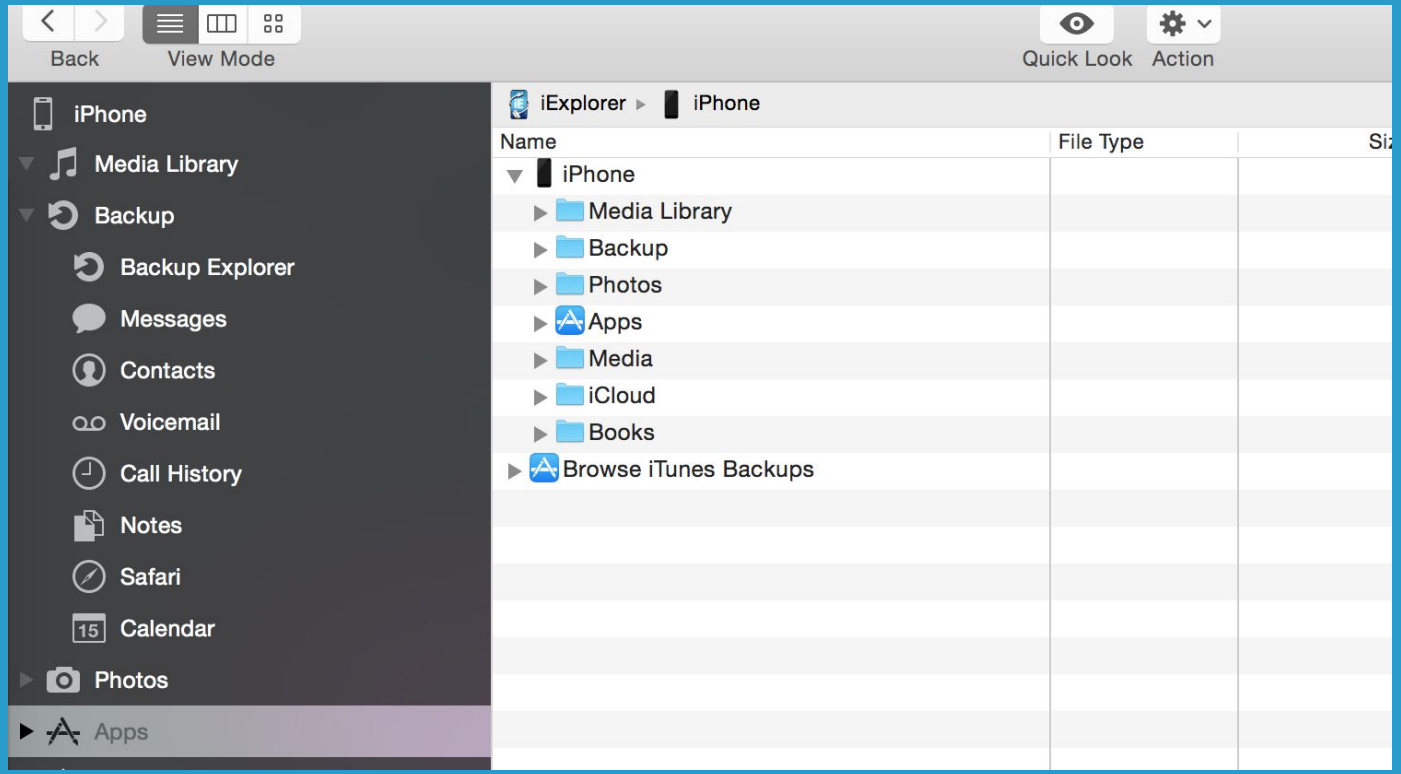
http://www.securitylearn.net/2012/10/18/extracting-data-protection-class-from-files-on-ios/

Investigating Local Storage – Cont'd

4. Application screenshots
   a. /private/var/mobile/Containers/Data/Application/<BundleID>/Library/Caches/Snapshots/
5. WebView caching
   a. /User/Library/Caches/*/Cache.db
   b. /Library/Caches/*/Cache.db
6. Forensic approach:
   a. `ls -lR --full-time` before application install, after install and after first use `diff` the results and check any files that changed
   b. use `strings` on any binary/unidentified file formats
   c. check for WAL files that may contain uncommitted DB transactions

http://www.securitylearn.net/2012/10/18/extracting-data-protection-class-from-files-on-ios/

iExplorer


Back
View Mode
Quick Look
Action
iPhone
Media Library
Backup
Backup Explorer
Messages
Contacts
Voicemail
Call History
Notes
Safari
Calendar
Photos
Apps
iExplorer
iPhone
Name
File Type
iPhone
Media Library
Backup
Photos
Apps
Media
iCloud
Books
Browse iTunes Backups

Introspy

Traced Calls

Potential Findings

Show / Hide | Show All | Hide All | DataStorage | Crypto | Network | IPC | Misc

1: CFBundleURLTypes CFBundleURLSchemes

2: CFBundleURLTypes CFBundleURLSchemes

Arguments:

```
{
  "CFBundleURLName": "com.path.path",
  "CFBundleURLScheme": "path",
  "CFBundleURLIsPrivate": "nil"
}
```

Return Value:

3: NSUserDefaults boolForKey:

Arguments:

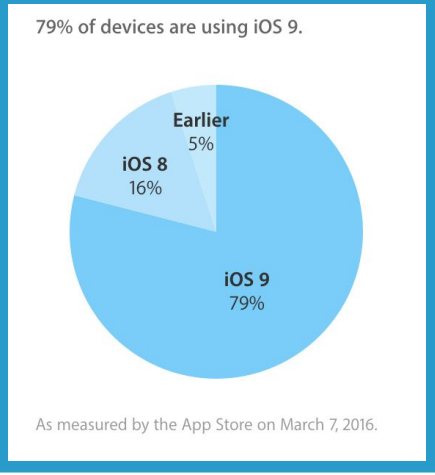
```
{
  "defaultName": "NSWriteOldStylePropertyLists"
}
```

Return Value:

```
false
```

Closing Thoughts – Threat Modelling

- Server-side bugs (LFI, SQLi, RCE) are still among most impactful
- Device data leakage is only meaningful on JB or NB devices without passcode
  - Starting from iPhone 5s the Secure Enclave protects from easy passcode bruteforcing
- Simple passcode might be an issue (1234, 0000, etc.)
- User may choose to wipe device after 10 attempts



Closing Thoughts – Pentester's Perspective

- iOS is an interesting, fast-moving ecosystem
- The iOS platform is pretty solid, but app design or implementation flaws will remain
- Moving from webapp to mobile testing is a good way to get into native (OS) security
  - This requires broad knowledge on: APIs, OS, ARM assembly, Objective C, RE

Closing Thoughts – Pentester's Perspective

- Keep up with new technologies:
  - Apple Pay
  - Health Kit
  - Yearly release of new iOS
- Lots of tools and materials for iOS < 8, but not so many for recent iOSes
- Debugger, decompiler, Frida and Mobile Substrate are your friends!

Closing Thoughts – Mobile App Security Landscape

- iOS is maturing – both from hardware and software perspective
  - Look at: Secure Enclave, Touch ID, Swift
- Still, common application flaws include:
  - home-grown crypto
  - security by obscurity
  - design flaws
  - trusting user input because “it cannot be changed”

Q&A

or drop me a line: me@skosowski.com

Interprocess Communication

- URL handlers, e.g. `mailto://`
  - open another app using its URL handler with:

    ```
    [[UIApplication sharedApplication] openURL:myURL];
    ```

  - sender can be authenticated :)
  - URL scheme can be hijacked :(

> *Note: If more than one third-party app registers to handle the same URL scheme, there is currently no process for determining which app will be given that scheme.* - Apple's Documentation

Read more:

https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html

Interprocess Communication

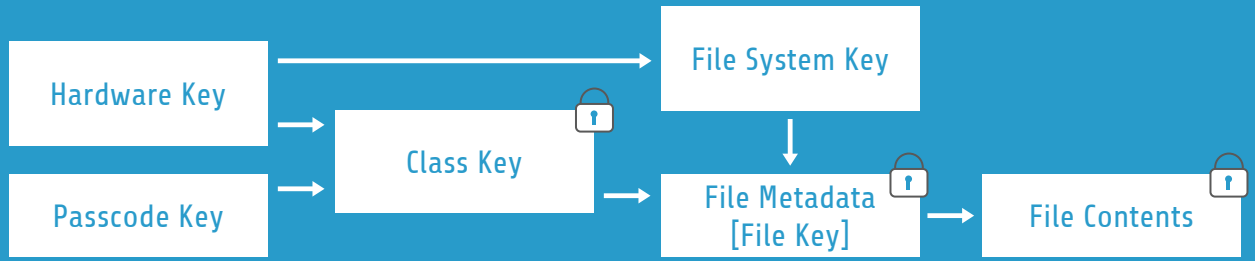
- Universal Links introduced in iOS 9
  - solve problems of `openURL`
  - no unregistered schemes - working over `https://`
- App Extensions introduced in iOS 8
  - are installed with host application and can communicate through extension points:

    Actions, Custom Keyboards, Document Providers, Photo Editing, Sharing, Today Widgets

Read more:

https://developer.apple.com/library/ios/documentation/General/Conceptual/AppSearch/UniversalLinks.html

File Data Protection

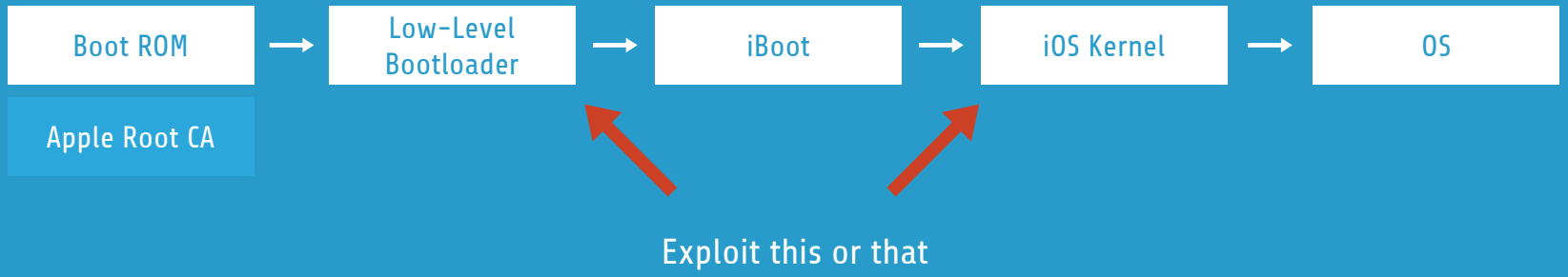


Secure Boot Chain

- Only Apple signed code is run on iDevice
- You need developer's account to write and run your apps
- Each level of boot is checked for a valid signature

Secure Boot Chain – Jailbreak

- Jailbreak permits running self-signed code
- It does not break Application Sandbox
- Usually several exploits are chained to perform jailbreak
- Look for: TaiG, Pangu, redsn0w



Protection Classes

Availability	File Data Protection	Keychain Data Protection
When Unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
When Locked	NSFileProtectionCompleteUnlessOpen	N/A
After First Unlock	NSFileProtectionCompleteUntil FirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode Enabled	N/A	kSecAttrAccessibleWhenPasscodeSetThis DeviceOnly