MOBILE PENETRATION TESTING: THE TRILOGY

Episode III
ATTACK OF THE CODE

NowSecure™

MOBILE PENETRATION TESTING: THE TRILOGY

Episode I
THE FORENSIC MENACE

Episode II
RETURN OF THE NETWORK/BACK-END

Episode III
ATTACK OF THE CODE

NowSecure™

Michael Krueger

Solutions Engineer | NowSecure

Jake Van Dyke

Mobile Security Researcher | NowSecure

Connect with NowSecure

Twitter: @NowSecureMobile

—

Subscribe to #MobSec5, blog updates, and more:

https://www.nowsecure.com/go/subscribe

Our weekly mobile security news digest

—

RSA Conference 2017: Visit us at **booth N3334**

And save a seat for NowSecure CEO Andrew Hoog's conference talk:

"How Android and iOS Security Enhancements Complicate Threat Detection"

Thursday, February 16 or Friday, February 17

Contents

- What is reverse engineering and source code analysis?
- How to do it and recommended tools
- Android app analysis example
- iOS app analysis example
- Questions

MOBILE PENETRATION TESTING AREAS OF ANALYSIS

1

Mobile forensics & data recovery

2

Network, web services, and API testing

3

Server-side penetration testing

4

Reverse engineering & code analysis

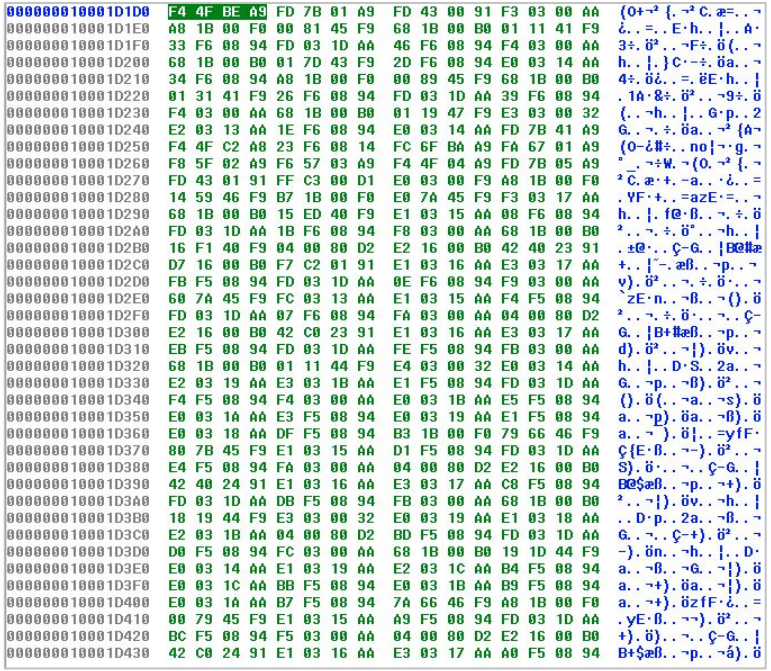
REVERSE ENGINEERING & CODE ANALYSIS

What does it mean to reverse engineer an app?

What is reverse engineering?	Taking executable code (not human readable), and translating it into something that's easier to understand. It also involves disassembling code into its component parts (e.g., classes, libraries, scripts, etc.).
What's the purpose?	• Analyzing an app's code • Identifying vulnerabilities • Finding hardcoded sensitive data • Analyzing malware • Modifying an app's functionality

Executable code compared to disassembled code

Executable code



```
0000000010001D1D0  F4 4F BE A9 FD 7B 01 A9  FD 43 00 91 F3 03 00 AA  ôO¾©ý{.©ý C.ó..ª
0000000010001D1E0  A8 1B 00 F0 00 81 45 F9  68 1B 00 B0 01 11 41 F9  ¨..ð..Eùh..°..Aù
0000000010001D1F0  33 F6 08 94 FD 03 1D AA  46 F6 08 94 F4 03 00 AA  3ö..ý..ªFö..ô...ª
0000000010001D200  68 1B 00 B0 01 7D 43 F9  2D F6 08 94 E0 03 14 AA  h..°.}Cù-ö..à..ª
0000000010001D210  34 F6 08 94 A8 1B 00 F0  00 89 45 F9 68 1B 00 B0  4ö..¨..ð.‰Eùh..°
0000000010001D220  01 31 41 F9 26 F6 08 94  FD 03 1D AA 39 F6 08 94  .1Aù&ö..ý..ª9ö..ö
0000000010001D230  F4 03 00 AA 68 1B 00 B0  01 19 47 F9 E3 03 00 32  ô..ªh..°..Gùã..2
0000000010001D240  E2 03 13 AA 1E F6 08 94  E0 03 14 AA FD 7B 41 A9  â..ª.ö..à..ª ý{A©
0000000010001D250  F4 4F C2 A8 23 F6 08 14  FC 6F BA A9 FA 67 01 A9  ôOÂ¨#ö..üoº©úg.©
0000000010001D260  F8 5F 02 A9 F6 57 03 A9  F4 4F 04 A9 FD 7B 05 A9  ø_.©öW.©ôO.©ý{.©
0000000010001D270  FD 43 01 91 FF C3 00 D1  E0 03 00 F9 A8 1B 00 F0  ýC.‘ÿÃ.Ñà..ù¨..ð
0000000010001D280  14 59 46 F9 B7 1B 00 F0  E0 7A 45 F9 F3 03 17 AA  .YFù·..ðàzEùó..ª
0000000010001D290  68 1B 00 B0 15 ED 40 F9  E1 03 15 AA 08 F6 08 94  h..°.í@ùá..ª.ö..ö
0000000010001D2A0  FD 03 1D AA 1B F6 08 94  F8 03 00 AA 68 1B 00 B0  ý..ª.ö..øh..°
0000000010001D2B0  16 F1 40 F9 04 00 80 D2  E2 16 00 B0 42 40 23 91  .ñ@ù...Ò...°B@#‘
0000000010001D2C0  D7 16 00 B0 F7 C2 01 91  E1 03 16 AA E3 03 17 AA  ×..°÷Â.‘á..ªã..ª
0000000010001D2D0  FB F5 08 94 FD 03 1D AA  0E F6 08 94 F9 03 00 AA  ûõ..ý..ª.ö..ù..ª
0000000010001D2E0  60 7A 45 F9 FC 03 13 AA  E1 03 15 AA F4 F5 08 94  `zEùü..ªá..ªôõ..ö
0000000010001D2F0  FD 03 1D AA 07 F6 08 94  FA 03 00 AA 04 00 80 D2  ý..ª.ö..ú..ª...Ò
0000000010001D300  E2 16 00 B0 42 C0 23 91  E1 03 16 AA E3 03 17 AA  â..°BÀ#‘á..ªã..ª
0000000010001D310  EB F5 08 94 FD 03 1D AA  FE F5 08 94 FB 03 00 AA  ëõ..ý..ªþõ..û..ª
0000000010001D320  68 1B 00 B0 01 11 44 F9  E4 03 00 32 E0 03 14 AA  h..°..Dùä..2à..ª
0000000010001D330  E2 03 19 AA E3 03 1B AA  E1 F5 08 94 FD 03 1D AA  â..ªã..ªáõ..ý..ª
0000000010001D340  F4 F5 08 94 F4 03 00 AA  E0 03 1B AA E5 F5 08 94  ôõ..ô..ªà..ªåõ..ö
0000000010001D350  E0 03 1A AA E3 F5 08 94  E0 03 19 AA E1 F5 08 94  à..ªãõ..à..ªáõ..ö
0000000010001D360  E0 03 18 AA DF F5 08 94  B3 1B 00 F0 79 66 46 F9  à..ªßõ..³..ðyfFù
0000000010001D370  80 7B 45 F9 E1 03 15 AA  D1 F5 08 94 FD 03 1D AA  €{Eùá..ªÑõ..ý..ª
0000000010001D380  E4 F5 08 94 FA 03 00 AA  04 00 80 D2 E2 16 00 B0  äõ..ú..ª...Òâ...°
0000000010001D390  42 40 24 91 E1 03 16 AA  E3 03 17 AA C8 F5 08 94  B@$‘á..ªã..ªÈõ..ö
0000000010001D3A0  FD 03 1D AA DB F5 08 94  FB 03 00 AA 68 1B 00 B0  ý..ªÛõ..û..ªh..°
0000000010001D3B0  18 19 44 F9 E3 03 00 32  E0 03 19 AA E1 03 18 AA  ..Dùã..2à..ªá..ª
0000000010001D3C0  E2 03 1B AA 04 00 80 D2  BD F5 08 94 FD 03 1D AA  â..ª...Ò½õ..ý..ª
0000000010001D3D0  D0 F5 08 94 FC 03 00 AA  68 1B 00 B0 19 1D 44 F9  Ðõ..ü..ªh..°..Dù
0000000010001D3E0  E0 03 14 AA E1 03 19 AA  E2 03 1C AA B4 F5 08 94  à..ªá..ªâ..ª´õ..ö
0000000010001D3F0  E0 03 1C AA BB F5 08 94  E0 03 1B AA B9 F5 08 94  à..ª»õ..à..ª¹õ..ö
0000000010001D400  E0 03 1A AA B7 F5 08 94  7A 66 46 F9 A8 1B 00 F0  à..ª·õ..zfFù¨..ð
0000000010001D410  00 79 45 F9 E1 03 15 AA  A9 F5 08 94 FD 03 1D AA  .yEùá..ª©õ..ý..ª
0000000010001D420  BC F5 08 94 F5 03 00 AA  04 00 80 D2 E2 16 00 B0  ¼õ..õ..ª...Òâ...°
0000000010001D430  42 C0 24 91 E1 03 16 AA  E3 03 17 AA A0 F5 08 94  BÀ$‘á..ªã..ª õ..ö
```

(by all appearances random junk)

Same code disassembled

```
; LoginDisclosureViewController - (void)accept

; void __cdecl -[LoginDisclosureViewController accept](struct LoginDisclosureViewController *self, SEL)
__LoginDisclosureViewController_accept_

var_20= -0x20
var_10= -0x10

STP             X20, X19, [SP,#var_20]!
STP             X29, X30, [SP,#0x20+var_10]
ADD             X29, SP, #0x20+var_10
MOV             X19, X0
ADRP            X8, #classRef_UIApplication@PAGE
LDR             X0, [X8,#classRef_UIApplication@PAGEOFF]
ADRP            X8, #selRef_sharedApplication@PAGE
LDR             X1, [X8,#selRef_sharedApplication@PAGEOFF]
BL              _objc_msgSend
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X20, X0
ADRP            X8, #selRef_displayModalViewWorking@PAGE
LDR             X1, [X8,#selRef_displayModalViewWorking@PAGEOFF]
BL              _objc_msgSend
MOV             X0, X20
BL              _objc_release
ADRP            X8, #classRef_ServiceCall@PAGE
LDR             X0, [X8,#classRef_ServiceCall@PAGEOFF]
ADRP            X8, #selRef_instance@PAGE
LDR             X1, [X8,#selRef_instance@PAGEOFF]
BL              _objc_msgSend
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X20, X0
ADRP            X8, #selRef_secondaryDisclosureWithDelegate_acceptDisclosure_@PAGE
LDR             X1, [X8,#selRef_secondaryDisclosureWithDelegate_acceptDisclosure_@PAGEOFF]
MOV             W3, #1
MOV             X2, X19
BL              _objc_msgSend
MOV             X0, X20
LDP             X29, X30, [SP,#0x20+var_10]
LDP             X20, X19, [SP+0x20+var_20],#0x20
```

(something that starts to make some sense)

Why reverse engineer an app **you developed**?

Know what's exposed if a third party reverse engineers your app

Find hard-coded API keys, credentials, etc. that make you vulnerable

Ensure debugging information is stripped from binaries

Verify that your toolchain built your code properly

Why reverse engineer an app **someone else developed**?

Clients pay you to assess the security of their mobile apps
(see previous slide)

To make sure an app is secure for your own personal use

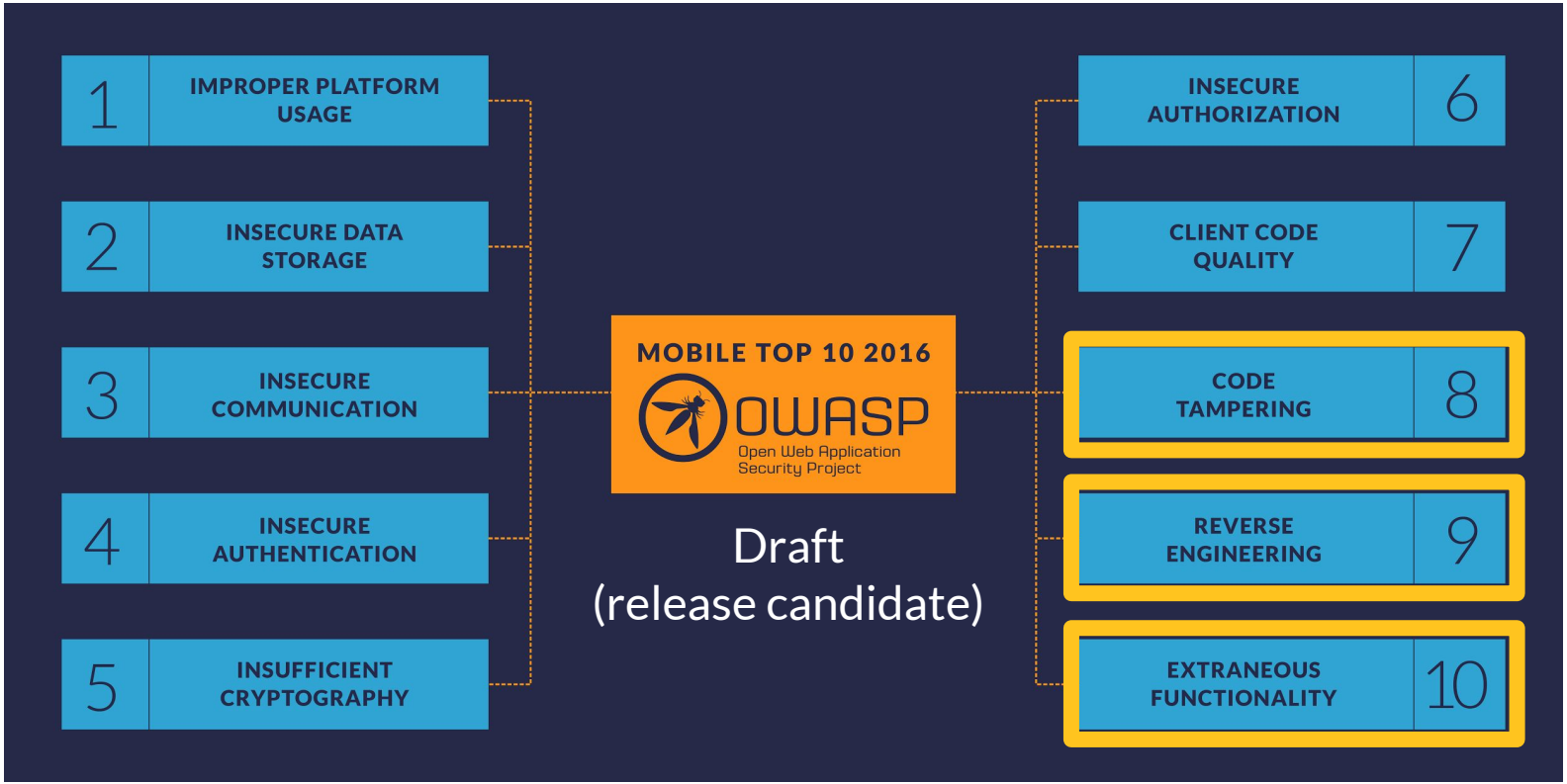
To participate in bug bounty programs offered by a vendor

To learn and practice reverse engineering

Reverse engineering answers questions such as:

- **Can you tamper with the app?**
- **Can you modify the app during runtime?**
- **Does the app allow you to pull it from a device, re-sign it, and re-install it?**
  - Can you also modify the app's main executable?
  - Can you also modify supplementary files like scripts and native libraries?
- **What can you see as a result of hooking APIs?**
  - Filesystem
  - Cryptography—dump keys, initialization vectors (IVs), identify cipher, dump the decrypted blob
  - Network—what servers and IP addresses does the app talk to?
  - Observe the app as it writes files or transmits data—is interesting/private data leaked?

Reverse engineering focus in the OWASP Mobile Top 10



Read more: "Building blocks for secure mobile development: Testing for the OWASP Mobile Top 10"

Specific vulnerabilities identified with reverse engineering

- **Insecure network communication / sensitive data leaking over the network**
- **Interprocess communication (IPC) issues**
  - Relevant to Android apps (iOS apps don't really talk to one another)
  - Content providers with directory traversal or SQLi vulns
- **Hard-coded encryption keys**
- **AES with null initialization vector (IV)**
- **Helpful logging is disabled at runtime**
  - But logging statements still available during static analysis
  - Can be used to determine names for classes and variables
  - Common in Java
  - Common in C, C++, Obj-C via non-standard toolchain (bootloaders and hypervisors)
- **Logic flaws or easily circumvented security**

Good reverse engineering tools

APKTool	For reverse engineering third party, closed, binary Android apps. Decodes resources to nearly original form and rebuilds them after making some modifications.
dex2jar	Suite of utilities for working with the classes.dex file.
jd-gui	Standalone graphical utility that displays Java source code from ".class" files.
Frida	Dynamic instrumentation framework that injects JavaScript and explores native apps on multiple platforms, including mobile platforms.
Radare (R2)	Portable reverse engineering framework.
classdump	Command-line utility for examining the Objective-C runtime information stored in Mach-O files. Also generates declarations for classes, categories, and protocols.
clutch	iOS decryption tool.
cycript	Allows you to analyze and modify running iOS or OS X apps.

A full suite of mobile tools (including most of these): Santoku Linux

ANDROID

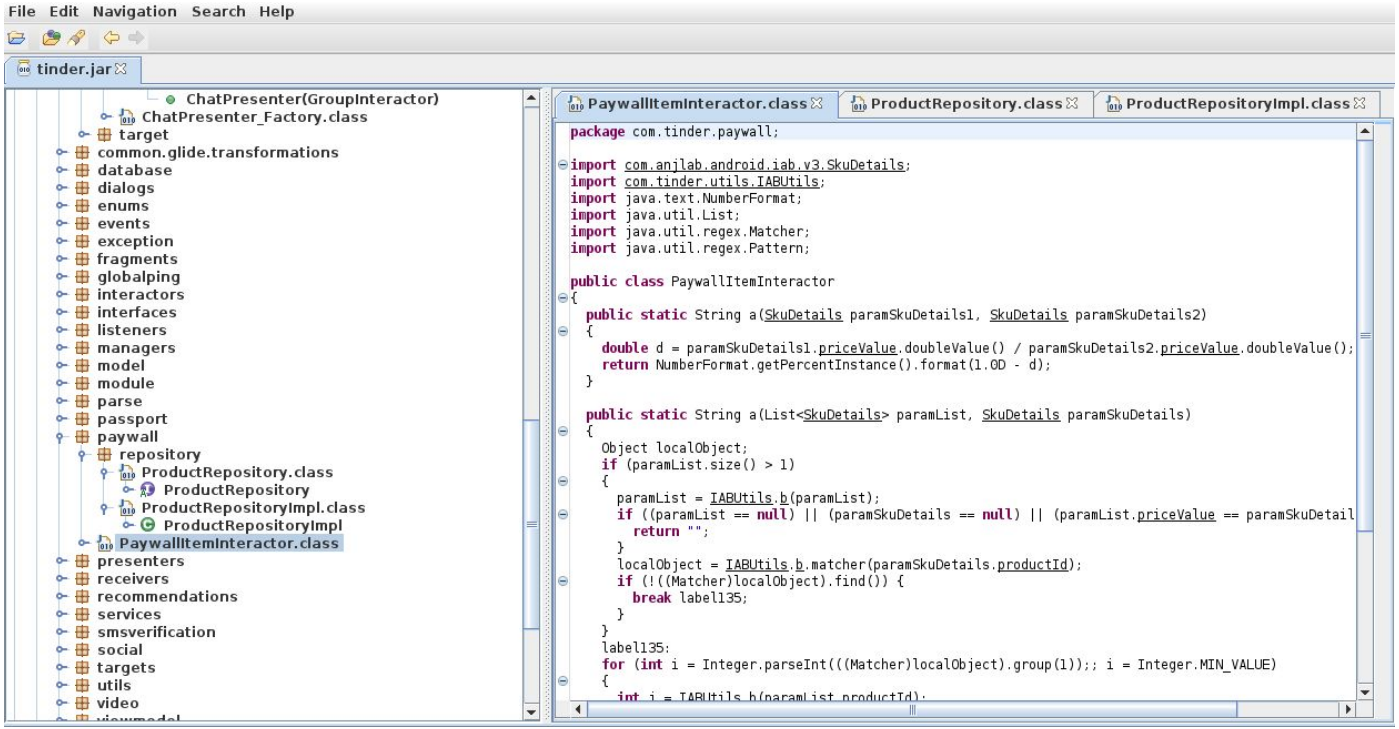
Using Apktool to get Smali code from an app

```
$ java -jar apktool_2.2.1.jar d com.tinder_1720.apk
$ cat com.tinder_1720/smali/com/tinder/paywall/PaywallItemInteractor.smali | sed '/^$/d'
.class public Lcom/tinder/paywall/PaywallItemInteractor;
.super Ljava/lang/Object;
.source "PaywallItemInteractor.java"
# direct methods
.method public constructor <init>()V
    .locals 0
    .prologue
    .line 21
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
    .line 23
    return-void
```

Analyzing an Android app's code

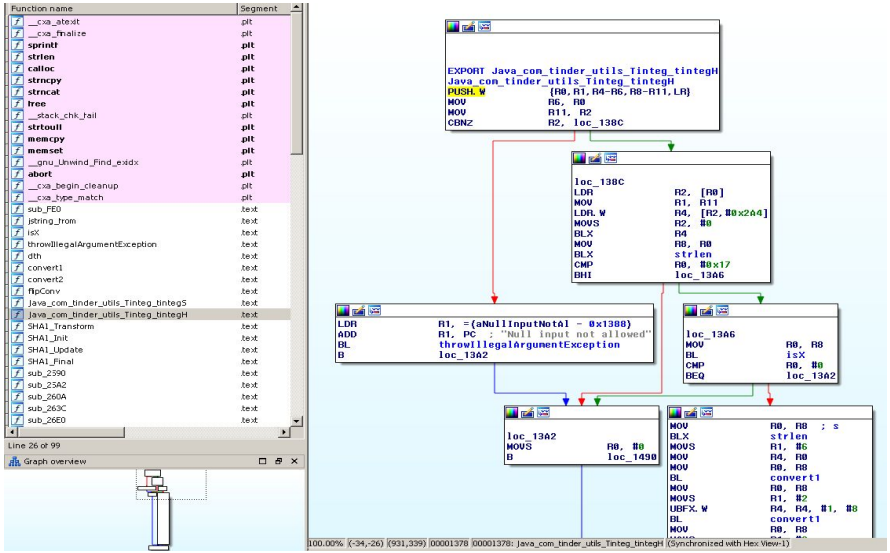
Using dex2jar, you get a .jar file that can be loaded in jd-gui for decompilation to produce Java code

```
$ dex2jar-2.0/d2j-dex2jar.sh -f -o tinder.jar com.tinder_1720.apk
```



Reverse Engineering Native Code

A disassembler takes compiled code (e.g., ELF and MACH-O files) and displays assembly and flow using boxes and arrows.



A decompiler tries to generate C code from those same binary formats.

```
while ( v13 != v18 )
{
  v19[v13] = v15[v13] | v14[v13];
  ++v13;
}
convert1(&v32, 2);
flipConv(v15);
convert1(v15, 3);
flipConv(v14);
v21 = strlen(v14);
convert1(v14, v21);
convert1(v14, 4);
v22 = (char *)calloc(v30 + n + 1 + v25 + v29 + v13, 1u);
strncpy(v22, v17, n);
strncat(v22, v20, v13);
strncat(v22, v15, v25);
strncat(v22, (const char *)&v32, v30);
strncat(v22, v14, v29);
SHA1_Init((int)&v35);
v23 = strlen(v22);
SHA1_Update((int)&v35, v22, v23);
SHA1_Final((int)&v35, (int)&v33, v24);
dth((int)&v33, (int)&v34);
convert2(&v34, 1);
```

IOS

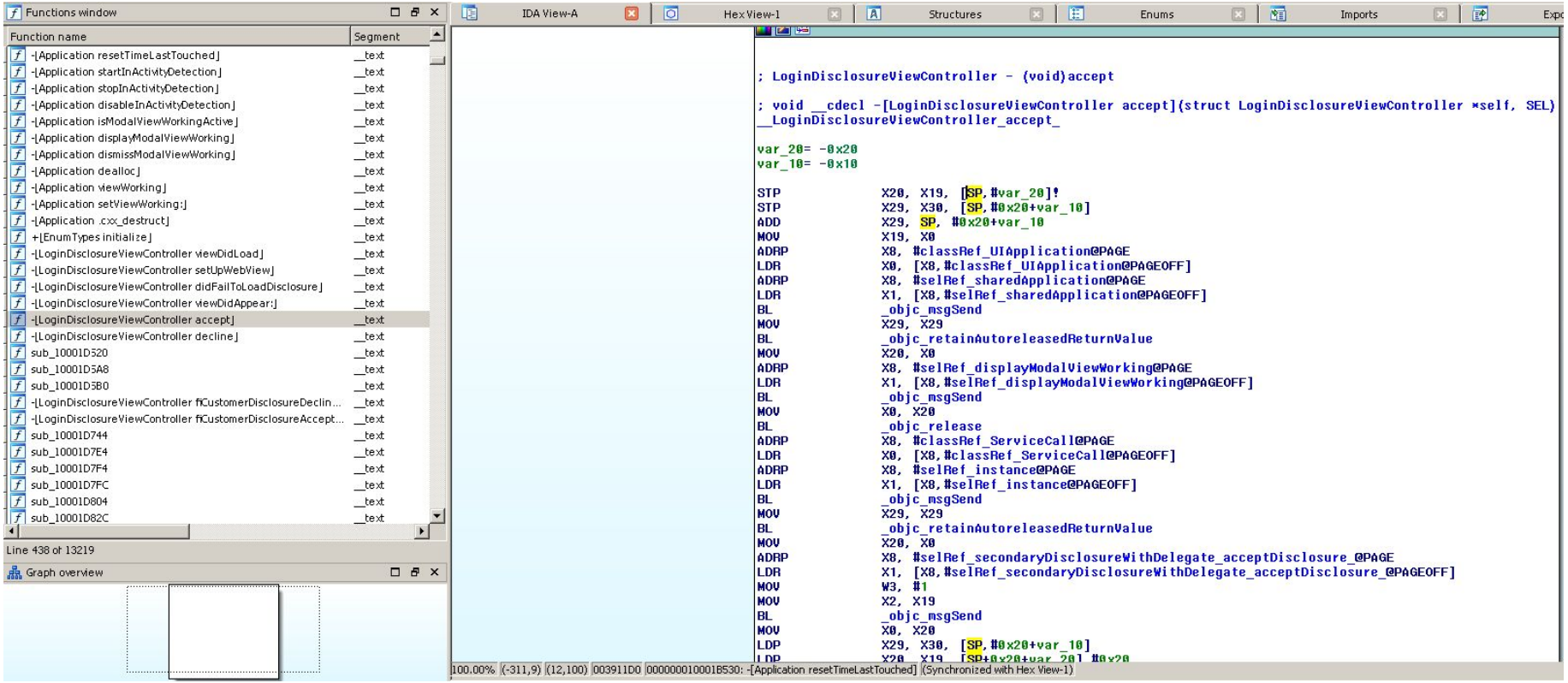
Analyzing an iOS app's code (similar concepts)

1. Get the app from the store
2. Decrypt the app on the device
3. Pull the app from the iOS device
4. Feed it to your disassembler and/or decompiler

```
iPhone:~ root# Clutch2 -i
Installed apps:
1:   Associated Credit Union <com.intuit.mobilebanking03919>
…
iPhone:~ root# Clutch2 -b 1
…
Finished dumping com.intuit.mobilebanking03919 to
/var/tmp/clutch/2734E965-9D88-41BC-830E-7B3E47623E4F
Finished dumping com.intuit.mobilebanking03919 in 2.6 seconds

$ scp -r
root@192.168.2.23:/var/tmp/clutch/2734E965-9D88-41BC-830E-7B3E47623E4F/com.intuit.mobilebanking03919
/ .
Associated Credit Union                                                    100% 7567KB
7.4MB/s   00:00
$ file com.intuit.mobilebanking03919/*
com.intuit.mobilebanking03919/Associated Credit Union: Mach-O universal binary with 2 architectures:
[arm_v7: Mach-O arm_v7 executable] [64-bit architecture=12]
```

Disassembled iOS app



ANALYZING THE OUTPUT

What Do I Do Next?

The answer is there's no right answer.

- Dig around
- Look for interesting keywords
- Start at an interesting function and follow the parameters and branches

While you're digging around, you look for things like the following:

- Logic bugs
- Additional debugging info
- Examine any closed source libraries you may be linking to in your app
- Hardcoded API keys and credentials
- Look in resources for other files that may be included in your .apk accidentally

TAKE-AWAYS: EPISODE III

1. Reverse engineering exposes flaws you might otherwise miss
2. Reverse engineer your app from the attacker's perspective ("black box" testing)
3. Apply your own creativity/ingenuity for the best results

MOBILE PENETRATION TESTING: THE TRILOGY

Episode I
THE FORENSIC MENACE

Episode II
RETURN OF THE NETWORK/BACK-END

Episode III
ATTACK OF THE CODE

Did you miss an episode?

Catch up at https://www.nowsecure.com/webinars/

Let’s talk

NowSecure
+1 312.878.1100
@NowSecureMobile
www.nowsecure.com

Subscribe to #MobSec5

A digest of the week’s mobile security news that matters

https://www.nowsecure.com/go/subscribe