

Reverse Engineering iOS Applications

Sept. 15, 2014

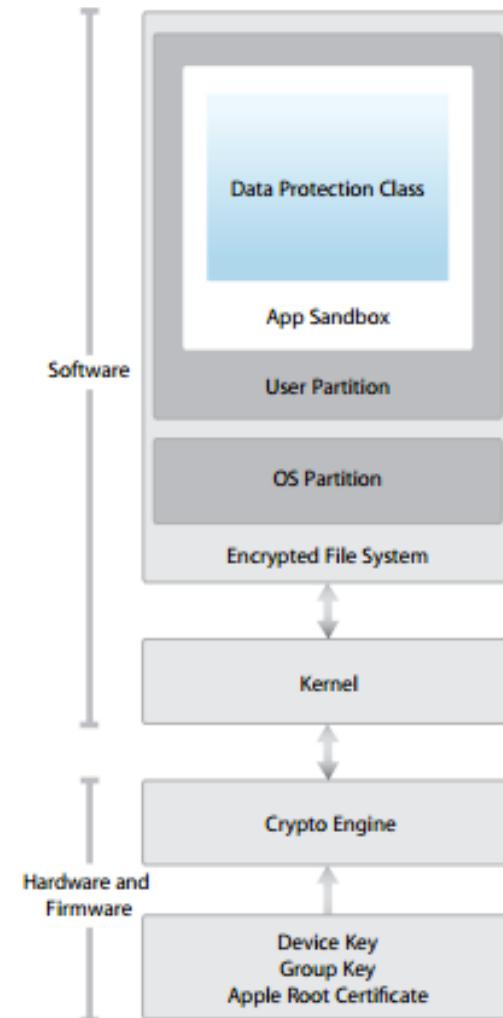
Agenda

- Technical overview
- Jailbreaking and accessing apps
- Exploring and attacking apps
- Mitigation strategies

Technical Overview

iOS Security Model

- Security is very important to Apple
- [“iOS Security” doc](#)
 - Black Hat 2012
- [Dev Center Security Overview](#)
 - Risk assessment/threat modeling
 - Coding Practices
 - Authentication



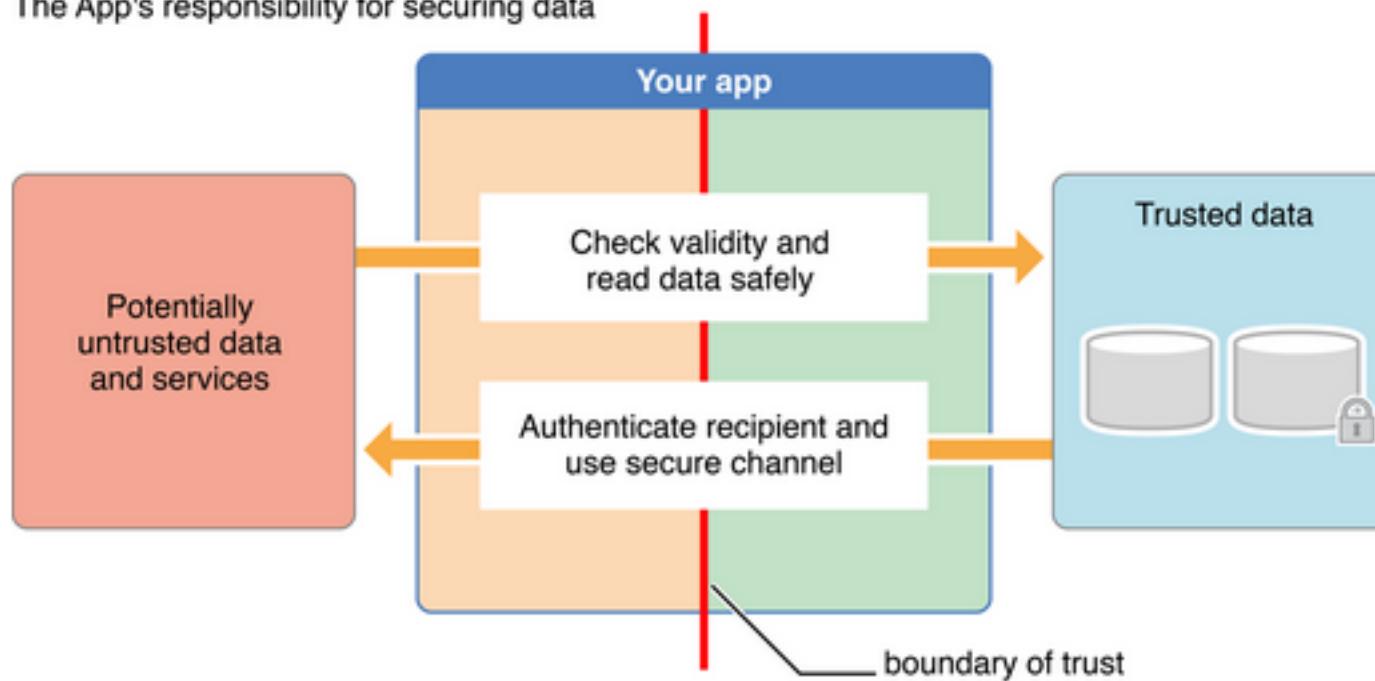
iOS Code Security

- Secure coding
 - Avoid buffer overflows, SQL injection, etc.
 - Rely on code signing, sandboxing, etc.
- Rely on OS-provided features
 - **“Don’t reinvent the wheel.** When securing your software and its data, you should always take advantage of built-in security features rather than writing your own if at all possible.”

iOS Data Security

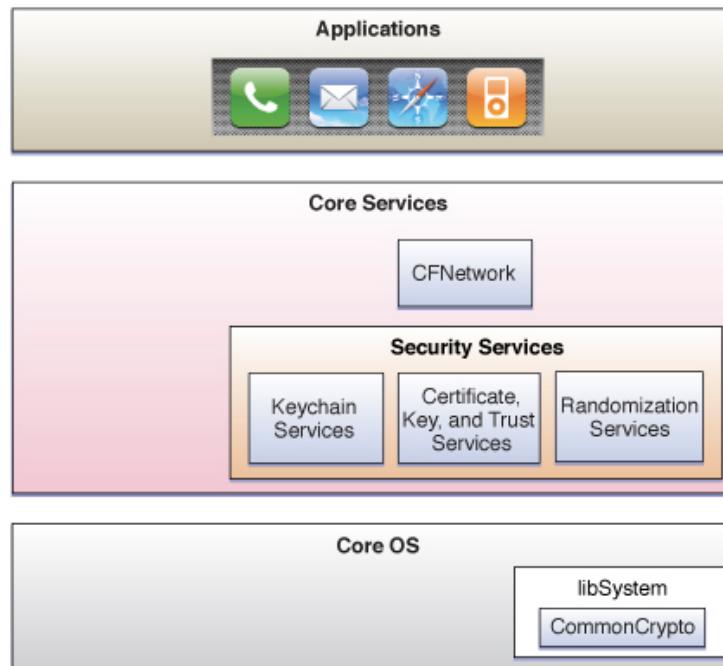
- Data security is the App's job
 - https://developer.apple.com/library/ios/DOCUMENTATION/Security/Conceptual/Security_Overview/Introduction/Introduction.html

The App's responsibility for securing data



iOS Security Overview

- Trust the OS!



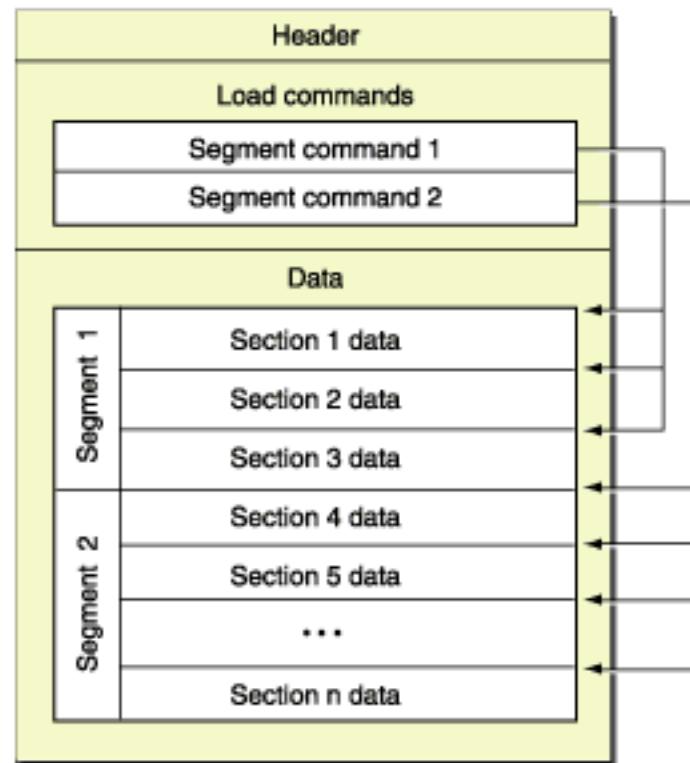
iOS Security Controls

- Why should we trust the OS?
 - Code signing
 - Anti arbitrary code execution policies
 - ASLR
 - Memory pages marked W^X
 - writable XOR executable
 - Stack canaries
 - Sandboxing
 - App encryption

iOS Security Controls

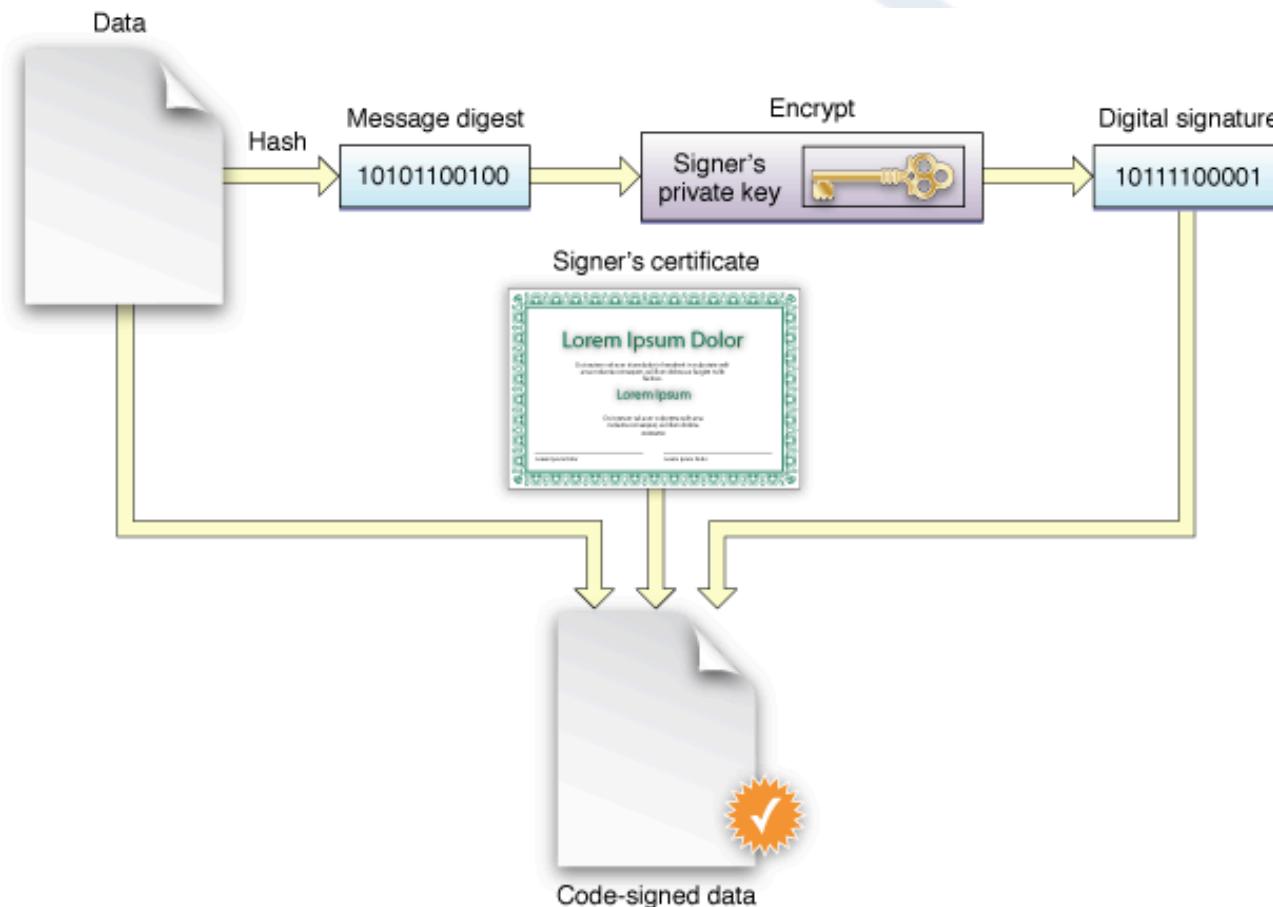
- App encryption

Figure 1 Mach-O file format basic structure



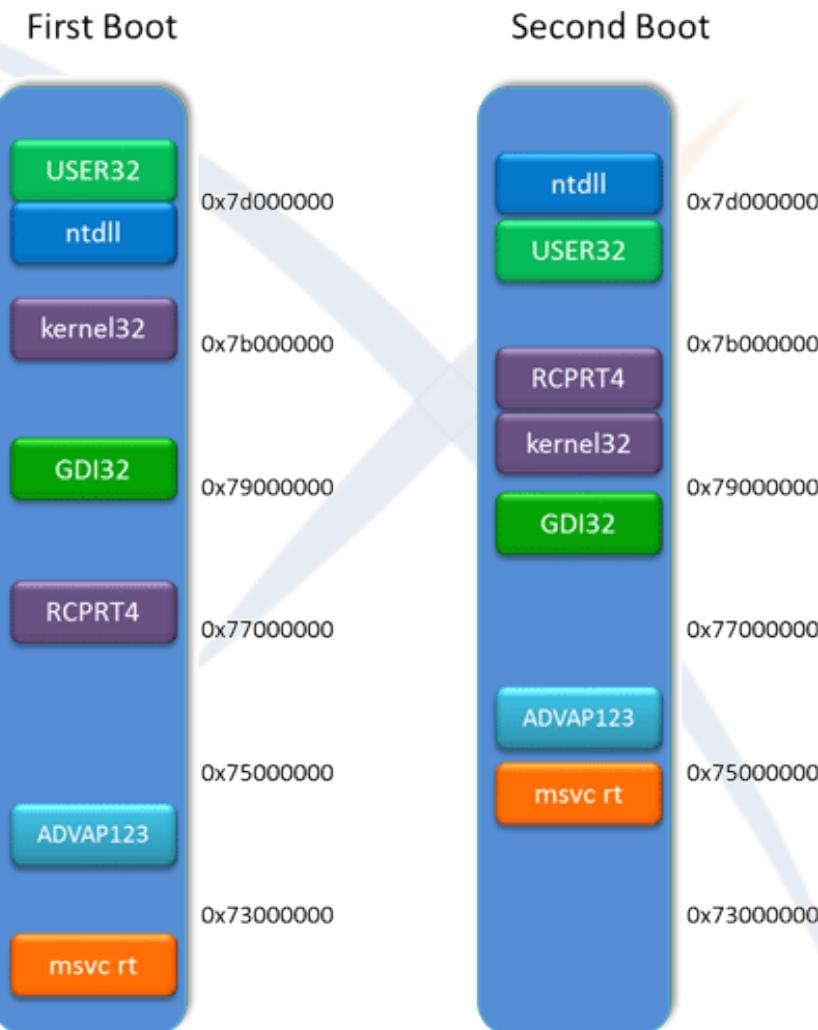
iOS Security Controls

- Code signing



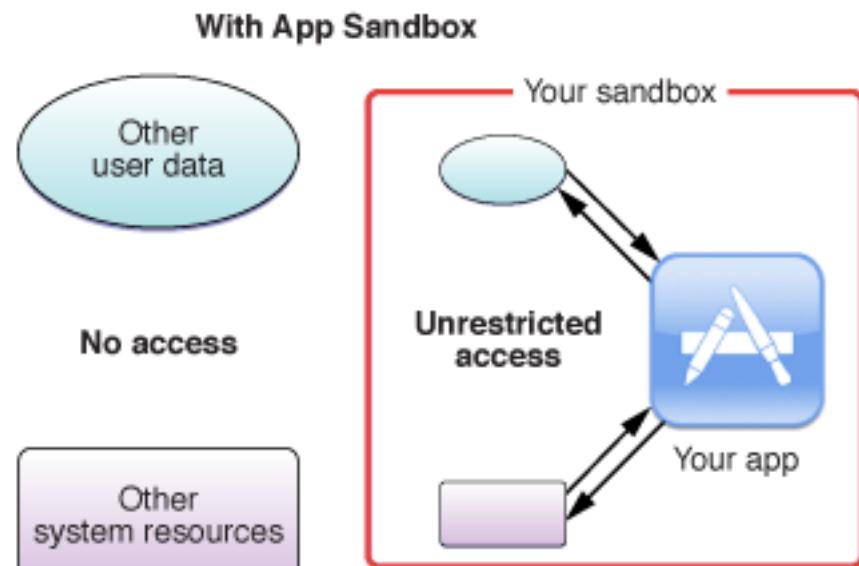
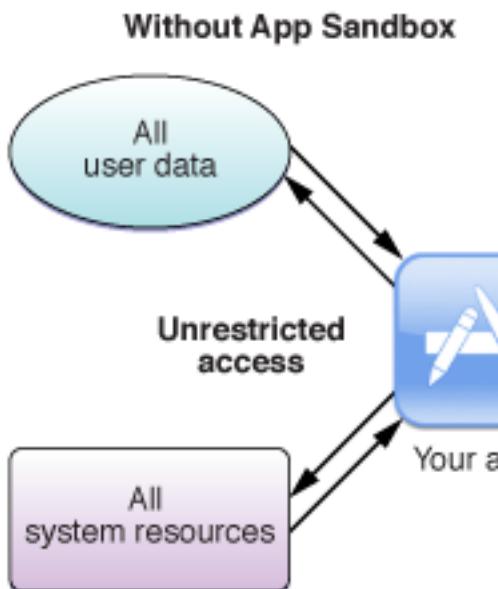
iOS Security Controls

- Code execution policies
 - ASLR
 - Address Space Layout Randomization
 - W^X Memory pages
 - No self-modifying code
 - Stack canaries



iOS Security Controls

- Sandboxing



Circumventing iOS Controls

- Jailbreaking
 - Remove iOS controls
 - Gain root access
 - Custom kernel
 - Privilege escalation

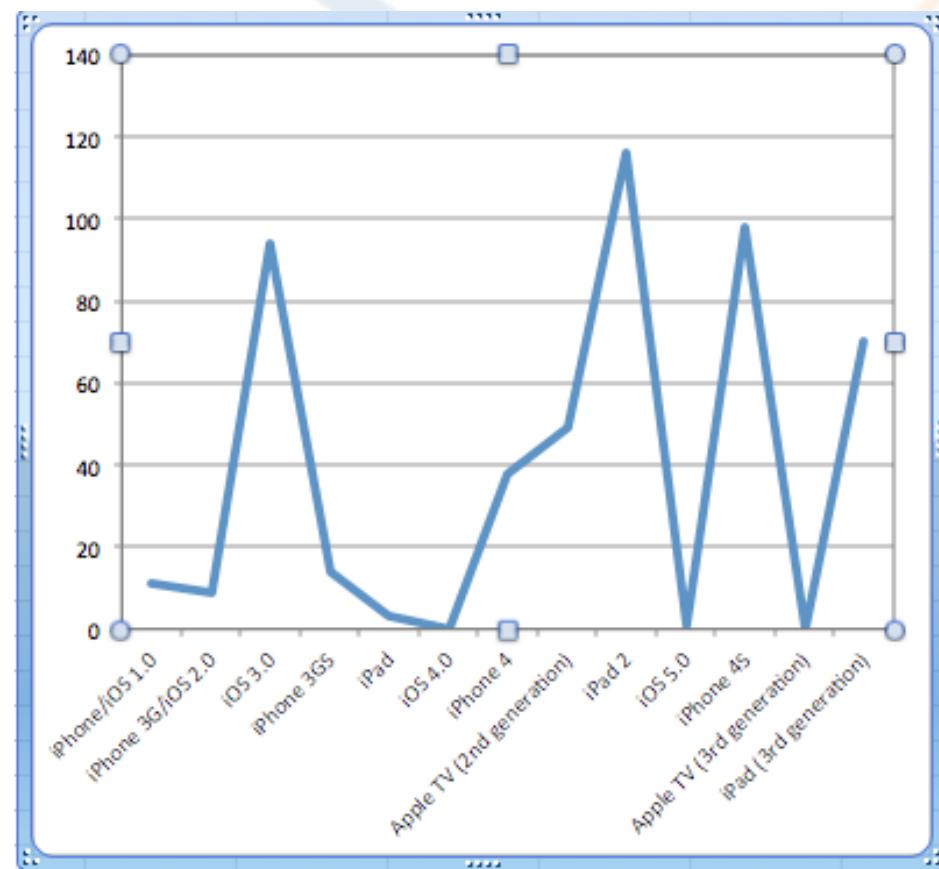


Jailbreak History

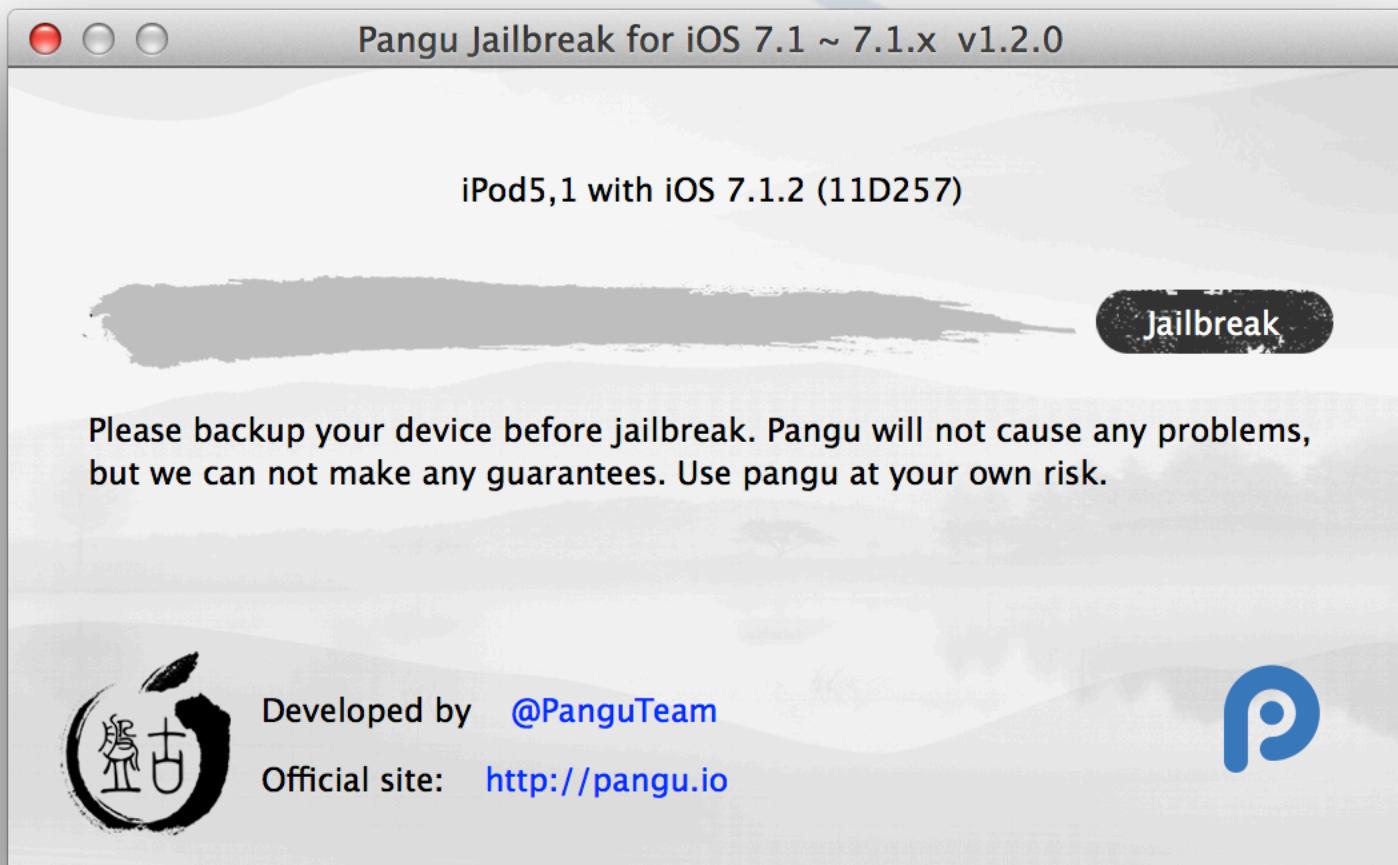
- iPhone 1.0 (June 29th 2007)
 - Jailbroken (July 10th 2007)
- 4.3.2
 - redsn0w 0.9.11x (April 2011)
- 4.3.3
 - jailbreakme.com remote jailbreak (July 2011)
- 5.1.1
 - absinthe 2.0.x (May 2012)
- 6.1
 - evasi0n (Jan 30 2013)
- 7.0
 - evasi0n7 (Dec 2013)
- 7.1
 - Pangu (Jun 23 2014)

Jailbreak History

- Time to jailbreak increases when:
 - New OS versions
 - New hardware versions
- Apple continually patches known exploits



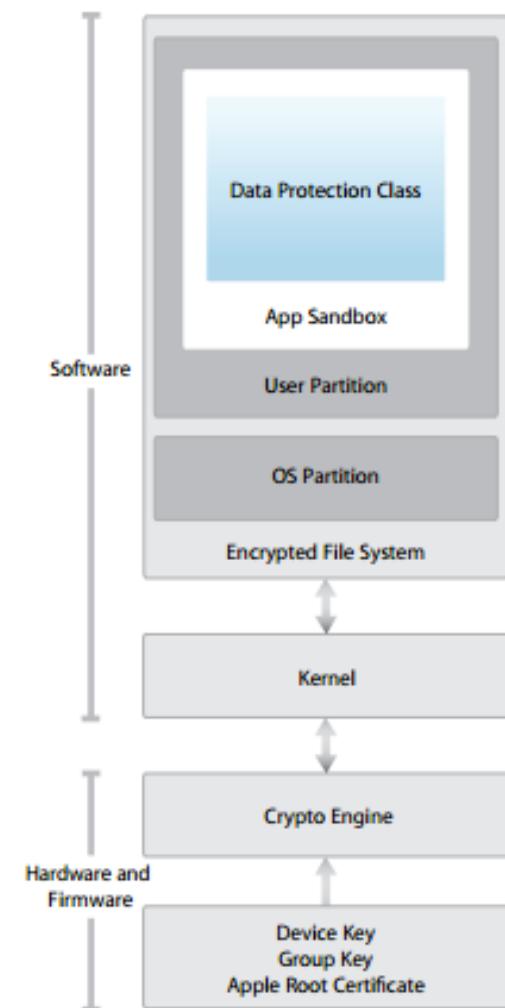
How Does Jailbreaking Work?



The screenshot shows a window titled "Pangu Jailbreak for iOS 7.1 ~ 7.1.x v1.2.0". Inside the window, it says "iPod5,1 with iOS 7.1.2 (11D257)". On the right side, there is a large, semi-transparent watermark of a savanna landscape with a lion and a zebra. In the top right corner of the window, there is a dark button labeled "Jailbreak". Below the watermark, a message reads: "Please backup your device before jailbreak. Pangu will not cause any problems, but we can not make any guarantees. Use pangu at your own risk." At the bottom left, there is a logo featuring a stylized Apple with Chinese characters inside it. Next to the logo, the text "Developed by @PanguTeam" and "Official site: <http://pangu.io>" is displayed. At the bottom right, there is a blue stylized letter "p" logo.

How Does Jailbreaking Work (really)?

1. Find an exploit
 - Exploit the bootrom (limera1n)
 - Exploit WebKit (Jailbreakme.com)
 - Privilege escalation
 - Need root to break the jail
2. Patch kernel
 - Disable signature checking, etc
3. Jailbreak the filesystem
 - Split partitions, setting +rw, remove nosu
4. Untether
 - Optional, various methods
5. Utility installation
 - tar, cp, mv, sh, etc
6. Cydia & post-install



Cydia

- Open Appstore
 - iOS dpkg



Jailbreaking Motivation

- Why jailbreak?!
 - Adding features
 - Carrier independence
 - OS customization
 - Security auditing
 - Piracy
 - Espionage/Forensics
- Why develop jailbreaks?

Exploit Types

- Remote exploit vs local exploit
 - jailbreakme.com exploit just requires a PDF download (<=4.3.3)
 - Current exploits require USB access... for now
- Certain attack vectors only require local jailbreaks
- Jailbroken devices in the field
- Discreet jailbreaking via malware
 - Requires a remote exploit
 - Removal of visible traces (Cydia etc)
 - Remote access to all iOS apps
- On Android, jailbreaking isn't necessary for app redistribution - there is no App Store or code signing

Apple's Threat Modeling

- https://developer.apple.com/library/ios/DOCUMENTATION/Security/Conceptual/Security_Overview/ThreatModeling/ThreatModeling.html
- Attacks on System Integrity
 - Attacks on system integrity [...] modify the system in such a way that it can no longer be trusted. [...] the attacker might be able to:
 - Execute malicious code
 - Impersonate a user or server
 - Repudiate an action

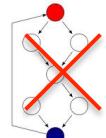
Common Application Integrity Risks



Compromise or circumvention of **security controls**, e.g., authentication, encryption, license management / checking, DRM, root / jailbreak detection



Exposure of **sensitive application information**, e.g., keys, certificates, credentials, metadata



Tampering with **critical business logic, control flows, and program operations**



Insertion of **malware or exploits** in the application and repackaging



Exposure of **application internals** (logic, vulnerabilities) via reverse-engineering



IP theft (e.g., proprietary algorithms) via reverse-engineering

Piracy and unauthorized distribution

Objective-C

What is it?

```
UIView *controllersView = [myViewController view];  
[window addSubview:controllersView];  
[window makeKeyAndVisible];
```

- `objc_msgSend(id, SEL, ...)`

Calls functions on classes using a messaging framework.

Objective-C

- C-style branching

```
sub_324FA4(34, 100, 107, "v3_ia5.c");
```



compiler

```
MOV      R1, R9
STR      R2, [R7,#0x34+var_44]
MOV      R2, R12
MOV      R3, LR
BL       sub_324FA4
```

Objective-C

- ObjC-style *messaging*

```
// When the user starts typing, show the clear button in the text field.  
textField.clearButtonMode = UITextFieldViewModeWhileEditing;
```



compiler

```
LDR      R0, [R0]  
LDR.W   R1, [R91] ; "setClearButtonMode:" metadata  
BLX      R3 ; __imp__objc_msgSend
```

MobileSubstrate

- Definition
 - Set of APIs that allow hooking of native or Obj-C functions
 - In-App or System functions
 - Installed during jailbreak
- Objective-C
 - MSHookMessage
 - Modifies message lookup table
- C/C++
 - MSHookFunction
 - Overwrites bytes to jump to custom code location

Mobile Substrate, con't

- Interfaces
 - Cycript
 - JavaScript interface to MS
 - Theos
 - Builds and installs apps/tweaks to MS
- Attack Vectors
 - Method swizzling
 - Information gathering (method names)
 - etc.

Mobile Substrate Extensions

- iOS first
- Now expanding cross-platform
 - iOS
 - Android
 - Java
 - etc
- <http://www.cydiasubstrate.com/>

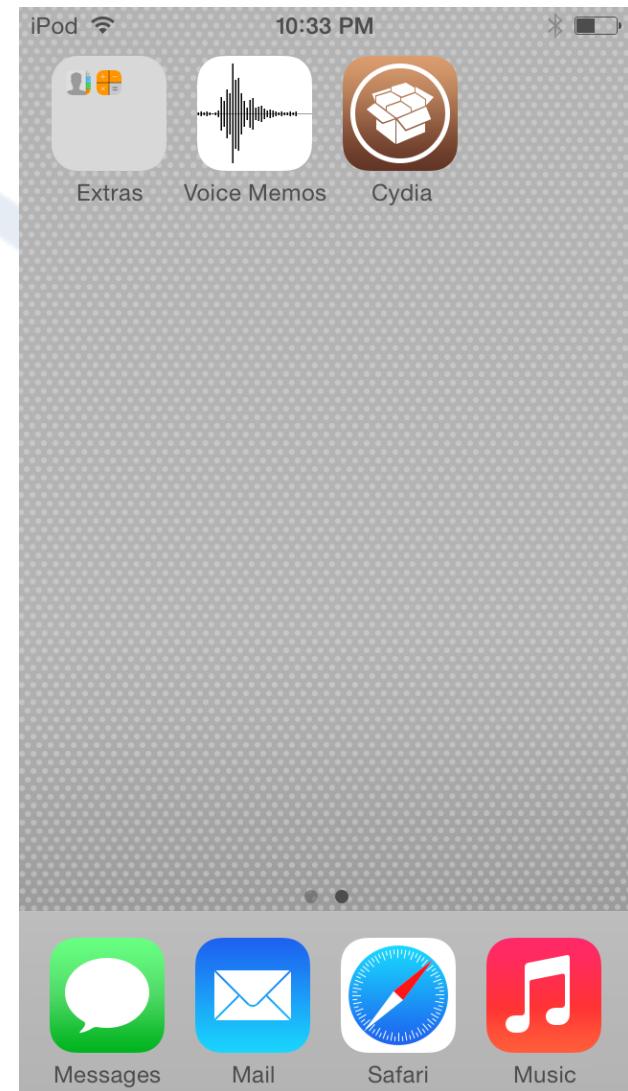
Technical Overview Wrapup

- Apple's Security Model
- Bypassing Apple's Security Model
- Objective-C
- MobileSubstrate
- Questions?

Hands-On Part 1 App Decryption

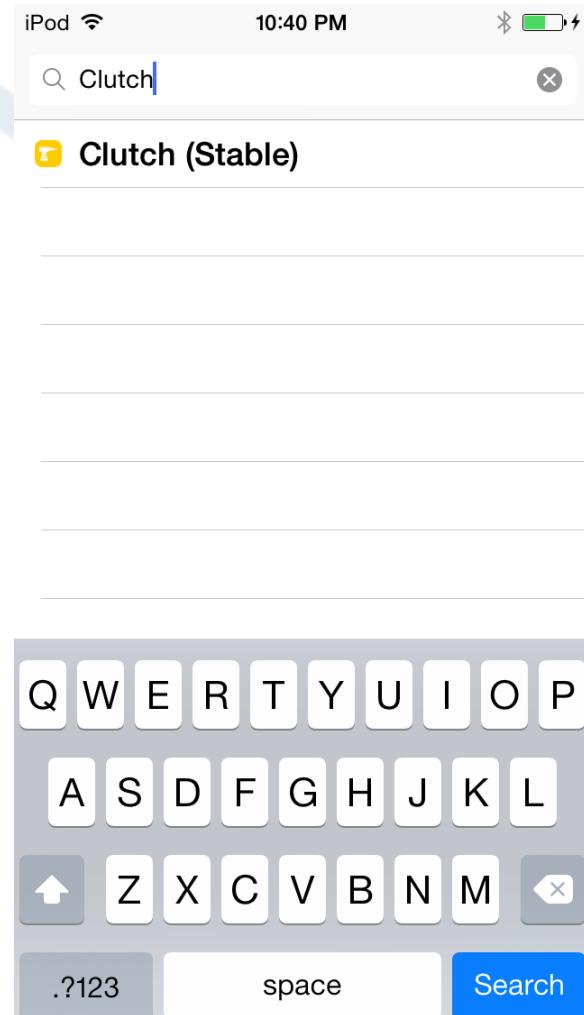
Jailbroken iPod

- iPod 5g
- iOS 7.1
- Cydia is pre-installed



Setup: Installing Cydia Apps

- All pre-installed on iPods
- Open Cydia
- Add a repo
 - <http://cydia.iphonecake.com/>
 - Default Repos host ‘known good’ Apps
- Install
 - Clutch
 - BigBoss Recommended Tools
 - AppSync

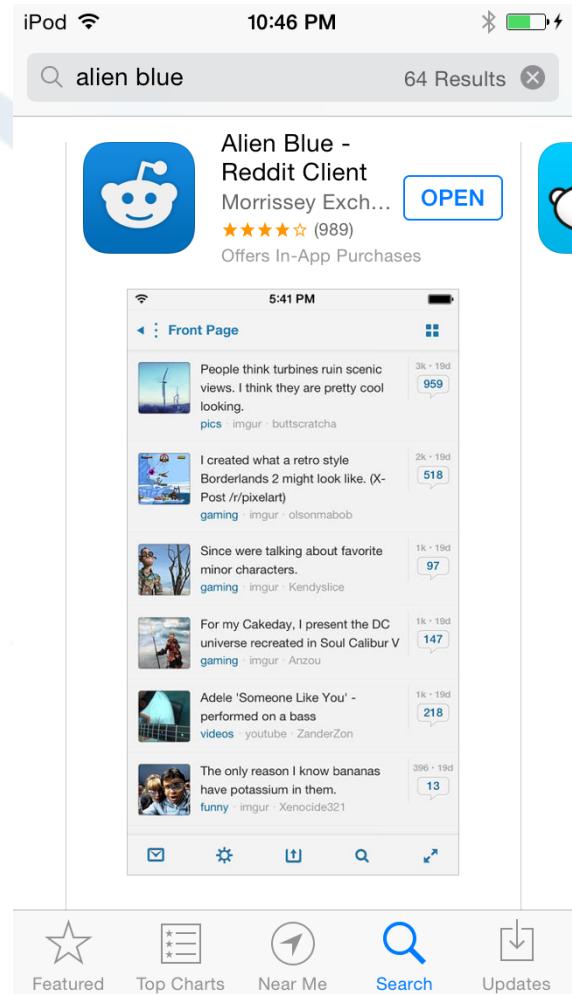


Setup: Installing Cydia apps (cont'd)

- Clutch
 - App decryption tool
- BigBoss Recommended Tools
 - otool and many other useful utilities (top, vi, etc)
 - OpenSSH
 - An ssh server so we can connect to the phone
- AppSync
 - Allows installation of arbitrary IPAs

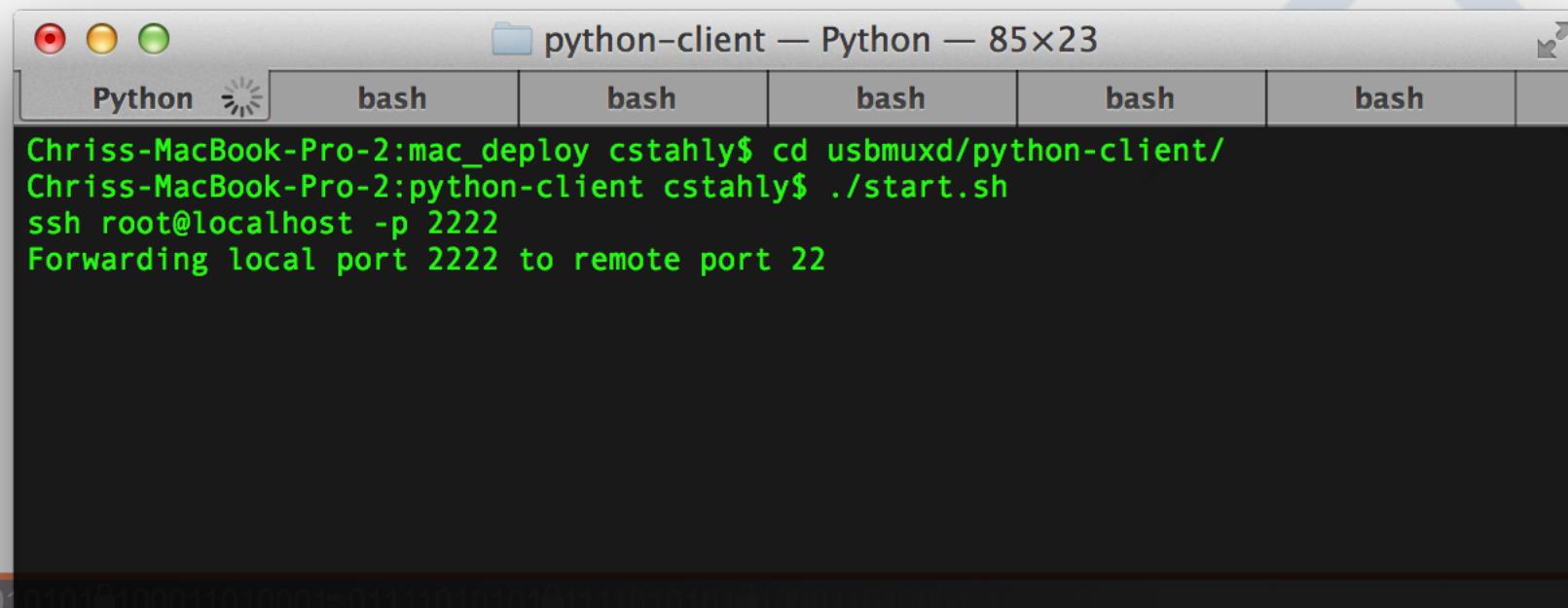
Setup: Install an App from the App Store

- Open the App Store
- Search for “Alien Blue”
 - This free app is also open-source
- This is also pre-installed



SSH to the device

- Open a Terminal
 - ⌘+Space for Spotlight
 - Type “**Terminal**”
- Start usbmuxd
 - **cd ~/usbmuxd/python-client/**
 - **./start.sh**



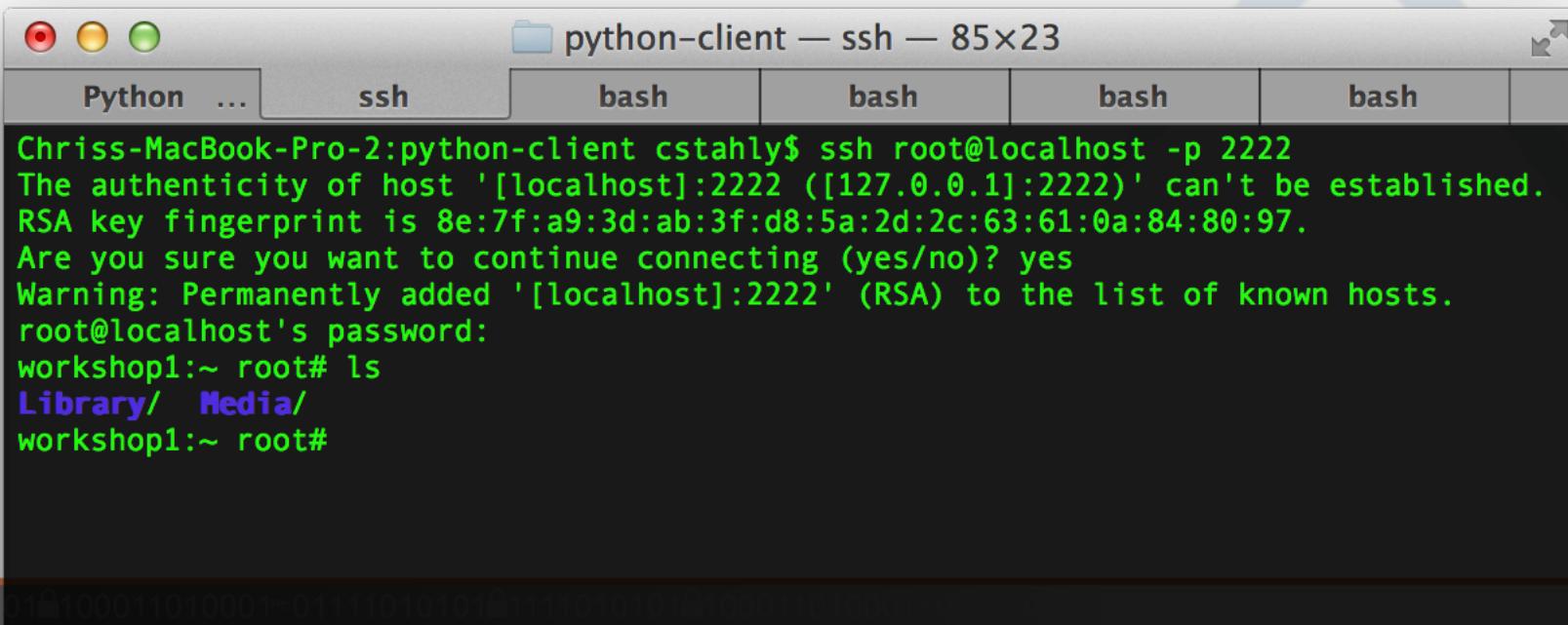
```
Chriss-MacBook-Pro-2:mac_deploy cstahly$ cd usbmuxd/python-client/
Chriss-MacBook-Pro-2:python-client cstahly$ ./start.sh
ssh root@localhost -p 2222
Forwarding local port 2222 to remote port 22
```

SSH to the device (cont'd)

- Open a new tab
 - ⌘+T
- ssh in
 - **ssh root@localhost -p 2222**
 - Default password is 'alpine'
 - Poke around the iPhone

Note:

- Keygen may take some time
- usbmuxd bridges localhost's network with the USB device



```
Chriss-MacBook-Pro-2:python-client cstahly$ ssh root@localhost -p 2222
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
RSA key fingerprint is 8e:7f:a9:3d:ab:3f:d8:5a:2d:2c:63:61:0a:84:80:97.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known hosts.
root@localhost's password:
workshop1:~ root# ls
Library/ Media/
workshop1:~ root#
```

iOS decryption

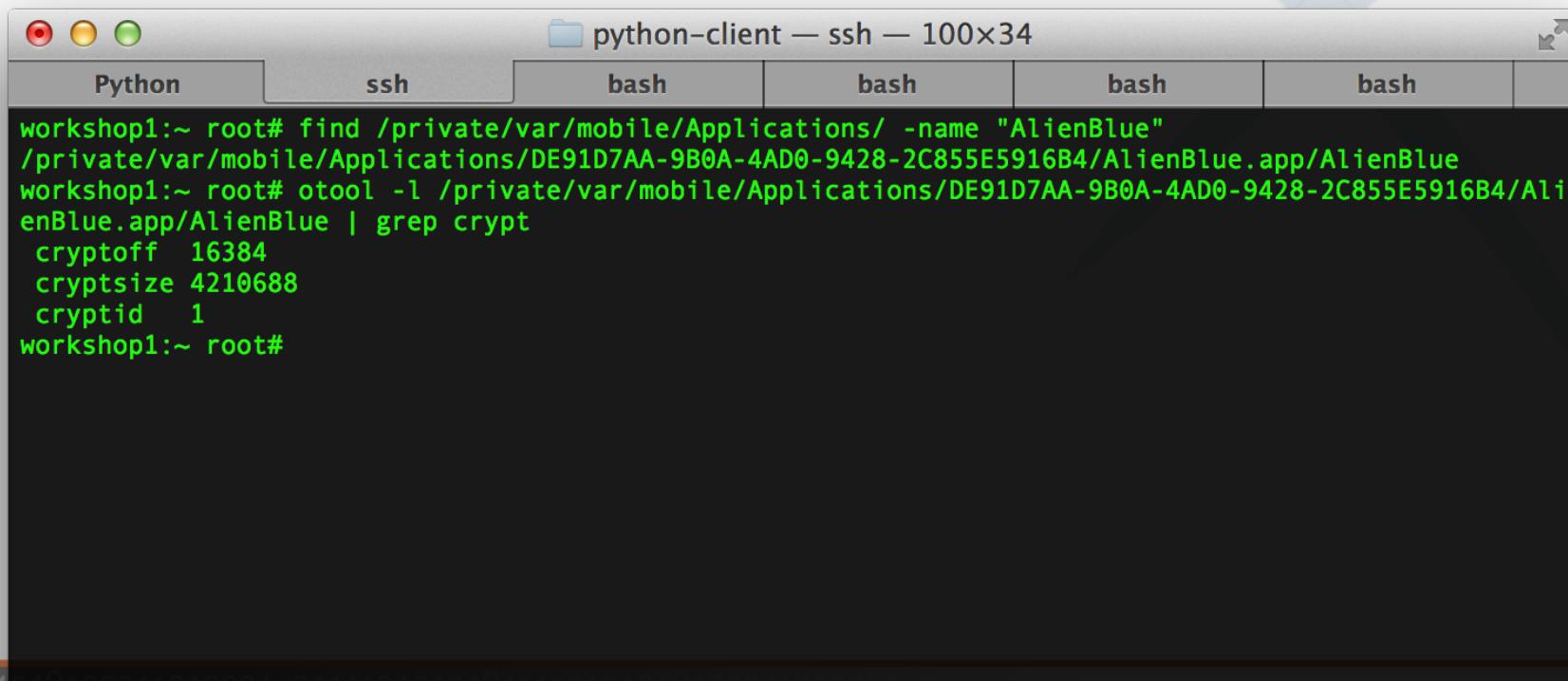
- IPAs
 - /Payload/
 - /Payload/Application.app
 - /Payload/Application.app/Application
 - (FairPlay encrypted)
 - /Payload/Application.app/[other]
 - /iTunesArtwork
 - /iTunesMetadata.plist
- Apps are installed by iOS into
“/private/var/mobile/Applications/”

Clutch

- Command-line tool to decrypt iTunes applications
 1. Loader decrypts app
 2. Clutch sets a breakpoint in loading process
 3. Dumps app from memory
 4. Fixes up load commands
- Graphical frontends exist
 - Crackulous

The Alien Blue App

- On the iPod shell
 - Find the AlienBlue installation in **/private/var/mobile/Applications**
 - Use “**otool -l**” to print load commands
 - cryptid == 1 tells the loader that this app is encrypted
 - Pipe through “| **grep crypt**” to get the crypto load commands

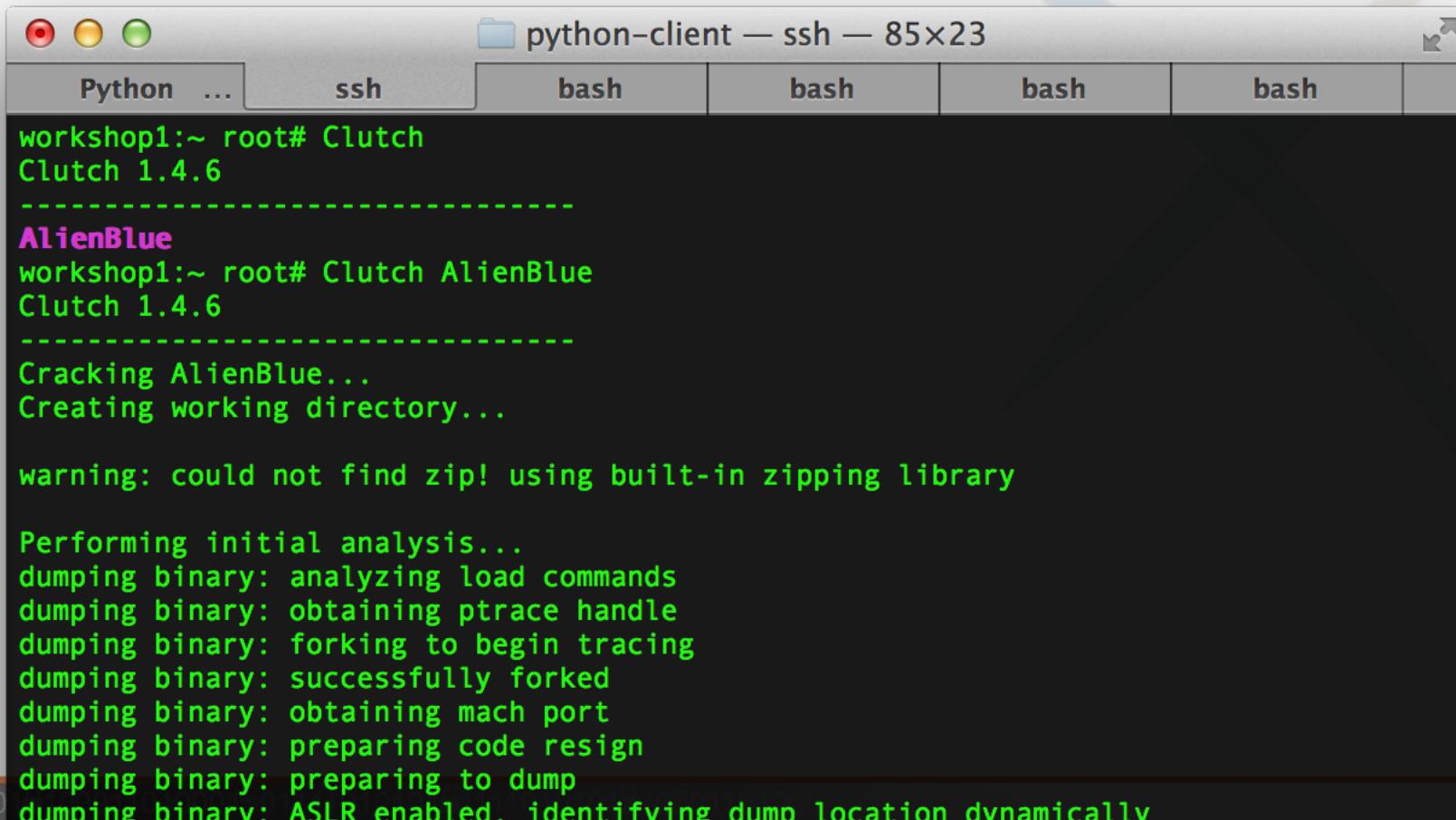


A screenshot of a terminal window titled "python-client — ssh — 100x34". The window has tabs at the top: "Python", "ssh", "bash", "bash", "bash", and "bash". The "ssh" tab is active. The terminal output is as follows:

```
workshop1:~ root# find /private/var/mobile/Applications/ -name "AlienBlue"
/private/var/mobile/Applications/DE91D7AA-9B0A-4AD0-9428-2C855E5916B4/AlienBlue.app/AlienBlue
workshop1:~ root# otool -l /private/var/mobile/Applications/DE91D7AA-9B0A-4AD0-9428-2C855E5916B4/AlienBlue.app/AlienBlue | grep crypt
cryptoff 16384
cryptsize 4210688
cryptid 1
workshop1:~ root#
```

Decrypting The App

- Run Clutch on the phone, specifying “AlienBlue” app
- App is decrypted into **/User/Documents/Cracked**



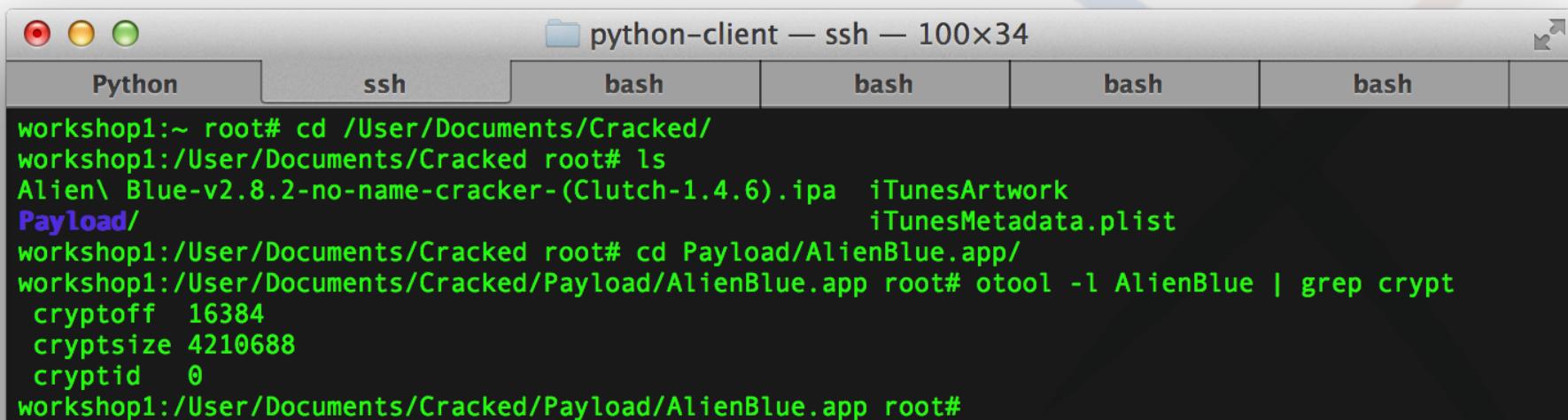
```
workshop1:~ root# Clutch
Clutch 1.4.6
-----
AlienBlue
workshop1:~ root# Clutch AlienBlue
Clutch 1.4.6
-----
Cracking AlienBlue...
Creating working directory...

warning: could not find zip! using built-in zipping library

Performing initial analysis...
dumping binary: analyzing load commands
dumping binary: obtaining ptrace handle
dumping binary: forking to begin tracing
dumping binary: successfully forked
dumping binary: obtaining mach port
dumping binary: preparing code resign
dumping binary: preparing to dump
dumping binary: ASLR enabled, identifying dump location dynamically
```

Decrypting The App (con't)

- Unzip the IPA (with “unzip” command)
- Run otool on the app again



```
workshop1:~ root# cd /User/Documents/Cracked/
workshop1:/User/Documents/Cracked root# ls
Alien\ Blue-v2.8.2-no-name-cracker-(Clutch-1.4.6).ipa  iTunesArtwork
Payload/                                              iTunesMetadata.plist
workshop1:/User/Documents/Cracked root# cd Payload/AlienBlue.app/
workshop1:/User/Documents/Cracked/Payload/AlienBlue.app root# otool -l AlienBlue | grep crypt
  cryptoff 16384
  cryptsize 4210688
  cryptid 0
workshop1:/User/Documents/Cracked/Payload/AlienBlue.app root#
```

Hands-On Part 2: App Attacking

Bank of Arxan

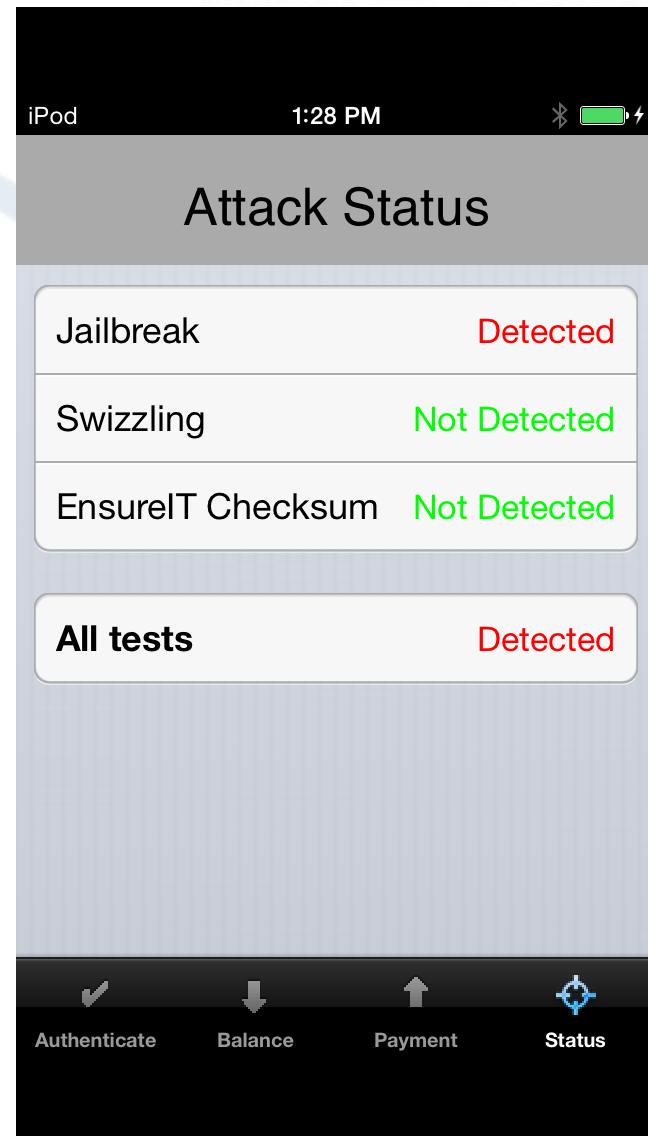
- Not Alien Blue
 - Can be decrypted the same way
- “Practice” banking app
 - Source code provided
 - ~/Desktop/Workshop/Source/
 - Client IPA
 - ~/Desktop/Workshop/Downloads/IPAs/BankDemo_client.ipa
 - We’ll install this via AppSync
 - Server at ~/Downloads/BankDemo_server
 - ~/Desktop/Workshop/Downloads/BankDemo_server
 - Runs on the Mac

Bank of Arxan Client

- Client Installation
 - Already installed via Xcode Organizer
 - AppSync facilitates this process
- Start client
 1. On first startup, set a PIN
 2. Review app

Attack Plan

- Goal
 - Remove jailbreak detection
 - Don't fail “All Tests” check



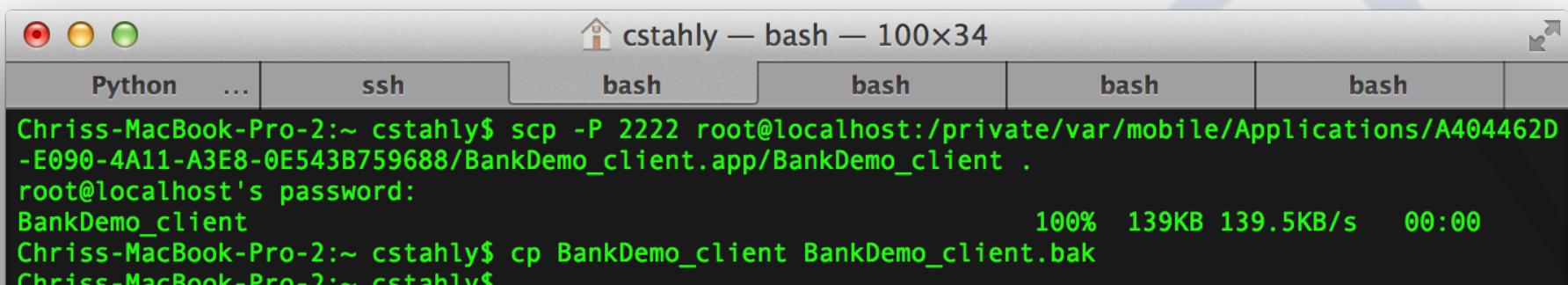
Other Attack Vectors

- Transactions
 - Modify transactions
 - Inject additional transactions
- Data gathering
 - Account information
 - Login information (username/password)
- etc

Phase 1 – Theos

Bank of Arxan Static Analysis

- Find installed app (as before, from ssh)
 - “/private/var/mobile/Applications”
- Copy app to the Mac (from the Mac)
 - “**scp -P 2222 root@localhost:[path_from_above]/BankDemo_client .**”
 - Make a backup!
 - cp BankDemo_client BankDemo_client.bak



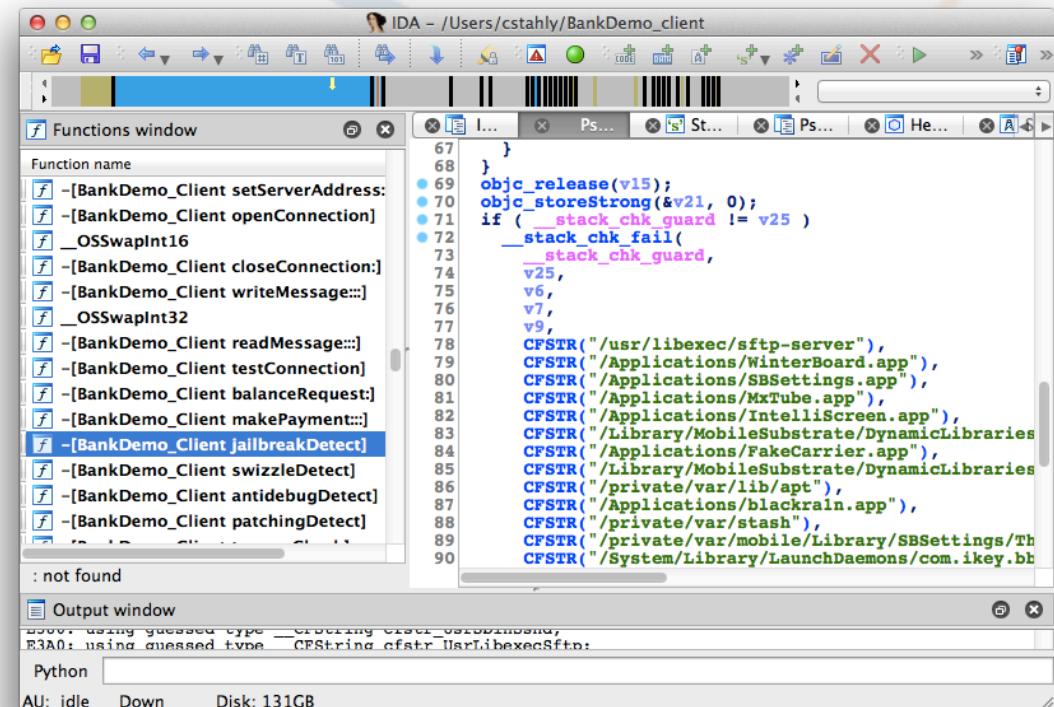
cstahly — bash — 100x34

| | | | | | | | |
|--------|-----|-----|------|------|------|------|--|
| Python | ... | ssh | bash | bash | bash | bash | |
|--------|-----|-----|------|------|------|------|--|

```
Chriss-MacBook-Pro-2:~ cstahly$ scp -P 2222 root@localhost:/private/var/mobile/Applications/A404462D-E090-4A11-A3E8-0E543B759688/BankDemo_client.app/BankDemo_client .
root@localhost's password:
BankDemo_client                                         100% 139KB 139.5KB/s  00:00
Chriss-MacBook-Pro-2:~ cstahly$ cp BankDemo_client BankDemo_client.bak
Chriss-MacBook-Pro-2:~ cstahly$
```

Bank of Arxan Static Analysis (cont'd)

- Load app in IDA
 - Strings
 - “View”
 - “Open Subviews”
 - “Strings”
 - “Search”
 - “Text”
 - Search for “Cydia”
 - Obj-C metadata
 - Functions Window
 - “Search”
 - “Text”
 - Search for “jail”



The screenshot shows the IDA Pro interface with the following details:

- Functions window:** Shows a list of Objective-C methods from the `BankDemo_Client` class.
- Assembly code:** The assembly code for the `-[BankDemo_Client jailbreakDetect]` method is displayed. It includes calls to `objc_release`, `objc_storeStrong`, and `_stack_chk_fail`. The assembly uses CFSTR to construct strings for file paths like "/usr/libexec/sftp-server" and "/Applications/WinterBoard.app".
- Output window:** Displays compiler messages such as "using guessed type - CFString cfstr libexecSftn".
- Python window:** An empty Python window is shown.
- Status bar:** Shows "AU: idle", "Down", and "Disk: 131GB".

Jailbreak Detection

```
- (int) jailbreakDetect
{
    int isJailbroken = 0;
    NSArray *jailbrokenPath = [NSArray arrayWithObjects:
        @"/Applications/Cydia.app",
        @"/usr/sbin/sshd",
        ...
        @"/private/var/lib/cydia", nil];
    for(NSString *string in jailbrokenPath)
        if ([[NSFileManager defaultManager] fileExistsAtPath:string])
            isJailbroken = 1;
        else
            isJailbroken = 0;

    return isJailbroken;
}
```

Bank of Arxan Static Analysis (cont'd)

- class-dump
 - Method prototypes
 - Class relationships
 - Field definitions
 - Etc
 - “**class-dump BankDemo_client**”
- Let's attack jailbreakDetect

```

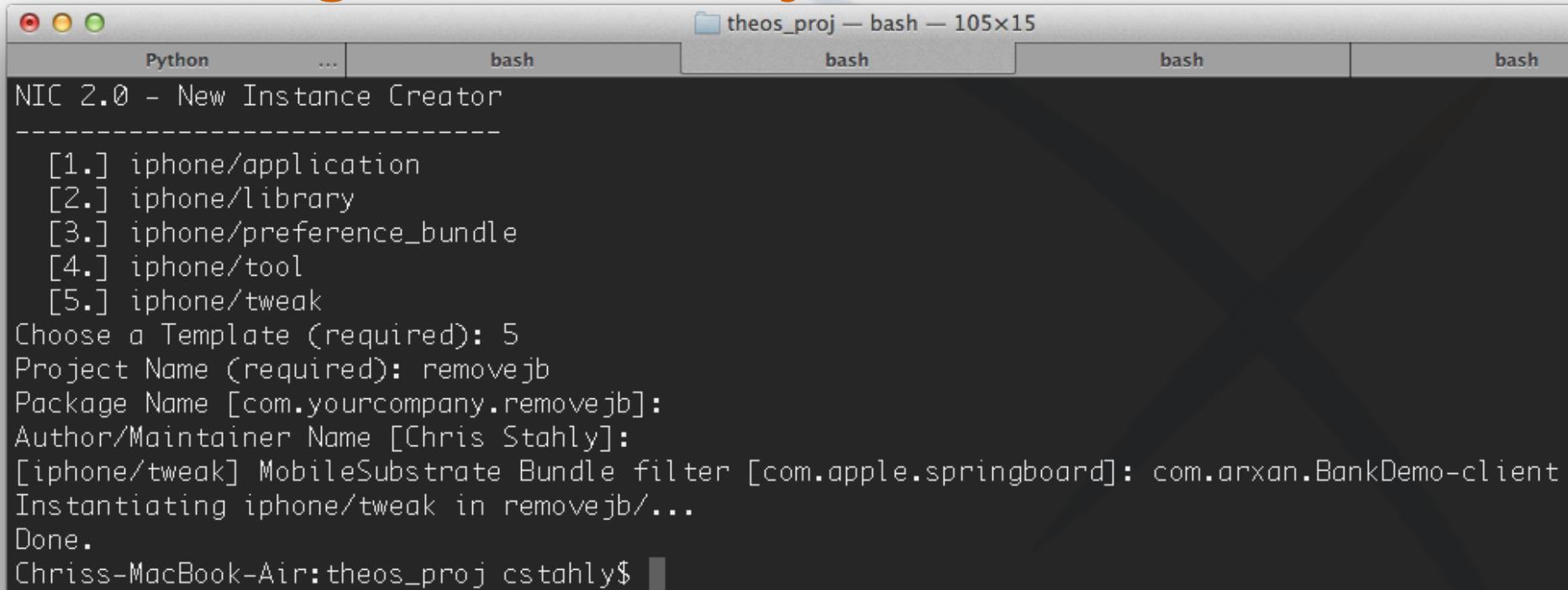
378 @interface ThirdViewController : UIViewController
379 {
380   UITextField *paymentTo;
381   UITextField *paymentNote;
382   UITextField *paymentAmount;
383   BankDemo_Client *m_BankDemoClient;
384 }
385
386 @property(nonatomic, retain) BankDemo_Client *m_BankDemoClient; // @synthesize m_BankDemoClient;
387 @property(nonatomic, retain) UITextField *paymentAmount; // @synthesize paymentAmount;
388 @property(nonatomic, retain) UITextField *paymentNote; // @synthesize paymentNote;
389 @property(nonatomic, retain) UITextField *paymentTo; // @synthesize paymentTo;
390 - (void).cxx_destruct;
391 - (void)sendButtonPressed:(id)arg1;
392 - (void)backgroundTap:(id)arg1;
393 - (BOOL)shouldAutorotateToInterfaceOrientation:(int)arg1;
394 - (void)viewDidUnload;
395 - (void)viewDidLoad;
396 - (void)didReceiveMemoryWarning;
397 - (id)initWithNibName:(id)arg1 bundle:(id)arg2;
398
399 @end
400
401 @interface BankDemo_Client : NSObject
402 {
403   char svrAddress[24];
404 }
405
406 + (id)theBankDemoClient;
407 - (int)tamperCheck;
408 - (int)patchingDetect;
409 - (int)antidebugDetect;
410 - (int)swizzleDetect;
411 - (int)jailbreakDetect;
412 - (unsigned char)makePayment:(const char *)arg1:(const char *)arg2:(int)arg3;
413 - (unsigned char)balanceRequest:(int *)arg1;

```

Using MobileSubstrate

- Attack with method swizzling
 - Jailbreak function returns 1/0
 - Swizzle to always return 0
- Theos review
 - MobileSubstrate interface
 - Works on iOS or Mac

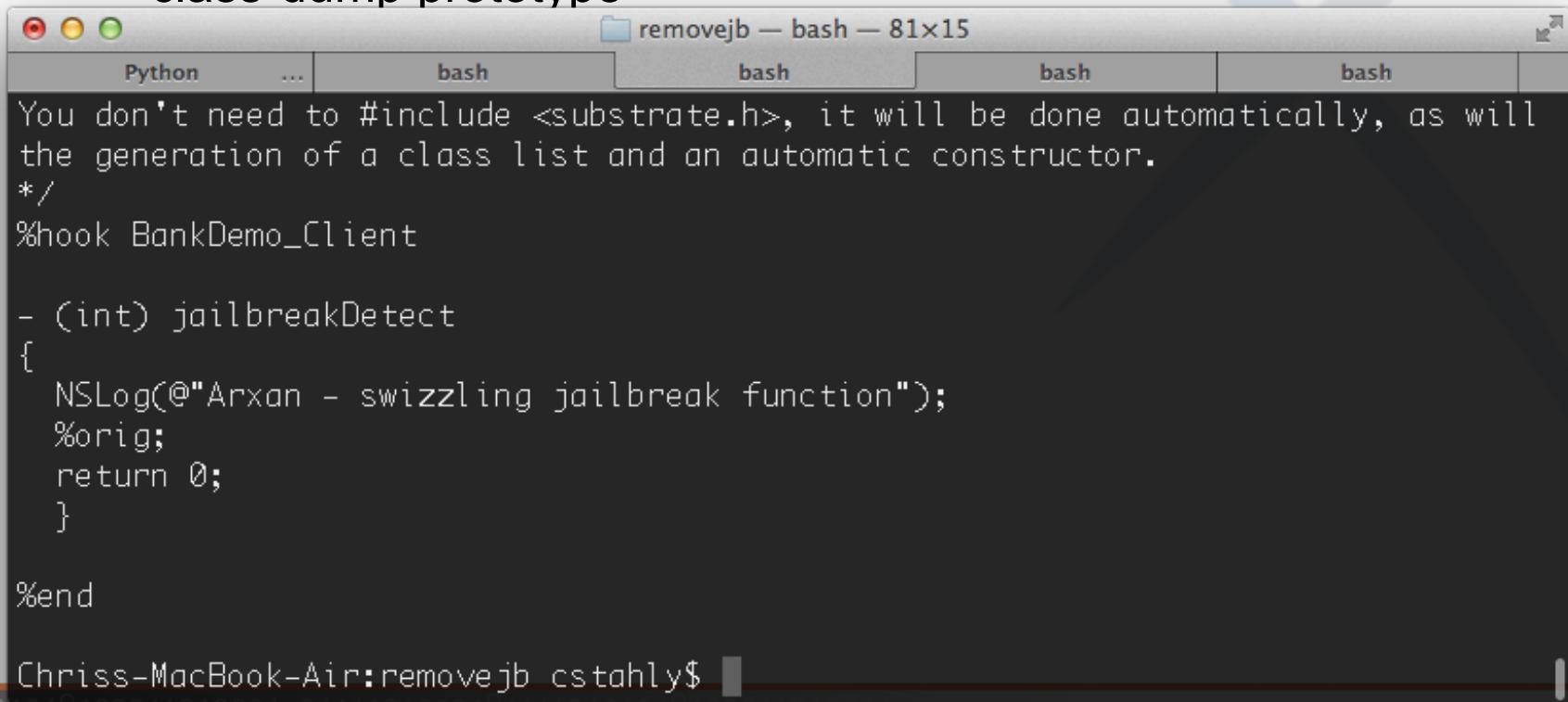
Creating a Theos Project



```
NIC 2.0 - New Instance Creator
-----
[1.] iphone/application
[2.] iphone/library
[3.] iphone/preference_bundle
[4.] iphone/tool
[5.] iphone/tweak
Choose a Template (required): 5
Project Name (required): removejb
Package Name [com.yourcompany.removejb]:
Author/Maintainer Name [Chris Stahly]:
[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]: com.arxan.BankDemo-client
Instantiating iphone/tweak in removejb/...
Done.
Chriss-MacBook-Air:theos_proj cstahly$
```

Using MobileSubstrate

- Existing project
 - `~/theos_proj/removejb`
- “**cat Tweak.xm**”
 - class-dump prototype



The screenshot shows a Mac OS X terminal window titled "removejb — bash — 81x15". The window has four tabs: Python, ..., bash, bash, bash, bash. The active tab is the third one. The terminal output is as follows:

```
You don't need to #include <substrate.h>, it will be done automatically, as will
the generation of a class list and an automatic constructor.
*/
%hook BankDemo_Client

- (int) jailbreakDetect
{
    NSLog(@"Arxan - swizzling jailbreak function");
    %orig;
    return 0;
}

%end
```

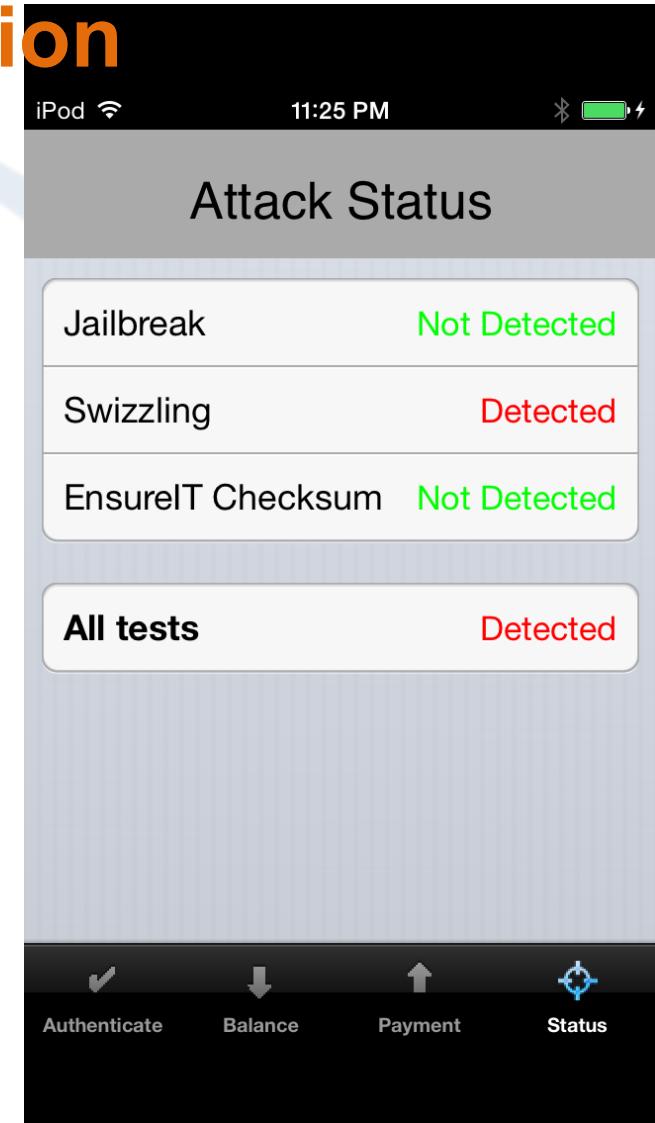
Chriss-MacBook-Air:removejb cstahly\$

Building Theos Tweaks

- Build app
 - “make”
 - “make package”
- Copy package to phone (on Mac)
 - “scp -P 2222 com.yourcompany[snip].deb root@localhost:”
- Install tweak (on iPod)
 - “dpkg -i com.yourcompany[snip].deb”

Removing Jailbreak Detection

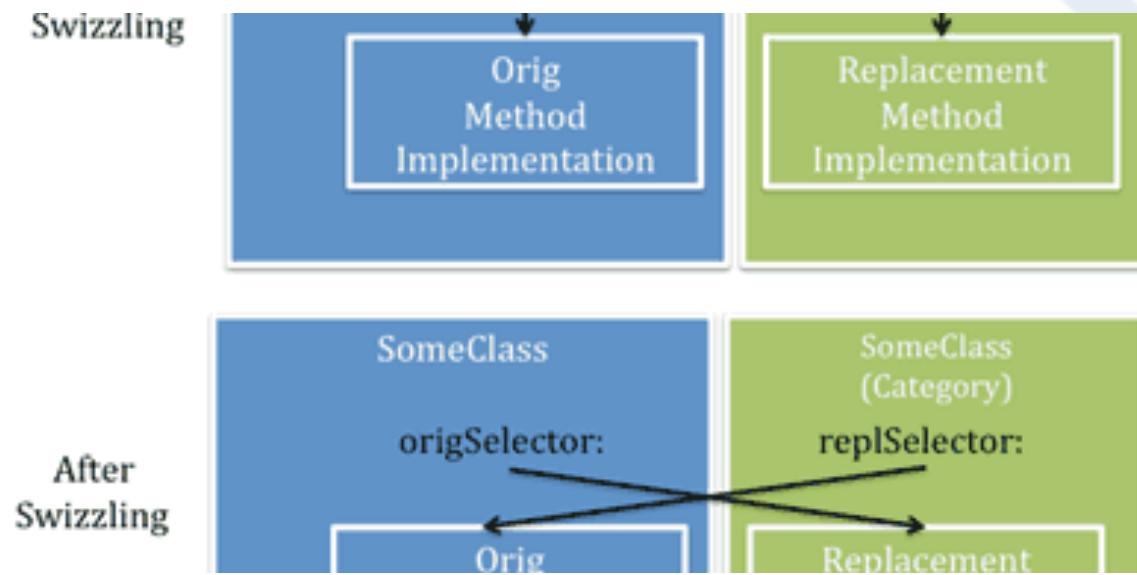
- Bounce SpringBoard
 - “**killall SpringBoard**”
- Rerun Bank of Arxan client
- Results?



Phase 2 - Patching

Swizzling Detection

- Where is the objc function?
 - Ask the loader (dyld)



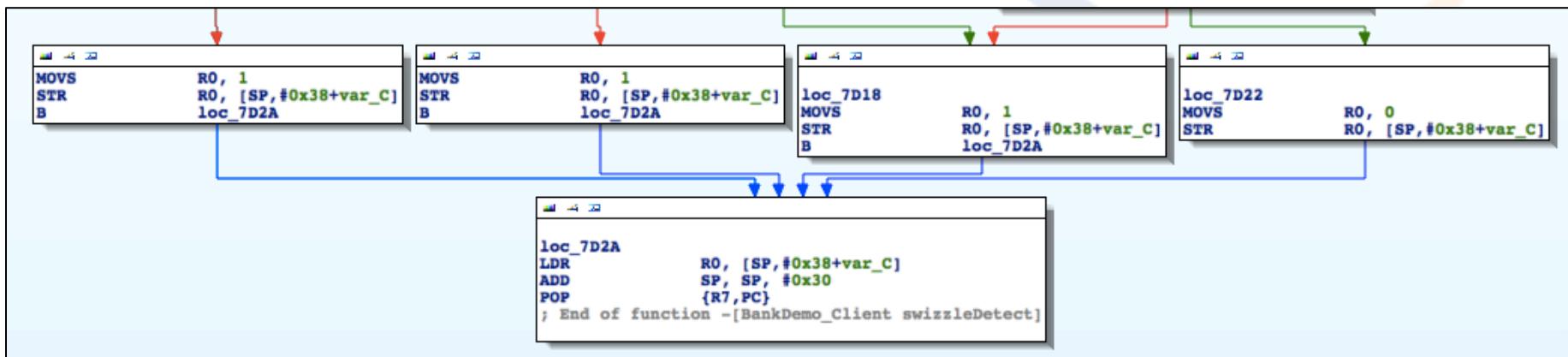
Swizzling Detection Analysis

- Back to IDA/Hex-Rays (or source code)

```
● 15  v11 = self;
● 16  v10 = a2;
● 17  v2 = _dyld_get_image_header(0);
● 18  v8 = getsectdatafromheader(v2, "__TEXT", "__text", &size);
● 19  v9 = _dyld_get_image_vmaddr_slide(0);
● 20  v8 += v9;
● 21  v6 = objc_getClass("BankDemo_Client");
● 22  if ( v6 )
● 23  {
● 24      v3 = class_getInstanceMethod(v6, "jailbreakDetect");
● 25      if ( v3 )
● 26      {
● 27          v5 = method_getImplementation(v3);
● 28          v12 = v5 < (unsigned int)v8 || v5 > (unsigned int)&v8[size];
● 29      }
● 30      else
● 31      {
● 32          v12 = 1;
● 33      }
● 34  }
● 35  else
● 36  {
● 37      v12 = 1;
● 38  }
● 39  return v12;
```

Patching the App

- Swizzle detection method control flow

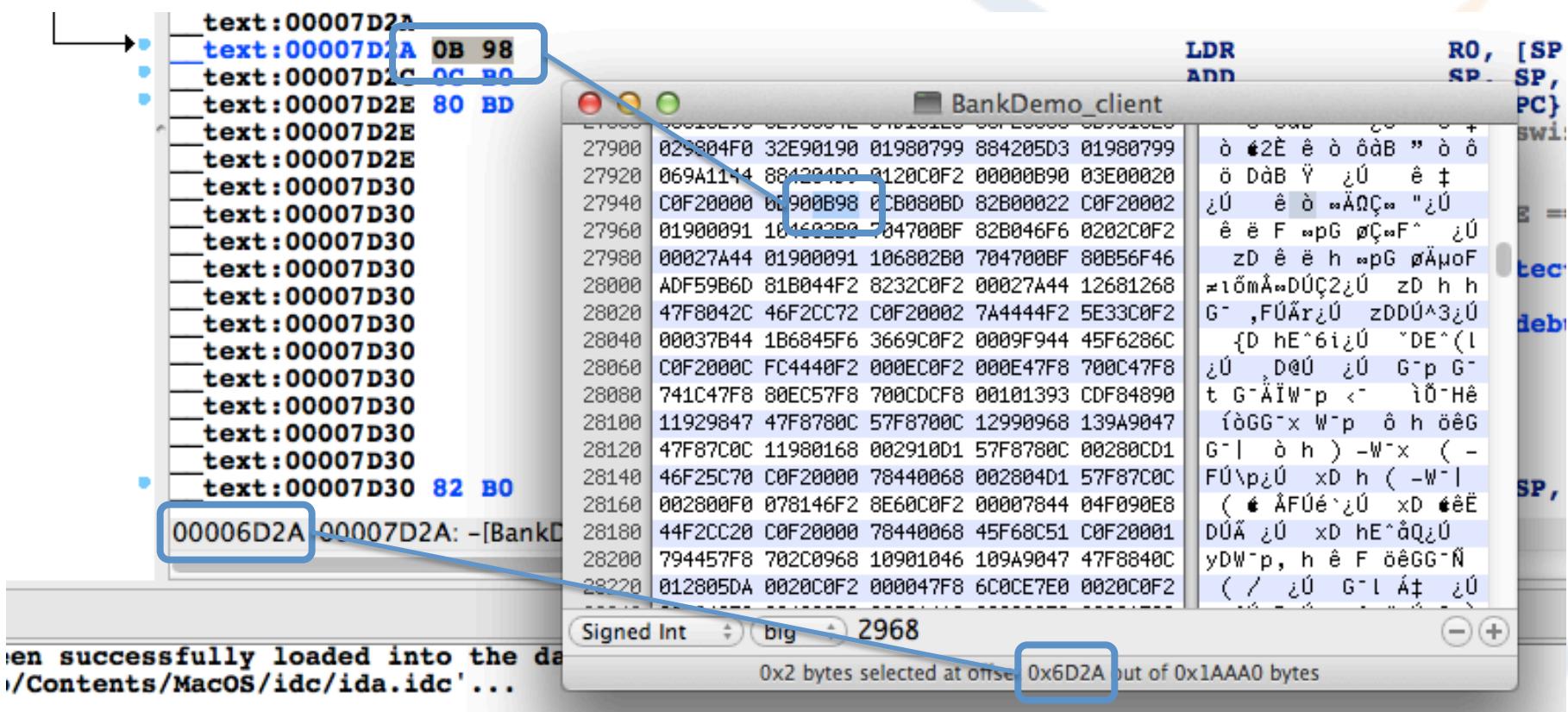


- Function wrapup + epilogue



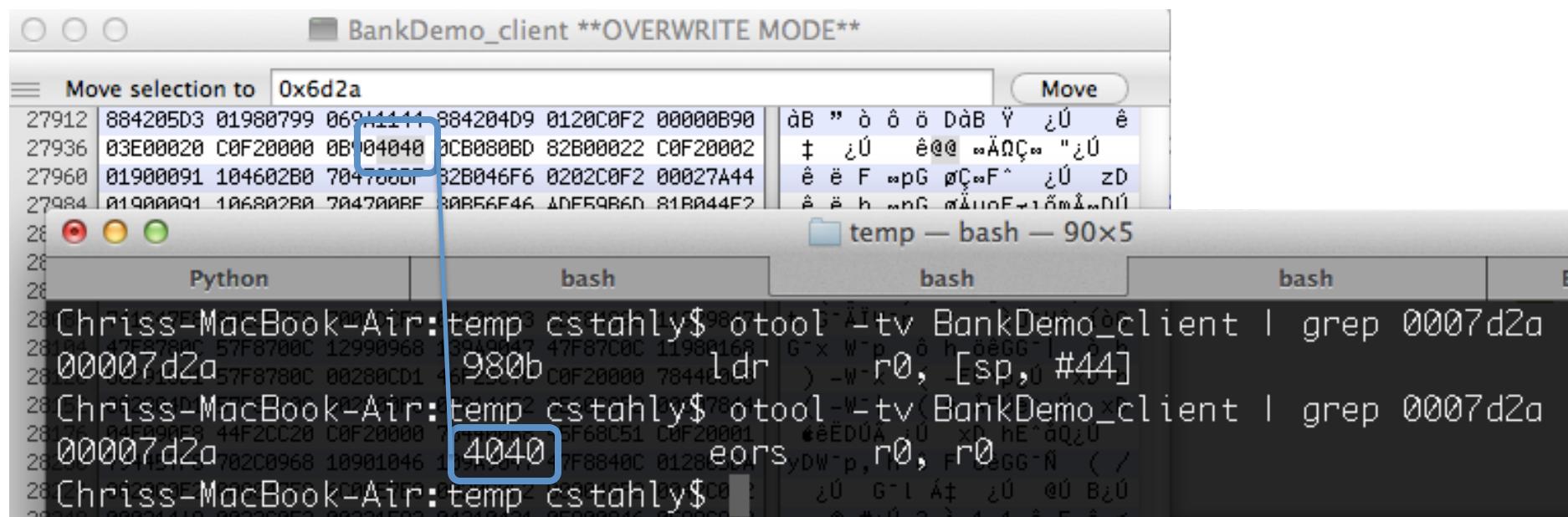
Patching the App

- Open app in Hex Fiend



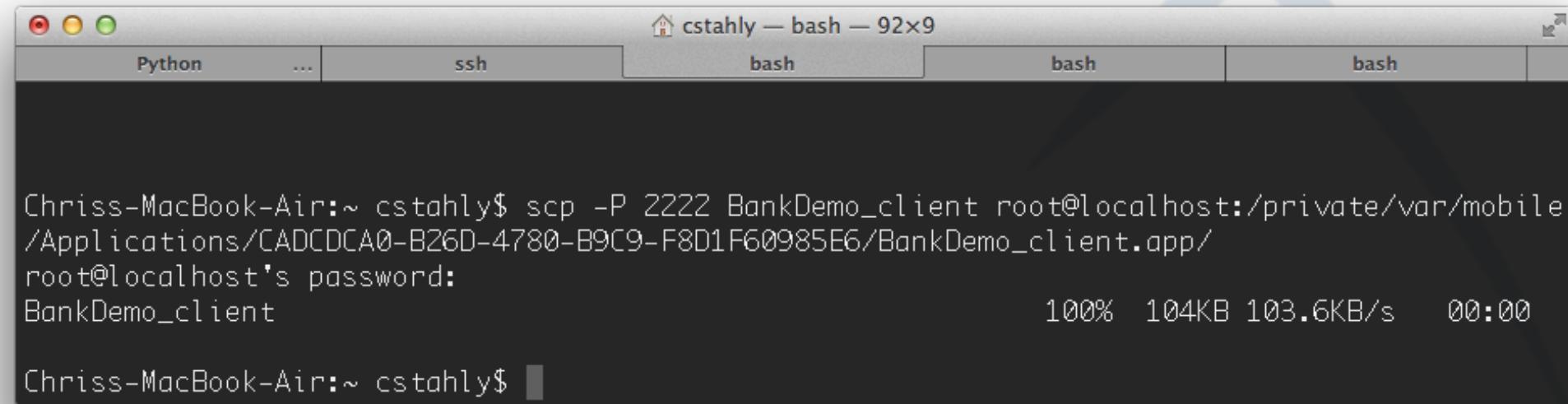
Patching the App (cont'd)

- Patch two bytes
 - 0x4040
 - Turn on Overwrite mode!
 - Edit->Overwrite Mode
 - “otool” will quickly show changes



Deploying the Modified App

- Copy back to the iPod
 - scp -P 2222 BankDemo_client root@localhost:[path to installed IPA]/

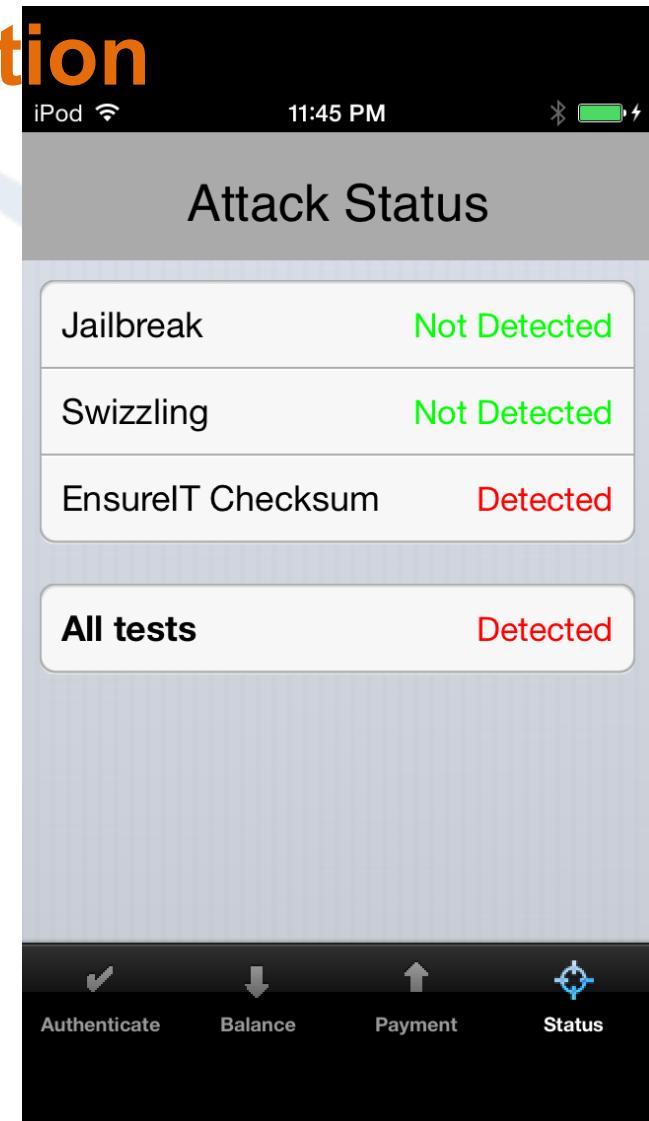


```
Chriss-MacBook-Air:~ cstahly$ scp -P 2222 BankDemo_client root@localhost:/private/var/mobile/Applications/CADCDCA0-B26D-4780-B9C9-F8D1F60985E6/BankDemo_client.app/
root@localhost's password:
BankDemo_client                                         100% 104KB 103.6KB/s  00:00

Chriss-MacBook-Air:~ cstahly$
```

Removing Swizzling Detection

- Kill app and restart
 - Results?

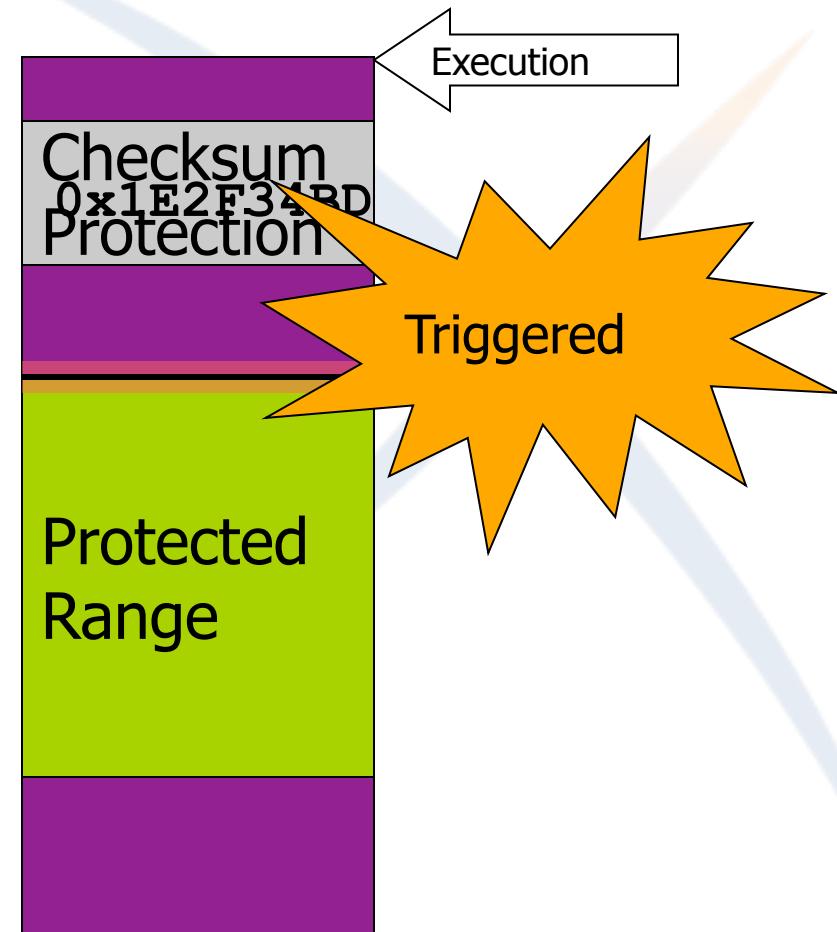


Detecting Code Modification

- Checksum
 - Hash areas of .text section at runtime

Checksum

Checksum ~~0x1E2F34BD~~ EA



Attacks and Defenses (what we covered)

- Jailbreaking
 - Jailbreak Detection
- MobileSubstrate
 - Swizzling Detection
- Application Patching
 - Checksumming

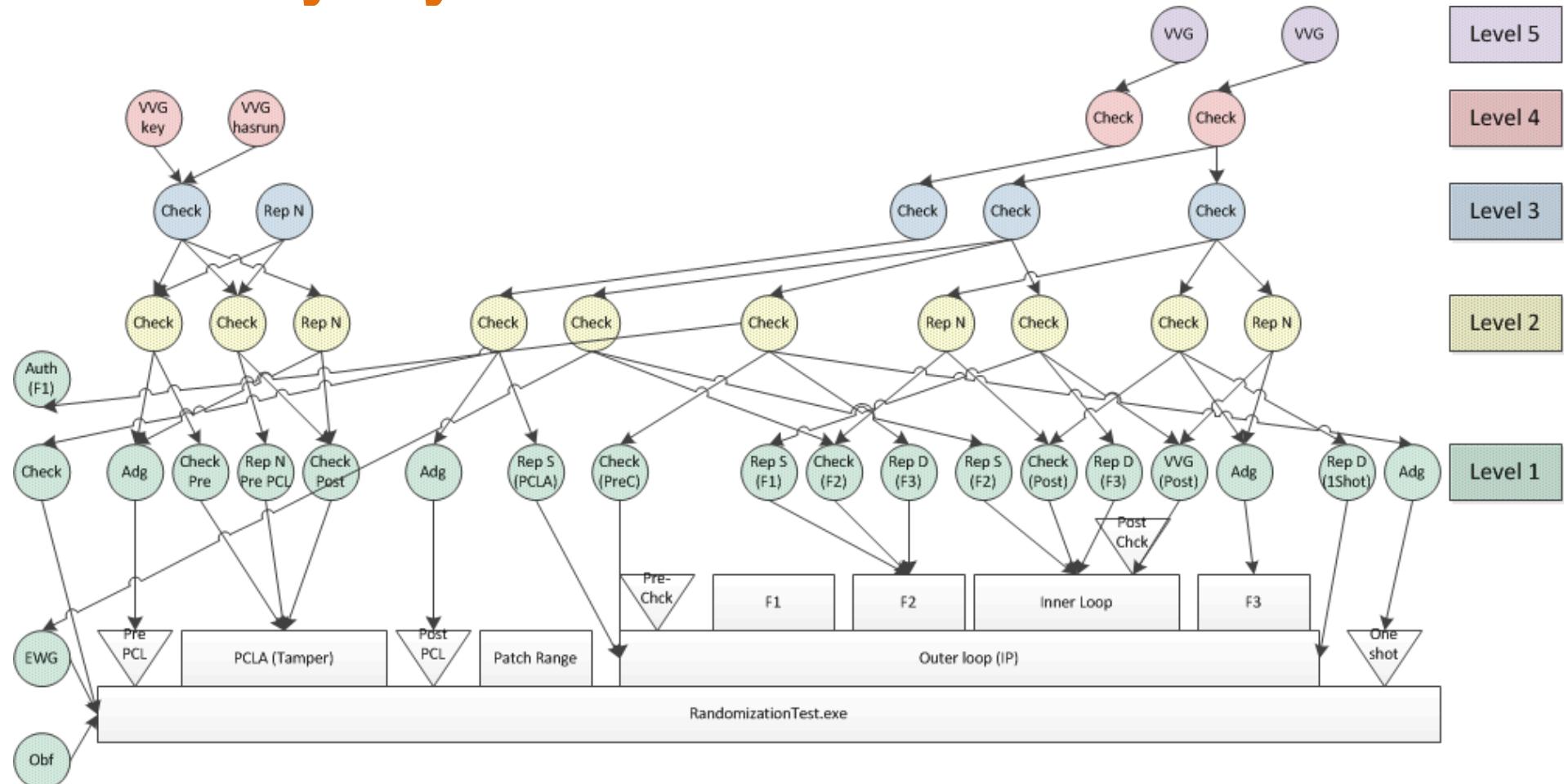
Attacks and Defenses (what we didn't cover)

- Dynamic Analysis with gdb
 - Antidebugging capability
- Static and dynamic analysis with IDA
 - Obfuscation capability
- IPA modification/redeployment
 - Resource verification (on-disk checksumming)

EnsureIT

- Provides these controls
 - Inline invocation
 - Active response
 - Networking ability
- Many other configurable features

Security Layers



Thanks!

Questions?