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
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
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?

Pangu Jailbreak for iOS 7.1 ~ 7.1.x v1.2.0

iPod5,1 with iOS 7.1.2 (11D257)

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.

Developed by @PanguTeam
Official site: http://pangu.io
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 nosuid

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


- **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

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

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

- ObjC-style *messaging*

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

![Compiler diagram](image)
MobileSubstrate

• Definition
  o Set of APIs that allow hooking of native or Obj-C functions
    o In-App or System functions
  o Installed during jailbreak

• Objective-C
  o MSHookMessage
    ▪ Modifies message lookup table

• C/C++
  o MSHookFunction
    ▪ Overwrites bytes to jump to custom code location
Mobile Substrate, con't

• Interfaces
  • Cycrypt
    • 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
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
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
Decrypted The App

- Run Clutch on the phone, specifying “AlienBlue” app
- App is decrypted into /User/Documents/Cracked
Decrypting The App (con’t)

- Unzip the IPA (with “unzip” command)
- Run otool on the app again
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`
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”
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
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]:
Instantiating iphone/tweak in removejb/...
Done.

Chriss-MacBook-Air:theos_proj csthaly$
Using MobileSubstrate

- Existing project
  - ~/theos_proj/removejb
- "cat Tweak.xm"
  - class-dump prototype

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.

```plaintext
/%hook BankDemo_Client

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

%/end
```

ChriSS-MacBook-Air:removejb csthaly$
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)

```c
015  v11 = self;
016  v10 = a2;
017  v2 = _dyld_get_image_header(0);
018  v8 = getsectdatafromheader(v2, "__TEXT", "_text", &size);
019  v9 = _dyld_get_image_vmaddr_slide(0);
020  v8 += v9;
021  v6 = objc_getClass("BankDemo_Client");
022  if ( v6 )
023      {
024          v3 = class_getInstanceMethod(v6, "jailbreakDetect");
025          if ( v3 )
026              {
027                  v5 = method_getImplementation(v3);
028                  v12 = v5 < (unsigned int)v8 || v5 > (unsigned int)&v8[size];
029              }
030              else
031                  {
032                      v12 = 1;
033                  }
034          }
035      else
036          {
037              v12 = 1;
038          }
039      return v12;
```
Patching the App

• Swizzle detection method control flow

• Function wrapup + epiologue
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]/
Removing Swizzling Detection

- Kill app and restart
- Results?
Detecting Code Modification

• Checksum
  – Hash areas of .text section at runtime
Checksum:

Checksum 0x1E2F34BD
Checksum Protection 0x7f3400EA
Triggered
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?