iOS Application Security

Trail of Bits

Est. 2012 to advance computer security

- Software Development
- Security Assessment
- Security Engineering
- Reverse Engineering
- Mobile Security
- Directed Original Research

Dan Guido
CEO
dan@trailofbits.com
@dguido

Sophia D'Antoine
Security Engineer
sophia@trailofbits.com
quend@freenode

Agenda

- iOS Security Overview
- Mobile Application Attacks
- Protections from Apple

- Jailbreak Development
- Attack Walkthrough
- Compiler-Driven Defenses

iOS Security Model

Application Layer

Transport Layer Security

- NSURLConnection and NSURLSession use ATS
- ATS requires TLS 1.2 and strong certificates
- Apps must opt-out*

It's easy to do network encryption right with iOS9

Data Protection

- Encrypt every file with a unique 256-bit key
- Per-file keys are wrapped with a "class" key
- class = security policy

Nearly all files are strongly encrypted on disk

iOS Security Model

Operating System Layer

Application code signing

- Verify your identity with Apple
- Sign your app
- Every code page in memory is checked

Every 4kb page is traceable back to a human owner

Runtime process security

- Apps run in a sandbox
- Apps are restricted from accessing other apps
- System files and resources are shielded

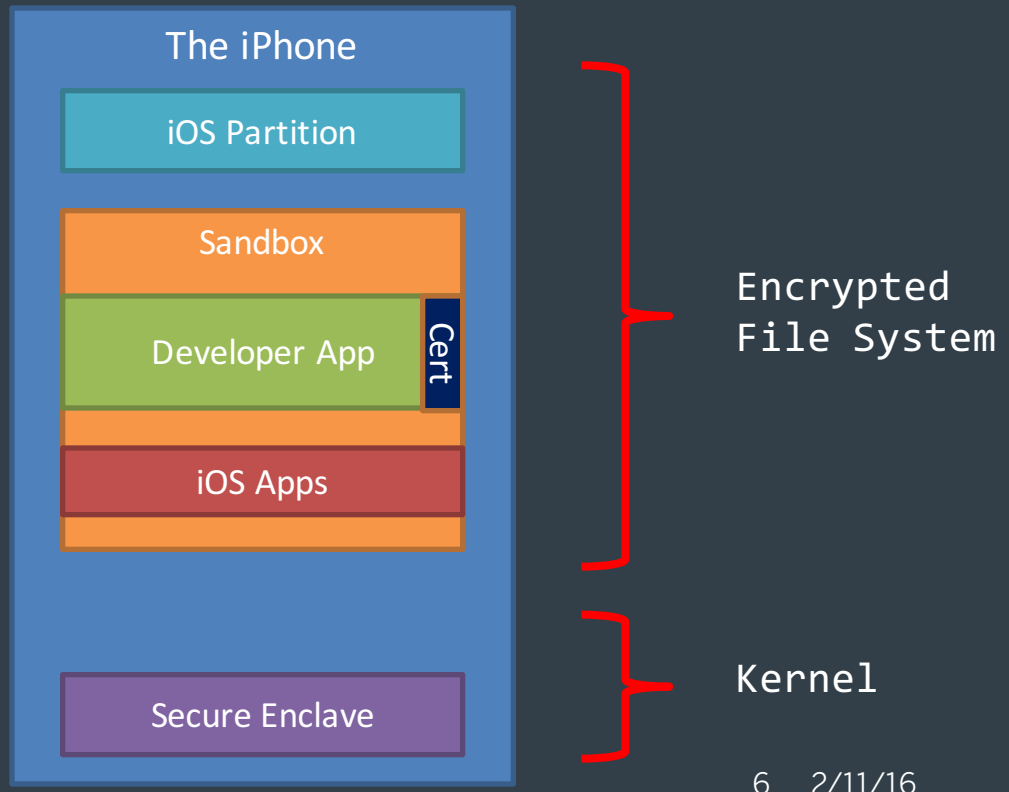
It's hard for even malicious apps to cause trouble

iOS Security Model

Secure Enclave

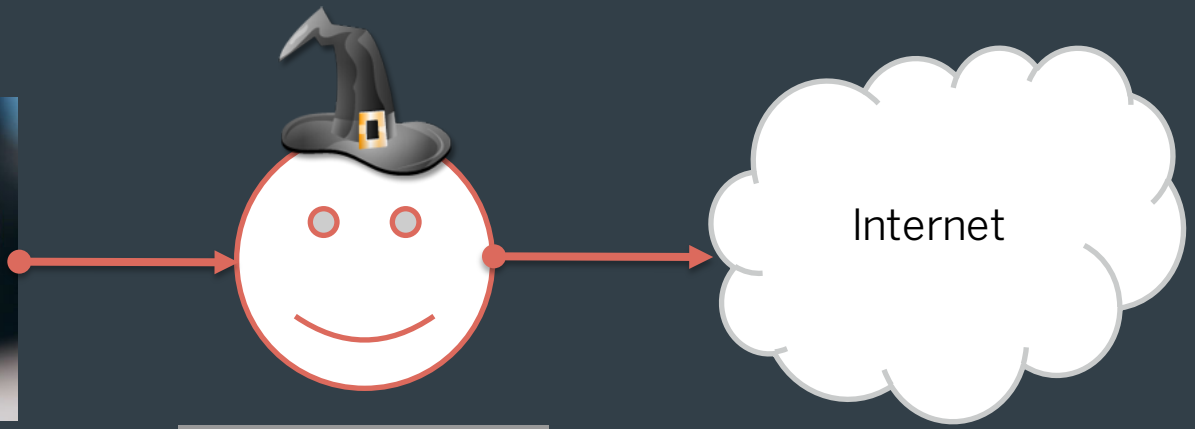
- Unique device key fused at manufacture time
- Keys can never be read out of the Secure Enclave
- Black box for encrypt and decrypt operations

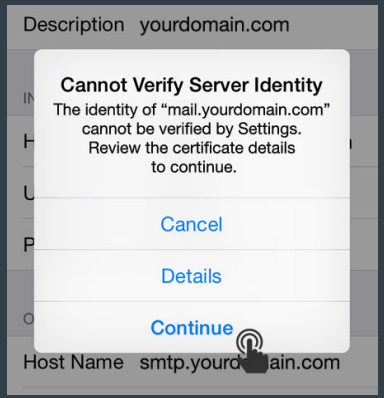
This protects Apple Pay and Data Protection (Keychain)



iOS Application Security

Wireless Interception





Data Leakage

Internal app data (e.g. session IDs) is easily lost in backups

- iTunes backups default to no encryption
- New backup made every time you plug in
- Windows and OS X commonly get malware

- See "BackStab" malware

Data Leakage

It's up to developers to use PIN-derived data protection

- Forensic acquisition software is available for iOS
  - Ex. Elcomsoft iOS Forensic Toolkit and NowSecure Forensics
  - All tout "3rd party app support" – acquire data from Snapchat, Outlook…
  - No passcode required (but even those are easy to brute force)
- This is data that's potentially exposed when you lose your phone
  - If you're making a business of stealing iPhones, $1k is worth paying

Malicious Applications

You may leave data in places where malicious apps can find it

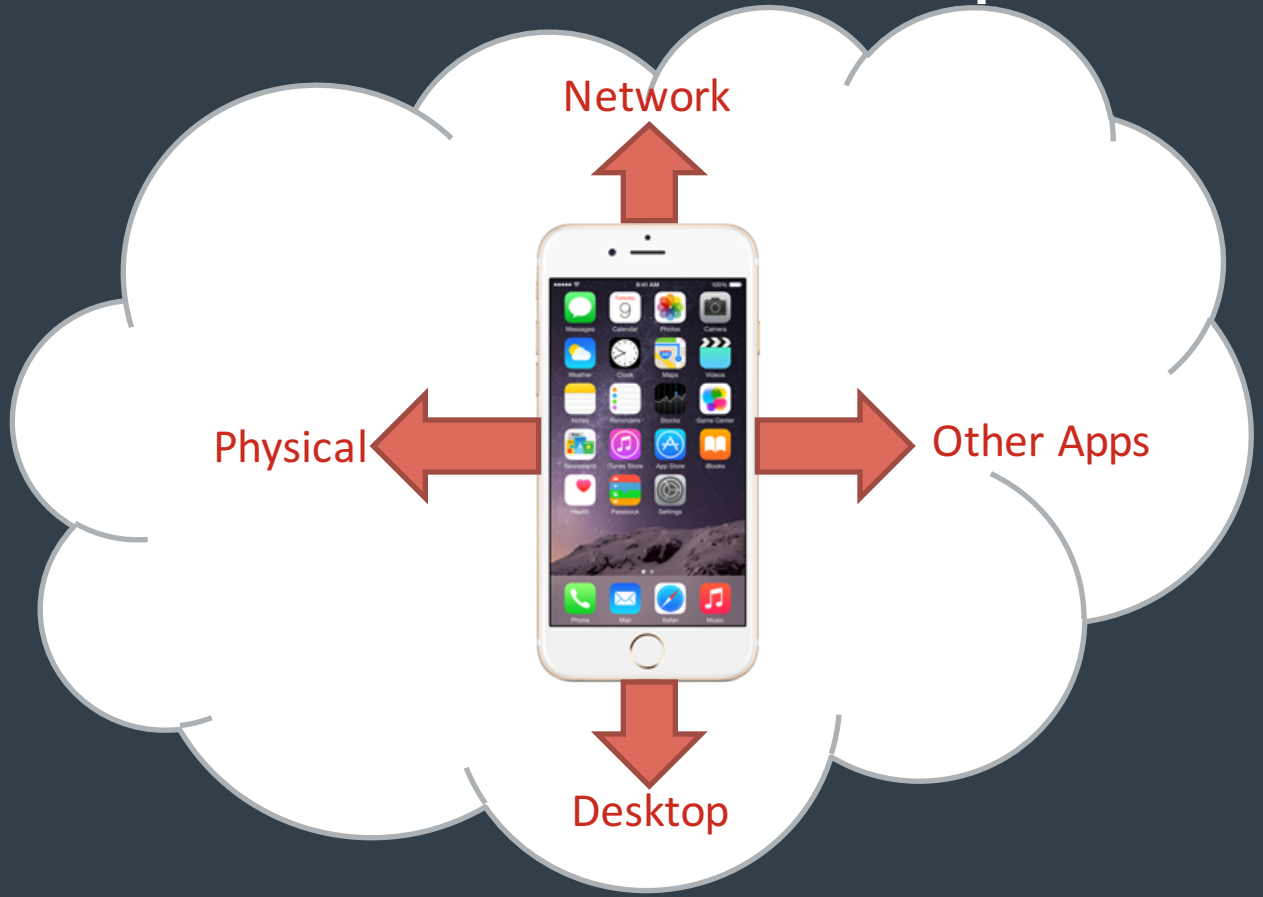
UIPasteboard Scenario

- Frequently used for sensitive data transfer
- 1Password, one-time tokens, visited URLs…
- Malicious apps can hook copy-paste to archive

Example Exposed Data

- URL cache
- Keyboard cache
- App backgrounding
- Logging
- HTML5 data storage
- Cookie objects
- Analytics

Attacks Recap



TLS & Certificate Pinning

Use TLS exclusively and pin your server cert inside your app

- Read the Apple docs for secure networking and adopt ATS
  - E.g., when using NSURLRequest, make sure to specify https in the URL
  - App Transport Security (ATS) fully enabled prevents nearly all failures

- Use TrustKit for effortless and universal certificate pinning
  - No code modifications, it swizzles NSURLConnection and Session
  - Certificate validation failures are reported to a configurable location
  - Can be easily deployed using CocoaPods or by hand

Data Protection & Keychain

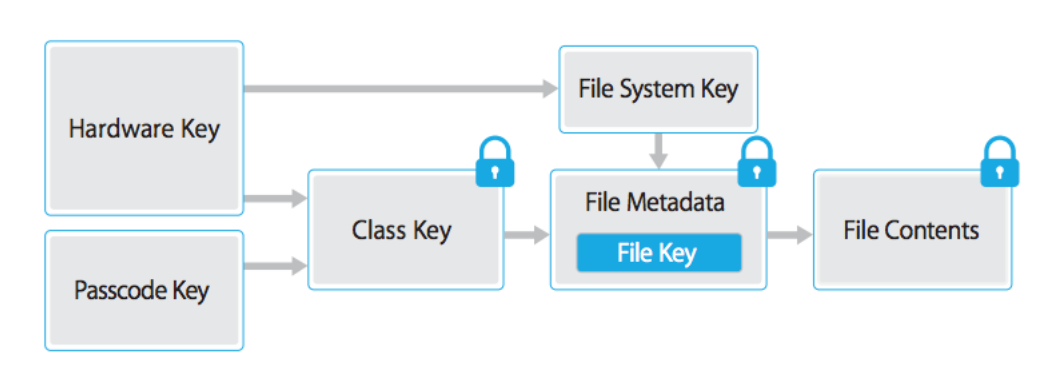
Encrypt ALL the things!

- DPAPI encrypts app data with the phone passcode + UID key*
  - Attribute added to NSData or NSFileManager
  - None, Complete, CompleteUnlessOpen, CompleteUntilFirstAuth
- Keychain Services are used to store passwords and tokens
  - SecItemAdd, SecItemUpdate, SecItemCopyMatching
  - Attributes determine encryption method:
    - Always, AfterFirstUnlock, WhenUnlocked, WhenPasscodeSet
- Don't use: Preferences, Cookies, files in /Library or /Documents

Data Protection API

Why Apple and the police can't read your phone contacts

- Passcode "tangled" with Hardware Key = must crack *on-device*
- Pre-iOS 7: Mail.app the only default app to use DPAPI
- iOS 8: Most Apple apps default to CompleteUntilFirstAuth
- iOS 9: 6-digit passcode required + exponential backoff



Data Minimization

iOS apps leave data in a lot of unexpected places

- Prevent sync to iCloud or iTunes
  - NSURLIsExcludedFromBackupKey/kCFURLIsExcludedFromBackupKey
- Clear background screenshots
  - Check applicationDidEnterBackground and set fields 'hidden' = YES
- Avoid using NSLog for sensitive or proprietary information
  - Use a dummy pre-processor macro `#define NSLog(...)`
- Keep sensitive data out of the Keyboard cache
  - secureTextEntry (password-style entry)
  - UITextAutocorrectionTypeNo (disable autocorrect)

App Security Recap

It's like BuzzFeed, for Mobile App Security

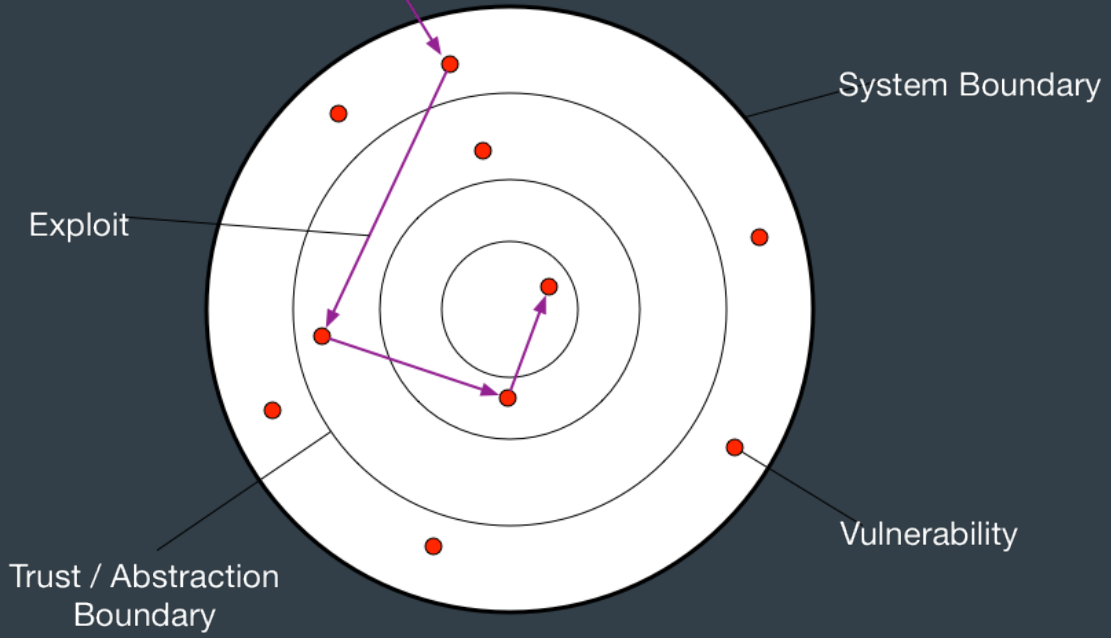
- Follow these 3 rules for every iOS app you make:
  1. *Use HTTPS exclusively*
  2. Store all files, passwords, and tokens with DPAPI or Keychain
  3. Clean up after your app and don't leave sensitive data lying around

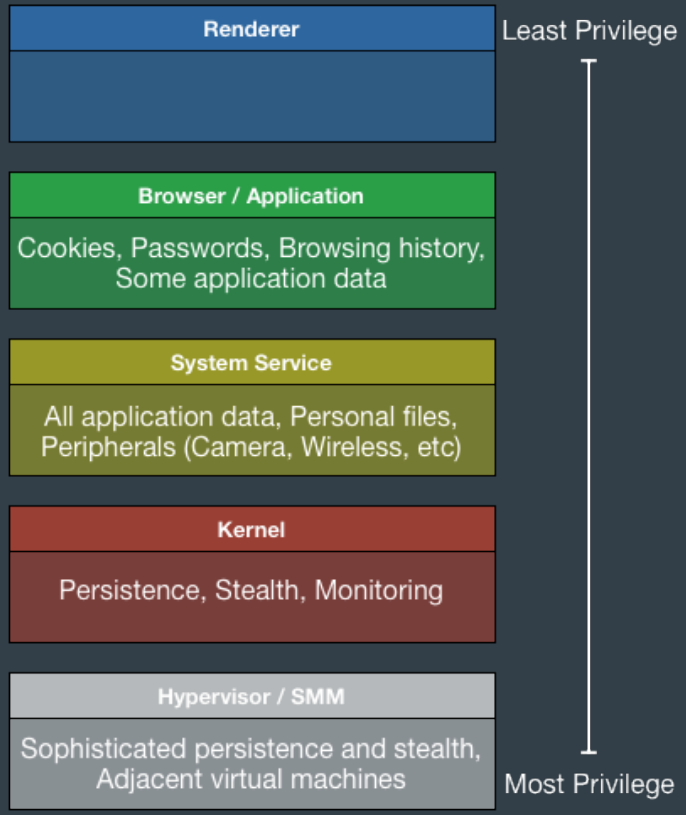
- These are just the basics. Level two includes:
  - Custom URL Handlers, XSS in UIWebViews, Format Strings,
  - Directory Traversal, Null bytes, XML parsing, SQL injection, and more!

Jailbreaks

What are Jailbreaks?

Exploit code that disables security mechanisms





Why Jailbreak?

- Jailbreak maliciously through browser or app
- If you want to:
  1. Obtain application code
  2. Reverse an API Spec
  3. Steal copyrighted content
  4. Pirate paid apps
  5. Commit fraud

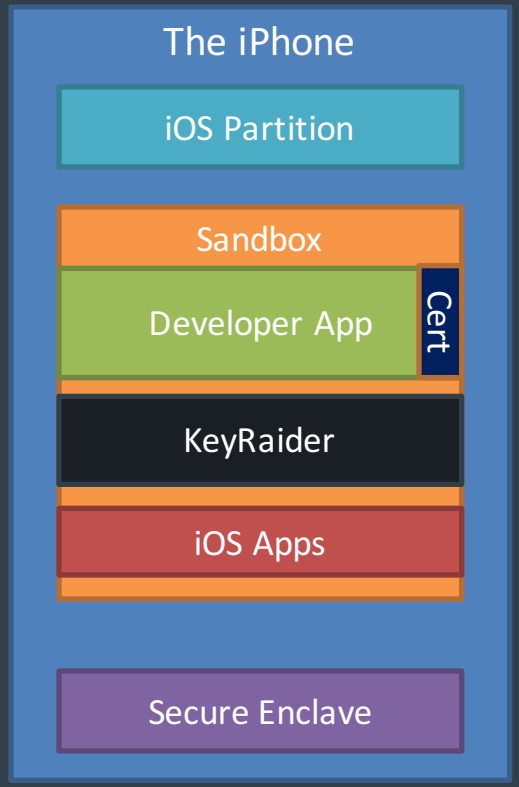
- Jailbreak voluntarily by downloading a kit
- If you want to:
  1. Access 3rd party appstores
  2. Tether for free
  3. Replace default apps
  4. Customize look and feel
  5. Device unlocking

The end result is the same. All application protections are dead.

KeyRaider

This is your phone. This is your phone on Jailbreaks.

- Application data is strongly protected by sandboxing and iOS security model
- Without these protections, a malicious app can bypass ALL device security
- Keyraider stole Apple ID creds from 225,000 devices for in-app purchases



Jailbreak Takeaways

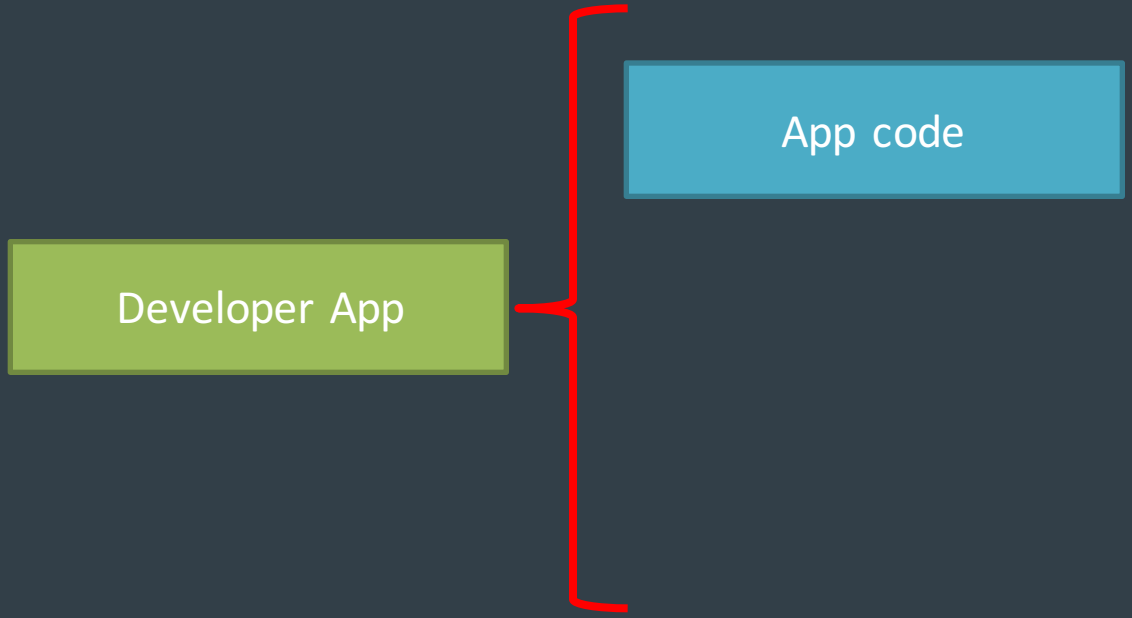
You cannot always depend on Apple's APIs to protect you

1. Users will 'attack' themselves
   - Up to 7 million voluntarily jailbreak
2. Existing Jailbreaks used for attacks
   - Delivered via browser or user-generated app content
3. New Jailbreaks not always public
   - Active community of people, knowledge, tools for hacking iOS

Jailbreak Protections

Injected Code

Application Code + Security Code



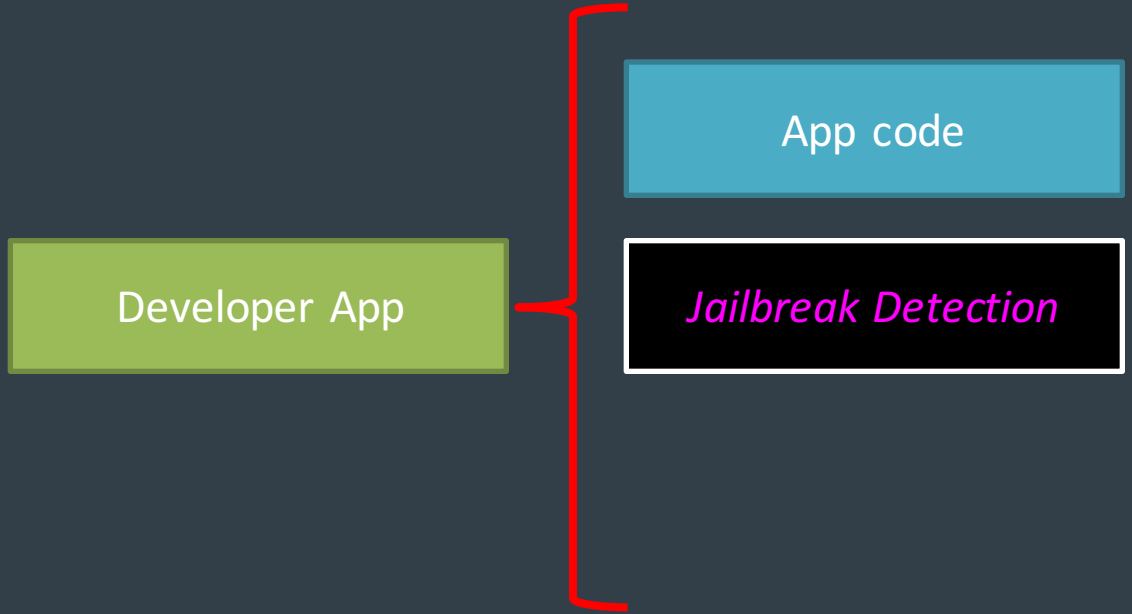
Jailbreak Checks

Detect an unsafe or tampered environment

- Detect system artifacts left over from successful jailbreaks
  - Relies on complete understanding of iOS internals and the jailbreak
- Many apps implement na ve checks
  - Check for MobileSubstrate.dylib, ssh, Cydia.app files
  - Check if the fork() system call is available
  - More basic ideas at project-imas/security-check

Injected Code

Application Code + Security Code



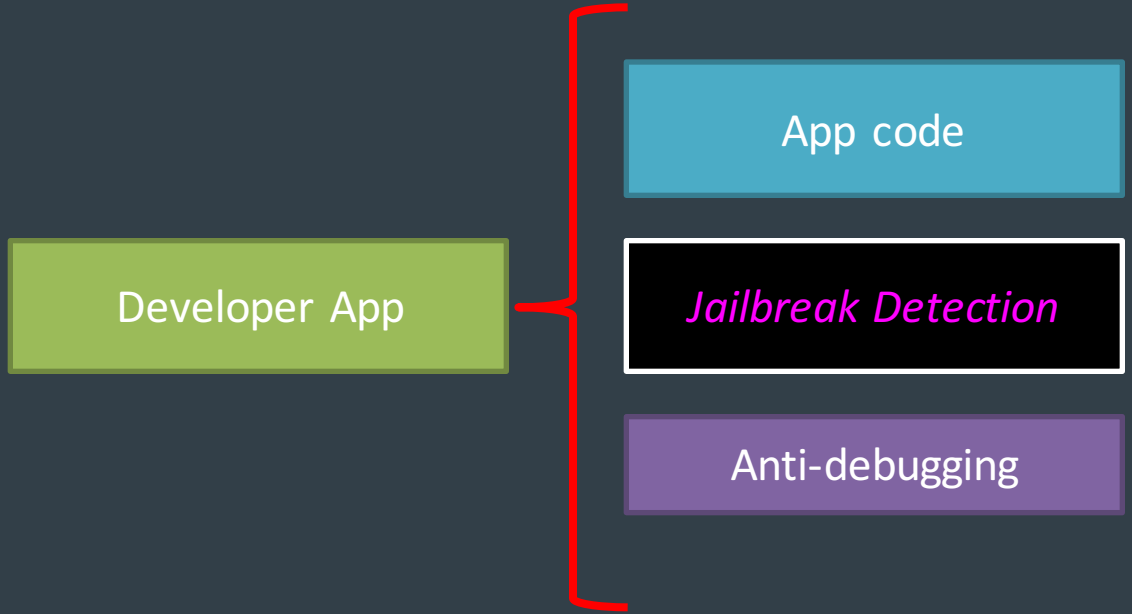
Anti-Debugging

Deny attempts to debug or hook the application

- Attackers can bypass Jailbreak checks with a debugger
  - Ex. Snapchat checks are bypassed by hooking filesystem calls
  - Download tsProtector 8 or xCon for point-and-click bypass!
- Dynamic, anti-debugging must accompany jailbreak checks
  - Ex. Use sysctl to ask who your parent is. If it's not launchd or the kernel, you're being debugged! Either exit or alter execution.

Injected Code

Application Code + Security Code



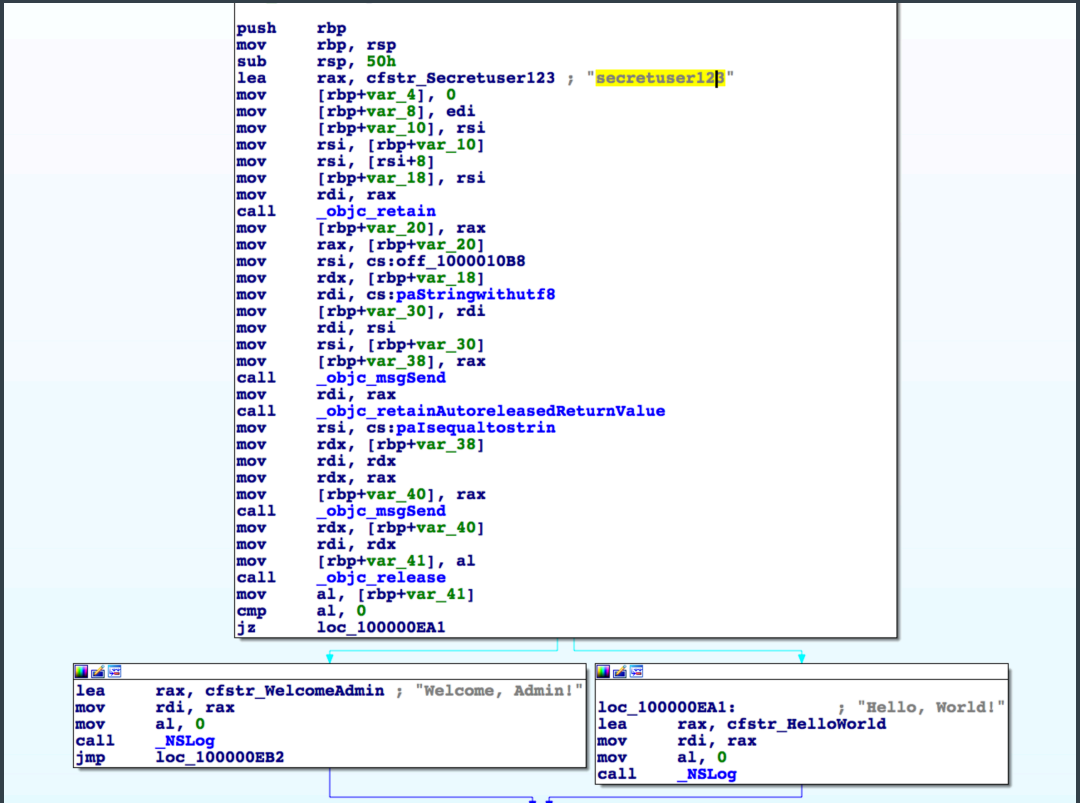
Anti-Reversing

Deny the ability to understand the recovered code

- Attacker could disassemble the app and patch out security checks
  - Simple to do with IDA Pro, Hopper, or Binary Ninja
- Static protections can destroy the utility of this approach
  - 100x more code to look through, inability to search through it
  - Symbol encryption, false predicates, and code diffusion…

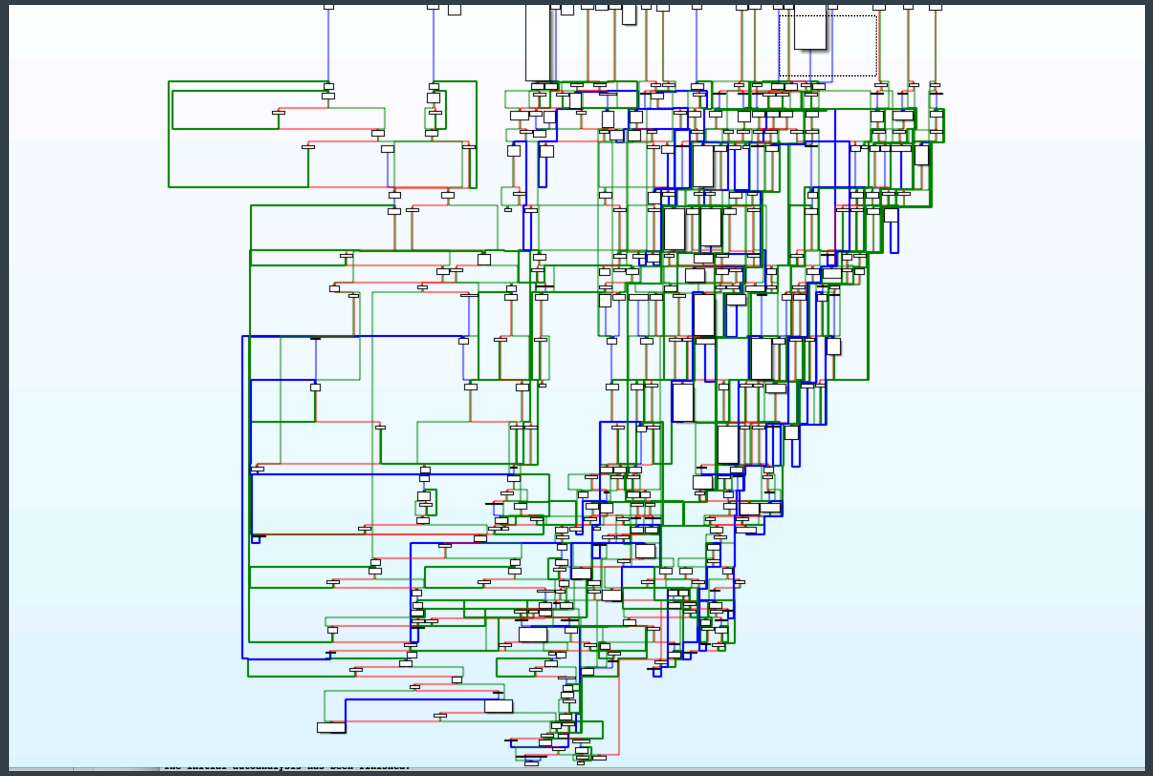
Anti-Reversing

Original application as viewed in IDA Pro



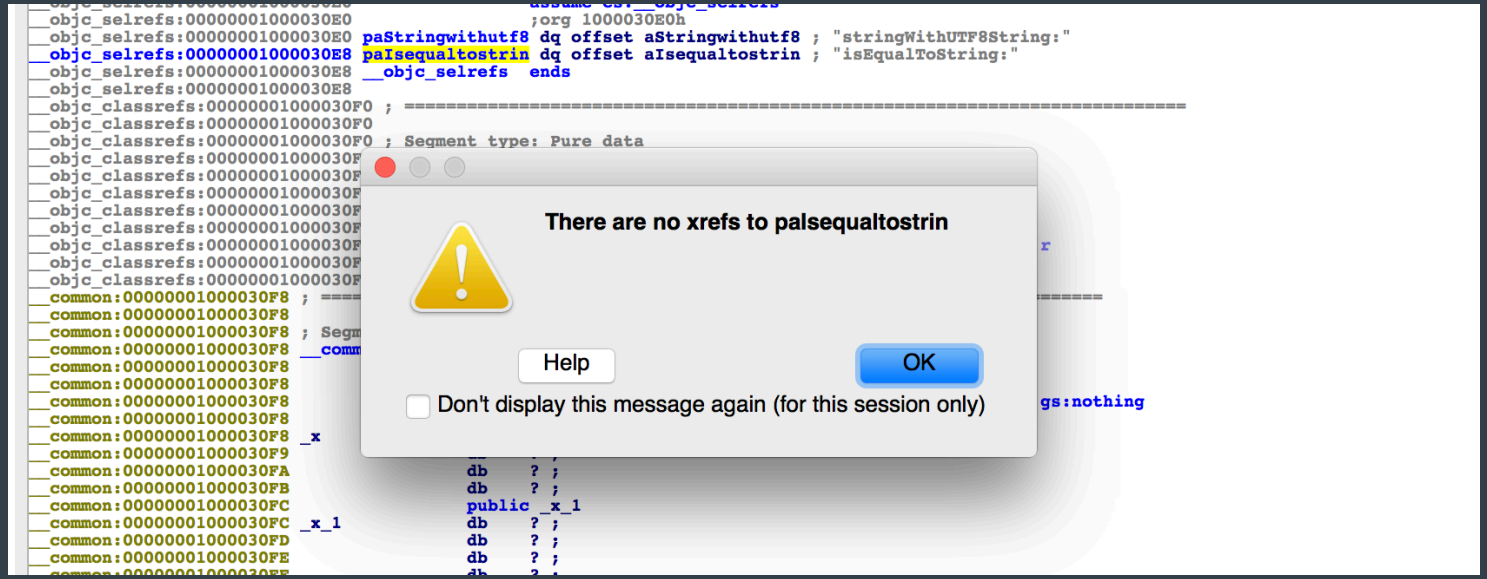
Anti-Reversing

Obfuscated applications have more code to reverse



Anti-Reversing

Symbol encryption makes binary code navigation impossible

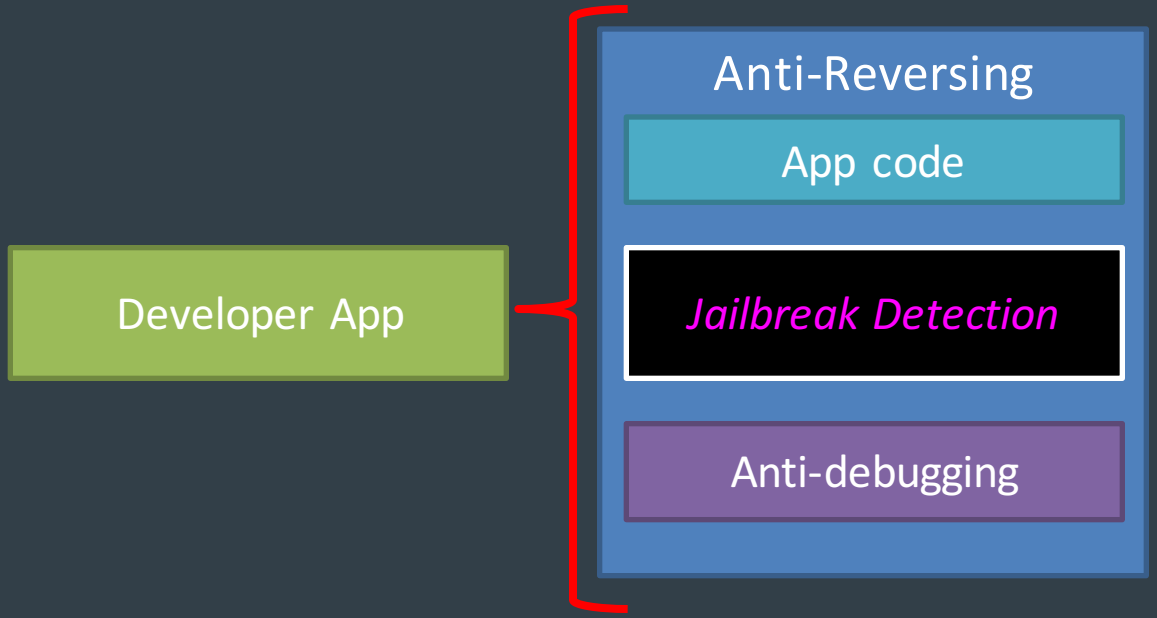


Imagine a consulting gig...

- The software development project from hell:
  - Sorry, you have to use *only* this Windows XP box for dev
  - Sorry, our devs were drunk and named all the vars a, aa, aaa
  - Sorry, we have 10mil LOC but only 10k are used. We don't know which.
  - Sorry, our code won't run in a debugger, it will crash
  - Sorry, ctags won't run on our code
  - ...
- Good luck adding new features to our app! You're going to do great!

Injected Code

Application Code + Security Code



Integration

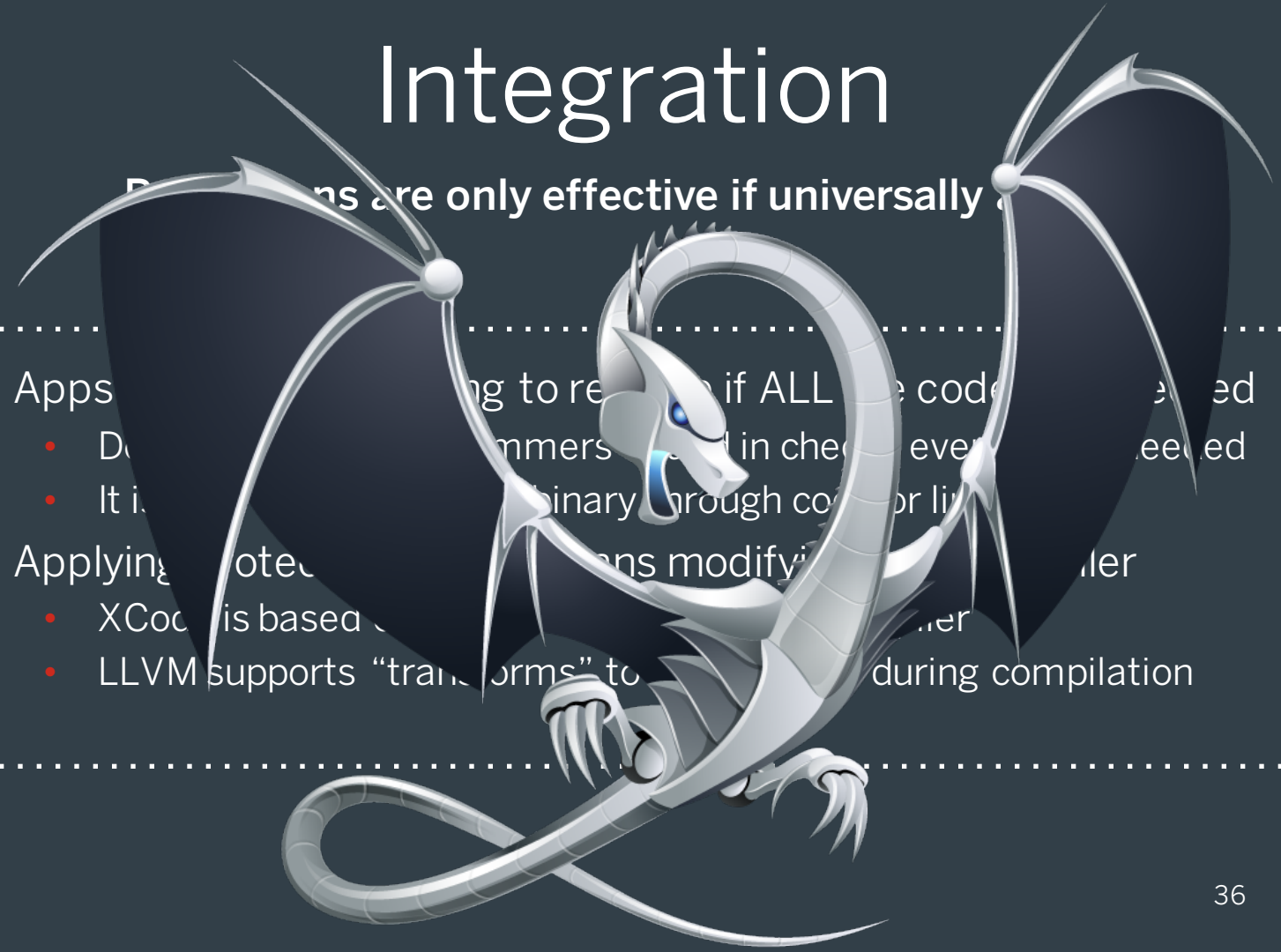
Protections are only effective if universally applied

- Apps are only challenging to reverse if ALL the code is protected
  - Don't leave it to programmers to add in checks everywhere needed
  - It is hard to influence the binary through code or linking
- Applying protections right means modifying your compiler
  - XCode is based on LLVM, an open-source compiler
  - LLVM supports "transforms" to modify code during compilation

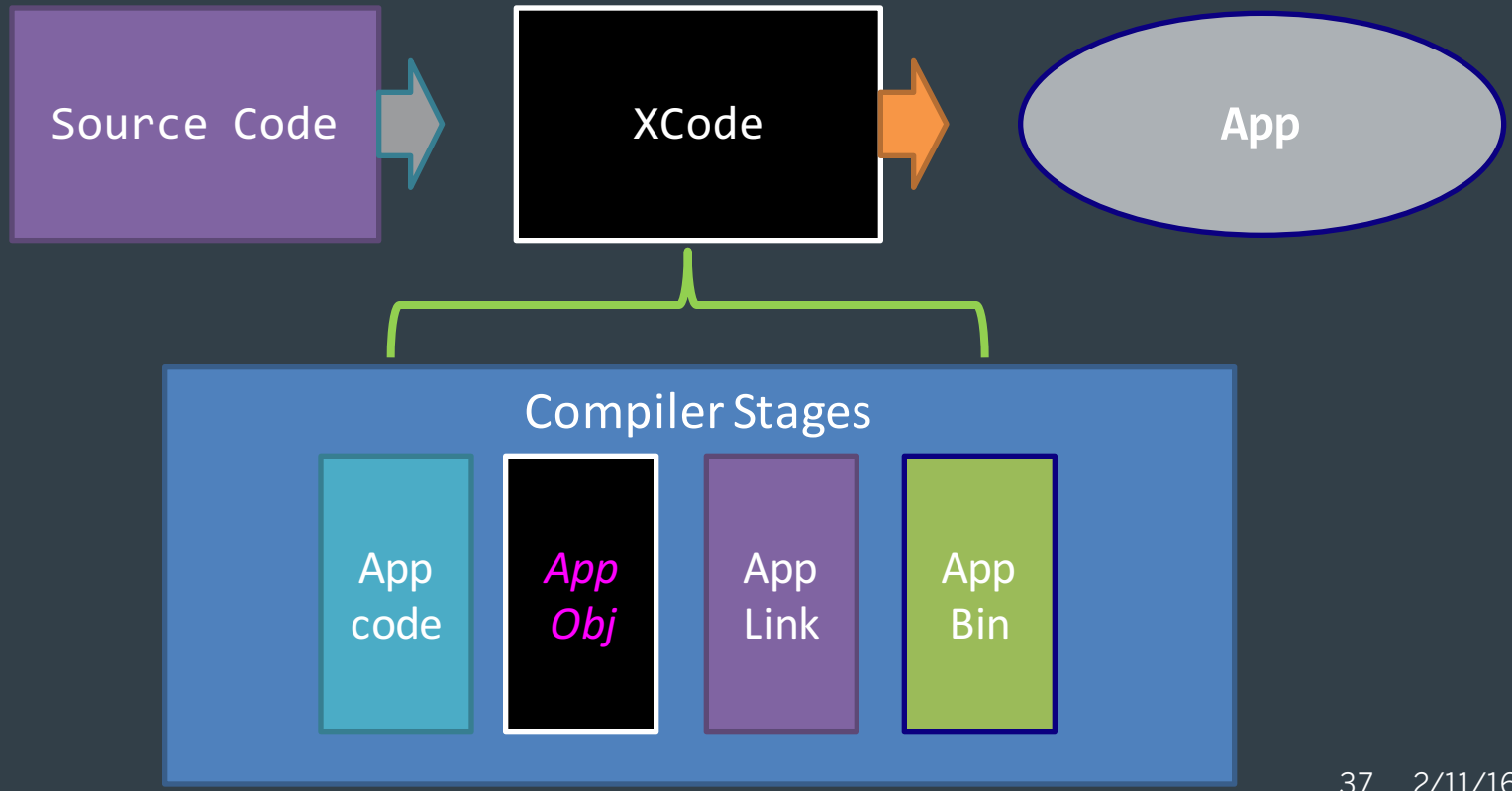
Integration

P ns are only effective if universally

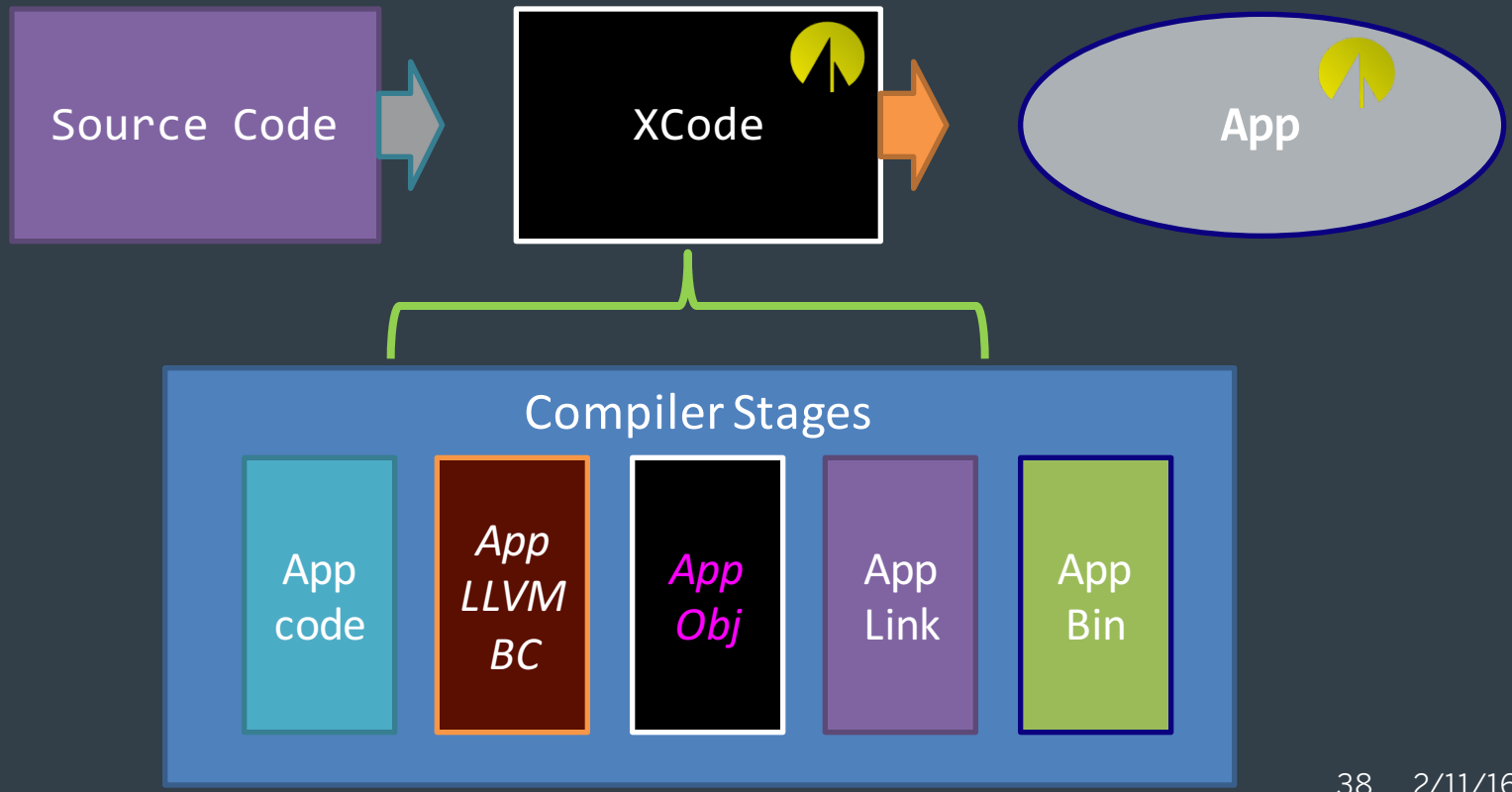
- Apps ng to re if ALL code ed
  - D mmers in che eve eeded
  - It i inary rough co or li
- Applying otec ns modifyi ler
  - XCod is based ler
  - LLVM supports "tra orms" to during compilation



Compiler Walkthrough



The End Result



Doing this yourself

LLVM is orders of magnitude easier to use than GCC

- LLVM Overview: www.aosabook.org/en/llvm.html
- Writing an LLVM Pass: llvm.org/docs/WritingAnLLVMPass.html
- LLVM Tutorial: cs.umd.edu/~awruef/LLVM_Tutorial.pdf

- Our interview process begins with an LLVM pass work-sample test

Getting it done for you

Contact us for more details (mast@trailofbits.com)

Conclusions

Protect Your App

Use all the tools that Apple gives you and then some!

- It's easier to abuse applications without basic protections
  - Use HTTPS *exclusively*: Apple Transport Security (ATS) and TrustKit
  - Use Data Protection or Keychain Services on all private data
  - Inventory the sensitive data in your app and eliminate it when possible
- However, Jailbreaks bypass all app protections
  - Embed protections that determine device integrity inside your app

Questions?

Thanks for having us!

- If you like this kind of work, come to Empire Hacking
- If you have high security needs, talk with us about MAST
- If auth is more your thing, www.passwordlessapps.com

Dan Guido
CEO
dan@trailofbits.com
@dguido

Sophia D'Antoine
Security Engineer
sophia@trailofbits.com
quend@freenode

References

More Information on iOS Application Security

Protecting Your App

- iOS Security Guide
- Security Changes in iOS9
- Protecting Against Screen Caching
- Writing Secure iOS Applications

Modern Threats to Your App

- Stealing an App's Custom API's
- Current Example of iOS Malware
- Simple Jailbreak Detection Bypass
- Mobile Threat Analysis: 2015

EnPublic Apps

How malware abuses Apple Private APIs

- Run class-dump on frameworks in iOS SDK --> private APIs
- Apps distributed w/ ad-hoc provisioning can use private APIs
- EnPublic attacks documented in paper at Asia CCS 2015

Method	Framework	Usage	Available on iOS 6.X	Available on iOS 7.X	Available on iOS 8.0
[[UIDevice currentDevice] U–niqueIdentifier]	UIKit	Get the UDID of the device.	Yes	No	No
CTSIMSupportCopyMobile–SubscriberIdentity()	coreTelephony	Get the IMSI of the device.	Yes	No	No
CTSettingCopyMyPhoneN–umber()	coreTelephony	Get the telephone number of the device.	Yes	No	No
CTTelephonyCenterAddOb–server()	coreTelephony	Register call back of SMS mes–sages and incoming phone calls.	Yes	Yes	Yes
CTCallCopyAddress()	coreTelephony	Get the telephone number of the phone call.	Yes	Yes	Yes
CTCallDisconnect()	coreTelephony	Hang up the phone call.	Yes	No	No
[[CTMessageCenter shar–edMessageCenter] incom–ingMessageWithId: result]	coreTelephony	Get the text of the incoming SMS message.	Yes	Yes	Yes

Bitcode App Submission

An opportunity for security improvements?

- Swift XCode projects automatically include LLVM BC
  - Bitcode is embedded into your app during archive builds (see 'ENABLE_BITCODE')
  - Apple can perform optimizations and transformations in the cloud for iWatch power usage or app thinning
- Bitcode submission likely to become standard
- Note: Compiling to bitcode does not introduce any new security issues into your application code.

Why not Android?

Anti-jailbreak code is nearly impossible to write for Android

- Let's ignore…
  - That < 5% run the latest version of Android…
  - HW fragmentation issues and lack of 'Secure Enclave'-alike…
  - The, usually insecure, OEM customizations…
  - The lack of whole-system code signing…
- Cyanogen Mod is an officially supported Android distribution!
  - 'su' is a valid, accessible binary for Android distributions
  - Millions of people run Cyanogen Mod and expect it to work
- That said, anti-reversing/debugging code works on Android

How about overhead?

Armoring gets in an attacker's way, not regular users

- Anti-jailbreak and anti-debug checks have very little overhead
  - They are single instructions that run between other code
  - Even so, you can add more or less of them via compiler options
- Anti-reversing adds lots of code, but hardly any is ever run
  - Opaque predicates insert dead code that never gets executed
  - Symbol encryption is not computationally difficult
- End of the day, expect ~5-10% CPU/memory increase
  - On the other hand, binary size may inflate with all the dead code