scademy
secure coding academy

iOS security

CL-IOS | Classroom | 2 days

Audience: iOS application developers, architects and testers
Preparedness: General iOS application development
Exercises: Hands-on

iOS is a mobile operating system distributed exclusively for Apple hardware and designed with security at its core; key security features including sandboxing, native language exploit mitigations or hardware supported encryption all offer a very effective environment for secure software development. The devil is however in the details – a programmer can still commit plenty of mistakes to make the resulting apps vulnerable. This course introduces the iOS security model and the usage of various components, but also deals with relevant vulnerabilities and attacks, focusing on the mitigation techniques and best practices for avoiding them.

Recommended for programmers developing apps who want to understand the security features of iOS as well as the typical mistakes one can commit on this platform.

Outline:

- IT security and secure coding
- iOS security overview
- iOS application security
- Common coding errors and vulnerabilities
- Buffer overflow protection on iOS
- Testing iOS code
- Principles of security and secure coding
- Knowledge sources

Participants attending this course will:

- Understand basic concepts of security, IT security and secure coding
- Learn the security solutions on iPhone
- Learn to use various security features of iOS
- Learn about typical coding mistakes and how to avoid them
- Get information about some recent vulnerabilities of iOS
- Get practical knowledge in using security testing tools for iOS
- Get sources and further readings on secure coding practices

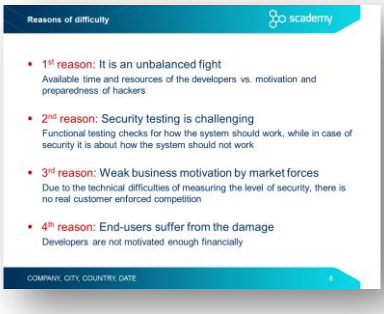
Related courses:

- CL-AND - Android security (Classroom, 3 days)
- CL-AAN - Android Java and native code security (Classroom, 4 days)
- CL-AIS - Android and iOS security (Classroom, 4 days)

Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of certain security problems. All exercises are prepared in a plug-and-play manner by using a pre-set desktop virtual machine, which provides a uniform development environment.

Detailed table of contents

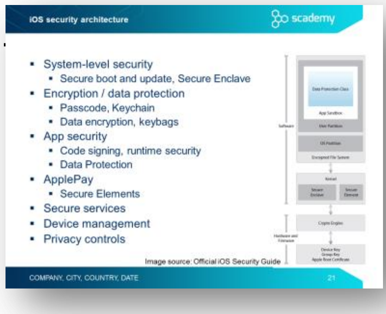
Day 1



Reasons of difficulty

- 1st reason: It is an unbalanced fight
  Available time and resources of the developers vs. motivation and preparedness of hackers
- 2nd reason: Security testing is challenging
  Functional testing checks for how the system should work, while in case of security it is about how the system should not work
- 3rd reason: Weak business motivation by market forces
  Due to the technical difficulties of measuring the level of security, there is no real customer enforced competition
- 4th reason: End-users suffer from the damage
  Developers are not motivated enough financially

COMPANY, CITY, COUNTRY, DATE 8



iOS security architecture

- System-level security
  - Secure boot and update, Secure Enclave
- Encryption / data protection
  - Passcode, Keychain
  - Data encryption, keybags
- App security
  - Code signing, runtime security
  - Data Protection
- ApplePay
  - Secure Elements
- Secure services
- Device management
- Privacy controls

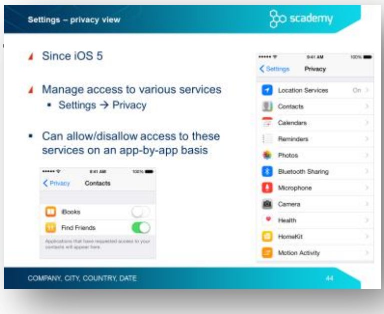
Image source: Official iOS Security Guide

COMPANY, CITY, COUNTRY, DATE 21

- Deploying applications
    - App verification
    - Developer registration
    - Code signing
    - Apple's review process
    - Illicit 'app store optimization'

iOS application security

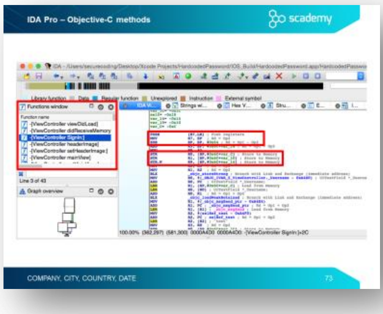
- iOS permissions
    - Using permissions
    - Settings – privacy view
    - iOS permissions – compared to Android
    - Entitlements vs permissions
    - Permissions – best practices
- Writing secure iOS applications
    - Privilege Separation
    - Local data storage
    - Storing local data in the Keychain
    - Exercise: managing local data
    - Keychain sharing vulnerability due to Apple provisioning bug
    - Developing secure networked applications
    - Local file encryption
- Protecting applications
    - FairPlay DRM for iOS apps
    - Third-party DRM pitfalls
    - Obfuscation and encryption
    - Securing inter-app communication – URL schemes
- Reverse engineering and debugging
    - Reverse engineering tools
    - Decrypting iOS Applications
    - Application Class dump
    - Reverse engineering with IDA Pro
    - IDA Pro basics
    - IDA Pro tips
    - IDA Pro string view
    - IDA Pro – Objective-C methods
    - IDA Pro – Objective-C method call



Settings – privacy view

- Since iOS 5
- Manage access to various services
    - Settings → Privacy
- Can allow/disallow access to these services on an app-by-app basis

COMPANY, CITY, COUNTRY, DATE 44



IDA Pro – Objective-C methods

Functions window
Function name
-[ViewController viewDidLoad]
-[ViewController didReceiveMemory
-[ViewController SignIn:]
-[ViewController headerImage]
-[ViewController setHeaderImage:]
-[ViewController mainView]
Line 3 of 43
Graph overview
Strings wi...
Hex V...
Library function
Data
Regular function
Unexplored
Instruction
External symbol

100.00% (382,297) (581,300) 0000A4D0 0000A4D0: -[ViewController SignIn:]+2C

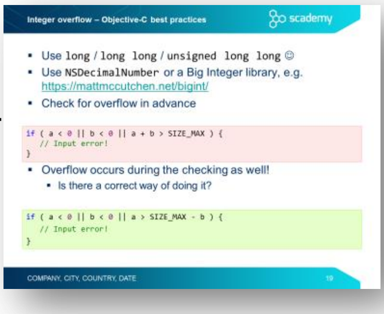
COMPANY, CITY, COUNTRY, DATE 73

- Hard-coded passwords
- Interacting with Objective-C
- Read process memory with Cycript
- Runtime analysis with GDB
- Some useful GDB commands
- Dumping Objective-C method calls with GDB
- Purpose of jailbreaking
- Types of jailbreaking
- Consequences of jailbreaking
- Jailbreak detection with file checking
- Jailbreak detection with sandbox checking
- Jailbreak detection with Cydia URL scheme
- Removing Jailbreak detection with Cycript
- Hook library calls with Cycript
- Jailbreak detection consequences
- Debug prevention with ptrace
- Debug prevention with assembly code
- Debugger detection
- Further debugger detection techniques
- Anti-debugging best practices

Day 2

Common coding errors and vulnerabilities

- Input validation
    - Input validation concepts
    - Injection flaws
        - JSON Injection
    - UIWebView vulnerabilities
        - Dangers of UIWebView
        - A real-world vulnerability: Skype for iOS (2011)
    - XML vulnerabilities
        - XML External Entity (XXE) injection
    - Integer problems in Objective-C
        - Integer overflow
        - Integer overflow – Objective-C best practices ....



Integer overflow – Objective-C best practices

- Use `long` / `long long` / `unsigned long long` ☺
- Use `NSDecimalNumber` or a Big Integer library, e.g. https://mattmccutchen.net/bigint/
- Check for overflow in advance

```
if ( a < 0 || b < 0 || a + b > SIZE_MAX ) {
   // Input error!
}
```

- Overflow occurs during the checking as well!
    - Is there a correct way of doing it?

```
if ( a < 0 || b < 0 || a > SIZE_MAX - b ) {
   // Input error!
}
```

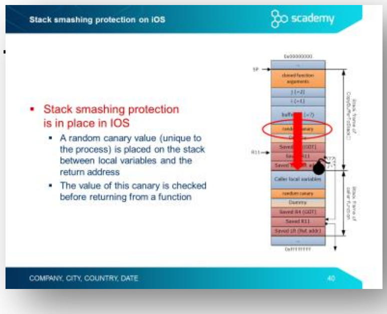
COMPANY, CITY, COUNTRY, DATE 19

- Other input validation problems
  - Path manipulation / path traversal
  - Unsafe reflection – spot the bug!
  - Format string vulnerability

Buffer overflow protection on iOS

- ARM architecture
  - ARM Processors – main registers
  - The function calling mechanism in ARM
- Buffer overflow
  - Buffer overflow on the stack
  - Exercises – trying to exploit a buffer overflow
  - Fortify_source check
  - Buffer overflow on the stack 2

- Protection techniques and their circumvention
  - Stack smashing protection on iOS..
  - Effects of stack smashing protection
  - Bypassing stack smashing protection
  - CVE-2011-1823 in vold's method – Spot the bug!
  - Exercise – vold vulnerability
  - Exercise – vold vulnerability exploit analysis
  - WWW exploit with .got overwrite
  - WWW exploit
  - Address Space Layout Randomization (ASLR)
  - Randomization with ASLR
  - Data Execution Prevention (DEP) / Execute-Never (XN) bit
  - Bypassing ASLR, XN and stack protection
  - Information leakage
  - Information leakage with use after free
  - Virtual method call
  - Code execution with use after free
  - Return oriented programming (ROP)
    - Creating ROP chain
    - Exploit using ROP
  - Signing and integrity protection
  - Access control
    - Storing sensitive data on iOS

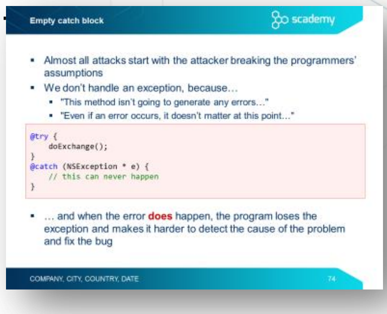


Stack smashing protection on iOS

- Stack smashing protection is in place in IOS
  - A random canary value (unique to the process) is placed on the stack between local variables and the return address
  - The value of this canary is checked before returning from a function

COMPANY, CITY, COUNTRY, DATE 40

- Improper error and exception handling
  - Typical problems with error and exception handling
  - Errors vs exceptions
  - Empty catch block ....
  - Overly broad catch
  - Null pointers in Objective C
  - Exercise ErrorHandling – spot the bug!
  - Exercise – Error handling
  - iOS and OS X HTTPS key verification


Empty catch block

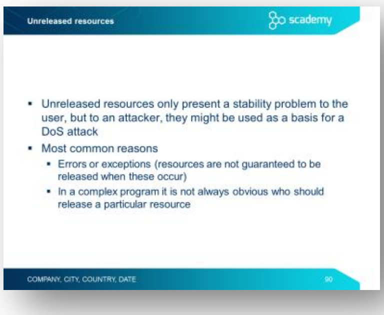
- Almost all attacks start with the attacker breaking the programmers' assumptions
- We don't handle an exception, because…
  - "This method isn't going to generate any errors…"
  - "Even if an error occurs, it doesn't matter at this point…"

```
@try {
    doExchange();
}
@catch (NSException * e) {
    // this can never happen
}
```

- … and when the error **does** happen, the program loses the exception and makes it harder to detect the cause of the problem and fix the bug

COMPANY, CITY, COUNTRY, DATE 74

- Time and state problems
  - Time and state related problems
  - Serialization errors (TOCTTOU)
  - Race condition in signal handling
- Code quality problems
  - Dangers arising from poor code quality
  - Poor code quality – spot the bug!
  - Unreleased resources..
  - Type mismatch – Spot the bug!
  - Regular expression DOS (ReDOS)


Unreleased resources

- Unreleased resources only present a stability problem to the user, but to an attacker, they might be used as a basis for a DoS attack
- Most common reasons
  - Errors or exceptions (resources are not guaranteed to be released when these occur)
  - In a complex program it is not always obvious who should release a particular resource

COMPANY, CITY, COUNTRY, DATE 90

- Cryptography
  - Cryptography on iOS
- Digital Rights Management (DRM)
  - DRM architecture
  - iOS DRM overview
  - Challenges of DRM protection
  - DRM protection without hardware support - hardening
  - DRM protection – decrypted content
- iOS-specific vulnerabilities and bugs

Testing iOS code

- General testing approaches
- Testing iOS code
  - Exercise – Clang
  - OCLint
  - Exercise – OCLint
  - CppCheck
  - Exercise – CppCheck

Principles of security and secure coding

- Matt Bishop’s principles of robust programming
- The security principles of Saltzer and Schroeder

Knowledge sources

- Secure coding sources – a starter kit
- Vulnerability databases
- iOS secure coding sources @ Apple Developer
- Recommended books – C/C++
- Recommended books – iOS