

Android and iOS security

CL-AIS | Classroom | 4 days

Audience: Android and iOS application developers, architects and testers

Preparedness: General Android and iOS application development

Exercises: Hands-on

As a developer, your duty is to write bulletproof code. However...

What if we told you that despite all of your efforts, the code you have been writing your entire career is full of weaknesses you never knew existed? What if, as you are reading this, hackers were trying to break into your code? How likely would they be to succeed?

This advanced course will change the way you look at code. A hands-on training during which we will teach you all of the attackers' tricks and how to mitigate them, leaving you with no other feeling than the desire to know more.

It is your choice to be ahead of the pack, and be seen as a game changer in the fight against cybercrime.

Outline:

- IT security and secure coding
- Android security overview
- Android application security
- Protecting Android applications
- iOS security overview
- iOS application security
- Common coding errors and vulnerabilities
- Practical cryptography
- Android native code security
- Security testing techniques and tools
- 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 Android
- Learn to use various security features of the Android platform
- Learn the security solutions on iPhone
- Learn to use various security features of iOS
- Learn about typical coding mistakes and how to avoid them
- Have a practical understanding of cryptography
- Get understanding on native code vulnerabilities on Android
- Realize the severe consequences of unsecure buffer handling in native code
- Understand the architectural protection techniques and their weaknesses
- Get practical knowledge in using security testing tools for iOS
- Get practical knowledge in using security testing tools for Android
- 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-IOS - iOS security (Classroom, 2 days)
- CL-CPS - C and C++ secure coding (Classroom, 3 days)

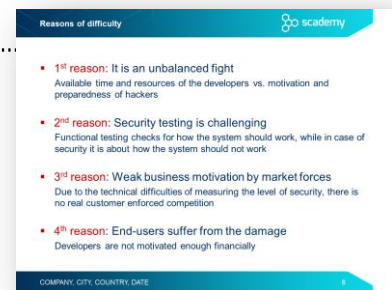
Note: Our classroom trainings come with a number of easy-to-understand exercises providing live hacking fun. By accomplishing these exercises with the lead of the trainer, participants can analyze vulnerable code snippets and commit attacks against them in order to fully understand the root causes of 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

IT security and secure coding

- Nature of security
- What is risk?
- IT security vs. secure coding
- From vulnerabilities to botnets and cybercrime
 - Nature of security flaws
 - Reasons of difficulty.....
 - From an infected computer to targeted attacks
 - The Seven Pernicious Kingdoms
 - OWASP Top Ten 2017
 - OWASP Mobile Top Ten 2016 (release candidate)



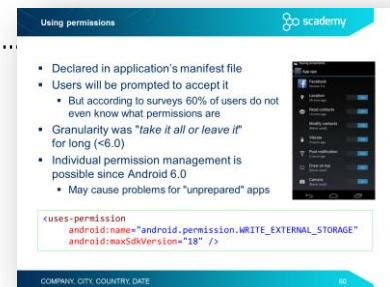
Android security overview

- Android fragmentation challenges
- The Android software stack
- OS security features and exploit mitigation techniques
- The Linux kernel
 - User and process separation
 - Anonymous shared memory (ashmem)
 - ANDROID_PARANOID_NETWORK kernel option
 - SELinux Type Enforcement policies
 - SELinux policies
 - SELinux policy example –
 - Adding custom policy files
 - Exercise: compiling and using SELinux policies
 - SELinux Role-Based Access Control
 - SELinux Multi-Level Security
- Filesystem security
 - Filesystems used for external storage
 - Filesystem encryption
 - Encrypting individual files and external SD cards

- Dalvik
 - Dalvik VM
 - VM Separation
 - Zygote
 - Bytecode verifier
- Android Runtime (ART)
 - ART architecture
 - ART backward compatibility
 - ART security features
 - Ahead-of-time (AOT) compilation
- Deploying applications
 - Application signing
 - No validation of developer identity
 - Google’s review process
 - Installing using Google Play
 - Installing outside of Google Play
 - Verify App

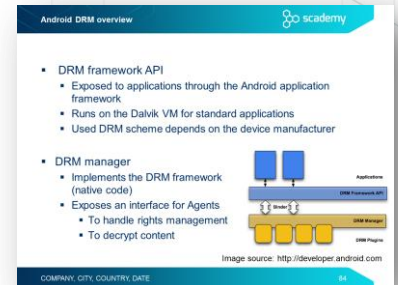
Android application security

- Permissions
 - Using permissions.....
 - Exercise – using permissions
 - Using custom permissions
 - Exercise – using custom permissions
 - Permissions – best practices
- Writing secure Android applications
 - Activity, Fragment and Service – basics
 - Intents
 - Implicit intents
 - Intent hijacking
 - BroadcastReceiver security
 - Activity hijacking
 - Best practices against activity hijacking
 - Sticky broadcasts
 - Content provider
 - Content provider permissions



Protecting Android applications

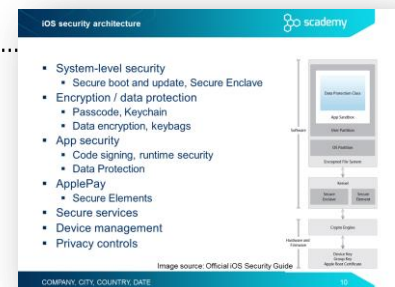
- Digital Rights Management (DRM)
 - DRM architecture
 - Android DRM overview
 - Challenges of DRM protection
 - DRM protection without hardware support - hardening
 - DRM protection – decrypted content
- Reverse engineering and debugging
 - Reverse engineering methods and tools
 - Getting the package name
 - Reverse engineering exercise



Day 2

iOS security overview

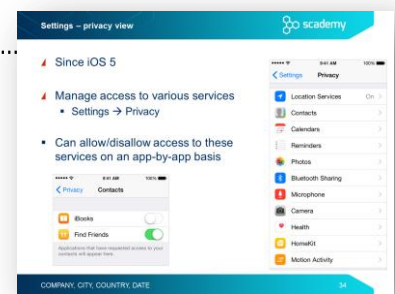
- iOS platform security basics
 - Evolution of iOS security features
 - iOS architecture
 - iOS security architecture
- iOS sandboxing and app interactions
 - Sandbox concepts
 - The iOS application sandbox
 - The iOS sandbox directories
 - Inter-app communication basics
 - Extensions and sandboxing
- Securing data storage
 - Data Protection overview
 - Key generation and storage
 - Disk and file encryption
 - Keybags
 - Data Protection classes
 - Keychain Data Protection classes



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

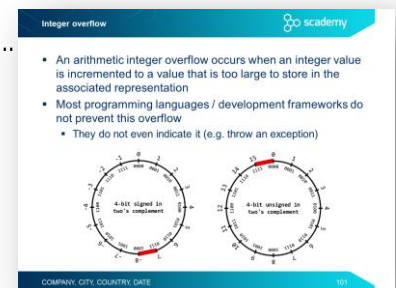
iOS application security

- 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
- 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



Common coding errors and vulnerabilities

- Input validation
 - Input validation concepts
- Injection
 - SQL Injection on Android
 - Typical SQL Injection attack methods
 - SQL Injection protection methods
 - Using parameterized queries in Android
 - JSON Injection
- Cross-site scripting
 - Mobile Web views
 - Dangers of UIWebView
 - Android WebView XSS
 - XSS prevention
 - Android WebView security best practices
 - iOS UIWebView security best practices
 - A real-world vulnerability: Skype for iOS (2011)
- Integer problems
 - Representation of negative integers
 - Integer overflow.....
 - Exercise IntOverflow
 - What is the value of Math.abs(Integer.MIN_VALUE)?
 - Integer problem – best practices
 - Avoiding arithmetic overflow – addition
 - Avoiding arithmetic overflow – multiplication
 - Java case study
 - A real-world integer overflow vulnerability in Java
 - The actual mistake in java.util.zip.CRC32
 - Integer problems in Objective-C
 - Integer overflow – Objective-C best practices
 - Case study – Android Stagefright
 - Stagefright – a quick introduction.....
 - Some Stagefright code examples – spot the bugs!
- Other typical input validation mistakes
 - Unsafe reflection
 - Implementation of a command dispatcher
 - Unsafe reflection – spot the bug!
 - Mitigation of unsafe reflection



- An arithmetic integer overflow occurs when an integer value is incremented to a value that is too large to store in the associated representation
- Most programming languages / development frameworks do not prevent this overflow
 - They do not even indicate it (e.g. throw an exception)

COMPANY, CITY, COUNTRY, DATE

191



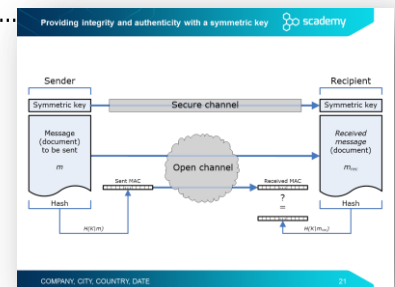
- "The Android Unicorn" revealed in August 2015
 - Affects Android since 2.2 (Froyo), fixed in 5.1 (Lollipop)
- Multiple vulnerabilities in the built-in media player
 - CVE-2015-1538, 1539, 3824, 3826-3829, 3864
 - Divide by zero
 - NULL dereference
 - String termination problems
 - Failing to check result of allocation
 - Integer overflow
 - Heap overflow
- Consequence: **arbitrary code execution!**
 - Remotely exploitable via MMS
 - Does not need any user interaction

COMPANY, CITY, COUNTRY, DATE

Day 3

Practical cryptography

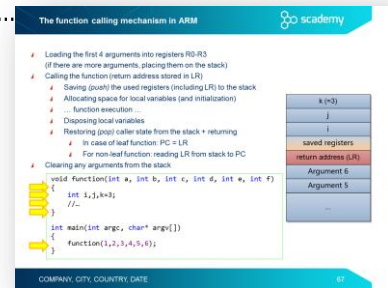
- Rule #1 of implementing cryptography.....
- Cryptosystems
 - Elements of a cryptosystem
- Symmetric-key cryptography
 - Providing confidentiality with symmetric cryptography
 - Symmetric encryption algorithms
 - Modes of operation
- Other cryptographic algorithms
 - Hash or message digest
 - Hash algorithms
 - SHattered
 - Message Authentication Code (MAC)
 - Providing integrity and authenticity with a symmetric key.....
 - Random numbers and cryptography
 - Cryptographically-strong PRNGs
 - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
 - Providing confidentiality with public-key encryption
 - Rule of thumb – possession of private key
 - The RSA algorithm
 - Introduction to RSA algorithm
 - Encrypting with RSA
 - Combining symmetric and asymmetric algorithms
 - Digital signing with RSA
 - Public Key Infrastructure (PKI)
 - Man-in-the-Middle (MitM) attack
 - Digital certificates against MitM attack
 - Certificate Authorities in Public Key Infrastructure
 - X.509 digital certificate



- Cryptography on Android
 - Java Cryptography Architecture / Extension (JCA/JCE)
 - Using Cryptographic Service Providers
 - Engine classes and algorithms
 - Cryptographic Service Providers in Android
 - Exercise Sign – Generating and verifying signatures

Android native code security

- Buffer overflow possibilities in Android
- ARM machine code, memory layout and stack operations
 - ARM Processors – main registers
 - ARM Processors – most important instructions
 - ARM Processors – control instructions
 - ARM Processors – stack handling instructions
 - ARM Processors – condition field
 - Understanding complex ARM instructions
 - The function calling mechanism in ARM.....
 - The local variables and the stack frame
 - Function calls – prologue and epilogue of a function
 - Stack frame of nested calls
 - Stack frame of recursive functions
- Buffer overflow on the stack
 - Classic buffer overflow on the stack
 - Exercises – trying to exploit a buffer overflow
 - Stack smashing protection in Android.....
 - Effects of stack smashing protection
 - Bypassing stack smashing protection
 - Lack of source checking
 - CVE-2011-1823 in vold's method – Spot the bug!
 - Exercise – vold vulnerability
 - Exercise – vold vulnerability exploit analysis
 - WWW exploit with .got overwrite
 - Exercise – overwrite .got with write-what-where
- Protection techniques – ASLR, XN, RELRO, ...
 - Address Space Layout Randomization (ASLR)
 - Randomization with ASLR
 - Access Control on memory segments
 - The Never eXecute (NX) bit
 - Read-only relocation and immediate binding – RELRO



The function calling mechanism in ARM

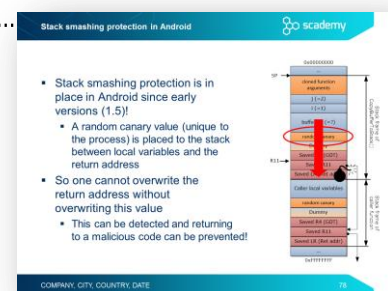
- Loading the first 4 arguments into registers R0-R3 (if there are more arguments, placing them on the stack)
- Calling the function (return address stored in LR)
- Saving (locally) the used registers (including LR) to the stack
- Allocating space for local variables (and initialization)
- ... function execution ...
- Disposing local variables
- Restoring (pop) caller state from the stack + returning
 - In case of leaf function: PC = LR
 - For non-leaf function: reading LR from stack to PC
- Clearing any arguments from the stack

```

void function(int a, int b, int c, int d, int e, int f)
{
    int i, j, k, x;
    //...
}

int main(int argc, char* argv[])
{
    function(1, 2, 3, 4, 5, 6);
}
  
```

Stack diagram showing: k (-3), saved registers, return address (LR), Argument 6, Argument 5.

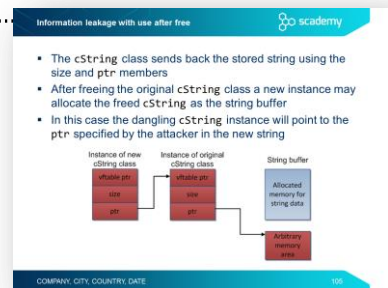


Stack smashing protection in Android

- Stack smashing protection is in place in Android since early versions (1.5)
 - A random canary value (unique to the process) is placed to the stack between local variables and the return address
- So one cannot overwrite the return address without overwriting this value
 - This can be detected and returning to a malicious code can be prevented!

Stack diagram showing: 0xFFFFFFFF, saved function arguments, saved registers, canary value, local variables, return address, saved PC, saved LR, saved PC, saved LR, saved PC, saved LR.

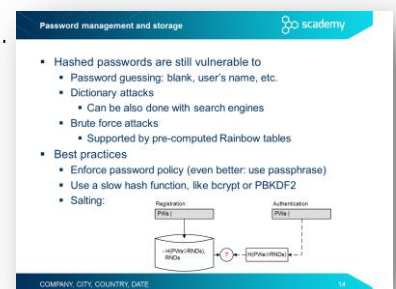
- Bypassing ASLR, XN, RELRO and stack protection
- Information leakage
- Spot the bug
- Exercise – exploit information leakage
- Spot the bug
- Use after free – Dangling pointers
- Use after free – Instance of a class
- cString class
- Information leakage with use after free
- Exercise – information leakage with use after free
- Exercise – control information leakage



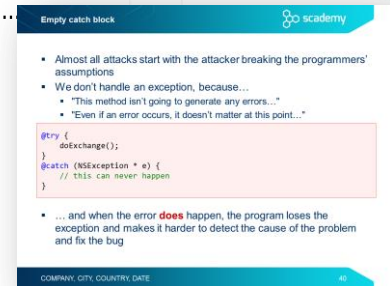
Day 4

Common coding errors and vulnerabilities

- Improper use of security features
 - Typical problems related to the use of security features
 - Insecure randomness
 - Weak PRNGs in Java
 - Exercise RandomTest
 - Using random numbers in Java – spot the bug!
 - Password management
 - Exercise – Weakness of hashed passwords
 - Password management and storage
 - Special purpose hash algorithms for password storage
 - Argon2 and PBKDF2 implementations in Java
 - bcrypt and scrypt implementations in Java
 - Password hash implementations on Android
 - KitKat changes concerning SecretKeyFactory
 - Password hash implementations on iOS
 - Case study – the Ashley Madison data breach
 - Signing and integrity protection weaknesses
 - Instagram vulnerability
 - Multiple file names in an APK
 - Access control weaknesses
 - Vulnerability in Skype for Android
 - Vulnerability in Firefox for Android
 - Google Wallet vulnerabilities
 - Storing sensitive data on iOS



- Improper error and exception handling
 - Typical problems with error and exception handling
 - Errors vs exceptions
 - Empty catch block
 - Overly broad throws
 - Overly broad catch
 - Using multi-catch
 - Catching NullPointerException
 - Null pointers in Objective C
 - Catching NullPointerException
 - Exception handling – spot the bug!
 - Exercise ScademyPay – Error handling
 - Exercise – Error handling
- Code quality problems
 - Dangers arising from poor code quality
 - Poor code quality – spot the bug!
 - Unreleased resources
 - Private arrays – spot the bug!
 - Private arrays – typed field returned from a public method
 - Exercise Object Hijack
 - Public method without final – object hijacking
 - Serialization – spot the bug!
 - Exercise Serializable Sensitive
 - Immutable String – spot the bug!
 - Exercise Immutable Strings
 - Immutability and security
 - Type mismatch – Spot the bug!



Security testing techniques and tools

- Testing Android code
 - General testing approaches
 - Testing Android code
 - Android Lint
 - Android Lint – Security features
 - Lint exercise
 - PMD
 - PMD exercise
 - FindBugs
 - FindBugs exercise

- Testing on iOS
 - OCLint
 - 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
- Java secure coding sources
- Android secure coding sources
- iOS secure coding sources @ Apple Developer
- Recommended books – Java
- Recommended books – Android
- Recommended books – iOS