scademy
secure coding academy

Android Java and native code security

CL-AAN | Classroom | 4 days

Audience: Android application developers, architects and testers
Preparedness: General Java and C/C++ development on Android
Exercises: Hands-on

Android is an open platform for mobile devices such as handsets and tablets. It has a large variety of security features to make developing secure software easier; however, it is also missing certain security aspects that are present in other hand-held platforms. The course gives a comprehensive overview of these features, and points out the most critical shortcomings to be aware of related to the underlying Linux, the file system and the environment in general, as well as regarding using permissions and other Android software development components.

Typical security pitfalls and vulnerabilities are described both for native code and Java applications, along with recommendations and best practices to avoid and mitigate them. In case of native code applications we go into more details, discussing memory management related issues, protection techniques as well as their circumvention (such as Return Oriented Programming). Finally, the most important cryptographic algorithms in symmetric cryptography, hashing, asymmetric cryptography and PKI are also discussed and put into the context of Android.

In many cases discussed issues are supported with real-life examples and case studies. Finally, we give a brief overview on how to use security testing tools to reveal any programming bugs.

Outline:

- IT security and secure coding
- Android security overview
- Practical cryptography
- Android application security
- Protecting Android applications
- Denial of service
- Input validation
- Android native code security
- Improper use of security features
- Improper error and exception handling
- Information leakage through error reporting
- Code quality problems

Testing Android 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 Android
Have a practical understanding of cryptography
Learn to use various security features of the Android platform
Learn about denial of service attacks and protections
Get information about some recent vulnerabilities in Java on Android
Learn about typical coding mistakes and how to avoid them
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 Android
Get sources and further readings on secure coding practices

Related courses:

- CL-AND - Android security (Classroom, 3 days)
- CL-IOS - iOS security (Classroom, 2 days)
- CL-AIS - Android and iOS security (Classroom, 4 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.

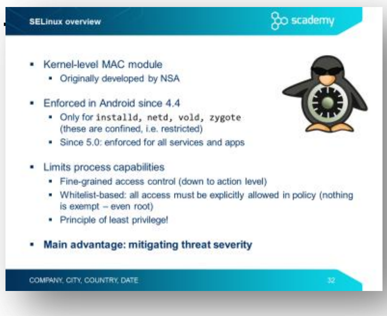
scademy
secure coding academy

Detailed table of contents

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 11



SELinux overview

- Kernel-level MAC module
  - Originally developed by NSA
- Enforced in Android since 4.4
  - Only for `installd, netd, vold, zygote` (these are confined, i.e. restricted)
  - Since 5.0: enforced for all services and apps
- Limits process capabilities
  - Fine-grained access control (down to action level)
  - Whitelist-based: all access must be explicitly allowed in policy (nothing is exempt – even root)
  - Principle of least privilege!
- **Main advantage: mitigating threat severity**

COMPANY, CITY, COUNTRY, DATE 32

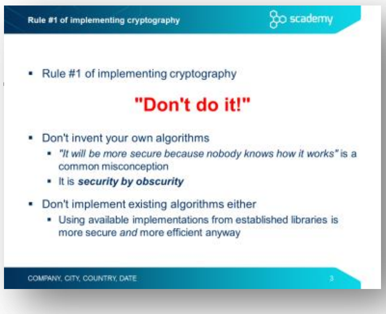
- SELinux policy example –
- Adding custom policy files
- Exercise: compiling and using SELinux policies
- SELinux Role-Based Access Control
- SELinux Multi-Level Security
- TrustZone

- 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

<u>Day 2</u>

Practical cryptography

- Rule #1 of implementing cryptography..
- Cryptosystems
  - Elements of a cryptosystem



Rule #1 of implementing cryptography

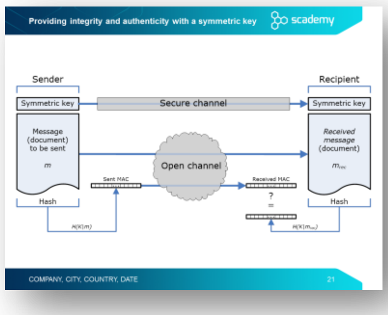
- Rule #1 of implementing cryptography

"Don't do it!"

- Don't invent your own algorithms
  - *"It will be more secure because nobody knows how it works"* is a common misconception
  - It is ***security by obscurity***
- Don't implement existing algorithms either
  - Using available implementations from established libraries is more secure *and* more efficient anyway

COMPANY, CITY, COUNTRY, DATE 3

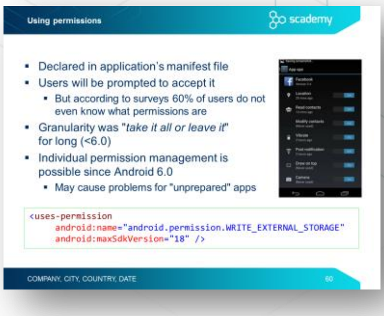
- Symmetric-key cryptography
  - Providing confidentiality with symmetric cryptography
  - Symmetric encryption algorithms
  - Modes of operation
- Other cryptographic algorithms
  - Hash or message digest
  - Hash algorithms
  - SHAttered
  - Message Authentication Code (MAC)
  - Providing integrity 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


Providing integrity and authenticity with a symmetric key
scademy
Sender
Recipient
Symmetric key
Secure channel
Symmetric key
Message (document) to be sent
m
Open channel
Received message (document)
m_rec
Sent MAC
Received MAC
?
=
Hash
Hash
H(K|m)
H(K|m_rec)
COMPANY, CITY, COUNTRY, DATE
21

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



Using permissions

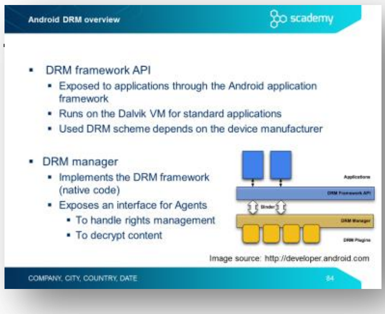
- Declared in application's manifest file
- Users will be prompted to accept it
    - But according to surveys 60% of users do not even know what permissions are
- Granularity was "*take it all or leave it*" for long (<6.0)
- Individual permission management is possible since Android 6.0
    - May cause problems for "unprepared" apps

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

COMPANY, CITY, COUNTRY, DATE 60

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



Android DRM overview

- DRM framework API
    - Exposed to applications through the Android application framework
    - Runs on the Dalvik VM for standard applications
    - Used DRM scheme depends on the device manufacturer
- DRM manager
    - Implements the DRM framework (native code)
    - Exposes an interface for Agents
        - To handle rights management
        - To decrypt content

Applications
DRM Framework API
Binder
DRM Manager
DRM Plugins

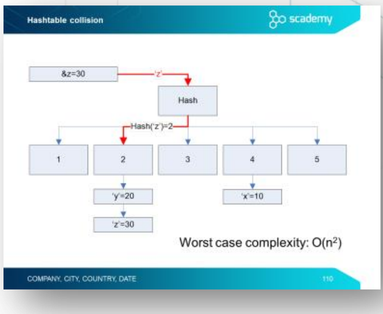
Image source: http://developer.android.com

COMPANY, CITY, COUNTRY, DATE 64

Denial of service

- DoS introduction
- Asymmetric DoS
- SSL/TLS renegotiation DoS
- Asymmetric DOS with JSON deserialization
- Regular expression DoS (ReDoS)
    - Exercise ReDoS
    - ReDoS mitigation
    - Case study – ReDos in Stack Exchange

- Hashtable collision attack
    - Using hashtables to store inputs
    - Hashtable collision



Hashtable collision

Worst case complexity: $O(n^2)$

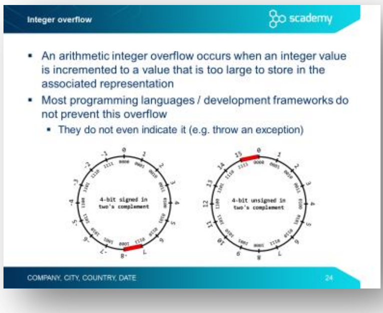
COMPANY, CITY, COUNTRY, DATE 110

    - Hashtable collision in Java

Day 3

Input validation

- Input validation concepts
- Injection
    - SQL Injection on Android
    - Typical SQL Injection attack methods
    - SQL Injection protection methods
    - Using parameterized queries in Android
- Cross-site scripting
    - Android WebView XSS
    - XSS prevention
    - Android WebView security best practices
- Integer problems
    - Representation of negative integers
    - Integer overflow



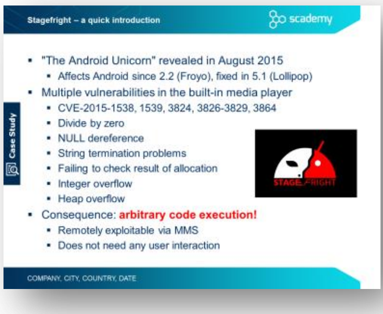
Integer overflow

- 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)

4-bit signed in two's complement
4-bit unsigned in two's complement

COMPANY, CITY, COUNTRY, DATE 24

    - 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.utils.zip.CRC32
    - Case study – Android Stagefright
        - Stagefright – a quick introduction



Stagefright – a quick introduction

Case Study

- "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

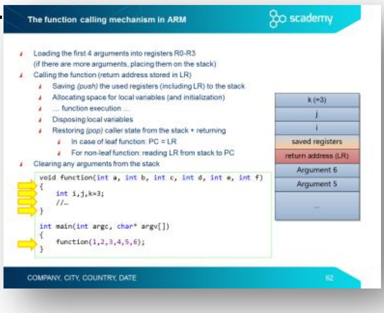
COMPANY, CITY, COUNTRY, DATE

        - Some Stagefright code examples – spot the bugs!
- Path traversal vulnerability
    - Path traversal – best practices

- Unsafe reflection
  - Implementation of a command dispatcher
  - Unsafe reflection – spot the bug!
  - Mitigation of unsafe reflection
- Log forging
  - Some other typical problems with log files

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
  - Exercise – overwrite .got with WWW after Android 4.1
- Protection techniques – ASLR, XN, RELRO, …
  - Address Space Layout Randomization (ASLR)
  - Randomization with ASLR
  - Access Control on memory segments
  - The Never eXecute (NX) bit



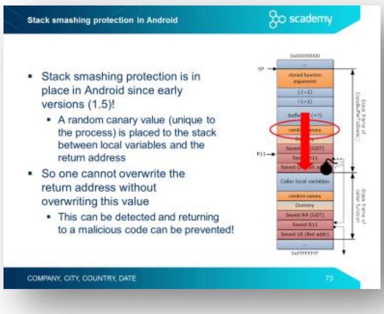
The function calling mechanism in ARM

- 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 (*push*) 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=3;
    //…
}

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

COMPANY, CITY, COUNTRY, DATE 62

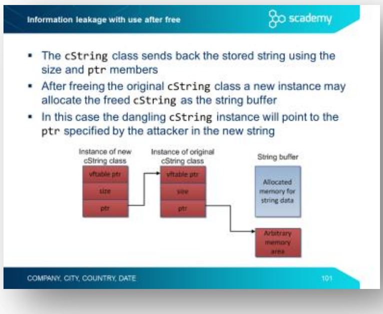


Stack smashing protection in Android

- 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!

COMPANY, CITY, COUNTRY, DATE 73

- Read-only relocation and immediate binding – RELRO
- 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
- Virtual method call
- Code execution with use after free
- App name memory corruption – caused Google Play DoS
- Buffer overflow in Android KeyStore



Information leakage with use after free

- The `cString` class sends back the stored string using the size and `ptr` members
- After freeing the original `cString` class a new instance may allocate the freed `cString` as the string buffer
- In this case the dangling `cString` instance will point to the `ptr` specified by the attacker in the new string

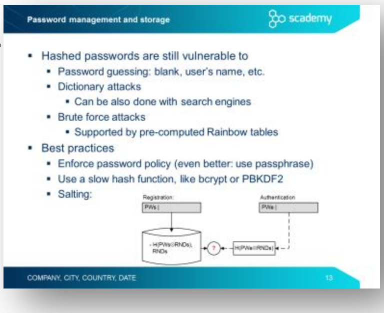
Instance of new cString class
vftable ptr
size
ptr
Instance of original cString class
vftable ptr
size
ptr
String buffer
Allocated memory for string data
Arbitrary memory area

COMPANY, CITY, COUNTRY, DATE 101

Day 4

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
  - Case study – the Ashley Madison data breach
    - The loginkey token
    - Revealing the passwords with brute forcing



Password management and storage

- Hashed passwords are still vulnerable to
  - Password guessing: blank, user's name, etc.
  - Dictionary attacks
    - Can be also done with search engines
  - Brute force attacks
    - Supported by pre-computed Rainbow tables
- Best practices
  - Enforce password policy (even better: use passphrase)
  - Use a slow hash function, like bcrypt or PBKDF2
  - Salting:

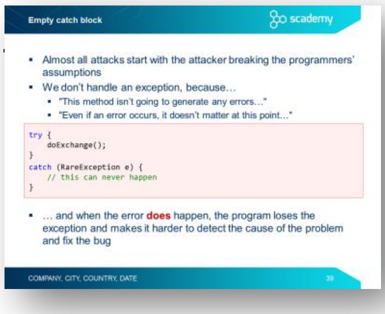
Registration:
PWs |
Authentication
PWa |
H(PWs||RNDs), RNDs
?
H(PWa||RNDs)

COMPANY, CITY, COUNTRY, DATE 13

- 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 Android
  - Exercise – Storing sensitive data without encryption
  - Storing sensitive data – Best practices

Improper error and exception handling

- Typical problems with error and exception handling
- Empty catch block
- Overly broad throws
- Overly broad catch
- Using multi-catch
- Returning from finally block – spot the bug!
- Catching NullPointerException
- Exception handling – spot the bug!
- Exercise ScademyPay – Error handling



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 (RareException 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 39

Information leakage through error reporting

- Exercise – Error handling
- Information leakage through logging (LogCat)
- GoToMeeting vulnerability
- Android best practices

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

Testing Android code

- Android Lint
- Android Lint – Security features
- Lint exercise
- PMD
- PMD exercise
- FindBugs
- FindBugs exercise

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
- Recommended books – Java
- Recommended books – Android