Mobile Security

Notes

Alberto Lazari

I Semester A.Y. 2023-2024

Index

# Lecture 2

1. Android OS

- Single user OS
- Custom Linux kernel at its core → multi-user support:
  - Each app is a different Unix user
  - Every app has a GID corresponding to the permission it has
  - App ID = UID GID1 GID2 GID3…

# Lecture 3

2. Binder driver

Android driver extension over Linux

Allows:
- Remote Procedure Call (RPC)
- Inter-Process Communication (IPC)

It's necessary to allow apps to make system calls, running in kernel space.
Example: an app wants to get the current location from the location manager service (which acts as a privileged process)

3. Permissions

Protection levels:
1. Normal: declared and automatically granted at installation (e.g. internet)
2. Dangerous: require manual user interaction (location, storage)
3. Signature: used for apps communication. Requires signature of the app to communicate with
4. SignatureOrSystem: granted to system apps

3.1. Dangerous permissions

- Before device's API level 23 or app's `targetSdkVersion` 23 they were granted at install-time
- They couldn't disable them

Permissions come in groups. Granting a specific actions grants the entire group

# Lecture 4

4. Signatures

- Used to sign certificates, to prove identity
- Hash of the data to sign, encoded with the user's private key

4.1. Certificates

Certificate Authority (CA) entity responsible to release certificates (within a hierarchy)

- Certificates contain:
  - User and CA metadata
  - User's public key (to decode signature)

Signing data: attaching a certificate and signature to a file. Guarantees:
- Sender identity: he's the only one who could encrypt with his private key
- Integrity of data: hash produced from received data has to be the same one obtained from the signature

There are two types:
- Public (trusted): generated from a CA
- Private (untrusted): self-signed

4.2. Android

- Android keys are private certificates (untrusted and self-signed). They are not released by a CA (unlike websites)
- Purpose of keys is to distinguish developers, not confirming identity
- Keys are also used to check application integrity (app not compromised: the files are the original ones, or at least signed by the official developer)

# Lecture 5

5. Services

- Components without a UI (running in the background)
- Started with `startService(explicitIntent)`
- Replies with broadcast intents (easiest way)
- Can be foreground with `startForeground()` → makes visible operations

5.1. Bound services

Allows a client-server communication between a component and its service

5.2. Broadcast receivers

- Catch broadcast intents
- Can declare filters at runtime, no need to declare in manifest file

5.3. Content provider

Exposes private data to the environment, through a URI resolved by a content resolver

# Lecture 6

6. Compilation

Android apps use a specific JVM, which is meant to be used on a mobile OS and device, the Dalvik Virtual Machine (DVM)

The DVM provides limitations and optimizations over the standard JVM bytecode, and it is directly derived from it, not from actual source code

Compilation steps of Android code:

1. **Source code** (`.java` / `.kt`)
2. **Java bytecode** (`.class` / `.jar`): compiled by `javac` or `kotlinc` compilers
3. **Dalvik bytecode** (`.dex`): compiled by `dx`, the DEX compiler

6.1. Shared Object files

Native C/C++ code compiled to `.so` files

6.2. JIT vs AOT

- The DVM was used up to Android 4.4 and used JIT
- From Android 5 Dalvik bytecode is compiled AOT during installation
- From Android 7 only *hot* methods get compiled (reduce installation time)

6.3. Decompilation

- Apk files are actually zip files with a different extension
- `apktool` can disassemble *smali* code from `.dex` files
- `jadx` can be used to decompile bytecode to java code

# Lecture 7

7. Reverse engineering

Static analysis: understand without running the app

7.1. Dynamic analysis

- Debugging: control execution flow. **Requires source code**

- Instrumentation: modify app’s code or run it in a custom emulator/art

7.1.1. Instrumentation

- Repackaging
- Frida framework: injects JS code. Can be detected and requires root
- Xposed

7.1.2. Program analysis

Automate analysis. Possibly static or dynamic

7.1.3. Evasion

Make static analysis more challenging with techniques:

- Use of reflection
- Load code dynamically
- Obfuscation
- Use native code

Identify if the app is running in an analysis environment

# Lecture 8

8. Malwares

Roles:

- Developer: writes the malware
- Customer: commissions a malware to developer
- Operator: actually starts the malware

8.1. Make users install apps

- Social engineering: free version of paid app. Similar name
- Repackaging: pirated paid apps with added malware
- Turning bad: legit app gets updated to malware (example: XcodeGhost)

# Lecture 9

9. Vulnerabilities

9.1. Types

- EOP: Elevation Of Privilege, run code that requires root privileges (from a third-party app). Ex:
  - Run code in TEE (Trusted Execution Environment)
  - Arbitrary files access
- RCE: Remote Code Execution, download and run code (e.g. from sms, web page, …)
- ID: Information Disclosure, leak sensitive data
- DOS: e.g. brick, crash device

9.2. Score

CCVS: Common Vulnerability Scoring System

9.3. Tracking

CVE: Common Vulnerabilities and Exposures

Format: `CVE-{YYYY}-{ID}`

# Lecture 10

10. Attacks subject

- User: social engineering

10.1. Apps

- Permissions abuse
- Sensitive informations leak

Possible attacks:

- Man in the middle, to catch communications between network backend
- Dynamic code loading
- External library vulnerability (cryptography)
- Confused deputy problem: an external app is able to access sensitive data of another:
  - Component hijacking: exploit exported components
  - Permission leak: exploit a permission through another app (that has it)
- Zip path traversal: exploit relative files in zip archives (`../../file`), to overwrite existing files
- Native code: every system language vulnerability (buffer overflow, …)

10.2. System

Android framework or OS itself

Examples:

- Media framework: automatically trigger code execution, by remote attack (send sms with contrived media content)