Lab-Manual

Digital Forensics

CY-334L

Prepared By: Ms.Memoona Sadaf

Instructor: Ms.Memoona Sadaf

Lab/Teaching Assistant: Muhammad Ahmad Ali Qureshi

Air University Islamabad

AIR UNIVERSITY

Department of Cyber Security

Lab Schedule

Week	Lab Topics
Week 12	**Android Application Reverse Engineering**

1. Introduction

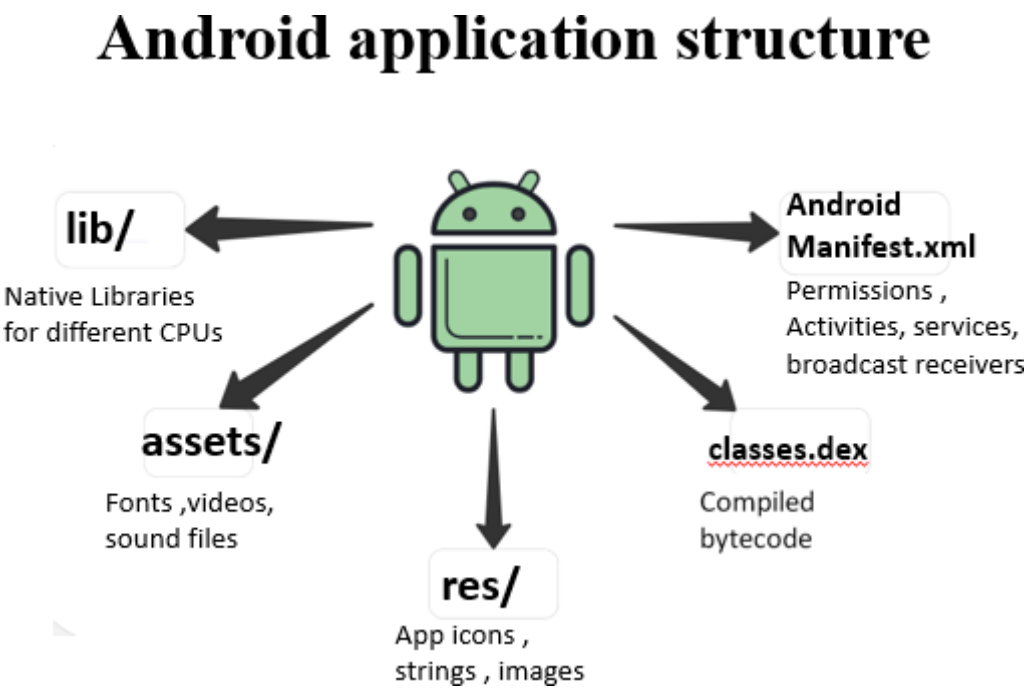
Reverse engineering in APKs means **breaking down an Android app (APK file)** to understand how it works **without having the original source code**.
APK = Android Package → it's like a ZIP file containing code, images, icons, permissions, etc.

Reverse engineering Android apps is essential for vulnerability assessment, malware analysis, and understanding undocumented APIs. This guide outlines a structured methodology for reversing Android APKs, focusing on static and dynamic techniques, tools, and common vulnerabilities.

2. Android Application Architecture

Android applications are developed using Java/Kotlin and compiled into bytecode (DEX files), then packaged into APKs. Key architectural elements include:

- **Activities:** UI components representing screens.
- **Services:** Background processes.
- **Broadcast Receivers:** Respond to system-wide events.
- **Content Providers:** Share data between apps.



3. Core Components of an Android App

Component	Description
Manifest File	Declares permissions, components, intent-filters
DEX Files	Contain compiled code
Native Libraries	C/C++ code compiled into .so files
Resource Files	XML-based layouts, strings, drawables
Assets	Custom data files accessed via AssetManager

Types of Android App Analysis

1. Static Analysis

What is it?

Looking at the app's files and code **without running the app**. Like reading a book instead of watching the movie.

What You Do:

- Unpack or decompile the APK
- Look at the code, resources, manifest, etc.
- Search for bugs, security issues, or secrets

Tools Used:

- **APKTool** – to unpack APK files
- **JADX** – to read Java code from DEX files
- **MobSF (Mobile Security Framework)** – automated scanner
- **grep / strings / VS Code** – to search through code and files

What You Can Find:

- API keys or passwords hardcoded in the app
- Dangerous permissions (camera, SMS, etc.)
- Debug mode enabled (android:debuggable="true")
- Exported activities or services that shouldn’t be public
- Insecure WebView usage
- Native libraries (.so files) to inspect

2. Dynamic Analysis

What is it?

Running the app on a phone (real or emulator) and **watching how it behaves**. Like observing how someone acts in real life instead of reading their diary.

What You Do:

- Install the app on a test device or emulator
- Interact with the app (click, login, navigate)
- Monitor network traffic, API calls, and data flows

Tools Used:

- **Frida** – for live code hooking and spying
- **Burp Suite / Wireshark** – to monitor network traffic
- **Xposed Framework** – to modify behavior at runtime
- **Logcat** – to see logs from the app
- **Magisk** – to root Android devices safely

What You Can Find:

- Is data being sent unencrypted (no HTTPS)?
- Does the app leak sensitive info like GPS, phone number?
- How does the app handle logins and sessions?
- Does the app try to detect if the device is rooted?
- Can you bypass login screens or unlock paid features?

3. Comparison Table

Feature	Static Analysis	Dynamic Analysis
Runs the app?	❌ No	✅ Yes
Needs emulator/device?	❌ No	✅ Yes
Looks at code?	✅ Yes	🔸 Some (only at runtime)
Finds network issues?	❌ Limited	✅ Yes
Detects runtime bugs?	❌ No	✅ Yes

Feature	Static Analysis	Dynamic Analysis
Best for...	Code review, secrets, permissions	Behavior, traffic, live testing

How Are Android Apps Made Today?

Modern apps aren't always written in just Java! Developers now use **frameworks** and **cross-platform tools**.

Tool/Language	Description
Java	Traditional Android language
Kotlin	Modern Android language (Google prefers this now)
React Native	JavaScript-based, builds apps for Android & iOS
Flutter	Uses Dart, also works on Android & iOS
C++	Used for performance-critical parts (games, media)

These tools generate APKs too, but the structure might be slightly different. For example:

- Flutter uses .dart files compiled into native code.
- React Native apps include a JavaScript bundle inside assets/.

GennyMotion Emulator Setup

The blog provides a step-by-step walkthrough for setting up Genymotion on Windows, covering:

https://medium.com/@nikhilbwr34/genymotion-101-how-to-sign-up-and-configure-it-for-windows-b237e7d36366

Reverse Engineering Modern Android Frameworks

Framework	Main Code Format	Where It's Stored	Tools to Use
Java	.dex	`classes.dex`, `smali/`	APKTool, JADX
Kotlin	.dex	`classes.dex`, `smali/`	APKTool, JADX

React Native	JavaScript bundle	`assets/index.android.bundle`	APKTool, JS beautifier, rn-bundle-inspector
Flutter	Native .so (Dart)	`lib/libapp.so`	Ghidra, IDA, Radare2
C++ NDK	Native .so	`lib/armeabi-v7a/*.so`	Ghidra, IDA, Frida

Conclusion

Android application reverse engineering is a powerful technique that helps us understand how mobile apps work, how they are built, and how secure they are. Through this process, we can uncover hidden behaviors, debug logic, analyze vulnerabilities, and learn from real-world code.

We explored two main analysis techniques:

- **Static Analysis**: Involves breaking down the APK without running it, using tools like APKTool and JADX to view the app's manifest, resources, and code.
- **Dynamic Analysis**: Involves running the app in a controlled environment to observe how it behaves in real-time, using tools like Frida or Burp Suite.

Modern Android apps are built using a variety of technologies such as Java, Kotlin, React Native, Flutter, and C++. Each framework introduces new structures and complexities that influence how we reverse engineer them.

Key takeaways include:

- APKs are structured packages containing app code, resources, and configuration files.
- Reverse engineering can expose hardcoded secrets, debug flags, insecure data handling, and poor encryption practices.
- Understanding the build technology (like React Native or Flutter) is critical, as each requires different tools and techniques.
- This practice is useful for security analysis, ethical hacking, and software learning — but must be done responsibly and legally.

In conclusion, reverse engineering not only enhances our technical skills but also helps us become more aware of app security, design flaws, and how mobile technologies operate behind the scenes.