

ISEA

Demystifying Mobile (Android) Application Security

**By - Venkat R Kodimela,
Project Manager, C-DAC Hyderabad**

Session - Agenda

1. Security Analysis
 - a. Static Analysis (SAST)
 - b. Dynamic Analysis (DAST)
 - c. Behavioural Analysis
2. OWASP MASVS & MASTG
3. Advanced Static Analysis Android Apps
4. Reverse Engineering of Android APKs
5. Reverse Engineering Tools
 - a. JADX-GUI
 - b. Apktool
6. Protection against Reverse Engineering

Security Analysis

Security Analysis

Mobile Landscape :

- Rampant proliferation of mobile applications through App Stores / Play Stores
- Mobile applications are a gateway to sensitive data and critical functionalities.
- Growing attack vectors due to increased adoption and reliance on mobile devices.

Why Security Analysis Matters:

- To prevent data breaches
- To build user trust and gain confidence
- To be compliant with industry regulations.
- To safeguard business reputation

Security Analysis - Key aspects

- ▶ **Data Storage:**
 - ▶ Verify encryption and avoid sensitive data in plaintext.
- ▶ **Communication:**
 - ▶ Ensure HTTPS/TLS and avoid untrusted certificates.
- ▶ **Authentication:**
 - ▶ Multi-factor authentication and secure token handling.
- ▶ **Third-party Libraries:**
 - ▶ Check for vulnerabilities using CVE databases.
- ▶ **Reverse Engineering Prevention:**
 - ▶ Check for Code Obfuscation, RASP (Runtime Application Self-Protection).

Key Challenges in Mobile Application Security

- ▶ **Platform-Specific Vulnerabilities**
 - ▶ Older versions of Android SDK may be vulnerable
- ▶ **Developer not adhering to secure coding practices:**
 - ▶ Common security flaws (e.g., improper encryption, insecure communication etc.)
- ▶ **Evolving Threat Landscape:**
 - ▶ Sophisticated attacks like mobile malware and application re-packaging.
- ▶ **Compliance Needs:**
 - ▶ Adhering to GDPR, CCPA, and industry standards like OWASP MASVS.

Security Analysis - Techniques

- ▶ **Static Analysis:**
 - ▶ Analyzing the application without executing i.e. examining the application's source code and binaries to find vulnerabilities.
 - ▶ Tools: MobSF, Reverse Engineering tools (JADX-GUI, Apktool)
 - ▶ Targets: Hardcoded credentials, misuse of permissions, insecure APIs etc.
- ▶ **Dynamic Analysis:**
 - ▶ Analyzing the application during runtime / execution mode.
 - ▶ Tools: Dynamic Instrumentation tools (Frida, Objection), Burp Suite, Drozer
 - ▶ Focus: Runtime behavior, insecure communication, root detection bypass, insecure data storage etc.

Static Analysis (SAST)

Static Analysis (SAST)

- ▶ **Static Application Security Testing (SAST),**
 - ▶ Critical process in security analysis performed early in the software development life cycle (SDLC)
 - ▶ It is a shift-left activity aimed to identify vulnerabilities, bugs, and coding issues before application is released
 - ▶ Performed manually & also using Automated tools
 - ▶ APK is decompiled to gain access to source code, bytecode for analysis
 - ▶ Detect hard coded credentials, insecure APIs, or weak cryptography.
 - ▶ Analyze critical or suspicious areas manually, focusing on application logic and sensitive operations.

Tools for Static Analysis

- ▶ **MobSF (Mobile Security Framework)** - Performs static analysis of the uploaded Android application i.e. APK file
- ▶ **JADX-GUI (Decompiler)** - Tool to decompile Android applications i.e. APK files into readable Java source code.
- ▶ **APKTool** - Reverse-engineering tool for analyzing Android applications i.e. APK files.
- ▶ **QARK (Quick Android Review Kit)** - Security tool specifically designed for static analysis of Android applications i.e. APK files.
- ▶ **Ghidra** - Advanced reverse engineering and static analysis of obfuscated APKs

Static Analysis - Types of Vulnerabilities

- ▶ **Security Flaws:**
 - ▶ Insecure data handling, hardcoded secrets, improper input validation.
- ▶ **Old Coding Conventions:**
 - ▶ Poor coding practices that may lead to maintainability issues.
- ▶ **Performance Issues:**
 - ▶ Inefficient algorithms or memory usage patterns.
- ▶ **Standards Violations:**
 - ▶ Non-adherence to coding guidelines or regulatory standards.

Key Benefits of Static Analysis

- ▶ **Key Benefits**
 - ▶ Early detection of issues reduces remediation costs.
 - ▶ Improves code quality and adherence to coding standards / best practices.
 - ▶ Supports secure software development by identifying vulnerabilities early.
 - ▶ Enhanced Security towards detecting critical vulnerabilities such as:
 - ▶ Hardcoded credentials.
 - ▶ SQL injection vulnerabilities
 - ▶ Cross-Site Scripting (XSS) risks
 - ▶ Insecure API calls etc.

Dynamic Analysis (DAST)

Android Applications - Dynamic Analysis

- ▶ **Dynamic Analysis :**
 - ▶ Analyzing application during runtime to identify security vulnerabilities, performance bottlenecks, and functional correctness.
 - ▶ Useful in detecting issues that depend on the execution environment, user interactions, or external systems
 - ▶ Simulates real-world conditions to detect security issues, vulnerabilities, and misconfigurations
 - ▶ Involves testing the functional aspects of an application to ensure that all features behave as expected under normal, edge, and adverse conditions

Android Applications - Dynamic Analysis

- ▶ **Focus Areas:**
 - ▶ Monitoring runtime behavior like
 - ▶ Storage of personal data
 - ▶ Runtime Permissions
 - ▶ System interactions
 - ▶ Intercepting and analyzing network traffic
 - ▶ Testing how the app responds to different inputs
 - ▶ Detecting runtime vulnerabilities like
 - ▶ Insecure Authentication / Authorization
 - ▶ Improper session handling, etc.

Key benefits - Dynamic Analysis

- ▶ Provides actionable insights into the runtime behavior of mobile applications such as.
 - ▶ Identification of Runtime Vulnerabilities
 - ▶ Improved Security Posture
 - ▶ Deeper insights into Network Security
 - ▶ Validation of Secure Data Storage
 - ▶ Detection of Exploitable Application Logic
 - ▶ Confirmation of Anti-Tampering and Root/Jailbreak Detection
 - ▶ Analysis of Application integrity implementation
 - ▶ Insights into Security of API Endpoints

Tools used in Dynamic Analysis

- ▶ **Commonly used tools :**
 - ▶ **Burp Suite** - Intercepts and analyzes network traffic to detect vulnerabilities
 - ▶ **Drozer** - Focuses on interacting with and manipulating Android components such as Activities, Services, Broadcast Receivers, and Content Providers
 - ▶ **Xposed Framework** - Allows to hook into app behavior at runtime and modify or analyze specific functions without altering
 - ▶ **Dynamic Instrumentation Tools** such as
 - ▶ **Frida** - Allows real-time analysis and debugging, including detecting function calls, analyzing API usage, and bypassing security mechanisms.
 - ▶ **Objection** - Allowing to interact with the app's processes, functions, and variables dynamically

Behavioural Analysis

Android Applications - Behavioural Analysis

- ▶ **Behavioural Analysis** : Monitoring the application's behavior in various runtime environments to detect anomalies or malicious activities.
 - ▶ Monitors how an application interacts with system resources such as the file system, network, memory, and CPU
 - ▶ Monitoring the permissions and access control mechanisms of the application during runtime / execution
 - ▶ Helps to understand how the app reacts to unexpected conditions, such as low battery, network outages, device rotation, or system-level events like app suspension.
 - ▶ Assess how the app handles cryptographic operations, such as encryption, decryption, and key management

Android Applications - Behavioural Analysis

- ▶ **Focus Areas:**
 - ▶ Monitoring resource usage
 - ▶ Battery
 - ▶ CPU
 - ▶ Memory
 - ▶ Detecting malicious activities like
 - ▶ Data exfiltration
 - ▶ Hidden background processes.
 - ▶ Testing for malware or spyware behavior in real-world conditions.

Android Applications - Behavioural Analysis

- ▶ Behavioral analysis in mobile application security are critical for understanding
 - ▶ Real-time operations of an app and identifying security, performance, and usability issues
 - ▶ Unencrypted network traffic - Sensitive data such as user credentials, credit card information and personally identifiable information (PII)
 - ▶ Identifying instances where the app or its components improperly grant elevated privileges, such as system-level access
 - ▶ Detecting issues like excessive CPU usage, memory leaks, or inefficient data processing that could degrade the app's performance

Android Applications - Behavioural Analysis

- ▶ **Tools:**
 - ▶ **Sandbox environments** - Execution of malware in multiple environments without affecting network resources
 - ▶ **TaintDroid** - Tracks and analyzes third-party apps on Android devices which collect and share users' private data in real time
 - ▶ **Quark Engine** - Automating analysis of suspicious Android application using dynamic analysis, network traffic interception, system call tracing, and code instrumentation
 - ▶ **AppMon** - Monitors the behavior of mobile apps and helps to identify memory leaks, crashes, performance bottlenecks, and other runtime issues

Advanced Static Analysis for Android Application Security Testing

Introduction to Advanced Static Analysis

- ▶ Sophisticated method of examining software or code without executing it
- ▶ Focusing on understanding its structure, behavior, and potential vulnerabilities at a deeper level
- ▶ It goes beyond basic static analysis (e.g., syntax checking or simple pattern matching)
- ▶ Employs advanced techniques to analyze the
 - ▶ code's logic
 - ▶ data flow
 - ▶ control flow, and
 - ▶ interactions with external systems

Key Aspects of Advanced Static Analysis

<p>Data Flow Analysis</p>	<p>Tracks how data moves through the program, identifying where sensitive data (e.g., passwords, encryption keys) is used, stored, or leaked. Eg. Insecure data Handling or data breaches</p>
<p>Control Flow Analysis</p>	<p>Examines the order in which instructions are executed, identifying loops, conditional branches, and function calls. Eg. Uncover logical flows, dead code or unexpected program behavior.</p>
<p>Taint Analysis</p>	<p>Identifies untrusted or "tainted" data (e.g., user input) and tracks how it propagates through the code. Helps in detect vulnerabilities like SQL injection, cross-site scripting (XSS), or command injection.</p>
<p>Symbolic Execution</p>	<p>Analyzes the program by simulating execution with symbolic values instead of concrete inputs. Explores multiple execution paths to identify potential vulnerabilities or edge cases</p>

Key Aspects of Advanced Static Analysis

Abstract Interpretation	Uses mathematical models to approximate the behavior of the program, ensuring a comprehensive analysis of all possible states. It helps identifying issues like buffer overflows, null pointer dereferences, or resource leaks.
Dependency Analysis	Examines relationships between different components, libraries, or modules in the code. It helps to identify unused dependencies, compatibility issues, or potential attack surfaces.
Pattern Matching and Heuristics	Uses predefined rules or machine learning models to identify known vulnerabilities, anti-patterns, or malicious code.
Cross-Platform and Multi-Language Support	Advanced static analysis tools often support multiple programming languages and platforms, enabling analysis of complex, heterogeneous systems.

Advantages of Advanced Static Analysis

- ▶ **Vulnerability Detection**
 - ▶ Identifying security flaws such as buffer overflow, injection vulnerabilities, or insecure API usage.
- ▶ **Compliance Checking**
 - ▶ Ensuring code adheres to security standards (e.g., OWASP, PCI-DSS) or coding guidelines (e.g., MISRA).
- ▶ **Malware Analysis**
 - ▶ Detecting malicious behavior in binaries or scripts without executing them.
- ▶ **Code Quality Improvement**
 - ▶ Identifying inefficiencies, dead code, or potential bugs.
- ▶ **Reverse Engineering**
 - ▶ Understanding the behavior of compiled binaries or proprietary software.

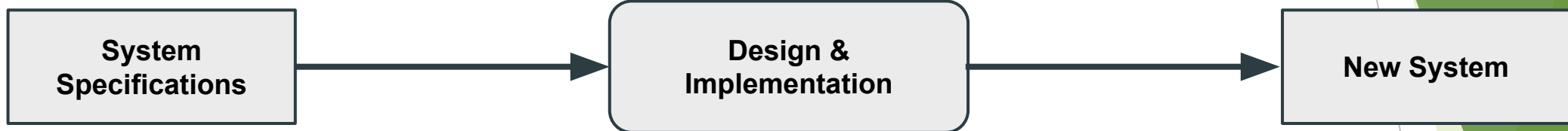
Key Challenges & Best Practices in Advanced Static Analysis

- ▶ **Key Challenges**
 - ▶ **False Positives:** Static analysis tools may report issues that are not actual vulnerabilities.
 - ▶ **Complexity:** Advanced techniques require expertise to configure and interpret results.
 - ▶ **Performance:** Large codebases can slow down analysis.
 - ▶ **Evasion Techniques:** Obfuscation or encryption can hinder static analysis.
- ▶ **Best Practices**
 - ▶ Use multiple tools to reduce false positives and increase coverage.
 - ▶ Regularly update rules and signatures to detect new vulnerabilities.
 - ▶ Integrate static analysis into the CI/CD pipeline for continuous testing.
 - ▶ Train developers to interpret and act on static analysis results.

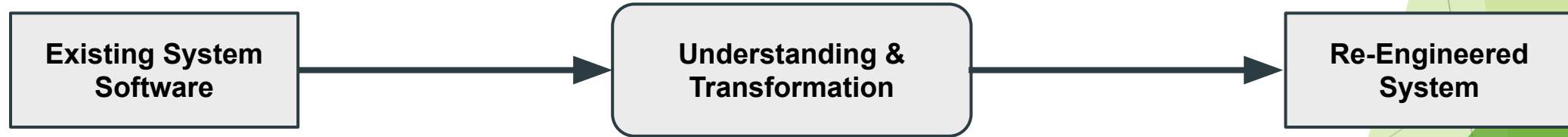
Reverse Engineering of Android APKs

Introduction to Reverse Engineering

- ▶ **Definition:** Reverse engineering involves analyzing the structure and behavior of an application to understand its components and functionalities.
- ▶ **Purpose:**
 - ▶ Security analysis
 - ▶ Malware Detection
 - ▶ Vulnerability Discovery
 - ▶ App Functionality Analysis
 - ▶ App Integrity Verification
 - ▶ Code Optimization and Refactoring
 - ▶ Intellectual property protection



Forward Engineering
VS
Reverse Engineering



Motivation to Perform Reverse Engineering

- ▶ **Interoperability**
 - ▶ An application or software to work on another platform.
- ▶ **Lost documentation**
 - ▶ Document the application's business functions & business logic implemented
- ▶ **Product analysis**
 - ▶ To determine how the application works
- ▶ **Security auditing**
 - ▶ An audit determines if systems are safeguarding assets, maintaining data integrity, and operating effectively.

Common Reverse Engineering Techniques

- ▶ **Decompilation & Disassembly:**
 - ▶ Using JADX, APKTool to analyze source code.
- ▶ **String Analysis:**
 - ▶ Searching for sensitive info (API keys, passwords).
- ▶ **Binary Analysis:**
 - ▶ Using tools like IDA Pro or Ghidra for native libraries.
- ▶ **Tampering:**
 - ▶ Injecting code or removing DRM.
 - ▶ Repackaging and resigning apps.

Reverse Engineering Tools

- ▶ **Apktool:**
 - ▶ Decompile and reassemble APKs, view resources.
- ▶ **dex2jar:**
 - ▶ Convert .dex files to .jar files for easier analysis.
- ▶ **JADX-GUI:**
 - ▶ Java decompiler to view source code from .jar files.
- ▶ **IDA Pro (Interactive Disassembler):**
 - ▶ Industry-standard disassembler and debugger for analyzing binaries.
- ▶ **Ghidra:**
 - ▶ Industry-standard disassembler and debugger for analyzing binaries.

Android App - Reverse Engineering - Steps to follow

- ▶ **Obtaining the APK:**
 - The first step is to download the APK file, which can often be done directly from various App stores that host APK files or by extracting it from a device.
- ▶ **Decompiling and disassembling:**
 - Tools like `apktool` are used to decompile the APK to extract the manifest and resource files. Decompilers like JADX or JD-GUI convert `.dex` files into readable Java code.
- ▶ **Analysis:**
 - The decompiled code is analyzed to understand the app's behavior and search for vulnerabilities such as hard-coded secrets, insecure implementations, and points of data leakage.

Reverse Engineering Tools

Reverse Engineering Tools - Walkthrough

- ▶ The following tools are really useful Reverse Engineering of an Android application
 - ▶ ADB (Android Debug Bridge)
 - ▶ ApkTool
 - ▶ JADX / JADX-GUI
- ▶ All above mentioned tools are Open Source Software (OSS) and are freely available to use

Tools Walkthrough - adb

- ▶ **What is ADB ?**
 - ▶ Android Debug Bridge (adb) is a command-line tool that lets us communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps.
- ▶ **General commands:**
 - ▶ adb devices - list connected devices
 - ▶ adb version - show version num
 - ▶ adb shell - Go to device's shell

Extract APK from the device using ADB

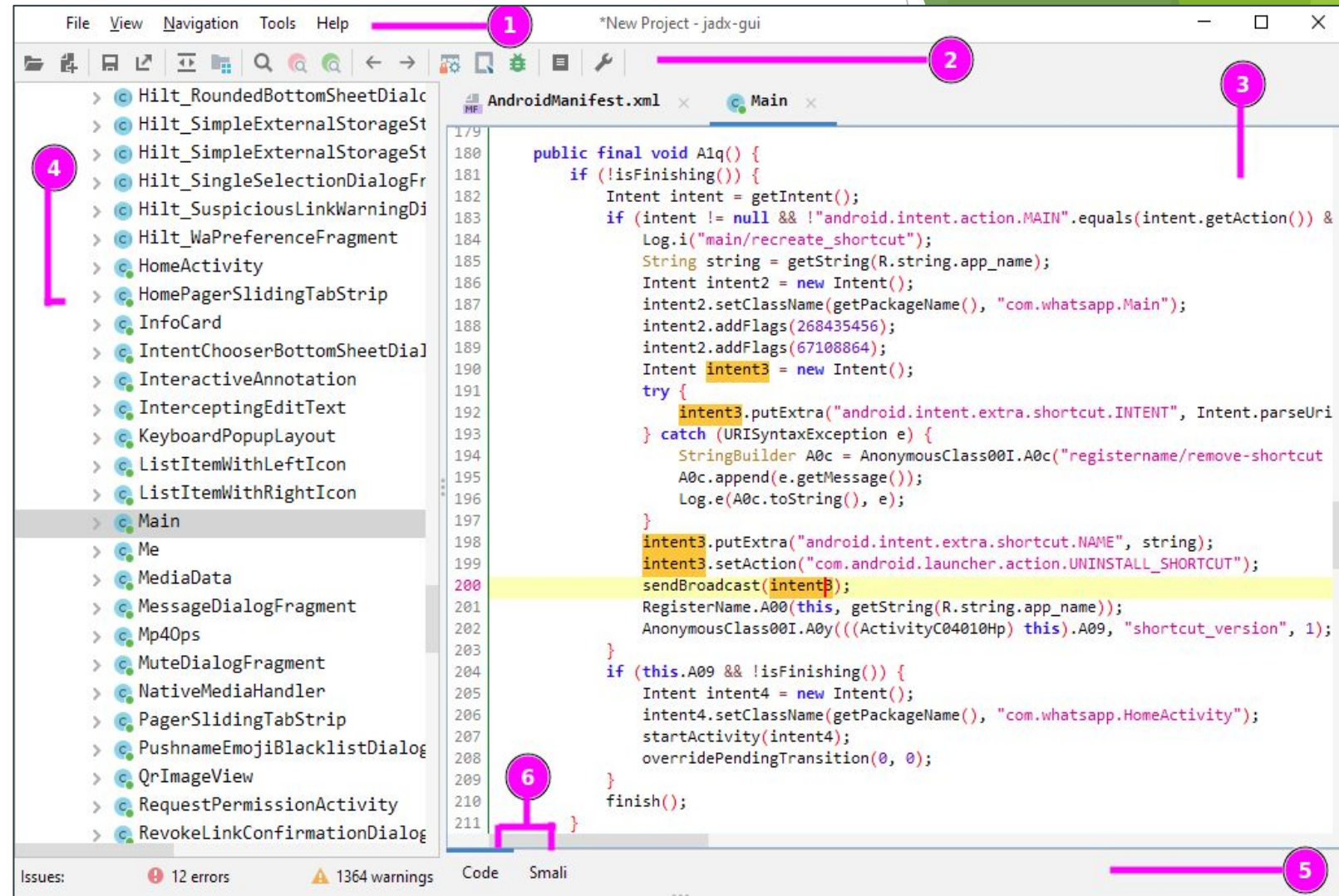
- ▶ You can extract an APK file from a device using **ADB** or download it from a trusted source.
 - ▶ Connect device using a USB cable and ensure that **USB debugging** is enabled.
 - ▶ List installed packages on the device:
 - ▶ *adb shell pm list packages*
 - ▶ Locate the specific app you want to extract (e.g., com.example.app), and find its path:
 - ▶ *adb shell pm path com.example.app*
 - ▶ Use the following command to pull the APK to your computer:
 - ▶ *adb pull /path/to/apk
/desired/local/directory/*

Tools Walkthrough | jadx

- ▶ **What is JADX ?**
 - ▶ JADX is a tool for reverse engineering Android applications.
 - ▶ This tool allows decompiling bytecode to Java source code from APK and DEX files.
 - ▶ It also allows you to decode resources such as XML files and images.
 - ▶ JADX provides CLI and GUI.

● Jadx User Interface

1. Menu Bar.
2. Toolbar.
3. Source Code Viewer.
4. Project View.
5. Status Bar.
6. Code Switch.



Tools Walkthrough | Jadx

- ▶ How to Use ?
 - ▶ Type the following command on the terminal. Browse the bin folder and execute below command
 - ▶ `./jadx-gui <../path/../../xxxx.apk>`

The screenshot shows the Jadx GUI interface. On the left, a file tree displays the decompiled structure of 'FotaProvider.apk', with 'MainActivity' selected. The main window shows the Java source code for 'MainActivity', which extends 'Activity' and overrides 'onCreate'. The code is as follows:

```
1 package com.adups.fota.sysoper;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5
6 /* loaded from: classes.dex */
7 public class MainActivity extends Activity {
8     @Override // android.app.Activity
9     protected void onCreate(Bundle bundle) {
10         super.onCreate(bundle);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

Explore Decompiled Code with JADX-GUI

- ▶ **JADX-GUI** automatically decompiles the APK into readable Java code and organizes the files in a project-like structure.
- ▶ **Source Code (src):** Decompiled Java code of the app. The Java classes, interfaces, and methods are represented as close as possible to the original code.
 - ▶ **MainActivity.java, AppConfig.java, etc.,** are found depending on the app's structure.
- ▶ **Resources (res):** The resources folder contains all the app's assets such as layouts, strings, images, and more.
 - ▶ Files such as
 - ▶ strings.xml
 - ▶ colors.xml
 - ▶ activity_main.xml

JADX - Analyze Decompiled Code

- ▶ **JADX-GUI** attempts
 - ▶ to extract the original **Java code**. This makes it much easier to read, understand, and analyze.
- ▶ **Examine Resources and Manifest**
 - ▶ **Resources:** Analyze layouts, strings, and other resources used in the app.
 - ▶ **AndroidManifest.xml:** Check the manifest to see the app's
 - ▶ Permissions, Components (Activities, Services, and Broadcast receivers etc.)
 - ▶ **Libs and Native Code (lib):** If the app uses native libraries, these files will be present here (e.g., .so files).
 - ▶ **Assets:** These include additional files that the app uses, such as HTML, JSON, or other configuration files.

Example of Decompiled Code - Java

```
package com.adups.fota.sysoper;

import android.app.Activity;
import android.os.Bundle;

/* loaded from: classes.dex */
public class MainActivity extends Activity {
    @Override // android.app.Activity
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.activity_main);
    }
}
```

Apktool - Reverse Engineering Tool

- ▶ **What is Apktool ?**
 - ▶ A tool for reverse engineering 3rd party, closed, binary Android apps.
 - ▶ Features include:
 - ▶ Disassembling resources to nearly original form
 - ▶ Rebuilding decoded resources back to binary APK/JAR
 - ▶ Organizing and handling APKs that depend on framework resources
 - ▶ Smali Debugging

Tools Walkthrough | Apktool

- ▶ Useful Apktool commands
 - ▶ apktool d <apk-file>
 - ▶ apktool b <decompiled-folder>

```
cdac@cdac-Precision-3640-Tower:~/LabApks$ cd ~
cdac@cdac-Precision-3640-Tower:~$ apktool d ~/LabApks/HelloAndroid.apk
I: Using Apktool 2.6.0 on HelloAndroid.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/cdac/.local/share/apktool/framework/1
.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes3.dex...
I: Baksmaling classes2.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
cdac@cdac-Precision-3640-Tower:~$
```

Understanding Decompiled Folder Structure of APK

- ▶ APK is decompiled, **APKTool** organizes the output into several key directories:
 - ▶ **smali/**: Contains the disassembled **Smali** code, which is the human-readable version of the **Dalvik bytecode**.
 - ▶ **res/**: Contains application resources, such as layouts, drawables, and strings.
 - ▶ **AndroidManifest.xml**: The manifest file, containing essential app configurations, permissions, and activities.
 - ▶ **assets/**: If the APK Contains non-compiled assets like HTML, text, and other file types.
 - ▶ **lib/**: If the APK contains native libraries (e.g., .so files), are resided in this folder.

Example of Decompiled Smali Code

```
##### Class com.adups.fota.sysoper.MainActivity  
(com.adups.fota.sysoper.MainActivity)  
.class public Lcom/adups/fota/sysoper/MainActivity;  
.super Landroid/app/Activity;  
# direct methods  
.method public constructor <init>()V  
    .registers 1  
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V  
    return-void  
.end method  
# virtual methods  
.method protected onCreate(Landroid/os/Bundle;)V  
    .registers 3  
    invoke-super {p0, p1},  
Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V  
    const/high16 v0, 0x7f030000  
    invoke-virtual {p0, v0},  
Lcom/adups/fota/sysoper/MainActivity;->setContentView(I)V  
    return-void  
.end method
```

Analyze the Decompiled Code

- ▶ **Analyze the Smali Code:**
 - ▶ Smali code is a low-level representation of the bytecode, resembling assembly language.
 - ▶ Each .smali file corresponds to a class in the app, with methods and fields defined similarly to Java code.
 - ▶ Smali code is widely used in reverse engineering, modifications (repackaging), and security analysis of Android apps.
 - ▶ Smali code can be modified to bypass security mechanisms, modify app behavior, or even disable certain features

Java Keywords and Corresponding Smali Instructions

Few mappings of Java Keywords to .smali equivalent instructions

Java Keyword	Description	smali equivalent
class	Declaration of a class	.class, .super, .source
extends	Inheritance of class	.super
implements	Interface implementation	.implements
new	Creation of new object instance	new-instance, invoke-direct
int, float, etc.	primitive data types	const, iput, iget (for fields)
if, else, switch	Conditional Statements	if-eq, if-nez, packed-switch, sparse-switch
try, catch	Exception Handling	.catch, throw, move-exception

Analyze the Decompiled Code

- ▶ **Review the AndroidManifest.xml:**
 - ▶ The **AndroidManifest.xml** file contains essential information about the app, such as
 - ▶ permissions,
 - ▶ services, and
 - ▶ intent filters.
 - ▶ This file is a good starting point for analyzing how the app functions and what permissions it requests from user for application's functionality.
- ▶ **Inspect Resources:**
 - ▶ Resources like layouts, drawables, and strings are available in the `res/` directory.
 - ▶ These resources can give insights into the app's UI and any hardcoded data.

Comparison b/w JADX-GUI vs APKTool

Features	JADX-GUI	APKTool
Decompiled Output	Java Source Code	Smali code (low-level bytecode)
Complexity	Easier to read and understand	Requires knowledge of Smali and bytecode
File Types	Decompiles Java code, resources, and manifest	Decompiles resources, manifest, and Smali code
Use Cases	Ideal for analyzing app logic, algorithms, and vulnerabilities in a readable format	Better for making modifications and rebuilding APKs
Recompiling Support	Does not support recompiling the APK	Supports recompiling APK after modifications

Protecting Apps from Reverse Engineering

- ▶ **Check for Rooted Devices:**
 - ▶ Use root-detection techniques (e.g., checking for su binary, Superuser APK) to identify if the device is rooted, as this can compromise security.
- ▶ **Code obfuscation:**
 - ▶ To make code harder to read by renaming classes and variables to meaningless names.
 - ▶ Tools such as ProGuard, DexGuard & Dasho may be used

Protecting Apps from Reverse Engineering

- ▶ **Encryption:**
 - ▶ Encrypting sensitive data within the app and
 - ▶ Using secure communication protocols to protect data in transit.
- ▶ **Secure coding practices:**
 - ▶ Avoiding hard-coded sensitive information,
 - ▶ Using secure storage mechanisms,
 - ▶ Implementing proper error handling to not expose sensitive information.

Protecting Apps from Reverse Engineering

- ▶ **Integrity Checks (Checksum/Hashing):**
 - ▶ Perform integrity checks by calculating a checksum or hash for critical app files to detect tampering.
- ▶ **Runtime protection:**
 - ▶ Implementing checks for jailbroken or rooted devices, debuggers being attached, or tampering with the app's runtime environment.

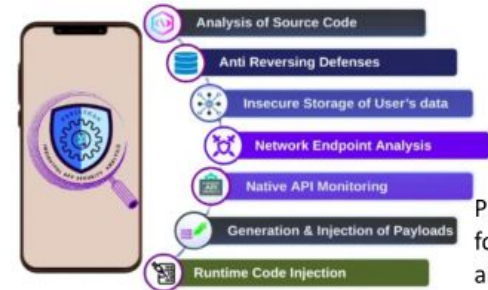
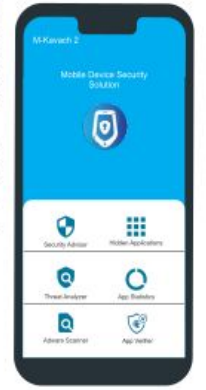
Protecting Apps from Reverse Engineering

- ▶ **Using advanced security solutions:**
 - ▶ Employing strong cryptography algorithms
 - ▶ RASP (Runtime Application Self-Protection) technologies can further enhance security against reverse engineering.
- ▶ **Prevent Emulator Detection :**
 - ▶ Implement emulator detection techniques to prevent reverse engineers from running the app in a simulated environment.

Mobile Security Products @ C-DAC Hyderabad



M-Kavach 2 is a comprehensive mobile device security solution addressing emerging threats related to Android based mobile devices. The major emphasis is on advising the users against security misconfigurations, detection of hidden apps and scanning the device for potentially risky apps installed on the user's mobile device.



Parikshan is an Automation tool mainly focused on performing static and dynamic analysis of mobile applications. The tool has the capabilities to identify the security vulnerabilities and perform penetration testing for few of them. As an outcome, a detailed security audit report is generated containing the information about the identified vulnerabilities which aids to carry out further analysis.



In recent times, mobile devices are not only used for personal purposes but also increasingly employed for managing businesses due to their convenience and mobility. This is where M-Prabandh MDM solution allows organizations to effectively monitor & manage mobile devices and reduce the risk of data breaches & theft. It also aids in safeguarding business critical information, and uphold data security standards



**Do you have any
questions?**

Thanks