**Reverse Engineer Android APKs and Auto-Modify Permissions/UI Strings**

**VERSION:** User Edition

**CREATED DATE:** 27/12/2024

**PREPARED BY:** Sai Datta Badam Roll No: 2210030422

**Table of contents**

|  |  |
| --- | --- |
| **s.no** | **List of contents** |
| 1 | Introduction |
| 2 | Project scope |
| 3 | Project objectives |
| 4 | Project Problems |
| 5 | Methodology |
| 6 | Techniques |
| 7 | Conclusion |

**Introduction**

Reverse engineering is the process of dissecting and analyzing software to understand its architecture, functionalities, and behavior without having access to its source code. In the context of Android applications, this technique plays a significant role in various domains such as cybersecurity, software testing, malware analysis, and app customization.

Android apps are distributed in the form of APK (Android Package Kit) files, which contain compiled code (classes.dex), resources (res/), native libraries (lib/), and the AndroidManifest.xml file. These APKs are usually compiled and signed by developers, making them difficult to inspect or modify directly. However, tools like **Apktool** make it possible to decode an APK into a human-readable format, enabling modification and rebuilding.

This project involves reverse engineering APK files to modify app permissions. The project demonstrates how Apktool can be used to decompile an APK, analyze its contents (especially permissions), make changes to the manifest file, and rebuild the application. This is highly relevant in the context of digital forensics and ethical hacking, where inspecting and controlling app behavior is crucial.

**Project Scope**

The scope of this project spans multiple technical operations related to Android reverse engineering and application modification:

**APK Decompilation**: Using Apktool to convert an APK into Smali code and human-readable resources. This step is essential for exposing the internal structure of the app, including manifest permissions, layout files, and more.

**Permission Identification**: The primary focus is on permissions declared in the AndroidManifest.xml file. By identifying and understanding these permissions (like access to location, contacts, camera, SMS, etc.), one can assess the potential risks or privacy concerns posed by an application.

**Modification of Permissions**: After identifying the required permissions, unnecessary or suspicious permissions are removed or altered. For example, an app requesting SMS access may be modified to eliminate that request.

**Rebuilding APK**: Once the desired modifications are made, the app is rebuilt using Apktool. This process re-packages the app into a valid APK format.

**APK Signing**: Android requires all APKs to be signed. Rebuilt APKs are unsigned by default, so they must be signed using a debug key or a custom keystore to allow installation on devices.

**Testing and Validation**: Finally, the modified APK is installed on an emulator or a physical device. Its behavior is observed to ensure that the permission changes are effective and that the app still functions correctly.

**Use Case Scenarios**: This technique is applicable for digital forensics, app customization, parental control, security auditing, and understanding the behavior of third-party applications.

## ****Project Objectives****

The main objectives of this project are:

**To understand the structure of Android APK files** including their components such as the manifest file, Smali code, resources, and compiled classes.

**To learn how to use Apktool** for decompiling and recompiling Android applications effectively.

**To analyze app permissions** declared in AndroidManifest.xml and understand their significance with respect to security and privacy.

**To modify or remove specific permissions** (e.g., INTERNET, ACCESS\_FINE\_LOCATION, READ\_SMS) from an APK to control the app's access to device features.

**To rebuild and re-sign the APK**, ensuring the app is installable and testable after modifications.

**To evaluate the effectiveness of permission changes** by comparing the original and modified versions in terms of behavior and access requests.

**To provide a practical demonstration** of reverse engineering skills relevant in the fields of ethical hacking, app security testing, and digital forensics.

## ****Project Problems****

During the execution of this project, several challenges and limitations were encountered:

**Obfuscated Code**: Some apps use code obfuscation techniques (like ProGuard) that make reverse engineering difficult by renaming classes and variables.

**Signature Verification**: Many apps verify their signature at runtime to prevent tampering. Modifying and re-signing such apps may cause them to crash or refuse to run.

**Resource Compilation Errors**: After modification, rebuilding an APK may fail due to resource merging or Smali code syntax errors.

**Unsupported APK Features**: Apktool might not fully support certain APK versions or features, resulting in incomplete decompilation or recompilation.

**Legal and Ethical Boundaries**: Reverse engineering can raise legal and ethical issues, especially when applied to commercial applications without permission.

**Compatibility Issues**: Modified apps may not be compatible with all versions of Android, especially if system-level permissions are altered.

These challenges were addressed through trial-and-error, updated versions of tools, and careful analysis of decompiled code and manifest files.

## ****Methodology****

The methodology followed in this project involves a structured, step-by-step approach as outlined below:

### Step 1: Setting Up the Environment

Installed **Java Development Kit (JDK)**, **Apktool**, and **keytool**.

Prepared a test APK file for reverse engineering.

Created a working directory for managing decompiled and rebuilt files.

### Step 2: Decompiling the APK

Command used:

apktool d appname.apk -o appname\_src

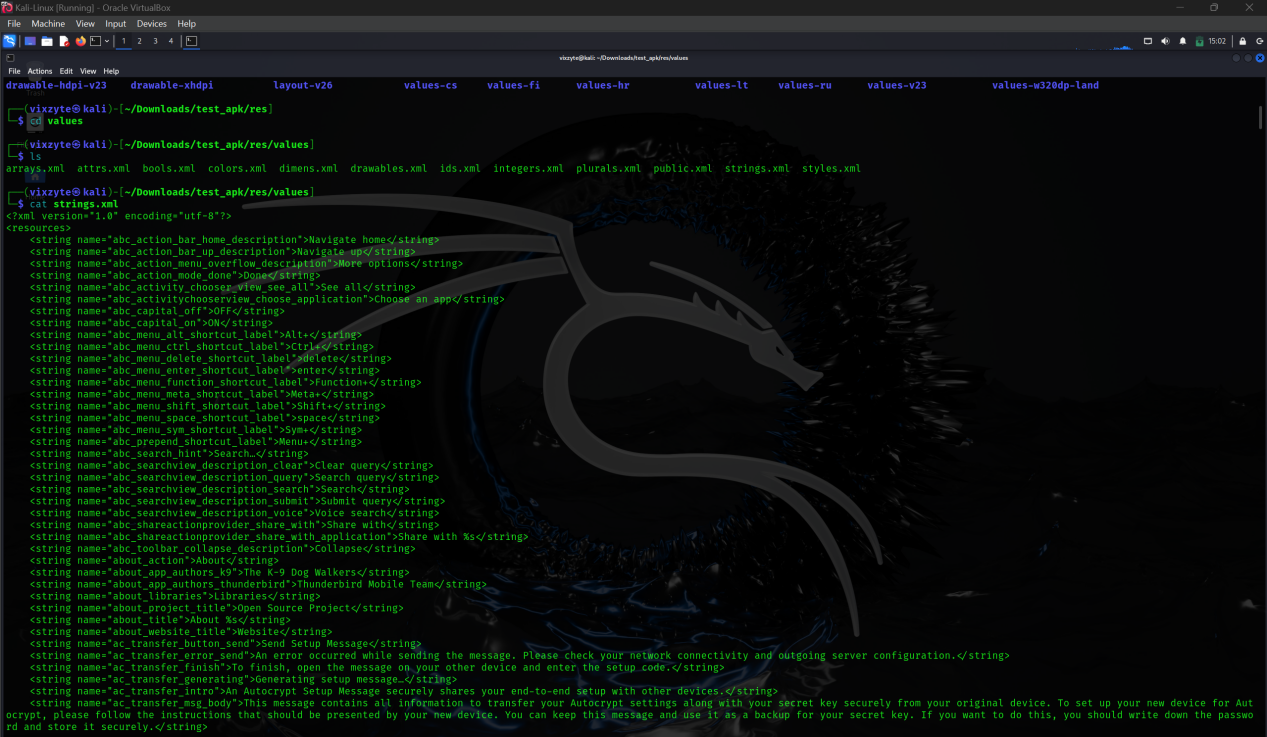
This produced a directory containing AndroidManifest.xml, Smali code, and resource files.



### Step 3: Modifying Permissions

Removed or commented out undesired permissions from the manifest file.

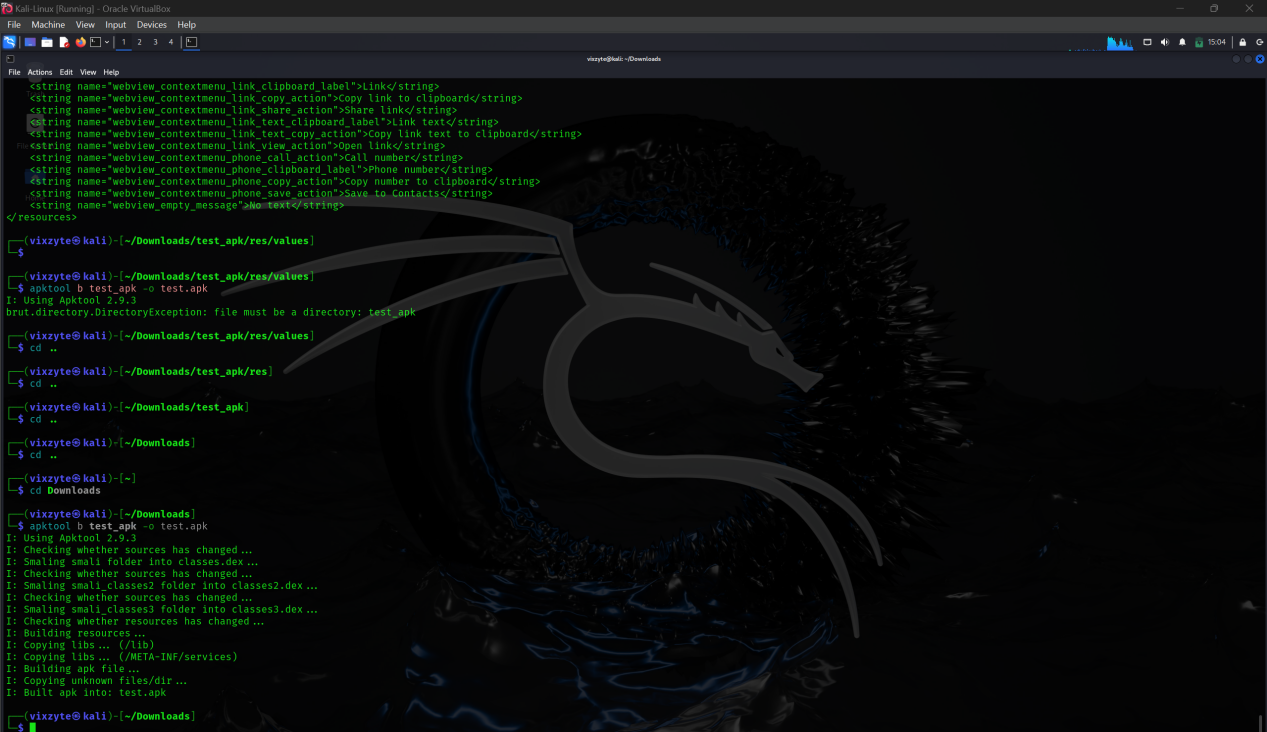
Saved the changes and ensured no structural issues were introduced.



### Step 5: Rebuilding the APK

Command used:

apktool b appname\_src -o appname\_modified.apk



### Step 6: Signing the APK

Generated a keystore (if not already present) and signed the APK:

keytool -genkey -v -keystore my-release-key.keystore -alias alias\_name -keyalg RSA -keysize 2048 -validity 10000

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore appname\_modified.apk alias\_name.

And finally testing the apk.

## ****Techniques****

The primary techniques and tools used in this project include:

### ****Apktool****

Tool used for decompiling and rebuilding APKs.

Allows inspection of Smali code and XML resources.

### ****Smali Code Analysis****

Smali is the human-readable form of Dalvik bytecode.

Useful for understanding internal logic when source code is unavailable.

### ****AndroidManifest Editing****

Direct modification of manifest file to change permission requirements.

### ****Keytool & Jarsigner****

Tools used to generate and apply digital signatures to APK files post-modification.

### ****Android Debug Bridge (ADB)****

Used to install and test APKs on devices and emulators.

### ****Ethical Hacking Techniques****

Permission analysis and modification simulate real-world ethical hacking and security auditing scenarios.

## ****Conclusion****

This project successfully demonstrated how reverse engineering can be applied to analyze and modify Android APKs using Apktool. The process of decompiling, analyzing, editing permissions, and recompiling provided deep insights into how Android applications are structured and how they can be controlled.

Modifying permissions is a powerful technique in scenarios where user privacy is a concern or when testing apps for security flaws. This project not only enhanced practical skills in Android reverse engineering but also emphasized the importance of ethical responsibility and legal considerations in such activities.

The techniques learned can be further extended to malware analysis, app customization, and automated security auditing tools. With the increasing number of mobile applications and their access to personal data, reverse engineering remains a vital skill in cybersecurity and digital forensics.