**Project Requirement Document (PRD)**

**Project Title:**

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

**Version:**

*Commercial Edition / User Edition*

**Date Created:**

*18/05/2025*

**Prepared By:**

*Sai Datta Badam*

**1. Overview**

**1.1 Project Summary**

This project focuses on reverse engineering Android applications (APKs) to analyse and automatically modify security-sensitive aspects such as **permissions** and **user interface (UI) strings**. The goal is to detect and remove potentially **dangerous permissions**, **ad libraries**, and **trackers**, while also customizing or obfuscating UI text as per user-defined rules.

The solution involves:

* Decompiling APKs using **Apktool** to gain access to source resources like AndroidManifest.xml and layout files.
* Parsing and modifying **AndroidManifest.xml** to remove or comment out sensitive permissions like ACCESS\_FINE\_LOCATION, READ\_SMS, etc.
* Editing XML files in the res/layout directory to update android: text attributes, using a user-defined mapping.
* Rebuilding the modified APK and re-signing it using **apksigner**, ensuring the application is installable and runs correctly.

A user-friendly Python script automates this entire workflow, enabling security-conscious users to vet and clean third-party apps before installation.

**1.2 Background and Context**

In today’s digital ecosystem, **Android applications** are ubiquitous and play a critical role in everyday user experiences. However, many of these applications request **excessive permissions**, integrate **advertising SDKs**, and collect **user data**, often without user awareness. This raises serious **privacy**, **security**, and **ethical** concerns.

**Reverse engineering** APKs (Android application packages) allows researchers, security professionals, and developers to inspect app behaviour, audit code, and gain insights into how applications handle user data. Tools like **Apktool** enable decompilation of apps into a human-readable format where XML-based UI layouts and the manifest file can be analysed and edited.

At the same time, modifying **UI strings** is a valuable tool for security auditors and developers. This can be used to:

* **Redact or anonymize labels** that may contain hardcoded sensitive content.
* **Customize the user interface** for specific contexts like internal enterprise use.
* Conduct **localization** or **obfuscation** for testing or compliance purposes.

This project was born out of the need to **automate the process** of reviewing, cleaning, and modifying APKs for safer usage. By building a script-driven toolchain to modify permissions and UI strings in decompiled APKs and then repackage them safely, this project aims to **empower users with greater control** over their Android app environments.

This project lies at the intersection of:

* **Cybersecurity**: Analysing permission misuse and minimizing attack surfaces.
* **Software Engineering**: Automating build pipelines for APKs.
* **Reverse Engineering**: Understanding and modifying app internals without source code.

**1.3 Stakeholders**

The successful execution of this project — focused on reverse engineering Android APKs to automatically modify permissions and UI strings — involves a diverse set of stakeholders. Each plays a vital role in contributing to the project’s objectives and its overall impact on privacy, security, and mobile app control.

**1.3.1 Organizations and Cybersecurity-Focused Businesses**

Organizations and businesses dealing with Android applications or user data have a vested interest in such reverse engineering projects. These stakeholders aim to:

* Audit third-party apps for compliance with privacy policies.
* Ensure applications don't misuse user permissions or contain dangerous trackers.
* Evaluate the ethical implications and security risks in Android ecosystems.

Key internal stakeholders include:

**1. System Administrators and IT Technicians**

* **Role:** They handle the deployment and maintenance of mobile applications in enterprise environments.
* **Interest:** They monitor app behaviour for potential misuse of permissions and verify whether modified APKs meet organizational security standards.
* **Contribution:** They may initiate APK scanning, track suspicious app activity, and collaborate in deploying cleaned versions.

**2. Internal Auditors and Compliance Officers**

* **Role:** Ensure apps adhere to internal privacy policies and external regulatory frameworks like GDPR.
* **Interest:** They are interested in tools that help expose unauthorized data collection or policy violations within apps.
* **Contribution:** May validate the modified APKs and use them for compliance audits or security certifications.

**3. Developers and Reverse Engineers**

* **Role:** Core executors of the project.
* **Interest:** Students and professionals involved in dissecting APKs, modifying permissions, automating UI string changes, and rebuilding functional apps.
* **Contribution:** Develop and refine automation scripts using tools like apktool, aapt, and Python-based XML parsers. They drive innovation in app reverse engineering techniques.

**4. Employees and Whistleblowers (in larger org contexts)**

* **Role:** Report potential misuse of apps in corporate settings.
* **Interest:** Ensuring apps do not collect unauthorized information or violate privacy policies.
* **Contribution:** May identify the need for APK analysis tools like this project and raise concerns that initiate forensic-like APK audits.

**5. Forensic Interns and Entry-Level Cybersecurity Analysts**

* **Role:** Assist in static and dynamic analysis of APKs.
* **Interest:** Gain practical experience in mobile app security, reverse engineering, and data privacy enhancement.
* **Contribution:** Help prepare datasets, run analysis tools, organize string mapping and permission logs, and support testing and rebuilding efforts.

**6. Faculty Advisors / Academic Mentors**

* **Role:** Guide and oversee the project.
* **Interest:** Ensure the learning objectives are achieved and technical goals are met ethically and effectively.
* **Contribution:** Provide feedback, academic resources, and validation support during development and review.

**7. External Reviewers / Industry Experts**

* **Role:** Evaluate the project during final reviews.
* **Interest:** Assess the practical relevance, industrial applicability, and technical sophistication of the solution.
* **Contribution:** Offer constructive feedback, assess real-world potential, and recommend improvements or future directions.

## ****1.4 Scope****

The scope of this project encompasses the **reverse engineering of Android APK files** to enable **automated modification of app permissions and UI strings**, with the primary goal of enhancing **transparency, privacy control, and usability**. The project focuses on providing both technical feasibility and practical utility for cybersecurity enthusiasts, auditors, and developers who wish to analyse or sanitize third-party Android apps.

**🔧 In-Scope Activities**

1. **Decompiling Android APKs:**
   * Use of tools such as apktool to decode APK files into a readable and editable form (e.g., smali code, XML files).
2. **Extraction and Identification:**
   * Automatically identifying dangerous or privacy-invading permissions in the AndroidManifest.xml file.
   * Extracting UI strings (like buttons, labels, and text hints) from res/layout and res/values directories.
3. **Automated Modification:**
   * Removing or replacing dangerous permissions (e.g., READ\_SMS, ACCESS\_FINE\_LOCATION).
   * Replacing specific UI strings that may include misleading or brand-specific content with neutral or customized alternatives.
4. **Rebuilding and Signing APKs:**
   * Repackaging the modified APKs and applying custom keys to enable installation and use on Android devices.
5. **Automation Scripting:**
   * Implementing Python-based scripts to automate the analysis and modification process, minimizing human effort and error.
6. **Documentation and Demonstration:**
   * Providing clear documentation of the tools, steps, and outcomes.
   * Demonstrating the effectiveness of the tool on real-world APK samples.

**🚫 Out-of-Scope Activities**

1. **Dynamic Analysis or Runtime Monitoring:**
   * This project does not analyse the app's runtime behaviour (e.g., network traffic, logs, etc.).
2. **Malware Detection or Classification:**
   * The tool does not aim to classify malware or conduct threat intelligence, but it may complement such tools.
3. **App Store Integration or Auto Updates:**
   * It does not publish apps to stores or manage versioning automatically.
4. **iOS App Analysis:**
   * The scope is limited to Android APKs; iOS .ipa files are not covered.

**2. Objectives and Goals**The primary objective of this project is to design and implement an automated solution that allows users to **decompile Android APKs, identify and remove sensitive permissions, and modify UI strings** to improve privacy, security, and usability. The tool aims to empower researchers, cybersecurity professionals, and developers to inspect and sanitize APKs before installation or analysis.

**🎯 Primary Objectives**

1. **Reverse Engineer Android APKs:**

* Develop a reliable pipeline to decompile APKs into human-readable formats using tools like apktool.

1. **Detect Dangerous Permissions:**

* Automatically scan AndroidManifest.xml for permissions that are classified as dangerous, such as access to contacts, location, messages, etc.

1. **Modify UI Strings:**

* Extract hardcoded UI strings from layout XML files and replace them with user-defined or neutral alternatives to remove branding, misdirection, or privacy-invasive language.

1. **Automate Rebuilding and Signing:**

* Rebuild the modified APK and sign it using apksigner or uber-apk-signer, ensuring it remains installable on Android devices.

1. **Develop User-Friendly Scripts:**

* Write Python scripts that allow non-expert users to automate the entire modification process with minimal manual intervention.

**🧭 Secondary Goals:**

* **Enable Customization:**  
  Allow users to provide custom keyword maps or regex rules to decide what UI elements or permissions to modify.
* **Support for Batch Processing:**  
  Extend the tool to support bulk modification of multiple APKs, saving time in forensic or enterprise use cases.
* **Logging and Reporting:**  
  Generate a log or report showing what changes were made for auditing and transparency.
* **Educational Utility:**  
  Help learners and cybersecurity students understand how Android apps are structured and how reverse engineering works in practice.

**3. Requirements**

**3.1 Functional Requirements:  
  
  
🔍 APK Decompilation and Inspection**Ability to decompile APKs using apktool or similar utilities.  
  
Extract XML files, resources, and code for analysis.  
  
Preserve original app structure for re-compilation. **🛡️ Dangerous Permission Detection**Identify permissions classified as dangerous in AndroidManifest.xml.  
  
Use pre-defined or customizable lists for permission classification.  
  
Flag or highlight suspicious permissions for removal.

**📝 UI String Extraction and Modification**Extract strings from res/values/strings.xml and layout XMLs.  
  
Replace or obfuscate selected strings using regex or keyword mapping.  
  
Support for multilingual string resources if applicable.  
 **⚙️ APK Rebuilding and Signing**Rebuild modified APK without breaking functionality.  
  
Sign the APK using either debug keys or custom keystores.  
  
Validate that the APK installs correctly on a test Android device or emulator.  
 **📁 Report Generation  
  
Generate logs summarizing:** Permissions removed/altered  
  
 UI strings changed  
  
 APK rebuild success/failure  
 **Optional export in .txt or .pdf format.  
  
  
🔄 Batch Processing**Support for processing multiple APKs in a single run.  
  
Maintain separate logs for each APK file processed.

* 1. **Technical Requirements  
       
       
     🧰 Reverse Engineering Tools**apktool for decompiling and rebuilding APKs.  
       
     uber-apk-signer or apksigner for signing APKs.  
       
     aapt for APK inspection (optional).  
      **🐍 Programming Tools  
       
       
     Python 3.x with libraries like:** os, subprocess – for command execution  
       
      re – for string pattern matching  
       
      argparse – for CLI tool creation  
       
      shutil, json, yaml – for file handling and configuration  
       
      **📊 Logging and Output**Use of logging or loguru Python module for tracking execution.  
       
     JSON-based configuration files to define keywords and replacement rules.  
      **💻 OS and Environment**Linux (Ubuntu/Kali), Windows, or WSL with support for Android SDK tools.  
       
     Compatibility with Python-based cross-platform automation. **🔐 Signing & Verification Tools**Java JDK (for keytool and jarsigner if needed).  
       
     Android SDK platform tools.
  2. **Infrastructure Requirements  
       
       
     🖥️ System Requirements** Minimum Quad-core CPU, 8 GB RAM, 100 GB SSD.  
       
      Pre-installed Android SDK, JDK, and Python environment.  
      **🔄 Version Control and Backup** Use Git for version control and team collaboration.  
       
      Periodic backup of APKs and modified builds.  
       
      **🧪 Testing Environment** Android emulator (e.g., Android Studio AVD) or test device for verifying rebuilt APKs.  
       
        
      Secure, sandboxed environment to prevent malware execution during APK analysis.  
      **🔒 Storage and Access** Local or encrypted cloud storage for APKs and logs.  
       
      Access controls to restrict modification of signed APKs or configuration files.
  3. **Operational Requirements  
       
     👨‍💻 Skilled Developers and Analysts** Basic to intermediate knowledge of Android app structure, XML, and reverse engineering.  
       
      Familiarity with Python scripting and shell commands.  
      **🤝 Collaboration and Review** Peer review of scripts and results to ensure consistency and correctness.  
       
      Use of shared repositories (e.g., GitHub) for team-based contributions.  
      **🚨 Legal and Ethical Compliance** Ensure tools are used for ethical purposes such as security audits, education, or testing.  
       
      Avoid repackaging apps for malicious distribution or copyright infringement.  
      **📚 Documentation and Help** Provide user documentation with step-by-step usage and example cases.  
       
      Include error handling, common troubleshooting steps, and FAQs.  
      **🔄 Continuous Improvement** Modular design to allow new features like malware detection, certificate spoofing alerts, etc.  
       
      Regular updates to permission lists and regex mappings based on Android updates.

**4. Use Case**

### ****4.1 Overview of Use Case****

### The proposed use case revolves around the reverse engineering of Android APK files with the primary objective of enhancing application security and transparency. Android applications often come bundled with excessive permissions and hidden trackers, which can compromise user privacy. This use case tackles that issue by automatically analysing APKs to detect and remove permissions that are unnecessary or potentially harmful, such as those that allow access to the microphone, SMS, contacts, or device location without clear justification. The core of the use case involves decompiling APK files using tools like Apktool to extract and inspect the application’s components, especially the AndroidManifest.xml and associated UI string resources. Once the files are in a human-readable format, the system scans for dangerous permissions and misleading or suspicious strings that may be linked to advertising SDKs, trackers, or unclear user prompts. This enables the tool to flag or directly modify elements that violate best practices for mobile app security or ethical user experience design. After analysis and modification, the system rebuilds the APK and re-signs it using a secure custom keystore. This allows the modified application to be installed on devices for further testing or forensic analysis, preserving the functional integrity of the app while ensuring it no longer includes the removed threats. In parallel, a detailed report is generated to summarize all changes made, including which permissions were removed, which strings were altered, and hash values for both the original and modified APKs to ensure forensic traceability. This use case is particularly valuable in real-world contexts such as mobile app auditing, digital forensic investigations, security testing, and regulatory compliance checks. It can be used by cybersecurity teams to audit third-party applications before deployment, by forensic analysts to examine apps involved in cybercrimes, or even by privacy-conscious users to sanitize APKs before installation. Ultimately, this automated reverse engineering and modification approach offers a powerful way to safeguard users and organizations from hidden threats within mobile applications.

### ****4.2 Scenario Description****

**Background:**  
Modern Android apps often request excessive permissions or include misleading UI elements that compromise user privacy and security. Manually analyzing such APKs is time-consuming, especially for cybersecurity researchers and forensic experts. This project proposes an automated tool that assists in reverse engineering APKs to uncover security issues, modify suspicious elements, and generate secure versions of the app.

**Objectives:**

* To automate the decompilation, analysis, and modification of Android APKs.
* To identify and remove or flag dangerous permissions in the AndroidManifest.xml.
* To scan and replace UI strings that are deceptive or related to trackers/ads.
* To rebuild and sign the modified APK for secure reinstallation.
* To generate an audit log/report for documentation and traceability.

**4.3 Forensic Data Analysis Process**

The project’s forensic analysis pipeline involves the following steps:

1. **Data Acquisition:**
   * The APK file is uploaded as input.
   * Metadata and manifest data are extracted using tools like apktool.
2. **Decompilation & Inspection:**
   * The APK is decompiled into human-readable formats (smali, XML, and assets).
   * Permissions and UI strings are programmatically extracted.
3. **Permission and String Analysis:**
   * AndroidManifest.xml is scanned for high-risk permissions (e.g., READ\_SMS, RECORD\_AUDIO).
   * UI strings are matched against predefined keyword sets to identify trackers or misleading terms.
4. **Modification:**
   * Dangerous permissions are removed or commented out

**4.4 Expected Outcomes  
  
Modified APK:** A safer, user-reviewed version of the original app with unnecessary permissions removed and UI strings clarified.  
  
**Automated Report:** Includes the permissions that were removed, the strings changed, original vs. modified hash values, and actions taken during processing.  
  
**Forensic Integrity:** Hash-based validation ensures that both the original and modified APKs maintain a verifiable chain of custody.  
  
**Reusability and Extensibility:** The tool can be adapted for large-scale analysis of multiple APKs in forensic or audit labs.

### ****4.5 Real-World Application of the Use Case****

* **Mobile App Security Testing**  
    
  Security analysts can use the tool to vet third-party APKs before installation on enterprise devices.
* **Digital Forensics and Incident Response**  
    
  During cybercrime investigations, forensic teams can reverse engineer malicious APKs to understand the attacker’s intent and methods.
* **User Privacy Audits**  
    
  Privacy-conscious users or regulators can use the tool to review apps for hidden trackers or data-leaking components.
* **Education and Research**  
    
  Cybersecurity students and researchers can study the internals of Android apps in a controlled lab setting without needing to write disassembly code manually.
* **Regulatory Compliance Verification**  
    
  Organizations subject to GDPR, HIPAA, or data privacy laws can use this tool to ensure apps comply with privacy norms before deployment.

1. **Technical Requirements**

### ****5.1 Technology Stack****

The technology stack for this project encompasses various domains such as cybersecurity, automotive engineering, and data analysis.   
  
Below is a breakdown of the tools and technologies that will be employed in this project to ensure a comprehensive approach to securing autonomous vehicles:

#### ****Digital Forensic Tools:****

|  |  |
| --- | --- |
| **Tool/Technology** | Purpose |
| Autopsy (optional use) | For integrating mobile forensics in future enhancements or full device data extraction if needed. |
| Volatility (optional) | Memory forensics tool used in broader forensic investigations, if app behavior needs runtime tracing. |
| Log File Analyzers | For reviewing generated logs and access trails during APK modifications. |
| Regex Engines / String Matching | Used within Python scripts to detect and replace suspicious UI strings |
| SQLite Browser | To inspect databases used in Android apps for any embedded trackers or sensitive data. |

1. **Cybersecurity Tools:**

|  |  |
| --- | --- |
| Tool/Technology | Purpose |
| MobSF (Mobile Security Framework) | For static and dynamic analysis of APKs. Helps in identifying security issues, tracker libraries, and permission abuse. |
| AndroGuard | Python-based tool for reverse engineering and analyzing Android apps, useful for scripting and automating inspection. |
| APKTool | To decompile and recompile APKs. Allows us to inspect and modify manifest files and resources. |
| keytool/jarsigner | For signing the modified APKs using a custom keystore, ensuring integrity and installability. |
| Frida (optional) | For dynamic instrumentation and runtime analysis (if future enhancements involve behavior monitoring). |
| SHA-256 Hash Generator | To verify evidence integrity and ensure that modified APKs can be traced back to the original version. |

1. **Programming and Automation Tools:**

|  |  |
| --- | --- |
| Tool/Language | Purpose |
| Python | Core language used for scripting automation, analysis of XML/UI strings, and log generation. |
| Libraries: os, shutil, re, hashlib, xml.etree.ElementTree, subprocess | Used for file manipulation, regex scanning, hash generation, XML parsing, and invoking tools. |
| Bash / Shell Scripts | Used for executing toolchains and command-line automation in Linux environments. |
| VS Code / PyCharm | For code development and debugging. |

1. **Operating Systems and Environment**

|  |  |
| --- | --- |
| OS / Platform | Purpose |
| Kali Linux / Ubuntu | Preferred platforms for cybersecurity tools and scripting environment. |
| Windows (optional) | For running tools like APKTool and Java-based signing utilities. |
| Android Emulator / VirtualBox | For testing modified APKs in a safe, sandboxed environment. |

**6. Timeline and Resources**

#### ****6.1 Project Timeline****

The project titled **"Reverse Engineer Android APKs and Auto-Modify Permissions/UI Strings"** is designed to be completed within **6–7 working days**, following a well-structured schedule:

| **Day** | **Task** |
| --- | --- |
| **Day 1** | Project planning, tool installation, and environment setup (MobSF, APKTool, Python). |
| **Day 2** | Data collection: APK acquisition and static analysis using MobSF. |
| **Day 3** | Reverse engineering: Decompiling APKs, modifying AndroidManifest.xml and UI strings. |
| **Day 4** | Automation scripting in Python for detecting & replacing malicious strings and permissions. |
| **Day 5** | Rebuilding APKs, signing, and verifying output. |
| **Day 6** | Testing on emulators, debugging, and refining outputs. |
| **Day 7** | Final documentation, report generation, and presentation preparation. |

#### ****6.2 Resource Allocation****

A **2-member team** is ideal to ensure the timely and successful delivery of the project. The resource allocation will focus on the following:

**1. Good System Configuration**

* Minimum of **8 GB RAM**, **Quad-core CPU**, and **SSD storage**.
* Required to run Android emulators, APKTool, and data analysis tasks efficiently.

**2. Stable High-Speed Internet**

Required for:

* + Tool downloads and updates.
  + GitHub repository access.
  + Real-time collaboration.
  + Testing via cloud/emulated platforms.

**3. Defined Team Roles**

| **Role** | **Responsibility** |
| --- | --- |
| **Member 1** | Reverse Engineering & Automation Developer |
| **Member 2** | Testing and Documentation Lead |

**Member1:**

1. Set up the working environment (MobSF, APKTool, Python, Java).
2. Perform APK decompilation and reverse engineering.
3. Develop Python scripts for detecting and modifying dangerous permissions and UI strings.
4. Handle rebuilding, signing, and verifying APK’s after modification.’

**Member2:**

1. Test modified APK’s on emulators and devices for functionality and integrity.
2. Identify and report bugs or build errors.
3. Document the entire process (tools used, workflow, issues, and solutions).
4. Prepare the final project report and presentation (including visuals and findings).

**7. Approvals and Signoffs**

| **Role** | **Name** | **Signature** | **Date** |
| --- | --- | --- | --- |
| **Project Manager** | Mohammed Zeeshan Farooq | \_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_ |
| **Client/Stakeholder** | *(To be filled)* | \_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_ |