

Automated Static Analysis of APKs

Subtitle if required

By

Alex Javier Garcia Chiquito

Submitted to

The University of Roehampton

In partial fulfilment of the requirements
for the degree of

BACHELOR OF SCIENCE IN COMPUTING

Declaration

I hereby certify that this report constitutes my own work, that where the language of others is used, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of others.

I declare that this report describes the original work that has not been previously presented for the award of any other degree of any other institution.

Alex Javier Garcia Chiquito

Date: 23/04/25

Signed : Alex Javier Garcia Chiquito

Acknowledgements

Here it is customary to thank the people who have supported this work and your studies in general. It is up to you who you thank!

Thank you very much to my supervisor Mastaneh Davis who has been very supportive with the Final Year Project and has given a lot of great ideas to add to the final artefact and has ensured that I stay on track with the meetings.

Abstract

GUIDANCE: Up to 300 words

Analysing android package files typically consumes a lot of time if manually performed which may result in a lot of human error which makes it challenging for developers and security professionals to gain access to accurate and reliable results. This project developed a tool that automatically analyses APKs for security issues and hidden sensitive data to not only save time but gain reliable and accurate results. OWASP's Mobile Top 10 research and android security studies indicated the demand of easier and automated tools as existing tools such as MobSF and Apktool have drawbacks when decompiling APKs. The tool was developed as a web application with Python (Flask), HTML 5, Bootstrap 5, and JADX alongside Java. It features a drag and drops interface with automatic analysis and CSV report generation which displays the extracted components which are permissions, activities, providers, services, API Levels, signatures and hardcoded sensitive strings. It has real-time dynamically updating permissions through web scraping the official android manifest website containing the official descriptions and names of all permissions known currently. The testing of the tool showed that it performed well both online and offline and was user friendly for non-technical users.

Table of Contents

1. Introduction	vi
Research Question or Problem statement	vi
Aims	vi
Objectives	vi
Legal, Social, Ethical and Professional Considerations	vi
Background	vii
Report overview	vii
2. Literature or Technology Review	ix
3. Methodology	xii
4. Implementation	xv
5. Results	xviii
Evaluation	11
Related Work	12
6. Conclusion	xix
Reflection	xx
Future Work	xix
7. References	xxi

8.	Appendices.....	xxii
----	-----------------	------

1. Introduction

GUIDANCE: Up to 1000 words

This project aims to develop an automated system which allows for the forensic analysis of android packages to scan for security vulnerabilities and sensitive information which could be hidden within them without the need of manual intervention which can result in time consuming, inaccurate and unreliable results. This project aims to deliver this automated system within a web-based application based on Flask that automatically scans and decompiles the android packages which are given via user input to then filter out important information regarding sensitive data and malicious code.

Problem Description, Context and Motivation

Android packages often tend to contain security vulnerabilities and hardcoded sensitive information that can be exploited by attackers. The problem being addressed here is the fact that forensic analysis of android packages now requires manual input which is time consuming can lead to inconsistency and human error. The affected include security analysts, forensic investigators, developers, studios and the users themselves who unaware of these vulnerabilities may come across insecure applications. The majority of these problems occur during the software development cycle whether it is during the development of an application where sensitive information could be left hardcoded, before the deployment of the application where the application has not been tested appropriately and post deployment where forensic analysts will need to assess the android packages manually and users install applications which may be insecure due to the problems mentioned earlier. It is important to eliminate the need for manual input when performing forensic analysis of android packages as it will save the time of the affected, reduce human error allowing for more consistent and accurate results and ensure that security vulnerabilities can be detected before they can be exploited by attackers.

Aims

1. Develop an automated system which will perform a forensic analysis on an android package to detect security vulnerabilities and hardcoded sensitive information.
2. Generate a report which will provide the permissions, activities and hardcoded sensitive data from the android package for risk mitigation strategies.
3. Provide an efficient forensic analysis tool that will require no manual input from users and will allow for accurate and detailed results.

Objectives

1. Perform researching and evaluate currently existing tools and techniques used for the forensic analysis of android packages.
2. Design the system which will automate the vulnerability detection within the android packages.
3. Develop a user-friendly interface and allow users to generate a report detailing all the permissions, activities and hardcoded sensitive information from the android package.
4. Test the final product against APK samples to evaluate the accuracy and efficiency of the system.

Legal

The project examines android packages which may contain confidential information which can raise legal concerns regarding the ownership and protection of data. Taking account of the legal requirements, this project aims to maintain legality by processing android packages and decompiling

them on the user's local machine which prevents sharing unauthorised data. Users are advised to look for permissions from the APK authors prior to initiating an analysis to adhere to copyright regulations. The tool uses currently available open-source command line tool JADX under its license and Java which is also needed to run the tool locally.

Social

The project aims to help society by making android applications more secure which in turn keeps users safe from potential data breaches and malicious applications. It allows non-technical users to perform an automated forensic analysis of an android package which makes security practices available to everyone whether online or offline. However, if the tool is misused to analyse android packages without permission it could hurt developers or studios so clear rules for its intended use must be created to encourage responsible use.

Ethical

Ethical concerns are the potential for the tool to be utilised in reverse engineering applications or exploiting weaknesses. To prevent this, the project features ethical use warnings and encourages transparency in analysis practice. The tool provides security of user data through secure file handling and timestamped storage to prevent session conflicts.

Professional

This project follows professional standards in software development. It uses secure coding practices, like validating file names to avoid path traversal attacks and is modularly designed so that it can be maintained easily. The tool fulfils the industry standards for good security analysis which will be useful for cybersecurity analysts and mobile application development. Collaboration with open-source tools such as JADX ensures that it follows professional standards in terms of transparency and contribution.

Background

The growing complexity of android applications and heightened cyber-attacks confirm the need for automated forensic tools. There are active solutions which involve virus total, MobSF and Ghidra that already have strong analysis but usually need more advanced technical knowledge and various forms of manual intervention, hence not being accessible. This project takes these solutions to the next level by incorporating their strengths into an accessible web-based solution which can be hosted locally and offline if needed which allows for the use of the tool without an active internet connection. Citations off earlier research including MobSF's automated analysis, help inform system design by solving already identified problems in mobile application security. The tool focuses on cybersecurity, application development, and digital forensic domains, where accurate analysis is most important.

Report overview

This report is built up of the following segments:

Literature review – This section focuses on outlining the different methods used by already existing tools through research and manual analysis of APK files. This also provides guidance to the design choices used to develop and evaluate the project's forensic analysis tool.

Methodology – This section outlines the development steps, tools and techniques used in creating and improving the tool. It shows the importance of the different selected technologies used so that the web application can function smoothly and achieve the objectives set.

Implementation – This section describes how the approach was employed in the development of the web-based application as it describes how it was created, its key features such as drag and drop, clean user-friendly interface and csv file generation.

Evaluation and Results – This section examines the performance of the tool through testing its strengths and identifying its weaknesses.

Conclusion – This section summarises what the project has accomplished, ensuring that all goals are achieved and thoughts on the learning journey alongside improvement ideas are listed.

References – This section lists the sources used in the report ensuring that credit is given.

Appendices – This section provides additional information such as the project schedule and meeting notes.

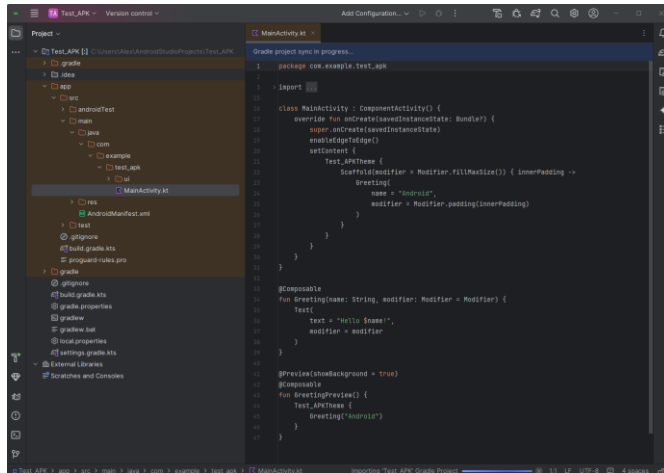
2. Literature - Technology Review

GUIDANCE: Up to 1500 words

Literature/Technology Review

As discussed in several meetings similar tools were evaluated based on how effective they were for APK forensic analysis involving decompilation of APKs, extracting components and identifying vulnerabilities via manual methods. The following tools were examined:

Android Studio/adbtool



Android studio is a development environment which is useful for android application developments as it provides the necessary features to code, debug, design and test the applications through a fully functional emulation station of a range of different devices but it lacks automated forensic capabilities.

```

Command Prompt
C:\Users\Alex>adb
Android Debug Bridge version 1.0.41
Version 35.0.2-12147458
Installed as C:\Android\platform-tools\adb.exe
Running on Windows 10.0.22631

global options:
-a                listen on all network interfaces, not just localhost
-d                use USB device (error if multiple devices connected)
-e                use TCP/IP device (error if multiple TCP/IP devices available)
-s SERIAL         use device with given serial (overrides $ANDROID_SERIAL)
-t ID             use device with given transport id
-H               name of adb server host [default=localhost]
-P               port of adb server [default=5037]
-L SOCKET         listen on given socket for adb server [default=tcp:localhost:5037]
--one-device SERIAL|USB only allowed with 'start-server' or 'server nodaemon', server will only connect to one USB device, specified by a serial number or USB device address.
--exit-on-write-error exit if stdout is closed

general commands:
devices [-l]      list connected devices (-l for long output)
help              show this help message
version           show version num

networking:
connect HOST[:PORT] connect to a device via TCP/IP [default port=5555]
disconnect [HOST[:PORT]] disconnect from given TCP/IP device [default port=5555], or all
pair HOST[:PORT] [PAIRING CODE] pair with a device for secure TCP/IP communication
  
```

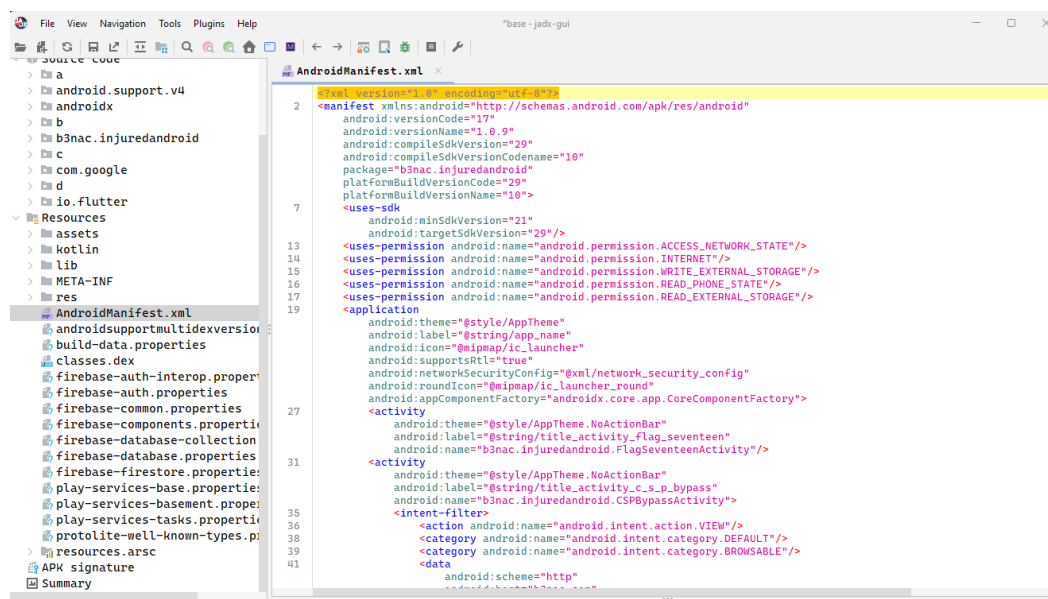
ADB is a command line tool that allows developers to communicate with and manage Android devices or emulators. It is part of the Android SDK and supports tasks like app installation, log capturing, debugging, file transfer, and shell access to the device.

Using Android studio alongside adbtool allowed for the retrieval of a sample APK file from an emulated device using a pull command which will be available below:

```
adb pull /data/app/~~K8ZFEhveJX96IYn7Dmp5RQ==/b3nac.injuredandroid-
_ybjUCp0fCv2FaHesOLTLA==/base.apk "C:\Users\Alex\OneDrive - University of
Roehampton\Documents\app"
```

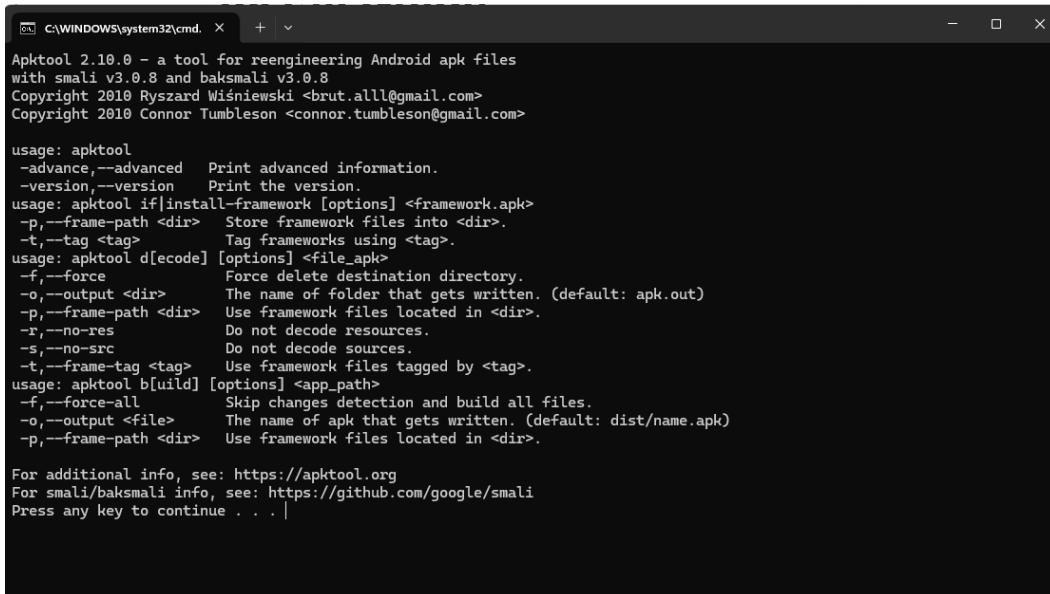
This command allows to retrieve the base sample APK from the application which was installed inside the emulator named **Injured Android** and saves it in the document's directory. Pulling the APK allows for the extraction of its contents through the tools apktool and JADX GUI.

JADX GUI/CLI



JADX GUI is an open-source tool built with java which is useful for decompiling android packages and other file extension types such as jar files. JADX allows for the inspection of content within an APK which is useful for identifying vulnerabilities, malicious code and gives an insight to the internal structure and functionality of the application. JADX GUI was used to obtain the java files within the application to extract hard coded sensitive information manually and review the AndroidManifest.xml which contained information on the permissions and activities.

Apktool



```

C:\WINDOWS\system32\cmd. X + v
Apktool 2.10.0 - a tool for reengineering Android apk files
with smali v3.0.8 and baksmali v3.0.8
Copyright 2010 Ryszard Wiśniewski <brut.all@gmail.com>
Copyright 2010 Connor Tumbleson <connor.tumbleson@gmail.com>

usage: apktool
  -advance,--advanced    Print advanced information.
  -version,--version      Print the version.
usage: apktool if|install-framework [options] <framework.apk>
  -p,--frame-path <dir>  Store framework files into <dir>.
  -t,--tag <tag>         Tag frameworks using <tag>.
usage: apktool d[ecode] [options] <file_apk>
  -f,--force             Force delete destination directory.
  -o,--output <dir>      The name of folder that gets written. (default: apk.out)
  -p,--frame-path <dir>  Use framework files located in <dir>.
  -r,--no-res            Do not decode resources.
  -s,--no-src            Do not decode sources.
  -t,--frame-tag <tag>   Use framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
  -f,--force-all        Skip changes detection and build all files.
  -o,--output <file>     The name of apk that gets written. (default: dist/name.apk)
  -p,--frame-path <dir>  Use framework files located in <dir>.

For additional info, see: https://apktool.org
For smali/baksmali info, see: https://github.com/google/smali
Press any key to continue . . . |

```

Apktool is a reverse engineering tool for android APK files which allows you to decompile them, modify its resources and recompile it once finished. It is particularly useful for analyzing app behavior, troubleshooting issues, or conducting penetration tests on Android applications.

MobSF

MobSF is an open-source tool which allows for the automated security analysis of mobile applications for android and iOS. It gives an insight to the application's API calls and permissions and allows developers and testers to secure their mobile applications by identifying the vulnerabilities before they are exploited.

Ghidra

Ghidra is a powerful reverse engineering tool used for the analysis of compiled applications, and it allows for the identification of security vulnerabilities, viruses and gives one an insight of how the application works without having access to the original source code. Ghidra is very useful for deep analysis however it does require a lot of advanced knowledge, and it is not suitable for the average user.

Normal Permission

Normal permissions allow access to data and actions that extend beyond an application's sandbox but present minimal risk to user privacy and the operation of other applications. By default, the system assigns normal protection levels to normal permissions.

Signature Permission

The system grants a signature permission to an app only when the app is signed by the same certificate as the app or the OS that defines the permission. Applications that implement privileged services, such as autofill or VPN services, also make use of signature permissions. These apps require service-binding signature permissions so that only the system can bind to the services.

Dangerous Permission

Dangerous permissions are permissions which require authentication from the user to run however if abused it can allow malicious applications to access sensitive data on affected devices.

Difficulties encountered when using the apktool command line interface was that it does not fully decompile java source code which is essential in identifying hardcoded sensitive information from the android packages. This limitation meant that JADX GUI had to be used to obtain a readable Java code and to get an insight of the full internal structure of the android package which apktool allowed but as a result of not being able to fully decompile the java code it meant that the manual forensic analysis of the APK's take even longer than necessary as you require to use both tools for an effective analysis. This difficulty was later resolved by integrating JADX and removing the need for apktool as JADX performed a deeper analysis retrieving Java and Kotlin files and more which allowed the analysis to go smoothly and obtain reliable results.

3. Methodology

GUIDANCE: Up to 1000 words

The development of an automated static analysis tool for android package files required a clear structured approach to design, implementation, testing and project management. This section details the methods, tools and processes used to produce the artefact and explains why they were chosen based on what was researched during the literature and technology review. The methodology looks at the project's aim of automating forensic analysis, generating comprehensive report and filtered CVSSs, and providing user accessibility while reducing the problems of older tools such as MobSF, Ghidra and Apktool.

Design

The system was developed as a web application with the Model View Controller pattern. MVC was used because it separates the tasks clearly and makes it both easier to develop and maintain. Previously, there had been a review of tech that considered scale out possibilities such as Flask for web development. The architecture is divided into three components:

- **View:** A minimalistic interface with HTML5, Bootstrap 5, and CSS with a drag and drop file upload system. Special emphasis was placed on keeping it easy for non-technical users, such as security analysts who the literature review describes looking at complex tools such as Ghidra.
- **Controller:** Back-end coding with Flask and Python that deals with file uploads, routing, and arranging of the analysis. Flask was selected as it is lightweight and is compatible with native tools in Python such as JADX CLI.
- **Model:** A file storage solution that keeps user uploads and analysis output time stamped and organised so that the data is separated and the sessions can run all together. This was a preference over database systems as they are less complex and more suitable for temporary storage, as the analysis of MobSF shows concerning its file processing capacity.

Wireframes were used throughout the process of creating a minimalistic and user focused interface with visual cues such as the loading spinners and alert boxes. The literature review identified the issues with manual tools such as Apktool that lack user interfaces as an example of the need for greater accessibility and automation.

Testing And Evaluation

The testing process proceeded step by step to validate how the system performed, its accuracy and how user friendly it is. The testing process involved:

- **Unit Testing:** Smaller components, such as requesting permissions with `xml.etree.ElementTree` were verified to ensure correct parsing of `AndroidManifest.xml` files. Unit tests ensured that permissions were properly labelled as “Normal” or “Dangerous” against the list of official android permissions.
- **Integration Testing:** The step from uploading APKs to the generation of CSV files was verified in ensuring that Flask, JADX and Pandas cooperated effectively. This revealed issues such as encoding problem is decompiled APKs, which were acknowledged as challenges in the implementation.
- **Usability testing:** A “think aloud” test with average users examined how user friendly the interface is, from file upload up to searching for results and previewing CSV files. Feedback resulted in changes to alert messages and the responsiveness of the DataTable functionality.
- **Performance Testing:** The system was tested with APK samples of different sizes to check how quickly it analysed it and how many resources it was consuming. This ensures it is capable of handling more load, a significant improvement from manual tools that took long with large APKs or didn’t accept them at all.

Project Management

The project was organised with an informal and flexible approach. It relied on GitHub and weekly meetings for code storage. Regular meetings occurred and they assisted in making significant progress as the findings and ideas were put into notes. A timeline was provided in the report’s appendices alongside the full list of important notes. GitHub was used to store code, like scripts and interface files all in one place for the project. Key meetings helped with making big decisions such as choosing manifest based analysis instead of Ghidra for simplicity and deciding to use Flask for the web interface. This setup allowed for quick modifications to the issues such as fixing how JADX works on different platforms. It also included feedback to improve features, like interactive CSV previews within the web application.

Technologies and Processes

The technologies and processes implemented were selected based on their strengths identified in the technology review:

- **Python (Flask):** This was used for the backend development since it is simple to use and has powerful libraries which allowed for the smooth integration of JADX and Pandas for decompilation and CSV filtering.
- **JADX:** This was used for decompiling of APKs into readable Java/Kotlin code, resolving apktool’s limitation of incomplete java decompilation identified in the literature review.
- **HTML5, Bootstrap 5, CSS:** These provided a responsive interface that functions on various devices such as desktop and mobile devices building on the command line interfaces of tools such as Ghidra.
- **Pandas:** This enabled for smooth and fast filtering of CSV files and mapping descriptions for permissions, making reports more user friendly compared to the previous manual CSV generation which was present in MobSF.
- **Web Scraping:** Python script using BeautifulSoup to gather the official permissions from android developers’ permission list for real time updates addressing the need of dynamic permission classification.

The process of development followed a modular workflow from requirement gathering from the literature review, iterative design and coding, continuous testing and debugging and documentation via GitHub for version control and artefact access. The modular way of designing and user experience directly tackles the aspect of issues noted in the literature review on the flaws of the manual tools and the demand for a more straightforward forensic analysis.

4. Implementation

GUIDANCE: Up to 3000 words

The web application of the automated tool for analysing android package files converted the planned methodology into a functional web application that simplifies forensic analysis by identifying security vulnerabilities as well as sensitive data that is hardcoded. In this section, the process followed to develop the system is described detailing the development process, key features and how significant challenges were addressed.

Setup and Environment

The development began with setting up the proper environment for the web application and analysis tools. Python 3.13 was set up as the primary programming language, and Flask was installed to handle web requests. The open-source tool JADX CLI was set up to decompile APKs into readable Java and Kotlin code. Code and interface files were maintained on GitHub, which acted as the central repository for project files.

A significant problem was making sure JADX worked properly across different operating systems as some platforms could not find the program for JADX which broke the analysis process. To fix this a python script to check the program's path and set up environment variables automatically was created which also sent any error outputs in a log file which could be inspected by the user at any point. This configuration allowed for reliable decompilation, setting the stage for further development.

User Interface

The user interface was made to be user friendly and easy to use for security analysts and QA testers with different technical abilities. A drag and drop file upload feature, written in JavaScript, allowed users to choose APK files, and preview before scanning the file. The design used Bootstrap 5 for responsiveness, ensuring compatibility across different browsers and devices. Important elements were:

- A loading spinner is displayed during analysis to show progress and avoid user input to prevent duplicate uploads.
- Error messages are set up using Flask's flash () function to notify users of successful or unsuccessful analysis attempts.
- An interactive DataTable, powered by jQuery DataTables to display analysis results with sorting enabled and filtering of permissions, components and sensitive strings.

There was a challenge with irregular styling on mobile phones where components misaligned in smaller screens. This was addressed by refining Bootstrap's grid classes and testing on different mobile resolutions. Feedback from early development highlighted unclear alert messages and recommended a new redesign with straightforward text and error codes for clarity.

Analysis Pipeline

The core functionality was created by implementing an analysis pipeline to examine the uploaded APK files. Files were received via POST requests and stored in timestamped folders to isolate user sessions. File names were validated using `werkzeug.utils.secure_filename()` to prevent security issues like path traversal attacks. A metadata file recorded analysis timestamps and progress and the pipeline also performed the following:

- **Decompilation:** A command was used with JADX CLI to decompile all the APK files to generate Java and Kotlin code along with its resources.

Jadx_path, "-log-level", "ERROR", "-d", output_folder, apkpath

- **Sensitive string detection:** Regular expressions searched decompiled .java and .kt files for patterns like URLs or keywords.

A challenge here was with non-UTF-8 characters in decompiled files which sometimes caused parsing failures. This was addressed by ensuring that the file reading function was modified to ignore errors and allow the pipeline to proceed without interruption which allowed the analysis to go smoothly. Another problem was with complex XML structures in manifests, requiring recursive parsing to ensure all components were captured accurately.

Analysis Reports

Results were saved as CSV files, stored in the user's local folder with a timestamp to enable further analysis in tools such as Excel. The CSV module produced the following results:

- **Apk_analysis.csv:** A full report showing all the permissions, activities, providers, services, signatures and API level.
- **Dangerous_permissions.csv:** A list of every permission found and whether it is dangerous or normal status alongside a description of each permission.
- **Exported_components.csv:** A list of exported components with the android: exported="true" tag.
- **Hardcoded_strings.csv:** A full list of sensitive strings along with line number and file paths.
- **Services_only.csv/Providers_only.csv:** A list of filtered services and providers found within decompilation.

Pandas improved on the CSVs by incorporating permission descriptions using a dictionary mapping done through `df['Description'] = df['Permission'].map(permission_descriptions)`.

ZIP archive of all the CSV files was created with `shutil.make_archive()`. The interactive DataTable displayed CSV data within the browser making it more accessible to users without currently existing tools such as Excel. An issue was ensuring CSV compatibility with excel due to the special characters that could result in formatting issues. This was resolved by implementing UTF-8 encoding with a BOM in the CSV files. The analysis was slowed down by large APKs, so regular expression searching was optimised to bypass non-code files which reduced the processing time.

Dynamic Permissions

To maintain updated permissions, a web scraping script with BeautifulSoup accessed the android developers' manifest permission website and retrieved permission names alongside their respective descriptions when beginning the analysis. Results were stored in a JSON file to avoid runtime delays, and this was proposed in later meetings were real-time integration of updating permissions were

suggested which would make the tool different from currently existing tools and their fixed lists such as MobSF. Issues with varied HTML structures on the permissions page resulted in parsing issues which were addressed by implementing a fallback logic that skips bad entries.

Development Phases

The implementation was split over several phases aligned with the meetings which took place:

Phase 1 – Environment and Scripting: Installation of Python, Flask, JADX and Java which are necessary components needed to run and create a CLI script for manifest parsing and permission extraction.

Phase 2 – Pipeline Development: Establish the analysis pipeline integrating decompilation, manifest parsing, string detection with encoding issues resolved.

Phase 3 – Interface and CSV generation: Designed the user interface and developed the CSV file outputs refining responsiveness and report formats.

Phase 4 – Improvements: Implemented web scraping for real-time permission status updates and descriptions and integrated an interactive DataTable to enhance performance with large APKs.

A challenge was managing concurrent user sessions, resolved by the timestamped directory structure. Another issue was slow analysis with complex APK files which was mitigated by streamlining file scans and maintaining saved information when scanning the same APK file.

The development process utilised a careful, step by step strategy focusing on user experience, performance, and versatility. Each component was tested individually before they were assembled. The program provides a useful, straightforward utility for verifying Android APKs, solving the main issue of minimising the work effort for forensic analysis. This ordered, data-based methodology offers a starting point that can later grow with features like active analysis, threat intelligence API integration, or real time cloud scanning services.

5. Evaluation and Results

GUIDANCE: Up to 2000 words

The automated static analysis tool for APK files was evaluated to see if it facilitates forensic analysis, presents comprehensive reports and if it is user-friendly. It was found from the tests that the tool performed well and effectively and this was possible due to the feedback given during the scheduled meetings. The web application accomplishes the objectives set however as discussed there is still a lot of room for expansion and further enhancements or optimisations.

Testing with a variety of APKs indicated that the permissions were being appropriately labelled “normal” or “dangerous” according to the official android list with little to no errors at all. The entire analysis process, from uploading APKs to creating csv reports functioned properly however upon discovering that if one stopped running the program, and scanned a previously analysed APK, it would present outdated data and sometimes not gathering the permission list along with the descriptions properly.

The performance testing indicated that small APKs were analysed very quickly in a matter of seconds however as one uploads larger APKs for example a 2GB APK, the analysis would last around 10 minutes or less. There was good feedback provided based on the simplicity of the drag and drop feature and quality of life changes meaning that users could download output files with one click and have a variety of different choices to view the csv files with.

The tool runs locally without needing access to the internet which ensures privacy and allows use for it in any location. It allows for fast automated analysis of small and larger android package files which when compared to manual existing tools is an advantage as with tools such as virus total you can't exceed the maximum file limit of 650MB. It allows for the creation of clear CSV reports listing the different components found within decompiled APKs and allows drag and drop features and a sortable table which allows easy access to filter specific keywords.

The web application is also very lightweight it can run on every device which makes it very accessible for every user. However, the web application lacks a variety of different graphical diagrams to display information which would make it more visually appealing and display information much clearer to the user.

The tool is more usable than most mobile security frameworks and can produce clear output and representation with no complex manual analysis. Whilst there are other tools available which may provide deeper analysis, they require a much more advanced knowledge and higher expertise. This tool is very good for ease of use and for non-technical users but still has the potential to improve providing a much deeper analysis of the APK files uploaded by the user.

6. Conclusion

GUIDANCE: Up to 1500 words

The project successfully achieved its primary objectives which was to develop an automated system for forensic analysis which was achieved with a robust pipeline that decompiled APKs, read manifests and searched for sensitive data which eliminated the need for manual automation. The second objective that was achieved was generating comprehensive reports which was achieved through the creation of numerous filtered CSV files using pandas containing permissions, activities, services, providers, hardcoded strings and API signatures and levels. The third objective was also achieved which was to provide a user-friendly tool which was achieved through the creation of a responsive interface with Bootstrap 5 which provided drag and drop uploads and interactive DataTable viewer which could be used by even the non-technical users.

The project allowed for a tool which not only could be accessible online but offline locally without depending on internet access, automated extraction of permission via website scraping to classify and gather the up to date permission list alongside their descriptions to classify permissions whilst analysing, detection of exported components such as services and providers, and lastly offers a built in interactive CSV viewer and a ZIP file download containing all the output files for further analysis using tools such as Excel.

Future Work

The project delivered a functional tool, but it is very much capable of improvement as there are things that weren't able to be fully implemented:

- **Performance Optimisation:** Large APK file scanning took longer than expected which could be addressed in the future via a multi-threaded scan for hardcoded strings or caching decompiled results to make the analysis faster compared to the current single threaded process.
- **Threat Intelligence Integration:** Through APIs like VirusTotal which was discussed in a meeting, it would enhance analysis by comparing APK files with databases of known malware if they have already been submitted before.
- **Cloud Based Deployment:** Local hosting limits scalability. A cloud-based version may make it more accessible for users whilst maintaining data privacy and addressing the project's aim of providing a security forensic analysis tool for everyone to use.
- **Further Vulnerability Detection:** Currently focuses on permissions and hardcoded strings however adding analysis for insecure API calls or incorrect cryptographic settings could provide better insights into understanding app security.
- **Graphical Visualisation:** The current web-based application can be improved through the addition of graphical diagrams representing different data types such as permission counts, activities, providers and services which would allow to make the results easier to understand.

These changes would take more time but in return it would make this a more viable alternative to currently existing tools which require more advanced knowledge compared to a tool that is accessible even to those without a lot of technical knowledge.

Reflection

The experience provided valuable lessons on software development, project management, and forensic analysis. The development of the web-based tool allowed to improve skills in Python, Flask and front-end technologies while resources such as JADX revealed the different components that an Android software is made up of. Dividing the work into phases and weekly meetings provided flexibility and the system was improved with user feedback and the solving of technical challenges which raised during the development of the tool.

The project did well in meeting usability and automation goals due to the different features which made this application user friendly and accessible such as the drag and drop interface and interactive DataTable for users who didn't have tools such as Excel for CSV viewing. There was a limitation for vulnerability detection regarding API call analysis which could make the analysis deeper and a limit file upload size for APKs strictly accepting under 2GB only.

Weekly meetings and storing the files on GitHub allowed to keep everything in line even in the absence of formal management. User testing provided valuable feedback that simplified the interface removing any unnecessary features and adding any quality-of-life features such as ZIP file archive containing all the CSV and generated output log files.

The lack of formal project management tools occasionally made tasks overlap and delayed completion of tasks and being unfamiliar at the start with the reverse engineering tools slowed down the process of setting the environment up when running JADX as previously apktool was being used for decompilation. What would have been best was to use a Kanban board to have tracked tasks and minimise delays.

7. References

The sources below supported the development and evaluation of the automated static analysis tool for android package files:

Android Debug Bridge (adb), Android Developers, 2025: <https://developer.android.com/tools/adb>

Android Studio, 2025. Available: <https://developer.android.com/studio>

Apktool, 2025: <https://ibotpeaches.github.io/Apktool/>

Beautiful Soup, 2025: <https://www.crummy.com/software/BeautifulSoup/>

Bootstrap, 2025: <https://getbootstrap.com/>

Flask, 2025: <https://flask.palletsprojects.com/>

Ghidra, National Security Agency, 2025: <https://ghidra-sre.org/>

JADX, 2025: <https://github.com/skylot/jadx>

Manifest Permissions, Android Developers, 2025:
<https://developer.android.com/reference/android/Manifest.permission>

Mobile Security Framework (MobSF), 2025: <https://github.com/MobSF/Mobile-Security-Framework-MobSF>

Mobile Top 10, OWASP, 2024: <https://owasp.org/www-project-mobile-top-10/>

Pandas, 2025: <https://pandas.pydata.org/>

8. Appendices

To access the web application files and set up guidance follow the link below it will also contain meeting data and notes that were recorded during meetings for project management. The video will also be found within the GitHub.

GitHub Repository: <https://github.com/garciaca1/APK-Analysis>