Final Year Interim Project Report

Full Unit – Interim Report

DEVELOP A SECURITY SUITE FOR ANDROID-BASED SMARTPHONES.

Abd El-Rahman Mohamed Hassan Abdou M Soliman

Supervisor: Christian Weinert

Department of Computer Science

Royal Holloway, University of London

March 28, 2023

Table of Contents

Declaration

This report has been prepared based on my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 16294 words

Student Name: Abd ElRahman Mohamed Hassan Abdou M Soliman

Date of Submission: 28/03/2023

Signature:
[illegible]

1. Abstract

For my final year project, I have chosen to develop a security suite for Android-based smartphones; where my objective is to create an application that implements multiple different modules, with each module handling a different aspect of security on Android devices.

Android security suites allow us to maintain and manage many aspects of our personal device's security from one all-encompassing application. This ability to secure our personal devices is one that is becoming more and more important as we rely more on technology and is an area where multiple industry-leading companies (such as McAfee and AVG) are investing heavily in, to ensure that security keeps up with frequency of vulnerabilities that are found. Throughout this project, I wish to analyse the feasibility of having one security package that handles multiple aspects of security on Android without the need to modify the ROM or Android OS, whilst developing an Android security suite. By the end of the project, I would have hoped to have explored the feasibility of an Android security suite without the need to modify the OS and developed a security suite that handles multiple aspects of security on an Android device. I would also like to be able to explore the use of Google's material design language to ensure that the security suite is user-friendly.

Throughout this report, I will be formally defining my objectives for the project as well as my motivations for the project and why I chose to embark on this project. I will be investigating the feasibility of my project by using various articles and online resources including Android documentation and academic literature that I have overviewed in a series of literature reviews.

I will also be going into detail regarding the development of AsecAV (my Android security suite) where I will be using various open-source Android security projects, and how I managed to implement the various modules into one unified Android security suite including how I tested my application to ensure that it is user-friendly and reliable. There will also be a section discussing any new technologies that I have used of which I consider to be a technical achievement. At the end of this report, I will be using my research and my security suite to evaluate the feasibility of an Android security suite without the need for modifying the ROM or Android OS.

2. Introduction

2.1 Motivations

Since the first release of the Android mobile operating system in 2008, Android smartphones have developed to become an integral part of day-to-day life and as smartphones have improved and added features, the amount of personal data they managed also increased. Android smartphones now can function as anything from a way to digitally sign contracts to complete bank branches in our pocket. They can replace anything from our debit cards to our laptops and as we become increasingly reliant on the personal computers in our pockets, the need for security becomes more and more crucial. Especially now more than ever as new malware is being developed and exploited quicker than exploits can be patched and as our devices become more and more connected to the outside world, an up-to-date fully featured Android security suite will assist in keeping our personal devices secure.

To combat this, various Android smartphone manufacturers have been guaranteeing Android security updates on a monthly, bi-monthly, or quarterly basis for a limited number of years depending on the age of the device. This is beneficial in practice; however different manufacturers choose to stop supporting legacy devices sooner than other manufacturers. For example, Google, which has a 0.5% market share (AppBrain.com, 2023), can promise at least 3 years of security updates across all their devices with their latest smartphones offering at least 5 years (Google Support, 2023) whereas Oppo, with 10.1% market share (AppBrain.com, 2023), uses a “the more you spend, the more you get policy” (C. Scott Brown, 2022) with mid-range and budget models receiving less and fewer security updates as you go down in price with their ultra-budget phones seeing no updates. Maintaining the security and increasing the longevity of these phones is one of my motivations for this project.

For most people, Android already comes with all the essential safety and security features such as encryption and app security and basic malware and anti-virus protection. However, the security level depends on whether the device is running the latest version of the OS. Unfortunately, the Android version market share is very fragmented with only 23.5% of devices running the latest Android 12.0 with more than 27% using a version of Android that is over 3 years (StatCounter GlobalStats, 2023). Naturally, owners of older Android devices look to the Google Play store for a 3rd party security solution such as those provided by McAfee or Malwarebytes and then soon realise that to cover every aspect of security on their device, they would need to download multiple apps as there is no full-featured security suite that can act as an all-in-one solution for your security needs on your device. This inconvenience may deter less tech-savvy users from maintaining the security of their older devices. Having an all-in-one security suite on Android will benefit and allow those users to protect their data more conveniently.

I have taken an interest in this project as I am hoping to pursue a career in the cybersecurity industry with a particular interest in mobile security. I am an avid Android user who has resisted the rise of IOS for many years and have taken an interest in secure messaging and encryption having programmed a secure messaging app (reminiscent of WhatsApp or Messenger) using Java and Google Firebase. Although I’m new to on-device Android security, it has always been a subject that I have wanted to pursue in the future as I do believe that the Play Store’s app requirements are a bit relaxed in the security department and this is a major issue as Android apps keep popping up on the news because of malware detection.

2.2 Aims & Objectives

With this project, my goal is to unite and package multiple existing Android security modules into a do-it-all mobile security suite that lives on the user's Android device of choice and conveniently shows the user an overview of the level of security that their device currently possesses in a user-friendly and understandable way. I aim to achieve this by using open-source implementations of various security features as the foundation that will allow me to develop my fit-for-purpose implementation that can be incorporated into my security suite without requiring a standalone app for each module or modifications to the OS. I aim to present the data being shown by my security suite in an understandable way that doesn't panic the user if not necessary yet will also allow the user to use multiple of the modules listed below with the press of a button to keep their security up to standards. A secondary goal is to have this security suite run automatically in the background, ensuring that all security definitions are up to date for the malware detection algorithm and provide day-to-day security without user input.

This project will analyse the possibility of having an all-in-one security suite on Android that:

- Requires no prior modification to the OS.
- Is user-friendly.
- Is up to date using the latest security definitions for the file scanner.
- Is compatible with older versions of Android.
- Handles multiple modules of Security on an Android device from this list:
    - Anti-Virus/malware Scanner
        - App-based malware scanner
        - File-based malware scanner
    - Overview of app permissions
    - App access control

2.3 Project Specification

Please find project specification provided by Royal Holloway, University of London in Appendix 11.1.

3. Literature Review

To aid the development of my security suite, I needed to do ample research to identify the functional and desirable requirements of the assignment. This initially began with decomposing the task and highlighting the three main features that I planned to implement throughout the project: the malware scanner, the permission manager, and the app locker. For each module, I explored the theoretical insights from various academic papers into the method of accomplishing the component to inform my development.

3.1 Malware Scanner

3.1.1 App-Based Malware Scanner

Most Android app-based malware scanners focus on the permissions aspect of the Android manifest file as each permission may equate to sensitive actions such as the sending of an SMS message or access to the device's contacts and files. Due to these permissions being so easily declared, they can be very helpful in recognising the true intentions of any given app. Permissions are known to be the "best single predictor of the app's malignity reaching the accuracy of about 96%" with its accuracy increasing as we add more metadata into the analysis (Sebastian Hahn et al., 2016). The main advantage of Android manifest analysis is that a practical implementation can be achieved on a device with automatic detection as apps are installed. Another advantage is that it can also scan APKs before the user installs the application. However, static analysis of apps can also lead to many false positives and even malicious apps being missed entirely as although the manifest file declares exactly all the sensitive actions the app is going to perform, the actual behaviour of the app/what it does with those permissions can only be derived from its actual code (Kimberly Tam et al., 2017). Dynamic analysis would be a more accurate way to detect malware without flagging false positives, however this would involve executing the app and observing its behaviour and results. This process is not safe for the user to perform and will take too long to process each application to get a definitive answer as to whether it is malware or benign (Kaijun Liu et al., 2020). Due to the impracticality of a dynamic analysis app-based malware for the reasons stated above, I set myself on looking into an implementation of an app-based malware scanner that performs a static analysis on the Android manifest file of an app.

3.1.2 File-Based Malware Scanner

A file-based malware scanner works by keeping a database of virus definitions that is updated frequently as an increasing number of viruses get profiled. Signature matching is a way of detecting malware using static virus definitions/signatures that have been generated from long sequences of code to minimize false positives that anti-virus software's use to check for presence of a virus' signature in a file (Stefan Katzenbeisser et al., 2011). A virus definition is a binary pattern that identifies a specific virus (TechTerms.com, 2022). By checking files and programs against a database of virus definitions, the malware scanner can determine whether the file/program is malicious or not. This database of virus definitions needs to be updated regularly due to the relentless pace that malware is being developed with at least once a week being the recommended update frequency of the database. Further development of the malware scanner can lead to the ability to generate heuristics allowing the detector to detect unknown viruses just based on their similarity to an existing virus definition and its behaviour (Zahra Bazrafshan et al., 2013). However, I determined that developing heuristic analysis into my security suite was outside of my scope as I lacked the technical knowledge to make this feature a reality. Instead, I decided to implement a file-based malware scanner that uses signature matching to detect malware on the device.

3.2 Permission Manager

Android applications request permissions from users during installation and runtime that the app needs to perform functionalities that require any user information or system resources. Access to these is blocked until the user approves the requested permissions and, most importantly, the app is not allowed to collect any sensitive information until after it has been granted permission (Vera Schmitt et al., 2022). As mentioned above, permissions that are required by an Android application must be listed in its Androidmanifest.xml which is fundamentally an XML file which can be obtained by decompiling the APK file which is the Android application installation package (Kaijun Liu et al., 2020). Fortunately, due to the manifest file being an XML file, it is very easy to parse through and filter the manifest file to find and extract all the permissions the app is using. XML Pull Parser is a common library that is used in Android development that acts as an interface that allows the user to go through an XML file line by line and access each tag/value pair (Stefan Haustein and Aleksander Slominski, 2022). Knowing this, my permission scanner can use the standard syntax for the manifest file to search for tags that contain "uses-permissions" and then extract their value (Google Developers, *Declare app Permissions | Android Developers*, 2022). By retrieving the permissions directly from the manifest file, we can show the user all the permissions an app is requesting, even those that are deemed as minor and as such are not shown to the user when accessing permissions settings on-device.

3.3 App Locker

All Android devices come with built-in locking mechanisms that allow the user to prevent malicious parties from easily gaining access to their device when said parties have physical access to the device that comes in the form of a lock screen (Razvan Stoleriu and Mihai Togan, 2020) . A lock screen is a view that appears upon start-up and every time the screen is woken up that prevents the user from accessing the rest of the device until they authenticate themselves with their pin, password, pattern or biometrics (ComputerHope, 2022). Once the user wakes up their device, they are greeted with their lock screen and apps and the rest of the device.

However, due to how often a user would unlock their phone each day, this is vulnerable to shoulder surfing and smudge attacks where a malicious party would look out for smudges on the screen to reverse engineer the password (Hazleen Aris and Wira Firdaus Yaakob, 2018). This is also vulnerable to user error in the case where the user leaves their phone unattended but unlocked. App locker is the obvious solution to this problem where each app has its own lock screen that the user can set to either require a different or the same password as the main lock screen. This will allow individual apps (chosen by the user) to have an extra layer of security on top of the already built-in lock screen which will ensure that the user's personal data is just that bit more secure. The App locker will also be able to allow the user to set a different password for each application if they so wish to further protect against shoulder surfing and smudge attacks.

4. Background Theory

The need for mobile security is becoming increasingly apparent as the number of mobile devices that are being used increases year after year. Over 70% of mobile devices in the world are powered by the Android operating system (StatCounter Global Stats, 2022) with over 2.8 billion active users potentially being at risk of a security vulnerability. There has never been a larger number of potential vulnerabilities in mobile security than today with how connected personal devices are with the rest of the world. mobile devices can be targeted on an OS, application, or network level. This report will identify and explain potential security apps that could be implemented in a security suite to mitigate the risk of a security breach.

4.1 Pre-existing implementations of an Android Security Suite

All the major computer security software companies have already developed and published some level of a feature-rich and popular Android Security Suite. For example, McAfee (a major leader in this space) has developed McAfee Security: Antivirus App which is their implementation of an anti-malware scanner on an Android device. These implementations tend to not include more niche security features such as an app locker. I used my research into pre-existing implementations to assist me on understanding what security modules are seen to be crucial in a security suite, with a malware scanner being the most crucial, and what features are more niche yet provide a lot of functionality for the user. This is how I settled on adding an app locker and permission viewer into my security suite.

I also chose to use pre-existing implementations such as Avast's and AVG's as inspiration for what my UI should look like and how the user interacts with my application. One of the key decisions that was influenced by pre-existing security suites was what colours to use throughout my application. Most security software companies already have their own well-known brand a long with colours that are heavily associated with that brand. For example, Norton360's colour is yellow whereas McAfee's is red. For my security suite, I wanted to have a colour theme that allowed my application to stand out compared to the big players in the field. Because of this, I settled on a blue and orange colour scheme for my logo.

Another aspect of my user interface that was heavily inspired by research into pre-existing security suites was how the main menu for my application should look like. Below are a series of figures showing the Main page of multiple popular anti-malware applications on Android. In all 4 figures, they share a roughly similar style of having some information or interaction in the top part of the screen followed by multiple large buttons arranged in a grid that is where the user can select what type of scan/feature they would like to run. Based on this research, I chose to base my main menu on this design philosophy by having some information in the main menu and having large buttons for each module of my application.

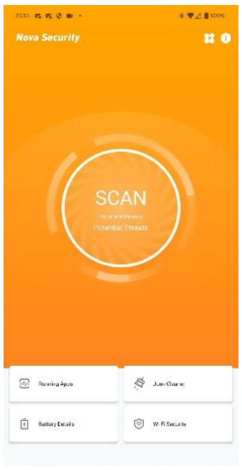


Figure 3, Main menu from Nova security suite on Android

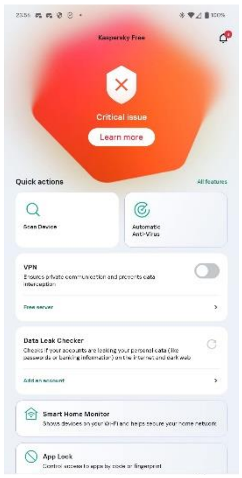


Figure 4, Main menu from Kaspersky security suite on Android

Figure 2, Main menu from AVG security suite on Android

Figure 1, Main menu from Avast security suite on Android

4.2 Malware Scanner

A traditional computer virus works by infecting local files on a particular device and then using that device's resources to spread (Roger A. Grimes, 2020). However, malware on an Android device is not likely to work in this way and is more likely to come in the form of a malicious app with the most common attack vector being repackaging popular benign apps with malicious payloads that the user will install unknowingly (Geoff Duncan, 2022). It is still possible for malware on Android to work in the traditional sense through infected files that pose as benign such as images. This leads to there being 2 different types of malware detection that can be run on Android:

- App-based malware detection
- File-based malware detection

4.2.1 App-Based Malware Scanner

Android is a privilege-separated operating system which fundamentally means that user-installed Android applications are not granted most permissions by default and must obtain more sensitive permissions to interact with OS services, hardware, and even other applications (Kaijun Liu et al., 2020). An Android app must define its requested permissions in its Androidmanifest.xml file. Every Android app has this file which describes any essential information about the application such as its activities, services and what device configurations and features it requires to run. However, the Information that we are focusing on for this scanner is:

- "The permissions that the app needs to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app." (Android Developers Guide, 2022)

We can use this knowledge to extract the permissions that an application is requesting from the device and compare this with its intents and static behaviour to predict whether an app is malicious or benign.

For my security suite, I have gone for an implementation that will use the permissions and intent filters from a given app's Android manifest file to attempt to detect malware. More specifically, I have chosen to implement LibreAV which is an open-source anti-malware application for Android that utilizes machine learning by retrieving datasets of malicious apps and benign apps along with the chosen app's permissions and intent filters and running them through a TensorFlow lite algorithm that will return a number from 0-1 with anything under 0.5 being safe and anything over 0.5 being classed as either risky or as malware (Project Matris, 2021). I am hoping that this prediction system along with showing the user the list of permissions an app is using will be enough to allow the user to make an informed decision as to whether they should keep or delete an app from their devices.

4.2.2 File-Based Malware Scanner

Unfortunately, the Android platform is also vulnerable to more traditional malware from infected files such as documents or images. Android devices can get compromised just by downloading infected/malicious files such as PDFs. In this instance, it can be harder for an average user to be able to tell whether a document they downloaded is malware or not, especially due to Android's default file manager not being as thorough or detailed compared to Windows or Linux. Thankfully, detecting file-based malware is a lot more straightforward than detecting malicious apps.

For this security app, I am using Hypatia which is an open-source implementation of a file-based malware scanner that works by hashing files on the device and comparing the hashes with a local virus definitions database that is being updated regularly. Hypatia is built on ClamAV databases and ESET databases and allows the user to choose which database to install as well as how detailed they would like the database to be. This provides the user with the choice of balancing the size of the database with their phone storage. Hypatia also allows for real-time scanning meaning if a new file was to be transferred or downloaded on the device, it will be scanned instantly and alert the user of its status.

4.3 Permission Viewer

When a user wants to modify the permissions settings for any app on their device, they are unable to change all permissions the app uses with the settings app not even showing the user a comprehensive list of all the permission the app is able to use. It only shows basic permissions that the user can change such as files and contacts' access. This is where a permission manager can be handy. Unfortunately, due to the sandbox protection in Android, unless the device is rooted, the only service that can change permissions for apps is the OS and by extension the settings itself (Daibin Wang et al., 2017). This means that for the user to change all permissions through an external app, they would need to modify the OS of their device. This goes against one of my aims & objectives for this project.

Because of this, I have settled on an app that can show the user all the permissions a particular app is using by grabbing all the requested permissions from the AndroidManifest.xml file and viewing them as a list to the user (Android Developers Guide, 2022). The user can also click on a permission string from the list to find out more information about those permissions. If the user wants to change any permissions, they can press the open in settings app which will directly open the settings page for that app and then the user can change the permissions Android allows them to change.

Although not ideal, this watered-down version of a true permission manager still provides value to the user as it ensures that the user knows exactly how and what permissions each app is using. This could also help the user identify malicious apps on their own. For example, if there was to be a news app installed but it requests the ability to send SMS messages. This can be logically seen as malicious as one can assume that a news app does need to be able to send messages to function (Pham Giang et al., 2015).

4.4 App Locker

An app locker is a form of access control on an Android device that will allow the user to set a lock screen to individual apps to ensure that unauthorised users are unable from accessing sensitive apps by adding an extra layer of security. This feature is especially useful for protecting social media applications where the user is always logged in and isn't verified every time the application is opened.

To implement an effective app locker into my security suite, I would need to implement a background thread that will check at set time intervals whether the user has opened an app on their device by using Android's built-in UsageStatsManager which "provides access to device usage history and statistics" and activity manager which is a class that interacts and provides information about different processes and services (Google Developer, 2022). This will allow my app locker to detect what app the user has opened in real time and then this thread would check this app package name against a list of 'locked' apps and then call the app lock screen before allowing the user to access the app. This list of 'locked' apps will be chosen by the user through the UI of the app locker which will list all the currently installed apps on the device and allow the user to click a padlock icon to lock the app and also allow the user to set different passwords for different apps (Stack Overflow, 2022). Since Android Oreo (8.0), Google has set stricter when it comes to handling background services that could lead to the background thread being terminated (Joe Birch, 2017). A possible solution to avoid this is to schedule a Job Service which is an asynchronous request and a broadcast receiver to restart the service whenever it is detected that the OS has killed it (Stack Overflow, 2022).

4.5 File Encryption

File encryption is a way of encoding data to prevent tampering or unauthorized access. In simple terms, the device uses a complex algorithm to change each individual bit in a file with the intention of making it impossible to work out the original contents of the file without using a decryption key. This ensures that any personal data that was stored on the chosen file is kept secure if a malicious party were to snoop in your device files.

Thankfully, since the release of Android 5.0, the developers of Android began taking encryption a lot more seriously and went as far as to implement support for full-disk encryption until Android 9.0 before switching to a more modern implementation through file-based encryption. Full-disk encryption on Android requires the user to enable the feature in settings and once encrypted, the user would need to input a pin/password/pattern before the device even boots. Once the user is authenticated, the device boots on to Android. From then on, any user-created files are automatically encrypted before being saved and any encrypted data is automatically decrypted when requested (Android Source, *Full-Disk Encryption | Android Source*, 2022). File-based encryption works in the same way but allows different files to be encrypted with different keys that can be unlocked independently which eliminates the need to authenticate the user before boot (Android Source, *File-Based Encryption | Android Source*, 2022).

Since file encryption has existed in Android for many years now, I have chosen not to pursue an implementation of file encryption into the security suite.

5. Technical Achievements

5.1 UI Design

Google Material Design 3 is the latest version of Google's open-source design system that gives developers an in-depth UX guidance and UI component implementation for Android with the intention of providing a consistent "personal, adaptive, and expressive experience" (Google Material Design, 2022).

Material design Is split into 3 main parts: **foundations, styles, and components**. Foundations are a set of standards that define what google considers to a great user interface including standards regarding Accessibility and even how the interface reacts to inputs. It also heavily pushes adaptive design which allows the interface to adapt to the device it is currently running on. Whether that be specific screen sizes or adapting to tablets and even desktops.

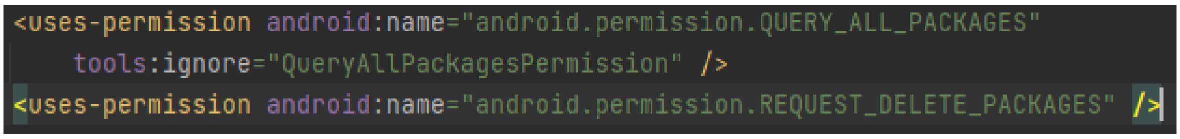
Styles dictates any visual aspects of a UI that allow the UI to have a distinct look and feel. This part of material design controls multiple attributes including:

- Colour – Mainly dynamic colour (keeping colours consistent with system colours)
- Elevation – Gives the UI a slight 3D looks like it's popping out of the screen.
- Icons – Standard buttons for actions such as Play or going back.
- Motion
- Shape – Style of shapes such as roundedness in containers
- Typography – Make writing legible and appeasing to look that.

Throughout my project and, mainly in Term 2, I was working on implementing Google's material design language into all aspects of my security suite to ensure that the user interface is as consistent and friendly as Android allows. In my experience, most anti-malware applications on Android have non intuitive user-interfaces which I found to be discouraging leading to me not opening them as much. I aim to eliminate this issue by simplifying the usage and look of the security suite to make the security suite more user-friendly.

5.2 Permission Requests in Android

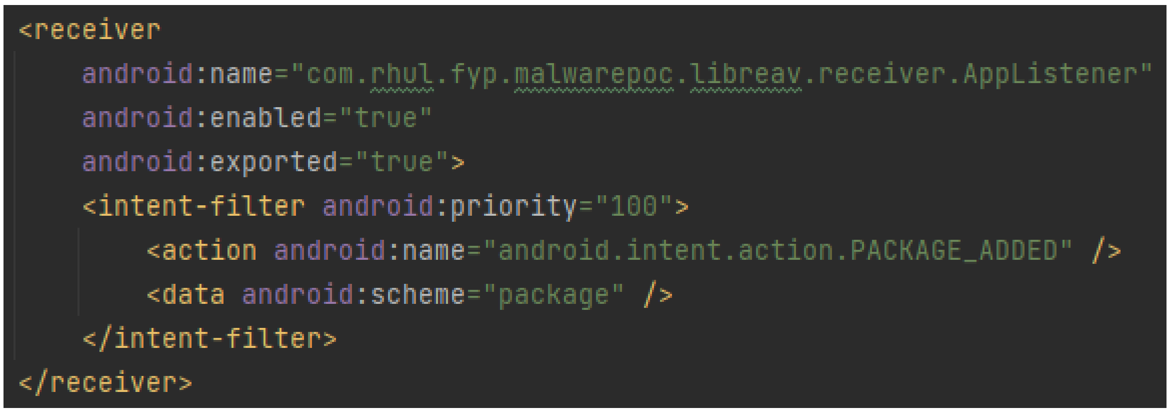
Another aspect of Android development that I have explored and discovered new things about throughout my project is the AndroidManifest.XML. The AndroidManifest.xml is a manifest file in the root of the project that defines all the components, permissions, and intents that an Android app may need. The aspect of this manifest file that I have been focusing on is the permissions. I've had to understand how to use more advanced permissions then what a conventional app may use. A couple of permissions that are commonly used across many of the apps in my security suite are:



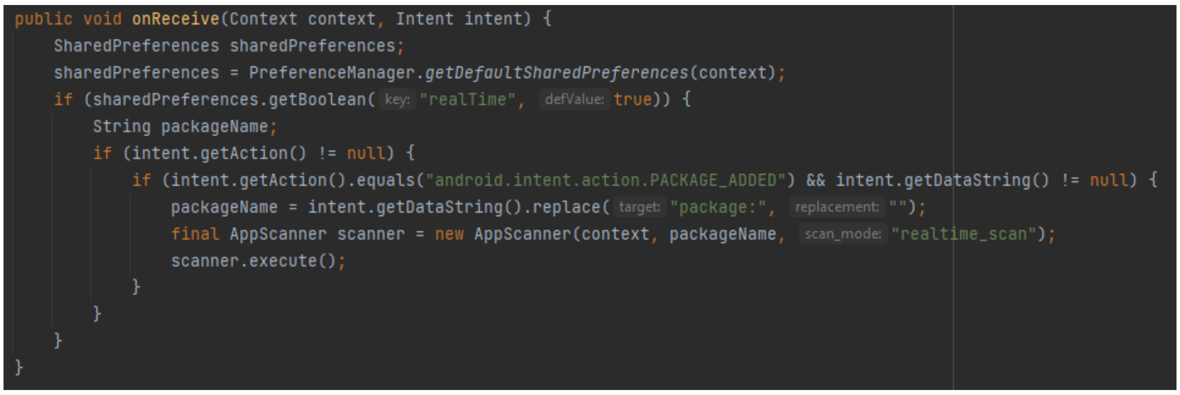
```
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"
    tools:ignore="QueryAllPackagesPermission" />
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES" />
```

The first permissions allow the app to query all the installed apps on the device and access a list of these apps whereas the 2nd allows the app to be able to delete selected apps at the discretion of the user without the need for the user to open the settings app and delete them manually.

I have also had to use different new components of the Android manifest file such as the <service> component as well as the <receiver> component. On Android, a service is like an activity component but lacks a UI that the user can interact with meaning services are usually background tasks or API's that are used by other apps. A receiver component allows the app to receive intents that are broadcast by the OS. An intent in its simplest form is a description of an operation that has been requested to be performed. LibreAV uses a receiver in order to check and thus scan when a new app has been installed.



```
<receiver
    android:name="com.rhul.fyp.malwarepoc.libreav.receiver.AppListener"
    android:enabled="true"
    android:exported="true">
    <intent-filter android:priority="100">
        <action android:name="android.intent.action.PACKAGE_ADDED" />
        <data android:scheme="package" />
    </intent-filter>
</receiver>
```



```
public void onReceive(Context context, Intent intent) {
    SharedPreferences sharedPreferences;
    sharedPreferences = PreferenceManager.getDefaultSharedPreferences(context);
    if (sharedPreferences.getBoolean( key: "realTime",  defValue: true)) {
        String packageName;
        if (intent.getAction() != null) {
            if (intent.getAction().equals("android.intent.action.PACKAGE_ADDED") && intent.getDataString() != null) {
                packageName = intent.getDataString().replace( target: "package:",  replacement: "");
                final AppScanner scanner = new AppScanner(context, packageName,  scan_mode: "realtime_scan");
                scanner.execute();
            }
        }
    }
}
```

5.3 Gradle

Gradle is a build automation tool that is known for its unrivalled flexibility when it comes to building software. The main selling point of Gradle is its ability to automate the creation of applications by handling any added libraries and dependencies and ensuring that the created application contains everything needed to function properly. It is especially popular due to its support for multiple languages ranging from Java to C/C++ and makes the process of building a program consistent amongst all supported programming languages. Gradle can also provide automatic testing for software on several platforms (Simplilearn, 2022).

Using Gradle to compile app resources and code is a heavily encouraged prospect throughout the industry with Android Studio using Gradle for many of its build and testing process by default going as far as starting any new project with Gradle already set up (Google Developers, *Configure your build | Android Developers*, 2022). Because of this, my exposure to Gradle has since severely increased. Since my project focuses on Android devices and Android development, having easy access to a tool such as Gradle in order to handle and install any extra dependencies and ensure that my build is successful and that I have an APK that works has made the technical aspect of my project that much easier as I can focus a lot more on just the code rather than how to build my Android applications correctly.

One of the major struggles I had throughout the development of AsecAV was merging all of the build.gradle files of all the different modules of my security suite. As I was using several different open-source projects that each had their own build.gradle file that declared any specific build options that were required for the module to build, I had to find a way to merge all the build options and settings into one build.gradle file. This proved to be particularly difficult as each module was using a different version of Gradle and had multiple conflicting libraries and their versions. This meant that I had to compare all the build.gradle files line by line in order to integrate the files together. This drastically expanded my knowledge of Gradle as I researched into the build options that I needed to implement for all the modules to build and work properly in one unified application.

5.4 Android Studio

Android Studio is the official IDE or integrated development environment that is well-equipped for fast-paced mobile development while ensuring a high quality of deliverables. It Is the go-to for any developer that is considering creating an app for Android from scratch. All the benefits that are mentioned below are areas of Android studio that I knew very little to none about in my experience and have improved my development process significantly.

Android Studio comes with an emulator that is used to test apps onto virtual Android devices to simulate the success of the application outside of a controlled environment. Crucially for me, Android Studio allowed me to set up Android emulators that were running older versions of Android. One of my objectives for this project is to make my security suite as compatible with as many older Android devices as possible. The emulator solves the issue as to how I would go about testing my security suite on an older device that I may not have physical access too (Google Developers, *Android Studio | Android Developers*, 2022).

Android Studio is also very well designed for developing quick iterations on your projects. For example, if a developer was tweaking a lot of their app's user interface, they would most likely want to see near instant updates on their emulator rather than wait for the app to be recompiled and Android Studio allows this by showing a real static view of the xml design without compiling (Juned Ghanchi, 2021). This was particularly useful throughout my development especially since the build time of my application for it to run on the emulator was reaching the 4-minute mark which meant waiting to see something as simple as a colour change was an inconvenience.

A particular feature of Android Studio that I became more accustomed too throughout my development is the Android profiler. The Android profiler provides real-time data that helped me analyse how intensive AsecAV is on the CPU and memory of a particular device as well as whether my security suite was inefficient and had a drastic effect on battery life. As one of the features of AsecAV was its ability to perform real-time scanning of both apps and files in the background, even if the app is closed. Being able to measure and ensuring that the background services are not slowing down the user's device or drastically reducing their battery life.

5.5 Android Hidden API

Android hidden API is a collection of classes, methods and resources that Google hides from you because of stability reasons or because they are likely to be changed in the next Android API release (anggrayudi, 2020). The Android hidden API was used in MaxLock which was my chosen implementation for an app locker. In this context, the API allows the app locker to react to an app being opened quicker as it is searching for an action that open an app on a System level. This is done by utilising the accessibility features of Android to look out for an app being opened. Once it detects an app loading, it will quickly react by setting a lock screen on the loading app. The Android hidden API is what allows the app locker to utilise the accessibility features of Android.

5.6 Kotlin

Kotlin is a free, open-source programming language designed by Jetbrains and recommended by Google that focuses on Android development however also has a cross-platform layer that allows Kotlin to be used to create native applications for IOS and the web. This is a relatively new programming language, having only been released in 2011 compared to Java, which was released in the 1990s. I did not have any exposure to Kotlin outside of this project as my preferred programming language for Android was Java. However, due to Maxlock (Implementation of an app locker) using a mixture of Kotlin and Java, I was made to use Gradle in its Kotlin form. This meant that I had to become accustomed to Kotlin's syntax and idioms.

A challenge I had come across throughout my project was translating the build.gradle files from using the Groovy programming language to using Kotlin. This was particularly challenging as Gradle was already using a very specific syntax with Groovy to define plugins and build options and I had to convert this to Kotlin using Gradle's syntax for Kotlin. This was a very time-consuming process where I also had to find Kotlin alternatives to some of the functions that Groovy supported. As I was already struggling with Gradle to combine all the build options for every module, having to add a layer of translating to Kotlin made the process a lot more challenging.

6. Software Engineering

6.1 Navigation Diagram

Please find a user flow navigation diagram illustrating the different pages of my application and how the user can navigate to those pages in Appendix 11.2.

6.2 Demo Video

You can find a demo of my proof of concept at the link below:

https://youtu.be/Kh30x65T0Fg

6.3 Evaluation

In this section, I will be evaluating each module that I have implemented into AsecAV (malware scanners, permission manager, app locker) as well as how well I have integrated each module together into a unified security suite.

6.3.1 Malware Scanner

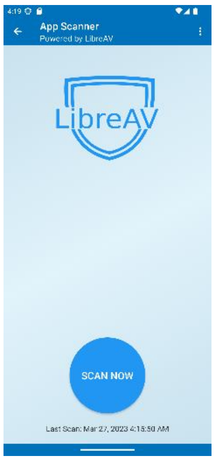


Figure 5, Screenshot of LibreAV running in AsecAV

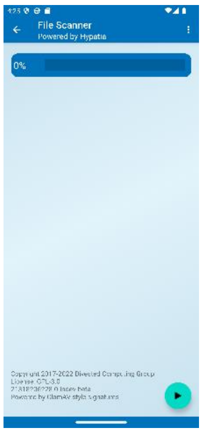
I concluded early on, after some preliminary research, that my malware scanner would be more successful if I was to consider using FOSS (free and open-source software) implementations of an Android malware scanner and I ultimately decided to use 2 different implementations of that were fundamentally different in what they are scanning on the Android device and how they detect malware in what they are scanning.

Originally, I believed that I should focus on an implementation that scanned installed applications on the Android device (LibreAV), however I came across an open-source application on GitHub (Hypatia) that was able to hash files on an Android device and compare their hashes to a database of known malware thus allowing it to scan individual files on an Android device for malware. The added value of using a file-based malware scanner alongside an app-based malware scanner was clear and I have implemented both into AsecAV.

If I had more time and expertise, I would look into updating/creating/using a more up to date dataset for the app scanner as the dataset LibreAV was using was created in 2017 and I was unable to find a more up to date dataset nor did I have the expertise or time to create my own dataset and deemed it a task too big to complete along with the rest of my project. If I decide to publish my app, I will investigate updating this dataset to provide the most effective security suite that is able to handle today's security risks. The dataset from 2017 still seemed to provide reliable results throughout my testing although it still shows some false positives such as the Galaxy Wearable app by Samsung. This is however expected as its main tell for whether an app is malicious is the number of permissions the app requests and since the Galaxy Wearable app requests a considerable number of permissions and the dataset is fairly old, this was too be expected.

Another improvement I would like to make would be the general UI of Hypatia (file scanner). I realised very late into overhauling the UI of my application that I would need to change a considerable amount of the backend of Hypatia and add an amount of extra functionality to achieve my vision for the user interface for Hypatia. I attempted to investigate this however realised that it would take too long for me to understand the inner workings of the module to a point where I would be able to modify its source code and add functionality without breaking the module.

Overall, I am happy with my implementation of malware scanners in AsecAV and believe I managed to achieve much of the functionality and accuracy I was envisioning for this part of AsecAV even if the UI of Hypatia could be improved. I am also pleased that all background features the file scanner and app scanner had been still intact and fully functional despite the merging process. I have still met my own personal targets for the malware scanner part of my app.


File Scanner
Powered by Hypatia
0%
Copyright 2017-2022 Divested Computing Group
License: GPL-3.0
Powered by ClamAV style signatures
Figure 6, Screenshot of Hypatia running in AsecAV

6.3.2 Permission Viewer

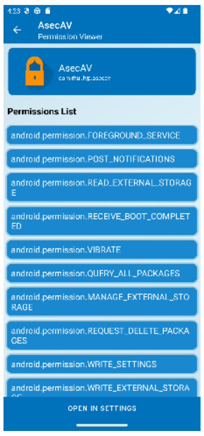

AsecAV
Permission Viewer
AsecAV
Permissions List
android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STORAGE
android.permission.REQUEST_DELETE_PACKAGES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORA
OPEN IN SETTINGS

Figure 7, Screenshot of Permission Viewer running in AsecAV

My vision for this app was to create an app that can not only show the user all the permissions that any selected app is using but can also allow the user to alter the app's permissions by even allowing the user to block permissions that the device's own settings don't allow. However, I had to simplify this vision of my permission manager to stay within the requirements I set myself in my aims and objectives.

The requirement in question was that the user does not need to modify their device in any way to be able to make full use of the security suite and for an external permission manager to be able to change permissions, the device would need to be rooted to give the permission manager the permissions necessary to be able to modify the Manifest files of other apps and remove permissions.

Despite this setback, I still saw value in creating an app that can show all the permissions and app uses, even permissions that the device's own settings hide from the user. This allows the user to know exactly what an app has access to and give users and indication as to whether the app is working as expected or if the app is performing malicious activity. Overall, I consider this implementation of a permission viewer a success as I have a permission viewer that can list all the permissions and app uses and provides the user a description of each permission and acts as a shortcut to the device's settings page for that app to allow them to change the permission the OS allows them to change by default.

6.3.3 App Locker

For my app locker, I originally investigated and even began programming the app locker implementation of my security suite myself but came across multiple hurdles almost instantly. Unfortunately, there was a discrepancy between what I knew and what I needed to know that, compounded by my already challenging to meet timeline, forced my hand into looking into an open-source implementation of an app locker. I came across an effective open-source application called MaxLock on Github.

Despite showing its age, MaxLock was by far the most effective implementation out of the shortlist I had researched and tested on my own personal devices. A couple of the key factors that separated MaxLock from the rest of the shortlist was its ability to allow the user to set different passwords for different applications as well as how quick it was detecting and then reacting to a user opening a locked app. This 2nd factor was one of the most difficult to find a good implementation for as many of the applications I tested had a multiple second delay between the app being launched and then being covered by the lock screen.

Throughout implementing MaxLock into AsecAV, I ran into multiple hurdles and errors that made MaxLock simply incompatible with my project without me modifying the properties of every other module in my application. Notably I had to downgrade the API versions and Gradle versions of every module to the version that MaxLock was doing. This meant doing a lot of downgrading and migrating of dependencies and libraries used in the other modules. Thankfully, after multiple trials and errors, I was able to get MaxLock implemented into AsecAV successfully whilst ensuring that all my other modules worked as intended. This task was time-consuming and threatened to put me behind schedule however after wrapping my head around Gradle and Kotlin, I was able to implement it into my application.

Below is a snippet of the function that oversees setting the lock screen in front of an application. Locking an application works by first checking whether the app the user has just clicked is on the list of apps to be locked, if the app is due to be locked than the below function is called with the app's package name being passed into it. This function then launches a new LockActivity over the application and sets all the necessary flags and extra information the lock activity needs such as the current pin/password/pattern. Without this function, the app locker would only know when an app is opened and would not be able to actually react.

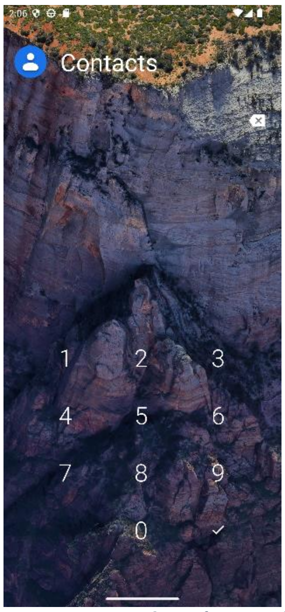
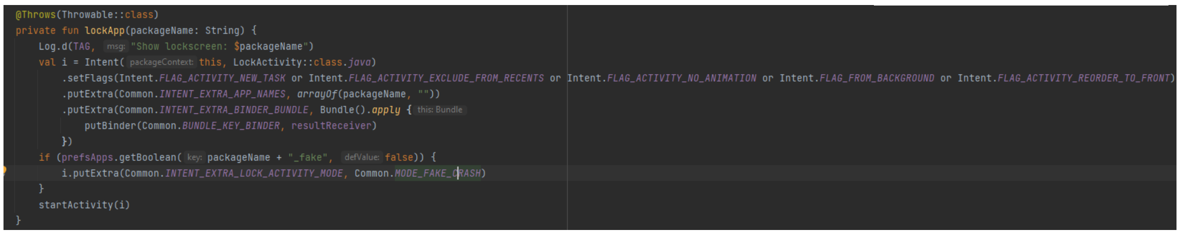


Figure 8, Screenshot of MaxLock running in AsecAV



```
@Throws(Throwable::class)
private fun lockApp(packageName: String) {
    Log.d(TAG, msg: "Show lockscreen: $packageName")
    val i = Intent( packageContext: this, LockActivity::class.java)
        .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS or Intent.FLAG_ACTIVITY_NO_ANIMATION or Intent.FLAG_FROM_BACKGROUND or Intent.FLAG_ACTIVITY_REORDER_TO_FRONT)
        .putExtra(Common.INTENT_EXTRA_APP_NAMES, arrayOf(packageName, ""))
        .putExtra(Common.INTENT_EXTRA_BINDER_BUNDLE, Bundle().apply { this: Bundle
            putBinder(Common.BUNDLE_KEY_BINDER, resultReceiver)
        })
    if (prefsApps.getBoolean( key: packageName + "_fake", defValue: false)) {
        i.putExtra(Common.INTENT_EXTRA_LOCK_ACTIVITY_MODE, Common.MODE_FAKE_CRASH)
    }
    startActivity(i)
}
```

Overall, I am very pleased with this implementation of the app locker as MaxLock incorporates all my required features of an app locker as well as the quality-of-life features that I wanted from a personal app locker. Despite having to jump over multiple hurdles to get Maxlock to work in AsecAV, I am glad that I haven't needed to sacrifice any of its functionality.

6.4 Methodology

After identifying my deliverables, by virtue of creating an Android application, I instantly settled on using Java as my programming language for much of the language whilst using Kotlin when any of my features require it. This is due to Java being one of my more confident languages which I have used to develop Android application in the past. The reason I was unable to program everything in Java was because some of the open-source implementations I was using used Kotlin to implement certain features. It was also clear that I needed to use Android Studio as my IDE of choice for both its access to Android emulators for testing, and Android Studio's tailored suite of Android related tools that improved the Android development experience.

I decided to take a modular approach to development, splitting my initial main application into three distinct functionalities, the malware Scanner, permission viewer and app locker. This allowed me to focus on each of the different mechanisms from a blank slate, so I could then integrate the features together into one security suite later. In this order, I implemented each module as its own standalone app with its own resource files, activities, and UI. This allowed me to debug each app individually and work on them separately.

When doing research on how I would be able to complete my deliverables and each module of my security suite, I assessed the feasibility of making each feature from scratch, versus cloning from an existing open-source project and then used this information to make informed decisions about which new technologies and frameworks I would need to learn to successfully program these features if I created them from scratch. My criteria for finding open-source projects were to evaluate whether the implementation matched or exceeded my own personal standards of effectiveness, if implemented. For example, for the malware scanner and app locker, I found open-source projects that fulfilled requirements of my application – such as the ability for unique patterns / pins to be created to protect user-selected 'sensitive' applications.

Some notable frameworks that I had to source to create the permission viewer include 'XML Pull Parser'. This package parses and filters xml files. In the context of the permission viewer, it reads the XML file of an app's Android Manifest file as a stream and allows me to check each line for conditions to filter through lines that are relevant to the app permissions. Specifically, it searches for the 'uses-permission' tag in the xml file. If it finds this tag, it reads and stores the permission name and adds it to an array. This allows me to collect requested / required permissions for each app.

Another framework that proved useful was 'Shared Preferences' which allowed the permission viewer to modify preference data returned by the context of an app. In the case of the malware scanner, it updates the last scanned key/pair value which logs the last time the device was scanned. Then, it stores this data in such a way that it persists even when the app is closed.

6.5 Testing

I made sure to continuously unit test each section by assessing it in accordance with my requirements. I investigated frameworks that I could use in the future to structure formal automated testing, such as Roboelectric which is a recommended fast and reliable framework that allows for Android app unit testing in the JVM workstation. However, after multiple attempts to implement frameworks such as Roboelectric and Espresso, I was unable to run any of the tests I've written due to a compilation error in Kotlin that I was unable to resolve even when following solutions found online. Because of this, I decided to switch from using frameworks to using traditional testing tables to test my app. I focused mainly on the components of the app that I had written rather than focusing on testing each security feature as I knew that each security feature had undergone its own testing by the author.

6.5.1 Operating System Compatibility Testing (Read Through)

As one of my requirements for the project was to ensure that AsecAV runs on all versions of Android from 5.0 onwards. I needed to find a way to test my application efficiently on multiple versions of Android. I decided to tackle this by performing my own in-depth tests on major versions of Android specifically the older versions of Android where I was less confident of the application working as expected. I also chose to use Google's own Firebase Test Lab which is a cloud-based app testing infrastructure that lets me test my app in a wide range of operating systems. This platform allows me to test my app on multiple devices simultaneously and I can choose whether to test my app on a physical device or virtual device.

As my main concern was whether AsecAV could compile and run successfully on different versions of Android, I chose to use Firebase's template Robo test which runs a 'robot' of sorts on my application that simulates random interactions with my application. By virtue of the interactions being random, the robo is incapable of testing every interaction possible on AsecAV however running the

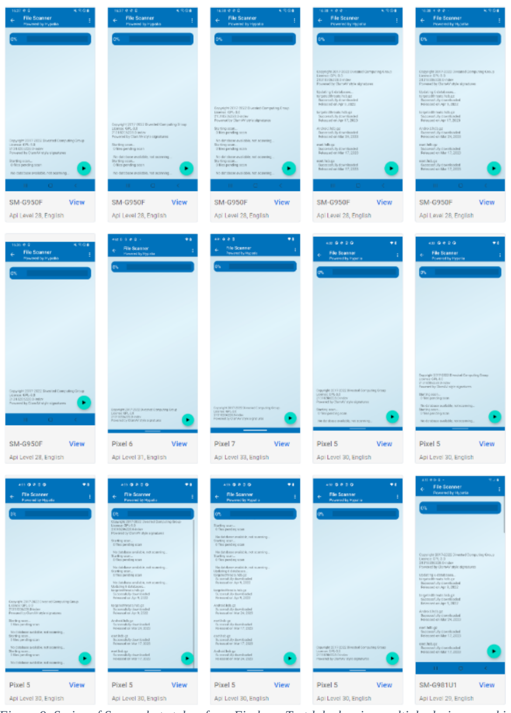


Figure 9, Series of Screenshots taken from Firebase Test lab showing multiple devices reaching and running the file-scanner tests and analysing the results still proved to be a great indicator as to whether there were any specific compatibility concerns regarding my application. It also shows me screenshots of how the app looked and ran on certain versions on Android that I can now have a look at and fix. For example, my UI for AsecAV didn't seem to load properly on devices that are running Android API version 26 and below as I was using a newer variation of setting colours on my UI. This was a great indicator that pointed me to exactly where I needed to modify my UI for the colours to show properly.

6.5.2 Android Profiling

After my interim presentation where I was able to explain the capabilities of my proof of concept to a live audience. I was presented with some feedback regarding another form of testing I should investigate to which was profiling my application to ensure that my application was running comfortably within the limits of the physical hardware of various Android devices. To measure this, I used Android Studio's built-in profiling tester. This allowed me to measure my applications CPU and memory usage when running various functions as well as what impact my application has on battery life. The figure below shows a sample of my profiling graph where I tested the impact of my application on Android.

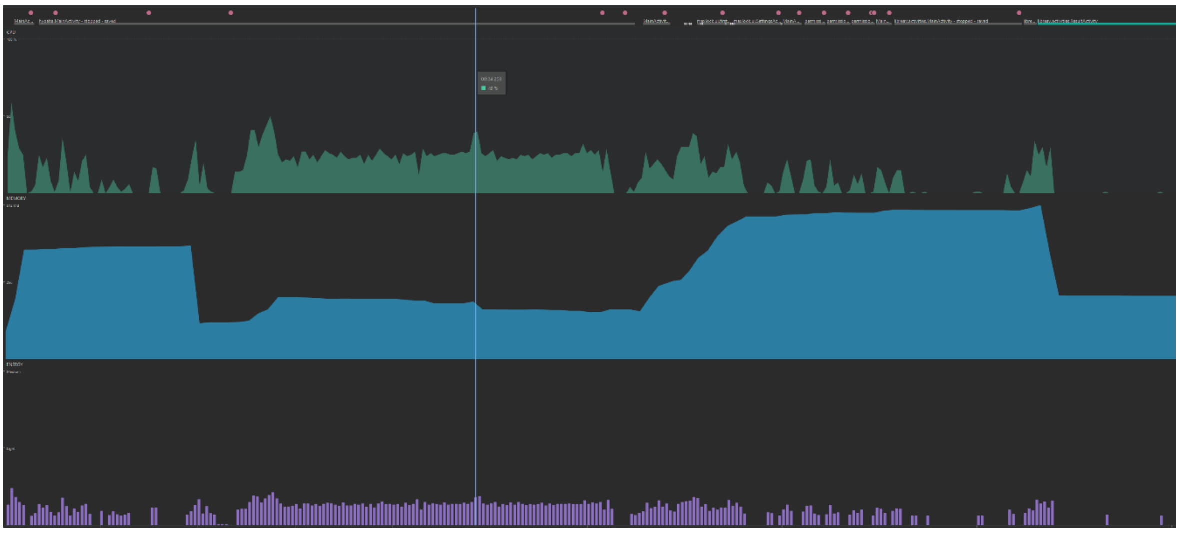


Figure 10, CPU/Memory/Energy Usage of AsecAV according to Android Profiling

6.5.3 Traditional Testing Tables

6.5.3.1 Navigation Testing

The following table shows my test cases for navigation testing where I tested possible ways the user can navigate AsecAV and check all buttons work as expected.

Test Number	Test Description	Expected Output	Actual Output	Pass/Fail
1	User can access each app by pressing on the relevant button	App takes user to module they selected	App takes user to module they selected	Pass

2	Users can exit app by pressing back button in app bar	App closes when button is pressed	App closes when button is pressed	Pass
3	Users can return to the main menu by pressing the back button in the app bar in any app	App returns to main menu when button is pressed	App returns to main menu when button is pressed in 3 of 4 modules. App locker does not present user with a back button in first page	Pass App locker: Fail (Figure 11)
4	Pressing the back button from any page in LibreAV will return the user to the previous page	App returns to previous page when button is pressed	App returns to previous page when button is pressed	Pass
5	Pressing scan button in LibreAV will load progress which once finished will load list of apps	App Scans apps on phone and outputs the results	App Scans apps on phone and outputs the results	Pass (Figure 12)
6	User can select an app from list to view its permissions in LibreAV	App shows list of permissions of selected app with an uninstall button at the bottom	App shows list of permissions of selected app with an uninstall button at the bottom	Pass
7	User can refresh list by pressing refresh button	App goes back to scan screen and rescans all apps and outputs result	App goes back to scan screen and rescans all apps and outputs result	Pass (Figure 12)
8	Users can navigate between all different pages in app locker and main menu	App locker goes to previous page whenever user presses back button	App locker goes to previous page whenever user presses back button	Pass

9	User can select an app from list to view its permissions in Permission Viewer	App shows user a list of permissions the app uses	App shows user a list of permissions the app uses	Pass (Figure 14)
10	User can go straight to settings page of app when they press on button in the permission Viewer.	App sends user to Android Settings permission page for chosen app	App sends user to Android Settings permission page for chosen app	Pass (Figure 15)
11	User can view all the open-source libraries used by app by selecting in menu on main page	App opens a list of libraries used and their licenses when button is clicked.	App opens a list of libraries used and their licenses when button is clicked.	Pass (Figure 16)

6.5.3.2 Functionality Testing (TO DO)

The following table shows my test cases for functionality testing where I tested the functionality of the modules in my application.

Test Number	Test Description	Expected Output	Actual Output	Pass/Fail
1	LibreAV: scans all apps on device and assigns each app a malware prediction score	List of apps with a status underneath each app and then a score when clicking on app	List of apps with a status underneath each app and then a score when clicking on app	Pass (*Figure 17*, *Figure 18*)
2	User can uninstall apps from LibreAV	Popup requesting uninstall appears when user presses button in LibreAV	Popup requesting uninstall appears when user presses button in LibreAV	Pass (*Figure 19*)
3	Hypatia can access selected folders and scan all apps	Hypatia completes scan and scans correct number of files.	Hypatia completes scan and scans correct number of files.	Pass (*Figure 20*, *Figure 21*)

4	Hypatia can update databases when user requests	Databases are shown to be updated in console	Databases are shown to be updated in console	Pass (*Figure 22*)
5	App locker allows user to set their own Pin	User can select to create a pin and enter pin	User can select to create a pin and enter pin	Pass (*Figure 23*)
6	App locker allows user to set their own pattern	User can select to create a pattern and enter pattern	User can select to create a pattern and enter pattern	Pass (*Figure 24*)
7	App locker allows user to set their own password	User can select to create a password and enter password	User can select to create a password and enter password	Pass (*Figure 25*)
8	App locker allows user to set their own knock code	User can select to create a knock code and enter knock code	User can select to create a knock code and enter knock code	Pass (*Figure 26*)
9	App locker allows user to lock apps of their choice	User can choose which apps to lock	User can choose which apps to lock	Pass (*Figure 27*)

10	App locker allows to user to set unique password for a chosen app	User can set a custom password for an app of their choice	User can set a custom password for an app of their choice	Pass (*Figure 28, Figure 29*)
11	App locker has minimal delay when showing lock screen when opening app	Lock screen appears near instantaneously when opening app	Lock screen appears near instantaneously when opening app	Pass (*Figure 30*)

6.5.3.3 Sample Malware Testing

The following table shows my test cases for sample malware testing where I tested that the malware scanning modules can detect sample malware files on the device.

Test Number	Test Description	Expected Output	Actual Output	Pass/Fail
1	LibreAV malicious application detection	App is flagged as Malware	App is flagged as Malware	Pass (Figure 31)
2	Hypatia malicious file detection	User is notified of Malicious File	User is notified of Malicious File in the console	Pass (Figure 32)

6.6 Project Diary

Please find the Project Diary in Appendix 9.2.

7. Professional Issues: Open-Source & Licensing

Open-Source software is software where the source code is freely and readily available for anyone to use, change, and distribute for any purpose. Open-Source software is released under an open-source license, such as MIT license and GNU General Public License (GPL), which dictates and guarantees the end user's right to use and modify the source code and the conditions under which they can share their modifications (Jerry Hildenbrand, 2012).

There are multiple open-source licenses that a developer can use with each license varying in the conditions required to be met when using/distributing the project. For example, the GPL license requires any further developments or modifications of the source code to be placed under the same license (Free Software Foundation, 2007) if the user chooses to distribute the software, whereas the Apache License provides more freedom when distributing by not requiring any modifications to be made open source (Apache, 2004).

The licenses available to use can be further split into 2 different groups, permissive licenses, and copyleft licenses. A permissive license allows software to be modified and copied without any obligation to distribute or share any modifications. In effect, it allows the developer to do so as they please with the software and distribute it however way they want including if they want to make their modified version of the software proprietary that a developer can sell (Joseph Morris, 2016). By contrast, a copyleft license gives the developer most of the same rights however any derivative work of the software can only be distributed under the same license meaning the developer who is generally required to share the source code (FOSSA Editorial Team, 2021).

There are two organisations that are seen in the industry as the reference point for defining Open Source, the Free Software Foundation (FSF) and the Open Source Initiative (OSI) with each promoting free software and playing different parts in the support of the open-source movement (Martin Callinan, 2020). The Free Software Foundation published the 'four freedoms' criteria that software needs to meet to qualify as free software. These four essential freedoms are:

- "The freedom to run the program as you wish, for any purpose." (Free Software Foundation, 2022)
- "The freedom to study how the program works and change it so it does your computing as you wish. Access to the source code is a precondition for this." (Free Software Foundation, 2022)
- "The freedom to redistribute copies so you can help others." (Free Software Foundation, 2022)
- "The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this." (Free Software Foundation, 2022)

If a program does not satisfy all these freedoms, the FSF considers the software unethical as they believe that "Cooperation is more important than Copyright."(Richard Stallman, 2021).

Compliance with open-source licenses should be a key legal concern for any user or organisation choosing to distribute or use open-source software as open-source software is still protected by copyright statements meaning that any violations of the license or if the author is not credited, the violation will constitute a copyright violation. If this were to be the case, the author of the software may have grounds to start legal proceedings against the perpetrator however this is at the discretion of the author. An issue with this is that the process of collecting evidence and then potentially hiring a lawyer to fight the case in court can be very expensive to the author and may dissuade them from doing anything more than asking the perpetrator to credit them. This barrier to defending their intellectual property can be seen as an ethical concern when developing open-source software.

Open-source development has grown significantly across all areas of software development, and this is because it provides developers with the ability to re-use and develop existing code to improve functionality quicker than writing new functionality from scratch. Open-source also promotes peer-review and collaboration between developers to improve their software.

Below is a list of the 3 open-source modules I used and what license they were under:

- MaxLock – Maxr1998 – GPL-3.0 License - https://github.com/Maxr1998/MaxLock
- Hypatia – Divested-Mobile – GPL-3.0 License - https://github.com/Divested-Mobile/Hypatia.
- LibreAV – Project Matris - GPL-3.0 License - https://github.com/projectmatris/antimalwareapp

Due to all 3 of the main modules in my application being licensed under the strong copyleft GPL-3.0 open source license, I had to also license my project under GPL-3.0 in order to adhere to their licenses. This also meant that I had to check that any other less important libraries I used was able to be covered by the GPL-3.0 license and ensure that my project and all its components and subcomponents adhered to any open-license rules.

As I used a considerable number of third-party components and libraries in my project, ensured that I adhered to the licenses of each individual module whilst making sure that they were all compatible with the license I was to write my project under. I also need to verify that all authors of these libraries are credited and any changes to the original source code was clearly indicated to allow anyone who was looking to use the source code of my project, to differentiate between my code and the original authors’ code.

8. User Manual

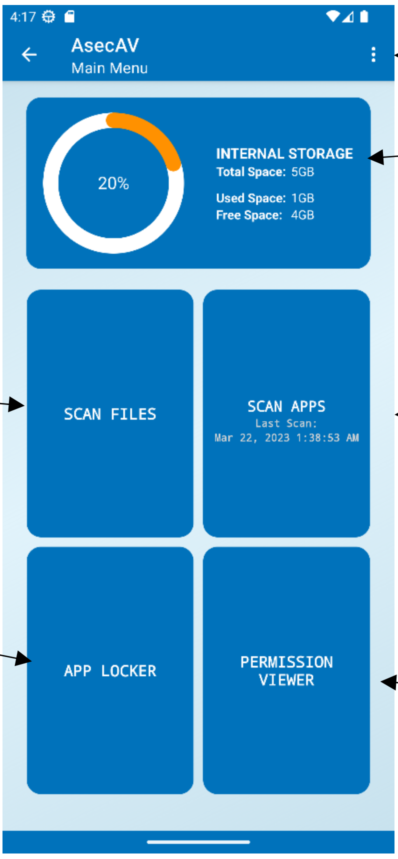
The following is a user manual that explains how to use AsecAV.

8.1 Installation

1. Copy APK onto Android device.
2. Click on APK to install in file manager.
3. Open AsecAV
4. Press accepts to any prompts asking for permission.

8.2 Main Menu

When you run AsecAV, the user will be greeted with the main menu that looks like the below. From this menu you can navigate to the various modules that are available on AsecAV:



Menu

Shows the current storage usage of the device.

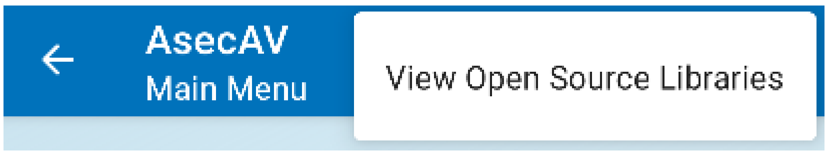
Click the "Scan Files" button to load up Hypatia which is the file scanner.

Click the "Scan Apps" button to load up LibreAV which is the app scanner.

Click the "App Locker" button to load up MaxLock which is the app locker.

Click the "Permission Viewer" button to load up the permission viewer.

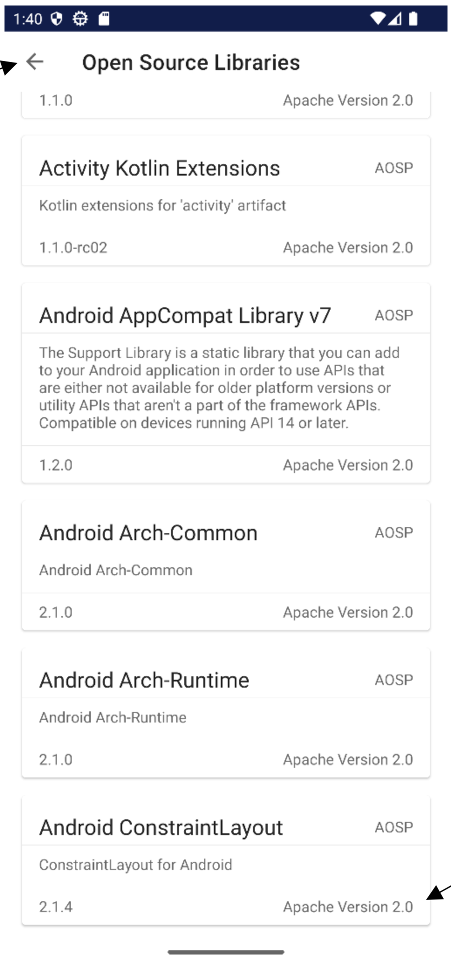
The main menu also has a menu button in the top right corner which provides the user with the option to view a list of all the open-source libraries I have used.



8.3 Open-Source Libraries View

The open-source libraries view provides the user with a list of all the open-source libraries that I have used in the development of AsecAV

Click here to navigate back to the main menu.



Open Source Libraries

1.1.0 Apache Version 2.0

Activity Kotlin Extensions AOSP
Kotlin extensions for 'activity' artifact
1.1.0-rc02 Apache Version 2.0

Android AppCompat Library v7 AOSP
The Support Library is a static library that you can add to your Android application in order to use APIs that are either not available for older platform versions or utility APIs that aren't a part of the framework APIs. Compatible on devices running API 14 or later.
1.2.0 Apache Version 2.0

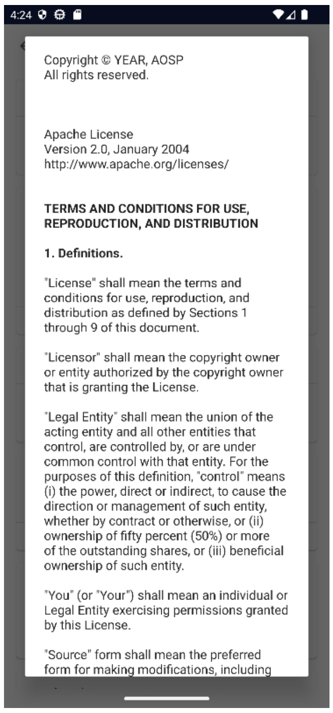
Android Arch-Common AOSP
Android Arch-Common
2.1.0 Apache Version 2.0

Android Arch-Runtime AOSP
Android Arch-Runtime
2.1.0 Apache Version 2.0

Android ConstraintLayout AOSP
ConstraintLayout for Android
2.1.4 Apache Version 2.0

Click here to view the license.

The user can also click on the license name in each tab to view and read the license.



Copyright © YEAR, AOSP
All rights reserved.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

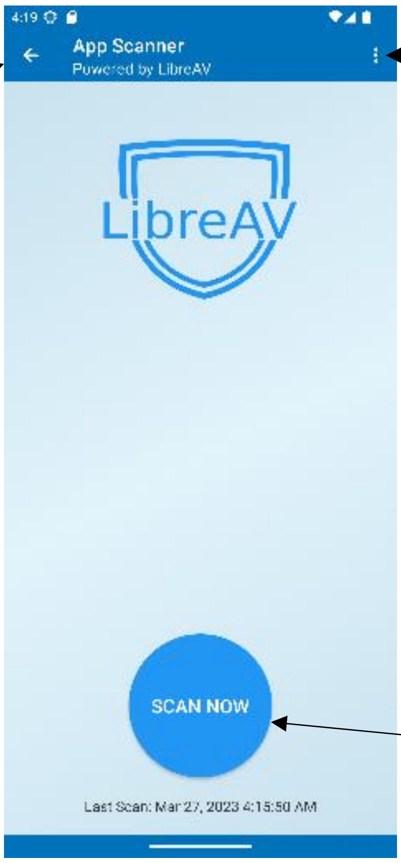
"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including

8.4 App Scanner (LibreAV)

The app scanner allows the user to execute a scan on all installed apps on their device.

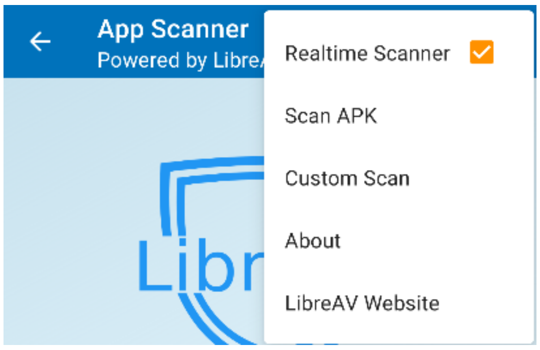
Press back arrow to return to previous page!



Menu

Press “Scan Now” button to initiate scan.

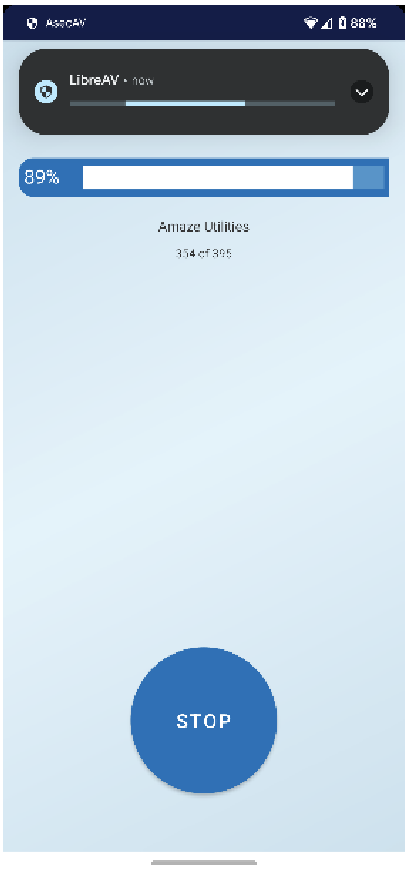
The homepage of LibreAV also has a menu that allows the user to select various options:



- “Realtime Scanner” – Enable/disable background scanner where LibreAV will scan new apps as they are installed on to the device.
- “Scan APK” – Choose a APK file on the device that LibreAV will scan.
- “Custom Scan” – Choos a specific app on the device that LibreAV will scan.
- “About” – Open an about page that briefly explains how LibreAV works.
- “LibreAV Website” – Hyperlink to open LibreAV’s website in your device’s default browser.

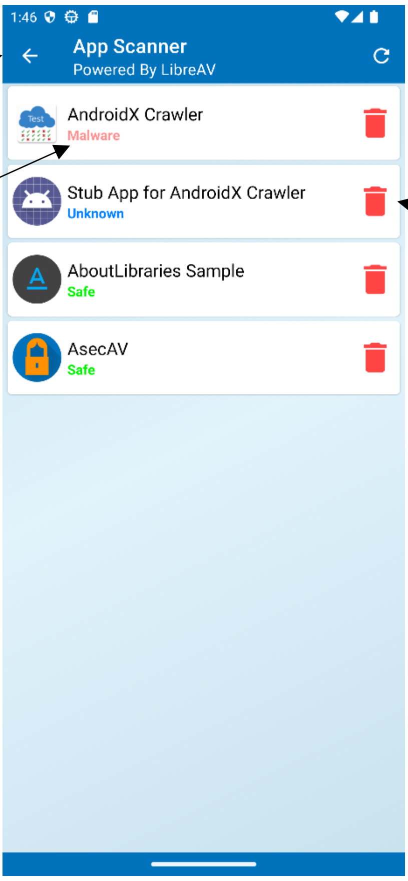
8.4.1 Scan View

If the user presses the “Scan Now” button in the home page of LibreAV, they will be greeted with a page that shows the progress of the scan and gives the user the ability to abort the scan.



8.4.2 List of Apps View/App Details view

Once LibreAV completes its scan, the user is greeted with a list of all their apps along with their determined classification by LibreAV.



Click here to return to previous page.

Classification as determined by LibreAV

Click here to refresh list.

Click here to uninstall application.

1:46

App Scanner
Powered By LibreAV

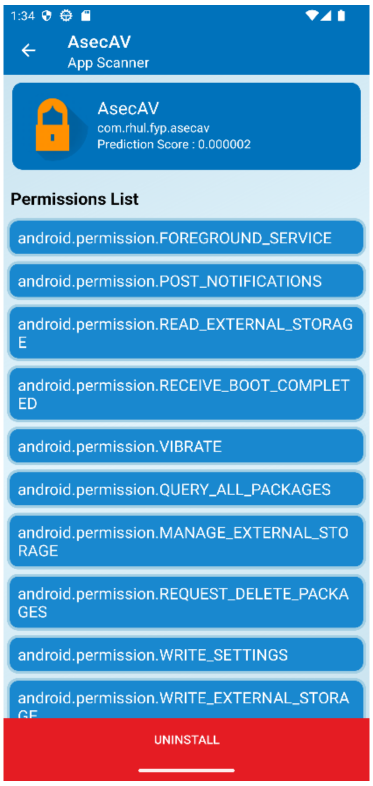
AndroidX Crawler
Malware

Stub App for AndroidX Crawler
Unknown

AboutLibraries Sample
Safe

AsecAV
Safe

The user is also able to click on one of the app tabs which will open the app details page with a list of the permissions that app uses.



1:34

AsecAV
App Scanner

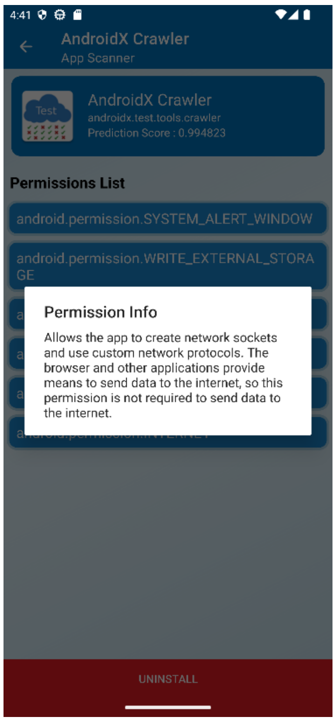
AsecAV
com.rhul.fyp.asecav
Prediction Score : 0.000002

Permissions List

android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STORAGE
android.permission.REQUEST_DELETE_PACKAGES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORAGE

UNINSTALL

From this page, the user can uninstall the application and can also find out more information about each permission by tapping on the permission.



4:41

AndroidX Crawler
App Scanner

AndroidX Crawler
androidx.test.tools.crawler
Prediction Score : 0.994823

Permissions List

android.permission.SYSTEM_ALERT_WINDOW

android.permission.WRITE_EXTERNAL_STORAGE

Permission Info

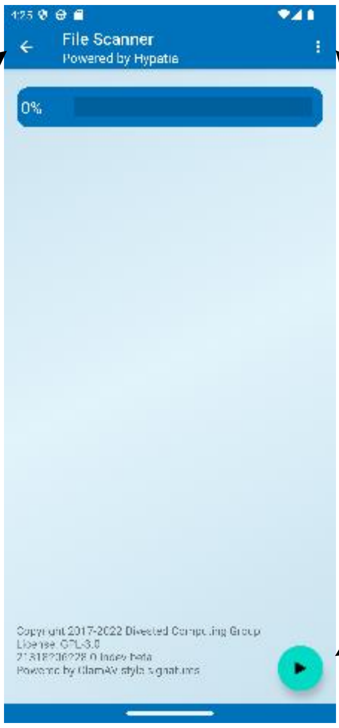
Allows the app to create network sockets and use custom network protocols. The browser and other applications provide means to send data to the internet, so this permission is not required to send data to the internet.

UNINSTALL

8.5 File Scanner (Hypatia)

If the user selects “Scan Files” button in the main menu, the application will launch Hypatia which is the file scanner for AsecAV.

Click here to return to Main Menu



File Scanner
Powered by Hypatia

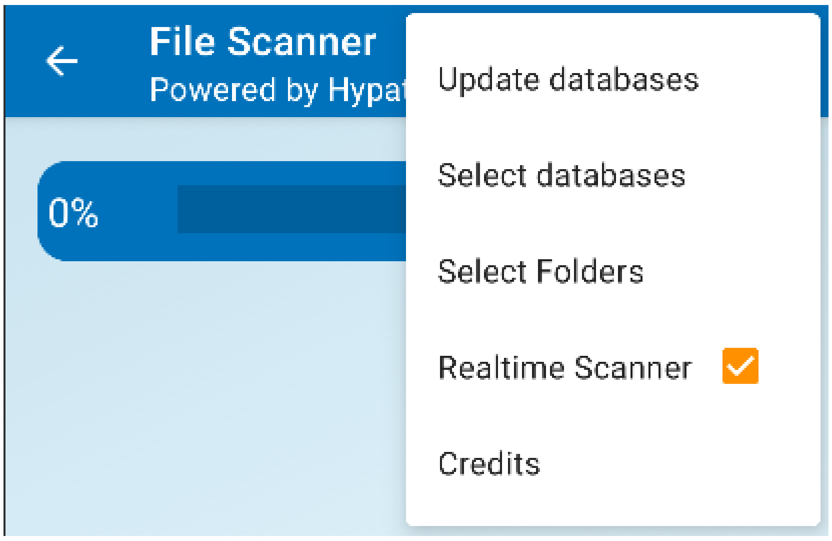
0%

Copyright 2017-2022 Divested Computing Group
License: GPL-3.0
21318206228.0 Index beta
Powered by ClamAV style signatures

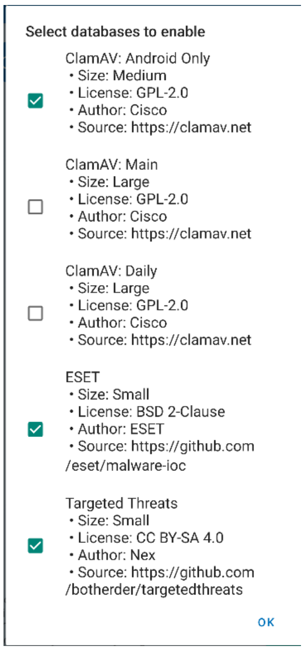
Menu

Click here to start scan.

Hypatia also includes a menu that allows the user to:



- "Update Database" – Check and update the virus signature databases that are stored locally on the device.
- "Select Database" – Select what databases they would like the scanner to use.

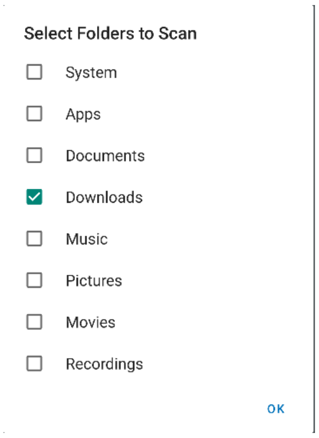


Select databases to enable

- [✓] ClamAV: Android Only
  • Size: Medium
  • License: GPL-2.0
  • Author: Cisco
  • Source: https://clamav.net
- [ ] ClamAV: Main
  • Size: Large
  • License: GPL-2.0
  • Author: Cisco
  • Source: https://clamav.net
- [ ] ClamAV: Daily
  • Size: Large
  • License: GPL-2.0
  • Author: Cisco
  • Source: https://clamav.net
- [✓] ESET
  • Size: Small
  • License: BSD 2-Clause
  • Author: ESET
  • Source: https://github.com/eset/malware-ioc
- [✓] Targeted Threats
  • Size: Small
  • License: CC BY-SA 4.0
  • Author: Nex
  • Source: https://github.com/botherder/targetedthreats

OK

- "Select Folders" – Select which folders they want the scanner to scan.



Select Folders to Scan

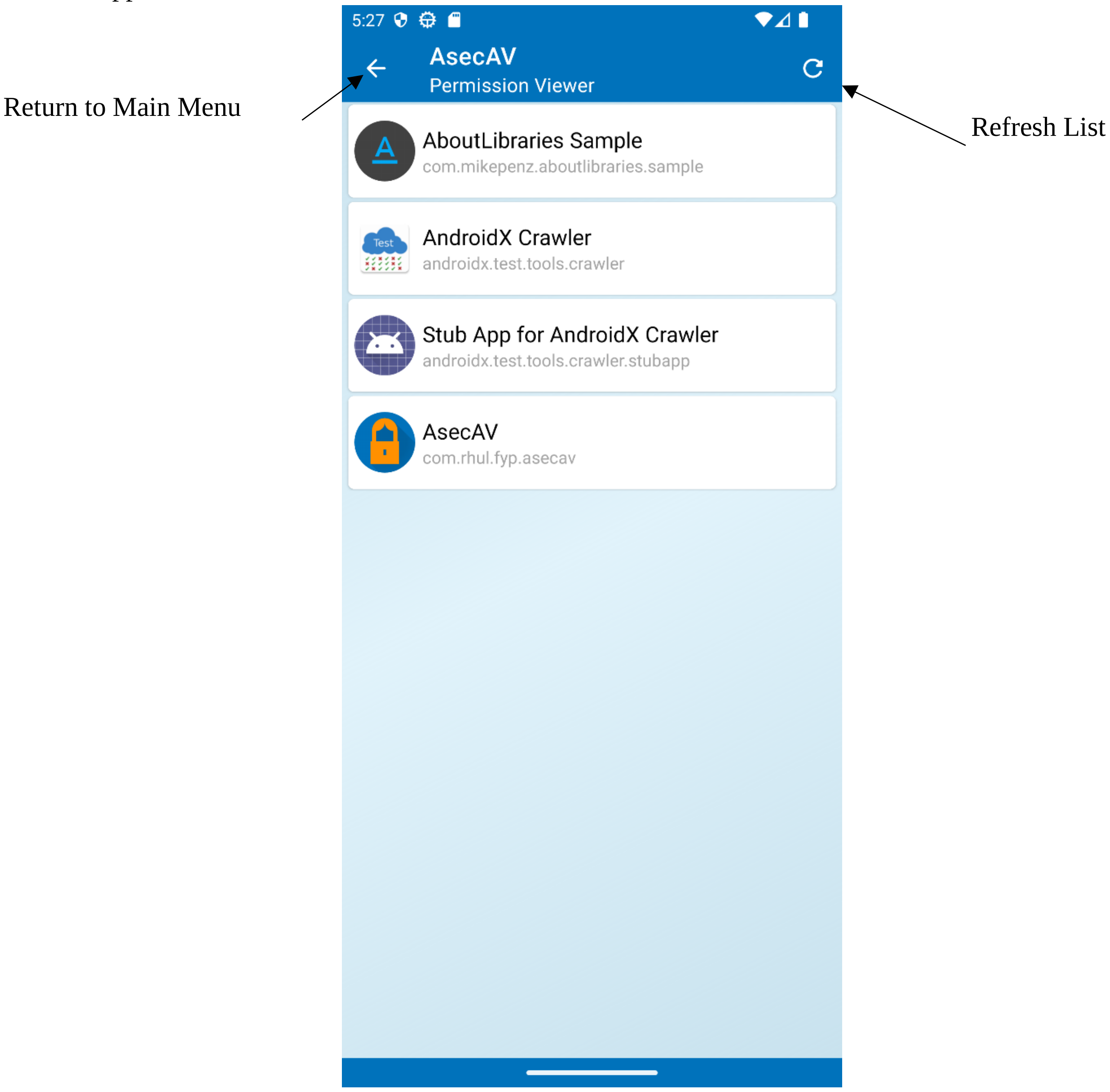
- [ ] System
- [ ] Apps
- [ ] Documents
- [✓] Downloads
- [ ] Music
- [ ] Pictures
- [ ] Movies
- [ ] Recordings

OK

- “Realtime Scanner” – Enable/disable the background scanner.
- “Credits” – Open the credits page for Hypatia.

8.6 Permission Viewer

If the user selects the “Permission Viewer” button in the main menu, they will be greeted with a of the applications installed on their device.



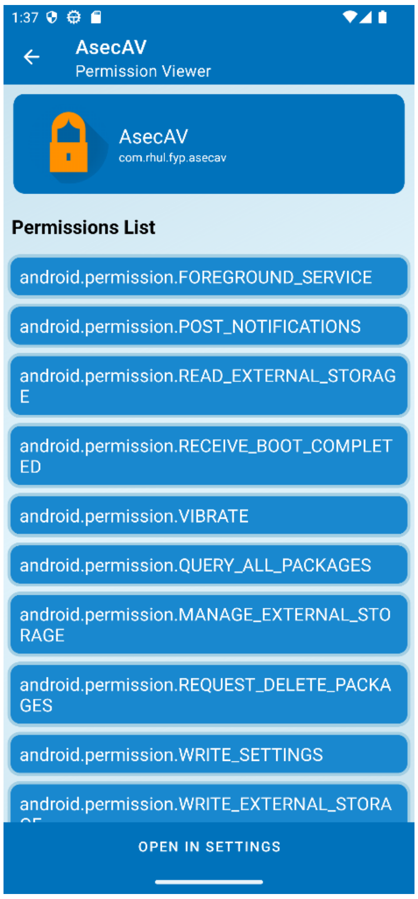
AboutLibraries Sample
com.mikepenz.aboutlibraries.sample

AndroidX Crawler
androidx.test.tools.crawler

Stub App for AndroidX Crawler
androidx.test.tools.crawler.stubapp

AsecAV
com.rhul.fyp.asecav

The user can then select and press on an application to view the app details page for their chosen application. From here the user can view the permissions the app uses and press on permissions for more information as well as press the button at the bottom of the page to open the settings page for the app.



1:37

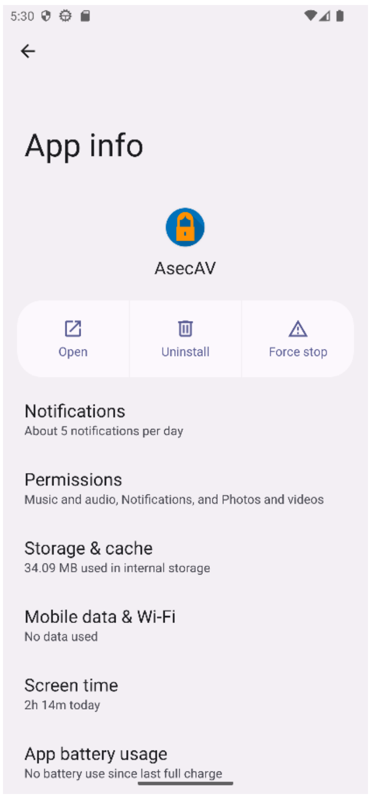
AsecAV
Permission Viewer

AsecAV
com.rhul.fyp.asecav

Permissions List

android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAG E
android.permission.RECEIVE_BOOT_COMPLET ED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STO RAGE
android.permission.REQUEST_DELETE_PACKA GES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORA

OPEN IN SETTINGS



5:30

App info

AsecAV

Notifications
About 5 notifications per day

Permissions
Music and audio, Notifications, and Photos and videos

Storage & cache
34.09 MB used in internal storage

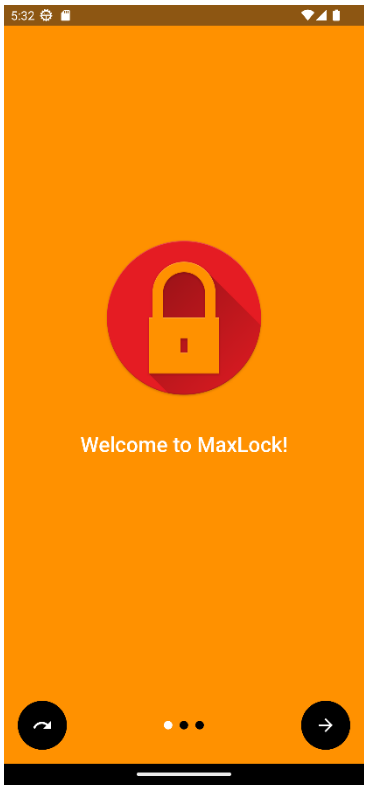
Mobile data & Wi-Fi
No data used

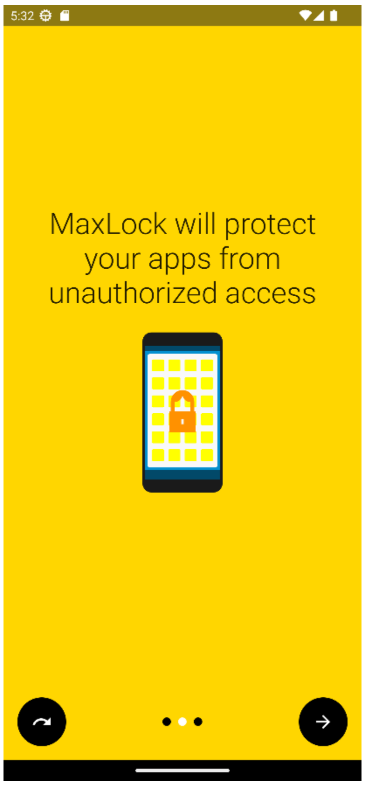
Screen time
2h 14m today

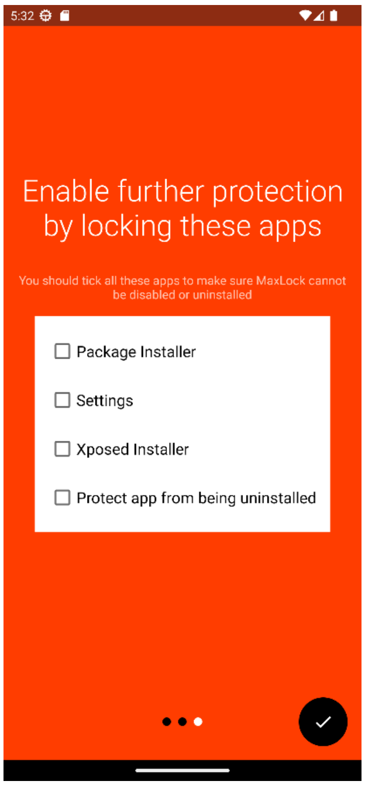
App battery usage
No battery use since last full charge

8.7 App Locker

On first launch of AsecAV, when the user clicks on the “App Locker button”, they will be greeted with a small intro for Maxlock which allows the user to set the app as an admin, so it is unable to be uninstalled.





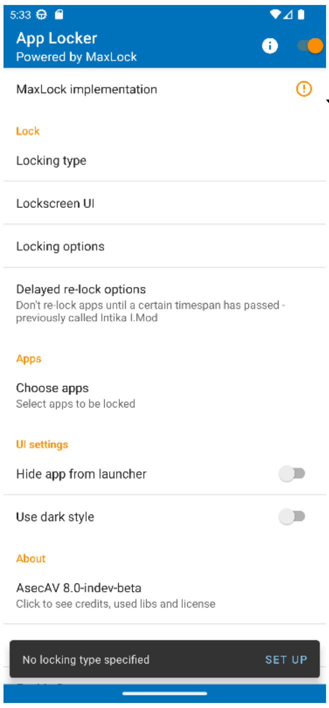


Enable further protection by locking these apps

You should tick all these apps to make sure MaxLock cannot be disabled or uninstalled

- ☐ Package Installer
- ☐ Settings
- ☐ Xposed Installer
- ☐ Protect app from being uninstalled

Once the user goes through these 3 pages, they will be greeted with the main menu for MaxLock.



App Locker
Powered by MaxLock

MaxLock implementation

Lock
Locking type
Lockscreen UI
Locking options
Delayed re-lock options
Don't re-lock apps until a certain timespan has passed - previously called Intika I.Mod

Apps
Choose apps
Select apps to be locked

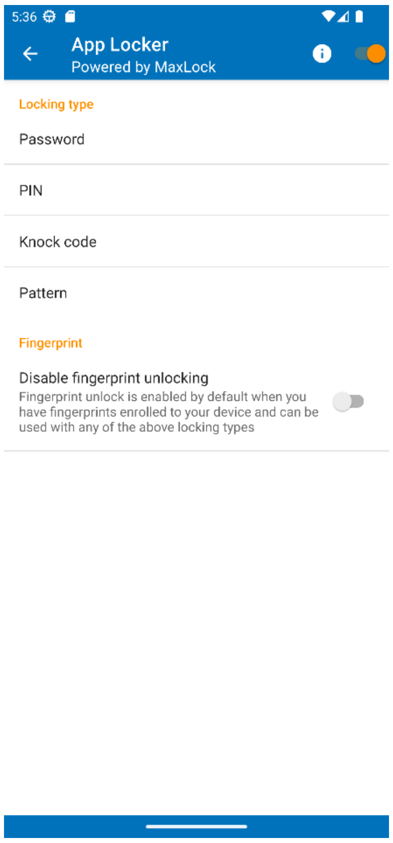
UI settings
Hide app from launcher
Use dark style

About
AsecAV 8.0-indev-beta
Click to see credits, used libs and license

No locking type specified SET UP

Click here for setup.

The very first thing the user needs to do to enable MaxLock is click on the orange exclamation mark in the top right corner and follow the instructions. Once completed MaxLock will be ready to lock apps on the device. The next thing the user needs to do is set their locking type and what their password is going to be. To do this, they need to click on “Locking Type” in the menu. They will be greeted with a list of different lock types.


5:36

App Locker
Powered by MaxLock

Locking type

Password

PIN

Knock code

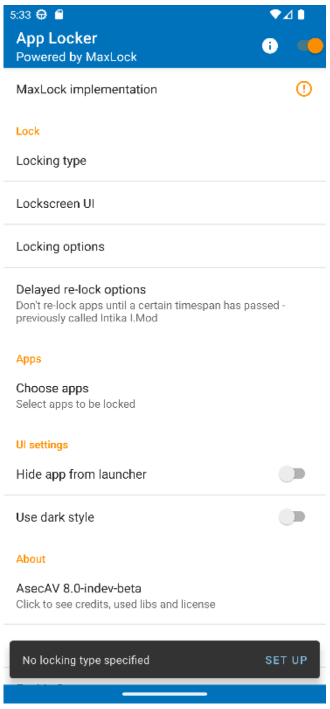
Pattern

Fingerprint

Disable fingerprint unlocking
Fingerprint unlock is enabled by default when you have fingerprints enrolled to your device and can be used with any of the above locking types

The user can then select one of the locking type’s and follow the instructions to create their password/pin.

Next the user wants to return to the menu and select what application they would like to lock.


5:33

App Locker
Powered by MaxLock

MaxLock implementation

Lock

Locking type

Lockscreen UI

Locking options

Delayed re-lock options
Don't re-lock apps until a certain timespan has passed - previously called Intika I.Mod

Apps

Choose apps
Select apps to be locked

UI settings

Hide app from launcher

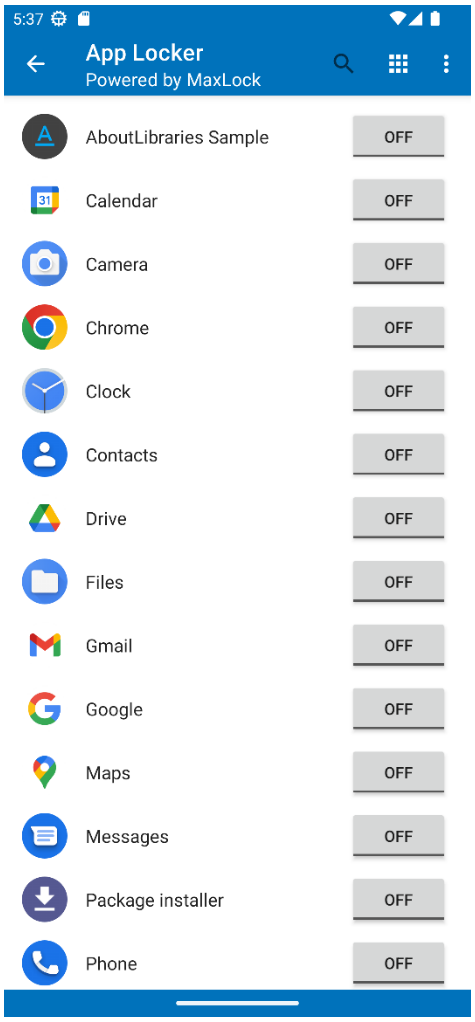
Use dark style

About

AsecAV 8.0-indev-beta
Click to see credits, used libs and license

No locking type specified SET UP

This is accomplished by selecting the “Choose Apps” button which will show the user a list of apps that they can toggle whether they want locked or not.


5:37

App Locker
Powered by MaxLock

AboutLibraries Sample	OFF
Calendar	OFF
Camera	OFF
Chrome	OFF
Clock	OFF
Contacts	OFF
Drive	OFF
Files	OFF
Gmail	OFF
Google	OFF
Maps	OFF
Messages	OFF
Package installer	OFF
Phone	OFF

The user then needs to toggle on/off which apps they would like to be locked. The next time the user launches their chosen locked app, they will be greeted with a lock screen where they need to input their password for the app to unlock.

To change the locking type/password the user needs to go to the main menu of the App Locker and click on “Locking Type” and change their lock.

9. Conclusion

To summarise this report, I have achieved all the aims and objectives that I had set out for myself at the beginning of the project with the only improvements remaining being UI. I believe that with some extra refinement and UI polishing, my app may be able to be publishable on the Google Play Store and this is something I may look to do over the summer.

Throughout the project, I enjoyed the process of decomposing tasks into smaller requirements and functional modules. This gave me the opportunity to explore and make use of open-source projects online. The downside of using open-source projects, is the need to tailor them to one’s specific needs. For some of the applications that I explored, this was easier to accomplish than in others. For example, Maxlock was ready to go once implemented into my security suite, however Hypatia UI requires a lot of modification in order to tailor to my objectives. So much so that I was unable to refine the UI as much as I would have liked.

Relating back to my aims & objectives, I believe that I have achieved all the aims & objectives that I set out for myself at the beginning of the project. Throughout the project, I placed a heavy emphasis in ensuring that AsecAV does not require any prior modification to Android OS and that it was compatible with as many versions of Android as possible. I set myself an original target for it to be compatible with Android 4.0 however I was unable to achieve this due to limitations regarding some of my open-source implementations. I was still able to achieve compatibility with Android 5.0 and above, so I am still satisfied. I consider this project to be a success and I have learnt a lot of new techniques and functions regarding managing security on Android as well as general Android development.

10. Bibliography

ANGGRAYUDI. 2020. *Android Hidden API* [Online]. Available: https://github.com/anggrayudi/Android-hidden-api [Accessed 15/03 2023].

APACHE. 2004. *Apache License, Version 2.0* [Online]. aPACHE. Available: https://www.apache.org/licenses/LICENSE-2.0 [Accessed 14/03 2023].

APPBRAIN.COM 2023. Android phone manufacturer market share. AppBrain: @TheAppBrain.

ARIS, HAZLEEN & YAAKOB, WIRA FIRDAUS. Shoulder Surf Resistant Screen Locking for Smartphones: A Review of Fifty Non-Biometric Methods. 2018 2018. IEEE.

BAZRAFSHAN, ZAHRA, HASHEMI, HASHEM, FARD, SEYED MEHDI HAZRATI & HAMZEH, ALI. A survey on heuristic malware detection techniques. 5th Conference on Information and Knowledge Technology (IKT), 2013 2013. IEEE.

BIRCH, JOE 2017. Exploring Background Execution Limits on Android Oreo.

BROWN, C. SCOTT. 2022. *Here are the phone update policies from every major Android manufacturer* [Online]. Android Authority. Available: https://www.Androidauthority.com/phone-update-policies-1658633/ [Accessed 24/03 2023].

CALLINAN, MARTIN. 2020. *Ethics of Open Software Source Licensing | Source Code Control Limited* [Online]. Available: https://sourcecodecontrol.co/ethics-of-open-source-licensing/ [Accessed 15/03 2023].

COMPUTERHOPE. 2022. *What is a Lock Screen?* [Online]. Available: https://www.computerhope.com/jargon/l/lockscreen.htm [Accessed].

DESIGN, GOOGLE MATERIAL. 2022. *Get Started | Material Design 3* [Online]. Available: https://m3.material.io/ [Accessed].

DEVELOPER, GOOGLE. 2022. *UsageStatsManager | Android Developers* [Online]. Available: https://developer.Android.com/reference/Android/app/usage/UsageStatsManager [Accessed 29/11/2022 2022].

DEVELOPERS, GOOGLE. 2022. *Android Studio | Android Developers* [Online]. Available: https://developer.Android.com/studio/features [Accessed 01/12/2022 2022].

DEVELOPERS, GOOGLE. 2022. *Configure your build | Android Developers* [Online]. Available: https://developer.Android.com/studio/build [Accessed 01/11/2022 2022].

DEVELOPERS, GOOGLE. 2022. *Declare app Permissions | Android Developers* [Online]. Available: https://developer.Android.com/training/permissions/declaring#:~:text=obvious%20to%20them.-,Add%20declaration%20to%20app%20manifest,has%20this%20line%20in%20AndroidManifest. [Accessed 30/11/2022 2022].

DUNCAN, GEOFF. 2022. *The ultimate Android malware guide: What it does, where it came from, and how to protect your phone or tablet | Digital Trends* [Online]. Available: https://www.digitaltrends.com/mobile/the-ultimate-Android-malware-guide-what-it-does-where-it-came-from-and-how-to-protect-your-phone-or-tablet/ [Accessed 24/11 2022].

FOUNDATION, FREE SOFTWARE. 2007. *The GNU General Public License v3.0 - GNU Project - Free Software Foundation* [Online]. GNU Operating System. Available: https://www.gnu.org/licenses/gpl-3.0.html [Accessed 14/03 2023].

FOUNDATION, FREE SOFTWARE. 2022. *What is Free Software? - GNU Project - Free Software Foundation* [Online]. Available: https://www.gnu.org/philosophy/free-sw.en.html [Accessed].

GHANCHI, JUNED 2021. The Major Advantages of Android Studio App Development.

GIANG, PHAM, DUC, NGUYEN & VI, PHAM 2015. *Permission Analysis for Android Malware Detection*.

GLOBALSTATS, STATCOUNTER 2023. Mobile & Tablet Android Version Market Share Worldwide | Statcounter Global Stats. @statcountergs.

GRIMES, ROGER A. 2020. 9 types of malware and how to recognize them | CSO Online.

GUIDE, ANDROID DEVELOPERS. 2022. *App Manifest Overview | Android Developers* [Online]. [Accessed 19/11/2022].

HAHN, SEBASTIAN, PROTSENKO, MYKOLA & MÜLLER, TILO. Comparative evaluation of machine learning-based malware detection on Android. Sicherheit, 2016.

HILDENBRAND, JERRY. 2012. *What is open source?* [Online]. Android Central. Available: https://www.Androidcentral.com/what-open-source-Android-z [Accessed 14/03 2023].

INITIATIVE, OPEN SOURCE. 2006. *The Open Source Definition* [Online]. Available: https://opensource.org/osd/ [Accessed 16/03 2023].

KATZENBEISSER, STEFAN, KINDER, JOHANNES & VEITH, HELMUT 2011. Malware Detection. Springer US.

LIU, KAIJUN, XU, SHENGWEI, XU, GUOAI, ZHANG, MIAO, SUN, DAWEI & LIU, HAIFENG 2020. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access,* 8**,** 124579-124607.

MATRIS, PROJECT. *LibreAV* [Online]. Available: https://github.com/projectmatris/antimalwareapp#readme.

OVERFLOW, STACK. 2022. *How to make app lock app in Android?* [Online]. Available: https://stackoverflow.com/questions/36261909/how-to-make-app-lock-app-in-Android [Accessed 30/11/2022 2022].

SCHMITT, VERA, POIKELA, MAIJA & MÖLLER, SEBASTIAN. Android Permission Manager, Visual Cues, and their Effect on Privacy Awareness and Privacy Literacy. ARES 22, 2022 2022. ACM.

SIMPLILEARN 2022. What is Gradle? Why Do We Use Gradle? [Updated].

SOURCE, ANDROID. 2022. *File-Based Encryption | Android Source* [Online]. Available: https://source.Android.com/docs/security/features/encryption/file-based [Accessed 20/11/2022 2022].

SOURCE, ANDROID. 2022. *Full-Disk Encryption | Android Source* [Online]. Available: https://source.Android.com/docs/security/features/encryption/full-disk [Accessed 20/11/2022 2022].

STALLMAN, RICHARD. 2021. *Why Software Should Not Have Owners - GNU Project - Free Software Foundation* [Online]. Available: https://www.gnu.org/philosophy/why-free.html [Accessed 15/03 2023].

STATS, STATCOUNTER GLOBAL. 2022. *Mobile Operating System Market Share Worldwide | Statcounter Global Stats* [Online]. @statcountergs. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed 19/11/2022].

STEFAN HAUSTEIN & SLOMINSKI, ALEKSANDER. 2022. *XML Pull Parsing* [Online]. Available: http://www.xmlpull.org/ [Accessed 30/11/2022 2022].

STOLERIU, RAZVAN & TOGAN, MIHAI. A Secure Screen and App Lock System for Android Smart Phones Using Face Recognition. 2020 2020. IEEE.

SUPPORT, GOOGLE. 2023. *Learn when you'll get software updates on Google Pixel phones - Pixel phone Help* [Online]. Available: https://support.google.com/pixelphone/answer/4457705?hl=en-GB#zippy=%2Cpixel-and-later [Accessed 24/03 2023].

TAM, KIMBERLY, FEIZOLLAH, ALI, ANUAR, NOR BADRUL, SALLEH, ROSLI & CAVALLARO, LORENZO 2017. The Evolution of Android Malware and Android Analysis Techniques. *ACM Computing Surveys,* 49**,** 1-41.

TECHTERMS.COM. 2022. *Virus Definition | TechTerms.com* [Online]. Available: https://techterms.com/definition/virus [Accessed 19/11/2022].

WANG, DAIBIN, YAO, HAIXIA, LI, YINGJIU, JIN, HAI, ZOU, DEQING & DENG, ROBERT H. 2017. A Secure, Usable, and Transparent Middleware for Permission Managers on Android. *IEEE Transactions on Dependable and Secure Computing,* 14**,** 350-362.

WU, DONG-JIE, MAO, CHING-HAO, WEI, TE-EN, LEE, HAHN-MING & WU, KUO-PING. DroidMat: Android Malware Detection through Manifest and API Calls Tracing. 2012. IEEE.

11. Appendix

11.1 Project Specification

Aims: Develop a security suite for Android based smartphones.

Background: This project will develop a security suite for Android smartphones. There are several security apps on the Google Play Store which take care of certain security aspects but there isn’t any all-in-one solution which covers the security. This project will analyse the possibility of having one such security package by using secure sensor management and enforcing authentication and access control policies without modifying the ROM or Android OS.

Prerequisites:

- Good programming skills, particularly Android app development, and knowledge of IDEs.
- Being able to autonomously download/compile/install tools from various repositories (e.g., GitHub).

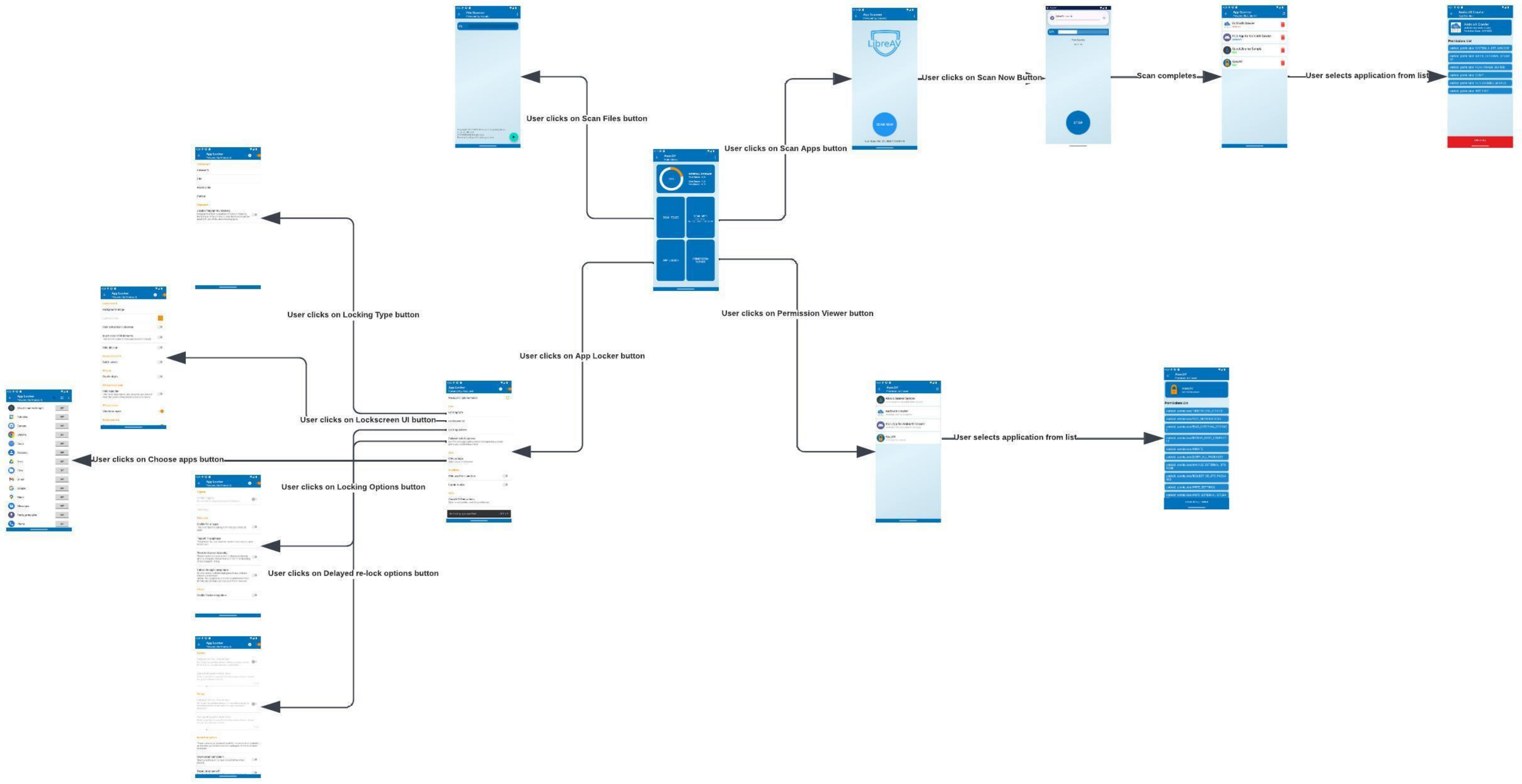
Early Deliverables

- A report describing different security apps for integrating into security suite.
- A proof-of-concept implementation of the security suite
- Testing the system

Final Deliverables

- Design and develop the final security suite.
- Complete report which must include a User Manual

11.2 Navigation Diagram


User clicks on Scan Files button
User clicks on Scan Apps button
User clicks on Scan Now Button
Scan completes
User selects application from list
User clicks on Permission Viewer button
User selects application from list
User clicks on App Locker button
User clicks on Locking Type button
User clicks on Lockscreen UI button
User clicks on Choose apps button
User clicks on Locking Options button
User clicks on Delayed re-lock options button
LibreAV

11.3 Testing Figures

11.3.1 Navigation Testing

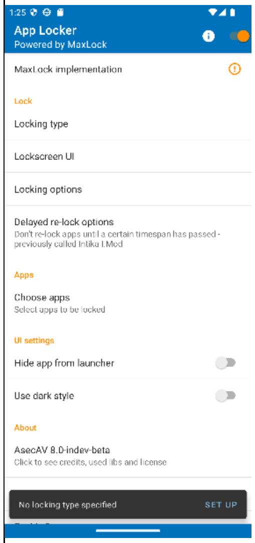

App Locker
Powered by MaxLock
MaxLock implementation
Lock
Locking type
Lockscreen UI
Locking options
Delayed re-lock options
Don't re-lock apps until a certain timespan has passed - previously called Intika I.Mod
Apps
Choose apps
Select apps to be locked
UI settings
Hide app from launcher
Use dark style
About
AsecAV 8.0-indev-beta
Click to see credits, used libs and license
No locking type specified
SET UP

Figure 11, Navigation Test 3

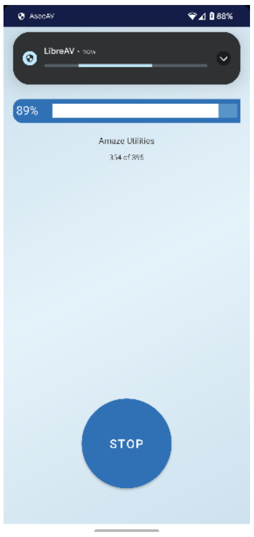

AsecAV
LibreAV • now
89%
Amaze Utilities
354 of 395
STOP

Figure 12, Navigation Test 5&7

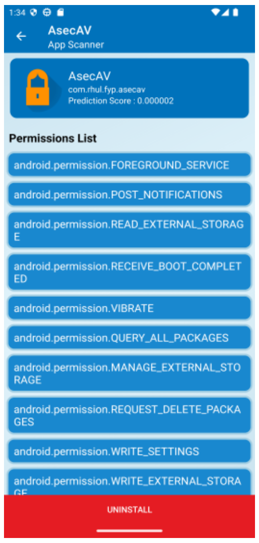

AsecAV
App Scanner
AsecAV
com.rhul.fyp.asecav
Prediction Score : 0.000002
Permissions List
android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STORAGE
android.permission.REQUEST_DELETE_PACKAGES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORAGE
UNINSTALL

Figure 13, Navigation Test 6

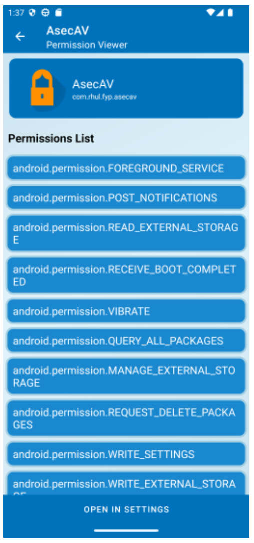

AsecAV
Permission Viewer
AsecAV
com.rhul.fyp.asecav
Permissions List
android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STORAGE
android.permission.REQUEST_DELETE_PACKAGES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORAGE
OPEN IN SETTINGS

Figure 14, Navigation Test 9

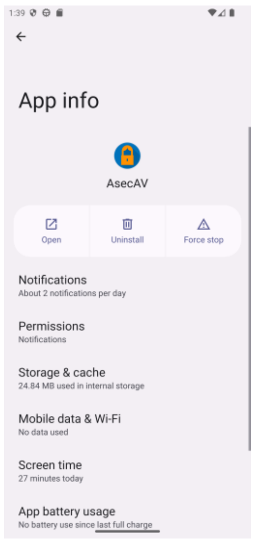

App info
AsecAV
Open
Uninstall
Force stop
Notifications
About 2 notifications per day
Permissions
Notifications
Storage & cache
24.84 MB used in internal storage
Mobile data & Wi-Fi
No data used
Screen time
27 minutes today
App battery usage
No battery use since last full charge

Figure 15, Navigation Test 10

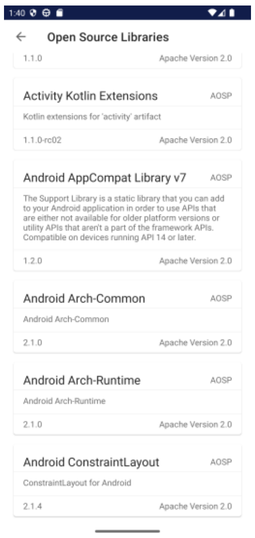
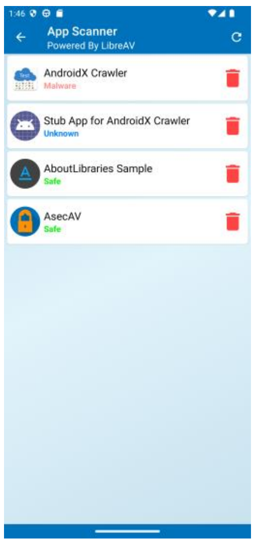

Open Source Libraries
1.1.0 Apache Version 2.0
Activity Kotlin Extensions AOSP
Kotlin extensions for 'activity' artifact
1.1.0-rc02 Apache Version 2.0
Android AppCompat Library v7 AOSP
The Support Library is a static library that you can add to your Android application in order to use APIs that are either not available for older platform versions or utility APIs that aren't a part of the framework APIs. Compatible on devices running API 14 or later.
1.2.0 Apache Version 2.0
Android Arch-Common AOSP
Android Arch-Common
2.1.0 Apache Version 2.0
Android Arch-Runtime AOSP
Android Arch-Runtime
2.1.0 Apache Version 2.0
Android ConstraintLayout AOSP
ConstraintLayout for Android
2.1.4 Apache Version 2.0

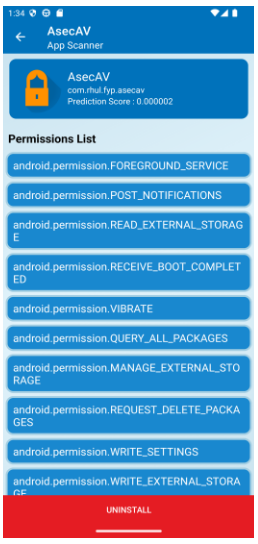
Figure 16, Navigation Test 11

11.3.2 Functionality Testing



AndroidX Crawler
Malware
Stub App for AndroidX Crawler
Unknown
AboutLibraries Sample
Safe
AsecAV
Safe

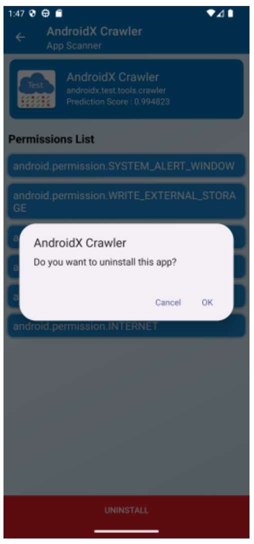
Figure 17, Functionality Test 1



AsecAV
com.rhul.fyp.asecav
Prediction Score : 0.000002

Permissions List
android.permission.FOREGROUND_SERVICE
android.permission.POST_NOTIFICATIONS
android.permission.READ_EXTERNAL_STORAGE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.VIBRATE
android.permission.QUERY_ALL_PACKAGES
android.permission.MANAGE_EXTERNAL_STORAGE
android.permission.REQUEST_DELETE_PACKAGES
android.permission.WRITE_SETTINGS
android.permission.WRITE_EXTERNAL_STORAGE

Figure 18, Functionality Test 1

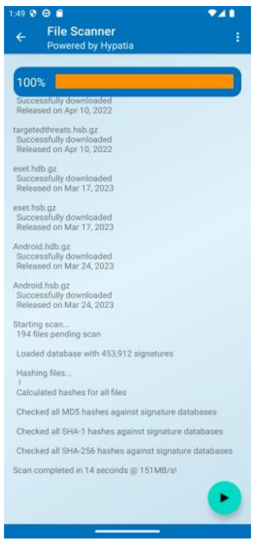


AndroidX Crawler
androidx.test.tools.crawler
Prediction Score : 0.994823

Permissions List
android.permission.SYSTEM_ALERT_WINDOW
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.INTERNET

AndroidX Crawler
Do you want to uninstall this app?

Figure 19, Functionality Test 2



Successfully downloaded
Released on Apr 10, 2022

targetedthreats.hsb.gz
Successfully downloaded
Released on Apr 10, 2022

eset.hdb.gz
Successfully downloaded
Released on Mar 17, 2023

eset.hsb.gz
Successfully downloaded
Released on Mar 17, 2023

Android.hdb.gz
Successfully downloaded
Released on Mar 24, 2023

Android.hsb.gz
Successfully downloaded
Released on Mar 24, 2023

Starting scan...
194 files pending scan

Loaded database with 453,912 signatures

Hashing files...
!
Calculated hashes for all files

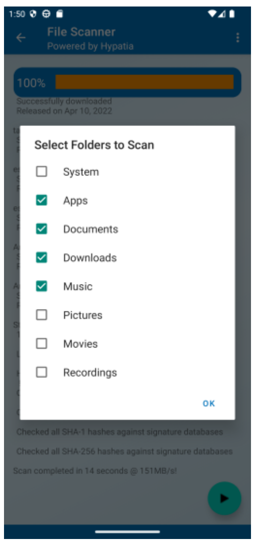
Checked all MD5 hashes against signature databases

Checked all SHA-1 hashes against signature databases

Checked all SHA-256 hashes against signature databases

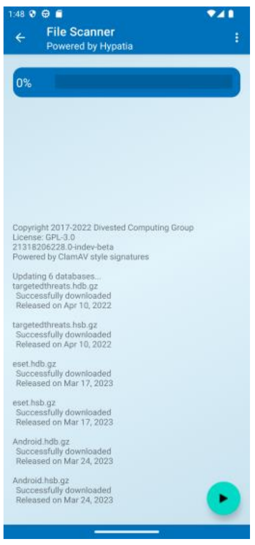
Scan completed in 14 seconds @ 151MB/s!

Figure 20, Functionality Test 3



Select Folders to Scan
☐ System
☑ Apps
☑ Documents
☑ Downloads
☑ Music
☐ Pictures
☐ Movies
☐ Recordings

Figure 21, Functionality Test 3



Copyright 2017-2022 Divested Computing Group
License: GPL-3.0
21318206228.0-indev-beta
Powered by ClamAV style signatures

Updating 6 databases...
targetedthreats.hdb.gz
Successfully downloaded
Released on Apr 10, 2022

targetedthreats.hsb.gz
Successfully downloaded
Released on Apr 10, 2022

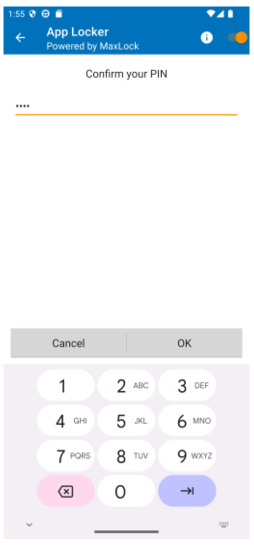
eset.hdb.gz
Successfully downloaded
Released on Mar 17, 2023

eset.hsb.gz
Successfully downloaded
Released on Mar 17, 2023

Android.hdb.gz
Successfully downloaded
Released on Mar 24, 2023

Android.hsb.gz
Successfully downloaded
Released on Mar 24, 2023

Figure 22, Functionality Test 4



Confirm your PIN
••••

Figure 23, Functionality Test 5

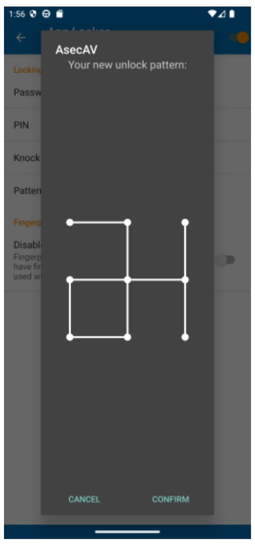


Figure 24, Functionality Test 6

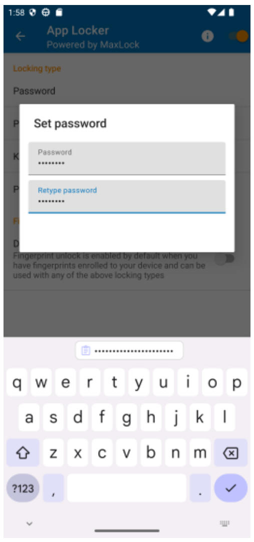

App Locker
Powered by MaxLock
Locking type
Password
Set password
Password
Retype password
Fingerprint unlock is enabled by default when you have fingerprints enrolled to your device and can be used with any of the above locking types

Figure 25, Functionality Test 7

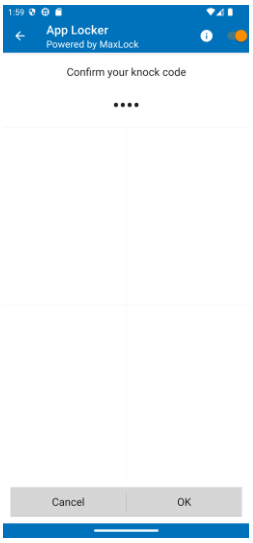


Figure 26, Functionality Test 7

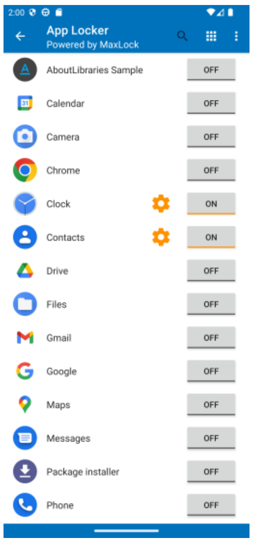


Figure 27, Functionality Test 9

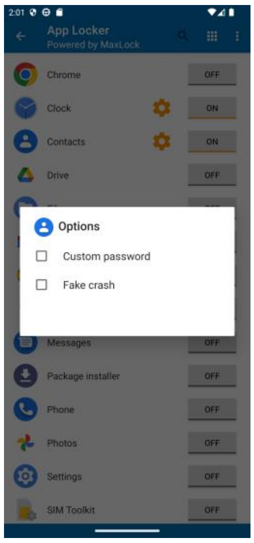


Figure 28, Functionality Test 10

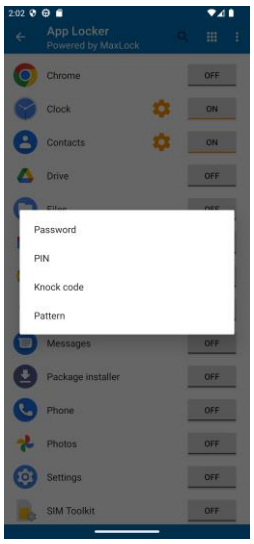


Figure 29, Functionality Test 10

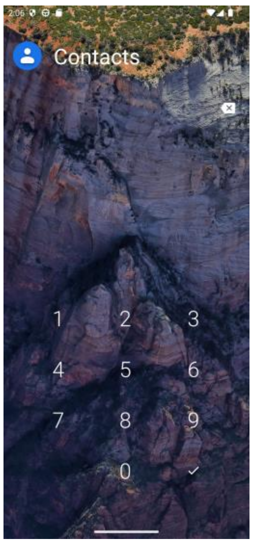
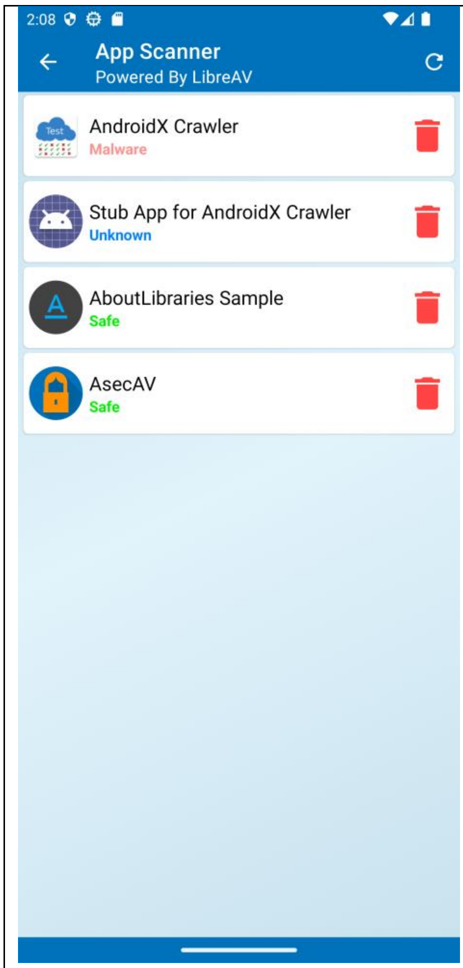


Figure 30, Functionality Test 11

11.3.3 Sample Malware Testing



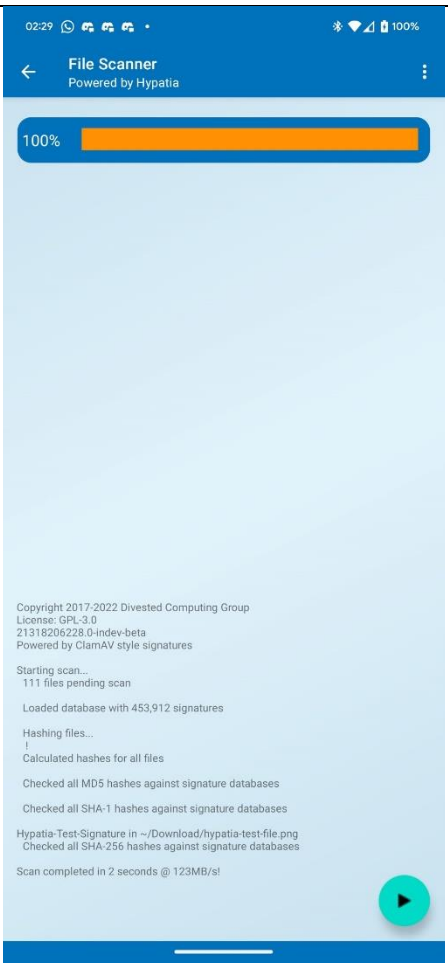
AndroidX Crawler
Malware

Stub App for AndroidX Crawler
Unknown

AboutLibraries Sample
Safe

AsecAV
Safe

Figure 31, Sample Malware Test 1



```
Copyright 2017-2022 Divested Computing Group
License: GPL-3.0
21318206228.0-indev-beta
Powered by ClamAV style signatures

Starting scan...
 111 files pending scan

 Loaded database with 453,912 signatures

 Hashing files...
  !
 Calculated hashes for all files

 Checked all MD5 hashes against signature databases

 Checked all SHA-1 hashes against signature databases

Hypatia-Test-Signature in ~/Download/hypatia-test-file.png
 Checked all SHA-256 hashes against signature databases

Scan completed in 2 seconds @ 123MB/s!
```

Figure 32, Sample Malware Test 2

11.4 Project Diary

05/10/2022

Cloned the GitLab Repository and Created my Diary.md and Created my Planning folder and moved relevant files such as my project plan into the folder.

06/10/2022

Completed My Project plan and uploading Submitted version to Repo.

11/10/2022

Found a couple of FOSS Anti-Malware for Android that I could potentially use for Malware Detection in my project. Currently (and over the next couple days) comparing the 2 apps to each other to see which one is better suited for my project and whether they both tackle Anti-Malware in different ways and if there is any benefit to implementing to finding a way to implement both into my project. For example, from my understanding, Hypatia focuses more on individual files on the device such as pictures whereas LibreAV focuses more on scanning the actual apps on the device. This means that theoretically, there is a security benefit to implementing both Open-Soruce malware scanners in my Security Suite. Currently am waiting on a response from the organizors as to how to correctly 'fork' the Open-Source Projects from Github to my Gitlab Repository.

13/10/2022

Have Received guidance on how to implment Open Source Projects from Github into my Gitlab. 'forked' the 2 FOSS Anti-Malware for Android Applications into my gitlab and added them as a submodule of my main project folder. Added my supervisor and the organizers to both submodules to ensure everyone who needs access has access. Now that this has been completed, I am able to start looking through the FOSS apps and understand how they work and how I could adapt them to work in my security suite. I have now setup Android Studio and both apps in Android Studio and am able to run both sucessfully on a virtual Android device running latest API of Android. Now will go through each version of Android to try and decide what is the earliest version of Android I can work on whilst keeping everything compatible.

16/10/2022

Have Tested both FOSS implementations for Malware Detection on multiple version of Android and determined that LibreAV will run on Android 4.1 and above with Hypatia running at Android 5 and above. Using these findings, I will be aiming to use Android 5 for the remainder of my project. This will ensure that I am able to support 98.8% of Android Devices according to Google API Version Distribution Chart. Have also begun going through the code of the FOSS and commenting in order to help me understand the code as their are no comments in the code. This should be completed in the next couple days at most.

18/10/2022

Have finished Commneting both FOSS projects to a point where I can understand and modify in order for them to work within my security Suite. Have also Created UML class diagrams for both which should help with this process. Have now begun thinking about how i am going to implement both together.

19/10/2022

Have begun (nearly completed) Implementing a Progress Bar into the Hypatia FOSS project. This will allow the user to see the progress of the scan and also allow me to implement a cancel button to stop the scan if the user wishes to do so. I chose to do this as I belive (for now) that a progress bar would be more effective than the orignal app's implementation of just printing out the logs into a Textview on the screen. Using a progress bar on both will also allow me potentially merge their implementations into one and have a single progress bar for both scans aka Hypatia will run up to 50% on the bar with LibreAV finishing the bar off. This means I can implement both into 1 seamless process for the user. Outside of code, I am still drawing up implementation ideas as to what is the best way to merge them together whilst making everything as cohesive as possible for the user.

20/10/2022

Completed corretly implementing Progress Bar in Hypatia UI. Struggled a little bit as was trying to find where the relevant data I need to track progress was. Over the weekend, will attempt to merge Hypatia and LibreAV together, However if proven difficult in time allocated, shall focus on getting 1 app to work as a proof of concept with a unified UI in order to save time.

22/10/2022

Decided to start over in implementing both apps into 1 app as my previous plan was taking too long. So now i have started my new implementation where I am making progress. Have also begun thinking about overall UI Design.

24/10/2022

Have created a UI page that is able to start the activities of another 'app package' and have implemented Hypatia. However, am yet to implement LibreAV however this should be completed by tommorow as I have streamlined the process and now know exactly what I am doing. Once LibreAV is implemented, I will focus slightly more on the UI looking a bit more conisistent and more user friendly before Wednesday. Have also begun looking into open source implementations of an app that is able to modify app permissions. I understand that more likely that not, I would need to make my security suite an admin on the device it is running on for this. I know how I can retrieve a list of all apps and then a list of each permission on each app. Just have to look into how I can change permission without having to send the user to the Android Settings page of that app. This shouldn't take too long and I am hoping I can quickly get back on track with my project timeline within the next week or so.

26/10/2022

Completed Merging LibreAV classes and resource files with Hypatia in order to get both implementatios to work in 1 app rather than have 2 seperate apps. Have run into an error using this app however where LibreAV is only showing the splash screen then crashing. I have a potential fix for this problem though which I will be trying tommorow. The UI currently consists of 2 buttons in the centre of the screen, 1 for hypatia and 1 for libreAv and the user decides which one to run. I have not yet implemented a back button in Hypatia/libreAV meaning user needs to restart app to change scanner. I am looking to add this soon however is not a priority due to it only being a proof of Concept. As the Malware Detection aspect is most likely to be a big part of my interim presentation, I will be working on refining its UI throughout the rest of the weeks however this is not a priority until closer to the interim presentation.

27/10/2022

Have resolved issue where LibreAV would crash after showing the splash screen. So now both Implementations are working succesfully in 1 app.

28/10/2022

Modified the UI of Hypatia slightly to be more consistent with LibreAV and the main menu. Have also removed the Splash screen for LibreAV which has made the app feel faster. I am in a position where i can declare this proof of concept complete and will be shifting almost my full attention to creating my proof-of-concept app for Viewing and Modifying App Permissions. This shouldn't be too difficult as LibreAV already gives me the foundation of this feature as it is able to seach apps and retrieve all their permissions meaning that viewing permissions is pretty much complete. Just need to think of implementation that will allow user to edit permissions potentially having to make the app as a administrator. I am hoping to make progress on this over the weeekend and be able to start my Active Sensor Usage POC early next week as well as make progress on my interim report as well as my early deliverable.

29/10/2022

After some Preliminary Research for Viewing and Modifying App Permissions, I'm slowly edging to the conclusion that an app that can modify app permissions may not be possible without the user having a rooted device or using ABD. Which may make this avenue I would need to simplify or drop. I am considering still pursuing a simplified version of this concept where the app will allow the user to view their permissions as well as group apps by permissions and if the user would like to change any permissions, the app could potentially redirect them to the built-in Android App Settings page where they can change the permissions from Settings which will have the ability to do this. I still feel like this could potentially add value to the user as they will be able to see the permissions for all their apps rather than click on one at a time. However as these permission lists can be very long, I would need to think long and hard about the most effective UI/Filtering System.

31/10/2022

Completed the UI for the PermissionsPOC App. Was a bit more complex than expected as need to make a card for each app that is clickable. I achieved this by using a card view with each card having a picture of the app Icon and some information about the app such as the name and package name. Have also nearly completed the 1st couple sections of my Interim Report (Timescale, Abstract, Aims & Objectives). I believe I will neeed to clarify a new up to date timeline for 2nd term to reflect the slightly slow progres I have been making. That being said, i am still on track to catch up, i just feel like I need to be slightly more generous with my timescale of each task now that I have a better view of how long each task will roughly take.

03/11/2022

The Backend for the PermissionsPOC app is taking a bit longer than expected. I have finished implementing all the necessary java code to make the views work as well as any adapters needed for the data regarding the apps. Just need to make it all work together. Really Hoping I can get this POC done by the end of tommorow at the latest as it is taking longer than expected. After the completion of this POC, I will instantly start creating my App Access Control POC which will lock certain apps until the user authenticates themselves. Hoping this would be done as well by early week at the latest. I am continuing to work on my Interim Report and just about to finish Section 1 and 2 and move on to writing my report about which features I could implement in my Security Suite.

06/11/2022

Have finally completed my Proof of Concept for App Permissions. This POC has taken a little longer than expected and will consider reorganising my timeline in order to provide myself more time per Proof Of Concept. However I have learnt a lot about ANdroid development which i didnt know and am hoping this new found knowledge will speed up development. Will attempt to complete the App Access Control POC as quick as possible (Thursday at the latest) and then I can switch my focus to the report, demo and presentation. Depending on how quick i can get my 3rd POC complete. I may attempt to complete a POC for file Encryption on Android but this is not confirmed. Have also bookmarked a few open source git hub projects that relate to app access control. The App Access control will most probably work by showing a lockscreen to the user on the applications they choose to require verification.

07/11/2022

Realised a mistake when sorting out my Resource files for my App Permissions in order to allow the app to work on Android 5 Devices. Have researched into ways I could implement App Access Control. As there is no direct way to listen to when an app open or closes so, from what i can tell, is to use a Usage Stats Manager to get the app that is at the top of the launcher which is most likely to be the app that the user has just opened.

16/11/2022

Have begun writing my background theory (which features I could implement in my Security Suite) and have made good progress on this report. Have looked into possible implementations and have begun programming my POC for App lock. This will most likely be my final POC of this term as I focus more on my report and my Demo and Presentation.

17/11/2022

Made progress on my 3rd Proof of Concept, is taking a longer than expected as I didnt anticipate how different usage stats worked between different versions of Android. I am now considering looking at using an Open Source project to fulfill my app access control.

18/11/2022

I have found potential candidates that I could use for app acccess control from github and just going through their feature set and evaluating their suitability as well as ensuring there are no licensing issues with the rest of my project.

19/11/2022

Have settled on an Open Source Implementation of App access control that I will be using as part of my Proof of Concepts.

20/11/2022

Completed my early delvierable report for my interim report and have begun looking for an open-source implementation/ way I could implement an app that alerts the user when any service on their device is using mic or camera (Active Sensor Usage). Have multiple section of my report left to write and am trying to get as much completed before draft look with my supervisor.

21/11/2022

Made more progress on my FYP interim report. Have begun and completed around half of my 2nd Section (Technical Achievements). Have questions about the report which I shall ask to my supervisor in Our meeting on Wednesday. Will get the demo of the Permission manager ready in order to show my supervisor in the same meeting.

22/11/2022 - 25/11/2022

Continued work on Final Year Report

28/11/2022

Begun New project on Android Studio to merge all of Proof Of Concept's together. Hoping I can get all merged by wednesday where I can record demo video on Thursday ready for submission. Continued work on Final Year Project

29/11/2022

Continued work on Final Year Report

01/12/2022

Completed majorirty of Proof Of Concept merging and completed report and demo video.

22/12/2022

Realised that I forgot to actually push my Interim Submission and AppLockPOC to the Gitlab so have done that. Over the past few days I have been looking into potentially using Google Flutter in order to create my UI to see if this would make the creation process any quicker and make the UI look more seamless and clean.

09/01/2023

Have continued to learn Google Flutter since last entry. Have now Gotten Maxlock to a state where it can be merged with the rest of the modules without causing any issues. Also modified MaxLock in order to allow it to be compatible with Android 5 and up instead of Android 6.1 and above. Will Finish Merging modules into 1 app tommorow and then full focus on UI.

10/01/2023

Have successfully merged all modules into 1 app with a basic menu to navigate between them. Have tested this with Android versions ranging from Android 5 to 13. Am going to add OnBackPressed functions before merging branch just to make navigating back to the main menu possible as well as change colours back to the intended colours rather than red. Still considering whether to commit to Flutter or not and am hoping to make decision by end of the week. I would only use Flutter for the main menu however it might be easier to use standard Android UI methods in order to keep everything consistent.

12/01/2023

Have fixed bug where pressing backspace from any of the home pages of each module to allow it to work.

14/01/2023

Been researching into how I can implement a way for the user to pick which folders they want Hypatia to scan rather than have Hypatia scan all the files on the device which takes a lot of time and battery power on the device which is an issue I ran into when testing Hypatia on my own personal device full of files rather than on the relatively empty emulator.

15/01/2023

Debugging Hypatia in order to find out where I could modify which folders for the scanner to include in it's scope. This is determined in it's main Activity where I can modify it to select specific folders. Still researching into how a user can select folders to scan but until then considering changing scope to include specific folders most likely to store downloaded files from the browser such as the Downloads and Documents Folder.

17/01/2023 - 22/01/2023

Researched into various methods and libraries I could use in order to allow the user to select specific folders to scan on Hypatia.

25/01/2023 - 29/01/2023

Created Proof Of Concept's of verious selectors but was unable to figure out how to pass a list of folders back to app. Then realised that I was trying to over engineer the solution and could just create a simple alert dialog with a list of User Folders (Documents, Downloads...) within the app and simplify the process significantly and avoid the need for a new library.

30/01/2023

Completed Implementation of the above and Merged Branch into Main.

31/01/2023 - 02/02/2023

Realised that my Permission manager scanner needed to be optimized as pressing the button on the main menu can lead to the app to crash due to the scanner not being able to finish in time. I have moved when the scanner starts to as soon as the app is ran on the device which has reduced the delay between the user interacting to the list being shown.

03/02/2023

Have begun working on the UI for the app and making the app look clean. Will start with the Permission Manager as it is currently easiest to modify in order to create a base theme. Have also begun using Figma which will with implementing Google's Material 3 design language.

05/02/2023

Have realised that using Figma would of slowed down my development so have limited use to jus organising components rather than implementing Figma into Android Studio.

07/02/2023

Created and added Toolbar to all aspects of the Permission Viewer to improve design. Redesigned the List page in order to make the list of apps look nicer and more understandable to look out

08/02/2023

Added a Back arrow to allow user to go back to main menu. Redesigned the App Details to make it easier to read for the user. Added a refresh button to the toolbar which allows the user to update their list of apps.

09/02/2023

Completed Branch for turning Permsision Viewer into Material UI. Have begun the same process on LibreAV

10/02/2023 - 11/02/2023

Completed desgining new LibreAV and begun implementing design into the app. Updated the Result List in order to match new design and be more material.

12/02/2023

Begun Modifying Toolbar in order to be more in line with rest of app and have modified Result list more to make Safety Classifications more obvious.

13/02/2023

Completed Result list and Toolbar and have also modified the scan progress bar to match Google's material design Guidelines and look more consistent with the rest of the app.

14/02/2023 - 17/02/2023

Completed the UI for the App Scanner module of my security suite. Have begun working on the file scanner UI which will take a lot of work.

18/02/2023/ - 22/02/2023

Created progress bar that links up with scanner to show user the progress of the scanner and completed the majority of the file scanner's UI.

05/03/2023 - 07/03/2023

Begun Working on my main menu layout by adding a widget to show user storage stats.

08/03/2023

Implemented Storage stats and outputting total/used/free space. Made buttons bigger in Main Menu to use more of the screen.Scan Apps button now shows the last time the scan was executed. Set up toolbar for Main Menu.

10/03/2023

Have begun focusing on my Final Report and adding/planning any headings that May of been missing since Interim. Have also started reviewing my interim report to find which sections that need to be updated/rewritten. Have begun looking into and implmenting Espresso Testing for my Android app.

11/03/2023

For a reason unknown to me, Espresso Testing seems to break my entire app and cause it to fail whilst compiling due to mismatch of kotlin versions. I have been unable to find a solution to this issue both using my own knowledge and online so have resorted to manual testing of my app. I have however used Firebase Test lab in order to test whether the app runs correctly on a number of Android OS versions ranging from API 21-33.

12/03/2023 - 15/03/2023

Have been fully focusing on my report.

16/03/2023

Have realised that I never set up Toolbar in Maxlock to be in line with rest of app. Have now done this however back arrow doesn't appear to show. Will need to investigate this before submission.

16/03/2023 - 27/03/2023

Have been focusing on my report.