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

December 02, 2022

Table of Contents

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 each handling a different aspect of security on Android devices.

Throughout this report, I will be delving into the full development process from the background theory I have accumulated throughout the term, to the deliverables and proofs of concept that I have created along the way. Additionally, I will discuss my aims and objectives for undertaking such a project, as well as my motivations to do so before moving onto my Literature review.

The purpose of my literature review is to write about how I aim to achieve my desired implementation of an Android Security Suite in a more high-level technical aspect, placing particular emphasis on theoretical and practical research that I have done towards the project. Following my literature review, this report will talk about my background theory which is essentially what modules I implement into the security suite and an investigation of how I have applied the theory and research in the development process.

The technical achievement section explores any new technologies that I have had to use to achieve my current progress in the project. Finally, I discuss how I used Software Engineering principles and methodologies, and how I have followed standard conventions such as Test-driven Development. Before ending the report with my conclusion and references.

2. Introduction

2.1 Project Specification

Please find Project Specification provided by Royal Holloway University of London in Appendix 9.1.

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
- User-Friendly
- Up to date using the latest security definitions
- 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/Management of app permissions
    - File Encryption (.zip, .rar encryption)
    - App Access Control

2.3 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 we become increasingly reliant on the palm-sized slabs of glass in our pockets, and 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 help keep our phones 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 until the variation in the number of years being supported from manufacturer to manufacturer is realised. For example, Google, which has a 0.5% market share (appbrain.com, 2022), can promise at least 3 years of Security updates across all their devices with their latest smartphones offering at least 5 years (Google Support, 2022) whereas Oppo, with 10.1% market share (appbrain.com, 2022), uses a "the more you spend, the more you get policy"(Android Authority, 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 (Android Version Market Share Worldwide | Stat counter Global Stats, 2022). 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 because there is no full-featured Security suite that can act as an all-in-one solution for your security needs on your Android 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 something like 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.

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 in Term 1: the Malware Scanner, the Permission Manager, and the App Locker. For each module, I explored the theoretical insights into accomplishing the components 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 manifest analysis is that a practical implementation can be done 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).

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).

3.2 Permission Manager

Android applications request permissions from users during installation and runtime and, 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

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 when ever it is detected that the OS has killed it (Stack Overflow, 2022).

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 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

4.1.1 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 Tensor Flow 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.

4.1.2 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 Realtime 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.

4.2 Permission Manager

It is common knowledge that when a user wants to modify permissions settings for any app on their device's settings, they are unable to change all permissions and the settings app does not even show the user a comprehensive list of all the permission the app can 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 and Objectives for this Security Suite.

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).

4.3 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.

4.4 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 implementations 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, I am/will be looking to implement Material Design 3 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 in order 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. I use a receiver in my Malware Detection app 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 it's 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.

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 emulator's that were running older versions of Android. One of my objectives for this project is to make my security suite as compatible with 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 like 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 before compiling (Juned Ghanchi, 2021).

6. Software Engineering

6.1 Demo Video

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

https://youtu.be/GkrAKXzwSx8

6.2 Evaluation

In this section, I will be evaluating each Proof of Concept I have completed this term (Malware Scanner, Permission Manager, App Locker) as well as how well I have integrated each POC together into a unified security suite.

6.2.1 Malware Scanner

For my malware scanner, 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 an Android malware scanner that were fundamentally different in what they are scanning on the Android device.

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 my malware scanner proof of concept.

Overall, I would say that my combined malware scanner app that implements both LibreAV and Hypatia with a simple UI at the front that allows the user to choose which one they would like to run was successful. I will be looking into refining the UI of Hypatia throughout term 2 as the current implementation just prints console logs to the screen to show the progress of the scan.

6.2.2 Permission Manager

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 hides 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 proof of concept a

success as I have a permission manager 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.

6.2.3 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 device. 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. Unfortunately, despite setting up the source code in Android Studio and ensuring that I can build and deploy MaxLock myself, I came short on time and was unable to implement MaxLock into my current draft of my security suite. This is something I am looking to complete over the holidays.

6.3 Methodology

After identifying my deliverables, by virtue of creating an Android application, I instantly settled on using Java as my programming language. This is due to Java being one of my more confident languages which I have used to develop android application in the past. 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 Manager 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 Manager include 'XML Pull Parser'. This package parses and filters xml files. In the context of the Permission Manager, 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 Manager 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.

With regards to Software Development principles, although I have no record of any formal testing, I made sure to continuously unit test each section by assessing it in accordance with my requirements. A potential framework that I could use in the future to structure formal scripted testing, could be Roboelectric. It is a recommended fast and reliable framework that allows for Android app unit testing in the JVM workstation. Finally, code commenting enables readability and a logical understanding of the program for any passing developers. It is also a fundamental example of a form of pair-programming which is often demonstrated in team-based development environments.

6.4 Project Diary

Please find the Project Diary in Appendix 9.2.

7. Conclusion

To summarise this report, I have achieved most of the goals that I set out for myself at the beginning of the project and for any goals I didn't manage to achieve, I am in a good position to efficiently reach them throughout the upcoming festive holidays and the beginning of next term. Over the Christmas period, I will be focusing on getting the Applocker implemented into the security suite and once this is complete; investigate refining the User Interface more to make it look more up-to-date and modern. Over next term, I will expect to stick to my Term 2 plan where I will be focusing on the User Interface and Testing of each module. I would like to investigate implementing one more module over Christmas, but this depends on whether I can find the material required to make it myself.

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.

Relating back to my Aims & Objectives, I believe that I am on track to achieve the majority by the end of Christmas and all by the end of the year. I have been placing a particular emphasis on ensuring that the security suite requires no prior user modification and has the ability to regularly keep these security modules updated to ensure security and increase the longevity of the Android device going into the future.

Despite not having a consistent UI and missing out on implementing the App locker into my current draft of the Security Suite, I consider this term to be a success and I have learnt a lot of new techniques and functions regarding managing security on Android, that I will be using throughout the rest of the project to finalise the security suite. I look forward to the trials and tribulations of development to come, and to the theoretical research opportunities on the horizon.

8. Bibliography

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.

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].

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

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

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.

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].

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.

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.

9. Appendix

9.1 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

9.2 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.