Android Security Development: SpywareDetection, Apps Secure Level and Data Encryption Improvement

Lim Wei Xian[1], Chan Shao Hong[2], Yap Ming Jie[3], Azween Abdullah[4], Mahadevan Supramaniam[5]
Taylor's University Lakeside Campus, School of Computing & IT (SoCIT), Subang Jaya, Selangor, Malaysia[1, 2, 3, 4]
Research and Innovation Management Center, SEGi University, 47810 Petaling Jaya, Selangor Darul Ehsan, Malaysia[5]

***Abstract*—Most Android users are unaware that their smartphones are as vulnerable as any computer, and that permission by Android users is an important part of maintaining the security of Android smartphones. We present a method that uses manifest files to determine the presence of spyware and the security level of apps. Furthermore, to ensure that no leaked data occurs in Android smartphones, we propose new method for the encryption of data from Google Suite applications.**

***Keywords*—Android; spyware detection; security level index; data encryption**

I. Introduction

In 2017, Android accounted for 85% of the smartphone market, and the Android operating system was also the most popular. However, Android is an open-source operating system that is often targeted by malicious software. In 2017, there were more than 3.5 million malware applications [1]. Since the development of Android 6.0 in 2015, Android has required permission for apps considered dangerous, and users can revoke this permission at any time [2].

The aim of the "Android Security Development" project is to provide a safer environment for Android smartphone users by detecting spyware more efficiently and effectively, prevent the leakage of personal information from Android smartphones, and raise awareness regarding permission for apps downloaded by Android users. Personal information in smartphones may include contacts, calendar schedule, and location, to name a few.

The first goal of this project was to prevent users from having spyware implanted in their smartphones and to prevent users from downloading malicious applications that cause their personal information to be leaked. As users may not be aware when their smartphones have been implanted with spyware or when they are downloading a malicious application, requiring permission does not effectively protect Android users' smartphones. As such, a spyware detection system is urgently needed that can detect and prevent malicious applications from being downloaded. We developed a Spyware Detection System than can alert users that a specific application has been implanted with spyware.

The second project goal was to provide an application security level index for users to access details about applications. With this index, users can assess the level of risk associated with using applications and be more informed regarding the permission request. The Application Security Level Index is a software program that can produce a report about the type of permission required, as well as the risk of specific data being leaked if permission is given for the application. The Application Security Level Index will be in place before the application is available at the Google Play store.

The third goal was to implement Hybrid, which encrypts data from Google Suite to Google's server. We called it Hybrid because the process of encryption uses two encryption methods, both the Advanced Encryption Standard (AES) and the RSA encryption method. AES performs symmetric-key algorithm encryption and RSA asymmetric key algorithm encryption. With these two encryption methods, the data being transferred can be made sufficiently secure. Hence, users need not worry that their data is being leaked during its transmission.

In summary, the "Android Security Development" target audience is all those who use Android smartphones. With these three implementations, we provide our target audience with a safe and more secure environment for their Android smartphones. This paper is presents in ten sections, including the Introduction, Related Work, Architecture Diagram, Method for Detecting Android Spyware, Method for Implement Application Security Level Index, Method for Hybrid–Cryptosystem, Experiment Setup, Experiment Results, Critical Analysis and Conclusion and Future Work.

II. Related Work

A. Android Permission Mechanism

Android is an operating system which used widely on billions of different devices, such as smartphones, tablets, wearable devices and intelligent appliances. However, such flexible supply of applications which causes vulnerable applications and malware easily obtains by users [3].

Various security mechanisms used in Android such as sandbox and permissions to solve Android related security threats. However the results of these security mechanisms are not satisfactory, as the malicious activities still targeting the Android applications. Android has improved the permission scheme since Android version 6 Marshmallow benefiting Android current users [3].

The permission mechanism in Android is to achieve a better security to Android platform. The permission mechanism is designed to separate the system and the

applications, which application have limited access to the system. The permission mechanism is essentially mandatory in Android control system based on permission labels, which will check the specific application, have the specific permission when attempting to access the protected resources such as gallery, phone and contact. Therefore the applications which needed 35 permissions is require to declare in its AndroidManifest.xml files and is mandatory to receives approval from the users to use the protected resources [3].

Furthermore, there is around 140 standard permissions in Android for protecting corresponding resources in an Android device. All the 140 permissions are classified into several categories which based on its sensitivities. Dangerous permissions are further grouped based on the functional relationship, for example, "READ_SMS" and "RECEIVE_SMS" permissions are comprise of "SMS" group. Moreover, permission system is workable on third party applications if their developers self-created permissions or apply standard to the interface of their applications [3].

Although, Android have permission system, permission protected resources are still contains vulnerabilities. Permission leak vulnerabilities is quite normal in third party applications, since the application is written by developers which are insufficient security background [3].

B. Android Request App Permissions

According to the Android developer website [2], every Android mobile application that requires a permission must put a <user-permission> element in the app manifest at the top level in the project view as a <manifest> element. For example, an app that requires permission to send SMS messages would have a code in the manifest such as that shown in Fig. 1.

Basically, the Android permission system is divided into various protection levels based on the sensitivity of the app requiring permission. Some permission that are considered "normal" or that must use permissions is not affected very much by the system. However, if permission is listed as "dangerous," the system will prompt the user to explicitly grant the app access. The protection levels of Android permissions that affect third party apps are categorized as either normal, signature, or dangerous. These protection levels are also affected whether or not a runtime permission request is required.

The ability of users to revoke their permission for any app at any time became available only with the introduction of Android 6.0 (API Level 23). For example, if gallery permission was given by a user for an application yesterday, it would only be valid for that day. If the application wanted to access the gallery again, it must request permission once again.

In another example, if an application wanted to request permission to access the calendar, a method known as the "ContextCompat.checkSelfPermission ()" method is called, as shown in Fig. 2.

If the corresponding app has permission to access the calendar, the method shown in the figure above will return PERMISSION_GRANTED, and only then can the application proceed to make changes in the calendar. However, if the corresponding application does not have permission, this method will return PERMISSION_DENIED, and the application must explicitly ask for user permission.

The reason Android has implemented this permission mechanism in the developer is to allow users to know which information apps are accessing their data and why corresponding apps need to access it. For example, if a user frequently denies permission requests by an app, this probably means that the user does not understand the reason the application is requesting such permission, and the user considers that the app does not need this access.

C. Android Permission Groups

The Android web area [4] shows that Android categorizes all of its permissions group by group. With our proposed system, permission requests are in charge at the group level and single permission groups correspond to several permission declarations in the app manifest. For example, the CALENDER group includes both READ_CALENDER and WRITE_CALENDER declarations. Fig. 3 shows an architectural view of how a permission group works [4].

```
<manifest xmlns:android=
          "http://schemas.android.come/apk/res/android"
          package="com.example.snazzyapp">

    <uses-permission android:name=
          "android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>

</manifest>
```

Fig. 1. Code to Request Permission [2].

```
if (ContextCompat.checkSelfPermission)thisActivity,
    Manifest.permission.WRITE_CALENDAR)

    != Package Manager.PERMISSION_GRANTED) {
    //Permission is not granted
}
```

Fig. 2. Code to Check for Permission [2].

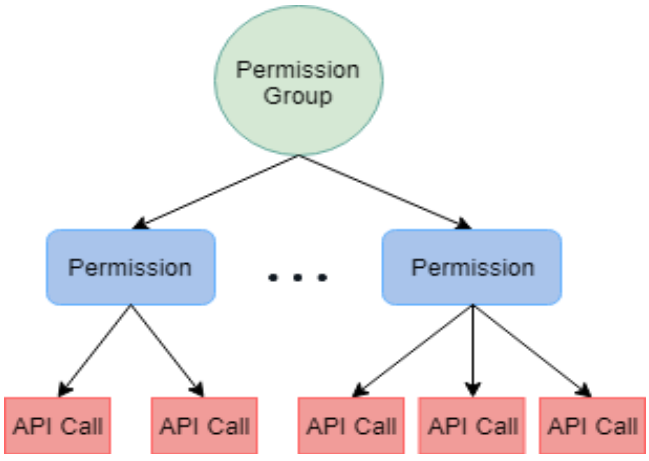


Fig. 3. Permission Group Architecture [4].

Permission groups in Android include all dangerous permissions. Although any permission that can belong to a permission group is assigned a protection level, a permission group only affects dangerous permissions that might affect the user experience, such that the system will protect the user's privacy.

D. Structure of Android Application Packages

Android applications are in APK file format. Fig. 4 shows that each APK file contains four important files, which include AndroidManifest.xml, classes.dex, META–INF, and resource files. Of these files, AndroidManifest.xml and classes.dex are often used in evaluating and analyzing threats and vulnerabilities [5].

The AndroidManifest.xml file contains application information described in XML, which is stored in binary form. Android Studio and Apktool can extract information from the AndroidManifest.xml file. Table I shows the main information stored in AndroidManifest.xml [5].

Android permissions in AndroidManifest.xml are categorized into three levels: normal, dangerous, and signature. Dangerous permissions will require approval from users. The version of Android operating system being used will determine the number of permissions requested [5]. Forty-four Android apps are in Java language and compiled in Java bytecode. These Java bytecodes are translated into Dalvik bytecode and stored in Dalvik executable format (DEX), for example in classes.dex. Dalvik bytecode enables code analysis without the use of source code, and it is also reverse-engineer friendly. APK files are stored in binary and since they are zip files, APK files cannot be analyzed directly. The Apktool has the ability to convert AndroidManifest.xml into text. So the bytecode in classes.dex can be reverse engineered to produce Smali code, a type of bytecode that is in human readable style and is useful for analysis [5].

E. Sandbox

Sandbox is a security mechanism for isolating app resources from each other to reduce system failures or the spread of malicious applications and to protect apps and the system from other malicious applications [6][7]. In general, a sandbox will allow an application to run in an isolated computing environment with limited resources. To use sandbox, Android assigns a unique user ID for each Android application and allows it to run its own process.

Sandbox is frequently used to run untrusted code or unverified programs obtained from third market applications that may contain malware. Typically, to run unfamiliar applications, sandbox will control resources, such as limiting the space for memory and permission access. In Android, the programmer must manually code the application that runs within the sandbox, so that the application will not be able to perform any unpermitted actions such as reading smartphone information without permission or any other malicious actions.

For example, if application A is downloaded from an untrusted source and tries to perform a malicious action, such as accessing a smartphone contact or gallery without permission, the Android operating system will prohibit this action since application A does not have the required permission.

F. SafeGuard

Safeguard is a real time anti-malware application that detects and blocks suspicious or malicious actions and behaviors. The SafeGuard database frequently updates types of malware threats and blocking rules. In general, SafeGuard monitors all applications that are running on the Android operating system in real time. If the SafeGuard library detects behavior that uses an API or combination of APIs, the database will detect it and alert the user [8].

As Android is an open-source operating system, its security is weaker and more vulnerable to attack. Most Android applications use Java as the official programming language, which makes it easier to use reverse engineering to allow the injection of malicious code and rewriting of code. This means Android users are at greater risk than those who use Apple's App Store [8]. Although normal signature-based detection can be used to easily detect malware from source code, malware is evolving rapidly. In addition, Android has developed and applied a new security model called Sandbox that prevents access by one application to other applications, based on the unique share ID created for all applications and those running in the virtual environment.

SafeGuard detects suspicious APIs such as accesses to GPS, conversation histories, galleries, private information etc. in real time. It then instructs users to block those malicious behaviors to protect their personal information. In addition, SafeGuard expands the reach of the behavior detection mechanism corresponding to the malicious behavior type against the target application and API behaviors. Moreover, SafeGuard is constantly being updated via the Internet to keep the database up to date and able to deal with the limitations of anti-virus software and prevent malicious activity by malicious applications. To deal with new malware that dynamically fetches codes, a heuristic detection method has been proposed that detects both original and dynamic codes. In existing mobile anti-virus software, old malware can be easily detected, but new malware is difficult to screen. To address this problem, SafeGuard monitors application behavior and the calling of malicious APIs in real time. If an application breaks a behavior-based rule, SafeGuard will block the application from running.

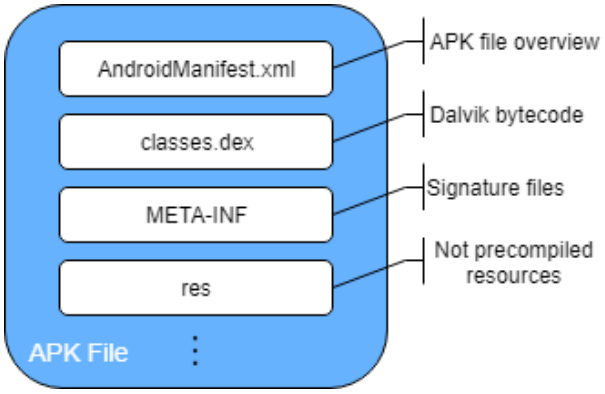


Fig. 4. APK File Structure [5].

TABLE I. ANDROID PERMISSION

Tag Name	Content
Application	General configuration of application, such as icons, labels, and display theme
Uses-sdk	Range of API levels needed to run the application
Uses-permission	Permissions requested by the application
Uses-library	Libraries used by the application

G. Types of File Analysis

There are two main major file analysis method, which are static analysis and dynamic analysis, which combine can become hybrid analysis. Static file analysis can inspect the files inside an APK file. AndroidManifest.xml and classes.dex contain data which are suitable use for analysis. The permission request information which are in Android Manifest.xml and can be extracted [5][9].

Static analysis is focus on application's code analyses without executing the code [10]. There are already quite many static analysers that can analyse Java source code, as we know that Android mobile application is wrote in Java language. Nevertheless, most of the static analysers are based on syntactical analyses or will use theorem providing some simplifying hypotheses. Unfortunately, most of the static tools do not support technologies such as XML, which will affect the control flow graph of an Android app, as we know that Android Manifest is a XML file type. However, Julia static analyser performs a semantic sound analysis. First of all, the apps are reverse engineer using dex2jar to able to extract the Java bytecode and will have apktool to extract the Android Manifest. The Android Manifest is use to determine the entry points for parsing the Java bytecode of the app [11].

The Julia analyser library provides a representation of Java bytecode which is suitable for interpretation (Mandal, Cortesi, Ferrara, Panarotto, & Spoto, 2018). Julia analyses the Java source code, which are already complied into Java bytecode inside Android Studio [12].

Dynamic analysis is the analysis that analyses the executing application on real time. It mainly focus on the behaviour of the application [8]. A dynamic analysis which presented by Taint Droid which it monitors the privacy of Android devices at real time by using privacy-sensitive data sources. Droid Box has extended the functionality of Taint Droid by modifying the Android framework; it can monitor the interesting API calls invoked by an application. It executes the application, and produce log of the behaviour that in the host operating system. After the executing which produces a more accurate analysis, however these approaches still contain problems which are overhead and require modification in the operating system and can cause a large part of Android users cannot use the system [13].

A type of Hybrid analysis called FlowSlicer, mixes a conservative static analysis with a dynamic analysis. The FlowSlicer allows a control over Android malicious applications with lower overhead and high accuracy. The idea behind FlowSlicer is that the static analysis use in filter elements that are important, while the dynamic analysis is use during the executing of application. The techniques used in the static analysis are instrumentation and program slicing are used while the techniques used in dynamic analysis is a tagging architecture [13].

Program slicing is a type of static analysis that has been used in many different purposes, such as information flow, software maintenance, program analysis and optimization. Program slicing is used in FlowSlicer with the objective of filtering and identifying the possible information-flow leaks in order to do a better analysis. Program slicing is a technique that creates an executable slice of the original program. Only the needed statement from the original program will be slice out, known as slicing criterion. FlowSlicer will discover the dependencies of each statement present in the reachable methods [13].

H. Advanced Encryption Standard (AES)

AES is a popular and widely used algorithm [14] that replaced DES following a public call in 1997 by the U.S. National Institute for Standards and Technology (NIST). The reason Triple-DES was replaced is that it required that DES be run three times to complete the encryption process, which it is not efficient, so a new and more efficient standard was needed. AES is a symmetric-key algorithm that uses the same key for both the encryption and decryption of data. The security of AES is directly proportional to the size of the key and the security level. This means that the longer the length of the key, the stronger the security. However, when the key is long, it also becomes slower.

I. Rivest-Shamir-Adleman (RSA)

RSA is a cryptosystem that is popular for securing the transmission of data by generating a public and a private key that are mathematically linked to each other but cannot be derived from each other [15]. It is an asymmetric algorithm, meaning that it consists of two different keys, one public and the other private. The public key can be given to everyone, whereas the private key must be kept private or given only to authorized personnel. Public keys encrypt data that can only be decrypted by the matching private key.

RSA works by multiplying two large prime numbers to produce a difficult form such that decryption is infeasible. Even with the best computers or super computers today, breaching the security of data being transmitted remains infeasible due to its complexity and large size. As technology continues to improve day by day, the ability to factor larger and larger numbers has also increased. As such, increasing the strength of data security becomes directly proportional to the size of the key, whereby the larger the size of the key, the stronger the security.

J. Objective

In Google Play Protect, there are still many flaws in their machine learning in terms of Spyware Detection. Spyware might not always be active as there passively infecting in the system without doing anything harmful at all to the infected system. However, once given command, the Spyware will only send the file to outside system. Because of this, the Spyware Detection is to be implemented. Secondly, the main idea of implement Application Security Level Index, is because of the current Android application's permission is not obvious on the application page; hence the implementation of Application Security Level Index is to create awareness for all the Android users, which the current system have not implemented. Thirdly, the reason why this hybrid-cryptosystem is to be implemented is mainly because, they were only a simple encryption using Application Level Transport Security (ALTS) between the transmission channels of Android user's phone to the server of Google. And so hybrid-cryptosystem is to make sure that the transmission channels between Android user's phone and the server of Google to be secure, so that the information and data being transmitted through the transmission channel will be able to be secured also to be able to prevent any man-in-the-middle to be listening and stealing information and data.

III. Architectural Diagram

Based on Fig. 5, starting from the left, which shows the developer of an app or apps, an app is published on the Google developer console via the Internet, is then connected and configured with the Google Services Cloud Server. The app is in the file format of .apk file. The app will then go through our implementation, which is a software of combine the Application Security Level Index and Spyware Detection mechanism.

Next, our software will use Apktool to reverse engineer the apk file, after the reverse engineer the apk file will produce few files, but in our implementation, we just used the AndroidManifest.xml file for our Android Security Level Index and Spyware Detection mechanism.

Furthermore, will have two separate parts which are Application Security Level Index and Spyware Detection mechanism. Regarding to Application Security Level Index, the Android uses permission is extracted out from the AndroidManifest.xml, and our program will analyses the extracted uses-permission and produce a permission report. For the Spyware Detection mechanism, the program will extract specific information from the AndroidMnifest.xml. The program will then compare with a list of keyword list and produce a result which identify whether the apps is benign or malignant from spyware.

After completing this process, it will produce a permission report and download approve for the app, and the app with permission report and download approve will be passed on to Google Play and then be published in the Google Play Store. The Google Play Store in the end user's devices is connected to Google Play Services. So, if the application requires updates, it will provide information to Google Play Services, which is connected to the Google Developer Console so the app developer can update the application.

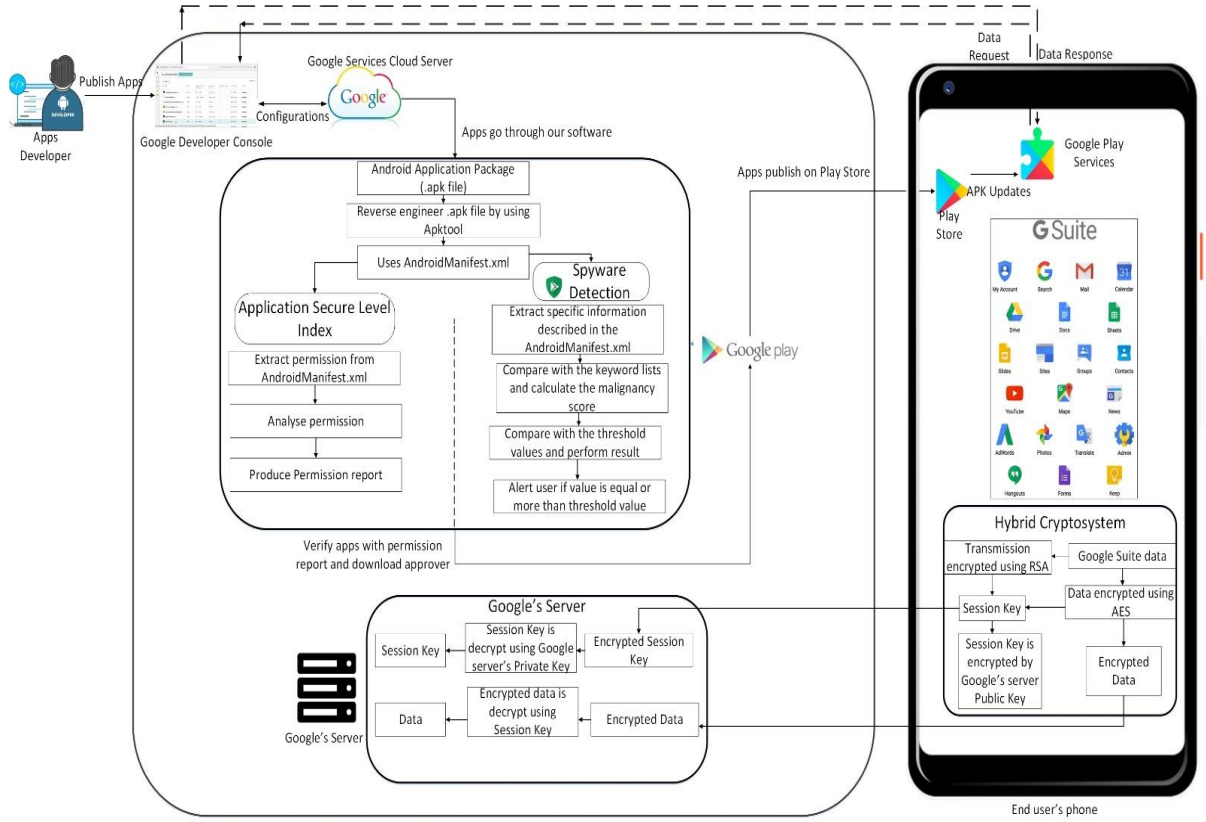


Fig. 5. Architectural Diagram of our System.

The last implementation, Hybrid Cryptosystem, is invoked when there are quite a number of Google Suite apps in Android OS devices that contain Google Search, Gmail, Google Account, YouTube, etc. These apps contain important information about the user of the smartphone. Hence, the Hybrid Cryptosystem will encrypt messages sent from the end user phone to Google's server.

IV. Method for Detecting Android Spyware

Our proposed method for detecting Android spyware analyzes AndroidManifest.xml files. The Android application package known as an APK file (.apk) contains the manifest file, application program for the Dalvik virtual machine (VM), and application resources. The manifest file takes the form of "AndroidManifest.xml," which occurs in all Android applications, while the application program is known as "classes.dex." Application resources contain pictures, music, and some xml files that provide layout information.

Android malware is detected by following the steps shown in Fig. 6:

- Extract specific information in the AndroidManifest.xml of the APK file.
- Compare the extracted information with that in the keywords list provided by our new method. Then, calculate the malignancy score of the sample by comparing the information in Step 1 with the list.
- Compare the malignancy score in Step 2 with the threshold values established by this new method. If the malignancy score exceeds the threshold value, the sample is judged to be malware.

A. Extraction of Information Items

Manifest files contain essential information about Android applications, such as the version number of the application, the name of a package, required permission, and the API level. The format of the manifest file is identical in benign and malicious applications. However, there are certain differences in the characteristics of several information items. In our research phase, we investigated benign and malware samples and obtained a total number of samples. We then selected specific information items that showed a wide variety of spyware as compared to benign applications. Based on our results, Table II shows six information items that are extracted from manifest files and used by our proposed method to detect Android malware. The items are represented as text strings or numbers.

B. Keyword Lists and Malignancy Score

With this new method, several keyword lists are compiled for an application. Benign or malicious strings in a manifest file are recorded in the keyword list. We generate four types of keyword lists: (1) permission, (2) intent filter (action), (3) intent filter (category), and (4) process name, as shown in Table III. Because items (5) intent filter (priority) and (6) number of redefined permissions are represented by an integer and not a text string, they have no associated keyword lists.

After we obtain the keyword lists, the malignancy score for the above four information items are calculated. This process is performed by classifying the keywords as either benign or malicious. The malignancy score is calculated using

Formula (1):

$$P = \frac{M-B}{E} \quad (1)$$

where P is the malignancy score, M is the number of malicious strings, B is the number of benign strings, and E is the total number of information items.

Of the five permissions listed in Table IV, READ_SMS, RECEIVE SMS, and SEND SMS are recorded in the keyword list and are classified as malicious strings, as shown in Table IV. Then, the malignancy score of this sample is calculated using

Formula (2):

$$P = \frac{3-0}{5} = 0.6 \quad (2)$$

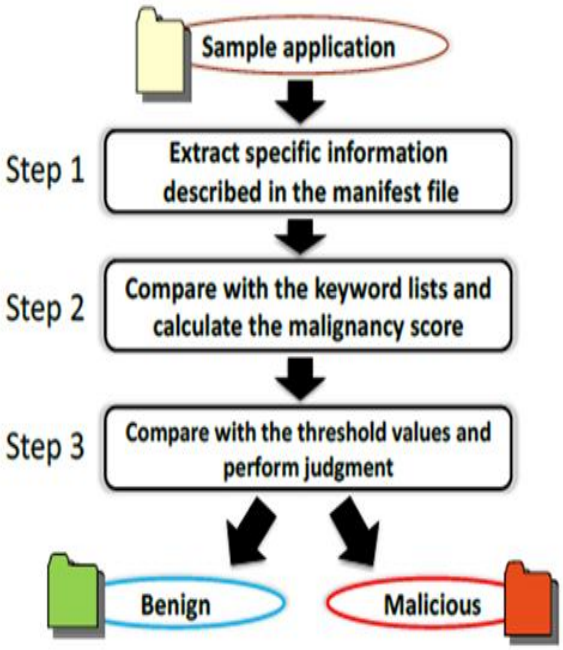


Fig. 6. Flowchart for Detecting Android Spyware.

TABLE II. List of Extracted Information Items

No.	Extracted information Items
1	Permission
2	Intent filter (action)
3	Intent filter (category)
4	Process name
5	Intent filter (priority)
6	Number of redefined permissions

TABLE III. KEYWORDS LISTS

(List 1) Permission	
1. READ_SMS	7. READ_HISTORY_BOOKMARKS
2. SEND_SMS	8. Write_HISTORY_BOOKMARKS
3. RECEIVE_SMS	9. READ_LOGS
4. WRITE_SMS	10. INSTALL_PACKAGES
5. PROCESS_OUTGOING_CALLS	11. MODIFY_PHONE_STATE
6. MOUNT_UNMOUNT_FILESYSTEMS	
(List 2) Intent-filter(action)	
1. BOOT_COMPLETED	8. Install_SHORTCUT
2. SMS_RECEIVED	9. left_up
3. CONNECTIVITY_CHANGE	10. right_up
4. USER_PRESENT	11. left_down
5. PHONE_STATE	12. right_down
6. NEW_OUTGOING_CALL	13. SIG_STR
7. UNISTALL_SHORTCUT	14. VIEW (benign keyword)
(List 3) Intent-filter (category)	(List 4) Process name
1. HOME	1. remote2
2. BROWSABLE (benign keyword)	2. main
	3. two
	4. three

TABLE IV. PERMISSION KEYWORDS IN A SAMPLE

```
<uses-permission android:name=”android.permission.INTERNET” />
<uses-permission android:name=”android.permission.READ PHONE STATE” />
<uses-permission android:name=”android.permission.READ SMS” />
<uses-permission android:name=”android.permission.RECEIVE SMS” />
<uses-permission android:name=”android.permission.SEND SMS” />
```

C. Thresholds and Judgement

The proposed method provides threshold values for the malignancy score. We use a data mining tool, Weka, to determine these threshold values. As with the four categories of information, the threshold values are set using the Weka J48 algorithm, which is based on a decision tree. We use both benign and malicious samples in the machine learning process.

Making a judgment about the safety of an application sample is based on conditions 1 and 2 and Formula (3), which are shown below. Condition 1 describes the characteristics of malware. Condition 2 is used to avoid incorrect judgments. In Formula (3), SCORE refers to the final malignancy score of the sample.

C1 and C2 are the number of items satisfied by a sample in conditions 1 and 2, respectively.

Condition 1:

- Malignancy score is greater than the threshold value determined by Weka.
- Count of intent filter (priority) is greater than the threshold value.
- Count of redefined permissions is greater than the threshold value.

Condition 2:

- Malignancy score of (2) intent filter (action) is negative (< 0)
- Malignancy score of (3) intent filter (category) is negative (< 0)

Criteria formula (3): SCORE = C1–C2

If the final score is greater than or equal to 1, the sample application is considered to be malware.

V. METHOD FOR IMPLEMENTING APPLICATION SECURITY LEVEL INDEX

Fig. 7 demonstrates on how “Application Security Level Index” works. Fig. 7 shows in a simple way to illustrate the main process of Application Security Level Index.

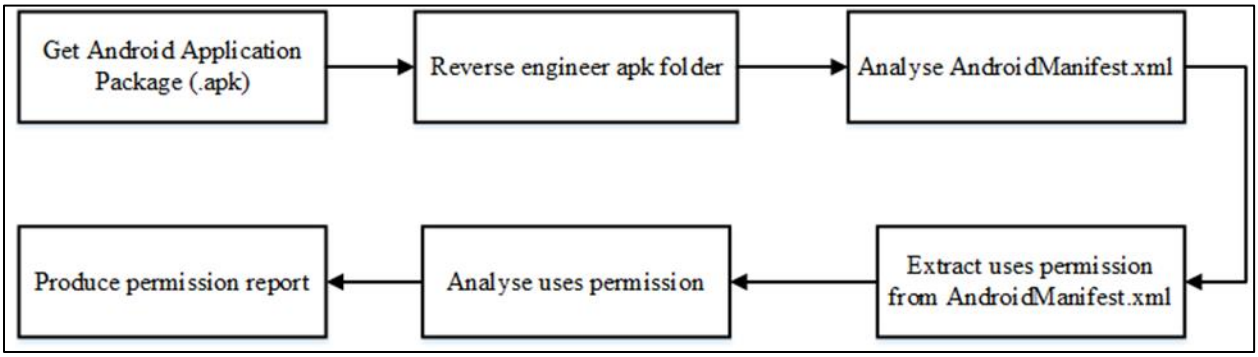


Fig. 7. Application Security Level Index's System Flow.

First of all, we will take the apk folder as an input to our system, the first step will be reverse engineer the apk folder, this is because apk folder is similar to zip folder, and cannot be unzip in a normal way, hence the purpose of reverse engineers the apk folder is to get the AndroidManifest.xml. The tool we used for reverse engineer the apk file is Apktool, which Apktool will produce the original folder, res folder, smali code folder, apktool.yml, assets folder, lib folder and AndroidManifest.xml. However, based on my system, I only need AndroidManifest.xml to analyses the Android uses permission of the apps. Hence, I will extract the Android uses permission from AndroidManifest.xml and yet analyses the permission uses. Based on the permission uses, the permission report will be ready to produce.

The purpose of permission report is to raise the awareness of Android users regarding to the permission uses for application that installed to their device. This is because based on my research, I found out that, majority of the users does not care about the permission requested from the app, as long the users can use the application. Furthermore, the nature of Google play store is also a problem. Apps description that in Google play store, does contain the permission that the specific apps require, however it is not easily to find the app permission from the description, users need to scroll all the way down to read more only can see the app permission. Moreover, the app permission does not tell the users that is the permission dangerous or normal. Hence users are not aware of dangerous permissions, such as calendar. Basically, with the app that request calendar permission, the particular app knows the schedule of users, if the users schedule is saved in the phone calendar. Here comes a bigger problem, with the technology today, people desire to make everything that around us to be simplified and convenient. Some of the smartphone users, store important data in their smartphone, for example, password, schedule, personal information such as identity card number, house address and etc. The examples above are important data to users. That's why in Application Security Level Index's system, is to raise the awareness of smartphone' users, we cannot prevent users from saving important data with their phone, but what we can do is, to provide a solution to the users, let them conscious about the permission uses in their smartphone.

If our system is implemented, the users can see the permission report at a glance of the app description in the Google play store. With the use of our system, we basically highlighted the uses permission in Google play store, to achieve our objective.

VI. Method for Hybrid-Cryptosystem

Fig. 8 is the flow of the hybrid-cryptosystem. The hybrid-cryptosystem will be using Advanced Encryption Standard (AES) and RSA (Rivest-Shamir-Adleman).

The client will first generate a secret key using the AES program automatically. Then, the client will be request for a public key of their partner, so that they will be able to encrypt the secret key that is generated which will be then to be sent to their partner.

The process of sending the secret key will be encrypted by the RSA program using the formula CT = PT^E mod N where N = p x q, and p and q are 2 large prime number. The public key will have its value produced by the formula gcd(ø(n), E) = 1; 1< E < ø(n) where ø(n) = (p – 1) (q – 1). Then the secret key received by their partner will be decrypted by their own private key using the formula PT = CT^D mod N, where value of the private key is produced by the formula (D x E) mod ø(n) = 1.

After the secret key has been received by the partner, the data that is needed to be sent to their partner will be encrypted using the secret key, and when the partner received the data, they will be decrypting it using the secret key.

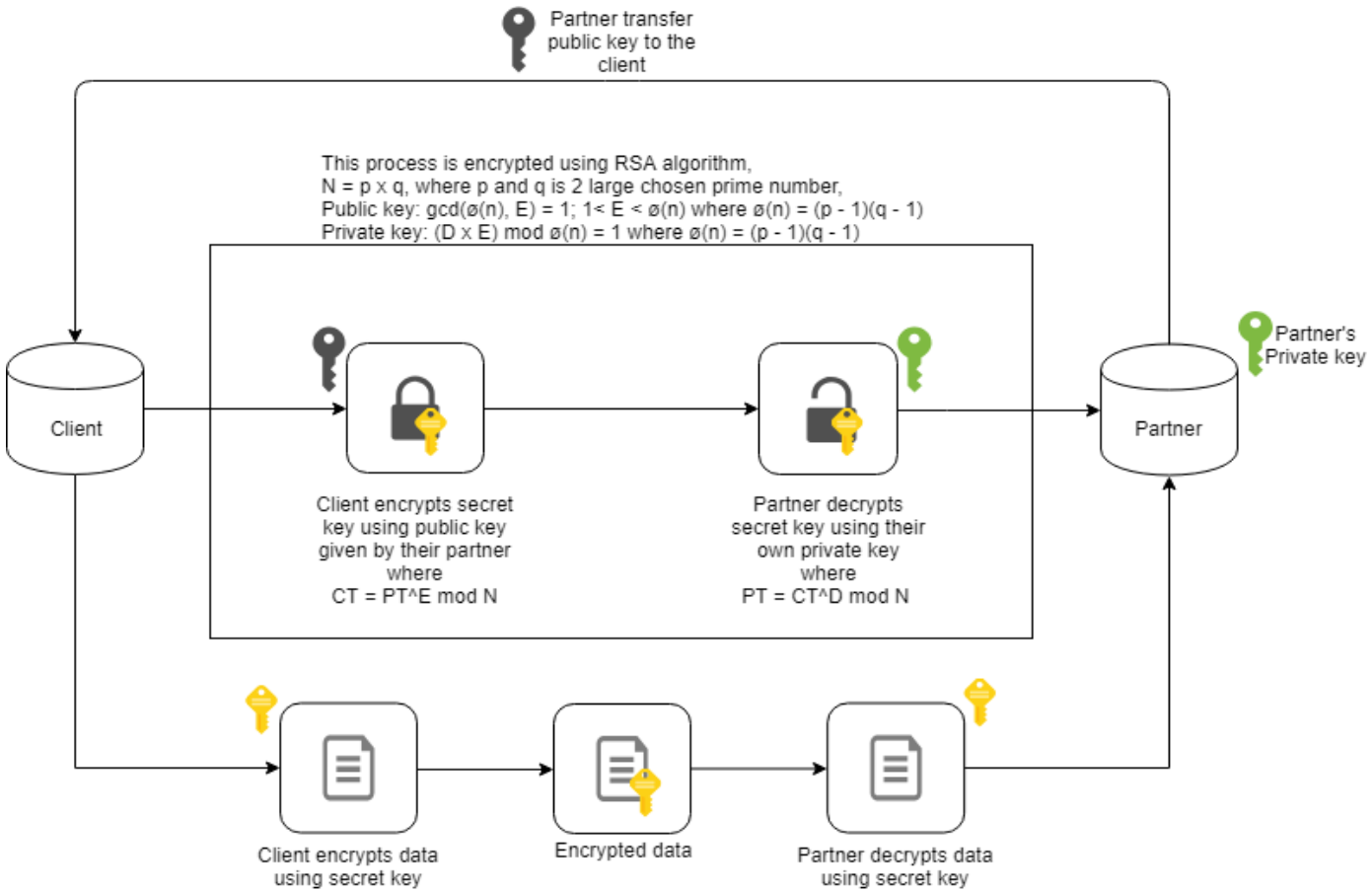


Fig. 8. The Flow of the System.

VII. Experimental Setup

A. *Experimental Setup for Second Implementation "Application Security Level Index"*

For this setup we used NetBeans and Apktool [16], [17]. NetBeans is an integrated development environment for Java and Apktool is a tool for reverse engineering Android apk files. Hence, the language for our program is Java.

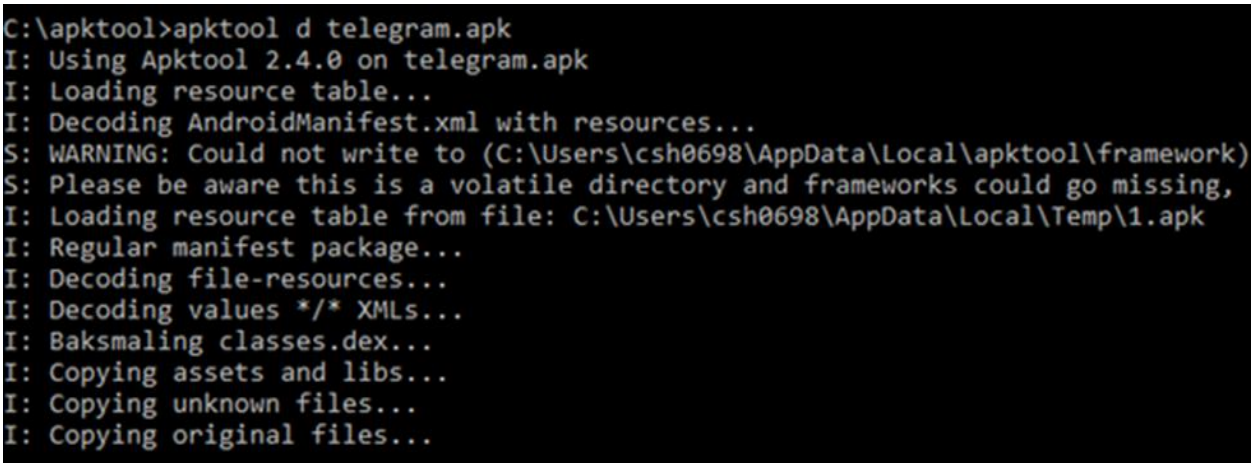
First, we installed Apktool on our computer via the Internet, and performed a setup on our computer to ready it for use. Next, we downloaded some popular applications used in Android devices, including WhatsApp, WeChat, Twitter, Subway Surf, Starbucks, Snapchat, Instagram, Clean Master, Clash of Clans, Google Chrome and Telegram [18] [19] [20]. These are the apps we used to test the program in this experiment.

The next step was to use the command prompt to run Apktool to reverse engineer the downloaded apps. Since the Android application package (.apk) file is actually a zip file, for our experiment, we needed to extract the AndroidManifest.xml from the apk file to analyze the Android permission used.

Fig. 9(i) shows an example of how Apktool reverse engineers the Telegram application. After the reverse engineering is complete, Apktool creates a folder name for the application, which in this case is telegram, and this folder contains res, smali, assets, lib, original, unknown, AndroidManifest.xml and apktool.yml, as shown above in Fig. 9(ii).

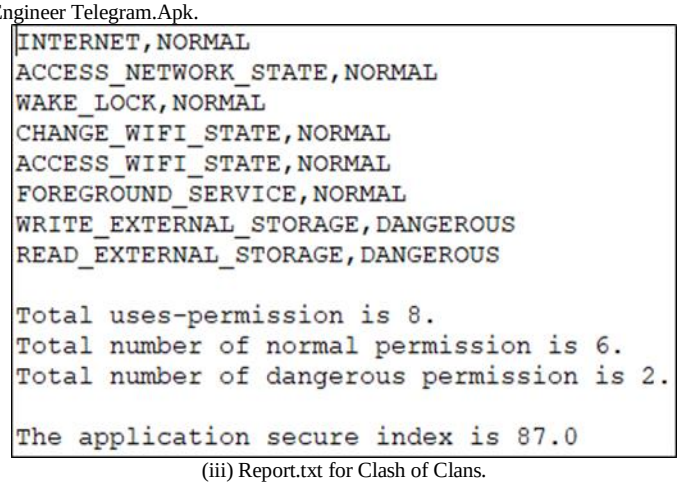
In our experiment we needed only the AndroidManifest.xml file. The AndroidManifest.xml file has plenty of lines of code, but our experiment requires only the Android use permissions.

First, we coded our program to analyze the AndroidManifest.xml file. This program extracted all of the Android use permissions from AndroidManifest.xml and compared them with the Android permission database we constructed of the permissions taken from Android [4]. After the comparison, we categorized the Android use permissions as either normal or dangerous. At the end of the program, two files are produced, i.e., Use-permissions.txt and report.txt. The Use-permissions file contains all the extracted Android Use permissions, and report.txt categorizes these permissions as either dangerous or normal, provides the number of permissions used, and shows the security level index. Fig. 9(iii) shows the generated Clash of Clans's report.txt.



(i) Apktool Reverse Engineer Telegram.Apk.

(ii) Telegram Folder.



(iii) Report.txt for Clash of Clans.

Fig. 9. (i) Apktool Reverse Engineer Telegram.Apk. (ii) Telegram Folder. (iii) Report.txt for Clash of Clans.

B. *Experimental Setup of Advanced Encryption Standard (AES)*

An AES encryption is a symmetric-key encryption, and like the traditional door, it uses the same key to lock and unlock itself, which is also known as encrypt and decrypt in cryptosystems. The following steps show how AES Encryption works.

- Generate a key using "javax.crypto.KeyGenerator;" and "KeyGenerator"
- Encode key into files using "writeKey," "FileOutputStream" and "write(key.getEncoded())"
- Receive the key using "getSecretKey," "SecretKeySpec"and "Files.readAllBytes(file.toPath())" from "javax.crypto.spec.SecretKeySpec;" to be encoded as a key.
- Encrypt using "Cipher," "SecretKey" and "Cipher.ENCRYPT_MODE."
- Decrypt using "Cipher," "SecretKey" and "Cipher.DECRYPT_MODE."

1) Testing: To test that the encryption and decryption were working, we used messages "message" and "getBytes()," as well as "encrypted" and "decrypted," then printed out the original message and the encrypted and decrypted messages.

a) Secret Key generated using "SecretKey" and "generateKey()"

b) Encryption and Decryption using the same key as "encrypted" and "decrypted"

c) IF (original, encrypted and decrypted message printed out without error)

Testing complete and successful

ELSE

Testing incomplete and unsuccessful.

C. *Experimental Setup of RSA (Rivest-Shamir-Adleman) Encryption*

- Generate pairs of keys using "generateKey" and "KeyPairGenerator"
- Encode the pair of keys named "Public Key" and "Private Key" using "DataOutputStream" and "getEncoded()"
- Encryption method using "PublicKey," "Cipher" and "ENCRYPT_MODE."

- Decryption method using “PrivateKey,” “Cipher” and “DECRYPT_MODE”
- Re-create “PublicKey” from serialized key using “getInstance,” “KeyFactory” and “generatePublic” for “publicKeyPath”
- Re-create “PrivateKey” from serialized key using “getInstance,” “KeyFactory” and “generatePrivate” for “privateKeyPath”
- Re-create “PublicKey” from public key byte array using “getInstance,” “KeyFactory” and “generatePublic” as “encryptedPublicKey”
- Re-create “PrivateKey” from private key byte array using “getInstance,” “KeyFactory” and “generatePrivate” as “encryptedPrivateKey”.

1) Testing: To determine if the encryption and decryption works, a message usingare working, we used the messages “data” and “getBytes()” will be used,(),” as well as “encrypted” and “decrypted” will also be used,,” then a message ofprinted out the original message, and the encrypted and decrypted message will be printed outmessages.

a) Public Key and Private Key generated using “generateKey()”

b) “publicKey” encoded using “getPublicKey” as encrypted

c) “privateKey” encoded using “getPrivateKey” as decrypted

d) IF (original, encrypted and decrypted message printed out without error)

Testing complete and successful

ELSE

Testing incomplete and unsuccessful.

VIII. Experimental Results

A. Test Results for Application Security Level Index

TABLE V. Application Security Level Index’s Test Results

Apk file	AUP	NP	DP	SLI
WhatsApp	30	18	12	31.00%
WeChat	30	19	11	35.50%
Twitter	18	9	9	50.50%
Subway Surf	6	5	1	92.50%
Starbucks	9	6	3	82.00%
Snapchat	20	12	8	54.00%
Instagram	18	10	8	55.00%
Clean Master	21	14	7	58.00%
Clash of Clans	8	6	2	87.00%
Google Chrome	23	15	8	52.50%
Telegram	25	14	11	38.00%

AUP = Total Number of Android Use Permissions

NP = Total Number of Normal Permissions

DP = Total Number of Dangerous Permissions

SLI = Security Level Index (where 100% is no permission use)

Table V shows the results of our program, with the apk files that we tested via our program in the left column. The results include the total number of Android use permissions, the total number of normal and dangerous permissions, and the security level index. The fewer permission requests by the application, the safer is that application. However, this does not mean that a larger number of permission requests by an application means that it is dangerous. More permission requests by an application simply mean that the specific application can access most of your phone utilities or data, which can but may not necessarily harm the end user.

Based on the above results, we found communications apps to require the most permission compared to other applications. These are apps such as WhatsApp, WeChat, and Instagram. The security level index of these applications is less than 50%, whereas the apps that request less permission include Subway Surf, the Starbucks app, and Clash of Clans, whose security level indexes are greater than 80%. By the color indicator, green indicates fewer use permissions with an index higher than 80%, yellow indicates a moderate number of use permissions with an index between 50% and 79%, and red indicates a high number of use permissions with an index less than 50%.

IX. Critical Analysis

Based on the above test result from the Application Security Level Index, where shows the outcome result are expected from what we plan in the system flow. The SLI in the Table V is the index where is use to alert the end user about the specific apps that the end users installed in their device. The apps that we tested are the apps which is popular in the market and most of the public are using these apps. Although, if the SLI is indicate red color where a lot of data is used by the app, which doesn’t mean that the specific apps is dangerous, the index is indicate to the users the how much of data privacy are they exposing to the specific app developer, whether the app developer is trusted or not.

Then from the experiment test and test result from the Advanced Encryption Standard (AES) and RSA (Rivest-Shamir-Adleman) above, we use a line of text as a simulation of the real data because that we wanted to test and experiment if the cryptosystem is working. Not only that, in our experiment test, all keys included “Secret Key”, “Public Key”, “Private Key” will be generated automatically as a simulation of different key that will be used by the real application. By doing this, we assure that the code will be running successfully even with different keys. Then for the cryptography, as if the code has run successfully, meaning that the original text has been encrypted into cipher text, and the cipher text has been decrypted into plain text, it means that the code has been running successfully and the encryption and decryption is also running smoothly without errors. In the real form of data, it is

much more complicated, and then it is why we perform the simulation of data using a line of text. With our testing and results, we assure that, by just convert the text into the form of a real data, the real data will also be able to be encrypted and decrypted.

X. Conclusions and Future Work

In this paper, we proposed a spyware detection method for determining the security level of Play Store applications and the data encryption of Google Suite applications. Both spyware detection and apps security use only manifest files to process the results. Since manifest files are required in all Android applications, the proposed method is applicable to all Android applications. The cost of analyzing the manifest file is quite low and combined with Google Play Protect; it provides a more precise detection method. These two implementations will ensure the security of smartphones, even those free of any malware. In response to the global concern about data privacy, we have provided a data encryption method for the Google Suite applications used by most Android users.

In future work, we plan to fix those APK files that cannot be reverse engineered to obtain useful information, such as the Facebook application, to ensure that our method is applicable to all Android applications. We will closely follow trends in hacking methods to ensure that our data encryption method provides sufficient data security.

References

[1] A. Arora and S. K. Peddoju, “NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions,” Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust. 2018, pp. 808–813, 2018.

[2] “Request App Permissions | Android Developers.” [Online]. Available: https://developer.android.com/training/permissions/requesting#java. [Accessed: 07- Nov- 2018].

[3] Xu, Y., Wang, G., Ren, J., & Zhang, Y. (2019). An adaptive and configurable protection framework against android privilege escalation threats. Future Generation Computer Systems,92, 210–224. https://doi.org/10.1016/j.future.2018.09.042

[4] "Permissions overview | Android Developers,” Android Developers. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview. [Accessed: 07- Nov- 2018].

[5] T. Takahashi and T. Ban, "Android Application Analysis Using Machine Learning Techniques,” Springer Link, 2018. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-98842-9_7. [Accessed: 05- Apr- 2019].

[6] "Application Sandbox | Android Open Source Project,” Android Open Source Project. [Online]. Available: https://source.android.com/security/app-sandbox. [Accessed: 11- Nov- 2018].

[7] A. Bryk, "Sandbox-Evading Malware: Techniques, Principles, and Examples", Apriorit, 2018. [Online]. Available: https://www.apriorit.com/dev-blog/545-sandbox-evading-malware. [Accessed: 06- Mar- 2019].

[8] Jeong E. S., Kim I. S. and Lee D. H., "SafeGuard: a behavior based real-time malware detection scheme for mobile multimedia applications in android platform,” 2017. [Online]. Available: https://link.springer.com/article/10.1007%2Fs11042-016-4189-1. [Accessed: 05- Apr- 2019].

[9] F. Shen, "Android Security via Static Program Analysis", Proceedings of the 2017 Workshop on MobiSys 2017 Ph.D. Forum - Ph.D. Forum '17, 2017. Available: https://dl.acm.org/citation.cfm?doid=3086467.3086469. [Accessed 7 March 2019].

[10] L. Tuan, N. Cam and V. Pham, "Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis", Cluster Computing, vol. 22, 2017. Available: https://link.springer.com/article/10.1007%2Fs10586-017-1364-8. [Accessed 10 March 2019].

[11] A. Mandal, A. Cortesi, P. Ferrara, F. Panarotto and F. Spoto, "Vulnerability analysis of Android auto infotainment apps", Proceedings of the 15th ACM International Conference on Computing Frontiers - CF '18, 2018. Available: https://dl.acm.org/citation.cfm?doid=3203217.3203278. [Accessed 10 March 2019].

[12] R. Salvia, P. Ferrara, F. Spoto and A. Cortesi, "SDLI: Static Detection of Leaks Across Intents", 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018. Available: https://ieeexplore.ieee.org/document/8456010. [Accessed 10 March 2019].

[13] L. Menezes and R. Wismuller, "Detecting information leaks in Android applications using a hybrid approach with program slicing, instrumentation and tagging", 2017 International Carnahan Conference on Security Technology (ICCST), 2017. Available: https://ieeexplore.ieee.org/document/8167856. [Accessed 10 March 2019].

[14] "Advanced Encryption Standard,” tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm. [Accessed: 10- Nov- 2018].

[15] "RSA Cryptography Demo Applet | Holowczak.com Tutorials,” Holowczak.com. [Online]. Available: https://holowczak.com/rsa-cryptography-demo-applet/2/. [Accessed: 27- Mar- 2019].

[16] “Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.” [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Accessed: 27- Mar- 2019].

[17] “Welcome to NetBeans.” [Online]. Available: https://netbeans.org/. [Accessed: 27- Mar- 2019].

[18] “AndroidAPKsFree - Free Apps (apk) Download for AndroidTM.” [Online]. Available: https://androidapksfree.com/. [Accessed: 27- Mar- 2019].

[19] “APKMirror - Free APK Downloads - Download Free Android APKs #APKPLZ.” [Online]. Available: https://www.apkmirror.com/. [Accessed: 27- Mar- 2019].

[20] “Download software about - Android ().” [Online]. Available: https://en.uptodown.com/android/top. [Accessed: 27- Mar- 2019.