Security source code analysis of applications in Android OS

Azam, Sami; Sumra, Rajvinder Singh; Shanmugam, Bharanidharan; Yeo, Kheng Cher; Jonkman, Mirjam; Samy, Ganthan Narayana

Published in:
International Journal of Engineering and Technology(UAE)

DOI:
10.14419/ijet.v7i4.15.21366

Published: 01/01/2018

Document Version
Publisher's PDF, also known as Version of record

International Journal of Engineering & Technology, 7 (4.15) (2018) 30-34

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Security Source Code Analysis of Applications in Android OS

Sami Azam[1]*, Rajvinder Singh Sumra[1], Bharanidharan Shanmugam[1], Kheng Cher Yeo[1], Mirjam Jonokman[1], Ganthan Narayana Samy[2]

[1]*School of Engineering and IT, Charles Darwin University, Australia*
[2]*Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia*
**Corresponding author E-mail: sami.azam@cdu.edu.au*

Abstract

It is a known fact that Android mobile phones' security has room for improvement. Many malicious app developers have targeted android mobile phones, mainly because android as an open operating system provides great flexibility to developers and there are many android phones which do not have the latest security updates. With the update of marshmallow in android, applications request permission only during runtime, but not all users have this update. This is important because user permission is required to perform certain actions. The permissions may be irrelevant to the features provided by an application. The purpose of this research is to investigate the use and security risk of seeming irrelevant permissions in applications available from Google store. Two different applications which seem to ask irrelevant permissions during installation were selected from Google store. To test these applications, static analysis, dynamic analysis and reverse engineering tools were used. Findings show potentially malicious behavior, demonstrating that downloading apps from Google play store do not guarantee security.

Keywords: *Android security; Android testing tools; Dynamic analysis; Information leakage detection; Static analysis.*

1. Introduction

Android has the largest mobile phone market share of around 80% of all mobile phones, but fewer than 5% of phones are running on the current version of Android Oreo, which was released recently. With the new update of Oreo in Android, the application now displays all permissions and request permission only during runtime. This is important because user permission is required to perform certain actions. Permissions can be manipulative and irrelevant to the features provided by an application. Thus, if not understood well by app users, can result in exploitation of confidential data [1]. Two different applications were selected from Google store, as they ask irrelevant permissions during installation. However, there are other apps that can perform the same functions, with fewer and only relevant permissions. The primary objective of this research is to test these applications by performing static and dynamic analysis and use reverse engineering tools to analyze the source code about the usage of the collected data and permissions.

2. Security Vulnerabilities

Attackers create a variety of applications of different behaviors as per the data they are targeting. Three main types of application behavior are: malicious behavior, moderate risk behavior and dangerous behavior [2]. Apps that show malicious behavior can gain root access of a device and steal secret credentials. These apps can also access device management and communicate with malicious IP address and domains. Moderate risk behaviors apps are generally deployed by entrusting and unknown sources which mainly focus on reading or sending emails, SMS messages and GPS information. Applications with dangerous behaviors can steal the user's information without notifying him and upload it on an app server or sell it to advertising companies or analytic companies or data aggregation companies.

The following section describes [3] where the collected data could be leaked to malicious actors.

2.1. Leaking Information via IPC

Applications broadcast private information in Inter-Process Communication (IPC) which is accessible to all other apps. Any apps that protect the broadcast with permissions or do not specify the target component can receive intent broadcasts. This is not safe if the intent contains sensitive information. A malicious application that eavesdrops on sensitive information in IPC can gain access to this information.

2.2. Leaking Information via Logs

Apps with the READ_ LOGS permission can read the log messages. Personal information is written for Android's general logging interface and URLs containing this personal information are logged just before a network connection is made. Therefore, apps can access the private information with the READ_LOGS permission [4].

2.3. Unprotected Broadcast Receivers

A broadcast receiver is a component [5] issued by applications to receive intent messages. A malicious application can forge messages if the receiver is not protected by permissions. Android has introduced "protected broadcasts" for system intent types to eliminate forging, but this appears to have limited impact.

2.4. Delegating Control

Apps use a "pending intent" to delegate actions to other applications. An app first generates an intent message as if it was performing some action. It then generates a reference to the intent, based on the goal component type (restricting its usage). The pending intent recipient can fill in the missing fields, but cannot change the values. Thus, if the intent address is not mentioned, the remote app with the original application's permissions can redirect an action that will be performed.

2.5. SD-Card Use

Any application which has access to the SD-card to write or read data can also write or read any other application's data on the SD-card. These apps misuse the SD-card by obtaining the path of the card's root and discovering the free space available on the SD-card.

2.6. Telephone Service Misuse

SMS API used in malicious applications [6] holds certain hard coded phone numbers that misuse SMS Manager class by implementing methods such as send Data Message(), send Multipart text message, send a Text Message() etcetera to pass values for a destination phone number.

2.7. Audio-Video Background Recording

Three key APIs are implemented for visual and audio information recording i.e. Media Recorder class, Audio Record class and Camera class. However, certain malicious applications misemploy these classes in the background to steal data since these codes are inaccessible to the activity component of the Android OS.

2.8. Information Database

Text messages and address book are sharing information which is basically stored in a file based database. Many applications attempt to hack these files and some of them succeed by bypassing android security checks [7].

3. Android Application Data Siphoning Process

Malicious android applications could use numerous methods to siphon important data from mobile devices. In this section, we will look at one example i.e., Android inter-app security vulnerability analysis. This is a vulnerability pattern example explaining how Inter-Process Communication (IPC) can also be used among Android Apps to hinder data security. Android provides a flexible model of IPC that uses a typical application level message, which is known as intent. An Android application comprises of multiple processes, i.e. Activity, Service that communicates using Intents. Additionally, an application process can also send messages (intents) to another application process to perform tasks such as send a stext message, take picture, etc. As we will see in the given example (Figure 1), how a combination of two benign applications that are installed on a mobile device can be used for malicious activities.

4. Potentially Harmful Android Applications

Android is continuously trying to improve the security of the user's personal data and the device. Google has decided to warn the users for potentially harmful applications (PHA) at the time of their installation (see Table 1). To illustrate this, an initial warning will be given to users about the applications that can disable the features of the Android Security such as SE Linux or root the device with disclosure and user consent [8-9]. However, some users ignore these warnings. Individuals tend to ignore warnings issued by the OS at the installation time. The ignorance of users to read the policies at the app installation time has made it easy for attackers to install malicious applications on the users' devices. Android malware that abuses possible vulnerabilities are a great challenge for the developers.

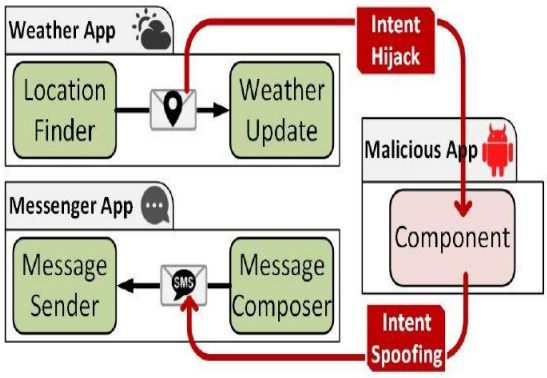

Fig. 1: A malicious application exploiting the vulnerability [3].

Table 1: Potentially harmful applications definitions

PHA	Definition
Hostile Downloader	An application that is not in itself unsafe, but downloads other potentially harmful apps. For instance, a gaming application that does not contain noxious code, but rather relentlessly shows a deceptive "Security Update" interface that installs unsafe applications.
Mobile Billing Fraud	Application that charges the user in a deliberate deceptive way. Mobile billing fraud is divided into SMS fraud, Toll fraud and Call fraud, based on the misrepresentation being committed.
SMS Fraud	Application that charges clients to send premium SMS without user knowledge, or tries to camouflage its SMS activities by concealing divulgence agreements or SMS messages from the device operator informing the user of charges or affirming membership.
Spam	An application that sends irrelevant commercial messages to the mobile user's contact list
Phishing	An application that pretends to be from trustworthy resource, request for mobile user's credentials or billing information, and sends this information to a third party. Most common targets of phishing attacks incorporate banking credentials, debit / credit card no. or online banking credentials

Some of the Android malware that made their way to the Google Play store and proved to be highly dangerous are spyware, adware, backdoors, commwarriors and botnets [10].

There is not much information about how the Google app review process works or what tools are used to test the applications. But, it has been mentioned in the Google developer website that apps are not filtered based on the permissions the app uses. Generally, the review process takes 2-3 hours. Recently, Google play started using a mixed approach which means that some apps can be manually tested as-well. Apps need to comply with Google developer policies which does not allow network and user data abuse, malicious behaviour, improper ways to display ads, insecure user data or offering rewards in return of leaving good reviews.

Google play store discourages the installation of apps from any third-party marketplace due to security concerns; however, it still permits these. Third party developer apps are made available from the official play store. Google uses Bouncer (a dynamic analysis sandboxed environment), which is a reasonably effective security mechanism to restrict any malware from entering the Google Play store. Android also has provided the facility of running a

verification service, while installing applications from other market places.
For this research two apps, Remit online money transfer app and Voice recorder app, were selected for detailed analysis. The selection was based on the fact that users often require financial apps and voice calling apps.

5. Tools

The purpose of this research is to investigate the use and security risk of seemingly irrelevant permissions in applications available from Google store. This can be done using static and dynamic analysis tools.

5.1. Static Analysis Tools

Static Analysis is performed without executing the code and hence does not influence the device's functioning. It is done by de-compilation and disassembly. The static analysis process involves several steps such as extracting n-gram statistics of .dex file, disassembling of an application, pattern search for malicious API calls and URLs and extraction of relevant information from an Android Manifest file such as permission, intents, activities, actions, services, and receivers [11].
In static analysis, reverse engineering tools are used to analyze the code. The reverse engineering approach helps to decompile and recompile an AK file. This is crucial to examine the code. Every apk file contains three files. First, AndroidManifest.xml, which defines the permissions, used in an application, secondly Classes.dex which contains all the Java files (Java source code needs to be retrieved with the help of another tool) and thirdly Resources.arsc which contains Meta information about the resources and nodes.

5.2. Dynamic Analysis Tools

Dynamic analysis observes the execution of instructions in a real time environment, thus looks at the information flow and interaction with other applications. This analysis has performance overheads as it performs real time monitoring of apps. Android applications facilitate several User Interface (UI) gestures, i.e. swipe, tap, pinch, back key, menu key keyboard. It is of utmost importance to address all these gestures during app development, but unfortunately some of them may unknowingly be missed. This may generate multiple entry points for malicious applications. Therefore, this analysis triggers these events to prevent serious issues. This analysis technique can be implemented on both virtual and actual android devices for real time processing [12-13].

6. Android Permissions and Application Analysis

To gain controlled access of certain system data, critical resources and features, Android apps request permissions from the operating system before its installation. Permission security models and Access Control Lists are the main mechanisms to provide access to these resources and maintain security for the users and system. However, various malicious android applications misuse these permissions (refer to Table 2).
For this research two apps were selected for detailed analysis, Remit online money transfer app and Voice recorder app. In the first step, static analysis tools called Androbugs [14] and Androwarn [15] were used. Based on the generated reports for each application, further analysis about attack surface, vulnerabilities and possible malicious code was performed. In the second step, with the help of reverse engineering tools, the application was decompiled and a comprehensive analysis was performed.

Table 2: Permission list [3].

Android Permission	Use	Misuse
INTERNET	Allows the applications to access the internet via opening network sockets for transferring data.	Granting this permission can be misused by sending confidential user data to unknown URL.
ACCESS_ NETWORK_ STATE	Allows the application to access information about the type of network available, type of network devices Connected, roaming or local network and no. of failed connection attempts.	Misused by a malicious application for maintaining user profile regarding his network information.
ACCESS_ WIFI_ STATE	Allows the application to access information about Wi-Fi network	Can help the malicious application in hacking the Wi-Fi network and sending user data by using this information
READ_ PHONE_ STATE	Gives access to critical data of phone like IMEI/IMSI device identifier, Phone Number, Network Operator, Voice Mail Box, SIM ID etc.	Helps the malware author to keep track of your phone and can involve your device in malicious activities using this Information.
READ/ WRITE_ EXTERNAL_ STORAGE	Allow to read or write external storage	Malware can read confidential data of the user and write its malicious code on External storage.
SEND_SMS, RECEIVE_ SMS, READ_SMS, WRITE_SMS	Allow the activities related to SMS	Help the malicious application read, write and send user's personal Information to the malware author.
ACCESS_ COARSE_ LOCATION, ACCESS_ FINE_ LOCATION	Permissions to access location related information of mobile device	These two permissions are used by the malicious applications for location based sniffing

6.1. Analysis of Remit Money Transfer App

This app is meant to transfer money from the US to Pakistan. Analysis of the app is discussed with the help of images of results from different tools. First the relevance of permissions required during installation was evaluated. As seen in Figure 2, the app showed many irrelevant permissions which are not justified by its functionality. An alternative app (right side image), World remit, requires no permissions; it is used to transfer money from any country to any place in the world. This app was also tested and did not show any suspect behaviour.

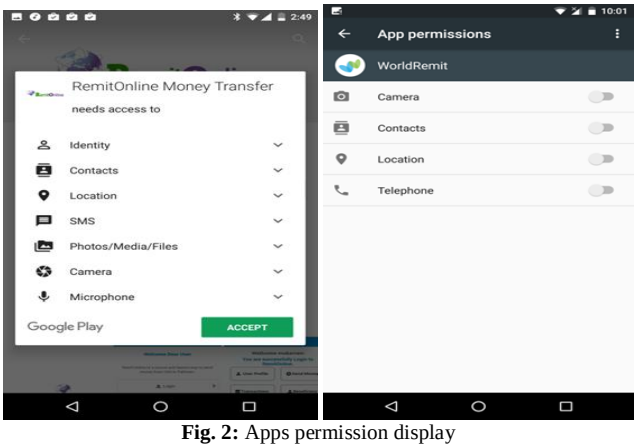

Fig. 2: Apps permission display

Androwarn report as shown in image Figure 3 gives useful information like android manifest.xml, API's used, application information and analysis results about possible malicious behaviour in an app.

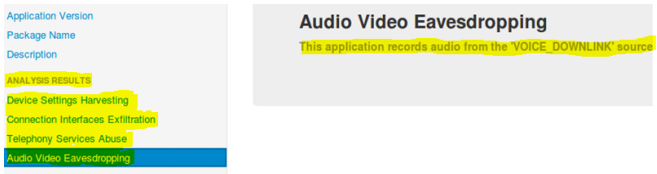

Fig. 3: Androwarn report for Remit money transfer app

Figure 4 is a screenshot of Androbugs report, which shows ADB backup/debugging are enabled. This means that anyone who has physical access to this phone can make a backup of all app data. For a money transferring app, it is a quite risky to have this feature enabled by the developer.

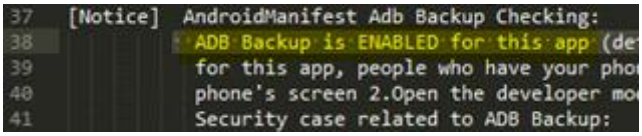

Fig. 4: Androbugs report for Remit money transfer app

Reverse engineering of application: Audio video eavesdropping- There are 5 classes for audio and video in this application. In one of the class Audiohandler.smali on line 146, it shows a method to record audio as shown in Figure 5.

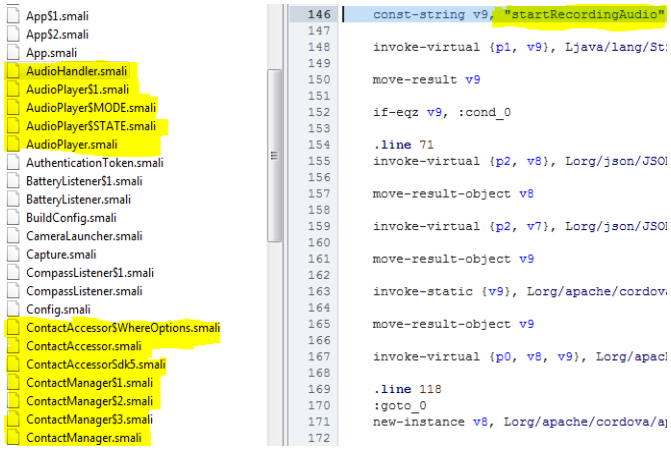

Fig. 5: Source code analysis of app

Figure 6 shows that the application has a code to check the phone status, whether the phone is ringing or in sleep mode.

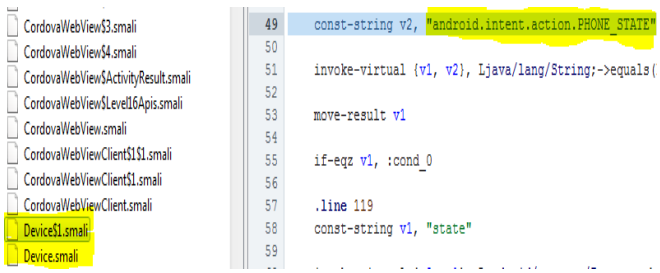

Fig. 6: Source code analysis of app

Based on Figure 7 showing the class NetworkManager.smali on line 612, it can be observed that the method connectivity change. This could be used in changing connection to send data to a different network. The connection interface ex-filtration means the use of malicious code to transfer data from the mobile phone. It also has a file storage.smali, which shows backing up of data, on line 38.

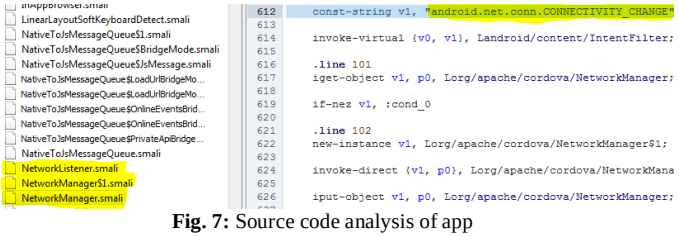

Fig. 7: Source code analysis of app

Figure 8 shows unencrypted transfer of data in Wireshark in which encryption should be used to transfer confidential information such as user name, account details, password and other information. Without encryption, it is easier for someone to steal information by performing attacks like the man in the middle.

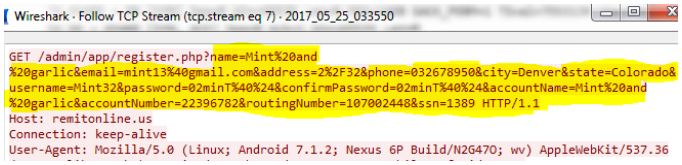

Fig. 8: Wireshark shows unencrypted transfer of data

To recapitulate, this app's analysis demonstrated surprising results such as user data misuse (audio video eavesdropping), data backup, network and phone abuse and insecure data transfer.

6.2. Analysis of Voice Recording App

Permissions displayed by the Voice recorder app as shown in Figure 9 during the installation seem quite irrelevant for the main features it means to provide. The second right-side image shows, an alternative app doing the same job with far less permission.

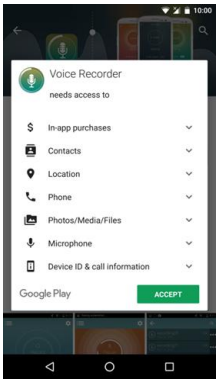

Fig. 9: Permissions displayed by Voice recorder app

The result in Figure 10 from virus total shows a type of Trojan malware android/generic.z.2ec701!tris.

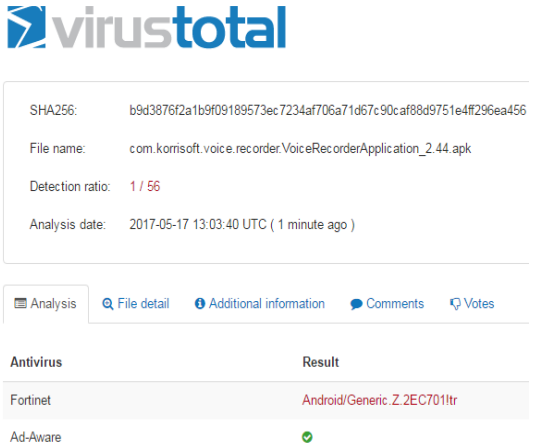

Fig. 10: Virus total detecting a Trojan in app

Figure 11 shows there are many classes that deal with phone numbers activities (Note a very recent update of this app provided the functionality of showing caller id like true caller app, while these functions were not available during the testing of this app. Still, this does not justify its behavior).

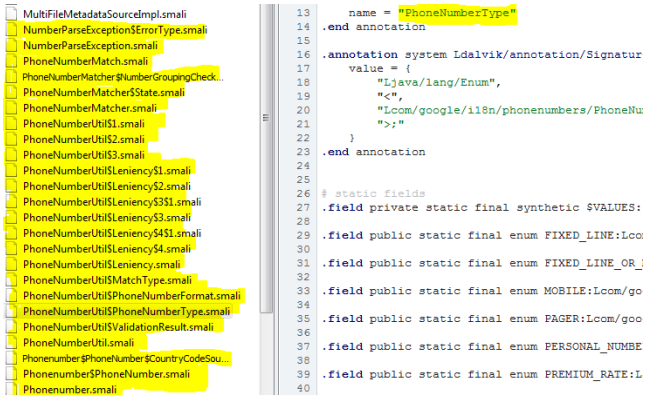

Fig. 11: Source code analysis of app

Figure 12 shows that there are many classes to get the user's exact location.

Fig.12: Source code analysis of app

As a summary, this app's main purpose is to record voice but it shows a lot of suspect behavior such as accessing external storage, contacts, network and user location.
Table 3 shows the list of Google developer policies violated by the selected applications. The following table is a summary of the list of vulnerabilities that has been discussed earlier.

Table 3: List of violated Google policies

	Violation of Google Developer Policies			
	User Data	Device and Network Abuse	Malicious Behavior	Wrong Way of Display-ing Ads
Remit Money Transfer	√	√	√	
Voice Re-corder	√	√		√

6. Conclusion

There is limited research on the security of data that is backed up to a server. This is one of the major concerns for security as the phone is a primarily an internet device and most of the information is backed up and synchronized with a remote server. Findings from analysis of applications demonstrate the insecure user data, malicious behavior, and device and network abuse by an application. Insecure transfer of confidential data without encryption speaks volume about an app security. Down-loading apps from Google play store do not guarantee security. These apps may be capable of stealing sensitive data from mobile phones without the user's knowledge. Android needs to strengthen its Google app review process by involving more trusted static and dynamic tools and accessing the results of red flag apps manually. The more secure phones are based on optimized android OS like Solarin, Blackberry dtek60, Blackphone2.

References

[1] Waddell, K. (2017). When apps secretly team up to steal your data. https://www.theatlantic.com/technology/archive/2017/04/when-apps-collude-to-steal-your-data/522177/.
[2] Sikorski, M., & Honig, A. (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. No Starch Press.
[3] Sadeghi, A., Bagheri, H., & Malek, S. (2015). Analysis of android inter-app security vulnerabilities using COVERT. Proceedings of the IEEE 37th International Conference on Software Engineering, pp. 725-728.
[4] Enck, W, Gilbert, P, Chun, B-G, Cox, L, Jung, J, McDaniel, P & Sheth, A. (2014). TaintDroid: An information flow tracking system for real-time privacy monitoring on smartphones. Communications of the ACM, 57(3), 99-106.
[5] Bhoraskar, R., Han, S., Jeon, J., Azim, T., Chen, S., Jung, J., Nath, S., Wang, R., & Wetherall, D. (2014). Brahmastra: Driving apps to test the security of third-party components. Proceedings of the USENIX Security Symposium, pp. 1021-1036.
[6] Enck, W., Octeau, D., McDaniel, P. D., & Chaudhuri, S. (2011). A study of android application security. Proceedings of the USENIX Security Symposium, pp. 1-38.
[7] Elenkov, N. (2014). Android security internals: An in-depth guide to Android's security architecture. No Starch Press.
[8] Dunham, K., Hartman, S., Quintans, M., Morales, J. A., & Strazzere, T. (2014). Android malware and analysis. Auerbach Publications.
[9] source.android.com. (2017). Security report. https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final .pdf.
[10] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2015). Android security: A survey of issues, malware penetration, and defenses. IEEE Communications Surveys and Tutorials, 17(2), 998-1022.
[11] Oberoi, S. (2014). AndroSAT: Security analysis tool for Android applications. PhD thesis, Concordia University.
[12] Wang, P., Lin, W. H., Chao, W. J., Chao, K. M., & Lo, C. C. (2015). Using dynamic taint approach for malware threat. Proceedings of the IEEE 12th International Conference on e-Business Engineering, pp. 408-416.
[13] Sikorski, M., & Honig, A. (2012). Practical malware analysis: The hands-on guide to dissecting malicious software. No Starch Press.
[14] Androbugs (n.d) Androbugs framework. https://github.com/AndroBugs/AndroBugs_Framework.
[15] Androwarn (n.d). Androwarn. https://github.com/maaaaz/androwarn.