ISSN 1870-4069

Armagedroid, APKs Static Analyzer Software

Luis Enrique Héctor Almaraz García, Eleazar Aguirre Anaya,
Ponciano Jorge Escamilla Ambrosio, Raúl Acosta Bermejo

Instituto Politécnico Nacional, Centro de Investigación en Computación, CDMX, Mexico

lalmarazg1400@alumno.ipn.mx, eaguirre@cic.ipn.mx,
pescamilla@cic.ipn.mx, racostab@ipn.mx

Abstract. Armagedroid, a software for static analysis of Android APKs, arises with the objective of assisting in the decision making by the user analyst, who must evaluate, thanks to the metadata obtained by the program, if it is a reliable package or a possible malware application, automating the procedures involved in this type of analysis. Consistent phases of the Armagedroid analysis consider the APK structure, its contents, its manifest file to extract the package, permissions and archive activities using action modules. The result obtained with the use of the tool is the gathered information from each module applied to a benign APK and one with malware, which, once compared, distinguish that the malicious package requests more permissions than the trusted APK and with just having an activity. The contributions of Armagedroid in comparison with other programs of static analysis are: the validation that the file loaded in memory is really an APK, checking its size, obtaining its content and generating the analysis report of the APK which consists of the information of the metadata obtained from the APK: the name, size in bytes, integrity checksums, which are MD5, SHA1 and SHA256, APK content , information of the files it contains, the name of the package, the list of activities and permissions of the APK in order to make the results known to the user.

Keywords: APKs, Android, mobiles, applications, static analysis, malware, software.

1 Introduction

The rise of the Internet and recently the Internet of Things, it is estimated that there are about 19 billion digital devices connected to it [1], has led to the growth of malware development by cybercriminals, to pursue certain purposes, mainly: data theft, cybercrime, espionage and hacktivism. Each quarter of 2016 found 600 million new samples of malware [2]. Currently, there is malware for all operating systems of personal computers like Windows, Linux and MacOS, to mention a few examples, but also malware has been developed for operating systems of mobile devices, the most attacked are Android and iOS [3].

Malware analysts on different platforms employ 2 types of malware analysis, static analysis and dynamic analysis. Static analysis identifies cryptographic hashes, fingerprints and metadata for a malware, while in the dynamic analysis malware execution is performed to study its behavior [4].

Android has become the operating system mostly used by smartphones, tablets, watches, smart TVs and even some vehicles, so it has caught the attention of malicious attackers to develop malware and take advantage of its users. A new Android application with malware is discovered every 10 seconds [5], it is estimated that by the end of this year 2017, the number of malware for this operating system will reach 3,500,000 samples [5]. These applications are formed by a variant of the Java JAR format, which is called APK [6]. An APK file contains the necessary resources for the installation of APPS, specifically: the manifest file (AndroidManifest.xml), the executable code (classes.dex) the compiled resources of the application (resources.arsc) and the directories of the application.

This paper describes a new approach called Armagedroid that performs the functions of a static analysis, obtaining: package name, size in bytes, integrity check sums, the hash functions MD5, SHA1 and SHA256, APK content, the files it contains, such as comments on them, last modified date, operating system in which they were created, ZIP version used to compress them, their size once they have been compressed and decompressed, the list of activities and permissions of the APK. The development of Armagedroid allows a user to automate the tasks of performing the static analysis of Android APKs that comprises the process of decompilation of the APK, the extraction of its metadata, its directories and files.

The remaining of this document is organized as follows: Section 2 defines the main concepts related to APKs: Android manifest file, package name, permissions, and packaging activities. The existing related works about static analysis of APKs are described in section 3, the phases of the software they developed and the results obtained. The description of the modules of Armagedroid static analysis are presented in section 4, which range from loading the APK, validating it, calculating its integrity checksums, obtaining its contents, decompiling its resources, reading the file of the packaging and the generation of the static analysis report. In section 5 the tests and results obtained by applying the developed software on a benign application and on a malicious APK are presented. Section 6 presents the conclusions reached in the implementation and use of the software, as well as the future scope regarding the improvement of the tool.

2 Android ApPlication PacKages

Applications are distributed and installed in the form of packaged application files, called APKs. They are containers that include the code and the resources of the application, as well as the manifest file (containing the permissions of the app).

APK is an extension of the JAVA JAR format, which in turn is an extension of the ZIP format. The contents of an AKP are presented in Table 1 [7].

Table 1. Contents of an APK and its description.

Content of an APK	Description
AndroidManifest.xml	Declares the name of the application package, its version, activities, services, message receivers and content providers that integrate it [8].
Classes.dex	Contains the executable code of the application in Dalvik VM format which is the file with extension .dex.
Resources.arsc	It has compiled application resources such as strings and styles.
Assets directory	It stores unprocessed resources, i.e. those that at the time of generating the APK retain their name and characteristics,
Lib directory	This directory is present in applications that use native libraries via JNI (Java Native Interface).
Res directory	Group resources that are directly referenced from the Android code, either directly using the android.content.res.Resources class or indirectly through the high-level APIs, they are split into separate subdirectories for each type of resource (animations, images, menu definitions, etc.).
META-INF dir.	It hosts the package manifest file and signature code.

2.1 Package that Contains the Android Application

It is the file that works as a unique identifier for the application and contains the classes that implement a specific function once the APK is installed [9]. The example of announcing a package is shown in Fig. 1, see that it begins with the <*manifest package* = tag and the name of the package in quotation marks with its tag close >.

```
1. <manifest package=”com.example.project”>
2. ...
3. </manifest>
```

Fig. 1. Declaration of a package in an APK.

2.2 Permissions on Android for APKs

They define the access rights of the apps to the resources of the device to perform an action, they can be used to access the hardware, to obtain connectivity to the Internet, to use data of the user or the services of security, memory and processes. Once installed the apps request these permissions through the file [10]**.** The declaration of a permission is shown in Fig. 2**,** it should be noted that, in its construction, they start with the label <*uses-permission android: name* = followed by the name of the permission to which the app must access, finally the label is closed with >.

```
1. <manifest
xmlns:android=”http://schemas.android.com/apk/res
/android”
2.     package=”com.android.app.myapp” >
3.     <uses-permission
android:name=”android.permission.RECEIVE_SMS” />
4      ...
5. </manifest>
```

Fig. 2. Declaration of a permit in an APK.

2.3 Activities of an APK

An activity is a component of the application that contains the screens that are presented to the user to interact and perform an action on it, thanks to the methods it implements. The APK developer encodes an activity for each window that is shown to the user.

All activities are declared in the AndroidManifest.xml file to be accessed by the application [11]. Fig. 3 shows the declaration of an activity is done by adding an element <*activity Android: name* plus the name of the activity in quotation marks and the closing of the element with >.

```
1. <manifest ...>
2.  <application ...>
3.    <activity android:name=”.ExampleActivity”/>
4.    ...
5.  </application>
6. ...
7. </manifest>
```

Fig. 3. Declaration of an activity in an APK.

3 Works Related to Static Analysis APKs

There have been several publications related to using computational tools to perform static analysis and dynamic analysis in the APKs, however, the focus of this document and the Armagedroid software takes only static analysis as a reference.

A program that performs static analysis extracting the most relevant features of an Android application [12] is performed by means of 4 stages: The first one involves obtaining the APK and decompressing it through APKTool, stage 2 is responsible for extracting the characteristics of the application through the manifest file (AndroidManifest.xml), taking into consideration the actions That request other actions of the different components of the application and also in the categories which contain additional information on the type of component that action must handle, in addition the software considers the permissions of the application, it is assigned a

category to the APK Analyzed in stage 3, according to a classification that the authors previously made with the largest number of permissions present in several applications and in stage 4 the user gives an assessment on the characteristics obtained on the APK and the category that will be assigned. The results that they obtained after analyzing several APKs were a list of categories: communications, games, social networks, utilities, education, multimedia, widgets and trips.

At the University of Technology, they developed a framework that detects malware in Android applications. They use the concept of machine-learning monitoring the permissions of the application and events, in this way, the machine-learning classifier is fed through 4 phases, the first is responsible for extracting the permissions requested by the APK, the second uses the clustering algorithm K-Means to provide information to the classifier, with the permits obtained before, the third phase applies decision tree algorithms to determine if the APK is malware or is benign and in the fourth phase evaluates the accuracy and accuracy of classification by means of true positive, false positive, true negative, false negative, and general accuracy formulas. They obtained 2 sets of data that differentiate between a malware APK and a benign one, the database was composed of a total of 500 malware APK features, which were extracted [13].

At the University of Luxembourg, authors used a set of 50,000 Android applications to feed a machine-learning that classifies a group of features for malware detection [14]. Firstly, the classifier is fed to the classifier with 2 types of sets, the set of APKs obtained from Google Play with a module of the program that is responsible for downloading them and the set of APKs with malware. They take the APK to analyze from which their permissions are obtained, a prediction is obtained thanks to the algorithms of RandomForest, J48, JRip and LibSVM, which, with the support of the characteristics of the machine-learning information, informs if the APK is malware or benign, together with the support of VirusTotal. As results obtained a classification accuracy radius with mean values of 0.94, this implies a better probability to differentiate between benign and malignant APKs.

4 Armagedroid System Modules

This section details the construction and operation of Armagedroid modules. The overview of its components can be seen in Fig, 4 which describes that the software currently has 7 modules of static analysis, which range from loading the APK into memory, validating it, calculating checksums integrity, obtaining information from its contents, Table 1, decompiling, reading the file AndroidManifest.xml and generating the static analysis report.

a) Module of loading APK in memory: It is the first phase of the analysis, the user selects the APK between their files; this is possible with the module of loading APK in memory.

b) Module of validating APK: A feature that Armagedroid has before executing the following phases of static analysis is to ensure that the selected file, in the previous

phase, is really an APK, for this, it observes the structure of the same and within its content looks for the magic number of the packaging. The magic number of a file is a numeric value that identifies it and associates it with a certain format. In the case of APKs, the magic number is the value represented in hexadecimal as *"PK \ x3 \ x4"*, see the structure of an APK in Fig. 5. If the file does not contain this value, then Armagedroid stops the analysis informing the user that the file is not APK, otherwise, proceed to calculate the size of the it and the condition at this time to continue with the analysis is to calculate its size in bytes, if it turns out to be 0 bytes, the analysis stops and the user is told that the APK size does not allow the analysis.

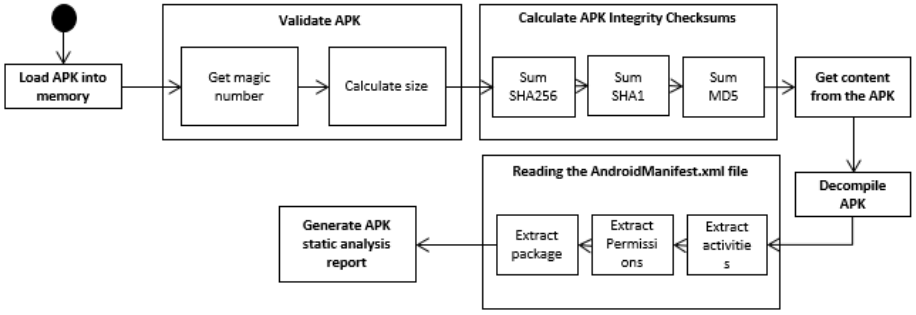

Fig. 4. Modules of Armagedroid analysis.

c) Module of APK Integrity Checksum Calculation: As shown in Fig. 4, the system determines 3 integrity check sums of the APK: MD5, SHA1, and SHA256 using the Integrity Checksum calculation module, it uses methods of the Python language, in its version 3, with which the system was implemented, included in the hashlib library, these are: hashlib.md5(), hashlib.sha1() and hashlib.256() respectively, each one receives as input the byte set of the APK, Fig. 5, outputting a 128-bit alphanumeric string for MD5, 160 bits for SHA1 and 256 bits for SHA256.

d) Module of obtaining the content of the APK: This module obtains the contents of the same one, the packaging contains the files of the Table 1 and of each one of them are extracted: comments on the same, its last date of modification, operating system in which they were created, version of the used ZIP to compress them, their size once they have been compressed and decompressed. Armagedroid takes this data informing the user analyst what the content of the APK, thus favoring the execution of the module of obtaining its content without the need to add a decompression function of it.

e) Module of APK decompilation: It uses the APKTool component for the APK decompilation task, reading the decoded AndroidManifest.xml and obtaining the name of the package, the permissions and activities corresponding to it, in the later stages of the analysis. It results in a directory with the name of the APK and the decompiled files in Table 1, if APKTool is not used, the files would not be readable.

f) Module of AndroidManifest.xml file reading: Reading module is responsible for reading the Android manifest file, previously decoded, extracting the package name from the APK, also gets the permissions and activities that compose it. The software uses regular expressions to find the format in which the 3 elements are found before the determination and thus their values, in the figures 1, 2 and 3 the declarations of a package, a permission and an activity are described respectively.

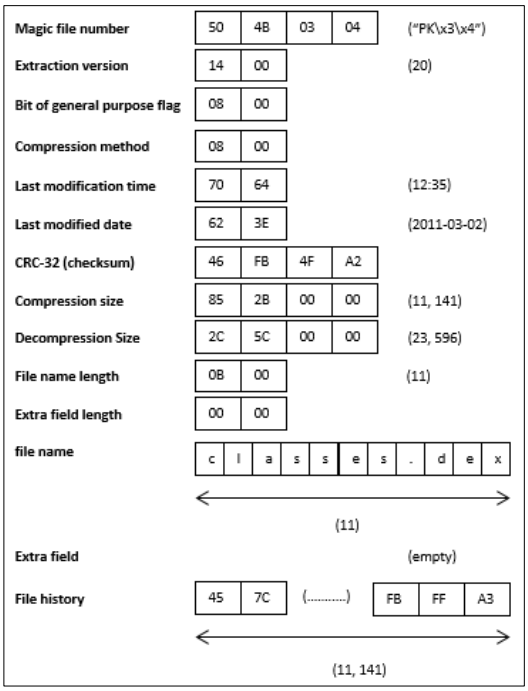

Fig. 5. Structure of an APK.

g) Module of APK static analysis report generation: In its last execution, Armagedroid performs a report in .txt file format containing the metadata obtained from the APK: the name, size in bytes, checksum integrity checksums, which are MD5, SHA1 and SHA256, APK content , information of the files it contains, such as comments on them, last modified date, operating system in which they were created, ZIP version used to compress them, their size once they have been compressed and decompressed, also included name the package, the list of activities and permissions of the APK, this task is performed by the module of generation of the report of static analysis of the APK.

5 Tests and Results

Armagedroid software was tested with 2 APKs, one of which corresponds to a video game of ships that destroy asteroids called *Asteroides.apk* and the second one is malware, with the name *badapk.apk*, the purpose of analyzing these 2 APKs is to

know the scope that would have each one in the device once installed. Table 2 shows the hardware and software features used for running Armagedroid.

Table 2. Hardware and software features used for running Armagedroid.

Software	
Operating System	Kali Linux 2017.1
Architecture	32 bits
Programming language	Python 3
Hardware	
RAM memory	3.8 Gb
Processor	Intel® Pentium(R) CPU N3520 @ 2.16GHz × 4
HDD	26.1 GB

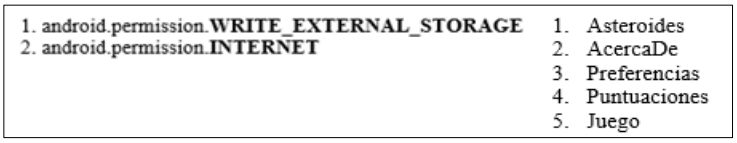

Fig. 6. Permissions and activities from the APK Asteroides.apk.

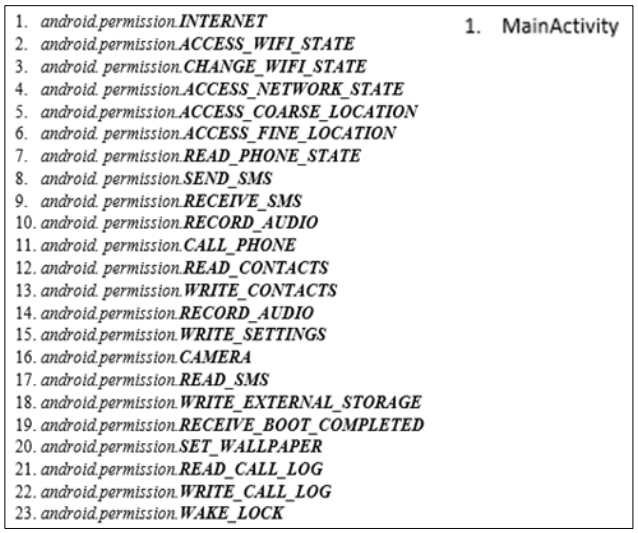

Fig. 7. Permissions and activities from the APK badapk.apk.

As seen in Fig. 6, the permissions of the APK *Asteroides.apk* request access to information storage and Internet access, are the permissions of a video game that writes the scores obtained in the memory of the device and that has the capacity to

access the Internet. Also, in Fig. 6**,** provides an overview of the activities that the user will see on the screen.

In the case of APK with malware, Armagedroid extracted the permissions of Fig. 7, in total there are 23, ranging from the request for Internet access, Wi-Fi network state settings, user location, reading of your contacts, control over the sending and receipt of both messages and phone calls, access to the camera and the microphone, to change the background image of the device and write to memory. The software detected a single activity, represented in Fig. 7, suggesting to the user that the APK is analyzed, that it is a main activity that runs in the background avoiding raising suspicions about their work.

6 Conclusions and future scope

Thanks to the implementation of Armagedroid, it was possible to create a tool that automates the task of decompiling the APK, to extract its metadata, its directories and files, procedures proper to a static analysis, the results of comparing a benign APK with a malware resulted in a large difference in the permissions requested by the malicious application as it requested a set of permissions on the configuration of the wireless network, as well as the Internet, access to calls, messages and contacts on the phone, also to write information in memory, manipulate the device's camera and microphone, all give full scope to the tasks of the Android operating system and give access to a cybercriminal to manipulate it.

As for benign APK, only two permissions were written in memory and Internet access, which gives an idea that the application is what it claims to be, a video game. An important point to say is that the tool does not yet predict probabilistically if the APK is malware or benign, but only assists in the decision making, through the obtained metadata, to the analyst user.

To get the APK valuation task, Armagedroid will be attached to the Garmdroid web application, developed in the Cyber Security laboratory of the Computer Research Center, which informs the user if the APK it analyzes is malware or benign. Due to this interaction with the web platform, a communication component will be created between both softwares, both Armagedroid and Garmdroid. In future tests Armagedroid will be applied to a malware database of APKs to know the characteristics that it obtains from them.

Aknowledgements. We thank the Instituto Politécnico Nacional and CONACyT who have made possible the development of the Armagedroid project, both in support of the necessary resources and in the provision of facilities for the realization of it.

References

1. Neely L.: Exploits at the Endpoint: SANS 2016 Threat Landscape Survey. (2016)

2. Universidad Internacional de Valencia: Ciberseguridad: Tendencias 2017. Jun 6, 2011, from Universidad Internacional from Valencia (2011)
3. Case A., Golden R.: Advancing Mac OS X Rootkit Detection. In: Digital Forensic Research Conference (2015)
4. Dunham K., Hartman S., Morales J., Quintans M., Strazzere T.: Android Malware and Analysis. E.U.: Auerbach Publications, pp. 7, 51, 52, 91, 92 (2014)
5. Lueg, C.: 8,400 new Android malware samples every day. April 27, 2017, from G DATA. https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day, last accessed 2017/08/14 (2017)
6. Elenkov N.: Android Security Internals: An In-Depth Guide to Android's Security Architecture. San Francisco, No Starch Press, pp. 51–52 (2014)
7. Elenkov N.: Android Security Internals: An In-Depth Guide to Android's Security Architecture. San Francisco, No Starch Press, pp. 51–52 (2014)
8. Developers Android: App Manifest, Developers Android. https://developer.android.com/guide/topics/manifest/manifest-intro.html, last accessed 2017/08/14
9. Developers Android: Package, Developers Android, https://developer.android.com/reference/java/lang/Package.html, last accessed 2017/08/14
10. Elenkov N.: Android Security Internals: An In-Depth Guide to Android's Security Architecture. San Francisco, No Starch Press, pp. 21–26 (2014)
11. Developers Android: Activities. December 4, 2016, https://developer.android.com/guide/components/activities.html, last accessed 2017/08/14
12. Zuhair, M., Nisar, A., Ullah, H.: Automatic Feature Extraction, Categorization and Detection of Malicious Code in Android Applications. Institute of Advanced Engineering and Science (2013)
13. Aung, Z., Zaw, W.: Permission-Based Android Malware Detection. March 3, 2013, from International Journal of Scientific & Technology Research (2013)
14. Allix K, Bissyandé T, Jérome Q, Klein J, State R, Le Traon Y. Large-Scale Machine Learning-based Malware Detection: Confronting the 10-Fold Cross Validation Scheme with Reality.University of Luxembourg (2014)