Android-App-Dissection on the Wandboard – Automatic Analysis of Malicious Mobile Code

H. Winkler¹ and C. Hummert¹
¹Hochschule Mittweida – University of Applied Sciences, Forensic Science Investigation Lab, Mittweida, Germany

Abstract—Android-based devices are the main target for cyber criminals. Especially, there is a broad range of malware for the Android platform. The analysis of the malicious code is an important challenge for the law enforcing agencies and developers of antivirus suites. To make the matters worse anti-forensic technologies are widespread in Android malware. To overcome this issue a new platform for the automatic analysis of mobile malicious code is presented. In contrast to existing frameworks the platform is based on the Wandboard hardware platform and supports various Android versions, the functionality is configurable. It is impossible for anti-forensic functions to detect the analysis environment because of the underlying hardware. The platform does both static and behavioural analysis of the malware. The results are presented in a report. The complete platform is easy to use and distributed as a web service. The result of an exemplary malware dissection is presented.

Keywords: malware analysis, mobile malware, Wandboard Cortex, Android, anti-forensic

1. Introduction

The importance of mobile devices is increasing since their introduction, due to the growing field of application and their ever improving functions. Smartphones have become a central element of our daily life no matter if private or business related. The core element of all these devices is the operating system (OS), whereby Android is the most popular mobile platform [1]. This particular operating system is dominating the international market for years now – in the third quarter of 2016 it had a share of over 87 percent [2]. The high distribution of this free platform is caused by a broad spectrum of compatible hardware but also by the open source structure of the operating system. The majority of all applications at all are available for Android [3] and especially apps are easy to create and distribute for this platform. For this reason Android-based devices are the main target for cyber criminals [4]. The systems are being compromised through exploitation of security gaps. Especially privilege escalation is easy in Android compared to other mobile OS [5], [6]. This problem can be treated with patches and optimized security measures in new Android versions. Because of the high degree of fragmentation among Android versions and the producers update policy zero-day-malware is still usable for attackers long time after detection [7]. Furthermore mobile malware is getting more and more complex in structure and process operation. This includes improved features and optimized methods to purposefully compromise devices and mobile systems.

Hence, there is a need for an efficient methodology overarching all Android versions to systematically analyze such threats. Relevant information can be gathered via static analysis of the source code or through dynamic analysis of the application during runtime. A static analysis of Android Application Packages (APK) is possible with some online services such as VirusTotal [8]. But the extracted data only partly gives information about the characteristic runtime behavior. Furthermore, the detection and comprehension of functions is difficult due to extensive protective and concealment mechanisms of the malicious application [9]. A combination of both techniques is therefore much more efficient [10]. Unfortunately dynamic analysis systems based on Android software emulators is not effective. Malware can avoid the examination with the implementation of intelligent anti-forensic technologies [11]. To get quality results the Android runtime environment needs to be simulated on real mobile hardware so that malware will show its characteristic behavior during the inspection [12]. Because of that, the necessity of this work arises to build a semiautomatic chain of analysis for mobile malware in a hardware environment. For that the following goals are defined:

- hardware-based Android Analysis System
- static analysis of a submitted Android package
- dynamic analysis to detect system manipulation and pattern of behavior of the executed APK, specifically:
  - running processes
  - reloaded data sets, files and libraries
  - network activities.
- platform independent analysis system for a version overarching inspection of Android malware
- creation of a clean runtime environment for every analysis
- individual setting of the parameters during runtime as well as the count of the simulated user entries
- adaptable and resilient system
- logging the process of analysis and reproducible results
- complete automatized and parallelizable inspection.
The Open-Source-Development-Platform *Wandboard*
*i.MX6* was chosen as the hardware basis for the system presented here. A systematic analysis methodology was developed for this Android platform.

2. Related work

The methodical inspection of mobile malware has already been thematized in many scientific works. A detailed list and a classification of important projects related to this work can be found in the publications of Tam *et al.* [13], Irolla *et al.* [14] as well as in the work of Meng [15]. The analysis system *DREBIN* [16] and the paper from Kang *et al.* [17] also show interesting approaches to the subject of tracing static informations from malicious applications. However, the methods mentioned in these works only partly analyze and deduce patterns of behavior.

Along commercial solutions there are also freely accessible systems available, which use an automatized malware analysis combining static and dynamic methods. The modular build framework *ANANAS* [18] bases its inspection on an Android software emulator. The web-application *Mobile Sandbox* [19] as well as the analytic system *ANDRUBIS* [20] – an extension of the web-service *Anubis* – both use outdated Android versions for the analysis of malware. *Anubis* also isn’t openly accessible anymore even though a further development of the project is planned.

Attempts for the inspection of runtime behavior of mobile malware on Android platforms were made by the Open–Source–Project *BareDroid* [21] and by the automatized analytic system *Glassbox* [14]. With *BareDroid* it is possible to create a scaleable infrastructure of real devices on the basis of bare–metal virtualization. However, no independent methodology or routine for dynamic or hybrid analysis of malware is implemented. The limit of *Glassbox* is reached with the automatized reconstruction of a clean runtime environment on the compromised Android device as soon the inspected malware gets privileged access (root access) to the device.

3. Methods

3.1 General system architecture

Fig. 1 shows the developed methodology. The submitted APK is downloaded and installed via web-interface onto the user chosen Android platform. The application performs over a predefined time span on the Wandboard, which overlooks and records the runtime behavior. The results of the analysis will be edited automatically and will be presented in form of a digital report. A detailed presentation of the used hardware and software as well as of the routine of analysis will take place in the following sections of the chapter.
devices simultaneously and control them. With that it’s possible to realize a parallelization of the inspection and a gradual extension of the analysis platform.

3.3 Applied techniques of the analysis

Mobile malware can be divided into different types [24] depending on the method of distribution and attack kind as well as its behavior and purpose [25], [5]. A multitude of circulating malicious apps are to be seen as hybrids or blended threats. Due to this the detection is even more complex.

Therefore, the static analysis of a submitted APK file can give a first insight but reaches its limits if it encounters mobile malware whose malicious code is reloaded during runtime. That is why the significance lays with the dynamic analysis of produced or monitored data of the malware, because these provide the basis for the detection of manipulation and reveal patterns of behavior. Important here is the following information of the Android Analysis System that is being documented during the process:

- documentation of system time and system date
- documentation of running processes
- documentation of installed applications
- documentation of registered and logged in users and
- documentation of established network connections

Furthermore, application specific process analysis and network analysis takes place so that dynamic runtime behavior, started services and relations to the potential malicious application can be identified. Like this it is possible to listen on connections to extern servers and collect non-persistent data of the application which is only available during runtime. In addition, a timeline analysis in relation to a file system analysis is performed. It follows a time-based surveillance of the accesses on the Android file systems to identify used and reloaded files that were created during the analysis. Even though the collection of manipulated files is CPU intensive and the evaluation very complex, through this process the patterns of behavior can be adequately detected and represented.

To effectively produce usable and profound information on the system level during the analysis, an access through the ADB Shell with extended privileges (root access), is needed [26].

3.4 Utilized software

The developed routine of analysis is solely based on free utilities and Android-integrated tools. Because root access is mandatory for some of them, the Android system has to be adjusted accordingly. The management of root authorizations for apps is done through SuperSU [27].

Linux-integrated tools usually are very restricted in their range of functions under Android or are not available at all. This gap is closed by the program BusyBox – the extension of important utility programs for Android [28].

For the recording of traffic in Linux-based systems the free tool tcpdump is an adequate solution for network analysis. This program intercepts the file packages directly at the network adapter and writes them into a file. Android tcpdump is a, for ARM-based Android systems, adapted extension of the program [29] and is used for the network recording in the Android Analysis Systems.

Information regarding the requested permissions and resources can be found in the Android-Manifest. However, in the APK this file only exists in binary format. A remedy for that problem presents – for Android devices – aapt for ARM [30] an extension of the Android SDK integrated Android Asset Packing Tool. The extension enables the user to readout the given APK on the Android device.

In the context of the analysis the host communicates with the Android on the Wandboard exclusively terminal-based via ADB. Therefore an installation of the utility program adb – a component of the Android SDK Platform Tools [31] – is required.

3.5 Applied methods of the analysis

General system information can be determined before installation, after execution and completion of the submitted application with the tool bugreport. These informations include the exact system time, data about running processes and threads including storage occupancy and system capacity among other things. Additionally installed packages are listed before, during and after the analysis. A process-dump is created after the start of the application via package manager (pm). All outputs get diverted into separated files, which are generated during the runtime of the analysis.

To automatize the process of analysis it’s needed to determine specific information from the APK. These will be temporarily stored in variables. The – through aapt for ARM extracted – package name is required to start the application via ADB Shell. Furthermore, the structure and content of the APK is being listed and the AndroidManifest.XML readout with this tool. Information such as package name from the installation, configuration- and process data, requested permissions of the application, resource usage and services can be researched through the .XML file. This data can be used to draw conclusions regarding the pattern of behavior and the intention of the to be analysed Android Malware.

In combination with dumpsys the User ID (UID) of the installed application is filtered from the package name.

Detailed information of established network connections can be traced with netstat. Because of the larger scope the busybox-integrated tool is used. That’s how periodic connection parameters like the kind of network protocol, status of connection or sockets, Process ID (PID) and application name of the connection owner, the local network address as well as the network address of the counterpart station can be requested and saved before and after installation as well as after the performance and completion of the application. The
Network traffic at the WLAN interface is captured with Android tcpdump. All package data is redirected to a previously generated .PCAP file, which builds the basis for the depiction and evaluation of the centralized data such as date and time as well as content of the transmitted and received packages. Ideally patterns of behavior of the monitored application can be detected and attacks reconstructed with this information.

Some types of malware reload additional – device specific – malicious code in the form of libraries or other files from distant servers or extract informations from the device and transmits them to distant systems during runtime. The detection and storage of this data is especially important for the later evaluation. All system and application data that is created or modified during the installation or during runtime of the inspected application is of especial interest. This data can be gathered through the analysis of the MAC times of the files. The tool find from the portfolio of busybox provides the opportunity to search files and directories within the given filesystem or partition. Through the setting of search criteria and the interlinking of activities all relevant files regarding the runtime can be filtered. In combination with the selected UID of the installed app especially those files are saved, that have been manipulated by the aforementioned application. This methodology a collection of all filetypes, especially application- and non-persistent process data as well as meta data, is ensured.

User activities during the application runtime are of importance for a dynamic analysis. Some types of malware activate malicious code or malevolent actions exclusively during the use of the app [19]. The input of the analysis system can be simulated through the command-line based utility monkey which is included in the Android SDK [32]. It starts the submitted application on the analysis-system via transmitted package name and after selection it runs the defined number of pseudo user events (clicks).

Table 1 shows the used methods or tools in the chronological context of the analysis.

### 3.6 Automation of the analysis process

One important goal of this project is the full automatization of the application analysis without user access to the Android system. The basic concept of this process is the terminal-based system entry to the host as well as to the Android system in addition to the scripting languages. This is realized by batch processing. So, the whole process routine of the Android Analysis Platform can be automatized based on scripting. To accomplish this the following files are of significance:

- A configuration file on the host to deliver user inputs (analysis parameter).
- A batch script on the host for the initialization of the Wandboard to the setup of the ADB connection as well as to initiate the collection routine on the Android Analysis System.
- A shell script on the Android Analysis System for the execution of the data collection.

The developed process of the malware analysis is shown in a simplified manner in Fig. 2. The APK is copied to the host through the web-based front-end. The Android version and user specific analysis parameters – name of the submitted APK, time frame of the application runtime and the number of clicks – are defined. The user initiates the batch script on the host through the website. The submitted parameters are transmitted and an ADB connection to a clean Wandboard is set up. Due to the script the data collecting process on the Android Analysis System is initialized. All standard streams and the results of the individual process steps are stored in
datafiles, which have been generated during runtime of the analysis. After completion of the data collection the results are packed into an archive and transferred to the host. After the host-based evaluation the results are presented on the website in the form of an edited report.

3.7 Validation
A proof of concept of the described analysis system and of the automatized process routine was reviewed in a first phase on several malware samples.

4. Results
The functionality of the developed platform will be demonstrated with analysis results gathered from a piece of malware: Android.Trojan.Iop.H\(^1\) [33] on an Android platform version using API Level 19. Because of the complexity and amount of collected data the information will only be explained in excerpts.

4.1 Overview of the system data
General information or system manipulations that are connected to the installation and use of malware can be determined through the secured outputs of the created system reports. Because of the complex protocol data only exemplary values are shown in Table 2. These depict the significant change of the system.

<table>
<thead>
<tr>
<th>Table 2: Compilation of the system reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Processes</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>PID of Zygote</td>
</tr>
<tr>
<td>UID (active)</td>
</tr>
<tr>
<td>Free RAM</td>
</tr>
</tbody>
</table>

The generated report of installed apps reveals, that a package called /data/app/com.android.sensma-1.apk=com.android.sensma was installed while the analysis.

4.2 Results of application- / process analysis
In addition to the archive structure, including file names and resources from the APK the evaluation of the AndroidManifest.xml also shows that the app has requested permissions which allow extensive control over the OS:

- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- android.permission.CHANGE_WIFI_STATE
- android.permission.GET_TASKS
- android.permission.INTERNET
- android.permission.READ_PHONE_STATE
- android.permission.WRITE_EXTERNAL_STORAGE

The application can modify network connections, readout private as well as device-specific information, reload data from the internet, manipulate extern media and edit and delete files. With the readout package name important information can be filtered from the dynamic generated dump of the running app:

- com.android.sensma/u0a45 pid=3384 uid=10045
- com.quick.swipe.activity.MainActivity.alias

In relation with this information it can be seen that four processes and two services have started during runtime of the application, which request periodical specific device parameters like IMEI, Device name and -model as well as Android version and location. This can be determined from the system reports in the context of the UID of the monitored app.

4.3 Results of the file system analysis
The results of the file system analysis show the reload of temporary data files from the internet into the directory: /data/data/com.android.sensma/files after execution of the app. The more interesting files are: onekey- rootseckill.sh, psneuter.script, realroot, install-recovery.sh and busybox. This selection illustrates that these are tools to gain root access to the Android. Furthermore, the file: mobclick_agent_cached_com.android.sensma4314 is created in the same folder. In this .XML file the requested hardware information and other details of the device like the current root status is listed. The detected modification of the file settings.db in the directory /data/data/com.android.providers/settings/databases stands in relation to this action. All affected files are stored during the runtime and are copied to the host after the analysis the temporary, non-persistent and modified files can be examined.

4.4 Results of the network analysis
As apparent from the network analysis after executing the malicious app a number of connections to distant servers are established.

<table>
<thead>
<tr>
<th>Table 3: Established remote server connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP-Address</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>106.38.193.23</td>
</tr>
<tr>
<td>178.162.216.179</td>
</tr>
<tr>
<td>110.173.196.36</td>
</tr>
<tr>
<td>140.205.159.56</td>
</tr>
<tr>
<td>103.235.46.39</td>
</tr>
<tr>
<td>140.205.156.81</td>
</tr>
</tbody>
</table>

With the use of the determined UID and PID the results of the network surveillance can clearly be attributed to the malware, which already reloads data right after the execution. Altogether 13 TCP-connection streams are identifiable. The
communication between the Android and the remote servers is distributed as listed in Table 3.

The evaluation of the recorded packets shows that the executed app reloads files to the Android system and transmits private device data to remote servers.

5. Discussion

The simulation of a runtime environment on real hardware is an important criteria for significant results because of anti-forensic methods in mobile malware. Generally an automatized research of different malware families is possible by the presented Android Analysis System and the implemented analysis routine. The limits of the developed methodology and process routine are reached with dynamic behavior analysis of types of malware like Filecoder/Ransomware, which encrypt whole directories or partitions of the operating system after activation. There also exists a necessity for the ability to root the particular Android version to be able to establish the analysis routine on the device. The Wandboard however doesn’t have a GSM module by default so the detection of certain types of malware is currently not possible. For example the transmission of Premium SMS can not be detected and a phone call can’t be reconstructed. This gap can be closed with the extension of the platform through a GSM module via UART interface. Further the dissection process becomes slower in bare metal analysis compared to the very easy process to reset an emulator to a clean state.

As described all process relevant information for dynamic analysis of malware can be determined independently from the API Level and can be collected for further evaluations by the Android Analysis Platform. Through the use of script-based instruction sequences the process operation is fully automatable, very efficient and resource-sparing.

To achieve significant results root access to the Android system is mandatory. But this is unsuspicous for most malware, because rooting of smartphones is very common nowadays. The Android Analysis System itself is unsucceptible against Rootkits because the memory card is completely deleted after analysis, unlike on real devices. This guarantees a clean runtime environment after the installation of a new Android image.

6. Conclusions and future work

In this work a hardware-based and automatized Android Analysis Platform on the based upon the Wandboard i.MX6 was presented. First the information of a submitted APK is determined and stored by means of static analysis via web-based front-end. In a second step a wide scale of information regarding runtime behavior of the potential malware is collected and completely gathered through dynamic analysis of the corresponding processes. The network activity is completely recorded as well as all modifications of the file system. In combination with the stored system conditions which were collected before, during and after the analysis, the pattern of behavior and intention of the analysed app is discernable. The results are given to the user in form of a edited and automatically generated report.

The whole process works script-based on the system level, whereby significant data can only be determined in combination with an established root access and in relation to that a command extension of the Android system. The functionality of the developed strategy has been tested on several types of malware. Through different test series the determined results can be comprehended and consistently be reproduced. Because of the individual behavior of malware adaptable parameters of analysis have been proven to be very effective. Furthermore, the productivity of the approach in the context of integrity or validity of the gathered data, the processing speed and use of resources of the analysis system was verified.

A major advantage compared to software-based analysis is to simulate the malware on a real Android runtime environment. This is a significant point that helps to prevent the manipulation of analysis results through the change of patterns of behavior.

With the use of removable media in the form of microSD cards as boot medium the application analysis can be executed in a low-cost and a time efficient way. Additionally, the Android system can be checked and completely reset after the analysis.

As a next step of development of the project it is planned to transfer the gathered knowledge from the malware analysis in combination with the Android version and used parameters to a database so as to use the data to improve future reports. An optimization of the evaluation routine and a selectable level of detail of the reports is also going to follow.

The system of analysis was first developed on the bases of Android 4.3 (Jelly Bean) and Android 4.4 (KitKat). Because of the strong fragmentation of the versions of the mobile platform the currently used software statuses as well as
Custom-ROM will be configured and prepared as target systems in the following project progression. In relation to that extensive test series with malware samples and an extension of static and dynamic methods of analysis like the processing of .DEX files or the recording of System Calls are intended.

The Wandboard is a PXE-capable system and that is why a central PXE-Boot-Server is planned to be build to distribute the various Android images.

As illustrated in Fig. 3 that aim is to run a parallel analysis on different Android versions or several isolated systems. Because of the hardware-based automatization of the individual Android systems and of the host the whole analysis platform can be gradually and cost effectively extended. This way future proof Android Analysis System is guaranteed.

References