Indian Journal of Science and Technology, *Vol 9(48), DOI: 10.17485/ijst/2016/v9i48/102493, December 2016*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Detecting Blacklisted IP Access from Android Phone

Parag H Rughani*

Institute of Forensic Science, Gujarat Forensic Sciences University, Gandhinagar – 382007, Gujarat, India; parag.rughani@gmail.com

Abstract

Objectives: To develop a method to detect access of blacklisted IP from Android. **Methods/Statistical Analysis**: Android Operating System source code under AOSP was modified and customized to achieve objective of the work. Work was tested on various simulators with variety of black listed and whitel listed IP to confirm outcome. **Findings:** A well-defined method was derived and verified based on the results of tests carried out during the work. Outcome of the work is an implementation on Android operating system, which customizes the way device connects to an IP through internet. The customization done on operating system helps in identifying interaction of any black listed IP to or from an android device. A supporting python script is also written to automate steps related to interception and interpretation. As being open source, the solution is also extendable to accommodate more features in the same domain. **Application / Improvements:** The method developed during this research can be used in behavioral analysis of android malware.

Keywords: Android, Android Malware, Black Listed IP, Malicious Website, Malware, Malware Analysis

1. Introduction

Android malware is becoming a major concern for android users and security professionals. As per Kaspersky's SecureList Report[1]: "From the beginning of January till the end of December 2015, Kaspersky Lab registered nearly 17 million attacks by malicious mobile software and protected 2,634,967 unique users of Android-based devices". These figures are increasing day by day as the popularity and use of android phones are increased.

Most of the android based malware communicate with some remote machine (command and control center) either for getting instructions as it does in ransomware or to send data / information stolen from device to the attacker. Whatever intension is there, these malware are mostly dependent on remote machine and always need to communicate with it.

This paper is based on detecting such malicious transaction with help of custom Android OS in simulated environment. The work done here is focused on intercepting the ip address and / or domains requested by application under test. The automated process based on python script with custom Android OS offers a powerful and customizable solution than other existing options.

The work done during this research can be considered as small aspect of Intrusion Detection System as it deals with malicious activities including attempts to communicate any black listed IP addresses.

Main objective of this work is to provide a first-step quick idea about malicious communication in android malware analysis. Many researchers are working in malware analysis and especially sufficient work is done in android malware analysis also, but very few of them have worked

*Author for correspondence

on intercepting ip address (URL). Main reason behind less work done in this regard is existence of interception tools like wireshark which provides in-depth analysis of data sent and received from android[2]. Though wireshark is very powerful it also has pros and cons. This section discusses work done in android malware analysis.

In malware analysis "host-based approaches are required, since network-based monitoring alone is not sufficient"[3]. Most of the researchers working on Android Malware Analysis and Intrusion Detection use variety of parameters in their research as some authors focused on call logs to detect malicious activities[4], while others discussed importance of Function Oriented approach in Malware Analysis[5]. Similarly, an extensive feature based approach is proposed by some authors[6]. One of very relevant works related to this paper discusses taint tracking to monitor privacy sensitive information[7].

Apart from intrusion based malware analysis there are various other approaches to analyze a malware. Code Clone using String Pattern Back Propagation Neural Network Algorithm[8], System Call Analysis[9] and use of Bayesian Technique and Nymble Algorithm in malware detection[10] are few other contribution in malware analysis.

2. Proposed Technique

Behavioral malware analysis on any platform is suggested to be carried out in sandbox or simulated environment. There are various sandboxes available for desktop computers including cuckoo. Similar to desktop computers, mobile malware analysis can safely and efficiently done in simulated environment. There is an advantage for mobile devices, as most of the Mobile Operating Systems provide their own emulators / simulators for application developers. Since, they are more reliable and efficient than third party sandbox these emulators / simulators can be used in malware analysis on mobile phones.

Importance of Mobile Sandbox in static and dynamic analysis of mobile malware including Android is discussed by some authors[11]. While some authors proposed and claimed that their sandbox is capable of performing static and dynamic analysis[12]. As various researchers have proposed various sandboxes for Android platform, a detailed Android Sandbox Comparison is also given by some authors[13], which can be used by researchers in choosing appropriate sandbox for their work. So far,[14] and[7] are the most popular sandboxes used by developers and researchers.

Keeping in consideration, importance of sandbox in behavioral analysis of malware, I have selected to work on Android Emulator by customizing AOSP and writing a python script to automate necessary steps for complete analysis.

Proposed work is focused on very basic but very important and crucial information required in any malware analysis. The idea behind the work is to provide customizable way of intercepting requested IP address and / or URL for detecting malicious transactions initiated by an application running on the android phone. The method further compares this IP Address with a list of blacklisted IPs.

To achieve my objective, I customized Android OS Version 5.0.2 by modifying its source code. The changes are made in files required for intercepting IP address. The requested IP addresses are printed using Log for interpretation at the runtime. Customized android source code is then compiled to get binaries (img files) required by the emulator. The detailed process on making code and running emulator can be referred from official android source website[15].

A python script is written to automate steps required in performing the task. Once the emulator is running the script can be executed before beginning behavioral analysis of sample. The script once started extracts emulator log automatically. These logs are than interpreted to filter IP Addresses. The script then compares details extracted from emulator log with the list of blacklisted IPs. Python script is capable of getting updated list of blacklisted IPs from predefined URL(s). Final results retrieved from the script are stored in a separate file called analysis.txt. The results are very user friendly and can help any layman in understanding them. Following steps with screen shots explains the proposed technique.

3. Implementation

Assuming that user has Android SDK configured on his computer with the custom img files generated as an outcome of this research, the first step in implementation is to start the android emulator. The emulator with help of custom image can be started using following command as shown in Figure 1.

```
WORKING_DIRECTORY$ emulator -sysdir out/target/product/generic/ -system out/target/product/generic/system.img -ramdisk out/target/product/generic/ramdisk.img -data out/target/product/generic/userdata.img -kernel prebuilts/qemu-kernel/arm/kernel-qemu-armv7 -sdcard out/target/product/generic/mysd.img -skindir sdk/emulator/skins -skin WVGA800 -scale 0.7 -memory 512 -partition-size 1024
```

Figure 1. Command to execute android emulator with custom images.

Once the emulator is loaded one can install sample apk (suspected malware) into the emulator using adb install command.

I have used an existing application called Server Status Checker which is designed to check status of a server based on ip addresses or URL, the sample application taken here is not a malware but it is used to illustrate purpose of IP Address interception. The Black listed IP used for testing (i.e. 221.214.10.16) is set to the testing application and can be seen in Figure 2.

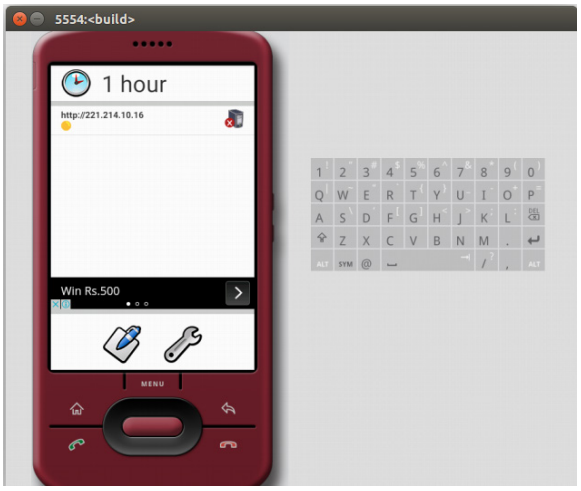

Figure 2. A sample application to test proposed technique.

After installing application or many times before the installation one can start the python script, which runs in the background and automatically intercepts, interprets, and compares IP address with existing list of black listed IP addresses.

I used open source feeder[16] as a source to get list of black listed IP Addresses. The reason behind selecting openbl is its real time updates and rich collection. The IP addresses are checked against the IP addresses intercepted using custom image and python script. The analyst based on the requirement can use any source instead of openbl.org. Since, the script is written in python, researchers and analyst can modify it to get list of black listed IP addresses from multiple sources (files or websites). Figure 3 shows a screen containing openbl data at the time of experiment.

Again the ip used for testing is highlighted as one of the blacklisted IP and can be seen in above figure.

The script keeps running in the background till the emulator is running. Figure 4 is a screen shot of running script.

Figure 3. Openbl.org data.

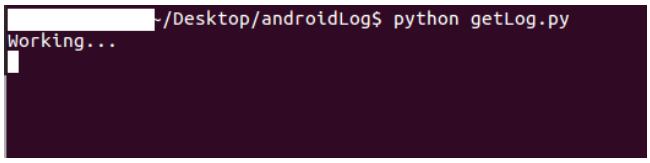

Figure 4. Python script is running in background.

As the analyst closes the emulator the script performs necessary steps to generate final analysis.txt file. The script after completing tasks displays appropriate message to the end user as shown in Figure 5.

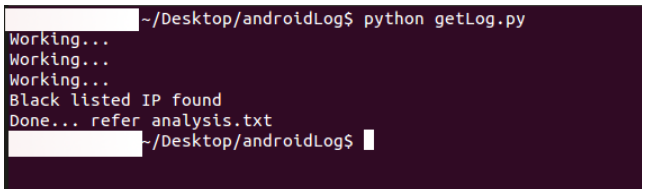

Figure 5. Output of the script.

The analysis.txt file contains list of black listed IP addresses (if there are any) which application tried to connect. A sample analysis.txt file generated from this experiment can be seen in Figure 6.

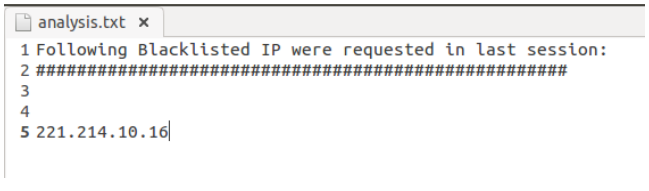

Figure 6. Analysis report generated by the script.

The analysis.txt file generated after the test run here shows IP address: 221.214.10.16, which was set in the application running into the emulator. The IP address

is blacklisted as per the openbl.org records. Now, when sample application will try to connect to the above IP, custom OS source code will intercept it with other IP access requests. While the automated python script running in the background will sense this IP address and will compare it will blacklisted IP addresses (as retrieved from openbl.org).

As a result it will mark this IP address and will put it in final resultant file called analysis.txt with appropriate message.

Same experiment can be done with any suspect application to check if it is connecting to any black listed IP. In case if it does then it becomes very easy for malware analyst to put it in malware category without going in detailed analysis. This quick and affordable technique in no time can tell whether the application under test is communicating to any unexpected IP or not.

4. Future Scope

The technique used in this research can be further enhanced to detect information sent and received to any IP address. Already, there are some solutions available in the market which intercepts packets including wireshark, but the solution used here gives more options for customization to researchers and analyst.

Another expansion to work can be related to detection of zombie running in an android phone. Based on frequency of the request sent to an IP address one can analyze whether the device is being used as a zombie for causing DoS / DDoS.

5. Conclusion

The final outcome lets users customize each component involved in achieving final objective. This not only makes the solution more acceptable, but it also leads to detailed investigation. Compared to other solutions like wireshark and bundled sandboxes available for Android, this solution is light weight and heavily customizable.

The results generated by the python script are very specific and user friendly, which can be easily understood by a non-technical person or a beginner. The proposed technique can also become first step for android malware researchers in developing new techniques by enhancing this solution or by creating new solution based on this concept.

6. References

1. Unuchek R, Chebyshev V. Mobile malware evolution in Securlist from Kaspersky Lab, 2016.
2. Banerjee U, Vashishtha A, Saxena M. Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection. International Journal of Computer Applications. 2010; 6(7):1–7.
3. Miettinen M, Halonen P, Hätönen K. Host-based intrusion detection for advanced mobile devices in Proceedings - International Conference on Advanced Information Networking and Applications, AINA. 2006; 2: p. 72–6.
4. Saudi M, Ridzuan F, Basir N, Nabila N, Pitchay S, Ahmad I. Android Mobile Malware Surveillance Exploitation Via Call Logs: Proof of Concept in 17th UKSIM-AMSS International Conference on Modelling and Simulation.2015;176–81.
5. Jang J, Kim H. Function-Oriented Mobile Malware Analysis as First Aid in Mobile Information Systems. 2016;1–11.
6. Muttik I, Yerima S, Sezer S. High Accuracy Android Malware Detection Using Ensemble Learning in IET Information Security. 2015; 9(6):313–20
7. Enck W, Gilbert P, Chun B, Cox L, Jung J, McDaniel P, Sheth A. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones in ACM Transactions on Computer Systems (TOCS). 2014; 32(2): 5.
8. Kaur S, Kaur A. Detection of Malware of Code Clone using String Pattern Back Propagation Neural Network Algorithm. Indian Journal of Science and Technology. 2016; 9(33):1–12.
9. Malik S, Khatter K. System Call Analysis of Android Malware Families. Indian Journal of Science and Technology. 2016; 9(21).
10. Jeyaseelan W, Hariharan S. Malware Detection and Elimination using Bayesian Technique and Nymble Algorithm. Indian Journal of Science and Technology. 2015; 8(34):1–7
11. Spreitzenbarth M, Freiling F, Echtler F, Schreck T, Hoffmann J. Mobile-Sandbox: Having a Deeper Look into Android Applications in Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013; 1808–15.
12. Asing T, Batyuk L, Schmidt A, Camtepe S, Albayrak S. An Android Application Sandbox System for Suspicious Software Detection in proceedings of Malicious and Unwanted Software (MALWARE). 2010;–55–62.
13. Neuner S, Veen V, Lindorfer M, Huber M, Merzdovnik G, Mulazzani M, Weippl E, Enter Sandbox : Android Sandbox Comparison in 3rd IEEE Mobile Security Technologies Workshop. 2014.
14. Droidbox website. 2015 September 25. Available from: https://github.com/pjlantz/droidbox
15. OfficialAndroid Source Code Website. http://source.android.com/.
16. Official OpenBL Project website. 2009 Dec 26. Available from: http://www.openbl.org/lists/base.txt