International Journal of Electrical and Computer Engineering (IJECE)
Vol. 6, No. 1, February 2016, pp. 249~256
ISSN: 2088-8708, DOI: 10.11591/ijece.v6i1.9012

Network Activity Monitoring Against Malware in Android Operating System

Luis M. Acosta-Guzmán*, Gualberto Aguilar-Torres, Gina Gallegos-Garcia***
* Department of Research and Graduate Studies, Electrical and Mechanical Engineering School
Instituto Politécnico Nacional, Mexico
** Comision Nacional de Seguridad. Secretaria de Gobernacion, Mexico

Article Info

Article history:

Received Sep 12, 2015
Revised Nov 15, 2015
Accepted Nov 30, 2015

Keyword:

Android
Malware
Methodology
Network Activity Monitor
Security

ABSTRACT

Google's Android is the most used Operating System in mobile devices but as its popularity has increased hackers have taken advantage of the momentum to plague Google Play (Android's Application Store) with multipurpose Malware that is capable of stealing private information and give the hacker remote control of smartphone's features in the worst cases. This work presents an innovative methodology that helps in the process of malware detection for Android Operating System, which addresses aforementioned problem from a different perspective that even popular Anti-Malware software has left aside. It is based on the analysis of a common characteristic to all different kinds of malware: the need of network communications, so the victim device can interact with the attacker. It is important to highlight that in order to improve the security level in Android; our methodology should be considered in the process of malware detection. As main characteristic, it does not need to install additional kernel modules or to root the Android device. And finally as additional characteristic, it is as simple as can be considered for non-experienced users.

Corresponding Author:

Gualberto Aguilar Torres,
Comision Nacional de Seguridad. Secretaria de Gobernacion,
Av. Constituyentes #947 Col. Belén de las Flores,
Del. Álvaro Obregón. Distrito Federal CP. 01110, Mexico
Email: gualberto.aguilar@cns.gob.mx

1. INTRODUCTION

Nowadays, personal computing is making a turn into mobile devices. While a few years ago cellphones were intended to provide people a way to make phone calls and communicate regardless the location; today, we talk about smartphones as a result of adding more and more functions as well as computational power with the passage of time.

Thanks to the smartphones people is now able to perform calls, video calls, use messaging services over the Internet, send and receive emails, use banking services and even use social networks wherever they are. Practically, everything done on a PC or Laptop can be accomplished now on a Smartphone; in most cases, thanks to the appearance of little pieces of software with a specific purpose commonly known as applications (Apps).Today's Smartphones can be classified depending on the Operating System running on the device into Google's Android, Apple's iOS, RIM's Blackberry OS, Windows Phone, and Symbian [1]. Each one of them with its own Application Store where the user can download paid and free Apps developed for people and enterprises around the globe.

According to Nielsen, leader Company in market research, 51.8% of smartphones were using Android by the end of June 2012 [2]. In other words, it has become the most used mobile operating system. Considering that and due to Android is the simplest option for developers of Apps (since it is free licensed). In addition to its Apps can be developed using Java and developers can publish their Apps making them

available immediately, this work focuses strictly on Google's Android as an operation system for smartphones and Tablets.An example of Android's growth is that until March 15th 2012 there were 450,000 Apps available in Google Play (Android's Application Store) while at the middle of 2010 there were only 100,000 [3]. Today, we are talking about more than 700,000 Apps according to Bloomberg's BusinessWeek [4, 5].

Unfortunately, not everything is good for Google's Android. As a result of its popularity and the lack of code validation and testing of the uploaded Apps, Android has become the weapon of choice for hackers to introduce malicious code into Smartphones. Malicious code, also commonly known as malware, could allow a remote attacker (hacker) to accomplish different things from stealing personal private information to taking full control of the device e.g. send text messages or perform phone calls [6]. In fact, this is the most important reason this work focuses on Android.

Although, Google has added many security features to its operating system such as application isolation, the permissions model, read-only access to the Android's kernel, no root permissions by default, among others, there is plenty of malware disguised as good Apps available for the user to be downloaded in Google Play.

The principal effort from Google to avoid the presence of malware in Google Play is called "Bouncer" and it was presented in February 2nd 2012. At its presentation, Google claimed that it had been already running for a while with the purpose of filtering malicious Apps even before they showed up into Google Play. At this point, Google said that between the first and second haves of 2011 they saw a decrease from about 40% in the number of potentially malicious downloads from Google Play (or Android Market as it was called then) [6].Even though, this was supposed to be the solution to make Android safer, some failures and weak points were found in Bouncer. John Oberheide and Charlie Miller, two white hat hackers, presented their analysis about Bouncer at the SummerCon 2012 with a work titled "Dissecting the Android Bouncer" [7].

Oberheide and Miller found out that Bouncer is nothing more than an Android Virtual Machine running in Google's Infrastructure analyzing each App before it is published. Nevertheless, the most important thing they were able to find out and demonstrate is a way to bypass Google Bouncer's validation.

Similarly, the malware problem has opened a window for Antivirus software to become mobile. Nowadays there is a wide variety of Apps intended to give protection against malware and other kinds of Information Security risks available in Google Play, in which is important to consider that whatever purpose malwareis, it will always need a network connection to accomplish its goal, this connection is required mainly so the victim sends back to the attacker what in general terms can be referred as stolen data and also the victim can receive instructions from the attacker.

AV-TEST Institute, an independent laboratory for Information Security and Antivirus research, published a test report in March 15th 2012 called "Anti-Malware Solutions for Android" where they put to test 41 different Anti-Malware solutions. As a result, they grouped the solutions in five sets according to its average capacity of detection [8].

As a part of the research performed in the development of this work, three solutions were tested from the set with an average detection rate of more than 90% (Avast, Mobile Security, Kaspersky Mobile Security and Lookout Security & Antivirus). That was done in order to see if they could identify a threat based on the presence of abnormal network activity. The three solutions were evaluated with a simple App coded to generate multiple connections in a loop to a remote server and also to open multiple ports in a loop receiving connections from a remote client. Unfortunately, none of the solutions were able to identify the custom App as a malicious or suspicious one. Moreover, none of them was even able to report the abnormal network activity generated by the custom App.

By analyzing the chosen Anti-Malware solutions was identified a Firewall feature, this function basically allows the user to choose which Apps can access the Internet over 3G or a Wireless connection and which cannot; it does not work strictly as a Firewall since it does not give an option to block or permit certain ports. Firewall feature could help but it becomes complicated to use since this particular component requires root permissions on the device as well as the support of netfilter/iptables (packet filtering framework) in the Kernel.

It is important to remember that Android's kernel has root permissions disabled by default but the user can gain this kind of permission by running a special crafted piece of code directly to the shell. From the standpoint of Information Security, enabling root permission on an Android device is not recommendable for two important reasons: the user cannot be sure if the rooting software contains any kind of malware such as a backdoor and a rooted device becomes more vulnerable if it gets compromised because any installed malware will get root permissions as well [9].

The present work shows the development and implementation of a methodology to analyze the network activity generated by an Android device in a way that a "complete" security solution or "Anti-

Malware" solution can take advantage of such analysis. That is because the analysis of the network activity by itself can allow the detection of a suspicious behavior in a device but it will never be able to indicate with a hundred percent certainty of malware presence. In other words, tracking the network activity can help to identify suspicious behavior. As in example, when an App intended to be used as a calculator starts to open communication ports in the device allowing any remote client to connect in. In addition to that, it will help to keep a dynamic continuous monitoring of the device searching for malware while actual Anti-Malware solutions principally focus on the detection of signatures at the moment a new App is installed. All of this is done by considering malware will always need a network connection to accomplish its goal, which states the basis to this work by the thought of considering the network activity analysis output as extremely valuable information to be used in the process to determine whether an App is malicious or not.

2. RELATED WORK

Actually, there are no works describing a methodology to gather the network activity in Android, how to take advantage of this information and its importance to determine the malignity of an App. Similarly, there are no Apps in Google play that could achieve our goal. However, application called "Connection Tracker Pro" [10] is the work that could be considered as related with the focus of this present work. Such App is designed to display the network activity of each App in an instant. After we ran some tests, we found that such App is actually a graphic representation of running a "netstat" command in a loop from the device's shell.

This App could be very helpful from the networking perspective point of view, but for people experienced in information security topics it may be considered as suspicious once, it is due to installation requires the "PHONE CALL PERMISSION", as can be seen in Figure 1. Such permission is not necessary and will allow the App to obtain IMEI and IMSI numbers, which in fact can reveal the device's location. It is also noticeable that the App demands a lot of CPU when running making the device really slow. As example we can mention a Sony Tablet S (where we made test) with a single CPU NVIDIA Tegra2 and 1 GB in RAM, which took from 5% to 60% average of its CPU capacity just by running the App.

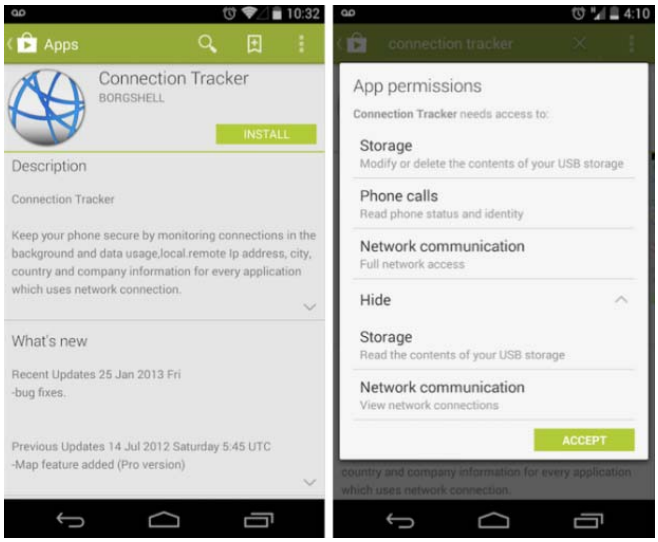

Figure 1. Description and permissions of "Connection Tracker" shown in Google Play

It is also important to mention that "Connection Tracker" was not able to identify a port opened by a custom App (listening on the device). Furthermore, this App does not have any documentation or even a web site where information about the development could be found, even tough, according to Google Play its website should be www.borgshell.com which leads to a non-existing webpage. Even though the goal, use and context of "Connection Tracker" is different and away from the one of the present work, it is mentioned as related work because in the description of the App showed by Google Play, it can be read that this application will help the user to keep the device secured by monitoring the connections.

3. CONSIDERATIONS IN OUR PROPOSED METHODOLOGY

It is important to emphasize that the approach of this methodology consists in the development of the mechanism to keep track of the network activity not the mechanisms to evaluate and detect any suspicious behavior and nor to identify malware. In other words, our methodology shows the implementation of a Network Activity Monitor (NAM) for Android that is capable of running all the time as a background service in order to identify new established connections and attempts of connection. It is also capable of listening open ports, waiting for a remote connection on a device and identifying, which installed App, is responsible for each new connection.

Considering the aforementioned our methodology improves the Information Security level in Android OS along with its three main attributes: confidentiality, integrity and non-repudiation. The NAM App works without the need of root permissions (rooted device) and only requires a few permissions from the user at the moment of installation.

As it has been clarified previously, the network activity monitor by itself cannot be used to identify malware but it could be used as a standalone App working in a blacklisting scheme where a user receives an alert if a selected App (blacklisted) generates network activity. As a consequence, the user running this NAM on its device may have an opportunity to kill the process or App that is generating dubious network activity. Alerting the user is the only reactive measure the NAM App can take without requiring root permissions.

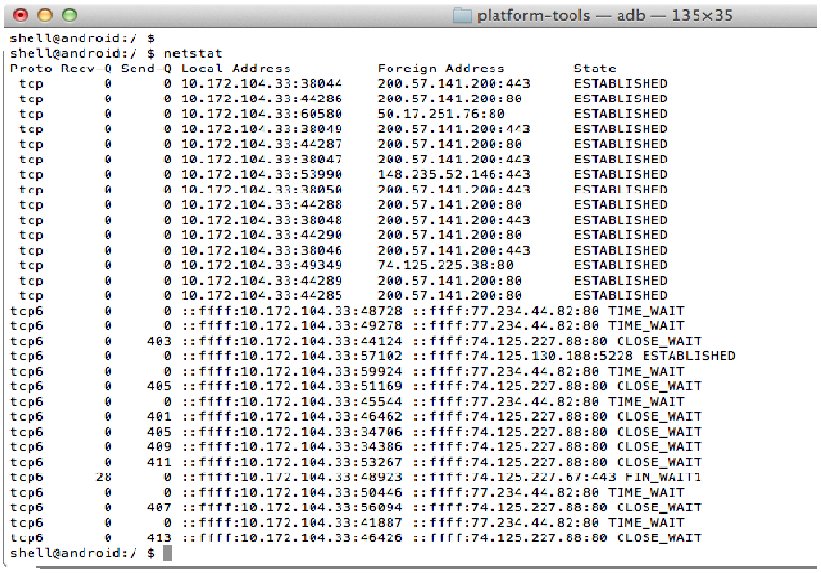

Figure 2. Our proposal uses a call to the system from the kernel

Tests were made trying to provide the NAM with the capacity of killing a blacklisted application automatically as an action triggered by the detection of network activity. It was found that in a rooted device it would be as simple as run the command “kill [process id]” directly to the shell (as in a regular Linux) but in a non-rooted device the user does not have permission to use the “kill” command and the means provided by the Android’s SDK in the classes android and ActivityManager with its methods “killProcess()” and “killBackgroundProcess()” only work to end the process itself and not external processes.

4. PROPOSED METHODOLOGY

This section will describe our entire methodology and the implementation of the Network Activity Monitor for Android as if it was meant to be a standalone application.

The first step is to get the Android device’s current network activity. It can be easily achieved in mobile devices running Android thanks to its Linux Kernel using the command “netstat” just like in an ordinary Computer.

The Android App this work has developed uses a call to the system invoking the command netstat directly from the kernel through the ”Process” and “Runtime” Java Classes and the method “getRuntime().exec()”, as can be appreciated in Figure 2. As can be seen in Figure 2, the netstat command

provides an output showing all active sockets in a table containing important information such as protocol (TCP, UDP, TCP6 or UDP6), the local address (source IP address), foreign address (destination IP address) and also the state of the socket.

The state parameter provides a way to differentiate connections that are already established ("ESTABLISHED") from connections that are initiating ("SYN_SENT"), connections ending ("CLOSE_WAIT", "TIME_WAIT", "FIN_WAIT" and many more) and listening ports that are waiting for a connection from a different host in the network ("LISTEN") [11].

Once the active sockets are obtained it is necessary to determine the state for each element of the output just between three types: active connections, attempts of connections, and listening ports; this will determine the treatment required on the next step. On the other hand, it is important to clarify that due the purpose of this work the rest of the sockets can be discarded (e.g. connections with a "TIME_WAIT" state) since detecting a closing connections would not make a big difference because it would be too late for the user to take actions if alerted; moreover, every active connections is originated with a state "SYN_SENT" or "LISTEN" and become active (begins the data transfer) with a state "ESTABLISHED".

Besides the "State" it is essential to identify the value on the parameter "Proto" of the output for each element. This parameter can only take one of the following values: "TCP", "TCP6", "UDP" or "UDP6". We will get to the importance of this at the moment of process-connection identification.The following is to identify the destination IP address as well as the source and destination ports for each element of the netstat output with a state "ESTABLISHED" of "SYN_SENT; in other words, for active connections and connection attempts. From the elements with a "LISTEN" state it is only possible to obtain the number of the listening port.

Identification can be accomplished by parsing the output just using the "String" Java Class and its methods such as: "trim()", "split()", "charAt()", "replace()" and "indexOf()". Still, it is not as simple as it looks because the character structure of each socket in the output depends on two factors: the value of the parameter "Proto" and the value of the parameter "State". As a result, each case must be considered, that is why it is important to differentiate the connections by its state, as is shown in Figure 3.

Figure 3. It is important to differentiate the connections over netstat output

The source port of the connections becomes really important in order to obtain all possible connections because there can be multiple connections to the same destination address and port but there cannot be two with the same source port. To sum up, each active or initiating connection requires a different non-used source port, as seen in Figure 4.

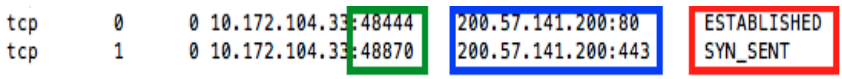

Figure 4. Source port, destination IP, destination port and state of the connections

Identifying address and ports is fundamental not only because it is the connection's detailed and important information by itself but also because it will lead to the identification of the App responsible for each socket. In a regular Linux Kernel or a Windows OS (where the netstat command can also be found) the responsible process or file for a connection can be found by the command itself but not in Android's kernel. Getting the local port and the remote address and port needs to be complemented with the identification of which process (App) is responsible for each connection (considering that one App could be responsible for multiple connections); this can be done by taking a look into certain files stored in the Linux file system containing dynamic information about the network activity. Which file to check depends on the value of the parameter "Proto" (earlier it was entrenched that this value must be identified), there is one file per possible value ("TCP", "UDP", "TCP6" or "UDP6") and all the files are located in the path "/proc/net/" e.g. in a connection with a value "TCP" the file that needs to be reviewed is "/proc/net/tcp". This can be easily done with the command "cat". Hence, the NAM must run the command "cat /proc/net/tcp" (for example) and store the output in a buffer so it can be analyzed, see Figure 5.

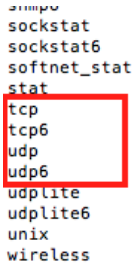

Figure 5. Existing files in the path "/proc/net/" have one possible value

Every file contains the information of current connections associated to that particular Protocol, this information includes the source address and port as well as the destination address and port provided in hexadecimal; moreover, it contains the information of the UID responsible per connection. This UID needs to be obtained and can be done by converting the addresses and ports to hex and searching for them inside the corresponding file, as shown in Figure 6.

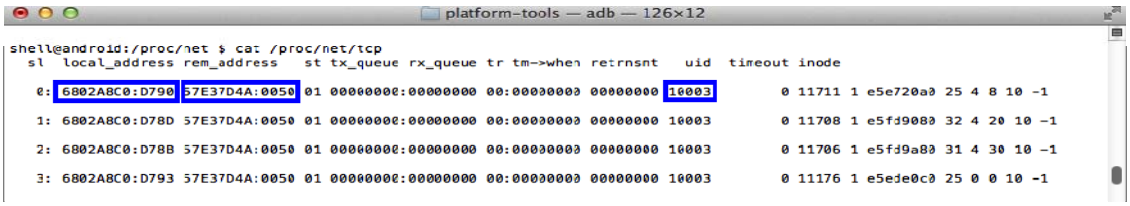

Figure 6. Content of the file "/proc/net/tcp" shows connections to port 80 (0050 in HEX) with the UID 10003

UID is an integer value provided in UNIX systems for user identification within the kernel. On a regular Linux all the Apps ran by the user would have the same UID; on the other hand, Android's kernel provide a different UID to each running App due to the concept of Virtual Machine (Dalvik Virtual Machine in Android), this is know as "Application Sandbox". Every App running in Android generates a new instance of Dalvik, as a result, every App has a different UID. Although it is possible to share a UID between two different App's, it can only be accomplished by signing the App s by the same Developer [11]. Hence; the identification of the UID can lead to the identification of the Process Name and to the App Name.

Once the UID has been obtained the acquisition of the Process Name can be accomplished through the Android's SDK by using the Class "ActivityManager" with its method "getRunningAppProcess()". This method will bring an Object's List where each element is a running process with attributes such as UID, PID (process id) and Process Name; therefore, a simple search for the UID in the List does the job. Because to regular users would not make much sense to see a Process Name, the App Name (called Label) can also be identified using the SDK

At the end, by using a loop to apply the presented mechanism to every element of the "netstat" output there would have been identified three sets of connections (established connections, attempted connections and listening ports), each element of these sets with a protocol, a source address and port, a destination address and port and a Process (App) responsible for that particular socket. The three sets should be stored in the device as data structures using non plain text files in order to keep a record on a previous state of network activity so the user is not alerted if the NAM detects the same connection many times as long as it is not closed. That is, the developed NAM App requires "android.permission.WRITE_EXTERNAL_STORAGE" being this permission the only required so the App does its assignment.

The average CPU usage of the Network Activity Monitor is the result of a designed test in which a device is operated normally using four different Apps (YouTube, WhatsApp, Google Chrome and Gmail) each one during one minute. During this four-minute operation the Network Activity Monitor keeps running performing all of its tasks while CPU usage information is gathered through the execution of the "Top" tool directly in the Android shell. This test was applied to four different devices of different market segments (from low capacity to high capacity devices) giving as particular results the average commented value of CPU usage.

"Top" is a tool contained in most of the Linux distributions that allow gathering the information about which processes are consuming most of the resources and giving the exact value of CPU usage. For our test the "tool" command was set up with particular values to only capture the top ten processes, checking every two seconds until reach 120 measures writing the collected information in a text file.

5. RESULTS AND DISCUSSION

The resulting App can run indefinitely as a background service without decreasing the performance of the device giving a good user experience. This test was done under a Sony Tablet S with a single CPU NVIDIA Tegra2 with 1GB in memory and the App runs perfectly in the background without making a significant difference to the user experience. Tests were made reporting only an increase of 2-5% of CPU usage when the detection is running. It's important to recall that the user can change the time between the performances of each detection process. Certainly, if we decrease the time, we can identify more dubious network activities.

The firewall feature presented as a feature or as a dedicated App requires root permissions and netfilter/iptables on the device. As a result, it is not easily accomplishable for a regular user; furthermore, rooting a device could decrease the security level on a device. In this sense, our proposal uses storage permissions in order to give agility to the user. Network activity-process identification can be accomplished without rooting the device or asking for permissions.

There is no way to kill a process that has been spotted as a suspicious App because of its network behavior without root permissions. The best chance is to alert the user based on a blacklist of Apps that shouldn't require network access. The user must set this blacklist.

Table 1. Results from the designed test to measure the NAM resource usage

Device	Model	Average CPU usage by App process	Average CPU usage by the system	Average total CPU usage	Average CPU usage by Network Activity Monitor
LG Optimus L7	LG P700	52%	36%	87%	6%
LG Optimus L7x	LG P714	44%	34%	79%	6%
Sony Xperia Tablet	SGPT12	23%	18%	40%	5%
Google Nexus 4	LG E960	9%	7%	16%	3%
				Average CPU usage	**5%**

It is important to mention a similar App was found in Google Play without any documentation available; tests made to this App reflect that it presents suspicious behavior; for example, it requires phone call permissions, it increases the CPU to around the 60% of its capacity and also it was not able to detect a Listening Port opened by a custom App. Moreover best rated Anti-Malware Apps were not able to identify suspicious network behavior.

As it is possible to see in Table 1, results we obtained report only an average 5% of CPU usage caused by the Network Activity Monitor and even when there is a high usage of CPU in a device only a small part of this consumption is caused by the Network activity monitor. This last point is really important since the Network Activity Monitor would not be useful if the device's performance was degraded decreasing the user experience.

6. CONCLUSIONS AND FUTURE WORK

Google's efforts to take down Malware still require to be improved due to Google should provide the Android OS with the Firewall feature by default as part of the system so the user can choose which Apps can access the network and which ones cannot.

Rooting a device to take advantage of a Firewall feature cannot be a good option from Information Security perspective because the benefit is less than the risk taken.

The NAM is truly an opportunity for the user to identify Malware disguised as some cool App downloaded from Google Play and provides a basis work that can be very useful in the future as many the tendency points to move to cloud services.

Considering that the research and fundamental pieces of code are finished in this work, in the future we leave as open research to take this code into a good user interface, to publish the App in order to it becomes available for the Android users along with all the documentation, in addition to the design and implementation of a mechanism to determine suspicious network behavior.

ACKNOWLEDGMENTS

The authors thank the Instituto Politecnico Nacional and the Consejo Nacional de Ciencia y Tecnologia. The research for this paper was financially supported by Project Grant No. SIP-2014-RE/123/CONACyT 216533.

REFERENCES

[1] Jiang Chun-mao, Qu Ming-Cheng, Wu Xiang-hu Hu. Optimization Design of a Realtime Embedded Operating System Based on ISO17356. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* Vol 11 No 10, 2013 pages 5763-5773.
[2] Patrick Mutchler, Adam Doupe, John Mitchell, Christopher Kruegel and Giovanni Vigna. *A Large-Scale Study of Mobile Web App Security*.In Proc. of the Mobile Security Technologies. Pp: 1 – 6. 2015
[3] Sheng-Wen Chen, Chung-Huang Yang, Chien-Tsung Liu. *Design and Implementation of Live SD Adquisition Tool in Android Smart Phone*. In Proc. of the International Conference on Genetic and Evolutionary Computing. Pp: 157 – 162. 2011.
[4] Muhammad Zuhair Qadir, Atif Nisar Jilani, Hassam Ullah Sheikh.Automatic Feature Extraction Categorization and Detection of Malicious Code in Android Applications. *International Journal of Information and Network Security (IJINS).* Vol 3 No 1.Pp: 12-17. 2014.
[5] Steffen Lortz, Heiko Mantel, Artem Starostin, Timo Bähr, David Schneider, Alexandra Weber. *Cassandra: Towards a Certifying App Store for Android*. In Proc. of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices. Pp: 93-104. 2014.
[6] Yajin Zhou, Xuxian Jiang, *Dissecting Android Malware: Characterization and Evolution*. In Proc. of the IEEE Symposium onSecurity and Privacy (SP). Pp: 95 – 109. 2012.
[7] Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, Georg Wicherski. *Android Hacker's Hanbook*. Published by John Wiley & Sons, Inc. Pp: 129 – 174. 2014.
[8] Hendrik. Pilz, *Mobile Security Apps*. AV-TEST the Independent IT Security Institute. Test Report. 2012
[9] Assem Nazar, Mark M. Seeger, Harald Baier. *Rooting Android - Extending the ADB by an Auto-Connecting WiFi-Accessible Service*. In Proc. of 16th Nordic Conference on Information Security Technology for Applications.Pp: 189 – 204. 2011.
[10] Borgshell Developer Team, *Connection Tracker Pro*.Google Play.London, UK 2014.
[11] IMS; Diagnosis Guide and Reference, IBM. *International Business Machines Corporation.* Ver 3. 6th Edition. 2005.

BIOGRAPHIES OF AUTHORS

M. Eng. Luis Miguel Acosta Guzmán received the BS Computer Science and Technology from The Monterrey Institute of Technology and Higher Education, Mexico City Campus in 2011. He holds two certifications: the Ethical Hacking from the EC Council and CCNA from Cisco. He is currently studying a Masters in Information Security. His areas of interest are: Hacking, Computer Forensics and the Android Operating System.

Dr. Gualberto Aguilar Torres received the BS degree on Electronic and Communications Engineer in 2002, the MS degree on Microelectronic Engineering, in 2004 and a Ph. D. degree in Electronic and Communications in 2008, from the National Polytechnic Institute. In 2005 he received the Best Thesis award from the National Polytechnic Institute of Mexico for his Master research work. In 2009 he joints the Graduate and Department of the Mechanical Engineering School of the National Polytechnic Institute of Mexico and nowadays he works at the National Safety Commission in Mexico City.

Dra. Gina Gallegos-Garcia received a MS Degree and Ph. D from the National Polytechnic Institute of Mexico in 2005 and 2011 respectively. She is currently Professor of Graduated Section of Mechanical and Electrical Engineering School and belongs to the National System of Researchers. During the summer of 2011 she performed a postdoctoral research at Yale University in the United States of America. Her areas of interest include The Electronic Voting, the Secure Cryptographic Application Design, Information Systems and Cryptography, Software Engineering.