O. Somarriba, L. C. Perez Ramos, U. Zurutuza and R. Uribeetxeberria, "Dynamic DNS Request Monitoring of Android Applications via Networking," 2018 IEEE 38th Central America and Panama Convention (CONCAPAN XXXVIII), 2018, pp. 1-6.

DOI: https://doi.org/10.1109/CONCAPAN.2018.8596558

Dynamic DNS Request Monitoring of Android Applications via networking

Oscar Somarriba†
Department of Electronics and Computer Science
Mondragon University
Mondragon, Spain
oscar.somarriba@alumni.mondragon.edu

Luis Carlos Perez Ramos
ICT Research Program
National University of Engineering (UNI)
Managua, Nicaragua
l.c.perezramos@gmail.com

Urko Zurutuza
Dept. of Electronics and Computer Science
Mondragon University
Mondragon, Spain
uzurutuza@mondragon.edu

Roberto Uribeetxeberria
Dept. of Electronics and Computer Science
Mondragon University
Mondragon, Spain
ruribeetxeberria@mondragon.edu

***Abstract*—Smart devices are very popular and are becoming ubiquitous in the modern society, with Android OS as the most widespread operating system on current smartphones/tablets. However, malicious applications is one of the major concerns and fast growing security menaces facing the use of Internet in the Android platform, today. So, we need techniques and methods to address the massive malware attacks. One of the most relevant techniques to disclose sensitive behavior of Android applications during their runtime execution is Dynamic Analysis. Here we proposed a malware detection tool, termed as *Network Sentinel*, that it can be used for Dynamic DNS request Monitoring of Apps via networking. The main motivation for this work, it is extensively abuse of the DNS by malevolent communities in order to provide Internet connection within malicious networks and botnets. Finally, the experimental results obtained are promising by allowing us to capture the DNS queries requested by the smartphones to remote servers from the collected network traces at very low battery usage.**

***Index Terms*—Smart devices, Android, Android Malware, Dynamic Analysis, Dynamic DNS request Monitoring, network-based analysis**

I. INTRODUCTION

In the last years, mobile devices with Android operating systems (OS) are widely being used, and now we observe the explosive growth of smart devices in the world due to rapid development of ubiquitous networking and wireless communications. According to forecasts [1], in 2018 the number of smartphones in the world will be over 50% of the total number of phones. Meanwhile, the increasing number of malicious applications (commonly referred to as malware) targeting Android smart devices raises the demand for analyzing them to find where the malware is triggered when user interacts with them. Of course, the outgrowth of mobile smart devices have also been supported by the enhancement of the OS technology upholding them. Therefore, the malware targeting smartphones has developed rapidly due to the popularity of Android OS, which has led to an astronomical increase in the spreading of this kind of malicious programs. Moreover, since mobile users increasingly rely on unofficial or third-party repositories in order to freely install paid Android applications (Apps), lots of security and privacy problems are generated. This raises strong security and privacy issues for both to users and operators [2]. So, logging information from Apps destined for smartphones is becoming essential for evaluating the security of an application or generating test cases for it.

Malware detection is a complex and sophisticated process pulling together monitoring, analysis and identification tasks. With the rise of a new and sophistication generation of smart malicious Apps, mobile malware detection faces several challenges. As it is proposed in [3], a challenging issue would be applying scalable technologies for efficient monitoring and analysis of mobile security events. On the other hand, the next generation of mobile malware detection systems will combine heterogeneous sources such as: malware analysis, anomaly detection, network analysis, log analysis, Distributed Denial of Service (DDOS) detection, among others to allow real time monitoring. Therefore, MMD (Mining Massive Data) or Big Data are part of the solution to tackle the current smart malware.

In the literature, existing malware detection (MD) techniques for smart devices comprise two opposite approaches, namely, static analysis and dynamic analysis (DA). This classification depends on how the code of the application is analyzed. The static case or code analysis means the attempt to identify malicious features, harmful strings or blocks of code, or suspicions code segments without executing Apps. However, the static methods suffer from the inherent constraints of static code analysis (e.g., this approach cannot detect zero-day vulnerabilities, obfuscation and evasion techniques). However, many attacks could happen after an App is installed

†This author has double affiliation. ICT Research Program. UNI, National University of Engineering. PO Box 5595. Managua, Nicaragua. Email: oscar.somarriba@fec.uni.edu.ni.

or executed. So, the latter case (DA) focuses on collecting information about the behavior of the App under analysis during its runtime execution. This type of detection method is also effective in discovering obfuscation and encrypted versions of malware, which is very useful in practical use. Hence, we focus on the dynamic detection of malware.

Of course, the first task in detecting network attacks and instructions is to collect security-related data. This paper proposes and evaluate one method of capturing network packets from Apps with emphasis on the DNS queries, in particular the URL (Uniform Resource Locator) requests, done by the smart devices to remote machine, this is crucial in order to monitoring the Android malware. Since most of the Android based malware communicate with some remote servers (command and control center) either for getting instructions as it does in a botnet or to send data/information stolen from device to the attacker. Here we present a malware detection method for Android using network traffic analysis, some kind of Android packet sniffer. As it stated in [4]: "A packet analyzer (also known as a packet sniffer) is a computer program or piece of computer hardware that can intercept and log traffic that passes over a digital network or part of a network". To the best of our knowledge, this first non-rooted "sniffer" publicly available for the Android ecosystem with capabilities to capture the URLs invoked by Apps in an easy manner. Furthermore, this method could be combined with machine learning techniques in order to design a network behavior analysis engine which able to monitor network patterns from packets and captures malevolent traffic at the network level, see for instance the work by Bae et al. [5], dissimilar to our approach they rooted the smartphones in theirs study.

In our experiments, we use the MalGenome dataset [6] in order to generate network traffic so that to evaluate malware behavior. Besides, most of the malware we examine use DNS in order to obtain as well the URLs of their command and control servers. Then, the problem of determining the DNS queries done by the malware through devices without modifying the firmware or rooting smartphone, is very important and it poses a big challenge. From traces we generated from Apps under test, we can extract malicious URLs invoked by the malware.

A. Problem definition

The problem to solve is how to capture network traffic from the smartphone without rooting or jail-breaking the phone. In our case, we are interested in identifying the URLs invoked through the DNS queries required by the Android malware, without altering the firmware or rooting the intelligent device. Here, we focus on Dynamic DNS request monitoring of Android Applications by capturing network traces from the smartphones.

So, at present, there is the challenge of developing a platform with scalable and efficient capability for the monitoring and analysis of security events that can compromise mobile devices or pose threats that affect the infrastructure operator. The management of the detection and reaction to new threats and their mass spread due to the ubiquity of the underlying communications network, will be done by extending security event management capabilities for the monitoring, detection, characterization, and mitigation of threats to mobile devices, as well as creating an early warning system for operators. In particular, this research will address part of this challenge, showing how to obtain the traces of network traffic from targeted App(s) at the mobile device under the restrictions mentioned above.

The main contribution of this paper, is the method description and the proposed malware detection tool (Android sniffer) for Dynamic DNS request Monitoring of Android Applications via networking without rooting or modified the phone under trial.

The rest of the paper is organized as follows: Section II discusses the related works. Section III presents the description of the method proposed here based on a VPN-solution to capture the network packets, in particular the DNS queries requested by the Apps to remote machines. Section IV evaluates the performance of the proposed method. Later, in Section V, we discuss some open issues. Finally, Section VI presents the remarking conclusions of this work.

II. Related Work

In this section, we introduce some previous works in this topic. The analysis and detection of Android malware has been a hot theme of research in the last years. Several concepts and techniques have been proposed to counter the growing amount and sophistication of this malware.

There are several ways to dynamically intercepting/obtaining the network packets transmitted by the mobile smart devices to remote servers, among others, namely: i) By using a proxy [7], ii) By utilizing an Android network log monitor [5], or packet analyzers (sniffer) such as *tPacketCapture Pro* [8] & *MalDetec* [9] based on virtual Private Networks (VPN) approaches, iii) By modification and customization of the Android OS as in [10], iv) By hooking function calls, such as library APIs, (e.g., OpenConnection method), and v) By exploiting the Logcat tool from Android OS such as the tool dubbed *Logdog*, where Logcat is the command to view and filter information from the Android logging system [11].

In [10], Rughani customized source code of Android OS available at AOSP (https://source.android.com/). The customization includes modifying code in needed files and rebuilding the code to make custom OS. Afterwards, the author utilizes a python script that captures and intercepts logs. It then extracts IP Address/URLs from the logs and puts them in a file. After extracting information, it compares the extracted information with existing blacklisted IP Address (which are downloaded from openbl.org automatically by the script). As the last step, the script creates result file containing suspicious IP Addresses (if any found). Collected information is not restricted. Unlike the work in [10], with the "approach" presented in this paper, we can obtain the URLs consulted by the smartphone, without modifying the firmware of the

Android OS and making use of the DA provided by our non-rooted Android sniffer.

In the approach by Bae et al. [5], they minimize the use of high overhead functions & replace them to lightweigt features (e.g., function call monitoring). They have leveraged those features instead of using high overhead operations. First, the monitor the network connections including the DNS queries requested by using an Android network log monitor. It is probably that malevolent Apps are trying to connect to some mistrusted remote servers, for example C&C machines with bad reputation. As a good rule of thumb for malware detection, we can watch whom, an App connects to, we can infer its malevolent behavior. Our approach is difference from the authors in [5], in that they need to root the smartphones while we are able to utilize our Android sniffer based on VPN-solution without root privileges as it is described in Section III.

Alternatively, in general, method hooking is a technique used to intercept the call of a certain method at runtime to change the behavior of the calling application. By dynamically (it is used for mechanisms that can dynamically apply a hook at runtime) intercepting function calls frameworks can analyze both single calls and sequences of calls to reconstruct behaviors for semantic representations, or monitor the function calls for misuse. Function hooks can also be used to trigger additional analyses. For instance, if a function was hooked and triggered, parameter analysis could then be applied to retrieve the parameter values of when the function was invoked. The analysis framework *InDroid* [12] inserted function call stubs at the start of each opcodes interpretation code in order to monitor bytecode execution and analyze Android behaviors. While it does require modifications to the Dalvik VM and it does not work on Android 5.0 (e.g., with ART), the method requires relatively light modifications and has been used on Android versions 4.0-4.2. However, *InDroid* requires to root the smartphone, which it is not necessary in our case. Dynamic hooking happens in volatile memory only. Furthermore, in [13], Brandolini designed and implemented a security library for Android applications exploiting the hooking of Java and native functions to enable runtime analysis. The library verifies if the application shows compliance to some of the most important security protocols and it tries to detect unwanted activities based on the Dalvik compiler. Testing of the library shows that it successfully intercepts the targeted functions, thus allowing to block the application malicious behaviour. He also assesses the feasibility of an automatic tool that uses reverse engineering to decompile the Android application, inject his library and recompile the security-enhanced App. A similar method hooking for Dynamic DNS request monitoring has been addressed in [14], with the Dalvik compiler.

On the other hand, of course, we could try to install a limited version of Wireshark [15] for Android, which allows us to identify the URLs invoked by means of the DNS queries from the phone, but this would imply ”rooting” the smart device; which is not acceptable for practical purposes in our case, since mobile operators could not maintain the guarantees of their users and also taking into account the fact that most of the Android users do not jailbreak their smartphones [9]. A very similar case to Wireshark is the tool*tcpdump* [16] because it also requires rooting the smart device. We have also done tests with *tPacketCapture Pro* [8] that allows the capture of Internet traffic back and forth on the smartphone without rooting privileges, but it has severed restrictions to show the DNS queries made by the Android phone. Also *tcpdump* requires ”rooting” the smart mobile device. Another possibility found in the literature survey is *Logdog* [11], which has been developed to detect botnets for smartphones using log analysis techniques. Again, *Logdog* is based on a collection of Android logs called *Logcat* and it is necessary to get the permission of superuser (root user) of the OS, to be able to operate with this App on the smart device. Which is out of the scope or restrictions required in this work.

Moreover, as aforementioned Android 5.0 introduced the new ahead- of-time compiling Android runtime termed ART. So, it is needed an advanced instrumentation approaches based on the new virtual machine ART which are addressed in [17]–[20].

In the forthcoming section, among the above approaches for sake of simplicity, we choose as used method for Dynamic DNS request monitoring of Apps the VPN-solution [8], [21] targeting Android versions using ART.

III. Implementation of the Dynamic DNS Request Monitoring method via networking

Although there are proposals that aim to control network traffic in smart devices [21], [22], current smartphones do not usually give importance to network requests and what they could represent to them in terms of mobile security. Also, ordinary users must resort to sophisticated systems and maintenance techniques to ensure that their smart devices are not infected with some malware. Therefore, it is of great value and importance to develop an application that can track and monitor network traffic in a more automated way and that performs the majority of actions necessary to give the state of reliability of the equipment to the user. In this way we can benefit both common users and some more advanced users who need specific administrative tools in their smart devices, some as developers or students of networks can make use of the application to help them understand how packets travel through the network .

So, the main goal here is develop an App that manages to keep track of network packets or network traces in order to ensure the degree of reliability of the applications installed in the smart device and avoid or warn possible malware problems associated with communication with malicious servers in the network. We have developed an App that can track the access to the network that other Apps can make, whether the user has proof of these accesses or not, capture DNS requests on the smartphone to a remote online server and create a check that defines to the user whether or not there is some unwanted access by the installed applications.

Let us describe the basics of the proposed method, where we develop a simple sniffer based on the Android Studio IDE version 2.3. Our application stars gathering the network traffic of the targeted App or Apps at the smartphone, through the creation of a local virtual private network (VPN), See Figures 1 and 2. We then need to use *VpnService* to redirect all the device's network traffic through our application. Of course, we can only capture DNS traffic from the App or Apps under test. Afterwards, the network traffic captured and all requests that exist within the device and are thus packaged it into PCAP files (this is an interface of a programming application for packet capture). Thereafter, the *jNetPcap* library is utilized; which is an open source Java library, used in order to capture and decode network packets. And, it uses native implementations to provide optimum packet decoding performance. In Figure 1, we have a diagram showing how the *VpnService* is called from *Main Activity* of the Android programming and its services that this utilized.

By doing so, the obtained IP traffic or URLs in the network traces can be checked through a blacklist server, and know if there is any malicious behavior on the device. Consequently, it is possible to define/declare which application under test is not suitable to use or uninstall. Note that *Network Sentinel* runs in its entirety on the local smart device and traffic is not routed through a remote VPN server. This is the basics of the proposed malware detection tool for Dynamic DNS request monitoring that works with the ART compiler. And, we named as to *Network Sentinel* which needs at least the Android version number 5.0 (code name Lollipop) to work properly.

Of course, VPN approaches [8], [21] have suggested in the literature to provide secured communications in Android ecosystems. For instance, *PrivacyGuard* [21], an open-source VPN-based platform for intercepting the network traffic of Apps. This Android application also requires neither root permissions nor any knowledge about VPN technology from its users. Thus, we implemented *Network Sentinel* on the Android platform by also taking advantage of the *VpnService* class provided by the Android SDK. *PrivacyGuard* is an App that alerts you when one of our Apps leaks sensitive information to a remote server, which has a little bit different of our scope of the proposed method presented here. Furthermore, we tested the paid App *tPacketCapture Pro* [8], but we did not succeed to get or capture any URLs at all. So, this was one of the reasons to develop *Network Sentinel* in order to fulfill this gap in the arsenal of the tools publically available for the Android analysis, in particular for Dynamic DNS request monitoring of Android malware. In Figure 3, it can be seen the icon on the *Network Sentinel* as it depicted in the platform of Google play. This Android "sniffer" is available through Google Play Store, at the web link [1].

IV. Experimental Results

In order to evaluate our approach (Android sniffer), we conduct several experiments with the two smartphones, Xiaomi

[1]https://play.google.com/store/search?q=Network%20Sentinel&c=appstext

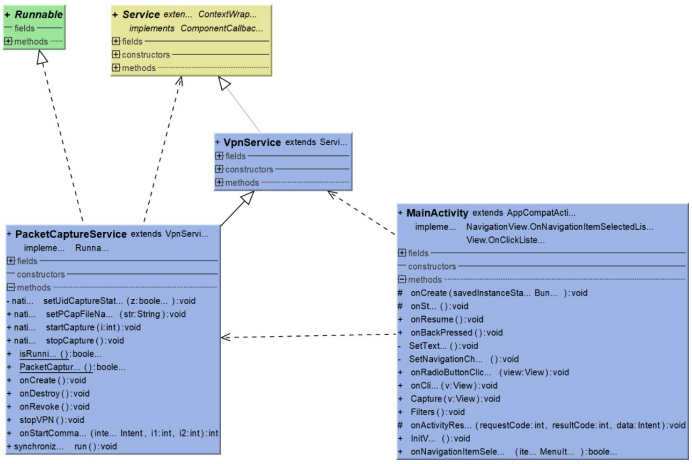

Fig. 1: The diagram of use and implementation of the capture service of the VPN at the smart device.

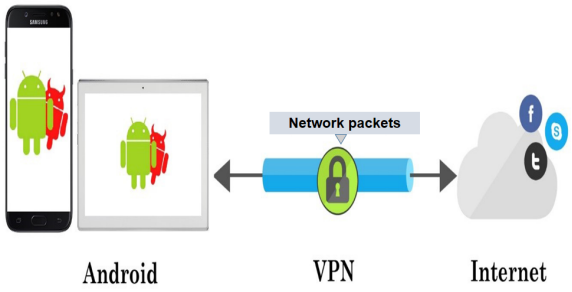

Fig. 2: Schema handling of packets via VPN (Virtual Private Network) from Android smartphone or Tablet. The main configuration of our Android sniffer is shown, capturing packets through a protected data tunnel.

Redmi 3S Prime and Samsung Galaxy Grand Prime. First of all, we validate the network traces (benign Apps and malware) provided by our tool. Second, we carry out interceptions of URLs by applying Dynamic DNS request monitoring of the App under test combined with existing blacklisted IP address of malicious Domain Name available in Internet, to detect the presence of malware. And third, we address the consumption of resources of the Android "sniffer", mainly through the power consumption of the device regarding the battery usage.

A. The capture of the DNS network packets

We first present the main menu of the *Network Sentinel* as can be shown in Figure 4. Here we star by choosing the configuration of the dynamic monitoring, e.g. we can select the App under examination (we can also monitor several Apps, if we want to do that) and the type of capture of the network traces. So far, the menu of *Network Sentinel* is available

Fig. 3: The Android sniffer termed *Network Sentinel* is available at Google Play Store.

Fig. 4: The main menu of *Network Sentinel* available at Google Play Store.

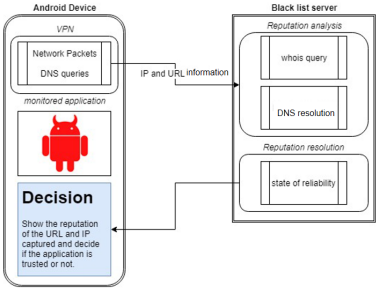

Fig. 5: Functional diagram of the developed App. Configuration of the structure of the App and connection to the blacklisted service.

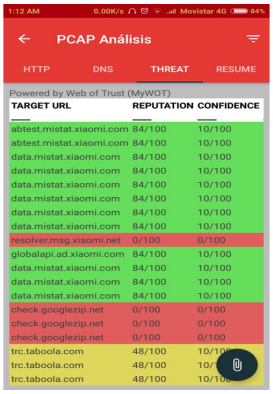

Fig. 6: Outcome of the queries to the blacklisted service WoT.

in Spanish language, however future versions are going to migrated to a full version in English language. For instance, in this case, "**INICIAR CAPTURA**" translated into English is "Star Capture", and "**SELECCIONAR APLICACIONES**" means in English "Choose the App or Apps to monitor". It is important to mentioned that the *Network Sentinel* provides information about the protocols in use in real-time, not the raw data. After obtaining the network traces, the results are being automatically saved in a file on the smartphone (Android sdcard) with an extension PCAP. We validate *Network Sentinel* by comparison its results against similar monitoring using the network protocol Wireshark [15] in an ad hoc set up with the smartphone Samsung Grand Prime rooted. We tested 10 benign Apps (they were taken from Google Play Store) and 10 malicious software.

B. The Maliciousness of the Android application (App)

Figure 5 depicted the functional block diagram employed the feature of inspecting the threat of the App under examination in *Network Sentinel*, utilizing blacklisted service provided by Internet. We can further extend the malware analysis selecting the option "Usar Servidor DNS" in the configuration menu of *Network Sentinel*, here we can either

In Figure 6, it is shown the monitoring results using the WoT, we reach this feature in *Network Sentinel* by using the *PCAP Analisis*, in particular with the option *THREAT*. Usually, a web site in red color could imply a suspicious or malicious URL.

C. Battery Usage with Network Sentinel working

Application performance is always a trouble for the everyone developing Apps. Here we review the battery usage of the *Network Sentinel* by utilizing the tools *Batterystats* and *Battery Historian*. *Batterystats* takes data from our smart device bout battery. On the other hand, *Battery Historian* converts this data to HTML format to be able to see it on Browser. *Batterystats* is a part of Android framework and *Battery Historian* is on Github as opensource at https://github.com/google/battery-historian.

The step by step procedure how we use *Batterystats* is described at the web site: https://medium.com/@elifbon/android-application-performance-step-4-battery-b1f88d096b1e.

During the experiments using *Batterystats* & *Battery Historian* we "play" with the App under test for about 15-20 minutes. The typical outcome of the battery usage during the

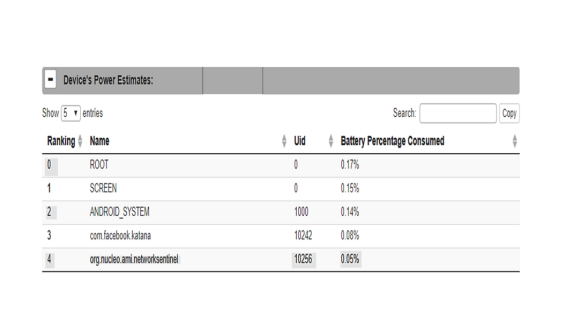

Fig. 7: Table of the battery usage with the smart device Xiaomi Redmi 3S, in particular the **org.nucleo.ami.networksentinel** (Ranking 4 in the Table has as **Battery Percentage Consumed**, 0.05%).

trials was roughly an average of 0.05%, as it is shown in the Table below 7, an excerpt of the *Battery Historian* tool.

V. OPEN RESEARCH TOPICS

According to the above mentioned, we envisage a number of open issues in this research field, i.e., Dynamic DNS request monitoring of Android malware. First of all, we would like to compare two methods that are interesting to this research, namely: i) method1 (Dynamic DNS request monitoring of Android malware via networking) proposed here as *Network Sentinel*, and ii) method2 (Dynamic DNS request monitoring of Android malware via instrumentation or hooking) as the one in [14]. Second, the proposed method2 aforementioned utilizes as compiler Dalvik in contract with presented method1, which it has as compiler ART. So, we need to update the method2 in [14] by using the Android runtime instrumentation toolkit ARTIST proposed in [19]. Third, CPU utilization and battery usage shall be the most common critical issues regarding the application performance in our future study cases.

VI. CONCLUSION

This paper gave an introduction to a simple Android "sniffer" designed and implemented on a VPN-approach inside the smart device as a method for Dynamic DNS request monitoring of Apps via networking. It is has been written in java programming language, under the umbrella of the Android Studio IDE. We dubbed this method as *Network Sentinel*. We conduct several experiments with *Network Sentinel* with real smartphones without jailbreaking them. The results of intercepting or capturing the URLs from DNS queries requested from the smartphones to remote servers have been validated. For users of the *Network Sentinel*, the graphical interface provides an easy and intuitive manner of using it, at very low battery usage. This tool is publically available at the Android distribution platform, Google Play Store, thus the research community can take advantage of it and give us further feedback about its performance.

REFERENCES

[1] J. K. Lee, "Research framework for ais grand vision of the bright ict initiative," *MIS Quarterly*, vol. 39, no. 2, 2015.

[2] O. Somarriba, U. Zurutuza, R. Uribeetxeberria, L. Delosières, and S. Nadjm-Tehrani, "Detection and visualization of android malware behavior," *Journal of Electrical and Computer Engineering*, vol. 2016, 2016.

[3] E. Markatos and D. Balzarotti, "The red book. the syssec roadmap for systems security research, the syssec consortium, 2013."

[4] U. Banerjee, A. Vashishtha, and M. Saxena, "Evaluation of the capabilities of wireshark as a tool for intrusion detection," *International Journal of computer applications*, vol. 6, no. 7, 2010.

[5] C. Bae and S. Shin, "A collaborative approach on host and network level android malware detection," *Security and Communication Networks*, vol. 9, no. 18, pp. 5639–5650, 2016.

[6] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 95–109.

[7] S. Garg, S. K. Peddoju, and A. K. Sarje, "Network-based detection of android malicious apps," *International Journal of Information Security*, pp. 1–16, 2016.

[8] Taosoftware, "tpacketcapture is the software that can capture communication packets on non-rooted device," *Online: http://www.taosoftware.co.jp/en/android/packetcapture/*, 2015.

[9] N. Trivedi and M. L. Das, "Maldetec: A non-root approach for dynamic malware detection in android," in *International Conference on Information Systems Security*. Springer, 2017, pp. 231–240.

[10] P. H. Rughani, "Detecting blacklisted ip access from android phone," *Indian Journal of Science and Technology*, vol. 9, no. 48, 2016.

[11] D. A. Girei, M. A. Shah, and M. B. Shahid, "An enhanced botnet detection technique for mobile devices using log analysis," in *Automation and Computing (ICAC), 2016 22nd International Conference on*. IEEE, 2016, pp. 450–455.

[12] J. Li, W. Yang, J. Shu, Y. Zhang, and D. Gu, "Indroid: An automated online analysis framework for android applications," *Crisis Intervention Team (CIT)*, 2014.

[13] F. A. Brandolini, "Hooking java methods and native functions to enhance android applications security," Ph.D. dissertation.

[14] O. Somarriba, "Detecting blacklisted urls from unmodified and non-rooted android devices," in *Central America and Panama Convention (CONCAPAN XXXVll), 2017 IEEE 37th*. IEEE, 2017, pp. 1–6.

[15] Wireshark, "Wireshark: A network protocol analyzer for unix and windows."

[16] V. Jacobson, C. Leres, and S. McCanne, "Tcpdump public repository," *Web page at http://www. tcpdump. org*, 2003.

[17] M. Wißfeld, "Arthook: Callee-side method hook injection on the new android runtime art," Ph.D. dissertation, Saarland University, 2015.

[18] M. Backes, S. Bugiel, O. Schranz, P. von Styp-Rekowsky, and S. Weisgerber, "Artist: The android runtime instrumentation and security toolkit," in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 2017, pp. 481–495.

[19] L. Dresel, M. Protsenko, and T. Müller, "Artist: the android runtime instrumentation toolkit," in *Availability, Reliability and Security (ARES), 2016 11th International Conference on*. IEEE, 2016, pp. 107–116.

[20] V. Costamagna and C. Zheng, "Artdroid: A virtual-method hooking framework on android art runtime." in *IMPS@ ESSoS*, 2016, pp. 20–28.

[21] Y. Song and U. Hengartner, "Privacyguard: A vpn-based platform to detect information leakage on android devices," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2015, pp. 15–26.

[22] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, 2014.