This is an Accepted Manuscript version of the following article, accepted for publication in:

O. Somarriba and U. Zurutuza, "A collaborative framework for android malware detection using DNS & dynamic analysis," 2017 IEEE 37th Central America and Panama Convention (CONCAPAN XXXVII), 2017, pp. 1-6.

DOI: https://doi.org/10.1109/CONCAPAN.2017.8278529

A Collaborative Framework for Android Malware Detection using DNS & Dynamic Analysis

Oscar Somarriba[†]
ICT Research Program
National University of Engineering (UNI).
Managua, Nicaragua.
oscar.somarriba@gmail.com.ni

Urko Zurutuza
Electronics and Computing Department
Mondragon University
Mondragon, Spain.
uzurutuza@mondragon.edu

***Abstract*—Nowadays, with the predominance of smart devices such as smartphones, mobile malware attacks have increasingly proliferated. There is an urgent need of detecting potential malicious behaviors so as to hinder them. Furthermore, Android malware is one of the major security issues and fast growing threats facing the Internet in the mobile arena. At the same time, DNS (Domain Name System) is widely misused by miscreants in order to provide Internet connection within malicious networks. Here, we propose an infrastructure for monitoring the Android applications in a platform-independent manner, introducing hooks in order to trace restricted API calls used at runtime of the application. These traces are collected at a central server were the application behavior filtering, string matching, and visualization takes place. From these traces we can extract malicious URLs and correlate them with DNS service network traffic, enabling us to find presence of malware running at the network level.**

***Index Terms*—Android malware, API calls, Dynamic behavior analysis, DNS queries, and Collaborative framework.**

I. INTRODUCTION

Android has become one of the dominant mobile operating systems. Thus, smart devices running Android Operating System (OS) represent an overwhelming majority of smartphones. In general, it is also the first targeted platform greatly impaired by malware writers pursuing to take the control, with well over a million of Android applications (apps), over the world. Due to the popularity of Android smartphones, its apps' security is a serious issue concerning more than 80% of smartphones users and the malware targeting smartphones has rocketed over the last few years [1]. The security issues of Android platform have recently drawn the attention of attackers seeking to exploit its vulnerabilities, and these concerns are being tackled both in academia and industry. The current security applications also scan apps for malware detection according to a malware signature database. This approach requires frequent network communication and support of up-to-date malware database, while almost certainly missing new malware or zero-day attack (which is an undisclosed computer-software vulnerability that hackers can exploit) samples. Furthermore, Android OS puts some limitations on apps for security reasons, in order to avoid easily gathering traces without reshaping the firmware or rooting the smartphone. Even so, we need to collect traces from apps on a large scale, herein this will be done under two constraints aforementioned above. Without these restrictions the warranty of the smart device may be invalidated or the Android phone will be further exposed to malware attacks. Furthermore, this paper extends the infrastructure proposed in [2], collecting the application traces using the hooks, and uploading the traces to a remote server for observation and analysis.

Many security mechanisms were proposed to detect mobile malware and protect targets from attacks. In general, most of these mechanisms are based on analyzing app elements such as permissions, the used application programming interface (API) function calls, the employed system calls, or its bytecode. Such mechanisms employ various detection techniques such as static dissection, dynamic analysis (DA), and cloud based analysis. In the static analysis, there is attempt to identify the malicious code by decompiling/disassembling the app and searching for suspicious strings or block of code. The DA implies the execution of the app performed through instrumenting or virtual machine monitoring to observe its behavior. In the cloud based approach, the app will be executed and dissected on a remote server.

Mobile devices have become major targets for smart malware due to their heavy network activity, including the Internet access. So, Domain Name System (DNS) is one of the key elements of the Internet that facilitates associating a domain name and hosting IP address. Besides, the DNS scheme is a query/reply based protocol where the authenticity of the response is not confirmed or confirmed by approaches that can be thwarted easily. However, in addition to the crucial role in functioning of the Internet, DNS is extensively misused by malware developers. Thus, the aggressors rely on DNS to provide adjustable and resilient communication between compromised client machines and malicious infrastructure. However, it is worth noting that we do not address or detect malicious DNS in this work, which is DNS traffic corrupted for illicit and malevolent reasons. In fact, we only take

[†]This author has double affiliation. Electronics and Computing Department, Mondragon University, 20500, Mondragon, Spain. oscar.somarriba@alumni.mondragon.edu

advantage of DNS to find malware without having to monitor all smartphones in a system.

This work focuses on monitoring Android applications' suspicious behavior at runtime, in particular adding to the app traces (described in [2]) the DNS queries to the remote servers done by the smart device. Later, we correlate these enhanced app traces with DNS traces taken from network traffic of the mobile infrastructure. Thus, we propose a platform-independent behavior monitoring infrastructure composed of five elements with the capacity of instrumenting DNS methods based on the hooking library named Android Dynamic Binary Instrumentation (ADBI) [3]: (i) an Android application that guides the user in selecting, instrumenting and monitoring of the application to be examined, (ii) an embedded client that is inserted in each application to be monitored, (iii) a cloud service that collects the app traces, (iv) a DNS service that provides data logs of the network-service traffic, (v) and finally an Elasticsearch [4] cluster including a visualization component that can generate dashboards of the top-ranking classification of malicious URLs based on *Kibana* (part of the Elastic Stack, it is an analytics and visualization platform that builds on Elasticsearch (*ES*) to give us a better understanding of the data). In addition, the DNS network-service send flow data (DNS logs) to the tool dubbed *Logstash* (an agent and server-side data pipeline processing that receives it, parses it and later sends the indexes into *ES*). An Overview of the monitoring system is shown in Fig.1. See further details about our infrastructure in Section III.

A. Problem Statement

This work aims to deal with the sophisticated and emerging threat of the Android malware in mobile ecosystems. We develop techniques to systematically explore and monitor the app traces generated from the execution of instrumented apps, thereafter they are sent to the cloud service in order to support the malware detection. Much of the research work surrounding mobile malware has been centered on either the in-depth analysis of malicious apps (host level) or the network-based approach (network level). Thus, developing a collaborative framework between the aforementioned approaches seem to be the next step, and it can increase the chance of malware detection. The main goal of this research is to explore, design and develop techniques that can be used to detect malicious mobile behavior from massive sets of heterogeneous sources. In particular, the DNS traffic activity produced by mobile malware will be inspected and correlated with device-related activity.

B. Contribution and Outline

The contributions of this paper are (1) extending the system for Android platforms described in [2] composed of an implementation on the smartphone side and on the remote server side. Herein, we focused on capturing requested URL for detecting malicious transactions initiated by an app running

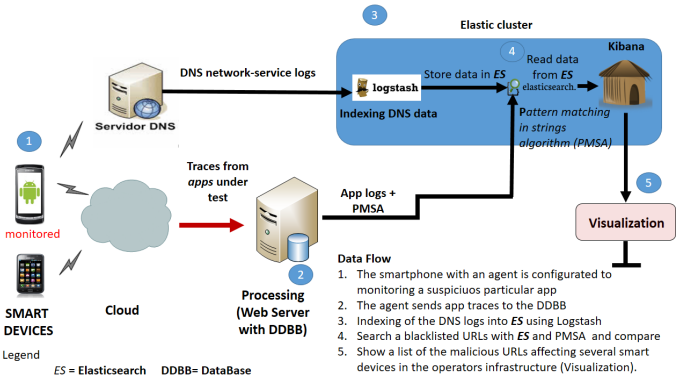

Fig. 1: Proposed approach.

on the Android phone; and (2) evaluation of combining and correlate the following two approaches: top-down detection by identifying malware domains using DNS network traffic and bottom-top detection using the classical DA on a number of apps to pinpoint the malware. The remainder of the paper is structured as follows: Section II discussed the related work. Section III reviews the research methodology. Section IV evaluates the performance of the monitoring system. Limitations and Conclusions are presented in Section V and Section VI, respectively

II. RELATED WORK

Since most of the Android malware resort to communicate with some remote server (e.g., a botnet master machine), there is the crucial need to detect fraudulent or malevolent operation with help of a collaborative malware analysis framework between the smart device and the network traffic involved. Besides, usually the malware analysis comprises the process of studying code and to obtain information about the behavior and functionality of the malware in its environment. Afterward, the results of the analysis will be used as an input to the Malware Detection (MD). The type of analysis for identifying malicious applications in Android platform can be classified as follows, namely: Host-based Analysis and Network-based Analysis. The so-called smart malware in current smartphones and tablets have mushroomed over the last few years, which is supported by sophisticated techniques intentionally designed to master security architectures in use by such devices. Let us review some of these approaches:

A. Host-based Android MD

Lu et al. in [5] describe a static tool that aim to detect Inter-Component Communication vulnerabilities (ICC) by analyzing the app bytecode. Another static approaches such as *Drebin* [6], and *DroidMat* [7], use learning-based or machine-learning (ML) systems to detect anomalies by considering permissions, APIs and bytecode instructions. For instance, DroidMat, is a tool that extracts its information from static feature-based mechanism as the bytecode. Next, applies k-means and expectation-maximization (EM) clustering algorithms to classify the app as malware or benign. It

detects Android malware using feature-based mechanism for characterizing the app through static information. Since [5]-[7] utilize static methods, they undergo from the inherent limitations of static analysis, e.g., they are unable to detect obfuscation techniques. Also, the network behavior-based malware detection for Android malware in smart devices got little attention with these approaches. Regarding the DA, we first consider virtual monitoring machines that analyze the systems calls to detect malicious behavior such as *Crowdroid* [8] and *CopperDroid* [9]. Alternatively, we then consider DA that can be carried out by instrumenting the Android OS, such as: [2] and *TaintDroid* [10], among others. Unlike [8]- [10], our framework targets at detecting attacks both at the host and network levels, and we do not need to root or to change the firmware the smart device, since we introduce hooks to trace API calls at the application level of the Android stack, so it is not necessary to invoke system calls at the kernel level that require rooting the smart device.

B. Network-based Android MD

Other approaches explicitly analyze network traffic for different goals. In [11], they address the network-based malware detection mechanisms for Android-based attacks, and they use *MalGenome* [12] dataset in their research. So, the authors used four different traffic categories (network traces), namely based on: DNS-based features, HTTP-based features, Origin-destination based features and TCP-based features. This trait analysis is used to train a detection app model for classification of apps based on ML algorithms. Furthermore, in *CREDROID* [13], it has been proposed an Android malware detection by network traffic analysis capturing packets in a remote server using the protocol analyzer WireShark [14]. They also introduced the reputation score of the URL. With all of this, the authors proposed a method which identifies malicious apps on the basis of their DNS queries and APK score computation through Virustotal [15], as well as the data it transmits to remote server by performing the in-depth analysis of network traffic logs in offline mode. Unlike [11] and [13], we profile only gathering the network traffic of app under test at the smartphone side. Regarding the host-based analysis, even though Android smartphones have attracted the users community for their feature rich apps to use with various applications like chatting, browsing, mailing, etc.; including a heavy activity use in the Internet. Most of the published approaches do not integrate the network traffic dimension at the device side into the analysis. Unlike the component-application analysis, we include the DNS traffic in our approach. So, to attain the goal of detecting the malware, we propose a dynamic inspection combining with DNS logs at the Android phone (app traces) and the network operator infrastructure levels.

Our proposed infrastructure is related to some of the research work mentioned above and employs similar traits for identifying malicious applications, such as API calls, DNS queries, and DA. However, our approach is different from aforementioned approaches in the following aspects. Firstly, we have a runtime malware detection (dynamic analysis) but abstain from reshaping the firmware or rooting the smart device as it is done by [8]- [10]. Also, the DNS queries of apps under test are captured at the smart device, but not on a remote server using Wireshark as it is done in [11] and [13]. Secondly, we combine in a collaborative or integrated environment the bottom-top analysis (host level) with top-down approach (DNS network service) in an easy-to-follow manner in the cloud service and *ES* cluster. Thirdly, moreover, we are able to monitor in real time not just the DSN queries for a particular app to be monitored, but we are able to focus on intercepting malicious URLs at DSN network service affecting others smart devices. Our platform is more dynamic and collaborative than other approaches mentioned above.

Our results are shown in a dashboard that visually render existing malicious URLs in the system enabling to warning a potential mobile operator about their presence in its traffic network. Noting that a mobile operator can easily or indirectly detect another infected devices that had not installed the monitoring application. This is due to they behave in the same manner, by doing the same DNS queries than the monitored devices. This is certainly a very valuable benefit, because we do not need to monitor all the smart devices at the same time. Since we collect the used URLs on the Android device instead of on a remote server or gateway, we shorten the time to detect malware as it is suggested in the hybrid analysis method dubbed *NeseDroid* [16].

III. RESEARCH METHODOLOGY

Hence the whole process is divided into four phases. First of all, the first phase implies the generation of the app traces, data collection and the analysis and monitoring of network traffic. Actually, the app traces are, in this particular case, the malware traces with the plus in this approach that we are able to capture the DNS queries done by the app under test, if any. The second phase includes the log aggregation and transport of data generated, the extraction of URLs from app traces and DNS network-service traces are done with the help of Python-language scripts. The third phase is the search and analytics task. And, the fourth phase is the visualization component of the system. The rest of the process is outlined in subsections that follows.

A. Introduction to the Malware Dataset

First let us to introduce the employed Malware Dataset. In our experiments we are using the MalGenome dataset [12], which it has 1260 Android application package (apks). So according to our approach we need to ”capture” the network traffic from 100 malware samples of these applications (apk files). It should be mentioned that there are malicious apps that are not generating network traffic, therefore these cases are not taken into account or ignore in our analysis.

B. Data Generation

Here we are interested in the data sources that will feed into our platform, namely: the app traces from smart devices conveying information about the DNS queries done by one app under test; and the traces of the DNS network-service in the mobile infrastructure, in particular the DNS logs from its DNS servers. To achieve our objective, we utilize the Android OS Version 4.3. In order to collect data at the smart device level, we need to setup an experimental testbed with multiple virtual machines (VM) which used VMWare WorkStation 12 and VirtualBox 5.0.28, respectively; to create a controlled environments. Afterwards, we use the Eclipse emulator on a host a machine which is employed for running the Android application; and we then run various tools to process the malicious apps under analysis, in particular in the Virtualbox VM we run the Elastic Stack [4]. Let us introduce the first data source, the processing of the traffic generated by the smartphones in the VMWare VM. Thereafter, we store some privacy data (e.g., contacts information, images, and some downloaded files) to the emulator, the next step performed is to capture the DNS queries of the samples from each MalGenome family in use. Samples from each of the malware family were executed on emulation environment for a short and fixed amount of time (10 mins). We expect some of samples to communicate to the remote server, since each sample itself is a malware. To separate network traffics of the smart devices, virtual machine is left idle for around 5 minutes between running and terminating of applications after the network traces from the app under examination are captured and saved. After the traces have been collected from an application, it is uninstalled and Eclipse emulator is rebooted. It is important to mention that the aforementioned procedure, it is just to check out which malware samples are connecting with remote servers. Regarding the second data source, for sake of simplicity, in this work we only consider two samples of DNS network-service logs (each file has around 30 MB in size, which are currently available for our experiments) provide by one mobile operator in 2015 and 2016. DNS records contain valuable information about domain names and associated IPs that clients query, whether mobile or not. It will therefore contain malicious URL records that could make mobile Apps malware.

C. Analysis and Monitoring of Mobile Traffic

The objective of this task will be to review and evaluate relevant sources of information security in the mobile network and to show how they could be used to detect subscriber security events in the network. It is important noting that we run Python scripts at the VMs and QPython scripts at the Emulator or the smartphone. First of all, we can query the app traces database using a Python script written to extract the involved URLs. So here we focus on detecting malicious URLs. And we then rely upon various third party APIs. With the help of these APIs such as URLVoid [17] and/or

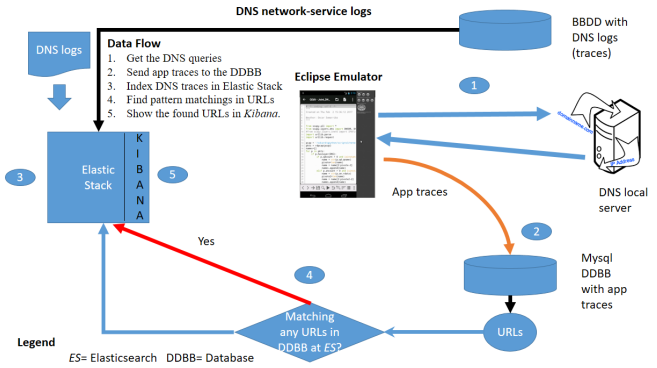

Fig. 2: Correlation of the used URLs from app traces with DNS logs.

Web of Trust [18]. These are free services that analyze a website through multiple blacklist engines or monitors and online reputation tools to facilitate the detection of fraudulent and malevolent websites. In the case of unavailability of the malware blacklist systems aforementioned, alternatively, selected features employed by the work of EXPOSURE [19] as it is suggested in [20] can be used to grade reputation of domains, where they use a host domain reputation analysis engine based on Support Vector Machine (SVM) classifier. For training the model, they collected two kind of sample domains (benign and malicious) from the local DNS server on their campus from July to August in 2013.

D. Log aggregation and transport

This task will focus on the description of the agent responsible for collecting and storing information on DNS network-service logs. For data collection, the data will be used in text format from the DNS Servers, and data from the instrumentation collected in a well-structured relational database of the apps. The collection of the logs of the DNS network service will be indexed through *Logstash* [4]. This tool, designed to collect and add events and logs created on multiple devices and services, sends information from DNS network-service logs to a system that indexes content for durable storage. Here, the main idea is to be able to have a telemetry correlator that will provide the analysis and correlation of all device telemetry data with all the sources that will be available to the system. Thus, on the other hand, when malware is executed, malicious URL queries will be logged, and sent to a SQL database (DDBB) to the server. For further details, see Figures 1 and 2.

E. Search and Analytics

In this task the correlation of the monitored events in the previous task has to be carried out. Here the utilized strategy is to index the content of data collection in order to develop a pattern matching system, in particular we do that with the DNS network-service logs that are expected to be huge amount of data. Thereafter, we proceed with the extraction of the malicious URLs include in the app traces of a particular sample of malware in use. So, we are able to identify another

smart devices who are running the same malicious URLs previously executed. Next, the following step is conducting a search of malicious URLs on the aforementioned content indexing built in with *ES*. By the way, here *ES* has been used as a method of content indexing. *ES* is a Lucene-based search server [4]. Also, our system is designed with an easy-to-use web interface supported by a Python script and *ES* to make it simple to search and perform various forms of analysis on the apps and their traces, as well as the DNS network-service logs from network infrastructure of the mobile operator. In our case, we search for pattern matching in strings for those common malicious URLs found both in the traces of the smart devices and in the DNS network-service records (logs) extracted in the network traffic of the mobile operator, see Fig. 2.

F. Visualization

The visualization of anomalous behavior is the last component of the proposed architecture. In order to performance a visual analysis of the platform. So, we use the Elastic stack, which is a versatile collection of open source software tools that make gathering insights from data easier [4]. Formerly referred to as the ELK stack (in reference to *ES*, *Logstash*, and *Kibana*). In particular, *Kibana* is a browser-based or web-interface visualization frontend for *ES*. It enables users to easily consume data in aggregate that would otherwise be difficult to process; making logs, metrics, and unstructured data searchable and more usable for humans. So because *Kibana* persists most of its data within *ES*, managing *Kibana* dashboards and visualizations is a similar exercise as managing other indexes in *ES*. Charts, graphs, and other visualizations sit atop *ES* APIs which can be easily inspected for closer analysis or use in other systems.

IV. EXPERIMENTAL RESULTS

To evaluate our framework, in this section we show the visualization results of the process of monitoring the malware behavior. Firstly, we assume that we have already instrumented and monitoring and running, one of malware samples that it does DNS queries, and we let it runs for a long time. By the way, from the 100 sample families explored, only 63 of them have connections with at least one remote server. We then proceed to apply the pattern matching in strings by using Python-language scripts developed for this purpose (see Fig. 2). Moreover, two programs written in Python language are used. The first Python script extracts one malicious URL upon the time from the MySQL database with the app traces, and then it connects with *ES* to look up through the whole indexed DNS records within it. For instance, if we search for the particular URL, **s0.2mdn.net**. The Python program obtained automatically this value from malware traces stored in the MySQL database in the service cloud. In Fig. 3, it is shown one of the DNS logs obtained after the processing (indexing) with the tool *Logstash* and stored in *ES*. Also, in this Fig. 3, we can appreciate the several fields that can be utilized to look up for a precise information, in our case we search for the field tagged ”URL”. In Table I, we can see the possibility of finding more malware not only in the smartphone under examination, but in other smart devices that are concurrently using the same malicious URLs and are also being detected in the DSN network-service traffic, so we can do a decisions correlation.

The second Python script allows us to do pattern matching in strings using a well-known algorithm (or KMP algorithm) [21]. This is done in order to compare the *Elasticsearch* processing against the KMP algorithm. In other words, the malicious URLs stored in the MySQL database in the cloud service are also read with a second Python script. Thereafter, we run the KMP algorithm, to conduct a pattern matching in strings, searching directly inside the DNS log in *ES* to look for the URLs (string) under examination. Comparison of the searching time on malicious URLs inside the indexed DNS log in *ES* using the first Python script are faster by approximately five times in average in 10000 trials, versus the searching time of using the second Python script.

V. LIMITATIONS

It is very-well known, there is no prefect detection system without limitations. In this work, we only consider URL/DNS traffic, however current applications and malware as well are using DNS tunneling techniques and HTT traffic so, taking into consideration the current communication landscape this proposal will be only covering partially the current malware. In addition to this, malware using certificate pinning will be totally able to evade our system. Also, working with sandboxes in DA instead of real Android phone could be a drawback since malware can detect the use of emulators as is discussed in [22]. For instance, most of the current mobile malware, specially bankers make use of geo-location techniques in order to prevent it being execute in sandboxes so this is one of the vectors the system must consider. Therefore, these issues will be addressed in future work.

VI. CONCLUSIONS

In this paper, we propose a collaborative framework for Android MD that allows to find events correlation among common malicious URLs from the app traces in the smart device and the DNS network-service logs from the mobile operator. This can be used to pinpoint the malware attacks in several unmonitored smartphones in the wireless cellular system. This platform provides a visualization component using the tool dubbed *Kibana* from the Elastic Stack, in particular the malicious URLs corresponding to malware behaviors are highlighted. Our infrastructure is composed of several components namely, the embedded client, the app that collects the network traffic of the application under examination, Python-language scripts that allow processing of the malicious URLs at the servers that gather the app traces and DNS logs, and the search and analytics cluster (Elastic Stack). So, any Android application (up to 19 API level) can be monitored without rooting the phone or changing its firmware. Further improvements on the visualization quality and the user

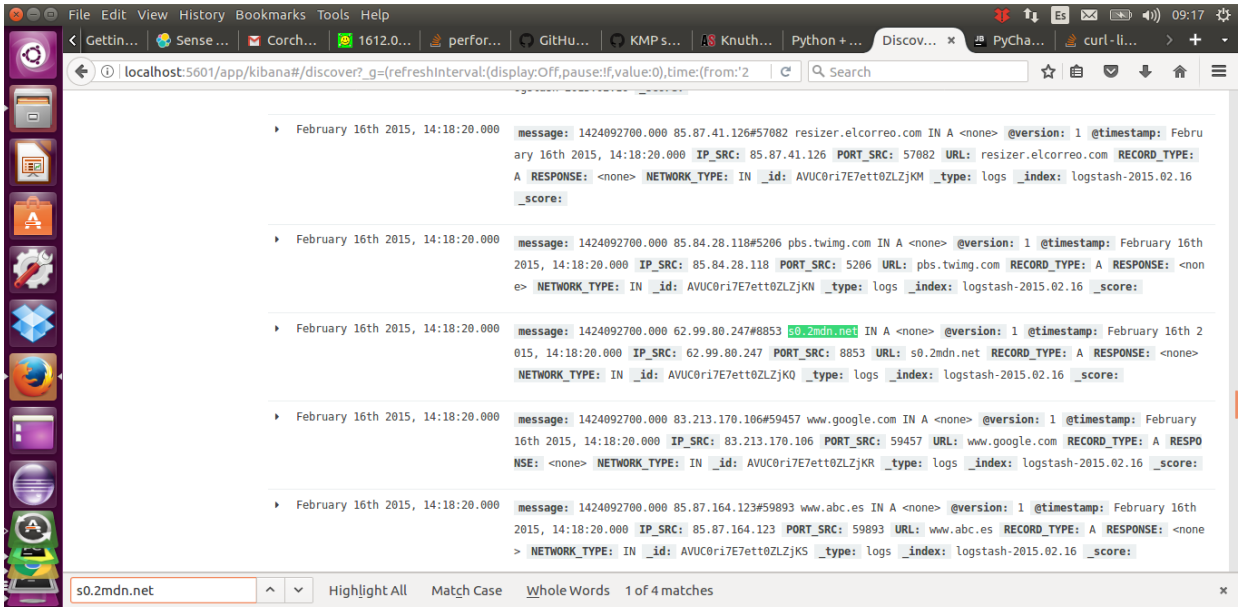

Fig. 3: Indexed DNS log after being processed by *Logstash* tool and stored in the *E*S, and it is shown using the tool for searching and data visualization called *Kibana*. The queried URL is **s0.2mdn.net**.

interface are possible, but the design and implementation of our platform demonstrated to be promising.

TABLE I: MALICIOUS URLs AFFECTING SEVERAL SMART DEVICES

Malicious URLs from smartphones	No. of times in the DNS logs
S0.2mdn.net	2161
alog.umeng.com	1630
thepiratebay.org	1024
servegame.com	288
pm-m.d.chango.com	30

REFERENCES

[1] IDC, ""smartphone shipments os market share, 2017 q1," http://www.idc.com/prodserv/smartphone-os-market-share.jsp." Available Online, 2017.

[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] C. Mulliner, "Adbi," 2016. [Online]. Available: https://github.com/crmulliner/adbi

[4] Elasticsearch. [Online]. Available: https://www.elastic.co/

[5] K. Lu, Z. Li, V. P. Kemerlis, Z. Wu, L. Lu, C. Zheng, Z. Qian, W. Lee, and G. Jiang, "Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting." in *NDSS*, 2015.

[6] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[7] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 2012, pp. 62–69.

[8] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 15–26.

[9] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors." in *NDSS*, 2015.

[10] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[11] 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.

[12] 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.

[13] J. Malik and R. Kaushal, "Credroid: Android malware detection by network traffic analysis," in *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*. ACM, 2016, pp. 28–36.

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

[15] V. Total, "Virustotal-free online virus, malware and url scanner," 2012.

[16] N. T. Cam and N. C. H. Phuoc, "Nesedroidandroid malware detection based on network traffic and sensitive resource accessing," in *Proceedings of the International Conference on Data Engineering and Communication Technology*. Springer, 2017, pp. 19–30.

[17] URLVoid. [Online]. Available: http://www.urlvoid.com

[18] W. of Trust. [Online]. Available: http://www.ywot.com

[19] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in *Ndss*, 2011.

[20] 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.

[21] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM journal on computing*, vol. 6, no. 2, pp. 323–350, 1977.

[22] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Emulator vs real phone: Android malware detection using machine learning," in *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*. ACM, 2017, pp. 65–72.