An Assistive System for Android Malware Analysis to Increase Malware Analysis Efficiency

Suyash Jadhav* and Tae (Tom) Oh+
Dept. of Information Sciences and Technologies+
Dept. of Computing Security*+
Rochester Institute of Technology
Rochester, NY 14623
Email: ssj8127@rit.edu and tom.oh@rit.edu

Jaehoon (Paul) Jeong
Dept. of Interaction Science
Sungkyunkwan University,
Suwon, Republic of Korea
Email: jaehoon.paul@gmail.com

Young Ho Kim and Jeong Neyo Kim
Mobile Security Research Section
Information Security Research Section
Electronics and Telecommunications Research Institute (ETRI),
South Korea

***Abstract*—After the comprehensive survey of existing malware analysis products in the market, the survey result has shown that an assistive tool is needed to help researchers to automatically predict an Android malware in a given large number of applications. Due to a large volume of Android applications being developed and distributed every day through third party application stores, it is difficult to detect a set of Android applications which might potentially indicate malicious behaviors. This situation results in high inefficiency in performing the manual analysis of applications that are uploaded to Android application stores. This research focuses on creating an Android malware analysis lab environment with assistive services that helps the prediction of malicious applications from a given large set of both benign and malicious Android applications. This lab environment is ideal for third party Android application stores and malware researchers in performing Android malware analysis on such an application set.**

Keywords-Malware; Assistive System, Analysis, Malware, Analysis

I. Introduction

Android application development and Android mobile market has been expanding tremendously in last few years. One of the core success reasons behind Android OS is easy application development and distribution. Google Play Store and many other Android application markets across the globe allow developers to upload and distribute their applications almost in real time. Publishing and updating applications is very easy and performs minimal manual review by the stores. The ease of distribution supports developers, but the distribution of malicious applications is considerably easy. Due to the tremendous volume and continuous increase in Android application uploads, it is highly difficult and costly to do the manual review of each newly submitted application. These Android market places resulted in the fairly easy creation and distribution of Android malwares.

Due to limitations on the manual review of applications, an Android market needs a fast and efficient way to automate malware analysis as an assistive system. Leading application stores like Google Play Store have developed their own automated malware analysis solutions and reduced the existing malwares, but small-sized application stores still mostly depend on traditional hash-based scanning techniques.

This research was initiated by a great need and scope for automated Android malware analysis. Therefore, this paper discusses an assistive system for Android malware analysis called AndroSandX. This AndroSandX is an assistive tool to reduce a set of possible malwares in a given random set of Android applications and provides a ticket-based malware analysis as a systematic approach for the creation of intrusion detection system (IDS) signature through the manual analysis, signature storage and testing.

II. Related Work

Almost 1.4 Billion mobile devices were sold in 2015, out of which 5/6 of the devices use Android OS [3]. The number of mobile devices running Android OS has been increasing tremendously in recent years. The increase attracted many sophisticated attacks and exploited development on Android platform [2, 3]. Cyber criminals are investing resources in a large amount of malwares and exploit an Android platform [4]. Malwares in Android are not just Short Message Service (SMS) Trojans or Spyware, but also multiple categories of Android malwares are increasing, such as Ransomware, Hostile Downloaders, and Botnet Trojan [1, 2].

978-1-5090-6231-7/17 $31.00 © 2017 IEEE
DOI 10.1109/WAINA.2017.26

Google categorizes Android malwares into 17 different categories as described in TABLE 1 [1]:

TABLE 1 : Google's Android Malware Categories

Backdoors	Commercial Spyware	Phishing Applications
Billing Fraud	Data Collection	Trojan
SMS Fraud	Denial of Service	Ransomware
Call Fraud	Hostile Downloaders	Rooting Apps
Spam	Spyware	Non-Android Threat
Wireless Access Protocol (WAP) Fraud	Privilege Escalation Apps	

Android malware analysis involves multiple stages. Real-world detection techniques are limited to a network-based detection and a host-based signature detection. The network-based detection includes a network data transfer level byte pattern match, network protocol, IP/Domain blacklisting, and URL pattern match. The host-based detection involves hash and byte code pattern based detection in most cases. Android application development and stores have a huge diversity, which leads to the rapid development of applications. A large number of applications are uploaded every day across the globe, targeting multiple variants on the Android platform. It is fairly difficult to perform manual analysis on all of the applications. The reusability of code and easy distribution of applications allows malware authors to easily avoid signature-based detection.

There are different approaches for malware detection suggested by Android security researchers around the world. Android malwares can be detected through code-level signatures and Android instruction code [5]. The detection through a certain pattern matching, such as application package name and methods/functions is possible. Some other interesting approaches include the visual representation of binaries, hybrid analysis using an attack tree based approach [6, 7]. Researchers have developed assistive malware prediction tools to allow malware researchers to find possible unknown malwares in a given set of applications [8, 9]. Multiple researches have been performed to find the feasibility to automatically categorize Android applications. Researchers utilized static code-level data to perform the categorization of Android applications [10-13].

III. A Proposed System Architecture

Figure 1 shows the overview of the hardware implementation and network structure. The system has a core server that runs the AndroSandX server applications mainly developed using Python. MongoDB is a primary database used by this system. The system allows the use of Android virtual instances as well as the actual Android devices. The assistive system is developed to empower researchers to use their laptops connected to a wired or WiFi network.

Our proposed system has a peripheral firewall, which is required to restrict the inbound connection to any compromised devices in order to avoid any Android backdoor-based exploitation. The peripheral firewall protects the research network from the unrestricted access from external networks. The server has a different VirtualBox which creates a virtual network to initiate all the Android virtual instances. These Android instances have access to the Internet though an IP table firewall, which allows very restrictive outbound connections. The IP table firewall on the server avoids the lateral infection which is caused by any Android malware. The system also has malware research archival facility and data sharing services, which allow for the collaboration among researchers.

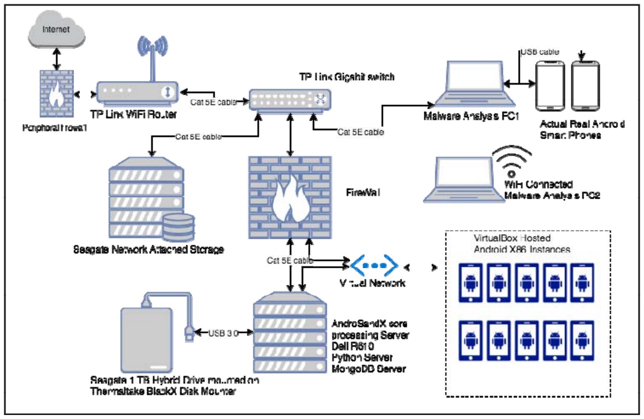

Internet
Peripheral Firewall
TP Link WiFi Router
Cat 5E cable
TP Link Gigabit switch
USB cable
Malware Analysis PC1
Actual Real Android Smart Phones
Cat 5E cable
Cat 5E cable
Seagate Network Attached Storage
FireWall
WiFi Connected Malware Analysis PC2
VirtualBox Hosted Android X86 Instances
Cat 5E cable
Virtual Network
USB 3.0
AndroSandX core processing Server Dell R610 Python Server MongoDB Server
Seagate 1 TB Hybrid Drive mounted on Thermaltake BlackX Disk Mounter

Fig. 1. A Proposed System Architecture

IV. The Features of the Proposed System

A. Static Analysis

Static analysis mainly focuses on the data that can be collected from an Android application executable .apk file. The focus of the static analysis is bound around the code, possible strings and sub routines in the application code. The host intrusion detection system usually utilizes this approach. In most cases, the static analysis focuses on the creation of hash signatures and byte code pattern signature for the detection of malicious applications. De-assembling and manual code reviews are core tasks in the static analysis. Hardcoded strings usually give an idea about the possible malware domains and IP addresses. Also, sometimes static strings which are a symmetric key used in the encryption of data. In case of botnet malware, static signatures reveal details about the command and control communication mechanism, capabilities and a possible instruction set that a bot can perform. Although much information is revealed in the static analysis, this information does not include complete data.

The static analysis in this research allows for a collection of Android application permission data, registered broadcasts, cryptographic functions, hardcoded strings, IP addresses, and functional calls to a system that allow for the access to private data.

B. Dynamic Analysis

Dynamic analysis focuses on the runtime behavior of the application. The analysis involves the execution and monitoring of the activity performed by the applications. Activity monitoring and data collection is carried out at the system, network and memory levels. The system-level data includes system calls and system functions invoked by the applications. The network-level data includes DNS requests, payloads, domains and IP addresses contacted by applications. Network data further reveals details about protocols and HTTP requests made by the applications. Memory-level data is usually obtained during the debugging of applications, which involves code-level breakpoints and monitoring of register, stack and heap data.

The dynamic analysis is sometimes a more powerful way than the static analysis as it reveals the actual behavior of malware and any intention hidden in functionalities. The further analysis of the dynamically downloaded payloads or code modules is possible, but most of the malwares nowadays download configuration files in runtime. For analyzing botnet and bot-network behaviors, the dynamic analysis is very helpful. In network-based intrusion detection/prevention system (NIDS/NIPS), signatures are most of the time collected through the dynamic analysis and a posterior infection activity is very important for the creation of network-based IDS/IPS.

In this research, every application under the analysis is triggered in a sandbox to monitor its runtime behavior. The collected behavioral data includes system calls, accessed files during execution, HTTP requests, network protocols, data transfer rates, suspicious broadcast receivers, fingerprinting activity, and high risk system call pattern.

C. Hybrid Analysis

Our system utilizes a hybrid analysis approach, which is a best way to gather all analysis relevant data. This hybrid analysis approach utilizes the combination of static and dynamic analysis. This approach might be resource intensive and time consuming, but provides most significant, accurate results due to the completeness of data. The static analysis is usually time-efficient, but the dynamic analysis is more depth-oriented. The hybrid analysis allows for the creation of robust signatures and the extraction of dynamic detection features as well as the gathering of the maximum number of features. As a result, this analysis performs more accurate detection as well as the in-depth analysis and signature creation.

D. Detection Techniques

Malware detection techniques are categorized into two approaches, such as a signature-based approach and a behavior-based approach. The signature-based approach is very accurate in detecting known malwares. On the other hand, the behavior-based approach allows for the detection of potential unknown malwares.

E. Malware Detection Evasion Techniques

Symantec 2016 Internet Security Threat Report stated that the Android malwares are using code obfuscation to avoid signature-based detection, and use multiple sandbox detection techniques [3]. This note from leading vendors like Symantec raises an urge of focusing on more than simple signature-based malware detection system. Also anti-evasion techniques at a robust sandbox level need to be utilized while analyzing malwares. A system integrates Cuckoo Droid [14] as a core sandbox solution. Cuckoo Droid provides an excellent Android sandboxing solution which considers possible anti-detection measures. Different techniques are implemented, such as the falsifying of the system properties, contacts, and network properties on an Android virtual device to overcome malware evasion techniques. The use of anti- evasion Android framework like Xposed makes Cuckoo Droid a robust Android sandbox.

F. Malware Sandboxing and Data Collection

Different Android sandboxing and malware analysis solutions have been created and tested for their efficiency. Automation and extraction of data from different phases of Android application execution are the core requirements of a sandbox. Analysis data can be extracted from static code level data, dynamic execution time data, network data, and Android OS memory related data. Cuckoo Droid is one of the most excellent, available Android malware sandboxes and provides data for static, dynamic and network level analysis. Additionally, Android system level function calls and application runtime data are provided as well.

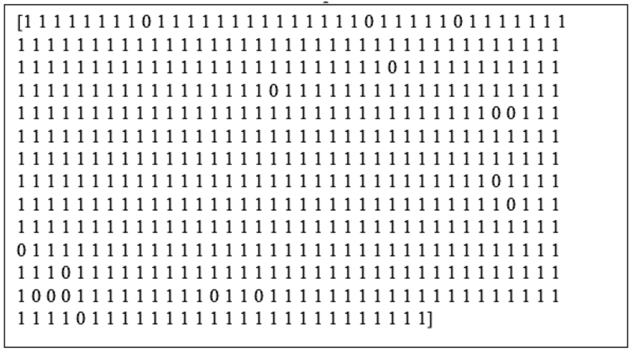
G. Android Malware Analysis using Classificaiton or Regression Techniques

Google uses Bouncer, a behavior-based analysis system in order to detect malwares in Google Play Store [15]. Bouncer utilizes artificial intelligence to perform behavior-based analysis to predict potential known malwares. Researchers utilize features that are extracted from the static analysis or hybrid analysis data. Machine learning algorithms have given very good results and are strong indicatives that more research needs to precede the utilization of this approach. Machine learning algorithms for Android malware detection are support vector machine (SVM), random forest, decision tree, and neural networks. The choice of a machine learning algorithm depends on the data used for the feature extraction. The feature extraction input data can be strings, floating values, continuously varying values, binary values or Boolean. The classification algorithm efficiency also varies with the type of analysis being used for extracting data. The static analysis, dynamic analysis or hybrid analysis needs different algorithms as the efficiency of classification changes significantly.

This research utilizes machine learning techniques to predict the possible subset of malicious applications from a given set of both benign and malicious applications. This technique uses 153 unique features that are extracted from the data, which was collected though the hybrid analysis. The function of AndroSandX provides a great help in minimizing the amount of time required for actionable threat intelligence instead of the manual malware analysis process. These 153 features extracted by the system have the data type of float or binary. Through the testing of multiple machine learning algorithms with a feature set, the results have shown that the Random Forest works best in the feature set.

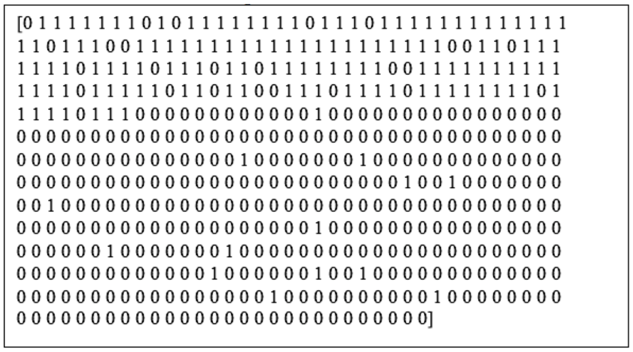
Following a data set array shows the predicted output for the classification and regression of a sample set. SVM and

Random Forest Algorithm prediction data set is given below. The total sample set length is 509. Number 1 in output represents “Malware” class and number 0 represents “Benign” class.



```
[1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Fig. 2. Support Vector Machine Classification Output

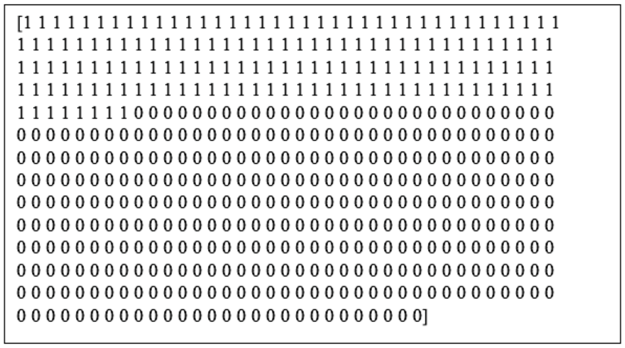


```
[0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1
1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1
1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Fig. 3. Support Vector Machine Classification Output

V. Sample Set Classification

The following data array shows the actual classification of the applications. Among 509 applications submitted for analysis, 156 applications were classified as “Malware” and 353 applications were classified as “Benign”.



```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Fig. 4. A Sample Set Classification

The following matrix indicates the efficiency of the prediction using created models:

Score indicates the predication accuracy of specific model, the higher score indicates the better success of guessing the correct classification of a submitted application.

Support Vector Machine Model Accuracy Score:
0.320235756385

Random Forest Classification Model Accuracy Score:
0.92141453831

Precision Score:
Precision indicates the accuracy from false positive, false negative, true positive and true negative score prospective:

Precision :
Precision = True Positive / (True Positive + False Positive)

For class malware and class benign respectively:
[0.92876712, 0.90277778]

Average precision: 0.896551724138

Recall :
Recall = True Positive / (True Positive + False Negative)

For class malware and class benign respectively:
[0.96033994, 0.83333333]

Average recall: 0.83333333

VI. Conclusion

Under this research, an Android malware analysis assistive system is created and developed. The result proves that the system successfully predicts malwares from a given set of randomly selected Android applications. The system increases the efficiency of Android malware analysis, and also the major benefit is producing actionable threat intelligence.

Acknowldegement

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (B0190-16-2032, Development of Operating System Security Core Technology for the Smart Lightweight IoT Devices). Note that Tae (Tom) Oh is the corresponding author.

References

[1] "Google Android Security PHA classification", *source.android.com*, 2016. [Online]. Available: https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_PHA_classifications.pdf. [Accessed: 09- Jul- 2016].

[2] "Google Android Security Report: Malware Down, Harmful Apps Still the Platform's Biggest Threat - Security News - Trend Micro USA",

Trendmicro.com, 2016. [Online]. Available: http://www.trendmicro.com/vinfo/us/security/news/mobile-safety/google-android-security-report-harmful-apps-the-biggest-threat. [Accessed: 09- Jul- 2016]

[3] "Internet Security Threat Report 2016 | Symantec", *Symantec.com*, 2016. [Online]. Available: https://www.symantec.com/security-center/threat-report. [Accessed: 09- Jul- 2016].

[4] "Android-Malware-Threat-Report", *www.bitdefender.com*, 2016. [Online]. Available: http://download.bitdefender.com/resources/files/News/CaseStudies/study/85/Android-Malware-Threat-Report-H2-2015.pdf. [Accessed: 09- Jul-2016].

[5] M. Zheng, M. Sun and J. C. S. Lui, "Droid Analytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware," 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, pp. 163-171, 2013.

[6] A. Jain, H. Gonzalez, and N. Stakhanova, "Enriching reverse engineering through visual exploration of Android binaries," Proceedings of the 5th Program Protection and Reverse Engineering Workshop on - PPREW-5, no. Article 9 , pp. 9 pages., 2015..

[7] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, pp. 1—12, Dec. 2015.

[8] S. Zhao, X. Li, G. Xu, L. Zhang and Z. Feng, "Attack Tree Based Android Malware Detection with Hybrid Analysis," 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, pp. 380-387, 2014.

[9] A. Apvrille and L. Apvrille, "SherlockDroid: A research assistant to spot unknown malware in Android marketplaces," Journal of Computer Virology and Hacking Techniques, vol. 11, no. 4, pp. 235–245, Jul. 2015.

[10] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST," Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks - WiSec '13, pp. 13—24, 2013.

[11] H. Y. Chuang and S. D. Wang, "Machine Learning Based Hybrid Behavior Models for Android Malware Analysis," 2015 IEEE International Conference on Software Quality, Reliability and Security, Vancouver, BC, pp. 201-206, 2015.

[12] S. Pai, F. Troia, C. Visaggio, T. Austin and M. Stamp, "Clustering for malware classification", Journal of Computer Virology and Hacking Techniques, pp. 1--13, 2016.

[13] L. Apvrille and A. Apvrille, "Identifying Unknown Android Malware with Feature Extractions and Classification Techniques," 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, pp. 182-189, 2015.

[14] "CuckooDroid Book — CuckooDroid v1.0 Book", Cuckoo-droid.readthedocs.io, 2016. [Online]. Available: http://cuckoo-droid.readthedocs.io/en/latest/. [Accessed: 09- Jul- 2016].

[15] D. Kaplan and D. Kaplan, "Google employs Bouncer to cleanse Android malware", iTnews, 2016. [Online]. Available: http://www.itnews.com.au/news/google-employs-bouncer-to-cleanse-android-malware-289242. [Accessed: 30- Jul- 2016].