International Journal for Information Security Research, Volume 9, Issue 2, June 2019, pp. 871-880
DOI:10.20533/ijisr.2042.4639.2019.0100

Host-Based Detection and Analysis of Android Malware: Implication for Privilege Exploitation

Moses Ashawa, Sarah Morris,
Centre for Electronic Warfare, Information, and Cyber,
Cranfield University,
Defence Academy of the United Kingdom,
Shrivenham
SN6 8LA
M.ashawa@cranfield.ac.uk, s.I.moriss@cranfield.ac.uk

Abstract

The Rapid expansion of mobile Operating Systems has created a proportional development in Android malware infection targeting Android which is the most widely used mobile OS. factors such Android open source platform, low-cost influence the interest of malware writers targeting this mobile OS. Though there are a lot of anti-virus programs for malware detection designed with varying degrees of signatures for this purpose, many don't give analysis of what the malware does. Some anti-virus engines give clearance during installations of repackaged malicious applications without detection. This paper collected 28 Android malware family samples with a total of 163 sample dataset. A general analysis of the entire sample dataset was created given credence to their individual family samples and year discovered. A general detection and classification of the Android malware corpus was performed using K-means clustering algorithm. Detection rules were written with five major functions for automatic scanning, signature enablement, quarantine and reporting the scan results. The LMD was able to scan a file size of 2048mb and report accurately whether the file is benign or malicious. The K-means clustering algorithm used was set to 5 iteration training phases and was able to classify accurately the malware corpus into benign and malicious files. The obtained result shows that some Android families exploit potential privileges on mobile devices. Information leakage from the victim's device without consent and payload deposits are some of the results obtained. The result calls proactive measures rather than proactive in tackling malware infection on Android based mobile devices.

1. Introduction

The evolution of information technology and its swift development of speedy transformation of wired and wireless communication in mobile computing especially smartphones have brought a proportional increase in malware development and attacks. Mobile devices today have played significant roles in our daily activities beyond making calls, sending SMS and MMS. Financial transactions such as mobile banking and online shopping are readily made easy using mobile devices. The dramatic growth in popularity and usage of the Android OS is as a result of their high computational abilities, openness and low cost of the OS [13]. In addition, Android has created a significant impact in the permeation of broadband. Android has become a vulnerable operating system which is heavily targeted by malware writers across the broad spectrum of the society. There are a lot of detection techniques proposed and developed by different security researchers and mobile companies to fight this menace. However, none of the detection techniques and anti-virus engines prove to be 100% (percent) accurate in malware detection. As powerful as those detection mechanisms are, many have significant limitations in their analytical domain. The design of Android security is in a way that the running applications are isolated in the sandbox; thus, prohibiting direct interaction from the system assets such as SMS, address book, MMS, GPS.

Before an application is given access to the device resources during installation by the user, a proper verification check is performed by the binder to ensure that the application in quote meets the prerequisite for accessing the system resources. Other system resources such as activity manager, package manager, network state, launcher permission and service monitor are also guided by the binder both at the user and kernel space. When device is been infected, transactions at the binder driver are been intercepted by malware.

This paper proposed a host-based detection of android malware by creating Linux Malware Detect rules on the virtual machine. This approach provides

analytical information about the activities of the malware on the android device, it can also be applied for other types of mobile operating system. This paper covers both old and newest Android malware samples and family distributions with a span of over four years.

2. Review of related literature

This section provided a brief related literature on some of the related works that were carried out on the Android malware.

Dimensionality reduction approach used in the study of [2] extracted malware features when APK files were monitored under random projection. Manifest files and Dalvik [19] executables were filtered using a logistic classifier. The activities performed by the malware was however not significantly identifiable. However, there was a pragmatic substantial reduction within large group of malware hashes with an initial step of pre-processing. Detecting android malware by looking at the device application packages, permission and system calls were performed in the study of [18]. The research of [3, 4, 5] looked at how malware can be avoided by looking at the installation permission spectrum and analysis of the application package before installation before the data availability in the application enhances detection with systematized prevalence [8, 9, 10, 11].

The study of [6] investigated how Android malware can be detected by close surveillance for overlapping their un-dynamic attributes. The study of [7, 14] applied visualization and disassembly to inspect Android malware chromatic similitude. The result shows that some Android malicious packages exhibit similar attributes to other families and has high propensity of evading detection in virtual environments especially.

3. Methodology

The detection methodology defined in this work uses dynamic approach, meaning that the malware sample was actually executed. The selected sample files were run in an isolated virtual environment and their behavior was subjected to observation and analysis. Due to the danger of malware, the methodology was separated into different approaches ranging from installation and configuration of tools and services. Details of the methodological approach is provided in the subsections below.

3.1 System setup

The experimental design of this system was based on some components and services. The setup lab was designed with open source tools which have the ability to extract Android Apps libraries and resources. The safety of the physical machine was ensured by isolating from the real machine and physical resources. The system setup and configuration were performed on Bionic Beaver which is an ubuntu codename for version 18.04 of the ubuntu Linux-based operating system. The structure of the experiment was based on different security open source tools which enhance application feature extractions during execution. The configured tools have ability for scanning, monitoring and detecting significant malware dynamic features ranging from battery consumption, GRP locations and others.

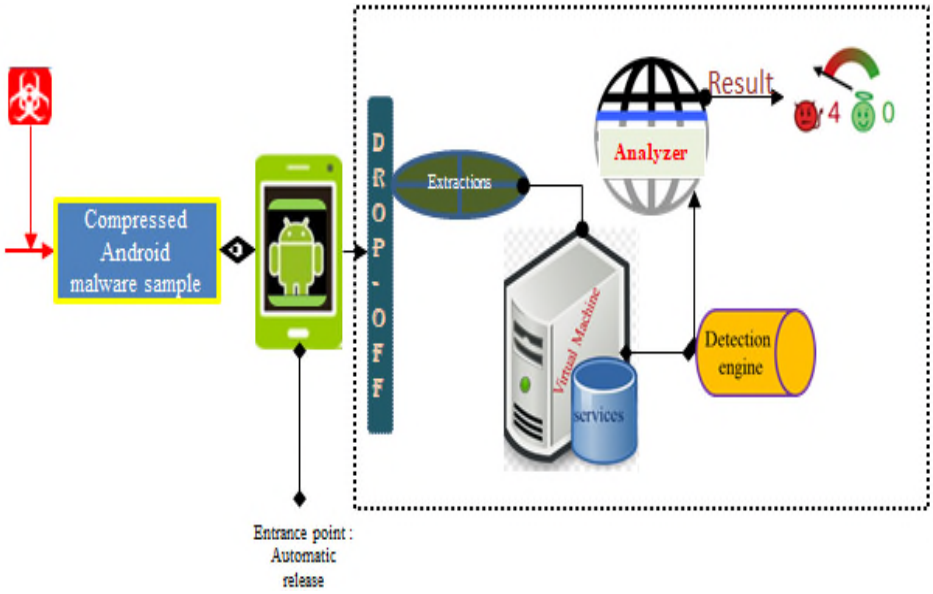

Figure 1. System setup for detection and analysis process

3.2 Dependencies

For the configuration to achieve the research aim, some services where installed and configured on the virtual machine. The system was designed with dependencies for repackaging and unpacking Android applications running on the devices. Python version 3.6.5, ClamAv, volatility, chkrootkit, rhunter and Linux Malware Detect (LMD) were installed as services for specific actions. The detected android malware file was then exported to virus total for analysis. LMD scans malware and other threats in a host-based environment [15] and behavioral monitoring [17]. It uses both HEX and MD5 hashes to generate the required signature from ClamAv for malware features extraction which are derivatives of malware files from users. In this experiment, the rules were written in five different segments after download and installation. It has unlimited capability to detect and analyse 8,888 malware hashes of Trojans, viruses, rootkits and other threats.

The **cron.daily** dependency helps in updating daily emerging signatures with constant variable release and changelog. Five most significant segments of the detection rules are summarily presented below (see Table 1 for details).

i. Setting manual email scan reports for enabling malware alert.
ii. Signature files for scanning multiple of weekly released malware with different signatures.
iii. Scan options to specify maximum file sizes that can be scanned per time interval.
iv. Quarantine segment for suspending suspected files.
v. Analysis mode for checking obfuscated files in an encoding mode.

Table 1. LMD rules for detection

Functions	Rule Explanation
Setting manual email scan reports	# ' create an email address called christfavour783@gmail.com'. # [0 = disabled, 1 = enabled] email_alert="1" email_addr="christfavour783@gmail.com" email_subj="Android malware alerts for $christfavour783 -$(date +%Y-%M-%d)" email_ignore_clean="1"

Enable signature	# Enabled as new signatures a released multiple times per-week.
	autoupdate_signatures="1"
	autoupdate_version="1"
	autoupdate_version_hashed="1"
	cron_prune_days="21"
	import_config_url=""
	import_config_expire="43200"
	import_custsigs_md5_url=""
	import_custsigs_hex_url=""

Scan options	# The maximum directory depth that the scanner will search.
	scan_max_depth="15"
	scan_min_filesize="24"
	scan_max_filesize="2048k"
	
	scan_hexdepth="65536"
	
	scan_hexfifo="1"
	
	scan_hexfifo_depth="524288"
	scan_clamscan="1"scan_tmpdir_paths="/tmp /var/tmp /dev/shm /var/fcgi_ipc"

Quarantine	# The default quarantine action for malware hits quarantine_hits="1"
	quarantine_clean="1"
	quar_susp="1"
	quarantine_suspend_user="0"
	quarantine_suspend_user_minuid="500"

Analysis	#This is useful as obfuscated code is often stored using encoding methods
	string_length_scan="0"
	string_length="150000" clam av="1"

3.3 Machine Learning algorithm

Machine learning [20] generally abbreviated as ML is a branch of the Artificial Intelligence which deals with computerisation of data logical prototypical construction. ML enables systems to identify and classify data arrays, making decisions that are minimally based on the interference of humans. Machine learning can be supervised, unsupervised, reinforced and semi-supervised learning. Supervised machine learning deals with training dataset to perform some regression or classification functions using algorithms such as decision tree, random forest Naïve Bayes, nearest neighbour, linear regression, neural networks.

The unsupervised ML searches and divides dataset of a particular sample based on its algorithm for mining, detecting, describing patterns for data grouping. The basic feature of this learning is that the target variables are usually not available. Commonly used unsupervised algorithms are associated rule and k-means clustering algorithms. This study used unsupervised K-means algorithm to classify the malware corpus. K-means classification produced different clusters on the malware dataset at each iteration phase of training.

4. Experiment

A sizeable Android malware dataset was obtained from Contagiodump project [1] for this study. The dataset was collected with the file name "Android-malware-master.zip" having a total size of 163 malware with 26 different malware families including benign genuine Android applications. Using Enguage Digitizer [16], the graphical illustration of the dataset distribution of the malware families was generated (see figure 3). In the distribution, it is observed that there is high degree of entropy in the sample size. Due to this visible distinctiveness in the elevation of this files' high level of entropy, other applications will be considered for detection in the experiment, but much credence will be given to the recent family to discover their determined characteristics.

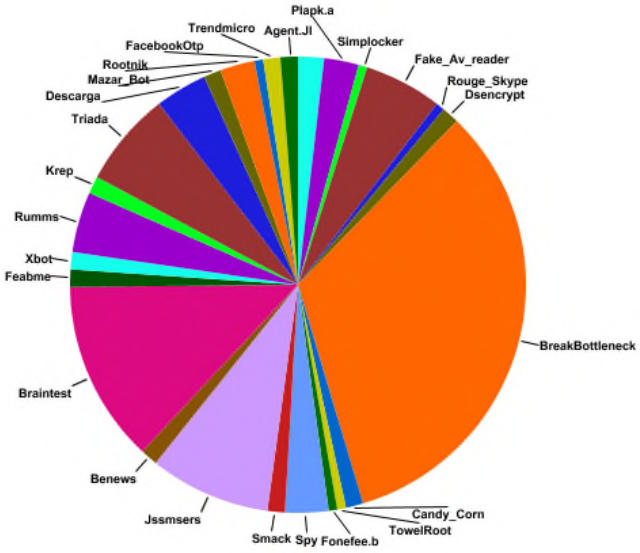

Figure 3. Distribution of the malware corpus using enguage digitizer

4.1 Malware Corpus Timeline

It is very important to know when a malware family immerged or was discovered in order to determined when the payload effects on the vulnerable device is been patched. Using Androguard, the malware corpus timeline was carefully acquired by inspecting the 26 malware families with 163 sample apk files.

Table 2. Dataset timeline consisting 26 android malware families and 163 sample files.

Android Malware	APKs	Discovered Date
Fave-Av-Reader	9	2013-08-31
Plapka	4	2013-09-07
Simplocker	1	2013-09-07
Rouge_Skype	1	2013-09-07
Dendroid	3	2013-12-20
Dsencrypt	2	2014-06-03
BreakBotleneck	54	2014-07-02
Candy_Corn	2	2014-01-26
TowelRoot	1	2014-11-30
Spy	5	2015-01-23
Simack	2	2015-02-21

Jssmsers	14	2015-02-21
Benews	2	2015-07-12
Braintest	21	2015-08-21
Feabme	2	2015-04-16
Xbot	2	2015-12-28
Rumms	7	2016-03-14
Krep	2	2016-05-23
Triada	11	2016-03-05
Descarga	6	2016-05-08
Mazar_Bot	2	2016-11-04
Rootnik	4	2016-11-24
FacebookOtp	1	2017-02-26
TrenMicro	2	2016-12-02
Agent.JI	2	2017-02-17
Fonefee.b	1	2017-04-20
Total samples	163	

4.2 Detection and feature extraction

Feature extractions from the dataset were done by installing the applications on 6.0.1 Android version through the root directory of the detection engine. This required authentication to reload the system services upon the virtual machine for execution. The basic authentication required was the password (see figure 4a), the permission that every application requires before installation on any device. After granting permission for applications installations, the system detected immediately that a change in the system privacy modification was tempered with and generated a warning signal when the account check was performed (see figure 4b).

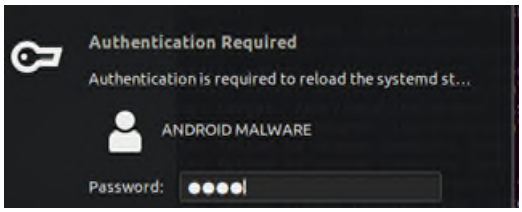

Fig. 4. (a)Required Authentication

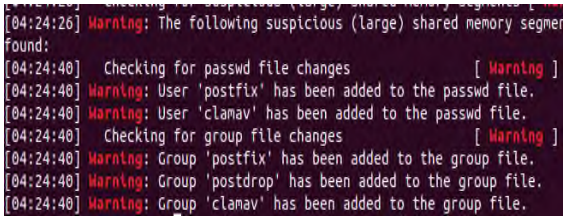

Fig 4. (b) Performing account check

Fig.4. (c) LMD captured detection results

4.3 Classifying malware corpus using K-Means

To find the similarity group in the Android malware data set and effectively classify the malware corpus K-means algorithm [12] was implemented in this work. This is an unsupervised machine learning algorithm used to group individual android application based on similarity in the population. K number of clusters were specified in the algorithm. Cluster 1 represented the group of benign applications in the data set while cluster 2 represented the malicious applications for classification in the data set.

Algorithm: K-Means clustering Algorithm

1. *Select K points, set of points X_1--- X_n*
2. *Randomly initialise the center point C_1---C_k*
3. *Assign malware corpus to the closest cluster centroids*
4. *Compute the center point of each cluster*
5. ***Repeat** step 2 to 3 until no more change to the center points*
6. *Exit*

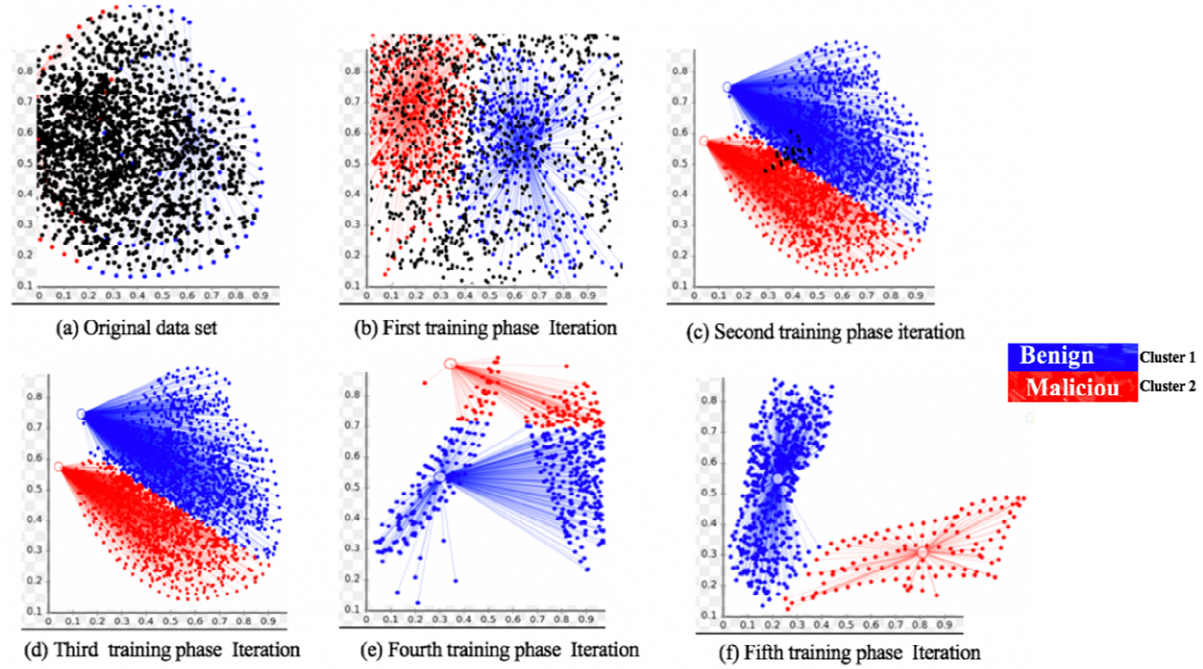

Figure 5. Classifying benign and malicious applications at each iteration

The algorithm was permitted to specify the closest, widest and marginal delta points at its space to observe and then append to each cluster at each iteration phase. The intention was to maximize the degree between the clusters and the given observation. If the delta point between a point to the two centroids is the same, it means the point could belong to the same group, thus classified as benign or malicious. At each iteration, the delta points between the clusters were systematically mirrored to group the dataset with similar attributes.

4.4 Metrics for performance evaluation

The following metrices were used for performance evaluation of the classification system.

Accuracy: This shows the ratio of the overall integral value of the applications sample which are classified in a correct measure as malicious or benign.

Precision: This is the ratio of the actual malicious applications that are correctly classified or detected to the total number of the applications that the LMD model detected as malicious.

Recall: This is a ratio of true positive to its function and the additive value of the false negative impact.

Confusion Matrix: This is the applications which are classified correctly or incorrectly. Our confusion

matrix (**see table**) summarised our classification model performance on the collected Android malware dataset. The true values with their respective derivatives are also represented

accordingly. The integer of the malicious applications the LMD model correctly detected as malicious (True positive). The integer of the benign applications the LMD model incorrectly detected as malicious (false positive). Also, the integer of the benign applications the LMD model correctly classified as not malicious (True Negative). Finally, the integer of the malicious applications the LMD model incorrectly classified as not malicious (False Negative).

Table 3: Confusion matrix

Measure	Value	Derivation
Sensitivity	0.7027	TPR = TP / (TP+FN)
Specificity	0.7053	SPC = TN / (FP+TN)
Precision	0.73581	PPV = TP / (TP+FP)
Negative Predictive Value	0.6700	NPV = TN / (TN+FN)
False positive rate	0.29472	FPR = FP / (FP+TN)
False Discovery Rate	0.2642	FDR = FP / (FP+TP)
False Negative Rate	0.2973	FNR = FN / (FN+TP)
Accuracy	0.7039	ACC = (TP+TN) / (P+N)
F-Measure (F1 Score)	0.7189	F1 = 2PT / (2PT+FP+FN)

The relationship between the sensitivity and specificity in the random classification gives an illustration of the Android malware detection by sequence in a custom analogous to the traditional Receiver Operating Characteristics (ROC) curve.

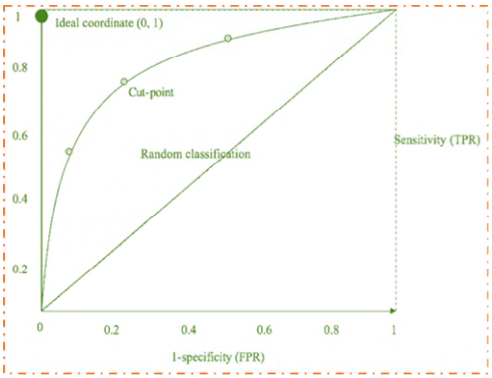

Figure 6. ROC Cure showing Android Malware dataset classification

5. Evaluation

The result analysis for this research was focused on the latest discovered Android malware family (Fonfee.b) in the Dataset which has the package name **com.c101421042723.apk.** The choice to concentrate on Fonefee.b in this study is to give an in-depth study about the research. Most importantly, Fonefee.b is more recent and might not have been patched probably. Another factor of interest is the high level of randomness and the information gain at the leaf nodes of Fonefee.b file during the research. The analysis was focused on permissions requested, files accessed, contacted urls, network connectivity, payload deposition and battery depletion by the malware.

5.1 Permissions requested and URLs contacted

Malicious applications have little leverage on Android devices if their permissions grants are controlled or constrained by users. The result obtained showed that the application requested different privileges when permitted to be installed in the root directory of the application. The research found that the malware established a connectivity with different C&C servers where http requests were made and contents such as images where uploaded at the background to those addresses without the victim's responsiveness (see 6b for details). In this study, permissions requested by the malware and the url contacted by the sample malicious application were shown in figure 6a and 6b respectively. Intent filters by category indicated that the malware is a launcher with the ability of sending SMS without victim's knowledge. Other infection capabilities of this malware were discovered in the bundled files of the device' META-INF/MANIFEST.MF, popup and automatic download are some special characteristics of this malware.

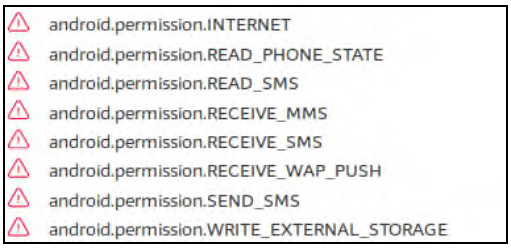

Figure 7. (a)Permission requested

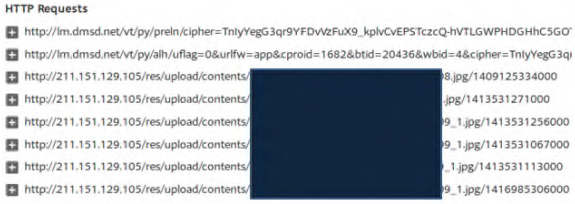

Figure 7. **(b)** Illustrating potential privileges exploited by Fonefee.b android malware

The permission requested remotely accessed files on the victim's device without his awareness. Accessed files include media, pictures, phone text messages and images.

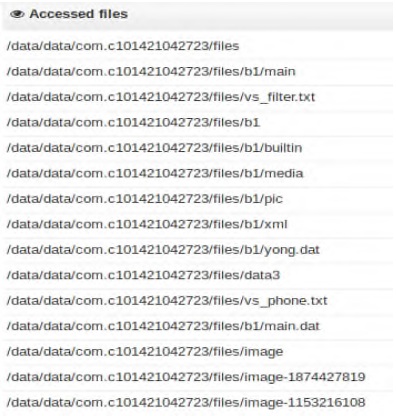

Figure 8. Files accessed by fonefee.b malware

5.2 Network Communication

Fonefee.b was found to be secretly communicating and uploading pictures to the particular IP address of a country. The malware loaded and invoked method **android.net.ConnectivityManager.getMobileDataEnabled** for background data connectivity and **android.net.conn.CONNECTIVITY_CHANGE** for synchronization mechanisms respectively. In addition, background calls, uploading of images and media files were made through the influence of the malware to a country ip address (name on hold for security reasons). The relationship between the contacted domains, **compressed parents, bundled files and the contacted ip address** of the country was obtained.

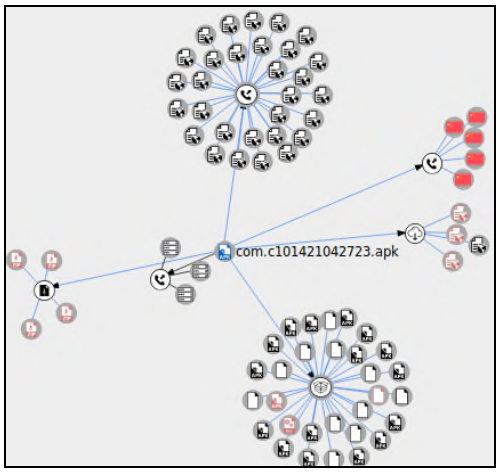

Fig. 9. Malware Activity connectivity

5.3 Payload

Payload is another interesting feature to look for when analysing malware. These are private text containing malware for victim's data encrypted, deletion and spamming. Fonefee.b deposited a payload in the device with an encrypted submitname filename"**cd9404f21e4bc52e477e62037db537c4239b9afcf1d490e1ea75d1c5aefb47ef".** This was primarily for performing phishing attacks while resident in the device. Part of the message was obtained during the analysis which was executed in the particular offset of the device address.

Figure 11. Payload

Name	Type	Address	Offset	Size	Flags
	NULL	0x00000000	0x00000000	0	
.dynsym	DYNSYM	0x00000114	0x00000114	1424	A
.dynstr	STRTAB	0x000006a4	0x000006a4	1517	A
.hash	HASH	0x00000c94	0x00000c94	632	A
.rel.dyn	REL	0x00000f0c	0x00000f0c	328	A
.rel.plt	REL	0x00001054	0x00001054	264	A
.plt	PROGBITS	0x0000115c	0x0000115c	416	A, X
.text	PROGBITS	0x000012fc	0x000012fc	77788	A, X
.ARM.extab	PROGBITS	0x000142d8	0x000142d8	396	A
.ARM.exidx	ARM_EXIDX	0x00014464	0x00014464	2224	A, L

Figure 11. (b) ELF Section Location of the

5.4 Location monitoring

Fonefee.b was found to is an Android Trojan capable of monitoring and tracking the victim's device by taking the coordinates of the victim's latitude. At the time of this experiment, the location of the device was captured at latitude 51.5595N and 1.7910W respectively. This ability of the Trojan to track the device location and send the details to the C&C server of the hacker can be used to launch terrorism or personal attacks on the individual. This is indeed a dangerous Trojan exhibiting almost all the actions performed by the rest of malware class.

Fig. 12. Location tracking

5.5 Battery depletion

The consumption rate per activity was monitored using volatility tool. The ratio of energy consumption to the bandwidth was not proportional

due to the draining effect exerted by the malware. The battery depletion was as a range of many activities performed at the background by the malware. The battery life of mobile device is determined by the rating of the input current of the phone in relation to the load current the circuit holds. By inference, the battery capacity of a mobile device under normal condition will be high when the load activities on the system are low. However, some external forces such as malware make allowances which the device battery life can be affected especially when the runtime is not equivalent to the ratio of load per second running on the device.

6. Conclusion

The results obtained from this study confirmed visibly the sombre menace malware has posed on Android platform operating system. There are many anti-virus programs modelled to tackle this threat. However, the empirical examination of our research revealed that many of these software lack efficacies in detecting malware which emerged with unknown signature by the existed anti-virus programs. Fonefee.b was only detected by 38 out of 59 anti-virus engines. The rest of the engines failed to extract the signature of the sample malware. in addition, more than one cryptographic function and hash values were used while writing this malware program. By implication, this has the ability to exhibit metamorphic attributes to evade detection by many detection techniques through self-repackaging.

Exploitation to gain privilege escalation as a result of the fragmentation problem observed in Android operating system can enhance mobile vulnerability risk before a security patch is initiated. It is observed in this research that Android lacks firm rheostat on some of the APIs such as “sendTextMessage” in order to hinder premium-rate sending of SMS at the background when affected with malware. Most of the permissions permitted by malicious applications have the ability of affecting the entire Android OS configuration and META-INF/MANIFEST.MF. It is evident that all the potentially exploited privileges where a function of this vulnerability. Some of the domains that were contacted include lm.dmsd.net, sdk2.cmvideo.cn and vsbbc.com using a mobile communication addresses 221.181.100.84 of the country’s network.

In conclusion, the results obtained and presented in this paper provided a detailed characteristics and threats that malware posed on users of Android based devices and information security at large. These threats range from malware payloads deposition, information harvest, privilege escalation, private communication without the victim’s awareness, upload of the device’s data contents to C&C servers and host of others. These outcomes are indeed graving and call for advance security measures to be implemented to provide a reactive defense mechanism on the OS rather than proactive defense. Future research can be carried out on using neural networks algorithms such as Bayesian regularization to determine the accuracy rate of both the target and output of malware classification sample. The researcher intends to carry further research on how to classify permissions request based on their threat and protection level. The researcher intends to perform further research on how to provide a security perimeter defence around the Google Bouncer in order to fight Android malware infections.

Acknowledgement

This research was fully sponsored by the Petroleum Technology Development Fund, Nigeria. Scholarship scheme year 2018_2021.

REFERENCES

[1] E. Yu. (2018) ‘Contagiodump project” Android MalwareSampleshttps://github.com/ashishb/android-malware/find/master (18 December 2018).

[2] S. Luiza, E. Eirola and J. Karhunen. "Android malware detection: building useful representations," In *Machine Learning and Applications (ICMLA) in 15th IEEE International Conference, Dec.* 201-206, 2016.

[3] X. Ju, "Android malware detection through permission and package," In *2014 International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*, 61-65, 2014.

[4] C. Tansettanakorn, S. Thongprasit, S. Thamkongka, and V. Visoottiviseth, "ABIS: a prototype of android botnet identification system, In *Student Project Conference (ICT-ISPC)*, 1-5, 2016.

[5] M. Yusof, M. Mohd Saudi, and F. Ridzuan, "A New Android Botnet Classification for GPS Exploitation Based on Permission and API Calls," In *International Conference on Advanced Engineering Theory and Applications*, Springer, Cham, 27-37, 2017.

[6] N. Maryam, S. Soltani, and S. A. Seno, "Android malware detection based on overlapping of static features," In *7th International Conference on Computer and Knowledge Engineering (ICCKE 2017), Ferdowsi University of Mashhad, Oct. 26-27*, 2017.

[7] P. Wonjoo, K. Lee, K. Cho, and W. Ryu, "Analyzing and detecting method of android malware via disassembling and visualization, " In *Information and Communication Technology Convergence (ICTC), 2014 IEEE International Conference*, 817-818, 2014.

[8] J. Xuxian, and Y. Zhou, "Dissecting android malware: Characterization and evolution," In *2012 IEEE Symposium on Security and Privacy*, 95-109, 2012.

[9] C. Jian, H. Manar, R. Thomas, and Y. Zou, "Detecting android malware using clone detection," *Journal of Computer Science and Technology, vol.* 30, no. 5, pp. 942-956, 2015.

[10] W. Zhaoguo, C. Li, Y. Guan, Y. Xue, and Y. Dong, "ActivityHijacker: Hijacking the Android Activity Component for Sensitive Data," In *Computer Communication and Networks (ICCCN), 25th IEEE International Conference*, 1-9, 2016.

[11] H. Shun-Wen, S. Yeali and M. C. Chen, "Behavior grouping of Android malware family." In *ICC*, pp. 1-6. 2016.

[12] Y. Chunyong, and S. Zhang, "Parallel implementing improved k-means applied for image retrieval and anomaly detection," Journal of *Multimedia Tools and Applications, vol.* 76, no. 16, pp. 16911-16927, 2017.

[13] G. Fei, J. Niu, Z. Qi, and M. Atiquzzaman, "Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions," *Journal of Network and Computer Applications*, *vol.* 3, no. 2, pp. 42-56, 2018.

[14] W. Markus, F. Fischer, R. Luh, A. Haberson, A. Rind, and A. Keim, "A survey of visualization systems for malware analysis," In *EG Conference on Visualization (EuroVis)-STARs*, 105-125, 2015.

[15] Linux Malware Detect - R-fx Networks, 2017. [Online] Available: https://www.rfxn.com/projects/linux-malware-detect/.

[16] Enguage Digitizer [Online]. Available http://markummitchell.github.io/engauge-digitizer/

[17] Asaf, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Anomaly: a behavioural malware detection framework for android devices," *Journal of Intelligent Information Systems* vol. 38, no.7, pp. 161-190, 2012.

[18] Method and Processes for Securely Autofilling Data Fields in A Software Application, by Caron, Etienne. (2018, January 4). U.S. Patent Application 15/707,647.

[19] J. Hao, H. Yang, S. Qin, Z. Su, J. Zhang, and J. Yan, "Detecting Energy Bugs in Android Apps Using Static Analysis," In *International Conference on Formal Engineering Methods*, Springer, Cham, 192-208, 2017.

[20] W. Andrew, P. K. Ferentinos, and P. G. Petropoulos, "A new synergistic approach for monitoring wetlands using Sentinels-1 and 2 data with object-based machine learning algorithms," Journal of *Environmental Modelling & Software*, vol. 6, no 104, pp. 40-54, 2018.