Journal of Organizational and End User Computing
Volume 32 • Issue 3 • July-September 2020

Android Botnets:
A Proof-of-Concept Using Hybrid Analysis Approach

Ahmad Karim, Bahauddin Zakariya University, Pakistan
Victor Chang, Teesside University, UK
https://orcid.org/0000-0002-8012-5852
Ahmad Firdaus, Faculty of Computer Systems and Software Engineering, Malaysia

ABSTRACT

Mobile botnets are gaining popularity with the expressive demand of smartphone technologies. Similarly, the majority of mobile botnets are built on a popular open source OS, e.g., Android. A mobile botnet is a network of interconnected smartphone devices intended to expand malicious activities, for example; spam generation, remote access, information theft, etc., on a wide scale. To avoid this growing hazard, various approaches are proposed to detect, highlight and mark mobile malware applications using either static or dynamic analysis. However, few approaches in the literature are discussing mobile botnet in particular. In this article, the authors have proposed a hybrid analysis framework combining static and dynamic analysis as a proof of concept, to highlight and confirm botnet phenomena in Android-based mobile applications. The validation results affirm that machine learning approaches can classify the hybrid analysis model with high accuracy rate (98%) than classifying static or dynamic individually.

KEYWORDS

Botnet Detection, Hybrid Analysis, Mobile Botnet, Mobile Malware

INTRODUCTION

Although Android OS being an open source has promoted mobile applications developers, yet malware programmers have also contributed to exploit its open source nature to carry out malicious acts. MacAfee, an antimalware platform, has diagnosed more than 700K mobile malware in the second quarter of 2014 (Weafer, 2014). Another report (Chebyshev, 2016) published in 2016 discovered that Internet access on smartphone devices had exceeded 61% in the first quarter of 2015. This study also revealed that 60.85% of Android users had started Internet access on their smartphone devices. Consequently, the similar growth is observed in malware program construction, i.e., 40,267 new mobile malware variants were analyzed and diagnosed by the security agencies at the end of 2015(Millman, 2015). In Q2 2016, it was observed that Android was used by 86.2% of smartphone users (Paul,

DOI: 10.4018/JOEUC.2020070105

2017). Similarly, its widespread adaptation to other platforms such as televisions, tablets, wearable, and vehicles opened the new dimensions for multi-platform attacks. In the similar pace, IoT (Internet of Things) would be the next target for malware programmers. A more recent report (BILIĆ, 2017) states that the growth of mobile malware is constantly increasing since 2013. On average 200 new malicious code variants have been discovered per month during 2015, this number rose to 300 per month by the end of 2016. As a result, on average 400 new malicious code variants are expected to evolve by the end of 2017 (Weafer, 2016).

Mobile botnet or SMARTbot (Karim, Salleh, & Khan, 2016) is a malevolent action which is inspired from traditional botnets (PC based). The basic motive behind botnet is to gain illegitimate access to someone's personal device (smartphone, tablets, etc.) and makes this device compromised by a bot binary (app). After becoming the part of a bot network, another pivotal role (botmaster) is responsible for controlling this device remotely and to initiate various attacks using some command and control (C&C) channel. Consequently, these devices are then participating in numerous malicious activities including DDoS, ransom, making premium calls, sending text messages and emails without user's consent.

There are two most common analysis strategies exist, static and dynamic analysis. In static analysis, structural properties of program code are observed including permission usage, CFGs, function call graphs and API calls, etc. For static analysis, reverse engineering tools (Lukan, 2012) are deployed to disassemble program code (Schmidt et al., 2009) or directly fetching parameters from executable binaries (Petsas, Voyatzis, Athanasopoulos, Polychronakis, & Ioannidis, 2014; Yousafzai et al., 2016). In contrast, dynamic analysis requires execution of malware binaries in a secure environment (called sandbox) to extract runtime behavior of these applications. Following are some of the parameters which are of interest during dynamic analysis: (a) file operations (b) network traces (c) initiated services (d) HTTP and DNS traffic etc. Currently, some mobile malware detection approaches (Arp, Spreitzenbarth, Hubner, Gascon, & Rieck, 2013; Chen, Rong-Cai, ZHENG, Jia, & Li-Jing, 2016; Fereidooni, Conti, Yao, & Sperduti, 2016; Yang, Wang, Ling, Liu, & Ni, 2017) are introduced targeting either program code or runtime execution traces. However, at a higher level of abstraction, these approaches are targeting mobile malware detection rather than mobile botnet. This is the extension of our previously proposed approaches (Ahmad Karim & Shah, 2015; Karim, Salleh, Khan, Siddiqa, & Choo, 2016) in a way that it can highlight the need for a hybrid analysis framework for the detection of botnet mobile binaries.

In this paper, we will investigate and highlight the problem of the mobile botnet by comparing botnet properties of known botnet applications with existing malware families and benign samples.

Thus, the basic motive of this research is to confirm the existence of mobile botnet phenomenon in Android-based system which is continuously progressing with the technological advancements. Moreover, we argued that hybrid analysis systems can detect mobile botnet binaries more accurately. Consequently, mobile botnet problem may supersede traditional PC based botnets if the precautionary measures have not been devised timely. Overall the paper objectives are summarized as follows:

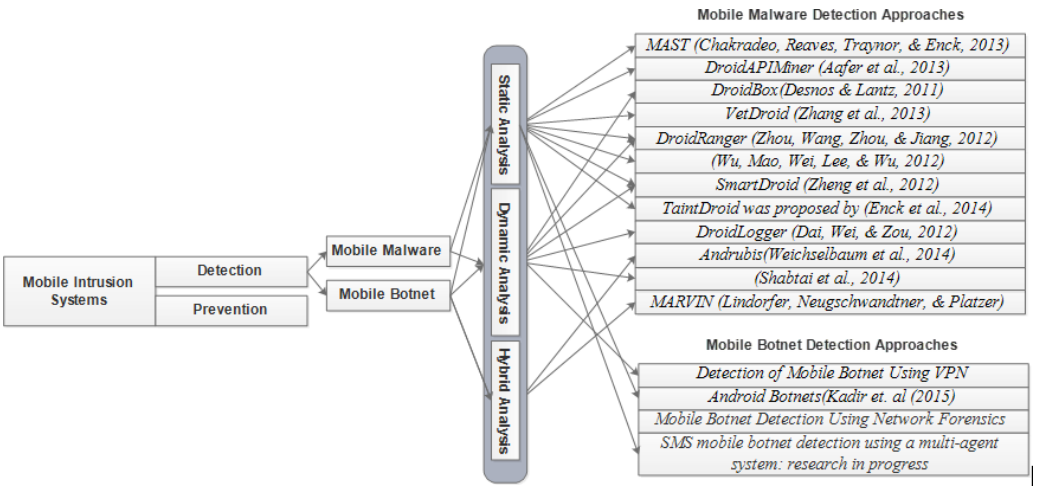
- To strengthen the existing efforts (mobile malware detection in general) towards mobile botnet (C&C) detection by identifying, highlighting and comparing the properties of known mobile botnets with existing malware and benign applications;
- We have divided our hybrid analysis framework into three different categories for mobile applications which includes botnet, malware and benign set of applications. Through this setup, we need to observe the most inherent properties and ongoing trends of mobile applications having botnet capabilities;
- Our claim of a hybrid analysis is more efficient to detect mobile botnet malware is verified by machine learning approaches;
- Conclusively, we have provided some guideline steps to avoid this growing hazard.

The rest of the paper is structured as follows: section 2 discusses related work, section 3 describes our proposed hybrid analysis framework, section 4 present evaluation results generated from code and runtime analysis, section 5 discuss some guidelines to avoid this growing hazard (mobile botnet), and conclude with section 6.

RELATED WORK

Mobile intrusion detection systems fall into the categories of detection and prevention as depicted in Figure 1. Overall mobile malware detection is a broader term used to identify the anomalous

Figure 1. Mobile malware detection approaches



behavior of apps with respect to benign apps. However, the motivation behind malware programmers is diverse, i.e., to gain illegitimate access, hinder the resource utilization, ransom, root exploit, spam dissemination and botnet. Therefore, in this article, we are specifically discussing mobile botnet phenomenon which is getting popularity in recent times for intrusive personals. By making a single device as a part of a bot network, the controller (C&C) can initiate various other types of attacks as well. As the mobile botnet phenomena are still not as popular as is predecessor (PC-based botnet), therefore the major work is carried out to detect mobile malware in general. However, recent reports state that the mobile botnet is also growing at the same pace as technology evolves.

Android permissions are considered as one the major entry point to breach smartphone security. Malware writers are often used dangerous permission set to exploit device by getting the benefit of users unawareness about the complexities associated with such dangerous permissions. In (Aswini & Vinod, 2014) authors, presented a static analysis model to mine prominent permissions that are of interest for malware writers. After extracting a feature vector from AndroidManifest.xml of 436 Android applications the future pruning method is applied to validate the accuracy of features. However, the approach is unable to extract detailed code features which result in greater false positives.

A mobile botnet detection approach using VPN (virtual private networks) is presented in (Choi, Choi, & Cho, 2013). The study revealed communication flow characteristics by investigating C&C communication flow over a VPN. The characteristics observed are a total number of bytes and the total number of packets. The basic theme of this research is to gain insight of a particular C&C traffic flow by comparing it with existing abnormal models, predefined signatures or whitelists. Whereas,

DeDroid (Karim, Salleh, Khan, et al., 2016) can detect zero-day attacks through static analysis of Android applications.

DroidRanger (Liang & Du, 2014), a hybrid analysis approach in which the app binaries are shortlisted based on the usage of dangerous permission set. Further, the malicious behavior is compared with existing malware's behavior by looking into the manifest, packages, function call graphs (FCG) and code sequence. Moreover, applications without a trusted signature are treated as zero-day attacks and selected for subsequent analysis. Nevertheless, the approach does not support to analyze network specific communication parameters. VetDroid (Pravin, 2015) is treated as dynamic analysis approach in which applications are scrutinized based on harmful permissions usage. The permission analysis component of VetDroid extracts all permissions an application is using and sketch a relationship among them. Consequently, system generates FCG to classify malicious applications. Further, each application is executed for a certain amount of time in the sandbox. The authors used monkeyrunner (Android, 2012) for triggering UI events. A sandbox known as DroidBox is proposed by Lantz (Desnos & Lantz, 2014) for behavioral analysis of applications for effective runtime analysis of applications. Similarly, it provides rich parameter support for dynamic analysis including network traces. One shortcoming of DroidBox is that it is not compatible with Android applications before version 4.2. We are also using DroidBox in our work to generate runtime analysis reports.

Another approach (Su & Fung, 2016) proposed a detection framework based on static analysis to detect malicious Android binaries. The features selected for evaluation are permissions, vulnerable functions applied by applications, intents, and native permissions. Moreover, the system validates the intrusive nature of applications by observing dynamic behavior through running and capturing runtime traces of applications. Further, the study highlighted the importance of short message (SM) in the detection of malicious intent. A more recent work on the detection of the mobile botnet is presented in (da Costa, Barbon, Miani, Rodrigues, & Zarpelão, 2017). The authors proposed an anomaly-based approach working on the host machine. They used machine learning classification algorithms to identify anomalous behaviors in Android applications using system calls as a feature vector. The approach uses dynamic properties of known malware (self-generated) and machine learning to detect botnet behavior. The said approach can detect malicious binaries with 92% of accuracy while relying only on dynamic features. However, we are analyzing both static and dynamic properties of botnet binaries. Moreover, in our hybrid analysis framework, Random Forest algorithm can detect more accurate results by achieving 98% of accuracy.

In a recent study (Zhi et al., 2017) the authors studied network traffic flow for botnets using Venn-abers predictor, K-nearest neighbor (KNN) and Kernel Density Estimation (KDE). The authors focused HTTP, IRC and P2P based botnet families for evaluation. Botrevealer (Khoshhalpour & Shahriari, 2018) is a behavior-based botnet detection framework based on botnet life cycle that analyzes network activities. Whereas, our proposed framework is based on hybrid analysis of android applications. A more recent work (Kirubavathi & Anitha, 2018) proposed structural analysis framework that can identify botnet applications from benign applications using machine learning classifiers. Contrary to this approach we are dealing with hybrid features of android applications using machine learning algorithms to detect C&C based Android applications.

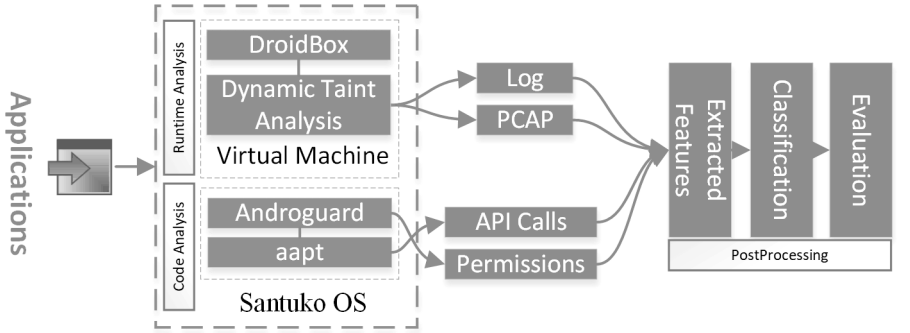
PROPOSED FRAMEWORK

As described above that currently Android is the most prevalent operating system for malware program authors. Therefore, we have focused Android platform in our analysis experimentation. To carry out this analysis task, we have taken (McIlroy, Ali, & Hassan, 2016; Parkour, 2011) ten samples of mobile applications from each of the categories (benign, malware, botnet). We have performed analysis using hybrid analysis approach, i.e., by combining static/code and dynamic/runtime analysis techniques. From code analysis, we will highlight various properties related to static code observation, which includes Permissions, API calls, Intents and hash values. Whereas, through run-time analysis, we

will run all thirty applications in a secure sandbox and log information during their execution. The properties we will observe during execution of applications are: opened connections, DNS queries, started services, HTTP request/response queries. After storing all feature values in comma separated files, we will then analyze and discuss the results of these properties. Finally, we will conclude the paper with our findings that applications having botnet intension has certain features which can be distinguishable. As a result, we should devise some proactive mechanisms to avoid such harmful threat.

We have carried out our experimentations on SANTOKU (a Linux distribution) which is specifically designed for task related to mobile malware analysis (NowSecure, 2015). For evaluation, Intel Xeon ® server (with 3.50GHz processing speed and 16GB of RAM) is deployed. The basic architecture of our analysis setup is depicted in Figure 2.

Figure 2. Proposed framework



Initially, applications are passed through static and dynamic analysis and stores respective log and trace files. For static analysis, Androguard (Desnos, 2011) tool is used, whereas dynamic analysis is performed with open source DroidBox (Desnos & Lantz, 2014) tool. Next, the static features (Ahmad Karim & Shah, 2015) and dynamic features (Karim, Salleh, & Khan, 2016) are extracted and labeled according to input class. Lastly, the results obtained from the hybrid analysis are evaluated and compared.

DATA ACQUISITION

As discussed above we have chosen ten mobile application samples from each category. To carry out our analysis task, we have downloaded those samples from online repositories including google play-store (Viennot, Garcia, & Nieh, 2014), contagion (Parkour, 2011) malware repository and well-known Drebin(Arp et al., 2013) dataset.

At the time of writing this article, it was the largest dataset publicly available through educational credentials. The dataset used for evaluation is described in Table 1.

PROGRAM CODE ANALYSIS

Usually, there are two approaches used in malware analysis: one is static/code analysis and the other is dynamic/behavior analysis. Through code analysis, an application is investigated through inspecting the downloaded mobile app's program code only. Signature-based systems mainly adopted by antivirus companies' falls in the category of static analysis. However, malware writers use obfuscation techniques to hinder the inspection of program code. Apart from various off-the-shelf obfuscation

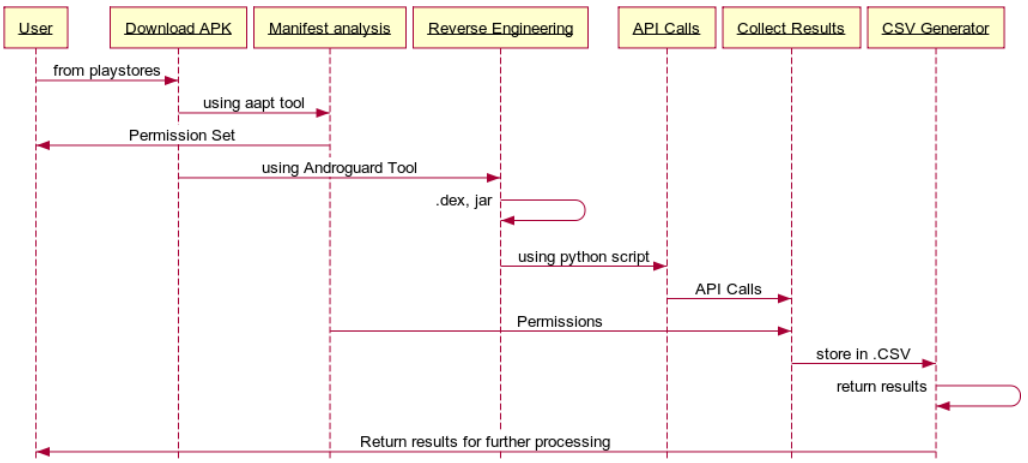
Table 1. Sample dataset used

Sample	Apps	Repository	Observation	Category	Family
Botnet	10	Contagio	Code/ Runtime	HTTP/ SMS-based	NotCompatible.C, Obad, SpamSold, Plangton, Geinime, DroidDream
Malware	10	Drebin/ Contagio	Code/ Runtime	Banking Trojan/ Information stealing / Premium SMS/self-installed binaries etc.	Foncy, Opfake, FakeNotify, Hippo, SMSreg
Benign	10	Google Play store	Code/ Runtime	Games, Entertainment, Web Browser, Wall-papers, GPS Tracking	-

methods available, malware programmers also utilize Native API calls by hiding system activities and calling the functions from outside the Java/Runtime library. As a result, our aim in this step is to apply different forensic techniques on a binary program of application and gather results for comparative analysis. The popular forensic techniques involved in static code analysis are reverse engineering, de-compilation, pattern matching, decryption and analysis of system calls. Using these techniques, the common thing is that the program is not executed at all.

The sequence diagram in Figure 3 represents the working flow of our code analysis method to extract and compare interesting features with malware, botnet, and benign binaries.

Figure 3. Code analysis workflow



The features selected for code analysis includes permission calls, and API calls which are already proven as interesting features of mobile botnet applications (Karim, Salleh, Khan, et al., 2016).

Therefore, we have selected permission calls, and functions call extracted from Manifest and .dex class respectively with the help of Androguard (Desnos) utility. To extract features automatically, we have executed a python script on all applications and stored all features on a CSV files. The contents of CSV files are binary numbers, i.e., stored "1" if the application has the particular feature enabled and "0" otherwise. SLet x and y be the number of applications and the set of features (included permissions and API calls, respectively. The feature vector for application i is $\left(a_{i,1}, a_{i,2}, a_{i,3}, \ldots, a_{i,y}\right)$ where:

$$a_{i,j} = \begin{cases} 1, & if\ application\ x\ uses\ feature\ k \\ 0, & otherwise \end{cases} \quad (1)$$

Similarly, we suppose class of an instance in the generated dataset is $c_i \in \{benign, malicious, botnet\}$ which shows the class of an application *i*. Then:

$$P\left(c_i \mid a_i\right) = P\left(c_i \mid a_{i,1}, a_{i,2}, a_{i,3}, \ldots, a_{i,y}\right) = \prod_{k=1}^{y} P\left(c_i \mid a_{i,k}\right) \quad (2)$$

Selected Feature Vector in Code Analysis

We can classify botnet working states into different phases depending upon their action type. These states are classified as (a) connection phase (b) communication phase and (c) status information phase which are shown in Table 2.

Table 2. Feature vector (permissions and API calls)

Phase	Permissions	API Calls
Connection Phase	INTERNET	openConnection(), execute(), connect(), openStream(), getInputStream(), Socket(), getContent()
Status Information Phase	READ PHONE STATE	getDeviceId(), getSimSerialNumber(), getSubscriberId(), getLine1Number()
	ACCESS NETWORK STATE	getActiveNetworkInfo(), getWifiState(),
	ACCESS WIFI STATE	getConnectionInfo(), getNetworkInfo()
	ACCESS COARSE LOCATION	getCellLocation()
	ACCESS FINE LOCATION	getLastKnownLocation(), isProviderEnabled(), requestLocationUpdates()
	READ CONTACTS	openOutputStream(), openInputStream(), openFileDescriptor()
	READ LOGS	exec()
Communication Phase	SEND SMS	getDefault(), sendTextMessage()

During connection phase, botnet applications try to establish a connection with the remote host. The permission and API calls for this phase include INTERNET, openConnection(), execute(), openStream(), etc. The bulk of negotiation and interactions happened once the connection is established. These interactions can be classified as (a) Direction of the Information (b) Communication Protocol. According to (Gu, Zhang, & Lee, 2008), the direction of C&C communication messages can be characterized as pull based or push based. In pull-based C&C mechanism, the bots periodically request for information in passive mode. In Contrast, for push-style C&C mode, bots explicitly send request messages to the server. Another important feature to recognize C&C communication patters is communication protocol. The most commonly used protocols in botnet communication are HTTP, IRC, and P2P. Moreover, communication phase causes initiation of malicious activity as well. Therefore, we have highlighted only those features which can cause communication and attack commencement. For

this purpose the permissions include: READ_CONTACTS, READ_LOGS and SEND_SMS whereas the API calls includes: openInputStream, openOutputStream, openFileDescriptor,exec, getDefault and sendTextMessage. To retain and expand attack vector, the attacker should constantly observe the active status of the device and changing network conditions. For this purpose, the following features are of interest for malware writers to actively monitor device status and network state, READ_PHONE_STATE, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, getDeviceId, getLine1Number, getSubscriberId, getSimSerialNumber, getDeviceSoftwareVersion, getLastKnownLocation, requestLocationUpdates, isProviderEnabled, getActiveNetworkInfo, getNetworkInfo, getConnectionInfo, getWifiState, getCellLocation.

BEHAVIOR/RUNTIME ANALYSIS

Behavior analysis requires the application to be processed at run-time. By this analysis, we have to detect and capture the system's execution states including communication patterns, SMS messages, HTTP traffic on network interfaces, network access to outside world and DNS requests. Behavior or dynamic analysis overcomes all possible problems that can incur during code analysis such as obfuscation, encrypted or confused communication and native API calls. This kind of analysis framework requires secure operational environment by sandboxing, virtual devices, or Cloud-based models (Chang, 2015) to gather the runtime execution traces by simulating and executing the application binaries. While certain types of malware are recognized using code analysis, many other kinds of malware can reliably be detected at looking the runtime execution traces. For instance, newly deployed malware may not have signatures introduced while they may be widely separated.

One of the major problems with the dynamic analysis is scalable solutions. For large datasets, it considered a most time consuming and difficult task to gather traces from run-time analysis due to various resource constraints. Consequently, to overcome this problem cloud-based systems (Chang, 2015; VirusTotal; Weichselbaum et al., 2014) are available for the research community to execute their apps in sandboxes (controlled environment) with maximum available resources which are sometimes difficult for an individual researcher to manage. However, for our behavioral analysis task, as we are not dealing with huge dataset, therefore, we have chosen the well-known sandbox known as DroidBox (Desnos & Lantz, 2011) which provides enough logging capabilities useful for identification of malicious intention. DroidBox also provides a comprehensive picture of the applications runtime activities provided by logging all the transmitted/accessed data of a particular application. The activities include, data read from or written to files, SMS messages sent and received, HTTP traffic initiated to/from, DNS queries, dataset or received over the network and much more.

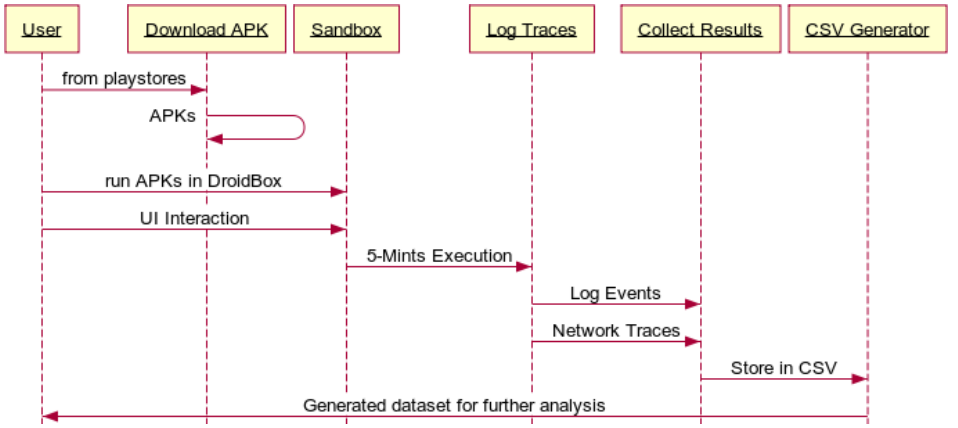
The steps we have taken to perform behavioral analysis are shown in Figure 4. Suppose, *A* is an instance of app comprising of, *n* number of applications and *m* number of features, then the runtime analysis collects all application with the following formula:

$$A = \left\{ app \mid app_i \ni \left| f_j \right| \ \forall \begin{matrix} i = 1,2,3,\ldots,n \\ j = 1,2,3,\ldots,m \end{matrix} \right\} \tag{1}$$

where:

n = total number of applications
m = total number of features

Figure 4. Behavior analysis workflow



Feature Vector for Behavioral Analysis

The features we have selected for behavioral analysis can be characterized as file activities, network operations, Information Leaks, Services, SMS operations, Cryptographic Operations, DNS Traffic, HTTP Traffic, unknown Conversations, which are listed in Table 3.

Table 3. Feature vector (runtime-analysis)

Features	Parameters
File Activity	Read/Write
Network Operations	Opened connections, Network Reads, Network Writes
Information Leaks	File Leaks, Data Leaks
Services	Started Services
SMS	Sent SMS
DNS Traffic	DNS Requests
HTTP Traffic	HTTP Conversations, HTTP Connection attempts

EVALUATION

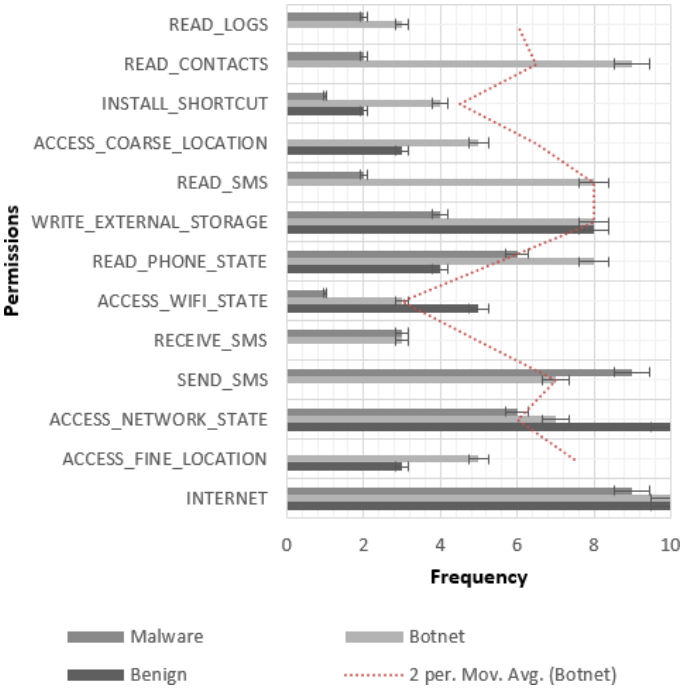
This section will evaluate static and dynamic analysis features among existing botnet, malware, and benign applications to highlight interesting trends in botnet applications.

Observations From Code Analysis

Malware families and benign application have different tendencies for requesting permissions. Malware generally requests more permissions than benign applications or even can request that permission which has risk associated with them. The risk factor provided with dangerous permissions is usually not descriptive to users at the time of installing an application. Therefore, user unknowingly installs such applications which have high risk related to them.

Figure 5 shows the average permissions used by benign, botnet and malware applications. The figure indicates that botnet applications are more frequently used feature vector specified in Table 2 than benign and malware applications.

Figure 5. Permission frequency analysis



The most prominent permissions used by botnet applications are INTERNET, ACCESS NETWORK STATE, READ PHONE STATE, READ SMS, READ CONTACTS, INSTALL SHORTCUT, and ACCESS COARSE LOCATION. These permissions are used by botnet applications to establish a remote connection with C&C by observing current status of device and network condition. Moreover, we have observed that botnet applications are using HTTP protocol as a medium of communication. Therefore, it is evident from the figure that on average SEND SMS permission is crucial for malware applications. Another interesting factor can be drawn from the figure is that on average benign application uses ACCESS WIFI STATE higher than botnet applications. Recently, WIFI has become the most widely used network access technology. Therefore, recent applications use this service more frequently. The Similar, trend is shown in a comparison of ACCESS WIFI STATE permission because our benign dataset is recently downloaded from official Google play store, whereas most botnet applications (60%) in our dataset are introduced during 2011-2017.

Another most valuable property which is exploited by boot master is to identify the active state of the mobile phone. Through observation of current status, the intruders can negotiate with bot clients and can instruct them accordingly. The same trend is observed that on average 80% of botnet applications utilize READ PHONE STATE permission than malware (60%) or benign (40%) applications.

The basic aim of a botnet initiator is not only to launch a botnet attack but also to expand its bot network. For this reason, botmaster tries to capture a broad audience through reading the contact list of affected mobile device and propagating the malware code to those contact persons. Consequently, on average 90% of botnet applications uses READ CONTACTS permission. In

contrast, on average 20% malware and 0% benign applications use READ CONTACTS permission. Similarly, 80% of applications exploit READ SMS in comparison with 20% and 0% of malware and benign applications respectively.

We have also examined API calls in order to analyze runtime execution traces. The usage frequency of API calls on botnet, benign and malware applications is depicted in Figure 6. The

Figure 6. API call frequency analysis

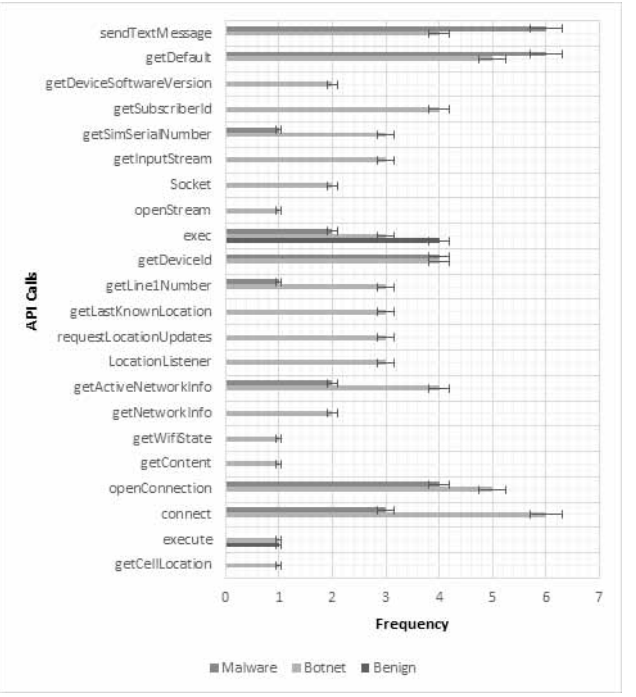


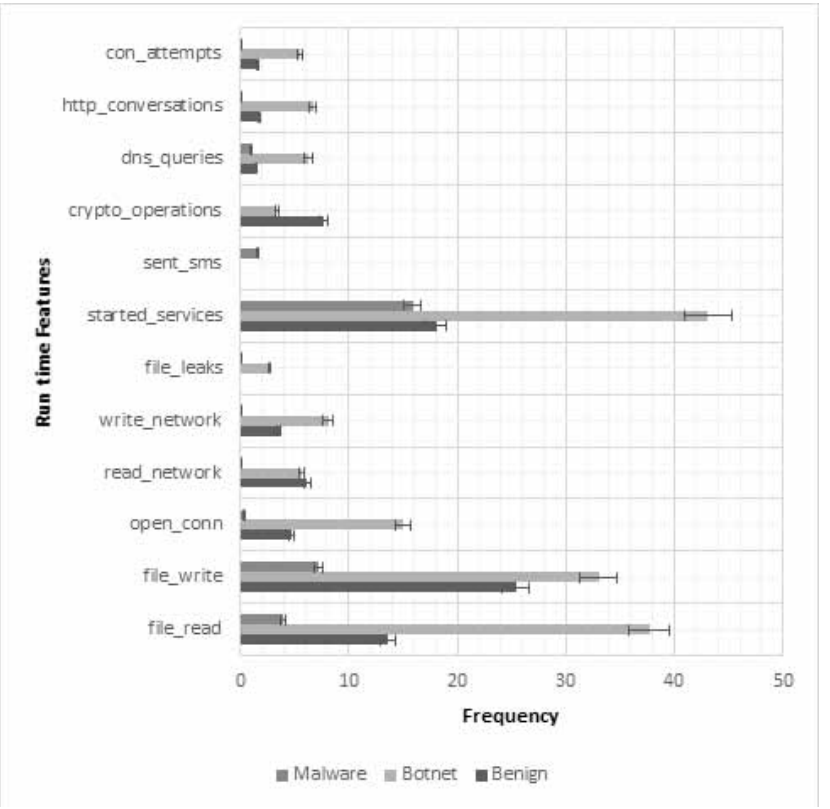
figure clearly indicates that botnet applications have frequent access to botnet specific API calls such as for connection establishment, connect () and openConnection() functions are called. Whereas, for continuous connection status and to gather network information, botnet apps use getNetworkInfo(), getConnectionInfo(), getConnectionInfo(), getNetworkInfor(), locationListener(), getLastKnownLocation(), getActiveNetworkInfo(), locationListener(), requestLocationUpdates(), getLine1Number() API calls.

To send/receive malicious commands to/from a particular bot, botmaster requires the unique identification of a device. For this purpose botnet application uses getDeviceID() and getSubscriberID() functions. We can observe the similar frequency of access for getDeviceID() function in both botnet and malware samples, i.e., on average 57% usage for each category. Whereas getSubscriberID() function is utilized on average 57% of botnet applications. In addition to that, as we have discussed above that mostly malware samples (60%) belongs to SMS Trojan. Therefore, we can observe the higher usage pattern (on average 90%) of sendTextMessage() function than in benign or botnet samples.

Observations From Run-Time Analysis

In this subsection, we will discuss the results obtained from runtime analysis of botnet, malware, and benign applications. Figure 7 shows the results generated from runtime analysis of botnet, benign

Figure 7. Features call frequency during runtime analysis



and malware applications. The results clearly indicate the high upward trend in botnet applications with the selected feature vector.

Android applications work under various service models to start background services. Malware programmers deployed this model to establish communication pathway among C&C and mobile device. Further, the personal information is acquired and sends this information to an adversary. From the figure, it can be depicted that, on average botnet, each application has initiated 43 background services, whereas malware and benign application initiated at an average of 15 and 18 services.

Android applications access internal and external storage of mobile devices to store and retrieve installation contents. Botnet applications access external storage to store execution code and initiate it on demand. According to the research conducted by (Weichselbaum et al., 2014) that 96% and 95% of malicious application access files for reading and writing respectively. In over findings, an overall upward trend is noticed for file reading and writing by botnet applications as well. On average each botnet application calls file_read 38 times and file_write 33 times.

Similarly, from opened connections, we can observe what communication channels are being used by botnet applications. For this purpose, we noticed that on average botnet applications opened 15 network connections mostly belongs to HTTP or HTTPS protocols, which clearly indicates malicious intention. In contrast, on average malware and benign applications opened one and four connections, respectively. In addition to that, the downward trend for opening a network connection in malware samples confirms our dataset choice as mostly malware samples belong to SMS Trojans that requires SMS as a communication channel rather than HTTP/s protocol to coordinate with C&C.

As we are targeting HTTP based mobile botnets, therefore, we also need to identify HTTP conversations. Through HTTP conversation we can notice, (a) actually what information is passing through the channel (b) what are the source and destination addresses (c) time stamp of each transaction. From the above figure, on average seven requests are made by botnet applications, in contrast, benign application on average call two times this feature.

One of the most important protocols used by botnet applications is DNS protocol. From the previous generation (PC Based) of botnets, it is proved that there are various parameters in DNS queries through which we can predict any botnet activity. Some of them includes: (a) failed DNS queries reveal botnet structure (b) DNS requests to backlist IP addresses (c) various suspicious DNS requests produced as round-robin DNS (c) DNS requests having short TTL for C&C domains (d) similar communication patterns, i.e., ratio of numerical characters in the DNS names. Based on these properties we can depict from the figure that, on average botnet, applications initiate 7 DNS requests. Whereas, a benign application on average calls one DNS request to resolve. The largest number of DNS queries that are resolved by the most recent mobile botnet NotCompatible.C which is 23 and among 4 of them are unsuccessful DNS queries. Moreover, we observed the similarity pattern in DNS names that are requested to resolve. Some of the queries are vhost240.181-4-8.telecom.net.ar, host26.186-126-172.telecom.net.ar, host105.181-4-113.telecom.net.ar, host30.186-127-146.telecom.net.ar, which clearly indicates a similar DNS query pattern.

Validation Through Machine Learning

In this section, we present validation of our proposed analysis platform to strengthen our claim that hybrid analysis is more effective to diagnose mobile malware having C&C features. The validation is performed through different machine learning classifier algorithms including J48, Naïve Bayes, Random Forest, and Logistic Regression. The j48 algorithm is typically applied on datasets to generate a pruned or unpruned classifier model which is based on decision trees algorithm logic. Similarly, another model which generates random trees to build a classifier model is called a random forest. Whereas Naïve Bayes algorithm derives prediction models based on independent assumptions. Moreover, logistic regression technique stems from the logistic model tree. In this paper, the motive behind the selection of diverse classifiers is to measure a higher level of accuracy. We have used 10-fold cross-validation on two different datasets. One of them is the dataset used in evaluation purpose (discussed above) and the second one comprises of 1371 botnet samples collected from (Kadir, Stakhanova, & Ghorbani, 2015) and 500 benign datasets collected from (Kang, Jang, Mohaisen, & Kim, 2014).

Table 4 presents the output of machine learning classifiers' performance on sample dataset which consists of 30 instances. The generated output comprises of correctly classified instances, incorrectly

Table 4. Classification on sample dataset

		Classifier Algorithms											
		J48			Random Forest			Logistic Regression			Naïve Bayes		
		Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)
Static Only	30	21	9	70	26	4	87	25	5	83	23	7	77
Dynamic Only	30	17	13	57	21	9	70	16	14	53	18	12	60
Hybrid	30	20	10	67	27	3	90	23	7	77	25	5	83

classified instances and accuracy in percentage against each classifier algorithm. The experiments involve one dataset of thirty botnet applications. Static-only classification experiment uses static features (as discussed above) of botnet applications, whereas dynamic-only experimentation takes dynamic features to pass through classifier models. Finally, hybrid experimentation involves the union of both static and dynamic features. The results indicate that the outcome of hybrid experimentation produced relatively good results other than static-only and dynamic-only experimentation. Similarly, based on accuracy results, random forest outperforms other classifier models with a classification accuracy of 90%, which is more than any other experimentation/classifier algorithm.

To validate our results on large-scale implementation we have applied the same classifier algorithm on Android botnet dataset. The results depicted from Table 5 confirm the viability of our hybrid

Table 5. Classification on android botnet dataset

		Classifier Algorithms											
		J48			Random Forest			Logistic Regression			Naïve Bayes		
		Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)	Correctly Classified	Incorrectly Classified	Accuracy (%)
Static Only	1871	1803	68	96	1825	46	98	1777	94	95	1622	249	87
Dynamic Only	1871	1685	186	90	1742	129	93	1562	309	84	1038	833	56
Hybrid	1871	1810	61	97	1827	44	98	1785	86	95	1621	250	87

analysis platform to detect botnet binaries on a large scale. Although all ML classifiers produced relatively good accuracy rates, however, simple Random Forest outperforms the other classifier algorithms. It correctly classifies 98% of Android botnet dataset using hybrid feature vector space to distinguish botnet applications. In difference, Naive Bayes, J48, and Logistic regression achieve an accuracy rate of 87%, 97%, and 95% respectively. However, the static-only experiment also produces almost the same accuracy result (fractionally lower).

GUIDELINES TO AVOID MOBILE BOTNET

The following guidelines are suggested to avoid such harmful hazard growing in mobile phone technologies:

- Google Play store provides advanced security mechanisms to keep app repository intact from malicious attempts. Therefore, it is recommended for users not to download apps from third-party application stores;
- Applications' permissions enable gateway to access smartphone resources. Therefore, permissions are considered as one of the interesting attributes for malware writers. The recommendation is to carefully look carefully requesting permissions by the app, for instance, if a weather application requests for contact list permission, then it seems to be fishy. Thus, the permissions should be skimmed before app installation. After installation, keep paying attention to the behavior of the app. Most often malicious application requests more permissions after their installation. Reputable security solutions for your mobile device will certainly protect your mobile phone from active threats;

- Similarly, make sure that your security software is up to date by installing latest patches and for software and firmware;
- The worldwide botnet risk is best defined by close national and international collaboration amongst governments and in fact with legislative and technically oriented organizations. For an efficient detection and mitigation mechanism to work, liaison between stakeholders must be reinforced and strengthened through political support, will, and negotiations.

CONCLUSION

In this paper, we have analyzed and identified the existence of botnet phenomenon in Android-based mobile applications. We have proposed a hybrid analysis framework which is divided into two different steps. During first step (code-based analysis), initially, we have highlighted all those permissions and API calls that can lead to a botnet activity. Further, we have extracted this permission from a dataset comprising of ten applications from each category: known-botnets, benign and malwares. Finally, we compare and evaluate the results and conclude that botnet applications have frequent access to botnet specific permissions and API calls in comparison with benign and malware applications.

Furthermore, as the second step of our analysis task, we have executed our selected dataset onto a sandbox (DroidBox) and collected the dynamic features from the log files. Among the various runtime analysis parameters, we have chosen and compared the most interesting parameters which can cause a botnet attack to expand exponentially or can persist for a long time. The results are then compared by applying machine learning classification models on both sample dataset and an Android Botnet dataset. The machine learning algorithms also affirm the viability of our framework by achieving 98% of accuracy in case of hybrid feature vector space.

Conclusively, botnet phenomenon is gaining popularity in the computationally intensive mobile environment. So, there is a need to devise some proactive mechanisms through which user should be aware of the consequences by unknowingly installing an application which has botnet capabilities. Ultimately, this can not only affect the overall performance of user's device, but also a user can naively become the part of a distributed attack.

REFERENCES

Android, S. D. K. (2012). *Tools: monkeyrunner*. Retrieved from https://www.android.com/

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2013). Drebin: Efficient and explainable detection of android malware in your pocket. *Paper presented at the Proc. of 17th Network and Distributed System Security Symposium, NDSS*.

Aswini, A. M., & Vinod, P. (2014). Droid permission miner: Mining prominent permissions for Android malware analysis. *Paper presented at the 2014 Fifth International Conference on Applications of Digital Information and Web Technologies (ICADIWT)*. Academic Press. doi:10.1109/ICADIWT.2014.6814679

Bilić, D.G. (2017). *Mobile security: The reality of malware … augmented.* We Live Security. Retrieved from https://www.welivesecurity.com/2017/03/10/mobile-security-the-reality-of-malware-augmented/

Chang, V. (2015). A cybernetics social cloud. *Journal of Systems and Software*.

Choi, B., Choi, S.-K., & Cho, K. (2013). Detection of mobile botnet using VPN. *Paper presented at the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS).* Academic Press. doi:10.1109/IMIS.2013.32

da Costa, V. G., Barbon, S., Miani, R. S., Rodrigues, J. J., & Zarpelão, B. B. (2017, May). Detecting mobile botnets through machine learning and system calls analysis. *Proceedings of the 2017 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE.

Desnos, A. (2011). Androguard: Reverse engineering, malware and goodware analysis of android applications... and more (ninja!).

Desnos, A. & Lantz, P. (2011). Droidbox: An android application sandbox for dynamic analysis.

Fereidooni, H., Conti, M., Yao, D., & Sperduti, A. (2016). ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. *Paper presented at the 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. Academic Press.

Gu, G., Zhang, J., & Lee, W. (2008). BotSniffer: Detecting botnet command and control channels in network traffic.

Kadir, A. F. A., Stakhanova, N., & Ghorbani, A. A. (2015). Android botnets: What urls are telling us. *Paper presented at the International Conference on Network and System Security*. Academic Press. doi:10.1007/978-3-319-25645-0_6

Kang, H. J., Jang, J. W., Mohaisen, A., & Kim, H. K. (2014). Androtracker: Creator information based android malware classification system. *Paper presented at the Information Security Applications-15th International Workshop, WISA*. Academic Press.

Karim, A., Salleh, R., & Khan, M. K. (2016). SMARTbot: A Behavioral Analysis Framework Augmented with Machine Learning to Identify Mobile Botnet Applications. *PLoS One*, *11*(3). doi:10.1371/journal.pone.0150077 PMID:26978523

Karim, A., Salleh, R., Khan, M. K., Siddiqa, A., & Choo, K.-K. R. (2016). On the analysis and detection of mobile botnet applications. *Journal of Universal Computer Science*, *22*(4), 567–588.

Karim, A., Salleh, R., & Shah, S. A. A. (2015, August). DeDroid: a mobile botnet detection approach based on static analysis. *Proceedings of the 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)* (pp. 1327-1332). IEEE.

Khoshhalpour, E., & Shahriari, H. R. (2018). BotRevealer: Behavioral Detection of Botnets based on Botnet Life-cycle. *The ISC International Journal of Information Security*, *10*(1), 55–61.

Kirubavathi, G., & Anitha, R. (2018). Structural analysis and detection of android botnets using machine learning techniques. *International Journal of Information Security*, *17*(2), 153–167. doi:10.1007/s10207-017-0363-3

Liang, S., & Du, X. (2014). Permission-combination-based scheme for android mobile malware detection. *Paper presented at the 2014 IEEE International Conference on Communications (ICC).* IEEE Press. doi:10.1109/ICC.2014.6883666

Lukan, D. (2012). *Reverse Engineering Tools*, Retrieved 8-9-2016, from http://resources.infosecinstitute.com/reverse-engineering-tools/

McIlroy, S., Ali, N., & Hassan, A. E. (2016). Fresh apps: An empirical study of frequently-updated mobile apps in the Google play store. *Empirical Software Engineering*, *21*(3), 1346–1370. doi:10.1007/s10664-015-9388-2

Millman, R. (2015). Updated: 97% of malicious mobile malware targets Android. *SC Magazine UK*. Retrieved from https://www.scmagazineuk.com/updated-97-of-malicious-mobile-malware-targets-android/article/535410/

NowSecure. (2015). *SANTOKU Operating System*. Retrieved from https://santoku-linux.com/

Parkour, M. (2011). *Contagio malware dump. blog sobre compartición de malware*. Contagiodump. Retrieved from http://contagiodump.blogspot.com/

Paul. (2017). Android Malware Doubled in 2016, Adding to Mobile Malware Problem. Security Ledger. Retrieved from https://securityledger.com/2017/03/android-malware-doubled-in-2016-adding-to-mobile-malware-problem/

Peng, C., Zhao, R.-C., Zheng, S., Xun, J., & Yan, L.-J. (2016). *Android Malware of Static Analysis Technology Based on Data Mining. DEStech Transactions on Computer Science and Engineering*. AICE-NCS.

Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., & Ioannidis, S. (2014). Rage against the virtual machine: hindering dynamic analysis of android malware. *Paper presented at the Seventh European Workshop on System Security*. Academic Press. doi:10.1145/2592791.2592796

Pravin, M. N. P. (2015). VetDroid: Analysis using permission for vetting undesirable behaviours in Android applications. *International Journal of Innovative and Emerging Research in Engineering*, *2*(3), 131–136.

Schmidt, A.-D. Bye, Rainer, Schmidt, H-G, Clausen, Jan, Kiraz, Osman, Yuksel, Kamer A, … Albayrak, Sahin. (2009). Static analysis of executables for collaborative malware detection on android. *Paper presented at the IEEE International Conference on* Communications *ICC'09.* IEEE Press.

Su, M.-Y., & Fung, K.-T. (2016). Detection of android malware by static analysis on permissions and sensitive functions. *Paper presented at the 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. Academic Press.

Unuchek, R., & Chebyshev, V. (2018). *Mobile malware evolution 2015*. AO Kapersky Lab.

Viennot, N., Garcia, E., & Nieh, J. (2014). A measurement study of Google Play. *Paper presented at the 2014 ACM international conference on Measurement and modeling of computer systems*.

VirusTotal. (2017) *VirusTotal*, Retrieved from https://www.virustotal.com/

Weafer, V. (2014). Threats Report. *McAfee Labs*.

Weafer, V. (2016). 2017 Threats Predictions. *McAfee Labs*.

Weichselbaum, L., Neugschwandtner, M., Lindorfer, M., Fratantonio, Y., van der Veen, V., & Platzer, C. (2014). Andrubis: Android malware under the magnifying glass. Vienna University of Technology.

Yang, M., Wang, S., Ling, Z., Liu, Y., & Ni, Z. (2017). Detection of malicious behavior in android apps through API calls and permission uses analysis. *Concurrency and Computation*, *29*(19), e4172. doi:10.1002/cpe.4172

Yousafzai, A., Gani, A., Noor, R. M., Naveed, A., Ahmad, R. W., & Chang, V. (2016). Computational offloading mechanism for native and android runtime based mobile applications. *Journal of Systems and Software*, *121*, 28–39. doi:10.1016/j.jss.2016.07.043

Wang, Z., Gao, H.-Z., Zhang, Y.-M., Hu, Yu-C., Qiu, K.-F., Cheng, X., & Jia, C.-F. (2017). Fortifying Botnet Classification based on Venn-abers Prediction. DEStech Transactions on Computer Science and Engineering Teaching and Research

Ahmad Karim received the M.S. degree (Hons.) in CS from Bahauddin Zakariya University, Multan, Pakistan, and the Ph.D. degree in computer security from the University of Malaya, Malaysia. He is currently a Lecturer with the Department of Information Technology, Bahauddin Zakariya University. His areas of research include mobile botnet detection, computer security, computer networks, software-defined networks, big-data analytics, and Internet of Things. He received Cisco International Certifications (CCNA, CCNP, and CCAI).

Victor Chang is an Associate Professor (Reader) and Director of PhD at IBSS, Xi'an Jiaotong-Liverpool University, Suzhou, China, after working as a Senior Lecturer at Leeds Beckett University, UK, for 3.5 years. Within 4 years, he completed a Ph.D. (CS, Southampton) and a PGCert (Higher Education, Fellow, Greenwich) while working for several projects at the same time. Before becoming an academic, he has achieved 97% on average in 27 IT certifications. He won a European Award on Cloud Migration in 2011, IEEE Outstanding Service Award in 2015, best papers in 2012 and 2015, the 2016 European award: Best Project in Research, Outstanding Young Scientist award in 2017 and numerous awards since 2012. He is widely regarded as a leading expert on Big Data/Cloud/IoT/ security. He is a visiting scholar/PhD examiner at several universities, an Editor-in-Chief of IJOCI & OJBD journals, Editor of FGCS, Associate Editor of TII, founding chair of two international workshops and founding Conference Chair of IoTBDS http://www.iotbd.org and COMPLEXIS http://www.complexis.org since 2016. He was involved in different projects worth more than £12.5 million in Europe and Asia. He has published 3 books as the sole author and the editor of 4 books on Cloud computing and related technologies. He has given 16 keynote speeches at international conferences.

Ahmad Firdaus has distinctively received his Master of Computer Science (Networking) from the University Teknologi Mara, Malaysia. He obtained his PhD from the Faculty of Computer Science and Information Technology at the University of Malaya, Kuala Lumpur. He is a senior lecturer at the Faculty of Computer Systems and Software Engineering at Universiti Malaysia Pahang, Gambang, Kuantan, Pahang. His area of research includes mobile security, Android, malware static analysis, and Blockchain technology.