Journal of Universal Computer Science, vol. 22, no. 4 (2016), 567-588
submitted: 1/10/15, accepted: 30/3/16, appeared: 1/4/16 © J.UCS

On the Analysis and Detection of Mobile Botnet Applications[1]

Ahmad Karim
(University of Malaya, Kuala Lumpur, Malaysia
ahmadkarim@um.edu.my)
(Bahauddin Zakariya University, Multan, Pakistan
ahmadkarim@bzu.edu.pk)

Rosli Salleh
(University of Malaya, Kuala Lumpur, Malaysia
rosli_salleh@um.edu.my)

Muhammad Khurram Khan
(King Saud University, Riyadh, Saudi Arabia
mkhurram@ksu.edu.sa)

Aisha Siddiqa
(University of Malaya, Kuala Lumpur, Malaysia
aasiddiqa@gmail.com)

Kim-Kwang Raymond Choo
(University of South Australia, Adelaide, Australia
raymond.choo@fulbrightmail.org)

Abstract: Mobile botnet phenomenon is gaining popularity among malware writers in order to exploit vulnerabilities in smartphones. In particular, mobile botnets enable illegal access to a victim's smartphone, can compromise critical user data and launch a DDoS attack through Command and Control (C&C). In this article, we propose a static analysis approach, DeDroid, to investigate botnet-specific properties that can be used to detect mobile applications with botnet intensions. Initially, we identify critical features by observing code behavior of the few known malware binaries having C&C features. Then, we compare the identified features with the malicious and benign applications of Drebin dataset. The results show against the comparative analysis that, Drebin dataset has 35% malicious applications which qualify as botnets. Upon closer examination, 90% of the potential botnets are confirmed as botnets. Similarly, for comparative analysis against benign applications having C&C features, DeDroid has achieved adequate detection accuracy. In addition, DeDroid has achieved high accuracy with negligible false positive rate while making decision for state-of-the-art malicious applications.

Keywords: Mobile Botnet, Botnet Detection, Malware, Botware, Mobile malware detection.
Categories: D.4.6, K.6.5, L.4

[1] This version extends our previous work (50% extension), which appeared in (UFirst2015), The 7th IEEE International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies August 10-14, 2015, Beijing, China.

1 Introduction

While open source Android OS has benefited mobile application (app) developers, malware writers have also exploited the open source nature to target such devices [Karim, Shah et al. 2014, Damshenas, Dehghantanha et al. 2015]. For example, [Hyppönen 2013] states that, more than 97% of mobile malware families are targeting Android operating systems. Estimations from antivirus (AV) vendors states that, Android malware is most rapidly evolving with diverse application logic. As an example, Sophos gathered in total of 650,000 distinct malware binaries, with everyday discovery of 2K new malware samples[Neugschwandtner, Lindorfer et al. 2013]. In addition to that, MacAfee has reported over 700K distinct mobile malware samples in the first quarter of 2014 [Weafer 2014] alone. A recent report [Data 2015] states that, Internet access on Android based smartphones and tablets has exceeded 61% in the Q1 2015. Consequently, almost 60.85% of worldwide Android users have started using Internet on thir cell phones. The similar growth shown in malware programs, as 40,267 new malware variants are identified and analyzed by the security experts at the end of Q1 2015. Another report [Shea 2015] states that, this mobile malware progression is three times more than that of found in previous quarter i.e Q4 2014. Moreover, 97% of mobile malware targeted Android platform[Millman 2015].

One common category of malware targeting mobile devices is bot malware. Similar to "traditional" botnets, mobile devices compromised by bot malware will be part of a botnet to carry out coordinated attacks upon the instructions of a botmaster, usually via a command and control (C&C) server [Choo 2007]. Such compromised devices can then be used to carry out distributed denial of service (DDoS) attacks and facilitate other cybercriminal activities, such as making premium number phone calls, sending of emails and text messages to others on the device's contact lists which contain a hostile payload (that looks like it is being sent from someone they trust; thus, infecting more devices and extending the reach of the botnet).

The first mobile bot malware, Yxes, targeted Symbian devices. Yxes was designed to collect private user information prior to sending the information to a C&C server under the remote control of the attacker. Currently, there are a large number of cross-platform mobile bot malware, such as ZeuS [IDC] which targets Android, Symbian, Blackberry and Windows devices, as well as bot malware that targets only specific devices (e.g. *NotCompatible.C* targets Android devices). Bot malware is getting more sophisticated. For example, to avoid the scrutiny of anti-malware companies, *NotCompatible.C* uses a peer-to-peer (P2P) C&C architecture. According to [Alcatel-Lucent 2015], *NotCompatible.C* is also the first Android bot malware to share a C&C infrastructure with a compromised Windows machine. Other examples of bot malware include IKee.B designed to scan IP addresses on iPhones, and TigerBot and BMaster designed to target Android application frameworks.

This is not surprising. Researchers [Choo 2011, Do, Martini et al. 2015, Nigam 2015, Walls and Choo 2015] [Karim, Shah et al. 2015] have noted that as the capabilities of smartphones and mobile devices become more powerful, cybercriminals will seek to compromise such devices (e.g. bot malware) in order to target data stored on such devices, etc. In addition, it has been observed that newer generations of bot malware uses techniques such as encryption, obfuscation and cryptographic functions to avoid detection. As a result, existing anti-malware

solutions are far from effectiveness. For example, in a recent systematic evaluation of ten popular free cloud-based anti-malware apps [Walls and Choo 2015], it was determined that:

> *"no single cloud anti-malware app can be solely relied upon to mitigate known malware. The findings were also concerning, particularly that malware threats are becoming more sophisticated and targeted, using various attack vectors to escalate permissions and exfiltrate data".*

In this paper, we propose a mobile bot malware detection approach (hereafter referred to as *DeDroid*), designed to effectively identify C&C communication patterns in Android apps. This is an extension of our previous work [Ahmad Karim 2015]. For this purpose, we study the properties of four known bot malware families, namely: DroidKungFu, Geinime, GoldDream, and Plangton. Then, we train our approach using 5,064 malware samples, a subset of the Drebin dataset [Daniel Arp], in our attempt to answer the following research questions:

1. What are the features of a mobile botnet which are critical in initiating and sustaining an attack?
2. How can we effectively detect bot malware characteristics in Android apps?
3. How do we implement the detection mechanism to provide real-time detection for large scale datasets?

Thus, the main aim of the research is to employ static analysis techniques for *botware* detection by identifying features that are most relevant to a botnet activity in smartphones. We define *botware* as a malware capable of communicating through C&C.

The rest of the paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 describe our proposed detection approach, and our research methodology, respectively. Section 5 presents our findings, and Section 6 concludes this paper.

2 Related Work

Mobile malware analysis tools can be broadly classified into two categories, namely: static analysis [Christodorescu and Jha 2006, Shabtai, Moskovitch et al. 2009] [Aswini and Vinod 2014] and dynamic analysis [Christodorescu, Jha et al. 2008, Shabtai, Tenenboim-Chekina et al. 2014]. However, majority of existing detection solutions are designed for mobile malware in general rather than mobile bot malware. The latter exhibits a somewhat different characteristic, due to the involvement of a C&C server (e.g. the need for the compromised device to "call home" to receive attack instructions). A cursory literature review suggests lack of studies focusing on identifying Android apps with bot malware capabilities.

[Aswini and Vinod 2014] proposed a static analysis approach to detect Android malware by mining prominent permissions from AndroidManifest.xml file. After extracting permissions from 436 Android package files, the feature pruning was applied to examine the accuracy with respect to the feature length. However, the

proposed approach is unable to deeply investigate the application code for possible malicious behaviour. Another mobile botnet detection approach is presented in [Choi, Choi et al. 2013]. The authors investigated anomalies by observing communication flow characteristics (total # of bytes, total # of packets) by passing C&C traffic over a secure virtual private network (VPN). The main feature of this approach is to detect mobile apps with potential bot malware characteristics by comparing traffic flow with abnormal models, whitelists or predefined signatures. However, this approach is not effective against zero-day attacks. In contrast, DeDroid can detect new malicious apps with bot malware characteristics via static analysis (see Section 5.6).

Recently, a hybrid multi-agent approach for the detection of SMS-based smartphone botnets was proposed in [Alzahrani and Ghorbani 2014]. The technique implements security services by combining signature-based and anomaly based approaches. Detection is achieved by performing behavioural analysis and correlating malicious SMS messages with already generated user profiles. However, the technique is still in the development phase. This approach complements our approach, as we focus on HTTP-based mobile botnet apps.

3 De-Droid: An Overview

Currently, *DeDroid* focuses on static code analysis considering permissions and API calls. The static analysis provides a lightweight approach as compared to the dynamic analysis. However, malware programmers can use different evasion techniques like reflections, code obfuscations or by dynamic code loading at runtime in order to hinder or bypass static analysis process. This code is shipped with the app itself or can be downloaded from external sources. These evasion techniques are not only deployed by malware writers but also benign applications often use these methods to secure premium features, application upgrades, copyright protection and statistical testing. We are dealing with this situation by relying on entry level structural information (AndroidMenifest.xml) where code obfuscation and other evasion techniques are impossible to apply. Similarly, the standard Java API classes may not be obfuscated. However, binary code can be effected by these approaches. Moreover, inducing reflection reduces the overall smartphone performance.

In order to deal with the above mentioned issues, it is significant to deploy dynamic analysis approaches for large-scale evaluations. Moreover, dynamic analysis is able to acquire complete behaviour of an application otherwise missed by static code analysis. However, effective dynamic analysis systems require compute intensive resources, sandboxing and rich code coverage [Karim, Salleh et al. 2016]. Similarly, dynamic analysis systems can be defeated by malware writers by evading and detecting sandboxing environment.

The DeDroid analysis approach used in this paper is shown in Figure 1. The first step examines the C&C features associated with the four well-known malware families including DroidKungFu [Lookout 2011], Plankton [Svajcer 2011], GoldDream [Jiang 2011], and Geinimi [Strazzere and Wyatt 2011]. After taking 5 samples from each of malware family, a static analysis is performed by reverse engineering the applications. The feature set consists for permissions and API calls having close relation with botnet like features. As an example, the INTERNET permission is an elementary feature used to establish connection with outside world.

Similarly, the *sendTextMessage()* API call is suspicious that can send private information to remote host without user intervention. Along the same lines, we have highlighted forty potential botnet-specific features for static analysis.

After identifying critical *botware* features in known-botnet applications, the systems repeats the process for 5064 malicious samples and compare the trends for malicious applications using botnet features. As an additional step of our previous work [Ahmad Karim 2015], we have also compared the botnet features with 14685 benign application in order to prove our claim that many benign applications also have C&C features. A comparative analysis is then performed to validate our results, which we will discuss in section-5.

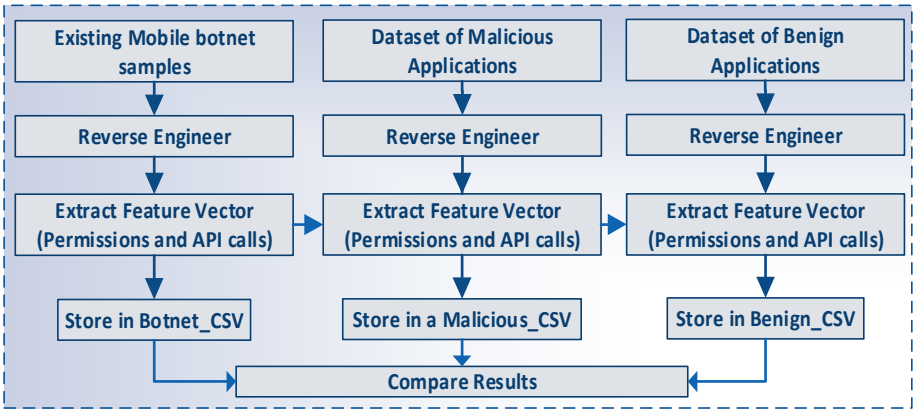

Figure 1: DeDroid System Overview

4 Methodology

Here, we describe the mobile botnet detection approach. We studied the architecture of four malware families which are known for their bot related malware activities. Moreover, we have taken five samples from each malware family, reverse engineered them and observed the behavior with respect to botnet C&C properties. Table 1 summarizes the properties of sample botnets.

Botnet Applications	**Year Introduced**	**C&C**	**Motivation**	**Propagation Technique**
DroidKungFu	2011	HTTP	Root exploits	Games
Plankton	2011	HTTP	Received commands from C&C and acted accordingly	Spam text messages
Geinimi	2011	HTTP/ SMS	Steal personal information	Games
GoldDream	2011	HTTP	Financial loss	Business Applications

Table 1: Summary of Mobile Botnets

4.1 Dataset Used

As a next step in DeDroid analysis, we took 5064 malicious binaries and 14865 benign samples from Drebin dataset. Drebin dataset is currently considered as the largest publically available dataset which was collected in the period of August 2010 to October 2012. Therefore, we have chosen Drebin dataset in order to measure the effectiveness of our analysis approach. Table 2 shows the total number of used samples and the length of the feature set.

Samples	***Source***	***# of Samples***	***Feature Set***
Botware	Drebin/Third Party	20	40
Malicious	Drebin	5064	40
Benign	Drebin	14865	40

Table 2: Dataset used

4.2 Botware Feature Selection

After manual inspection of 20 *botware* applications, we have observed the most important permissions and API calls which are of interest for *botware* writers. The permissions along-with their API calls and their rationale with respect to the botnet activity are mentioned in Table 3.

After identifying critical features related to botware applications, we have reverse engineered all malicious applications from Drebin dataset. As an outcome, we have gathered static features (permissions) from Manifest file and function calls (API Calls) from .dex class, using *Androguard* [Desnos 2011] tool. To accomplish this task automatically, we applied a python script to Android binary code, and stored all extracted features into a CSV file for further analysis. The values of CSV file are binary numbered such that, "1" refers to applications with enabled features and "0" for disabled features. Formally:

$$F_i = \begin{cases} 1, & feature\ enabled \\ 0, & otherwise \end{cases} \qquad (1)$$

As an example, below is the format of CSV file of the Plankton botware. The file starts with hash function of the application and ends with the sum of all enabled features. The values "1" and "0" corresponds to enabled and disabled features respectively. We use the sum of enabled features (i.e. 30) to further classify our malicious dataset with respect to botware applications. We have observed from all samples of botware applications that the maximum number of accumulated features used by any application is 30, whereas the minimum number is 18. Thus, for our analysis we use (18) as a threshold such that malicious application having accumulated sum less than or equal to 18 are classified as malware/benign and botware otherwise.

Permissions	API Calls	Rationale
INTERNET	getContent() openConnection() connect() execute() HttpResponse HttpUriRequest getInputStream() Socket;-><init> openStream()	Most network-connected Android apps use HTTP to send and receive data. Android includes two HTTP clients: (a) HttpURLConnection and (b) Apache HttpClient. Similarly, it can establish a remote connection and can execute commands accordingly. Moreover, TCP sockets can also be utilized to establish connections.
READ_ PHONE_ STATE	getDeviceId() getLine1Number() getDeviceId() getSimSerialNumber getSubscriberId() getDeviceSoftware Version	This is a read-only permission which is used to get information with respect to current phone state. This permission is crucial in a way that it can send identity and location information of the effected mobile device (bot) to C&C.
ACCESS_ NETWORK_ STATE	getActiveNetworkInfo() getNetworkInfo()	This feature is applied in conjunction with INTENET permission, and used to view the current status of the associated networks.
SEND_ SMS	getDefault() sendTextMessage()	Application uses this feature to send SMS message to C&C servers without user intervention.
ACCESS_ WIFI_ STATE	getConnectionInfo() getWifiState() isWifiEnabled()	To access information about Wi-Fi networks and send this information to remote site.
ACCESS_ COARSE_ LOCATION	getCellLocation()	This permission allows an app to access *estimated* location identified from some network location sources i.e WiFi.
ACCESS_ FINE_ LOCATION	getLastKnownLocation() isProviderEnabled() requestLocationUpdates()	This permission allows an application to retrieve a *precise* location from GPS, WIFI or cell towers.
READ_ CONTACTS	openOutputStream() openInputStream() openFileDescriptor()	This feature allows an application to read user's contact information. This information is then propagated to C&C to perform infection.
READ_ LOGS	exec()	This feature allows an application to access system log files.

Table 3: Selected Feature Set and Their Rationale

*Plankton (D0C35F26B94F67D9AF189D3050541EC7971A88858913E52A334480CEA4434085),<1,1,1,1,0,1,1,1,0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,1,**30**>*

Generally applications using more features pretend to have a malicious intension accompanied with them. Therefore, malware detection systems calculate score which is based on the accumulative number of static and dynamic features. For example, Andrubis' [Technology 2012] malice sore calculation depends upon the relationship between total number of features and accumulative value of static and dynamic features. Therefore, for DeDroid, minimum threshold value is crucial to separate C&C specific applications from the rest. For this purpose, we have observed applications in our training dataset and considered minimum accumulative value as threshold value.

Based on the above mentioned criteria, we have applied this logic to Drebin dataset and observed that out of 5064 malicious binaries 1795 binaries have C&C features, which are 35% of total malicious applications as shown in Figure 2. Similarly, the same logic has been applied to 14865 benign application set to strengthen our claim about the existence of bot behavior even in case of benign applications. The result for this comparative analysis which is shown in Figure 3, states that almost 8% of benign applications contain botware behavior.

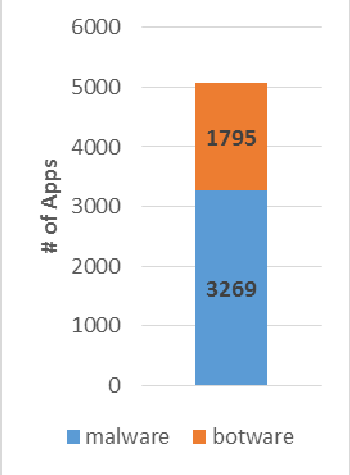

Figure 2: Botware vs Malware

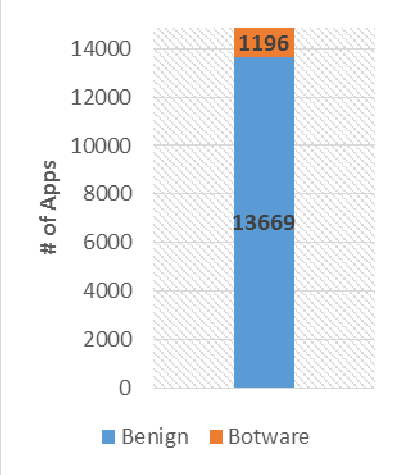

Figure 3: benign vs botware

4.3 Feature Extraction

For feature extraction, we applied the same python script on a dataset containing 5064 applications and reverse enginered each application. From Manifest.xml file we have extracted Permissions and API calls are collected from .dex file. After extracting the required features, the malicious.CSV file is generated. The steps are highlighted in Figure 4.

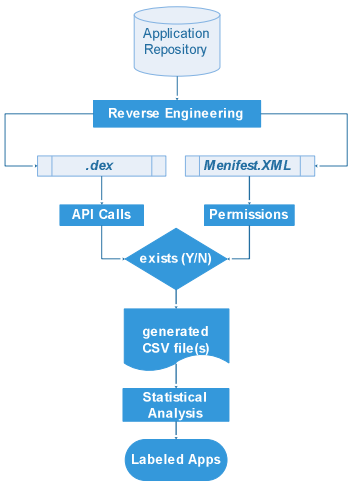

Figure 4: Feature extraction Process

As a proof of concept, we have analyzed the behavior of malicious as well as benign applications with respect to properties and features described in the previous section. Ultimately, the result of the evaluation process will confirm the applications with botnet motivations. As a matter of fact, Drebin dataset does not provide binaries for benign applications, however, it has provided rich feature set for each benign application. Further, we also have applied our python script to extract our required information which is related to permission requested and API calls from the benign dataset using regular expressions. Similar to as previous step, we have stored all gathered information to a CSV file for further analysis.

5 Evaluation

This section presents evaluation of experimental results and discusses them. Initially, we compare the attributes of trained dataset (botware dataset) with respect to malicious application set. The outcome of this method shows the number of applications having botnet behavior in a malicious dataset. Further, we have applied the same comparative analysis to 14865 benign application set to strengthen our claim about the existence of bot behavior even in case of benign applications. The ultimate aim of *DeDroid* analysis approach is to investigate the trends in malicious as well as benign applications with respect to botnet intensions.

5.1 Botware vs Malicious Dataset

Android security architecture heavily relies on permission-based system[Barrera, Kayacik et al. 2010, Felt, Greenwood et al. 2011]. At the time of writing, there are about 147 permission set available in Android platform that can allow access to various system and user resources. Whenever, a user installs some applications, he

would be prompted whether to allow these permissions prior to installation or not. However, in normal practice, users are unaware of the complexities and harmful affect associated with permissions which they are going to enable. The users should be given extra information to make correct decision.

The Figure 5 shows the percentage of permissions used by *botware* and malware applications. The figure clearly indicates that applications having C&C features most often have access to the botware permissions than other types of malicious applications. The permissions used by *botware* are *INTERNET, ACCESS_NETWORK_STATE, READ_PHONE_STATE,* and *ACCESS_FINE_LOCATION*. These permissions are used by *botware* applications to establish a remote connection and to persist those connections in order to observe state of the device and the network. Another interesting fact we have observed is that, as our training dataset consists of malware samples that belong to botnets having HTTP based C&C mechanism. However, 70% of the malware applications using *SEND_SMS* which come as no surprise because sending SMS to premium numbers is a popular method of mobile malware programmers [59]. In contrast, *botware* applications utilize 37% *SEND_SMS* permission to periodically update bots for new instructions. *Botware* applications try to utilize network connectivity to launch the attacks. For instance, in our observation, 82% of *botware* applications gain insight of WIFI state by initiating *ACCESS_WIFI_STATE* command. Whereas, only 26% malwares use this permission. Similarly, detecting the current state of cell phone is also an important point for *botware* programmers, in this way they can be well aware of the current status of the mobile device, if it is active then botmaster can start negotiating with cell phone. In our observation, 98% *botware* applications use *READ_PHONE_STATE* and 85% of malware applications using this permission for their malicious intensions. On the same lines, 77% *botware* uses *ACCESS_COARSE_LOCATION* to be aware about the Internet but only 12% other malicious applications use this permission.

In order to detect malicious code execution capabilities, *DeDroid* examines the API calls. Figure 6 shows the impact of malicious API calls on malware and *botware* applications. The results clearly indicate that botware applications most often have access to commands such as *execute(), connect()* and *openConnection()* in order to build and propagate bot network. Similarly, in order to get connected and to take network information of the devices, *botware* have used *getConnectionInfo(),getNetworkInfor(),getActiveNetworkInfo(),locationListener(), requestLocationUpdates(),getLastKnownLocation(),getLine1Number()andgetDeviceID()* API calls. In contrast, API calls having file transfer are least significant w.r.t *botware* and malware applications. Moreover, socket API is used 35% by *botware* applications, whereas usage for normal malicious applications is 6%. Another important feature for *botware* applications is to get bot identification information and send it to remote host. This can be done through the following API calls: getSimSerialNumber() and *getSubsriberID()*. *Botware* applications utilize *getSimSerialNumber()* and *getSubscriberID()*47% and 56% respectively. Whereas, normal malicious applications use these API calls 11% and 35% respectively. As we have already described, our training dataset is based on HTTP based botnets, therefore, *sendTextMessage()* is least utilized by *botware* applications as compared to

malware applications. For *sendTextMessage()* API, we have seen 33% usage by *botware* applications, in contrast to 45% in malwares.

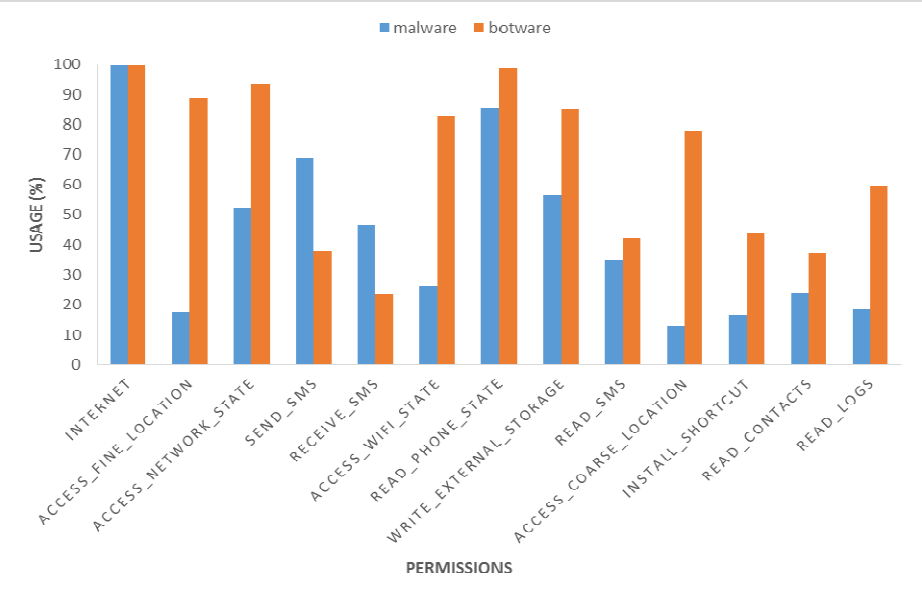

Figure 5: Permissions comparison between Botware and Malware

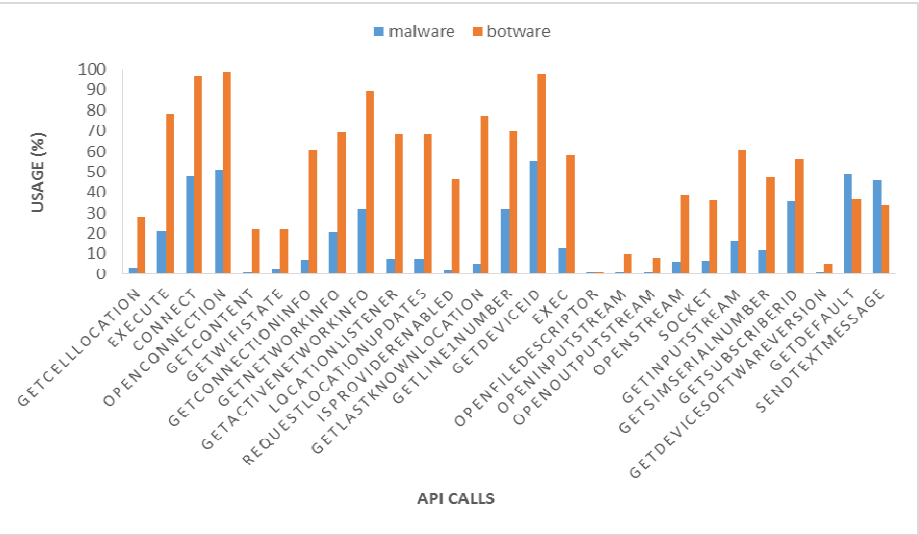

Figure 6: API calls comparison b/w botware and malware

5.2 Botware vs Benign Dataset

The results drawn by comparison of *botware* vs benign applications are shown in Figure 7 and Figure 8. Figure 7 depicts the permission usage (%) of benign applications and *botware* applications. Previous research work states that the frequency of permission requests in malware is much more than that in benign applications. However, malware writers write fewer explicit permission requests [Barrera, Kayacik et al. 2010, Felt, Greenwood et al. 2011, Aswini and Vinod 2014]. Indeed, the logic behind requesting maximum permissions is that, malware writers are trying to evade detection as calling those permissions indirectly through other code of the program. This behavior can certainly hinder the detection of malicious codes. Therefore, our system focuses on requested permissions, which certainly show indication of botnet motivation in long run.

It is interesting to note that, almost 100% of the benign and *botware* applications are exploiting the *INTERNET* permission. Therefore, we are not considering this permission for comparison for the purpose of botnet detection. Several other factors are of interest, for instance, *ACESS_FINE_LOCATION and ACCESS_NETWORK_STATE* are used by 96% and 90% of the *botware* applications while 34% and 48% of benign applications use these permissions. For connection to be persistent, *botware* applications manage to send and receive commands to C&C using *SEND_SMS* and *RECEIVE_SMS*, and the trend shows that 24% and 12% are *botware* applications in benign dataset. However, this trend is almost negligible in terms of benign applications' usage as depicting 5% and 3%. Similarly in malware dataset, *ACCESS_COARSE_LOCATION* permission is called by 70% of the botware applications. Whereas this trend is reduced to 20% of benign applications. Similarly, *READ_CONTACTS* and *READ_LOGS* is called by 46% and 59% of the botware applications. In contrast, 11% and 6% benign applications use this permission.

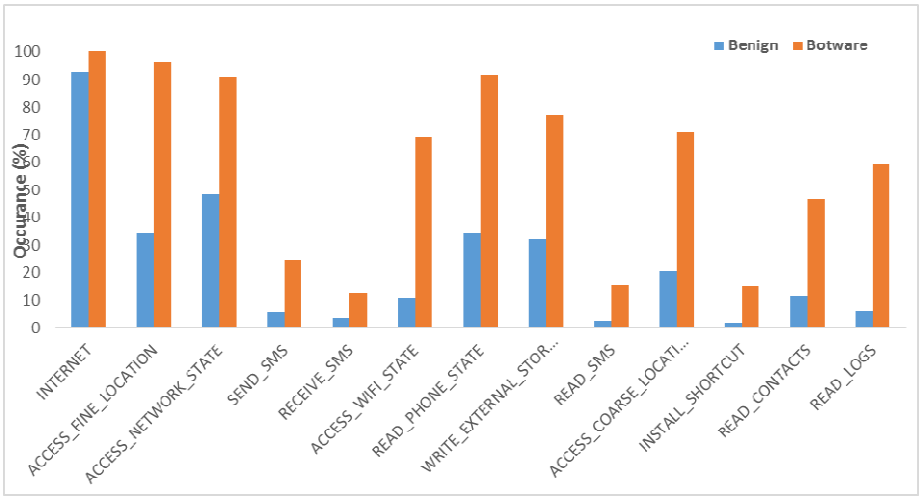

Figure 7: Comparison of benign and botware permissions

The comparison of *botware* and benign application with respect to API call functions is shown in Figure 8. The results show that *botware* deployed *execute(), connect* and *openConnection()* commands 73%, 90% and 96% respectively for the sake of establishing a remote connection. Whereas, in order to recognize bot location and identity information, *botware* commonly deployed *getDeviceID(), getLastKnownLocation(), getActiveNetworkInfo() and requestLocationUpdates()*. Therefore, we have seen the same trend in botware application, where 89% of the applications use *getDeviceID()*, 84% of apps request getLastKnownLocation(), 88% of the apps use *getActiveNetworkInfo()* and 85% of apps call *requestLocationUpdates()* . In contrast, this trend is minimal in benign application i.e. 17% of the applications use *getDeviceID()*, 14% of apps request *getLastKnownLocation(),* 23% of the apps use *getActiveNetworkInfo()* and 20% of apps call *requestLocationUpdates()*. Sending text message through GPRS service is accomplished by the function call *sendTextMessage()*. Around 21% of the applications call this function, while just 3% of benign applications invoke this method.

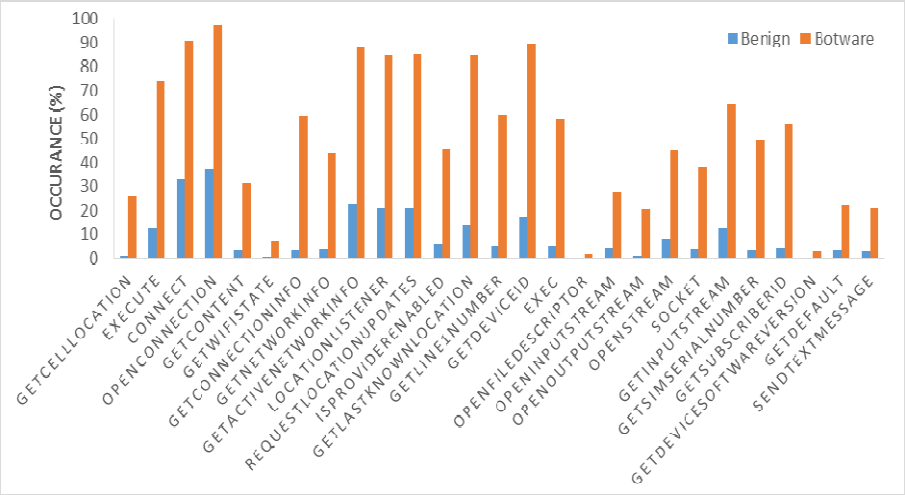

Figure 8: Comparison of benign and botware API calls

5.3 Performance Evaluation

Feature extraction from large datasets can be time consuming. Therefore, we examine the time consumption of our program logic for feature extraction. The program logic is written in python language, which uses regular expressions to extract features from *Manifest* and *.dex* file simultaneously. We have performed tests on *SANTUKO OS*, a Linux distribution especially designed for mobile malware analysis. For efficiency, we have used Intel Xeon® server 3.50GHz with 16GB RAM. The feature extraction process for botnet training set requires 1.090 seconds to search through and generate comma separated file. Whereas, the same python script requires 230.35 seconds in processing of malicious dataset to extract features and generate file for further analysis. Since the benign dataset consists large set of applications as compared to

malware dataset, this process requires more time, i.e, 476.53 second to mine feature vector and generate CSV file. For our analysis, we have also extracted malware families with closure to botnet behavior. For malware families' extraction, the script takes 114.94 seconds to find and store information to another CSV file. Table 4 shows the execution time elapsed against each data extraction process.

Dataset Scanned and Store as comma separated values	**Time Taken(seconds)**
Feature set scanned from training dataset and store in Botnet.csv file.	1.090
Feature set scanned from malicious dataset and store in Malware.csv file.	230.35
Feature set scanned from benign dataset and store in Benign.csv file.	476.53
Families scanned and stored on Family.csv file	114.94

Table 4: Execution Time Comparison

5.4 Effectiveness

We evaluate our analysis approach with *VirusTotal*[Nigam 2015], which provides a reliable malware scanning and detection service. It includes more than 50 off-the-shelf antivirus software. As a matter of fact, to-date no benchmark (mobile botnet dataset) is available to compare our findings. In addition, the direct comparison between VT and DeDroid is infeasible, because VT's detection criteria is based on various factors (static, dynamic) whereas DeDroid in particular is dealing with malware detection having C&C motivations. Thus, the only way to measure the effectiveness of our approach is – if a sample is detected by VT and the sample belongs to a particular malware family that is already proven as botnet family, DeDroid should mark that application as botware. However, many malware variants of a family may have accompanied C&C features in their following versions. Therefore, DeDroid's decision is subject to the initiation of C&C features regardless of their families.

During evaluation, all 5064 malware binaries which belong to 20 malware families [Daniel Arp] are effectively identified by VirusTotal. Among them, DeDroid detected 1795 malware samples having C&C features. To validate results, we have taken top 6 malware families with the highest detection ratio by DeDroid. The results affirm that, 5 out of top 6 malware families are known for their botnet related activities [Jiang 2011, Lookout 2011, Svajcer 2011, wyatt 2011]. Table 5 shows the top 6 malware families detected by DeDroid in malware dataset. From the table, we can observe the performance of DeDroid with respect to families having botnet C&C features. FakeRun family is 100% detected by DeDroid, whereas FakeDoc, DroidKungFu, Plankton, Geinimi and GoldDream are 98%, 78%, 84%, 90% and 80% detected by Dedroid respectively.

	Families	# of malwares in each family	Botwares Detected by DeDroid	Detection Accuracy of DeDroid (%)
From Malware Dataset	FakeRun	62	62	100
	FakeDoc	126	120	95
	Geinimi	88	79	90
	Plankton	581	493	84
	GoldDream	64	51	80
	DroidKungFu	629	493	78
From Benign Dataset	FakeRun	7	7	100
	GingerBreak	2	2	100
	Geinimi	14	13	93
	GoldDream	9	8	89
	Plankton	89	73	82
	KungFu	83	65	78
	DroidKungFu	134	92	69

Table 5: DeDroid detection accuracy of top 6 botware families for malware and benign applications

An interesting fact we have observed while looking the results generated by DeDroid is that, although literature [AVGThreatLab 2012] states that, FakeRun acts as adware, yet our system has detected it with high accuracy. Therefore, we decided to take one step ahead by observing its runtime behavior. For the sake of clarity, we have performed behavioral analysis using *DroidBox* [Lantz, Desnos et al. 2012] on a subset of FakeRun applications. In order to evaluate the results obtained from dynamic analysis, we have chosen ten applications from each of benign, malware, and FakeRun datasets. We have considered the following feature vector for analysis: read/write operations, file leaks, started services, DNS queries and HTTP conversations. Figure 9 shows that, on average, FakeRun has highest values against each of the said features. Similarly, the majority of IP addresses collected during HTTP communication from all FakeRun applications are marked blacklisted by more than one servers in the blacklist databases [Karim 2016]. Thus, behavior of FakeRun shows certain C&C communication patterns. Due to its structural properties, which are also proven by behavioral analysis, DeDroid has detected FakeRun as botware.

Therefore, we can conclude that, every malware family has certain number of applications that uses command and control (C&C) features. So, our system i.e *DeDroid* can effectively diagnose those applications as *botware*.

Further, as a second step of evaluation, among 14864 of benign binaries, DeDroid has detected 1196 samples as having C&C features. For validation, from 1196 binaries detected by DeDroid, we have taken top 7 malware families [Svajcer 2011, Xuxian 2012, Karim 2016] and draw the same conclusion that all families are known for their C&C features. The detection ratio against each malware family from benign dataset is also presented in Table 5. It can be depicted from Table 5 that, Dedroid has deteted malware samples in FakeRun, and GingerBreak families with 100% acuracy. Similarly, Geinimi, GoldDream, Plankton, KungFu, and DroidKungFu have the

detection accuracy of 93%, 89%, 82%, 78% and 69% respectively on DeDroid system.

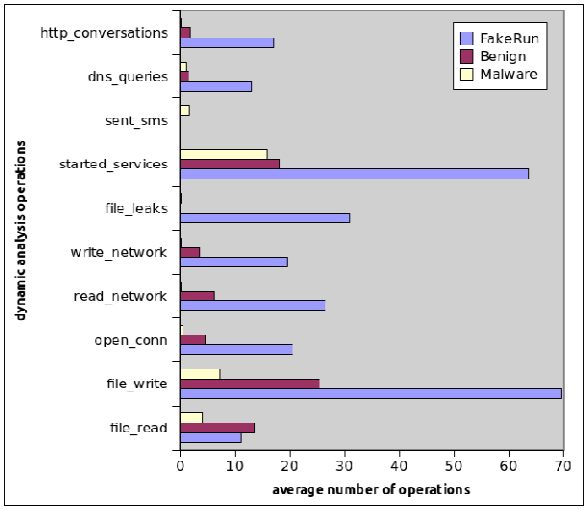

Figure 9: Dynamic analysis result comparison b/w Fakerun, benign and malware samples

5.5 Scalability

As *DeDroid* system is an offline analysis framework based on static properties of applications, therefore, it can work for large scale market place. Currently, *DeDroid* is using Drebin dataset for its evaluation purpose. Instead of using 500 malicious and 10,000 random binaries in a recent study [Spreitzenbarth, Freiling et al. 2013], our system consists of 5065 malware samples and 14865 benign binaries in order to effectively identify *botware* application. As a result of this large dataset, *DeDroid* system took almost 8 days to execute and collect all those features and stored in a database for further statistical analysis which is better than [Spreitzenbarth, Freiling et al. 2013]. Once the trace and log files has been obtained, the analysis process took few hours to analyze the results. As a future work, we are going to augment *Andrubis* [Technology 2012] by embedding the *DeDroid* logic into their system. Moreover, we can also deploy this "*as a service*" to Mobile Network Operator/ located in MNO core network, OS vendor market stores/ located in App Store, after market security product vendor/ located in device and or Cloud etc.

5.6 Adaptability

In order to measure the effectiveness of DeDroid approach according to growing sophistication in malware programs we need to observe its efficacy with state-of-the-art malwares and botwares. For this purpose, we collected some malware and botware samples/families from [Parkour 2015] that are diagnosed during 2015. Table 6 shows the details of each malware/botware including Hash value, its actual class according to literature and DeDroid's predicted class. The results affirm the adaptability of our

approach with evolving malware and botware samples. DeDroid achieves detection with 94% accuracy, 88% TPR (recall) and 0% FPR.

File Name	Hash	Actual Class	DeDroid Class
Godwon	79309179DB63D2B505398ABCB4DD1AE0	Malware	Malware
FacebookOTP	021D55C415FF951C8E7B1CE3F94399BB	Botware	Botware
Cajino	B3814CA9E42681B32DAFE4A52E5BDA7A	Botware	Botware
Gazon	4A56C7ABDC455C82E95753BDB1934285	Malware	Malware
HackingTeamRSC	904ED531D0B3B1979F1FDA7A9504C882	Botware	Botware
Cajino	39581735EE24D54F93C8C51D8C39B506	Botware	Botware
FBI_Ransom	F836F5C6267F13BF9F6109A6B8D79175	Malware	Malware
Com.studio.proxy	D05D3F579295CD5018318072ADF3B83D	Malware	Malware
Podec	72ADCF52448B2F7BC8CADA8AF8657EEB	Botware	Botware
Cajino	9342B4ECBB7EB045EDCDB6E0E339E415	Botware	Botware
Save_me	78835947CCA21BA42110A4F206A7A486	Botware	Botware
Android-Locker-qqmagic	735B4E78B334F6B9EB19E700A4C30966	Malware	Malware
Remote_control_smack	370FE3D8E9B40702B08A5F93003DE0D3	Botware	Botware
Hijack_Rat	A21FAB634DC788CDD462D506458AF1E4	Botware	Malware
FakeApp.AL	ACB66E858D54C61AA10E60276001C02B	Malware	Malware

Table 6: DeDroid detection accuracy for contemporary malicious applications

We have presented the TPR and FPR for 15 applications. However, in order to prove the effectiveness of DeDroid approach for large repositories, we need to perform more experiments. For this purpose, we have downloaded latest malware and benign binaries from recent published work [Kang, Jang et al. 2015] and botware applications from open repositories [Kadir, Stakhanova et al. 2015, Parkour 2015]. As a result, we gathered 4906 benign, 1084 malware and 100 botware samples. We have taken 100 botware samples from each of the five botware families as shown in Figure 10. It can be depicted that, from these 100 known botware binaries DeDroid can classify them with high accuracy, i.e 99% accuracy is achieved. Similarly, DeDroid's performance was not hindered even in case of applications using native code, dynamic code loading, java reflection or cryptographic operations. Moreover, the average threshold value observed by each of the botware family is between 20 and 26.

Likewise, the DeDroid performance with respect to contemporary malware (not having C&C) and benign applications is also acceptable which is shown in Table 7 and Table 8 respectively. Among 1168 malware binaries, 98% of them found below the threshold value which endorses the efficacy of our logic behind DeDroid system. On average, the threshold value observed in malware samples is between 2.5 to 10. Similarly, 99% of benign samples are correctly classified from the latest mobile malware dataset [Kang, Jang et al. 2015] comprising of 4906 applications.

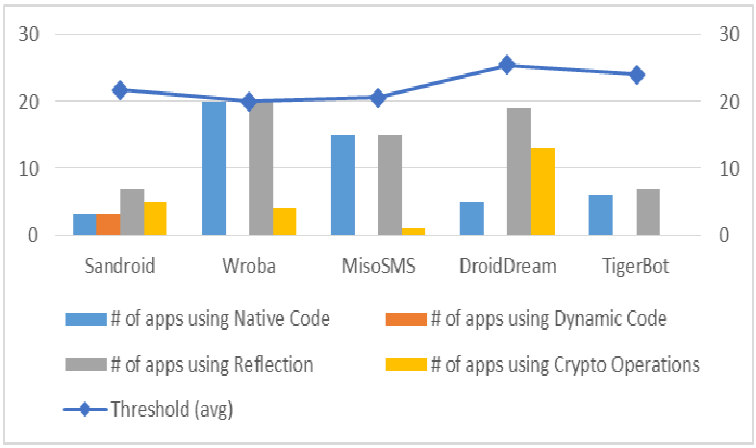

Figure 10: DeDroid detection with respect to contemporary mobile botware applications

Malware Families [Kang, Jang et al. 2015]	**Total Samples**	**Detected by DeDroid**	**Threshold (avg)**
AdWo	100	86	10.24
Airpush	35	34	9.14
Boxer	28	28	2.46
FakeInstaller	921	921	4.97
FakeNotify	84	84	5.00

Table 7: DeDroid detection with respect to contemporary Mobile Malware Applications

# of Benign Samples [Kang, Jang et al. 2015]	**DeDroid detected as benign**	**DeDroid detected as Botware**	**Threshold (avg)**
4906	4835	71	3.8

Table 8: DeDroid detection with respect to contemporary Benign Applications

6 Conclusion and future work

Botnet phenomenon is migrating progressively from previous PC based platform generation to new emerging computational intensive mobile platform. Therefore, the urge is to devise some proactive mechanisms to avoid this hazard. In this paper we have introduced a novel approach, *DeDroid*, which can effectively detect botnet C&C communication features in malicious and benign Android binaries using static analysis. The feature selection is carried out by observing static behavior of known

mobile botnet applications where static feature vector comprises of permission modules and API calls from Android operating systems. After feature selection, feature extraction process is accomplished through a program logic written in python. For proof of concept, we used *Drebin* dataset (which is currently a largest Android malware dataset) to evaluate and compare our findings with malicious and benign applications. The results affirm that *DeDroid* provides a lightweight solution to effectively identify botnet capabilities in malicious and benign mobile applications. Moreover, DeDroid's effectiveness is tested on large scale contemporary botnet and malware datasets and achived high accuracy in terms of detecting C&C enabled binaries. However, since the detection is solely based on static feature vectors; the detection accuracy can further be improved by analyzing the communication patterns at runtime. Therefore, as part of future work, we aim to expand this analysis to dynamic and network layer exploration, which will certainly enhance the botnet detection rate.

Acknowledgements

This research was supported by a research grant from the R&D program (IPPP) of the University of Malaya -- Project No. FP034-2012A. The authors also extend their sincere appreciations to the Deanship of Scientific Research at King Saud University for its funding this Prolific Research Group (PRG-1436-16).

References

[Ahmad Karim 2015] Ahmad Karim, R. S., Syed Adeel Ali Shah (2015). DeDroid: A Mobile Botnet Detection Approach Based on Static Analysis. The 7th International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies. Beijing, China, IEEE.

[Alcatel-Lucent 2015] Alcatel-Lucent. (2015). "NotCompatible – Android Web Proxy Bot." Retrieved June 15, 2015, from http://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2014/9594-notcompatibleandroid-web-proxy-bot-malware-analysis-report.pdf.

[Alzahrani and Ghorbani 2014] Alzahrani, A. J. and A. A. Ghorbani (2014). SMS mobile botnet detection using a multi-agent system: research in progress. Proceedings of the 1st International Workshop on Agents and CyberSecurity, ACM.

[Aswini and Vinod 2014] Aswini, A. and P. Vinod (2014). Droid permission miner: Mining prominent permissions for Android malware analysis. 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), IEEE.

[AVGThreatLab 2012] AVGThreatLab. (2012). "Android/Ev-trojan-fakerun." Retrieved January 2014, 2014, from http://www.avgthreatlabs.com/ww-en/virus-and-malware-information/info/android-ev-trojan-fakerun/.

[Barrera, Kayacik et al. 2010] Barrera, D., H. G. Kayacik, P. C. van Oorschot and A. Somayaji (2010). A methodology for empirical analysis of permission-based security models and its application to android. Proceedings of the 17th ACM conference on Computer and communications security, ACM.

[Choi, Choi et al. 2013] Choi, B., S.-K. Choi and K. Cho (2013). Detection of Mobile Botnet Using VPN. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), IEEE.

[Choo 2011] Choo, K.-K. R. (2011). "The cyber threat landscape: Challenges and future research directions." Computers & Security **30**(8): 719-731.

[Choo 2007] Choo, K. (2007). Zombies And Botnets. Trends And Issues In Crime And Criminal Justice, Australian Institute of Criminology Canberra.

[Christodorescu and Jha 2006] Christodorescu, M. and S. Jha (2006). Static analysis of executables to detect malicious patterns, DTIC Document.

[Christodorescu, Jha et al. 2008] Christodorescu, M., S. Jha and C. Kruegel (2008). Mining specifications of malicious behavior. Proceedings of the 1st India software engineering conference, ACM.

[Damshenas, Dehghantanha et al. 2015] Damshenas, M., A. Dehghantanha, K.-K. R. Choo and R. Mahmud (2015). "M0droid: An android behavioral-based malware detection model." Journal of Information Privacy and Security **11**(3): 141-157.

[Daniel Arp 2013] Daniel Arp, M. S., Malte Huebner, Hugo Gascon, and Konrad Rieck. (2013). "The Drebin Dataset." Retrieved Octobar 6, 2014, from http://user.informatik.uni-goettingen.de/~darp/drebin/.

[Data 2015] Data, G. (2015). "MOBILE MALWARE REPORT." Retrieved September 2, 2015, from https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/G_DATA_MobileMWR_Q1_2015_US.pdf.

[Desnos 2011] Desnos, A. (2011). "Androguard: Reverse engineering, malware and goodware analysis of android applications... and more (ninja!)." Retrieved June 10, 2014, from http://code.google.com/p/androguard/.

[Do, Martini et al. 2015] Do, Q., B. Martini and K.-K. R. Choo (2015). "Exfiltrating data from Android devices." Computers & Security **48**: 74-91.

[F-Secure 2009] F-Secure. (2009). "Threat Description Worm: ?iPhoneOS/Ikee.B." Retrieved September 6, 2013, from http://www.f-secure.com/v-descs/worm_iphoneos_ikee_b.shtml

[Felt, Greenwood et al. 2011] Felt, A. P., K. Greenwood and D. Wagner (2011). The effectiveness of application permissions. Proceedings of the 2nd USENIX conference on Web application development.

[Hyppönen 2013] Hyppönen, M. (2013). "Threat Report H2 2013." Retrieved December 8, 2013, from https://www.f-secure.com/documents/996508/1030743/Threat_Report_H2_2013.pdf.

[IDC 2014] IDC. (2014). "Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013." Retrieved December 16, 2014, from http://www.idc.com/getdoc.jsp?containerId=prUS24676414.

[Jiang 2011] Jiang, X. (2011). "Security Alert: New Android Malware -- GoldDream -- Found in Alternative App Markets." Retrieved June 17, 2013, from http://www.csc.ncsu.edu/faculty/jiang/GoldDream/.

[Kadir, Stakhanova et al. 2015] Kadir, A. F. A., N. Stakhanova and A. A. Ghorbani (2015). Android Botnets: What URLs are Telling Us. Network and System Security, Springer**:** 78-91.

[Kang, Jang et al. 2015] Kang, H., J.-w. Jang, A. Mohaisen and H. K. Kim (2015). "Detecting and classifying android malware using static analysis along with creator information." International Journal of Distributed Sensor Networks **2015**: 7.

[Karim 2016] Karim, A. (2016). "FakeRun Blacklisted IP addresses." Retrieved December 2015, 2015, from https://docs.google.com/document/d/1msfl0xO9DRFzIrqov1d925tnfILgjC3A8IRa42SaFgg/pub

[Karim, Salleh et al. 2016] Karim, A., R. Salleh and M. K. Khan (2016). "SMARTbot: A Behavioral Analysis Framework Augmented with Machine Learning to Identify Mobile Botnet Applications." PloS one **11**(3): e0150077.

[Karim, Shah et al. 2014] Karim, A., S. A. A. Shah and R. Salleh (2014). Mobile Botnet Attacks: A Thematic Taxonomy. New Perspectives in Information Systems and Technologies, Volume 2, Springer**:** 153-164.

[Karim, Shah et al. 2015] Karim, A., S. A. A. Shah, R. B. Salleh, M. Arif, R. M. Noor and S. Shamshirband (2015). "Mobile Botnet Attacks–an Emerging Threat: Classification, Review and Open Issues."

[Lantz, Desnos et al. 2012] Lantz, P., A. Desnos and K. Yang (2012). DroidBox: Android application sandbox.

[Lookout 2011] Lookout. (2011). "Security Alert: New Malware Found in Alternative Android Markets: DroidKungFu." Retrieved May 4, 2013, from https://blog.lookout.com/blog/2011/06/06/security-alert-new-malware-found-in-alternative-android-markets-legacy/.

[Millman 2015] Millman, R. (2015). "Updated: 97% of malicious mobile malware targets Android." Retrieved December 9, 2015, from http://www.scmagazineuk.com/updated-97-of-malicious-mobile-malware-targets-android/article/422783/.

[Neugschwandtner, Lindorfer et al. 2013] Neugschwandtner, M., M. Lindorfer and C. Platzer (2013). A View to a Kill: WebView Exploitation. LEET.

[Nigam 2015] Nigam, R. (2015). "A timeline of mobile botnets." Retrieved June 5, 2015, from https://www.virusbtn.com/virusbulletin/archive/2015/03/vb201503-mobile-botnets.

[Nigam 2015] Nigam, R. (2015). "A timeline of mobile botnets, https://www.virusbtn.com/virusbulletin/archive/2015/03/vb201503-mobile-botnets."

[Parkour 2015] Parkour, M. (2015). Contagio Mobile. Mobile malware mini dump.

[Shabtai, Moskovitch et al. 2009] Shabtai, A., R. Moskovitch, Y. Elovici and C. Glezer (2009). "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey." Information Security Technical Report **14**(1): 16-29.

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

[Shea 2015] Shea, S. (2015). "Mobile malware statistics highlight unknown state of mobile threats." Retrieved December 10, 2015, from http://searchsecurity.techtarget.com/news/4500245950/Mobile-malware-statistics-highlight-unknown-state-of-mobile-threats.

[Spreitzenbarth, Freiling et al. 2013] Spreitzenbarth, M., F. Freiling, F. Echtler, T. Schreck and J. Hoffmann (2013). Mobile-sandbox: Having a deeper look into android applications. Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM.

[Strazzere and Wyatt 2011] Strazzere, T. and T. Wyatt. (2011). "Geinimi trojan technical teardown." Lookout Mobile Security Retrieved June 20, 2013, from https://blog.lookout.com/_media/Geinimi_Trojan_Teardown.pdf.

[Svajcer 2011] Svajcer, V. (2011). "Plankton malware drifts into Android Market." Retrieved June 17, 2013, from https://nakedsecurity.sophos.com/2011/06/14/plankton-malware-drifts-into-android-market/.

[Svajcer 2011] Svajcer, V. (2011). "Plankton malware drifts into Android Market."

[Technology 2012] Technology, V. U. o. (2012). "Andrubis-analysis of android apks." Retrieved June 5, 2013, from http://anubis.iseclab.org.

[Walls and Choo 2015] Walls, J. and K.-K. R. Choo (2015). A Review of Free Cloud-Based Anti-Malware Apps for Android. Trustcom/BigDataSE/ISPA, 2015 IEEE, IEEE.

[Weafer 2014] Weafer, V. (2014). "McAfee Labs Threats Report." Retrieved August 9, 2014, from http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2014.pdf.

[wyatt 2011] wyatt, t. (2011). "Security Alert: Geinimi, Sophisticated New Android Trojan Found in Wild." Retrieved December 15, 2013, from https://blog.lookout.com/blog/2010/12/29/geinimi_trojan/.

[Xuxian 2012] Xuxian, J. (2012). "Security Alert: New RootSmart Android Malware Utilizes the GingerBreak Root Exploit." Retrieved January 8, 2015, from http://www.csc.ncsu.edu/faculty/jiang/RootSmart/.