A Smart Framework for Mobile Botnet Detection Using Static Analysis

Shahid Anwar[1], Mohamad Fadli Zolkipli[2], Vitaliy Mezhuyev[3,*], Zakira Inayat[4]

[1] Department of Software Engineering, The University of Lahore | 1-Km, Defence Road
Bhobatian Chowk, Lahore, Pakistan.
[e-mail: shahidanwar.safi@gmail.com]
[2] Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang, Lebuhraya Tun Razak Gambang, 26300 Kuantan, Malaysia
[e-mail: fadli@ump.edu.my]
[3] FH JOANNEUM University of Applied Sciences, Institute of Industrial Management
Werk-VI-Straße 46, 8605 Kapfenberg, Austria
[e-mail: vitaliy.mezhuyev@fh-joanneum.at]
[4] Department of Computer Science, University of Engineering and Technology,
Peshawar 2500, Pakistan
[e-mail: zakirainayat@uetpeshawar.edu.pk]
*Corresponding author: Vitaliy Mezhuyev

Received May 12, 2019; revised November 11, 2019; accepted March 18, 2020; published June 30, 2020

Abstract

Botnets have become one of the most significant threats to Internet-connected smartphones. A botnet is a combination of infected devices communicating through a command server under the control of botmaster for malicious purposes. Nowadays, the number and variety of botnets attacks have increased drastically, especially on the Android platform. Severe network disruptions through massive coordinated attacks result in large financial and ethical losses. The increase in the number of botnet attacks brings the challenges for detection of harmful software. This study proposes a smart framework for mobile botnet detection using static analysis. This technique combines permissions, activities, broadcast receivers, background services, API and uses the machine-learning algorithm to detect mobile botnets applications. The prototype was implemented and used to validate the performance, accuracy, and scalability of the proposed framework by evaluating 3000 android applications. The obtained results show the proposed framework obtained 98.20% accuracy with a low 0.1140 false-positive rate.

Keywords: Android Botnets, Smart Framework, Static Analysis, Botnet Detection Technique

1. Introduction

In the recent era, explosive growth has been seen in the sale and adoption of smartphone devices. Gartner stated in their report that the sale of smartphones is increased by 87% worldwide in the fourth quarter of 2018 [1, 2]. In order to take a maximal benefit from the smartphones facilities, relevant software applications should be installed. Correspondingly, users have been downloading an increasingly large number of applications in response to advancements in smartphones [3]. The Android platform becomes the most popular with an estimated 4.839 million applications in the official Google's Android market, having downloads in excess of 25 billion by November 2018 [4]. Meanwhile, the malicious software (malware) is on the top with the adding of 2.8 million samples to the end-users libraries [5]. The increased number of goodware applications also leads to a greater chance of installing malware to smartphones. It reflects the harmful intention of cybercriminals to create malware like adware, botnets, bugs, rootkits, spyware, Trojan horses, viruses and much more [67; 71; 75]. Among this malware, the botnets are quickly gaining the attention of researchers belonging to both academia and industry. The term "bot" is derived from the "robot" and "net" is derived from the "network" [6, 7]. A bot is a type of malware that runs automatically after installation in the victim device and get the full control of that device. It is not required for the malware program to manipulate by a remote command and control servers. Therefore, the major difference between botnet and malware is the unconditional control of a remote machine through the bot codes. The botnets can be a platform for a slave, provided by Internet-connected computers. In general, botnets can be classified into traditional and mobile ones [65]. In the case of the traditional botnets, computers provide the platform for the slave bots. However, in the mobile botnet, a mobile device provides the platform for the slaves. Furthermore, the mobile botnet can be generally classified as hypertext transfer protocol (HTTP) based, Internet relay chat (IRC) based, and peer to peer (P2P) based, according to the underlying C&C (Command and Control) communication protocol [8]. Unlike the traditional cybercrime, a mobile botnet can attack and propagate itself through various methods and may cause much greater losses to smartphones [9, 10].

Once a smartphone is being infected, it can be converted to bot zombie [11, 12]. An infected smartphone can cause drastic threats to the end-users or the service-provider company. As far as researchers are aware, the first mobile botnet named SymbOS/Yxes appeared in 2009 [13]. The target of SymbOS/Yxes botnet was SYMBIAN OS by using a rudimentary HTTP-based C&C channel: both Trojanized game applications with bot-like capabilities that compromised mobile devices [14]. Since this study focuses on the Android OS, the terms "Android botnet" and "mobile botnet" will be used in the same meaning. In this case, the bots infect smartphones, personal digital devices (PDAs), smart watches, and tablets that have Android OS. These mobile botnets have the capability to steal sensitive information like phone numbers, International mobile equipment identity (IMEI), network operator details, international mobile subscriber identity (IMSI), voice mail numbers, location information and much more from the infected devices using the connection with command and control servers [15-19]. Phishing, SMS fraud, Spyware, Fake installation, Banking Trojans, and premium dialers are the most common threats of botnets.

A mobile botnet has four basic components, such as botmaster, command and control (C&C) server, bots, and communication channel. The one, who owns the mobile botnet, known as botmaster. C&C server is the most important component of a mobile botnet. Most of these botnets use short messaging service (SMS), hypertext transfer protocol (HTTP), and HTTP & SMS together as a command and control server [20-22]. The C&C

servers are responsible for sending and receiving commands from botmaster to the infected devices known as bots. Communication channel refers to the network through which these bots are connected in order to perform illegal activities. **Fig. 1** shows the basic architecture of the mobile botnet.

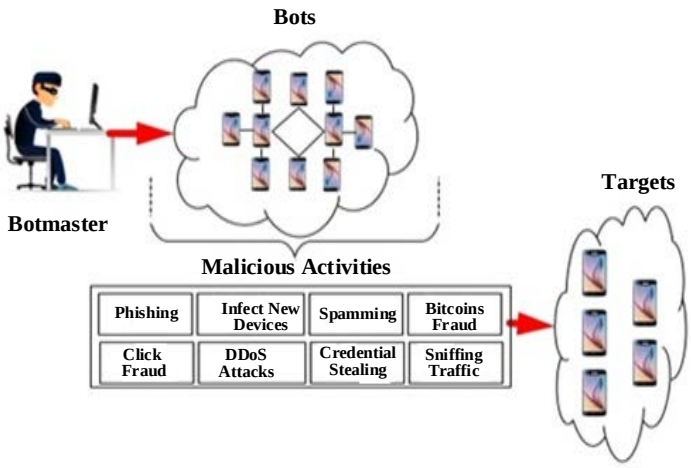


Fig. 1. Mobile Botnet Basic Architecture

Recently, there is a rapid growth seen in the mobile botnets to damage the performance of smartphones. At this time, NotCompatible.C is the most dangerous botnet (which belongs to AnServer bots) affected a huge amount of smartphones to access their services [23]. P2P communication architecture and various evasion approaches discriminate NotCompatible.C botnet from other mobile botnets. Similarly, TigerBot, Zeus Botnet, BMaster perform the attacks on smartphones for financial profit, identity theft, and stealing personal information [24]. The most significant feature of the Zeus botnet is the ability to perform malware activities in different operating systems. However, other mentioned botnets are built only for Android OS [25]. The Obad.a is another dangerous botnet, which uses SMS and HTTP for the distribution among the devices. In this case, the bots were infected by sending users a message with the text “MMS message has been delivered, download from www.otkroi.com” [26]. Similarly, the botmaster uses the attack vectors, such as SMS, MMS, or USB. Commonly, Bluetooth and Wi-Fi are used to spread the infected commands among the targeted devices [27]. Successful execution of an infected command on a targeted smartphone empowers a botmaster to access a victim device for the above-mentioned illegal activities. In addition, mobile botnets have extra features as compare to PC botnets, such as portability, 24/7 connectivity to the Internet, access to the most credential information, possibility of sending and receiving SMS and MMS, and dialling capabilities [28]. Therefore, botnets are expected to maintain their severe effects on mobile devices.

The rapid growth of mobile botnet applications realizes the need to develop an efficacious solution for mobile botnet detection. This study proposes a new framework for mobile botnet detection using static feature extraction and analysis. In the static analysis, the applications are decompiled, making possible exploration of the suspicious features and code [38]. To effectively identify the most significant botnet features, a huge number of benign and botware applications were surveyed and machine-learning approach was employed. The main contribution of this paper is given below.

- This paper proposed a smart framework for botnet detection in Android devices using static feature extraction and analysis.
- This study analysed a significant number of mobile botnet applications to identify the botnet C&C structure and their most relevant features.
- By using the information gain (IG) and support values approach, the most prominent features were selected and then used for classification. A selected machine-learning approach was used to classify the botnet attributes.

This paper is organized as follows: Section 2 provides an overview of related work; Section 3 describes the data collection process and the experimental study; Section 4 explains the proposed framework in more detail. Experimental results and comparison with prior work are given in Section 5. Conclusion and the reference list finalize the study.

2. Related Work

Nowadays, mobile botnet detection becomes a crucial task. Different studies proposed a number of techniques for botnet applications detection and prevention [1, 18, 29-32, 73, 79]. For easy understanding, the existing detection techniques can be classified into two main categories, such as dynamic and static analyses [33, 34]. In the dynamic analysis technique, the applications are installed and run on a real device or on an emulator to observe the malicious behaviour of the malware [35]. In order to inspect the suspicious behaviour of applications sandboxes are used [36]. These are special security tools used to inspect and stop the illegal activities of malicious applications. Commercially available sandboxes provide sustainable analysis and logging facilities to observe a particular bot and to generate the report [37]. However, the generated data may not be focused on the pivotal activities of the bots. Bots can generate an overwhelming amount of information extensively calling particular functions observed by the sandbox.

On the other hand, in the static analysis technique, the applications are decompiled to find the suspicious features and lines of code [38]. This technique has been used widely due to the fast processing approach as compared to the dynamic one. Permissions and API calls are the most commonly used features in static analysis. Permissions are the gateways to interact with system resources. Each Android application requests for specific permissions, including permission to access SMS and calling modules, internal and external memory etc. AndroidManifest.XML and meta-data contain these and other important features for all Android applications available from the Google Play store [39]. In static analysis, in order to extract these features, the APK files are decompiled without executing the applications itself.

Yerima et al. [40] proposed a static analysis approach to detect Android malware by extracting the most important features, such as permissions, API calls from the AndroidManifest and Classes of Android applications. In order to extract these features, they used 1000 malicious and 1000 benign applications. The Bayesian Classifier was applied to perform the classification. The authors claimed that they achieved 92.1% accuracy by the consideration of 30 static features. However, this approach is incompetent to thoroughly scrutinize the Android applications code for suspicious behaviours. Canfora, et al. [41] claimed that the classification of malware families is not allowed in this approach.

Another mobile botnet detection approach was proposed by Peiravian and Zhu [42]. The authors utilized API calls, permission, and the combination of features. In their study, permissions were categorized into two types, such as requested and required permissions. The $API_{j,i}$ is represented with *1*, such as $API_j = 1$ if the j^{th} API call is made in the

application. Similarly, the required permission is represented by *1*, such as $P_i = 1$. The third feature was obtained by the concatenating of these two features. In the experimental part of the study, there were 2510 samples including 1250 benign and 1260 malicious applications. The authors achieved 96.88% accuracy by selecting a high number of features. This study shows that the accuracy can be improved when the features are selected from both malware and benign applications. It also can be concluded that Bagging (an ensemble classification method) had the best performance in classifying the datasets.

De-Droid is another mobile botnet detection approach proposed by Karim et al. [43]. The authors claimed that this is a lightweight approach focusing on static analysis. This approach considers the API calls and permissions to detect suspicious binaries. In this study, 5064 malicious and 14865 benign applications were used to decode the API calls and permissions. Rashidi & Fung proposed a BotTracer, a clustering-based approach to detect bot users by the same master [44]. The key idea behind this technique is to collect the expert users' responses to a permission request and recommend them to inexperienced users. The first step of BotTracer is the analysis of the common features of bot users resulting in the similarity graph. Then, a hierarchical clustering method employed to group the users based on the distance, defined using similarity. However, the proposed approaches are unable to detect zero-day attacks and to deeply scrutinize the applications for possible malicious behaviour.

3. Data Collection and experimental study

For the experiments, the clean (benign) and infected third-party mobile applications were used. Currently, there are various types of applications in the market, such as web tools, native applications, and widgets [18]. The web applications are those developed through HTML, JavaScript, and CSS, while the natives are developed with Android SDK. The widget applications are those which are displayed on the Android desktop. Moreover, experiments use benign applications of native nature. The samples were obtained from the Google Android market [39]. The VirusTotal Malware Intelligence Service allows the researchers to obtain samples from their databases [45]. Initially, 1865 applications were obtained and tested with the VirusTotal tool to confirm their cleanness. Finally, to avoid the duplication the Monte Carlo sampling method [66] was used giving in total 1330 benign applications, as shown in **Table 1**.

Table 1. Number of Benign Applications for each category

Category	Number of Apps	Category	Number of Applications
Education	120	Business	62
Music	103	Maps	42
Health and Fitness	94	Social	90
Finance	112	Photography	52
Medical	90	Puzzles	61
News	115	Sports	79
Shopping	117	Entertainment	122
Wallpaper	38	Communication	33

The Botnet applications, which also known as botware, are collected from the different open source repositories, such as DREBIN, ISCX Android Botnet Dataset, and Android Malware Genome Project [46-48]. The DREBIN dataset consists of 5560

applications from different 179 malware families, the ISCX dataset includes 1929 botnet samples, and the Android malware genome project consists of 1200 malware samples. To the best of our knowledge, these are the largest Android malware and botnet datasets, which include all the publicly available samples and open-source malware repositories. The same number of benign and botware applications are considered to facilitate machine-learning modelling. **Table 2** shows the number of malware applications and the name of the corresponding malicious activities. Symbol ✓ shows that the malware family can perform this activity, while the × shows that the malware cannot perform it.

Table 2. Number of Samples for Botware Applications with Malicious Activities

Category	**Number of Apps**	**IMEI/IMSI**	**SMS**	**Call**	**Location**	**Root Exploit**	**Repacking**	**DDoS**	**Banking Information**
AnserverBot	95	✓	✓	✓	✓	✓	✓	✓	✓
Bmaster	6	✓	×	✓	×	✓	✓	✓	✓
DroidDream	143	✓	×	✓	×	✓	✓	✓	×
Geinimi	126	✓	×	✓	✓	✓	✓	✓	×
MisoSMS	92	✓	✓	✓	×	✓	✓	✓	✓
NickySpy	112	✓	✓	✓	✓	✓	✓	×	×
Cot Compatible	72	✓	✓	✓	×	✓	✓	✓	✓
PJapps	118	✓	×	✓	×	✓	✓	✓	✓
Pletor	80	✓	✓	✓	✓	✓	✓	✓	✓
RootSmart	25	✓	✓	✓	✓	×	✓	×	×
Fjcon	42	✓	✓	✓	✓	×	✓	✓	✓
FakeInstaller	90	✓	×	✓	×	✓	✓	×	✓
Twikabot.A	90	✓	×	✓	✓	×	✓	×	✓
Saiva	92	✓	✓	✓	✓	✓	✓	✓	✓
Sweaweth	82	✓	×	✓	✓	✓	✓	×	✓
AndroRat.A	65	✓	✓	✓	✓	✓	✓	✓	✓

4. Smart Framework for Android Botnet Detection

This section demonstrates the proposed smart framework for mobile botnet detection, which is based on the static features extraction in Android devices (**Fig. 2**). The proposed framework has five layers of mobile security that can detect applications having the features of mobile botnets. These five layers of detection technique are the decompiler, extractor, smart learner, features refiner, and the machine learning module respectively. Using the framework, whenever a user completes the installation of an application, a popup window appears on the screen if the installed application contains an injected code from the third party (i.e. from a cyber-criminal).

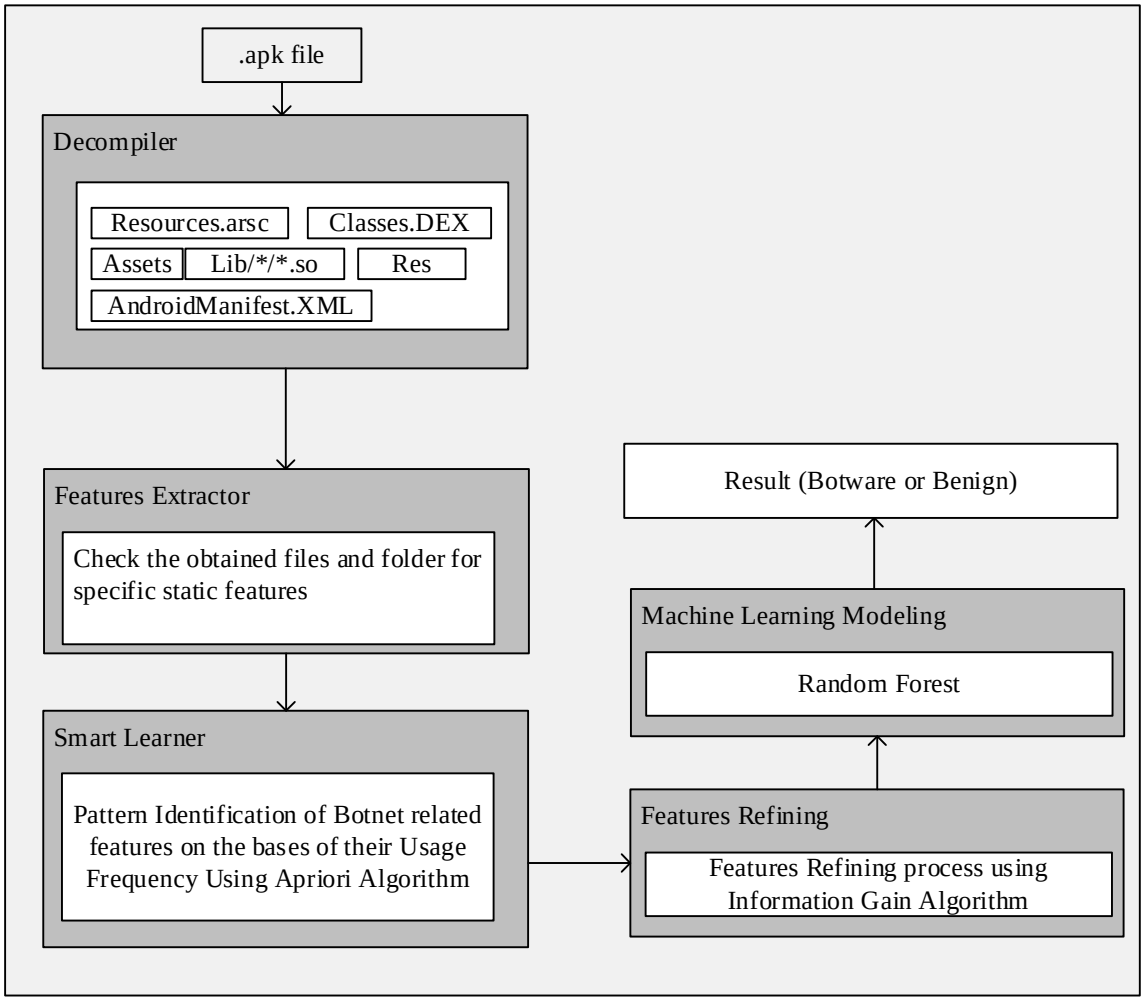


Fig. 2. A Proposed Framework for Android Botnet Detection

4.1. App Decompiler

Decompiler is responsible for the translation of the machine code into the source code. The Android applications are distributed as a machine code that is called Android package file (APKs). In order to decompile the APK files, the Android asset packaging tool (AAPT) was used (available within the Android SDK [49]). Each APK file contains the AndroidManifest.xml and the classes.DEX files that are decoded to make them human-readable. The AndroidManifest.xml describes an Android application and encloses the essential information about the APK file, such as permissions, activities, services, and intent-filters. The same procedure was applied to decompile the classes.DEX files, which are the java source code.

4.2. Feature Extractor

This component performs reverse engineering of each decompiled application to extract static features from the AndroidManifest.xml and the classes.DEX files. Besides, tags to interact with the features inside and outside of the Android applications were extracted. For

example, <uses − permission> was employed to get the permissions for accessing hardware and other components of smartphones, while <uses − activities> was used to switch between different activities inside the application. Permission tags can only interact with the smartphones when a user grants permissions to the applications during the installation or later on. However, not all requested permissions may be used by the application. E.g. <uses − features> is required to interact with other components of the application, while some of the features may not be used.

In order to extract these features from an Android application, the Androguard open-source tool was used [50]. After extraction of required features, for each Android application, a CSV file was generated for further processing. The static features, namely, permissions, activities, broadcast receivers, services, and API calls, were extracted from the AndroidManifest.XML and the classes.DEX files. In order, to automatically perform extraction of the features a Python script was applied to all Android applications. Let l and m be correspondingly the number of Android applications and the set of features including permissions, activities, broadcast receivers, services, and API calls. The features vector for the application i is $(X_{i,1}, X_{i,2}, X_{i,3}, \dots\dots, X_{i,n})$ where:

$$F(i,j) = \begin{cases} 1 & if\ the\ application\ l\ uses\ the\ feature\ k \\ 0 & Otherwise \end{cases} \quad (1)$$

Similarly, suppose an instance of the class in the generated dataset is $F_i \in$ {Botware, Benign}, where i shows the class of application [51]. Features of each extracted application were saved in the CSV file for further analysis (**Fig. 3**). The file begins with the hash function of the application and ends with the sum of all the enabled features. The hash value represents the MD5 values of the Android applications, while the values "1" and "0" correspond to enabled and disabled features, respectively. The sum of the enabled values was utilized for further analysis. In general, the analysis shows that the applications that were using more features had a higher probability to have a botware intention.

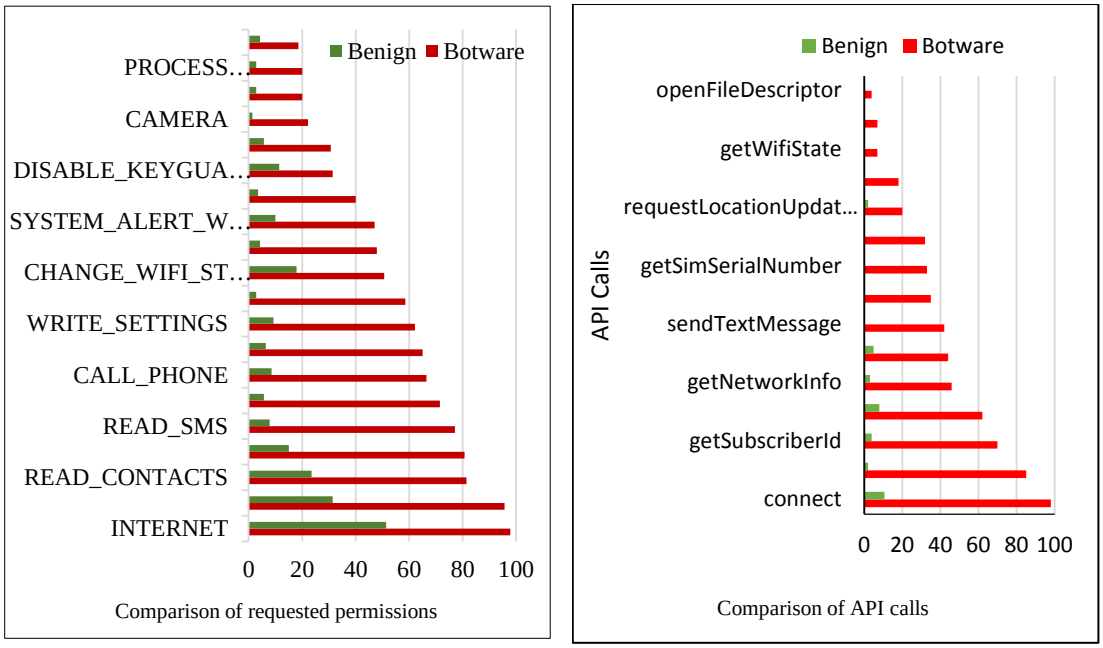
(00DA00BA346A4B1AB452651A003A0BA37A463E4A4BAB452651A),<1,1,1,0,1,1,
0,0,1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,1,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,0,1,1,0,
0,1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,1,0,0,0,1,0,1,0,1,0,1,0,11,1,1,0,1,1,0,0,
1,0,1,0,1,1,1,1,1,1,1,0,0,1,0,1,0,1,0,0,1,0,1,1,1,1,1,1,0,1,1,1>,(80)

Fig. 3. Example of the CSV file of Extracted Application

Fig. 4 shows the frequencies of the permissions requests and API calls feature. The figure demonstrates that the number of the used by botware applications API calls and requested permissions is higher as compared to the benign applications. At the same time, requesting more features did not mean that the botware developers would be able to use all these features. The logic behind requesting a large number of features is that malware developers are trying to evade detection by calling these features indirectly through the code of another program. This strategy can certainly hinder the detection of malware codes.

Fig. 4(a) illustrates the number of requested and used permissions by botware and benign applications. The botware features range from 18.570 % to 97.860 % while from 1.430% to 51.430% for benign applications. The average percent of requested permissions of botware is 54.249% and 11.323% for benign applications. This enormous difference shows

that botware applications request a much bigger number of permissions features as compared to benign applications. For instance, one of the common features of botware and benign applications is the INTERNET permission as shown in **Fig. 4(a)**. Here, the requests generated by botware applications was 97.86%, while 51.43% for benign applications. It shows that botware applications are usually generating more requests as compared to benign applications. Therefore, the smartphones users should be aware of the botware's susceptible permissions requests during the installation of an Android application. The comparison of benign and botware applications with respect to API calls is given in **Fig. 4(b)**. The obtained results show that botware applications use more API calls as compared to benign applications. At the same time, almost the same usage ratio can be seen for the services and the broadcast receivers.



(a) Permissions (b) API Calls

Fig. 4. The number of requested permissions and activities of botware and benign applications

4.3. Smart Learner

The smart learner takes data from the feature extractor component and generates the pattern, based on the assigned frequencies, by using the Apriori algorithm in WEKA tool [52]. The Apriori algorithm was chosen to identify the patterns of significant features combination [53]. This algorithm deals with the subset of events beyond examining the specific order of events. For easy processing, all extracted features were indexed, e.g. INTERNET with P_1, READ_PHONE_STATE with P_2, and so on. Similarly were indexed activities, services, broadcast receivers, and API calls.

The Apriori algorithm takes as an input the dataset B_{bot} that contains a full set of features of n botnet applications. Let $I = \{P_1, P_2, P_3, \ldots\ldots, P_n, A_1, A_2, A_3, \ldots\ldots, A_n, B_1, B_2, B_3, \ldots\ldots, B_n, S_1, S_2, S_3, \ldots\ldots, S_n, AP_1, AP_2, AP_3, \ldots\ldots, AP_n, \}$ be an instance of B_{bot}. The Apriori algorithm

begins by pinpointing the individual repeated items in the B_{bot} dataset and extending them to substantial sets as much as that item sufficiently often appears. For example, $A = \{P_1, A_1, B_1, S_1, AP_1\}$ is a candidate item set. There are two values need to be known in advance for the Apriori algorithm, which are the support and the confidence levels, for the calculation of the frequency of features, used in the B_{bot} dataset. In this case, the support value of the candidate itemset $\{P_1, A_1, B_1, S_1, AP_1\}$ was computed by formula (2).

$$Support\ (P1, A1, B1, S1, AP1) = \frac{\text{Number of applications that contains } P1,A1,B1,S1,AP1 \text{ in } B_{bot}}{Total\ number\ of\ Applications\ in\ B_{bot}} \quad (2)$$

The candidate item set is considered as a frequent itemset or a relevant pattern, only if $support\ (P1, A1, B1, S1, AP1) \geq$ threshold (t), where t is a user-defined threshold. In this study, we set 0.5 as a minimum threshold [54]. The same process was applied for frequent itemset identification for benign applications. **Table 3** shows the identified unique patterns for botware applications. The pattern ID represents the indexed ID of used features pattern, while the support values are calculated for each used feature pattern (UP).

The support values in **Table 3** show that the botware applications mostly utilize the combination of the features, such as the INTERNET, RECEIVE_SMS, WRITE_EXTERNAL_STORAGE, com.clientsoftware.ServiceStartr, com.phone.callcorexy.x-y.SReceiver, and SYSTEM_A-LERT_WINDOW to perform the malicious activities on the smartphone and to steal a piece of sensitive information from it. Once this mentioned malicious activity is performed, it sends the stolen information to the C&C server through the communication channel. In this malicious activity, INTERNET permission provides the connection between smartphone and C&C server, while the RECEIVE_SMS permission received the updates and commands about the activity. Botware applications having the INTERNET, WRITE_SMS and SEND_SMS permissions enable can send SMS and MMS to premium numbers with the combination of MAIN ACTIVITY and TOUCHSCREEN. It is reported that sending of SMS and MMS to the premium numbers can cause financial losses [31].

Furthermore, the location related to permissions, such as ACCESS_COARS_LOCATION and ACCESS_FINE_LOCATION is used for the smartphone data collection and network location data gathering. The pattern UP29 is the combination of INTERNET, ACCESS_NETWORK_STATE, com.google.android.mms.-LiveReceiver and com.clie-ntsoftware.SDCardServiceSt-arter is used to handle the connection between bots and botnets.

Table 3. Unique Patterns for the Botware and Benign used Features

Used Pattern	Botware	Benign	Used Pattern	Botware	Benign	Used Pattern	Botware	Benign
UP1	0.9731	0.0269	UP15	0.7349	0	UP29	0.6269	0.005
UP2	0.9374	0.1059	UP16	0.722	0	UP30	0.6254	0
UP3	0.9151	0.0773	UP17	0.7214	0	UP31	0.6235	0.0556
UP4	0.9045	0.137	UP18	0.7202	0.0216	UP32	0.6229	0.0235
UP5	0.9009	0.1831	UP19	0.7178	0.1945	UP33	0.6216	0
UP6	0.8856	0	UP20	0.7173	0.0731	UP34	0.605	0
UP7	0.8806	0.1059	UP21	0.7089	0	UP35	0.6014	0
UP8	0.7949	0.0731	UP22	0.7015	0.0556	UP36	0.5945	0.1718
UP9	0.7867	0.1363	UP23	0.6974	0	UP37	0.5831	0.0269
UP10	0.7796	0.0235	UP24	0.6828	0.0773	UP38	0.5565	0
UP11	0.7774	0.1831	UP25	0.6773	0	UP39	0.5363	0.1363

UP12	0.7758	0.2055	UP26	0.6731	0.1945	UP40	0.5338	0.1718
UP13	0.7712	0.1831	UP27	0.6544	0	UP41	0.5059	0
UP14	0.7705	0.0216	UP28	0.6534	0	UP42	0.5000	0

4.4. Features Refiner

This component takes input from the smart learner and use it to achieve high accuracy in a specified model of the problem domain. In the feature's extraction process, many features are irrelevant and redundant. Such redundant features may create an issue for the learning algorithm, which includes misleading, overfitting, reducing accuracy and generality, time and space complexity [55]. It may increase the crucial effects since mobile devices have limited resources [56]. Selecting the most appropriate features to form a large feature vector space can significantly change the accuracy of the predicted model. In order to increase the efficiency and accuracy of botnet detection, it is important to implement an effective feature refining process. The basic idea of feature refining is to select the most related to botware applications features.

For this purpose, the information gain (IG) algorithm was applied to the malicious dataset [57]. It is the most common feature selection method in botware detection techniques [58]. While most of the methods followed the feature ranking approach based on specific metrics, the value was computed to score each feature individually. According to this measure, a feature Y is regarded as a better indicator than a feature Z for a class X, if $IG(X|Y) > IG(X|Z)$. Approach ranks the features by the information gained and select the high ranked features. For the computing of information gain value, WEKA is used [52]. The Android applications are inspected manually, and then the permission, activities, broadcast receivers, services, and API_Calls are noted. All these features have a specific relationship with respect to botnet activities.

4.5. Machine Learning Modelling

This component takes input data from the smart learner component. Once the features are refined, the next stage is to train the machine learning classifier. In anomaly detection, various machine-learning techniques can be used to train a model and test the datasets [60; 68; 80]. Since the dataset used in this research study is labelled and has two target classes - Benign and Botware, supervised learning technique was selected. In supervised learning, the algorithm uses a labelled dataset, which provides a possibility to evaluate the accuracy of the model on training data. This why a supervised learning technique is generally more accurate and reliable as compare to unsupervised learning. In this study, the selecting classifier is based on the performance, a number of classes and the ranking criteria of features. Since the prototype of the framework was implemented in Java, WEKA, which is a data mining software also written in Java [52, 53], was used.

During the classification phase, several machine-learning algorithms were selected to recognize botware applications based on classification accuracy. Choosing an appropriate classifier ultimately demonstrates the accuracy of the detection framework. To choose a proper machine-learning algorithm for classification, the following requirements were considered: (1) diversity (the number of static features were taken from the multiple domains); (2) sparse feature set (the supreme features set were finally selected for the framework evaluation); (3) scalability (the system should be able to deal with future requirements of users); and (4) performance (algorithms for testing and training should provide a prompt response to a user). Given these considerations, support vector machine (SVM), Random Forest, J-48, simple logistic regression (SLR), and Naïve Bayes were selected as the classification algorithms to establish and test the proposed framework.

5. Experimental Results

This section discusses the obtained results for individual and combined features.

5.1. Performance Evaluation Parameters

This section describes the evaluating process of the proposed botnet detection framework. The performance has been evaluated using five matrices, namely True Positive Rate (TPR), False Positive Rate (FPR), Precision, F-measure, and Accuracy, and next compared to the existing state-of-the-art detection techniques. Literature review shows that these parameters are commonly used for the evaluation of the malware detection frameworks and tools. **Table 4** shows the evaluation parameters with the description and the corresponding formulas.

Table 4. Performance Evaluation Parameters of the Proposed Framework

Parameters	Description	Formula
True Positive Rate (TPR)	When it is actually Botware, how often it predicted as Botware	$TPR = TP/(TP + FN)$
False Positive Rate (FPR)	When it is actually Botware, how often it predicted as Benign	$FPR = FP/(FP + TN)$
Accuracy (ACC)	The number of occurrences, correctly classified	$ACC = \frac{(TP + TN)}{(TP + TN + FP + FN)}$
F-Measure (F)	A measure that combines precision (P) and recall (R) is the harmonic mean of precision and recall	$F - Measure = 2(\frac{P * R}{P + R})$
Positive Predictive Value (PPV), Precision	The ratio of predicted positives, which are actually positive	$PPV = \frac{TP}{TP + FP}$

5.2. Results

First, the experiments were conducted separately on the permissions, the activities, the broadcast receivers, the services, and the API_calls features. These features were next united to conduct experiments on the combined features set. The obtained results are summarized in **Table 5**. In general, random forest obtained higher accuracy rate for all selected features as compared to other classifiers (except the services feature). Considering only the accuracy of evaluation, the random forest was found the best classifier. TPR, FPR, precision, and F-measure were used to evaluate the proposed framework and to obtain more accurate results. The experiments were performed on all selected classifiers by considering the mentioned parameters.

Once researchers obtained the results for all selected features by considering different classifiers, they combined the features to perform further experiments. **Table 6** summarizes the obtained results. Random forest achieved the highest accuracy 98.2%, while TPR is 0.7880, precision is 0.8893 and FPR is 0.1140. However, experiments produce a low F-measure of 0.7457. **Table 6** shows that the random forest achieved more accurate results by considering the combined features.

Table 5. Results, obtained by considering separate features

Features	Algorithms	TPR	FPR	Precision	F-Measure	Accuracy
Permissions	Random Forest	**0.9114**	0.0626	**0.9127**	**0.9358**	**93.4000**
	Naïve Bayes	0.8802	0.0803	0.9016	0.9110	91.1200
	SVM	0.8912	0.1118	0.8555	0.8832	87.0300
	SLR	0.9000	0.1100	0.8443	0.7633	86.9300
	J-48	0.9011	**0.0100**	0.9123	0.9128	90.6400
Activities	Random Forest	**0.7927**	0.1673	**0.8814**	**0.7997**	**81.9300**
	Naïve Bayes	0.7250	0.1772	0.8809	0.7499	66.0200
	SVM	0.7918	**0.1138**	0.7598	0.7634	79.9100
	SLR	0.7111	0.1889	0.7741	0.7259	77.1800
	J-48	0.7674	0.1326	0.8684	0.7912	80.7100
Broadcast Receivers	Random Forest	**0.8567**	**0.0816**	**0.8940**	**0.8602**	**86.6800**
	Naïve Bayes	0.6600	0.3267	0.7960	0.5250	81.0200
	SVM	0.8180	0.0882	0.7932	0.8461	81.8000
	SLR	0.6667	0.2700	0.6360	0.6250	83.0200
	J-48	0.7923	0.1069	0.8714	0.7971	85.2300
Services	Random Forest	0.7150	**0.0519**	**0.8632**	**0.7517**	73.8600
	Naïve Bayes	0.6753	0.5820	0.5780	0.7281	68.1800
	SVM	**0.7319**	0.1272	0.8503	0.7359	**74.4800**
	SLR	0.7142	0.1452	0.8461	0.6800	66.1800
	J-48	0.6620	0.2662	0.6380	0.6470	71.4200
API_Calls	Random Forest	0.7580	0.1840	0.8393	**0.7457**	**78.2600**
	Naïve Bayes	0.7550	0.1850	0.8270	0.7330	74.5700
	SVM	**0.7630**	0.2400	**0.8410**	0.7510	76.2900
	SLR	0.7533	0.1441	0.8237	0.7224	73.5400
	J-48	0.7377	**0.1288**	0.8348	0.7402	75.3300

Table 6. Results, obtained by considering combined features

Algorithms	TPR	FPR	Precision	F-Measure	Accuracy
Random Forest	**0.7880**	**0.1140**	**0.8893**	0.7457	**98.2000**
Naïve Bayes	0.7550	0.1850	0.8270	0.7330	92.5700
SVM	0.7630	0.2400	0.8410	**0.7510**	88.2900
SLR	0.7533	0.1441	0.8237	0.7224	83.5400
J-48	0.7377	0.1288	0.8348	0.7402	84.3300

5.3. Comparison with prior work

The need for new botnet detection techniques is a crucial security requirement of Android devices. The proposed framework is an effective and efficient botnet detection approach for Android devices. The proposed framework has high accuracy with the obtained value of 98.20%, compared to Rashidi & Fung (2016), obtained 89.30% by utilizing the permissions feature. Sanz et al. (2013), obtained 94.83% accuracy by using the permissions and API_Calls features as shown in **Table 7**. The accuracy obtained by Yerima et al., (2014) was 92.10 % and Peiravian & Zhu (2013) showed the accuracy score of 96%. Overall, the proposed framework is more accurate when compared with Rashidi & Fung, (2016), Sanz et al., (2013), Yerima et al., (2014) and Peiravian & Zhu, (2013). The main reason for the better results of the proposed framework is adding the refining component and use of the features selection approach. With using of refining component, the proposed framework ignored the features, which are not vulnerable to Android botnets attacks.

Table 7. Comparison with Existing Techniques

References	Approach	Platform	Features	Accuracy
[18]	Static	Android	Permissions & API_Calls	94.83%
[42]	Static	Android	Permissions & API_Calls	96%
[59]	Static	Android	Permissions	92.10%
[44]	Static	Android	Permissions	89.30%
Proposed Framework	Static	Android	Permissions, Activities, Broadcast Receivers, Services & API_Calls	98.20%

Fig. 5 shows the accuracy represented by a bar graph. It depicted that the proposed framework has high accuracy as compared to other considered studies.

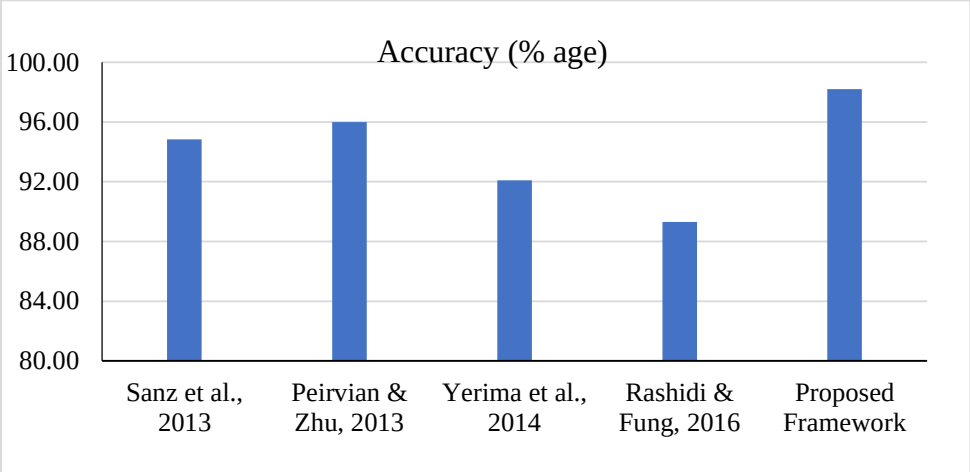


Fig. 5. Comparison of accuracy of the proposed framework with existing solutions

The study [18] proposed to discover malicious applications by using the difference in the Android application permissions that the application request upon installation. Android permissions are coarse-grained [60]. For example, the INTERNET permission does not have the capability to restrict access to a particular Uniform Resource Locator (URL). READ_PHONE_STATE allows an app to identify whether the device rings or is on hold. At the same time, it also allows the app to read sensitive information, such as device identifiers. Permissions, as WRITE_SETTINGS, CAMERA are broadly defined, violating the least privilege access principle. Access to WRITE_CONTACTS or WRITE_SMS does not imply access to READ_CONTACTS or READ_SMS permissions [49]. Thus, permissions are not hierarchical, and a developer must separately request them. In addition, at the installation time, a user is forced to either grant all permissions or deny the application installation. Hence, the dangerous permissions cannot be avoided at the installation time. Moreover, the users cannot differentiate between the necessity and its misuse, which may expose exploitation.

In approach [61], authors extracted static features from the androidmanifest.xml file of 666 Android applications. They used machine-learning techniques, such as K-Nearest Neighbors (K-NN), Decision Trees, Bayesian networks, Support Vector Machines (SVM) to detect malicious applications. K-NN was used due to its simplicity in classifying instances into different classes; at the same time, a decision tree allowed an easy implementation. The

Bayesian network was employed for determining the probability of a hypothesis certainty. On the long run, SVM was used to solve the problem of kernel functions, which may lead the technique to the non-linear classification surface [62].

The study [42] developed a machine-learning framework to analyze benign and malicious applications by using the permissions and API calls as the features. In their study, authors examined in total 2400 Android malware and benign applications to extract the requested permissions and API calls. Authors achieved 96.88% of accuracy by selecting a high number of features. This approach shows that the accuracy can be improved when the number of features is selected from both malware and benign applications. Extracting and selecting a bigger number of features allow to improve the accuracy, while the memory and time use are increased [60]. This framework was specifically focused on malware applications. However, using these approaches mobile botnets cannot be detected in Android devices. This study has almost the same nature as an aforementioned study on MAMA [18], which use the same features. Researchers achieved 94.83% accuracy when 130 features were examined.

Yerima et al. (2014) developed a proactive machine-learning approach based on Bayesian classification and aimed to detect zero-day Android malware attacks using static analysis approach [59]. This approach has three main components, which are decompression, identification, and classification. First, an application is decompiled by reverse engineering by using Dalvik VM to extract features from the AndroidManifest.xml and .DEX classes [63]. All extracted features are stored in a file with the .csv extension for further analysis. In the identification step, this component converts the extracted features file to a readable form for further analysis. In the classification process, the Bayesian algorithm classifies the malware and benign applications as a result [52]. This approach was based on the large existing set of malware of 49 families. Specifically, this technique achieved approximately 92.1% accuracy by using a set of 30 static features.

Karim et al. (2015) research work focused on Android malware identification only. By using their approach, a mobile botnet cannot be detected. Authors used a few numbers of features as compared to the study [42], which significantly affects the accuracy of malware detection. According to the study [64], the time is taken for features extraction and computation is decreased to 77%. However, another study showed that by using these features the time taken for features extraction is increased at 28% [64]. Thus accuracy and time consumption are the main issues need to be addressed.

BotTracer [44] is a clustering-based method to detect bot users controlled by the same masters. The main method of the proposed method is to plot the Android users in various groups based on their similarity. It is an Android permissions recommendation framework, which allows users to grant application permissions requests in a fine-grained manner. In Android applications, permissions are the main factor during installation that cannot be ignored. For example, the INTERNET permission does not have the capability to restrict access to a particular Uniform Resource Locator (URL), READ_PHONE_STATE allows an app to identify whether the device rings or is on hold [65].

The key idea behind BotTracer technique [44] is to collect the expert users' responses to a permission request and recommend them to inexperienced users. BotTracer uses two phases with the first analysis of the common features of the bot users for the construction of a similarity graph. Then, a hierarchical clustering method groups those users based on the distance, which is defined using similarity. This approach has many limitations in terms of botnet detection. First, it detects the simulated bot from user profiles only. Secondly, it considers the DEX bytecode, while it ignores the native code and app resources. Thirdly, the

opcode sequence does not include high-level semantic information and hence generates false negatives. With these limitations, the smart adversary can easily bypass this technique using code transformation, such as inserting junk bytecode, restructure methods, and alter control flow to evade the BotTracer prototype.

6. Conclusion and Future Work

Botnets have become one of the most significant threats to Internet-connected smartphones. An increasing number of botnet attacks brings challenges for the detection of harmful software. To contribute to this domain, this study proposes a new botnet detection framework using static feature extraction and machine learning algorithms. In order to identify the C&C structure and the most relevant features, a considerable number of botnet applications were analyzed. The static features, such as permissions, activities, broadcast receivers, services, and API calls were extracted and further analysed. The extraction of the features, the used patterns and the feature refining component differentiate the proposed technique from others. By using the information gain and the Apriori algorithms, the most prominent features were identified that gives the possibility to classify an Android application as a mobile botnet and to prevent an attack. The selected features were used in the machine-learning algorithm for classification, which achieved 98.2% accuracy with very low FPR (0.1140). Among the different classifiers used, the random forest gave the best results. The obtained results show that the proposed framework accurately identifies botnet applications. The limitation of the proposed approach is that it is based on static features; the accuracy of prediction can be improved by expanding the functionality by dynamic properties. It will be a direction of our future work.

References

[1] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features," *IEEE Access*, vol. 6, pp. 31798-31807, 2018, Article (CrossRef Link).

[2] L. Goasduff and C. Pettey, "Gartner says worldwide smartphone sales soared in fourth quarter of 2019 with 47 percent growth," Visited March, 2019.

[3] W. Fan, Y. Sang, D. Zhang, R. Sun, and Y. a. Liu, "DroidInjector: a process injection-based dynamic tracking system for runtime behaviors of android applications," *Computers & Security*, vol. 70, pp. 224-237, 2017. Article (CrossRef Link).

[4] I. Statista, "Number of mobile phone users worldwide (in billions) Retrieved from http://www.statista.com/, Accessed on 21 January 2019.

[5] V. Vanitha, "Android Malware Analysis: A Survey," *International Journal of Innovations & Advancement in Computer Science*, Vol. 6, no. 1. January 2017.

[6] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection," in *Proc. of USENIX Security Symposium*, vol. 5, no. 2, pp. 139-154, 2008. Article (CrossRef Link).

[7] Z. Inayat, A. Gani, N. B. Anuar, S. Anwar, and M. Khurram Khan, "Cloud-Based Intrusion Detection and Response System: Open Research Issues, and Solutions," *Arabian Journal for Science and Engineering*, 42, 399-423, 2017. Article (CrossRef Link).

[8] R. Nigam, "A timeline of mobile botnets," *Virus Bulletin*, March, 2015. Retrieved from https://www.virusbulletin.com/blog/2015/03/paper-timeline-mobile-botnets/

[9] D. F. Guo, A.-F. Sui, and T. Guo, "A behavior analysis based mobile malware defense system," in *Proc. of 6th International Conference on Signal Processing and Communication Systems (ICSPCS), IEEE*, pp. 1-6, 2012. Article (CrossRef Link).

[10] A. Naser, M. A. Majid, M. F. Zolkipli, and S. Anwar, "Trusting cloud computing for personal files," in *Proc. of International Conference on Information and Communication Technology Convergence (ICTC), IEEE,* pp. 488-489, 2014. Article (CrossRef Link).
[11] S. Anwar, M. F. Zolkipli, Z. Inayat, B. Odili, M. Ali, and J. M. Zain, "Android Botnets: A Serious Threat to Android Devices," *Pertanika Journal of Science and Technology*, 26(1), 37-70, 2018. Article (CrossRef Link).
[12] S. Anwar, J. M. Zain, Z. Inayat, R. U. Haq, A. Karim, and A. N. Jabir, "A Static Approach Towards Mobile Botnet Detection," in *Proc. of Electronic Design (ICED), 2016 3rd International Conference on, Phuket, Thailand, IEEE*, pp. 563-567, 2016. Article (CrossRef Link).
[13] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee, "Evaluating bluetooth as a medium for botnet command and control," *Detection of Intrusions and Malware, and Vulnerability Assessment: Springer*, pp. 61-80, 2010. Article (CrossRef Link).
[14] Y. Zeng, K. G. Shin, and X. Hu, "Design of SMS commanded-and-controlled and P2P-structured mobile botnets," in *Proc. of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 137-148, 2012. Article (CrossRef Link).
[15] S. Anwar, J. Mohamad Zain, M. F. Zolkipli, and Z. Inayat, "A Review Paper on Botnet and Botnet Detection Techniques in Cloud Computing," in *Proc. of ISCI 2014 - IEEE Symposium on Computers & Informatics*, 2014.
[16] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar, "Intrusion Response Systems: Foundations, Design, and Challenges," *Journal of Network and Computer Applications*, 62, 5374, 2016. Article (CrossRef Link).
[17] S. Anwar et al., "Cross-VM Cache-based Side Channel Attacks and Proposed Prevention Mechanisms: A survey," *Journal of Network and Computer Applications*, vol. 93, pp. 259-279, 2017. Article (CrossRef Link).
[18] B. Sanz et al., "MAMA: manifest analysis for malware detection in android," *Cybernetics and Systems*, vol. 44, no. 6-7, pp. 469-488, 2013. Article (CrossRef Link).
[19] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for android," *Pervasive and Mobile Computing*, vol. 32, pp. 15-25, 2016. Article (CrossRef Link).
[20] S. Anwar, J. M. Zain, M. F. Zolkipli, Z. Inayat, A. N. Jabir, and B. Odili, "Response Option for Attacks Detected by Intrusion Detection System," in *Proc. of The 4th International Conference on Software Engineering and Computer System*, 195-200, 2015. Article (CrossRef Link).
[21] S. Anwar et al., "From intrusion detection to an intrusion response system: fundamentals, requirements, and future directions," *Algorithms*, vol. 10, no. 2, p. 39, 2017. Article (CrossRef Link).
[22] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-art," *IEEE Communications Surveys & Tutorials*, 19(1), 303-324, 2017. Article (CrossRef Link).
[23] G. Suarez Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961-987, 2014. Article (CrossRef Link).
[24] X. Meng and G. Spanoudakis, "MBotCS: A mobile botnet detection system based on machine learning," in *Proc. of International Conference on Risks and Security of Internet and Systems*, pp. 274-291, 2015. Article (CrossRef Link).
[25] H. Binsalleeh et al., "On the analysis of the zeus botnet crimeware toolkit," in *Proc. of Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on,* pp. 31-38, 2010. Article (CrossRef Link).
[26] M. Ge, J. B. Hong, W. Guttmann, and D. S. Kim, "A framework for automating security analysis of the internet of things," *Journal of Network and Computer Applications*, vol. 83, pp. 12-27, 2017. Article (CrossRef Link).
[27] A. Flo and A. Josang, "Consequences of botnets spreading to mobile devices," in *Proc. of Short-Paper Proceedings of the 14th Nordic Conference on Secure IT Systems (NordSec 2009)*, pp. 37-43, 2009. Article (CrossRef Link).

[28] M. Kwon et al., "Development and validation of a smartphone addiction scale (SAS)," *PloS one*, vol. 8, no. 2, p. e56936, 2013. Article (CrossRef Link).
[29] G. Suarez-Tangil, J. E. Tapiador, P. Pens-Lopez, and J. Blasco, "DENDROID: A text mining approach to analyzing and classifying code structures in Android malware families," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104-1117, Mar 2014. Article (CrossRef Link).
[30] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android Botnets: What URLs are Telling Us," *in Network and System Security: Springer*, pp. 78-91, 2015. Article (CrossRef Link).
[31] E. Johnson and I. Traore, "Sms botnet detection for android devices through intent capture and modeling," in *Proc. of IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, pp. 36-41, 2015. Article (CrossRef Link).
[32] G. Kirubavathi and R. Anitha, "Structural analysis and detection of android botnets using machine learning techniques," *International Journal of Information Security*, 17, 153-167, 2018. Article (CrossRef Link).
[33] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *information security technical report*, vol. 14, no. 1, pp. 16-29, 2009. Article (CrossRef Link).
[34] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1-18, 2014. Article (CrossRef Link).
[35] M. B. Mollah, M. A. K. Azad, and A. Vasilakos, "Security and privacy challenges in mobile cloud computing: Survey and way ahead," *Journal of Network and Computer Applications*, vol. 84, pp. 38-54, 2017. Article (CrossRef Link).
[36] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proc. of 5th international conference on Malicious and unwanted software (MALWARE), IEEE*, pp. 55-62, 2010. Article (CrossRef Link).
[37] H. Tiirmaa-Klaar, J. Gassen, E. Gerhards-Padilla, and P. Martini, "Botnets: how to fight the ever-growing threat on a technical level," *Botnets: Springer*, pp. 41-97, 2013. Article (CrossRef Link).
[38] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22-37, 2015. Article (CrossRef Link).
[39] Google Play Store, Retrieved from https://play.google.com/store?hl=en_GB. Access on 24 Feb, 2020.
[40] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *Proc. of 27th International Conference on Advanced Information Networking and Applications (AINA), IEEE*, pp. 121-128, 2013. Article (CrossRef Link).
[41] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An hmm and structural entropy based detector for android malware: An empirical study," *Computers & Security*, vol. 61, pp. 1-18, 2016. Article (CrossRef Link).
[42] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *Proc. of 25th International Conference Tools with Artificial Intelligence (ICTAI), IEEE*, pp. 300-305, 2013. Article (CrossRef Link).
[43] A. Karim, R. Salleh, M. K. Khan, A. Siddiqa, and K. K. R. Choo, "On the analysis and detection of mobile botnet applications," *Journal of Universal Computer Science*, 22(4), 567-588, 2016. Article (CrossRef Link).
[44] B. Rashidi and C. Fung, "BotTracer: Bot user detection using clustering method in RecDroid," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 1239-1244, 2016. Article (CrossRef Link).
[45] VirusTotal-Free online virus, malware and URL scanner. Retrieved from https://www.virustotal.com/en
[46] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," *Ndss*, vol. 14, pp. 23-26, 2014. Article (CrossRef Link).

[47] UNB ISCX Android Botnet DataSet. Retrieved on April, 2018. http://www.unb.ca/research/iscx/dataset/iscx-android-botnet-dataset.html
[48] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," *IEEE Symposium on Security and Privacy (SP)*, pp. 95-109, 2012. Article (CrossRef Link).
[49] Android developers permissions. Retrieved by April, 2018 from Article (CrossRef Link).
[50] Reverse engineering, Malware and goodware analysis of Android applications. Retrieved from by April 2018 https://github.com/androguard/androguard
[51] N. Kheir, F. Tran, P. Caron, and N. Deschamps, "Mentor: Positive DNS reputation to skim-off benign domains in botnet C&C blacklists," *IFIP Advances in Information and Communication Technology*, vol. 428, pp. 1-14, 2014. Article (CrossRef Link).
[52] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10-18, 2009. Article (CrossRef Link)
[53] T. C. Smith and E. Frank, "Introducing machine learning concepts with WEKA," *Statistical genomics: Methods and protocols*, pp. 353-378, 2016. Article (CrossRef Link).
[54] A.-D. Schmidt et al., "Static analysis of executables for collaborative malware detection on android," in *Proc. of Communications, 2009. ICC'09. IEEE International Conference on*, pp. 1-5, 2009. Article (CrossRef Link).
[55] E. Alparslan, A. Karahoca, and D. Karahoca, "BotNet Detection: Enhancing Analysis by Using Data Mining Techniques," *Advances in Data Mining Knowledge Discovery and Applications*, 2012. Article (CrossRef Link).
[56] M. Ali, J. M. Zain, M. F. Zolkipli, and G. Badshah, "Mobile cloud computing & mobile battery augmentation techniques: A survey," in *Proc. of IEEE Student Conference on Research and Development (SCOReD)*, pp. 1-6, 2014. Article (CrossRef Link).
[57] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161-190, 2011. Article (CrossRef Link).
[58] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in API calls with machine learning algorithms for malware detection," in *Proc. of 2nd ACM Proceedings workshop on Security and artificial intelligence*, pp. 55-62, 2009. Article (CrossRef Link).
[59] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25-36, 2014. Article (CrossRef Link).
[60] K. Sokolova, C. Perez, and M. Lemercier, "Android application classification and anomaly detection with graph-based permission patterns," *Decision Support Systems*, vol. 93, pp. 62-76, 2017. Article (CrossRef Link).
[61] P. Narang, C. Hota, and H. T. Sencar, "Noise-resistant mechanisms for the detection of stealthy peer-to-peer botnets," *Computer Communications*, vol. 96, pp. 29-42, 2016. Article (CrossRef Link).
[62] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Generation Computer Systems*, 78, 987-994, 2018. Article (CrossRef Link).
[63] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121-134, 2017. Article (CrossRef Link).
[64] A. Karim, S. A. A. Shah, R. B. Salleh, M. Arif, R. Md Noor, and S. Shamshirband, "Mobile botnet attacks - an emerging threat: Classification, review and open issues," *KSII Transactions on Internet and Information Systems*, vol. 9, no. 4, pp. 1471-1492, 2015. Article (CrossRef Link).
[65] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. of the Eighth Symposium on Usable Privacy and Security*, pp. 1-14, 2012. Article (CrossRef Link).
[66] Peskun, Peter H, "Optimum monte-carlo sampling using markov chains," *Biometrika*, 60(3), 607-612, 1973. Article (CrossRef Link).

[67] Thakur, Deepak; Gera, Tanya; Singh, Jaiteg, "Android Anti-malware Techniques and Its Vulnerabilities: A Survey," *Smart Innovations in Communication and Computational Sciences, Springer Singapore*, pp. 315-328, 2018. Article (CrossRef Link).
[68] Narayan, V., & Shaju, B, "Malware and Anomaly Detection Using Machine Learning and Deep Learning Methods," *Handbook of Research on Machine and Deep Learning Applications for Cyber SecurityHershey, PA: IGI Global*, pp. 104-131, 2020. Article (CrossRef Link).
[69] Awad, S. G. Sayed and S. A. Salem, "Collaborative Framework for Early Detection of RAT-Bots Attacks," *IEEE Access*, vol. 7, pp. 71780-71790, 2019. Article (CrossRef Link).
[70] Aminuddin, N. I., & Abdullah, Z. "Android Trojan Detection Based on Dynamic Analysis," *Advances in Computing and Intelligent System*, 1(1), 2019. Article (CrossRef Link).
[71] Attia Qamar, Ahmad Karim, Victor Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Generation Computer Systems*, Vol. 97, pp. 887-909, 2019. Article (CrossRef Link).
[72] Adebayo, O. S., & Aziz, N. A., "The Trend of Mobile Malwares and Effective Detection Techniques," *Multigenerational Online Behavior and Media Use: Concepts, Methodologies, Tools, and Applications, Hershey, PA: IGI Global*, pp. 668-682, 2019. Article (CrossRef Link).
[73] Pedram Amini, Reza Azmi, Muhammad Amin Araghizadeh, "Analysis of Network Traffic Flows for Centralized Botnet Detection," *Journal of Telecommunication, Electronic and Computer Engineering*, Vol 11, No 2, pp. 7-17, 2019. Article (CrossRef Link).
[74] Shisrut Rawat, Aishwarya Srinivasan, Vinayakumar R, "Intrusion detection systems using classical machine learning techniques versus integrated unsupervised feature learning and deep neural network," *Cornell University Preprint*, 2019. Article (CrossRef Link).
[75] Talal, M., Zaidan, A.A., Zaidan, B.B. et al., "Comprehensive review and analysis of anti-malware apps for smartphones," *Telecommun Syst*, vol. 72, pp. 285-337, 2019. Article (CrossRef Link).
[76] Yaocheng Zhang, Wei Ren, Tianqing Zhu, Yi Ren, "SaaS: A situational awareness and analysis system for massive android malware detection," *Future Generation Computer Systems*, Vol. 95, pp. 548-559, 2019. Article (CrossRef Link).
[77] Luo, Shiqi, et al., "Android Malware Analysis and Detection Based on Attention-CNN-LSTM," *Journal of Computers*, vol. 14, no. 1, pp. 31-43, 2019. Article (CrossRef Link).
[78] Seul-Ki Choi; Taejin Lee; Jin Kwak, "A Study on Analysis of Malicious Code Behavior Information for Predicting Security Threats in New Environments," *KSII Transactions on Internet & Information Systems*, Vol. 13, Issue 3, pp. 1611-1625, Mar2019. Article (CrossRef Link).
[79] S. Esmaeili and H. R. Shahriari, "PodBot: A New Botnet Detection Method by Host and Network-Based Analysis," in *Proc. of 2019 27th Iranian Conference on Electrical Engineering (ICEE), Yazd, Iran*, pp. 1900-1904, 2019. Article (CrossRef Link).
[80] Laraib U. Memon, Narmeen Z. Bawany, Jawwad A. Shamsi, "A comparison of machine learning techniques for android malware detection using apache spark," *Journal of Engineering Science and Technology*, Vol. 14, No. 3, 1572 – 1586, 2019. Article (CrossRef Link).

SHAHID ANWAR is an Assistant Professor in the Department of Software Engineering, the University of Lahore, Pakistan. He received PhD degree in Computer Science from the Universiti Malaysia Pahang in the year of 2019. He has completed his Master in Computer Science (M.Sc.-CS) from Hazara University Pakistan in the year of 2009. He has published over 18 research peer review articles in different journals and International conferences. His current research interests include Android security and network security.

Mohamad Fadli Zolkipli is an Associate Professor at the Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang. He completed his doctorate degree in Computer Science at Universiti Sains Malaysia (USM) in 2012. His career in academia started when he joined KUKTEM in July 2002 as academician. His teaching expertise includes Data Communication and Networking, Switching & Routing, and Network Security. His research interests cover the broad area of digital security. He has published numerous articles in the area of computer systems and networking especially in security domain such as intrusion detection systems, malware analysis and cloud security. As a part of research community, he also involves as a reviewer for conferences and journals. He is currently active in supervising research students of master and doctorate degrees.

Vitaliy Mezhuyev received a specialist degree in informatics from Berdyansk State Pedagogical University (BSPU), Ukraine, in 1997. In 2002, he received a PhD in Educational Technology from Kiev National Pedagogical University and, in 2012, an ScD in Information Technology from Odessa National Technical University, Ukraine. From 2004 until 2014, he was a Head of the Department of Informatics and Software Engineering at BSPU, Ukraine. From 2014 until 2019 he was a Professor at Faculty of Computer Systems and Software Engineering in University Malaysia Pahang, Head of the Software Engineering Research Group. Now he is a Professor at Institute of Industrial Management in FH JOANNEUM University of Applied Sciences, Austria. During his career, Vitaliy Mezhuyev participated in the multiple international scientific and industrial projects, devoted to the formal modelling, design, and development of advanced software systems as a network-centric real-time operating system; IDEs for the automation of development of parallel real-time applications; tools for specification, verification and validation of software products; visual environment for metamaterials modelling and others. His current research interests include formal methods, metamodeling, safety modelling and verification of software systems, IoT, and the design of cyber-physical systems.

ZAKIRA INAYAT is a Senior Lecturer in Department of Computer Science and Information Technology, University of Engineering and Technology Peshawar, Pakistan. She received a Ph.D. in Computer Science from University of Malaya, Malaysia. She served as a Research Assistant in High Impact Research Project (Mobile Cloud Computing) fully funded by Malaysian Ministry of Higher Education. She has completed her Master degree in Computer Science (M.Sc-CS) from Hazara University of Mansehra, Pakistan in 2006. She has published over 15 research articles in different journals and international conferences. Her current research interests include networks security, Smartphone Security, Cloud Computing, IoT, and Big Data.