




Root Exploit Detection and Features Optimization: Mobile Device and Blockchain Based Medical Data Management

Ahmad Firdaus¹ · Nor Badrul Anuar² · Mohd Faizal Ab Razak^{1,2} · Ibrahim Abaker Targio Hashem³ · Syafiq Bachok^{1,2} · Arun Kumar Sangaiah⁴ 

Received: 7 February 2018 / Accepted: 19 April 2018 / Published online: 4 May 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract

The increasing demand for Android mobile devices and blockchain has motivated malware creators to develop mobile malware to compromise the blockchain. Although the blockchain is secure, attackers have managed to gain access into the blockchain as legal users, thereby comprising important and crucial information. Examples of mobile malware include root exploit, botnets, and Trojans and root exploit is one of the most dangerous malware. It compromises the operating system kernel in order to gain root privileges which are then used by attackers to bypass the security mechanisms, to gain complete control of the operating system, to install other possible types of malware to the devices, and finally, to steal victims' private keys linked to the blockchain. For the purpose of maximizing the security of the blockchain-based medical data management (BMDM), it is crucial to investigate the novel features and approaches contained in root exploit malware. This study proposes to use the bio-inspired method of practical swarm optimization (PSO) which automatically select the exclusive features that contain the novel android debug bridge (ADB). This study also adopts boosting (adaboost, realadaboost, logitboost, and multiboost) to enhance the machine learning prediction that detects unknown root exploit, and scrutinized three categories of features including (1) system command, (2) directory path and (3) code-based. The evaluation gathered from this study suggests a marked accuracy value of 93% with Logitboost in the simulation. Logitboost also helped to predicted all the root exploit samples in our developed system, the root exploit detection system (RODS).

This article is part of the Topical Collection on *Mobile & Wireless Health*

Keywords Blockchain · Root exploit · Static analysis · Android · Machine learning

✉ Arun Kumar Sangaiah
arunkumarsangaiah@gmail.com

Ahmad Firdaus
firdausza@ump.edu.my

Nor Badrul Anuar
badrul@um.edu.my

Mohd Faizal Ab Razak
faizalabrazak@siswa.um.edu.my

Ibrahim Abaker Targio Hashem
targio_123@yahoo.com

Syafiq Bachok
syafiqbachok@siswa.um.edu.my

Introduction

In today's environment, people use mobile devices as their main gadgets to connect and communicate. In the healthcare sector, mobile devices are utilized for blockchain-based medical data management (BMDM) [1–3]. Blockchain is a digital ledger that records digital money transactions as well as other important patient-information that involves human lives [4]. A respectable security framework in blockchain attracts the healthcare sector particularly when it involves the Internet of Medical Things (IoMT) or the Internet of Health Things (IoHT), both of which, offer better security and privacy for medical data processing [5]. Nevertheless, current news recorded many hacked cases involving blockchain [6]. This is because once the attacker obtains the victim's private key, the hacker is able to sign into any legal transaction in the blockchain and from there, the attacker able to transfer all the medical data obtained to his/her address. In order to obtain

¹ Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, 26300 Kuantan, Pahang, Malaysia

² Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia

³ School of Computing & IT, Taylor's University, 47500 Subang Jaya, Selangor, Malaysia

⁴ School of Computing Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu 632014, India

this private key, the hacker needs to take over the operating system (OS) of the mobile device and this is done through the use of root exploit malware.

Root exploit is a type of malware attackers use to modify the Android OS kernel so that the attackers able to gain super-user privileges. When attackers gain root of the OS kernel, they also gain access to full administrator privileges. Through this, attackers are able to install other malware types, such as botnets, worms, or Trojans into the system. Attackers are able to evade detection by modifying the OS code, running its process stealthily, and bypassing permissions [7–9]. The number of root exploit has increased because of malware creators and homebrew communities (smartphone users who break the OS kernel to obtain a customized version of the OS). When a new version of the OS is released, malware creators may develop their own root exploit or they may wait for the homebrew community to determine ways to break the OS [10]. To combat this, security analysts have implemented two types of analysis namely, dynamic and static analysis, to detect root exploit.

Dynamic analysis investigates malware characteristics by running the application [11, 12]. This is exemplified by a study which inspected the network movement whilst detecting malware [13]. Another example is traced to DyHAP [14] which analyzed mobile network traffic to detect malware. Dynamic analysis has its own drawbacks. Its coverage is limited since it monitors the application's behavior within a limited time only. Malware behaviors which exceed the time of the experiment are thus unexplored thereby obscuring certain malware activities and overlooking certain parts of the investigation. In contrast, static analysis scrutinizes the application codes without executing the malware. In addition, this analysis only requires fewer resources (low specifications of hardware) and exhibits fast processing speed [15]. During static analysis, the malware is unable to modify or hide its malicious behavior [16] because it is unexecuted. Nonetheless, static analysis needs distinct features in minimal amounts so as to be able to detect malware.

Discovering the relevant list of features in lesser amounts increases the accuracy of the machine learning prediction model. This is because static analysis decreases the multifaceted nature of the predictive model, hence, reducing the machine learning processing time. It also removes irrelevant data as well as minimizes the dimensionality of the datasets [17, 18]. In relation to this, the current paper adopts the particle swarm optimization (PSO) approach, which is a bio-inspired algorithm, to swarm the overall features so as to gain an optimized list of features that specifically detect unknown root exploit. This study applies three types of features encompassing: (1) system command, (2) directory path, and (3) code-based features. System command is a command in UNIX-based operating system. It is an efficient type of feature because of the rare changes in its characteristics although the

kernel version is updated regularly. Security practitioners possibly use this feature for a long period of time. It consists of terminal commands, android debug bridge (ADB) commands and executing processes. The ADB is a novel feature that was not covered by literature search during writing this paper. The next type of feature is the directory path. It consists of Linux kernel directories and system paths. The last feature is code-based features; they are tools used for executing the commands, such as standard error (stderr), standard input (stdin), and standard output (stdout).

To conduct this study that uses the bio-inspired PSO features through the machine learning classifiers, this research applied four type of boosts (Adaboost, Realadaboost, Logitboost, and Multiboost). These were used to train and convert the Decision Stump classifier into a strong learner that detect root exploit in mobile devices. The contributions of this study are as follows:

- a) The assessment experiment applied 550 benign and 550 root exploit samples which were extracted from the Malgenome dataset. To avoid bias results in detecting the unknown root exploit, this study also evaluates other samples which were taken from Drebin which were excluded from the simulation stage.
- b) The experiment employed the bio-inspired PSO to swarm, optimize and select the best root exploit features for the machine learning classifiers.
- c) The study also used multiple categories of features such as: (1) system command (i.e., terminal and process); (2) directory path; and (3) code-based features. To the best of the author's knowledge, the first category include the novel ADB features, which had not been discovered in previous Android static analysis research.
- d) This study employed multiple types of boosts (Adaboost, Realadaboost, Logitboost, and Multiboost) for comparison purposes so as to discover which among them, is the most suitable boost that also synchronizes well with the PSO selected features.
- e) This study also developed a root exploit detection system (RODS) which was then used to evaluate the detection rate of the root exploit prediction.
- f) The evaluation measured the effectiveness of the results in both simulation and RODS to increase the security of the blockchain-based medical data management (BMDM).

The remainder of this paper is organized as follows. Second section surveys the related works. Third section provides information about the techniques utilized in this analysis. It involves data collection, reverse engineering application, feature extraction and machine learning classification. Fourth section exhibits the assessment and result. Fifth section compares the results with previous findings. Sixth section provides the development of the root exploit detection system.

Seventh section presents the discussion of the paper. Finally, eighth section expounds on the conclusion and gives suggestions for future works.

Related work

This section briefly describes the blockchain, awareness of the root exploit, types of analysis to counter it, previous works involving static analysis and machine learning, the bio-inspired method in optimizing features and boosting method that helps to develop efficient root exploit prediction.

Blockchain based medical data management (BMDM)

Deep investigations focusing on medical applications [19–23] are noted in previous studies. These investigations have made contributions to medical data management by enhancing time proficiency in processing data of the heartbeat. Its effectiveness concerns low resource usage, great time calculations, more vitality, less power and low memory consumption. All these are important because the condition of each patient is dependent on each second of the time. Therefore, security in healthcare devices is vital in protecting patients’ data from being compromised. One possible alternative that able to protect and secure the data management of healthcare applications is the blockchain technology [5].

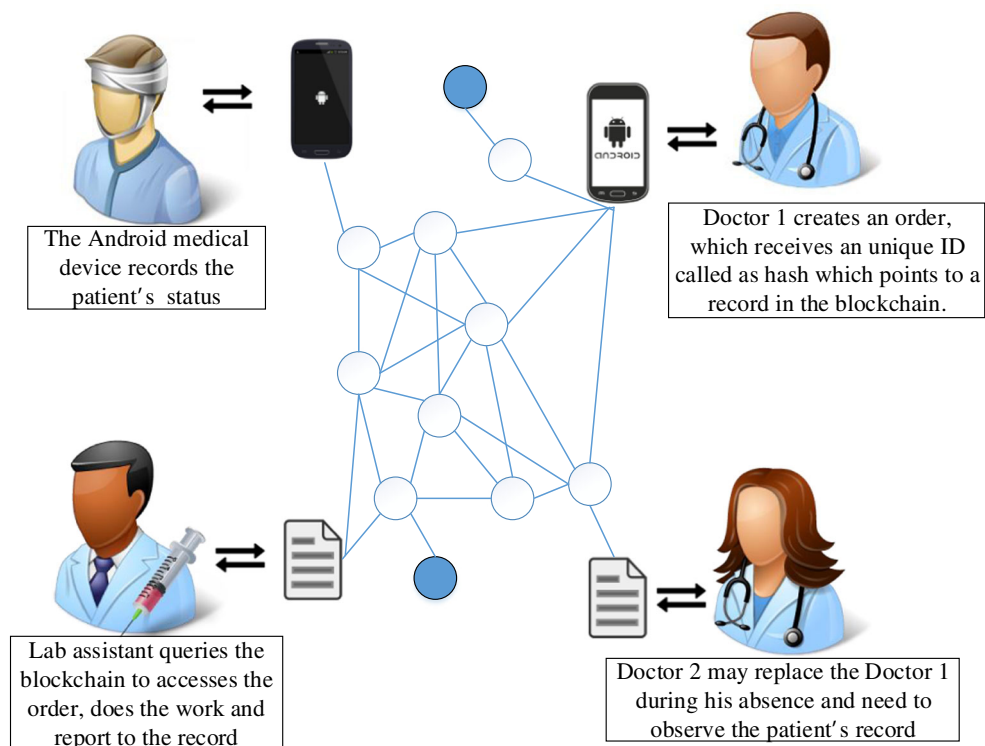
Figure 1 depicts blockchain in the medical environment. Blockchain refers to a block that records the new transaction of the medical data. Once the new block finishes its computerization, it will be linked to the chain, further constructing a chain of multiple blocks. The blockchain then records this transaction in a distributed ledger on a peer to peer network. It excludes the requirement of a middle man or third party thereby, enhancing processing speed whilst also minimizing costs involved. Compared to other approaches, blockchain is more secure because the data are chained in the blockchain, and decentralized. Therefore, attackers are unable to attack the node which they would like to compromise. Even if attackers are able to attack the node, they need to compute the overall involved block of the blockchain at a certain time and this is highly impossible.

Nonetheless, root exploit is capable of bringing risks into the blockchain data management. This is because, the root exploit would take over the whole operating system (OS) that contains the healthcare applications; it steals all the password including the private keys and proceed to signing legal transactions in the blockchain data transaction. The attackers may disguise as an administrator of the OS and start transferring the data with the help of the root exploit.

Root exploit

As mentioned above, unscrupulous authors design malware to compromise the system so as to make private gains. Among

Fig. 1 Blockchain in medical situation



the common types of malware available such as root exploit, botnet, spyware, worm, and Trojan, the most dangerous is root exploit, also known as rootkit [24, 25]. Figure 2 depicts how root exploit affects the victim’s OS. Once the attackers compromise the kernel of the OS, they are able to control all layers of the system including applications and libraries. As a result, the OS may allow the attackers to do whatever they please including installing multiple types of malware and stealing all the passwords that are contained in the blockchain transaction. Figure 3 shows the two situations of before and after the root exploit attacks. Once the attackers had obtained all the required passwords, they will proceed with the transactions in the blockchain and begin to steal all the private medical data and transfer these to the desired address of the blockchain destination. Therefore, security practitioner conduct malware analysis to investigates and determines ways of preventing, detecting, and responding to malware activities [26–28].

Malware analysis

Malware analysis comprises dynamic and static analysis. Many studies such as [29–37] have employed dynamic analysis to execute applications and to monitor the malware runtime behavior, for example, network access, correlating user input, applications and network traffic, user behavior, thread–gain system call sequences (activated by applications), and memory modifications. By monitoring the running applications, security analysts are able to detect any unknown

malware or the type of malware that transforms from benign to malware on-the-fly (during the application runs). Dynamic analysis is convenient but it is limited by its high requirements and low performance. This method is unable to detect the malware types which hide their malicious behaviors during analysis. In contrast, static analysis examines malware samples without executing them [38].

Static analysis [39] is an experiment which focuses on the malware application code, hence it covers all possible activities without any scope of time. This is because static analysis does not execute any application. Its main activity is to reverse engineer the applications, with the aim of retrieving the entire native code and to further inspect the structure and substance within it without executing the applications [40–42]. The benefit of this approach is that since it considers the overall code, it also provides the big picture of the application cycle. In addition, its processing phase is short and fast because static analysis is done without running the application.

Using static analysis, [15] had proposed family signatures as a method to detect unknown malware in the Android platform. Their study concentrated on using code strings to detect new malware variants. The signature code consists of methods, classes, character strings, and method bodies. These signatures were extracted to classify each set of variant of a malware family by estimating its similarity to the signatures. Unlike the targeted malware discussed in their study, our features are specifically adjusted to detect root exploit.

Fig. 2 Root exploit exploitation

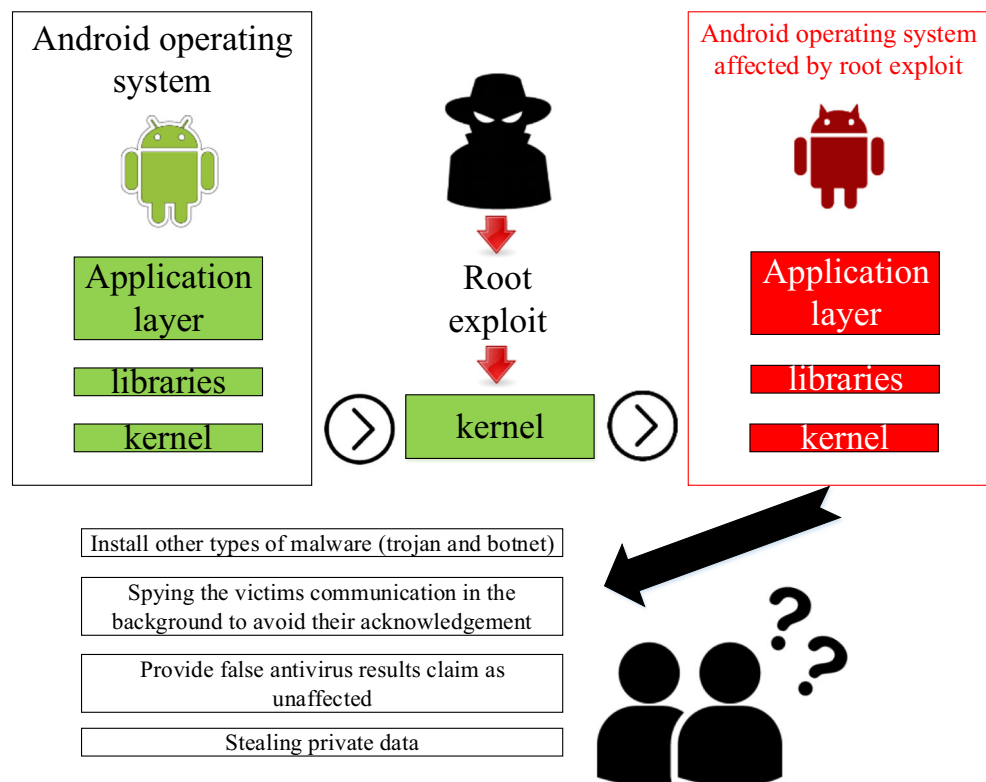
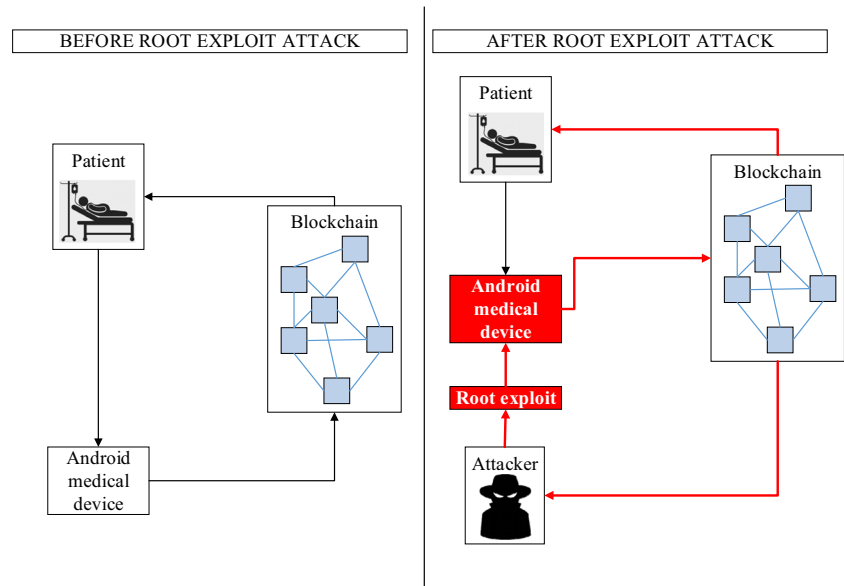


Fig. 3 Before and after root exploit attack

Apk Auditor [43] introduced a way to detect malware by using permission as features. This system utilizes static features to distinguish the benign or malicious applications. The study used permission as features and logistic regression algorithm to detect the malware score threshold. When an application exceeds its threshold limit score, Apk Auditor classifies it as malicious and so categorizes it as malware. Two other studies namely, [44, 45] have also adopted permission as their features and machine learning algorithms in their approach, unlike our current study which excludes permissions but focus mainly on code strings in the Java code as features. This is because root exploit is a malware type that is able to evade permissions when it gains super-user privileges [7–9].

Droid analyzer [46] combines API, rooting and botnet commands as features to detect root exploit and mobile botnet. Their study analyzed risky API and strings by using particular features and keywords to identify malware. Their algorithm calculates the MD5 hash value which were cross-referenced in the database of signatures. The current study, however, adopts machine learning specifically to detect root exploit.

Droidexec [47] proposes a framework with opcode components as features to detect root exploit. The framework uses similarity recognition by adopting structural graph constructor (function–relation graph extraction and opcode component graph constructor). The current study, in comparison, uses strings of features to detect root exploit malware; it also combines system commands, directory path, and code-based features with machine learning to detect root exploit.

Static analysis and machine learning

Most security practitioners adopt static analysis with proactive machine-learning approach to discover unknown malware thereby, overcoming the drawbacks of static analysis [48].

Machine learning is a mathematical mechanism that explores the study of algorithms and predicts decisions based on sample data. It is capable of predicting future judgements in light of the encounters seen through past data sources (learning set) hence, predicting the yields [49]. The learning set depends on a given dataset. Insightful choices are made by the connected calculation of the applied algorithms. One such machine learning types is supervised machine learning, which depends on the data of a training stage to create a function. The training information contains input (features or attributes) and output yields (between malware and benign). This information is then estimated for constructing the model. As indicated by the model, supervised machine learning is capable of characterizing the unknown status of the applications as malware or benign. Security analysts have adopted different types of machine learning classifiers for this purpose, as illustrated in Table 1.

The authors in [38] detected malware by adopting the Bayesian classification. Their study used permissions from Androidmanifest.xml and code-based application as features. Among their code-based features (in the top 25 mixed attributes), only one feature, chmod, is similar to ours. This command is used in a Unix-type OS to change the permission of the file system and objects (files and directories).

In [55], the authors proposed using static analysis to extract features from the .apk, .xml, and .dex file properties (including strings, types, classes, prototypes, methods, fields, static values, inheritance, and opcodes). The authors included string types into their features, similar to our study approach. By applying strings with examples applied in their paper, we unable to compare which features they exactly used for their study. In our study, we list the exact 31 features in the experiment to ease the reader to compare and investigate our features.

Table 1 Examples of machine learning classifiers in previous studies

References	Machine learning classifiers
[38, 50, 51]	Bayes, Support Vector Machine (SVM), Decision Tree, and MP
[52, 53]	Support Vector Machine (SVM)
[54]	K-Nearest Neighbor, Decision Tree, Bayesian Network, and SVM
[42]	K-Nearest Neighbor, ID3, Decision Tree, C4.5, and linear SVM
[44]	Adaboost, NB, C4.5, Decision Tree, and SVM
[55]	Decision Tree, NB, Bayesian Network, Part, boosted Bayesian network, boosted Decision Tree, RF, and Voting Feature Interval (VFI)
[45]	Simple Logistic, NB, Bayesian Network, Sequential Minimal Optimization, Instance-based learning with parameter k , J48, Random Tree, and RF
[56]	NB, Part, Ridor, Decision Tree, and Simple Logistic
[57]	SVM, J48, and Bagging. Prism and Nearest Neighbor
[58]	Adaboost
[59]	RF, Random Tree, NB, Decision Tree, and Simple Logistic
[60–63]	K-means
[64, 65]	Normalized Compression Distance (NCD).

Drebin [53] implemented the static analysis approach together with the support vector machine (SVM). They used the .xml files, permission, application programming interface (API) calls, and network addresses as their features in detecting malware. However, the research excluded strings or keywords as features. Different from Drebin who uses an SVM classifier only, the current study adopts multiple types of boosts for the machine learning classifiers. While Drebin targeted general malware types, we aimed to detect root exploit only.

The authors in [56, 59] also used static analysis to detect malware. They used three types of features (API calls, permissions, and commands) in their experiments. Five features in their command including `chmod`, `/sys/bin/sh`, `chown`, `pm install`, and `createSubprocess` are similar to our study. The authors used a total of 179 features (including API calls, commands, and permissions) in their experiments but only the top 20 features were mentioned in their paper. This restriction created some difficulties in comparing the command of features used for machine learning classifiers.

Another type of machine learning, which is a clustering technique with unlabeled data, is unsupervised machine learning. This technique is used in computer security applications, such as malware detection and forensics [66]. Clustering involves dividing a large dataset into similar and smaller datasets. This method classifies a given object set through a certain number of clusters (assume k clusters) so as to find the k centroids (assigned for each cluster). This algorithm randomly chooses the centroid from the set application; it collects each application which belong to a given dataset, and it then assigns a centroid to the nearest centroid. Sherlockdroid [60] and Droidmat [61–63] applied the K-means clustering algorithm in the static analysis of Android malware detection.

The similarity method in clustering [64] is also capable of detecting malware. The similarity distance, based on real-world compressors, is called normalized compression distance (NCD). This method has been used in [67] to determine the similarities among the malware families. The NCD is a parameter-free data mining method which excludes features from the dataset. Another method that is able to detect clone applications (benign applications convert clone applications to malware applications) is Dnandroid [65]. Clone applications may imprint similar characteristics which can be used to expose malware infections.

The studies mentioned above use static analysis and various features to detect malware. None of these studies except Droidanalyzer [46] and Droidexec [47] had discussed root exploit malware. This shows that the investigation of root exploit in the Android platform is rarely conducted. Both Droidanalyzer and Droidexec had adopted methods that were different from the current study which uses machine learning intelligence as the prediction method. Droidanalyzer had utilized API calls and keywords as sets of features for detecting malware while Droidexec [47] had adopted the graph constructor which uses opcode components, as features. Both had listed the keywords as an example only. The current study also uses similar keywords such as `only mount -o remount` and `chmod` but unlike the two studies, it lists the exact total of 31 lines of features that were used in the experiment.

To the best of our knowledge, at the time of writing this paper, the ADB command is one of the novel types of features which were unexplored in existing studies that had employed static analysis with machine learning classifier. It is crucial to have a list of optimized features to enhance the machine learning classification. Therefore, this study utilized the particle swarm optimization (PSO) to select the best features.

Particle swarm optimization (PSO)

In accordance with selecting the most relevant features to enhance the machine learning detection accuracy [68], this study adopted the particle swarm optimization (PSO), a bio-inspired based to optimize features in detecting root exploit. A study [69] which recommended this method to detect malware managed to unveil outstanding results.

PSO was inspired by the flocking and schooling examples of birds or fish [70]. For instance, the PSO algorithm is based on the birds rushing around the nourishment sources. Over different cycles, it has gathered a variable of values that are close to the member with a value that close to the target @ solution. It envisions a group of bird’s hovering over a zone where they noticed a hidden source of food. The individual that is closest to the food chirps the loudest and the other birds will swing around towards him/her. When another circling bird close to the target chirps, it becomes even louder.

In Fig. 4, the individual bird would endeavor to be nearer to the direction of the flying bird that is close to the food’s coordinate which is called gBest. The data information is represented as a pattern or sequence, so the individual pieces of information are being controlled until the example coordinates. In the bird’s case, the general population most remote from the sustenance would attempt to stay mindful of the others by flying faster than the gBest bird. The value of gBest changes for every particle’s pBest value when it gets closer to the objective than gBest. For every emphasis of the algorithm, the gBest moves increasingly closer and closer to the objective until one of the particles achieves the objective. Once we have achieved the optimized features with the PSO, we utilized it to classify and predict the root exploit with the boosting method.

Boosting

The term, Boosting, alludes to meta-calculations @ a group of algorithms in the machine learning to transform from

powerless to a solid classifier. The benefits of Boosting are its quick execution in classification, makes lesser mistakes in the ensemble method strategy and it is reasonable for any stage regardless of whether the underlying model is effective or ineffective [71]. This paper utilized four types of boost comprising Adaboost, Logitboost, Multiadaboost, and Realadaboost.

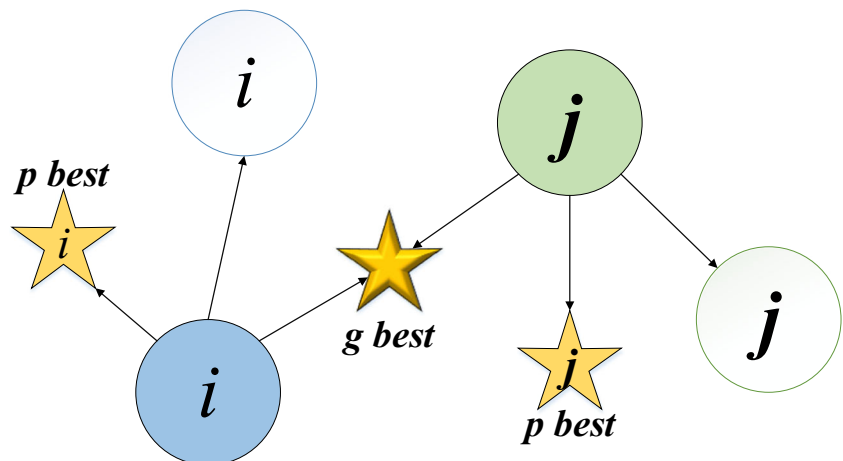
Adaboost

Adaboost is Adaptive Boosting and it was presented by Schapire and Freund [61]. The boosting calculation of the algorithm is created by learning the feeble calculation more than once in a round arrangement. This helps to deliver more precise and exact outcomes. Adaboost allocates each perception, x_i an underlying weight esteem, $w_i = \frac{1}{n}$, where n is the aggregate number of perceptions. For every incorrect perception, w_i is expanded while for every correct prediction, w_i is diminished. Adaboost also prepares another powerless model where perceptions with more prominent weight are given priority. It processes the calculation over and over again so as to consider the heaviness of the frail classifier until the point where the observations are splendidly anticipated.

Realadaboost

The Realadaboost calculation utilizes weighted likelihood evaluations to refresh the added substance strategic model, instead of the groupings. It fits an added substance strategic relapse shown by the stagewise and estimated advancement of $J(F) = E[e^{-yF(x)}]$ [72]. For instance, $F(x)$, is a present gauge; it further locates an enhanced gauge of $F(x) + f(x)$ by limiting $J(F(x) + f(x))$ at every x . It refreshes the populace and consistently applies it to the information by approximating contingent desire through terminal-hub midpoints in trees. It uses rough approximations to restrict desire, for example, decision trees or other compelled models.

Fig. 4 Particle swarm optimization (PSO)



Logitboost

It is an update from Adaboost and it contains multiple class capabilities [72]. The Logitboost algorithms (two classes, population version) are used to fit the additive logistic regression model. It begins with weights $w_i = \frac{1}{N}$ $i = 1, 2, 3, \dots, N$, $F(x) = 0$ and probability estimates $(x_i) = \frac{1}{2}$. The populace calculation portrayed here makes an instant interpretation of a usage on information when $E(.|x)$ is supplanted by a relapse technique, for example, relapse trees. While the part of the weights are manufactured in the populace case, they are not in usage; $w(x)$ is steady when adapted on x , however the $w(x_i)$ in a terminal hub of a tree, for example, rely on the present esteems $F(x_i)$, and will be regularly inconsistent. It utilizes the Newton calculation ventures to fit an added substance symmetric strategic model using the most extreme probability.

Multiboost

Multiboost is an accession to the AdaBoost method. It combines with wagging, which is a part of bagging, to fit the bridling of Adaboost’s high predisposition and differences, which decreases through the lessening of wagging’s prevalent change. It uses C4.5 as the base learning calculation to create choice panels with less errors. It gives more favourable positions than the Adaboost in suiting parallel executions. It is able to

accomplish a large portion of Adaboost’s incredible predisposition decrease when combined with the greater part of the packing’s unrivalled difference reduction. With this mix, the Multiboost is proficient in preparing cases with various weights so as to deliver an exact machine learning expectation [73].

In order to increase machine learning’s prediction in detecting root exploit, these multiple types of boosting have to be combined with the weak rules of algorithm. In the current study, the Decision Stump is boosted from being a weak to a strong classifier. Since the earlier section of this paper has explained static analysis, the PSO and the boosting method, Table 2 tabulates the differences of this study and previous static analysis studies which use machine learning as an approach. Based on the comparison, it seems evident that the current study serves as the only experiment that had scrutinized the novel ADB features by utilizing multiple types of boosts to detect root exploit. The following methodology section describes these novel features and boosts in detail.

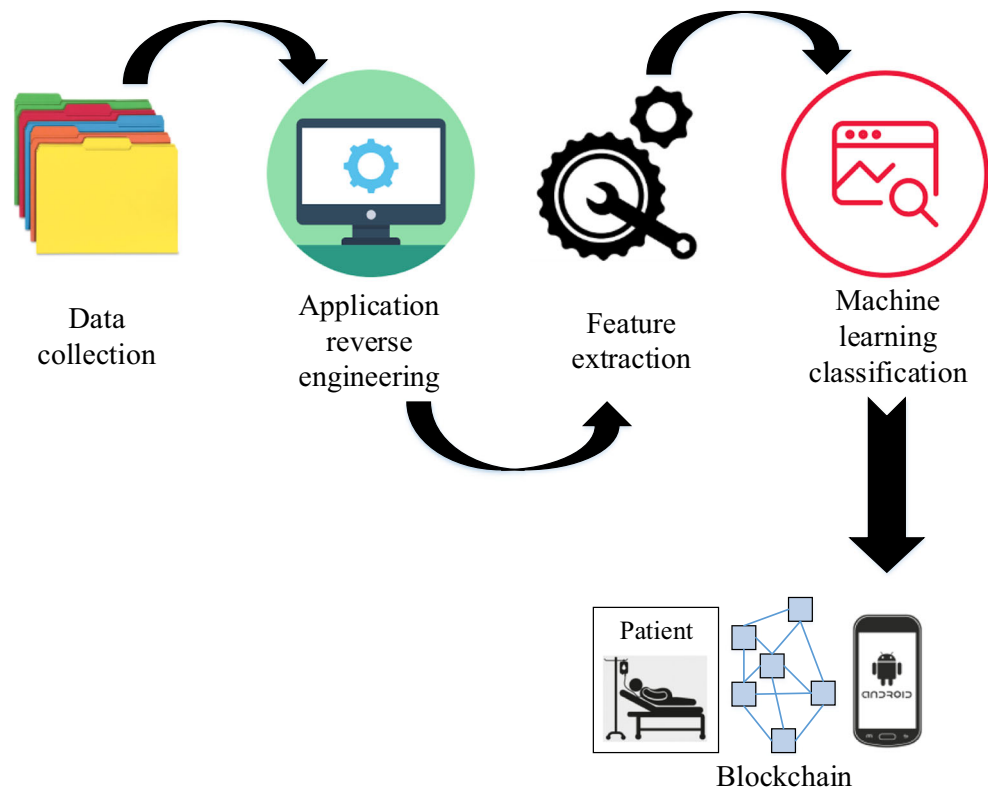
Methodology

The methodology of this study comprises data collection, application of reverse engineering, feature extraction, and machine learning classification. Figure 5 shows how malware and benign applications were collected. This is followed by

Table 2 Previous studies comparison

References	Type of features	Feature selection	Classification method	Targeted malware	Malware dataset
[61]	<i>f</i>	Information gain	m	g	mal
[40]	<i>f</i>	Mutual information	m	g	mal
[55]	<i>f</i>	Vector space	m	g	đ
[76]	<i>f</i>	Range algorithm and information gain	m	g	mal, đ
[77]	<i>f</i>	Accumulative value	c'	b	đ
[71]	<i>f</i>	PSO	a	g	đ
This paper	A	PSO	B	Ř	mal, đ
Legends:					
g = general type of malware	<i>f</i> = other features without ADB	B = multiple types of boosts (Adaboost, Realadaboost, Logitboost and Multiboost)	mal = malgenome	Ř = root exploit	c' = Comparative analysis
m = machine learning without boosts	A = additional features with ADB	đ = drebin	a = adaboost	b = botnet	

Fig. 5 Methodology stages



the reverse engineering application (to retrieve the codes) and then the extracting and identifying of the relevant features. In the final stage, we evaluated these features by using machine learning classifiers. The aim is to take note of the accuracy of the machine learning classification which utilizes the best features in the Android-based healthcare mobile device. The idea is to prevent any root exploit attacks and to protect the blockchain environment from being compromised.

Data collection

A dataset is organized into certain types of data structure so as to support the experiment which strives to identify malware. For this purpose, the experiment needs two classes of applications: malware and benign. *Malware* is an abbreviated term for “malicious software.” Unethical application authors design malware for harmful purposes, such as damaging the operating system of the computer and gaining unauthorized access into users’ private data without user consent. In contrast, benign software refers to applications that legally provide services and activities to fulfill an owner’s requests. When it is desirable to obtain the private data, requests are sent to the user for authentication. Hence, the initial phase of this methodology involves gathering the malware and benign software.

As the experiment requires malware dataset, 1260 samples of Malgenome were extracted and utilized. These samples consist of 49 different malware families [76] and have been used in many studies [12, 14, 16]. They include several

malware types, such as botnet and root exploit. This study specifically focuses on root exploit malware hence; the experiment considers all samples of the corresponding types in an effort to obtain a total of 550 samples. Table 3 [77] lists the samples’ family and descriptions.

The current study also utilizes benign applications collected from Google Play store [78]. The play store provides Android applications of various categories, such as business, books, comics, communication, education, entertainment, family, lifestyle, medical, music, shopping, transport, tools, and social interactions. These applications provide many types of contents for users of the Android-powered phones, tablets, and Android TV devices. Table 4 lists the benign samples based on frequency. To achieve an equal condition and to obtain unbiased content, 25 applications were unloaded into each of the 34 categories.

VirusTotal scan was used to exclude the malware from the benign applications [79]. This subsidiary of Google delivers a free online service, which analyzes suspicious files, URLs, and various malware types, including Android packages. A total of 850 applications were downloaded and 300 applications were discarded based on the following reasons. First, we only considered applications with a VirusTotal scan result of 0, suggesting that the application is malware-free. We also excluded applications with scan results of more than 0 in the benign dataset. Second, certain applications were placed in multiple categories, for instance, some applications in books and references were placed in the comic category.

Table 3 List of Android root exploit malware

Root exploit family	Frequency	Descriptions
Asroot	8	Asroot is similar to the word in Unix terminal, that is, login <i>as root</i> . Asroot is a standalone program that is capable of executing without OS service and installation procedure.
BaseBridge	120	BaseBridge conducts a silent installation of additional applications without user approval.
DroidDream	16	Whenever the user clicks the application icon on the home screen or when an intent ACTION_MAIN is received by the application, DroidDream directly hijacks the entry activity of the host application.
DroidDeluxe	1	Without querying the user to grant the root privileges, DroidDeluxe leverages known root exploits to bypass the built-in security sandbox.
DroidCoupon	1	DroidCoupon obfuscates the file names that are associated with root exploits (e.g., impersonate as picture file with .png file type).
DroidKungfu 1	34	DroidKungfu contains encrypted root exploit scripts and decrypts these scripts during runtime conditions. It remotely downloads and updates a new version via a network.
DroidKungfu 2	30	
DroidKungfu 3	309	
DroidKungfu 4	20	
zHash	11	zHash contains <i>exploid</i> file names, which are exactly the same as the publicly available file names.
Total	550	

Accordingly, we excluded similar applications in any category so as to avoid duplicates. Third, we set our total target

Table 4 List of benign applications

Category	Frequency
Books and Reference	15
Business	20
Comic	21
Communication	23
Education	11
Entertainment	16
Finance	24
Games (Action)	18
Games (Adventure)	12
Games (Arcade)	10
Games (Board)	14
Games (Card)	15
Games (Casino)	18
Games (Casual)	13
Games (Education)	15
Games (Family)	15
Games (Music)	20
Games (Puzzle)	11
Games (Racing)	11
Games (Role Playing Games)	23
Games (Simulation)	12
Games (Sports)	9
Games (Strategy)	16
Games (Word)	15
Health and Fitness	19
Live Wallpaper	17
Media and Video	18
Medical	17
Music and Audio	11
News and Magazine	23
Personalization	16
Photography	11
Productivity	17
Shopping	24
Total	550

frequency of samples as 550 for each of the malware and benign applications. We set this target to observe the result by using a similar number of samples. The combined malware and benign datasets, therefore, amounted to 1100 samples.

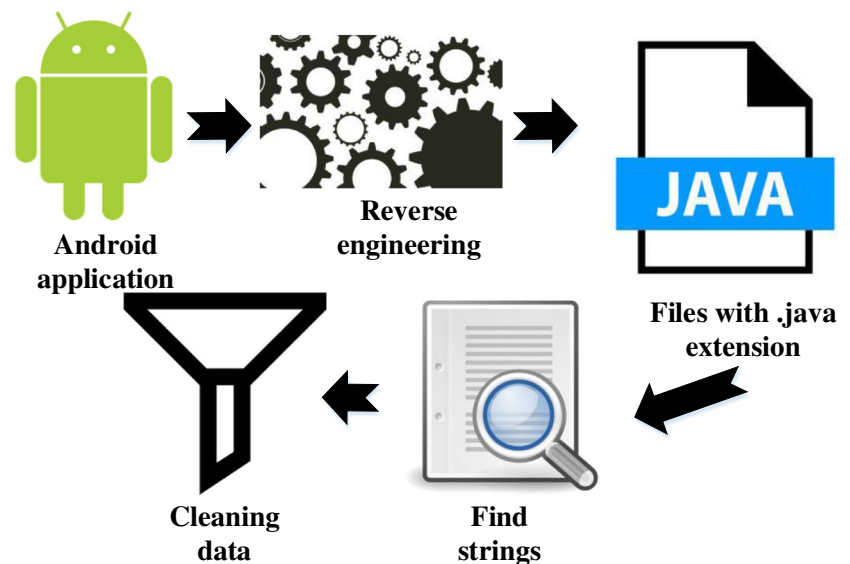
Application of reverse engineering

The general process in static analysis is reverse engineering, which involves reversing the application compilation to attain its programming codes. This was applied for the purpose of analyzing the malware and benign applications. The Android OS depends on the Android application package (APK) as a file system. This operating system uses Java programming language with the .apk file extension. Figure 6 illustrates the reverse engineering procedure. This method reverses the files to the Java programming codes through a tool called Jadx [80]. It then selects the features from the codes. In detail, this engineering tool is able to reverse the compiled .apk to .java extension files (java code).

Given that many lines of code are involved, there is a need to investigate the keywords used in the malware and benign datasets. This is done via the “grep” command and the output is saved in the .csv file. Unix users normally use this command to find and grab any desired keyword according to user demand. This study uses the same command to find malicious strings and keywords for the features. Figure 7 depicts a sample screenshot of the extracted information, showing a malware application and its secure hash algorithm (SHA) name, folder name, and java file. The string contains “/system/bin/chmod”. The figure also shows five malware samples that have one java file contains “/system/bin/chmod” string.

After we grabbed the strings, it is essential to clean the data. In this regard, certain strings were confused with one another,

Fig. 6 Reverse engineering process



for example, the word *cat*, one of the Linux commands, may be confused with other words, such as concatenate, locate, or similar words which contain *cat* strings. Following the cleaning of data, this experiment proceeded to identifying and observing the full exact line of the codes. The *cat* command, which is specifically used for Linux platform, was pulled out. Once the filter process data are completed, the next step is to extract the features.

Feature extraction

The process of feature extraction involves searching for any suspicious strings in all the samples (malware and benign). This process of searching for the feature in the 1100 samples involved a period of 1 month. We managed to discover only 31 features. This was accomplished based on the time constraint of the current study. The 31 features were derived from the system command, directory path, and code-based features. Their proportion amounted to 12, 10 and 9, respectively.

System command

The system command feature consists of a terminal, process, and an ADB command. Some examples are *adb_enabled* and *cat*. As an illustration, *cat* is a maintained command; *cat* is an abbreviation for “concatenate”. This command is most frequently used in the Unix-type OS. It allows users the access

to view files, create single or multiple files, concatenates files, and redirects output via a terminal.

The ADB command is a terminal command line tool that allows communication between the user and the Android emulator to be connected to the Android-powered device [81]. This communication tool allows users to easily connect to their own mobile devices via desktop computers or notebooks. As a result, malware practitioners misuse this tool to gain malicious actions, particularly in gaining root privileges. These system commands are unique elements because they are unchanged and similar to other Linux-based OS commands globally. The architecture of the Android depends on the Linux layer, hence, this feature increases the reliability of future detection methods, for a longer period of time. Figure 8 depicts the system command features’ existence. As an example, *cat* appeared 21 times in the malware samples but only once in the benign samples. The *startservice -n* appeared 359 times in the malware samples, but none in the benign samples.

Directory path

Android has its own OS directory path and its architecture is similar to the Linux kernel. For instance, */system/bin/mount* and */proc.*, are paths that authorize an attempt to enter and gain access to the kernel directories for the purpose of obtaining root privileges without user consent. These sensitive directory paths were included in the current experiment as features.

```
5f7a40015a1f3f42802424ec776799f5e096/com/ UpdateService.java: String str_d = "/system/bin/chmod";
3ac954c94c9c420578777c4aa878b12cd89/com/google1.java: str = "/system/bin/chmod";
4f1cbb091dcde0cd0e8fe0d4bd27134750b/com/Service.java:str5 = "/system/bin/chmod";
5fb8cabd3dd8d4caca3f626f96419ea70e4f/update/Update.java: dir = "/system/bin/chmod";
6a1431f60a704ca729d36c7edf2b8142b03/com/google/UpdateService.java: TCP.execute("/system/bin/chmod");
```

Fig. 7 Example screenshot of *chmod* directory feature

Fig. 8 System command occurrences

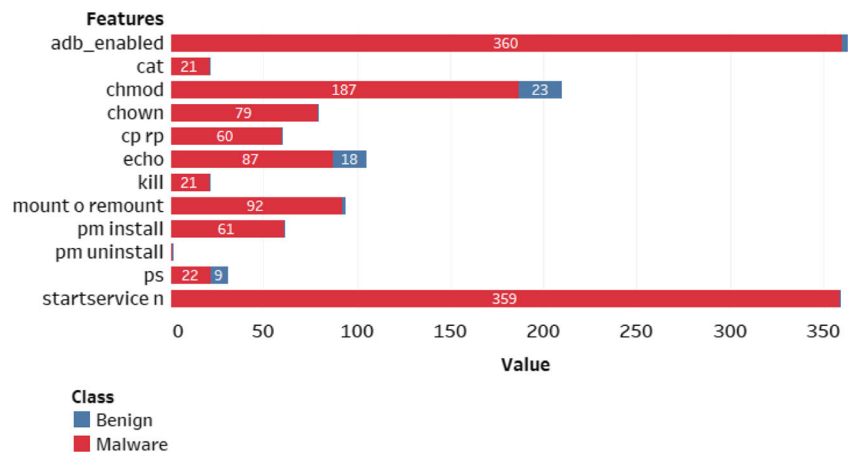


Figure 9 shows one of the directory paths (/system/xbin/su) that appeared 361 times in the malware samples but only 38 times in the benign samples.

Code-based features

The third type of feature is code-based features. Figure 10 illustrates the tool for executing the command, createSubprocess, which appeared 83 times in the malware samples but not at all in the benign samples. One other feature which did not appear in the benign samples is Forked. This feature is the string in an argument or a parameter, which occurred 76 times in the malware samples. Other code-based features that are included in static analysis encompass: setPtyWindowSize (code to execute process), three code execution processes (exec(), exec(“sh”), exec(“su”)), stderr (to detect standard error), stdin (standard input), and stdout (standard output). After discussed these three categories of features, the next section combines all these categories for exploratory analysis.

Exploratory analysis

Exploratory analysis is an approach that analyzes a dataset and summarizes its characteristics with visual methods. It is also known as exploratory data analysis (EDA). This approach discloses an analysis that is beyond formal modeling or any hypothesis-testing task. It was originally proposed by John Tukey [82] to encourage researchers and statisticians to show a graph and to explore a dataset by highlighting the interesting features.

Figure 11 depicts a graph that combines the malware and benign samples, their occurrences, and categories. The figure indicates that from 60 to 500, malware class elevates the area, except for one feature (exec ()). This is because exec () has the highest occurrences in both categories. However, malware class only exists from 300 to 500 which indicates that root exploit utilizes this feature the most.

From another perspective that looks at similar distance, the directory path type showed four features which include two features of system command and two features of code-based fea-

Fig. 9 Directory path occurrences

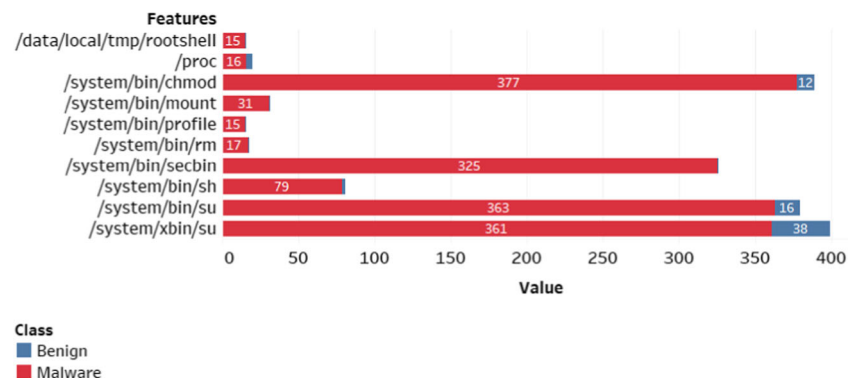
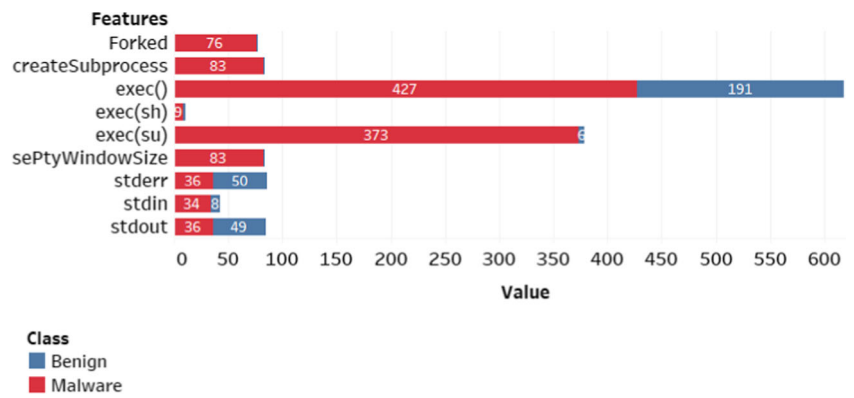


Fig. 10 Code-based occurrences



tures. The graph provided demonstrates that the directory path is more significant than system command and code-based feature. Once all the features are recognized, the next stage involve using the bio-inspired PSO method for feature selection.

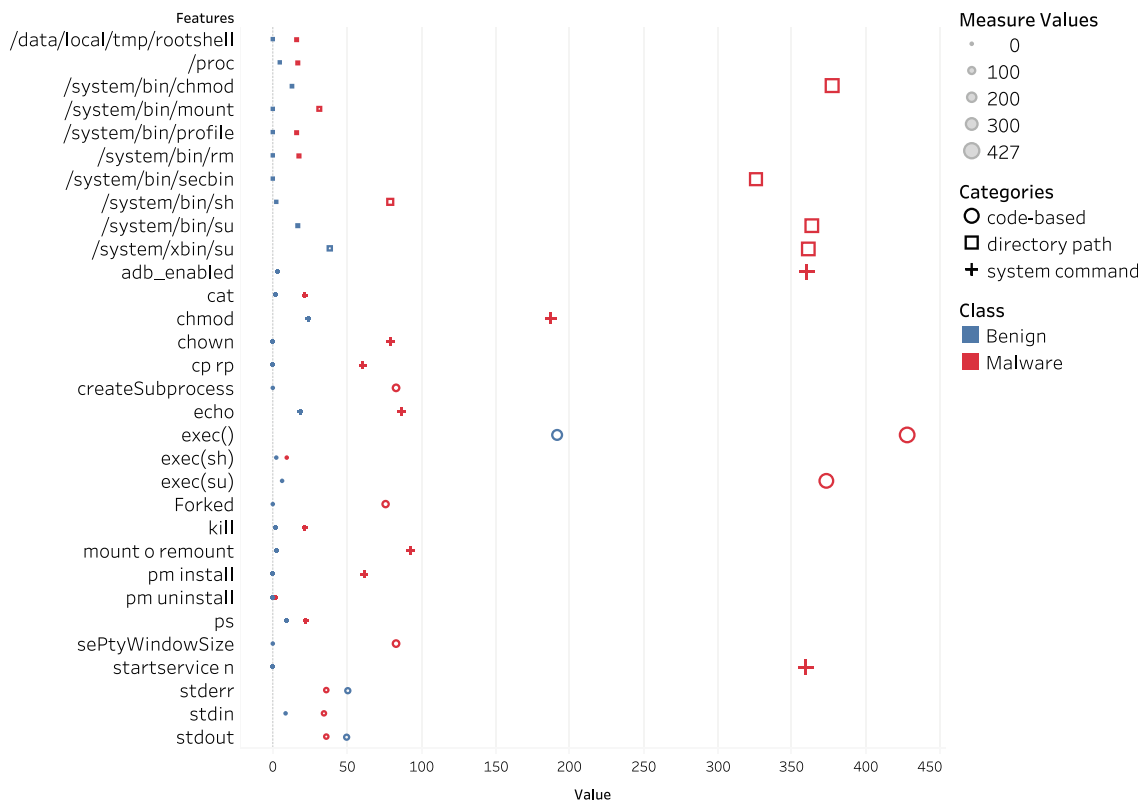
Feature selection

To optimize the efficiency of the root exploit detection, it is important to select unique features because they play a crucial

part in the machine learning prediction. This process also helps to remove noisy and irrelevant data, thereby, increasing the accuracy of the results of the machine learning algorithms [69, 83–85]. This study adopts the bio-inspired PSO to select the unique features of which eight were selected out of 31. Table 5 tabulates these features before and after the PSO decision.

Table 5 shows that the PSO had selected eight relevant features for detecting the unknown root exploit. It is worth

all



Benign and Malware for each Features. Color shows details about Benign and Malware. Size shows Benign and Malware. Shape shows details about Categories.

Fig. 11 All 31 features in categories

noting that at the time of writing, two of the eight features, `adb_enabled` and `startservice -n` are novel elements, undiscovered in previous studies. They are the android debug bridge (ADB) type of features. Once this stage is completed, the step that follows is the machine learning classification phase.

Machine learning classification

In this phase, the steps involved building the machine learning predictive model. In constructing the machine learning model, the classifiers were run in the Waikato Environment for Knowledge Analysis (Weka) [86]. The initial step of building the model was to prepare the Comma Separated Values (.csv) file with the static features (0 and 1). This file contains nine columns and 1101 rows. The nine columns consist of eight features with one class column at the end (M for malware and B for benign). The 1101 rows represent the samples used in this experiment (1100 samples) with an addition of one feature header name, hence, the total number of rows equaled to 1101. Given that our experiment uses static analysis, each sample takes the number of 1 or 0 only, where 1 is if the feature exists @ occur and 0 if the feature is non-existence @ non-occur.

Following the setting of the total number of features, the .csv files were converted to Attribute-Relation File Format (.arff) file using Weka. This is because the ARFF is an ASCII text file format, which was developed specifically for Weka. The .arff file loads faster when compared with the .csv

file loads [87]. As mentioned earlier, boosting (adaboost, logitboost, multiboost and realadaboost) was applied as a method for the machine learning classification. Figure 12 shows the multiple types of boost used to convert the Decision Stump classifier into a strong learner that enhances the results of detecting root exploit.

We conduct this root exploit detection experiment on a desktop computer that was equipped with Intel Core i7-4770 CPU of 3.40 GHz, 16 GB of RAM, and Microsoft Windows 7 Professional as an operating system. The following section presents the evaluation results obtained from the experiment.

Evaluation

Evaluation involves two benchmarks: (1) training and testing; and (2) cross validation. In the first benchmark, the evaluation utilized 70% of the samples for training the machine learning algorithm. To detect unknown root exploit for future needs, it is crucial to use the samples that were excluded from the training. In this regard, we utilized the remainder 30% of the samples for testing the detection.

Subsequent to testing and training is cross validation. In this process, we applied 10-fold cross validation, a technique that randomly selects parts of the data for training and the remainder for testing. These actions (training and testing) were repeated 10 times so as to achieve significant results. In particular, the datasets were randomly split into ten subsets of equal sizes and this was repeated ten times. In each repetition, nine subsets were combined to form the training set for constructing the predictive model, while the remainder of one subset, was used as the test set. This test set was excluded from the training set as they were used to detect unknown root exploit malware in this study.

In order to evaluate the root exploit detection with eight bio-inspired PSO features, we assessed the performance matrix of the machine learning classifiers. Table 6 lists each evaluation in terms of accuracy, True Positive Rate (TPR), recall, precision, f-measure and False Positive Rate (FPR), ROC, MCC, and PRC. It further lists the benchmark performance evaluation and its descriptions. The results in both training and testing, and cross validation benchmark are according to these evaluation measures.

Training and testing

Table 7 projects that Logitboost is the best boost for the evaluation conducted in this study. It lists the best results which are highlighted in bold. Here, Logitboost serves as the best boost for the Decision Stump machine learning classifier; it jotted the best accuracy, f-measure, MCC, ROC, and PRC. (Fig. 13 depicts the multiple results of all the boost used to illustrate a clearer comparison).

Table 5 Before and after PSO features selection

Before PSO		After PSO
.exec()	chown	.exec("su")
.exec("su")	cp -rp	/system/bin/chmod
.exec(sh)	createSubprocess	/system/bin/secbin
/data/local/tmp/rootshell	echo	adb_enabled
/proc	Forked	cat
/system/bin/chmod	kill	chmod
/system/bin/mount	mount -o remount	setPtyWindowSize
/system/bin/profile	pm install	startservice -n
/system/bin/rm	pm uninstall	
/system/bin/secbin	Ps	
/system/bin/sh	setPtyWindowSize	
/system/bin/su	startservice -n	
/system/xbin/su	Stderr	
adb_enabled	Stdin	
cat	Stdout	
chmod		

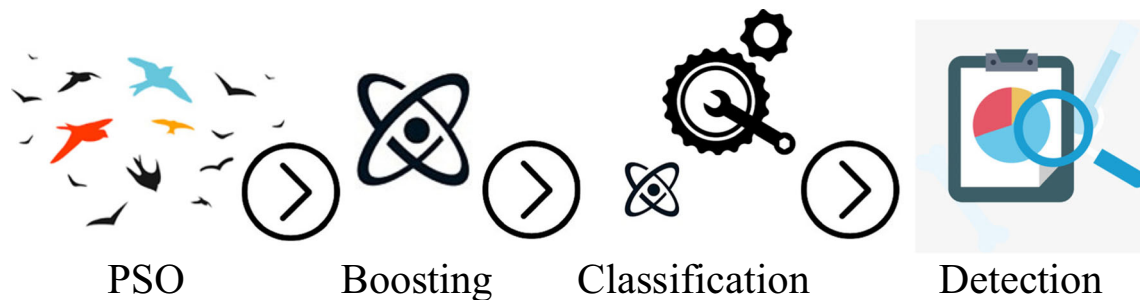


Fig. 12 Machine learning classification

Figure 13 shows the advantage of using each boost for each evaluation. Although Logitboost was the best in several aspects, it also gained the worst FPR mark, among the other boosts (0.006), with the highest record of mistakes in classifying the wrong root exploit. The next best boost in line is Adaboost which jotted good values in all aspects including the FPR. The section below uncovers more discoveries based on the cross validation evaluation benchmark.

Cross validation

The cross validation results are arranged similarly to the previous section, where Table 8 highlights the best value in bold and Fig. 14 depicts the overall results. It was observed that Logitboost performed well in accuracy (90%), f-measure (89.3%), MCC (0.812), ROC (0.905), and PRC (0.937).

In contrast, multiadaboost gained the best value in FPR (0.4), classifying lesser mistakes among other boosts. The range difference in FPR between them is significantly large as noted in adaboost (1.3), logitboost (2.2), and realadaboost

(3.3). Following this, we compare the best results gathered from this study with the results of past studies.

Comparison

The best results gathered from this study were compared with the results of past research using static analysis. The purpose is to investigate the features’ performance in distinguishing malware and benign samples as well as to obtain the capability of the machine learning classifiers. The identified previous studies were used for comparison based on two reasons. One is that they too had adopted Malgenome as their malware dataset (similar to our dataset). Second is that these selected studies had published in reputable journals which is reliable for research comparison. Table 9 presents the comparison in terms of accuracy and TPR results.

In this study, the static analysis offers results that contained the accuracy and TPR of Logitboost (eight features) which was noted to be of the highest value among Bayesian and Naïve Bayes (both ten features). This outcome shows that

Table 6 List of evaluation measures

	Evaluation measure	Descriptions	Equation
Higher value indicates better performance	Accuracy	Correctly predicts instances as either malware or benign	$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
	True Positive Rate (TPR)	Correctly predicts instances as malware	$TPR = TP/(TP + FN)$
	Recall (similar to TPR)	Measures the algorithm performance in identifying malicious samples	$Recall = TP/(TP + FN)$
	Precision	Measures whether the prediction is true or otherwise	$Precision = TP/(TP + FP)$
	F-measure	Measures the weighted harmonic mean of precision and recall	$F-measure = \frac{2 \times precision \times recall}{precision + recall}$
	MCC	The value closer to 1 indicates good classifier performance.	Takes true and false negative and positive into account
	ROC		Tradeoff between the TPR and FPR values
	PRC		Measures the precision and sensitivity value.
Lower value indicates better performance	False Positive Rate (FPR)	Incorrectly predicts the sample as malware, when it is actually benign	$FPR = FP/(FP + TN)$

Table 7 Classifier result in training and testing

	Evaluation measure Percent (%)	Adaboost	Logitboost	Multiadaboost	Realadaboost
Higher value indicates better performance	Accuracy	91	93	82	92
	True Positive Rate (TPR)	81.4	85.7	63.4	87
	Recall	81.4	85.7	63.4	87
	Precision	99.6	99.3	99.6	96.6
	F-measure	89.7	92	77.6	91.5
	0.001 until 1				
	MCC	0.831	0.862	0.685	0.846
	ROC	0.907	0.932	0.842	0.932
	PRC	0.905	0.932	0.843	0.931
Lower value indicates better performance (0.001 until 1)	FPR	0.001	0.006	0.001	0.03

fewer features are capable of increasing the efficiency of machine learning prediction in detecting unknown malware. Moreover, the PSO-inspired method is suitable in selecting the best features in minimal amounts to assist the machine learning prediction.

However, in the TPR comparison, the differences between Logitboost (87%) and others (85% and 85.4%) were small,

about 2% in difference. These results are different from previous studies because of the dataset samples. As mentioned earlier, the current study only utilized a malware set of 550 as compared to other studies in the table, which utilized a malware set of 1000 and 2925. In this regard, future studies may consider finding more root exploit samples to improve the rate of detection. However, the experiments in [38, 59]

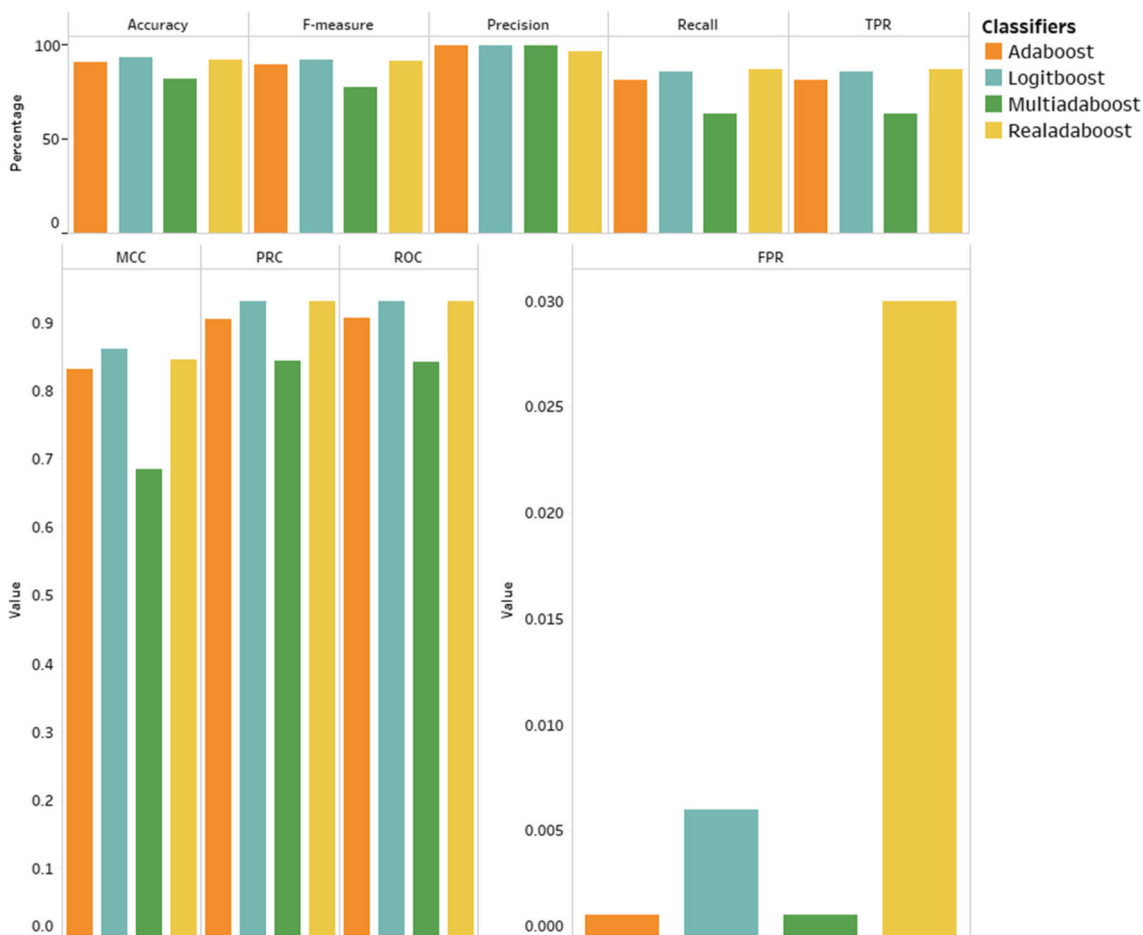


Fig. 13 Visual results of training and testing

Table 8 Classifier results in cross validation

	Evaluation measure Percent (%)	Adaboost	Logitboost	Multiadaboost	Realadaboost
Higher value indicates better performance	Accuracy	90	90	83	90
	TPR	80.7	82.4	66.2	82.7
	Recall	80.7	82.4	66.2	82.7
	Precision	98.4	97.4	99.5	96.2
	F-measure	88.7	89.3	79.5	89
	0.001 until 1				
	MCC	0.808	0.812	0.698	0.802
	ROC	0.903	0.905	0.891	0.903
	PRC	0.933	0.937	0.922	0.936
Lower value indicates better performance	FPR	1.3	2.2	0.4	3.3

focused mostly on general malware types, whereas, in this study, we focused specifically on root exploit.

In a simulated environment, our study was capable of detecting root exploit and this would further protect the mobile device from being utilized in blockchain. The accuracy rate was 93% and the TPR was 87%. In order to test the effectiveness of a practical environment, we developed a system called the Root exploit detection system (RODS).

Root exploit detection system (RODS)

In the previous section, it was mentioned that Logitboost had recorded the best detection rate in identifying zero-day root exploit in the simulated evaluation stage. For this reason, in the interest to test the bio-inspired features, we designed a system called as RODS with Logitboost, as displayed in Fig. 15.

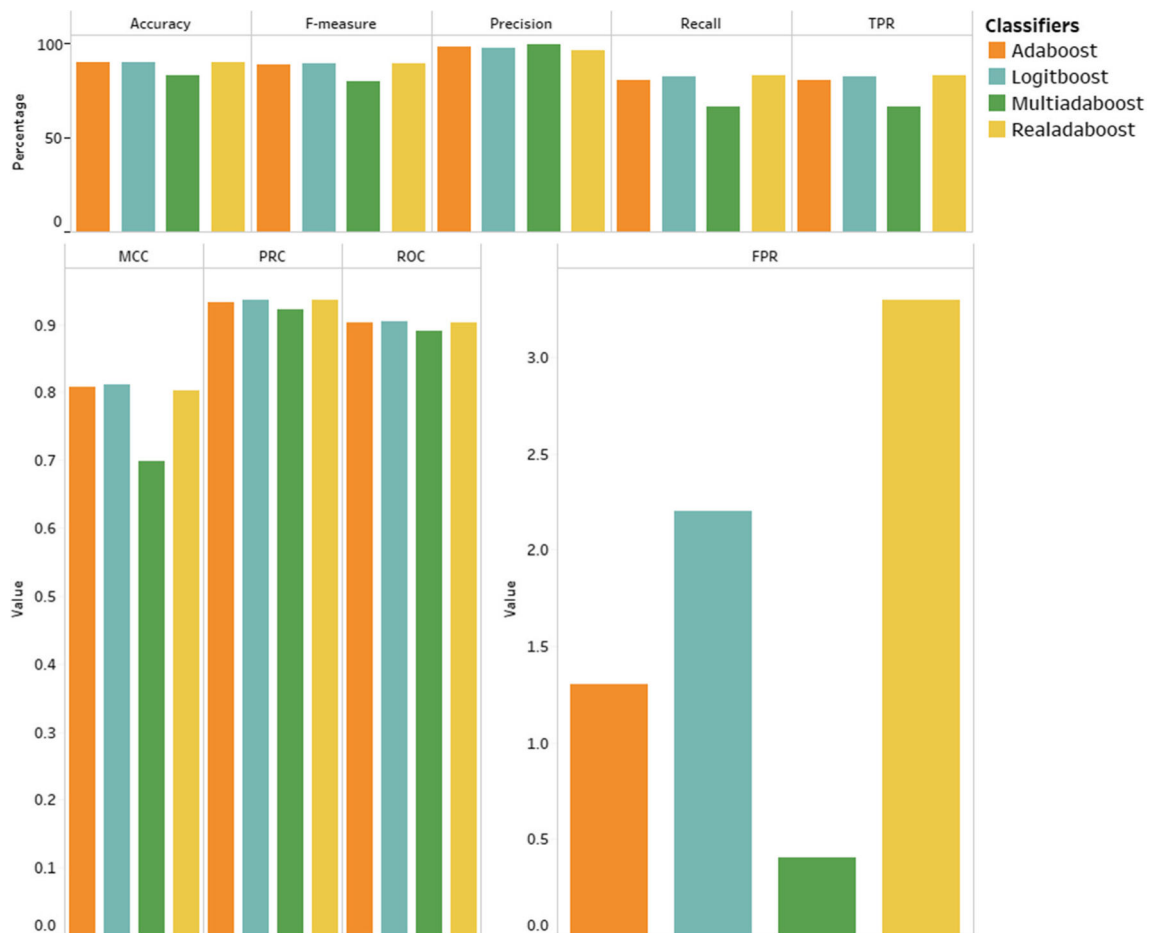


Fig. 14 Visual results for cross validation

Table 9 Result comparison

Type of analysis	References	Classifiers	Feature selection	Result (%)		Dataset		Number of features
				Accuracy	TPR	Benign	Malware	
Static	Ours	Logitboost	PSO	93	87	550	550	Eight
	[38]	Bayesian	Mutual information	89.7	85	1000	1000	Ten
	[59]	Naïve bayes	Information gain	88.4	85.4	3938	2925	

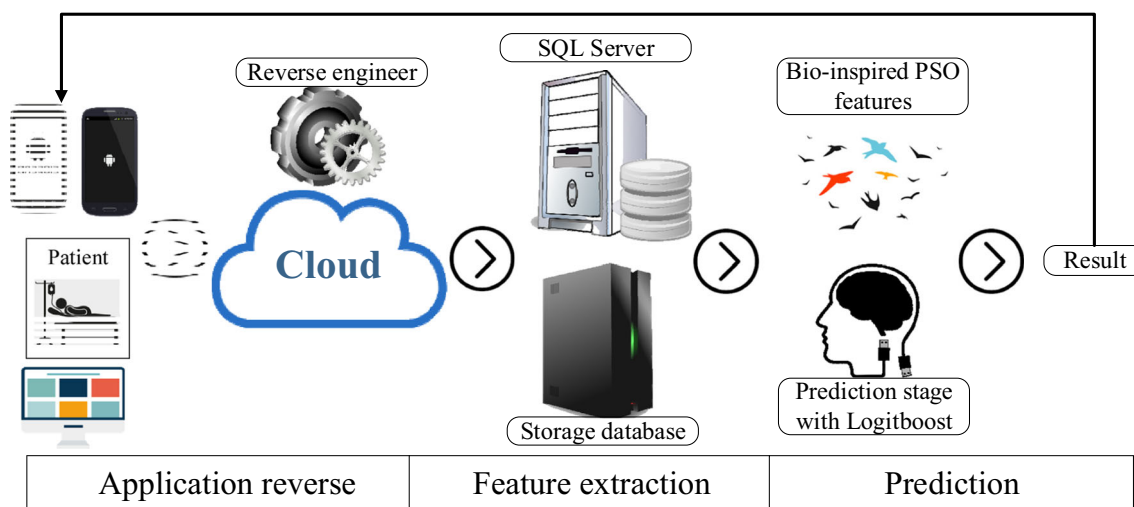
We constructed the framework of the system on a PC that was outfitted with Intel Core i7-4770 CPU of 3.40 GHz, 16 GB of RAM, MySQL database and Microsoft Windows 7 proficient as a working framework with Java as the principle programming language. In mobile device, the user may use Web browser such as Opera or Chrome to use this system. The RODS was developed based on three vital stages – application reverse, feature @ attribute extraction and prediction of the root exploit. At the initial stage, the RODS will analyse the Android application (.apk extension) by assigning a unique id along with the name of the application file. This is to keep the file from being duplicated with the same application in the database. Thereafter, the system inspects the whole code that ends with Java augmentation (.java). This step proceeds with the procedure by searching in all the files that were incorporated in the nested folders in each application. This is to extract our proposed PSO-inspired root exploit features as parameters (exec(“su”), /system/bin/chmod, /system/bin/secbin, adb_enabled, cat, chmod, setPtyWindowSize, and startservice -n). Finally, the system utilizes features for the Logitboost to predict the class of the inspected application as malware or benign. Figure 16 demonstrates the upload zone stated on the main page of the site.

System results

To assess the effectiveness of our prediction root exploit system, it is essential to utilize a distinctive dataset which are different from those of the simulation stage. As this study had utilized Malgenome for the simulation, this section applied a different dataset which is the root exploit in Drebin [88]. It means that these samples were excluded from our machine learning detection model. This is important to detect unknown root exploit. Table 10 lists the root exploit in detail. These samples include the Droidrooter and Rooter families with each family representing three applications. Meanwhile, Fig. 17 displays the prediction results (i.e. M indicates as malware and B stands as benign).

Once the system finishes processing, our system is able to predict all the five samples as root exploit, as shown in Fig. 17. The figure shows the red line which indicates the sha256 of each root exploit. Based on this, it proves that our system is capable of predicting root exploit that had been undiscovered before as the Drebin samples were different from the evaluation part.

Based on previous evaluations, (fourth section) the effectiveness of our proposed bio-inspired PSO features and the boost machine learning classifier used in the

**Fig. 15** Root exploit system architecture

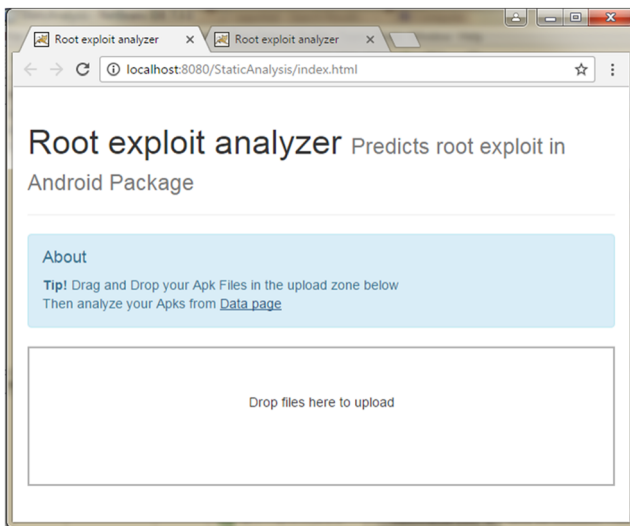


Fig. 16 First page of the system

simulation stage during training and testing, as well as cross validation, is evident. Furthermore, the RODS is also able to detect root exploit in the Drebin dataset on a practical basis.

Therefore, the prediction noted in this section proves that the PSO proposed features with a unique ADB type, and the boost classifier, are capable of predicting unknown root exploit in Android’s medical device. This would protect the device from root exploit attacks, hence, making it safe for users in the blockchain environment.

Discussion

As has been informed, this study investigated the root exploit features by utilizing machine learning classifiers with multiple types of boost to detect unknown root exploit. This study also developed the RODS to evaluate the proposed method practically.

System detection for the blockchain

Figure 18 shows the system being installed in the medical device based on the Android operating system. Clearly, the proposed system would hinder the attackers from deploying root exploit and hacking into the medical device. As a result, the medical field and their professionals may utilize the medical data in the blockchain system with assurance because it is safe. One example drawn from this study showed that a patient who utilizes the Android-based mobile device to monitor his/her heartbeat and send the heartbeat record to the blockchain. By installing the RODS with our proposed bio-inspired features combined with boost machine classifiers, the patient is able to avoid from the root exploit attacks. Conversely, if the mobile device is unable to avoid an attack by root exploit, the attacker may gain control of the Android operating system and install multiple types of malware such as botnet, Trojan, key logger and spyware. Consequently, the attacker would be able to steal all the passwords in the mobile device and start observing the data in the blockchain for personal gains.

However, in the interest to achieve better performance in detecting root exploit, we need to consider certain imperative issues. The first issue is the difference in method: static and dynamic analysis. As this study employed static analysis (inspecting the malware code) as a method, it had also omitted the dynamic behaviour observation (execute the malware and observe its behaviour). In particular, static analysis is unable to detect benign application class that receives updates and further evolves to root exploit form. To avoid this situation, we plan to retrieve the code of the application in two circumstances: (1) once after the application update, and (2) to capture the code at least once a month. The scanning process consumes a minimal amount of time prior to static analysis thereby, accelerating the process rapidly. Root exploit is a type of malware that alters the kernel of the operating system to further gain control of all levels of the system including fake behavior

Table 10 Root exploit information in Drebin

Sha256	Size (kilobyte)	Family
1f5a97fb0cbaa2e10e1f080571ae081d9d85fc95519ef59a85b83ca366b10df2	13	Rooter
226dc739a76faf5127a245b9cc759d4db3086710d4e71594c5578ae642774f5c	950	DroidRooter
94112b350d0feceff0a788fb042706cb623a55b559ab4697cb10ca6200ea7714	862	DroidRooter
94ea44688feb558e2786e52fbfa46d90984e40c0980e28035fd2311d5f17f8e3	13.7	Rooter
add10b0368753ec38de0dca15550d824ac141f0c86f2f123f30551bd82e82415	13	Rooter
edf568790907e970da583855e9b923b2f897fbeb4faf41b87436b23e262b821a	953	DroidRooter

#	Apk Name	Model 1 Result
0	0316ec50-a067-434f-a319-1673a4902832_add10b0368753ec38de0dca15550d824ac141f0c86f2f123f30551bd82e82415	M
1	3a020142-8f47-45fa-aa70-657c4be85774_edf568790907e970da583855e9b923b2f8977beb4faf41b87436b23e262b821a	M
2	4a11033b-400e-41dc-a7e1-4d1b1d723e1b_94ea44688feb558e2786e52bfa46d90984e40c0980e28035fd2311d5f17f8e3	M
3	4a756d01-4705-47d5-9eaf-a9da9150ba64_94112b350d0fecedf0a788fb042706cb23a55b559ab46697cb10ca6200ea7714	M
4	7296eff5e-0ab3-4ea4-bc4c-b37ad4730aae_1f5a97fb0cbaa2e10e1f080571ae081d9d85fc95519ef59a85b83ca366b10df2	M
5	180deea1-e404-418a-8c00-38b67c3fb480_226dc739a76faf5127a245b9cc759d4db3086710d4e71594c5578ae642774f5c	M

Fig. 17 Second page displayed the results

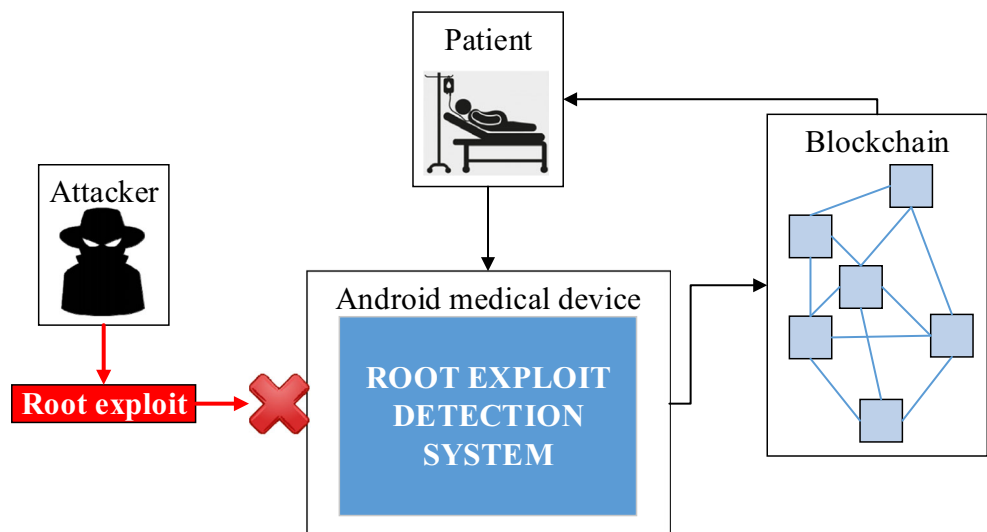
and hiding its presence from dynamic analysis methods. Therefore, static analysis is encouraged to capture the application code before root exploit takes over the OS.

The second issue is that static analysis, on its own, is unable to detect all types of malware thoroughly [89] while dynamic analysis is able to evade obfuscation [90]. Thus, it is advisable to conduct static with dynamic analysis together so as to effectively detect malware. This is because each approach has its own strengths and weaknesses. Combining both types of analysis increases the efficiency of malware countermeasures.

Conclusion and future work

Root exploit is one of the most dangerous malware types; it attacks victim’s mobile device to gain root privileges.

Fig. 18 The proposed system for medical device for blockchain ecosystem



Once it gains root privileges, the attackers are capable of running malicious process stealthily, bypassing permission security, installing any possible types of malware to a victim’s mobile device and then stealing the private keys to compromise the blockchain-based medical data management (BMDM) transactions. As a result of this, it is important to detect unknown root exploit malware in Android-based devices. In this study, we presented a bio-inspired method with machine learning to detect root exploit. We also investigated three types of features, namely system command, directory path, and code-based features, along with the novel android debug bridge (ADB). This study adopted the PSO algorithm to select the best strings and we used four types of boost (adaboost, realadaboost, logitboost, and multiboost) for the machine learning classifiers. The best accuracy of the experimental result of this study exceeded 92% in detecting unknown root exploit in the simulation. This study also developed a system called the RODS which had successfully detected all the samples of root exploit in another dataset, Drebin.

In the data collection phase, this study used a total of 550 malware and 550 benign samples. Thus, future work may include more samples in the dataset to expand the accuracy of static detection. For malware samples, Androzoo [91] may be considered as one of the interests for future research or other samples obtained from any antivirus companies.

The future study also possibly adds more types of features. This may increase the accuracy and robustness of the results [92–94]. For instance, the novel features in ADB commands in this paper are available for addition; and would enhance the future research in detecting root exploit. Finally, implementing the proposed feature set (i.e.,

system command, directory path, and code-based features) may be beneficial in detecting malware variants, such as botnets, worms, and Trojans.

Acknowledgements This work was supported by Universiti Malaysia Pahang, under the Grant Faculty of Computer Systems and Software Engineering (FSK1000), RDU180360.

References

- Zapata, B. C., Fernández-alemán, J. L., Toval, A., and Idri, A., Reusable software usability specifications for mHealth applications. *J. Med. Syst.* 42:1–9, 2018.
- Imtiaz, S. A., Krishnaiah, S., Yadav, S. K., Bharath, B., and Ramani, R. V., Benefits of an android based tablet application in primary screening for eye diseases in a rural population, India. *J. Med. Syst.* 41(4):49, 2017.
- Elhoseny, M., Abdelaziz, A., Salama, A. S., Riad, A. M., Muhammad, K., and Sangaiah, A. K., A hybrid model of internet of things and cloud computing to manage big data in health services applications. *Futur. Gener. Comput. Syst.*, 2018.
- Greene, T., Blockchain can help secure medical devices, improve patient privacy, 2017. [Online]. Available: <https://www.networkworld.com/article/3184614/security/blockchain-can-help-secure-medical-devices-improve-patient-privacy.html>. [Accessed: 06-Feb-2018].
- Puthal, D., Malik, N., Mohanty, S. P., Kougiianos, E., and Yang, C., The blockchain as a decentralized security framework. *IEEE Consumer Electronics Magazine* 7(2):18–21, 2018.
- De, N., Hacks, scams and attacks: Blockchain's 2017 disasters, 2018. [Online]. Available: <https://www.coindesk.com/hacks-scams-attacks-blockchains-biggest-2017-disasters/>. [Accessed: 01-Apr-2018].
- Ma, Y., and Sharbaf, M. S., Investigation of static and dynamic android anti-virus strategies. In: *10th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada, 2013*, 398–403.
- Schmidt, A. et al., Smartphone malware evolution revisited: android next target? In: *IEEE Conference Publications, Montreal, Quebec, Canada, 2009*, 1–7.
- Bickford, J., O'Hare, R., Baliga, A., Ganapathy, V., and Liviu, I., Rootkits on smart phones: attacks, implications and opportunities. In: *HotMobile '10 Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, Annapolis, Maryland, 2010*, 49–54.
- Felt, A. P., Finifter, M., Chin, E., Hanna, S., and Wagner, D., A survey of mobile malware in the wild. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Illinois, USA, 2011*, 3–14.
- Khan, S., Gani, A., Wahab, A. W. A., and Singh, P. K., Feature selection of denial-of-service attacks using entropy and granular computing. *Arab. J. Sci. Eng.*, 2017.
- Tahaei, H., Salleh, R., Razak, M. F. A., Ko, K., and Anuar, N. B., Cost effective network flow measurement for software defined networks: A distributed controller scenario. In: *IEEE Access*, 2018, 1–17.
- Narudin, F. A., Feizollah, A., Anuar, N. B., and Gani, A., Evaluation of machine learning classifiers for mobile malware detection. *Soft. Comput.* 20(1):343–357, 2014.
- Affifi, F., Anuar, N. B., Shamshirband, S., and Choo, K.-K. R., DyHAP: Dynamic hybrid ANFIS-PSO approach for predicting mobile malware. *PLoS One* 11(9):1–21, 2016.
- Lee, J., Lee, S., and Heejo, L., Screening smartphone applications using malware family signatures. *Comput. Secur.* 52:234–249, 2015.
- Apvrille, A., and Strazzere, T., Reducing the window of opportunity for android malware Gotta catch 'em all. *J. Comput. Virol.* 8(1):61–71, 2012.
- Feizollah, A., Anuar, N. B., Salleh, R., and Wahab, A. W. A., A review on feature selection in mobile malware detection. *Digit. Investig.* 13:22–37, 2015.
- Alhendawi, K. M., Predicting the effectiveness of web information systems using neural networks modeling: framework & empirical testing. *International Journal of Software Engineering and Computer Systems (IJSECS)* 4(1):61–74, 2018.
- Pirbhulal, S., Zhang, H., Wu, W., Mukhopadhyay, S. C., and Zhang, Y.-T., Heart-beats based biometric random binary sequences generation to secure wireless body sensor networks. *IEEE Trans. Biomed. Eng.*:1–9, 2018.
- Pirbhulal, S., Zhang, H., Mukhopadhyay, S., Li, C., Wang, Y., Li, G., Wu, W., and Zhang, Y. T., An efficient biometric-based algorithm using heart rate variability for securing body sensor networks. *Sensors* 15(7):15067–15089, 2015.
- Pirbhulal, S., Zhang, H., Wu, W., and Zhang, Y. T., A novel biometric algorithm to body sensor networks. In: *Wearable Electronics Sensors, Smart Sensors, Measurement and Instrumentation*. Vol. 15, 2015, 57–79.
- Pirbhulal, S., Zhang, H., Wu, W., and Zhang, Y.-T., A comparative study of fuzzy vault based security methods for wireless body sensor networks. In: *Proceedings of the International Conference on Sensing Technology (ICST), Nanjing, China, 2016*, 1–6.
- Pirbhulal, S., Zhang, H., and Wu, W., HRV-based biometric privacy-preserving and security mechanism for wireless body sensor networks. In: *Wearable Sensors Applications, Design and Implementation*, 2017, 12-1-27.
- Ullah, F., Edwards, M., Ramdhany, R., Chitchyan, R., Babar, M. A., and Rashid, A., Data exfiltration: A review of external attack vectors and countermeasures. *J. Netw. Comput. Appl.* 101:18–54, 2017 2018.
- Tian, Z., Wang, B., Zhou, Z., and Zhang, H., The research on rootkit for information system classified protection. In: *2011 International Conference on Computer Science and Service System (CSSS)*, 2011, 890–893.
- Anuar, N. B., Papadaki, M., Furnell, S., and Clarke, N., An investigation and survey of response options for intrusion response systems (IRSs). In: *Proceedings of the 9th Annual Information Security South Africa Conference*, 2010, 1–8.
- Razak, M. F. A., Anuar, N. B., Salleh, R., and Firdaus, A., The rise of 'malware': Bibliometric analysis of malware study. *J. Netw. Comput. Appl.* 75:58–76, 2016.
- Zin, S. M., Anuar, N. B., Kiah, M. L. M., and Pathan, A.-S. K., Routing protocol design for secure WSN: Review and open research issues. *J. Netw. Comput. Appl.* 41:517–530, 2014.
- Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., Ma'arof, R. R., and Shamshirband, S., A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malays. J. Comput. Sci.* 26(4):251–265, 2013.
- Yaakob, N., Khalil, I., Kumarage, H., Atiquzzaman, M., and Tari, Z., By-passing infected areas in wireless sensor networks using BPR. *IEEE Trans. Comput.* 64(6):1594–1606, 2015.
- Shabtai, A., Mimran, D., Rokach, L., Shapira, B., and Elovici, Y., Mobile malware detection through analysis of deviations in application network behavior. *Comput. Secur.* 43:1–18, 2014.

32. Lin, Y., Lai, Y., Chen, C., and Tsai, H., Identifying android malicious repackaged applications by thread-grained system call sequences. *Comput. Secur.* 39:340–350, 2013.
33. Feizollah, A., Shamshirband, S., Anuar, N. B., Salleh, R., and Kiah, M. L. M., Anomaly detection using cooperative fuzzy logic controller. In: *16th FIRA RoboWorld Congress (FIRA), Kuala Lumpur, Malaysia*, 2013, 220–231.
34. Xie, L., Zhang, X., Seifert, J.-P., and Zhu, S., pBMDs : A behavior-based malware detection system for cellphone devices. In: *3rd ACM Conference on Wireless Network Security Location: Stevens Institute Technology, Hoboken, NJ*, 2010, 37–48.
35. Burguera, I., Zurutuza, U., and Nadjm-Tehrani, S., Crowdroid: behavior-based malware detection system for android. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Chicago, Illinois, USA*, 2011, 15–26.
36. Feizollah, A., Anuar, N. B., Salleh, R., and Amalina, F., Comparative study of K-means and mini batch K-means clustering algorithms in android malware detection using network traffic analysis. In: *International Symposium on Biometrics and Security Technologies (ISBAST)*, 2014.
37. Allahham, A. A., and Rahman, M. A., A smart monitoring system for campus using Zigbee wireless sensor networks. *International Journal of Software Engineering and Computer Systems (IJSECS)* 4(1):1–14, 2018.
38. Yerima, S. Y., Sezer, S., and McWilliams, G., Analysis of Bayesian classification-based approaches for android malware detection. *IET Inf. Secur.* 8(1):25–36, 2014.
39. Chess, B., and McGraw, G., Static analysis for security. *IEEE Security & Privacy Magazine* 2(6):76–79, 2004.
40. Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., and Lee, W., Eureka: a framework for enabling static malware analysis. In: *Lecture Notes in Computer Science*. Vol. 5283, 2008, 481–500.
41. Chang, T.-K., and Hwang, G.-H., The design and implementation of an application program interface for securing XML documents. *J. Syst. Softw.* 80(8):1362–1374, 2007.
42. Aafer, Y., Du, W., and Yin, H., DroidAPIMiner: mining API-level features for robust malware detection in android. In: *Security and Privacy in Communication Networks*, 2013, 86–103.
43. Talha, K. A., Alper, D. I., and Aydin, C., APK auditor: Permission-based android malware detection system. *Digit. Investig.* 13:1–14, 2015.
44. Huang, C.-Y., Tsai, Y.-T., and Hsu, C.-H., Performance evaluation on permission-based detection for android malware. In: *Proceedings of the International Computer Symposium ICS 2012 Held at Hualien, Taiwan*. Vol. 21, 2012, 111–120.
45. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., and Alvarez, G., PUMA: permission usage to detect malware in android. In: *Advances in Intelligent Systems and Computing*, 2013, 289–298.
46. Seo, S.-H., Gupta, A., Mohamed Sallam, A., Bertino, E., and Yim, K., Detecting mobile malware threats to homeland security through static analysis. *J. Netw. Comput. Appl.* 38:43–53, 2014.
47. Wei, T., Lee, H., Tyan, H.-R., Liao, H. M., Jeng, A. B., and Wang, J., DroidExec: root exploit malware recognition against wide variability via folding redundant. In: *17th International Conference Advanced Communication Technology (ICACT), PyeongChang, Korea*, 2015, 161–169.
48. Anuar, N. B., Sallehudin, H., Gani, A., and Zakari, O., Identifying false alarm for network intrusion detection system using hybrid data mining and decision tree. *Malays. J. Comput. Sci.* 21(2):101–115, 2008.
49. Kotsiantis, S. B., Supervised machine learning: A review of classification techniques. *Informatica* 31:249–268, 2007.
50. Yerima, S. Y., Sezer, S., McWilliams, G., and Muttik, I., A new android malware detection approach using Bayesian classification. In: *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain*, 2013, 121–128.
51. Peng, H. et al., Using probabilistic generative models for ranking risks of android apps. In: *ACM Conference on Computer and Communications Security, (CCS), Raleigh, North Carolina, USA*, 2012, 241–252.
52. Sarma, B., Li, N., Gates, C., Potharaju, R., Nita-rotaru, C., and Molloy, I., Android permissions: a perspective combining risks and benefits. In: *SACMAT '12 Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, New Jersey, USA*, 2012, 13–22.
53. Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., and Rieck, K., DREBIN: effective and explainable detection of android malware in your pocket. In: *21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2014, 1–15.
54. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., and Álvarez Marañón, G., Mama: Manifest analysis for malware detection in android. *Cybern. Syst.* 44(6–7):469–488, 2013.
55. Shabtai, A., Fledel, Y., and Elovici, Y., Automated static code analysis for classifying android applications using machine learning. In: *Ninth International Conference on Computational Intelligence and Security, Nanning, Guangxi Zhuang Autonomous Region China*, 2010, 329–333.
56. Yerima, S. Y., Sezer, S., and Muttik, I., Android malware detection using parallel machine learning classifiers. In: *Eight International Conference on Next Generation Mobile Apps, Services and Technologies, (NGMAST), St. Anthony's College of the University of Oxford, UK*, 2014, 37–42.
57. Peiravian, N., and Zhu, X., Machine learning for android malware detection using permission and API calls. In: *International Conference on Tools with Artificial Intelligence (ICTAI), Herndon, VA, USA*, 2013, 300–305.
58. Sheen, S., Anitha, R., and Natarajan, V., Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing* 151:905–912, 2015.
59. Yerima, S. Y., Sezer, S., and Muttik, I., High accuracy android malware detection using ensemble learning. *IET Inf. Secur.* 9(6): 313–320, 2015.
60. Aprville, L., and Aprville, A., Pre-filtering mobile malware with heuristic techniques. In: *The 2nd International Symposium on Research in Grey-Hat Hacking (GreHack), Grenoble, France*, 2013, 1–16.
61. Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., and Wu, K.-P., DroidMat: android malware detection through manifest and API calls tracing. In: *Seventh Asia Joint Conference on Information Security, Tokyo, Japan*, 2012, 62–69.
62. Samra, A. A. A., Kangbin, Y., and Ghanem, O. A., Analysis of clustering technique in android malware detection. In: *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Taichung, Taiwan*, 2013, 729–733.
63. Aung, Z., and Zaw, W., Permission-based android malware detection. *International Journal of Scientific & Technology Research* 2(3):228–234, 2013.
64. Cilibrasi, R., and Vitányi, P. M. B., Clustering by compression. *IEEE Trans. Inf. Theory* 51(4):1523–1545, 2005.
65. Crussell, J., Gibler, C., and Chen, H., Attack of the clones: detecting cloned applications on android markets. In: *Computer Security –*

- ESORICS 2012. Lecture Notes in Computer Science*. Vol. 7459, 2012, 37–54.
66. Beverly, R., Garfinkel, S., and Cardwell, G., Forensic carving of network packets and associated data structures. *Digit. Investig.* 8: S78–S89, 2011.
 67. Paturi, A., Cherukuri, M., Donahue, J., and Mukkamala, S., Mobile malware visual analytics and similarities of attack toolkits. In: *Collaboration Technologies and Systems (CTS), San Diego, CA, USA, 2013*, 149–154.
 68. Spolaôr, N., Cherman, E. A., Monard, M. C., and Lee, H. D., A comparison of multi-label feature selection methods using the problem transformation approach. *Electron. Notes Theor. Comput. Sci.* 292:135–151, 2013.
 69. Razak, M. F. A., Anuar, N. B., Othman, F., Firdaus, A., Affi, F., and Salleh, R., Bio-inspired for features optimization and malware detection. *Arab. J. Sci. Eng.*, 2017.
 70. Kennedy, J., and Eberhart, R., Particle swarm optimization. In: *IEEE International Conference on Neural Network, Perth, WA, Australia*. Vol. 4, 1995, 1942–1948.
 71. Ng, W. W. Y., Zhou, X., Tian, X., Wang, X., and Yeung, D. S., Bagging-boosting-based semi-supervised multi-hashing with query-adaptive re-ranking. *Neurocomputing* 275:916–923, 2017.
 72. Friedman, J., Hastie, T., and Tibshirani, R., Additive logistic regression. *Ann. Stat.* 28(2):337–374, 2000.
 73. Webb, G. I., MultiBoosting: A technique for combining boosting and wagging. *Mach. Learn.* 40(2):159–196, 2000.
 74. Firdaus, A., Anuar, N. B., Razak, M. F. A., and Sangaiah, A. K., Bio-inspired computational paradigm for feature investigation and malware detection: Interactive analytics. *Multimedia Tools and Applications* 76(280):1–37, 2017.
 75. Karim, A., Salleh, R., Khan, M. K., Siddiqa, A., and Choo, K.-K. R., On the analysis and detection of mobile botnet. *Journal of Universal Computer Science* 22(4):567–588, 2016.
 76. Zhou, Y., and Jiang, X., Android malware genome project, 2012. [Online]. Available: <http://www.malgenomeproject.org/>.
 77. Zhou, Y., and Jiang, X., Dissecting android malware: characterization and evolution. In: *IEEE Symposium on Security and Privacy, San Francisco, CA 2012*, no. 4, 95–109.
 78. Google, Google play store, 2014. [Online]. Available: <https://play.google.com/store?hl=en>. [Accessed: 01-Jan-2014].
 79. VirusTotal, VirusTotal, 2016. [Online]. Available: <https://www.virustotal.com/>. [Accessed: 24-Aug-2016].
 80. Skylot, Jadx, 2015. [Online]. Available: <https://github.com/skylot/jadx>. [Accessed: 01-Feb-2014].
 81. Android Developer, Android debug bridge (ADB), 2017. [Online]. Available: <http://developer.android.com/tools/help/adb.html>. [Accessed: 01-Jan-2017].
 82. Tukey, J. W., *Exploratory data analysis: past, present, and future*, 1993.
 83. Jensen, R., and Shen, Q., Computational intelligence and feature selection: rough and fuzzy approaches. Wiley-IEEE Press, 2008.
 84. Adewole, K. S., Anuar, N. B., Kamsin, A., Varathan, K. D., and Razak, S. A., Malicious accounts: Dark of the social networks. *J. Netw. Comput. Appl.* 79:41–67, 2017.
 85. Firdaus, A., Anuar, N. B., Karim, A., and Razak, M. F. A., Discovering optimal features using static analysis and genetic search based method for android malware detection. *Front. Inf. Technol. Electron. Eng.* 9:184:1–27, 2017.
 86. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., The WEKA data mining software: An update. *ACM SIGKDD Explorations* 11(1):10–18, 2009.
 87. Williams, G., ARFF Data, 2010. [Online]. Available: http://datamining.togaware.com/survivor/ARFF_Data0.html. [Accessed: 10-Sep-2015].
 88. Technische Universität Braunschweig, The Drebin dataset, 2014. [Online]. Available: <https://www.sec.cs.tu-bs.de/~danarp/drebin/>. [Accessed: 01-Jan-2015].
 89. Moser, A., Kruegel, C., and Kirda, E., Limits of static analysis for malware detection. In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 2007, 421–430.
 90. Louk, M., Lim, H., Lee, H., and Atiquzzaman, M., An effective framework of behavior detection- advanced static analysis for malware detection. In: *International Symposium on Communications and Information Technologies (ISCIT)*, 2014, 361–365.
 91. Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y., AndroZoo: collecting millions of android apps for the research community. In: *MSR '16 Proceedings of the 13th International Conference on Mining Software Repositories, Austin, Texas, 2016*, 468–471.
 92. Amin, M. R., Zaman, M., Hossain, M. S., and Atiquzzaman, M., Behavioral malware detection approaches for android. In: *IEEE International Conference on Communications, ICC 2016*, 2016.
 93. Enck, W., Defending users against smartphone apps: techniques and future directions. In: *Proceedings of the 7th International Conference on Information Systems Security, Kolkata, India, 2011*, 49–70.
 94. Zhongyang, Y., Xin, Z., Mao, B., and Xie, L., DroidAlarm: an all-sided static analysis tool for android privilege-escalation malware. In: *Proceedings of Computer and Communications Security (CCS), Hangzhou, China, 2013*, 353–358.