IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 20, 2019, accepted May 9, 2019, date of current version May 30, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2916886

A Multimodal Malware Detection Technique for Android IoT Devices Using Various Features

RAJESH KUMAR[1], XIAOSONG ZHANG[1], WENYONG WANG[1], RIAZ ULLAH KHAN[1], JAY KUMAR[1], AND ABUBAKAR SHARIF[2,3]

[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2]School of Electronic Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[3]Department of Electrical Engineering, Government College University Faisalabad, Faisalabad 38000, Pakistan

Corresponding author: Xiaosong Zhang (johnsonzxs@uestc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572115, and in part by the Sichuan Department of Science and Technology under Grant 2018GZ0218, and in part by the the National Key R&D Plan under Grant CNS 2016QY06X1205.

ABSTRACT Internet of things (IoT) is revolutionizing this world with its evolving applications in various aspects of life such as sensing, healthcare, remote monitoring, and so on. Android devices and applications are working hand to hand to realize dreams of the IoT. Recently, there is a rapid increase in threats and malware attacks on Android-based devices. Moreover, due to extensive exploitation of the Android platform in the IoT devices creates a task challenging of securing such kind of malware activities. This paper presents a novel framework that combines the advantages of both machine learning techniques and blockchain technology to improve the malware detection for Android IoT devices. The proposed technique is implemented using a sequential approach, which includes clustering, classification, and blockchain. Machine learning automatically extracts the malware information using clustering and classification technique and store the information into the blockchain. Thereby, all malware information stored in the blockchain history can be communicated through the network, and therefore any latest malware can be detected effectively. The implementation of the clustering technique includes calculation of weights for each feature set, the development of parametric study for optimization and simultaneously iterative reduction of unnecessary features having small weights. The classification algorithm is implemented to extract the various features of Android malware using naive Bayes classifier. Moreover, the naive Bayes classifier is based on decision trees for extracting more important features to provide classification and regression for achieving high accuracy and robustness. Finally, our proposed framework uses the permissioned blockchain to store authentic information of extracted features in a distributed malware database blocks to increase the run-time detection of malware with more speed and accuracy, and further to announce malware information for all users.

INDEX TERMS Android malware detection, blockchain, Internet of Things (IoT), clustering, secure machine learning.

I. INTRODUCTION

Recently, the scope of the internet of things (IoT) is expanding with the introduction of a variety of applications such as healthcare, smart agriculture, smart home, smart shopping, etc. as shown in Figure 1. Most of the devices in a shared IoT network are based on the Android platform due to flexibility, robustness and hardware support, which is pivotal for sensors interface. The different type of IoT devices provides several distinguished services related to sensing, monitoring and controlling tasks (as illustrated in Figure 1). Consequently, in our inexorably associated society, the number and scope of Android devices keep on increasing. It is assessed that there will be roughly 6.1 billion smartphone clients by 2020 [1], [2]. Therefore, malicious activity can affect the working of many devices connected in a network. The recent studies focus on malware detection for Android-based devices.

Furthermore, the Android devices are attractive target for hackers due to extensive use of the Android platform in IoT devices [3]. The hacker exploits android application features to break the security and privacy of devices, which

The associate editor coordinating the review of this manuscript and approving it for publication was Gang Mei.

poses a serious threat towards leakage of personal data such as the location of the user, contact information , accounts, photos,and etc. In addition to this, most of the Android devices do not use anti-virus or malware detection applications [4]–[6]. The previous literature reports several techniques to address the problem of malware attacks, such as a sandbox, access control, signature mechanism, authority mechanism [7]–[9]. Furthermore, in [3], an M0Droi framework based on API calls has been proposed to generate signatures which are pushed toward the client devices for danger identification. However, TaintDroid [10] developed an exception-based malware discovery system based on usage of application data behavior. Moreover, Canfora *et al.* [11], designed a structure to monitor Android Dalvik activity codes regarding frequencies, which can detect malware applications. Besides, Burguera *et al.* [12], proposed Crowdroid framework for the client and server components. The client uses the strace mechanism of the Linux system for monitoring android system calls. Kim *et al.* [13], proposed CopperDroid framework, which detects the behavior of Java code and local code execution. Moreover, most of the previous studies proposed for Android malware detection were based on the limited types of feature selection to detect malware [5], [6], [14]–[18]. Additionally, most of existing malware detection methods cannot be directly employed for IoT devices due to their computation complexity and limited resources such as cost, memory and limited processing capabilities [5], [6], [14]–[18]. Several techniques are developed for selecting the features of malware [14], [19]–[21] for instance information gain technique [19], but they are based on limited types of feature to detect the malware.

To solve the problem, blockchain and machine learning are posing a momentum around the world towards their use for intelligent model training and secure information exchange. Blockchain technique ensures secure and reliable IoT data sharing; the reason for adaption includes high credibility and high efficiency.The internal architecture of this new type of technology is based on a distributed computing paradigm. The proposed technique combines blockchain and machine learning for effective malware detection. Machine learning automatically extracts the malware information using clustering and classification techniques, and further store the information into the blocks. In this structure, all malware information stored in the blockchain, each type of latest malware can inspect by machine learning methods using our enhanced clustering and Multi-class Naive Bayes classifier. A blockchain consensus rules for validating the malware using the p2p network, as the machine learning is used to train the model, and automatically share the malware information to all users in the network. Thereby, we enhanced malware detection accuracy by combining machine learning and blockchain. Machine learning was used to detect and train the model for Android malware detection, creating a vast amount of malware database. The blockchain-based framework was used to store the trained model results in the blockchain.Thus, our proposed framework can identify the new type of malware for IoT devices.

In addition, our proposed framework detects malware using different features information to reflect various characteristics of applications in numerous aspects. Our proposed methodology first distinguishing the malware and benign, and refines them using enhanced clustering methods. Our clustering method calculates the weights of each feature set and iterative reduced the unnecessary features that can effectively distinguish malware and benign applications even though malware have many properties similar to benign applications. Moreover, our proposed clustering method handles multidimensional data by reducing the features using weight learning procedure. This property enables our method to fit for applying on many types of clustering problems. For example, it can be applied on image recognition, cell formation, topic identification, text summarization and many more [22]–[27]. In the second phase, we propose a multi-feature Naive Bayes algorithm for classification of malware. The input matrix for decision tree based Naive Bayes is the output matrix of the clustering process which includes feature-weighting matrix and clustering labels. Among many useful classification algorithms, we analyzed that the Naive Bayes is the most suitable classification approach for various types of features. Additionally, it extracts the most vital characteristics of input matrix for removing noisy objects from the data. Finally, our proposed framework integrates permissioned blockchain database for storing de-tracked information of malware features, which are automatically generating new blocks that can identify the new type of malware for IoT devices. The permissioned blockchain provides actual information in a distributed malware database to increase the run-time detection of malware more efficiently. The advantage of this method is can be applied directly to the mobile device. Our contribution includes major folds as follows:

1) The proposed technique consist of both blockchain technology and machine learning techniques for malware detection of IoT devices. Machine learning automatically extracts the malware information using clustering and classification technique, and store the information into the blockchain.
2) We enhance the clustering algorithm for feature selection by: 1) calculating the weights for each feature set 2) developing a parametric study for optimization, and 3) iterative reduction of unnecessary features having small weights.
3) We propose the multi-feature Naive Bayes algorithm for classification based on decision trees for extracting more important features to provide classification and regression for achieving high accuracy and robustness.
4) Furthermore, our framework also proposes the use of permissioned blockchain, which provides actual information in a distributed malware database to increase the run-time detection of malwares with more speed and accuracy by storing the extracted features in blocks to announce malware features information for all users.

FIGURE 1. Integration of Android platform and apps with IoT devices to realize IoT dreams.

After the brief introduction of the paper, the following sections are arranged as follows. Section II contains the previous reported research work on Android malware detection. Section III gives a brief overview to the proposed methodology based on clustering, classification, and blockchain. Conducted experiments and results are discussed in Section IV. Finally, The section V concludes the contribution of paper and results.

II. LITERATURE REVIEW

In this section, we briefly review the related literature work. Firstly, we discuss the static analysis, which consists of two methods i) Permission-based analysis ii) API Call based analysis. Secondly, we elaborate the dynamic analysis that is used to extract the training characteristics of the model. Also, we consider the hybrid analysis that combines the static and dynamic analysis. Finally, we compare the static, dynamic and hybrid analysis.

A. STATIC ANALYSIS

The recent use of static features of machine learning to detect Android malware include the following: Cen *et al.* [28] proposed a model based on API call and permissions. This method also utilize regularized logistic regression (RLR). Moreover, RLR was compared to different machine learning techniques such as SVM, KNN, decision tree and naive Bayes. DroidMat [29] classified the malware and benign according to the intents, permissions and API calls. For extracting static features, the author used k-nearest neighbors (k-NN) and k-means clustering algorithm. In [30], SVM classifier was developed for on-device malware detection. The proposed algorithm based on API calls, permissions and network access. Yerima *et al.* [9], [31], used random forest ensemble learning models to detect the malware, which was based on permissions, API calls, embedded commands and intents. Wang *et al.* [32] applied SVM, decision trees and random forest to analyses the use of vulnerable permissions for malware detection. Varsha *et al.* [33] extract static features from the manifest and application executable apk files. DAPASA [34] utilized sensitive sub-graphs to construct five features depicting the performance random forest algorithm achieved the best detection.

Wang *et al.* [32], used logistic regression, linear support vector machines, random forest and decision trees. Furthermore, the author achieved the TPR (true positive rate) of 96% and FPR (false positive rate) of 0.06% with the logistic regression, which was highest as compared with other used classifiers. The dataset used in [32], was composed of 18 363 malware applications and 217 619 benign applications, to describe the particular static signature and stage particular static features. Most of the literature explore machine learning approaches for malware

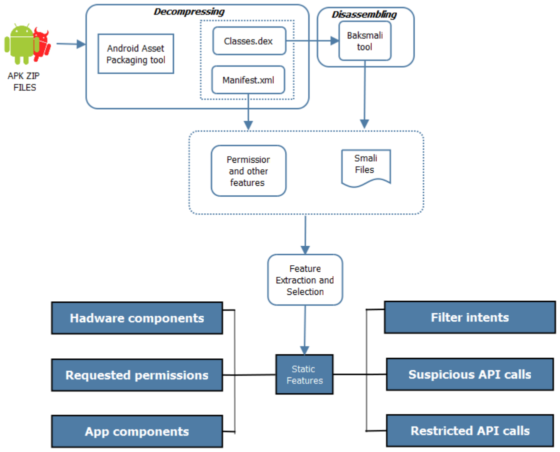


FIGURE 2. Static feature extraction.

TABLE 1. Overview of feature sets.

Feature sets		
manifest	S1	Hardware components
	S2	Requested permissions
	S3	Application components
	S4	Filtered intents
dexcode	S5	Restricted API calls
	S6	Used permission
	S7	Suspicious API calls
	S8	Network addresses

detection [32], [35]–[39]. The feature extraction methods are shown in Figure 2. Also, Table 1 describes some features sets of static methods.

Furthermore, some static features detection methods are shown in Table 2. The k-nearest neighbors machine learning classifier achieves better performance and accuracy in the detection of the malware. However it takes more processing time with a large amount of data. That's why most of the authors used Support Vector Machine and Random Forest classifiers. Therefore, we use and enhance the Naive Bayes algorithm which is based on decision tree(Random forest is based on decision tree) for Android malware detection.

1) PERMISSION-BASED ANALYSIS

Since the Android security model is based on application permissions, the permission set was extracted from the manifest file. Every application must have the privileges needed to access different features. During the application installation, the Android platform asks the user whether to grant the requested permissions. There are some permissions, which can be exploited by malicious applications. For example, a malicious application may use the permissions to access the SD card and the Internet, in order to access and filter sensitive information on an SD card.

Among the 145 permission set, 48 permission are risky permissions which are mentioned in previous literature [19]–[21] as shown in Table 3. Moreover, Li *et al.* [43], developed a SIGPID framework to detect the risky permissions. Moreover, the author extract top 22 risky permission as mentioned in Table 4. Furthermore Kumar *et al.* [18], used a data-mining technique to extract the risky permission, based on association rule set of risky permission shown in Table 5.

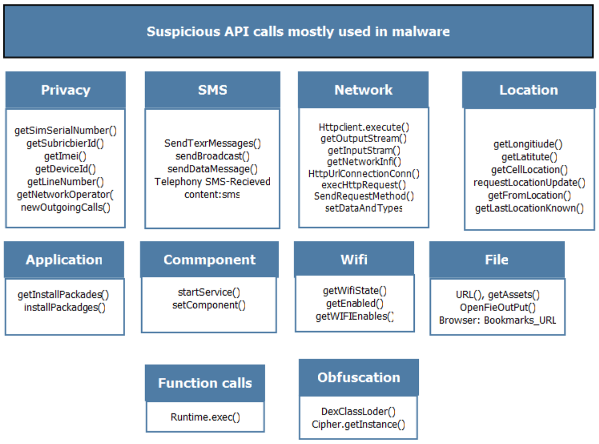


FIGURE 3. Suspicious API calls.

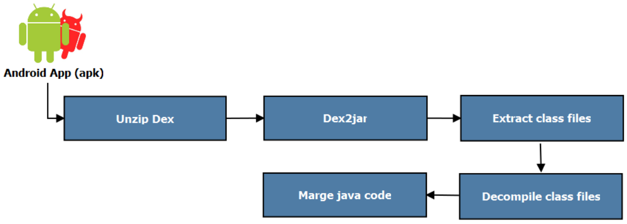


FIGURE 4. Workflow of android file decompiling.

2) SUSPICIOUS API CALLS

The second solution is a static analysis of the source code of the app. Malicious codes usually use a combination of services, methods and API calls that is not common for non-malicious applications [12]. To differitiat malicious and non-malicious applications, the machine learning algorithms are able to learn common malware services such as combinations of APIs and system calls. Figure 3 shows the some of suspicious API calls, which are mostly used by malware applictions. Figure 4 shows the extracted features from the APK file that contains the classes.dex file.

B. DYNAMIC ANALYSIS

AntiMalDroid [63] was a machine learning method to extract dynamic features, which uses the detection technique based on behavioral sequences as feature vectors with SVM. Also, DroidDolphin [45] use Support Vector Machine with features that obtained dynamically. Afonso *et al.* [64] proposed a dynamic API calls and framework calls to track and study bolster vector machines, J48, IBk (an example based approach), BayesNet Pool, BayesNet K2, Random Forest, and Naive Bayes.

Figure 5 shows the feature extraction method and detection technique of the dynamic analysis. Many machine learning

TABLE 2. Static features detection methods.

Ref	Features	Accuracy	Machine Learning Models	Contribution	Limitation
[40]	Permission	91.75%	Random Forest	Permission based approach using KNN clustering	Risky permission not founded
[41]	Permission	81%	C4.5, SVM	The framework quick identify the malicious permission	It uses the limited number of malware. it require the evidence
[42]	Permission	88.20%	HMNB	Probabilistic generative models for ranking the permission. it identifies ranging from the simple Naive Bayes, hierarchical mixture models	Susceptible to adversarial attack
[14]	Permission	-	AHP	a global threat score deriving set of permissions required by the app	Only depends on permissions with known limitations- susceptible to attack
[43]	Permission	98.6	J48	Build a framework for based on SIGPID. It extracts top 22 permissions.	Susceptible to impersonate attack
[17]	Permission	92.79%	Random Forest	Design a model which score the malicious permission	Susceptible to adversarial attack
[44]	Permission	94.90%	Random Forest	It uses the classification algorithm to detect the malware.	Susceptible to adversarial attack
[19]	Permission, API calls	92.36%	Random Forest		Susceptible to adversarial attack
[45]	Permission, API calls, intent	97.87%	k-nearest neighbors	Design a DroidMat Framework which is based on manifest and API call tracing	Susceptible to adversarial attack
[46]	API call	99%	k-nearest neighbors	It mitigate Android malware installation through providing lightweight classifiers	Susceptible to impersonate attack
[47]	API call	93.04%	Signature matching	It measure the similarity of malware	Susceptible to impersonate attack
[48]	API call	96.69%	SVM	The paper use malicious-preferred features and normal-preferred features for the detection of malware	Susceptible to impersonate attack
[49]	ICC related features	97.40%	SVM	Design a ICCDetector framework which classify the malware based on android intent filters	Susceptible to impersonate attack
[50]	Permission, command, API calls	98.60%	Parallel classifier	This paper combine the machine learning classifiers to classify the malware.	Susceptible to impersonate attack
[51]	Requested permissions-used permission-ssensitive API calls-Actions-app components	F1 97.3 Prec. 98.2 Recall 98.4	DBN	DroidDeep for detection of malware using deep belief network	Susceptible to adversarial attack
[52]	Risky Permissions-dangerous API calls	F1- 94.5 Recall-94.5 Prec-93.09	DBN	Proposed DroidDeepLearner combines risky permission and dangerous API calls to build a DBN classification model.	Susceptible to adversarial attack
[53]	API call blocks	ACC 96.66%	DBN	DroidDelver Detection system is used to identify malware using an API call block.	Susceptible to adversarial attack
[54]	Requested permission	Acc 93%	CNN-AlexNet	Proposed a detection system that converts the requested permissions into an image format and then uses CNN for classification	Only depends on permissions with known limitations- susceptible to attack
[55]	323 features	F1 95.05	DBN	An identification system designed by FlowDroid uses data flow analysis to identify malware.	Susceptible to adversarial attack
[56]	Learn to detect sequences of opcode that indicate malware	ACC 98 Prec. 99 Recall 95 F1 97	CNN	Developed a detection system that uses automatic functions to learn from raw data and to treat the disassembled code as text	Although trained on a large dataset, performance dropped when tested on a new dataset- Susceptible
[57]	API call sequence	Acc 99.4 Prec. 100 Recall 98.3 Acc 97.7	CNN	The proposed method based on API call sequence that can use the multiple layers of CNN.	Susceptible to impersonate attack
[51]	Extract features from the transferred images		CNN	Proposed a RGB scheme based on color representation.	Results showed that human experts are still needed in the collection and updating of long- term samples. Susceptible to an attack

TABLE 2. ***(Continued.)*** **Static features detection methods.**

Ref	Features	Accuracy	Machine Learning Models	Contribution	Limitation
[58]	Dangerous API calls-risky permissions	Recall 94.28	DBN	DBN was used to create an automatic malware classifier	Susceptible to adversarial attack
[59]	API calls Permissions-Intent filters	Prec 96.6 Recall 98.3 ACC 97.4 F1 97.4	CNN	Presented system detection of malware DeepClassifyDroid Android based on CNN	Susceptible to impersonate attack
[60]	API calls	Acc 95.7	DBN	Suggested approach to image texture analysis for malware detection	Risky permission not founded
[61]	Permissions requested permissions filtered intents restricted API calls-hardware features-code related features suspicious API calls	Acc 98.8 Recall 99.91 F1 99.82	CNN	A hybrid malware detection model has been developed using CNN and DAE	It uses the limited number of malware. it require the evidence
[16]	API sequence calls	F1 96.29 Prec 96.29 Recall 96.29	CNN	MalDozer used natural language processing technique to detect Android malware, that can identify the malware family attributes.	Susceptible to adversarial attack
[62]	The semantic structure of Android bytecode	Acc 97.74	CNN LSTM	DeepRfiner was proposed to identify the malware. The structure of method use the LSTM for semantic byte code	Only depends on permissions with known limitations- susceptible to attack
[63]	Permissions API Calls	Prec 97.15 Recall 94.18 F1 95.64	DNN	Implemented DNN - based malware detection engine	Susceptible to impersonate attack
[15]	Code Analysis	Acc 95.4	CNN	The proposed method for analyzing a small portion of raw APK using 1-D CNN	Susceptible to adversarial attack

TABLE 3. **Permission set mostly used in malware.**

Risky Permissions	
ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
CAMERA	DISABLE_KEYGUARD
CHANGE_NETWORK_STATE	RESTART_PACKAGES
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
WRITE_SETTINGS	GET_TASKS
RECEIVE_MMS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
READ_HISTORY_BOOKMARKS	RECEIVE_BOOT_COMPLETED
ACCESS_NETWORK_STATE	CALL_PHONE
READ_EXTERNAL_STORAGE	ACCESS_FINE_LOCATION
EXPAND_STATUS_BAR	ADD_SYSTEM_SERVICE
PERSISTENT_ACTIVITY	INTERNET
GET_ACCOUNTS	WRITE_SMS
PROCESS_OUTGOING_CALLS	CHANGE_CONFIGURATION
READ_HISTORY_BOOKMARKS	GET_PACKAGE_SIZE
WAKE_LOG	ACCESS_MOCK_LOCATION
WRITE_CALL_LOG	WRITE_HISTORY_BOOKMARKS
READ_PHONE_STATE	RECEIVE_WAP_PUSH
SET_ALARAM	WRITE_SMS
RECEIVE_SMS	READ_SMS

TABLE 4. **Top 22 permissions.**

Random Forest Based Malware Detection for Permissions	
ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
RESTART_PACKAGES	DISABLE_KEYGUARD
READ_EXTERNAL_STORAGE	CHANGE_NETWORK_STATE
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
CAMERA	GET_TASKS
READ_HISTORY_BOOKMARKS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
WRITE_SETTINGS	RECEIVE_BOOT_COMPLETED

algorithm used for dynamic analysis for instance, Logistic regression (LR), K-means Clustering, SVM, KNN_E,KNN, Bayesian network (BN), and Naive Bayes. Table 6 illustrates the accuracy level, dynamic features and related detection methods.

C. HYBRID ANALYSIS

To improve the performance of learning algorithms, the hybrid analysis was developed, which utilizes the dynamic and static features as shown in Figure 6. Some researches proposed multi-classification techniques [70], [71] to obtain high accuracy in the hybrid analysis. Furthermore, The static features include Publisher ID, API call,

TABLE 5. Permission patterns malware and benign.

Permission Patterns	Benign	Malware
Common Android request permission		
READ_PHONE_STATE, ACCESS_WIFI_STATE	2.36	63.08
INTERNET, ACCESS_WIFI_STATE	5.05	63.49
READ_PHONE_STATE	31.87	93.4
ACCESS_WIFI_STATE	5.22	63.49
ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	3.99	60.31
INTERNET, WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE	13.28	65.44
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE	24.21	78.97
INTERNET, READ_PHONE_STATE	31.21	93.078
WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE	13.37	65.53
READ_PHONE_STATE, ACCESS_NETWORK_STATE	24.21	79.05
Common Android Run-time Permissions		
READ_PHONE_STATE, ACCESS_NETWORK_STATE	23.63	77.18
INTERNET, READ_LOGS	6.85	6.85
READ_PHONE_STATE	30.32	91.69
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE	26.36	77.18
READ_PHONE_STATE,VIBRATE	21.92	65.28
INTERNET, READ_PHONE_STATE	29.9	91.52
READ_PHONE_STATE, READ_LOGS	5.38	46.86
READ_LOGS	6.93	47.6
INTERNET, READ_PHONE_STATE, VIBRATE	21.68	65.12
Unique Android request permission		
READ_PHONE_STATE, WRITE_SMS	0	50.94
INTERNET, READ_PHONE_STATE, ACCESS_WIFI_STATE	0	63.09
ACCESS_NETWORK_STATE, RECEIVE_BOOT_COMPLETED	0	51.68
ACCESS_NETWORK_STATE, WRITE_SMS	0	49.64
RECEIVE_BOOT_COMPLETED, ACCESS_WIFI_STATE	0	42.63
INTERNET, RECEIVE_BOOT_COMPLETED	0	44.75
WRITE_EXTERNAL_STORAGE, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	0	54.53
READ_PHONE_STATE, RECEIVE_BOOT_COMPLETED	0	43.12
INTERNET, SEND_SMS	0	43.12
INTERNET, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	0	60.31
Unique Android Runtime Permissions		
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE, VIBRATE	0	55.42
ACCESS_NETWORK_STATE, VIBRATE, READ_LOGS	0	38.55
READ_PHONE_STATE, ACCESS_NETWORK_STATE, READ_LOGS	0	43.2
READ_LOGS, INTERNET, ACCESS_NETWORK_STATE	0	43.2
READ_PHONE_STATE, VIBRATE, READ_LOGS	0	41.33
INTERNET, VIBRATE, READ_LOGS	0	41.49
READ_LOGS, INTERNET, READ_PHONE_STATE,	0	46.87
ACCESS_FINE_LOCATION, READ_PHONE_STATE, VIBRATE,INTERNET	0	34.23
INTERNET, SEND_SMS	0	33.58
INTERNET, ACCESS_FINE_LOCATION, READ_LOGS	0	28.45

TABLE 6. Dynamic features detection methods.

Ref	Features	Accuracy	Machine Learning Models
[66]	System call	91.75%	Signature Matching
[12]	System call	81%	K-Means
[67]	System call	88.2%	Frequency
[68]	System call	-	Pattern matching
[39]	API call	97.6	KNN_M
[41]	Native Size	99.9%	RF, SVM

Class structure, Java Package name, Crypto operations, Intent receivers Services, Receivers, and Permission, and dynamic are Crypto operations, File operations, Network activity. The APK file extracted static features from classes.dex files, and dynamic features from Androidmanifest.xml file. Hybrid Analysis combines static features and dynamic features. These features are used to detect malicious applications. In [72], following features are selected form static (permission and APICall) and dynamic (SystemCall). Liu *et al.* [72] used the SVM and Naive Bayes machine learning classifier. The SVM classifier used for static analysis achieved 93.33 to 99.28 % accuracy, while the Naive Bayes used for dynamic analysis achieved accuracy up to 90 %. Furthermore, Kim *et al.* [73], used the J48 machine learning classier, and further utilized the features selected from static (permission) and dynamic (API Call). Saracino *et al.* [74], achieved 96.9 % accuracy based on KNN by selecting the static feature (permission) and dynamic (critical API, SMS, User activity System call) features.

D. A COMPARISON OF STATIC, DYNAMIC, AND HYBRID ANALYSIS

1) STATIC ANALYSIS

1) Single category features: The advantages of single category features are easy to extract, and low power computation. The limitations associated with this method are code obstruction, imitation attack and low precision.
2) Multi-category features: The advantages of multi-category features are easy to extract, and poses

Name	Used in malicious	Used in benign
PTRACE	Most often utilized [69], [67]	Utilized in benign applications [67]
SIGPROCMASK	Most often utilized [69], [67]	Utilized in benign applications [67]
CLOCK	Most often utilized [69], [70]	-
CLOCK-GETTIME	Utilized in malicious applications [67]	Utilized in benign applications [67]
RECV	Most often utilized [70], [67]	Not Utilized [67]
RECVFROM	Most often utilized [68], [69], [70]	Not Utilized [67]
WRITE	Most often utilized [68], [69], [70]	Utilized in benign applications [67]
WRITEV	Most often utilized [70], [67]	Utilized in benign applications [67]
WAIT4	Most often utilized [69]	
SEND	Most often utilized [70]	
SENDTO	Most often utilized [69], [70]	
MPROJECT	Most often utilized [68], [69], [70]	Utilized in benign applications [67]
FUTEX	Most often utilized [69], [67]	Utilized in benign applications [67]
IOCTL	Most often utilized [69], [67]	Utilized in benign applications [67]
FCNTL64	Most often utilized [67]	Utilized in benign applications [67]
GETPID	Most often utilized [69], [67]	Utilized in benign applications [67]
GETUID32	Most often utilized [69], [67]	Utilized in benign applications [67]
EPOLL	Most often utilized [67]	Utilized in benign applications [67]
EPOLL-CTL	Most often utilized [67]	Utilized in benign applications [67]
EPOLL-WAIT	Most often utilized [70], [68]	Utilized in benign applications [67]
CACHEFLUS	-	-
READ	Most often utilized [69], [70]	Utilized in benign applications [67]
READV	Most often utilized [70]	-
STAT64	-	-
GETTIMEEOFDAY	utilized in malicious applications [67]	Utilized in benign applications [67]
ACCESS	Most often utilized [70], [68]	Utilized in benign applications [67]
PREAD	-	-
UMASK	Most often utilized [67]	Not Utilized [67]
CLOSE	utilized in malicious applications [67]	Utilized in benign applications [67]
OPEN	Most often utilized [70], [67]	Utilized in benign applications [67]
MMAP2	utilized in malicious applications [67]	Utilized in benign applications [67]
MUNMAP	-	-
MADVISE	utilized in malicious applications [67]	Utilized in benign applications [67]
FCHOWN32	Most often utilized [67]	Not Utilized[67]
PRCTL	Not Utilized [67]	Utilized in benign applications [67]
BRK	Most often utilized [67]	Not Utilized[67]
LSEEK	Utilized in malicious applications [67]	Utilized in benign applications [67]
DUP	Utilized in malicious applications [67]	Utilized in benign applications [67]
GETPRIORTY	Utilized in malicious applications [67]	Utilized in benign applications [67]
PIPE		
CLONE	Utilized in malicious applications [67]	Utilized in benign applications [67]
FSYNC	Most often utilized in [67]	Not Utilized[67]
GETDENTS64	Utilized in malicious applications [67]	Utilized in benign applications [67]
GETTID	Utilized in malicious applications [67]	Utilized in benign applications [67]
LSTA64	Utilized in malicious applications [67]	Utilized in benign applications [67]
FORK	-	-
NANOSLEEP	Not Utilized [67]	Only Utilized in benign applications [67]
RECVMSG	-	-
CHMOD	Utilized in malicious applications [67]	Utilized in benign applications [67]
SENDMSG	Most widely Utilized[69]	-
FLOCK	Not Utilized [67]	Only Utilized in benign applications [67]
MKDIR	Most often utilized [67]	Not Utilized [67]
CONNECT	Most often utilized [67]	Not Utilized [67]
POLL	Not Utilized [67]	Only Utilized in benign applications [67]
RENAME	Most widely Utilized [70]	Not Utilized [67]
SETPRIORITY	-	-
SETSOCKOPT	Most often utilized [67]	Not Utilized [67]
SOCKET	Most often utilized [67]	Not Utilized [67]
UNLINK	-	-
GETSOCKOPT	-	-
BIND	Most often utilized [67]	Not Utilized[67]
FTRUNCATE	Utilized in malicious applications [67]	Utilized in benign applications [67]
GETSOCKNAME	-	
INOTIFY	-	
RESTART	-	

Name	Used in malicious	Used in benign
SCHED	Utilized in malicious applications [67]	Utilized in benign applications [67]
GETRLMIT	-	
LGETXATTR	-	
READLINK	-	
SOCKETPAIR	-	
SATAFS64	Utilized [67]	Not Utilized[67]
FDATSYNC	Utilized [67]	Not Utilized[67]
GETPPID	-	
KILL	-	
PWRITE	Utilized [67]	Not Utilized[67]
MSGGET	Used [67]	Not Utilized[67]
RMDIR	Used [67]	Not Utilized[67]
SELECT	Used [67]	Not Utilized[67]
SEMGET	Used [67]	Not Utilized[67]
SEMOP	Used [67]	Not Utilized[67]
CHDIR	Not Utilized [67]	Only Utilized in benign applications [67]
GETCWD	Not Utilized [67]	Only Utilized in benign applications [67]
RT_SIGRETURN	Utilized [67]	Only Utilized in benign applications [67]
SIGACTION	Not Utilized [67]	Only Utilized in benign applications [67]
SYS_281	Not Utilized [67]	Only Utilized in benign applications [67]
SYS_283	Not Utilized [67]	Only Utilized in benign applications [67]
SYS_224	Utilized [67]	Not Utilized[67]
SYS_248	Utilized [67]	Not Utilized[67]
SYS_290	utilized in malicious applications [67]	Utilized in benign applications [67]
SYS_292	utilized in malicious applications [67]	Utilized in benign applications [67]
SYSCALL_903042	utilized in malicious applications [67]	Utilized in benign applications [67]
FSTAT64	utilized in malicious applications [67]	Utilized in benign applications [67]

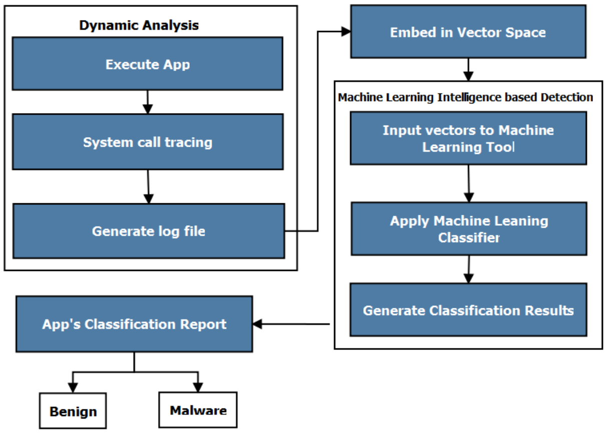

Dynamic Analysis
Execute App
System call tracing
Generate log file
Embed in Vector Space
Machine Learning Intelligence based Detection
Input vectors to Machine Learning Tool
Apply Machine Leaning Classifier
Generate Classification Results
App's Classification Report
Benign
Malware

FIGURE 5. Dynamic feature extraction and detection.

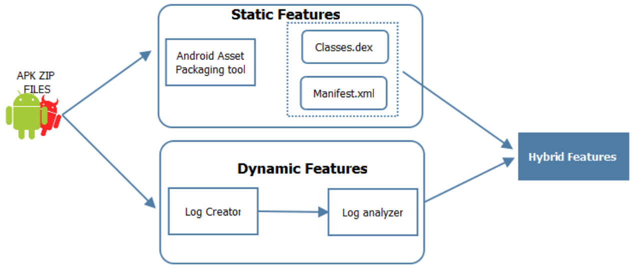


FIGURE 6. Hybrid feature extraction.

high accuracy. The limitations associated with this method are Mimicry attack, high computation, code obfuscation, and difficult to handle multiple features.

2) DYNAMIC ANALYSIS

1) Single category features: it poses a better accuracy and easy to recover code obfuscation as compared with static analysis. However, its feature extraction process is difficult, and it consumes high resources.
2) Multi-category features: It gives better accuracy and easy to recover code obfuscation as compared with a static and dynamic single category. The limitations of this approach are: 1) difficult to handle multiple features, 2) high resources, and 3) more time computation.

3) HYBRID ANALYSIS

The main benefits of hybrid analysis are to perform the highest accuracy as compared to static and dynamic analysis. The limitations are 1) highest complexity, 2) framework requirement to combine the static and dynamic features, 3) more resources utilization, and 4) time-consumption.

III. PROPOSED SCHEME AND METHODOLOGY

In this section, we propose the malware detection framework which is based on three techniques, i) Clustering Algorithm ii) Naive Bayes Classifier for Multi-Feature iii) Blockchain based malware detection framework.

Overall architecture of the proposed system is shown in Figure 7. A new blockchain-based framework was presented to evaluate the performance of malware detection. The proposed machine learning technique provides an efficient approach to train the model, further stores and exchanges the trained model results throughout the blockchain network for spreading the information of newly generated malware. Our proposed framework holds the information about the new types of malware. So, the shared database can control new types of malware and secure devices. It helps to increase malware detection accuracy by providing a dynamic way of sharing information about the new types of malware using application layer. The ledger employs a link list or chain of

TABLE 7. Hybrid analysis methods.

Ref	Methodology	Tools	Achements	Limitations
[76]	Decompress and decompile the Android app using the tool Baksmali. Scans decompiled samli files to extract static patterns. Generate static behavior vector. Installs and executes the applications on emulator Runs monkey to give user inputs Hijacks system calls using LKM logs the system calls	Baksmali Monkey tool Emulator	can detect the malicious system calls at kernel space	Insufficient test results for malware detection No comparison of the system is provided against any other malware detection techniques. Not any classification results are available Increase in malware detection rate is not shown Incomplete evaluation system
[77]	Detects known malware samples by filtering and foot printing based on permission. Detects zero-day malware through heuristic filtering and dynamic monitoring of execution	–	Successfully detects 211 malicious apps among 204,040 apps. +Detect two zero-day malware Droid Dream light and Plankton Achieves 86.1 accuracy.	This study is limited to two heuristics Permission based filtering only considered the essential permission of 10 malware families
[65]	Pre-process the App through API Monitor to obtain static features such as API calls. Install the app on AVD. Uses APE_BOX, combination of DroidBox and APE, to collect the runtime activities and simulation of GUI based event. Combines the static and dynamic features and apply SVM classification.	API Monitor APE DroidBox LIBSVM	Achieves 86.1% accuracy	Time consuming due to use of emulators High resource consumption in log collection. Malware can easily evade-anti-emulator techniques.
[78]	Extract the static features from manifest file and disassembled dex file using Aapt Extracts dynamic features using CuckooDroid Maps the features into vector space and perform vector selection. Uses LinearSVC classifier in Misuse detection to classify the application, if app is malware uses signature based detection to identify the malware. Applies anomaly detection if App is not classified by misuse detection and use signature based detection to identify the family of malware.	Android Asset Packaging Tool	Detects known malwares and their variants with 98.79% true positive rate. Detects the zero-day malwares real positive rate with 98.76 percent accuracy	Comparison of proposed scheme with other well-known malware detection schemes e.g., RiskRanker, Drebin, Kirin etc. is not provided.
[79]	Parameters related to permissions, such as broadcast receivers , intents and services, are decompiled from the manifest file in the static analysis phase using Aapt. In the behavior analysis phase, the Android emulator app is executed and the functions related to user interactions, java- based and native function calls are extracted. Performs feature on the basis of information gain and record them in CSV file. Rule generation module uses CSV file to create rules and maps the permission against the function calls for classification	Android Asset Packaging Tool.	Achieves 96.4% detection rate	High time for scanning. High electricity consumption. High consumption of resources / storage.
[80]	Extracts PSI from binary code files as static features sort features according to the frequency of occurrence in each file. Selects feature with occurrence frequency above certain threshold value and create static feature vector. For dynamic feature use cuckoo malware analyzer. For each file, create API call grams and analyze API call sequences based on the n- gram method. Selects grams of API call above a certain threshold value and creates a dynamic function vector. Concatenates both feature vector for each file and input them to Machine learning classifiers.	WEKA	Classifies 98.7 percent accurate unknown applications.	Comparison of proposed scheme with other well-known malware detection schemes e.g., RiskRanker, Derbin, DroidRanger etc is not provided
[81]	Extracts sensitive API calls and permissions as static features. Logs dynamic action for dynamic analysis Applies deep learning model for classification	7ZIP, XML-printer2 Tinyxml , DropidBOX Baksmali	Detects 96.7 percent accurate malware.	Unrealistic malware for dynamic analysis that does not display malicious behavior throughout the monitoring interval can evade the detection system
[73]	Decompiles applications using Akptool and analyze the decompiled results. Automatically switches to static analysis if app is correctly decompiled. Performs extraction of static features, permission and API calls, from manifest and smali files. Inputs the feature vectors to machine learning classifiers, SVM, KNN and Naive Bayes. If application do not correctly decompile then it performs dynamic analysis by operating the app with monkey tool and monitoring the app's actions using strace. Generates the feature vector of traced system call logs and apply the machine learning classifier on the feature vector for classification.	APK tool Strace Monkey ool	Achieves 99% accuracy as a result of static analysis and 90% accuracy as a result of dynamic analysis.	Only static or dynamic analysis can be performed on the application, so that the dynamically labelled data cannot be detected in an easy way for static analysis Only the executed code is analyzed when dynamic analysis is carried out. The non-executed code remains undetected.

TABLE 7. ***(Continued.)*** **Hybrid analysis methods.**

Ref	Methodology	Tools	Achements	Limitations
[75]	Extracts features at four different levels: user level, application level, kernel level, and package level user activities at user level and market information and riskiness of application at package level Generates feature vectors consisting of 14 features and input the vector to KNN classifier. Notifies the user about malicious apps and helps the user to block and remove them through UI			Only runs on rooted devices with a carnal having module support due to which it has not been conceived for distribution in the mass market. Pre-installed apps are not analyzed by the app evaluator. Thus, will not be included in apps suspicious list and so will not be dejected against known malware behavior patterns. only the apps identified as risky or added to the apps suspicious list. 9.4% memory overhead because classifier requires the training data and memory.
[82]	Feature collector collects static features of at the application at installation. GramDroid a web tool that extracts the features of applications and provides their visual representation in order to identify the threads posed by the application Local detector classifies the application as legitimate, malware or risk using static features. Response manager gives control to use if app as detects as malware. Cloud detector performs detailed dynamic analysis at a remote server if app is detected is risk by local detector updates the database if app is detecting malware.		From top 20 enlisted frequently requested permission	
[83]	The Android device's client application captures the application's specific information and sends it to the server. Detailed analysis and application execution based on emulation is carried out. Otherwise, the APK file will be sent from the client device to the server.	Androgaurd	Detects 99% accurate malware applications.	The malware can easily evade emulation- based detection
[84]	User permission to detect malware behavior as static analysis. The signature data type contains all applications signature. Android user offers users a malware analysis service.. The central server connects the Android client to the signature database.		Archives 92.5% specificity	It lacks the advantages of dynamic analysis, as dynamic malicious payloads cannot be detected
[85]	Uses static functions, manifest file and code files assembled. Uses system calls and binder transactions as dynamic behavior features. The user and the application monitor and se signature are forwarded to the server which applies generate the signature. The signature matching algorithm.		Achieves 99% accuracy	Overall causes 7.4 percent overhead performance and 8.3 percent overhead memory.

blocks; each block contains the information about number of malware that was validated to the network in the given timestamp. The distributed ledger records all types of malware in history. Using the previous history, we trained our model and detected malware efficiently and fast.in addition to this, it also stores the malware information single federated datasheet, which can help to detect the new type of malware quickly. Putting information about malware in the blockchain improves the performance in terms of runtime malware detection for Android IoT devices. The blockchain executes the machine learning classifier to evaluate the speed of detection of malware.

The proposed framework runs in recurring way described in following three steps as follows:

1) Hackers create a new type of malware.
2) Machine learning identify the malware and re-train the model.
3) The new type of malware information stores in the blockchain database.

More precisely, the first method based on a clustering algorithm, which reduces the high dimensional data and removes unnecessary features. Secondly, we use classification method based on Naive Bayes for multi-feature classification. Finally, a blockchain based framework to detect run time malware is proposed.

A. MODIFIED FEATURE REDUCTION USING CLUSTERING ALGORITHM

The classification algorithm classifies the malware application for each cluster. KNN and c-mean is a very popular algorithm for clustering. Also, the feature extraction is an essential factor for high-dimensional data. To address the problem of irrelevant features, we enhance the clustering algorithm for feature reduction that takes less time for the large dataset

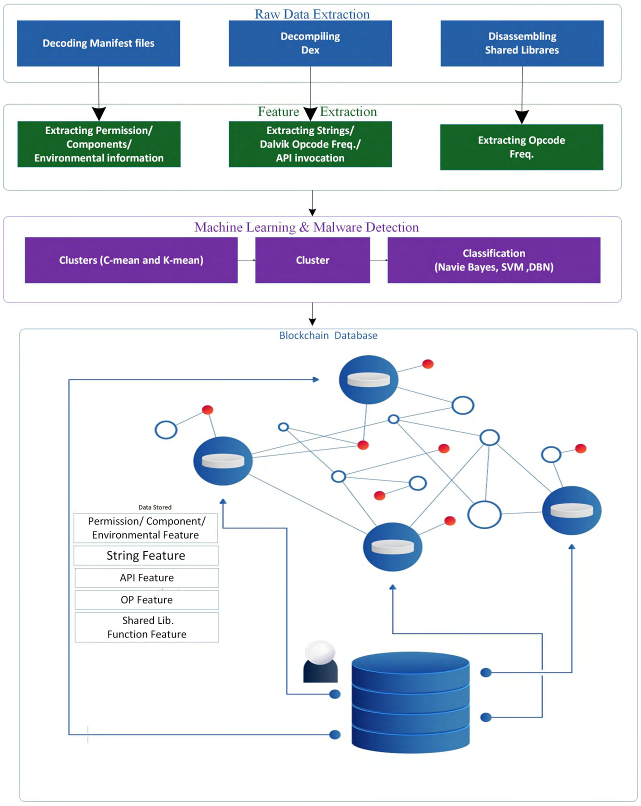

Raw Data Extraction
Decoding Manifest files
Decompiling Dex
Disassembling Shared Librares
Feature Extraction
Extracting Permission/ Components/ Environmental information
Extracting Strings/ Dalvik Opcode Freq./ API invocation
Extracting Opcode Freq.
Machine Learning & Malware Detection
Clusters (C-mean and K-mean)
Cluster
Classification (Navie Bayes, SVM ,DBN)
Blockchain Database
Data Stored
Permission/ Component/ Environmental Feature
String Feature
API Feature
OP Feature
Shared Lib. Function Feature

FIGURE 7. Arcitecture of proposed framework.

and low-cost computation in the training process. Our proposed scheme reduce the high dimensional data and remove unnecessary features. The method calculates the weights of each feature set and iterative reduced the unnecessary features that effectively distinguish malware and benign applications even though malware has many properties similar to benign applications. It handles multidimensional data by reducing the features by weight learning procedure. This property enables our method as a suitable candidate for many types of clustering problems. For example, it can be applied on image recognition, cell formation, topic identification, text summarization and many more. The following are objective of our proposed scheme:

1) Calculating the weights for each feature set.
2) Developing a parametric study for optimization.
3) Iterative reduction of unnecessary features having small weights.

Suppose $x = \{x_{1,}.x_{2,......}x_n\}$ is the dimensional dataset. w_j is the weight of thejth features. The following equation defined in [22]–[27].

$$J(X,Y,Z) = \sum_{p=1}^{c}\sum_{q=1}^{n}\sum_{r=1}^{d}\mu_{pq}^{m}\delta_r w_r(x_{qr}-v_{pr})^2 + \frac{n}{c}\sum_{r=1}^{d}(w_r\log\delta_r w_r) \quad (1)$$

subject to

$$\sum_{p=1}^{c}\mu_{pq}=1, 0\le\mu_{pq}\le 1, \quad \sum_{r=1}^{d}w_r=1,\ 0\le w_r\le 1$$

We solve the problem in three steps. Firstly fix the Y=Yˆ then Z=Zˆ, in the lst minimize the J˜(X,Yˆ,Zˆ) with respect to X. Larangain function obtain by below equation.

$$\tilde{J}(X,Y,Z) = \sum_{p=1}^{c}\sum_{q=1}^{n}\sum_{r=1}^{d}\mu_{qp}^{m}\delta_r w_r(x_{qr}-v_{pr})^2 + \frac{n}{c}\sum_{r=1}^{d}\times(w_r\log\delta_r w_r)+\lambda_1\left(\sum_{p=1}^{c}\mu_{qp}-1\right) + \lambda_2\left(\sum_{r=1}^{d}w_r-1\right) \quad (2)$$

The partial derivative of the equation is taken from above equation concerning shown in below

$$\frac{\partial\tilde{J}}{\partial\mu_{pq}} = \sum_{r=1}^{d}m\mu_{qp}^{m-1}\delta_r w_r(x_{qr}-v_{pr})^2+\lambda_1=0 \quad (3)$$

From the above equation the enhance equation shown below

$$\mu_{pq} = \frac{\left(\sum_{r=1}^{d}\delta_r w_r(x_{qr}-v_{pr})^2\right)^{-1/m-1}}{\sum_{p=1}^{c}\left(\sum_{r=1}^{d}\delta_r w_r(x_{qr}-v_{pr})^2\right)^{-1/m-1}} \quad (4)$$

Secondly, we fix the X=Xˆ and Z=Zˆ, and then minimize the J(Xˆ,Y,Zˆ) from below equation

$$\frac{\partial\tilde{J}}{\partial v_{pr}} = -2\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r w_r(x_{qr}-v_{pr})=0. \quad (5)$$

From the above equation the enhance equation shown in below.

$$v_{pr} = \frac{\sum_{q=1}^{n}\mu_{pq}^{m}x_{qr}}{\sum_{q=1}^{n}\mu_{pq}^{m}} \quad (6)$$

Finally, we fix the X=Xˆ and Y=Yˆ, and then minimize the J(Xˆ,Yˆ,Z) from below equation is obtained from above equation

$$\frac{\partial\tilde{J}}{\partial w_r} = \sum_{p=1}^{c}\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r(xqr-v_{pr})^2 + \frac{n}{c}(\log\delta_r w_r+1)+\lambda_2=0. \quad (7)$$

From the above equation the enhance equation shown below.

$$w_r = \frac{\frac{1}{\delta_j}\exp\left(\frac{-c\sum_{p=1}^{c}\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r(x_{qr}-v_{pr})^2}{n}\right)}{\sum_{p=1}^{d}\frac{1}{\delta_p}\exp\left(\frac{-c\sum_{p=1}^{c}\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r(x_{qr}-v_{pr})^2}{n}\right)} \quad (8)$$

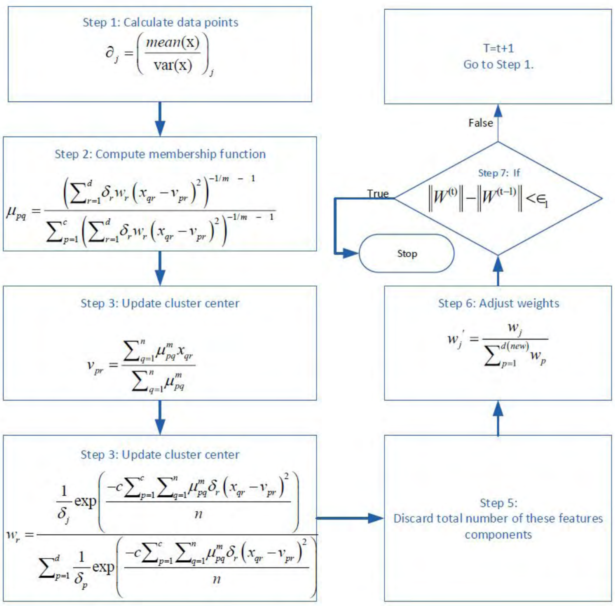

Step 1: Calculate data points
$\partial_j = \left(\frac{mean(\mathrm{x})}{\mathrm{var(x)}}\right)_j$
Step 2: Compute membership function
$\mu_{pq} = \frac{\left(\sum_{r=1}^{d}\delta_r w_r\left(x_{qr}-v_{pr}\right)^2\right)^{-1/m\ -\ 1}}{\sum_{p=1}^{c}\left(\sum_{r=1}^{d}\delta_r w_r\left(x_{qr}-v_{pr}\right)^2\right)^{-1/m\ -\ 1}}$
Step 3: Update cluster center
$v_{pr} = \frac{\sum_{q=1}^{n}\mu_{pq}^{m}x_{qr}}{\sum_{q=1}^{n}\mu_{pq}^{m}}$
Step 3: Update cluster center
$w_r = \frac{\frac{1}{\delta_j}\exp\left(\frac{-c\sum_{p=1}^{c}\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r\left(x_{qr}-v_{pr}\right)^2}{n}\right)}{\sum_{p=1}^{d}\frac{1}{\delta_p}\exp\left(\frac{-c\sum_{p=1}^{c}\sum_{q=1}^{n}\mu_{pq}^{m}\delta_r\left(x_{qr}-v_{pr}\right)^2}{n}\right)}$
Step 5:
Discard total number of these features components
Step 6: Adjust weights
$w_j' = \frac{w_j}{\sum_{p=1}^{d(new)}w_p}$
Step 7: If
$\left\|W^{(t)}\right\|-\left\|W^{(t-1)}\right\|<\in_1$
True
False
Stop
T=t+1
Go to Step 1.

FIGURE 8. Flow diagram of proposed clustering algorithm.

To achieve the constraint, $\sum_{p=1}^{d} wj = 1$ we adjust W_j by

$$w_j' = \frac{w_j}{\sum_{p=1}^{d(new)} w_p} \tag{9}$$

Compared with the feature set before optimization, the feature set obtained by the above method is mixed with the feature of uncertainties. This feature set can reduce the training time of the model and the prediction time. Our proposed algorithm for removing unnecessary features shown in Figure 8.

The similarity-based features are extracted from the API and Opcode features, which are further used during the feature vector generation to achieve reduced feature set. Firstly, we normalized the values in the range of [0,1]. After that, we measure the distance of malicious features by using the centroid of the cluster. The similarity is measured by calculating the minimum distance among various features. Moreover, by evaluating the similarity values, it can be observed that each feature matrix can contain similarities to the centroids of multiple clusters computed with known malware applications. Consequently, the input malware's feature can belong to a certain similarity based feature cluster.

To avoid the irrelevant features and improve the performance, we select the small weights for the feature reduction. In this process, we used threshold function to determine unimportant features. It can be noticed that the collected dataset has a number (n) of data points (d). Every data point has a different feature, to set the weight constraint $\sum_{j=1}^{d} W_j = 1$. If the data points are separated by a distance d, it reduces the the features by $1/d$. In addition, the proposed feature reduction algorithm fit for the small points. $1/\sqrt{nd}$ is a suitable threshold for removing the irrelevant features. Furthermore, the value of ∂_j can be to estimated by calculating the distance between the cluster center and data points, that can reduce the distance when the center is small, and entropy function $\sum_{j=1}^{d} W_j = \log w_j$ was used to control the feature weights.

TABLE 8. Attribute of naive bayes classifier.

Attribute	Description
T	training dataset
D	The attributes from the data collection appeared in the tree
D_t	to find the attribute subset in the training dataset
W_i	weight
A_i	Each Attribute
$P(C_i)$	prior probability for each class
x_i	each training instance $x_i = \{x_{i1}, x_{i2}, ...x_{ih}$
C	Set of classes $C = (C_1, C_{2,.......,}C_n)$

There are various criteria for evaluating classifiers and every criteria is based on the selected goals. According to our dataset the parameters are $\alpha = 0.5$, $u = (33)^t$, $\sum k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Additionally, since the weights of clustering depend on initialization so we initialize the weights as $W = [0.250]$.

B. NAIVE BAYES CLASSIFIER FOR MULTI-FEATURE

DREBIN [30] method was used for static analysis to build datasets based on application permissions, API and other features. Our approach is more efficient than DREBIN when combining the features. Also, the support vector machine (SVM) algorithm was used to classify malware datasets. It is challenging to enhance DREBIN. Previous research [17], [19], [40], [44] shows the Random Forest achieved the best performance in malware detection. Random forest is the collection of decision trees or more precisely it makes the forest of decision trees.Therefore, we enhanced the Naive Bayes classifier based on decision trees. It can be utilized for both classification and regression that can achieve high robustness and accuracy. Table 8 shows the attribute of the proposed method and the Figure 9 shows the description of algorithm.

C. BLOCK–CHAIN BASED MALWARE DETECTION FRAMEWORK

Recently, blockchain technology has gained more attention. It is a vital technology that comes from the consensus mechanism, such as fault-tolerant distributed computing system [85].The objective of IoT data analysts is to gain a deep insight into the IoT data recorded on the blockchain, which is based on Delegated Proof of Stake (DPOS), Proof of Work (PoW), Practical Byzantine Fault Tolerance (PBFT), Proof of Stake (PoS),and etc. Therefore, to address the multi-feature malware detection problem, we design a technique which uses permissioned block-chain based framework for storing information of malware features. The overall architecture consist four layer: 1) Network Layer, 2) Storage Layer 3) Support Layer 4) Application Layer, this design scheme

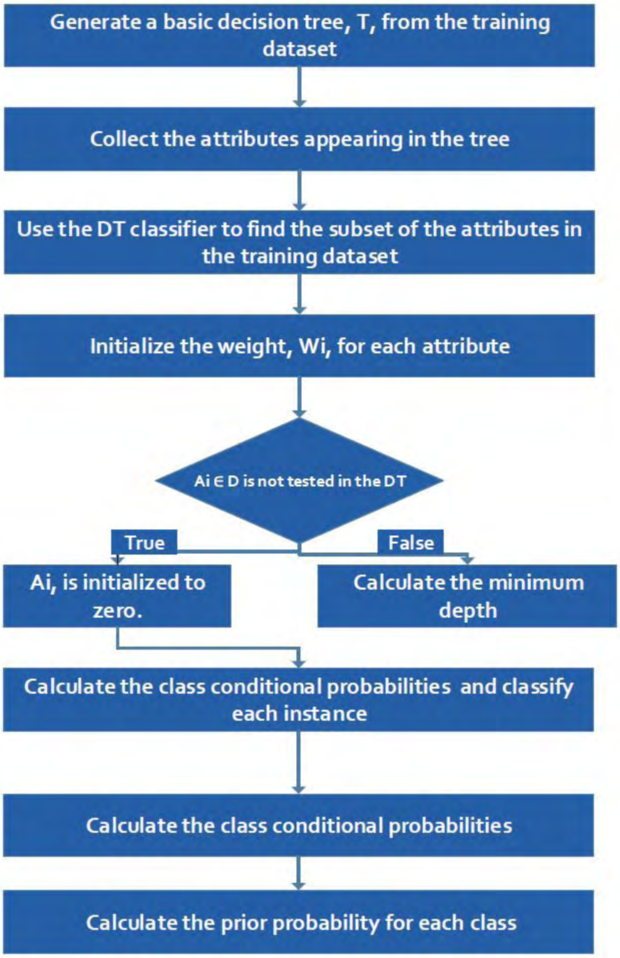

Generate a basic decision tree, T, from the training dataset
Collect the attributes appearing in the tree
Use the DT classifier to find the subset of the attributes in the training dataset
Initialize the weight, Wi, for each attribute
Ai ∈ D is not tested in the DT
True
False
Ai, is initialized to zero.
Calculate the minimum depth
Calculate the class conditional probabilities and classify each instance
Calculate the class conditional probabilities
Calculate the prior probability for each class

FIGURE 9. Flow chart of naive bayes algorithm.

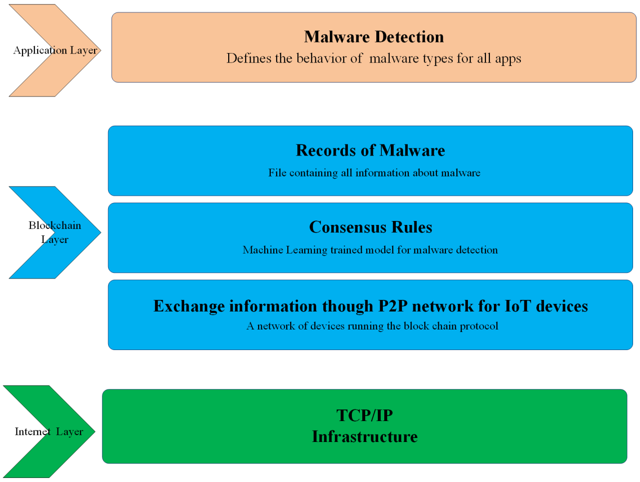

Application Layer
Malware Detection
Defines the behavior of malware types for all apps
Blockchain Layer
Records of Malware
File containing all information about malware
Consensus Rules
Machine Learning trained model for malware detection
Exchange information though P2P network for IoT devices
A network of devices running the block chain protocol
Internet Layer
TCP/IP
Infrastructure

FIGURE 10. Blockchain based malware detection for IoT devices.

cost-creditable for the IoT devices. Figure 10 shows the proposed architecture of blockchain technology for malware detection for IoT devices. The blockchain layer (described in Figure 10) consists of network layer, support layer and storage layer.

Algorithm 1 Hybrid Feature Extraction Naive Bayes Classifier

```
function Naïve Bayes classifier
    attr = attribute
    cls = class
    determine the attribute that best classifies the training data;
    T = Build root node holds a class label after splitting on attribute;
    T = From the root node and adding the arc with splitting predicate;
    for arc do
        D = spliting the dataset with predicate to D;
        if stopping criteria is reached then
            T = each leaf node is assigned a class label
        end if
        if stopping criteria is not reached then
            T = DecisionTreeBuild(D);
        end if
        T = add arc from tree T to subtree;
    end for
    for each x, $A_i \in D$ do
        if $A_i <> T$, then
            $W_i = 0$;
        end if
        if $A_i$ is tested in T, then
            d lowest depth of $A_i \in T$, and;$W_i = \frac{1}{\sqrt{d}}$
        end if
    end for
    for each cls, $C_i \in D$ do
        Search for previous probabilities, P($C_i$).
    end for
    for each attr, $AC_i \in D$, and $W <> 0$ do
        for each attr, $A_{ij} \in A_i$ do
            Search for previous probabilities, $P(A_{ij}/C_i)^{W_i}$
        end for
    end for
    for instance, $x_i \in D$ do StateSearch for previous probabilities, $P(C_i/x_i)^{W_i}$.
    end for
end function
```

The proposed architecture describes that the Link nodes can communicate through a P2P connection between devices at the network layer. The characteristics of decentralization consensus mechanism offers some critical benefits such as security, fault tolerance between the communication nodes. Every node can freely join and exit the network. Every node connected with each other and transfer the information to their neighbouring node. The synchronous block uses information through the nodes and build a request and response pattern.

The blockchain can store the malware features of malicious code in the storage layer. The permissioned blockchain provides the all information of malware through the distributed blocks. Every block stores the malware features such as risky permission, suspicious API. The features cannot be modified once stored in the blocks.

Furthermore, the support layer of the Android platform used blockchain technology for data encryption, access control, and key management, etc. The advantage of using the blockchain technology is to identify illegal activity and illegal manipulating of the user's data. Finally, in the application layer blockchain technology protect the user from malicious applications. The structure of the blockchain block is shown in Figure 11. The block mainly composed of two section: 1)Block head section 2) Data section. The block head describes the information of the blocks which can store the Markle root, hash value, version number, and so on. The data section of block stores the malware information such as name, family type, permission feature, sensitive behavior,

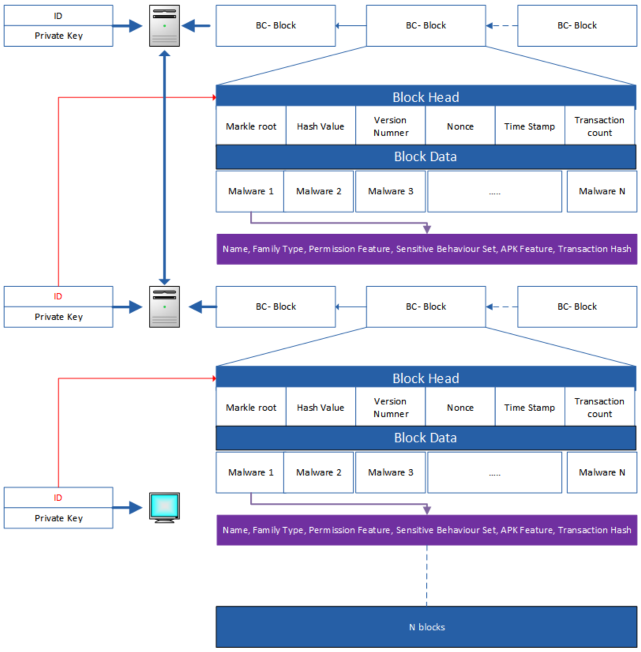


FIGURE 11. Blockchain-based framework for storing multi-features of malware.

TABLE 9. Blockchain attributes.

Keywords	Size	Definition
Pre- Hash	32 bytes	preceding block hash value
Version number	4 bytes	track the protocol or software updates
Timestamp	5 bytes	records the time a block
Transaction_count	15 bytes	number of malware results in the current block
Merkle root	32 bytes	it calculate the malicious codes which detected by block
Nonce	15 byte	randomly recognized as a formal block

APK feature and transaction hash. Table 9 shows the attributes and definitions of all block head contents.

Additionally, the permissioned blockchain provides actual information in a distributed malware database to increase the run-time detection of malware with more speed and accuracy by storing the extracted features in blocks to announce malware features information for all users.

To ensure the reliability of blockchain, unique hash values effectively prevent the fraud. Each data block of detection results contains 1) Permission Feature of the malware, 2) API Call Feature of the Malware, 3) Name of Malware, 4) Installation Packages Features and 5) hash value which uniquely recognized the detection record.

IV. EXPERIMENTAL RESULTS

The proposed framework poses a strong evidence over acquired experiments results. Here, we discuss major aspects for experimentation which include statistics and source of dataset, evaluation measures to understand the performance criteria for exploited machine learning algorithm and result outcomes which gives strong evidence towards the significance of our proposed model.

TABLE 10. Clusters having same label.

	Original Label	Benign or Malware Label
String	89.10%	95.05%
API freq	96.53%	99.36%
Shared Lib.Opcode	75.47%	98.88%
Manifest	61.50%	78.55%
Opcode freq	73.65%	91.88%

A. DATASET

In order to excavate practical significance, it can cover most android permission models,which have their own characteristics. In this paper, our dataset composed of malware and benign applications.The dataset includes 6192 benign and 5560 malware apps, collected from the Google Play Store and Chinese App store. Table 10 shows the shared libraries in applications which helps to exploit for clustering. Besides, the existence in Manifest contains lowest percentage during analysis.

B. EVALUATION MEASURES

Python programming language contains tools for data preprocessing, classification, clustering, regression, association rules, and visualization, which make it the best tool for the data scientist to measure and test the performance of classifiers. There are various criteria for evaluating classifiers and criteria is set based on the selected goals. For the classification methods are evaluating such as True Positive Rate (TPR) and False Positive Rate (FPR) and classification accuracy. we used the following standard measurements: Given the number of true positives for malicious applications using the following formulas:

$$TPR = \frac{T_p}{T_p + F_n}$$

False Positive rate is the proportion of negative instance for the benign apps

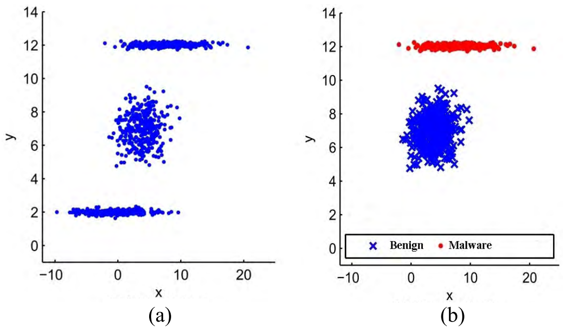
$$FPR = \frac{F_p}{F_p + T_n}$$

The accuracy is defined as below equation

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

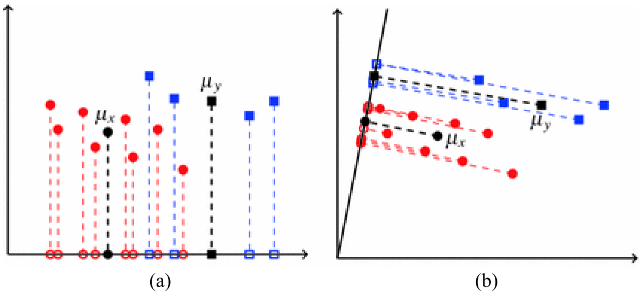
C. RESULTS DISCUSSION

The experimental results are carried out to prove the significance of extracted features of proposed framework. The high-dimensional and noisy dataset is optimized according to the clustering algorithm, while the clustering results can be seen in Figure 12. In clustered data, the red color shows the malicious samples and blue colour shows the benign samples over two axis after feature reduction. As we can see in Figure 13 (a) shows the means widely separated cluster is not efficient but Figure 13 (b) shows the efficient clustering.



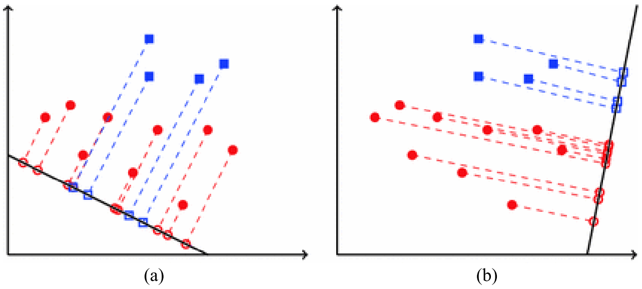
(a) (b)

FIGURE 12. Clustering of the benign and malware apps.



(a) (b)

FIGURE 13. Efficient clustering based feature analysis. (a) Means widely separated. (b) Means closer together.



(a) (b)

FIGURE 14. Efficient projection feature analysis. (a) A projection. (b) A better projection.

TABLE 11. Comparison different machine learning classifiers.

Alogrthim	TPR	FPR	ACC
Improved Naive Bayes	98.2	98.2	98.0
SVM	95.2	95.2	95.0
KNN	75.8	87.5	92.0
Naive Bayes	80.5	80.7	90.5
DBN	84.6	85.0	87.0

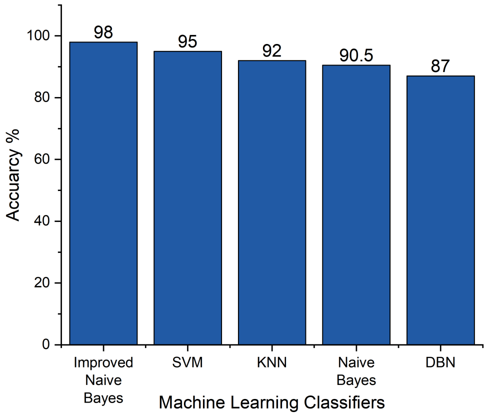


FIGURE 15. Machine learning classifiers accuracy.

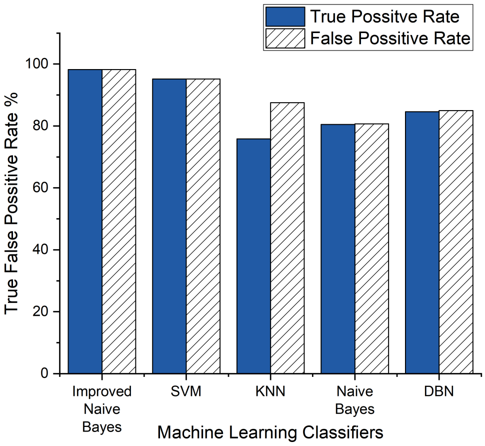


FIGURE 16. Comparison false and true positive rates.

Smoothing over subspace area of feature clearly dominates the visualization of two different cluster. However, it is difficult to distinguish the two clusters by visualization of points distribution. Projections over data points make visualization of two clusters easy for the placement of hyperplane. It is the challenging task to draw such projection of data because it includes deep analysis of data point during separation process. The results of different projections of sample data can be seen in Figure 13 and Figure 14, which shows that our proposed technique can distinguish the benign and malware samples effectively.

As we can see in Table 11, enhanced Naive Bayes achieved better accuracy compression with other classifiers. The proposed algorithm is more efficient in terms of speed and multi feature extraction. Moreover, proposed framework classify and cluster the malware apps. Naive Bayes algorithm achieves more than 98% accuracy; we achieve the highest recall rate for detection the dangerous features of the Android platform such as permission,API calls,string and etc. Table 11 shows the improved naive Bayes achieved the best performance and provided the highest efficiency and lowest false alarm rate. Figures 15 and 16 show the curacy and false and true positive rates of different machine learning classifiers, respectively.

D. COMPARISON WITH OTHER WORKS

To prove the significance of proposed framework, its performance is compared with state-of-art detection systems. In this context, we investigated the similar approaches that have been previously proposed. From the machine learning-based methods to the general classification-based methods,

TABLE 12. Compersion accuracy with other works.

Authors	Algorthim	Capicity for feature diversity	Accuracy	F-measure
OURs	Proposed NB	High	98%	0.98
[87]	DNN/RNN	medium	90%	NA
[56]	CNN	low	90%	NA
[88]	XGBoost	low	97%	0.97
[41]	Adaboost/ NB/ DT	Low	N.A	0.78
[89]	NB	low	93%	NA
[30]	SVM	low	93.9	NA
[29]	KNN+Kmeans	low	NA	0.91
[90]	Bayesian	low	92%	NA
[81]	SVM	low	NA	0.98
[91]	RF	low	97.5%	NA

TABLE 13. Comparison of accuracy with other works.

Authors	Benign	Malware
OURs	6192	5560
[41]	480	124769
[36]	45	300
[49]	5264	12026
[42]	378	324658
[46]	3978	500
[19]	175	621
[44]	1446	2338
[30]	5560	123453
[50]	2925	3938
[47]	238	1500

various kinds of the Android malware detection methods were studied. As shown in Table 12, the detection accuracy or the F-measure values of our framework were higher than the other methods including the deep learning based methods [43], [56], [70], [73]. Furthermore, the comparison of number of benign and malware apps used in previous and our work are shown in Table 13. The performance comparison shows the significance of the proposed framework (which is based on machine learning and blockchain) over previously designed methods, which further paves a way for its application for android malware detection in IoT devices. More precisely, the performance of this framework overpass other models in term of runtime malware detection. In addition to this, when a certain new type of malware or new feature is added the accuracy is increased further. Therefore, our proposed method can handle the multi-features by combing the advantages of machine learning and blockchain that effectively detect the malware for Android IoT devices.

V. CONCLUSION

The IoT devices are advancing this world towards a new paradigm with its exciting applications such as sensing, smart healthcare, remote monitoring, smart agriculture, etc. Android platform based IoT devices and applications are working hand to hand to realize IoT dreams. Therefore, this research work proposes a framework that combines blockchain and machine learning for effective malware detection of Android IoT devices. The malware information was extracted using machine learning techniques such as clustering and classification, and further this information was stored into the blockchain. Thereby, all malware information stored in the blockchain history can be communicated through the network, and therefore any latest malware can be detected effectively. Our proposed clustering method calculates the weights of each feature set and iteratively reduced the unnecessary features that can be very effective to distinguish between malware and benign applications even though malware has many similar properties of benign applications. Secondly, the classification algorithm is implemented using the naive Bayes classifier based on a decision tree to address multi-feature problems for achieving high accuracy and robustness. Finally, the permissioned blockchain used in our framework provides actual information in a distributed malware database to increase the run-time detection of malware more efficiently.

Furthermore, the experimental results show the proposed framework can achieve higher accuracy for malware detection with a low number of false-negative and false-positive rates. Besides the significance of proposed approach towards malware detection, the limitations associated with this approach are its inability to handle some obfuscation techniques and the feature hiding techniques when decompile the apk using Dex2jar. In future, we plan to develop a framework based on the deep neural network to combine static and dynamic analysis for malware detection using blockchain. Finally, our proposed framework can be applied directly to the Android-based mobile and IoT device to achieve more security and privacy.

REFERENCES

[1] J. S. Park, T. Y. Youn, H. B. Kim, K. H. Rhee, and S. U. Shin, “Smart contract-based review system for an IoT data marketplace,” *Sensors*, vol. 18, no. 10, p. 3577, 2018.

[2] B. L. R. Stojkoska and K. V. Trivodaliev, “A review of Internet of Things for smart home: Challenges and solutions,” *J. Cleaner Prod.*, vol. 140, no. 3, pp. 1454–1464, 2017.

[3] M. Damshenas, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, “M0Droid: An android behavioral-based malware detection model,” *J. Inf. Privacy Secur.*, vol. 11, no. 3, pp. 141–157, Sep. 2015.

[4] J. Walls and K. K. R. Choo, “A review of free cloud-based anti-malware apps for android,” in *Proc. 14th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, vol. 1, Aug. 2015, pp. 1053–1058.

[5] H. Chen *et al.*, “Malware collusion attack against SVM: Issues and countermeasures,” *Appl. Sci.*, vol. 8, no. 10, p. 1718, Sep. 2018.

[6] I. Dogru and K. Ömer, “Web-based android malicious software detection and classification system,” *Appl. Sci.*, vol. 8, no. 9, p. 1622, Sep. 2018.

[7] *Strategy Analytics: Android Captures Record 88 Percent Share of Global Smartphone Shipments in Q3 2016*, Businesswire, San Francisco, CA, USA, 2016.

[8] A. Demontis *et al.*, “Yes, machine learning can be more secure! A case study on android malware detection,” *IEEE Trans. Dependable Secure Comput.*, to be published.

[9] S. Y. Yerima, S. Sezer, and I. Muttik, “Android malware detection using parallel machine learning classifiers,” in *Proc. 8th Int. Conf. Next Gener. Mobile Apps, Services Technol.*, Sep. 2014, pp. 37–42.

[10] W. Enck *et al.*, “TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,” *Commun. ACM*, vol. 57, no. 3, pp. 99–106, 2014.

[11] G. Canfora, F. Mercaldo, and C. Visaggio, "Mobile malware detection using op-code frequency histograms," in *Proc. 12th Int. Joint Conf. E-Bus. Telecommun. (ICETE)*, Jul. 2015, pp. 27–38.

[12] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for android," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices*, Oct. 2011, p. 15.

[13] D. Kim, C. Park, J. Oh, S. Lee, and H. Yu, "Convolutional matrix factorization for document context-aware recommendation," in *Proc. 10th ACM Conf. Recommender Syst.*, Sep. 2016, pp. 233–240.

[14] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Risk analysis of Android applications: A user-centric solution," *Future Gener. Comput. Syst.*, vol. 40, pp. 505–518, Mar. 2018.

[15] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for android malware detection," in *Proc. IEEE 14th Int. Colloq. Signal Process. Appl.*, Mar. 2018, pp. 99–102.

[16] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, Mar. 2018.

[17] A. Kumar, K. S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic analysis of MObile devices using scoring of application permissions," *Future Gener. Comput. Syst.*, vol. 83, pp. 158–172, Jun. 2018.

[18] R. Kumar *et al.*, "Research on data mining of permission-induced risk for android IoT devices," *Appl. Sci.*, vol. 9, no. 2, p. 277, Jan. 2019.

[19] P. P. K. Chan and W.-K. Song, "Static detection of android malware by using permissions and API calls," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 1, Jul. 2014, pp. 82–87.

[20] S.-H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," *J. Netw. Comput. Appl.*, vol. 38, pp. 43–53, Feb. 2014.

[21] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," 2014, *arXiv:1408.2927*. [Online]. Available: https://arxiv.org/abs/1408.2927

[22] W. Peizhuang, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Philadelphia, PA, USA: SIAM, 1983.

[23] M. S. Yang, "A survey of fuzzy clustering," *Math. Comput. Model.*, vol. 18, no. 11, pp. 1–16, 1993.

[24] D. M. Witten and R. Tibshirani, "A framework for feature selection in clustering," *J. Amer. Stat. Assoc.*, vol. 105, no. 490, pp. 713–726, 2010.

[25] X. Wang, Y. Wang, and L. Wang, "Improving fuzzy *c*-means clustering based on feature-weight learning," *Pattern Recognit. Lett.*, vol. 25, no. 10, pp. 1123–1132, 2004.

[26] H. Frigui and O. Nasraoui, "Unsupervised learning of prototypes and attribute weights," *Pattern Recognit.*, vol. 37, no. 3, pp. 567–581, 2004.

[27] D. S. Yeung and X. Z. Wang, "Improving performance of similarity-based clustering by feature weight learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 4, pp. 556–561, Apr. 2002.

[28] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 400–412, Jul. 2015.

[29] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in *Proc. 7th Asia Joint Conf. Inf. Secur.*, Aug. 2012, pp. 62–69.

[30] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2014, pp. 23–26.

[31] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, Nov. 2015.

[32] Z. Wang *et al.*, "Orientation invariant feature embedding and spatial temporal regularization for vehicle re-identification," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 379–387.

[33] M. V. Varsha, P. Vinod, and K. A. Dhanya, "Identification of malicious android APP using manifest and opcode features," *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 2, pp. 125–138, May 2017.

[34] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, "DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1772–1785, Aug. 2017.

[35] A. Westyarian, Y. Rosmansyah, and B. Dabarsyah, "Malware detection on android smartphones using API class and machine learning," in *Proc. 5th Int. Conf. Elect. Eng. Inform.*, Aug. 2015, pp. 294–297.

[36] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun.*, Oct. 2014, pp. 354–358.

[37] B. Kang, S. Y. Yerima, S. Sezer, and K. Mclaughlin, "N-gram opcode analysis for android malware detection," *Intl. J. Cyber. Situational Awareness*, vol. 1, no. 1, pp. 231–254, 2016.

[38] T. Ban, T. Takahashi, S. Guo, D. Inoue, and K. Nakao, "Integration of multi-modal features for android malware detection using linear SVM," in *Proc. 11th Asia Joint Conf. Inf. Secur.*, Aug. 2016, pp. 141–146.

[39] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow APIs and machine learning," *Inf. Softw. Technol.*, vol. 75, pp. 17–25, Jul. 2016.

[40] Z. Aung and W. Zaw, "Permission-based android malware detection," *Int. J. Sci. Technol. Res.*, vol. 2, no. 3, pp. 228–234, 2013.

[41] C. Y. Huang, Y. T. Tsai, and C. H. Hsu, "Performance evaluation on permission-based detection for android malware," *Smart Innov., Syst. Technol.*, vol. 2, pp. 111–120, Feb. 2013.

[42] W. Peng, L. Huang, J. Jia, and E. Ingram, "Enhancing the naive bayes spam filter through intelligent text modification detection," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Aug. 2018, pp. 849–854.

[43] Y. Li, Y. Li, H. Yan, and J. Liu, "Deep joint discriminative learning for vehicle re-identification and retrieval," in *Proc. Int. Conf. Image Process.*, Sep. 2017, pp. 395–399.

[44] U. Pehlivan, N. Baltaci, C. Acartürk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based Android malware detection," in *Proc. SSCI*, Dec. 2014, pp. 1–8.

[45] W.-C. Wu and S.-H. Hung, "DroidDolphin: A dynamic android malware detection framework using big data and machine learning," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Oct. 2014, pp. 247–252.

[46] Y. Aafer, W. Du, and H. Yin, *DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android*. New York, NY, USA: Springer, 2014.

[47] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proc. Black Hat, Abu Dhabi*, 2011, pp. 77–101.

[48] H. Y. Chuang and S. D. Wang, "Machine learning based hybrid behavior models for android malware analysis," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, Aug. 2015, pp. 201–206.

[49] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on android," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1252–1264, Jun. 2016.

[50] S. Hou, A. Saas, L. Chen, Y. Ye, and T. Bourlai, "Deep neural networks for automatic android malware detection," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Aug. 2017, pp. 803–810.

[51] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying android malware using deep learning," in *Proc. IEEE 37th Sarnoff Symp.*, Sep. 2016, pp. 160–165.

[52] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: An android malware detection system using deep belief network based on API call blocks," in *Proc. Int. Conf. Web-Age Inf. Manage.*, Cham, Switzerland: Springer, 2016, pp. 54–66.

[53] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based android malware detection," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2017, pp. 60–65.

[54] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data," in *Proc. IEEE Symp. Comput. Commun.*, Jul. 2017, pp. 438–443.

[55] N. McLaughlin *et al.*, "Deep android malware detection," *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, Jul. 2017, pp. 301–308.

[56] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, May 2017, pp. 1871–1878.

[57] W. Li, Z. Wang, J. Cai, and S. Cheng, "An android malware detection approach using weight-adjusted deep learning," in *Proc. Int. Conf. Comput., Netw. Commun.*, Mar. 2018, pp. 437–441.

[58] Y. Zhang, Y. Yang, and X. Wang, "A novel android malware detection approach based on convolutional neural network," in *Proc. 2nd Int. Conf. Cryptogr., Secur. Privacy*, New York, New York, USA, 2018, pp. 144–149.

[59] L. Shiqi, T. Shengwei, Y. Long, Y. Jiong, and S. Hua, "Android malicious code classification using deep belief network," *KSII Trans. Internet Inf. Syst.*, vol. 12, no. 1, pp. 454–475, 2018.

[60] P. Wang and B. Li, "Vehicle re-identification based on coupled dictionary learning," in *Proc. 2nd Int. Conf. Robot. Automat. Sci. (ICRAS)*, Jun. 2018, pp. 1–5.

[61] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: Multi-layer android malware detection system applying deep neural networks," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2018, pp. 473–487.

[62] D. Li, Z. Wang, and Y. Xue, "Fine-grained android malware detection based on deep learning," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–2.

[63] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An efficient SVM-based malware detection framework for android," *Commun. Comput. Inf. Sci.*, vol. 243, pp. 158–166, Sep. 2011.

[64] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying android malware using dynamically obtained features," *J. Comput. Virol. Hacking Techn.*, vol. 11, no. 1, pp. 9–17, Feb. 2015.

[65] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Proc. 7th Int. Conf. Comput. Intell. Secur.*, Dec. 2011, pp. 1–6.

[66] Y. J. Ham and H.-W. Lee, "Detection of malicious android mobile applications based on aggregated system call events," *Int. J. Comput. Commun. Eng.*, vol. 3, no. 2, p. 149, 2014.

[67] Y. J. Ham, D. Moon, H. W. Lee, J. D. Lim, and J. N. Kim, "Android mobile application system call event pattern analysis for determination of malicious attack," *Int. J. Secur. Its Appl.*, vol. 8, no. 1, pp. 231–246, 2014.

[68] S. Malik and K. Khatter, "System call analysis of android malware families," *Indian J. Sci. Technol.*, vol. 9, p. 21, Jun. 2016.

[69] F. Tchakounté, and P. Dayang, "System calls analysis of malwares on android," *Int. J. Sci. Technol.*, vol. 2, no. 9, pp. 669–674, 2013.

[70] S. Huda, R. Islam, J. Abawajy, J. Yearwood, M. M. Hassan, and G. Fortino, "A hybrid-multi filter-wrapper framework to identify run-time behaviour for fast malware detection," *Future Gener. Comput. Syst.*, vol. 83, pp. 193–207, Jun. 2018.

[71] A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, "Extinguishing ransomware—A hybrid approach to android ransomware detection," in *Proc. Int. Symp. Found. Practice Secur.*, 2018, pp. 242–258.

[72] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in *Proc. IEEE Int. Conf. Consum. Electron.*, Jan. 2016, pp. 155–156.

[73] D. Kim, J. KIm, and S. Kim, "A malicious application detection framework using automatic feature extraction tool on android market," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, 2013, pp. 1–4.

[74] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2018.

[75] B. Thomas, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proc. 5th IEEE Int. Conf. Malicious Unwanted Softw.*, Oct. 2010, pp. 55–62.

[76] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious Apps in official and alternative android markets," in *Proc. NDSS*, Feb. 2012, pp. 50–52.

[77] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1960–1700, 2014.

[78] K. Patel and B. Buddadev, "Detection and mitigation of android malware through hybrid approach," in *Proc. Int. Symp. Secur. Comput. Commun.*, 2015, pp. 455–463.

[79] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Comput. Sci.*, vol. 46, pp. 804–8111, Aug. 2015.

[80] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.

[81] A. Rodriguez-Mota, P. J. Escamilla-Ambrosio, S. Morales-Ortega, M. Salinas-Rosales, and E. Aguirre-Anaya, "Towards a 2-hybrid Android malware detection test framework," in *Proc. Int. Conf. Electron., Commun. Comput.*, Feb. 2016, pp. 54–61s.

[82] J. W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," *Comput. Secur.*, vol. 58, pp. 125–138, May 2016.

[83] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digital Invest.*, vol. 13, pp. 1–14, Jun. 2015.

[84] M. Sun, X. Li, J. C. Lui, R. T. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for android," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1103–1112, May 2017.

[85] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, 2017.

[86] W. Yu, L. Ge, G. Xu, and X. Fu, *Towards Neural Network Based Malware Detection on Android Mobile Devices*. Cham, Switzerland: Springer, 2014, pp. 99–117.

[87] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications," in *Proc. 8th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Nov. 2016, pp. 1–5.

[88] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual API dependency graphs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 1105–1116.

[89] K. Chen *et al.*, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-play scale," in *Proc. 24th {USENIX} Secur. Symp.*, 2015, pp. 659–674.

[90] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android anti-malware against transformation attacks," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, 2013, p. 329.

RAJESH KUMAR was born in Sindh, Pakistan, in 1991. He received the B.S. and M.S. degrees in computer science from the University of Sindh, Jamshoro, Pakistan. He is currently pursuing the Ph.D. degree in computer science and engineering from the University of Electronic Science and Technology of China (UESTC). His research interests include machine learning, deep learning, malware detection, the Internet of Things (IoT), RFID, and blockchain technology.

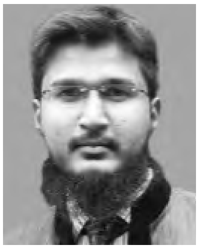
XIAOSONG ZHANG was born in Sichuan, China, in 1968. He received the M.S. and Ph.D. degrees from the University of Electronic Science and Technology of China, in 1999 and 2011, respectively, where he has been a Professor in information security, since 2011. His areas of interest include cryptography, dynamic program analysis, and information security. As of 2017, he has published more than 60 academic papers in the field of network and information security, including CCF Class A top-level journal articles such as IT, TSE, and TIFS. He has published Network Security Protocol, Malware Analysis and Testing, Software testing, and other monographs, textbooks, and translations. He held 28 authorized international and domestic invention patents. A ten of software were registered as copyright. He has received the Best Achievement Awards, in 2012, 2013, 2014, 2015, 2016, and 2017.

WENYONG WANG was born in 1967. He received the B.E. degree in computer science from the Beijing University of Aeronautics and Astronautics, Beijing, China, and the M.E. degree in computer science and the Ph.D. degree in communications engineering from UESTC, where he is currently a Professor of computer science. His research interests include computer network architecture, software-defined networking, and the Internet of Things (IoT).

RIAZ ULLAH KHAN was born in Malakand, KPK, Pakistan, in 1980. He received the bachelor's degree from the University of Peshawar, Pakistan, in 2003, the M.Sc. degree from Hazara University, Pakistan, in 2006, the Diploma degree in information technology from the Metro College of Management Studies, in 2008, and the M.S. Degree from the University of Sunderland, U.K., in 2010. He is currently pursuing the Ph.D. degree with the University of Electronic Science and Technology of China. He is also pursuing the Ph.D. degree with the Lab of Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China.

Since 2010, he has been with in various industrial and educational institutions. From 2010 to 2012, he was appointed as an IT Analyst in a reputable organization in U.K. (IU consultant, U.K.). From 2012 to 2016, he was a Lecturer in computer science with the Abdul Wali Khan University Mardan, Pakistan. In addition, he has published over 16 articles in various international journals and conference proceedings. His research interests include cyber security, big data, dynamic program analysis, cloud computing, data management, distributed systems, blockchain, and the Internet of Things.

JAY KUMAR received the master's degree in computer science from Quaid-i-Azam University, Pakistan. He is currently pursuing the Ph.D. degree in computer science and engineering with the University of Electronic Science and Technology of China (UESTC). His research interests include text classification and subspace clustering in stream mining applied to various domains.



ABUBAKAR SHARIF received the M.Sc. degree in electrical engineering from the University of Engineering and Technology at Lahore, Lahore, Pakistan. He is currently pursuing the Ph.D. degree in electronics engineering with the School of Electronic Science and Engineering, University of Electronic Science and Technology of China (UESTC). From 2011 to 2016, he was a Lecturer with Government College University Faisalabad (GCUF), Pakistan. His research interests include wearable and flexible sensors, compact antenna design, antenna interaction with the human body, antenna and system design for RFID, passive wireless sensing, machine learning, blockchain technology, and the Internet of Things (IoTs).