Android Malware Detection Techniques

Rajesh Kumar and Mamoun Alazab

June 30, 2020

Android Malware Detection Techniques

Rajesh Kumar, Mamoun Alazab

***Abstract*—The revolution of smart devices such as smartphones, smart washing machines, smart cars is increasing every year, as these devices are provided connected with the network and provide the online functionality and services available with the lowest cost. In this context, the Android operating system (OS) is very popular due to its openness. It has major stakeholder in the smart devices but has also become an attractive target for cyber-criminals. In this chapter, we present some current methods and results in the research area of Android malware detection and analysis of Android malware. We begin by briefly describing the background of the static, dynamic and hybrid analysis of the Android malware detection techniques which provides a general view of the analysis and detection process to the reader. After that, the most popular framework to detect malware is discussed. Then, the most popular and basic algorithm and techniques are discussed which is mostly an analysis of malware. Finally, some conclusions about Android malware detection techniques.**

I. Introduction

With the growth of smartphone and the services they provide such as online shopping, health monitoring system, money transaction and many more. The android has largest global market in the world. The frequent use of mobile devices with that facilities encourage people to store and share their personal and critical information through using mobile devices, and the wide use of devices with Android system make Android-based mobile devices a target for malicious application developers [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. Therefore, malicious activity can affect the working of many devices connected in a network. Malware is a program or a set of programs that can cause harm to financial forgery, identity, sensitive information or data, and resources.These malicious applications may leak user's private information without their knowledge or consent.

Personal data leakage: People are not concerned with the security of data or personal information in mobile devices while they are normally very concerned for the same in PC environments [11]. Some apps steal personal information and at the same time demand payments. Such Trojan apps have been downloaded 9,252 times and 211 affected users paid a total of $250,000 to the malware developers [12]. Malware developers successfully stole personal data such as contacts, emails, SMS, and device information which can be used in identity theft and spamming [12].

Social: GPS location, call log, and contact lists can be captured by malware [12]. The contact list and location are user-sensitive information. This information can be captured by malware and can do harm by leaking social identity that can be used in various ways to threaten the security of a user's social image.

Business: Business organizations have their own apps to run their business. Malware can capture user information or business data which will put the business organization at a risk. The business owner will be at a risk of financial loss as well as reputation

Financial loss: The motive of malware development has changed and now focuses on financial gain [13]. Capital expenses related to malware average $6–7bn dollars in a fiscal year [13]. "Zeus in the Mobile" is a Trojan that captures the authentication code of the user in a banking application, which may cause financial losses to the user. It is also expensive to remove, where a security firm charged $21/s for the first detection in 2010 [11]. This type of malware can cause user financial losses as well as large financial losses to a business owner in detection fees. In some cases, a user may have to pay large phone bills for premium rate services because of the malicious activity of an app [12].

Every day has various new applications in the market. It is assessed that there will be roughly 6.1 billion smartphone clients by 2020 [14], [15]. Google, the manufacturers of the Free Phone Alliance, and the open source community of Android developers have made great efforts to enhance security for Android. However, a major concern tends to be the proliferation and development of emerging security threats. Hence, in this context, we discuss the static, dynamic and hybrid analysis detection Android malware features extraction techniques.After that, the most popular framework to detect malware is discussed. Then, the most popular and basic algorithm and techniques are discussed which is mostly an analysis of malware. Finally, some conclusions about Android malware detection techniques. Additionally, this chapter identify many elements of security threats involved in using mobile phones and applications, and the user will feel confident in using these applications.

II. Static, Dynamic and Hybrid Analysis of Android Malware Background

In this chapter we discuss the background of the Android malware detection techniques. There are three basic techniques to detect the android malware. i) Static analysis, ii) Dynamic analysis, iii) Hybrid analysis

A. Static Analysis

The static analysis method refers to analyzing source code files or executable files without running applications. There are several features such as API call and permissions to analysis the static analysis. The feature extraction methods are shown in Figure 1.

Furthermore, some static features detection methods are shown in Table II. The k-nearest neighbours machine learning classifier achieves better performance and accuracy in the detection of the malware. However it takes more processing

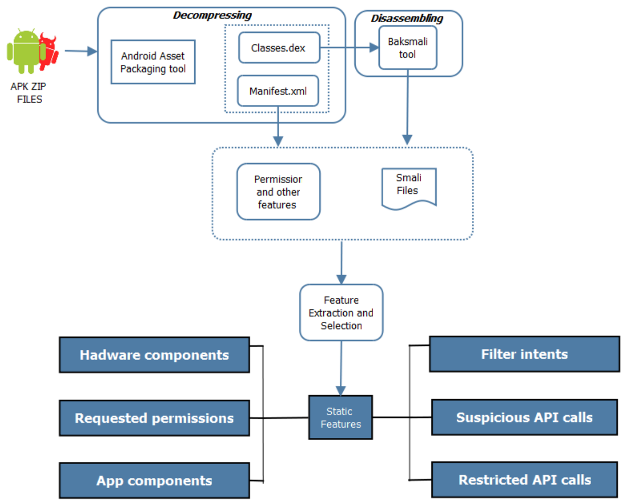

Figure 1. Static Feature Extraction of method

Table I
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

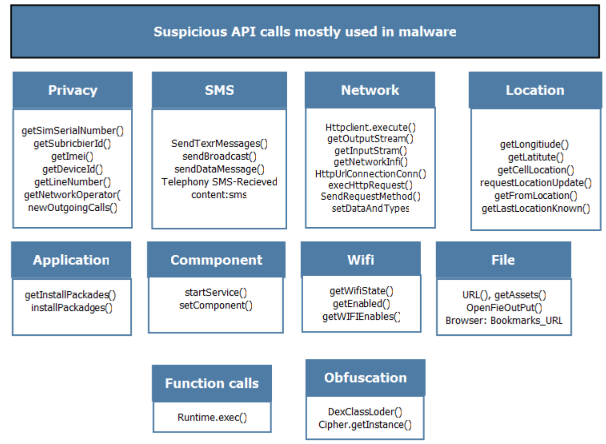

Figure 2. Suspicious API calls

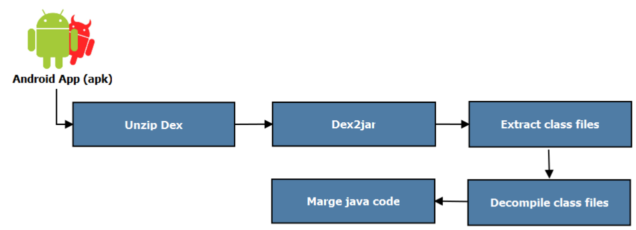

Figure 3. Workflow of Android file decompiling

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 Random Forest algorithm for Android malware detection.

1) Permission-based analysis: Permission-based access control mechanism is a major component of the Android platform security mechanism. On the Android platform, applications are separated from applications, and applications and systems are isolated. When applications perform certain operations or access certain data, they must apply for corresponding permissions. This means that permissions defined in the manifest file can indicate the behavior of the application. Developers can declare the permissions that need to be applied in the <uses-permission> tag or <permission> tag. The permissions in the <usespermission> tag are predefined by android, and the permissions in the <permission> tag are customized by the developer and belong to third-party permissions. According to Android's official documentation, the level of protection of permissions implies the potential risks involved, and points out the verification process that should be followed when the system decides whether to grant application permissions. The four protection levels are described as follows: Normal defines the low-risk permissions to access the system or other applications, which does not require user confirmation and is automatically authorized. Dangerous can access user data or control the device in some form, such as READ_SMS (allowing applications to read SMS). When granting such permissions, the system will pop up a confirmation dialog box and display the permission information requested by the application. The user can choose to agree or cancel the installation. Signature is the most severe permission level and requires an encryption key. It only grants applications that use the same certificate as the declared permissions. Therefore Signature usually only appears in applications that perform device management tasks, such as ACCESS_ALL_- EXTERNAL_STORAGE (access to external storage). SignatureOrSystem can be granted either partial applications of the system image or applications with the same signature key as the declaration permission.

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 [45]. To differentiate 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 2 shows the some of suspicious API calls,which are mostly used by malware applictions. Figure 3 shows the extracted the features from the APK file that contains the classes.dex file.

B. Dynamic Analysis

The dynamic analysis method is not affected by code transformation technologies such as bytecode encryption, reflection, and native code execution, and can deeply analyze the malicious behaviors of the application. Therefore, it makes

Table II
STATIC FEATURES DETECTION METHODS

Ref	Features	Accuracy	Machine Learning Models	Contribution	Limitation
[16]	Permission	91.75%	Random Forest	Permission based approach using KNN clustering	Risky permission not founded
[17]	Permission	81%	C4.5, SVM	The framework quick identify the malicious permission	It uses the limited number of malware. it require the evidence
[18]	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
[19]	Permission	-	AHP	a global threat score deriving set of permissions required by the app	Only depends on permissions with known limitations- susceptible to attack
[20]	Permission	98.6	J48	Build a framework for based on SIGPID. It extracts top 22 permissions.	Susceptible to impersonate attack
[21]	Permission	92.79%	Random Forest	Design a model which score the malicious permission	Susceptible to adversarial attack
[22]	Permission	94.90%	Random Forest	It uses the classification algorithm to detect the malware.	Susceptible to adversarial attack
[23]	Permission, API calls	92.36%	Random Forest		Susceptible to adversarial attack
[24]	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
[25]	API call	99%	k-nearest neighbors	It mitigate Android malware installation through providing lightweight classifiers	Susceptible to impersonate attack
[26]	API call	93.04%	Signature matching	It measure the similarity of malware	Susceptible to impersonate attack
[27]	API call	96.69%	SVM	The paper use malicious-preferred features and normal-preferred features for the detection of malware	Susceptible to impersonate attack
[28]	ICC related features	97.40%	SVM	Design a ICCDetector framework which classify the malware based on android intent filters	Susceptible to impersonate attack
[29]	Permission, command, API calls	98.60%	Parallel classifier	This paper combine the machine learning classifiers to classify the malware.	Susceptible to impersonate attack
[30]	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
[31]	Risky Permis-sionsdangerous 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
[32]	API call blocks	ACC 96.66%	DBN	DroidDelver Detection system is used to identify malware using an API call block.	Susceptible to adversarial attack
[33]	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
[34]	323 features	F1 95.05	DBN	An identification system designed by FlowDroid uses data flow analysis to identify malware.	Susceptible to adversarial attack
[35]	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
[36]	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
[30]	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
[37]	Dangerous API calls-risky permissions	Recall 94.28	DBN	DBN was used to create an automatic malware classifier	Susceptible to adversarial attack
[38]	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

Ref	Features	Accuracy	Machine Learning Models	Contribution	Limitation
[39]	API calls	Acc 95.7	DBN	Suggested approach to image texture analysis for malware detection	Risky permission not founded
[40]	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
[41]	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
[42]	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
[43]	Permissions API Calls	Prec 97.15 Recall 94.18 F1 95.64	DNN	Implemented DNN - based malware detection engine	Susceptible to impersonate attack
[44]	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 III
DYNAMIC FEATURES DETECTION METHODS

Ref	Features	Accuracy	Machine Learning Models
[46]	System call	91.75%	Signature Matching
[45]	System call	81%	K-Means
[47]	System call	88.2%	Frequency
[48]	System call	-	Pattern matching
[49]	API call	97.6	KNN_M
[17]	Native Size	99.9%	RF, SVM

sense to collect dynamic features, which can effectively compensate for the limitations of static analysis. Figure 4 shows the feature extraction method and detection technique of the dynamic analysis. Many machine learning algorithm used for dynamic analysis for instance, Logistic regression (LR) , K-means Clustering, SVM, KNN_E,KNN, Bayesian network (BN), and Naïve Bayes. Table III illustrates the accuracy level, dynamic features and detection methods. For example, some malware may obtain malicious files through the network or other means during the running process, and then write them into the system files to perform malicious behaviors. These means can escape static detection and affect the accuracy of detection. DroidBox is an Android application sandbox that extends TaintDroid. It can perform dynamic stain analysis at the application framework level, and monitor various operations of the application, such as information leakage, network, file input / output, and encryption operations. DroidBox provides two scripts, startemu.sh and droidbox.sh. The former is used to start a simulator dedicated to the dynamic analysis of Android applications, and the latter is used to perform specific dynamic analysis. We obtain the dynamic operation log of each application by installing and running each application in DroidBox for 30s, and extract features from them.

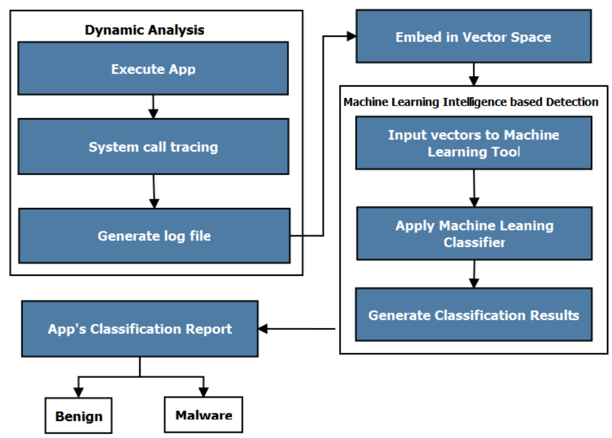

Figure 4. Dynamic Feature Extraction and Detection

C. Hybrid Analysis

To improve the performance of learning algorithms, the hybrid analysis was developed, which utilises the dynamic and static features as shown in Figure fig: Hybrid Analysis. Some researches proposed multi-classification techniques [52], [53] to obtain high accuracy in the hybrid analysis. Furthermore, The static features are Publisher ID, API call, 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 [54], following features are selected form static (permission and

Name	Used in malicious	Used in benign
PTRACE	Most often utilized [50], [47]	Utilized in benign applications [47]
SIGPROCMASK	Most often utilized [50], [47]	Utilized in benign applications [47]
CLOCK	Most often utilized [50], [51]	-
CLOCK-GETTIME	Utilized in malicious applications [47]	Utilized in benign applications [47]
RECV	Most often utilized [51], [47]	Not Utilized [47]
RECVFROM	Most often utilized [48], [50], [51]	Not Utilized [47]
WRITE	Most often utilized [48], [50], [51]	Utilized in benign applications [47]
WRITEV	Most often utilized [51], [47]	Utilized in benign applications [47]
WAIT4	Most often utilized [50]	
SEND	Most often utilized [51]	
SENDTO	Most often utilized [50], [51]	
MPROJECT	Most often utilized [48], [50], [51]	Utilized in benign applications [47]
FUTEX	Most often utilized [50], [47]	Utilized in benign applications [47]
IOCTL	Most often utilized [50], [47]	Utilized in benign applications [47]
FCNTL64	Most often utilized [47]	Utilized in benign applications [47]
GETPID	Most often utilized [50], [47]	Utilized in benign applications [47]
GETUID32	Most often utilized [50], [47]	Utilized in benign applications [47]
EPOLL	Most often utilized [47]	Utilized in benign applications [47]
EPOLL-CTL	Most often utilized [47]	Utilized in benign applications [47]
EPOLL-WAIT	Most often utilized [51], [48]	Utilized in benign applications [47]
CACHEFLUS	-	-
READ	Most often utilized [50], [51]	Utilized in benign applications [47]
READV	Most often utilized [51]	-
STAT64	-	-
GETTIMEEOFDAY	utilized in malicious applications [47]	Utilized in benign applications [47]
ACCESS	Most often utilized [51], [48]	Utilized in benign applications [47]
PREAD	-	-
UMASK	Most often utilized [47]	Not Utilized [47]
CLOSE	utilized in malicious applications [47]	Utilized in benign applications [47]
OPEN	Most often utilized [51], [47]	Utilized in benign applications [47]
MMAP2	utilized in malicious applications [47]	Utilized in benign applications [47]
MUNMAP	-	-
MADVISE	utilized in malicious applications [47]	Utilized in benign applications [47]
FCHOWN32	Most often utilized [47]	Not Utilized[47]
PRCTL	Not Utilized [47]	Utilized in benign applications [47]
BRK	Most often utilized [47]	Not Utilized[47]
LSEEK	Utilized in malicious applications [47]	Utilized in benign applications [47]
DUP	Utilized in malicious applications [47]	Utilized in benign applications [47]
GETPRIORTY	Utilized in malicious applications [47]	Utilized in benign applications [47]
PIPE		
CLONE	Utilized in malicious applications [47]	Utilized in benign applications [47]
FSYNC	Most often utilized in [47]	Not Utilized[47]
GETDENTS64	Utilized in malicious applications [47]	Utilized in benign applications [47]
GETTID	Utilized in malicious applications [47]	Utilized in benign applications [47]
LSTA64	Utilized in malicious applications [47]	Utilized in benign applications [47]
FORK	-	-
NANOSLEEP	Not Utilized [47]	Only Utilized in benign applications [47]
RECVMSG	-	-
CHMOD	Utilized in malicious applications [47]	Utilized in benign applications [47]
SENDMSG	Most widely Utilized[50]	-
FLOCK	Not Utilized [47]	Only Utilized in benign applications [47]
MKDIR	Most often utilized [47]	Not Utilized [47]
CONNECT	Most often utilized [47]	Not Utilized [47]
POLL	Not Utilized [47]	Only Utilized in benign applications [47]
RENAME	Most widely Utilized [51]	Not Utilized [47]
SETPRIORITY	-	-
SETSOCKOPT	Most often utilized [47]	Not Utilized [47]
SOCKET	Most often utilized [47]	Not Utilized [47]
UNLINK	-	-
GETSOCKOPT	-	-
BIND	Most often utilized [47]	Not Utilized[47]
FTRUNCATE	Utilized in malicious applications [47]	Utilized in benign applications [47]
GETSOCKNAME	-	
INOTIFY	-	
RESTART	-	
SCHED	Utilized in malicious applications [47]	Utilized in benign applications [47]
GETRLMIT	-	
LGETXATTR	-	
READLINK	-	
SOCKETPAIR	-	
SATAFS64	Utilized [47]	Not Utilized[47]
FDATSYNC	Utilized [47]	Not Utilized[47]
GETPPID	-	
KILL	-	
PWRITE	Utilized [47]	Not Utilized[47]

Name	Used in malicious	Used in benign
MSGGET	Used [47]	Not Utilized[47]
RMDIR	Used [47]	Not Utilized[47]
SELECT	Used [47]	Not Utilized[47]
SEMGET	Used [47]	Not Utilized[47]
SEMOP	Used [47]	Not Utilized[47]
CHDIR	Not Utilized [47]	Only Utilized in benign applications [47]
GETCWD	Not Utilized [47]	Only Utilized in benign applications [47]
RT_SIGRETURN	Utilized [47]	Only Utilized in benign applications [47]
SIGACTION	Not Utilized [47]	Only Utilized in benign applications [47]
SYS_281	Not Utilized [47]	Only Utilized in benign applications [47]
SYS_283	Not Utilized [47]	Only Utilized in benign applications [47]
SYS_224	Utilized [47]	Not Utilized[47]
SYS_248	Utilized [47]	Not Utilized[47]
SYS_290	utilized in malicious applications [47]	Utilized in benign applications [47]
SYS_292	utilized in malicious applications [47]	Utilized in benign applications [47]
SYSCALL_903042	utilized in malicious applications [47]	Utilized in benign applications [47]
FSTAT64	utilized in malicious applications [47]	Utilized in benign applications [47]

APICall) and dynamic (SystemCall). Y. Liu, et al. [54] used the SVM and Navie Bayes machine learning classifier. The SVM classifier used for static analysis achieved 93.33 to 99.28 percent accuracy,while the Naive Bayes used for dynamic analysis achieved accuracy up to 90 percent. Furthermore, Kim et al. [55], used the J48 machine learning classier, the features are selected from static (permission) and dynamic (APICal l) . A. Saracino el al. [56], achieved 96.9 % accuracy based on KNN by selecting the static feature (permission) and dynamic (critical API, SMS, User activity System call) feature.

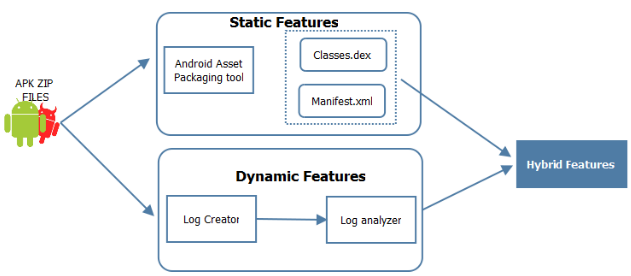

Figure 5. Dynamic Feature Extraction and Detection

D. A Comparison of Static, Dynamic, and Hybrid Analysis

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) Multiple categories of Features: The advantages of multiple category features are easy to extract, and high accuracy. The limitations associated with this method are Mimicry attack, high computation, code obfuscation, and difficult to handle multiple features

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) Multiple categories of 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.

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.

Table IV
HYBRID ANALYSIS METHODS

Ref	Methodology	Tools	Achements	Limitations
[57]	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
[58]	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
[59]	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 Moni-tor 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.
[60]	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 Pack-aging 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.
[61]	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 Pack-aging Tool.	Achieves 96.4% detection rate	High time for scanning. High electricity consumption. High consumption of resources / storage.
[62]	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
[63]	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 , Dropid-BOX 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
[54]	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.

Ref	Methodology	Tools	Achements	Limitations
[56]	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.
[64]	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	
[65]	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
[66]	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
[67]	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.

III. Deep Learning Based Android Malware Detection Schemes

A. Basic proposed framework to detect android malware

In this section, We discusses the methodology to detect the malicious codes detection techniques based on deep learning and machine learning. Kim et al. [1] proposed an multi-model malware detection based malware analysis system to automatically analyze and classify malware behaviors. Figure 6 shows the overall architecture of the developed framework. The multimodal deep learning framework uses seven kinds of the feature; String feature, method opcode feature, method API feature, shared library function opcode feature, permission feature, component feature, and environmental feature. Using those features, the seven corresponding feature vectors are generated first, and then, among them, the permission/component/predefined setting feature vectors are merged into one feature vector. Finally, the five feature vectors are fed to the classification model for malware detection.

Moreover, Tao Lei et al. [68] proposed an Graph based malware detection model based on three components: 1) call graph extraction; 2) event group building; and 3) NN training. These three phases shown in 7. In call graph phase it extract the call graphs of every sample from the training samples by using the static analysis tools and then filter out repetitive API calls. The event group building component aims to build the event group for apps, which consists of event subgraph traverse, API calls encoding and clustering. Finally, the event group (clustering result) are fed into the NN to train the parameters.

Andrea Saracino et al. [56] detect malicious behavioral-patterns extracted from several categories of malware.The features at the four system levels, and to detect and prevent a misbehavior. It consist of 4 steps shown in Figure 9. The first one is the App Risk Assessment, which includes the App Evaluator that implements an analysis of metadata of an app package (apk) (permission and market data), before the app is installed on the device. The second block is the Global Monitor, which monitors the device and OS features at three levels, i.e., kernel (SysCall Monitor), user (User Activity Monitor) and application (Message Monitor). The third block is the Per-App Monitor, which implements a set of known behavioral patterns to monitor the actions performed by the set of suspicious apps (App Suspicious List), generated by the App Risk Assessment, through the Signature-Based Detector

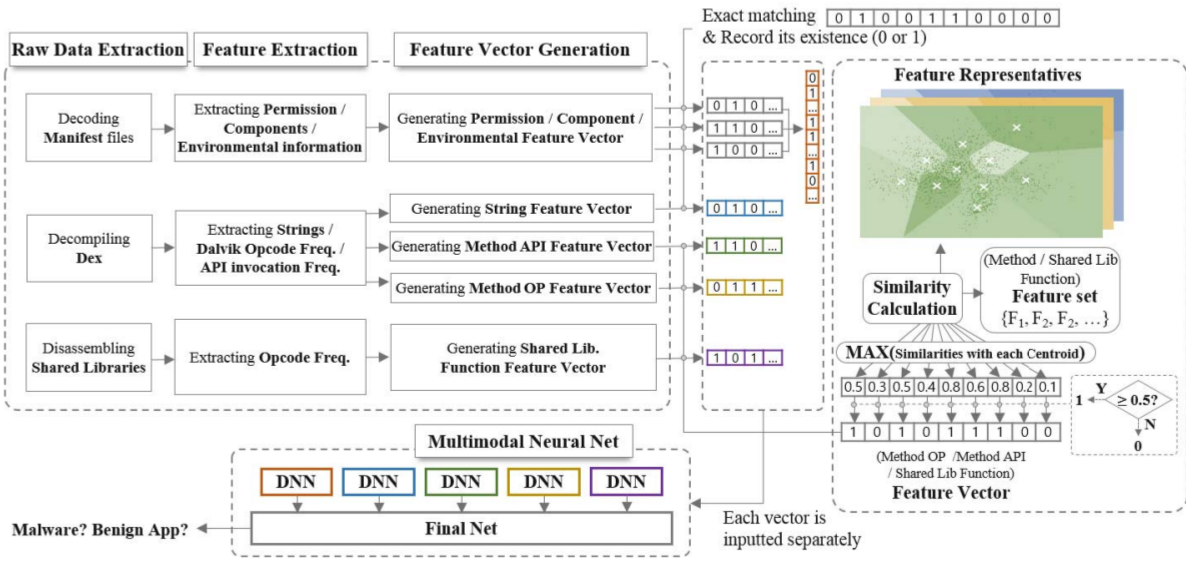

Figure 6. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features [1].

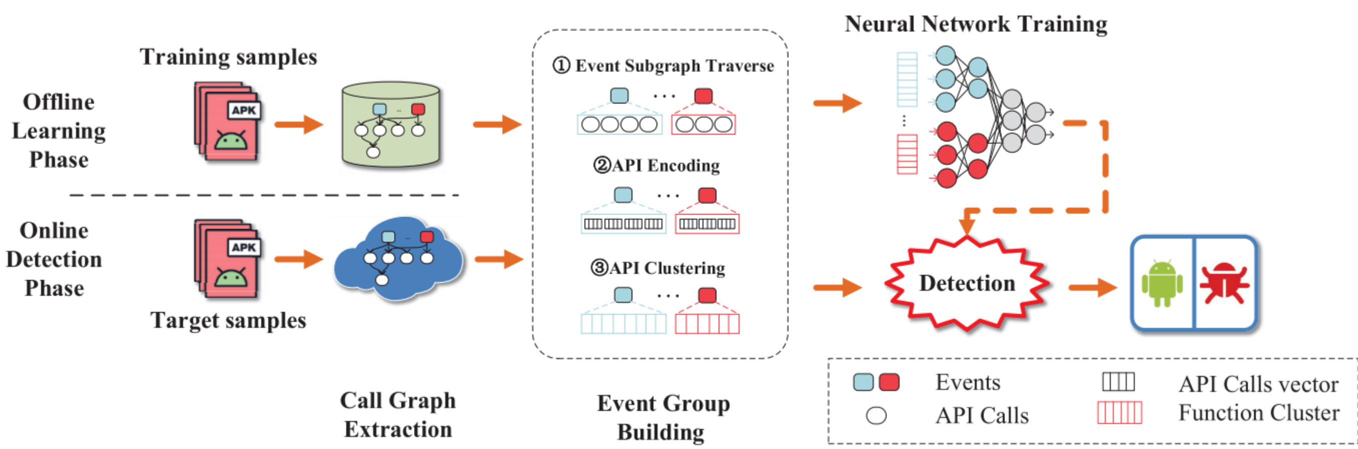

Figure 7. EveDroid: Event-Aware Android Malware Detection Against Model Degrading for IoT Devices [68].

Huijuan Zhu et al. [70] raises a stacking ensemble framework SEDMDroid to identify Android malware. Specifically, to ensure individual's diversity, it adopts random feature subspaces and bootstrapping samples techniques to generate subset, and runs Principal Component Analysis (PCA) on each subset. The accuracy is probed by keeping all the principal components and using the whole dataset to train each base learner Multi-Layer Perception (MLP). Then, Support Vector Machine (SVM) is employed as the fusion classifier to learn the implicit supplementary information from the output of the ensemble members and yield the final prediction result. The Figure 9 shows the overall proposed framework of the SEDMDroid.

Ming Fan et al. [71] in first step , the apk file is given as the input, whose classes.dex file is converted into .smali files (an interpreted language that syntactically approaches pure source codes) with apktool. By scanning the .smali files, the possible functions and the calling relations between them can be obtained. Thus, StaticFunction-CallGraph (SFCG) can be constructed in manner in which nodes denote the functions and edges denote the calls. Second, two key steps are performed: measuring the sensitivity coefficient of each sensitive API and mining the Sensitive Subgraph (SSG shown in Figure 11) in the generated SGS. Lastly, five features of SSG are constructed and fed into machine learning algorithms to detect whether the app is piggybacked or benign shown in Figure 10.

Jin Li, et al. [69] propose the malware detection framework based on static analysis for permission feature. The proposed framework consist three technique to collect the risky permissions. i) Permission Ranking With Negative Rate ii)Support-Based Permission Ranking iii) Permission Mining With Association Rules. It extracts significant permissions from apps and uses the extracted information to effectively detect malware using supervised learning algorithm.

Kumar et al. [2] propose the malware detection framework which is based on three techniques, i) Clustering Algorithm

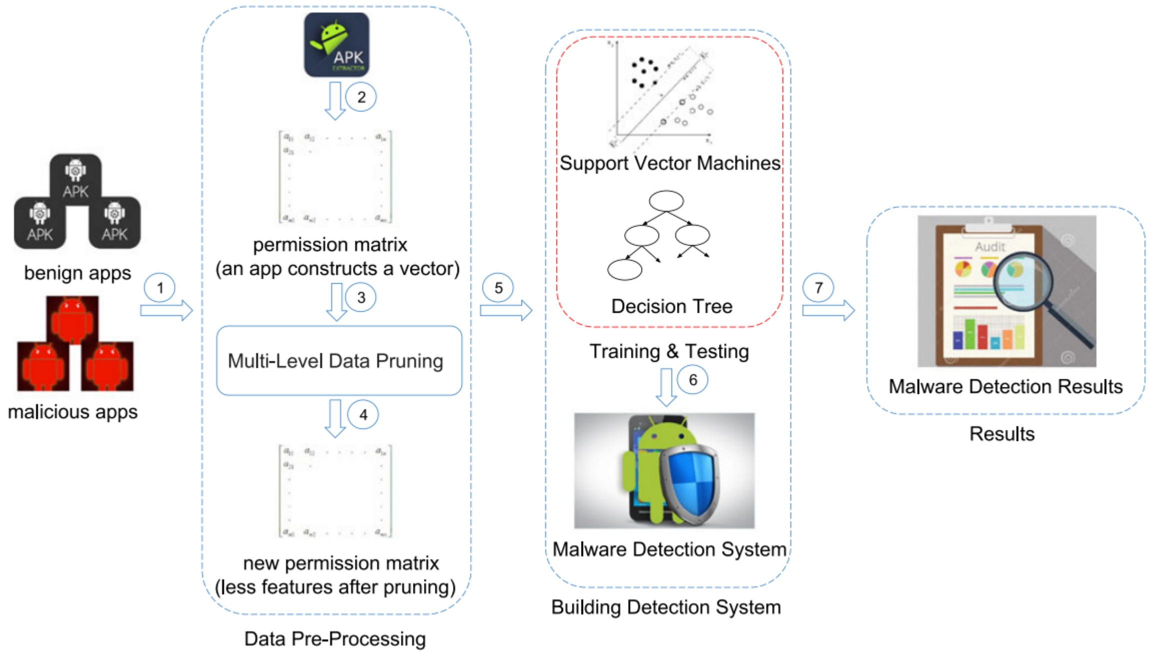

Figure 8. Significant Permission Identification for Machine-Learning-Based Android Malware Detection [69].

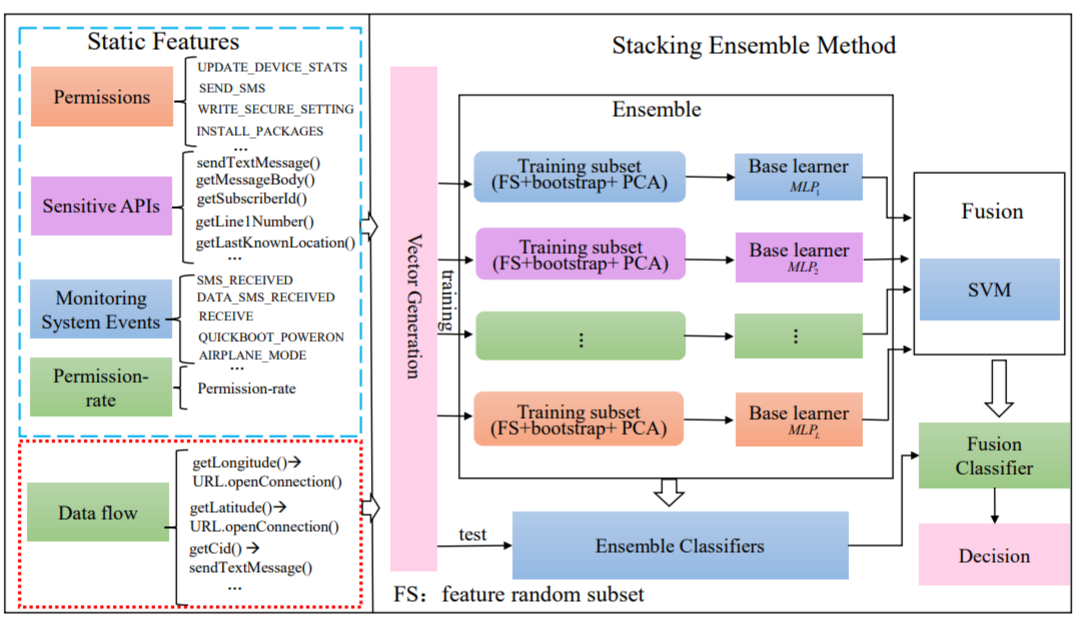

Figure 9. SEDMDroid: An enhanced stacking ensemble framework for Android malware detection[70]

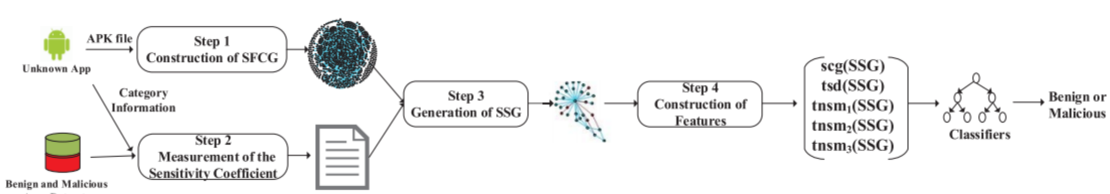

Figure 10. DAPASA: Detecting Android Piggybacked Apps Through Sensitive Subgraph Analysis [71].

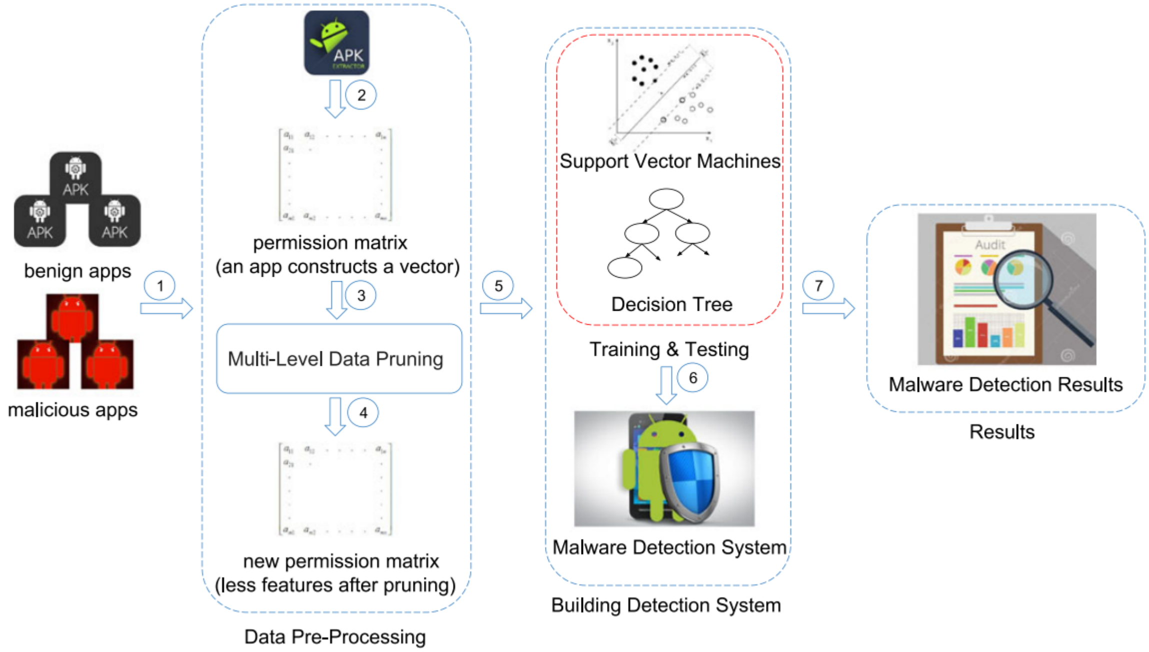

Figure 12. Significant Permission Identification for Machine-Learning-Based Android Malware Detection[69]

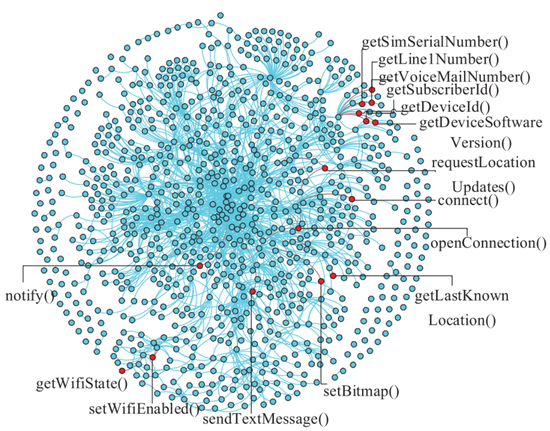

Figure 11. DAPASA: SSG Graph [71].

ii) Naive Bayes Classifier for Multi-Feature iii) Blockchain based malware detection framework. Overall architecture of the proposed system shown in Figure 13. A new blockchain-based framework was presented to evaluate the performance of malware detection. The newly proposed machine learning technique provides an efficient approach to train the model and then stores and exchanges the trained model results throughout the blockchain network for spreading the information of newly generated malware.

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 naïve Bayes for multi-feature classification. Finally, a blockchain database store the malware information.

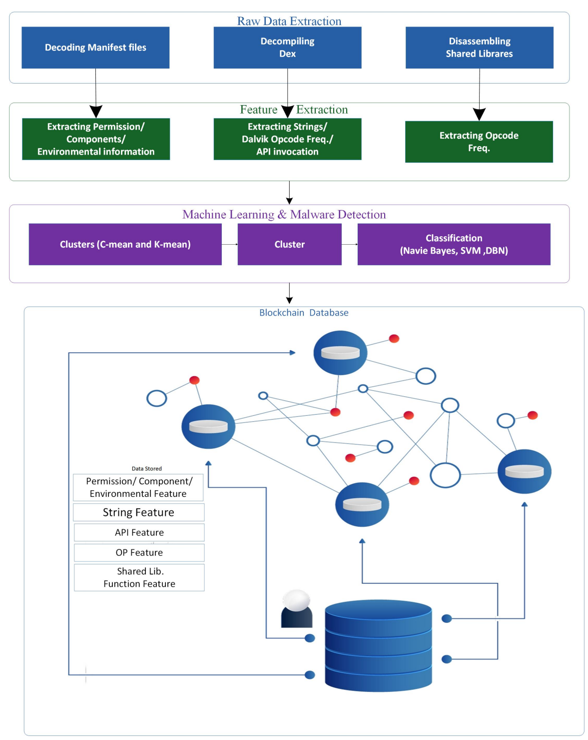

Figure 13. A multimodal malware detection technique for Android IoT devices using various features [3].

B. *Basic proposed algorithms for android malware features*

This section discuss the basic algorithms and techniques which is used commonly.

1) Clustering Techniques to classify the malware : The centroids of the clusters which are calculated using the basic

K-Means Algorithm

1. Select k centroids arbitrarily (called as seed) for each cluster Ci , i ε [1, k]
2. Assign each data point to the cluster whose centroid is closest to the data point.
3. Calculate the centroid Ci of cluster Ci, i ε [1, k].
4. Repeat steps 2 and 3 until no points change between clusters.

1. Select k random instances from the training data subset as the centroids of the clusters C1; C2; ...Ck.
2. For each training instance X:
   a. Compute the Euclidean distance D(Ci,X), i=1... k:Find cluster Cq that is closest to X.
   b. Assign X to Cq. Update the centroid of Cq.(The centroid of a cluster is the arithmetic mean of the instances in the cluster.)
3. Repeat Step 2 until the centroids of clusters C1;C2; ...Ck stabilize in terms of mean-squared- error criterion.
4. For each test instance Z:
   a. Compute the Euclidean distance D(Ci,Z), i=1... k. Find cluster Cr that is closest to Z.
   b. Classify Z as an anomaly or a normal instance using the Threshold rule

The Threshold rule for classifying a test instance Z that belongs to cluster Cr is:
Assign Z-->1 if P(w|Z)> Threshold;
Otherwise Z →0 where "0" and "1" represent normal and malware classes [6]

Figure 14. K-Means Algorithm

```
Input: Feature set in DB, F_db & Feature set of an App, F_app
Output: A existence based feature vector
  feature_vector ← [0 | 0 | . . . | 0]
  index ← 0
  for ∀f1 ∈ F_db do      // for all features in database
    if f1 ∈ F_app then
      feature_vector[index] ← 1
  return feature_vector
```

Figure 15. Existence Based Feature Vector Generation

```
Input: Feature set in DB, F_db & Feature set of an App, F_app
Output: A similarity based feature vector
  Centroids←k_means(k, F_db)    // preprocessing
  feature_vector ← [0 | 0 | . . . | 0]
  index ← 0
  for ∀c ∈ Centroids do      // for all centroids
    min_sim←0
    for ∀f ∈ F_app do      // for all features
      dist ← get_euclidean_dist(c, f)
      sim ← 1/(dist+1)
      if sim < min_sim then
        min_sim ← sim
    if min_sum > threshold then
      feature_vector[index] ← 1
    else
      feature_vector[index] ← 0
    index ← index+1
  return feature_vector
```

Figure 16. Similarity Based Feature Vector Generation

Input: $F = \{f_{ij}\}_{1\le i\le m, 1\le j\le n}$, G: number of clusters desired, Clu a clustering algorithm, $\oplus$ associative and commutative feature combination algorithm;

Cluster the n basic features into G groups accordingly by considering each feature to be a column vector in F;

for *each sample APK* i **do**

  for *each feature group* g **do**

    $f_{ig}^{FC} = \oplus\{i.f \mid f \in g\}$ % combine values of APK i's value of feature f for each f in feature group g;

  $\mathbf{f}_i^{FC} = (f_{i1}^{FC}, \cdots, f_{iG}^{FC})$ % FC feature vector for sample i;

return $F_{FC} = \{\mathbf{f}_i^{FC} \mid 1 \le i \le m\}$ (feature value clustering based G-dimensional feature vectors for m sample APKs);

Figure 17. Feature Value Clustering based Feature Transformation

K-means [72] clustering algorithm shown in Figure 14The process of future generation values in the malicious feature database correspond to the elements of the feature vector, and every feature value is searched in the features extracted from input applications. If there is no certain feature value in the extracted features, its absence is represented as zero. Otherwise, the existence of the feature value is represented as one in the vector. The overall process of future generation shown in Figure 15. Additionally, the similarity-based feature vectors are generated in Figure 16

2) Feature Ranking-Based Algorithms: **Average Accuracy-Based Ranking Scheme:** The ranking is designed to be directly proportional to the average prediction accuracies across the classes.

Let P_{base} be the set of performance accuracies $P_{k,c} \epsilon\ P_{base}$ of K base classifiers. If m denotes malware and b, benign then the average accuracy of the kth base classifier is given by

$$a_k = 0.5 \times \sum_{c=m,b} P_{k,c} | k \in \{1, \ldots, K\}, 0 < P_{k,c} \le 1 \quad (1)$$

.

Let $A \leftarrow a_k, \forall k \in \{1, \ldots, K\}$ be a set of the average predictive accuracies, to which a ranking function $Rank_{\text{desc}}(.)$ is applied

$$\bar{A} \leftarrow Rank_{\text{desc}}(A) \quad (2)$$

Thus,$\bar{A}$ contains an ordered ranking of the level-1 base classifiers average predictive accuracies in descending order. Next, the top Z rankings are utilized in weight assignments as follows

$$\omega_1 = Z, \omega_2 = Z - 1, \ldots, \omega_Z = 1, Z \le K \quad (3)$$

Class Differential-Based Ranking Scheme: let the average accuracy of each base classifier be given by a_k in 1 and define $\bar{D}$ with cardinality K as a set of ordered rankings in descending order of magnitude. Calculate d_k proportional to average accuracies and inversely proportional to absolute difference of interclass accuracies

$$d_k = \frac{a_k}{|P_{k,m} - P_{k,b}|}, k \in \{1, \ldots, K\} \quad (4)$$

$$\bar{D} \leftarrow Rank_{\text{desc}}\,(D) \tag{5}$$

Ranked Aggregate of Per Class Accuracies-Based Scheme: With $\bar{F}$ defined as the set of ordered rankings with cardinality K, given the initial performance accuracies of $P_{p,c}$ of the K base classifiers

$$\begin{cases} P_m \leftarrow P_{k,c} \text{ where } c \neq b \\ P_b \leftarrow P_{k,c} \text{ where } c \neq m \end{cases}, k \in \{1, \ldots, K\}, c \in \{m, b\} \tag{6}$$

$$\begin{cases} \bar{P}_m \leftarrow Rank_{\text{desc}}\,(P_m) \\ \bar{P}_b \leftarrow Rank\,(P_b) \end{cases} \tag{7}$$

$$\begin{cases} f_k \leftarrow \bar{P}_{k,m} + \bar{P}_{k,b} \\ F \leftarrow f_k \end{cases}, \forall k \in \{1, \ldots, K\} \\ \bar{F} \leftarrow Rank_{\text{desc}}\,(F) \tag{8}$$

C. Feature Selection-Based Algorithms

Feature selection is extremely important in static, dynamic and hybrid analysis. The appropriate feature set is selected using different selection methods such as information gain, mutual information,fisher score, similarity function, etc.

Information gain (IG) feature ranking approach to rank the features and then selecting the top n features. IG evaluates the features by calculating the IG achieved by each feature. Specifically, given a feature X, IG is expressed as

$$\text{IG} = E(X) - E(X/Y) \tag{9}$$

where $E(X)$ and $E(X/Y)$ represent the entropy of the feature X before and after observing the feature Y, respectively. The entropy of feature X is given by

$$E(X) = -\sum_{x \in X} p(x) \log_2(p(x)) \tag{10}$$

where $p(x)$ is the marginal probability density function for the random variable X. Similarly, the entropy of X relative to Y

$$E(X/Y) = -\sum_{x \in X} p(x) \sum_{x \in X} p(x \mid y) \log_2(p(x \mid y)) \tag{11}$$

Similarity based feature selection shown in below equation, B represents the benign and M represents the malware. X is the feature list and γ is the similarity between the features.

$$S_B\left(X_j\right) = e_p \sum_{i=1}^{n} \gamma^{S_B}\left(X_j^{sb}\right) \psi\left(X_j^{sb}\right), \left(X_j\right) \in X^{sb} \tag{12}$$

$$S_{score} = S_p + S_j \tag{13}$$

D. Association Rule-Based Algorithms

Association rule mining is used to discover meaningful relationships between variables in huge databases. For example, if events A and B always occur at the same time, then the two events are likely to be associated. for instance, we found that many permission are always together i.e., READ_CONTACTS and WRITE CONTACTS are always used together. These dangerous Android permissions belong to permission Google's list. As we know that those permission always together. So we only need one of them to characterize certain behavior.

STEP1: Find out the frequent two-permissions sets

STEP2: Diversity-based interestingness measures for association rule using frequent two itemsets that was developed by Piatetsky- Shapiro[73]

- When support$(Y \bigcup Z)$ $\approx$ $support(Y)support(Z)$, the two-item sets(Y, Z) are mutually independent. That is, the association rule $Y \Rightarrow Z$ is uninteresting.

$$interest(y,z) = \frac{support(Y \cup Z)}{support(Y \cup Z)} - 1 \\ = \frac{P(Y|Z)}{P(Z)}$$

- if interest(Y, Z) $>$ 0 ,Y and$Z$ are correlated positively.
- if interest(Y, Z) $\approx$ 0 , Y and$Z$ are commonly independent, and the common two- item sets should be rejected.
- if interest(Y, Z) $<$ 0 , Y and Z are negatively correlated.

STEP3: Create the association rule based on the permission

Algorithm 1 Association Rule set R For Permission Based

```
input ← 1 Associaion Rule Set R
minSub ← minimum thershold of support cofficient
minConf ← minimum thershold of confidence cofficient
for Z=D do
    r = null
    r.PushTail(Z)
    for Y in D do
        if Y ⇒ Z ∈ L2 and support(Y ⇒ Z) > minsup and confidence(Y ⇒ Z) > mincof then
            r.PushTail(Y)
        end if
        r.PushTail(r)
    end for
end for
output ← Association Rule R
```

STEP4: Calculate probability table of the association rules.

E. Model 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

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

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.

A. Publicly Available Most Popular Dataset

In order to excavate practical significance, we introduce 10 most popular dataset in Table V. The more description of the dataset are discuss in provides links.

B. Dataset other reasearch work

The comparison the number of benign and malware apps used in previous work shown in Table VI.

Authors	Benign	Malware
[17]	480	124769
[74]	45	300
[28]	5264	12026
[18]	378	324658
[25]	3978	500
[23]	175	621
[22]	1446	2338
[75]	5560	123453
[29]	2925	3938
[26]	238	1500

Table VI
COMPERSION ACCURACY WITH OTHER WORKS

C. Results Discussion

1) Permission-Based Resutls : Among the 145 permission set, 48 permission are risky permissions which are mentioned in previous literature [23], [76], [77] and Table VII. Moreover, Jin Li, et.al, [20], developed a SIGPID framework to detect the risky permission, the author generate top 22 risky permission mentioned in Table VIII. Furthermore Kumar et.al[2], used a data-mining technique to extract the risky permission, based on association rule set of risky permission shown in Table IX.

Table VII
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 VIII
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

2) Cluestring based Results:

3) Classification Results: From the machine learning-based methods to the general classification-based methods, various kinds of the Android malware detection methods were surveyed. As shown in Table X, the detection accuracy or the F-measure values of our framework were higher than the other methods including the deep learning based methods[52], [55], [20], [36].

Table V
PUBLICLY AVAILABLE MOST POPULAR DATASET

	Original Label	Sources	Description
1	Android Malware Genome Project	http://www.malgenomeproject.org	2539 benign, 1260 malware
2	M0Droid Dataset	http://m0droid.netai.net/modroid/	
3	The Drebin Dataset	http://user.informatik.uni-goettingen.de/~darp/drebin/	5560 benign ,9476 malware
4	AndroMalShare	http://sanddroid.xjtu.edu.cn:8080/#home	
5	Kharon Malware Dataset	http://kharon.gforge.inria.fr/dataset/	
6	AMD Project	http://amd.arguslab.org	
7	AAGM Dataset	http://www.unb.ca/cic/datasets/android-adware.html	
8	Android PRAGuard Dataset	http://pralab.diee.unica.it/en/AndroidPRAGuardDataset	
9	AndroZoo	https://androzoo.uni.lu/	
10	A Dataset based on ContagioDump	http://cgi.cs.indiana.edu/~nhusted/dokuwiki/doku.php?id=datasets	

Table IX
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

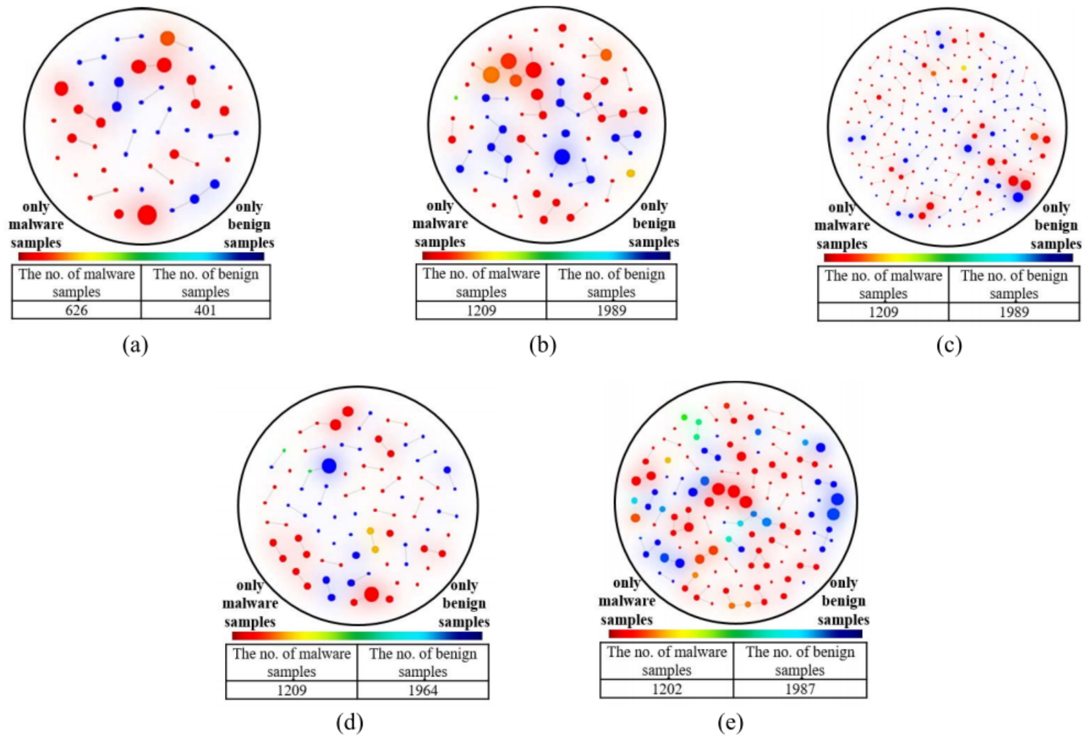

Figure 18. Topological data analysis (TDA) result of each feature data. Density-based spatial clustering algorithm was utilized in the TDA. (a) - (e): the visualized result for each feature type. Malicious samples from Malgenome project were used.[1]

Table X
CLASSIFICATION RESULTS

Authors	Algorthim	Capicity for feature diversity	Accuracy	F-measure
[1]	Multimodal deep neural network	High	98%	0.99
[78]	Ranking Based	High	98%	0.98
[3]	KNN & Navie Bayes	High	98%	0.98
[79]	DNN/RNN	medium	90%	NA
[35]	CNN	low	90%	NA
[80]	XGBoost	low	97%	0.97
[17]	Adaboost/ NB/ DT	Low	N.A	0.78
[81]	NB	low	93%	NA
[75]	SVM	low	93.9	NA
[82]	KNN+Kmeans	low	NA	0.91
[83]	Bayesian	low	92%	NA
[63]	SVM	low	NA	0.98
[84]	RF	low	97.5%	NA

V. CONCLUSION

This chapter analyzed a broad variety of mechanisms for analyzing and detecting Android malware, highlighting evolving patterns in its methods. This article also addressed the potential of Android malware to hinder research and escape detection, including deep learning and machine learning approaches. This chapter assessed the effectiveness of current methods for analyzing the malware and detection techniques. That's different from previous surveys that usually study mobile attacks only, this chapter introduce static, dynamic and hybrid analysis techniques and proposed algorithms.

REFERENCES

[1] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2018.

[2] R. Kumar, X. Zhang, R. Khan, A. Sharif, R. Kumar, X. Zhang, R. U. Khan, and A. Sharif, "Research on Data Mining of Permission-Induced Risk for Android IoT Devices," *Applied Sciences*, vol. 9, p. 277, jan 2019.

[3] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A multimodal malware detection technique for android iot devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[4] M. Alazab and R. Broadhurst, "An analysis of the nature of spam as cybercrime," in *Cyber-Physical Security*, pp. 251–266, Springer, 2017.

[5] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and api calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.

[6] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, 2020.

[7] S. Venkatraman and M. Alazab, "Use of data visualisation for zero-day malware detection," *Security and Communication Networks*, vol. 2018, 2018.

[8] M. Alazab, S. Huda, J. Abawajy, R. Islam, J. Yearwood, S. Venkatraman, and R. Broadhurst, "A hybrid wrapper-filter approach for malware detection," *Journal of networks*, vol. 9, no. 11, pp. 2878–2891, 2014.

[9] K.-N. Tran, M. Alazab, R. Broadhurst, *et al.*, "Towards a feature rich model for predicting spam emails containing malicious attachments and urls," 2014.

[10] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Information security governance: the art of detecting hidden malware," in *IT security governance innovations: theory and research*, pp. 293–315, IGI Global, 2013.

[11] S. Mansfield-Devine, "Paranoid android: just how insecure is the most popular mobile platform?," *Network Security*, vol. 2012, no. 9, pp. 5–10, 2012.

[12] S. Mansfield-Devine, "Android malware and mitigations," *Network Security*, vol. 2012, no. 11, pp. 12–20, 2012.

[13] S. Gold, "Android insecurity," *Network Security*, vol. 2011, no. 10, pp. 5–7, 2011.

[14] 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 (Switzerland)*, 2018.

[15] B. L. Risteska Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," 2017.

[16] Z. Aung and W. Zaw, "Permission-Based Android Malware Detection," *International Journal of Scientific & Technology Research*, 2013.

[17] C. Y. Huang, Y. T. Tsai, and C. H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," *Smart Innovation, Systems and Technologies*, 2013.

[18] W. Peng, L. Huang, J. Jia, and E. Ingram, "Enhancing the Naive Bayes Spam Filter Through Intelligent Text Modification Detection," in *Proceedings - 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018*, 2018.

[19] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Risk analysis of Android applications: A user-centric solution," *Future Generation Computer Systems*, 2018.

[20] Y. Li, Y. Li, H. Yan, and J. Liu, "Deep joint discriminative learning for vehicle re-identification and retrieval," *Proceedings - International Conference on Image Processing, ICIP*, vol. 2017-Septe, pp. 395–399, 2018.

[21] A. Kumar, K. S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic Analysis of MObile devices Using Scoring of application permissions," *Future Generation Computer Systems*, 2018.

[22] U. Pehlivan, N. Baltaci, C. Acarturk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based Android malware detection," in *IEEE SSCI 2014: 2014 IEEE Symposium Series on Computational Intelligence - CICS 2014: 2014 IEEE Symposium on Computational Intelligence in Cyber Security, Proceedings*, 2014.

[23] P. P. Chan and W. K. Song, "Static detection of Android malware by using permissions and API calls," in *Proceedings - International Conference on Machine Learning and Cybernetics*, vol. 1, pp. 82–87, 2014.

[24] W.-C. Wu and S.-H. Hung, "DroidDolphin: A dynamic android malware detection framework using big data and machine learning," in *2014 Conference on Research in Adaptive and Convergent Systems, RACS 2014*, pp. 247–252, 2014.

[25] Y. Aafer, W. Du, conference on security, H. Y. I. privacy In, and undefined 2013, "Droidapiminer: Mining api-level features for robust malware detection in android," *Springer*.

[26] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proceeding of Black Hat, Abu Dhabi*, 2011.

[27] H. Y. Chuang and S. D. Wang, "Machine Learning Based Hybrid Behavior Models for Android Malware Analysis," in *Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015*, 2015.

[28] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," *IEEE Transactions on Information Forensics and Security*, 2016.

[29] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *Proceedings - 2014 8th International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2014*, 2014.

[30] S. Hou, A. Saas, L. Chen, Y. Ye, and T. Bourlai, "Deep Neural Networks for Automatic Android Malware Detection," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017 - ASONAM '17*, 2017.

[31] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning," in *2016 IEEE 37th Sarnoff Symposium*, pp. 160–165, IEEE, sep 2016.

[32] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: An Android Malware Detection System Using Deep Belief Network Based on API Call Blocks," pp. 54–66, Springer, Cham, 2016.

[33] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-Based Android Malware Detection," in *2017 International Conference on Software Security and Assurance (ICSSA)*, pp. 60–65, IEEE, jul 2017.

[34] 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 *Proceedings - IEEE Symposium on Computers and Communications*, 2017.

[35] N. McLaughlin, A. Doupé, G. Joon Ahn, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, and Z. Zhao, "Deep Android Malware Detection," *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17*, pp. 301–308, 2017.

[36] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2017.

[37] W. Li, Z. Wang, J. Cai, and S. Cheng, "An Android Malware Detection Approach Using Weight-Adjusted Deep Learning," in *2018 International Conference on Computing, Networking and Communications, ICNC 2018*, 2018.

[38] Y. Zhang, Y. Yang, and X. Wang, "A Novel Android Malware Detection Approach Based on Convolutional Neural Network," in *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy - ICCSP 2018*, (New York, New York, USA), pp. 144–149, ACM Press, 2018.

[39] L. Shiqi, T. Shengwei, Y. Long, Y. Jiong, and S. Hua, "Android malicious code classification using deep belief network," *KSII Transactions on Internet and Information Systems*, 2018.

[40] P. Wang and B. Li, "Vehicle Re-Identification Based on Coupled Dictionary Learning," 2018.

[41] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, 2018.

[42] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 473–487, IEEE, apr 2018.

[43] D. Li, Z. Wang, and Y. Xue, "Fine-grained Android Malware Detection based on Deep Learning," in *2018 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–2, IEEE, may 2018.

[44] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," in *Proceedings - 2018 IEEE 14th International Colloquium on Signal Processing and its Application, CSPA 2018*, 2018.

[45] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, p. 15, 2011.

[46] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Proceedings - 2011 7th International Conference on Computational Intelligence and Security, CIS 2011*, 2011.

[47] Y. J. Ham and H.-W. Lee, "Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events," *International Journal of Computer and Communication Engineering*, 2014.

[48] 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," *International Journal of Security and its Applications*, 2014.

[49] 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," *Information and Software Technology*, vol. 75, pp. 17–25, 2016.

[50] S. Malik and K. Khatter, "System call analysis of Android Malware families," *Indian Journal of Science and Technology*, vol. 9, jun 2016.

[51] F. Tchakounté, P. D. I. J. o. S. And, and U. 2013, "System calls analysis of malwares on android," *International Journal of Science and Technology*, vol. 2, no. 9, pp. 669–674, 2013.

[52] 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 Generation Computer Systems*, vol. 83, pp. 193–207, 2018.

[53] A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, "Extinguishing ransomware - a hybrid approach to android ransomware detection," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[54] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in *2016 IEEE International Conference on Consumer Electronics, ICCE 2016*, 2016.

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

[56] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[57] B. Thomas, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010*, 2010.

[58] 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 *NDSS (Network and Distributed System Security Symposium)*, 2012.

[59] 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," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[60] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, 2014.

[61] K. Patel and B. Buddadev, "Detection and Mitigation of Android Malware Through Hybrid Approach," in *International symposium on Security in Computing and Communication*, 2015.

[62] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," in *Procedia Computer Science*, 2015.

[63] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[64] 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 *2016 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pp. 54–61, IEEE, feb 2016.

[65] J. W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "AndroDumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," *Computers and Security*, 2016.

[66] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digital Investigation*, 2015.

[67] 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 Transactions on Information Forensics and Security*, vol. 12, no. 5, 2017.

[68] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "Evedroid: Event-aware android malware detection against model degrading for iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[69] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[70] H. Zhu, Y. Li, R. Li, J. Li, Z.-H. You, and H. Song, "Sedmdroid: An enhanced stacking ensemble of deep learning framework for android malware detection," *IEEE Transactions on Network Science and Engineering*, 2020.

[71] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, "Dapasa: detecting android piggybacked apps through sensitive subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1772–1785, 2017.

[72] I. B. Mohamad and D. Usman, "Standardization and its effects on k-means clustering algorithm," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 17, pp. 3299–3303, 2013.

[73] G. Piatetsky-Shapiro, "Discovery, analysis and presentation of strong rules," *Knowledge Discovery in Databases*, 1991.

[74] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *International Conference on Wireless and Mobile Computing, Networking and Communications*, pp. 354–358, 2014.

[75] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proceedings 2014 Network and Distributed System Security Symposium*, 2014.

[76] S. H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," *Journal of Network and Computer Applications*, vol. 38, pp. 43–53, feb 2014.

[77] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," vol. 3, no. 1, pp. 1–122, 2014.

[78] S. Y. Yerima and S. Sezer, "Droidfusion: a novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.

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

[80] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, nov 2016.

[81] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs," 2014.

[82] 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 *Proceedings of the 2012 7th Asia Joint Conference on Information Security, AsiaJCIS 2012*, 2012.

[83] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pp. 659–674, 2015.

[84] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: evaluating Android anti-malware against transformation attacks," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, (New York, New York, USA), p. 329, ACM Press, 2013.