IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 13, 2017, accepted November 20, 2017, date of publication January 12, 2018,
date of current version February 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2792941

SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System

SABA ARSHAD[1], MUNAM A. SHAH[1], ABDUL WAHID[1], AMJAD MEHMOOD[2], HOUBING SONG[3], (Senior Member, IEEE), AND HONGNIAN YU[4, 5]

[1]Department of Computer Science, COMSATS Institute of Information Technology, Islamabad 45550, Pakistan
[2]Institute of Information Technology, Kohat University of Science and Technology, Kohat 26000, Pakistan
[3]Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114 USA
[4]School of Computer Science and Network Security, Dongguan University of Technology, Shongshanhu 523808, China
[5]Faculty of Sciences and Technology, Talbot Campus, Bournemouth University, Bournemouth BH12 5BB, U.K.

Corresponding author: Amjad Mehmood (dramjad.mehmood@ieee.org)

ABSTRACT For the last few years, Android is known to be the most widely used operating system and this rapidly increasing popularity has attracted the malware developer's attention. Android allows downloading and installation of apps from other unofficial market places. This gives malware developers an opportunity to put repackaged malicious applications in third-party app-stores and attack the Android devices. A large number of malware analysis and detection systems have been developed which uses static analysis, dynamic analysis, or hybrid analysis to keep Android devices secure from malware. However, the existing research clearly lags in detecting malware efficiently and accurately. For accurate malware detection, multilayer analysis is required which consumes large amount of hardware resources of resource constrained mobile devices. This research proposes an efficient and accurate solution to this problem, named SAMADroid, which is a novel 3-level hybrid malware detection model for Android operating systems. The research contribution includes multiple folds. First, many of the existing Android malware detection techniques are thoroughly investigated and categorized on the basis of their detection methods. Also, their benefits along with limitations are deduced. A novel 3-level hybrid malware detection model for Android operating systems is developed, that can provide high detection accuracy by combining the benefits of the three different levels: 1) Static and Dynamic Analysis; 2) Local and Remote Host; and 3) Machine Learning Intelligence. Experimental results show that SAMADroid achieves high malware detection accuracy by ensuring the efficiency in terms of power and storage consumption.

INDEX TERMS Accuracy, android operating system, dynamic analysis, efficiency, hybrid malware detection, machine learning, memory usage, performance overhead, power consumption, static analysis.

I. INTRODUCTION

Android operating system is known to be the most popular and widely used operating system. According to the Gartner report, Android dominated the operating system market by capturing 81.7% of total market shares by the end of 2016 [1]. Figure 1 shows the market shares of Android operating system on yearly basis. It can be observed that Android has become the most widely used operating system over the years. With the increasing popularity of Android OS every year, Android malware attacks are also growing rapidly. TrendMicro declared that number of Android malwares has increased to 10.6 million [2].

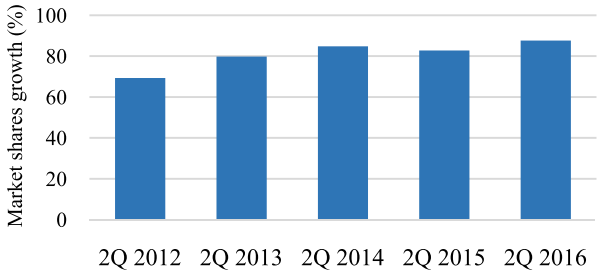

FIGURE 1. Android market shares.

Figure 2 depicts the yearly increase in Android malwares. A lot of antimalware techniques have been proposed to protect Android devices from malwares. These techniques can be classified as two main approaches: Static Analysis Approach and Dynamic Analysis Approach. In Static analysis, applications are analyzed by scanning all the code included in the application package instead of executing them. This is

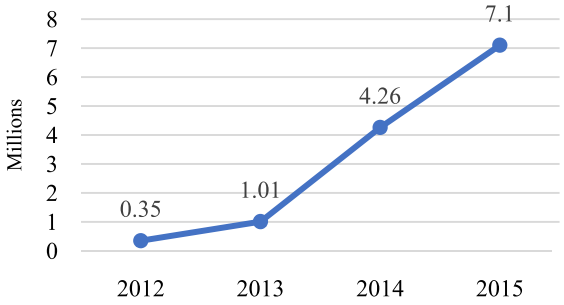

FIGURE 2. Android malware growth.

useful for identifying the malicious code of applications which only executes at specific conditions, e.g., system reboots etc. but it cannot detect the encrypted and dynamically loaded malicious code. While in Dynamic analysis, only runtime behavior of application is analyzed, e.g., system call tracing etc. It can detect the encrypted and dynamically loaded malicious data. A better solution to overcome the limitations of both analysis approaches is to use a hybrid analysis, which is a combination of the static and dynamic analysis technique. These limitations force the researchers to develop the hybrid analysis mechanisms to achieve high detection rates. Although hybrid analysis techniques succeed to achieve high accuracy by combining the benefits of static and dynamic analysis approaches but they fail to ensure the efficiency of mobile resources. Existing Antimalware techniques that are hybrid of static and dynamic analysis are either performed on the local host (on-device) or at remote host (off-device). On-device hybrid analysis helps to generate the quick analysis results but mobile devices are usually resource constrained and requires more hardware resources to perform the hybrid analysis on the device. Instead, analysis on the remote server helps to reduce the resource consumption of mobile devices but it generates unrealistic results, thus lowering the detection accuracy.

In this paper, a novel 3-level hybrid Android malware detection model is proposed named as SAMADroid. It is a hybrid between three levels for malware analysis and detection: i) Static & Dynamic Analysis; ii) Local & Remote Host; iii) Machine Learning Intelligence. In static analysis phase, experiments are performed for feature selection, in order to obtain the features which can provide maximum and useful information about the application behavior. For this purpose, Drebin's features sets are used, with little alterations. Drebin is a static analysis malware detection framework which detects malwares with high accuracy, although it lacks dynamic analysis and cannot detect encrypted and dynamically loaded malicious code [3]. For dynamic analysis, system calls are traced at runtime. Different machine learning algorithms are applied and their performance is compared to get the most accurate machine learning technique. The proposed scheme is designed to overcome the resource efficiency problem of the existing anti-malware systems. Our contribution includes multiple folds.

1) We thoroughly investigate most of the existing antivirus programs that act against malwares to protect Android systems and categorize them on the basis of their detection methods.
2) We provide an easy and concise view of the existing malware detection and protection mechanisms and deduce their benefits and limitations.
3) We developed a novel 3-level hybrid malware detection model for Android operating systems, that can provide high detection accuracy by combining the benefits of three different levels and ensure the resource efficiency.
4) Through SAMADroid, we provide explanation to Android users about the behavior of application. This feature of SAMADroid helps the Android users to become aware of behavior of different applications.

The rest of the paper is organized as follows: Section II contains the literature review of existing Android malware detection techniques that uses hybrid analysis. In Section III, we explained our proposed solution, SAMADroid, which overcomes the resource consumption problem and achieves high detection accuracy at low mobile resource consumption. Section IV comprises of experiments performed during development of SAMADroid and the results obtained and finally the paper is concluded in Section V.

II. LITERATURE REVIEW

Standalone static and dynamic analyses have their own limitations due to which efforts have been made towards the development of hybrid antimalware techniques. Although the hybrid detection schemes have resolved the problem of accuracy in malware detection up to some extent but still they are inefficient. We have categorized the existing Android malware detection hybrid techniques as follows:

1) Static and Dynamic Analysis
2) Static, Dynamic Analysis and Machine Learning
3) Static, Dynamic Analysis and Local, Remote Host.

A. STATIC AND DYNAMIC ANALYSIS

This category includes Android malware detection techniques that uses hybrid of static and dynamic analysis to achieve high malware detection accuracy.

Bläsing *et al*. [4] proposed an Android application sandbox which can detect suspicious behavior in Android applications by analyzing the app both statically and dynamically. In static analysis phase, Android application is decompressed first and then decompiled by using *Baksmali* tool [5]. Static patterns are then extracted by scanning the decompiled smali files. Static patterns include usage of *Runtime.Exec()* method, java native libraries, reflection, permissions, service and IPC provision. While in the dynamic analysis phase, the application is installed and executed on the Android emulator. Monkey tool is used to operate the application with random user inputs e.g. gestures, touches and clicks. AASandbox placed in the kernel space uses loadable kernel module for execution of the application under fully controlled environment and creates the logs by hijacking the system calls. Mathematical behavior

vectors are created from the static behavior logs and dynamic behavior logs. These vectors can be used for the detection of suspicious behavior of the application and classification of application either manually or automatically. Through experimentation, they have shown that the proposed method performs correctly when they have applied a handwritten program named fork bomb that uses *Runtime.Exec()* and generated the static behavior vector and dynamic behavior vector. The output generated by the system clearly shows that the program uses *Runtime.Exec()*. The major limitation of AASandbox is that it provides security to Android devices by hijacking the system calls. It cannot be distributed in the mass market because it requires the root privileges to hijack the system calls on a device. They have not provided accuracy and efficiency measures either.

Zhou *et al.* [6] have proposed DroidRanger for detection of known and zero day malwares in popular Android markets. They have collected 204,040 Android applications among which 75% are from Android official market [7] and 25% are collected from four alternative Android application markets, eoeMarket [8], gfan [9], alcatelclub, and mmoovv. By using static features, DroidRanger detects known malwares in two steps: Initially it uses essential permissions, required by the malware to perform the intended functionality, to filter the malicious Apps. This process greatly reduces the number of Apps for further examination in second step. After permission filtering, behavioral filtering is performed where App's behavior is analyzed by using information from manifest file and API calls. This information is then mapped against the behavior rules to detect and filter the known malwares on the basis of their behavior. Detection of unknown malwares is performed in two phases: In the first phase heuristic based filtering is carried out. Two heuristics have been considered by the authors in this scheme. In the first heuristic, they focused the dynamic loading of java binary code from remote server. Android uses DexClassLoader [10] for dynamic loading of java code. While evaluation, authors have found that 1055 applications used this class. This heuristic enabled the system to detect zero-day malware, *Plankton*. The other heuristic is related to dynamic loading of native code locally such that if an app containing native code stored in the directory other than default directory such app's behavior is unusual. This heuristic helps to discover the malwares which attacks the OS kernel to get access on the root privileges. Through this heuristic they discovered *DroidDreamLight* [11], a zero-day malware from Android's official market. In the second phase, it performs dynamic execution monitoring to inspect the runtime behavior of the applications on the basis of heuristics. API calls and their arguments are recorded in order to detect the malicious behavior of dynamically loaded java code and system calls are traced for detection of malicious behavior performed by the app by dynamically loading native code. Among 204,040 applications DroidRanger successfully detected 211 infected applications where 40 apps contained zero-day malware and 171 apps were infected from known malware. This malware detection technique is only developed for Android markets, not for Android devices.

B. STATIC, DYNAMIC ANALYSIS & MACHINE LEARNING

Antimalwares that are hybrid of static and dynamic analysis and machine learning techniques fall into this category. Wu *et al*. [12] proposed DroidDolphin, a cloud based malware detection framework that uses both static and dynamic analysis for malware detection. The proposed scheme consists of four phases: preprocessing, emulation and testing, feature extraction and machine learning. In preprocessing phase they used *APIMonitor* [13] for monitoring malicious API calls. First of all, *APIMonitor* reverse engineers the APK file, wraps the API call with *DroidBox* [14] version and then repackages the application. This process helps to identify the malicious API calls by tracing the call logs as it generates the message whenever the application triggers an API call. Authors have recorded 25 API calls in this process. For emulation and testing they installed the applications on Android Virtual Devices and *APE_BOX* which is a combination of *APE* [15] and *DroidBox*. Through *DroidBox*, 13 runtime activities were recorded. *APE* enables the proposed system to simulate GUI based events and helps to traverse application's code path in order to identify the malicious behaviors. Ngram model is used for feature representation. Features extracted through *APIMonitor*and *APE_BOX* are given as input to the *SVM* [16] classifier and *LIBSVM* [17] is used for malware detection model implementation. In order to evaluate the performance of proposed scheme authors have used big training dataset which includes 32000 malicious and 32000 benign apps. Test set consists of 1000 benign and 1000 malicious apps. They have compared the system's performance with the *STREAM* [18]. Through experimental results it is shown that the proposed scheme can achieve 86.1% accuracy which is higher than that of *STREAM* but at the cost of low efficiency.

Wang *et al*. [19] proposed a hybrid malware detection scheme that detects known malwares and their variants through signature based misuse detection and zero-day malwares through anomaly detection. In static analysis, static features are extracted from manifest file and disassembled dex files through *Android Asset Packaging Tool* [20]. For dynamic features extraction, *Cuckoo Droid* [21] is used and a simulation tool *Robotium* [22] is used to enhance *Cuckoo Droid*. These features, static and dynamic, are then mapped into vector space where each dimension has a value 0 or 1. In order to enhance the accuracy and performance, different feature selection methods are applied for misuse and anomaly detection. After feature selection, *Linear SVC* classifier [23] is applied. If the application is classified as a malware, the system further detects whether it is a known malware or a variant of known malware family. On the basis of similarity in signature it classifies the malware and updates the training database. On the other hand, if misuse detection cannot detect the malicious behavior in application then anomaly detection is performed. It uses *One-Class SVM classifier* [24]. If any abnormal behavior is detected it classifies the app as

zero-day malware and update the training database. For evaluation, they have used 12000 benign apps from different china App stores and 5560 malware samples. Results shows that the misuse detection can detect with 98.79% true positive rate and anomaly detector can detect with 98.76% true positive rate.

Patel and Buddadev [25] proposed a malware detection system that performs both static and dynamic analysis of application under observation and then applies machine learning technique in order to create rules. These rules are then used to classify the application as benign or malware. In static analysis phase, it extracts the permission related parameters e.g. intents, services and broadcast receivers from the manifest file by using *Android Asset Packaging Tool* [20]. In dynamic analysis, the system executes the application on *Android emulator* [26] and extracts the features related to user interactions, java based and native based function calls. Once the permission based and behavior based features are collected, they are stored into CSV file. In the next stage, feature selection is performed on the basis of information gain. Selected features are recorded in the CSV file along with the name of the application and class of application e.g. malware or benign. This CSV file is then used by the rule generation module for creating rules that are used to correctly classify the application. It classifies the application by mapping the permissions against the function calls e.g. if an app tries to send message or access device id by API calls and does not mention the permission request for message sending or access to device id then such an application is classified as malicious. For experiments authors have used datasets of various sources (*Droidkin* [27] and *contagioDump* [28]). Experimental results show that they achieved high detection rate of 96.4% but at low efficiency because they performed both static analysis, dynamic analysis and machine learning based detection on local device which results in high scanning time, high power consumption and high resource/storage consumption.

Shijo and Salim [29] have proposed a hybrid malware detection technique which integrates the static analysis and dynamic analysis and then applies machine learning for detection. In static analysis phase, the system extracts PSI as static features from the binary code files. On the other hand, in dynamic analysis Cuckoo malware analyzer is used for dynamic feature extraction by execution of application. API call logs are extracted from the binary executable files and their sequence is used to distinguish between malicious and benign apps. These API call sequence are analyzed on the basis of n-gram based method in which API call grams are created for each file and sorted according to the occurrence frequency in each file. API call grams below the threshold value are eliminated and remaining call grams are used for feature vector creation. Once the static and dynamic feature vectors are created the proposed scheme concatenates the two feature vectors for each file in order to produce an integrated feature vector and input these feature vectors to two different machine learning techniques for classification. For evaluating the performance of proposed system authors have used *Weka* tool [30] and applied the two machine learning techniques, *SVM* [31], [32] and *Random Forest* [33]. The data set consists of 997 malicious files and 490 benign files. Results show that the proposed scheme can classify the unknown applications with 98.7% accuracy but this scheme requires large storage and computation resources.

DroidDetector is an online system developed for detection of Android malwares [34]. It performs both static and dynamic analysis at remote server and then applies deep learning for separation of malwares from legitimate apps. For static analysis, permissions and sensitive API calls are used. These static features are extracted using *7-zip* [35], *AXML-Printer2* [36], *TinyXml* [37] and *Baksmali* tools [4]. Dynamic behaviors are extracted for dynamic analysis through *DroidBox* tool. Each Android application is executed in the *DroidBox* for specific period of time and its dynamic behavior is monitored. The extracted features, both static and dynamic, are then embed into feature vectors. These features are given as input to deep learning model for detection of malicious applications. For evaluation of DroidDetector, authors have used 20,000 legitimate apps from Google Play store and malware apps from Contagio mobile, 500 apps, and Genome project, 1260 apps. Experimental results show that it can detect malwares with 96.7% accuracy. Dynamic analysis is performed at remote server for a specific period of time. Major limitation of this technique is that malwares which do not show malicious behavior during that monitoring interval may evade the detection system.

Liu *et al*. [38] proposed a hybrid malware detection scheme which can detect the malicious behavior by static or dynamic analysis of the application. It initially decompiles the application by using *Apktool* [39] and applies a program that analyses the decompiled results of the application and automatically switches the application to static or dynamic analysis procedure. If the app is successfully decompiled then static analysis is performed where manifest file [40] is scanned and permissions and API keywords in Smali files are extracted as static features. Feature vector is generated from these features and passed to machine learning classifier for classification of the application as benign or malware. On the other hand, if application is not correctly decompiled, as a result of any transformation technique [41], then dynamic analysis is performed on the application. Firstly, it installs and executes the application on the experimental device and traces the system call logs by using *Strace* tool [42]. Feature vector is generated from these logs and passed to machine learning classifier for evaluation. For performance evaluation, authors collected malware dataset from Gnome Project and benign apps dataset form Wandoujia [43], an Android app market, and applied *SVM* [32], *Naive Bayes* [44] and *kNN* [45], [46], machine learning algorithms, on the dataset. Experimental results of static analysis show that the proposed scheme can achieve 99% accuracy while 90% accuracy is achieved as a result of dynamic analysis. Limitations of this technique includes that static analysis is performed if the application is correctly decompiled; in that case, dynamic analysis is not performed

on the application. Detection system will not be able to detect the dynamically loaded data. Also, when dynamic analysis is performed, only the executed code is analyzed. The code that is not executed remains undetected, thus lacking the benefits of static analysis if dynamic analysis is performed and vice versa.

Saracino *et al.* [47] have defined the malware types on the basis of their behavioral class. The proposed scheme identifies the misbehavior performed by each malware type by correlating the features extracted at four different levels: package level, application level, user level and kernel level. At kernel level, it monitors and hijacks the system calls. At application level, it monitors the critical APIs to detect the malicious behaviors. User activities are monitored at user level and malicious events are detected when the user is idle or not interacting the device. At the fourth level, named as package level, proposed system identifies if the application under observation is risky or not, on the basis of permissions requested by the app and market information. MADAM performs App Risk Assessment statically where App evaluator analyzes the app's metadata, collected locally and remotely from the app market, finds if the app is risky or not and generates the list of suspicious Apps. Static features analyzed to assess the app's risk level are: permissions in the manifest file, market place from where the app is downloaded, no of downloads of this app, developer's reputation and user ratings. Once the riskiness of application is identified, it is added to the app suspicious list for monitoring at runtime for malicious behavior detection. Global monitoring block includes system call monitor, user activity monitor, message monitor, activity logger and classifier. System call monitor intercepts the system calls related to file operations and networks access. Message monitor hijacks the API calls *sendTextMessage*() and *sendDataMessage*() while the user activity monitor identifies if the user is active or idle through the Android APIs. The user is considered to be active if the screen is on and user is actively interacting with it or the screen is off and the phone call is ongoing. The action logger collects the features from three monitors and generates a feature vector consisting of 14 features among which 11 features represent the system calls. The values corresponding to these system calls shows the number of times each system call is triggered in specific time interval. User idleness feature value is '0' if the user is idle and '1' if the user is active. The last two features are collected from message monitor which shows the number of messages sent in a specific time interval and messages sent to the number not in device contact list. Finally, the vector is given as input to the *kNN* classifier which uses *Euclidean Distance Function* [48]. The proposed scheme is tested on dataset obtained from Genome [49], contagio mobile [50] and virus share [51] and achieves 96.9% detection rate which is comparable with VirusTotal [52]. Also, it is able to detect 9 such malware families that remain undetected by VirusTotal. MADAM has some limitations too but the major limitation is that it requires root privileges on the device to perform detection due to which it cannot be distributed in the mass market. Only markets can use it for detection of malicious apps on their store. Also, it is not very much efficient in terms of memory usage.

Chuang and Wang [53] proposed a hybrid behavior model for malware detection. In a proposed technique, APK file of each Android application is preprocessed initially. The preprocessing phase includes following steps: first of all, the application is disassembled by the reverse engineering tool, *Androgaurd* [54], in order to get the frequency of each API call used by the application. Then the statistics are generated for API calls in malicious and benign Apps. These statistics are then used for ranking the APIs on the basis of usage difference percentage of the API. Through the statistics, it is observed that the call frequency of dangerous APIs is higher in malicious apps than benign Apps. As a result of application preprocessing two feature vectors are generated, one includes APIs preferred by the malicious apps and other includes API preferred by the benign apps. These feature vectors are classified by the two behavior models, normal behavior model and malicious behavior model. *LIBSVM* [17] is used to build these models. The normal behavior model is trained on the feature set of APIs preferred by the benign apps. If new application, under analysis, does not act like other legitimate apps then it is classified as malware. On the other hand, feature set of APIs preferred by the malicious apps is used for the training of malicious behavior model. If an application does not act like other malicious applications, then malicious behavior model will classify it as benign app. After the separate analysis by each trained *SVM* model [23], the proposed scheme combines the two models by using fusion logic where score is calculated for each application. If the score value is larger than zero, the application is classified as True (malicious), otherwise False (benign). For performance evaluation of proposed model, they have collected 3368 malware samples from Contagio mobile [55] and 6005 benign Android apps [56]. This dataset is divided into training set (dataset A) and test set (dataset B). Experimental results show that the proposed hybrid model can detect the malicious apps with 96.69% accuracy and 95.25% detection rate. The proposed scheme cannot detect the dynamically loaded malicious code as it only performs static analysis.

C. STATIC, DYNAMIC ANALYSIS & ON-DEVICE, OFF-DEVICE DETECTION

Rodríguez-Mota [57] proposed a 2-Hybrid malware detection test framework which is an ongoing project and performs the analysis and detection of malwares on the device and remote server. The framework includes a feature collector unit which collects different features of the application, at installation. By analyzing these features local detector classifies the application as legitimate, malware or risky. If application is classified as benign, the response manager allows the installation of the App and if the App is classified as malware, response manager allows user to cancel the installation process. In the case of potential risk, the host based analysis does not completely classify the application as benign or malware, then

detailed analysis is performed at remote server. The cloud manager obtains the app's information from feature collector and performs dynamic analysis on the data at remote server. If any malicious behavior is detected in the application the remote server sends data to local device to be stored in local database for future detection of such apps. At this initial stage, they have collected 39 Trojan samples and listed 69 permissions requested in these malware samples. They compared their results with [58], [59] and found that from top 20 frequently requested permissions, 17 permissions are reported in these cited studies. Efficiency and accuracy measures cannot be determined at this stage.

Jang *et al.* [60] have proposed Andro-Dumpsys, and Android antimalware system. In Andro-Dumpsys, client application running on the Android device captures the application specific information and sends it to the server where detailed analysis and emulation based execution of application is performed. The application specific information includes the hash code of application and package name. If the application is available in the repository, the analysis is performed. Otherwise apk file is forwarded to the server from client device. Key features used for the analysis includes the serial number of the developer certificate, which is allocated to each developer during account registration. This feature helps to recognize the applications developed by malware developers quickly. Other features include suspicious API sequence, intent filters, permissions, system commands and the forged files. Using these features, malicious behavior of applications is identified. Experimental results show that Andro-Dumpsys can detect malware applications with 99% accuracy. The major limitation of this technique is that it uses emulation based detection, which can be easily evaded by the malwares. Also, the sending of complete apk file to server consumes lot of battery power and money at client device.

Talha *et al.* [61] proposed *APK Auditor*, an Android malware detection system that uses permissions as static analysis features for malware behavior detection. It consists of three components: 1) Signature database, which contains the signatures of all the applications; 2) Android client, which provides a service of malware analysis to the users; 3) Central server, which is responsible for connecting android client with signature database. Central server performs the analysis without downloading the application on the client device, thus saving the hardware resources. It extracts the permissions requested by the application and computes the permission malware score (PMS). Then it combines the PMS for each application and classifies the application as malware if the application malware score exceeds threshold value. APK Auditor uses *Logistic Regression* [62]–[64] for calculating threshold value by using 70% of available dataset as training data. Performance of proposed scheme is evaluated on 30% dataset. Results shows that APK Auditor achieves 92.5% specificity but it lacks the benefits of dynamic analysis as it cannot detect the dynamic malicious payloads.

In [65], a detection system is proposed named *Monet*, which can detect the variants of known malwares. Monet uses static features, from manifest file and disassembled code files, and dynamic behavior of applications for detection of variants of known malware families. It consists of client end application that executes on Android device, monitors the applications and generates the signatures. These signatures are forwarded to server which performs further detection by applying signature matching algorithm and sends back the detection results. For evaluation of Monet, authors used dataset of 3723 malwares and 500 legitimate applications. Through experimental results, it is shown that Monet can detect variants of known malware families with 99% accuracy. Monet causes overall 7.4% performance overhead and 8.3% memory overhead which is not negligible for resource constrained smart phones. Thus, Monet is good at accuracy but lacks resource efficiency.

A detailed comparative study of existing Android malware detection hybrid techniques that lie in these three categories, discussed above, is described in Table 1.

III. PROPOSED SCHEME

SAMADroid is a 3-level hybrid malware detection model for Android devices. It is hybrid between following three levels for malware analysis and detection.

A. LEVEL 1: STATIC AND DYNAMIC ANALYSIS

At Level 1, the hybrid of static and dynamic analysis provides a highly accurate analysis as it combines the benefits of two analysis techniques. Through static analysis, it scans all the code of application and analyzes the malicious behavior of application without executing it. In static analysis phase, static features are extracted from the manifest file and dex code files of the application. Motivated from Drebin [3], same sets of static features are used for static analysis, with a little alteration, in order to achieve high detection accuracy. These static feature sets are grouped as follows:

S_1 : Requested Hardware Components
S_2 : Requested Permissions
S_2 : Application Components (Service, Receiver, Content Provider)
S_4 : Intent filters
S_5 : Suspicious API calls
S_6 : Restricted API calls

All the above static features are extracted using *Android Asset Packaging Tool* and*Baksmali tool.*

In dynamic analysis phase, system executes the application on the real device and analyzes its runtime behavior which also includes the monitoring of dynamically loaded and decrypted code. System calls are used as dynamic features. Applications installed on real Android devices are analyzed by system call tracing. These system calls allow us to overcome the limitations of static analysis and analyze the application's behavior in real time environment.

B. LEVEL 2: LOCAL AND REMOTE HOST

Level 2 is a hybrid of local and remote host. Detailed static analysis is performed on the remote host to achieve highly

TABLE 1. Hybrid malware detection techniques.

Ref.	Title	Year	Methodology	Hybrid of	Tools	Achievements	Limitations
[4]	An Android Application Sandbox System for Suspicious Software Detection	2010	• Decompresses and decompiles android app by using Baksmali tool. • Extracts static patterns by scanning decompiled smali files. • Generates static behavior vector. • Installs and executes the application on emulator. • Runs monkey to give user inputs. • Hijacks system calls using LKM. • Logs the system calls. • Generates dynamic behavior vector.	• Static analysis • Dynamic analysis (emulation based)	• 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 of the system.
[6]	Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets	2012	• Detects known malware samples by permission based filtering and behavioral foot printing. • Detects zero-day malwares by heuristic based filtering and dynamic execution monitoring.	• Static analysis • Dynamic analysis	-------	• Successfully detects 211 malicious apps among 204,040 apps. • Detected two zero-day malwares DroidDream Light and Plankton.	• This study is limited to two heuristics • Permission based filtering only considered the essential permissions of 10 malware families.
[12]	DroidDolphin: A Dynamic Android Malware Detection Framework Using Big Data and Machine Learning	2014	• Preprocesses the App through APIMonitor to obtain static features such as API calls. • Installs 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 for classification	• Static analysis • Dynamic analysis • Machine learning	• APIMon-itor • APE • DroidBox • LIBSVM	• Achieves 86.1% accuracy.	• Time consuming due to use of emulators • High resource consumption in log collection. • Can be easily evaded by malwares using anti-emulator techniques.
[19]	A Novel Hybrid Mobile Malware Detection System Integrating Anomaly Detection with Misuse Detection	2015	• Extracts the static features from manifest file and disassembled. dex files 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 family of malware. • Applies anomaly detection if App is not classified by misuse detection and uses One-class SVM to classify the app as benign or zero-day malware.	• Static analysis (signature based) • Dynamic analysis (behavio r based) • Machine Learning	• Android Asset Packaging Tool. • Cuckoo Droid • Robotium	• Detects known malwares and their variants with 98.79% true positive rate. • Detects the zero-day malwares with 98.76% true positive rate.	• Comparison of proposed scheme with other well-known malware detection schemes e.g. RiskRanker, Drebin, Kirin etc. is not provided.
[25]	Detection and Mitigation of Android Malware Through Hybrid Approach	2015	• In static analysis phase, it extracts the permission related parameters e.g. intents, services and broadcast receivers from the manifest file by using Aapt. • In behavior analysis phase, executes the App on android emulator and extracts the features related to user interactions, java based and native based function calls. • 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 permissions against the function calls for classification.	• Static analysis (permission based) • Dynamic analysis (behavior based) • Machine learning	Android Asset Packaging Tool	Achieves 96.4% detection rate	• High scanning time • High power consumption • High resource/ storage consumption

TABLE 1. *(Continued.)* **Hybrid malware detection techniques.**

[29]	Integrated static and dynamic analysis for malware detection	2015	• Extracts PSI as static features from the binary code files • Sorts features according to occurrence frequency in each file • Selects feature with occurrence frequency above certain threshold value and create static feature vector. • Uses Cuckoo malware analyzer for dynamic feature, API call, extraction. • Creates API call grams for each file and analyzes API call sequences on the basis of n-gram based method • Selects API call grams above certain threshold value and create dynamic feature vector. Concatenates both feature vectors for each file and input them to Machine learning classifiers.	• Static analysis • Dynamic analysis • Machine Learning	Weka	Classifies the unknown applications with 98.7% accuracy.	• Comparison of proposed scheme with other well-known malware detection schemes e.g. RiskRanker, Drebin, DroidRanger etc. is not provided.
[34]	DroidDetector: Android Malware Characterization and Detection Using Deep Learning	2016	• Extracts permissions and sensitive API calls as static features • Logs dynamic actions for dynamic analysis • Applies deep learning model for classification	• Static analysis • Dynamic analysis • Machine learning	• 7-zip • XML-Printer2 • Tinyxml • Baksmali • DroidBox	Detects malwares with 96.7% accuracy	• Unrealistic dynamic analysis Malwares which do not show malicious behavior during monitoring interval may evade the detection system
[38]	A Hybrid Malware Detecting Scheme for Mobile Android Applications	2016	• Decompiles application using Akptool and analyzes the decompiled results. • Automatically switches to static analysis if App is correctly decompiled. • Performs extraction of static features, permissions and API calls, from manifest and smali files. • Inputs the feature vectors to machine learning classifiers, SVM, kNN and Naïve 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.	• Static analysis (permission based) • Dynamic analysis (behavior based) • Machine learning	• Apktool • Strace • Monkey Tool	Achieves 99% accuracy as a result of static analysis and 90% accuracy as a result of dynamic analysis.	• Only performs static or dynamic analysis on the application thus detection system will not be able to detect the dynamically loaded data in case of static analysis. • When dynamic analysis is performed, only the executed code is analyzed. The code that is not executed remains undetected.
[47]	MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention	2016	• Extracts features at four different levels: package level, application level, user level and kernel level. • Monitors the system calls at kernel level, critical APIs at application 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 vectors to KNN classifier. • Notifies the user about malicious apps and helps the user to block and remove them through UI.	• Static analysis • Dynamic analysis On-device	• Superuser • X-posed Installer	Achieves low performance (1.4%) and energy overhead (4%) Achieves 96.9 % detection rate	• Only runs on rooted device with a kernel 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 App suspicious list and so will not be detected against known malware behavior patterns • Only the apps identified as risky are added to the App suspicious list • 9.4% memory overhead because classifier requires the training data in memory.
[57], [66]	Towards a 2-hybrid Android Malware Detection Test Framework	2016	• Feature collector collects static features of the application, at installation. • GarmDroid, a web tool that extracts the features of Applications and provides their visual representation in order to identify the threats posed by the Application.	• Static analysis • Dynamic analysis • On-device • Off-device	---------	From top 20 enlisted frequently requested permissions, 17 permissions are reported in [58] and [59]	----------

TABLE 1. ***(Continued.)*** **Hybrid malware detection techniques.**

	Improving Android Mobile Application Development by Dissecting Malware Analysis Data		• Local detector classifies the application as legitimate, malware or risk using static features. • Response manager gives control to user if App is detected as malware. • Cloud detector performs detailed dynamic analysis at remote server if App is detected as risk by local detector. • Updates the database if App is detected a malware.				
[60]	Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information	2016	• Client application running on the Android device captures the application specific information and sends it to the server. • Detailed analysis and emulation based execution of application is performed. • If the application is available in the repository, the analysis is performed. Otherwise apk file is forwarded to the server from client device.	• Static analysis • Dynamic analysis • On-device • Off-device	Androgaurd	Detects malware applications with 99% accuracy.	• Emulation based detection can be easily evaded by the malwares.
[61]	APK Auditor: Permission based Android malware detection system	2015	• Uses permissions as static analysis features for malware behavior detection. • Signature database contains the signatures of all the applications • Android client provides a service of malware analysis to the users • Central server is responsible for connecting android client with signature database	• Static analysis • On-device • Off-device	---------	Achieves 92.5% specificity	• It lacks the benefits of dynamic analysis as it cannot detect the dynamic malicious payloads
[65]	Monet: A User-oriented Behavior-based Malware Variants Detection System for Android	2016	• Uses static features, from manifest file and disassembled code files. • Uses system calls and binder transactions as dynamic behavior features. • Client end application monitors the applications and generates the signatures. • These signatures are forwarded to server which applies signature matching algorithm.	• Static analysis • Dynamic analysis • On-device • Off-device	----------	Achieves 99% accuracy	• Causes overall 7.4% performance overhead and 8.3% memory overhead.

accurate results. On the local host, dynamic analysis is performed to take the realistic inputs from the user instead of using any programmed tool, *Monkey Runner*, which generates non-realistic random input events. On the basis of user inputs, system call logs are generated and forwarded to the remote server. Remote server keeps on analyzing the behavior of application on the basis of logs and extracted static features.

C. LEVEL 3: MACHINE LEARNING INTELLIGENCE

At Level 3, the feature vectors built from analyzed features are given as input to the machine learning intelligence unit to perform the detection of malicious behavior of unknown apps and to correctly classify them. All the applications are classified as malicious or benign. In SAMADroid, the detection operation is performed at remote host, thus keeping all the training dataset in memory of server, which is a resource rich system. This ultimately reduces the memory overhead.

All the three levels of SAMADroid model are shown in Figure 3. Following are the major components of this model.

1) CLIENT END

An Android interface application is developed for SAMADroid client end. It provides an interface which includes all the applications currently installed on the device either they are system applications or user applications. This interface allows the users to use any application through SAMADroid client application. When a user opens, and runs any other application through SAMADroid, the application runs smoothly. SAMADroid monitors the applications in background and do not affects the operations of other applications running on the device. If running application is system application, there is no need to check whether it is malicious or not, because system apps are added in the device by the manufacturers. On the other hand, if the application is user application, installed by user from any app-store, then

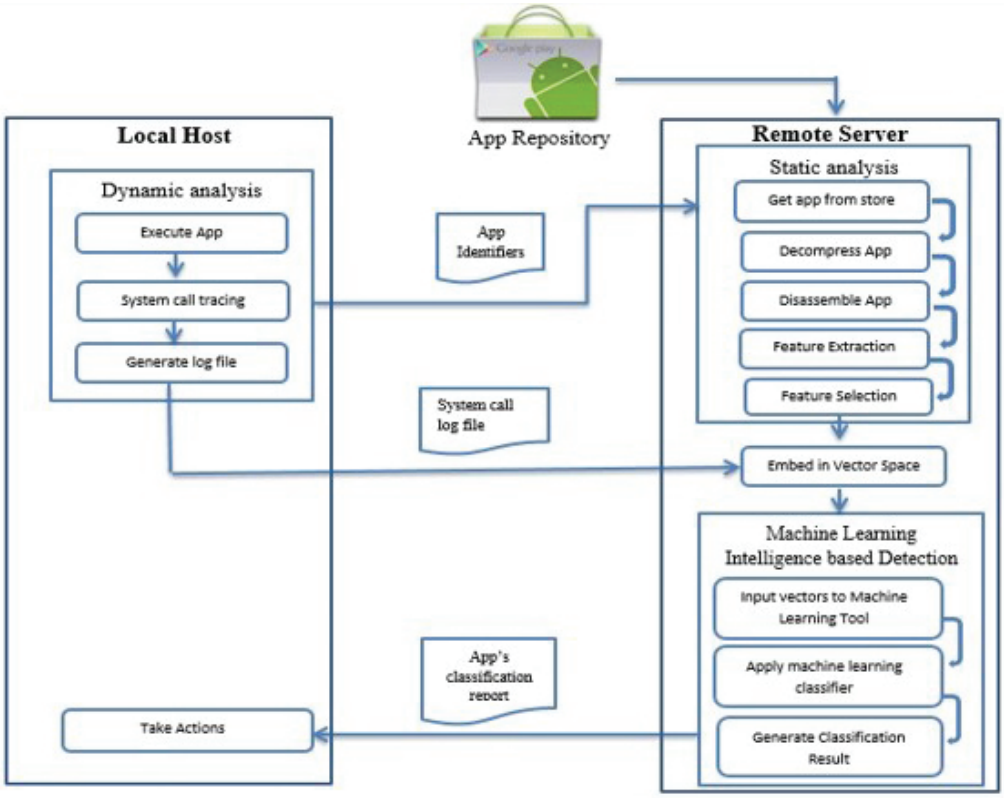

FIGURE 3. Architectural diagram of SAMADroid.

SAMADroid hooks the *Strace* tool with that application and starts tracing the system calls invoked by the user application. For example, Messaging application is system application, when it is run by the user, SAMADroid does not hooks the Strace and when SubwaySurf, a user application downloaded from Google Play Store, runs SAMADroid immediately hooks the *Strace* with it. *Strace* only traces the system calls instead of intercepting them. SAMADroid is designed to trace only 10 specific system calls which includes system calls related to file operations and network access e.g. *open, ioctl, brk, read, write, close, sendto, sendmsg, recvfrom, recvmsg.*

As long as the user applications keeps running on the device, SAMADroid keeps on tracing the system calls of that application and generates the log file of system calls invoked by the user application. This log file contains the summary of system calls such as system call names, count for each system call, time utilized by the system calls and seconds of the time for which the system call is executed.

Also, the identifiers of that application are sent to the server so that it can immediately start the static analysis for that application. Application Identifiers includes the package name, version and market name from which the application is downloaded. After the log file is generated by the *Strace*, count for each system call is forwarded to the server. If any system call is not found in the log file, '0' value is forwarded for that system call. SQL server database is maintained at the server where it stores the system calls recorded for specific applications downloaded from different app stores.

2) SERVER END

In order to make SAMADroid resource efficient for memory and power constrained Android devices, detailed static analysis is performed at remote host. Figure 4 shows the workflow of static analysis. On receiving identifiers from SAMADroid client application, firstly server looks into its database of previously classified applications and searches the application using package name. If the application is found in the database, its classification report is forwarded to the client application. If the required application package name is not found, then its package is downloaded from the app-store. The installer name is different for different app-stores. This helps to get the application from that specific app market from which the user downloaded the application.

Once the application package is downloaded, static features are extracted from it. The application package is decompressed using *Android Asset Packaging Tool*. This module outputs the classes.dex file and also unpacks the manifest file. All the requested permissions, application components, filtered intents and hardware features used by the application are extracted from manifest file.

Baksmali tool, a tool for assembling and disassembling of applications, is used to disassemble the application code from classes.dex file. The output generated by this module contains

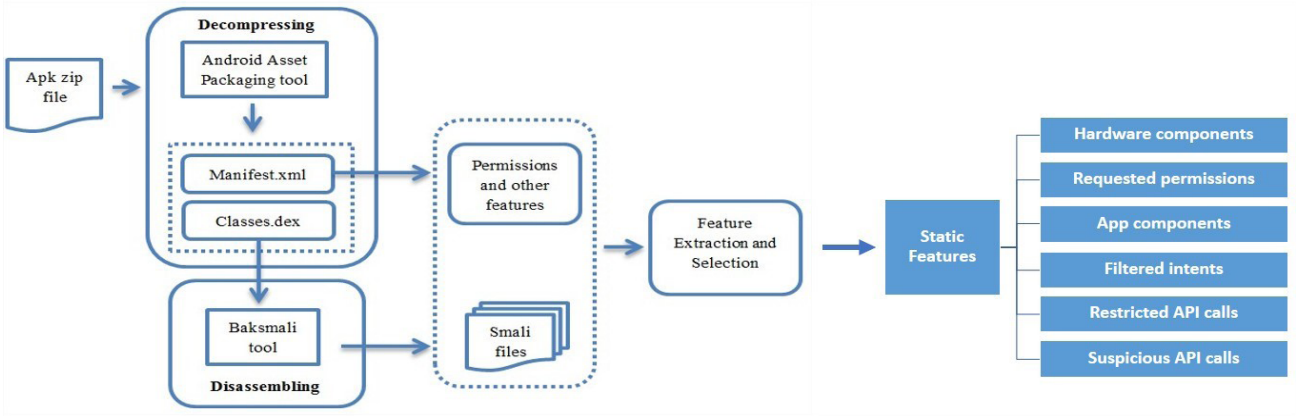

FIGURE 4. Static analysis flow diagram.

the smali files. These smali code files contains the Android application java code in smali language, which is assembly like language. From the smali code files, the suspicious API calls and restricted API calls are extracted.

Once the features are extracted, feature selection is performed. Arp *et al.* [3] of Drebin used maximum of the features in the static analysis and achieved high accuracy results using these features. Same sets of features are used in this research and different experiments are performed to get the most useful features among them in feature selection process. The features selected for the SAMADroid static analysis are hardware components, requested permissions, application components, filtered intents, restricted API calls and suspicious API calls.

3) EMBED EXTRACTED FEATURES INTO VECTOR SPACE

The extracted features, both static and dynamic, are mapped into vector space.

a: STATIC FEATURE VECTORS

For static feature vectors, we define a joint set S that comprises all observable strings contained in the 6 feature sets.

$$S := S_1 \cup S_2 \cup \ldots \cup S_6$$

It is ensured that elements of different sets do not collide by adding a unique prefix to all strings in each feature set. In our evaluation, the set S contains roughly 44,000 different features. Using the set S, an $|S|$-dimensional vector space is defined, where each dimension is either 0 or 1. An application x is mapped to this space by constructing a vector $\varphi(x)$, such that for each feature s extracted from x the respective dimension is set to 1 and all other dimensions are 0. Formally, this map φ can be defined for a set of applications X as follows:

$$\varphi : X \to \{0, 1\}^{|S|}$$

For example, a malicious application sends user's personal data such as pictures and contacts information to remote server. It needs to request certain permissions and hardware components. A corresponding vector $\varphi(x)$ for this application looks like this

$$\varphi(x) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \end{pmatrix} \begin{matrix} \cdots \\ \texttt{android.hardware.wifi} \\ \texttt{android.hardware.telephony} \\ \cdots \\ \texttt{SEND\_SMS} \\ \texttt{DELETE\_PACKAGES} \\ \cdots \end{matrix} \begin{matrix} \\ \\ \} S_1 \\ \\ \} S_2 \\ \\ \end{matrix}$$

b: DYNAMIC FEATURE VECTORS

Once the system call features are delivered to the server, these features are then embedded into vector space. The frequency of system call occurrence shows the behavior of application. Malicious applications invoke specific system calls more frequently than legitimate applications. Thus, system call frequency representation is used in order to capture such behavior of malware applications.

Let $M = \{m_1, m_2, \ldots, m_n\}$ represent the set of system calls, related to Android operating system, that are used as dynamic features such as:

$$\begin{aligned} m_1 &= open \\ m_2 &= ioctl \\ m_3 &= brk \\ m_4 &= read \\ m_5 &= write \\ m_6 &= close \\ m_7 &= sendto \\ m_8 &= sendmsg \\ m_9 &= recvfrom \\ m_{10} &= recvmsg \end{aligned}$$

The set M contains 10 different system calls, related to file operations and network access, as dynamic features. Using the set M, an $|M|$-dimensional vector space is defined, where

each dimension is either 0 or 1. An application x is mapped to this space by constructing a vector $\sigma(x)$, such that for each feature m extracted from x the respective dimension is set to 1 and all other dimensions are 0. Formally, this map σ can be defined for a set of applications X as follows:

$$\sigma : X \rightarrow \{0, 1\}^{|M|}$$

Following is an example of the system calls frequency values extracted from application x and embedded into vector space.

Let $x = com.kiloo.subwaysurf$

$\sigma(x) = 0, 6219, 0, 3391, 4531, 9, 334, 0\ 4913, 0$

c: MACHINE LEARNING INTELLIGENCE BASED DETECTION

Once the feature vectors are generated, both static feature vector and dynamic feature vector, they are given as input to the machine learning tool for classification. Machine learning Intelligence is used for automatically learning a separation between malicious and benign applications. The application of machine learning spares us from manually constructing detection rules for the extracted features. *Weka tool, v.3.6*, [66] is used for machine learning classification.

SAMADroid uses *Linear Support Vector Machine* (*SVM*) machine learning classifier for detection of malicious applications. *SVM*, trained on the Drebin's dataset of malicious applications and legitimate applications, is applied on both the vectors for a particular application, static feature vector and dynamic feature vector. SVM classifies the application as legitimate or malware.

As SVM is applied on static and dynamic feature vectors separately, there may be three possibilities:

1) Application is classified as legitimate for both static and dynamic feature vectors.
2) Application is classified as malicious for both static and dynamic feature vectors.
3) There exists contradiction in the classification results for the two analyses, e.g. Application is classified as legitimate for static feature vector and malware for dynamic feature vectors and vice versa.

Thus, the final classification decision is taken as follows:

1) Application is classified as 'legitimate' if both the static analysis and dynamic analysis results shows that application is legitimate.
2) Application is classified as 'malware' if classification results for both analysis declares the application as malicious.
3) Application is classified as 'risky' if one of the two analyses declares application as malicious and the other declares the application as legitimate.

These classification results are forwarded to the client application where the notification is generated for the user if application is malicious or risky.

4) APPLICATION BEHAVIOR EXPLANATION

Once the classification results are produced at server, they are communicated to the client application, running at real Android device. SAMADroid notifies the users about application's behavior, under examination, without interrupting other activities of user. Also, it provides the explanation to the user about application, for the awareness of users about applications behavior. In other words, this application not only detects the legitimate and malicious behavior of application but also provides sufficient information to the Android users about application's behavior. The application information includes the application name, package name, permissions, services and hardware features used by the application, version number, install date of application and also the date when it was last updated.

Users can view the details of applications in order to make decision of whether to continue using application or uninstall the application

IV. EXPERIMENTATION & RESULTS

In this section, series of experiments are discussed that were carried out during the development of SAMADroid and the results obtained from these experiments. Different machine learning classifiers are used in order to obtain high detection accuracy and also their results are evaluated. There are many evaluation parameters for machine learning classifiers on the basis of which performance of classifiers can be evaluated. In this research, true positive rate, false positive rate and accuracy are used as evaluation metrics for accurate detection of malwares.

1) *True positive rate or recall:* It is defined as the ratio of positive instances correctly classified among all available positive instances.

$$True\ Positive\ Rate = \frac{True\ Positive}{Total\ Positives}$$

In this research, malicious applications in dataset are termed as positive instances because we are interested in the detection of malicious applications. Thus, true positive rate is the ratio between the number of malicious applications correctly classified as malicious and total number of malicious applications.

2) *False positive rate:* It is defined as the ratio of negative instances incorrectly classified as positive instances over total number of negative instances.

$$FalsePositiverate = \frac{FalsePositive}{TotalNegatives}$$

Here, false positive rate is the ratio between number of legitimate applications incorrectly classified as malicious and total number of legitimate applications.

3) *Accuracy:* Accuracy is defined as the total number of instances correctly classified, both positive and negative, among all the available instances.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TotalInstances}$$

Where,

True Positives = number of malware applications classified as malware

False Positives = number of legitimate applications classified as legitimate

Thus, Accuracy is the ratio between sum of correctly classified legitimate and malicious applications and total legitimate and malware applications.

Besides malware detection accuracy, efficiency is also measured in terms of performance overhead, caused by SAMADroid application on Android device, memory consumption in terms of space required by the application on the device and power consumption.

A. DATASET

For evaluation of detection performance of proposed system, Drebin's dataset [3] is used. It is claimed to be the largest dataset of real malware applications collected from Google Play Store, Chinese App stores, Russian App markets and other app sources such as forums, blogs and websites. Also, the dataset contains the applications from Malware Genome Project [49]. Another reason for using this dataset in evaluation is that in training and testing partitions of dataset, malware and legitimate applications are distributed in such a way that it avoids the overfitting of classifier. This feature of Drebin's dataset helps the classifiers to achieve high detection rate on test set.

B. STATIC ANALYSIS EXPEIRMENTS

For the sake of effectiveness of static analysis module, in SAMADroid, against malicious apps, an experiment is performed to identify the most useful features. In Drebin, authors have used maximum of the static features which they have grouped into eight sets.

S_1 : Hardware components
S_2 : Requested permissions
S_3 : App components
S_4 : Filtered intents
S_5 : Restricted API calls
S_6 : Used permissions
S_7 : Suspicious API call
S_8 : Network Addresses

During feature selection phase, it is observed that used permissions, in maximum of the applications, are the subset of ones that are requested by the applications. As requested permissions feature set is already selected so the used permissions feature set is dropped. Also, the network addresses used in application are either the addresses of that specific application location at google play store or the address of remote server which hosts the application. Thus, network addresses found, in any application, are different for each application developed by different developers and there is no any standardized way of identifying the remote host network addresses to be malware or legitimate. On the basis of this fact, the network addresses feature set is also dropped. The remaining six feature sets are used and for static analysis.

TABLE 2. Detection performance comparison with Drebin.

Malware Detection System	True Positive Rate (%)	False Positive Rate (%)
SAMADroid	98.5	0.5
Drebin	94	1

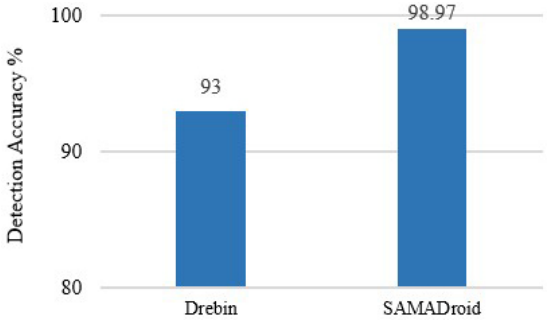

FIGURE 5. Performance comparison with Drebin.

An experiment is performed to check that whether the selected six feature sets can generate high accuracy than the eight feature sets. Thus, all the six sets of static features are extracted from malicious and non-malicious applications and embedded into vector space.

In the first experiment, *Linear SVM* machine learning classifier is applied on the dataset. The dataset is randomly split into known partition (66%) and unknown partition (34%). The known partition is used for the training of classification model and unknown partition is used for testing. Same process is repeated 10 times and computed the average of the obtained results for each run. The average accuracy achieved is 98.97% at a false positive rate of 0.005 or 0.5%. Table 2 reports the comparison of average true positive rate and false positive rate achieved by dataset of eight feature sets, used by Drebin, and dataset of six feature sets, used by SAMADroid.

Figure 5 depicts the comparison of average malware detection accuracy achieved by SAMADroid and Drebin, which used the same dataset but different number of feature sets. It can be observed that classifier learned on dataset of selected six feature sets, extracted from applications, can detect malicious applications more accurately than the dataset of eight feature sets. Also, the false alarm rate of SAMADroid is low i.e. 0.5%, in comparison to that of Drebin.

In order to test the performance of other classifiers on the dataset of six feature sets, second experiment is performed. In this experiment, different machine learning algorithms such as *Random Forest, Decision tree* and *Naive Bayes* are applied on dataset. These classifiers are applied on the dataset with random percentage split where 66% data is used as training set and 34% data is used for testing. Same process is applied 10 times for each classifier. Table 3 reports the average true positive rate, false positive rate and accuracy values achieved by *Random Forest, Decision Tree* and *Naive Bayes* classifiers.

TABLE 3. Comparison between different machine learning classifiers on static features.

Classifier	True Positive Rate (%)	False Positive Rate (%)	Accuracy (%)
SVM	98.5	0.5	98.97
Random Forest	80.6	0.03	99.07
Decision Tree	81.3	0.5	98.56
Naive Bayes	81.1	7.8	91.6

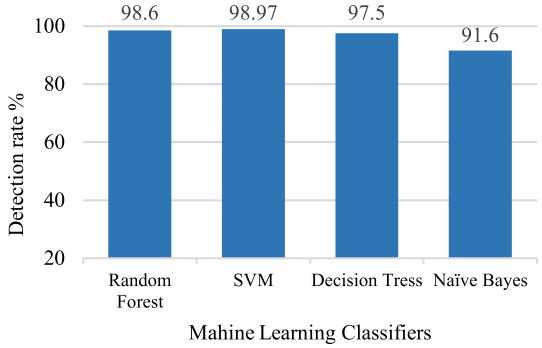

FIGURE 6. Performance comparison of different classifiers on static features.

TABLE 4. Comparison between different machine learning classifiers on dynamic features.

Classifier	True Positive Rate (%)	False Positive Rate (%)	Accuracy (%)
SVM	0.79	0.1	82.76
Random Forest	0.79	0.1	82.76
Decision Tree	0.79	0.3	72.41
Naive Bayes	0.79	0.5	62.07

It is observed that *Random Forest* achieves highest accuracy 99.07% at very negligible false positive rate of 0.03%. Although *Random Forest* provides highest accuracy in comparison to *SVM* but at the cost of low true positive rate. *SVM* provides the highest true positive rate which shows the potential of a classifier to detect the malicious application accurately. Figure 6 illustrates the visual representation of comparison between different machine learning classifiers.

C. DYNAMIC ANALYSIS EXPERIMENT

In this experiment, SAMADroid is evaluated on the basis of system calls. Feature vectors of system call frequency are given as input to the machine learning tool, *Weka*. Different machine learning classifiers such as *Random Forest, Decision tree* and *Naive Bayes* are applied on the system calls dataset. 5-fold cross validation is applied on the dataset for each classifier and compared the results. Table 4 and Figure 7 reports the true positive rate, false positive rate and accuracy obtained from these classifiers. *SVM* and *Random Forest* gives highest accuracy with lowest false positive rate.

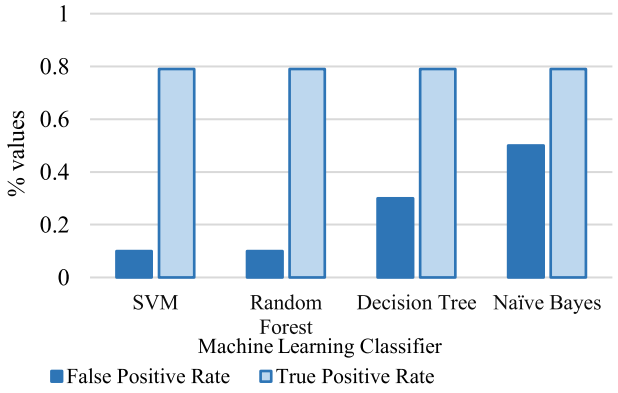

FIGURE 7. Comparison between different machine learning classifiers on dynamic features.

TABLE 5. Performance overhead caused by SAMADroid.

Test	SAMADroid not running	SAMADroid running	Overhead
Total Score	7566	7522	0.6 %
CPU	19723	19620	0.5 %
Memory	5304	5206	1.8 %
I/O	10043	10023	0.2 %
2D Graphics	500	500	0 %
3D Graphics	2259	2259	0 %

D. PERFORMANCE OVERHEAD

This section explains two experiments that are performed to determine the performance overhead caused by SAMADroid on real Android device. Firstly, the performance of Android device is observed through Benchmark tool before and after running SAMADroid. This experiment provides the overhead caused by the SAMADroid client application while running on the real Android device. Secondly, it provides the performance overhead of SAMADroid with MADAM [47]. Results shows that performance overhead caused by MADAM is high in comparison to SAMADroid.

1) PERFORMANCE OVERHEAD BY CAUSED BY SAMADROID

In this experiment, performance overhead of SAMADroid is measured through a standard benchmark tool, i.e. the *Quadrant Standard Edition Application* [67], which is distributed through Google Play [68]. It performs benchmark tests which cover the CPU, Memory, I/O operations, 2D graphics and 3D graphics in order to measure the performance of the device. Total score is the average of all the performance scores obtained from the tests performed on processor, memory, I/O, 2D and 3D. Benchmark tests have been performed on Samsung Galaxy Grand Prime (CPU Quad-core 1.2 GHz Cortex-A53, RAM 1GB). The device runs Android 5.1 Lollipop. Table 5 and Figure 8 shows the average results obtained from the five experiments performed when the SAMADroid was running on the device and when it was not. It is observed that the benchmark values decreased after running SAMADroid on the device which reflects the

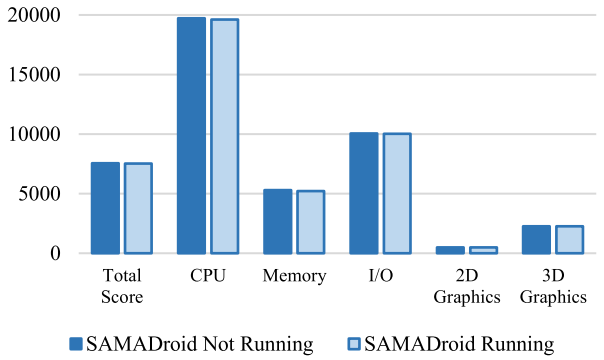

FIGURE 8. Performance degradation caused by SAMADroid.

TABLE 6. Comparison of overhead caused by SAMADroid with other related frameworks.

Antimalware System	Overall Performance Overhead	CPU Overhead	Memory Overhead	I/O Overhead
SAMADroid	0.6 %	0.5 %	1.8 %	0.2 %
MADAM	1.4 %	0.9 %	9.4 %	4.0 %

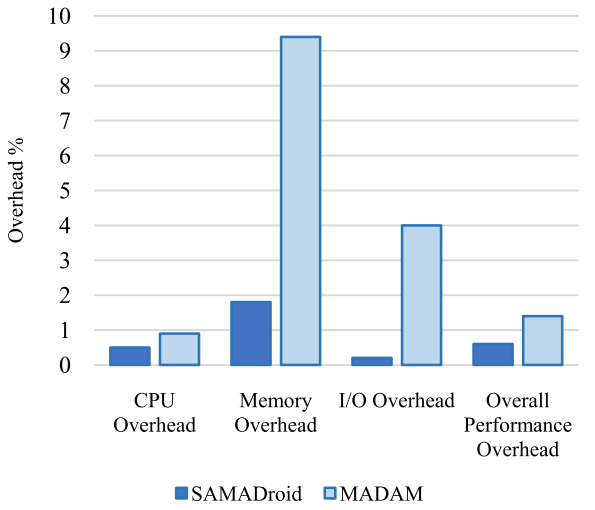

FIGURE 9. Comparison of SAMADroid performance overhead with MADAM.

decrease in performance of the device. This performance degradation is computed as percentage overhead between the performance before and after running SAMADroid. The overall performance impact of SAMADroid on the system is 0.6%.

2) PERFORMANCE OVERHEAD COMPARISON WITH MADAM

In this experiment, the performance overhead caused by SAMADroid and MADAM are compared. Table 6 and Figure 9 explains the comparison of performance overhead between SAMADroid and MADAM. It is observed that overall performance overhead in SAMADroid is low in comparison to MADAM i.e. 0.6% which is acceptable for smartphone users. Also, in SAMADroid, memory overhead is reduced to 1.8% in comparison to MADAM. This

TABLE 7. Comparison of device memory required for different security applications.

Applications	Memory (MB)
SAMADroid	15.07
Avast Mobile Security	62.32
AVG AntiVirus	52.74
Avira	29.09
Kaspersky Internet Security	64.57
McAfee Security	45.76
360 Security	56.0

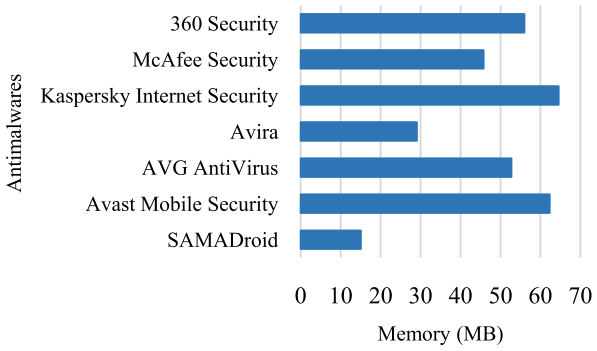

FIGURE 10. Memory required for security applications.

is because in MADAM, classifier requires training set in the memory which causes 9.4% memory overhead while SAMADroid performs classification at remote host which reduces the memory overhead at Android device. I/O overhead caused by SAMADroid is also low in comparison to MADAM.

E. RESOURCE CONSUMPTION

This section discusses about the experiments performed to evaluate the SAMADroid on the basis of resource consumption. These experiments are aimed to determine the efficiency of SAMADroid in comparison to the other existing antimalware applications for Android operating system. Results obtained from these experiments shows that SAMADroid consumes low resources, i.e. memory and power, in comparison to other detection systems and is more suitable for resource constrained Android devices.

1) MEMORY CONSUMPTION

In this experiment, SAMADroid is evaluated on the basis of device memory consumed by the client application. Memory consumption of SAMADroid client application is compared with some famous antivirus applications provided for the mobile apps such as Avast Mobile Security [69], AVG AntiVirus [70], Avira [71], Kaspersky Internet Security [72], McAfee Security [73] and 360 Security [74], [75]. Table 7 and Figure 10 shows the memory required for each application on the Android device before any scanning and detection process. It can be observed that SAMADroid is the light weight malware detection application for resource constrained Android devices. It consumes very low mem-

TABLE 8. Power consumption by different antimalwares on android device.

Applications	Power (mW)
SAMADroid	114
Avast Mobile Security	858
AVG AntiVirus	1287
Avira	1500
Kaspersky Internet Security	677
McAfee Security	800
360 Security	1404

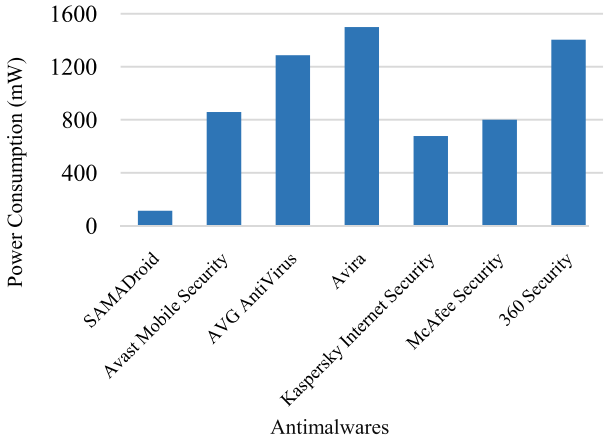

FIGURE 11. Illustration of power consumed by different antimalwares.

ory space in comparison to the other malware detection applications.

2) POWER CONSUMPTION

Another evaluation parameter used is power consumption. This experiment is performed to measure the power consumption of SAMADroid on real Android device through *App Tuneup Kit* [75] and compare it with other related real applications such as *Avast Mobile Security, AVG AntiVirus, Avira, Kaspersky Internet Security, McAfee Security* and *360 Security*. Table 8 and Figure 11 describes the power consumed by different antimalware applications for the interval of 5 minutes. Results shows that power consumed by SAMADroid is moderate in comparison to the other Antimalwares. This happens because SAMADroid monitors only running applications. While the other anti-malwares scan all the applications either they are running in the background or not. Another reason for the low power consumed by our security system is its distinguishing feature that it only analyzes the user applications and does not scan the system applications. As system applications are added in the device by the device manufacturers and are not malicious, due to this reason SAMADroid does not scan system applications. On the other hand, other anti-malwares scan all the user and system applications because of which they consume more battery power. Also, SAMADroid performs static analysis at server and dynamic analysis on the device while the other anti-malwares listed in the Table 8 perform only static analysis and do not analyze the runtime behavior of user applications.

V. DISCUSSION & FUTURE WORK

Although SAMADroid provides high detection accuracy at low resource consumption but it has some limitations too. First of all, the whole system is dependent on server communication. Classification of applications is performed at server and the results of which are delivered to the Android device for the sake of security provision. No malicious behavior detection is performed at local host i.e. Android device. Thus, if the network link gets down or congestion occurs at channel due to which Android device cannot communicate with the server then the performance of SAMADroid will be reduced. Secondly, Drebin's dataset of malicious applications was used for training the classification model which does not contain the most recent variants of malware types. In future, we aim to enhance the malware dataset for SAMADroid, including the recent malwares so that SAMADroid effectively secures the Android applications against recent malware applications.

VI. CONCLUSION

This research work is based on the development of a malware detection system that can detect the malwares on the Android devices while ensuring the low resource consumption. In this research, we thoroughly investigated many of the existing malware detection and prevention techniques, developed during the period of 7 years, 2010 to 2016. Based on the benefits and limitations of existing antimalware techniques, we formulated the problem that existing research lags in detection of Android malwares accurately while ensuring the low consumption of hardware resources of Android devices. Thus, we proposed 3-level hybrid malware detection model for Android operating system. To the best of our knowledge, there does not exist any 3-level hybrid malware detection system. Thus, SAMADroid is a novel malware detection model which combines the benefits of static analysis, dynamic analysis and machine learning Intelligence. Also, it operates both on local host and remote host to achieve the resource efficiency. SAMADroid client application is developed for Android devices. It performs dynamic analysis on the device and communicates with the server for static analysis and detection results. Remote server performs the static analysis and machine learning based detection. Through extensive experimentation, we have shown that SAMADroid performs better than Drebin for static analysis. It is also observed that SAMADroid causes very negligible performance overhead on the Android device. SAMADroid is also efficient in terms of hardware resource usage.

REFERENCES

[1] (2017). *Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016*. Accessed: Apr. 28, 2017. [Online]. Available: http://www.gartner.com/newsroom/id/3609817

[2] *Trend Micro Q2 Security Roundup Report | Androidheadlines.Com*. Accessed: Dec. 8, 2015. [Online]. Available: http://www.androidheadlines.com/2015/08/trend-micro-q2-security-roundup-report.html

[3] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Symp. Netw. Distrib. Syst. Secur. (NDSS)*, 2014, pp. 23–26.

[4] T. Bläsing, 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., Malware*, Oct. 2010, pp. 55–62.

[5] *[Utility][Tool][Windows] Baksmali/Smali Ma... | Android Development and Hacking*. Accessed: Dec. 22, 2015. [Online]. Available: http://forum.xda-developers.com/showthread.php?t=2311766

[6] 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. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2012, no. 2, pp. 5–8.

[7] *Android Apps on Google Play*. Accessed: Aug. 30, 2016. [Online]. Available: https://play.google.com/store?hl=en

[8] *Android Games Android Application Android Phones__Excellent Software Download Center Official Website-Billion Market*. Accessed: Aug. 30, 2016. [Online]. Available: http://www.eoemarket.com/

[9] *Machine Front Network—Technology News, all in the Machine Front*. Accessed: Aug. 30, 2016. [Online]. Available: http://www.gfan.com/

[10] *DexClassLoader | Android Developers*. Accessed: Aug. 30, 2016. [Online]. Available: https://developer.android.com/reference/dalvik/system/DexClassLoader.html

[11] *Update: Security Alert: DroidDreamLight, New Malware From the Developers of DroidDream | Lookout Blog*. Accessed: Aug. 30, 2016. [Online]. Available: https://blog.lookout.com/blog/2011/05/30/security-alert-droiddreamlight-new-malware-from-the-developers-of-droiddream/

[12] 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.

[13] *API Monitor: Spy on API Calls and COM Interfaces (Freeware 32-Bit and 64-Bit Versions!) | Rohitab.Com*. Accessed: Aug. 22, 2016. [Online]. Available: http://www.rohitab.com/apimonitor

[14] *DroidBox*. Accessed: Aug. 22, 2016. [Online]. Available: https://github.com/pjlantz/droidbox

[15] S. Chang, "APE: A smart automatic testing environment for Android malware," Dept. Comput. Sci. Inf. Eng., Nat. Taiwan Univ., Taipei, Taiwan, Tech. Rep., 2013.

[16] A. Ng, *Support Vector Machines for Machine Learning*. Stanford, CA, USA: Stanford Univ., 2008

[17] *LIBSVM—A Library for Support Vector Machines*. Accessed: Aug. 17, 2016. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[18] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jul. 2013, pp. 1666–1671.

[19] X. Wang, Y. Yang, Y. Zeng, C. Tang, J. Shi, and K. Xu, "A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection," in *Proc. 6th Int. Workshop Mobile Cloud Comput. Services*, Sep. 2015, pp. 15–22.

[20] *Android Aapt—eLinux.Org*. Accessed: Aug. 13, 2016. [Online]. Available: http://elinux.org/Android_aapt

[21] *CuckooDroid Book—CuckooDroid V1.0 Book*. Accessed: Aug. 19, 2016. [Online]. Available: http://cuckoo-droid.readthedocs.io/en/latest/

[22] *10 Open Source Mobile Test Automation Tools*. Accessed: Aug. 19, 2016. [Online]. Available: http://www.testingexcellence.com/open-source-mobile-test-automation-tools/

[23] S. R. Gunn. (1998). *UNIVERSITY OF SOUTHAMPTON Support Vector Machines for Classification and Regression*. Accessed: May 16, 2017. [Online]. Available: http://ce.sharif.ir/courses/85-86/2/ce725/resources/root/LECTURES/SVM.pdf

[24] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class SVM for anomaly detection," in *Proc. Int. Conf. Mach. Learn.*, Nov. 2003, pp. 3077–3081.

[25] K. Patel and B. Buddadev, "Detection and mitigation of Android malware through hybrid approach," in *Security in Computing and Communications*. Cham, Switzerland: Springer, 2015, pp. 455–463.

[26] *Control the Emulator from the Command Line | Android Studio*. Accessed: Aug. 13, 2016. [Online]. Available: https://developer.android.com/studio/run/emulator-commandline.html

[27] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "DroidKin: Lightweight detection of Android apps similarity," in *Security and Privacy in Communication Systems* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 152. Cham, Switzerland: Springer, 2015, pp. 436–453.

[28] M. Parkour. *Contagio*. Accessed: Aug. 13, 2016. [Online]. Available: http://contagiodump.blogspot.in/

[29] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Comput. Sci.*, vol. 46, pp. 804–811, Jan. 2015.

[30] *Weka 3—Data Mining With Open Source Machine Learning Software in Java*. Accessed: Dec. 16, 2015. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

[31] A. Andrew, N. Cristianini, and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[32] A. Andrew, "An introduction to support vector machines and other kernel-based learning methods," in *Kybernetes*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[33] D. Dittman, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "Random forest: A reliable tool for patient response prediction," in *Proc. BIBMW*, Nov. 2011, pp. 289–296.

[34] 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.

[35] *7-Zip*. Accessed: Jun. 10, 2017. [Online]. Available: http://www.7-zip.org/

[36] *AXMLPrinter2 | Android Tales*. Accessed: Jun. 10, 2017. [Online]. Available: http://android.amberfog.com/?tag=axmlprinter2

[37] *TinyXML Download | SourceForge.Net*. Accessed: Jun. 10, 2017. [Online]. Available: https://sourceforge.net/projects/tinyxml/

[38] 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. (ICCE)*, Jan. 2016, pp. 155–156.

[39] *Apktool—A Tool for Reverse Engineering Android Apk Files*. Accessed: Aug. 13, 2016. [Online]. Available: https://ibotpeaches.github.io/Apktool/

[40] *App Manifest | Android Developers*. Accessed: Aug. 13, 2016. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro.html

[41] 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. (ASIA CCS)*, 2013, pp. 329–334.

[42] *Strace Download | SourceForge.Net*. Accessed: Dec. 22, 2015. [Online]. Available: http://sourceforge.net/projects/strace/

[43] *'Pea Pod' Official Website*. Accessed: Aug. 13, 2016. [Online]. Available: https://www.wandoujia.com/

[44] *Naive Bayesian*. Accessed: Aug. 13, 2016. [Online]. Available: http://www.saedsayad.com/naive_bayesian.htm

[45] *KNN Classification*. Accessed: Aug. 13, 2016. [Online]. Available: http://www.saedsayad.com/k_nearest_neighbors.htm

[46] L. Kozma, *K Nearest Neighbors Algorithm (kNN)*. Espoo, Finland: Helsinki Univ. of Technol. Press, 2008.

[47] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based android malware detection and prevention," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2016.

[48] P.-E. Danielsson, "Euclidean distance mapping," *Comput. Graph. Image Process.*, vol. 14, no. 3, pp. 227–248, Nov. 1980.

[49] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.

[50] *Contagio Mobile*. Accessed: May 17, 2017. [Online]. Available: http://contagiominidump.blogspot.com/

[51] *VirusShare.Com*. Accessed: May 17, 2017. [Online]. Available: https://virusshare.com/

[52] *VirusTotal—Free Online Virus, Malware and URL Scanner*. Accessed: May 17, 2017. [Online]. Available: https://www.virustotal.com/

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

[54] *Google Code Archive—Long-Term Storage for Google Code Project Hosting*. Accessed: Aug. 17, 2016. [Online]. Available: https://code.google.com/archive/p/androguard/

[55] *Contagio Mobile*. Accessed: Aug. 17, 2016. [Online]. Available: http://contagiominidump.blogspot.tw/

[56] *Downloading Free Apks From Google Play and Alternate Markets to Your Desktop. LID: Lost In Droid*. Accessed: Aug. 17, 2016. [Online]. Available: https://machiry.wordpress.com/2012/10/01/downloading-apks-from-google-play-to-your-desktop/

[57] A. Rodríguez-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. (CONIELECOMP)*, Feb. 2016, pp. 54–61.

[58] 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. IEEE Symp. Comput. Intell. Cyber Secur. (CICS)*, Dec. 2014, pp. 1–8.
[59] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905–912, Mar. 2015.
[60] 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.
[61] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digit. Invest.*, vol. 13, pp. 1–14, Jun. 2015.
[62] A. Agresti, "Building and applying logistic regression models," in *Categorical Data Analysis*, 2nd ed. Hoboken, NJ, USA: Wiley, 2003.
[63] F. Pampel, *Logistic Regression: A Primer*, vol. 132. Newbury Park, CA, USA: Sage, 2000.
[64] D. W. Hosmer, Jr., S. Lemeshow, and R. Sturdivant, *Applied Logistic Regression*. Hoboken, NJ, USA: Wiley, 2013.
[65] M. Sun, X. Li, J. C. S. Lui, R. T. B. 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.
[66] *Weka 3—Data Mining With Open Source Machine Learning Software in Java*. Accessed: May 17, 2017. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
[67] *Quadrant Standard Edition—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.aurorasoftworks.quadrant.ui.standard
[68] *Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store
[69] *Mobile Security & Antivirus—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.avast.android.mobilesecurity
[70] *AVG AntiVirus FREE for Android - Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.antivirus
[71] *Avira Antivirus Security—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.avira.android
[72] *Kaspersky Antivirus & Security—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.kms.free
[73] *McAfee Mobile Security—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.wsandroid.suite
[74] *360 Security—Antivirus Boost—Android Apps on Google Play*. Accessed: May 16, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.qihoo.security
[75] A. Mehmood, A. Khanan, A. H. H. M. Mohamed, and H. Song, "ANTSC: An intelligent Naïve Bayesian probabilistic estimation practice for traffic flow to form stable clustering in VANET," *IEEE Access*, to be published. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7994591/, doi: 10.1109/ACCESS.2017.2732727.

SABA ARSHAD received the B.S. degree (Hons.) in computer science from Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan, and the M.Sc. degree in computer science from the COMSATS Institute of Information Technology, Islamabad, Pakistan. Her research interest includes smart device security, malware analysis and detection, machine learning, intelligent traffic system, distributed computing, and social aware delay tolerant networks.

MUNAM A. SHAH received the B.Sc. and M.Sc. degrees in computer science from the University of Peshawar, Pakistan, in 2001 and 2003, respectively, the M.S. degree in security technologies and applications from the University of Surrey, U.K., in 2010, and the Ph.D. degree from the University of Bedfordshire, U.K., in 2013. Since 2004, he has been a Lecturer with the Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan. He has authored over 50 research articles published in international conferences and journals. He is an HEC Approved Supervisor. His research interests include MAC protocol design, QoS, and security issues in wireless communication systems.

ABDUL WAHID received the Ph.D. degree from Kyung pook National University, South Korea. He is currently an Assistant Professor with the Department of Computer Science, CIIT, Islamabad, Pakistan. He is also Reviewer and a TPC member of many conferences and journals. His research interests include, but are not limited to, vehicular ad hoc network, wireless sensor network, underwater wireless sensor network, cyber physical systems, software defined networking, and information-centric networking. He is currently an Associate Editor of the IEEE Access.

AMJAD MEHMOOD received the Ph.D. degree in wireless networks from the Kohat University of Science and Technology, Kohat, Pakistan, in 2014. He held a virtual post-doctoral position at the University of Virginia, USA. He is currently holding a post-doctoral position with the Guangdong Provincial Key Laboratory on Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology, Maoming, China. In 2003, he joined the Kohat University of Science and Technology, where he is currently a Senior Faculty Member and the Coordinator of M.S./Ph.D. program. He has authored over 42 academic articles in peer-reviewed international journals and conferences around the world. His is research interest include cyber physical systems, IoT, connected vehicles, wireless communications and networking, optical communications and networking, smart grid communications and networking, security issues in wireless networks, big data, cloud computing, and fault diagnosis in WSNs. His research was supported by the Guangdong University of Petrochemical Technology, Maoming, China. He supervised many students of B.C.S., M.C.S., M.S., and Ph.D. in the above-mentioned interests. He has also been a part of reviewing and organizing different workshops, seminar, and training sessions on different technologies. Furthermore, he has served as a TPC Reviewer and the Demo Chair for numerous international conferences, including CCNC, SCPA, WICOM, INFOCOM, and SCAN. Additionally, he is a reviewer or an Associate Editor for many peer-reviewed international journals.

HOUBING SONG (M'12–SM'14) received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, USA, in 2012. In 2017, he joined the Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL, USA, where he is currently an Assistant Professor and the Director of the Security and Optimization for Networked Globe Laboratory. He served as the Faculty Member with West Virginia University from 2012 to 2017. In 2007, he was an Engineering Research Associate with the Texas A&M Transportation Institute. He has authored over 100 articles. His research interests include cyber-physical systems, cybersecurity and privacy, Internet of Things, edge computing, big data analytics, connected vehicle, smart and connected health, and wireless communications and networking. He was the very first recipient of the Golden Bear Scholar Award, and received the Highest Faculty Research Award from West Virginia University Institute of Technology in 2016. He serves as an Associate Technical Editor for the *IEEE Communications Magazine*. He is the Editor of four books, including *Smart Cities: Foundations, Principles, and Applications*, (Wiley, Hoboken, NJ, USA, 2017), *Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications*, (Wiley-IEEE Press, Chichester, U.K., 2017), *Cyber-Physical Systems: Foundations, Principles, and Applications*, (Academic Press, Boston, MA, USA, 2016), and *Industrial Internet of Things: Cyber manufacturing Systems*, (Springer, Cham, Switzerland, 2016). He is a Senior Member of the ACM.

HONGNIAN YU has held academic positions with the Universities of Sussex, Liverpool John Moor, Exeter, Bradford, Staffordshire, and Bournemouth in the U.K. He is currently a Professor in computing with Bournemouth University. He has authored over 200 journal and conference research papers. He has extensive research experience in mobile computing, modeling, scheduling, planning, and simulations of large discrete event dynamic systems with applications to manufacturing systems, supply chains, transportation networks, computer networks and RFID applications, modeling and control of robots, mechatronics, and neural networks. He has graduated over 20 Ph.D./M.Phil. and MRes research students, is supervising eight Ph.D. students, and has examined over 20 Ph.D./M.Phil. students' theses as both internal and external examiner. He has held several research grants from the UK EPSRC, the Royal Society, EU, AWM, and from the industry. He was a recipient of the F. C. William Premium for his paper on adaptive and robust control of robot manipulators by the IEE Council. He is a member of EPSRC Peer Review College. He has strong research collaboration with partners from China, France, Germany, Hungary, Italy, Japan, and Thailand.