mDroid : A Machine Learning Based Malware Identification System For Android

P M Dhanasree

Abstract—

The explosive growth of smart phones and wide acceptance of android brings a large scope for android mobile malwares. As the number of applications in android platform increases, number of malwares is also increasing rapidly. Since today's smartphones are a ubiquitous source of confidential datas, hackers are very much interested in targeting mobile phones. So security of mobile phones is a major issue. This paper presents a new machine learning based approach for detecting malwares in the android platform. We perform hybrid analysis over downloaded applications to specify whether it is of malicious behaviour or not.

Index Terms— Android Malware, APIs, Behaviors, Hybrid analysis, Information flow, Machine learning, System calls.

1 INTRODUCTION

In this epoch of wireless communication, smart phone is a vital part of our life. It is meant not only for communications but also performs almost all functions of a PC, which made the smart phones so popular. A life without a smart phone is too difficult for people of today's.

In the year of 2018, around 2.53 billon people in the world use smart phones[1].There is a lot of mobile OSs existing today which includes android, iOS, windows, kaiOS etc. As per International Data Corporation (IDC) market share of android phones in the 3rd quadrant of 2018 is 86.8%[2]**.** This shows the wide acceptance of Android OS among other mobile phone OSs.

Why it succeed? There are many reasons. Only some of the main reasons are mentioned here. First one, it is cheap and supported by many vendor hardware. Second, its open source nature and support of multiple platforms. Third reason is that it provides a large number of APIs. Framework for developing collaborative applications is the fourth reason.

Android application's security is a major concern since it is the largely used smart phone OS in the world. In January 2019 there are more than 2.5 million apps in Google Play store[3]. Additional to play store, android apps are also available through third party stores. Such stores provides more freedom to developers and faster speed to market. The approval processes are less strict than the Google Play Store and so approvals are processed much faster.

Google is trying to ensure security to applications in many ways. A well known example is the vetting process (Process of checking an application before uploading to Play Store).

Another one is Google's app security improvement program[4]. In this, Google scans the applications uploaded on Google Play for some known vulnerabilities and gives detailed information on the issues it detects. Also provides a guidance to fix them. Developers who fail to fix the problems within the deadline provided by Google cannot release future updates for their affected applications through Google Play Store.

Google recently introduced a new feature to make the users aware about application's security. For any applications that haven't gone through Google's application verification system, they project a new screen to alert users that the application they are about to use hasn't been verified yet and its their own risk to use that application. Users even have to tap in "continue" to move beyond the warning screen.

There is always a risk with third party stores. The apps in such stores may include malewares. So downloading a software application from a third-party app store may infects your smartphone or tablet with malicious software. Such malware could allow someone to take control of your device. It may provide hackers access to your contacts, passwords, and financial accounts.

There is a 40% increase in android malwares in 2018. Approximately, 3.2million malicious apps are identified by the end of 3rd quarter off 2018[5]. In general, as the review and supervision of Android applica-

- *P M Dhanasree, currently pursuing masters degree program in computer science and engineering in APJ Abdul Kalam University, India, E-mail: dhanasreepmohan@gmail.com*
- *Laiji George, Guide, assistant professor, AWH EC, Calicut, India . E-mail: laijivgeorge@gmail.com*

tions are inadequate, a reasonable chance exists that users will download malware. This malware can lead to monetary loss, information leakage, and other damages. Thus, the analysis of applications in android platform has become an important topic.

At present, there are mainly three analysis methods: static, dynamic and hybrid analysis. Static analysis, as it name suggests perform a statical analysis without executing the application being tested. It extracts necessary informations from source code and binary files and evaluates those informations to find malicious activities. In contrast dynamic analysis detects malicious behaviour by executing the applications.

Static and dynamic analysis, both have some disadvanatges. For examples, static analysis may face challenges of dynamic code loading and code obfuscation where dynamic have disadvantages such as path explosion and incomplete detection results. Hybrid, the more complicated one combines static and dynamic analysis.

Even though a variety of malware detection softwares are available, majority of them cannot show specific malicious behaviours. Moreover, some of them are based on the database of viruses, so it can identify only known malwares which are defined in the database and cannot identify unknown malware. Such malware detecting apps fails when a new malware is entered.

This work introduces a hybrid analysis technique to solve the above problems. An Android application to detect, analyze, and identify applications is implemented and evaluated using many sample applications. The key contributions of our paper are as follows:

1) A new approach for real-time behavior detection based on Android kernel by using kernel-level monitoring mechanisms.
2) A new approach to identify malwares by combining static and dynamic analysis which can identify both known and unknown malwares.
3) This approach uses a data-centric technique which make it capable for reconstructing the behavior of applications with less overhead.

The rest of this paper is organized as follows. In Section II, Related works are given. Section III & IV explains android system and risk with android system. Overview of the proposed system is in Section V. Section VI gives a detailed description of our mDroid system. Then, we present the discussion and evaluation in Section VII. Finally, Section VIII presents the conclusion.

2 RELATED WORKS

Lanzi et al. [6] characterizes the general interactions between benign programs and the OS and models the way in which benign programs access OS resources. Even though it possess minimal impact on the performance of testing machines it faces significant challenges due to the diverse nature of system calls invoked by different applications.Also, it can detect only the cases in which malicious code attempts to tamper the settings of other applications or the core OS itself.

In [7], William Enck et al. proposes an efficient, system-wide dynamic taint tracking and analysis system capable of simultaneously tracking multiple sources of sensitive data and provides real-time analysis by leveraging Android's virtualized execution environment. Taintdroid monitors downloaded, third party applications by tracking the flow of privacy sensitive data through third-party applications. It incurs only 14% performance overhead on a CPU-bound microbenchmark and imposes negligible overhead on interactive third-party applications.But it can be circumvented through leaks via implicit flows.

Droidscope [8] by Yan et al. is an android analysis platform that continues the tradition of virtualization-based malware analysis. It reconstructs both the OS-level and Java-level semantics simultaneously and seamlessly. DroidScope is built on top of QEMU and is able to reconstruct the OS-level and Java-level semantic views completely from the outside. It is enriched with the semantic knowledge and provides a set of APIs to help analysts implement custom analysis plugins. Low instrumentation overhead is its advanatage. But it is unable to obtain real behaviors of applications and are faced with problems such as antiforensic techniques.

A proactive scheme to spot zero-day Android malware without relying on malware samples and their signatures is proposed by Grace et al. [11]. It divides the potential risks into three categories: high-risk, medium-risk, and low-risk and performs a two-order risk analysis to assess the risks from existing apps for zero-day malware detection based on this risk classification. This scalable & fast approach doesn't consider real time behavior.

In [10] Zhang et al. proposes VetDroid, a dynamic analysis platform for generally analyzing sensitive behaviors in Android apps from a novel permission use perspective. It overcomes several key challenges to completely identify all permission sensitive behaviors and utilizes accurate permission use information during the runtime. VetDroid can find more information leaks than TaintDroid.

A machine learning based system for the detection of malware on Android device is proposed by Sahs et al. [11]. This uses an open source project, Androguard, to extract features from packaged Android

applications (APKs) and then uses these extracted features to train a One-Class Support Vector Machine, using the Scikit-learn framework. It has very low false negative rate but imited to permissions and CFGs of the input application.

Slicing Droids [12] is a Static Android Analysis Framework presented by Hoffmann et al. for android apps. Based on static backtracking it recognizes suspicious behavior patterns in an automated way. Although it works fast usage of encryption methods and obfuscation put a heavy burden on it. Also, Slicing Droids can't deal with informations that are available only at runtime.

Reina et al. [15] proposes a VM based dynamic system call-centric analysis based on the observation that malicious behaviors are achieved through the invocation of system calls. It dynamically observes the interactons between the android components and underlying Linux system and reconstructs higher level behaviour. However, attackers can easily modify an application for detecting whether it is running on a VM and then leak no datas at that time.

Dehghantanha et al. in [14] suggests data centric approaches for mobile security. Similarly, Choudhary et al. in [15] suggests hybrid analysis techniques for malware detection. A dynamic android gaming malware detection system based on system call analysis to classify malicious and legitimate game is presented in [16]. It is based on difference in system call pettern and system call frequency inhibited by legitimate and malware games. This approach does not require source code of the android application being tested. It can be easily be adopted and implemented.

In [17] Zhao et al. proposes a behavior based quick and accurate Android malicious detection scheme based on sensitive API calls. It goes through sensitivity score calculation and eigenvector creation. Then uses kNN classifier and the decision tree classifier for traning. The linear weighted sum method (LWSM) is used to calculate the final result. The disadvantage is that it does not focus on the function and class defined by the application developers.

RanDroid [18] is a machine learning-based malware detection system for Android platform. The system extracts requested permissions, vulnerable API calls along with the existence of app's key information such as: dynamic code, reflection code,native code, cryptographic code and database from applications. RanDroid is built on concept of static analysis and lacks dynamic inspection. The results of proposed system may vary with increasing size of training and testing data sample set

Li et al. in [19] uses fine grained deep neural network for android malware detection. It shows the detail malware families but doesn't consider the dynamic features of Android applications.

3 ANDROID SYSTEM

3.1 Components

As demonstrated in this document, the numbering for sections upper case Arabic numerals, then upper case Arabic numerals, separated by periods. Initial paragraphs after the section title are not indented. Only the initial, introductory paragraph has a drop cap.

Basic build blocks of an android application are called components. It makes intra as well as inter application communication possible through messaging. There are four types of components.

1) Activity - alludes a user screen or a user interface
2) Service – for performing long-running background tasks
3) Broadcast receiver - manages system or application generated events
4) Content provider - tackles access to a structured set of data

It can complete a task independently and can interact with each other to complete the task Communication between components of different applications are also possible.

3.2 Intents

It describes an operation to be performed by a component. Divided into two, explicit and implicit. If the intent knows which its target component is, such intents are called explicit intents. If the intent has no idea about its target components, such intents are categorized as implicit intent.

3.3 Intent Filter

Intent filter is used to publicize the capability of a component to perform a specific action. When an implicit intent request came, system compares the operation to be performed by intent with the publicized operation of components. If there is a match that particular component will be assigned as the target component for that implicit intent. Here the collaborating application does not know each other.

3.4 Manifest File

All the components of an application should be declared in a configuration file called manifest file. By exposing at least one of its components, an application can offer its services to other applications. This can be done by setting exposed component's exported flag attribute to true. If a component (activity, service, or broadcast receiver) contains intent filter(s), that component will be exported by default. For a content provider, if the application uses an API level less than 17 it will be treated as exported. Even though the entire components must be declared in manifest file, it is possible to declare a broadcast receiver components can in the source code also

4. RISK WITH ANDROID

4.1 Permission Based System

An external application need certain permissions from user to access system features, this is the concept of permission based system in android. When a user wants to install a third party application in his device, he should grant some set of permissions requested by the application before installation. If he/ she didn't approve the permissions, then the installation of the application will be cancelled automatically. This permission request and permission approval constitutes the permission based system.

Collaborative model makes android so acceptable for both users and developers. Simply, it is the system of accessing an application from another application. Consider an example. Suppose a user is using a social media application and he wants to capture an image and send it to a particular recipient. What he/ she generally do is described in steps from 1 to 7. 1) close the social media application, 2) opens the camera application, 3) capture photo & save to gallery, 4)close camera application, 5) again opens the social media application, 6) attach the photo from gallery, 7) send to the recipient. With the collaborative model, the user can access the camera application without closing facebook application and can send the captured photo directly to the user instantaneously.

4.2 Security In Android

There are two levels of security implementation in Android. One is at system level and another is at application level. In the system level Sandboxing technique is used. Each application is treated as a separate entity with its own set of resources and datas. No one other process can interrupt to the area of another process. But problem with this technique is that this boundary is virtual and can be broken at the time of inter application communication. In the application level Permission Based System is used. Components that participate in inter process communication can be protected with permissions. The service provider application will declare permission and protect its exposed component with this protection. If an application wants to use the services it should acquire the permission.

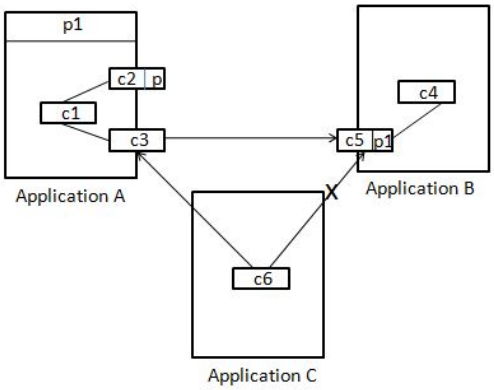

Fig 1. An example illustration of collaborative model and its security

In Fig. 1 there are three applications A, B and C. Component c2 and c3 of Application A and c5 of Application B are exposed. Component c2 is protected with permission p and c5 is protected with permission p1 while component c3 is not protected. Application A holds the permission p1. Suppose Application C is a malicious application trying to access component c5 of Application B. Since Application C have not acquired permission p1, required to access c5 it cannot access c5 directly. However Application C can access c5 indirectly through component c3 of Application A. This is possible because c3 is a public component without any protection. Application C can access c3 without any restriction (where as it cannot access c2 because it is protected with permission p) and since Application A has already acquired the permission p1, c3 can access c5 component of Application B. So by accessing c3, Application C can indirectly access c5.

This example shows that every component of an application should be protected in order to ensure security. Leaving at least one component unprotected is risky. An unprotected application may put a well protected application at risk.

4.3 Four Threat Levels Of Android Permissions

- Normal- minimum security risk permission, granted automatically by the system
- Dangerous-higher security risk permission, requires explicit user consent
- Signature- used by applications signed with same certificate, granted by the system after comparing the certificates & found a match
- Signature Or System-similar to signature level except that it can also be granted to applications reside in the system image

4.4 Permission Based Attack

Suppose a user trying to install an application. Then the application will ask certain permissions and installation will proceed only if the user grants the permission. Some applications may ask for extra permissions

that may include access to almost all features of device and user data. This is an attack itself called permission based attack. It goes through the following procedure

1. The developer develops the application.
2. Uploads the application to application store.
3. User downloads the application.
4. Before the installation, application asks for certain set of permissions.
5. User can either reject the permissions and cancel the installation or he can grant the permissions and proceed to installation
6. Usually, user grants the permissions with or without understanding the list of permissions asked.
7. Once the application is installed, application/ developer will get access to device features and user data.

Sometimes, after a while an installed app may prompt for update. If device is running in Android 5.0 or lower, this update will be installed automatically. For higher versions, there will be an extra permission request to be granted by the user [20].

In permission based systems extra permissions are also a way to insecurity. Given example shows how an extra permission make the system prone to attacks. Suppose user A installs an image to pdf converter application with the following permission approval.

1. To read gallery
2. To access camera
3. To read contacts
4. To get complete network access.

It is clear that the 3rd and 4th permissions asked here are extra permissions. By installing this application, it can read contacts in user A's phone and can transfer it to server or to any other locations without the knowledge of user A. So the security breaks.

Also Permission Re-Delegation attack, which is an attack in which less privileged application misuses privileged applications to perform the malicious task can be occurred[21].

4.5 Risk Of Apis & Information Flows

A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service are called as APIs. In general, it is a set of precisely defined methods of communication among various components. Applications have two ends generally. Front-end and back-end. Front-end deals with UI/UX, creating the components which is directly visible to the user and backend deals with data storages, data manipulation etc. The front end and back-end is linked by the APIs. Front-end will collect the required datas and call the required APIs for data manipulation and storage.

APIs bring a lot of adavantages in developing an android application. It make the development procedure speedy and easier. Although, it has some security risks associated with them. Simply, it is a great door way for hackers to collect confidential datas and to perform malicious activities.

Information flow is simply the movement of information. An application, which does not exhibit any malicious behaviour explicitly can track the datas and informations in the device in which it is installed and send over network to a server or store it in a database for further manipulations without the owner's knowledge. So, there lies a big security risk related with the information flow.

There will be differences in the information flows of malicious and benign applications. We cannot neglect the chances for information flow in the two applications being same. Eventhough the information flows are same , the structure of these flows may be quite different.

5. OVERVIEW OF PROPOSED SYSTEM

The proposed system composed of six modules. Two of them are on client side and remaining four on server side. App behavior tracker module in the android device will track the behaviour of installed applications in the phone and log creation module will record those behaviours as a log file. Later, this log file will be send to server over the network for behaviour rebuilding purpose and for further analysis. The log analyser module will analyse the record and rebuilds the behaviour as a graph. Code analyser module will perform APK decompilation and static analysis. The obtained features will stored in a database(feature library)and later it is used for training the classifiers. The architecture of the system is in Fig 2.

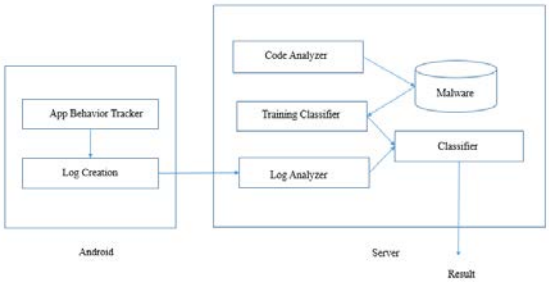

Fig.2. Architecture of mDroid

6. MDROID

6.1 Initialization

The initialization module is in responsibility of generating resource files when the mDroid application is used for the first time. Clearly, some resource files may need to be copied, and some files may need to be created, such as the uid_file and directories for log files and behavior graphs. Nonsystem applications are shown as a list each time the user opens the mDroid application. The detailed information of each nonsys-

tem application need to collected. The information includes the name of the application and the application package, version(name & number) of the application and id of the application.

6.2 Behavior Tracking

6.2.1 System Call Interception

We first obtained the address of system call table and intercepted the specific system calls. Then source addresses of required system calls are noted and let it be handled as usual. Four types of system calls are intercepted in our system: Android interprocess communication, file operations, network operations, and process management. Among these system calls, Android interprocess communication can be parsed by Binder Parser, as most of the system calls depend on the Android Binder mechanism; other system calls can be directly parsed using System Call Parser.

6.2.2 Information Flow Extraction

Next step is the discovery of Information-Flows. Information-Flow analysis described here is different from the traditional information flow analysis which mainly focuses on the discovery of a flow from a single source to a single sink. Here individual single source to a single sink flows are aggregated and connected. We leverage data flow analysis techniques to extract paths contained within each simple flow. If two information flows share a common path then these two information flows are grouped together. Each group can contain multiple information flows, which means it can contain multiple sources and multiple sinks. We then analyze these flow groups and extract API sequences present within them.

While doing so, we analyze control-flow paths in each flow group to extract API call sequences. , All the code paths (control paths with branches and loops) are examined statement by statement, in the execution order to extract APIs. Here Android Intents are also considered. Intent will be treated as a sink since it's a potential point to leak data outside.

6.2.3 Other Feature Extraction

Features considered here are information code segment and resource segment. In order to obtain information of code segment we need to extract API call information and data stream information. As we gathered API information already, data stream information is the only thing that we need to extract. For that, user definition chain, which contains all the activities of a data used in a software is used.

6.3 Behavior Rebuilding

The behaviour rebuilding module in the server is responsible for rebuilding the application behaviour as a graph so that it will be more convenient for the users to understand the behaviors. The behaviour graphs are generated by using log records prepared by the behaviour analysis module in the client device. So these records must be sent to server for rebuilding the bahavior.

6.4.1 Graph Creation Algorithm

The graph creation algorithm is presented below. In the algorithm, "uid" stands for the unique id of an Android application, "pid'" stands for the process id, and "cid" stands for the id of child process. Since we create a node for each record in the log file, the time complexity of the algorithm is O(n), where n stands for the number of records in the log file.

Algorithm : Graph Creation

1) Start
2) Create a root node
3) For each record in the log file, repeats steps from 3 to 8 until the log file become empty
4) Extract uid, pid, cid(if the system call is clone) the name of the function, and parameters from each log record
5) Check whether there is a node corresponding to this uid, if yes go to step 6 else goto step 9
6) Check whether there is a node corresponding to the pid obtained in step 4, if yes go to step 7 else goto step 8
7) If there is a cid, create a node corresponding to this cid and make it the child of the node corresponding to its pid
8) If there is no node created for the pid, create one node and make it as the child of the node with its uid
9) If no node for uid, create a node and make itas the child of root node.
10) Stop

6.4 Constructing A Feature Library

After the feature extraction we need to create a library of these features. The extracted features(ie. Information flow features, API features, data stream features and resource segment information) of both malwares and benign applications are stored in this library for training and identification purpose. During the training process, we converted the feature library into a feature vector of length n where the i-th dimension of the vector is the i-th functional feature in the library.

6.5 Apk Decompilation

The server receives the APK of an Android application and extract permissions and APIs from it[23]. For decompiling APKs, a decompiler provided by Google called, APKTool is used. The tool generates files such as source codes, a configuration file, and resources. By traversing the configuration file (AndroidManifest.xml) list of permissions can be obtained. Android applications are often built using java. So the android system own a java virtual machine implementation called Dalvik VM itself. The Dalvik VM uses the dex format and smali/baksmali is the assembler for the dex format used by Dalvik. The APK decompilation tool decom-

piles the .dex file into .smali files. No tools to decompile back to java. Hence, we traversed all smali files and each and every line starting with an "invoke," (which represents a function call) is extracted. Then, the APIs are extracted by parsing those lines.

6.6 Permission, Api And Application

As the first step, four numbers need to be calculated: the number of malware applications with the permission, the number of benign applications with the permission, the number of malware applications without the permission, and the number of benign applications without the permission. Similarly, these four numbers are calculated for each APIs. In order to find whether there is a relation between presence of a permission and the nature of an application owning the permission a chi-square test is applied. The below given equations (1) & (2) are used to calculate the chi-square values[22]. In the equations, a stands for the number of malware applications with the permission, b stands for the number of benign applications with the permission, c stands for the number of malware applications without the permission, d stands for the number of benign applications without the permission, and n stands for the number of applications. In the case where the value of a, b, c, or d is less than 5 and value of n is greater than 40, the correction equation (1) is used. In normal cases, equation (2) is used.

$$\chi^2 = \frac{\left(|ad-bc|-\frac{n}{2}\right)^2 n}{(a+b)(c+d)(a+c)(b+d)} \quad (1)$$

$$\chi^2 = \frac{(ad-bc)^2 n}{(a+b)(c+d)(a+c)(b+d)} \quad (2)$$

This chi-square test considers permissions and APIs that are used more than 50 times. Chi-square value and the correlation between the presence of permission and the nature of an application is directly proportional. As the value increases, correlation also increases. After ordering the chi-square test values first 80 are selected and defined as characteristic attributes. The final step is to train our naïve Bayes classifier with these characteristic attributes.

We calculate the conditional probabilities of the characteristic attributes. Using the data in the database, we can determine the probability of an application being malware with a certain characteristic attribute, probability of an application being malware due to without a certain characteristic attribute, the probability of an application being benign with a certain characteristic attribute, and the probability of an application being benign without a certain characteristic attribute.

6.7. Classifier Training & Malware Identication

We collected samples of malware and benign application and extracted features of both category. We used support vector machine as well as naïve Bayes classifier in this work. Support vector machine can be trained by feature vectors mentioned in Section 6.4. Our system learn how malwares leverage information flows and what types of behavior it contains and notes the same in case of benign applications also. We use this as the features to train the SVM. So that, our system will be able to specify if an app leverages information flows in a benign or malicious way. In general, our classification system will detect if an information flow is suspicious or not based on the app's behavior along the information flow.

In short, the app is benign if the flows perform meaningful operations similar in structure to other benign apps and malicious if the structure is similar to other malicious apps. With the information flow features, we also considered code segment and resource segment informations to train SVM. Since our proposed system classifies both categories of applications training using samples of both are preferred.

The steps for training naive Bayes classifier is as follows. First, the server receives the APK and the log file of an application from the client. Then, the APK is parsed according to Section VI. E.. In addition, we also extract the permissions and APIs recorded by behavior tracker from the log file. Then, using the probabilities from before, the probability of the application being malware as well as the probability of the application being benign can be determined. Finally, the two probabilities are compared to identify the nature of application.

Now we have two set of results, one from SVM and one from NB. Finally by comparing these two results we finalize the application into either malicious or benign. If any of the two result specifies the application has malicious activity, we categories it as malware. Our two set identification is intented to ensure that no application could overcome the test. Even if it could escape from one test by any chance, it should be caught by other one.

7. DISCUSSION & EVALUATION

In this paper, we have proposed and implemented a real time behaviour analysis system to identify malware applications in android platform. We have considered features such as permissions, APIs, information flow, code segment and resource segment information. The multiple features are expected to improve the efficiency of the malware identification. We trained the system using samples of both malware and benign application. On testing, our proposed system is proved to be efficient and obtained an accuracy rate of 95.39%.

In future, we think the system can be improved by following ways.

1) Improve the graphical representation by integrating informations from both static and dy-

namic analysis . i.e. by integrating all the features considered.

2) Extend the analysis to identify malware games.
3) Improve the accuracy by including different type of classifiers such as kNN, DT and RF.
4) Enhance the system by considering more features such as
    a. Sytem call frequency
    b. Presence of key information such as: crypto code; dynamic code; native code; reflection code and database

8. CONCLUSION

Based on several features such as permissions, APIs, and information flow a new system for identifying android applications is proposed. This real time system performs a hybrid analysis on applications by leveraging machine learning to detect malicious behaviors exhibited by them. Required features are extracted and trained classifiers with these features. On evaluation, our system found to be efficient and obtained an accuracy rate of 95.39%.

REFERENCES

[1] STATISTA - https://www.statista.com/statistics/467163/forecast-of-smartphone-users-in-india/.W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)

[2] IDC - https://www.idc.com/promo/smartphonemarket-share/os

[3] APPBRAIN - https://www.appbrain.com/stats/number-of-android-apps

[4] DEVELOPER - https://developer.android.com/google/play/asi

[5] GDATASOFTWARE - https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise

[6] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu and E. Kirda, "AccessMiner: using system-centric models for malware protection," in *Proc. 17th ACM Conf. Computer and Communications Security*, Chicago, IL, USA, 2010, pp. 399-412.

[7] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Y. Jung, P. McDaniel and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX Conf. Operating systems design and implementation*, Vancouver, BC, Canada, 2010, pp. 393-407.

[8] L. K. Yan and H. Yin, "DroidScope: seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis," in *Proc. 21st USENIX Conf. Security symposium*, Bellevue, WA, 2012, pp. 29-29.

[9] M. Grace, Y. Zhou, Q. Zhang, S.H. Zou and X. X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proc. 10th Int. Conf. Mobile systems, applications, and services*, Low Wood Bay, Lake District, UK, 2012, pp. 281-294.

[10] Y. Zhang, M. Yang, B. Q. Xu, Z. M. Yang, G. F. Gu, P. Ning, X. S. Wang and B. Y. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proc. ACM SIGSAC Conf. Computer and communications security*, Berlin, Germany, 2013, pp. 611-622.

[11] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," in *Proc. EISIC*, Odense, Denmark, 2012, pp. 141-147.

[12] J. Hoffmann, M. Ussath, T. Holz and M. Spreitzenbarth, "Slicing droids: program slicing for smali code," in *Proc. 28th Annual ACM Symposium on Applied Computing*, Coimbra, Portugal, 2013, pp. 1844-1851.

[13] A. Reina, A. Fattori and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proc. ACM European Workshop on Systems Security*, Prague, 2013, pp. 1-6.

[14] A . Deghantanha, N . Izura and R. Mahmod, *"Towards Data Centric Mobile Security"*, 2011 17th International Conference on Information Assurance and Security (IAS)

[15] M . Choudhary and B . Kishore, "*HAAMD:Hybrid Analysis for Android Malware Detection*", *2018 International Conference on Computer Communication and Informatics (ICCCI -2018), Jan. 04 – 06, 2018, Coimbatore, INDIA*

[16] M . Jaiswal et al., "Android Gaming Malware Detection Using System Call Analysis", *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*

[17] C . Zhao et al., "Quick and Accurate Android Malware Detection Based on Sensitive APIs", *2018 IEEE International Conference on Smart Internet of Things.*

[18] J. D. Koli, " RanDroid:Android Malware Detection Using Random Machine Learning Classifiers", *IEEE International Conference on Technologies for Smart-City Energy Security and Power (ICSESP-2018), March 28-30, 2018, Bhubaneswar, India*

[19] D . Li et al., "Fine-grained Android Malware Detection based on Deep Learning", *2018 IEEE Conference on Communcations and Network Security (CNS): IEEE CNS 2018 – Posters.*

[20] Faiz Mohammad Faqiry. , Rizwanur Rahman. , Deepak Singh Tomar. , 2017. Scrutinizing Permission Based Attack On Android Os Platform Devic-

es. International Journal of Advanced Research in Computer Science. Vol 8, No. 7. pp. 421-426.

[21] Arushi Jain. , Prachi. , 2016. Android Security: Permission Based Attacks. In Proceedings of the International Conference on Computing for Sustainable Global Development. IEEE, pp. 2754-2759.

[22] S. Sun et al., "Real-Time Behaviour Analysis and Identification for Android Application", IEEE Access.Vol.6,July2018,**doi:** 10.1109/ACCESS.2018.2853121.

[23] F. Yu, S. Anand, I. Dillig and A. Aiken, "Apposcopy: semantics-based detection of Android malware through static analysis," in *Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, China, 2014, pp. 576-587.

[24] F . Shen et al., "Android Malware Detection Using Complex-Flows", *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*

[25] J . Du et al. "A Dynamic and Static Combined Android Malicious Code Detection Model Based on SVM", *The 2018 5th International Conference on Systems and Informatics (ICSAI 2018).*