International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-10 Issue-7, May 2021

R-MFDroid: Android Malware Detection using Ranked Manifest File Components

Kartik Khariwal, Rishabh Gupta, Jatin Singh, Anshul Arora

Abstract*: With the increasing fame of Android OS over the past few years, the quantity of malware assaults on Android has additionally expanded. In the year 2018, around 28 million malicious applications were found on the Android platform and these malicious apps were capable of causing huge financial losses and information leakage. Such threats, caused due to these malicious apps, call for a proper detection system for Android malware. There exist some research works that aim to study static manifest components for malware detection. However, to the best of our knowledge, none of the previous research works have aimed to find the best set amongst different manifest file components for malware detection. In this work, we focus on identifying the best feature set from manifest file components (Permissions, Intents, Hardware Components, Activities, Services, Broadcast Receivers, and Content Providers) that could give better detection accuracy. We apply Information Gain to rank the manifest file components intending to find the best set of components that can better classify between malware applications and benign applications. We put forward a novel algorithm to find the best feature set by using various machine learning classifiers like SVM, XGBoost, and Random Forest along with deep learning techniques like classification using Neural networks. The experimental results highlight that the best set obtained from the proposed algorithm consisted of 25 features, i.e., 5 Permissions, 2 Intents, 9 Activities, 3 Content Providers, 4 Hardware Components, 1 Service, and 1 Broadcast Receiver. The SVM classifier gave the highest classification accuracy of 96.93% and an F1-Score of 0.97 with this best set of 25 features.

Keywords*: Android Security, Machine Learning, Malware Detection, Manifest File Components, Mobile Malware, Static Solution.

I. INTRODUCTION

With smartphones being a regular part of our lives these days, we can say that nearly every individual has a smartphone in the 21st century. They are one among many of the cheapest and simplest ways to connect to the World Wide Web, i.e., Internet. As per a recent report [1], the share of smartphones and desktop systems is 54.49% and 45.51% in the business market respectively. This highlights the popularity of smartphones in users across the globe. Moreover, the presence of feature rich apps in smartphones make them more popular than desktops and laptops. Among all mobile operating systems, Android has the highest shareholder with 81% [2] in the current market. These reports portray the developing popularity of mobile devices, specifically Android based, and it is predicted to increment further in the upcoming years. With the expanding interest of users in smartphones, the danger to these smartphones is likewise expanding. As per the reports [3], nearly 10 million mobile devices were infected with malware through a "Samsung Update". Another recent report [4] shows that thousands of malignant applications are randomly uploaded on various app stores, which are disguised as they are legitimate applications. A malware known as Dress-code was carried out by these malicious apps. Dress-code is programmed in such a way that it is able to infiltrate deep networks and is able to steal data from the system. These examples depict how weak the current security of smartphones is and how easy it is to get targeted by attackers. These malicious apps pose threats like data leaks, system damage, huge monetary losses, etc. Since more users are opting for Android over other mobile OS nowadays, that's why it is coming out to be a major target by attackers. In 2017, around 26 million malicious applications were detected on Android platform [5]. Whereas, In May 2019, around 45,000 users with Android OS were infected by an unknown malware named *xhelper* [6]. Presence of such attacks in Android demand stronger and stealthier mechanisms to detect malicious apps with greater accuracy.

A. Motivation

A great amount of research has been done in the detection of Android apps containing malicious behavior which can be categorized into three detection types: Static, Dynamic, and Hybrid detection. Dynamic and Hybrid malware detection focuses on analysing the runtime behavior of apps which is very computationally complex. Therefore, our research focuses on the static detection of malicious apps.

Among the components present in the manifest file, permissions is the most used feature in static detection of Android malware, followed by Java Code, intents, and then other components. Manifest file components have been used in a few research works for Android malware detection.

Manuscript received on May 11, 2021.
Revised Manuscript received on May 17, 2021.
Manuscript published on May 30, 2021.

* Correspondence Author

Kartik Khariwal*, Discipline of Mathematics and Computing, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: kartik.khariwal@gmail.com

Rishabh Gupta, Discipline of Mathematics and Computing, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: rishabhmgupta@gmail.com

Jatin Singh, Discipline of Mathematics and Computing, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: jatinsingh0710@gmail.com

Anshul Arora, Discipline of Mathematics and Computing, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: anshul15arora@gmail.com

For example, authors in [7], [8], [9], [10], and [11] have used features from manifest files for Android malware detection. However, to the best of our knowledge, none of the previous works done on the static detection of malware apps have aimed to find the best set of different manifest file components, i.e., best set comprising of Permissions, Intents, Hardware Components, Activities, Services, Broadcast Receivers and Content Providers which can distinguish between benign and malicious apps with better accuracy.

Therefore, in this paper, we aim to identify the best feature set of manifest components which gives better detection accuracy. This work is an extension of our previous work [21] where we studied permissions and intents for Android malware detection. In this extended version, we have considered seven features that can be extracted from an Android manifest file, i.e., Permissions, Intents, Hardware components, Activities, Services, Content Providers, and Broadcast Receivers, for Android malware detection.

B. Contributions

The proposed work is divided into three basic yet effective steps which are highlighted below:

1. Firstly, for each feature, i.e., for each manifest file component, we created a Bag-of-Words model which is a matrix representation with all apps as rows and each unique feature represented as a column.
2. We ranked all the features based on Information Gain (I.G.) Score by calculating I.G. for each unique feature.
3. Further, to find the best set of combined features from different manifest file components, we proposed a novel algorithm to find the best set comprising of all the features by executing several machine learning models along with deep learning techniques.

C. Organization

The rest of the paper is organized as follows. In section II, we review the related works in the field of Android malware detection. The proposed methodology is elaborated in Section III. Section IV discusses the experimental results and we conclude with future work directions in Section V.

II. RELATED WORK

In this section, we review the research works put forward for Android malware detection. Related works can be divided into three different categories - Static, Dynamic, and Hybrid detection. We review each of the detection types in the upcoming subsections.

A. Static Detection

In static techniques, the Android malware are detected without executing the malicious apps on smartphones. Static features like manifest file components, Java code, API calls, etc. are used in static detection.

i. Manifest File Based Detection

In this subsection, we discuss the related works that have studied various manifest file components for Android malware analysis or detection. The authors in [10] calculated a specific score of each permission by dividing the number of malware samples containing that permission by the total number of malware present in the dataset. They further used this permissions score to detect malicious apps. Moonsamy et al. [11] analysed the harmful and dangerous patterns of permissions defined within the malicious apps.

Idrees et al. [12] identified the significant intents and permissions defined in malicious samples as well as the normal apps and used them for detecting malware. The authors in [13] applied Hamming Distance to find the similarity between the malware and normal apps based on the static features of permissions, intents, and API's. Qiu et al. [14] applied multi-label classification models on the set of extracted features like permissions, API calls, network addresses, etc. with an aim to detect zero-day malware families. FAMD (Fast Android Malware Detector) model proposed in [15] extracted permissions and Dalvik code sequences from the Android samples. They further applied N-gram technique to reduce the feature dimensionality and CatBoost classifier for malware detection. The authors in [16] extracted several static features from manifest files and source code and applied a linear SVM algorithm for malicious apps detection. Few more works like [17], [18], [19] and [20] extract static features for malware detection on Android platform. In our previous work [21], we ranked permissions and intents with Information Gain and further used that ranking for malware detection. In that work, we aimed to identify the best set of intents and permissions for better detection accuracy. In this paper, however, we aim to identify the best set of all the manifest features to further improve the detection accuracy.

The authors in [22] applied Factorization Machine architecture on the set of manifest components to detect malicious Android apps. Sato et al. [23] evaluated the malign score for each of the manifest file components depending upon the number of malicious applications the component is present in. They further used that malign score for malware detection. The work done in [24] is about a Feature transformation-based Android Malware detector which takes well-known features and introduces three new types of feature transformations that transform these features irreversibly into a new feature domain.

None of the above-discussed works have aimed to find the best feature set comprising manifest file components for malicious apps detection in Android. However, our work aims to find the best feature set that could give better detection accuracy.

ii. API Calls Based Detection

Some of the researchers have used static API calls to detect Android malware. User-trigger dependence and sensitive APIs in malicious apps were analyzed by [25] while the authors in [26] constructed the dependency graphs of API calls and categorized the malicious apps into their own corresponding Android malware families with similarity metrics. Similarly, the work proposed in [27] analyzed API calls and their call graphs for malware detection. A model was proposed with the capabilities of both binary file feature representation and feature selection for malware detection in [28] while in [29] the authors have used the code semantic structure features to reflect deep semantic information and proposed a pre-processing method of APK files to generate graphics that reflect the code semantic features.

B. Dynamic Detection

Static detection mechanisms do not run or execute the applications, hence, such solutions may leave the malicious component undetected because such malicious components

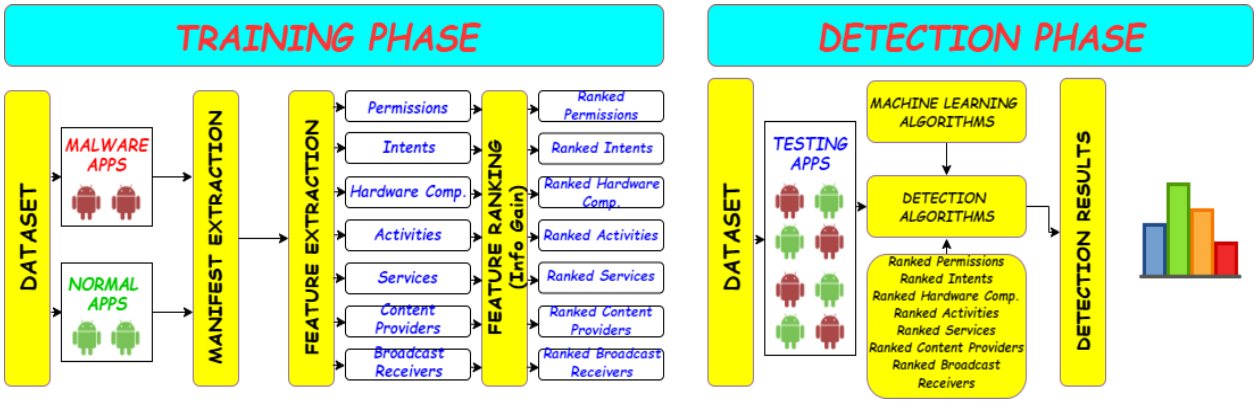


Figure 1: System Design of R-MFDroid Model

May be downloaded at the update time. Therefore, dynamic solutions were proposed by the authors for Android malware detection. We can further split these solutions into two types: OS-based malware detection and Network Traffic based malware detection.

i. OS-Based Detection

The API and system calls were analyzed by the authors in [30] to detect malicious apps in Android. Feng et al. [31] analyzed dynamic system-level behavior traces in addition to application-level malicious behaviors like premium service subscription, stealing of personal information, and malicious service communication to detect malicious activities at run-time. The authors in [32] analyzed the system calls of apps to distinguish malware from benign ones. Iqbal et al. [33] analyzed several dynamic features such as CPU usage, memory consumption, and system call events to detect malicious apps. The authors in [34] leveraged customized deep neural networks to provide a real-time and responsive detection environment on mobile devices against malware while in [35] the authors proposed a robust, scalable and efficient Cuda-empowered multi-class malware detection technique leveraging Gated Recurrent Unit (GRU) to identify sophisticated Android malware. Surendran et al. [36] found the occurrence of common malicious system call codes in the system call sequence of several malware families.

The dynamic features extraction, for example system calls, is computationally complex and has huge overheads as compared to static techniques, therefore, in this paper, only a static detection model is presented.

ii. Network Traffic Based Detection

Now, we review the related works that have used Internet traffic features for Android malware detection. Wang et al. [37] applied Natural Language Processing techniques on the HyperText Transfer Protocol (HTTP) headers to detect the malware. The authors in [38] extracted network-level features and applied multiple classifiers to detect malicious activity in the network traffic. Igor et al. [39] observed the patterns of 14 features from TCP / IP headers of the normal and malicious traffic files to detect malware network traffic. On similar lines, the authors in [40] and [41] proposed models to detect Android malware based upon network traffic analysis.

All the above-discussed related works can solely be used to detect those malware that produce some amount of network traffic. However, these solutions fail for those which do not generate any traffic, for example, malware that only sends SMS in the background. Therefore, in this work, we have neither
considered a network traffic based or an OS-based detection solution.

C. Hybrid Detection

To combine the advantages of both static and dynamic solutions, few hybrid solutions exist in the literature wherein both static and dynamic features can be combined to propose a hybrid detection model.

The authors in [42] applied deep artificial neural networks on both static and dynamic layers for malware detection. The authors used static permissions and intents, and dynamic API calls for detection. Mahindru et al. [43] extracted features such as permissions, dynamic API calls, apps rating, and the number of users download and further applied machine learning techniques to detect malware. AdDroid model presented in [44] analyzed Android actions such as Internet connections, uploading of a file to a server, installing packages on the device, etc. to detect malicious activities on the device. Zhu et al. [45] observed permissions, sensitive APIs, and run-time system-related events to detect Android malware. Apart from this, the authors in [46] and [47] proposed two completely different hybrid detection techniques by merging and combining the network traffic features with permissions. In [48], a novel hybrid analysis approach for identifying malicious behaviors concealed within dynamically loaded code through reflective calls is proposed.

Because the hybrid detection involves both static as well as dynamic features, they involve computational overheads too, similar to dynamic mechanisms. Therefore, we have aimed to propose a static detection in our model.

III. METHODOLOGY

In this section, we discuss the proposed methodology of our model *R-MFDroid*. There are two phases in our approach: The training phase and the detection phase, as summarized in Figure 1. We discuss both of them in detail in the upcoming subsections.

A. Training Phase

In the training phase, we extract several features from the Android manifest files of the apps in the dataset, and further aim to rank those features using Information Gain. The training phase is further divided into various sub-phases as discussed below.

i. Dataset

We gathered 10,414 benign apks & 5707 malicious apks for our research work. We took the normal dataset of apps from the Android Play Store. Trending apps like Whatsapp, BeautyCam, etc. were also part of our dataset. Hereafter, we checked the normal dataset from the VirusTotal website to ensure that the normal apps downloaded from the Google App Store are not malignant. We collected the malicious dataset from the following sources: Drebin[16], Genome[49] and Koodous[50].

ii. Manifest File Extraction

Static features that we intend to use in our research work are defined inside the manifest file of Android applications. Hence, to begin with, we first extract the manifest files of the applications using Apktool. This tool breaks the application (apk package file) into several components, including its manifest file.

iii. Features Extraction

After extracting manifest files, we wrote scripts in Python to extract the required features, i.e., Intents, Permissions, Hardware Components, Services, Activities, Content Providers and Broadcast Receivers. We extract these features from all the benign and malicious apps in our dataset. We briefly explain each of the features below:

Permissions

An Android smartphone's various resources can only be accessed by an app if permissions are granted by the user at the time of its installation. Android's resources include the data access stored in the mobile device, for example, media files, contact list, or access to hardware components such as a microphone or a camera. These permissions required by an app are not granted on their own, instead, they are needed to be granted by the user.

Intents

An intent is similar to a messaging system that can request an action from a component of another app. Some basic functions of intents include the start time activity, launch activity, web page display, contact list display, and message broadcasting.

Activities

An Android activity is nothing but the screen of an Android app's UI which is the same as of the windows in any desktop application. An Android application might contain two or more activities. An application's activity goes through a life cycle in which the Android application is started with main activity followed by three other states served to the user: created, started and resumed.

Broadcast Receivers

Receivers or Broadcast Receivers are the components that allow you to register for system or application related events. All receivers, registered for an event, are notified at runtime of the Android app, once this particular event happens.

Content Providers

They help to supply the content among applications when requested. Such requests can be handled by the methods of the class called ContentResolver. Content providers can use different techniques to store their data and this data can be stored in files, databases, or even over a network.

Hardware Components

Hardware components, as the name suggests, are one of the most important components of an Android app. They provide essential support to hardware features such as cameras and other possible sensors.

Services

A Service is a component of an application that can perform tasks even in the background. It doesn't have any UI. When initiated, service assistance may keep running for quite a while, even after the client changes to another application. Moreover, a component can tie to service, an interface with it, and hence inter-process communication can be performed.

iv. Bag-of-Words Model

After extracting the seven features from the manifest files of all the apps, we use the Word Embedding Technique in *NLP* for creating the *Bag-of-Words* Model for each of the seven features, represented by the sets *S(1), S(2), S(3),....S(7)*, i.e., *S(i)* for *i* ranging from 1 to 7. A bag-of-words model is just a matrix that represents the occurrence of various words that are present in a corpus or document. The current model only focuses on whether a particular word, i.e., feature in our case, occurs in the manifest file or not.

The proposed methodology helps in the matrix representation of each manifest file component, which is later used by machine learning algorithms; wherein each unique entry in the feature set is represented as a column and applications as rows. Say the matrix is represented as *M[A][B]*. The entries in *M[A][B]*, are either 0 or 1.

The entry of 1, i.e., *M[A][B] = 1* signifies that *Ath* app of the dataset contains the *Bth* feature of *S(i)* feature set. The entry of 0, i.e., *M[A][B] = 0* signifies that *Ath* app of the dataset does not contain the *Bth* feature of *S(i)* feature set.

v. Feature Ranking

After creating the Bag-of-Words model for each manifest component in the matrix form, we calculate the Information Gain (I.G.) score for each unique feature present in *S(i)* feature set. Information Gain gives the measure of the information gained about a random variable by observing another random variable. In other words, it can be used to determine, amongst a given set of features, which one is the most distinguishing, i.e., the feature that can better differentiate between several categories. Entropy defines the uncertainty in all the features and I.G. determines the reduction in that uncertainty. The I.G. of the feature *F* for *C* is evaluated by the following equation, where H(C) represents entropy for class C, and H(C/F) denotes the uncertainty in class C for any feature F.

$$I.G.(C/F) = H(C) - H(C/F) \quad (1)$$

We calculate I.G. for each unique feature in each feature set, i.e, *S(1) ..S(7)* using the above equations. Higher the value of I.G. score, lower the entropy, and better the feature in distinguishing between malware & benign Apps. We rank the features in each feature set in the descending manner of the I.G. score. We rank all the feature sets separately. Hence, we get seven ranked lists, one each for each feature set, at the end of this phase.

B. Detection Phase

For our experiments, we have separately kept the training and testing samples, i.e., the samples used for training and testing are different. We extracted all seven manifest components from each of the testing samples. We used three machine learning algorithms: Random Forest, XGBoost, and Support Vector Machines (SVM) and Deep Learning technique of Neural Networks for classifying apps as benign or malicious.

We have divided the detection phase into two Algorithms: Algorithm 1 summarizes the method that gives the best set of each manifest component. For instance, with Algorithm 1, we get one best set of permissions that gives better detection results as compared to all other permissions. Similarly, we get one best set of each of the other manifest components that give better accuracy.Ranked feature sets obtained from the I.G. scores during the Training phase are used by Algorithm 1. Let us see the working of Algorithm 1 with the case of Intents. Firstly, the

Algorithm 1: Returns Best Set of Individual Manifest File Components

1. **Input:** Testing Dataset (T_{Set}), Ranked Permissions List (P_{List}), Ranked Intents List (I_{List}), Ranked Activities List (A_{List}), Ranked Services List (S_{List}), Ranked Content Providers List (CP_{List}), Ranked Broadcast Receivers List (BR_{List}), and Ranked Hardware Components List (HC_{List})
2. **Output**: Best Set of Permissions ($Best_{SetP}$), Intents ($Best_{SetI}$), Activities ($Best_{SetA}$), Services ($Best_{SetS}$), Content Providers ($Best_{SetCP}$), Broadcast Receivers ($Best_{SetBR}$), and Hardware Components ($Best_{SetHC}$)
3. **Parameters Initialization:** $Acc_{MaxI} = 0$, $I_{Set} \leftarrow \phi$, $Best_{SetI} \leftarrow \phi$
4. **for** each intent I_j in I_{List} **do**
5. $I_{Set} \leftarrow I_{Set} \cup I_j$
6. Train Machine Learning Classifiers on I_{Set} and identify the detection accuracy(Acc_{Det})
7. **if** $Acc_{Det} > Acc_{MaxI}$ **then**
8. $Best_{SetI} \leftarrow Best_{SetI} \cup I_j$
9. $Acc_{MaxI} \leftarrow Acc_{Det}$
10. **else**
11. $I_{Set} \leftarrow I_{Set} - I_j$
12. **end if**
13. **end for**
14. Return $Best_{SetI}$

Repeat the entire algorithm again for each of the manifest component to obtain $Best_{SetP}$, $Best_{SetA}$, $Best_{SetS}$, $Best_{SetCP}$, $Best_{SetBR}$, and $Best_{SetHC}$

Algorithm 2: Returns Best Set of Combined Manifest File Components

1. **Input:** Testing Dataset (T_{Set}), Best Set of Permissions ($Best_{SetP}$), Intents ($Best_{SetI}$), Activities ($Best_{SetA}$), Services ($Best_{SetS}$), Content Providers ($Best_{SetCP}$), Broadcast Receivers ($Best_{SetBR}$), and Hardware Components ($Best_{SetHC}$)
2. **Output**: Best Combined Set of Manifest Components ($Best_{SetCOMB}$)
3. **Parameters Initialization:** $Acc_{Max} = 0$, $F_{Set} \leftarrow \phi$, $Best_{SetCOMB} \leftarrow \phi$
   $F_{List} \leftarrow Best_{SetP} \cup Best_{SetI} \cup Best_{SetA} \cup Best_{SetS} \cup Best_{SetCP} \cup Best_{SetBR} \cup Best_{SetHC}$
4. Sort the features in F_{List} in decreasing order of I.G. score.
5. **for** each intent F_i in F_{List} **do**
6. $F_{Set} \leftarrow F_{Set} \cup F_i$
7. Train Machine Learning Classifiers on F_{Set} and find the detection accuracy(Acc_{Det})
8. **if** $Acc_{Det} > Acc_{Max}$ **then**
9. $Best_{SetCOMB} \leftarrow F_{Set}$
10. $Acc_{Max} \leftarrow Acc_{Det}$
11. **else**
12. Continue
13. **end if**
14. **end for**

Return $Best_{SetCOMB}$

Algorithm selects the highest-ranked intent, i.e., Intent with the highest Information Gain score. We then train classification algorithms on the selected intent and find the initial classification accuracy. $Best_{SetI}$ denotes the best set of Intents. The top-ranked Intent is put into the set $Best_{SetI}$. Thereafter, we select the second top Intent, put it into our $Best_{SetI}$, train the classification algorithms on the merged set, and determine the detection accuracy. If this detection accuracy, obtained after adding the second intent, is greater than the previous one, i.e., with the top ranked Intent alone, only then we merge the second top Intent with the top ranked intent in the $Best_{SetI}$.

Otherwise, we ignore the second top intent, go to the third top Intent, and follow the same procedure again. After iterating through all the Intents, the algorithm, as an output, gives the $Best_{SetI}$, i.e., the best set of Intents that returns better detection accuracy. We repeat the procedure for each of the manifest component to find the best set of other components, i.e., best set of Permissions $Best_{SetP}$, best set of Hardware Components $Best_{SetHC}$, best set of Content Providers $Best_{SetCP}$, best set of Activities $Best_{SetA}$, best set of Broadcast Receivers $Best_{SetBR}$, and best set of Services $Best_{SetS}$.

Now, in this work, we aim to find the best set of combined manifest file features, hence, we further use the sets: $Best_{SetP}$, $Best_{SetI}$, $Best_{SetA}$, $Best_{SetS}$, $Best_{SetCP}$, $Best_{SetBR}$, and $Best_{SetHC}$. Algorithm 2 describes the method that generates, as an output, the best set of combined features that gives better detection results. We merge all seven sets into a single set denoted by F_{List}.

Furthermore, we sort the features present in F_{List} in descending order of I.G. score. The proposed algorithm returns the best set of features combined, i.e., $Best_{SetCOMB}$.

We should keep in mind here that to find this best set $Best_{SetCOMB}$, we do not use the whole of individual feature sets. With the help of Algorithm 1, firstly, we find the best set of feature sets separately. Thereafter, in Algorithm 2, we make use of these best sets, i.e., $Best_{SetP}$, $Best_{SetI}$, $Best_{SetA}$, $Best_{SetS}$,

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

$Best_{SetCP,}$ $Best_{SetBR}$, and $Best_{SetHC}$ to identify the combined best set $Best_{SetCOMB}$. Different machine learning classifiers give different $Best_{SetP}$, $Best_{SetI}$, $Best_{SetA}$, $Best_{SetS}$, $Best_{SetCP,}$ $Best_{SetBR}$, and $Best_{SetHC}$, and hence, different accuracy. We have applied three machine learning classifiers: Random Forest, XGBoost , and SVM and Neural networks in Deep learning for classification. Therefore, for each of the classifiers, we have a different $Best_{SetCOMB}$. In the next section, we discuss the detection results obtained from the proposed approach.

IV. RESULTS AND DISCUSSION

In this section, we discuss the results obtained with our proposed methodology. After extraction of features, i.e., Intents, Permissions, Hardware Components, Activities, Services, Content Providers, and Broadcast receivers from the manifest file of all the apps, we identified the unique/distinct features from each feature set. For instance, there are 280 unique permissions in the entire permission set of our dataset. Table I, on the similar lines, summarizes the unique number of entries in each feature set.

Table II summarizes the top 10 features in each of the feature sets ranked in the decreasing order of Information Gain score. Note that, this ranking is obtained when we have kept the seven feature sets separately from each other. However, in Table III, we show the top 20 features, if we combine all the seven feature sets into a single set, ranked in the decreasing order of information gain.

Table- I: Unique Entries in Different Feature Sets.

Feature Set	Number of Unique Entries
Permissions	280
Intents	815
Hardware Components	72
Activities	43381
Services	7462
Content Providers	1275
Broadcast Receivers	5568

A. Permission Analysis

First, we discuss the detection results if we use the permissions alone for detection. We use Algorithm 1 to find the best set of permissions, i.e., $Best_{SetP}$ and further calculate the detection accuracy with permissions alone. We used SVM, Random Forest, XGBoost, and Neural Networks for classification, and we get different results for each of the classifiers. Table IV summarizes the results for each of the classifiers.

Let us interpret the results obtained. SVM classifier gave the detection accuracy of 95.13% and an F1-score of 0.9523 with the best set of 24 permissions. Corresponding to the SVM classifier, the last column in Table IV summarizes the rank of those 24 permissions which form the best set of permissions and hence give better accuracy as compared to any other set of permissions with the SVM classifier. On closely analyzing the results, we observe that the top three permissions are present in the best set. When the algorithm adds fourth ranked permission into the best set, the detection accuracy decreases. Therefore, the fourth-ranked permission gets discarded by the algorithm and is not added into the best set. We find that with the case of SVM, the proposed detection algorithm went up to 70th ranked permission.

Similarly, the results with other classifiers can be understood. When we compare the results of SVM with those obtained from XGBoost, Random Forest, and Neural Network, we find that Random Forest gave its best accuracy of 95.13% and F1-Score of 0.9510 on the best set of 25 permissions. Neural Networks gave its best accuracy of 94.62% and an F1-score of 0.9459 on the best set of 21 permissions.

B. Intents Analysis

Similar to the analysis done with permissions in the previous subsection, we review the results, in this subsection, if we use the intents

Table- II: Top 10 Ranked Manifest File Components (Seven Feature Sets Kept Separately).

Top Permissions	Top Intents	Top Activities	Top Hardware Components	Top Services	Top Broadcast Receivers	Top Content Providers
Read SMS	Browsable	GoogleApi	TouchScreen	Firebase Instance Id	Firebase Instance Id	FirebaseInit
Read Phone State	View	Ad Activity	Touchscreen Multitouch	App Measurement	App Measurement	FacebookInit
Send SMS	Boot Completed	Facebook Activity	Touchscreen Distinct	App Measurement Job	App Install Referrer	Crashlytics Init
Write SMS	Sig Str	Custom Tab Main	Camera	Component Discovery	Access Token Expiration	Content.File Provider
Receive SMS	Default	Sign In Hub	Location GPS	Firebase Messaging	Analytics	Mobileads Init
Wake Lock	User Present	Audience Network	Location	Revocation Bound	Update Receiver	Facebook Content
Write APN Settings	Battery Changed	CustomTab	Camera Autofocus	Analytics	Basea	Marketing Init
Restart Packages	Input Method	RemoteAN	Wi-Fi	Analytics Job	Multiple Install	Arch.Process Lifecycle Owner
Read External Storage	My Package Replaced	AppLovin Confirmation	Location Network	Ads Messenger	Network State	Firebase Perf
Read Logs	UMS Connected	AppLovin Interstitial	Sensor Accelerometer	AdsProcess Priority	Campaign Tracking	Process Lifecycle Owner Initializer

Table- III: Top 20 Ranked Manifest File Components (Seven Feature Sets Combined).

Rank	Feature Name	Feature type
1	GoogleApi	Activity
2	Firebase Instance Id	Broadcast Receiver
3	Firebase Init	Content Provider
4	Firebase Instance Id	Services
5	App Measurement	Broadcast Receiver
6	App Measurement	Services
7	App Measurement Install Referrer	Broadcast Receiver
8	App Measurement Job	Services
9	Component Discovery	Services

10	Ad Activity	Activity
11	Read SMS	Permission
12	Firebase Messaging	Services
13	Read Phone State	Permission
14	Send SMS	Permission
15	Facebook Activity	Activity
16	Write SMS	Permission
17	Browsable	Intent
18	Receive SMS	Permission
19	Custom Tab Main	Activity
20	Facebook Init	Content Provider

Feature alone for detection. Again, we apply Algorithm 1 to find the best set of intents and the corresponding accuracy. Table V summarizes the results obtained with intents. Similar to the results obtained with the permissions, we observe that Neural Networks gave the highest accuracy of 85.66% amongst all the classifiers. And the best set consisted of only 17 intents. The best set generated with other classifiers was higher than that obtained with Neural Networks. Hence, we can argue that the Neural Networks was better than all other classifiers as far as Intents feature is concerned.

C. Activities Analysis

On similar lines, we review the results if we use the activities alone for malware detection. Again, we apply Algorithm 1 to find the best set of activities and the detection accuracy with that best set of activities. Table VI summarizes the results obtained with activities. We observe that SVM gave the highest accuracy of 95.52% amongst all the classifiers. However, the detection algorithm went up to 601st ranked activity and the best set consisted of 24 activities, higher than all other classifiers.

D. Broadcast Receivers Analysis

In this subsection, we review the results if we use the broadcast receivers alone for Android malware detection. Again, we apply Algorithm 1 to find the best set of broadcast receivers and the detection accuracy with the best set of receivers. Table VII summarizes the results obtained with receivers. Again, we observe that SVM gave the highest accuracy of 91.04% amongst all the classifiers. However, the detection algorithm went up to 675th ranked broadcast receiver and the best set consisted of 35 receivers, higher than all other classifiers.

TABLE- IV: Detection Results with Permissions

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Permissions Involved
SVM	95.13	0.9523	24	1, 2, 3, 6, 7, 8, 11, 13, 15, 16, 18, 21, 22, 24, 30, 32, 35, 36, 38, 47, 63, 68,69, 70
Random Forest	95.13	0.9510	25	1, 2, 3, 6, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19, 20, 28, 32, 35, 37, 41, 47,62, 74, 101
XG Boost	95.26	0.9534	23	1, 2, 3, 6, 7, 8, 11, 13, 14, 15,18, 21, 26, 32, 33, 35, 36, 37, 38, 47, 60, 63,65
Neural Networks	94.62	0.9459	21	1, 2, 3, 6, 7, 8, 9, 13, 15, 24, 25, 29, 30, 32, 33, 35, 62, 63, 66, 101, 214

TABLE- V: Detection Results with Intents

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Intents Involved
SVM	86.43	0.8567	19	1, 3, 4, 6, 9, 10, 12, 13, 19, 21, 23, 29, 33, 43, 45, 50, 65, 102, 778
Random Forest	86.55	0.8605	18	1, 3, 4, 6, 9, 10, 12, 13, 16, 18, 19, 21, 29, 35, 38, 44, 47, 70
XG Boost	86.56	0.8591	17	1, 3, 4, 6, 9, 10, 12, 13, 19, 21, 23, 29, 44, 50, 56, 74, 797
Neural Networks	85.66	0.8486	17	1, 3, 4, 6, 7, 9, 10, 12, 15, 18, 19, 29, 33, 44, 50, 79, 83

TABLE- VI: Detection Results with Activities

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Activities Involved
SVM	95.52	0.9567	24	1, 2, 3, 9, 10, 22, 24, 27, 31, 36, 44, 45, 79, 82, 96, 103, 128, 135, 157, 166, 167, 182, 337, 601
Random Forest	94.11	0.9436	14	1, 2, 3, 22, 36, 71, 72, 79, 128, 129, 135, 167, 168, 975
XG Boost	93.73	0.9402	9	1, 2, 3, 44, 67, 71, 128, 337, 975
Neural Networks	94.37	0.9466	20	1, 2, 5, 6, 9, 24, 25, 36, 52, 55, 66, 78, 88, 125, 150, 296, 298, 346, 540, 975

TABLE- VII: Detection Results with Broadcast Receivers

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Broadcast Receivers Involved
SVM	91.04	0.9188	35	1, 2, 4, 5, 9, 12, 16, 17, 25, 26, 28, 30, 35, 51, 57, 67, 82, 93, 106, 121, 156, 179, 237, 264, 285, 344, 345, 407, 539, 542, 557, 590, 611, 651, 675
Random Forest	89.50	0.9064	24	1, 2, 4, 5, 9, 12, 16, 17, 25, 26, 30, 35, 51, 57, 82, 93, 106, 156, 179, 264, 344, 345, 611, 651
XG Boost	87.84	0.8929	14	1, 2, 4, 5, 9, 12, 16, 17, 30, 51, 57, 106, 179, 285
Neural Networks	89.50	0.9062	26	1, 2, 4, 5, 9, 12, 16, 17, 25, 26, 28, 30, 32, 35, 51, 57, 59, 82, 83, 93, 156, 179, 264, 285, 345, 611

E. Content Providers Analysis

In this subsection, we review the results if we use the content providers alone for Android malware detection.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

Again, we apply Algorithm 1 to find the best set of content providers and the detection accuracy with the best set of providers. Table VIII summarizes the results obtained with content providers. Again, we observe that SVM gave the highest accuracy of 90.91% amongst all the classifiers. However, the detection algorithm went up to 956th ranked content provider and the best set consisted of 23 providers, higher than all other classifiers.

F. Hardware Components Analysis

In this subsection, we review the results if we use the hardware components alone for Android malware detection. Again, we apply Algorithm 1 to find the best set of hardware components and the detection accuracy with the best set of components. Table IX summarizes the results obtained with hardware components. Again, we observe that SVM gave the highest accuracy of 76.31% amongst all the classifiers. However, the best set with SVM consisted of 17 components, higher than all other classifiers.

G. Services Analysis

In this subsection, we review the results if we use the services alone for Android malware detection. Again, we apply Algorithm 1 to find the best set of services and the detection accuracy with the best set of services. Table X summarizes the results obtained with services. Again, we observe that SVM gave the highest accuracy of 90.65% amongst all the classifiers. However, the best set with SVM consisted of 29 services, higher than all other classifiers.

TABLE- VIII: Detection Results with Content Providers

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Content Providers Involved
SVM	90.91	0.9179	23	1, 2, 3, 4, 5, 6, 8, 17, 19, 23, 51, 52, 64, 67, 303, 331, 406, 607, 730, 753, 764, 904, 956
Random Forest	89.50	0.9064	12	1, 2, 3, 4, 5, 6, 8, 17, 19, 23, 52, 67
XG Boost	89.12	0.9033	9	1, 2, 3, 4, 5, 6, 8, 17, 52
Neural Networks	89.50	0.9064	12	1, 2, 3, 4, 5, 6, 8, 17, 19, 23, 52, 67

Table- IX: Detection Results with Hardware Components

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Hardware Components Involved
SVM	76.31	0.7996	17	1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 18, 20, 21, 25, 29, 36
Random Forest	75.03	0.7923	11	1, 3, 4, 5, 6, 7, 8, 10, 13, 20, 22
XG Boost	75.03	0.7923	13	1, 3, 4, 5, 6, 7, 8, 10, 11, 15, 16, 18, 31
Neural Networks	74.52	0.7885	9	1, 3, 4, 5, 9, 14, 25, 27, 36

TABLE- X: Detection Results with Services

Classifier	Detection Accuracy (in %)	F1 Score	Best Set	Rank of Services Involved
SVM	90.65	0.9158	29	1, 2, 4, 6, 7, 9, 18, 27, 29, 36, 40, 41, 56, 61, 71, 72, 136, 149, 152, 196, 247, 249, 288, 443, 523, 663, 734, 755, 815
Random Forest	89.24	0.9043	21	1, 2, 4, 6, 7, 9, 18, 27, 29, 40, 41, 56, 61, 71, 72, 136, 137, 249, 250, 443, 451
XG Boost	87.45	0.8901	12	1, 2, 4, 6, 7, 9, 18, 27, 40, 56, 71, 149
Neural Networks	89.88	0.9095	24	1, 2, 4, 6, 7, 9, 18, 27, 29, 36, 40, 41, 56, 61, 71, 72, 136, 149, 152, 216, 247, 249, 288, 755

TABLE- XI: Detection Results with Merged Best Set of Manifest File Components

Classifier	Best Set Count								Accuracy (in %)	F1-Score
	Permissions	Intents	Activity	Broadcast Receivers	Content provider	Hardware Components	Services	Merged		
SVM	24	19	24	35	23	17	29	171	96.80	0.9690
Random Forest	25	18	14	24	12	11	21	125	96.41	0.9650
XG Boost	23	17	9	14	9	13	12	97	95.52	0.9569
Neural Networks	21	17	20	26	12	9	24	129	96.54	0.9664

H. Combined Manifest Components Analysis

In this subsection, we discuss the results when we merge all the manifest components together for malware detection. Algorithm 2 aims to find the best set of manifest components that give better accuracy than any other set of components. However, before reviewing the results from Algorithm 2, we discuss the detection results if we merge the best set of each manifest component obtained from Algorithm 1. Table XI summarizes the results if we merge the best set of each manifest component and then find the accuracy.

We observe that the SVM classifier gave the best accuracy of 96.80% amongst other classifiers when we merge the best set of each manifest component. However, the number of components in the best set was relatively higher with SVM, i.e., 171 as compared to other classifiers. Other classifiers like Random Forest and Neural Networks also gave nearly the same accuracy (around 96%) with a lower merged best set count.

Next, we review the results obtained with the proposed Algorithm 2, i.e., rather than combining the best set of each component, the proposed approach itself finds the best set of all manifest components by iterating through the ranked list of components. Table XII summarizes the detection results with the proposed approach. Again, the SVM classifier gave better accuracy (96.93%) as compared to other classifiers. We also observe that other classifiers also gave nearly the same accuracy (around 96%). From these results we make two important observations:

- On comparing the detection accuracy obtained from the best set of permissions (Table IV), best set of intents (Table V), best set of activities (Table VI), best set of Broadcast receivers (Table VII), best set of Content Providers (Table VIII), best set of Hardware Components (Table IX), best set of Services (Table X), and merged best set of all components (Table XI) with the accuracy obtained from the proposed Algorithm 2 (Table XII), we observe that the proposed approach gives better detection accuracy as compared to all other cases for all the classifiers.
- When we compare Tables XI and XII, we note that the number of manifest file components in the best set is also less with Algorithm 2 as compared to merging the best set of each component.

TABLE- XII: Detection Results with Combined Manifest File Components (Algorithm 2).

Classifier	Best Set Count								Accuracy (in %)	F1-Score
	Permissions	Intents	Activity	Broadcast Receivers	Content provider	Hardware Components	Services	Merged		
SVM	5	2	9	1	3	4	1	25	**96.93**	**0.9702**
Random Forest	6	4	3	1	3	2	0	19	96.41	0.9655
XG Boost	4	2	3	2	3	3	0	17	96.16	0.9631
Neural Networks	3	2	3	3	1	1	2	15	96.80	0.9689

Therefore, we conclude that combining all components using the proposed approach for malware detection is better than using the individual components

V. CONCLUSION AND FUTURE WORK

In this work, we proposed a static model named *R-MFDroid* for Android malware detection by analyzing the manifest file components. To begin with, we ranked all the manifest file components using Information Gain. Further, we proposed a novel method that applied various machine learning and deep learning algorithms on the ranked components. The algorithm produced the best set of components that gave better accuracy as compared to any other set of components. The results highlighted that combining all the manifest file components for malware detection gave better results than the individual components. In our future work, we will put forward a novel technique that combines, with manifest file components, other features such as API calls, system calls, network traffic, etc., to detect the stealthier malware samples.

ACKNOWLEDGMENT

The authors would like to thank the authors of Genome, Drebin and Koodous for providing us the malware samples for our experiments

REFERENCES

1. Desktop vs Mobile vs Tablet Market Share Worldwide, Available Online. https://gs.statcounter.com/platform641 market-share/desktop-mobile-tablet/.
2. Android dominates 81% of the world smartphone market, Available Online. https://www.cnet.com/news/android643 dominates-81-percentof-world-smartphone-market/.
3. Critical Warning Issued Regarding 10 Million Samsung Phone Updates, Available On line. https://www.forbes.com/sites/daveywinder/2019/07/05/critical-warn ing-issued-regarding-10-million-samsung-phone-updates/.
4. Hundreds of Malicious Apps are showing up on the Google Play Store, disguised as legitimate Applications, Available Online.https://us.norton.com/internetsecurity-emerging-threats-hund reds-of-android-apps-containing-dresscode-malware-hiding-in-googl e-play-store.html/.
5. Development of new Android malware worldwide from June 2016 to May 2019, Available Online. https://www.statista.com/statistics/680705/global-android-malware-volume/.
6. 45,000 Android devices infected by new unremovable xHelper malware, Available On line. https://thenextweb.com/security/2019/10/30/45000-android-devices-infected-by655 new-unremovable-xhelper-malware/.
7. A. Feizollah et al., "A review on feature selection in mobile malware detection", Digital Investigation, vol. 13, pp. 22-37, 2015.
8. M. Grace, W. Zhou, X. Jiang, and A. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements", 5th ACM WiSec, 2012.
9. W. Enck, M. Ongtang, and P. McDaniel, "On Lightweight Mobile Phone Application Certifi661 cation", 16th ACM CCS, 2009.
10. K. Talha, D. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system", Digital Investigation, vol. 13, pp. 1-14, 2015.
11. V. Moonsamy, J. Rong, and S. Liu, "Mining permission patterns for contrasting clean and malicious android applications", Future Generation Computer Systems, vol. 36, pp. 122-132, 2014.
12. F. Idrees, and M. Rajarajan, "Investigating the Android Intents and Permissions for Malware detection", 7th International Workshop on Selected Topics in Mobile and Wireless Computing, 2014.
13. R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features", Future Generation Computer Systems, vol. 105, pp. 230-247, 2020.
14. J. Qiu et al., "A3CM: Automatic Capability Annotation for Android Malware," IEEE Access, vol. 7, pp. 147156-147168, 2019.
15. H. Bai, N. Xie, X. Di and Q. Ye, "FAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation," IEEE Access, vol. 8, pp. 194729-194740, 2020.
16. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket",NDSS,2014.

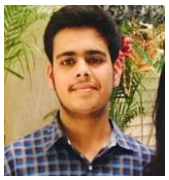
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

17. M. Varsha, P. Vinod, and K. Dhanya, "Identification of malicious android app using manifest and opcode features", Journal of Computer Virology and Hacking Techniques, vol. 13, pp. 125–138, 2017.
18. A. Mahindru, and A. Sangal, "FSDroid:- A feature selection technique to detect malware from Android using Machine Learning Techniques", Multimedia Tools and Applications, 2021.
19. V. Dharmalingam, and V. Palanisamy, "A novel permission ranking system for android malware detection—the permission grader", Journal of Ambient Intelligence and Humanized Computing, 2020.
20. A. Arora, S. K. Peddoju and M. Conti, "PermPair: Android Malware Detection Using Permission Pairs," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1968-1982, 2020.
21. K. Khariwal, J. Singh and A. Arora, "IPDroid: Android Malware Detection using Intents and Permissions," 4th World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, United Kingdom, pp. 197-202, 2020.
22. C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang and H. Kinawi, "Android Malware Detection Based on Factorization Machine," IEEE Access, vol. 7, pp. 184008-184019, 2019.
23. R. Sato, D. Chiba, and S. Goto, "Detecting Android Malware by Analyzing Manifest Files", Proceedings of the Asia-Pacific Advanced Network, vol. 36, pp. 23-31, 2013.
24. Q. Han, V. S. Subrahmanian and Y. Xiong, "Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 3511-3525, 2020.
25. K. Elish et al., "Profiling user-trigger dependence for Android malware detection", Computers & Security, vol. 49, pp. 255-273, 2015.
26. M. Zhang et al., "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs", ACM CCS, 2014.
27. H. Zhang, S. Luo, Y. Zhang and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis," IEEE Access, vol. 7, pp. 69246-69256, 2019.
28. M. Y. -Azar, L. Hamey, V. Varadharajan and S. Chen, "Byte2vec: Malware Representation and Feature Selection for Android," The Computer Journal, vol. 63, no. 1, pp. 1125-1138, 2020.
29. Y. Zhang and B. Li, "Malicious Code Detection Based on Code Semantic Features," IEEE Access, vol. 8, pp. 176728-176737, 2020,
30. V.M. Afonso et al., "Identifying Android malware using dynamically obtained features", Journal of Computer Virology and Hacking Techniques, vol. 11, pp.9-17,2015.
31. P. Feng, J. Ma, C. Sun, X. Xu and Y. Ma, "A Novel Dynamic Android Malware Detection System With Ensemble Learning," IEEE Access, vol. 6, pp. 30996-31011, 2018.
32. M. Jaiswal, Y. Malik and F. Jaafar, "Android gaming malware detection using system call analysis," 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, pp. 1-5, 2018.
33. S. Iqbal and M. Zulkernine, "SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android," 13th International Conference on Malicious and Unwanted Software (MALWARE), Nantucket, MA, USA, pp. 1-8, 2018.
34. R. Feng, S. Chen, X. Xie, G. Meng, S. -W. Lin and Y. Liu, "A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices," IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1563-1578, 2021.
35. I. Bibi, A. Akhunzada, J. Malik, J. Iqbal, A. Musaddiq and S. Kim, "A Dynamic DL-Driven Architecture to Combat Sophisticated Android Malware," IEEE Access, vol. 8, pp. 129600-129612, 2020.
36. R. Surendran, T. Thomas and S. Emmanuel, "On Existence of Common Malicious System Call Codes in Android Malware Families," IEEE Transactions on Reliability, vol. 70, no. 1, pp. 248-260, 2021.
37. S. Wang, et al., "Detecting Android Malware Leveraging Text Semantics of Network Flows", IEEE Transactions On Information Forensics And Security, vol. 13, pp. 1096-1109, 2018.
38. J. Feng, L. Shen, Z. Chen, Y. Wang and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic," IEEE Access, vol. 8, pp. 125786-125796, 2020.
39. I. J. Sanz, M. A. Lopez, E. K. Viegas and V. R. Sanches, "A Lightweight Network-based Android Malware Detection System," IFIP Networking Conference (Networking), Paris, France, pp. 695-703, 2020.
40. A. Arora, S. Garg, and S.Peddoju,"Malware detection using network traffic analysis in android based mobile devices", 8th IEEE NGMAST,2014.
41. A. Arora, and S. Peddoju, "Minimizing Network Traffic Features for Android Mobile Malware Detection", 18th ACM ICDCN, 2017.
42. S. Imtiaz, S. Rehman, A. Javed, Z. Jalil, X. Liu, and W. Alnumay, "DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network", Future Generation Computer Systems, vol. 115, pp. 844 – 856, 2021.
43. A. Mahindru, A. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques", Neural Computing & Applications, 2020.
44. A. Mehtab et al., "AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis", Mobile Networks and Applications, vol. 25, pp. 180–192, 2020.
45. H. Zhu et al., "HEMD: a highly efficient random forest-based malware detection framework for Android," Neural Computing & Applications, vol. 30, pp. 3353–3361, 2018.
46. A. Arora, and S. Peddoju, "NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions", 17th IEEE TrustCom, 2018.
47. A. Arora, S. Peddoju, V. Chauhan, and A. Chaudhary, "Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning", 24th ACM MobiCom, 2018.
48. M. Alhanahnah et al., "DINA: Detecting Hidden Android Inter-App Communication in Dynamic Loaded Code," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 2782-2797, 2020.
49. Y. Zhou, and X. Jiang, "Dissecting android malware: Characterization and evolution", IEEE Symposium on Security and Privacy, 2012.
50. Koodous Malware Dataset, "www.koodous.com".
51. A. Taha, and S. Malebary, "Hybrid classification of Android malware based on fuzzy clustering and the gradient boosting machine", Neural Computing and Applications, 2020.

AUTHORS PROFILE

Kartik Khariwal, is a final year undergraduate student at Delhi Technological University, India. He is pursuing B.Tech in the Discipline of Mathematics & Computing. He has the experience of internship with Samsung Research and Development, India. His areas of interests include Machine Learning, Competitive Programming, and Android Security.

Rishabh Gupta, is a final year B.Tech student in the Discipline of Mathematics and Computing at Delhi Technological University, India. His areas of interest include Mathematics, Machine Learning, Data Analysis, Android Security and Competitive Coding. He also has work experience with American Express as a Risk Analyst.



Jatin Singh, is a final year B.Tech student in the Discipline of Mathematics and Computing at Delhi Technological University, India. His interest lies in competitive coding and Android Security, and he is enthusiastic about new technology.

Anshul Arora, is currently working as Assistant Professor in Discipline of Mathematics and Computing, Delhi Technological University Delhi, India. He has pursued Masters and Ph.D. from Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India. His areas of research include Mobile Security, Mobile Malware Detection, and Network Traffic Analysis.