SFDroid: Android Malware Detection using Ranked Static Features

Gourav Garg, Ashutosh Sharma, Anshul Arora

Abstract: ***Over the past few years, malware attacks have risen in huge numbers on the Android platform. Significant threats are posed by these attacks which may cause financial loss, information leakage, and damage to the system. Around 25 million smartphones were infected with malware within the first half of 2019 that depicts the seriousness of these attacks. Taking into account the danger posed by the Android malware to the users' community, we aim to develop a static Android malware detector named SFDroid that analyzes manifest file components for malware detection. In this work, first, the proposed model ranks the manifest features according to their frequency in normal and malicious apps. This helps us to identify the significant features present in normal and malware datasets. Additionally, we apply support thresholds to remove the unnecessary and redundant features from the rankings. Further, we propose a novel algorithm that uses the ranked features, and several machine learning classifiers to detect Android malware. The experimental results demonstrate that by using the Random Forest classifier at 10% support threshold, the proposed model gives a detection accuracy of 95.90% with 36 manifest components.***

Keywords: Mobile Malware Detection, Mobile Network, Mobile Privacy, Mobile Security.

I. INTRODUCTION

Similar to computers, mobile phones are developed on a mobile OS that, nowadays, has advanced computational power and connectivity, and hence their demand has outperformed the desktops [1]. Amongst different mobile OS, Android is the best market seller due to its open architecture and a large number of apps in the Play Store [2]. This clearly shows the popularity of Android over other mobile OS. App downloads on the Android platform are made easier by the availability of the application stores such as Google Play and other third-party app markets. However, all apps available on these markets are not *safe*. Normal operations of mobile phones are interrupted by some applications known as *malware*. They may cause damage to the system, collection, and leakage of sensitive information, and may even cause a financial loss by sending SMS in the background without users' knowledge.

Manuscript received on May 06, 2021.
Revised Manuscript received on May 15, 2021.
Manuscript published on May 30, 2021.
***** Correspondence Author
Gourav Garg, Student, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: gauravgarg4000@gmail.com
Ashutosh Sharma*, Student, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: ashutosh8110@gmail.com
Anshul Arora, Assistant Professor, Department of Applied Mathematics, Delhi Technological University, Delhi, India. Email: anshul15arora@gmail.com

According to a report [3], in 2018, 5.3 million malicious mobile installation packages were detected. Moreover, 66.4 million attacks were using malicious mobile apps in 2017 which then almost doubled to 116.5 million in 2018 [3].

In 2019, 25 million Android smartphones were infected by malware named Agent Smith [4]. In general, there are three ways by which malicious applications can get into smartphones: (i) Repackaging in Google Play Store, (ii) Update Attacks, and (iii) Drive-by-Downloads.

A. Motivation

Taking into account the danger posed by the Android malware to the users' community, in this work, we aim to propose a static detection mechanism that combines Android manifest file features for malware detection. Static features like permissions, intents, hardware components, content providers, broadcast receivers, etc., are present within the manifest file of any Android app. Permissions' access is one of the security control mechanisms provided by Android, which constraints certain operations that an app can perform. For the app to execute smoothly, permissions must be granted. Permissions are granted when the app is installed by the user (on devices running Android 5.1 and lower) or while the app is running (on devices running Android 6.0 and higher). Intent feature in the manifest file declares the types of intents that an activity or service responds to. Its parent component's properties and abilities, i.e., what an activity or service can perform and what types of broadcasts a receiver can manage, are declared by the intent-filter. Hardware component declares a hardware feature by the developer that the application is using. Content providers provide structured access to application-managed information. Intents that are broadcasted by other applications or the system are received by broadcast receivers. A broadcast receiver can be made known to the system by two methods: one by declaring it in the manifest file, the second is to create the receiver in code dynamically. A Service is a component where the visual user interface is absent. Long-running background operations are implemented by a service. Service is unbounded to the activity's lifecycle. Operations like Internet downloads, checking for new data, data processing, and updating content providers use services.

Whenever a new application is installed on the mobile phones by the users, a list of features is prompted for access, which is mostly ignored and access is granted to the application by the user.

This weakness is exploited by malware developers and dangerous features from manifest files are integrated to generate malicious activity.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

Because manifest file features are needed to be declared explicitly by malicious app developers for their attacks' execution, hence, in this study, we have focused on analyzing manifest file components for Android malware detection. There exist several related works that have analyzed manifest file components for Android malware detection such as [5], [6], [7], [8], [9], [10], [11], etc. In comparison to all these related works, our objective is to find a minimal set of manifest file features that gives better detection accuracy. We rank the manifest features according to their frequency in malware and normal dataset. We aim to use only the top-ranked features instead of using all the features for detection. We get two separate feature rankings: one representing the features that are significantly present in malware apps; and the second representing the features that are significantly present in the normal dataset. To the best of our knowledge, no other work has used such manifest feature rankings for malware detection.

B. Contributions

The main contributions of the proposed *SFDroid* model are summarized below:

1. We ranked the manifest file components based upon their frequency in the normal and malware dataset.
2. We constructed two feature rankings: one for significant features present in malware apps, and the other for significant features present in normal apps.
3. Further, we applied support thresholds to eliminate the lower-ranked features.
4. We proposed a novel algorithm to detect Android malware that applies several machine learning classifiers on two feature rankings.

C. Organization

The rest of the paper is organized as follows. In Section 2, we discuss the related work in the field of Android malware detection. We explain our proposed *SFDroid* model in Section 3. Section 4 reviews the results obtained from the proposed approach, and we conclude with future work redirections in Section 5.

II. RELATED WORK

In this section, we review the research works put forward in the field of detection of Android malware. Static, Dynamic, and Hybrid detections are three different categories in which the related works are classified. In the upcoming subsections, we review each of the detection types.

A. Static Detection

In the static solutions, malware samples are identified in Android devices without installing the apps. It makes them faster and relatively less expensive in comparison to dynamic methods. Static features including manifest file components and API calls are used in this technique.

(i) Manifest File-Based Detection

In this subsection, we discuss the related works that have studied different features in the manifest file for Android malware analysis and detection. Taheri et al. [5] applied hamming distance to obtain the similarity between the benign and malware apps based on features like permissions, APIs, and intents. Qiu et al. [6] applied multi-label classification models on the collection of extracted features like API calls, network addresses, permissions, etc., intending to detect zero-day malware. In [7], the model named *FAMD* (Fast Android Malware Detector) was proposed that extracted Dalvik code sequences and permissions from the samples and applied CatBoost classifier to detect malware apps. In [8], the authors extracted various static features from source code and manifest files, and a linear SVM algorithm was applied to detect malicious apps. Similarly, the authors in [9] applied machine learning algorithms on several static features and manifest components for detection. In some other works like [10], and [11] malware detection is performed by extraction of static features on the Android platform. The identification of the most important permissions to differentiate malware from benign apps is done by the *SIGPID* model presented in [12]. The authors applied pruning with association rules mining for ranking and identifying the important permissions.

The dangerous and harmful patterns of permissions within the malicious apps were analyzed by Moonsamy et al. [13]. The authors in [14] combined permissions and intents for detection using PCA and machine learning techniques. The authors in [15] used permissions for malware detection using gain ratio, J48, Multilayer Perceptron, Sequential Minimal Optimization (SMO), and Randomizable filtered classifiers. Permissions and intents were ranked with information gain in [16] and further, that ranking was applied to detect malicious apps. Likewise, in [17] and [18], authors applied various machine learning algorithms on manifest components for malware detection. In [19], factorization machine architecture was applied by the authors on the collection of manifest features for malware detection in apps. The malign score was further used for malware detection. Some recent techniques like [20] and [21] have also examined manifest features for malware detection. However, none of the above-mentioned works have aimed to find the best set of manifest features for effective Android malware detection. In this work, we aim to find the best set of manifest features which can give better accuracy in the detection of malicious apps.

(ii) API Calls Based Detection

Static API calls have been used by some researchers to detect malicious apps in Android devices. In [22], the authors analyzed the sensitive APIs and user-trigger dependence in malicious apps, and in [23], the authors designed the API calls dependency graphs and classified the malicious apps into their corresponding malware families based on similarity metrics. In [24], the authors proposed a model named *Apposcopy* which analyzed data-flow and control-flow properties from API calls of malicious apps. Wang et al. [25] analyzed structural features like API calls, intents and permissions to detect malware. Similarly in [26], the proposed work analyzed API calls and their call graphs for malicious app detection. Vu et al. [27] constructed an adjacency matrix for each application from its APK source code.

API calls were used to convert APK source code into call-graphs which were further used to train CNN to detect malicious apps. Chen at al. [28] applied API invocation sequences to detect malware.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

However, these API calls are not used in our proposed malware detection technique because they may result in noise if they are not combined with any permission or other manifest file component.

B. Dynamic Detection

Android applications are not executed in static detection techniques, hence, malicious content that gets downloaded at run-time / update time may remain undetected in these types of solutions. Therefore, some authors have proposed dynamic solutions for malware detection. These solutions can be further divided in two categories: OS-based and Network Traffic based detection.

(i) OS-Based Detection

Research works that have used Android OS-based features for malware detection are discussed in this subsection. In [28], the authors analyzed the APIs and system calls for Android malicious app detection. A behavioral signature-based detector named *MADAM* model was presented in [29] that extracted features at four levels: kernel, package, user, and application-level and implemented parallel classifiers to detect malicious apps. Feng et al. [30] examined dynamic system-level behavior traces along with application-level behaviors like premium service approval, theft of personal details, and dangerous service communication to detect malicious activities at run time. In [31], the authors distinguished benign and malware apps by analyzing the system calls of apps. Iqbal et al. [32] analyzed several dynamic features like memory consumption, CPU usage, and system call events for malware detection. The authors in [33] used features that depict the utilization of mobile run-time resources like memory, processor, battery, and network traffic for malware detection. Extraction of dynamic features like system calls is computationally complex and has large overheads in comparison to static methods, therefore, a static malware detection model is presented in this paper.

(ii) Network Traffic Based Detection

Now, we review the related works that have used Internet traffic features for Android malware detection. Wang et al. [34] implemented Natural Language Processing methods on headers of the HyperText Transfer Protocol (HTTP) for malware detection. Network-level features were extracted by authors in [35] and multiple classifiers were applied to detect malicious features in the network traffic. Patterns of 14 features from headers of TCP/IP of benign and malware traffic files were observed by Igor et al. in [36] for malicious network traffic detection. Similarly, in [36], [37], and [38], the authors proposed models based upon analysis of network traffic to detect Android malware apps. All related works discussed above can be used to detect those malicious features that produce a significant amount of network traffic. But, fail for those who do not generate any network traffic; for instance, a malware app that only sends messages. Therefore, network traffic-based detection is not taken into account in this proposed work.

C. Hybrid Detection

Some hybrid solutions do exist in the literature so that advantages of both static and dynamic techniques can be combined. Deep artificial neural networks were implemented for malware detection by authors in [39] on both static and dynamic layers like static permissions, intents, and dynamic API calls. Features like permissions, apps ratings, dynamic API calls, and count of users downloaded were extracted by Mahindru et al. [40], and machine learning techniques were applied for malware detection. Android actions like the Internet connections, installing packages on the device, file uploading to a server, etc. were analyzed by the *AdDroid* model presented in [41] to detect malicious activities. Zhu et al. [42] inspected permissions, run-time system-related events, and sensitive APIs for malware detection in Android. Static features like permissions and dynamic features like network traffic packets were analyzed by the authors in [43] to detect malicious activities in apps. Apart from this, in [44] and [45], the authors introduced two distinct hybrid detection methodologies by joining the network traffic with permissions. Similarly, in [46] and [47], the combinations of static and dynamic features were analyzed for malware detection. Because both static and dynamic features are involved in hybrid detection techniques, similar to dynamic mechanisms, computational overheads are also involved in hybrid ones. Therefore, a static detection method is proposed in our model.

III. PROPOSED METHODOLOGY

Now, we discuss the proposed *SFDroid* model. The detailed methodology is classified into several stages as summarized in Figure 1. We explain all the stages in the following subsections.

A. Training Phase

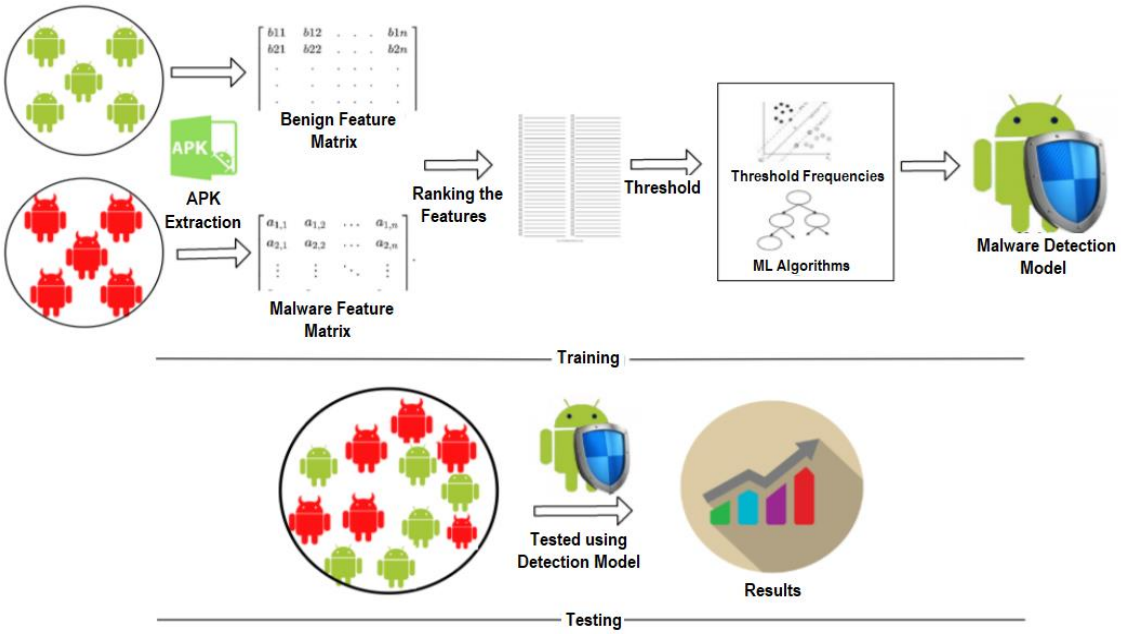
The main aim of the training phase is to rank the static manifest file features so that we can use the top-ranked distinctive features for malicious app detection. The static features required by any Android application are defined in its manifest file that can be extracted using *apktool*. All static features required by each app from their manifest files were extracted using the python script. This phase is further subdivided into two phases, as discussed below.

(i) Features Ranking: We extract six manifest features, i.e., permissions, intents, hardware components, content providers, broadcast receivers, and services from benign and malicious apps. Benign Matrix, *B* and Malware matrix, *M* are used to represent the extracted features from benign and malware apps respectively. In matrices *B* and *M*, the columns represent the six manifest features and rows represent Android applications. Either 0 or 1, are the elements in the matrices. The value of any index [i, j] is 1 if the *jth* permission exists in the *ith* app, else the value is 0.

As the benign and malware applications may differ in the count, unbalanced matrices may give asymmetrical results. So balanced matrices are created by calculating the relative frequency of each static feature in both the matrices.

To obtain the relative frequency of a feature, the frequency of that feature in benign apps is divided by the total number of benign apps and similarly, the frequency of that feature in malware apps is divided by the total number of malware apps. The relative frequency is calculated using the equations mentioned below, where Pj and Qj represent the relative frequency of a feature in benign and malware datasets respectively.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication



Relative Frequency (Pj) = (∑ Bij) / sizeof (Bj) (1)

Relative Frequency (Qj) = (∑ Mij) / sizeof (Mj) (2)

We need to obtain those significant features which are present in benign apps but not in malware ones and vice-versa. Those static features which are significantly present in malware apps are denoted using the term, *dangerous features*. Likewise, those static features which are significantly present in benign apps, are denoted by the term *clean features*. A score is calculated for every feature using the following equation to get the clean and dangerous feature set.

$$R(Fj) = Qj - Pj \quad (3)$$

The value of R(Fj) will lie in the interval [-1, 1], for each feature. The value of R(Fj) as 1, for any feature Fj, denotes that static feature is present only in malicious apps and thus it is the most *dangerous feature*. Whereas the value of R(Fj) as -1 denotes the most *clean feature*. Furthermore, we rank all the features using their scores obtained. For all the features, two rank lists were formed, one in increasing and the other in decreasing order of their scores, represented by *AList* and *DList* respectively. The topmost feature in the *AList* of each category is the one that is frequently present in benign apps. And similarly, the topmost feature in the *DList* represents the one that is frequently present in the malware apps. Note that, for our experiments, first we rank the six features individually, i.e., we have separate rankings for permissions, intents, hardware components, content providers, services, and broadcast receivers. Then, we merge all the features and generate a common ranking consisting of all the manifest components.

(ii) Features Threshold: There are numerous features in the Android Manifest file out of which some features may decrease the detection accuracy. Thus, in our model, features were ranked to remove the lower-ranked features from both *AList* and *DList*, for each of the feature sets. Threshold values vary from 0% to 50% in the intervals of 5%. A threshold value of x% in the interval [0, 50] denotes that only those features which are present in at least x% of both benign and malware apps will be considered in the experiment and the remaining ones will be eliminated. The model is evaluated on each of the threshold values ranging in [0, 50].

B. Testing Phase

In this phase, the proposed model for detecting malicious apps is discussed. Algorithm 1 explains the steps involved in the detection method. After applying the k% support threshold on the ranked lists *AList* and *DList*, Algorithm 1 is applied to find the best feature set to get the highest accuracy for malware detection. Now, to test any app, we extract all of its static features from all categories - hardware, intents, permissions, providers, receivers, and services. Let us consider *AList* first, its topmost feature is selected, and then machine learning algorithms are applied to get the detection accuracy, say x%. Then the first feature is added in the Best Set. Further, the top two features of *AList* are merged to get the detection accuracy, say y%. If y > x, it means the merging of features gives better results in comparison to the individual ones. Hence, both features are put into Best Set, otherwise, if y < x, the second feature is not added into the Best Set. Similarly, a feature is added to the Best Set if it improves accuracy, otherwise, Algorithm 1 moves on to the next feature and the process continues for all features in *AList* to get its Best Set. The same procedure is followed to get the alternative Best Set on other ranked list *DList*. Three machine learning classifiers are used for our evaluation: Naive Bayes, Random Forest, and SVM.

The Best sets obtained from both the lists at several thresholds are discussed in the Results section.

Algorithm 1 - Detection Algorithm that Returns Best Set of Permissions

1: **Input:** Testing Dataset (TSet), Ranked Features Lists (AList and DList)

2: **Output:** Best Set of Features (BestSetA) and (BestSetD), and Highest Detection Accuracy (MaxAccA) and (MaxAccD) obtained from AList and DList respectively

3: **Parameters Initialization:** BestSetA ← φ, BestSetD ← φ, MaxAccA ← 0, MaxAccD ← 0, PSetA ← φ PSetD ← φ

4: Let the number of features in AList and DList be represented by PA and PD after applying the k% support threshold.
5: **for** i = 1− > PA **do**
6: PSetA ← Pi
7: Apply ML Algorithms using the features present in PSetA and find detection accuracy D
8: **if** D > MaxAccA **then**
9: MaxAccA ← D
10: BestSetA ← PSetA
11: **else**
12: PSetA ← PSetA − Pi
13: **end if**
14: **end for**
15: **for** j = 1− > PD **do**
16: PSetD ← Pj
17: Apply ML Algorithms using the features present in PSetD and find detection accuracy D
18: **if** D > MaxAccD **then**
19: MaxAccD ← D
20: BestSetD ← PSetD
21: **else**
22: PSetD ← PSetD − Pj
23: **end if**
24: **end for**
25: Return BestSetA and BestSetD

IV. RESULTS AND DISCUSSION

Now, we discuss the results obtained from the proposed approach. For our experiments, we get the malware samples from three different sources namely Drebin [8], Genome [48], and Koodous [49]. And we downloaded the benign apps from the official Google Play Store. Moreover, we checked these benign apps with Virus Total that contains multiple anti-virus scanners, to make sure that none of the apps downloaded from Play Store is malicious. Firstly, we review the ranking of different features, obtained from the methodology, summarizing the significant features. Then, we highlight the detection results obtained from the *SFDroid* model.

A. Features Ranking

In this subsection, we discuss the ranking of all manifest file components. First, we review the ranking of permissions obtained from the proposed methodology, in both increasing and decreasing order in Table 1. As summarized in the Table 1, *Wake Lock* is the most clean permission, i.e., appearing in the majority of the normal apps and not present in much of the malicious apps. On similar lines, *Read Phone State* is the most dangerous permission, present in the majority of the malicious apps. For every permission, Table 1 also summarizes the score obtained for that permission, i.e., the difference in relative frequencies for that permission in malware and normal dataset. This ranking helps us to identify the significant permissions present in normal and malware dataset. This list of top permissions is obtained at 0% support threshold. Next, we review the ranking of intents, in both increasing and decreasing order. Table 2 summarizes the top clean and top dangerous intents. As Table 2 summarizes, *Browsable* and *Boot Completed* are the top clean and dangerous intents respectively, i.e., appearing in the majority of normal and malware apps respectively. Again, for every intent, Table 2 also highlights the difference in relative frequencies for that intent in normal and malware dataset.

Ascending Order (Top Clean Permissions)		Descending Order (Top Dangerous Permissions)	
Permission	**Score**	**Permission**	**Score**
Wake Lock	-0.427	Read Phone State	0.608
Read External Storage	-0.328	Read SMS	0.553
Camera	-0.166	Send SMS	0.500
Access Network State	-0.158	Write SMS	0.444
Foreground Service	-0.125	Receive SMS	0.444

Table 1. Top 5 Permissions in Ascending and Descending Order

Ascending Order (Top Clean Intents)		Descending Order (Top Dangerous Intents)	
Intent	**Score**	**Intent**	**Score**
Browsable	-0.486	Boot Completed	0.389
View	-0.477	User Present	0.271
Default	-0.370	Sig Str	0.207
My Package Replaced	-0.172	Battery Changed Action	0.175
Leanback Launcher	-0.118	Input Method Changed	0.163

Table 2. Top 5 Intents in Ascending and Descending Order

Now, we review the ranking of hardware components, in both increasing and decreasing order. Table 3 summarizes the top clean and top dangerous hardware components. As Table 3 summarizes, *Touch Screen* and *Refinish Receiver* are the top clean and dangerous hardware components respectively. Table 3 highlights the significant hardware components present in normal and malware dataset. Further, we review the ranking of content providers, in both increasing and decreasing order. Table 4 summarizes the top clean and top dangerous content providers. As Table 4 summarizes, *Firebase Init* and *Launcher* are the top clean and dangerous content providers respectively. Table 4 highlights the significant content providers present in normal and malware dataset.

Ascending Order (Top Clean Hardware Components)		Descending Order (Top Dangerous Hardware Components)	
Hardware Component	**Score**	**Hardware Component**	**Score**
Touch Screen	-0.249	Refinish Receiver	0.0011
Camera	-0.179	Wake Activity	0.0011
Touch Screen Multi Touch	-0.178	Screen Service	0.0011
Touch Screen Multi Touch Distinct	-0.172	Screen Receiver	0.0011
Location GPS	-0.128	Releases	0.0011

Table 3. Top 5 Hardware components in Ascending and Descending Order

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

Ascending Order (Top Clean Providers)		Descending Order (Top Dangerous Providers)	
Content Provider	**Score**	**Content Provider**	**Score**
Firebase Init	-0.635	Launcher	0.024
Internal Facebook Init	-0.359	Downloads	0.004
Android Crash Analytics	-0.335	Interface Stats	0.003
Support V4 Content	-0.284	Download Task	0.002
Mobile Ads Init	-0.232	RCMD Config Provider	0.002

Table 4. Top 5 Content providers in Ascending and Descending Order

Next, we review the ranking of broadcast receivers, in both increasing and decreasing order. Table 5 summarizes the top clean and top dangerous broadcast receivers. As Table 5 summarizes, *Firebase Instance* and *Update Receiver* are the top clean and dangerous broadcast receivers respectively. Table 5 highlights the significant broadcast receivers present in normal and malware dataset.

Next, we review the ranking of services, in both increasing and decreasing order. Table 6 summarizes the top clean and top dangerous services. As Table 6 summarizes, *Firebase Instance Service* and *Update Service* are the top clean and dangerous services respectively. Table 6 highlights the significant services present in normal and malware dataset. As can be seen from Tables 1-6, none of the manifest file components gets the score of +1 or -1, i.e., no manifest feature exists that is present in only one type of the dataset. Because no single distinguishing feature exists, hence, we aim to find the best set of distinguishing features.

Ascending Order (Top Clean Services)		Descending Order (Top Dangerous Services)	
Service	**Score**	**Service**	**Score**
Firebase Instance Service	-0.594	Update Service	0.117
App Measurement Service	-0.587	Custom First Service	0.106
App Measurement Job Service	-0.562	Custom Third Service	0.106
Component Discovery	-0.520	Custom Second Service	0.106
Firebase Messaging	-0.488	Custom Fourth Service	0.105

Table 5. Top 5 Broadcast Receivers in Ascending and Descending Order

Ascending Order (Top Clean Receivers)		Descending Order (Top Dangerous Receivers)	
Broadcast Receiver	**Score**	**Broadcast Receiver**	**Score**
Firebase Instance	-0.640	Update Receiver	0.117
App Measurement	-0.587	Custom Base Broadcast	0.106
App Measurement Install	-0.578	Adservice	0.026
Access Token Expiration	-0.279	Battery Base Broadcast	0.025
Analytics Receiver	-0.252	Boot Receiver	0.024

Table 6. Top 5 Services in Ascending and Descending Order

B. Features Ranking with Individual Features

In this subsection, we discuss the detection results with the proposed SFDroid model. We analyze the results from different machine learning classifiers at different support thresholds. The proposed model aims to find the best set of manifest features that gives better detection accuracy. However, before analyzing the best set of manifest features, we first study the detection results with individual manifest features.

Results with Permissions: Firstly, we review the results if we use permissions alone for detection. We use the permissions' ranking (both increasing and decreasing order) in the proposed detection algorithm to get the best set of permissions that gives better accuracy. Tables 7 and 8 summarize the detection results with permissions at different support thresholds. Table 7 shows the results when permissions are ranked in ascending order. We get the accuracy of 85.66% at 0% threshold with Naive Bayes classifier, and this accuracy is achieved at best set of 15 permissions. On similar lines, we can observe the other entries of the table. With permissions ranked in ascending order, we get the best accuracy of 93.85% at 0% threshold with SVM classifier and this accuracy is achieved at best set of 42 permissions. Similarly, from Table 8, when permissions are ranked in descending order, we get the best accuracy of 94.49% with Random Forest classifier at 0% threshold and this accuracy is achieved at best set of 31 permissions. Hence, we can conclude that when we use permissions for detection, we get the best accuracy of 94.49% with Random Forest classifier at best set of 31 permissions.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	15	85.66	46	92.57	42	93.85
5	4	79.39	7	88.09	8	88.48
10	4	80.03	6	82.84	6	83.74
15	3	75.42	3	81.69	3	81.69
20	3	75.42	3	81.69	3	81.69
25	3	75.42	3	81.69	3	81.69

Table 7. Detection Results with Permissions Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	16	86.43	31	94.49	23	92.83
5	4	83.10	4	86.04	4	86.04
10	5	85.28	4	85.53	4	85.53
15	4	85.53	4	85.53	4	85.53
20	2	81.82	2	81.82	2	81.82
25	2	81.82	2	81.82	2	81.82

Table 8. Detection Results with Permissions Ranked in Descending Order.

Results with Intents: Now, we review the results if we use intents alone for detection. We use the intents' ranking (both increasing and decreasing order) in the proposed detection algorithm to get the best set of intents that gives better accuracy.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

Tables 9 and 10 summarize the detection results with intents. Table 9 shows the results when intents are ranked in ascending order. We get the best accuracy of 85.53% at 0% threshold with SVM classifier at best set of 21 intents. Similarly, from Table 10, when intents are ranked in descending order, we get the best accuracy of 83.35% with Random Forest classifier at best set of 28 intents. Hence, we can conclude that when we use intents for detection, we get the best accuracy of 85.53% with SVM classifier at best set of 21 intents. We also observe that the best accuracy with intents alone is lower than obtained with permissions alone.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	19	83.48	20	85.15	21	85.53
5	3	75.42	3	79.13	3	79.13
10	1	69.40	3	73.62	3	73.62
15	1	69.40	3	73.62	3	73.62
20	1	69.40	3	73.62	3	73.62
25	1	69.40	3	73.62	3	73.62

Table 9. Detection Results with Intents Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	16	75.80	28	83.35	24	82.71
5	4	75.16	7	78.87	7	79.00
10	3	72.47	4	73.50	4	73.50
15	3	72.47	4	73.50	4	73.50
20	3	72.47	4	73.50	4	73.50
25	3	72.47	4	73.50	4	73.50

Table 10. Detection Results with Intents Ranked in Descending Order.

Results with Hardware Components: Next we review the results if we use hardware components alone for detection. Tables 11 and 12 summarize the detection results with components. Table 11 shows the results when components are ranked in ascending order. We get the best accuracy of 76.31% at 0% threshold with SVM classifier at best set of 17 hardware components. Similarly, from Table 12, when hardware components are ranked in descending order, we get the best accuracy of 76.06% with SVM classifier at 0% threshold at best set of 35 hardware components. Hence, we can conclude that when we use hardware components for detection, we get the best accuracy of 76.31% with SVM classifier at best set of 17 components. We also observe that the best accuracy with hardware components alone is lower than obtained with permissions and intents.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	10	74.90	15	75.42	17	76.31
5	9	73.24	14	74.39	18	75.03
10	9	73.24	14	74.39	18	75.03
15	13	72.47	14	72.60	20	73.75
20	13	72.47	14	72.60	20	73.75
25	13	72.47	14	72.60	20	73.75

Table 11. Detection Results with Hardware Components Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	22	59.15	34	75.29	35	76.06
5	22	59.15	33	74.01	34	74.65
10	22	59.15	33	74.01	34	74.65
15	22	59.15	32	72.47	33	73.11
20	22	59.15	32	72.47	33	73.11
25	22	59.15	32	72.47	33	73.11

Table 12. Detection Results with Hardware Components Ranked in Descending Order.

Results with Content Providers: Next we review the results if we use content providers alone for detection. Tables 13 and 14 summarize the detection results with content providers. Table 13 shows the results when content providers are ranked in ascending order. We get the best accuracy of 91.29% with SVM classifier at best set of 26 content providers. Similarly, from Table 14, when content providers are ranked in descending order, we get the best accuracy of 89.76% with SVM classifier at best set of 59 content providers. Hence, we can conclude that when we use content providers for detection, we get the best accuracy of 91.29% with SVM classifier at best set of 26 content providers. We also observe that the best accuracy with content providers alone is lower than obtained with permissions and higher than obtained with intents and hardware components.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	14	89.76	12	89.50	26	91.29
5	19	87.71	16	87.20	34	89.63
10	19	87.71	16	87.20	34	89.63
15	19	87.71	16	87.20	34	89.63
20	19	87.71	16	87.20	34	89.63
25	19	87.71	16	87.20	34	89.63

Table 13. Detection Results with Content Providers Ranked in Ascending Order.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	7	51.22	58	89.50	59	89.76
5	7	51.22	57	87.20	58	87.71
10	7	51.22	57	87.20	58	87.71
15	7	51.22	57	87.20	58	87.71
20	7	51.22	57	87.20	58	87.71
25	7	51.22	57	87.20	58	87.71

Table 14. Detection Results with Content Providers Ranked in Descending Order.

Results with Broadcast Receivers: Further, we review the results if we use broadcast receivers alone for detection. Tables 15 and 16 summarize the detection results with broadcast receivers at different support thresholds. Table 15 shows the results when broadcast receivers are ranked in ascending order. We get the best accuracy of 89.63% with SVM classifier at best set of 25 broadcast receivers. Similarly, from Table 16, when broadcast receivers are ranked in descending order, we get the best accuracy of 83.61% with SVM classifier at every threshold. This accuracy is achieved at best set of 62 broadcast receivers. Hence, we can conclude that when we use broadcast receivers for detection, we get the best accuracy of 89.63% with SVM classifier at best set of 25 broadcast receivers. We also observe that the best accuracy with broadcast receivers alone is lower than obtained with permissions and content providers, and higher than obtained with intents and hardware components.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	23	88.86	20	88.86	25	89.63
5	25	88.48	23	88.99	27	89.63
10	10	78.75	30	83.99	36	84.89
15	10	78.75	30	83.99	36	84.89
20	10	78.75	30	83.99	36	84.89
25	10	78.75	30	83.99	36	84.89

Table 15. Detection Results with Broadcast Receivers Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	62	82.97	59	83.10	62	83.61
5	62	82.97	59	83.10	62	83.61
10	62	82.97	59	83.10	62	83.61
15	62	82.97	59	83.10	62	83.61
20	62	82.97	59	83.10	62	83.61
25	62	82.97	59	83.10	62	83.61

Table 16. Detection Results with Broadcast Receivers Ranked in Descending Order.

Results with Services: Next, we review the results if we use services alone for detection. Tables 17 and 18 summarize the detection results with services at different support thresholds. Table 17 shows the results when services are ranked in ascending order. We get the best accuracy of 89.76% with SVM classifier at best set of 23 services. Similarly, from Table 18, when services are ranked in descending order, we get the best accuracy of 83.48% with SVM classifier at best set of 61 services. Hence, we can conclude that when we use services for detection, we get the best accuracy of 89.76% with SVM classifier at best set of 23 services. We also observe that the best accuracy with services alone is lower than obtained with permissions and content providers, and higher than obtained with intents and hardware components.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	23	89.76	22	89.37	23	89.76
5	23	89.37	21	89.12	23	89.37
10	22	88.61	21	88.22	22	88.61
15	23	87.58	19	86.68	23	87.58
20	32	84.63	30	83.99	32	84.63
25	32	84.63	30	83.99	32	84.63

Table 17. Detection Results with Services Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	61	83.35	59	83.23	61	83.48
5	61	83.35	59	83.23	61	83.48
10	61	83.35	59	83.23	61	83.48
15	61	83.35	59	83.23	61	83.48
20	61	83.35	59	83.23	61	83.48
25	61	83.35	59	83.23	61	83.48

Table 18. Detection Results with Services Ranked in Descending Order.

From the above results, we can conclude that we get the highest accuracy of 94.49% with 31 permissions. Moreover, we find that SVM and Random Forest classifiers gave relatively better accuracy than Naive Bayes, for all the individual features.

C. Detection Results with Combined Features

In this subsection, we analyze the results when we combine all the manifest file components, rank them all in ascending and descending order, and then apply the proposed detection algorithm. The proposed approach aims to find the best set consisting of all the manifest components that give relatively better accuracy than any other set of features. We summarize the detection results with combined manifest components in Tables 19 and 20.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

Table 19 highlights the detection results when we arrange all the manifest file components in increasing order of their relative frequencies. We observe that we get the best accuracy of 95.90% at 10% support threshold with 36 manifest features. In this best set of 36 features, there are 10 permissions, 5 intents, 6 services, 4 broadcast receivers, 7 content providers, and 4 hardware components. We analyze that eliminating the features (ranked in ascending order) at the 10% threshold increases the accuracy as compared to the 0% threshold (where no feature is eliminated). Hence, we can argue that eliminating the irrelevant features with a support threshold helps in improving the detection accuracy. From Table 20, where features are ranked in descending order of their relative frequencies, we observe that the proposed approach gives the highest accuracy of 94.70% with SVM classifier at support thresholds of 10%, 15% and 20%. This accuracy is obtained with 58 manifest features. However, this accuracy is lower than obtained with features ranked in increasing order of frequencies.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	35	90.56	32	94.48	36	94.39
5	35	90.56	31	94.21	39	95.13
10	36	90.78	33	94.89	36	95.90
15	36	90.78	33	94.89	38	95.90
20	39	91.46	33	94.89	33	95.39
25	35	90.56	33	94.89	33	95.39

Table 19. Detection Results with Combined Manifest Components Ranked in Ascending Order.

Threshold (in %)	Different Machine Learning Classifiers					
	Naive Bayes		Random Forest		SVM	
	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)	Best Set	Accuracy (in %)
0	45	89.66	52	93.28	56	93.79
5	45	89.66	51	93.28	59	93.23
10	47	89.88	53	93.26	58	94.70
15	47	89.88	53	93.26	58	94.70
20	49	90.68	53	93.26	58	94.70
25	55	89.26	53	93.26	53	93.70

Table 20. Detection Results with Combined Manifest Components Ranked in Descending Order.

To summarize the detection results, Table 21 highlights the highest accuracies obtained with individual features and all the features combined. As can be seen from the table, we observe that combining the manifest features gives better detection accuracy than individual manifest features.

Features Used for Detection	**Highest Accuracy (in %)**	**Best Set of Features**
Permissions	94.49	31 Permissions
Intents	85.53	21 Intents
Hardware Components	76.31	17 Components
Content Providers	91.29	26 Providers
Services	89.76	23 Services
Broadcast Receivers	89.63	25 Receivers
All Manifest Features	**95.90**	**36 Features (10 permissions, 5 intents,6 services, 4 broadcast receivers, 7content providers, and 4 hardware components)**

Table 21. Comparison of Detection Results.

D. False Result Analysis

In this section, we review the false positives, i.e., normal apps detected as malware and false negatives, i.e., malware apps detected as normal. Because we have got the highest accuracy of 95.90% with SVM classifier on all manifest features combined, we discuss the false results with reference to this highest accuracy achieved. We observe that many normal apps with the functionality of blocking phone calls and SMS have been detected as malicious. For instance, normal apps available on Play Store such as *Calls Blacklist-Call Blocker*, *AntiNuisance- Call Blocker and SMS Blocker*, *Call Control*, etc., have been detected as malicious because of the presence of dangerous functionalities of blocking phone calls and SMS. Hence, to correctly detect such apps, we need to analyze, in addition to manifest components, the Java source code and the API calls being generated within the code. Furthermore, some malicious apps such as *AnserverBot*, and *BaseBridge* have been detected as normal because such apps download malicious components at run time. The proposed method is a static approach, hence, the method does not analyze the run time behavior of the apps. To correctly detect such apps, we need to include dynamic analysis as well, such as observing dynamic calls or network traffic of the apps. However, this will increase the computational complexity of the detection model.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a static model named *SFDroid* to detect malicious Android applications by analyzing manifest file components. The proposed model aimed to rank each of the manifest component using the concept of relative frequency, i.e., by comparing its frequency in both malware and normal dataset.

We constructed, for each of the manifest components, two rankings, one highlighting the features significantly present in malware apps and the other representing the features significantly present in normal apps. Thereafter, we proposed a novel algorithm to find the best set of manifest features that gives highest detection accuracy. The proposed detection algorithm applied several machine learning classifiers on the ranked list of manifest features to get their best set. We achieved the accuracy of 95.90% with the proposed model on the best set of 36 features. In our future work, we will analyze, in addition to the manifest components, the Java source code and API calls of the apps to further improve the detection accuracy.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

ACKNOWLEDGMENT

The authors would like to thank the authors of Genome, Drebin and Koodous malware datasets 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/platform-market-share/desktop-mobile-tablet/.
2. App Stores List(2020), Available Online. https://www.businessofapps.com/guide/app-stores-list/
3. Critical Warning Issued Regarding 10 Million Samsung Phone Updates, Available Online .https://www.forbes.com/sites/daveywinder/2019/07/05/critical-warning-issued-regarding-10-million-samsung-phone-updates/.
4. Agent Smith virus hides in WhatsApp, infests 1.5 crore Android phones in India, Available Online. https://www.indiatoday.in/technology/news/story/agent-smith-virus-whatsapp-infects-android-phones-in-india-what-is-it-1566668-2019-07-11
5. R. Taheri et al., "Similarity-based Android malware detection using Hamming distance of static binary features", Future Generation Computer Systems, vol. 105, pp. 230-247, 2020.
6. J. Qiu et al., "A3CM: Automatic Capability Annotation for Android Malware," IEEE Access, vol. 7, pp. 147156-147168, 2019.
7. 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.
8. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket", NDSS,2014.
9. H. Fereidooni, M. Conti, D. Yao and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications," 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Larnaca, 2016.
10. A. Firdaus, N.B. Anuar, A. Karim,and M. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection", Frontiers of Information Technology and Electronic Engineering, vol. 19, pp. 712–736, 2018.
11. 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.
12. J. Li et al., "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, 2018.
13. 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.
14. A. Sangal, and H. K. Verma, "A Static Feature Selection-based Android Malware Detection Using Machine Learning Techniques", International Conference on Smart Electronics and Communication , 2020.
15. S. K. Jhansi, et al., "Feature Selection and Evaluation of Permission-based Android Malware Detection", 4th International Conference on Trends in Electronics and Informatics, 2020.
16. K. Khariwal, J. Singh and A. Arora, "IPDroid: Android Malware Detection using Intents and Permissions," 4th IEEE WorldS4, London, United Kingdom, pp. 197-202, 2020.
17. S. Feldman, D. Stadther and B. Wang, "Manilyzer: Automated Android Malware Detection through Manifest Analysis," 11th IEEE MASS, Philadelphia, PA, pp. 767-772, 2014.
18. M. Kumaran and W. Li, "Lightweight malware detection based on machine learning algorithms and the android manifest file," IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, pp. 1-3, 2016.
19. 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.
20. T. Kim et al., "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features", IEEE Transactions on Information Forensics and Security, vol. 14, 2019.
21. H. Zhua et al., "DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model", Neuro- computing, vol. 272, pp. 638-646, 2018.
22. K. Elish et al., "Profiling user-trigger dependence for Android malware detection", Computers & Security, vol. 49, pp. 255-273, 2015.
23. M. Zhang et al., "Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs", ACM CCS, 2014.
24. Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics- based detection of android malware through static analysis", 22nd ACM SIGSOFT Symposium on Foundations of Software Engineering, 2014.
25. W. Wang et al., "DroidEnsemble: Detecting Android Malicious Applications With Ensemble of String and Structural Static Features," IEEE Access, vol. 6, pp. 31798-31807, 2018.
26. 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.
27. L. N. Vu, and S. Jung, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification", IEEE Access, vol. 9, pp. 39680-39694, 2021.
28. 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.
29. A. Saracino, D. Sgandurra, G. Dini and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Preven- tion," in IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 1, pp. 83-97, 2018.
30. 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.
31. M. Jaiswal, Y. Malik and F. Jaafar, "Android gaming malware detection us- ing system call analysis," 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, pp. 1-5, 2018.
32. 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), Nan- tucket, MA, USA, pp. 1-8, 2018.
33. J. Ribeiro, F. B. Saghezchi, G. Mantas, J. Rodriguez, and R. A. Abd-Alhameed, "HIDROID: Prototyping a Behavioral Host-Based Intrusion Detection and Prevention System for Android", IEEE Access, vol. 8, pp. 23154-23168, 2020.
34. 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.
35. 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.
36. 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.
37. A. Arora, S. Garg, and S.Peddoju,"Malware detection using network traffic analysis in android based mobile devices", 8th IEEE NGMAST,2014.
38. A. Arora, and S. Peddoju, "Minimizing Network Traffic Features for Android Mobile Malware Detection", 18th ACM ICDCN, 2017.
39. S. Rahmat, Q. Niyaz, A. Mathur, W. Sun and A. Y. Javaid, "Network Traffic-Based Hybrid Malware Detection for Smartphone and Traditional Networked Systems," 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, New York City, NY, USA, pp. 0322-0328, 2019.
40. 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.
41. A. Mahindru, A. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques", Neural Computing & Applications, 2020.
42. A. Mehtab et al., "AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis", Mobile Networks and Applications, vol. 25, pp. 180–192, 2020.

43. 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.
44. Y. Shyong, T. Jeng and Y. Chen, "Combining Static Permissions and Dynamic Packet Analysis to Improve Android Malware Detection," 2nd International Conference on Computer Communication and the Internet (ICCCI), Nagoya, Japan, pp. 75-81, 2020.
45. A. Arora, and S. Peddoju, ”NTPDroid: A Hybrid Android Malware Detec- tor Using Network Traffic and System Permissions”, 17th IEEE TrustCom, 2018.
46. A. Arora, S. Peddoju, V. Chauhan, and A. Chaudhary, ”Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learn- ing”, 24th ACM MobiCom, 2018.
47. S. Arshad et al. "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," IEEE Access, vol. 6, pp. 4321-4339, 2018.
48. W. Zhang, H. Wang, H. He, and Peng Liu, “DAMBA: Detecting Android Malware by ORGB Analysis”, IEEE Transactions on Reliability, vol. 69, pp. 55-69, 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”.

AUTHORS PROFILE

Gourav Garg, is a student at Delhi Technological University, Delhi, India. He is pursuing Bachelor of Technology in Mathematics and Computing branch from the Department of Applied Mathematics. He has dsone research work and published a paper in the field of Mobile Security and Android Malware Detection, using static components. He has done project work in the fields of Machine Learning, Deep Learning, and Android App Development. He has gained work experience at Paytm as an SDE intern. He has a creative mind and is always eager to learn new skills and apply them to solve problems that make people's lives easier and better.

Ashutosh Sharma, is a student at Delhi Technological University. He is pursuing Bachelor of Technology in Mathematics and Computing from the Department of Applied Mathematics. He has immense interest in software Development and research in mobile malware Detection and Network Traffic Analysis. He is a tech enthusiast and an innovative thinker who's always up to learning new technologies which would improve the standard of life. He is a quick learner and always ready to work on projects that requires technical skills for solving real-life problems and Presentations skills such as programming. His goal is to learn and evolve as much as possible.

Anshul Arora is currently working as Assistant Professor in Discipline of Mathematics and Computing, Delhi Technological University Delhi, India. He is pursuing Ph.D. from Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India. Previously, he has done M.Tech in Computer Science and Engineering from IIT Roorkee and B.Tech in Computer Engineering from Kurukshetra. His areas of research include Mobile Security, Mobile Malware Detection, and Network Traffic Analysis. He has published several papers in the field of Android malware detection. He is also a reviewer for various journals such as IEEE Transactions on Information Forensics and Security, IEEE Access, Computers and Security, Elsevier, etc.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication