Minimization of Attributes for the Detection of Vulnerabilities in Android Applications

Jigna Rathod, Dharmendra Bhatti

Uka Tarsadia University, India

Corresponding author: Dharmendra Bhatti, Email: dgbhatti@utu.ac.in

Bluetooth, MMS, SMS, e-mail, and other mobile-specific applications may constitute deliberate threats, resulting in financial losses as well as the revealing of sensitive user data. To circumvent this, different approaches have lately been applied, such as static analysis-based detection or dynamic analysis-based detection. However, some of these strategies have become saturated as a result of massive increase in malware code authoring techniques. As a result, there is a need for an effective android privacy-leakage vulnerability detection technique, for which this research used hybrid features such as standard permissions, non-standard permissions, system call traces, and network traffic to conduct experimental tests. We presented a method to prioritize the attributes in order to select the most prominent features, with the purpose of minimizing the number of attributes to be investigated while retaining good detection accuracy. To rank the attributes in this dimension, we used a variety of statistical approaches. The findings of the experiments reveal that selecting attributes for vulnerability detection resulted in higher detection accuracy than assessing Accuracy using all attributes. To assess the efficacy of the suggested attribute selection procedure, we deployed a neural network. With the prominent attributes chosen by the proposed algorithm, the neural network attained an accuracy of 96.41 percent, which is 2% higher than the accuracy we achieved with all of the attributes combined.

Keywords: Android, Mobile Phones, System Calls, Network Traffic, Permissions, Feature Ranking, Neural Networks, Malware, Detection, Privacy-leakage.

2021. In Rahul Srivastava & Aditya Kumar Singh Pundir (eds.), *New Frontiers in Communication and Intelligent Systems*, 475–487. Computing & Intelligent Systems, SCRS, India. https://doi.org/10.52458/978-81-95502-00-4-49

1 Introduction

Android vulnerability, also known as Android vulnerable application, is always developed with the intent of stealing sensitive information from users or harming Android devices. Since 2012, Android has remained the leading operating system [1]. Android's market share was 84.1 percent in 2020, and it is expected to increase to 84.9 percent by 2025 [2]. Mobile devices are now used for more than just sending messages or dialing numbers; they are also used for gaming, downloading files, social networking, chatting, web browsing, online shopping, and online financial transactions. Mobile users store sensitive data in their devices, such as bank account numbers, contacts, usernames and passwords for online banking, and credit card numbers. Because Android is an open-source operating system, many contributors are contributing to its development in order to improve the graphical interfaces and optimize the performance. In order to gain more popularity, a large number of refined versions are being offered to mobile device users, and their evaluations will allow it to become more dominant and convenient. Nonetheless, this has caught the interest of many vulnerable code writers in gaining unsigned access to the user's sensitive information.

There are a lot of Android apps that are susceptible. In order to avoid malware penetration, Google provides the Bouncer [3] technique to detect dangerous Android applications during the application submission process. However, malware that loads its malicious content during the runtime cannot be identified by Google's Android malware detection approach during installation, thus a variety of approaches have lately been utilized to limit this, including static and dynamic analysis-based detection. Because static analysis-based detection has limits, such as malware that uses obfuscation, encryption, or the use of runtime libraries going undetected, we employed dynamic analysis in our research. As a result, a proper approach to identifying Android malware is required, which is addressed in this work by experimental tests with hybrid features such as normal permissions, non-standard permissions, network traffic, and system call tracing. The exceptional attributes are selected using a novel technique introduced in this work. The proposed framework was created using neural network algorithms and is based on the concept of feature selection techniques. To find the optimum attributes for training, we applied two different approaches: attribute ranking and attribute subset selection. Gain-Ratio [4], OneR [4], Information Gain [5], and Chi-Squared Test [5] are four distinct attribute ranking methodologies we look at in this research to find the top attributes. In addition, we examine CFS Subset Evaluation [6] and Filtered Subset Evaluation [7], two unique attribute subset selection algorithms. We used neural network algorithms to train them once we chose the best trait.

By collecting data from the emulator, we attempted to evaluate the network traffic behavior of android malware in this study. By integrating network traffic attributes with permission and system call traces, as well as utilizing them alone, we evaluated the impact of network traffic attributes in the discovery of vulnerabilities. Initially, we used feature selection algorithms to train our model by selecting attributes. Furthermore, the permissions, network traffic attributes, and system calls have been prioritized such that only a subset of the whole attribute set can be used for vulnerability identification.The advantage of this type of attribute prioritization is that it allows for the decrease of attributes that may cause detection problems. The outcomes of this study, which used statistical techniques to rank attributes and locate subsets of traits, were not particularly noteworthy.

Below is a summary of the remainder of the paper. Previous research on Android malware detection is discussed in Section 2. Section 3 describes the technique employed in this study. The performance of the suggested method is discussed in Section 4.

2 Related Work

According to a recent literature review, there are several approaches available to detect vulnerability in Android applications. Some have used signature-based detection methods, but their main limitation is that they can only detect known malwares whose signatures are present in the repository. Because signature-based approaches have limitations, many tactics, such as static and dynamic analysis, have emerged to detect malicious Android applications [8-11].

Static analysis has been used by many authors. The majority of existing static analysis-based android malware detection techniques make use of permissions and API-calls as features, with a few others making use of intents, metadata information, row op-codes, and so on. As a result, we chose permission as the static attribute out of all the attributes in this work. Sahal [12] proposed a permission-based Android malware detection technique. To select features, the TF-IDF-CF weighting method is used. They compared TF-IDF-CF to Information Gain and PCA and found that TF-IDF-CF outperformed the others. They tested the effectiveness of feature selection methods such as J48, Naive Bayes, SVM, and KNN and found that SVM achieved 98.6 percent accuracy.

Jung [13], proposed an Android malware detection system based on API-call features, and they identified a subset of Android APIs that are effective as features. They have created two top-rankedlists: one for the top API calls used by benign samples and another for the top API calls used by malicious samples. According to their observations, the APIs in the benign list is quite different from the APIs in the malicious list. They used a random forest classifier and achieved a 99.98 percent accuracy.

In the past, many writers have employed a combination of static analysis features to detect malware. Standard permissions, non-standard permissions, and API call features were used to create various feature sets such as API calls with standard permissions, API calls with non-standard permissions, and API calls with both standard and non-standard permissions. Singh [3] focused on static analysis by considering standard permissions, non-standard permissions, and API call features, which were used to create various feature sets such as API calls with standard permissions, API calls with non-standard permissions, and API calls with standard and non-standard permissions. They chose the top characteristics using four techniques: BI-Normal Separation, Mutual Information, Relevancy Score, and the kullback-Leibler method. They employed a linear support vector machine classifier to assess the performance. According to their findings, the combined features set indicated by the BI- Normal Separation technique had the maximum accuracy of 99.6%.

Only a few authors have employed dynamic analysis. System call traces, network traffic, control flow graphs, and other data are used in existing dynamic analysis research. In this study, we used system call and network traffic occurrences as attributes. Aminuddin [14] presented an Android malware detection mechanism based on dynamic analysis of system calls. They attained an accuracy of 81.2 percent using the random forest technique. [15] Radoglou-Grammatikis demonstrated an intrusion detection method for identifying aberrant behavior in Android mobile devices. They used network traffic and an artificial neural network as a classifier as a feature.

Some of the authors have presented ways that take both static and dynamic analysis into account. The extent to which adversarial attacks can degrade classifier performance was examined by Anupama [16]. They proposed a system built on static and dynamic properties learned using machine learning and deep neural networks. The researchers discovered that integrating static and dynamic features enhances accuracy. They achieved 100 percent accuracy using CART and SVM for hybrid features, 97.59 percent accuracy using SVM for static features, and 95.64 percent accuracy using Random Forest for dynamic features. They achieved 99.28% accuracy with static features, 94.61 percent accuracy with dynamic features, and 99.59 percent accuracy with hybrid features and deep neural networks. They

assessed the classifier's resistance to evasion and poisoning attacks.

Afonso et al [17] propose another Android malware detection system that combines static and dynamic features such as API calls and system call traces with machine learning to detect malware with a high detection rate. Yerima [18] presented a longitudinal study of machine learning classifier performance for android malware detection. They extracted features from applications that were first seen between 2012 and 2016. When tested on sets of samples from a later time period, the experimental results of their study show progressive diminishing performance. When older models were tested on newer app samples, they discovered that misclassification of benign samples as malicious increased significantly more than the fall in correct classification of malicious apps.

Yang [19] proposed a CNN-based malware detection approach based on images. They took Dalvik Bytecode as a feature and converted it to image. The transformed images are then fed into CNN for malware detection. According to the results of the experiment, they achieved an accuracy of 93 percent.

By considering previous approaches for android vulnerability detection, an experimental study was conducted to compare the method's evaluation with other existing methods in Table 1. A summary is provided below.

- Using dynamic analysis, standard and non-standard permissions were extracted from the manifest file, system call traces and network traffic were recorded, and Java code was written to support this. These permission sets are then combined with system calls and network traffic to create a combined attribute set. Non-standard permissions are declared by application developers, whereas standard permissions are predefined in the authoritative android library.
- An algorithm is proposed in section 3.1 to select the nominal set of attributes.

Table 1. Comparison of proposed approach with existing approach

Method	Attributes	Attribute Selection Techniques	Accuracy (%)
Our Approach	**Permissions + System Calls + Network Traffic**	**Proposed Approach**	**96.41%**
Shahzad, R. K. [20]	Permissions + Intents + Hardware Components	Not Mentioned	94%
Alshahrani, H [21]	System Calls + System Information + Network Traffic + Requested Permission	Not Mentioned	95%
Afonso, V. M. [17]	API Calls + System Calls	Not Mentioned	95.66%

3 Proposed Methodology

Figure 1 depicts the proposed operation flow diagram, in which dynamic analysis is used to detect android vulnerabilities. Static analysis was used because recent malwares are vulnerable to code obfuscation, encryption, and the use of runtime libraries. So, in order to overcome the limitations of static analysis, dynamic analysis is proposed in this work. Static analysis does not necessitate the execution of an application, whereas dynamic analysis necessitates the execution of an application, which takes significantly longer than static analysis but allows for the observation of the application's runtime behavior. The first step in dynamic analysis is to select a dataset that contains a uniform distribution of benign and malicious applications. To extract attributes from an Android application, an

independently developed monitoring component is installed in the device, and that monitoring component is responsible for capturing network traffic, system calls, and permissions requested by the application. Section 3.1 discusses the attribute selection algorithm, which is proposed to select distinguishable attributes. These characteristics were combined to create training data for the neural network. Our dataset contains total of 329 attributes. For example, the average packet size in normal apps is 1-2515, while it is 1-18143 in vulnerable apps. Normal and vulnerable apps share the same permissions and system calls. There is no distinguishable attribute set because attributes and ranges overlap in both types of apps. As there is no such distinguishing attribute, determining a nominal set of attributes that improves detection accuracy is the most difficult task. The attributes should be ranked in order to distinguish the best subset of attributes. We used various feature selection and ranking algorithms to rank the attributes, which resulted in higher detection accuracy. To rank the attributes, we used various feature ranking algorithms such as Information Gain, Gain Ratio, OneR, and correlation. In addition, we used subset selection algorithms like CFSSubset and FIlteredSubset.

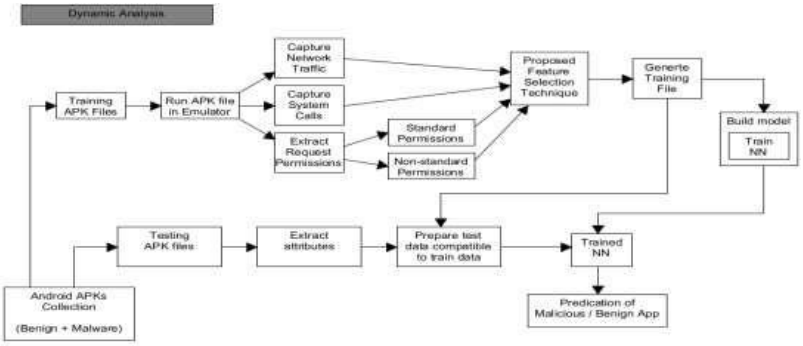

Fig. 1. Proposed Operational Flow Diagram

To perform dynamic analysis of benign and vulnerable applications, a Google Nexus 4 emulator running Android version 4.4 is used. As a result of dynamic analysis, the tracking component installed in the emulator will record the permission, system calls, and network traffic generated by each of the applications. Each sample will run in the emulator for 20 minutes. There are a total of 2008 samples used for training and 502 samples used for testing. The monitoring component will capture network traffic in .pcap format, which can be easily analyzed with the Wireshark tool. Each system call trace, permission, and network traffic file is sent to a desktop computer running Windows 10 with 8 GB RAM and a 256 GB SSD for further analysis. The attribute selection algorithm proposed in this paper is discussed in the following subsection.

3.1 Selection of Attributes

The selection of relevant attributes among the ranked attributes is obtained through CHI tests, I.G., G.R., OneR, correlation, and subset features obtained through CFS subset, Filtered Subset, and are presented in algorithm 1. We will have seven various rankings of attributes obtained from mentioned algorithms. As input, seven different ranking algorithms are used to compute the nominal set of top-ranked attributes from all seven rankings. We want to prioritize the attributes so that only a few of them are used for vulnerability detection rather than the entire set of 329 attributes. I made three lists: the Primary List (Plist), the Subset List (SubsetList), and the Nominal Attributes List (NominalAList). The ordinary attributes from all five rankings are contained in Plist, the common attributes from two subset

selection algorithms are contained in SubsetList, and the nominal set of attributes is contained in NominalAlist. We evaluate detection performance in terms of ACC.

The goal of this work is to find the highest ACC from the nominal set of attributes used. As shown in algorithm 1, we find the common attributes from the top-ranked attributes from all seven lists for each value of I. The common attributes from each of the five rankings are listed in Temp Common, and this set is then added to Plist. In Plist, identical attributes are removed. This step is necessary because, as I increment, Temp Common contains the attributes from the top-ranked attributes of all the lists containing attributes that have already been added to Plist. If we discover any ordinary attribute during any iteration of I, Plist is updated with the newly discovered common attributes. For the detection of test data using the attributes in Plist, a neural network detection algorithm is used.

Algorithm 1 Selection of nominal attributes from ranked features

Input: Five rankings of attributes: IGList, GRList, CHIList, OneRList, CorelationList obtained from IG, GR, CHI, OneR, Correlation respectively

Two subsets of features: CFSList, FilteredList obtained from CFSSubset, Filtered Subset respectively

Output: Nominal set of attributes which gives best detection accuracy

PList <- {}, NomialFList <- {}, Amax = 0; common_list <- common attributes of all 5 ranking algorithms, SubsetList <- {}

```
For I = 1 -> common_list.length do
        Temp_common <- (IGList)I∩ (GRList)I ∩ (CHIList)I ∩ (OneRList)I ∩
        (CorelationList)IDiff = PList – Temp_common
        PList =
        PList U
        (Diff)IF I
        <= 41
                Subset_Temp <- (CFSList)I ∩
        (FilteredList)IPList = PList U (SubsetList –
        Subset_Temp)
        Apply normalization on data
        Run NN on test data using
        attributes in PListCalculate ACC of
        detection results
        If ACC > Amax
           then Amax
           <- ACC
           NominalALis
           t <- PList
    End if
    End for
Return Nominal A List
```

4 Performance Assessment

4.1 Dataset used

A dataset is required to carry out the experimental work. The Drebin dataset [22], which contains 5560 samples, was used to collect vulnerable apps. The benign samples were obtained from the Google Play store and the CICMALANAL2017 [23] dataset. 2008 samples from each category (Benign and Malware) were used to train the neural network.

For the current work, Accuracy, True positive rate and False positive rate were considered as the evaluation as shown in Equation 1.

$$\text{Accuracy} = \frac{TP+TN}{TP-TN-FP+FN} \quad TPR = \frac{TP}{TP+FN} \quad FPR = \frac{FP}{FP+TN} \tag{I}$$

True Positive (TP) represents the malicious samples correctly classified as malicious. True Negative (TN) represents the benign samples correctly classified as benign. False Positive (FP) represents the benign samples misclassified as malicious. False Negative (FN) represents the malicious samples misclassified as benign. Table 2 represents the confusion matrix which is used to compute the evaluation metrics.

Table 2. Confusion Matrix

Class	Prediction	
	Malware	**Benign**
Malware	True Positive (TP)	False Positive (FP)
Benign	False Negative (FN)	True Negative (TN)

In Section 4.2, we discussed the attribute rankings obtained using I.G., G.R., Chi squared test, OneR, and Correlation on the entire set of 329 attributes. The detection results are also discussed in this section, where we show how using the nominal attribute set improves accuracy over the entire attribute set. Section 4.3 discusses the efficacy of nominal attribute set on unknown samples. Except for a few exceptions, the results show that unknown samples with a nominal set of attributes can be detected with the same accuracy as all features. Section 4.3 demonstrates that the nominal attribute set produced by algorithm 1 that considers all seven rankings is a more appropriate set with higher detection accuracy than the other attribute set produced by individual rankings. Last section discusses the limitations of this approach.

4.2 Ranking of Attributes

Values are calculated for all ranking algorithms by considering both normal and vulnerable app attributes. AUTHENTICATE ACCOUNTS permission attribute is used only by normal samples, so, this has the lowest chi value. On the other hand, the range of attributes Average number of packets sent per second for both normal and vulnerable samples is very similar: 5-10553 for normal samples and 8.1-10586 for vulnerable samples. Because the range of attributes is very similar, the I.G. for this attribute is the lowest while the CHI test has the highest value. As a result, it is possible that in one test, an attribute receives the highest priority while in another, it receives the lowest priority. As a result, we must select the attributes from all of the tests that will provide better detection accuracy.

We get the results as shown in Table 3 when we apply our attribute selection algorithm to all of the

ranked attributes obtained from all seven tests. The first column displays the value of I, i.e., common attributes from all seven lists, calculated by taking the top I attributes from all seven rankings. Table 3 shows that until k=9, there is no single common attribute from all seven lists. The second column contains the total number of attributes in the primary list derived from the top I attributes of each ranked list. This column will provide us with the final output of the number of attributes that are sufficient to produce a higher ACC score than the previous one. We get the same accuracy even after adding new attributes to the primary list, so we won't consider those attributes because our goal is to get a nominal attribute set that gives better detection results. Only when the ACC score is higher than the previous one, attributes are added to the final list.

At I=267 we get the highest ACC score considering 191 attributes as given in the second column of the Table 3. The results show that it correctly detects 283 normal samples and 202 vulnerable samplesout of a total of 298 and 204. Furthermore, by adding a greater number of attributes to this list, the same detection result is obtained until the last interaction in which attributes such as socket, REQUEST_PASSWORD_COMPLEXITY, Average number of packets sent per flow, and so on are added, which are ranked higher in OneR and lower in Information Gain. We obtained 94.57 percent accuracy with the entire attribute set, and 96.41 percent accuracy with the nominal attribute set of 191 attributes, which is better than the entire attribute set.

Table 3. Detection results of proposed attributes Selection algorithms

Value of I	**Total attributes in nominal subset**	**TPR%**	**FPR%**	**ACC**
1, 2, 3, 4, 5, 6, 7, 8, 9	-	-	-	**-**
10 to 19	1	0	0	59.36%
20 to 27	3	0.01	0	59.965%
28 to 33	6	0.01	0	60.15%
34 to 47	8	0.34	0	73.30%
48, 49	14	0.57	0.11	75.89%
50 to 54	15	0.70	0.19	76.49%
55 to 61	20	0.75	0.11	83.26%
62 to 81	23	0.75	0.07	85.65%
82 to 86	37	0.86	0.13	86.65%
87 to 133	41	0.82	0.10	87.05%
134 to 159	62	0.82	0.09	87.25%
160 to 180	70	0.86	0.09	88.84%
180 to 200	109	0.98	0.11	92.43%
201 to 203	110	0.98	0.10	93.02%
204 to 224	112	0.94	0.05	94.62%
224 to 235	152	0.95	0.05	94.82%
236 to 260	153	0.99	0.07	95.01%
261 to 266	179	0.98	0.06	95.61%
267 to 329	**191**	**0.99**	**0.05**	**96.41%**

4.3 Unknown Samples

To look over the usefulness of our proposed algorithm on unknown samples, we kept some vulnerable samples like AccuTrack, Adrd, Aks, GoldDream, FakeLogo, FakePlayer and FakeRun separated from the training as well as the testing dataset. The samples from these malware families are not included in the training database while we extracted attributes using dynamic analysis and therefore, they known as unknown. Total of 36 samples as given in Table 4 are tested using the set of 191 attributes selected by proposed algorithm. The detection consequences are shown in Table 5. It is possible that the simplest vulnerable pattern from the FakeLogo family is not correctly detected. Multilayer perceptron neural network with 191 attributes can able to detect malware from all the other unknown families given for testing. Except for this one sample, detection accuracy using 191 attributes is equivalent to that of using 329 attributes.

Table 4. List of malwares used in these experiments

Set I		Set II	
Malware	Total Samples	Malware	Used for testing
DroidDream	32	AccuTrack	02
DroidKungFu	172	Adrd	08
FakeInstaller	54	Aks	02
GinMaster	36	GoldDream	08
Iconosys	24	FakeLogo	04
Kmin	18	FakePlayer	04
Opfake	40	FakeRun	08
Plankton	116		
Gappusin	52		
GinMaster	36		
BaseBridge	308		

Table 5. Detection results on unknown samples

	AccuTrack	Adrd	Aks	GoldDream	FakeLogo	FakePlayer	FakeRun
Using 191 attributes	100%	100%	100%	100%	90%	100%	100%
Using 329 attributes	95%	93%	100%	95%	92%	98%	100%

4.4 Nominal Attribute Set

In this section, we will discuss the nominal set obtained from proposed algorithm contains 191 attributes is suitable set that can give high detection accuracy. We argue whether a different combination of or less than 191 attributes can provide better detection accuracy or not. To put this argument to the test, we created four other sets of 191 attributes, each of which was derived from a different ranking algorithm. Set1 is the set of the top 191 attributes obtained from Chi-Squared ranking, i.e., Set1 = Top 191 CHI Squared attributes. Set2 is the set of the top 191 attributes derived from Gain Ratio, i.e., Set2 = Top 191 GR attributes. Set3 is the collection of the top 191 attributes obtained from Information Gain, i.e., Set3 = Top 191 IG attributes. Set4 is the set of the top 191 attributes obtained from OneR, i.e., Set4 = OneR's top 191 attributes. Set5 = Top 191 attributes of proposed algorithm are the nominal attribute set obtained from the proposed algorithm. We compare the detection results from these five different

sets to determine which one has the highest detection accuracy. Table 6 compares the outcomes of the aforementioned experiments.

Table 6. Result comparison of proposed algorithm with existing feature ranking

algorithmsAlgorithm	ACC	FN	FP	TN	TP
Chi (Set 1)	83.46%	35	48	263	156
GR (Set 2)	90.23%	27	22	271	182
IG (Set 3)	85.65%	39	33	259	171
OneR (Set 4)	92.43%	36	7	262	197
Proposed Algorithm (Set 5)	**96.41%**	**16**	**2**	**282**	**202**

The above result compares all five sets' False Negative and False Positive rates. Set 5 produces the fewest false negatives and achieves the highest accuracy of 96.41 percent when compared to all other algorithms. Set5 correctly detects 202 vulnerable samples out of 204 samples, with only two samples misclassified as benign. Set 5 correctly detects 202 samples compared to 156 from Set 1, 182 from Set 2, 171 from Set 3, and 197 from Set 4. Set5 generates 16 false negatives compared to 35 from Set1, 27 from Set2, 39 from Set3, and 36 from Set4. As a result, Set5 with 191 attributes obtained from the proposed selection algorithm from various rankings is superior to other sets of the same size obtained from individual attribute rankings.

Next, we must demonstrate that this is the appropriate nominal attribute set, i.e., no other set with fewer than 191 attributes can produce better detection accuracy. When several attributes are considered as 70 from Set1, Set2, Set3, and Set5, we find that Set1 detects the highest true positives and an equal number of false negatives with Set 5 and lower false negatives when compared to Set2 and Set3, as shown in Table 7. The number of True positives from Set1 and Set5 is 185 and 177 respectively. In addition, the number of false negatives from Sets 1 and 5 is 27. Sets 1 and 5 produce 54 and 29 false positives, respectively. As a result, attributes in Set5 misclassified fewer malware samples as benign as attributes in Set1, and overall accuracy in Set5 is higher than in all other sets. We believe that in terms of security, our goal should be to ensure that no vulnerable application goes undetected. Keeping this in mind, we believe that employing 191 attributes with high detection accuracy is preferable to employing 70 attributes with low detection accuracy. As a result, we conclude that this is the nominal subset and that no other set of attributes with fewer than 191 attributes can produce better detection results.

Table 7. Result comparison of proposed algorithm using less attributes with other existing feature ranking algorithms

Algorithm	ACC	FN	FP	TN	TP
Chi – 70	83.86%	27	54	236	185
GR – 70	86.85%	39	27	280	156
IG – 70	79.68%	61	41	255	145
Proposed Algorithm – 70	**88.84%**	**27**	**29**	**269**	**177**

4.5 Limitations and Discussion

According to the results of the experiments, 202 of the 204 malware samples were successfully detected. There were 282 benign samples correctly detected and 16 false negatives out of a total of 298 samples.

All of these samples produce attributes that closely match those of malware. The average packet size, for example, is around 700 bytes; the average number of packets sent per flow is around 400; the permission sets used by the examples are similar to those used by malware; the frequency of system calls is high, and so on. As a result, normal samples with high attribute values can be classified as malware samples.

Malicious families such as BaseBridge and AnserverBot wait for system-related events such as a phone reboot to complete before launching their malicious payload at the device. Because we were aware of their behavior, all such occasions were completed concurrently while performing dynamic analysis. In many cases, normal users are not aware of such activities and thus do not complete that activity or do so after a long period of time; as a result, the malware, despite having infected the phone, will now no longer produce any network traffic and thus will evade detection for that period of time.

The proposed framework is an off-device detection framework, which means that dynamic analysis performed in the device is analyzed at another system rather than on a mobile device. Doing on-device detection consumes smartphone assets such as memory, CPU, battery, and so on, but it provides real-time detection.

The experiments are carried out on malware samples discovered around 2013. There has been a significant increase in the number of android malwares since then. As a result, we will broaden our research to include more recent malware samples.

We conducted a dynamic analysis in the emulator. Emulators have limitations, such as the inability to generate booting events, phone calls, or SMS-related events. So, if any vulnerable sample is waiting for such events to trigger their malicious activity, they will go undetected because they have not executed any malicious activity. As a result, dynamic analysis can be performed in real-world devices, which will be part of future work.

5 Conclusion and Future Work

In this work, we've developed our Android application system to capture the behavior of system call traces and network traffic, as well as the permission made by means of each application throughout their run time. We conducted experiments with various feature ranking and feature subset selection algorithms, but we were unable to obtain the desired results. So, in order to redesign the detection rate, we prioritized the attributes from a total of 329 attribute sets primarily based on the rankings from the IG, GR, CHI, OneR, and Correlation tests. As an attribute set, CFSSubset and FilteredSubset provide 44 attributes to detect malware. An algorithm is written to extract the most relevant attributes for vulnerability detection from the rankings obtained by IG, GR, CHI, OneR, Correlation, and CFSSubset and FilteredSubset subsets. The proposed algorithm extracts 191 relevant attributes from a total of 329, resulting in high detection accuracy. However, few samples remain undetected as they are using the same attribute range as benign. When the attribute prioritization algorithm is used, we achieve a detection accuracy of 96.41 percent.

The attacker is continually creating harmful software in order to steal the user's personal information. In this work, 38 different malware families were used to train the model, and our model can detect malware from known families. This study can also be developed to produce a multiclass malware training model that can detect more real-world dangerous apps.

This work will be expanded in the future in different aspects, such as the analysis of a larger data set, the ability to undertake dynamic analysis in real devices to bypass emulator restrictions, and so on. In addition, by adding new susceptible apps to our database, we can identify them.

References

[1] Mobile OS market share 2021. Statista

[2] IDC – Smartphone Market Share – Market Share

[3] Singh, A. K., Jaidhar, C.D. and Kumara, M.A. (2019). Experimental analysis of android malware detection based on combinations of permissions and API-calls. *Journal of Computer Virology and Hacking Techniques*, 15(3): 209-218.

[4] Plackett, R. L. (1983). Karl Pearson and the chi-squared test. *Int Stat Rev/Revue Int Stat*, 51(1):59-72.

[5] Hall, M. A. (1999). Correlation-based feature selection for machine learning. *Doctoral Dissertation*.

[6] Kohavi, R. (1997). Wrappers for feature subset selection. *Artif Intel*, 97: 273-324.

[7] Dash, M. (2003). Consistency-based search in feature selection. *Artif Intel*, 155-176.

[8] Zhang, J., Zhuang, X. and Chen, Y. (2019). Android Malware Detection Combined with Static and Dynamic Analysis. In *Proceedings of the 2019 the 9th International Conference on Communication and Network Security*, 6-10.

[9] Wang, Z. et al. (2019). Android malware detection based on convolutional neural networks. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, 1-6.

[10] Tuan, L. H., Cam, N. T. and Pham, V. H. (2019). Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis. *Cluster Computing*, 22(1): 1079-1085.

[11] Yerima, S. Y., Alzaylaee, M. K. and Sezer, S. (2019). Machine learning-based dynamic analysis of Android apps with improved code coverage. *EURASIP Journal on Information Security*, 2019(1): 1-24.

[12] Sahal, A. A., Alam, S. and Soğukpinar, I. (2018). Mining and Detection of Android Malware Based on Permissions. In *3rd International Conference on Computer Science and Engineering,* 264- 268.

[13] Jung, J. et al. (2018). Android malware detection based on useful API calls and machine learning. In *IEEE First International Conference on Artificial Intelligence and Knowledge Engineering*, 175-178.

[14] Aminuddin, N. I. and Abdullah, Z. (2019). Android trojan detection based on dynamic analysis. *Advances in Computing and Intelligent System*, 1.

[15] Radoglou-Grammatikis, P. I. and Sarigiannidis, P. G. (2017). Flow anomaly-based intrusion detection system for Android mobile devices. In *6th International Conference on Modern Circuits and Systems Technologies*, 1-4.

[16] Anupama, M. L. et al. (2021). Detection and robustness evaluation of android malware classifiers. *Journal of Computer Virology and Hacking Techniques*, 1-24.

[17] Afonso, V.M. et al. (2015). Identifying Android malware using dynamically obtained features. *J. Comput. Virol. Hacking Tech.* 11(1): 9–17.

[18] Yerima, S. Y. and Khan, S. (2019). Longitudinal performance analysis of machine learning based Android malware detectors. In *International Conference on Cyber Security and Protection of Digital Services*, 1-8.

[19] Xiao, X. (2019). An image-inspired and cnn-based android malware detection approach. In *34th IEEE/ACM International Conference on Automated Software Engineering*, 1259-1261.

[20] Shahzad, R. K. (2018). Android malware detection using feature fusion and artificial data. In *IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, 702-709.

[21] Alshahrani, H. et al. (2018). Android application threat detection using static and dynamic analysis. In *IEEE*

International Conference on Consumer Electronics, 1-6.

[22] Arp, D. et al. (2014). Drebin: effective and explainable detection of android malware in your pocket. In: *NDSS*.

[23] Lashkari, A. H. et al. (2018). Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In *the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology*.