Android Malware Detection (AMD) Based on Shallow Feature and Permission Correlation

Jung-San Lee[1,2], Yun-Yi Fan[1], Gah Wee Yong[1] and Ying-Chin Chen[1]
[1]Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan
[2]Cybersecurity Technology Institute, Institute for Information Industry, Taipei, Taiwan

leejs@fcu.edu.tw
a19990101152@gmail.com
yonggahwee@gmail.com
P1136287@o365.fcu.edu.tw (corresponding author)

Abstract: There are apps for everything, from online banking, social software to shopping. They have become one of the most important tools in our daily lives. This even implies that a mobile device has stored most of personal information, including photos, credit cards, and communications. If an intruder succeeds in hacking into the mobile device, all private properties must suffer from the leakage threats. Undoubtedly, a malware is the commonest tool used by an attacker to compromise a mobile phone. In particular, it is often disguised as a popular application through an obfuscated or packed form. That is the main reason why it is difficult to distinguish a malware from the legal ones. In this article, we have adopted machine learning technique to develop a static analysis mechanism for Android malware detection based on shallow feature and permission correlation (AMD). AMD first analyses the Application Programming Interfaces of the target to detect all possible and hidden privilege threats. It then filters this obfuscation information using permission correlation to eliminate noise and identify meaningful malicious indicators. The proposed approach leverages the correlation patterns between permissions and API calls to distinguish suspicious behaviours from legitimate ones. Thus, AMD can extract all the representative shallow features to achieve the high detection rate. Simulation results have shown that AMD can outperform related works under the datasets of CICAndMal2017 and CICMalDroid2020, which confirms the effectiveness of shallow features and permission correlation.

Keywords: Machine learning, Malware detection, Static analysis, Shallow features, Permission correlation

1. Introduction

With the explosive development of the Internet, thousands of apps have been designed for enriching people daily lives, including social media, communications, online banking and shopping. This clearly indicates that a mobile device has recorded most of personal information, including photos, credit cards, and communications. The popularity of these applications not only brings convenience to the public, but also shows how people rely on them. According to the Statcounter statistics, the open source adoption enables Android operating system to reach a 71.96% occupation in the global smart phone marketing at the end of 2024 [1].

The high coverage rate of Android operating system has brought smart phones the extremely diverse and numerous apps developments. Nevertheless, subsequently malicious attacks for Android mobile devices have occurred all the time, such as ransomware, adware, and spyware. They have caused a serious and great threat to user privacy and property. The Kaspersky statistic from 2021 to 2022 [2] points out that the continuous price decline of virtual currency has led to the increase of malware. But the numbers of detected malware have lowered down, as shown in Figure 1. It is due to that hackers who create malware have actively enhanced the concealment and infection attributes. A distribution of various malware has been depicted in Figure 2 [2]. Obviously, the number of Trojan-Dropper which is able to escape from malware detection to infect users lacking of security awareness has highly increased.

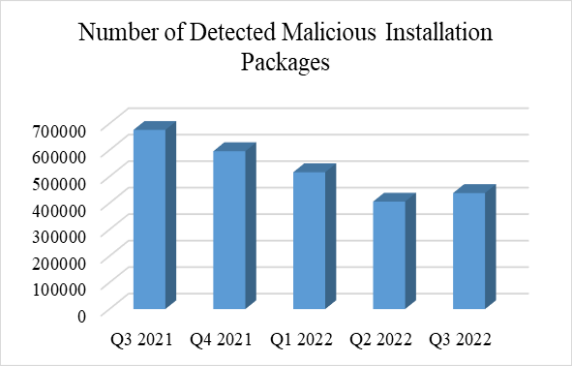

Figure 1: The numbers of detected malware

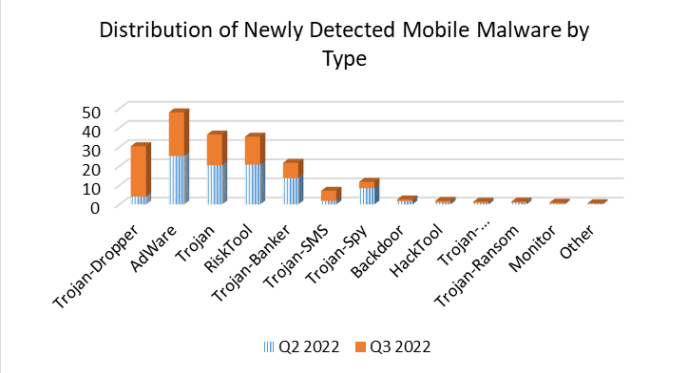

Figure 2: The distribution of mobile malware

The techniques of detecting malware could be classified into two types, including static analysis and dynamic analysis. Actually, hackers are of course familiar with these methodologies, which implies that they might be able to make up the malware to be a legal Apps through a serial of confusion or camouflage. That is the crucial challenge to malware detection. Without loss of generality, it takes the integrity verification and source code reviewing to achieve the determination, concerning the first category [3, 4]. The static features could be extracted from a malware without executing [5, 6]. For instance, researchers could confirm the integrity of APK (Android Package) or dig out the APK signature to classify whether the API (Application Programming Interface) or permission used are exceeded. However, Apps developers often obfuscate or pack the program before the APK is put on the online store to prevent from being illegal pirated. These additional operations usually lead to the detection ineffectiveness of static analysis. In 2021, Ateeq and Moreb have applied the neural network to design a malicious URL detection method [7]. Actually, it is easy to counterfeit a malicious URL through replacing some letters or digits by resembling symbols so that the length of malicious URL is often longer than that of a legal one. Therefore, it verifies whether an URL is valid or not via semantic analysis in [7]. Later on, Amenova et al. adopted convolutional neural network (CNN) and long short term memory (LSTM) to detect malicious Apps [8]. CNN is used to analyse the API to extract feature, while LSTM is applied to enhance the accuracy via correlation. Nevertheless, the prediction accuracy remains a crucial challenge in the category of static analysis.

As to the category of dynamic analysis, it retrieves the behaviour signature of a running malware. Due to the executing characteristics, behaviours of the obfuscated and packed malicious process can still be recorded to conclude illegal features. The obtained signature, however, relies on the length of analytic period, which is an extremely time-consuming task and becomes the main constraint on malware detection in this category. In 2020, Mahdavifar et al. have applied semi-supervised deep learning technique to develop a dynamic malware analysis model [9]. Here, fake levels and true labels are input into the training model to enhance the prediction accuracy. After that, Luo et al. further designed a DCGAN_1D-CNN (Deep convolution generative adversarial networks one-dimensional convolutional neural networks) model to distinguish encrypted malicious streams from normal ones [10], which also belongs to the dynamic type. To eliminate the problem of imbalance category distribution, authors have used 1D-CNN to increase the number of samples for those with a small number. By this way, the determination accuracy for new malware and those with a small number could be effectively enhanced. Whether the static or dynamic analysis is used, it is extremely inefficient to manually

analyse each malware. That is the reason why the technique of machine learning has been widely introduced into the malware detection.

Actually, the determination result of machine learning is often affected by the malware characteristics. In particular, lots of features may lead to the overfitting phenomenon; thus, increasing the computation overhead and lowering down the accuracy rate. To address these two issues, eliminating parts of unnecessary features while figuring out significant characteristics are very critical in the training model. In [11], it has demonstrated that removing seldom performing benign process is helpful to accuracy and efficiency enhancement. Regarding [12], it has tuned the machine model to dig out malware after being shuffled and noised. As to [13], all the programs are categorized according to the corresponding actions; thus, a pre-processing framework for machine learning is imperative.

So far, there exist lots of challenges in the detection of malware. Among them, how to increase the detection accuracy is the most crucial and emergent one. No doubt that the malware is constantly evolving thanks to the rapid development of Internet technology. Specifically, it is hard to dig them out as the stealth has become more serious. Therefore, we aim to design a static detection framework based on the API and program access privilege provided by Android. One of the main contribution of the new framework is that it is able to analyse the APK to filter out the necessary features for training model even though many meaningless APIs and access rights have been appended to the target program. Taking the trade-off between analysing time overhead and detection accuracy into consideration, the main target of this work is to fully dig out the malware as it has caused serious damage for general systems. According to the simulation results, the new method outperforms related works in terms of efficiency and accuracy based on the concept of shallow features and access right analysis.

The rest of this article is listed as follows. Related works and preliminary are described in section 3, followed by the detail of new method in section. Discussions on the simulation outcomes and comparisons are explained in section 4, while conclusions are made in section 5.

2. Related Works

Technology of machine learning has been well-developed, and sufficient models have been produced according to individual needs, which have been applied in various fields with satisfactory results. Actually, it also plays an important role in the detection of malware. Traditional expert analysis cannot approach the explosive development of malware anymore. More precisely, rule-based detection has come out to replace conventional methods to fulfil the real demand. Undoubtedly, the incredible increasing of malware variety and number have become an unbearable burden on traditional detection methods. That is the reason why the solution of machine learning model is urgent to be imported to this field. To achieve a higher detection accuracy, researchers have applied machine learning models to classify malware based on specific features. As to the new method, multi-layer perception (MLP) and random forest are applied to figure out the connection between distinct features, which is able to ensure the malware classification correctness and practicability.

2.1 Multilayer Perceptron (MLP)

In MLP, neurons of each layer are fully connected and transferred to the next layer. Multi-layer adoption has been used to expand or compress features, and neurons of each layer obtain specific feature through individual neuron weight. During the training process, the backpropagation algorithm has been applied to gradually tune a better weight to increase the prediction accuracy; thus, mitigating the difference between the real value and the predicted one. The structure of multi-layer perceptron is depicted in Figure 3.

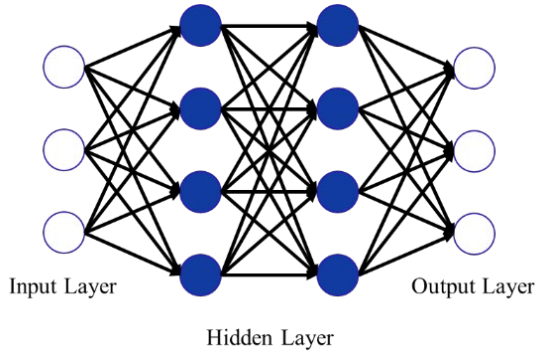

Figure 3: The model of multi-layer perceptron

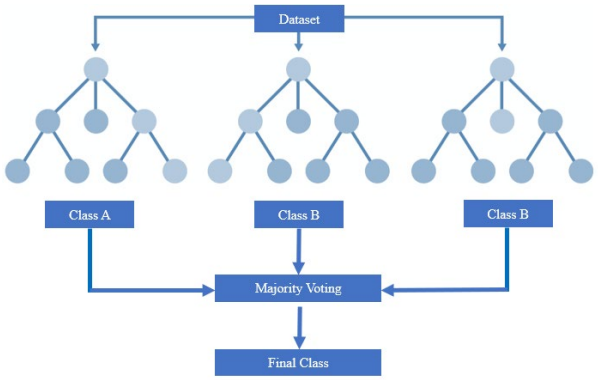

Figure 4: The model of random forest

Generally, the numbers of hidden layer and neuron could be adjusted to extend or compress the features according to the user demand. The weight of each neuron is applied to determine the significance of target feature. No matter in the hidden or output layer, the neuron value is derived from the previous layer neurons based on Eq. (1), where y_i represents the output of the *i*-th neuron, *p* is the number of previous layer neuron, w_{ji} denotes the weight between the *j*-th neuron of previous layer and the *i*-th neuron of the current layer, and x_i is the value of previous layer neuron.

$$y_i = \sum_{j=1}^{p} w_{ji} x_j \tag{1}$$

2.2 Random Forest (RF)

The model of RF consists of one or multiple decision trees, and each decision tree is constructed through the random feature selection of data cluster. After completing the training process of decision trees, the majority rule has been employed to determine the prediction type in RF. As the feature selection is random, the results predicted by each decision tree are different. Thus, the ensemble learning mechanism can increase the prediction stability and accuracy, while the structure of RF model is displayed in Figure 4.

Without loss of generality, RF can help lay out a satisfactory various prediction due to the adoption of random sampling along with the majority rule of decision trees. Meanwhile, the over-fitting problem could be mitigated effectively through the strategy. The eventual prediction outcome is determined according to the majority vote of decision trees, as shown in Eq. (2),

$$p = arg \max_{n} \sum_{i=1}^{n} w_i I(f_i(x) = n) \tag{2}$$

where *p* denotes the final outcome, *n* means the number of decision trees, w_i displays the weight of the *i*-th tree, *I* is the indicator function, and $f_i(x)$ depicts the prediction on the input *x* from the *i*-th tree.

3. Proposed Scheme (AMD)

Here AMD imports two significant behaviours, shallow feature and permission correlation, to determine the legality of a suspicious App. Androguard [14] is used to analyze all possible and hidden privilege threats, while Axplorer [15, 16] is applied to dig out the permission correlation. The flowchart is illustrated in Figure 5. The feature analysis of APK is described in Subsection 3.1, the permission estimation of APK is explained in Subsection 3.2, the feature correlation is examined in Subsection 3.3, while how to extract the shallow feature is introduced in Subsection 3.4.

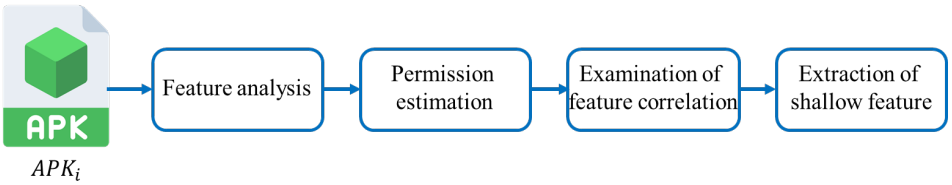

Figure 5: The flowchart of AMD

3.1 The Feature Analysis of APK

Step 1. Aggregate all the dataset applications into $APKs$, where

$APKs = \{APK_1, APK_2, \dots, APK_n\}$, n is the total number of applications.

Step 2. Apply Androguard [14] to analyze each APK_i, where $APIs^{APK_i}$ and $Permissions^{APK_i}$ denote the application interface and permission, respectively, $i \in \{1, 2, \dots, n\}$.

Step 3. Repeat Steps1 to 2 until all application analyses are completed.

3.2 The Permission Estimation of APK

Actually, parts of APIs can occupy a higher permission to complete a specific operation without advance authority, and such feature cannot be dug out through previous analysis. Here AMD concentrates on figuring out these additional information, called Extra_Permissions, using the tool Axplorer [16].

Step 1. Apply the Axplorer [16] to extract the hidden permission of ${API_j}^{APK_i}$,$Extra\_Permissions^{API_j^{APK_i}}$, where $i \in \{1, 2, \dots, n\}$, $j \in \{1, 2, \dots, m\}$.

Step 2. Concatenate $Extra\_Permissions^{API_j^{APK_i}}$ with $Permissions^{APK_i}$ to form $ALL\_Permissions^{APK_i}$, and integrate the result with $APIs^{APK_i}$ to obtain the total feature of APK_i.

Step 3. Repeat Steps 1 to 2 until all additional information are retrieved to layout a feature set $Features$ containing all APKs and APIs permissions.

3.3 The Examination of Feature Correlation

Here is to reserve the uncommon features of BAPKs and those used by malware.

Step 1. Refine $APIs$ by removing API_j which does exist in API_{ALLOW}, such that $APIs \in API_{ALLOW}$.

Step 2. Calculate the frequency of $Feature_j$ occurring to $BAPKs$ based on Eq. (3).

$$Feature\_rate_j = \frac{1}{n}\sum_{i=1}^{n} {Feature_j}^{BAPK_i}, j \in \{1, 2, \dots, m-1, m\} \quad \textbf{(1)}$$

Step 3. Select $Feature_j$ according to TH=0.8 and Eq.(4).

$$\begin{cases} \text{Remove } Feature_j, TH \le Feature\_rate_j \\ \text{Keep } Feature_j, otherwise \end{cases} \quad \textbf{(2)}$$

3.4 The Extraction of Shallow Feature

After the correlation analysis, AMD further extracts the shallow feature through APIs with the same classification.

Step 1. Generate $Package_k^{APK_i}$ by extracting the same $APIs^{APK_i}$ of $Features^{APK_i}$, $k \in \{1, 2, \dots, l-1, l\}$.

Step 2. Repeat above procedure until all extraction have been done and concatenate the results to form $Packages^{APKs}$.

Step 3. Train AMD model with $Packages^{APKs}$ and $ALL\_Permissions^{APKs}$.

4. Experimental Result and Analysis

Here we used the Python language to conduct the simulation and ensure that the detection system can support the latest Android version 14. Furthermore, we applied Scikit-learn [19] to perform the classification with the kits of MLP and RF. The simulation environment is equipped with Windows 10 64bit, Intel i7-7700HQ 2.8GHZ CPU, 8GM RAM, and NVIDIA GTX 1050 GPU. The testing dataset is discussed in Subsection 4.1, the detection outcomes under different THs are shown in Subsection 4.2, while the analyses and comparisons are explained in Subsection 4.3.

4.1 Dataset Selection

The paper uses two Android malware datasets, including CICAndMal2017 [18] and CICMalDroid2020 [9]. The CICAndMal2017 dataset consists of 2125 APK files, containing 426 malicious APKs and 1699 benign ones, where malicious APKs comprise advertising, ransomware, intimidation, and SMS malware. As to CICMalDroid2020, it covers more APK files and types of malicious APKs, including 9803 malicious APKs and 1795 benign ones, whereas malicious APKs include advertising, banking, SMS, and risky malicious APKs.

We analysed and preprocessed the two aforementioned datasets, resulting in 218 features. Among these, 183 features were Packages integrated by the API used, and the remaining 35 features were Permissions used in the dataset. To evaluate the detection effectiveness of AMD, we divided the datasets based on the Pareto principle, using 20% as the test set and the remaining 80% as the training set. The distribution is shown in Table 1.

The valuation indicators in this research are composed of True positive (TP), False positive (FP), True negative (TN), and False negative (FN).

- TP: The actual number of samples that are malicious APK and are determined to be malicious APK.
- FP: The actual number of samples that are benign APK but are determined to be malicious APK.
- TN: The actual number of samples that are benign APK and are determined to be benign APK.
- FN: The actual number of samples that are malicious APK but are determined to be malicious APK.

Using the four evaluation metrics above, we calculate *Accuracy*, *Precision*, *Recall*, and F_β according to Eq. (5) to (8). Here, β is set to 1, representing the weight of *Precision*, indicating that *Precision* and $Recall$ have the same importance in the classification of benign and malicious APK.

$$Accuracy = \frac{(tp+tn)}{(tp+fp+fn+tn)} \quad (3)$$

$$Precision = \frac{tp}{(tp+fp)} \quad (4)$$

$$Recall = \frac{tp}{(tp+fn)} \quad (5)$$

$$F_\beta = (1+\beta^2) * \frac{Precision*Recall}{(\beta^2*Precision)+Recall} \quad (6)$$

Table 1: Dataset distribution

Name of Dataset / APK Class	CICAndMal2017 [18]		CICMalDroid2020 [9]	
	Training Set	Test Set	Training Set	Test Set
Benign	1359	340	1436	359
Adware	83	21	1002	251
Ransomware	81	20	-	-
Scareware	90	22	-	-
SMS Malware	87	22	3123	781
Banking	-	-	1680	420
Riskware	-	-	2037	509
Total	1700	425	9278	2320

4.2 The Detection Outcomes Under Different *TH*s

The AMD employs machine learning to identify benign or malicious APKs, achieving high accuracy and stability the heavily relies on the quality of the data used for training the model. If too many features are provided, it can lead to overfitting, increasing the time consumption and reducing the classification effectiveness. Conversely, too few features are provided can lead to underfitting, rendering the model ineffective in classifying malicious and benign APKs. This study proposes a method based on feature correlation analysis to reduce the features required for detecting malicious APK under different thresholds (*TH*). This approach reduces the training time for machine learning and improves *Precision* and *Recall*. The experimental process places greater emphasis on *Precision* and *Recall*. *Precision* represents the proportion of benign APKs

misclassified as malicious among those identified as malicious APKs. Conversely, *Recall* represents the proportion of actual malicious APK among those identified as malicious. Hence, a higher *Recall* indicates a stronger ability to identify malicious APK.

Table 2: Result of CICAndMal2017 based on different *TH*

Metrics	*Accuracy*		*Precision*		*Recall*		$F1-score$	
Model / *TH*	MLP	RF	MLP	RF	MLP	RF	MLP	RF
0.5	87.32	86.15	87.43	87.07	98.24	97.06	92.52	91.79
0.6	90.14	90.14	90.93	90.93	97.35	97.35	94.03	94.03
0.7	91.78	91.31	92.24	92.20	97.94	97.35	95.01	94.71
0.8	92.72	92.49	93.28	93.02	97.94	97.94	95.55	95.42
0.9	91.08	91.31	92.42	91.97	96.76	97.65	94.54	94.72
1.0	91.08	89.20	91.71	90.38	97.65	96.76	94.59	93.47

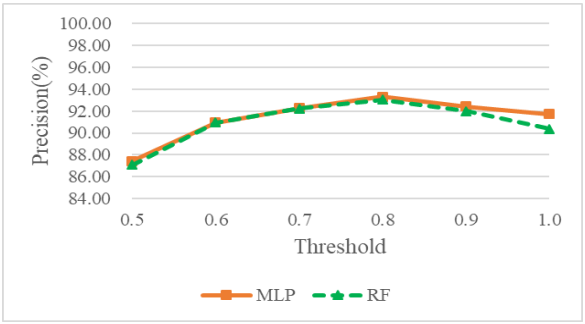

Figure 6: Comparison of $Precision$ at different TH in the CICAndMal2017

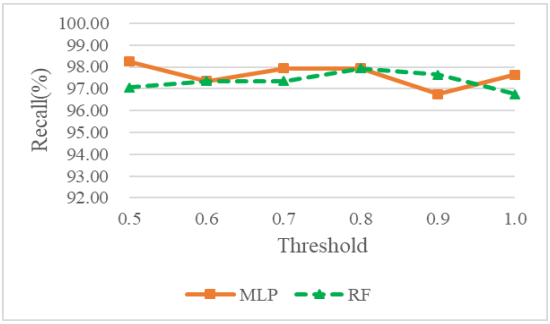

Figure 7: Comparison of $Recall$ at different TH in the CICAndMal2017

Using APIs in benign applications as features for detecting malicious APKs is insufficient for effective classification. Therefore, the number of features discarded through feature correlation analysis depends on changes in the *TH* value. In the CICAndMal2017 dataset, there are 1699 benign and 426 malicious samples. Reducing unnecessary features can improve machine learning *Precision* and *Recall*, as shown in **Table 2**. The analysis of the CICAndMal2017 dataset, shows that when the *TH* value is 1, features present in all benign APKs need to be removed. However, these features are rare since there are slight differences among different APKs. This results in only a slight improvement in detection after removing these features. When the *TH* value is continuously lowered to 0.8, both *Precision* and *Recall* achieve results, as the removal of some meaningless features effectively improves detection efficiency and accuracy. As shown in **Figures 6** and **7**, *Recall* decreases when the *TH* value is below 0.8. The lower the *TH* value, the more features are removed. Some of these removed features include important or distinctive features, affecting the accuracy of classifying malicious APKs. Therefore, AMD can avoid the overfitting problem encountered in machine learning for APK detection. It also adjusts the *TH* value according to the dataset to achieve the best classification results.

Table 3: Result of CICAndMal2020 based on different *TH*

Metrics	*Accuracy*		*Precision*		*Recall*		*F1-score*	
Model / *TH*	MLP	RF	MLP	RF	MLP	RF	MLP	RF
0.5	95.32	95.38	96.21	96.39	97.68	97.56	96.94	96.97
0.6	96.63	96.78	97.35	97.62	98.23	98.15	97.79	97.88
0.7	96.30	96.51	97.23	97.57	97.92	97.84	97.57	97.70
0.8	96.06	96.33	96.53	97.12	98.35	98.07	97.43	97.60

Metrics	*Accuracy*		*Precision*		*Recall*		*F1-score*	
Model / *TH*	MLP	RF	MLP	RF	MLP	RF	MLP	RF
0.9	96.12	96.39	96.78	97.34	98.15	97.92	97.46	97.63
1.0	96.63	96.99	97.50	97.85	98.07	98.19	97.79	98.02

As shown in **Table 3**, compared to CICAndMal2017, the CICMalDroid2020 dataset, which has more samples, demonstrates significant improvements in various metrics based on the feature correlation analysis. With the increase in sample size, *Precision* and *Recall* also achieve favourable results at different *TH* values, as illustrated in **Figures 8** and **9**. In AMD, features with high usage rates in benign APKs are removed to reduce the chance of the detector being confused. Consequently, the CICMalDroid2020 dataset, which contains more malicious APK samples, is less likely to remove important features, effectively classifying malicious APKs. Thus, *Precision* and *Recall* have outstanding performance at different *TH* values. AMD enhances classification efficiency and stability by removing unnecessary features through feature correlation analysis in datasets with more samples and a greater proportion of malicious APKs.

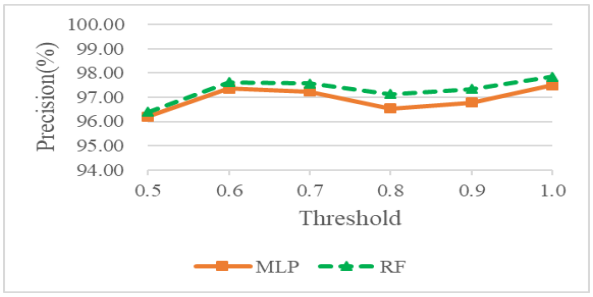

Figure 8: Comparison of $Precision$ at different TH in the CICAndMal2020

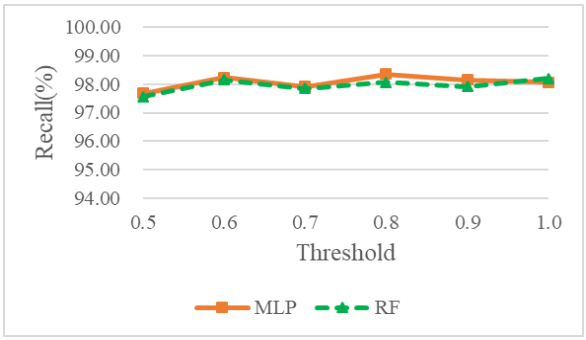

Figure 9: Comparison of $Recall$ at different TH in the CICAndMal2020

As shown in **Figure 10**, whether in the CICAndMal2017 dataset or the CICMalDroid2020 dataset with a larger sample, AMD avoids poor classification due to model overfitting by setting the *TH* value to 0.8, thereby effectively improving the accuracy of classifying malicious APKs. This indicates that adjusting the *TH* value according to different datasets and sample sizes can further enhance the detection rate of malicious APKs. Regarding the choice of model, although the *Precision* of RF in CICMalDroid2020 improved by 0.59 compared to MLP, the *Recall* decreased by 0.28. This suggests that RF might misclassify some benign applications in the CICMalDroid2020 dataset, which has many malware samples but is more accurate in identifying malicious APKs. Conversely, in the CICAndMal2017 dataset, which has fewer malware samples, both MLP and RF maintain better classification results. Therefore, we adopt MLP as the classification model.

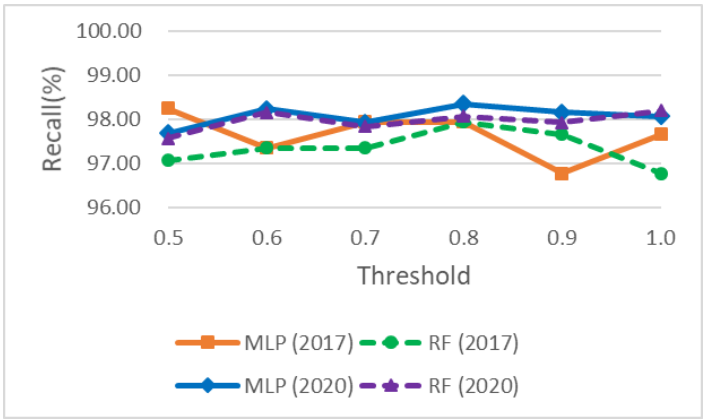

Figure 10: Comparison of $Recall$ between CICAndMal2017 and CICAndMal2020

4.3 The Analyses and Comparisons

The AMD classifies API calls and APK permissions through static analysis. To evaluate classification performance on different datasets, we use the CICAndMal2017 dataset proposed by Lashkari and Rahali et al. [18] and the CICMalDroid2020 dataset [9]. The proposed AMD method is compared with the static analysis method [8] and dynamic analysis methods [9, 10], focusing on the detection of malicious APKs. Misclassification of benign softwares has relatively minor impacts, whereas misclassifying malwares leads to issues like data leakage. Therefore, this paper aims to achieve higher *Recall*, improving the detection accuracy of malicious APKs. As shown in **Table 4**, *Recall* indicates the proportion of malicious that are accurately detected.

Table 4: Comparison between related works

Dataset	CICAndMal2017 [18]			CICMalDroid2020 [9]		
Metrics	*Precision*	*Recall*	*F1-score*	*Precision*	*Recall*	*F1-score*
[8]	-	-	-	93.30	94.45	95.78
[9]	-	-	-	99.16	96.54	97.84
[10]	95.87	97.23	96.55	-	-	-
AMD (*TH* = 0.8)	93.28	97.94	95.55	96.53	98.35	97.43

AMD uses static analysis, which takes less time than dynamic analysis and avoids wasting time due to malicious APKs deliberately hiding their overprivileged behaviours. For example, malicious APKs intentionally hide their malicious behaviours until certain conditions are met to evade detection [20]. In the CICAndMal2017 dataset, although this method's *Precision* is slightly lower than the method in DCGAN_1D-CNN [10], its Recall is 0.71 higher, and static analysis is more efficient than dynamic analysis. In the CICMalDroid2020 dataset, which has many malicious APK samples, AMD shows significant improvements in various metrics compared to the static analysis method [8]. Malicious APKs have minor correlations but continuously evolve to evade detectors, making them difficult to detect. Compared to the dynamic analysis method [9] and the CICMalDroid2020 dataset they proposed, although AMD has lower *Precision*, it improves *Recall* by 1.81. While our method may misclassify benign softwares, it is more effective in detecting softwares in a dataset with more malicious samples. These results demonstrate that AMD is better at identifying malicious softwares and effectively blocking malicious behaviours. Additionally, AMD surpasses dynamic analysis time, making it superior in time efficiency and detection effectiveness. Since our primary concern is detecting malicious APKs, we should focus on detecting malicious programs. Misclassifying a malicious APK as benign can lead to users downloading malicious APK, resulting in potential asset loss.

5. Conclusions

This paper proposes a machine learning-based static malware detection method, which primarily relies on the correlation between shallow features and APK permissions. It conducts an in-depth analysis of the relationships between applications and their features. Experimental results show that AMD detects malicious APKs accurately. Even when malicious softwares hide their behaviours within various APIs, our method effectively analyses and detects them. Moreover, in feature correlation analysis, our method can effectively prevent hackers from interfering with security analysts' detection processes. Extracting shallow features allows our method to focus on identifying representative behavioural features without being affected by commonly used but non-informative data. Our research results indicate that, compared to related works, our method detects malicious APKs more accurately, demonstrating its feasibility in practical applications. Users can enjoy the convenience of mobile devices while ensuring a secure environment with AMD.

Ethics Declaration: This research involves no human subjects and uses only publicly available malware datasets. All experiments were conducted in isolated environments following ethical cybersecurity research standards.

AI Declaration: AI tools were used solely for grammar checking and language polishing. All research methodology, analysis, and findings are entirely human-generated.

References

(2022) “Welcome to Android Developers”, [Online], Google, Android Studio, December, https://developer.android.com/reference/packages.

(2024) “Mobile Operating System Market Share Worldwide”, [Online], Statcounter, https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201701-202406.

S. Amenova, C. Turan, and D. Zharkynbek (2022) “Android Malware Classification by CNN-LSTM,” *2022 International Conference on Smart Information Systems and Technologies (SIST)*, November, pp. 1-4.

J. H. Ateeq and M. Moreb (2021) “Detecting Malicious URL using Neural Network,” *2021 International Congress of Advanced Technology and Engineering (ICOTEN)*, July, pp. 1-8.

M. Backes et al (2016) “On Demystifying the Android Application Framework: Re-visiting Android Permission Specification Analysis”, *The 25th USENIX Conference on Security Symposium (SEC'16)*, August, pp. 1101-1118.

A. Demontis et al (2019) “Yes, Machine Learning can be more Secure! A Case Study on Android Malware Detection”, *IEEE Transactions on Dependable and Secure Computing*, Vol. 16, No. 4, July, pp. 711-724.

A. Desnos, G. Gueguen, and S. Bachmann Revision (2022) “Welcome to Androguard’s Documentation!”, [Online], Androguard, November, https://androguard.readthedocs.io/en/latest/.

P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal (2015) “AndroSimilar: Robust Signature for Detecting Variants of Android Malware”, *Proceedings of the 6th International Conference on Security of Information and Networks, Journal of Information Security and Applications*, Vol. 22, June, pp. 66-80.

H. Gonzalez, N. Stakhanova, and A. A. Ghorbani (2014) “Droidkin: Lightweight Detection of Android Apps Similarity”, International Conference on Security and Privacy in Communication Systems, pp. 436-453.

M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang (2012) “RiskRanker: Scalable and Accurate Zero-day Android Malware Detection”, *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, June, pp. 281-294.

A. H. Lashkari et al (2018) “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification”, *2018 International Carnahan Conference on Security Technology (ICCST)*, December, pp. 1-7.

M. Leslous et al (2017) “GPFinder: Tracking the Invisible in Android Malware”, *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, October, pp. 39-46.

J. Li et al (2018) “Significant Permission Identification for Machine-learning-based Android Malware Detection”, *IEEE Transactions on Industrial Informatics*, Vol. 14, July, pp. 3216-3225.

X. Liang et al (2017) “Facial Feature Extraction Method based on Shallow and Deep Fusion CNN”, *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, July, pp. 50-53.

W. Luo et al (2022) “Malicious HTTPS Traffic Classification Algorithm Based on DCGAN_1D-CNN”, *2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, January, pp. 20-25.

S. Mahdavifar et al (2020) “Dynamic Android Malware Category Classification using Semi-supervised Deep Learning”, *2020 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, November, pp. 515-522.

Y. Pan, X. Ge, C. Fang, and Y. Fan (2020) “A Systematic Literature Review of Android Malware Detection using Static Analysis,” *IEEE Access*, Vol 8, June, pp. 116363-116379.

F. Pedregosa et al. (2022) “Machine Learning in Python”, [Online], Scikit-learn, December, https://scikit-learn.org/stable/.

T. Shishkova and A. Kivva (2022) “IT Threat Evolution in Q3 2022. Mobile Statistics”, [Online], Securelist, https://securelist.com/it-threat-evolution-in-q3-2022-mobile-statistics/107978/.

R. Surendran, T. Thomas, and S. Emmanuel (2021) “On Existence of Common Malicious System Call Codes in Android Malware Families”, *IEEE Transactions on Reliability*, Vol. 70, No. 1, March, pp. 248-260.