Hindawi
IET Information Security
Volume 2024, Article ID 2850804, 12 pages
https://doi.org/10.1049/2024/2850804

Research Article

MFEMDroid: A Novel Malware Detection Framework Using Combined Multitype Features and Ensemble Modeling

Wei Gu, Hongyan Xing, and Tianhao Hou

School of Electronics and Information Engineering, Nanjing University of Information Science and Technology, Nanjing 210044, China

Correspondence should be addressed to Hongyan Xing; xinghy@nuist.edu.cn

Received 21 July 2023; Revised 29 January 2024; Accepted 3 February 2024; Published 17 February 2024

Academic Editor: Leandros Maglaras

The continuous malicious attacks on Internet of Things devices pose a potential threat to the economic and private information security of end-users, especially on the dominant Android devices. Combining static analysis methods with deep Learning is a promising approach to defend against that. This kind of method has two limitations: the first is that the current single-permission mechanism is not insufficient to regulate interapplication resource acquisition; another problem is that current work on feature learning is dedicated to modifying a single network structure, which may result in a suboptimal solution. In this study, to solve the abovementioned problems, we propose a novel malware detection framework MFEMDroid, which combines multitype features analysis and ensemble modeling. The Provider feature, facilitating information requests between applications (apps) and serving as an indispensable data storage method, plays a vital role in characterizing app behavior. Hence, we extract permissions and Provider features to comprehensively characterize app behavior and probe potentially dangerous combinations between or within these features. To address oversparse datasets and reduce feature learning overhead, we employ an auto-encoder for feature dimensionality reduction. Furthermore, we design an ensemble network based on SENet, ResNet, and the evolutionary convolutional neural network Squeeze Excitation Residual Network (SEResNet) to explore the hidden associations between different types of features from multiple perspectives. We performed extensive experiments to evaluate its method performance on real-world samples. The evaluation results demonstrate that the proposed framework can detect malware with an accuracy of 95.38%, which is much better than state-of-the-art solutions. These promising experimental results show that MFEMDroid is an effective approach to detect Android malware.

1. Introduction

The diversified functions and services of Internet of Things (IoT) devices have drastically altered our daily habits [1]. Internet of Everything has also raised IoT security issues, particularly in the mobile market, where attackers are increasingly drawn to the attraction of digital wallets and sensitive user data [2]. The Android operating system (OS) is rapidly gaining popularity in IoT devices, such as smartphones, smart homes, car navigation systems, and smart watches [3, 4]. Android OS has gained 84.1% support from mobile phone manufacturers and end-users since 2021 [3]. The open source and sizable market of Android also appeal to the development of Android malicious software. Android, as the main target of malicious developers, undertakes approximately 98% of attacks in the mobile market [5]. Malicious apps, especially on Android, are the main threat to end-user security [6]. According to statistics, nearly 12,000 new malware are detected every day [7]. Besides, the third-party app markets without official trust guarantees allow users to download Android apps, further complicating the scenario. Hence, the urgency and importance of Android malware detection is unprecedented.

Previous detection approaches can be divided into dynamic and static analysis [8]. Dynamic analysis records the interaction with the system and network traffic consumption during the app's execution. Due to their reliance on the execution environment, dynamic methods are time-consuming. With the current exponential growth in the number of apps, static analysis is more suitable due to its lower requirements on the execution environment. Considering our aim is to detect

malware rather than discover hidden taints, our proposed approach relies on static analysis.

The existing detection methods mainly focus on the static features extracted from the AndroidManifest.xml file, especially on permission features, to analyze the capabilities and behaviors of Android apps [9]. Ganesh et al. [10] studied permission mode and detected Android malware based on a convolutional neural network (CNN). The security of an Android device depends heavily on its permission mechanism. To access sensitive resources (e.g., get a photo album) or perform system operations (e.g., invoke Bluetooth), the application must obtain appropriate permissions from the user. This access mechanism explains why permission-based detection methods are so widespread. After an in-depth analysis of permission-based works, we find that existing efforts fail to manage access to cross-application resource scenarios. According to the permission mechanism, an App that requires access to the contact information must declare the following element: <uses-permission android: name = "android.permission. READ_CONTACTS"/>. However, unauthorized apps can access the data if the following conditions are met. One is that an undeclared app obtains the Provider's relevant uniform resource identifier (URI) information from an authorized app. The other is that the exported attribute of the corresponding Provider is "True." Successful unauthorized access undermines the permissions mechanism's effectiveness.

In our study, to strengthen the regulation of interapplication resource access scenarios, we specifically emphasize Provider features rather than simply analyzing permissions. The Provider manages data sharing and access across apps, which is an indispensable data storage method. Further statistical analysis of our dataset shows that the permission named "android.permission.REQUEST_INSTALL_PACKAGES" and the Provider feature with the name "android.support.v4.content.FileProvider" often co-occur in the malicious sample set. This statistical data further confirms that it is worthwhile to consider the combined analysis of permission and Provider to reflect the sample characteristics.

It is vital to analyze the interdependences of the extracted features, including permission and Provider features, to optimize the classification performance. Feature learning [11] allows a machine to automatically explore and learn appropriate feature transformations from raw data. In recent years, deep Learning (DL) techniques for feature learning, such as CNN, have achieved considerable success in object detection [12, 13] and anomaly detection [14, 15]. Combining CNN with static analysis has made excellent achievements in Android malware detection tasks [16, 17]. Most earlier research focused on modifying individual network architecture, such as skip connection [18] and attention mechanism [19], to enhance the model's fitting learning capability. However, a single deep neural network converges to a local minimum, potentially yielding a suboptimal solution [20]. The ensemble methods provide a new perspective to address the challenge. The concept behind ensemble is to train multiple networks, which is less explored in malware detection. Some works have demonstrated that using an ensemble of models provides superior performance compared to a single model [21].

Based on the above observations, we propose a novel network architecture that utilizes an ensemble of deep CNNs to adaptively explore the intrinsic connection between the multiple features from various perspectives. The base learners in our ensemble network comprise ResNet, SENet, and the evolutionary CNN algorithm Squeeze Excitation Residual network (SEResNet). Each base learner has unique capabilities to mine the intricate connections within the data. Our proposed network combines SENet, ResNet, and SEResNet algorithms into a fully connected neuron layer for the final classification of the input data.

Based on the above works, we propose a novel malware detection framework MFEMDroid. In the raw feature extraction stage, we first extract two types of features with natural links (permission and Provider features). The auto-encoder (AE) is then applied to reduce feature dimensionality, with the purpose of reducing the overhead of feature learning and avoiding over-sparse datasets. Finally, the proposed ensemble network is applied to adaptively learn the proposed multitype feature combination (permission and Provider) from multiple perspectives to improve the malware detection performance. To our knowledge, the existing malware detection approaches pay less attention to the combination or internal relationship between the permission and Provider features. We have conducted extensive experiments to evaluate the effectiveness of the proposed framework based on the collected real-sample datasets and compared them with similar frontier work. The promising experimental results verify that our proposed framework is an effective solution to protect end-users away from malware attacks. In brief, our paper makes the following contributions:

(1) We propose a novel malware detection framework, MFEMDroid. The framework leverages permission and Provider features with rich semantics and a customized ensemble network to enhance the effectiveness of malware detection.
(2) We propose an ensemble network architecture that uses multiple base models such as SENet, ResNet, and evolutionary CNN SEResNet to adaptively explore the intrinsic connections between multiple features (permissions and Providers) from various perspectives.
(3) We collected benign and malicious apps from Google Play and Virus Share, respectively, to construct a time-sensitive dataset. We conducted extensive experiments to evaluate the performance of our proposed model. The experimental results show that our proposed model is superior to the state-of-art research.

The remainder of this paper is structured in the following way: Section 2 introduces the related work in the field of Android malware detection; Section 3 explains the overall architecture of the proposed model; Section 4 describes the experiment's specific design, results, and evaluation in detail;

Table 1: Summary of proposed approaches with the relevant works.

Paper	Year	Input data			Classifier/algorithm			
		D	S-S	M-S	I-ML	I-DL	E-ML	E-DL
Kim et al. [22]	2018	√			√			
Bhat et al. [23]	2023	√					√	
Li et al. [24]	2022	√						√
Ganesh et al. [10]	2017		√			√		
Alslan et al. [25]	2019		√		√			
Alazab et al. [26]	2020			√	√			
Khariwal et al. [27]	2020			√	√			
Wang et al. [28]	2019			√		√		
Wu et al. [29]	2021			√		√		
Zhu et al. [30]	2023			√		√		
Bakhshinejad et al. [31]	2020		√					√
Ficco et al. [32]	2021	√						√
This paper				√				√

Note: D: dynamic features; S-S: single static feature; M-S: multitypes of static features; I-ML: individual machine learning classifier; I-DL: individual deep learning algorithm; E-ML: the ensemble of multiple machine learning classifiers; E-DL: the ensemble of multiple deep learning algorithms.

and finally, Section 5 summarizes the paper and proposes possible future studies.

2. Related Work

Researchers have proposed various static and dynamic tools to detect malware and designed many data-driven malware detection methods based on DL. Among those existing efforts, a few of them are reviewed below. The summary of our proposed models with the related works is shown in Table 1.

2.1. Dynamic and Static Analysis Methods. Dynamic analysis methods monitor and capture the runtime behavior of apps to identify malicious patterns. For instance, Kim [22] presented an ML-based model to analyze the native API system calls for malware detection. Bhat et al. [23] proposed a dynamic analysis approach using an ensemble of multi-ML approaches to identify malicious attacks. Li et al. [24] presented a novel dynamic malware detection mechanism that extracts API sequence intrinsic features as the input of CNN and Bi-LSTM. Due to the widespread use of malicious samples with obfuscation techniques, dynamic analysis methods require more expensive hardware resource matching. Besides, the number of apps in the market is growing exponentially, which further expands the requirements of dynamic detection approaches for the execution environment. We aim to identify malware accurately rather than tracking specific malicious behaviors. Thus, the dynamic analysis method does not meet our requirements of quickly identifying malware.

Unlike dynamic feature analysis, static analysis requires less execution environment. Mainstream static analysis work cannot be separated from permissions. For example, Ganesh et al. [10] were dedicated to studying permission patterns and utilized CNN to detect Android malware. Arslan et al. [25] designed a permission-based malware detection mechanism combining classical ML algorithms such as decision tree (DT), logistic regression, and random forest (RF). The existing static detection work is limited to a single feature (mainly permission), which poses the reflected one-sided characteristics of malicious samples and makes it hard to match the update iteration speed of malicious samples. Therefore, some scholars began to pay attention to the hidden patterns among multiple types of features and have achieved some achievements. For instance, Alazab et al. [26] designed a novel malware detection framework, which combined the characteristics of permission and API. The experiment results of the F1-score on 27,891 real apps reached 94.3%. Khariwal et al. [27] analyzed the combination of static features permission and intent based on classical ML methods such as SVM and RF.

As far as we know, it is still not found some scholars have combined or analyzed permission and Provider. Considering that both are related to controlling access to resources, we filled out the mixed features of permission and Provider for joint analysis. In addition, our proposed malware detection method has good scalability and can be used to mine more types of feature combinations in the future.

2.2. DL-Based Malware Detection. In recent years, DL technology combined with static or dynamic analysis has made excellent achievements in malware detection. For example, Ganesh et al. [10] proposed a malware detection model based on classical LeNet and evaluated the model performance on 2,500 collected real Android apps (including 2,000 malicious samples and 500 ones), with an accuracy of 93%. Wang et al. [28] proposed a new CNN-based malware detection framework. The static features are extracted and converted into a 2D matrix, which is then used as input for the CNN to detect malware. Wu et al. [29] proposed an interpretable malware detection method. The extracted permissions and API calls are fed into the attention layer and multilayer perceptron for malware detection. Zhu et al. [30] employed a customed CNN multihead squeeze-and-excitation residual network to learn the intrinsic correlation between multitype static features. These efforts utilizing an individual network have made significant strides in identifying and preventing

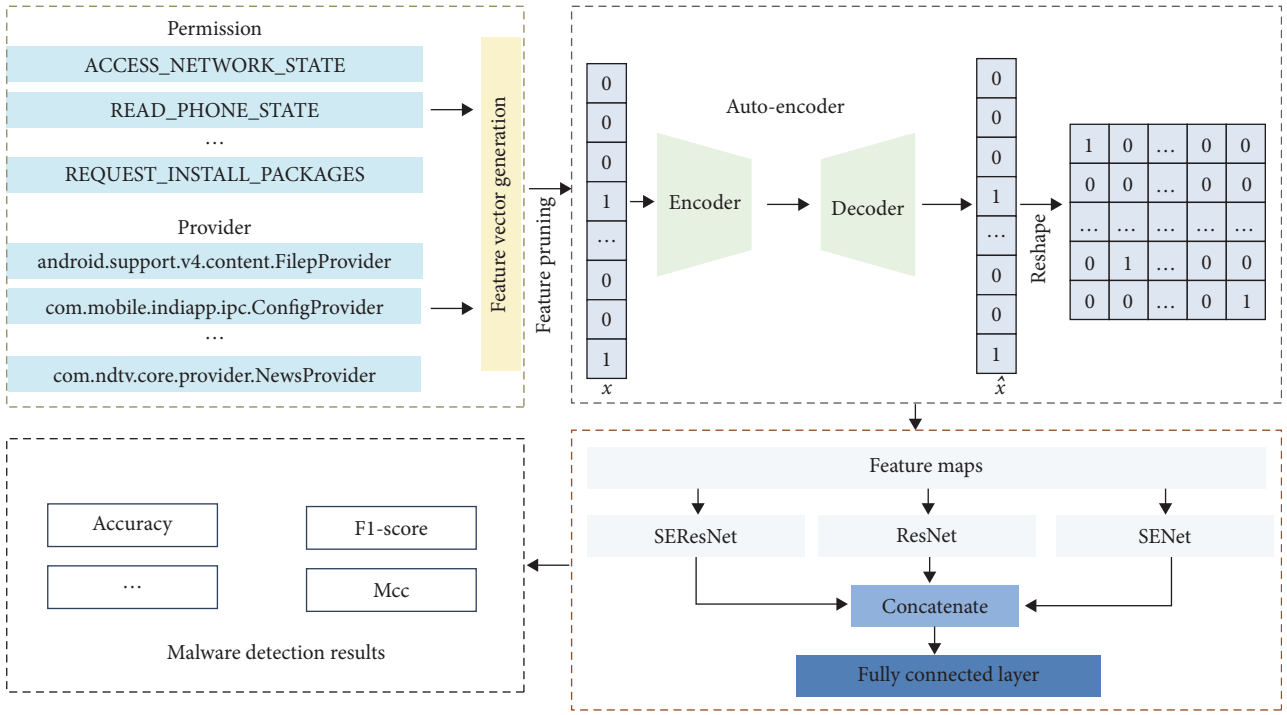

Figure 1: The overall framework of MFEMDroid.

malware. However, it is important to note that a single deep neural network may converge to a local minimum, resulting in suboptimal results.

Ensemble modeling using multiparallel networks has proven to be an effective way to solve the above issues. Bakhshinejad and Hamzeh [31] employed a new parallel architecture of two CNNs to automatically extract and learn n-gram patterns. Ficco [32] proposed an ensemble detector to exploit multitype features from multiple dimensions. Naeem et al. [33] developed an ensemble of neural networks using stacked generalization to learn basic local and global image features for malware detection. Naeem et al. [34] employed a deep-stacked ensemble network to learn the fuzed handcrafted features for malware detection. Inspired by the above work, we propose an ensemble model using multiple parallel CNNs to adaptively mine the potential associations of multitype features. Multiple parallel CNNs involve an evolutionary CNN SEResNet and classical SENet and ResNet.

3. Proposed Model

Our proposed malware detection model mainly conducts three parts, as shown in Figure 1. We will describe each part in detail in the following sections.

3.1. Raw Feature Extraction. The quality of the extracted features directly affects the performance of the malware detection model. Therefore, in the feature extraction stage, we are committed to mining the static features with strong associations to provide high-quality input data for the training model. Static feature analysis work is inextricably linked to AndroidManifest.xml, which is an indispensable configuration file for each app, consisting of permissions, Providers, and other features. To obtain the required static features, such as permission and Provider, the open-source tool Androguard [35] is selected to decompile AndroidManifest.xml and extract in batches. The process of feature extraction and vector generation method is shown in Algorithm 1.

3.1.1. Permission Feature. There are countless outstanding works on permission [8, 16], which is due to the inseparable relationship between permission and Android security. The permission mechanism serves as the primary protective barrier for Android security, manages apps to access sensitive resources (such as SMS), and performs system-level operations (such as camera) [36]. After Android 6.0 [37], permission features can be subdivided into normal, dangerous, and signature levels based on protection level. The protection level of permission is normal by default. It is automatically authorized during app installation without any execution operation instructions from the user. Signature-level permission will be automatically granted to access the specified resource merely when it matches the app's signature that claims this permission. Permissions at the level of dangerous are closely related to the user's private information, which requires attracting sufficient attention from end-users. Dangerous-level permissions, such as READ-CONTACTS and READ-SMS, respectively, manage the operations of reading user contact and SMS messages, which involve personal privacy. During the development process, developers need to uniformly define the fields starting with <uses-permission> for all applied permissions in advance in the AndroidManifest.xml file. For the irreplaceable position of permission in Android security, we select the unique identifier tag "name" in <uses-permission> for analysis and

Input:
X: Both malware and benign samples
Output:
PE: Permission feature set
PR: Provider feature set
FV: Feature vector
Function:
1. For each apk file ∈ X do
2. Decompile to obtain the AndroidManifest file;
3. PR←collect all Providers from the tag <Provider> line by line;
4. PE←collect all permissions from the tag <uses-permission> line by line;
5. End for
6. FV←∅;
7. BF←PE∪PR;
8. For each apk ∈ X do
9. index ← 0, F_i← [0|0|…|0]
10. For $\forall F \in BF$ do
11. If $f \in$ apk do
12. F_i[index] ← 1;
13. index ← index + 1;
14. End for
15. FV← FV∪$F_{i;}$
16. End for
17. Return FV

Algorithm 1: Feature extraction and vector generation method.

generate a permission feature set based on Google's official documents for subsequent analysis.

3.1.2. Provider Feature. As one of the significant components [36] of Android apps, the Provider is an indispensable data storage method for sharing information between different apps. Provider features are often applied to manage information transmission between multi-apps, such as sharing contact between different apps. Specifically, the Content Provider serves a public URI as a unique mark to tag resources, and then other apps access or even delete information through the entrance. The default value of "android: exported" for Content Providers is set to "True" when the API Level is below 17. This default configuration inadvertently provides a hidden entry point for malicious attackers, putting end-users at risk of arbitrary data access, SQL injection, and directory traversal attacks. The Provider acts a fine complementary role in resource control, especially in cross-application resource access. Man-in-the-middle attack (MITM) refers to the interception or even malicious tampering of normal network communication data without the awareness of both parties. MITM is closely related to the Provider that manages the communication between apps. Therefore, we extract and analyze Provider features. We draw support from statistical methods to further estimate the rationality and effectiveness of extracting Provider. We counted the Provider with the top 10 occurrence frequencies on our all-malicious apps. The results show that half of the Providers were prefixed with "com.mobile.Indiapp." This prefix points to a resourceful third-party app store, named 9apps, which has an average daily circulation of more than 26 million. The 9apps is not a malicious site. However, the website VirusTotal [38] was applied to ergodic the domain of the store, and huge viruses were found in the domain. Besides, malicious developers [39] prefer to name components with similar confusing names to trick end-users into installing or avoiding detection. For instance, the high-frequency Provider named "com.qihoo.util.CommonProvider" in our dataset misleads end-users through the tag of "Qihoo" disguised as Qihoo 360 apps. This instance further indicates that it is beneficial to identify malware by analyzing Providers.

3.2. Feature Filtering and Selection. Given the huge number of features extracted and the sparse dataset, we perform a dimensionality reduction operation via AE to eliminate redundant features and reduce the load on subsequent feature learning. As shown in Figure 2, AE consists of an encoder and a decoder. The encoder is responsible for mapping the input data x into a hidden layer h. The decoder maps the representation of the latent space back to the space of the original data, reconstructing the input data as much as possible. For the input x, the encoding and decoding process is shown in Equations (1) and (2).

$$h(x) = f(Wx + b), \tag{1}$$

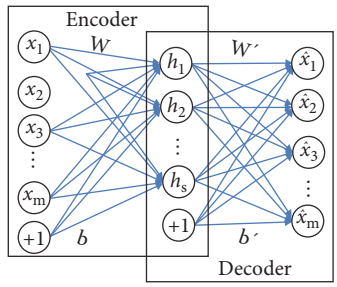

FIGURE 2: The architecture of an auto-encoder.

$$\widehat{x} = f'(W'h(x) + b'), \tag{2}$$

where f and f' are sigmoid function, W and W' denote the weight of the encoder and decoder, respectively, b and b' represent the bias of the encoder and decoder, respectively. The loss function of the AE is shown in Equation (3).

$$L = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}\left[x_{j,i}\log\widehat{x}_{j,i} + \left(1 - x_{j,i}\right)\log\left(1 - \widehat{x}_{j,i}\right)\right], \tag{3}$$

where M denotes the total number of samples input into the AE method, and N means the feature dimension of each sample. $x_{j,i}$ means the jth component of the ith observation.

After the AE method, we reserve 147 permissions and 253 Providers. There is a unified naming rule for permission features. Therefore, after the AE operation, we retained all 147 extracted permission features for further study. Due to the lack of official naming rules, the dimensionality of the Provider was reduced by more than 90%. Compared with the Provider features before filtering, the filtered features based on multiple evaluation indicators still have advantages.

To adapt the extracted features to the input of our ensemble network, an adaptive vector dimension transformation mechanism is designed. Specifically, the input dimension $(X, Y, 1)$ is converted into $(X, \sqrt{Y}, \sqrt{Y})$, where X represents the number of samples, and Y is the number of features corresponding to each sample. To ensure that $\sqrt{Y}$ is an integer, a common operation is to fill 0 columns at the end or reduce the number of columns. Reducing the number of columns will cause the loss and defect of the relationship of the original features. Therefore, we adapt the dimensions by adding 0 columns.

3.3. Ensemble Multi-CNN Modeling and Classification. As a feature extractor or classifier, CNN has made excellent achievements in object detection [12, 13] and anomaly detection [14, 15]. Many excellent CNNs have been introduced into malware detection models, including ResNet [40] and SENet [41]. ResNet tackles deteriorated learning performance by leveraging residual connectivity, enabling the network to learn identity mappings. SENet is a milestone in the field of CNN, which won the championship of the ILSVRC classification competition in 2017. The Squeeze and Extraction (SE) block in the SENet enables the network to have the capability to adaptively calibrate features, that is, selectively identify features with strong identifications and suppress features with weak discriminations. Recent CNN-based methods relied on improving the internal architecture of the network to efficiently detect malware. A single network poses the potential risk of achieving suboptimal results. Ensemble modeling, which uses multiple networks to learn the hidden patterns from the raw input features, provides a new solution to solve this issue. In this paper, inspired by the success of ensemble modeling and CNN, we propose an ensemble network utilizing multiple base models, including SENet, ResNet, and evolutionary CNN SEResNet, to enhance the overall accuracy rate. The structure of the ensemble network is shown in Figure 3.

ResNet, as part of our ensemble network, has a residual structure at its core. The residual structure ensures that the performance of the deep network is not inferior to a relatively shallow network, as shown in Equation (4).

$$H(X) = F(x, \{W_i\}) + x, \tag{4}$$

where $F(x, \{W_i\})$ is residual mapping, $F = W_2\sigma(W_1x)$ and σ is ReLU, while W_1 and W_2 represent two weight layers, respectively.

SENet is another important part of our ensemble network. The SE block is an important part of SENet and consists of two parts: Squeeze and Exception. As shown in Figure 3, the F_{tr} is responsible for transforming the input feature map X to a squeezed feature vector U_c following Equation (5). The Squeeze block, depicted in Figure 3, transforms the input feature using Convolution, Batch Normalization, and ReLU, as described in Equation (6). The Squeeze block mainly utilizes Global Average Pooling to generate channel-wise statistics, capturing important information from each channel. Another important component, the Excitation operation, denoted in Equation (7), aims to further capture channel correlation. By combining the Squeeze block and the Excitation operation, the SE block in SENet adaptively calibrates features and enhances discrimination. The final output result is shown in Equation (8).

$$Uc = V_C * X = \sum_{s=1}^{c'} V_C^S * X^s, \tag{5}$$

$$Zc = Fsq(uc) = \frac{1}{H}\sum_{i=1}^{H} uc(i), \tag{6}$$

$$S = Fex(z, W) = \sigma(g(z, W)) = \sigma(W2\delta(W1z)), \tag{7}$$

$$\widetilde{X}c = F_{\text{scale}}(uc, sc) = s_c u_c. \tag{8}$$

In Equation (5), "$*$" represents convolution, "V_C" denotes the parameters of the cth filter, and V_C^S is a 1D convolutional kernel. In Equation (7), δ is ReLU, W_1 and W_2 represent the weights of two fully connected layers. The F_{scale} in Equation (8) represents channel-wise multiplication.

By integrating the SE module and the residual connections, we design SEResNet, which aims to improve the detection of Android malware by effectively calibrating features

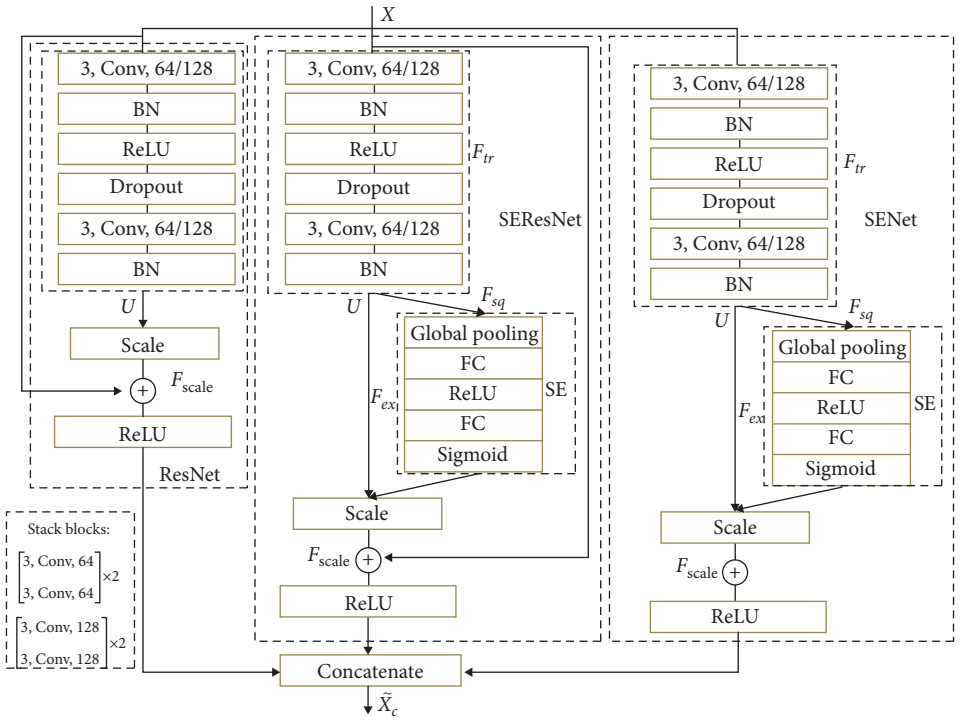

FIGURE 3: The structure of our ensemble network.

and leveraging the learning performance of ResNet. We learn permission and Provider features latent associations jointly by three parallel base models (SENet, ResNet, and SEResNet) and feed the learning results into a fully connected neuron layer for final classification.

4. Experiment Evaluation

To estimate the effectiveness of MFEMDroid, we conduct multigroup experiments on real-world samples. The effectiveness of the AE module is verified in the first group experiment using two types of features (namely, the raw Provider and the filtered Provider). The second group experiment evaluates the performance to characterize the malware with semantically rich multitype features using three categories of characteristics (i.e., Provider, permission, and combination features). The final set of experiments verifies the performance of the proposed network and compares it with the traditional single network, confirming that our ensemble network can fully exploit the potential associations among features. The dataset utilized in these experiments is described in Section 4.1.

4.1. Dataset Collection. To discover the hidden patterns of malware [42], we analyze the behavior of malicious and benign Android apps in the real world. Self-collection of samples rather than obtaining samples from current mainstream datasets (such as Drebin [43], DCL [44], etc.) mainly takes into account the following common issues: first, the raw sample is not published, which leads to the study scope focusing only on the existing published features, and mainstream dataset samples are generally outmoded. Therefore, we obtained real-world apps from the official market Google Play Store and the authoritative malicious sample-sharing website VirusShare to form a preliminary sample set. Compared with the third-party app market with uneven quality, Google Play Store provides strict security verification for all uploaded APKs [41]. Therefore, we merely consider samples from the official market Google Play Store. For the samples obtained from the Google Play Store, we selected VirusTool (a detection website integrating more than 70 antivirus scanners) [38] to traverse one by one, and finally, 8,981 samples passed the detection successfully. Owing to some apps downloaded from VirusShare failing to decompile, our dataset eventually consisted of 7,896 malicious samples and 8,981 benign ones. It is worth noting that the corresponding label of malicious apps is “1”, and the value of ”0” represents the benign samples.

4.2. Evaluation Criteria. In this section, we systematically introduce the evaluation criteria used in comprehensive experiments to evaluate the performance of the proposed Android malware detection model. The common evaluation criteria, including Accuracy, Precision, Recall, F1-score (F1), and Matthews correlation coefficient (Mcc), are shown as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}}, \tag{9}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{10}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{TN}}, \tag{11}$$

Table 2: Results based on ensemble network and different providers.

Feature	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Mcc (%)
Provider	75.41	67.21	94.53	78.56	55.82
Provider(AE)	**76.42**	**67.73**	**95.87**	**79.38**	**58.20**

Bold values signify the best results.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{12}$$

$$\text{Mcc} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TP} + \text{FP}) \times (\text{TN} + \text{FN})}}, \tag{13}$$

where TP is the number of malicious samples properly predicted, FP represents the number of benign apps incorrectly predicted, TN represents the number of benign apps correctly predicted, and FN is the number of malicious samples incorrectly predicted.

F1-score is determined by both Accuracy and Recall; hence, it can judge the performance of the model more scientifically. Mcc comprehensively considers TP, TN, FN, and FP, which are also relatively balanced indicators. To comprehensively evaluate the experimental results, the Auc is also introduced, which represents the area under the ROC curve.

4.3. Performance Evaluation. In this section, we conducted several groups of experiments under the same parameter settings to evaluate the effectiveness of the malware detection framework MFEMDroid. The relevant hyperparameters are set as follows: Adam Optimizer uses the default, the activation function is ReLU, batch_size is 200, and the number of iterations is 100. Our ensemble network uses Dropout to prevent over-fitting, setting a value of 0.1. We strictly follow the protocol of fivefold cross-validation in all experiments to obtain stable and convincing experimental results.

The aim of the first group of experiments is to evaluate the effectiveness of AE utilizing two different features (i.e., the raw Provider and the filtered Provider). Through the AE mechanism, 3,978 Providers are filtered into 253 features. We generate a feature set named Provider(AE) based on the reserved features. Our proposed network was selected to conduct feature learning on Provider and Provider(AE), respectively, and the relevant experimental results are presented in Table 2. We notice that compared with the raw Provider, the experiment based on the proposed Provider(AE) increases the Accuracy by 1.01%, Precision by 0.52%, Recall by 1.34%, F1-score by 0.82%, and Mcc by 2.38%.

The experimental results presented in Table 2 show that the Provider or Provider(AE)-based model performs mediocrely on most indicators. The Provider generally serves merely as an intermediary for the attack rather than being directly involved in the potentially malicious activities of the app. This is the main reason for the poor performance of models based on single-Provider features. Poor precision metric refers to the large number of benign apps that are incorrectly classified as malware. This can be attributed to the fact that some benign apps share similar Providers with

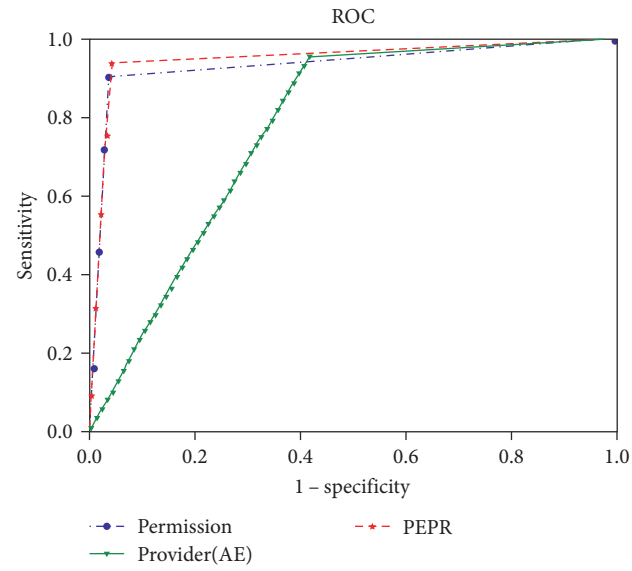

Figure 4: The ROC curves achieved by ensemble network on the Permission, Provider(AE), and PEPR.

malicious apps, leading to misclassification. Therefore, we consider the joint analysis of the Provider with the permission features that are directly involved in the interaction with the malicious behavior.

To verify the characterization ability of features extracted under the multitype feature idea, we compare each component by ablation study. We use our proposed ensemble network to learn the hybrid features PEPR (the combination of Permission or Provider(AE)) and its single component Permission or Provider(AE). To visually demonstrate the performance-mentioned models, the average ROC curves are shown in Figure 4. The area under the ROC curve is positively correlated with the model performance. Figure 4 preliminarily presents the effectiveness of PEPR features for improving malware detection performance. Meanwhile, we found that the detection performance of a model based on a single Provider was significantly weaker than that of a model based on permissions or a combination of both. This is because while both Providers and permissions are relevant to information security, permission-based features are often more informative and directly related to the potential risks and malicious activity an app may exhibit in a malware detection task. In Figure 4, the red curve representing the PEPR-based model has a markedly larger area under the curve than the blue curve referring to the Permission-based model. This confirms that Providers can be better combined with permissions for joint analysis.

Table 3: Results of a single feature or hybrid feature based on our ensemble network.

Feature selection	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Mcc (%)
Provider(AE)	76.42	67.73	**95.87**	79.38	58.20
Permission	93.66	95.15	90.87	92.97	87.28
PEPR	**95.38**	**95.78**	94.47	**95.12**	**90.74**

Bold values signify the best results.

Table 4: Results based on different networks or classifiers.

Network	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Mcc (%)	Training time (min)
KNN	92.31	**96.28**	86.96	91.37	84.84	—
DT	93.05	93.61	91.40	92.49	86.05	—
SVM	92.20	93.27	89.84	91.51	84.37	—
ResNet	93.99	94.74	92.28	93.50	87.93	0.46
SENet	94.57	95.02	93.28	94.14	89.09	**0.44**
SEResNet	95.20	96.65	93.16	94.87	90.42	0.45
MFEMDroid	**95.38**	95.78	**94.47**	**95.12**	**90.74**	3.79

Bold values signify the best results.

Extensive qualitative experimental results are shown in Table 3. From Table 3, we can notice that our hybrid feature, PEPR, is more effective than the single feature. Compared with the Provider(AE), our model MFEMDroid based on PEPR has significantly improved the performance on most evaluation indicators. In addition, compared with Permission, another momentous component of PEPR, our model MFEMDroid increased the Accuracy by 1.72%, Precision by 0.63%, Recall by 3.60%, F1-score by 2.15%, and Mcc by 3.46%. Therefore, these results show that MFEMDroid is effective in malware detection and works as a multiperspective representation of samples by hybrid features. Furthermore, Table 3 confirms the effectiveness of permissions in detecting malware.

To verify that the use of ensemble modeling is necessary for the detection performance improvement, we compare it with the classical machine learning method and individual DL algorithm. The proposed ensemble network's effectiveness is verified by feeding the same hybrid features, consisting of permissions and Providers, to different networks. The relevant parameters are set as follows: the value of k in the K-nearest neighbor (KNN) is 2, and the kernel in the support vector machine (SVM) is linear. The default values for the sklearn libraries are used for the rest of the parameters in KNN, DT, and SVM. Resnet, SENet, and SEResNet each consist of two convolutional blocks. The convolutional layers in block1 have 3 kernels of size 64 each, while block2 has 3 kernels of size 128 each. SENet and SEResNet both use a reduction parameter of 16.

The results are shown in Table 4. The second and third rows of Table 4 show the experimental results based on KNN, DT, SVM, ResNet, SENet, and SEResNet, respectively. The experimental results demonstrate that our ensemble network characterizes malicious samples more accurately than KNN, SVM, DT, ResNet, SENet, and SEResNet. The above experiment results verified the efficacy of the ensemble modeling in enhancing the overall performance of the model. Additionally, we analyzed the training time for different algorithms on our real-sample dataset. Table 4 illustrates that while our ensemble network incurs greater time overhead than a single model, it is still viable to sacrifice a certain amount of time overhead to enhance detection performance.

4.4. Comparison to State-of-the-Arts. To further evaluate the effectiveness and generality of the proposed model, we conducted three groups of comparative experiments to verify the detection capability of the proposed model. We compare with the frontier representative research work based on machine learning and the DL model, respectively. All comparative experiments are also conducted based on our collected dataset, and the relevant parameter configuration is set strictly according to the original author's paper. The fivefold cross-validation scheme is also applied to all experiments. Table 5 shows extensive experiment results of our model and a comparison with the state-of-the-art models.

Arslan et al. [25] proposed a malware detection model KNN-P based on static feature Permission, which makes use of the classical machine learning method KNN as a classifier and achieves optimal performance under the condition of $k = 2$. The remaining KNN parameters utilize the default values of the sklearn library. The experimental results are shown in the first row of Table 5. It is worth mentioning that the Accuracy, Recall, F1-score, and Mcc of MFEMDroid are 5.26%, 5.58%, 2.85%, and 11.65% higher, respectively, compared to the KNN-P model. It confirms that the proposed malware detection model MFEMDroid performs better than KNN-P, no matter both in terms of Accuracy, Recall, F1-score, and Mcc. While our model's precision metric falls slightly short of that of KNN-P, the comprehensive evaluation metrics of Mcc and F1-score indicate significant improvement.

Ganesh et al. [10] designed the model LeNet-P to research the permission features based on the LeNet. The parameter in LeNet is based on standard LeNet-5 architecture. Different from the 138 permission features mentioned in the original author's paper, 147 permissions were extracted from the

Table 5: Comparison with other detection models.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	Mcc (%)
KNN-P [25]	90.12	**95.95**	88.89	92.27	79.09
LeNet-P [10]	93.38	95.28	90.31	92.73	86.76
SigPID [45]	89.32	93.81	82.64	87.87	78.91
MFEMDroid	**95.38**	95.78	**94.47**	**95.12**	**90.74**

Bold values signify the best results.

reproduction experiment. The reason for choosing to extract more permissions is that our samples are more novel. To adapt the input dimension of LeNet, we fill in 0 columns and convert the dimension to 13×13. The experimental results are shown in Table 5. We can see that our model has achieved significant performance improvement in the mainstream evaluation indicators. It is worth mentioning that, especially in the comprehensive evaluation index Mcc, our improvement has reached 3.98%.

Li et al. [45] presented a malicious software detection system, SigPID, which calculates and filters important permission features through multi-level pruning. First, the permissions are filtered with permission ranking with a negative rate, and 78 permissions are retained. After the second support-based permission ranking operation, 36 permissions are reserved, and then the last association rule based on Apriori [46] is performed to eliminate redundant permissions. Finally, 35 permissions are retained and trained by SVM. The relevant parameters in SVM use the default values from the sklearn library. The relevant experimental results are shown in the third row of Table 5. Compared with SigPID, our proposed model MFEMDroid obtains superior performance on the basis of five popular evaluation indicators.

The comparison work presented in Table 5 (i.e., KNN-P, LeNet-P, and SigPID) is all related to permissions. The results of these comparative experiments demonstrate that permissions consistently play a crucial role in malware detection. Our proposed model achieves superior results by emphasizing Provider features in addition to permissions, which effectively compensates for the lack of supervision of permissions for inter-application resource access scenarios. Additionally, the improved performance of the proposed model MFEMDroid can be attributed to our customized network. Our ensemble network has the ability to explore more internal corrections among multitype features from multiple perspectives. Considering the above comparative experiments, the contribution of the proposed model MFEMDroid based on hybrid feature PEPR and ensemble modeling in malware detection is further verified.

5. Conclusion

In this paper, we proposed a novel malware detection framework MFEMDroid. The multitype feature extraction method is one of its essential elements. We introduced a novel malware detection method by combining the permissions and Providers with inherent relations to reflect the hidden patterns of malware. To reduce the overhead of feature learning, we employ an autoencoder for feature dimensionality reduction. The ensemble modeling is another essential component. We utilize an ensemble network based on SENet, ResNet, and SEResNet to explore more internal relationships between multitype features from different perspectives, thereby improving the model detection capability. In the experiment, the Accuracy, Precision, Recall, F1-score, and Mcc of the model's detection results are 95.38%, 95.78%, 94.47%, 95.12%, and 90.74%, respectively. The results of comprehensive experiments and analysis show that the MFEMDroid outperforms the single feature type-based and individual network-based detection models. Comparison with state-of-the-art malware detection research works further verifies the effectiveness of our framework in malware detection. In the future, we will concentrate on the combinations of more static features to obtain more efficient malware detection.

Data Availability

Data will be made available on request.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Authors' Contributions

Wei Gu has contributed to the conceptualization, methodology, investigation, software, and writing—original draft. Hongyan Xing has contributed to supervision, funding acquisition, and writing—review and editing. Tianhao Hou contributed to software and validation.

Acknowledgments

The authors would like to thank all anonymous reviewers for their insightful feedback. The authors would like to appreciate Nanjing University of Information Science and Technology for supporting this research work. This work is supported by the National Natural Science Foundation of China (62171228) and the National Key Research and Development Program of China (2021YFE0105500).

References

[1] A. Bacci, F. Martinelli, E. Medvet, and F. Mercaldo, "VizMal: a visualization tool for analyzing the behavior of Android malware," in *2nd International Workshop on FORmal Methods for Security Engineering*, 2018.

[2] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, 2019.

[3] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: event-aware Android malware detection against model degrading for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[4] F. Aloraini, A. Javed, O. Rana, and P. Burnap, "Adversarial machine learning in IoT from an insider point of view," *Journal of Information Security and Applications*, vol. 70, Article ID 103341, 2022.

[5] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: effective and efficient behavior-based Android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[6] İ. Atacak, K. Kazım, and A. D. İbrahim, "Android malware detection using hybrid ANFIS architecture with low computational cost convolutional layers," *PeerJ Computer Science*, vol. 8, Article ID e1092, 2022.

[7] W. Niu, Y. Wang, X. Liu, R. Yan, X. Li, and X. Zhang, "GCDroid: Android malware detection based on graph compression with reachability relationship extraction for IoT devices," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11343–11356, 2023.

[8] A. Arora, S. K. Peddoju, and M. Conti, "Permpair: Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2019.

[9] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[10] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based Android malware detection," in *2017 International Conference on Software Security and Assurance (ICSSA)*, pp. 60–65, IEEE, 2017.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] Y. Xu, D. Xu, X. Hong et al., "Structured modeling of joint deep feature and prediction refinement for salient object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, IEEE, 2020.

[13] S. Wang, H. Lu, and Z. Deng, "Fast object detection in compressed video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7104–7113, IEEE, 2020.

[14] R. Hinami, T. Mei, and S. i. Satoh, "Joint detection and recounting of abnormal events by learning deep generic knowledge," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2017.

[15] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed, and R. Klette, "Deep-anomaly: fully convolutional neural network for fast anomaly detection in crowded scenes," *Computer Vision and Image Understanding*, vol. 172, pp. 88–97, 2018.

[16] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[17] X. Li, K. Kong, S. Xu, P. Qin, and D. He, "Feature selection-based Android malware adversarial sample generation and detection method," *IET Information Security*, vol. 15, no. 6, pp. 401–416, 2021.

[18] L. Xiaofeng, J. Fangshuo, Z. Xiao, Y. Shengwei, S. Jing, and P. Lio, "ASSCA: API sequence and statistics features combined architecture for malware detection," *Computer Networks*, vol. 157, pp. 99–111, 2019.

[19] N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang, and Y.-A. Tan, "Hybrid sequence-based Android malware detection using natural language processing," *International Journal of Intelligent Systems*, vol. 36, no. 10, pp. 5770–5784, 2021.

[20] H. Du, H. Yuan, P. Zhao et al., "Ensemble modeling with contrastive knowledge distillation for sequential recommendation," in *46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2023)*, ACM, Taipei, China, July 23–27, 2023.

[21] R. Lazzarini, H. Tianfield, and V. Charissis, "A stacking ensemble of deep learning models for IoT intrusion detection," *Knowledge-Based Systems*, vol. 279, Article ID 110941, 2023.

[22] C. W. Kim, "Ntmaldetect: a machine learning approach to malware detection using native api system calls," 2018.

[23] P. Bhat, S. Behal, and K. Dutta, "A system call-based Android malware detection approach with homogeneous & heterogeneous ensemble machine learning," *Computers & Security*, vol. 130, Article ID 103277, 2023.

[24] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, and Y. Qiao, "A novel deep framework for dynamic malware detection based on API sequence intrinsic features," *Computers & Security*, vol. 116, Article ID 102686, 2022.

[25] R. S. Arslan, İ. A. Doğru, and N. Barişçi, "Permission-based malware detection system for Android using machine learning techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 1, pp. 43–61, 2019.

[26] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.

[27] K. Khariwal, J. Singh, and A. Arora, "IPDroid: Android malware detection using intents and permissions," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020.

[28] Z. Wang, G. Li, Y. Chi, J. Zhang, T. Yang, and Q. Liu, "Android malware detection based on convolutional neural networks," in *3rd International Conference on Computer Science and Application Engineering*, 2019.

[29] B. Wu, S. Chen, C. Gao et al., "Why an Android app is classified as malware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–29, 2021.

[30] H.-J. Zhu, W. Gu, L.-M. Wang, Z.-C. Xu, and V. S. Sheng, "Android malware detection based on multi-head squeeze-and-excitation residual network," *Expert Systems with Applications*, vol. 212, Article ID 118705, 2023.

[31] N. Bakhshinejad and A. Hamzeh, "Parallel-CNN network for malware detection," *IET Information Security*, vol. 14, no. 2, pp. 210–219, 2020.

[32] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1276–1290, 2021.

[33] H. Naeem, X. Cheng, F. Ullah, S. Jabbar, and S. Dong, "A deep convolutional neural network stacked ensemble for malware threat classification in internet of things," *Journal of Circuits, Systems and Computers*, vol. 31, no. 17, Article ID 2250302, 2022.

[34] H. Naeem, S. Dong, O. J. Falana, and F. Ullah, "Development of a deep stacked ensemble with process based volatile memory forensics for platform independent malware detection and classification," *Expert Systems with Applications*, vol. 223, Article ID 119952, 2023.
[35] Androguard, *Androguard*, 2022, https://github.com/androguard/androguard.
[36] T. G. Kim, B. J. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.
[37] Android 6.0, *Google Developer Documentation*, 2022, https://developer.android.google.cn/guide/topics/manifest/permission-element.
[38] VirusTotal, *VirusTotal*, 2022, https://www.virustotal.com/ko.
[39] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2016.
[41] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2018.
[42] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355–369, 2018.
[43] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: effective and explainable detection of Android malware in your pocket," *Network & Distributed System Security Symposium*, vol. 14, pp. 23–26, 2014.
[44] Y. Nishimoto, N. Kajiwara, and S. Matsumoto, "Detection of Android API call using logging mechanism within Android framework," in *International Conference on Security and Privacy in Communication Systems*, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds., pp. 393–404, vol. 127 of *SecureComm 2013. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, Cham, 2013.
[45] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.
[46] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487–499, ACM, 1994.