Hindawi
Security and Communication Networks
Volume 2023, Article ID 9890472, 1 page
https://doi.org/10.1155/2023/9890472

Retraction

Retracted: Anticoncept Drift Method for Malware Detector Based on Generative Adversarial Network

Security and Communication Networks

Received 12 December 2023; Accepted 12 December 2023; Published 13 December 2023

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] Y. Dai, H. Li, Y. Qian, Y. Guo, and M. Zheng, "Anticoncept Drift Method for Malware Detector Based on Generative Adversarial Network," *Security and Communication Networks*, vol. 2021, Article ID 6644107, 12 pages, 2021.

Hindawi
Security and Communication Networks
Volume 2021, Article ID 6644107, 12 pages
https://doi.org/10.1155/2021/6644107

Research Article

Anticoncept Drift Method for Malware Detector Based on Generative Adversarial Network

Yusheng Dai,[1] **Hui Li**,[1] **Yekui Qian**,[2] **Yunling Guo**,[3] **and Min Zheng**[4]

[1]*School of Electronics and Information, Northwestern Polytechnical University, Xi'an, China*
[2]*Network Security Laboratory, PLA Army Academy of Artillery and Air Defense, Zhengzhou, China*
[3]*Electronic and Electrical Experiment Teaching Center, Shaanxi University of Technology, Hanzhong, China*
[4]*Zhongke Yuxin Technology Development (Xuchang) Co., Ltd., Xuchang, China*

Correspondence should be addressed to Hui Li; lh@nwpu.edu.cn

Received 3 December 2020; Revised 22 December 2020; Accepted 30 December 2020; Published 19 January 2021

Academic Editor: Chin-Ling Chen

The number of new malware has been increasing year by year, and the construction of the malware sample space is also changing with time. The existing research studies on malware detection mainly focus on how to improve detection performance and how to effectively detect the evasion malware and improve the detection performance of adversarial samples, while ignoring the concept drift of malware samples over time. The concept drift of the sample will lead to the aging of the detector model, thus resulting in the reduction of the detection accuracy. Concerning this problem, we proposed a malware sample generator based on auxiliary classifier GAN, according to the malware samples generated, to train the detection model. In this paper, the API call sequence is used as a feature to train the improved generative adversarial network, and the trained generator model is used to generate samples that simulate concept drift for the purpose of training detection models. Meanwhile, using the detection results of the detector as the training set again, the generator is used to generate samples, so as to repeatedly train the detection model and improve the anticoncept drift performance of the monitoring model. In this paper, real malware samples and generated samples are used to train the detector model, and malware samples are segmented in a linear time sequence as test sets to verify the effectiveness of the proposed method. The results reveal that the framework can maintain good detection accuracy and effectively mitigate the aging of the detector in a longer time dimension.

1. Introduction

Due to the increasingly complex network environment and the emerging attack methods, malware, as an important carrier of network attacks, is attached to various functions such as destruction, theft, and extortion. Kaspersky's 2019 annual report [1] shows that the number of new malware detected in 2019 reaches approximately 24 million, posing a serious threat to network security worldwide. Therefore, being able to effectively detect malware is of great significance.

Currently, using commercial anti-virus software is the regular method of defense against malware. Commercial anti-virus software used signature-based or heuristic-based methods [2], which offer advantages of high accuracy and fast speed. However, it cannot effectively detect the new malware, and in some application scenarios, cannot timely and effectively defend against malware due to its signature database, which needs to be updated online in real time. There were two types of methods for detecting malware: dynamic detection and static detection [3]. These methods can achieve good detection rate in detection performance [4, 5]. At the same time, some researchers took into account some variant malware [6] and some malware that were difficult to detect [7]. Ensuring the accuracy of the detection algorithm is an important goal of research. However, with the continuous updating of malware, the detection efficiency of new samples will gradually decrease through the detection algorithm trained by the old samples. This phenomenon is called *concept drift* [8].

Recently, more and more researchers have begun to pay attention to the problem of the emerging malware causing a deviation in the sample set [9, 10]. The deviation of the sample usually caused the aging of the detector model, which was an unavoidable problem in machine learning. The purpose of most researches in malware was to improve the precision of detecting malware and the detection of evasion problems that occurred in the malware itself [7, 11, 12]. In spite of the continuous improvement of malware technology, new malware of the same type or in the same family emerged one after another (sample concept drift), which was an evasion problem in itself [8], and most researches failed to deal with it. A detection method based on API call and using Markov chain [13, 14] can effectively mitigate the sample drift for a long time; however, this method was based on static feature extraction and detection, and encountering encrypted malware samples will decrease the generation ability of this type of sample. Ensemble learning has advantages in detecting concept drift compared to other detection models [15]. However, compared with the research using Markov chain or other generation methods, ensemble learning still has the problem of rapid decline in detection performance in a longer time dimension. At the current stage of research on malware sample generation, most of them focus on defending against detector attacks [16–18], and Grosse et al. [19] have proposed that the use of distillation can solve the problem of attack detectors and, to a certain extent, mitigate the decrease in detection rate caused by concept drift. The above methods are based on static detection and have proposed a more effective method to prevent concept drift, but we believe that the statically extracted features have greater weaknesses, and there is some room for improvement in the performance and time length of preventing concept drift.

Our research is inspired by Xu et al.'s [20] research on satellite cloud image prediction. Considering the above problems, this paper proposes a method for detecting malware by using ACGAN [21] combined with recurrent neural network (GRU) to deal with the aging of detectors. The API call sequence of the malware is used, the word vectors in the call sequence are extracted as the features, and the sample features are used as raw data to train ACGAN. The trained ACGAN generator is used to generate new malware samples, which simulate the probability distribution of the original data category, and the generated malware samples are combined with the real samples to train the GRU network. The newly generated samples and real samples are used as training samples to train an untrained GRU network. The samples of concept drift are detected in the GRU network that has been trained and will be retrained according to the detected malware samples, in order to achieve long-term resistance to the samples of concept drift.

Our contributions are described as follows:

(1) We propose an improved ACGAN sample generation framework to mitigate the problem of decreased detection rate caused by concept drift of malware samples

(2) We use the API call sequence as a feature, combining actual malware samples, to generate samples to verify the effectiveness of the proposed method in a long time span, and the performance is better than other recent research

(3) Evaluating the performance of this experiment through experiments and comparisons, and discussing the basis for setting various parameters in this article's model in dealing with the adversarial concept drift

The structure of the remaining sections is as follows: Section 2 introduces related work; Section 3 describes the methods used to explain adversarial concept drift in detail; Section 4 evaluates the quality of the generated samples and the anticoncept drift performance; Section 5 summarizes the full text and proposes further work.

2. Related Work

Deep neural networks have been widely used in various areas of network security, and malware detection is an important branch of this field. Malware detection is usually divided into two types in the literature: dynamic detection and static detection [3]. Static detection usually analyzes the source code of the malware directly, or extracts disassembly code, so that researchers can analyze the program from the aspects of semantics and behavior, which is simple and efficient. The DREBIN [22] method proposed by Arp et al. uses a linear support vector machine (SVM) to analyze high-dimensional binary features. Kapoor and Dhavale [23] used opcodes combined with control flow graphs generated by extracting source code to perform malware classification. Nataraj and Manjunath [24] cleverly used another method to detect malware by converting binary files into grayscale images. Although static detection methods are simple and efficient for most cases, they are vulnerable to mixed attacks such as packaging and encryption. Dynamic detection could effectively overcome the defects of static detection technology by monitoring the running samples and recording various behaviors of samples in a virtual environment. Common approaches included behavior-based malware research [25] and dynamic detection research based on API call sequence [4]. Dynamic detection could effectively deal with the defects of static detection to a certain extent, but this method was likely to suffer from evasion behaviors such as environmental detection, imitation attacks, and injection attacks.

Athiwaratkun and Stokes [5] proposed a method that combined dynamic analysis and static analysis techniques, extracted features from applications, and monitored data flow, network behavior, and other operations of applications in the sandbox. However, this method was still restricted by dynamic detection and the defects of static detection. Therefore, some researchers have taken a different approach to the defects of dynamic detection. Ozsoy et al. [26] proposed using hardware performance counters as features to confront the evasion behavior of malware; however, it had strict requirements on the hardware environment [27]. Smutz and Stavrou [7] proposed using the method of ensemble learning to detect malware through mutual agreement analysis. Dai et al. [11, 28] also used a variety of features

combined with the ensemble learning methods to achieve good detection rates in terms of detection performance and anti-evasion.

However, all the above methods were limited to selecting only a set of known malware datasets for experiments, which tend to cause a deviation in the results of the malware detector [9]. In intrusion detection systems, the model usually changes rapidly due to concept drift. The use of hidden Markov model methods can generate samples to assist classifier learning and reduce the performance degradation of the classifier caused by concept drift [29–31]. It was very difficult to build a model for the continuous detection of malware because the malware would rapidly develop new attack capabilities based on certain vulnerability; therefore, the detector trained by older malware often made biased decisions in dealing with the new emerging malware, which was the concept drift of malware samples [8]. In order to cope with the aging of the detector, that is, the concept drift of malware sample, Onwuzurike et al. [14] proposed the MaMaDroid method, which uses API call sequence combined with Markov chain to build models for multiple types of malware, imitating the unknown malware samples in the mobile terminal predicted to emerge in several years. Similarly, Ficco [13] used a similar method to deal with malware of IoT devices. Hu et al. [32] eliminated the subclassifiers with low accuracy in ensemble learning and retrained them to solve the problem of concept drift. Grosse et al. [19] used the distillation method to effectively resist the aging of the detector caused by concept drift to a certain extent. In addition, research on preventing concept drift focuses on the method of ensemble learning because the diversity of internal detectors in ensemble learning can well summarize the conceptual diversity of samples and improves the generalization ability of models [15]. But, in the field of malware detection, although these researches also use ensemble learning methods [7, 11], their focus is on the behavioral deviation of malware and does not consider the impact of time deviation [10].

3. Methods of Adversarial Concept Drift

The tool used to generate the training set used in this paper is ACGAN and the classification model used is the GRU network. The main steps for detecting evolved malware code are divided into the following: First, the API call sequence of the malware code is extracted from the sandbox, and word2vec is used to convert the sequence into a feature vector that can be used for training. The extracted feature vectors are then used to train the adversarial network. Finally, the trained generative adversarial network is used to generate samples, together with the added actual samples to train the detection model. The main flow of the experiment is shown in Figure 1.

3.1. Features of API Sequence. The features used in this article extract the API call sequence from a large amount of malware code. Cuckoo is an automated malware analysis system commonly known as a sandbox. The sandbox records

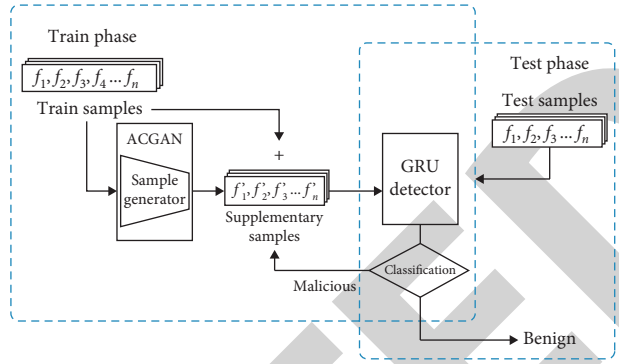

Figure 1: Malware code classification framework with concept drift.

the functions and data executed by the malware code by monitoring the operation of the malware inside it. The function sequence extracted by the sandbox is taken as the raw data, and the word2vec method is used to convert the API sequence into a feature vector. This vector is used as a training feature for the adversarial generation network and the GRU detection network.

Due to the different order of API call sequence for different types of malware, the combination of some functions can express stronger maliciousness. The word2vec method is a shallow neural network used to map the sequence of API functions into an N-dimensional real number vector according to their order, and use real numbers to express their similarity through the distance between functions. In this paper, the CBOW model of layered SoftMax, the corpus of CSDMC 2010 API, and the samples we used are adopted to supplement the corpus incrementally. We arrange the extracted API call sequence into a group in the form of $F = \{f_1, f_2, f_3, \ldots, f_n\}$, in which f_n represents the function corresponding to the position in the sequence. In the call sequence F, set the current function f_n to average $2c$ of the surrounding functions, and the context constant $c = 2$, then the f_n vector is expressed as

$$\text{context}(f)_n = \frac{1}{2c} \sum_{i=1}^{2c} f_i. \tag{1}$$

In this paper, the number of training iterations is set to 15, and the average value calculated for each training iteration is updated in a small range after gradient rise. Finally, it outputs a set of vectors $V = \{\text{label}, v_1, v_2, v_3, \ldots, v_n\}$ combined with the label of the current samples, where label is in the form of one-hot. The vector value of each API is mapped to an array of fixed length, and the other functions that did not appear in this training are filled with 0, making it a vector of fixed length.

We use the Gensim package to implement the training and generation of word2vec and incrementally update each newly learned function, that is, new words are added into the existing model without needing to relearn all.

3.2. Network Model. We intend to use two neural network models to conduct this experiment. The experiment is divided into two stages: in Stage 1, improved ACGAN is used to

combat the malware data of generating network training to save the generating model; in Stage 2, the generated data is combined with the actual data into a whole complete dataset that is trained by using the GRU network. Since the discriminator in ACGAN needs to be retrained, it cannot be used for direct classification. Therefore, the GRU classifier we set separately here is used to generalize and adapt to longer-term evolution malware through continuous retraining.

3.2.1. ACGAN. It has been found in the previous classification detection research of malware that malware samples can reflect the similarity of such malware samples through clustering in high-dimensional space [28]. However, in the case of insufficient samples, even though the most advanced current detector is used, it cannot fully fit the distribution of the generated data. Meanwhile, by analyzing the source code of some malware samples, it can be seen that the malware Sample A and the malware Sample B belong to two different families in family classification. However, there are similarities in certain details that are also a part, which is not easy to be handled by the general classifier.

This paper uses ACGAN, which is a variant of generative model classification based on likelihood estimation. Based on the original GAN, the generator and discriminator are trained by using the label data and the label information is reconstructed at the end of the discriminator at the same time, so as to improve the generation effect of the generator. Therefore, ACGAN is more suitable for the experimental scenario in this paper and can try to describe the edge of the sample data distribution.

ACGAN adds a random number that conforms to the Gaussian distribution to each class sample at the input of the generator G to assist in generating the class condition samples as well as improving the quality of the generated samples. The inputting of discriminator D is a real-type sample and a type of generated sample. The samples input discriminator contained a part of the fake data produced by the generator which is only used to train the discriminator. Our ACGAN model is shown in Figure 2.

The generator is set to include the GRU cells and the fully connected layer, and the last layer of the fully connected layer sent data into the two activation functions, sigmoid and SoftMax, respectively, to map the generated data into vectors and obtain true and false outputs and category outputs. A batch normalization layer is connected between the fully connected layers of the generator and the discriminator to ensure that the input of each layer of the neural network has the same distributions. Leaky-rectified linear unit (LeakyReLU) is used for the generator and discriminator. The dropout strategy is selected for discriminator, with a value of 0.5, which means that only half of the neurons in each layer participate in the calculation and make of the discriminator overfit. The internal model of the generator and discriminator is shown in Figure 3.

The objective function for training ACGAN is divided into two parts. L_S is the log likelihood of whether the data are true or not. L_C is the log likelihood of the accuracy of the data category.

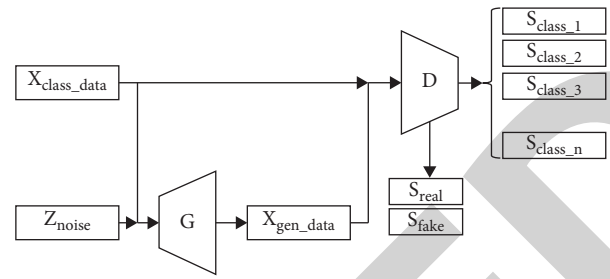

Figure 2: ACGAN model.

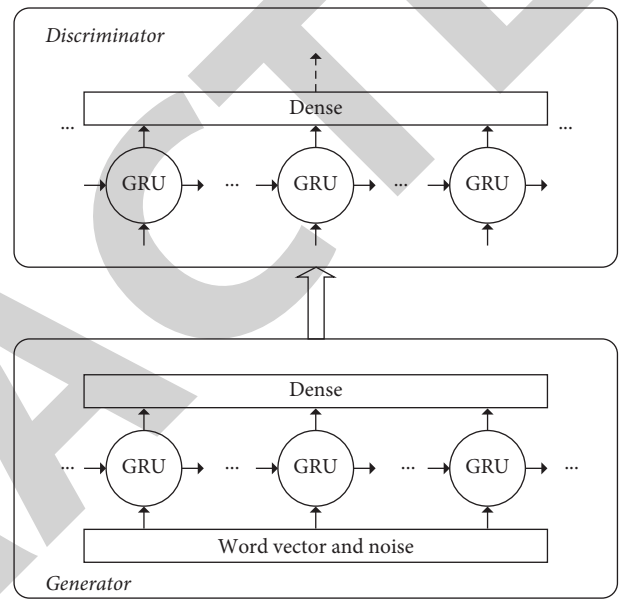

Figure 3: The internal structure of the generator and discriminator.

$$L_S = E\left[\log P\left(S = \text{real}|X_{\text{real}}\right)\right] + E\left[\log P\left(S = \text{fake}|X_{\text{fake}}\right)\right],$$
$$L_C = \sum_{i=1}^{n} E\left[\log P\left(C = C_i|X_{C_i}\right)\right] + E\left[\log P\left(S = C|X_{\text{fake}}\right)\right]. \tag{2}$$

In this paper, the purpose of ACGAN is to optimize the category generation of the generator as much as possible; so the weight of the discriminator game is slightly reduced and the weight of the cost function of the generator is increased, so that the purpose of the overall network is to minimize the $L_C - L_s$ of discriminator G and maximize the $L_C + L_s$ of discriminator D. ACGAN uses the Adam optimizer to optimize the loss function. The learning rate of the Adam optimizer is 0.0002, and the exponential decay rate of the first-order moment estimation is 0.5. Furthermore, Adam can automatically adjust the learning rate to prevent converging of a local optimal risk effectively.

3.2.2. GRU Network. Because the sequence of API calling is time-dependent and the calling sequence of different functions before and after would result in different levels of

malicious expression, it is more appropriate to use the RNN-type network model; the validity of using two-layer GRU network to detect malware is confirmed by previous research [11, 33]. The GRU network is a time-circle neural network and GRU is easy to converge by using fewer parameters compared with LSTM.

Our GRU network adopts a double-layered GRU network as the backbone of the neural network. When a vector $V = \{f_1, \ldots, f_i, \ldots, f_t\}$ of an API call sequence is obtained, f_i is representing a certain point-in-time on the calling sequence vector. We embed the vectorVin the embedding layer as an input and designate the embedding to input dimension. Then, the vector of this layer will initialize the smaller random numbers, and this layer will be updated in the subsequent training.

The word vector $f_{(t-i)}$ is mapped by the embedding layer to output the vector x_i. The GRU layer takes the output x_i of the embedding layer as input. The number of cells in each GRU layer is consistent with the input word vector, the API calling sequence, that is, if the length of the word vector extracted from the sample is l, the number of GRU cells also is l. In a single GRU cell, each cell receives the output of the hidden layer from the previous cell as an input and the value obtained after calculation is used as the output of the hidden layer of the cell, which is recorded as h_i. There are two gates included in one GRU cell; one is the reset gate which is used to determine the amount of information for forgetting the past and is recorded as r_i. The other is the update gate, which is to decide what information is to be discarded and what information is to be added, with recording of such information as z_i. Figure 4 shows our double-layer GRU neural network model, with the API sequence propagating in order from front to back.

In Figure 4, each GRU cell should output the hidden state of the current cell h_i, which is calculated by the current sequence of the embedded API x_i and the state of the $i-1$ GRU cell jointly h_{i-1}. The reset gate and update gate are represented by adopting the sigmoid function, with the representation of σ. The results of reset gate and update gate are calculated by the tanh function to obtain the candidate hidden state of $\tilde{h}_i$. The process of calculating a GRU cell is shown as follows:

$$\begin{aligned} z_i &= \sigma\left(W_z x_i + U_z h_{i-1}\right), \\ r_i &= \sigma\left(W_r x_i + U_r h_{i-1}\right), \\ \tilde{h}_i &= \tanh\left(W x_i + U\left(r_i^* h_{i-1}\right)\right), \\ h_i &= \left(1 - z_i\right) * h_{i-1} + z_i * \tilde{h}_i, \end{aligned} \tag{3}$$

where, W and U are the weight matrices that represent the function of the API sequence at the current time node in the GRU neural network. When more GRU layers are added, the learning ability will be improved to a certain extent, but the detection performance cannot be improved once the number of layers is increased to a certain number. Therefore, the GRU neural network we use in this article is set to double-layer to achieve the best results.

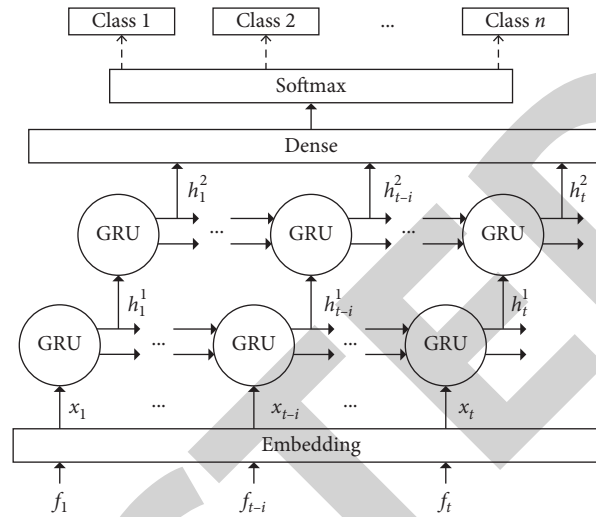

Figure 4: Double-layer GRU neural network model.

After all the hidden states of the GRU layer are output, it is necessary to pass them to the fully connected layer. The fully connected layer takes all GRU hidden states as the input to carry out a weight calculation and obtain a new vector value through the activation function. The fully connected layer in this paper adopts the ReLU as the activation function. The output of each neuron in the fully connected layer is shown as follows:

$$h_i^l = \max\left(0, W_i^l h_i^{l-1} + b_i^l\right), \tag{4}$$

where, l represents the number of layers, i represents the i-th neural cell of the current layer, and W and b are the weight and offset of the current cell, respectively. After the fully connected layer is output, the SoftMax function regression is used to obtain the results of this classification.

4. Experiment Evaluation

4.1. Experimental Environment and Dataset. The experimental environment of this paper is divided into two parts. One part is the sandbox environment, used to extract the API sequence of the malware. The other part is the algorithm-operating environment. Sandbox environment is operated by using a regular PC with a CPU of Intel (R) Core (TM) i5-6500 @ 3.20 GHz, 8 GB memory, and a 64-bit version of the operating system of Ubuntu 16.04, where the guest environment of the sandbox is set as the 2 GB of memory, with a 32-bit sp1 operating system of Windows 7. The algorithm operation platform uses the CPU of Intel (R) Core (TM) i7-6800K, with the NVIDIA 1080Ti graphics card and the 11 GB video memory as well as the CUDA 10 toolkit.

The dataset used in this paper should be consistent with the dataset of the study [11], with a total sample size of about 27 k. All the samples can be divided into 214 families, 8 large types of malware samples, and a small number of unclassifiable samples according to the detection results of VirusTotal [34]. See Table 1 for information of samples that are divided based on the sample types. Because Hacktool has

Table 1: Malware sample category.

Sample category	Category no.	Quantity
Backdoor	0	4652
Flooder	1	527
Worm	2	3821
Trojan	3	2047
Adware	4	3226
Exploit	5	541
Constructor	6	602
Hacktool	7	655
Other malware	8	662

various functions, and the "Other Malware" are malware that cannot be classified or undefined, it cannot be used. We only use the data of the first 7 categories in this experiment.

In addition, according to the results of VirusTotal, we can obtain some sample generation information including the sample creation time, the first submission time, and so on. Since the creation time may be affected by the local timestamp of the malware author, resulting in an inaccurate time information, we take the time of VirusTotal's first submission analysis as the sample discovery time to obtain the relationship between our sample number and time (Table 2).

4.2. Generate Sample Classification Performance. This section discusses the effects of training improved ACGAN (hereinafter referred to as ACGAN-GRU) and using labeled samples generated by ACGAN-GRU on the classification of malware. The number of training cycles of ACGAN-GRU is set to 10000 epochs, and the loss results of the generator G and discriminator D are as shown in Figure 5.

As can be seen from the figure, the loss results of the generator and discriminator tend to be stable after about the 3000th epoch. The samples are set to generate after the 5000th epoch, and a large number of generated samples are clustered, based on which the selected generated samples have close Euclidean distance to that of actual samples, while the number is twice that of the actual training samples. The generated samples are fused with the actual samples and added into the GRU network for training. The training results are presented with the confusion matrix, as shown in Figure 6.

Figure 6(a) shows the classification results of using GRU directly for training without using generated samples. Figure 6(b) shows the results of training using generated samples. The confusion matrix is mainly used to compare the classification results with the actual measured value. The actual accuracy is displayed on the diagonal, with each row representing the proportion of the cases being predicted as such classes and each column representing the proportion of the actual classification. In the ideal state, each value on the diagonal should be 1. As can be seen from the training matrix, GRU network, without the training for generated samples, can obtain the classification results maintained with an average accuracy of 97%. While after using ACGAN-GRU for generating samples, which are trained together with

Table 2: Number of malware samples distributed by time.

Annual	In and before 2010	2011	2012	2013	2014	2015
Number of samples	4365	5899	3438	1702	906	423

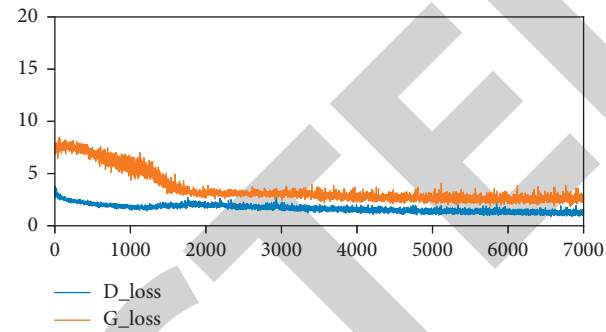

Figure 5: Loss results of ACGAN generator and discriminator.

the actual samples, GRU has a slight improvement in overall classification accuracy. It shows that the samples generated by ACGAN-GRU are more authentic. Also, the training in which the samples generated by ACGAN-GRU are fused leads to a slight decrease in classification accuracy for some classes. As can be seen from the figure, the detection accuracy for backdoor malware decreases by nearly 1%, for which the reason lies in the various functions of backdoor malware. In contrast, the malware of the 2nd, 3rd, and 4th classes, corresponding to worm, trojan, and adware, respectively, have more simple functions than backdoor malware.

4.3. Analysis on Generated Samples. Every year, the generation of new malware brings a lot of difficulties to the analysis and research of malware [10], and the malware of the same class will use new coding tools as time goes by and coding technique advances, which might result in different code optimization, leading to the detection failure. A lot of researches have not taken into account the effect of malware with time going by, i.e., the original dataset for training. As the malware samples are continuously added into the dataset, the whole dataset will deviate from the original space.

The experiment first verifies the generation effect of the sample. We use five generative models: variational autoencoders (VAE), condition generative adversarial network (CGAN) [35], auxiliary classifier generative adversarial network (ACGAN) [21], and ACGAN-GRU and ACGAN-LSTM, the variants of ACGAN. CGAN neural network model is set basically consistent with ACGAN neural network model. CGAN is an expansion of traditional GAN, that is, in the original network structure, the input of discriminator and generator are added with label information to help improve the quality of the generated samples. The discriminator and generator of ACGAN are consistent with those of CGAN at the input,

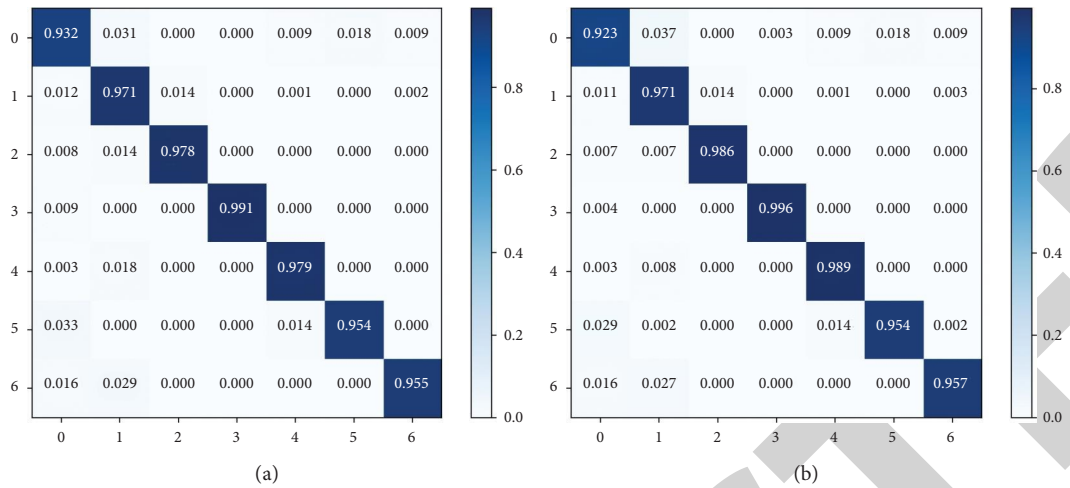

Figure 6: Loss results of ACGAN generator and discriminator. (a) Training unused generated samples and (b) Training used generated samples.

while the output is added with reconstruction of label information, so as to enhance the generation effect through cost function.

The basic structure of VAE (%) is a multilayer perception neural network, there being multiple hidden layers of full connections between input and output, with the whole network being divided into two parts, encoder and decoder. Within the network structure of the encoder, the neurons decrease layer-by-layer from the input layer to the coding layer, which is actually a dimension reduction process. The neurons increase layer-by-layer from the coding layer to the output layer of the decoder, so as to restore the data.

We will compare the five trained generated models. The loss functions used by these models and the network parameters of the generator and discriminator are listed in Table 3. The results of the generated data are shown in Table 4.

As can be seen from the experimental results, VAE has no advantage in generating results because of data loss in the process of encoding and decoding. However, the data generated in the two methods based on GAN can achieve the highest accuracy of over 98% on the discriminator, but the information the discriminator could give is just to provide a measurement index for the improvement of the generator. After CGAN and ACGAN have experienced a training of over 5000 epochs, only the generated data can be input and train GRU. The highest accuracy of the data generated by ACGAN, nearly 90%, can be obtained by classifying the test sets. But if the generated datasets are added into real datasets, there will be no significant difference in the classification results. In this paper, the main reason for choosing ACGAN and the use of GRU instead of MLP in the generator and discriminator is that GAN uses a neural network with sequence features to better adapt to API sequence features [36]. Because the number of image features is more than the number of API sequence features, using convolutional layers in the scenarios described in this paper can easily cause the model to collapse.

Table 3: Training parameters.

Models	Learning rate	Batch size	Optimizer
VAE	0.00	64	RSProp
CGAN	0.002	64	Aam
ACGAN	0.002	64	Aam
ACGAN-LSTM	0.002	32	Aam
ACGAN-GRU	0.0002	32	Adam

Table 4: Generated data classification.

Generation method	Discriminator accuracy (%)	Accuracy without real data (%)	Accuracy with real data (%)
VAE	—	72.1	95.4
CGAN	96.1	87.3	95.9
ACGAN	96.4	89.9	96.2
ACGAN-LSTM	98.4	90.1	98.0
ACGAN-GRU	98.3	90.1	97.9

We will use the malware samples of the same period (in 2011 and before) to use the above 5 models to generate new sample sets. Then, we use malware samples (before 2010 to 2012) to train the GRU classifier, and use the samples generated by the model for testing, and use the ROC curve and AUC value to evaluate generated sample quality. ROC is shown in Figure 7. As can be seen from the results in the figure, the samples generated by VAE in several models have the worst detection efficiency and lower fidelity. Comparing two GANs without the sequence model, ACGAN's AUC value is slightly higher than CGAN. The ACGAN model with sequence generates the highest degree of fitting, and the generated sample has a higher degree of fitting with the sample with concept drift. At the same time, it can be seen from the results in the figure that GRU and LSTM have similar generating capabilities in the dynamic API call sequence. However, from the model structure of GRU and LSTM, the calculation amount of GRU is relatively low.

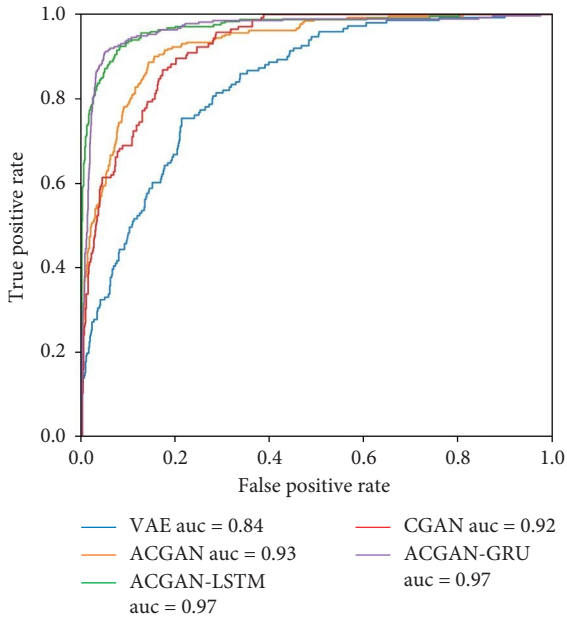

Figure 7: Sample-generated fidelity.

4.4. Drift Detection Evaluation. The moment when the experimental data is submitted to VirusTotal for the first time is regarded as the moment when the samples appear for the first time, which certainly will not affect the experimental results [10]. The sample drift detection is aimed at evaluating the classifier over time, for which a general neural network detector (GRU network, without training-generated samples), an adversarial detector, and a detector to adversarial drift are selected in this paper. We chose MaMaDroid [14] and Grosse et al. [19] but used dynamic API call sequences as features for comparison experiments. We also used Smutz and Stavrou [7] research, the ensemble learning method, as the compared experiment in this paper. This paper refers to the MaMaDroid model to reproduce the prediction process of the Markov chain. The distillation method sets the distillation temperature $T = 20$ of the SoftMax layer. The ensemble learning method in article [7] is based on 100 SVM classifiers, and the parameters of the classifiers, such as penalty, kernel function, gamma, and other settings, are kept as inconsistent as possible to ensure the diversity of classifiers.

In this experiment, samples earlier than 2011 are used as training sets, and samples in each year from 2012 are used as test sets to evaluate the accuracy of the detector. First, we need to compare and verify the impact of ACGAN-generated samples in different training epochs on the classifier. We selected two groups of samples ("2011" and "2012") and the samples generated by the generator to train the classifier. The experimental results are shown in Figure 8.

In the figure, "Classification" represents the accuracy of the classifier after training, and "Evasion" represents the accuracy of detecting concept drift. It can be seen from the figure that ACGAN using GRU can improve the performance of the classifier under continuous training, reaching the maximum at about the 8000th epoch. However, the classification performance of the classifier for concept drift reaches the maximum when it is close to the 5000th epoch, and then gradually decreases. The reason is that the generator mode collapses and cannot output more samples, which leads to the training of the classifier tending to overfitting. Figure 9 shows the relationship between the accuracy of various detectors and samples at different times.

It can be seen from Figure 9 that the concept drift of the long time dimension causes the detector to greatly affect the accuracy of malware detection. Samples with detectors that do not use adversarial samples for more than two years will result in a detection accuracy of less than 90%. The method in this paper can maintain a relatively accurate detection result within a two-year time range, and the detection accuracy rate is better than other research. And for more than 3 years, the detection rate will be reduced due to the concept drift of malware samples. It can be seen from the figure that since the ensemble SVM method is a passive defense, in the face of the concept drift caused by the new samples, the detection accuracy rate is above 81% in samples of two years or more. The accuracy of the MaMaDroid method is similar to the distillation method, but it is slightly lower than the method in this paper. We use the F1-score to evaluate our method with other methods. After cross-validation, the best performance statistics are shown in Table 5. It can be seen from the data obtained from the experiment that our method is able to obtain a higher F1-score in detecting the performance of long-term evolution malware.

The main goal of this paper is to mitigate the performance degradation of the classifier caused by the concept drift. We set up a retraining classifier to solve this problem. We conducted the following two experiments: Exp. 1 Learn new samples automatically and retrain. Step (1) samples from the previous year are taken as the original training set and twice as many samples are generated for training. Samples from the second year are used as the test set for testing; (2) malware data detected from the test sets are taken as the training sets of the current year and twice as many samples are generated for training. Samples from the next year are used as the test set; (3) steps (1)-(2) are repeated. Note that the retraining mentioned was not to train GRU network again but to train on a previously trained GRU network. But, the ACGAN model requires the use of old samples combined with new samples to retrain, rather than repeated training based on the original trained model. Exp. 2 Verify the generalization ability of the retrained network to detect old samples. Whenever the newly generated samples are used for model retraining, the old real samples are used as the test set to verify the accuracy of the classifier. We conducted experiments according to the above method, the results of which can be seen in Figure 10.

Because the new sample and its generated sample retrain the classifier model, it may cause the migration of the model so that the classification performance of the classifier for early samples is reduced. The figure legend 'reverse test' evaluation will use the earlier sample group in Table 2 as the test set. From the figure, we can see that the classifier model can basically maintain a high accuracy rate, proving the generalization performance of the model to be good. After

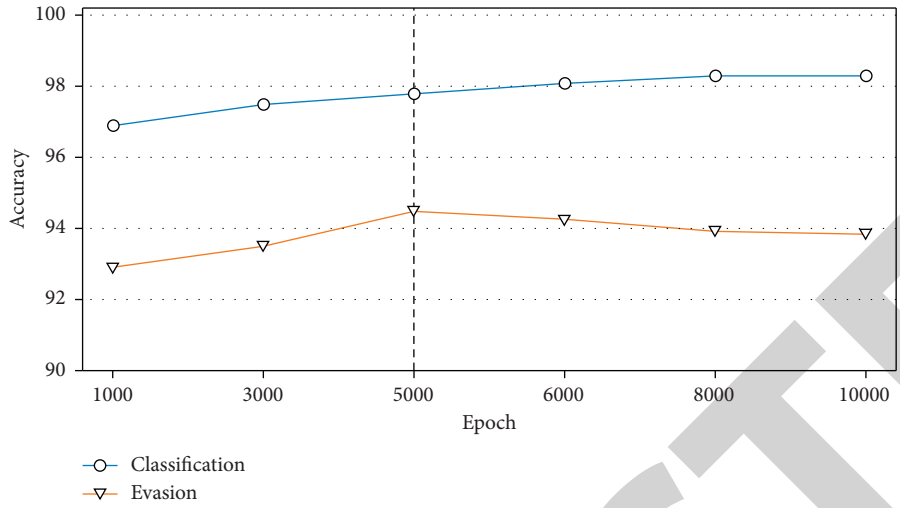

Figure 8: Classification accuracy and evasion detection accuracy in different epochs.

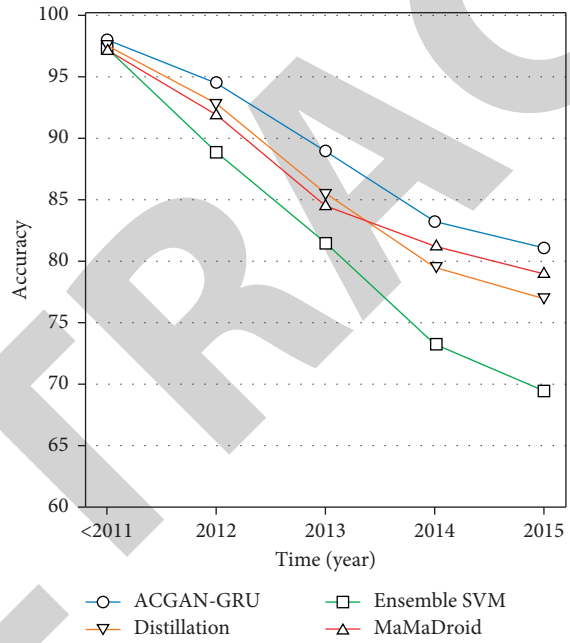

Figure 9: Influence of time concept drift on accuracy.

Table 5: Comparison of our method with MaMaDroid, distillation method, and integrated SVM method in precision, recall, and F1-score value.

Method	Precision, recall, F1-score														
	<2011			2012			2013			2014			2015		
ACGAN-GRU	0.981	0.978	0.979	0.948	0.953	0.950	0.891	0.897	0.893	0.839	0.843	0.841	0.808	0.817	0.812
MaMaDroid	0.979	0.976	0.977	0.937	0.934	0.935	0.856	0.851	0.853	0.821	0.818	0.819	0.792	0.789	0.791
Distillation	0.980	0.978	0.979	0.939	0.940	0.940	0.861	0.862	0.862	0.803	0.798	0.8	0.776	0.769	0.772
Ensemble SVM	0.976	0.981	0.978	0.922	0.924	0.923	0.828	0.839	0.833	0.749	0.753	0.751	0.711	0.718	0.714

all the actually detected malware samples are extracted and retrained, it can see from the figure that the detection accuracy after retraining is significantly improved. The accuracy in the second year remains above 90%, and in the third year, it still remains around 85%. To sum up, the proposed method has good detection performance in confronting time-varying drift samples, confirming the feasibility of the proposed method in this paper.

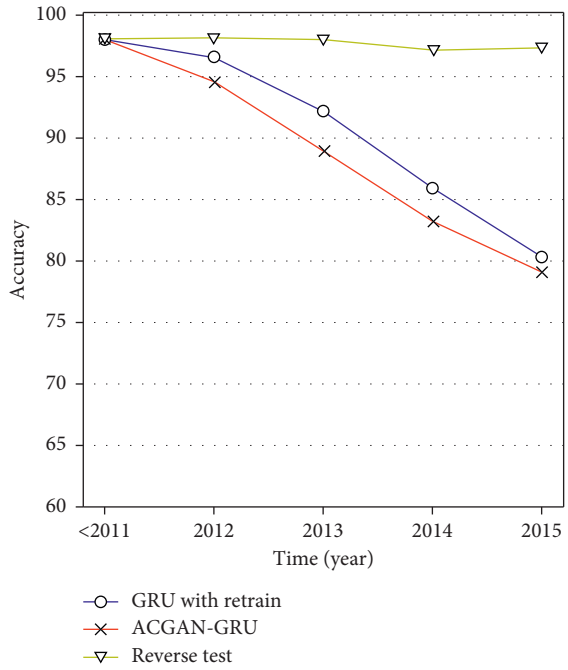

Figure 10: Retraining sample and reverse test results.

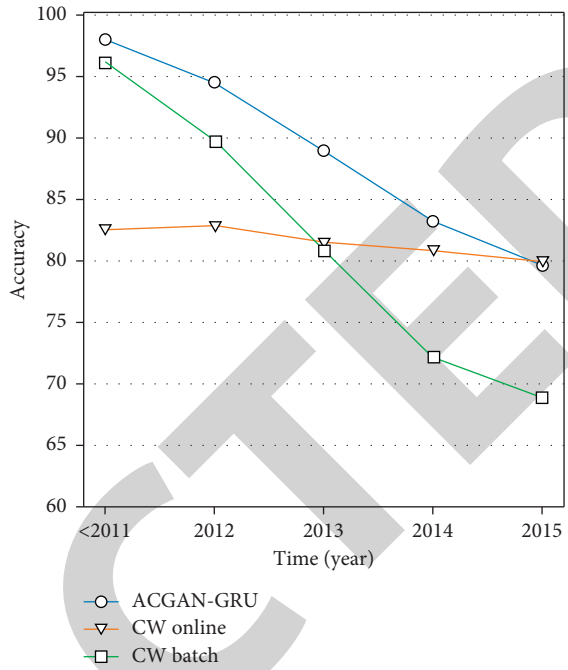

Figure 11: Retraining sample and reverse test results.

4.5. Compared with Online Classifier. In addition, as an effective means to deal with the concept drift of input data, online classifiers are widely used in research work. We use the same ACGAN-GRU configuration as in Section 4.4 for experiments. We also chose an adaptive and scalable online malware detection method, CASANDRA [37]. We use the confidence-weighted linear classifier in the article and the same parameters as the original article. As far as we know, current online malware detection research is based on static analysis methods. In order to be used in the comparative experiments in this article, we use the same feature sequence as the method in this article for online method training and detection. In order to compare the difference between the online method and the offline method (batch), we also compared the offline training accuracy of the CW classifier. The experimental results of using long-term evolution malware samples are shown in Figure 11.

It can be seen from the figure that the overall accuracy of the offline learning method is higher than that of online learning, but the online learning method can continuously learn by itself to continuously update the weight of the learning model, so that the detection classification accuracy is maintained at a relatively stable level. If you use the method combined with retraining in this article, you can obtain a higher classification accuracy than online learning in a long time range. Usually, static analysis uses features that contain more information than dynamic analysis, but the shortcomings contained in static analysis have been described above. Because the features used in this article are obtained by dynamic analysis methods, dynamic analysis is more time-consuming in malware detection and classification research; therefore, the use of offline training is more suitable for the application scenarios of dynamic analysis.

5. Conclusion

The existing researches on malware detection do not effectively deal with the aging of the detector and the concept drift of time-varying samples. This paper takes advantage of the special performance of ACGAN in generating samples with class labels and proposes an improved ACGAN generator model for malware samples training, which uses the API call sequence extracted dynamically as the feature. This model can maximize the trend of concept drift of malware samples and can effectively mitigate the impact of concept drift on the performance of malware classifiers. In this paper, real malware samples are used as input data for the generation model, and the quality of the generated samples is verified. At the same time, samples are generated according to the detection results of the classifier, and models are retrained based on the old classifiers, so that the classifier model can maintain good detection accuracy in a longer time range. After the verification of the method designed in this paper, the experimental results show that the model proposed in this paper achieves the expected effect against sample concept drift and in mitigating the aging of malware detectors, and the experimental results are better than similar studies.

In the next step, we expect to carry out research in two aspects: firstly, the generation of malware samples and the concept drift of malware samples will be further studied, and the relationship between the spatial probability distribution of samples and the generation model will be studied in detail; secondly, research will be carried on the detection of malware of special types to evaluate their impact on detector performance.

Data Availability

The malware data used to support the findings of this study have not been made available because the owner has no authorization.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61571364) and the Innovation Foundation for Doctoral Dissertation of Northwestern Polytechnical University (CX201952).

References

[1] *Kaspersky Security Bulletin 2019*, 2019, https://securelist.com/ksb-2019/.

[2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *Proceedings of The 5th Conference on Information and Knowledge Technology*, pp. 113–120, IEEE, Shiraz, Iran, May 2013.

[3] N. Idika and A. P. Mathur, "A survey of malware detection techniques," vol. 48, Purdue University, West Lafayette, IN, USA, 2007.

[4] E. Amer and I. Zelinka, "A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence," *Computers & Security*, vol. 92, 2020.

[5] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2482–2486, IEEE, New Orleans, LA, USA, March 2017.

[6] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: a large-scale, automated approach to detecting ransomware," in *Proceedings of the 25th USENIXSecurity Symposium (USENIXSecurity 16)*, pp. 757–772, Austin, TX, USA, March 2016.

[7] C. Smutz and A. Stavrou, "When a tree falls: using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2016.

[8] R. Jordaney, K. Sharad, S. K. Dash et al., "Transcend: detecting concept drift in malware classification models," in *Proceedings of the 26th USENIXSecurity Symposium (USENIXSecurity 17)*, pp. 625–642, Vancouver, British Columbia, Canada, January 2017.

[9] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Are your training datasets yet relevant?" in *Proceedings of the International Symposium on Engineering Secure Software and Systems*, pp. 51–67, Springer, Milan, Italy, March 2015.

[10] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: eliminating experimental bias in malware classification across space and time," in *Proceedings of the 28th USENIXSecurity Symposium (USENIXSecurity 19)*, pp. 729–746, Santa Clara, CA, USA, August 2019.

[11] Y. Dai, H. Li, Y. Qian, R. Yang, and M. Zheng, "Smash: a malware detection method based on multi-feature ensemble learning," *IEEE Access*, vol. 7, pp. 112588–112597, 2019.

[12] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Rhmd: evasion-resilient hardware malware detectors," in *Proceedings of the Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 315–327, Boston, MA, USA, October 2017.

[13] M. Ficco, "Detecting IoT malware by Markov chain behavioral models," in *Proceedings of the 2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 229–234, IEEE, Prague, Czech Republic, June 2019.

[14] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: detecting android malware by building Markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, p. 14, 2019.

[15] L. L. Minku and X. Yao, "Ddd: a new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, 2012.

[16] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," in *Proceedings of the Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, February 2018.

[17] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art api call based malware classifiers," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 490–510, Springer, Heraklion, Crete, Greece, September 2018.

[18] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14, IEEE, San Francisco, CA, USA, May 2019.

[19] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proceedings of the European Symposium on Research in Computer Security*, pp. 62–79, Springer, Oslo, Norway, September 2017.

[20] Z. Xu, J. Du, J. Wang, C. Jiang, and Y. Ren, "Satellite image prediction relying on gan and Lstm neural networks," in *Proceedings of the ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, Shanghai, China, May 2019.

[21] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651, Sydney, Australia, August 2017.

[22] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: effective and explainable detection of android malware in your pocket," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, vol. 14, pp. 23–26, San Diego, CA, USA, February 2014.

[23] A. Kapoor and S. Dhavale, "Control flow graph based multiclass malware detection using bi-normal separation," *Defence Science Journal*, vol. 66, no. 2, pp. 138–145, 2016.

[24] L. Nataraj and B. S. Manjunath, "Spam: signal processing to analyze malware [applications corner]," *IEEE Signal Processing Magazine*, vol. 33, no. 2, pp. 105–117, 2016.

[25] Y. Ding, X. Xia, S. Chen, and Y. Li, "A malware detection method based on family behavior graph," *Computers & Security*, vol. 73, pp. 73–86, 2018.

[26] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE*

Transactions on Computers, vol. 65, no. 11, pp. 3332–3344, 2016.

[27] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: myth or fact?," in *Proceedings of the Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 457–468, Incheon, Korea, June 2018.

[28] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[29] S. Shitharth, K. Sangeetha, and B. P. Kumar, "Integrated probabilistic relevancy classification (prc) scheme for intrusion detection in scada network," in *Design Frameworks for Wireless Networks*, pp. 41–63, Springer Nature, NY, USA, 2019.

[30] S. Shitharth and D. P. Winston, "A new probablistic relevancy classification (prc) based intrusion detection system (ids) for scada network," *Journal of Electrical Engineering*, vol. 16, no. 3, pp. 278–288, 2016.

[31] S. Shitharth and D. Prince Winston, "An enhanced optimization based algorithm for intrusion detection in scada network," *Computers & Security*, vol. 70, pp. 16–26, 2017.

[32] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The concept drift problem in android malware detection and its solution," *Security and Communication Networks*, vol. 2017, Article ID 4956386, 2017.

[33] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang, "nlsalog: an anomaly detection framework for log sequence in security management," *IEEE Access*, vol. 7, pp. 181152–181164, 2019.

[34] *Virustotal*, 2019, https://www.virustotal.com/.

[35] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[36] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (gans): a survey," *IEEE Access*, vol. 7, pp. 36322–36333, 2019.

[37] A. Narayanan, L. Yang, L. Chen, and L. Jinliang, "Adaptive and scalable android malware detection through online learning," in *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 2484–2491, Vancouver, British Columbia, Canada, July 2016.