2025 IEEE 6th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT) | 979-8-3315-2228-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/AINIT65432.2025.11035526

GAN-Based Adversarial Attack Against Malware Detection

Liyan Tao*
Beijing University of Technology
BeiJing, China
*e-mail: 641217688tly@gmail.com

Hanshu Rao
Beijing University of Technology
BeiJing, China

Linyu Zhang
Beijing University of Technology
BeiJing, China

Ruotong Sun
Beijing University of Technology
BeiJing, China

Shizheng Wang
Beijing University of Technology
BeiJing, China

***Abstract*—Traditional malware detection methods based on signatures or behaviors are increasingly struggling to cope with new types of attacks, prompting a shift towards detection methods based on machine learning and deep learning. Compared to traditional detection mechanisms, machine learning and deep learning-based methods possess stronger feature extraction and generalization capabilities. This paper presents a malware detection model implemented using a Residual Neural Network (ResNet), capable of accurately distinguishing between malicious and benign software. However, detection systems based on machine learning or deep learning have issues with robustness, as malware authors can adjust the features of their malware to evade detection. We implemented an attack method based on Generative Adversarial Networks (GAN), targeting a trained ResNet detector, and used GAN to generate adversarial malware samples. The results demonstrate that this attack strategy can dynamically generate adversarial malware samples while realistically replicating actual attack scenarios.**

Keywords-Malware detection; GAN; ResNet; Deep learning; Adversarial malware

I. INTRODUCTION

In the realm of malware detection, traditional methods based on signatures or behaviors have begun to falter due to their reliance on existing malware signatures and the limitation to only detect known malware, struggling to keep pace with new attacks and variants [6]. An increasing number of researchers have shifted their focus to machine learning-based detection methods [4]. Benefiting from the generalization capabilities of models trained with machine learning algorithms, these methods generally perform well in detecting or classifying malware.

Although traditional machine learning methods have achieved significant results in malware detection, they are time-consuming in feature engineering and may miss critical feature information. At the same time, machine learning detection models also suffer from poor performance when dealing with unknown or Zero-Day malware [6]. Compared to machine learning, deep learning architectures possess stronger capabilities for handling unstructured data and model generalization. Deep learning models are capable of automatically extracting and processing complex features from raw data, providing a more comprehensive and efficient solution for the detection and classification of malware. Consequently, solutions for machine learning-based malware detection are gradually evolving towards deep learning architectures. This paper presents a detection model based on the Residual Neural Network (ResNet), which accepts grayscale images converted from arrays of system-level APIs called by software as input, to distinguish between benign and malicious software.

While the aforementioned machine learning or deep learning-based malware detection methods have achieved relatively high accuracies, the defense strategies against malware continue to face security threats. Hu and Tan [9] pointed out in their article that detection models are often designed and trained with a focus on detection performance without considering the possibility of malware authors using adversarial techniques to bypass the models. Li et al. [12] also argued that machine learning algorithms are vulnerable to adversarial attack techniques that attackers could use to evade or circumvent detection.

In response to these security vulnerabilities in machine learning or deep learning-based malware detection, many researchers have targeted detection models with effective attack strategies. However, the aforementioned strategies for generating adversarial malware examples typically have some drawbacks. First, If the developers of malware detection models can collect adversarial malware examples and use these new samples to retrain the detection models, then the accuracy of the detection models will once again be improved [15]. Moreover, some studies treat the attacked detection model as a white box, assuming the internal structure of the model is known [7]. However, in actual scenarios, malware detection models are often deployed to the cloud or encrypted, preventing attackers from accessing the details of the models, making the target effectively a black box, which can lead to the ineffectiveness of certain white box-oriented attacks.

To address the above flaws, our team have implemented an attack scheme that dynamically generates adversarial malware samples. We chose to use Generative Adversarial Networks

(GAN) to generate adversarial samples because the adversarial training mechanism of GANs ensures that the generator can continuously iterate and update to breach the detector's defense, while the GAN's detector can also act as a surrogate detector during simulated attacks to fit the detection logic of the black box detector, allowing us to launch attacks without understanding the internal structure of the black box detector. Our model primarily consists of a GAN and a ResNet, where the ResNet assumes the role of the black box detector, and the GAN dynamically generates adversarial examples against the black box detection. The results show that our attack strategy can effectively bypass the detector's detection.

II. Related Work

A. Malware Detection

Traditional detection methods based on signatures or behaviors, which highly rely on existing malware samples and have poor iteration speed, are gradually being replaced by machine learning-based detection methods.

Buriro et al. [4] proposed a lightweight portable executable (PE) malware detection and prevention method, MALWD&C, which can detect and classify new files into one of 14 malware families by learning from a large dataset of benign and malicious software programs. Among these, the classifier based on Random Forest (RF) shows the best performance in detection and classification scenarios. Selamat and Ali [13] used the PEview tool to inspect PE files and extracted 28 optimal features for malware classification. They designed a malware defense system based on multiple machine learning algorithms, where the classifier based on the Decision Tree (DT) algorithm achieved extremely high accuracy on smaller datasets. Chen and Ren [5] adopted the forward feature stepwise selection technique on the BIG2015 dataset to obtain a new efficient set of features fusion. The machine learning algorithms trained on this fusion set of features, including RF, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Extreme Gradient Boosting (XGBoost), all achieved excellent performance. Hussain et al. [10] implemented a malware detection system based on machine learning. The system extracts features from the header of PE files and deployed machine learning models such as RF, DT, AdaBoost, and Gaussian Naive Bayes (GNB) to combat malware. Among them, RF achieved a malware detection accuracy of 99.44%, outperforming other machine learning detectors.

It is evident that, due to effective feature extraction and generalization capabilities, classifiers based on Random Forest often perform well in malware detection tasks.

Furthermore, to reduce the time spent on feature engineering, capture more key features of samples, and enhance the generalization ability of detection systems, deep learning- based detection methods have been widely adopted.

Akhtar and Feng [1] constructed a deep neural network CNN-LSTM by stacking Convolutional Neural Networks (CNN) and Long Short-Term Memory networks (LSTM). Compared to DT and SVM, CNN-LSTM achieves higher true positive rates and accuracy in malware detection. Anand et al. [3] implemented a detection model, CNN-DMA, for defending against malicious attacks on electronic medical applications. Based on a CNN classifier and using dense layers, dropout layers, and flattening layers, it accurately detects Alueron.gen!J type malware in a dataset containing 25 families. Almaleh, Almushabb, and Ogran [2] proposed a new hybrid model. This model used functions called by malware as features and employed Recurrent Neural Networks (RNN) for malware classification. To address the challenges of model weight initialization techniques, they proposed using logistic regression to initialize weights for the neural network model. Fu, Ding, and Godfrey [14] built an LSTM model for mobile malware and then used GAN to generate enhanced examples. By retraining parts of the hidden and fully connected layers of the LSTM network with these enhanced examples, the model could distinguish newly merged malware. To capture the relationships between API calls, Li et al. [11] implemented a malware detection and classification method, DMalNet, based on Graph Neural Networks (GNN) combined with API feature engineering. DMalNet can achieve high accuracy in detecting and classifying malware.

B. Adversarial Attacks Against Malware Detection

Machine learning and deep learning-based malware detection methods have vulnerabilities in robustness, allowing malware creators to evade detection models by crafting adversarial malware.

Grosse et al. [7] used feedforward neural networks for adversarial attacks. Treating the target as a white box, they improved existing algorithms and modified the adversarial sample creation method proposed by Papernot et al., effectively increasing the misclassification rate of malware classifiers. Zhao et al. [15] proposed a gradient-based instruction replacement strategy, EFGSM, which effectively overcomes the drawback of adversarial samples being detected during preprocessing due to the removal of adversarial noise. Malware processed by EFGSM effectively misled the open-source detection framework Malconv and increased the rate of adversarial sample generation. He et al. [8] proposed a query- based attack framework. This framework uses a perturbation selection tree and perturbation adjustment policy, capable of generating adversarial Android malware that can deceive machine learning detection models using static program analysis under a zero-knowledge setting. Hu and Tan [9] also proposed an attack strategy using GAN to generate adversarial samples, MalGAN. They extracted system-level API calls from software and selected 160 of them as features. After training, the malicious software samples generated by MalGAN could effectively evade detection by various machine learning algorithms. However, as mentioned earlier, machine learning-based malware detection is gradually evolving towards deep learning architectures, and more detection models are opting to use neural networks to improve their generalization and detection capabilities. Therefore, it is particularly important to attack deep learning algorithm-based black-box detectors to verify the effectiveness of this attack strategy.

In this context, we used ResNet to build the black-box detector and constructed feature vectors for the dataset based on 128 system-level APIs. Our experimental results demonstrate that adversarial samples generated by GAN can also effectively evade detection by detectors equipped with deep learning algorithms.

III. Preliminary Knowledge

A. ResNet

Residual Networks (ResNets) are a class of deep neural networks in the field of deep learning, primarily used to address the problem of vanishing gradients. Compared to traditional neural networks, ResNets introduce a unique architectural feature known as residual blocks. These residual blocks incorporate skip connections that bypass one or more layers, enabling the training of deeper networks.

Mathematically, a residual block models the function F as $F(x) = f(x) + x$, where $f(x)$ is the learned residual function and x is the data input to the block. This formulation allows the network to learn an identity function, ensuring that higher layers can perform at least as well as lower layers, hence, preventing performance degradation with increased network depth.

Deep ResNets consist of multiple stacked residual blocks, each refining the feature representation progressively. Thus, the entire network can be formalized as a series of function compositions, similar to traditional neural networks, but with added skip connections:

$$F: \tilde{x} \to f_n\big(\cdots\big(f_2\big(f_1(\tilde{x}, \theta_1) + \tilde{x}, \theta_2\big)\big) + \tilde{x}, \theta_n\big) \qquad (1)$$

where each θ_i includes the parameters of the ith residual block and its shortcut connections. The entire set of parameters $\theta = \{\theta_i\}$ is optimized during training, typically using back-propagation with stochastic gradient descent or its variants. For example, in supervised learning scenarios, parameters are adjusted by minimizing a loss function calculated from the difference between predicted and actual outputs over pairs of training data $(\tilde{x}, \tilde{y})$.

B. GAN

Generative Adversarial Networks, introduced by Goodfellow et al., are a machine learning framework consisting of two neural networks: a generator G and a discriminator D. G and D are trained simultaneously in a competitive manner: G generates samples highly similar to real data, while D attempts to discern the authenticity of the generated data. Specifically, the generator G takes a random noise vector z and real data as input, generating new instances by capturing the distribution of real data. The discriminator D, on the other hand, is a classifier aimed at distinguishing between real data instances and synthetic data generated by G. The training process involves adjusting the parameters of G and D such that G gradually produces more realistic samples while improving D 's ability to detect generated instances.

The architecture of GANs can be described as a minimax game with a value function $V(G,D)$, where G aims to minimize the probability of D making correct authenticity classifications, while D tries to maximize it. The objective function for this process can be written as:

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}(x)}[\log(x)] + E_{x\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right] \qquad (2)$$

where x is a real instance from the data distribution p_{data}, and z is a noise instance from distribution p_z. The generator G is trained to produce outputs that the discriminator D will classify as real. Conversely, D is trained to better distinguish between real data and fake data produced by G. The training continues until the data generated by G cannot be distinguished by D as real, indicating that G has learned the distribution p_{data}.

IV. Black-Box Malware Detector

Given that malware detection algorithms based on machine learning or deep learning are typically integrated into antivirus software or hosted in the cloud, it's challenging for malware developers to know which classifier and classifier parameters the malware detection system uses. Therefore, we assume that the malware authors face a "black box", where the only information they might obtain is which features the black box detector uses as the basis for judgment. Based on this fact, I have introduced a black box detector into our system. This detector classifies input samples to determine if they are malicious, while maintaining the invisibility of its internal structure.

In our research, the black box detector is used to simulate a real attack scenario, ensuring that our trained attack model remains effective and applicable when facing a black box detector in real-world situations. At the same time, the black box detector provides a realistic benchmark for evaluating the effectiveness of adversarial samples generated by GAN. During the training of GAN, it labels the adversarial samples generated by the generator as either malicious or benign. This evaluation result is then used as a label for the training set to update the surrogate detector's weights and parameters, allowing the surrogate detector to better fit the judgment logic of the black box detector.

We chose to use Residual Neural Network to train the black box detector. This ResNet model is designed with multiple residual blocks, each including two convolutional layers followed by batch normalization and ReLU activation functions. These blocks are interconnected through skip connections, enabling the network to effectively learn complex feature representations.

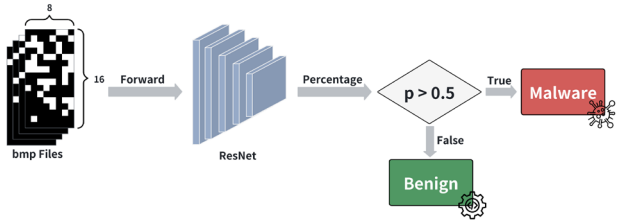

Figure 1. ResNet Architecture.

As shown in Figure 1, our ResNet model is designed to receive 16x8 grayscale images, with its first convolutional layer adjusted to suit this input size. The final stages of the network

include an adaptive average pooling layer to handle inputs of varying sizes, culminating in a fully connected layer and Sigmoid activation function to output the prediction probability.

During the prediction phase, the ResNet model outputs a probability value indicating the likelihood that the input sample is malicious. By setting a threshold (here, 0.5), we can convert this probability into a binary label, thus classifying the sample. If the probability is greater than the threshold, the sample is judged to be malicious; otherwise, it is considered benign.

V. GAN-Based Adversarial Attacks

We employed a GAN to train our attack model, with the goal of dynamically generating malware samples that can effectively evade detection by machine learning or deep learning-based malware detectors. The adversarial training mechanism of the GAN endows the model with the ability to evolve and adapt, enabling it to dynamically generate increasingly challenging malware samples for detection systems. Additionally, the GAN's detector can simulate the judgment logic of black box malware detectors, allowing it to generate adversarial samples capable of deceiving the black box detector, even without direct access to the actual malware detection systems. Its flexibility, scalability, and the neural network's generalization ability make it adaptable to different types and architectures of malware detection systems, while its powerful generative capability ensures the high quality and realism of the generated adversarial samples.

A. Substitute Detector

During the training of the GAN generator, it is imperative to update the generator parameters based on the discriminator's gradient information, facilitating the generation of more challenging adversarial samples. Given the difficulty in accessing the internal structure and parameters of real-world malware detectors, using a black-box detector as the GAN discriminator would impede gradient acquisition. Consequently, we employ a substitute detector to approximate the black-box detector and provide gradient information for generator updates.

Our substitute detector takes the feature vector x of a program as input, classifying the program and predicting the probability $D_{\theta d}(x)$ that x is malware.

Since the goal of the substitute detector is to approximate the judgment logic of the black box detector as closely as possible, the actual labels of the training set are not used for training the substitute detector. Instead, we first use the black box detector to make predictions on the training data, and then use the results as labels for training the substitute detector.

B. Generator

In our GAN model, the generator receives the malware feature vector m and a noise vector z as inputs. Here, m is a binary vector of dimension M, where different indices of the vector represent different API features. If $m[i]$ is 1, it indicates the usage of that API in the program sample; if $m[i]$ is 0, it indicates the non-usage of that API. The noise vector z is a Z-dimensional vector, with each element being a random number sampled from a uniform distribution within the [0, 1) range.

We set the number of neurons in the output layer of the generator to M and use the sigmoid function as the activation function. This allows the generator to produce a vector o of the same dimension as the feature vector m, with values between (0, 1). Subsequently, we binaryize the values of vector o to obtain a binary vector o'. Finally, the generator performs an OR operation between the malware feature vector m and the binary vector o' to output the adversarial example m'.

However, since m' is a binary vector obtained through m OR o', we cannot directly backpropagate gradients from the substitute detector to the generator. Therefore, we introduce $G_{\theta g}(m,z)$ [9] to provide a continuous, differentiable approximation that allows for effective weight updating in the generative model's neural network during training. We define:

$$G_{\theta_g}(m,z) = \max(m,o) \tag{3}$$

Specifically, $G_{\theta g}(m,z)$ is obtained by taking the maximum value for each pair of elements at the same index in m and o. In this way, even though our goal is to generate a binary vector m', we can use these continuous values to effectively update the network weights during the training process.

Furthermore, in the process of generating adversarial examples for binary malware features, we can consider adding some irrelevant features to the malware. However, it is advisable not to remove original features of the malware, as this could lead to the loss of key functionalities, rendering the malware benign. For instance, ransomware relying on the EncryptFile API to encrypt user files would be rendered non-functional if the EncryptFile API feature were removed from its feature set.

VI. Experiment

A. Set up

The dataset used in this paper originates from the research work of Hu and Tan [9]. They collected around 1.8 million programs from software sharing websites and utilized *Cuckoo Sandbox* to extract the APIs invoked by these software. Ultimately, they selected 160 API features as dimensional vectors for model training. Similarly, this study obtained data from 441 benign programs and 1,368 malicious programs, choosing 128 APIs as the data features.

Here, we demonstrate how to represent a program using API feature vectors with an example. Suppose we select M APIs as the features of a program; then a feature vector of M dimensions needs to be constructed for this program, including:

$$[API_1,\ API_2,\ \cdots,\ API_M]$$

If the selected program calls the dth API, then the feature value API_d is set to 1; otherwise, it is set to 0.

Before starting the experiments, the dataset was divided. We chose to share the same dataset between the black box detector and GAN. For the black box detector, 80% of the dataset was

used for training the model, and 20% for testing. Meanwhile, the dataset was processed into 16x8 grayscale images (i.e., converting .npy format data into .bmp format). This conversion adapts the one-dimensional feature vector for processing by convolutional neural networks. For GAN, 60% of the dataset was used as the training set, and the remaining data was evenly distributed between the validation set and the test set.

Adam was selected as the optimizer for the black-box detector, Generator, and Discriminator, with initial learning rates set to 0.001, 0.0001, and 0.00001, respectively. Additionally, binary cross-entropy was chosen as the loss function for the black-box detector. The loss function for the GAN's discriminator D was defined as:

$$L_D = -E_{x \in RN_{benign}} \log\left(1 - D_{\theta_d}(x)\right) - E_{x \in RN_{Malware}} \log D_{\theta_d}(x) \quad (4)$$

Here, $x \in RN_{Benign}$ refers to data identified as benign by the ResNet black-box detector, while $x \in RN_{Melware}$ denotes data identified as malware. The substitute detector D receives data from both the Malware and benign software datasets. When D incorrectly classifies data with benign labels as malicious, or vice versa, its loss function value increases. Similarly, the loss function for the generator G is defined as:

$$L_G = E_{m \in S_{Malware}, z \sim p_{uniform}[0,1)} \log D_{\theta_d}\left(G_{\theta_g}(m,z)\right) \quad (5)$$

Here, $m \in S_{Malware}$ represents real malware from the dataset, not those labeled by the black-box detector. The generator G produces adversarial examples $G_{\theta_g}(m,z)$. The loss value of G decreases when D incorrectly predicts a high probability of the adversarial sample being benign, and vice versa.

In GAN, the sizes of the input, hidden, and output layers of discriminator D are 128, 256, and 1, respectively. For the generator G, the length of the noise vector z in the input layer is set to 10, making the input layer size $M + Z = 180$. The sizes of the hidden and output layers are 256 and 128, respectively.

During the experiments, training the ResNet black-box detector for 18 epochs was deemed effective. The training epochs for GAN ranged from 90 to 110, seemingly adequate for achieving GAN convergence and favorable training outcomes.

B. Model Training

1) ResNet Black Box Detector: The training of the ResNet black-box detector primarily encompassed the following steps:

i) Data Processing: Initially, the malware and benign software datasets were transformed into 16x8 grayscale images.

ii) Initialization: The ResNet model and datasets were loaded, with the selection of a loss function and optimizer.

iii) Iterative Training: The model underwent training across multiple epochs, each comprising the steps below:

• Batch Processing: In each epoch, data were segmented into batches of size 64. For each batch, the following operations were performed:

a) Feeding images and labels into the model.

b) Calculating the loss between predicted outputs and actual labels.

c) Computing gradients via backpropagation.

d) Updating model weights using the Adam optimizer.

• Validation: Post every training epoch, the model was evaluated on an independent validation set with batch size 256. The system automatically saved the model state if higher accuracy was achieved in any training epoch.

iv) Model Saving: Upon the completion of training epochs, the model was automatically saved.

2) GAN: As shown in Figure 2, the training of the GAN primarily encompassed the following steps:

i) Initialization:

The generator model, discriminator model, and datasets were loaded, with the selection of loss functions and optimizers. A pre-trained black-box detector was imported.

ii) Iterative Training:

Repeat the following steps until the end of the training epochs:

a) Extract a small batch of samples M from the malware dataset.

b) Generate adversarial examples M' for the malware samples M using the generator.

c) Extract a small batch of samples B from the benign software dataset.

d) Employ the black-box detector to discriminate and label both M' and B.

e) Update the weights θ_d of the substitute detector based on the loss function L_D.

f) Update the weights θ_g of the generator based on the loss function L_G.

Conclude the training process when the training epochs are over and the model converges.

iii) Model Saving: After the completion of training, the model was automatically saved.

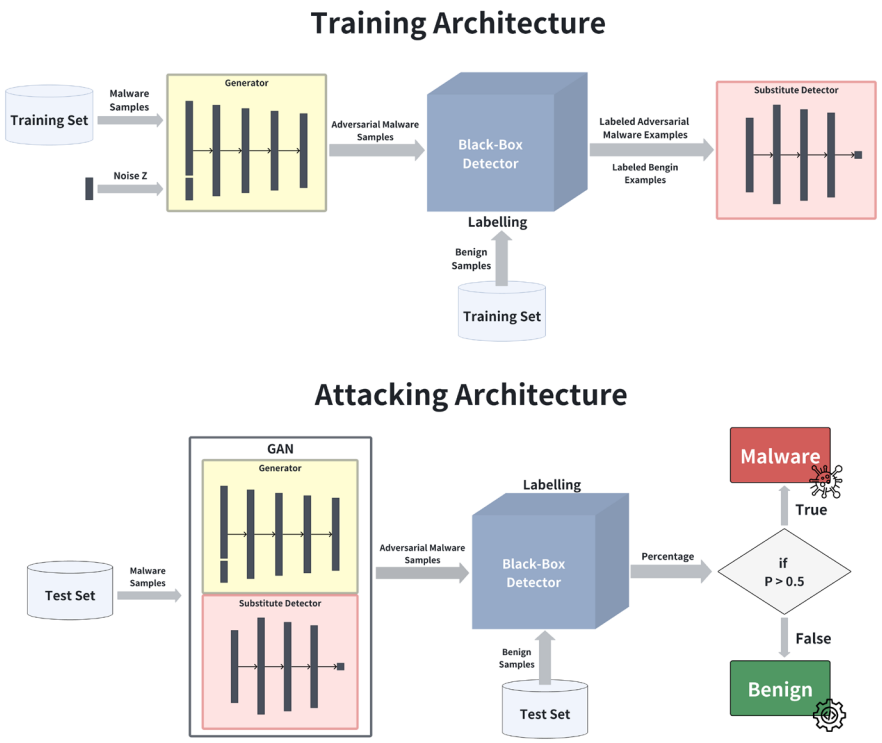

Figure 2. GAN Architecture.

C. *Results*

To better present the experimental results, we first introduce the concepts of confusion matrix, accuracy, and true positive rate (TPR). In the context of this study, the black box detector and GAN's discriminator may have the following discrimination results: correctly identifying malware, mistakenly identifying benign software as malicious, correctly identifying benign software, and mistakenly identifying malware as benign. These four discrimination results are respectively called true positive (TP), false positive (FP), true negative (TN), and false negative (FN) in the confusion matrix, that is shown in Table 1.

TABLE I. CONFUSION MATRIX

	Predicted as Benign	Predicted as Malicious
Actual Benign	TN	FP
Actual Malicious	FN	TP

With the help of the confusion matrix, the accuracy of the malware detection model can be defined as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

At the same time, the true positive rate (TPR) can be defined as:

$$TPR = \frac{TP}{TP+FN} \tag{7}$$

The level of accuracy directly reflects the overall performance of the detection model in processing all samples. High accuracy means that the model excels in identifying both malicious and benign software, correctly distinguishing the vast majority of samples. The true positive rate is another key indicator, used to measure the proportion of malware correctly identified by the malware detection model out of all actual malware. A high TPR means that the detection model has strong recognition capabilities, accurately identifying most malware and thus providing effective security protection for users.

1) Black Box Detector: After completing the training of the black box detector and the GAN, we first assessed the performance of the ResNet black box detector using the test set. As shown in Figure 3 under *Original ResNet*, the black box detector we trained achieved an accuracy of 97.51% and a true positive rate of 98.53% in identifying malware. This indicates that it can effectively distinguish between malicious and benign software.

Subsequently, we also implemented black box detectors using various machine learning algorithms, and their performance is also presented in Figure 3. Overall, whether deploying ResNet or other machine learning algorithms, the black box detectors all maintained an accuracy of no less than 92.5% and a true positive rate of at least 90.8%. This ensures that the black box detectors can accurately identify common malware, to a certain extent replicating the malware detection systems in real-world scenarios.

2) GAN: After testing the performance of the black box detector, we assumed the role of malware developers, treating the aforementioned black box detector as the target of our attack, and first chose to attack the detector deploying ResNet. We utilized GAN to convert the malicious software samples in the test set into adversarial samples, and then merged these adversarial samples with benign samples from the test set to

form a new test set. This process is illustrated in Figure 2. Finally, we used the black box detector to predict on this new test set again, with its performance shown in Figure 3 under *Against ResNet*. It is evident that the black box detector deploying ResNet could only achieve an accuracy of 30.1% and a true positive rate of 9.5% in distinguishing adversarial samples generated by GAN. After multiple tests, adversarial samples generated by GAN managed to reduce the true positive rate of the black box detector to between 50.5% and 5.9%, indicating that malware processed by GAN is more difficult for the black box detector to recognize.

Subsequently, we adopted the same strategy to attack the black box detectors deployed with machine learning algorithms, and their confusion matrices are also displayed in the above figure. By comparing the accuracy and true positive rates of the black box detectors before and after the attack, we obtained the following Table 2.

It is evident that, except for Random Forest, the TPRs of other machine learning algorithms were reduced to below 23%. The TPR of Random Forest was only reduced to around 60%, which might be due to the GAN's substitute detector using a feedforward neural network, while the structure of Random Forest differs significantly from neural networks. This difference led to the substitute detector not being able to effectively mimic the recognition logic of the black box detector deployed with Random Forest. Nevertheless, the TPR of Random Forest was still reduced by about 40%, indicating that the substitute detector could still use neural networks to some extent to simulate black box detectors with non-neural network structures.

TABLE II. PERFORMANCE COMPARISON OF BLACK-BOX DETECTORS.

Black-Box Detector	TPR		Accuracy	
	Original	Adversarial	Original	Adversarial
ResNet	94.87 ± 4.03	28.21 ± 22.34	95.15 ± 2.91	43.35 ± 17.87
Decision Tree	96.34 ± 0.73	0.073 ± 0.07	94.74 ± 0.83	22.16 ± 0.83
Logistic Regression	96.34 ± 0.37	20.15 ± 11.36	94.32 ± 0.69	36.29 ± 8.59
MultiLayer Perceptron	97.80 ± 0.73	14.10 ± 8.61	96.26 ± 0.69	32.69 ± 6.65
Random Forest	99.45 ± 0.18	59.70 ± 7.33	97.23 ± 0.55	66.48 ± 5.54
SVM	97.07 ± 0.01	18.68 ± 4.03	93.63 ± 0.01	34.35 ± 3.05

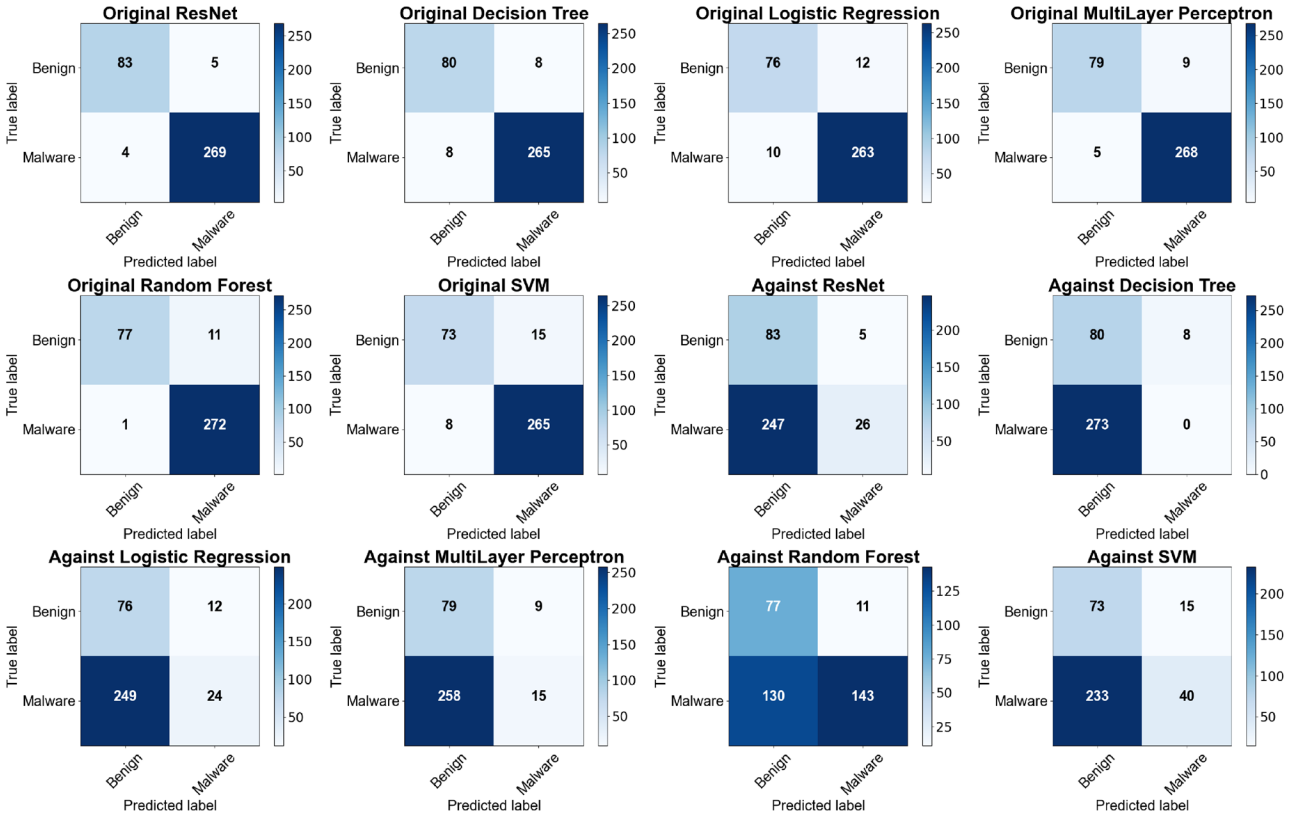

Figure 3. Confusion Matrixes.

VII. Conclusion

This paper first implemented a detector based on ResNet, capable of effectively distinguishing between malicious and benign software. Subsequently, we employed GAN to develop an attack strategy capable of dynamically generating adversarial samples. Future work could focus on constructing larger datasets and re-attacking retrained black box detectors.

References

[1] Muhammad Shoaib Akhtar and Tao Feng. Detection of malware by deep learning as cnn-lstm machine learning techniques in real time. Symmetry, 14(11), 2022.

[2] Abdulaziz Almaleh, Reem Almushabb, and Rahaf Ogran. Malware api calls detection using hybrid logistic regression and rnn model. Applied Sciences, 13(9), 2023.

[3] Ankita Anand, Shalli Rani, Divya Anand, Hani Moaiteq Aljahdali, and Dermot Kerr. An efficient cnn-based deep learning model to detect malware attacks (cnn-dma) in 5g-iot healthcare applications. Sensors, 21(19), 2021.

[4] Attaullah Buriro, Abdul Baseer Buriro, Tahir Ahmad, Saifullah Buriro, and Subhan Ullah. Malwd&c: A quick and accurate machine learning-based approach for malware detection and categorization. Applied Sciences, 13(4), 2023.

[5] Zhiguo Chen and Xuanyu Ren. An efficient boosting-based windows malware family classification system using multi-features fusion. Applied Sciences, 13(6), 2023.

[6] Amir Djenna, Ahmed Bouridane, Saddaf Rubab, and Ibrahim Moussa Marou. Artificial intelligence-based malware detection, analysis, and mitigation. Symmetry, 15(3), 2023.

[7] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick Mcdaniel. Adversarial perturbations against deep neural networks for malware classification. ArXiv, abs/1606.04435, 2016.

[8] Ping He, Yifan Xia, Xuhong Zhang, and Shouling Ji. Efficient querybased attack against ml-based android malware detection under zero knowledge setting. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS '23, page 90–104, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. arXiv e-prints, page arXiv:1702.05983, February 2017.

[10] Abrar Hussain, Muhammad Asif, Maaz Bin Ahmad, Toqeer Mahmood, and M. Arslan Raza. Malware detection using machine learning algorithms for windows platform. In Abrar Ullah, Sajid Anwar, A´ lvaro Rocha, and Steve Gill, editors, Proceedings of International Conference on Information Technology and Applications, pages 619–632, Singapore, 2022. Springer Nature Singapore.

[11] Ce Li, Zijun Cheng, He Zhu, Leiqi Wang, Qiujian Lv, Yan Wang, Ning Li, and Degang Sun. Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning. Computers & Security, 122:102872, 2022.

[12] Xiangjun Li, Ke Kong, Su Xu, Pengtao Qin, and Daojing He. Feature selection-based android malware adversarial sample generation and detection method. IET Information Security, 15(6):401–416, 2021.

[13] Nur Selamat and Fakariah Ali. Comparison of malware detection techniques using machine learning algorithm. Indonesian Journal of Electrical Engineering and Computer Science, 16:435, 10 2019.

[14] Musaazi Godfrey Zhangjie Fu, Yongjie Ding. An lstm-based malware detection using transfer learning. Journal of Cyber Security, 3(1):11–28, 2021.

[15] Jiapeng Zhao, Zhongjin Liu, Xiaoling Zhang, Jintao Huang, Zhiqiang Shi, Shichao Lv, Hong Li, and Limin Sun. Gradient-based adversarial attacks against malware detection by instruction replacement. In Lei Wang, Michael Segal, Jenhui Chen, and Tie Qiu, editors, Wireless Algorithms, Systems, and Applications, pages 603–612, Cham, 2022. Springer Nature Switzerland