


Article

# Adversarial Attacks with Defense Mechanisms on Convolutional Neural Networks and Recurrent Neural Networks for Malware Classification

Sharoug Alzaidy and Hamad Binsalleeh \* 

Department of Computer Science, College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 13318, Saudi Arabia; saalzaidi@sm.imamu.edu.sa

\* Correspondence: hmbinsalleeh@imamu.edu.sa

**Abstract:** In the field of behavioral detection, deep learning has been extensively utilized. For example, deep learning models have been utilized to detect and classify malware. Deep learning, however, has vulnerabilities that can be exploited with crafted inputs, resulting in malicious files being misclassified. Cyber-Physical Systems (CPS) may be compromised by malicious files, which can have catastrophic consequences. This paper presents a method for classifying Windows portable executables (PEs) using Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). To generate malware executable adversarial examples of PE, we conduct two white-box attacks, Jacobian-based Saliency Map Attack (JSMA) and Carlini and Wagner attack (C&W). An adversarial payload was injected into the DOS header, and a section was added to the file to preserve the PE functionality. The attacks successfully evaded the CNN model with a 91% evasion rate, whereas the RNN model evaded attacks at an 84.6% rate. Two defense mechanisms based on distillation and training techniques are examined in this study for overcoming adversarial example challenges. Distillation and training against JSMA resulted in the highest reductions in the evasion rates of 48.1% and 41.49%, respectively. Distillation and training against C&W resulted in the highest decrease in evasion rates, at 48.1% and 49.9%, respectively.

**Keywords:** adversarial example; artificial intelligence; malware analysis; security; cyber-physical systems



**Citation:** Alzaidy, S.; Binsalleeh, H. Adversarial Attacks with Defense Mechanisms on Convolutional Neural Networks and Recurrent Neural Networks for Malware Classification. *Appl. Sci.* **2024**, *14*, 1673. <https://doi.org/10.3390/app14041673>

Academic Editor: Andrea Prati

Received: 15 November 2023

Revised: 9 December 2023

Accepted: 17 February 2024

Published: 19 February 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

As Internet technologies have grown and improved, malicious activities and threats have also increased, and consequently, defensive approaches are facing increasing challenges. One of the advantages of deep learning (DL) approaches is the ability to automatically extract features from raw data such as malware binary files, which helps in building byte-based malware classification models. Despite the success of DL in several fields, recent studies have proven that adversarial examples are effective against DL models [1–6]. For instance, by using X-Adv, Liu et al. created physically printable metals capable of deceiving X-ray detectors when placed inside luggage [7] and Li et al. proposed a novel structured-light attack against structured-light-based 3D face recognition [8]. Moreover, when adversarial examples are attacked, this can also compromise the stability and reliability of cyber-physical systems (CPS), causing malfunctions or failures in critical systems or operations. Furthermore, according to [9], small perturbations added to any waveform can attack voice recognition systems. In the literature, many approaches have been developed to generate adversarial examples ( $x^*$ ), such as fast gradient sign method (FGSM), L-BFGS, Carlini and Wagner (C&W), and Jacobian-based saliency map attack (JSMA). These studies open the door to investigating the effectiveness of adversarial attacks against malware classifiers by considering domain-specific characteristics in the malware domain. For instance, binary adversarial examples were generated for Android [10], and portable executable (PE) [11–14]. Because of the semantic structure of PE and the independence

of bytes introducing another factor in generating crafted samples, few empirical studies have successfully generated executable PE adversarial examples as shown in Table 1. Most adversarial attacks against malware detection systems have been exclusively carried out through continuous features such as pixels in malware images. Meanwhile, perturbing different types of features (e.g., API, string, URL) in cybersecurity has varying difficulty levels. For instance, perturbing pixels have the same difficulty level, as the binary itself is not modified, only its intensity. These attacks on malware images cannot preserve malware functionality. Therefore, attackers need effective adversarial attack techniques to perturb malware samples while preserving the PE binary file functionalities. To perturb malware files, an attacker must apply practical manipulations on the input  $x$  without breaking the binary files' semantics. We denoted practical manipulations by a  $h : Z \times \mathcal{T} \rightarrow X$ , which is the output, which is an executable program without losing its malicious functionality. Here,  $x \in X$  is an input program and a vector  $\tau \in \mathcal{T}$ , indicating the function  $h$ 's parameters. As a result, we determine how likely the adversarial example  $x^*$  is classified to the target label  $y^*$  from a labeled space  $Y = \{0, 1, 2, \dots, n\}$  where  $Z \subseteq \mathbb{R}^d$  represents the feature space. The function  $f : Z \rightarrow \mathbb{R}$  is used for prediction. This function provides a probability of the input sample being classified as malware family  $y^*$ . However, previous studies have primarily concentrated on the FGSM approach, although FGSM is a random starter attack that modifies a large portion of input features [11,12,15]. To design robust models, several practical countermeasures have been proposed, including training [1], distillation [16], and feature squeezing [17]. However, to date, none of these countermeasures can be treated as a one-stop solution. Due to the nonlinear nature of DL models, it is difficult to prove these countermeasures theoretically. Moreover, these proposed countermeasures are not adaptive to all types of attacks that use adversarial examples ( $x^*$ ). Each mechanism has shown various impacts on different attack types. For example, most attack types are fooled by distillation mechanisms, with the exception of the C&W approach.

**Table 1.** Summary of adversarial examples articles.

Article	Model						Dataset						Classification Types		Features Type					Attack Types				Evasion Rate		Norm Distance	Preserving Functionality	Defense Mechanisms									
	DNN	MalConv (CNN)	CNN	SVM	Boosted Trees	InceptionV3	Random Forest	EMBER	Drebin	VirusTotal	VirusShare	Microsoft	Maling	Others	Binary-Class	Multi-Class	API Calls	Binaries	Grayscale	Other	JSM	C&W	FGSM	Others	Before Defending			After Defending	Training	Distillation	Ensemble	Feature Squeezing	Dimensionality Reduction	Weight Decay	Random Feature Nullification		
[18]	✓													✓			✓			✓																	
[11]		✓							✓					✓	✓		✓								71%												No
[12]		✓									✓			✓	✓		✓								99.21%		$L_2, L_\infty$										Yes, without verification
[13]		✓															✓							60%												No	
[14]		✓						✓						✓	✓										86.6%											Yes, without verification	
[17]	✓								✓					✓	✓		✓										$L_2$									No	
[10]	✓			✓				✓	✓					✓	✓		✓			✓	✓				96%, 83%											Yes	
[19]	✓							✓						✓	✓					✓	✓			69%	67%, 38.5%	$L_1$										No	
[15]	✓							✓						✓	✓					✓	✓			100%	27%, 100%	$L_0$									No		
[20]			✓	✓										✓										100%		$L_0, L_2, L_\infty$									No		
[21]			✓											✓	✓									99.45%	26.12%	$L_2$									No		
[22]			✓		✓	✓								✓										50%		$L_0, L_2$									No		
[23]		✓	✓							✓				✓	✓										91.6%										Yes, without verification		
[24]	✓								✓					✓	✓		✓								100%											Yes	

The FGSM approach is one of the most widely used against malware detection systems. It has been extensively used to evade malware detectors for PE files. Ref. [11] proposed different append techniques, namely random append, gradient append, benign append, and FGSM append. In addition, they proposed an FGSM slack-based attack. Their attacks initialized the adversarial examples using random noises and then perturbed them using FGSM. They observed that these one-time examples were not transferable between models. Furthermore, FGSM attack required a large number of bytes to be modified. Ref. [12]

proposed an injection FGSM attack using a discrete input set to evade a neural network. Ref. [13] investigated the neural network's vulnerability against a gradient-based perturbation approach and proposed an append-based attack. Ref. [14] applied the attack presented in [13] by only perturbing the bytes inside the DOS header. However, Ref. [14] attack can be easily recognized by a human expert. Ref. [17] presented white-box and gray-box evasion attacks, perturbing 419 API features using JSMA with an  $L_2$  distance norm. Ref. [20] showed and analyzed the vulnerability of convolutional neural network (CNN), support vector machine (SVM), and random forest (RF) by C&W. To evaluate these detectors, the Microsoft BIG dataset was converted into binary texture grayscale images. The evasion rate of C&W was 100% against all presented ML-based malware detectors. In addition, the highest transferability rate for CNN reached up to 88.5%. This attack was limited by the lack of verified file executability.

To overcome this limitation, the COPYCAT framework is presented in [21]. C&W is one of six crafting approaches used by the COPYCAT to generate executable adversarial malware samples. To ensure the executability of the adversarial examples, they applied the adversarial examples padding method, with the adversarial examples appended at the end of the original sample image. These researchers targeted Windows and Internet of Things (IoT) DL-based malware detectors. The evasion rate for C&W was 99.45%, with the minimum number of modified pixels being 4.09. After applying the padding, an overall misclassification rate of 73.5% was achieved. In 2023, fast-generation adversarial malware (FGAM) was proposed, introducing a method for quickly generating adversarial malware [23]. According to the gradient sign, this iteratively perturbs bytes, resulting in the enhanced adversarial capability of the perturbed bytes. In [24], a novel black-box AE attack towards the FCG-based malware detection system is presented, called BagAmmo. BagAmmo adopts the architecture of generative adversarial network (GAN). BagAmmo achieves an average attack success rate of over 99.9% on MaMaDroid, APIGraph, and GCN. The attack methodology involves the use of generative models to generate adversarial examples that successfully evade detection by BagAmmo.

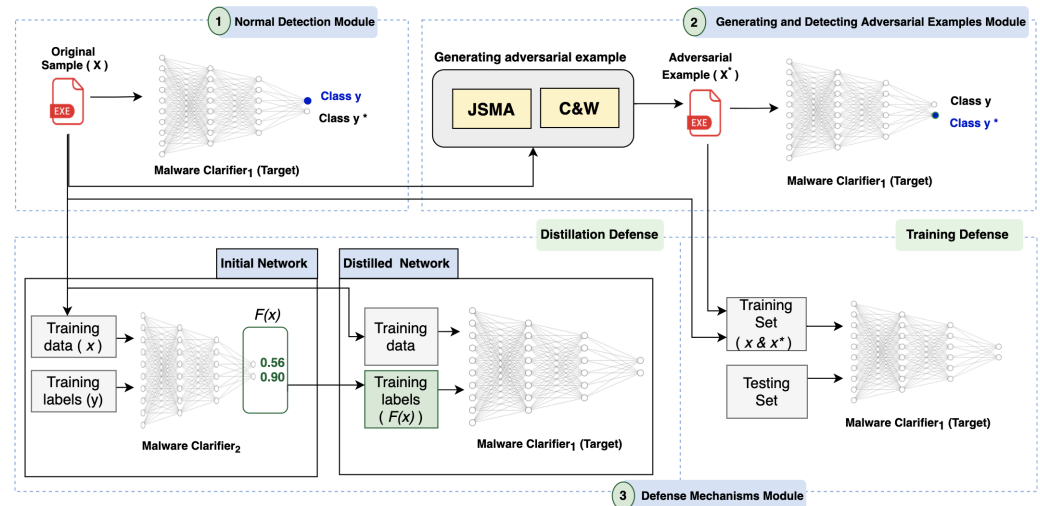
In this work, we contribute to the current literature via the following contributions:

- **Investigate the vulnerabilities of DL-based malware classification using two approaches to crafting adversarial examples: C&W and JSMA.** Previous JSMA works used API calls as input features to indicate how much each input feature's perturbations would affect the output. On the other hand, although previous works on C&W investigated the impact of the approach on continuous data, such as image representations of the malware binaries, we investigate its impact on discrete input, that is, raw bytes of the PE file. To our knowledge, this is the first work on C&W that uses discrete data input into the malware classification domain.
- **Maintain malware while producing and crafting adversarial examples using C&W and JSMA.** Previous studies mainly focused on malware adversarial images and differentiable domains, which differ from PE adversarial. This paper presents a practical functionality-preserving approach to the PE file format based on injecting the adversarial payload into the DOS header and adding a section of perturbation.
- **Evaluate two defense mechanisms for adversarial examples attacks.** We investigate the impact of distillation and training defense mechanisms against nonlinear models and our generated adversarial examples by JSMA and C&W approaches.
- **Propose a practical method for preventing malware detection classifiers from being evaded.** Our approach effectively increases the complexity for hackers attempting to bypass malware classifiers. This addresses all methods of producing adversarial examples in the high-security malware industry.

The remainder of this paper is organized as follows. Section 2 presents our methodology. Section 3 provides the results. Section 4 discuss the results. Finally, Section 5 concludes this paper.

## 2. Materials and Methods

In this section, we discuss the methodology for attacking PE malware detectors using the JSMA and C&W approaches. In addition, the effects of implementing current defense mechanisms are discussed. To illustrate our methodology, we have divided it into the following three consecutive modules: normal detection (module 1), generating and detecting adversarial examples (module 2), and defense mechanisms (module 3). Figure 1 illustrates the methodology of the three modules. Module 1 is described in Section 2.2; module 2 in Section 2.3; and module 3 in Section 2.4, while the dataset is described in Section 2.1.



**Figure 1.** General structure of adversarial attacks. As shown, adversarial attacks consist of three modules. Module 1 is the malware detectors; and module 2 generates adversarial samples. Module 3 is the defense mechanisms.

### 2.1. Dataset

The Endgame Malware Benchmark for Research (EMBER) [25], one of the most widely used open source datasets in cybersecurity research, was used to evaluate our proposed attacks. To collect raw malware files, we utilized the VirusShare platform (<https://virusshare.com/> (accessed on 1 July 2020)). VirusShare is a public repository of malware samples through which security researchers, incident responders, forensic analysts, and the morbidly curious gain access to samples of malicious files. The EMBER dataset labels samples as malware, benign, or unlabeled. Since our paper is based on multiclassification, we focused on malware and unlabeled samples. Table 2 shows the distribution of our collected dataset.

**Table 2.** The distribution of the collected dataset divided into training and testing sets.

Set	The Collected Dataset			
	Before Preprocessing	After Removing Duplicates	After Removing Packed Files	After Labeling
Training	81,820	81,584	77,952	76,985
Testing	13,271	13,244	13,006	12,943

We were able to label malware samples with a family name using the widely used AVClass labeling tool [26]. A total of 16,643 samples were labeled successfully.

### 2.2. Malware Classification

Our adversarial approaches were compared with two different neural networks: CNN and a recurrent neural network (RNN). In this module, we only trained the networks with clean/original input ( $x$ ).

### 2.2.1. CNN

The CNN model consists of an embedding layer, two 1D convolution layers, and temporal max pooling followed by a fully connected layer, as shown in Figure 2. In our experiment, the MalConv model presented in [27] was used.

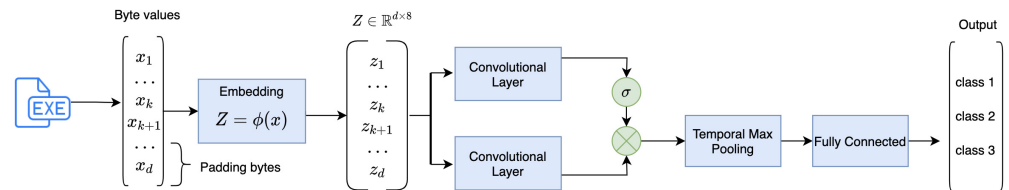


Figure 2. Architecture of the CNN malware classifier.

### 2.2.2. RNN

In [28], the authors contributed to the field of malware detection by classifying a part of the header’s components. We expanded the features’ space beyond the header and evaluated the attack’s performance against an RNN classifier. As shown in Figure 3, the RNN model architecture consisted of an embedding layer, three last long short-term memory (LSTM) layers, and an attention mechanism, the latter focusing on a part of the sequence (the relevant features) and ignoring the rest. For example, in face detection applications, the attention mechanism focuses on human features rather than other details.

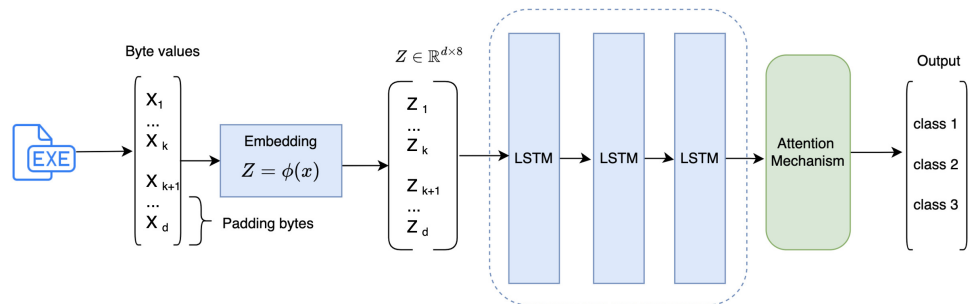


Figure 3. The RNN architecture. The embedding layer and blue squares indicate LSTM layers, and the activations of all time steps are merged into the attention mechanism.

#### Attention Mechanism

The attention mechanism in RNNs is a common technique that enables the network to generate output by focusing on specific parts of the input sequence [29], significantly improving performance. The attention mechanism is particularly useful for long input sequences or when the relevant information for the output is not always located at the same place in the input. The attention mechanism used in our LSTMs was derived from the approaches used by [28].

Attention mechanisms assign a weight to each input sequence element based on its relevance to the current output. A weighted sum is then computed from these input elements, generating the output. The output of the attention mechanism is a weighted average of the hidden states, as shown in (1).

$$\text{Attention output} = \sum_{i=1}^T \alpha_i h_i \tag{1}$$

At step  $i$ ,  $h_i$  represents the hidden activation of the LSTM layer. A vector  $\bar{h}$  provides information about the entire sequence, while each  $h_i$  provides information about one step.

As shown in Equation (2), these are combined in a small one-layer network (that is, a part of the larger RNN).

$$\alpha_i = v^T \tanh(W_0^T h_i + W_1^T \bar{h} + b) \quad (2)$$

Matrix  $W_0$  and matrix  $W_1$  represent the activations at local and global hidden states for the attention mechanism's hidden layer. The softmax function is applied as the final step to sum the weighted importance over each time step, as shown in Equation (3).

$$\alpha_i = \frac{\exp(\tilde{\alpha}_i)}{\sum_{j=1}^T \exp(\tilde{\alpha}_j)} \quad (3)$$

### 2.3. Adversarial Attacks

In this section, we discuss *module 2*, which, as illustrated in Figure 1, is responsible for constructing an adversarial example ( $x^*$ ) that evades our CNNs and RNNs. The goal of crafting an adversarial approach is to find a perturbation ( $\delta$ ) for the original sample ( $x$ ) with the label ( $y$ ) and classify it with a target label ( $y^*$ ), where  $y \neq y^*$ . To generate adversarial examples, we use two approaches, namely JSMA and C&W. Adversarial examples crafted by these approaches are fed into the CNN and RNN. The following describes the crafting approaches used to attack malware detectors.

#### 2.3.1. Jacobian-Based Saliency Map Attack (JSMA)

Ref. [3] presented a targeted JSMA that utilizes the impact of forward derivatives to craft adversarial examples with minimally perturbed features. Their attack highlighted the benefit of directly computing the mapping between input and output. The number of perturbed features can drive extreme variations in the neural network's output. The neural network trains this forward derivative, called the Jacobian matrix of function  $F$ . The Jacobian matrix used to construct the saliency map of the most important features should be included in the perturbation to evade the neural network. In this approach, no knowledge of training data is needed; only a known network architecture is required. The approach scenario consisted of three main steps.

- (a) **Computation of the forward derivative  $\nabla F(x^*)$  for the original sample  $x$** , as given by:

$$\nabla F(X) = \frac{\partial F(X)}{\partial X} = \left[ \frac{\partial F_j(X)}{\partial x_i} \right]_{i \in 1 \dots M, j \in 1 \dots N} \quad (4)$$

where  $\frac{\partial F_j(X)}{\partial x_i}$  is obtained by:

$$\frac{\partial F_j(X)}{\partial x_i} = \left( W_{n+1,j} \cdot \frac{\partial H_n}{\partial x_i} \right) \times \frac{\partial f_{n+1,j}}{\partial x_i} (W_{n+1,j} \cdot H_n + b_{n+1,j}) \quad (5)$$

where  $W$  is the weights vector,  $H$  is the output vector of the hidden layers, and  $b$  is the bias.

- (b) **Construction of the saliency map.** We used an increasing saliency map, that is, as the map increases, so does the probability of the target label ( $y^*$ ), while the probabilities of other labels decrease.
- (c) **Perturbation of the most salient pixels/features with  $\theta$ .** The original paper used  $\theta = 1$ , whereas we set  $\theta = 0.7$ .

These steps were repeated until we reached the maximum  $Y$  or successfully misled the model. The JSMA approach is described in Algorithm 1.



---

**Algorithm 1** Crafting the adversarial example for malware detection with the JSMA approach

---

```

1: Input:  $x, y^*, F, \theta, Y, x \leftarrow x^*$ 
2:  $\Gamma = 1 \dots |x|$ 
3: while  $F(x^*) \neq y$  and  $\|\delta_x \leq Y\|$  do
4:   Compute forward derivative  $\nabla F(x^*)$ 
5:    $S = \text{Saliency Map}(\nabla F(x^*), y^*, \Gamma)$ 
6:   Modify  $x_{i_{max}}^*$  by  $\theta$  s.t  $i_{max} = \text{argmax}_i S(x, y^*)[i]$ 
7:    $\delta_x \leftarrow x^* - x$ 
8: end while
9: return  $x^*$ 

```

---

### 2.3.2. Carlini and Wagner (C&W)

The C&W approach is one of the most widely used in generating adversarial examples and has been extensively used in computer vision studies. The C&W approach’s goal is to generate the adversarial example ( $x^*$ ) by optimizing the objective function for the distance between the target class ( $y$ ) and the most likely class ( $y^*$ ). In [5], seven objective functions were presented; the objective function  $f_6$  outperformed all functions. Thus, they selected  $f_6$  to perform their approach, as follows:

$$f_6(x^*) = (\max_{i \neq y^*} (Z(x^*)_i) - Z(x^*)_{y^*})^+ \tag{6}$$

The original C&W paper proposed three norm-based attacks, whereas we focused on an  $l_2$ -based attack. To ensure that C&W generated a valid adversarial example, the  $\delta$  was constrained by the *box constraints*:  $0 \leq x_i + \delta_i \leq 1$  for all  $i$ . In C&W, different methods for *box constraints* were evaluated, and it was found that the best method was that of the change of variables. They added a new variable  $w$  and then applied a change of variables to optimize over  $w$  instead of ( $\delta$ ), as follows:

$$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i \tag{7}$$

where  $-1 \leq \tanh(w_i) \leq 1$ . It follows that  $0 \leq x_i + \delta_i \leq 1$ .

Thus, the C&W formulation using the  $l_2$  distance metric can thus be defined:

$$\begin{aligned} &\text{minimize } \left\| \frac{1}{2}(\tanh(w) + 1) - x \right\|_2^2 + c.f(\frac{1}{2}(\tanh(w) + 1)) \\ &\text{where } f(x^*) = \max(\max\{Z(x^*)_i : i \neq y^*\} - Z(x^*)_{y^*}, -k) \end{aligned} \tag{8}$$

where  $Z(x^*)$  represents the *logits*, which are the outputs of the layers-unnormalized probability predictions for each class; except for the softmax layer, parameter  $k$  sets a lower limit for the loss value to generate an adversarial example with confidence that the adversarial example is correctly classified to the target class ( $y^*$ ). The highest  $k$  value generates a high-confidence adversarial example but is far from the original input. We used  $k = 0$  to conduct our experiment, as described in the original paper [5]. The C&W approach is described in Algorithm 2.

### 2.3.3. Preserving Functionality

Many studies have focused on generating adversarial examples without preserving the malicious files and modifying the PE files [10,15,17,19–22]. To move forward with the practical approach, we clarify the attack formalization below.

**Algorithm 2** Crafting adversarial examples for malware detection with the C&W approach

---

```

1: Input:  $x, y, F(\cdot), \varepsilon$ 
2:  $x \leftarrow x^*$ 
3:  $\delta = \frac{1}{2}(\tanh(w) + 1) - x$ 
4: while  $F(x^*) \neq y$  and  $\operatorname{argmin}(D(x, x + \delta))$  do
5:    $\min_w \|\frac{1}{2}(\tanh(w) + 1)\|_2^2 + c.f(\frac{1}{2}(\tanh(w) + 1))$ 
6:    $f(x^*) = \max(\max_{\delta}(Z(x^*)_i) - Z(x_{y^*}, -k),$ 
7:   if  $w_{\max} \leq 0$  then
8:     return Failure
9:   end if
10:   $\delta = \varepsilon \cdot \frac{1}{2}(\tanh(w) + 1) - x$ 
11:   $x^* \leftarrow x + \delta$ 
12: end while
13: return  $x^*$ 

```

---

A model attempts to classify input  $x$  into one of the dataset labels. The inputs are then passed through a feature extractor, where the attack is carried out. The feature extractor is denoted by the function  $\phi : X \rightarrow Z$ , where  $Z$  is a feature space denoted by  $Z \subseteq \mathbb{R}^d$ . This requires the manipulation of the PE without compromising its functionality. In this paper, we denote these manipulations as  $h : Z \times \mathcal{T} \rightarrow X$ , which causes a return to the functioning program with a different representation. Due to the non-differentiable embedding layer of our models, the attack is applied inside the feature space. Our approach for generating executable adversarial examples consists of three steps.

1. **Determining the indexes of allowed bytes that can be freely modified without affecting the PE functionality.** These indexes must be cautiously determined due to the byte dependencies and the PE semantics. To achieve this, we identified two areas for modification:
  - (a) The padding bytes of the embedding space are perturbed, and then the perturbation is injected into a PE zone that is not used by the program and has no effect on its functionality. A wide variety of features in the DOS header can be modified without affecting the PE file. A modification to the DOS header requires that neither the MZ magic number nor the value at offset  $0x3c$  is modified. Altering these two fields in the DOS header results in inexecutable programs.
  - (b) The padding bytes in the embedding space are perturbed, and the perturbation is appended to the end of the PE file. The perturbation is injected into a nonexecutable section of the program, generating an executable program.
2. **Performing the C&W and JSMA attacks on the list of indexed bytes generated from the previous step.** The attacks are performed inside the feature space  $Z \subseteq \mathbb{R}^d$ . We refer to our attacks as  $JSMA_{Header}$ ,  $JSMA_{Section}$ ,  $C\&W_{Header}$ , and  $C\&W_{Section}$  in this paper.
3. **Mapping back the modified values to the byte value.**

To verify the functionality of the adversarial example, we used VirusTotal sandboxes (<https://www.virustotal.com/> (accessed on 1 January 2023)). VirusTotal was used to analyze the original ( $x$ ) and adversarial malware samples ( $x^*$ ) and compare their detection scores. Moreover, we verified the executability of the adversarial samples by successfully running them with VirusTotal.

#### 2.4. Defense Mechanism

There are two defense mechanisms discussed in this paper: distillation and training. The effectiveness of these two mechanisms was evaluated by comparing the evasion rates before and after their application.



### 2.4.1. Training

This paper analyzed the training defense mechanism by adding the adversarial examples generated in *module 2* to the training set. We then trained the networks with the original/clean input ( $x$ ) and the adversarial example ( $x^*$ ). The mechanism consisted of the three main steps as follows:

1. Training the model  $F$  on the clean dataset  $D$ .
2. Generating adversarial examples ( $x^*$ ) using the JSMA and C&W approaches presented in *module 2*.
3. Iterating the training epochs on our model  $F$  with the generated adversarial examples ( $x^*$ ) from the previous step.

The number of adversarial examples that should be included in training defense is currently an important discussion topic [15,30]. We added 400 adversarial examples to the training set. Ten samples were selected at random from 20 classes for each attack.

### 2.4.2. Distillation

The existing body of research on the distillation defense has shown that it can increase a neural network's resilience to adversarial examples [15,17–19]. However, these results were based on the JSMA approach, and whether this mechanism has the same impact on the C&W approach is unclear. As part of this paper, we demonstrate the effectiveness of distillation when used against adversarial examples in malware detection. The distillation defense scenario consisted of the following steps:

1. Building an initial network for each model (CNN and RNN); these are the networks presented in *module 1*.
2. Training initial networks  $F$  with a clean dataset  $D$ .
3. Producing the probability vector using a  $T$  value of 10. The probability vector prediction is given as

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (9)$$

4. Using the probability vector computed in step 3 as soft training labels for the new dataset  $D'$ .
5. Training the distilled networks  $F'$  on the new dataset  $D'$ .

## 3. Results

We evaluated the effectiveness and efficiency of the proposed adversarial attacks using JSMA and C&W to generate executable malware adversarial examples. Also, training and distillation defense mechanisms were examined.

### 3.1. Model Performance

In this subsection, we evaluate the effectiveness of our CNN and RNN models. We report classification accuracy and loss metrics to quantify the proposed models' effectiveness. In CNN, the used loss function was cross-entropy, and Stochastic Gradient Descent (SGD) was used for optimization. The training and validation sets in this experiment demonstrated the high performance of the trained CNN model regarding the classification accuracy, with a training accuracy of 84% and a validation accuracy of 77%.

The RNN training accuracy was 76%. In our experiment, we demonstrated that the trained RNN model performed similarly to how it did in the original paper that introduced the RNN architecture in terms of classification accuracy by [28]. The highest RNN accuracy they reached was 88%.

### 3.2. Attacks and Defense Mechanisms

This section evaluates the performance of JSMA and C&W attacks using the evasion rate. As described in Section 2.3.3, this paper presents two experimental manipulation

approaches that preserve functionality, namely a partially perturbed PE header and a new section of perturbations.

The evasion rate measures how many samples evaded the classifier after adversarial examples were previously correctly classified as malicious, as given in Equation (10).

$$\text{Evasion Rate} = \frac{TP_{adv}}{TP} \quad (10)$$

where  $TP_{adv}$  is the number of  $TP$  samples correctly classified as target labels ( $y^*$ ). Table 3 shows the evasion rates for adversarial examples generated by JSMA and C&W attacks.

**Table 3.** Results of JSMA and C&W attacks in terms of evasion rate.

Model	$JSMA_{Header}$	$JSMA_{Section}$	$C\&W_{Header}$	$C\&W_{Section}$
CNN	63.9%	49.6%	91.06%	80.21%
RNN	37.58%	32.24%	84.6%	60.3%

An adversarial example generated by  $JSMA_{Header}$  successfully evaded the CNN model with a 63.9% evasion rate while evading the RNN model with a rate of 37.5%. In a  $JSMA_{Section}$  attack, the CNN was evaded with a rate of 49.6%, while the RNN was evaded with a rate of 32.2%. The C&W method had a higher evasion rate than the JSMA method when applied to both CNNs and RNNs. Furthermore, C&W performed better against the CNN rather than the RNN. With the  $C\&W_{Header}$  attack, CNN and RNN detection was evaded with rates of 91.06% and 84.60%, respectively, while with the  $C\&W_{Section}$  attack, they were evaded with rates of 80.21% and 60.30%, respectively. It can be seen that C&W was less affected by the complexity of the model structure, compared to JSMA, specifically against RNN. Moreover, in C&W attacks, the evasion rate and distortion were found to be oppositely correlated. As distortion increased, the evasion rate decreased. Thus, the size of perturbed headers resulted in a higher evasion rate than the perturbed section.

As shown in Table 3, based on our results for both attacks, we demonstrated that the header approach outperformed the section approach against the CNN and the RNN. In this paper, defense mechanisms against adversarial attacks are examined and analyzed. In this section, we examine the ability of these defense mechanisms to strengthen the resistance of DL models to a variety of adversarial attacks compared to other studies [10,15,20]. Therefore, we applied two types of mechanisms in response to both attacks, namely training and distillation. Evasion rates before and after defense mechanisms were applied against JSMA and C&W and are shown in Figures 4 and 5.

A training defense mechanism reduced the evasion rate of  $JSMA_{Header}$  attacks against the CNN to 1.47%. In the case of  $JSMA_{Section}$  attacks, the training defense reduced the evasion rate to 16.7%. As a result of a distillation defense, the evasion rate of  $JSMA_{Header}$  attacks that evaded CNN detection was reduced to 26.48%, while the evasion rate of  $JSMA_{Section}$  attacks was reduced to 22.41%. As seen from Figure 4, the training defense reported significantly higher effectiveness than distillation against our white-box attacks.

In the case of  $JSMA_{Header}$  attacks against the RNN, training defenses reduced the evasion rate to 5.03%. The training defense was able to reduce the evasion rate to 16.7% against the  $JSMA_{Section}$  attacks that evaded the RNN. In a distillation defense mechanism,  $JSMA_{Header}$  attacks against RNN were defended with an 11.3% evasion rate, while  $JSMA_{Section}$  attacks were defended with a 13.34% evasion rate. The training defense mechanism performed better against the  $JSMA_{Section}$  attack than the  $JSMA_{Header}$  attack in both the CNN and RNN models. A distillation defense performed better on the CNN than on the RNN model when defending against the JSMA. Furthermore, the distillation defense performed better against  $JSMA_{Section}$  attacks than against  $JSMA_{Header}$  attacks.

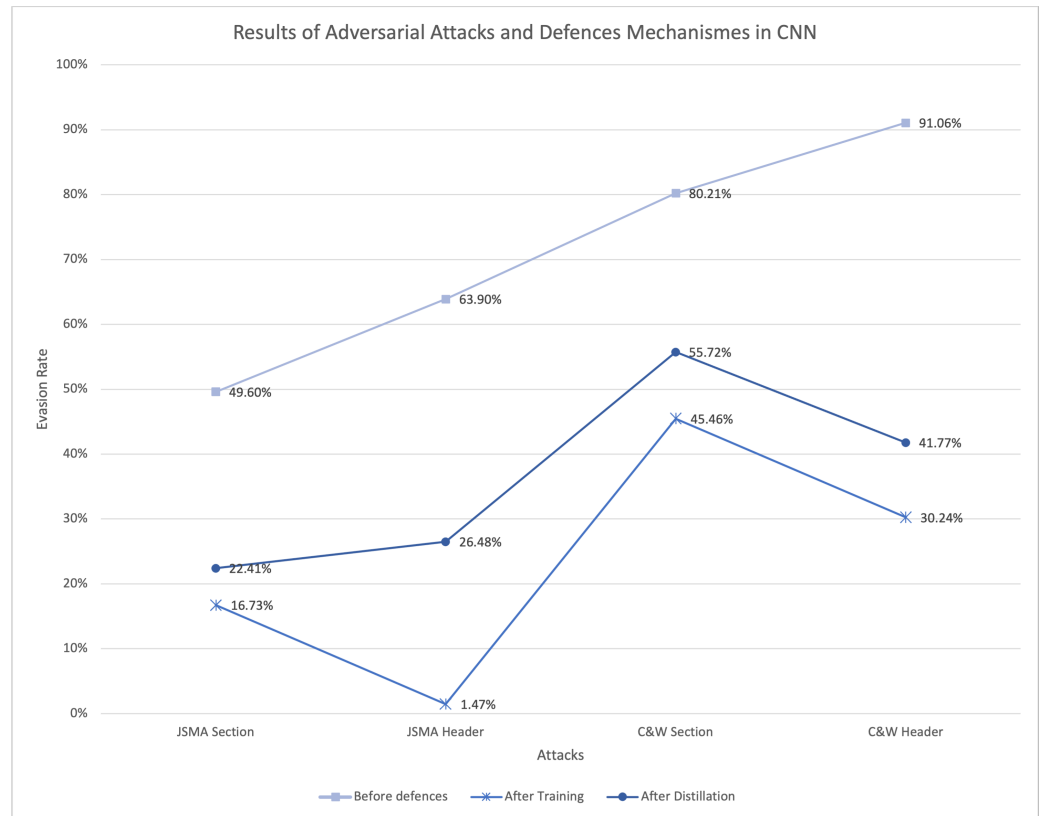


Figure 4. Defense mechanism results against JSMA and C&W attacks in CNN.

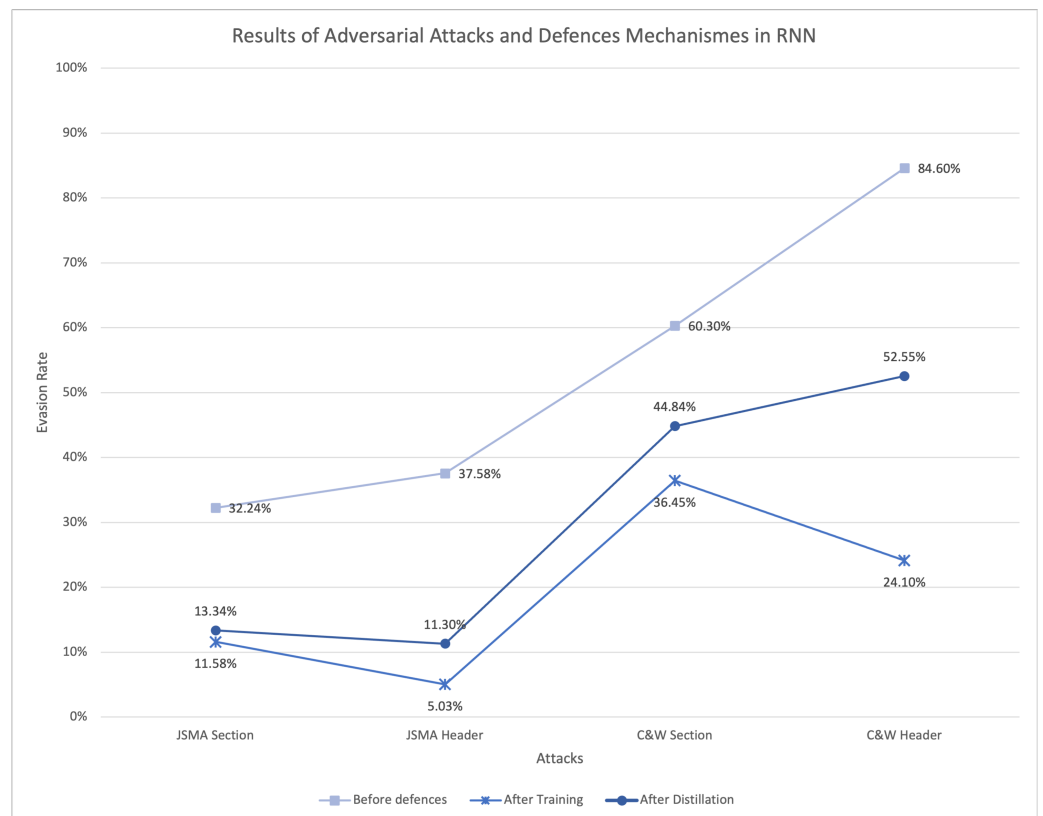


Figure 5. Defense mechanisms results against JSMA and C&W attacks in RNN.

Interestingly, training defense mechanisms outperformed distillation against all attacks, as shown in Figures 4 and 5. Most adversarial studies of computer vision have demonstrated that distillation is more effective than training defense mechanisms [5,15,17]. Our results show a notable difference between the efficiency of the attacks against continuous and discrete data. To our knowledge, none of the recent studies proved the opposite viewpoint when referring to binary data, which is the subject of our research.

#### 4. Discussion

Our experiments demonstrate that adversarial attacks are possible if small changes are made to the PE to evade the malware detection model. JSMA and C&W provide the capability to attack malware detection models. Nevertheless, these attacks are still limited by the PE structure. The semantic structure of PE limits attacks to attacking PE and raises the difficulty of developing a universal defense mechanism.

However, this paper describes how adversarial attacks exploit certain PE areas, such as DOS header and appending a section with perturbation, making it a prime target for attackers. These areas can be particularly effective for attackers, bypassing detection systems. To address this vulnerability, our experiment was designed by categorizing an attacker's influence over the PE's structure based on knowledge about the malware domain and designing a classifier that can prevent these manipulations by default. This mechanism is not restricted in its application to a specific type of adversarial attack on PEs. It builds a robust classifier in response to adversarial examples as part of a proactive defense mechanism.

Thus, with these areas removed, we demonstrated that a robust classifier can be constructed without affecting the performance. Most importantly, this mechanism does not leave any areas susceptible to attack by an adversary. In fact, by eliminating the vulnerable area in the malicious file, the attacker will no longer be able to evade the classifiers. Therefore, we stripped the areas in our experiment and then trained the CNN and RNN models. Thus, we could achieve high accuracy rates. The reported accuracy for CNN was 95% and for RNN was 97%.

#### 5. Conclusions

In this paper, we introduced a practical approach for placing adversarial payloads directly into PE to bypass DL-based malware detectors. The effectiveness of our technique was demonstrated by crafting a white-box adversarial attack against CNN and RNN models. Our targeted models were byte-based malware classifiers that took PE file inputs. In this paper, we introduced two preservation functionality approaches to perturb PE, namely, perturbing the DOS header and appending a section. An extensive dataset was collected from EMBER [25] by crawling VirusShare. All unlabeled samples were then assigned a label by querying AVClass provided by [26]. The challenge of creating an executable adversarial example without altering the semantics was met in this paper. By attacking our models, we could successfully generate executable functional adversarial examples. The attack effectiveness was demonstrated by our ability to cause the target classifier to misclassify all of the evaluated adversarial examples generated by JSMA and C&W with accuracies of 63.9% and 91.06%, respectively.

To prevent evasion attacks, we addressed two countermeasures, namely training and distillation. Furthermore, we presented a proactive defense mechanism that targets DL-based PE detection systems. Since this mechanism focuses on the PE format, it can be generalized to all adversarial attacks. Our success in the white-box adversarial attack highlights the need for continued research on and the enhancement of defenses against adversarial attacks on DL-based malware detection frameworks. In the future, we recommend exploring whether other areas of PE can be perturbed by practical manipulation to generate adversarial malware.

**Author Contributions:** During the entire research process, S.A. participated in literature reviews, experiment design, data collection, and manuscript preparation. The supervisor, H.B., provided guidance in research design, methodology, and data analysis, providing valuable insights throughout the all research phases. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors extend their appreciation to the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University for funding and supporting this work through Graduate Students Research Support Program.

**Data Availability Statement:** Publicly available datasets were analyzed in this study. These data can be found at: <https://virusshare.com/> (accessed on 1 July 2020). The dataset is a part of the EMBER dataset. A list of hashes is available upon request.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

CPS	Cyber-Physical Systems
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
JSMA	Jacobian-based Saliency Map Attack
C&W	Carlini and Wagner
DL	Deep Learning
FGSM	Fast Gradient Sign Method
PE	Portable Executable
SVM	Support Vector Machine
IoT	Internet of Things
EMBER	Endgame Malware Benchmark for Research
LSTM	Long Short-Term Memory
SGD	Stochastic Gradient Descent
RF	Random Forest
GAN	Generative Adversarial Network

## References

1. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. In Proceedings of the 2nd International Conference on Learning Representations, ICLR 2014—Conference Track Proceedings, Banff, AB, Canada, 14–16 April 2014; pp. 1–10.
2. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015; pp. 1–11.
3. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, 21–24 March 2016; pp. 372–387. [[CrossRef](#)]
4. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2574–2582.
5. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 39–57. [[CrossRef](#)]
6. Kurakin, A.; Goodfellow, I.J.; Bengio, S. Adversarial examples in the physical world. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017—Workshop Track Proceedings, Toulon, France, 24–26 April 2017; pp. 1–14.
7. Liu, A.; Guo, J.; Wang, J.; Liang, S.; Tao, R.; Zhou, W.; Liu, C.; Liu, X.; Tao, D. XX-Adv: Physical Adversarial Object Attacks against X-ray Prohibited Item Detection. In Proceedings of the 32nd USENIX Security Symposium, Anaheim, CA, USA, 9–11 August 2023; Volume 6, pp. 3781–3798.
8. Li, Y.; Li, Y.; Dai, X.; Guo, S.; Xiao, B. Physical-World Optical Adversarial Attacks on 3D Face Recognition. In Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 24699–24708. [[CrossRef](#)]
9. Yao, J.; Chen, X.; Zhang, X.L.; Zhang, W.Q.; Yang, K. Symmetric Saliency-Based Adversarial Attack to Speaker Identification. *IEEE Signal Process. Lett.* **2023**, *30*, 1–5. [[CrossRef](#)]

10. Chen, X.; Li, C.; Wang, D.; Wen, S.; Zhang, J.; Nepal, S.; Xiang, Y.; Ren, K. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Trans. Inf. Forensics Secur.* **2018**, *15*, 987–1001. [[CrossRef](#)]
11. Suci, O.; Coull, S.E.; Johns, J. Exploring adversarial examples in malware detection. In Proceedings of the 2019 IEEE Symposium on Security and Privacy Workshops (SPW), San Francisco, CA, USA, 19–23 May 2019; pp. 8–14. [[CrossRef](#)]
12. Kreuk, F.; Barak, A.; Aviv-Reuven, S.; Baruch, M.; Pinkas, B.; Keshet, J. Deceiving end-to-end deep learning malware detectors using adversarial examples. *arXiv* **2019**, arXiv:1802.04528.
13. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; pp. 533–537. [[CrossRef](#)]
14. Demetrio, L.; Biggio, B.; Lagorio, G.; Roli, F.; Jan, C.R. Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv* **2019**, arXiv:1901.03583.
15. Podschwadt, R.; Takabi, H. Effectiveness of adversarial examples and defenses for malware classification. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Orlando, FL, USA, 23–25 October 2019; pp. 1–16.
16. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
17. Huang, Y.; Verma, U.; Fralick, C.; Infantec-Lopez, G.; Kumar, B.; Woodward, C. Malware evasion attack and defense. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Portland, OR, USA, 24–27 June 2019; pp. 34–38. [[CrossRef](#)]
18. Stokes, J.W.; Wang, D.; Marinescu, M.; Marino, M.; Bussone, B. Attack and defense of dynamic analysis-based, adversarial neural malware detection models. In Proceedings of the MILCOM 2018–2018 IEEE Military Communications Conference (MILCOM), Los Angeles, CA, USA, 29–31 October 2018; pp. 1–8. [[CrossRef](#)]
19. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial examples for malware detection. In Proceedings of the 22nd European Symposium on Research in Computer Security, Oslo, Norway, 11–15 September 2017; Springer: Cham, Switzerland, 2017; pp. 62–79. [[CrossRef](#)]
20. Liu, X.; Zhang, J.; Lin, Y.; Li, H. ATMPA: Attacking machine learning-based malware visualization detection methods via adversarial examples. In Proceedings of the International Symposium on Quality of Service, Phoenix, AZ, USA, 24–25 June 2019; pp. 1–12. [[CrossRef](#)]
21. Khormali, A.; Abusnaina, A.; Chen, S.; Nyang, D.; Mohaisen, A. COPYCAT: Practical adversarial attacks on visualization-based malware detection. *arXiv* **2019**, arXiv:1909.09735.
22. Park, D.; Khan, H.; Yener, B. Short Paper: Creating adversarial malware examples using code insertion. In Proceedings of the 18th IEEE International Conference on Machine Learning and Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; pp. 1283–1290. [[CrossRef](#)]
23. Li, K.; Zhang, F.; Guo, W. FGAM: Fast Adversarial Malware Generation Method Based on Gradient Sign. *arXiv* **2023**, arXiv:2305.12770.
24. Li, H.; Cheng, Z.; Wu, B.; Yuan, L.; Gao, C.; Yuan, W.; Luo, X. Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information. In Proceedings of the 32nd USENIX Security Symposium, Anaheim, CA, USA, 9–11 August 2023; Volume 2, pp. 1181–1198.
25. Anderson, H.S.; Roth, P. Ember: An open dataset for training static pe malware machine learning models. *arXiv* **2018**, arXiv:1804.04637.
26. Sebastián, M.; Rivera, R.; Kotzias, P.; Caballero, J. Avclass: A tool for massive malware labeling. In Proceedings of the 19th International Symposium on Research in Attacks, Intrusions, and Defenses, Paris, France, 19–21 September 2016; Springer: Cham, Switzerland, 2016; pp. 230–253.
27. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware detection by eating a whole EXE. In Proceedings of the Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2017; pp. 268–276.
28. Raff, E.; Sylvester, J.; Nicholas, C. Learning the PE header, malware detection with minimal domain knowledge. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2017, Dallas, TX, USA, 3 November 2017; pp. 121–132. [[CrossRef](#)]
29. Bahdanau, D.; Cho, K.H.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015; pp. 1–15.
30. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial perturbations against deep neural networks for malware classification. *arXiv* **2016**, arXiv:1606.04435.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.