MRA2: Malicious-function Reserved Automated Attacks on Malware Detection

Journal:	*Transactions on Dependable and Secure Computing*
Manuscript ID	TDSC-2021-10-0780
Manuscript Type:	Regular Paper
Keywords:	Adversarial example, malware detection, I.2.6.g Machine learning < I.2.6 Learning < I.2 Artificial Intelligence < I Computing Methodologies

MRA2: Malicious-function Reserved Automated Attacks on Malware Detection

Jiliang Zhang, *Senior Member, IEEE*, Zihao Luo, Honggang Yu, *Student Member, IEEE*, He Li, *Member, IEEE*, Xin Zhang, Kenli Li, *Senior Member, IEEE*, Yupeng Hu, *Senior Member, IEEE*

Abstract—Machine Learning (ML) based malware detection methods are becoming increasingly common due to their ability of achieving high accuracy and low response time while addressing potential malware threats. Unfortunately, recent studies demonstrated that these ML models are vulnerable to adversarial examples which fool the models to output incorrect results. Previous malware adversarial attacks mainly focus on adding particular perturbations to input feature vectors in order to generate adversarial examples to evade the model detection. However, these generated vectors can not be executed directly to perform malicious actions and even destroy original malware functionalities in some cases.To address this issue, in this paper we propose an effective black-box attack methodology named **M**alicious-function **R**eserved **A**utomated **A**ttack (MRA2) against ML-based malware detection models. Our attack can automatically locate the function-irrelevant parts of executable malware files and generate a subset of executable adversarial examples with malicious function well-reserved. Experimental results on a series of popular ML models demonstrate that our attack achieves up-to 100% success rate for all tested detectors and 100% malware-function reservation with online sandbox tests. In addition, three defense strategies, such as adversarial training, ensemble training and SafetyNet, are also employed in this paper to further evaluate the effectiveness of MRA2.

Index Terms—Adversarial example, malware detection, machine learning

1 Introduction

Malware has posed a great threat to computer devices and brought huge economic loss in recent years. For example, AV-TEST [1] reported that over 3.12 million malware masqueraded as benign software were counted in 2020. In 2021, Acer was attacked by REvil malware and demanded the largest known ransom of $50,000,000. [2].

In order to mitigate these threats, some malware detection approaches have been developed, mainly including digital signatures-based malware detection [3], heuristic-based malware detection [4] and ML-based malware detection [5–7]. Unfortunately, due to economic benefits, adversaries quickly developed automated malware development toolkits (*e.g.*, Zeus [8]), resulting in a massive proliferation of new malware samples. These new malware samples can easily bypass traditional signatures or heuristic detection methods. Besides, traditional signatures or heuristic detection methods are time- and labor-consuming, which makes the anti-malware software tools less effective to new threats. To address these issues, machine learning shows great advantages in malware detection thanks to its high detection accuracy, the ability of classifying massive potential malware and reducing the response time to new malware threats.

However, these machine learning based detection models are usually vulnerable to adversarial attacks [9–13]. For example, Lin *et al.* [11] introduced a new attack method named BMI-FGSM which generates adversarial examples of malware to escape the detection of the ML-based malware detector. Liu et al. [14] proposed an adversarial texture malware perturbation attack (ATMPA) method that crafts malicious examples by adding tiny perturbations on the transformed dataset.

Although these existing methods achieve significant progress, they remain impractical in the real-world scenarios due to the following reasons: 1) Current adversarial attacks cannot effectively generate executable AEs against ML-based detection models and some of them even damage the function of malware [14–16]; 2) The transferability of these AEs is low (e.g., average 65% [14]) among different ML models; 3) Current adversarial attacks require repeated query to the victim model for constructing the synthetic dataset, which introduces huge computation overhead and can be easily discovered by the model vendor [17, 18].

To address these challenges, we propose a new black-box adversarial attack methodology named malicious-function reserved automated attack (MRA2) against ML-based malware detection models. MRA2 is the first malicious function reserved adversarial attack on ML-based visualization malware detection methods. Compared to existing works, our MRA2 can locate the function-irrelevant segments of executable malware files automatically. As a result, the generated AEs can not only preserve the malicious function but also be executed directly without repackage, recompiling and disassembly. Moreover, to address the issue of lack of gradients in black-box attack scenario, we propose a new attack which uses the improved genetic algorithm (GA) [19] to craft the particular perturbation. In comparison with existing works, the GA algorithm used in our attacks has the following advantages:

- In the black-box scenario, the gradient of ML models cannot be directly calculated, while the optimal solution can be obtained without calculating the gradient in the genetic algorithm because the search process of GA is random and controllable.

- *J. Zhang, Z. Luo, X. Zhang, K. Li and Y. Hu are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410012, China. E-mail: zhangjiliang@hnu.edu.cn.*
- *H. Yu is with the School of Information and Communication, Guilin University of Electronic Technology, Guilin, 541004 China. E-mail: honggang.yu.uf@gmail.com.*
- *H. Li was with the Computer Lab, University of Cambridge, Cambridge, CB3 0FD, UK. E-mail: he.li@ieee.org.*

- The search of GA starts from the population, which has potential parallelism, i.e., multiple individuals can be compared at the same time.
- GA is a modular algorithm, which can be easily modified and adaptable to our problem.

Versus the state-of-the-art black-box attacks [15, 17], MRA2 avoids the access to the architecture and parameters of ML models and training a substitute model. For the MRA2, we implement two types of attacks, including injection and non-injection attack. Specifically, injection attack aims at injecting Perlin noise variant to the original malware for generating AEs without damaging the malicious function of the malware simultaneously. In this paper, we generate the optimal Perlin noise variant using the genetic algorithm and fool the victim ML model without damaging the malicious function of the malware. In contrast, the non-injection attack does not require an adversary to inject additional noises to the original inputs. In the non-injection attack, an automated searching method is proposed to identify the function-irrelevant parts in the original malware. Since executable malware files in Windows OS include aligned sections, there are several automatically filled parts which are independent of malware's function at the end of unfilled sections. Therefore, an adversary can modify these filled parts and generates adversarial examples for fooling victim ML-based detection models. In the byte level, the automatically filled parts are consistent and regular, so the proposed non-injection attack employs this property to identify function-irrelevant parts. Then, we utilize genetic algorithm to generate the optimal individual. With the information of the position individual, attackers can modify the executable malware files directly to fool the victim model.

In this paper, MRA2 is designed to bypass ML-based malware detection. Although MRA2 is ML-algorithmically independent, we implemented three popular ML malware detectors based on Inception(CNN), support vector machine (SVM) and random forest (RF), respectively. ML-based malware detectors are trained with the kaggle dataset [20], VirusShare files and Windows executable files. Experimental results demonstrate that our attack can achieve an up-to 100% attack success rate for the tested ML malware detectors and show 100% malware-function reservation. Moreover, we also evaluate the effectiveness of MRA2 on three state-of-the-art defenses, including adversarial training [21], ensemble adversarial training [22] and SafetyNet [23], and find that our attack still reaches up-to 96% success rate under the defense-deployed models.

In conclusion, we mainly make the following contributions in this paper:

- A new black-box adversarial attack is proposed to generate executable malware AEs against ML based detection models. The malicious function of the generated malware AEs can be well-reserved and executed directly without repackage, recompiling and disassembly.
- An automatic algorithm is designed to identify the function-irrelevant parts of executable malware, without disassembly and accessing to the source code of malware.
- The effectiveness of MRA2 is evaluated on three popular ML-based malware detectors, including SVM, RF and CNN. The experimental results show that our attack achieves 100% success rate. Moreover, we also validate the robustness of MRA2 against three commonly used defense mechanisms such as adversarial training, ensemble adversarial training and SafetyNet, and find that our MRA2 can still bypass these mechanisms with high success rate.
- To enhance the robustness of ML-based detection models, we introduce a novel defense strategy which introduce a random translation layer into the ML-model. The experimental results show that the success rates of MRA2 under the deployed model are less than 37%.

The rest of the paper is organized as follows. Section 2 of this paper introduces the relevant work in the field of malware detection and adversarial example. Section 3 describes the MRA2 in detail. Section 4 evaluates and analyzes MRA2. Then we make a conclusion in Section 5 .

2 Related Work

2.1 Traditional Malware Detection Methods

Some defense mechanisms have been proposed to reduce the impact of the malware during the inference stage. In this paper, we roughly divide them into the following two categories: signatures-based malware detection [3] and heuristic-based malware detection [4].

2.1.1 Signatures-based Malware Detection

To protect legitimate users from malware threats, anti-malware companies developed a lot of malware detection products. Typically, they use the signature-based method to identify the known threats. A signature is a short sequence of bytes unique to each known malware, which allows newly encountered files to be correctly identified with a small error rate [24].

Signature-based detection method identifies unique strings from the binary code [3]. Whenever a new type of malware is unleashed, anti-malware companies need to obtain an instance of the new malware, analyze the instance, create new signatures, and deploy in their clients [3]. However, the signature bases are often manually generated, updated, and disseminated by domain experts. This process is usually time and labor consuming, and the speed of malware creation (e.g., over 100,000 new malware/day) is faster than the signature generation [4], which makes the anti-malware software tools less responsive to new threats. In additional, signature-based detection method may allow some malware samples to bypass the detection and stay undetected for a long time. For example, a typical time window between a malware's release and its detection by anti-malware software tools is about 54 days [25]. In the worse case, after 180 days, 15% of samples are still undetected [26]. By applying the counter-measures such as encryption, packing, obfuscation, polymorphism and metamorphism, malware writers can easily bypass the signature-based detection. Therefore, more powerful malware detection methods are urgent.

2.1.2 Heuristic-based Malware Detection

To make attackers harder to bypass the malware detection, the heuristic-based detection was proposed in recent years. From the late 1990s until 2008, the heuristic-based detection method was the most important way for malware detection [4]. Heuristic-based detection is based on rules/patterns determined by experts to discriminate malware samples and benign files. These rules/patterns should be generic enough to be consistent with variants of the same malware threat, but not falsely matched on benign files [27]. However, the analysis of malware samples and the construction of rules/patterns by domain experts is often error-prone and time-consuming. More importantly, as introduced above, driven by economic benefits, the malware industry has invented automated malware development toolkits (such as Zeus [8]) to create and mutate thousands of malicious codes per day which can bypass such traditional signature-based or heuristic-based detection [4]. Therefore, manual analysis has become a major bottleneck in the workflow of malware detection, demanding for intelligent techniques to detect incoming malware automatically. Such intelligent techniques should allow anti-malware vendors to keep pace with the rapid malware deployment and also reduce their response time to new malware threats.

2.2 Machine Learning-based Malware Detection

Compared with the above traditional malware detection methods, machine learning shows great advantages in terms of high detection accuracy, ability to classify massive potential malware and reducing the response time to new malware threats. In ML-based detection systems, the detection process generally includes two-steps, feature extraction and classification [5]. In the first step, various features such as API calls, binary strings, and program behaviors are extracted statically or dynamically to capture the characteristics of the malware. In the second step, ML algorithms such as CNN, DNN, SVM, RF are used to classify the malware samples into different classes (malicious or benign) based on the feature vectors. In this paper, We implemented three popular ML malware detectors based on Inception(CNN), support vector machine (SVM) and random forest (RF), respectively.

In recent works, many kinds of features are extracted statically to represent malware for detection. Analyzing a specific byte sequence in Portable Executable(PE) is a widely used static technology. Raff *et al.* use the byte blocks with specific size as features to represent malware and classify them by ML algorithms [28] and most researches [29–32] use byte sequences to represent malware based on n-grams; Opcode frequency is one of the most commonly used features. It measures the number of times for each specific opcode appearing in the program. Khodamoradi*et al.* [33] and Ye *et al.* [34] used opcodes frequency to represent malware. Similarly to opcodes, APIs and system calls enable the analysis of samples' behaviour at a higher level. They can be either extracted statically or dynamically by analysing the disassembly code or the execution traces. The associations and relationships among the APIs may capture the underlying semantics of the malware behaviors and can be used as essential features for malware detection, *e.g.* The Windows API calls in the "KERNEL32.DLL" of "OpenProcess", "CopyFileA", "CloseHandle", "GetVersionEx A", "GetModuleFileNameA" and "WriteFile" always co-occur in malware samples while rarely show together in a benign file set. Mansour *et al.* extracted and selected a set of novel features including API calls, for the effective representation of malware samples. However, these works are susceptible to code obfuscation and requires a series of operations such as unpacking, decompilation, feature extraction and selection.

Dolly *et al.* [35] proposed a novel malware detection approach based on extracting API sequences, where an API monitor is used to keep track of the execution of input samples and capture the API calls from these samples to get the API call sequences. Then, feature selection method based on N grams and odds ratio selection is used to capture unique and distinct API sequences from the extracted API calls. Next, a model is built using ML algorithms to classify the final API call sequences. In addition to API call, a huge number of relevant features, such as the statistics on used protocols, TCP/UDP ports, HTTP requests, DNS-level interactions, can be obtained by observing how the malware interacts with the network. Many works require dynamic analysis to extract these features [36–38]. Mohaisen *et al.* proposed an automated and behavior-based malware detection system which provides tools to collect features of malware including network-related features by running malware samples in virtualized environments. File operations which executed by malware is also important features for detecting malware. Features of interest mainly concern how many files are read or modified, what types of files and in what directories, and which files appear in not-infected/infected machines [37, 39–41]. Although these works can identify the unique characteristics of malware [42–44], but need to execute malware in a controllable environment, such as virtual machines or sandboxes, to trace suspicious behaviors, and hence it is time-consuming [27].

2.3 Malware Visualisation Detection

The above ML-based malware detection methods have limited detection ability for different feature types. In order to accelerate the malware detection process, ML-based visualization detection methods have attracted great attention in recent years [7, 45, 46]. The process of ML-based visualization detection is as follows. Firstly, malware visualization methods [46, 47] are used to transform the hard-to-identify malware information into image data with certain feature information. We can then train ML models using transformed grayscale images. Finally the trained models can detect the new grayscale images of malware. The transformed malware grayscale images can be easily detected by the detectors based on RF [48], SVM [47], CNN [49]. For example, Han *et al.* [45] proposed to convert malware into a grayscale image and then employed converted grayscale images to entropy graphs which are stored in a reference database. In this case, malware can be detected by calculating the similarity between entropy graphs in the reference database and a newly converted entropy graph. Go *et al.* [46]improved the detection accuracy through using the state-of-the-art Convolution Neural Network (CNN) model after transforming the attributes of raw malware binary executable files to greyscale images, while the overhead is not increased significantly.

2.4 Adversarial Attacks

ML-based malware detection performs well with the high detection accuracy, the ability of classifying massive potential malware and reducing the response time to new malware threats. However, ML-based malware detectors suffer from adversarial attacks. Many adversarial attacks have been proposed to deceive the ML-based malware detection. For instance, Grosse *et al.* employed the FGSM to generate adversarial examples of Android malware to escape the detection of a DNN-based malware detector [16]. Demontis *et al.* construct the adversarial examples by manipulating the features of the malware, such as the strings that are present (absent) in the malware and the API calls [50]. Rosenberg *et al.* inserted APIs in the API-call sequence to produce adversarial examples [15]. In these attacking methods, malware source code is required for constructing the feature vector of malware, and the generated adversarial examples are feature vectors which cannot be executed directly. Therefore, it requires extra manual operations to generate executable malware adversarial examples, such as repackage, recompiling and disassembly. In order to avoid the access to the source code of malware to construct the feature vector, Kolosnjaji *et al.* injected bytes at the end of the malware and modified the injected bytes to generate the gradient-based AEs [51]. However, only 60% attack success rate was achieved and analysing the API call dynamically caused more overhead than static analysis-based methods. Anderson *et al.* were the first to use reinforcement learning to escape ML-based malware detection [52]. A series of functionally irrelevant modification operations, such as injecting entry into the code and importing irrelevant packages, are predefined. Reinforcement learning is then used to find the optimal sequence of modification operations. However, attack success rate is only 24% and the design of some complicated functionally irrelevant operations requires access to the source code of the malware. Recently, Liu *et al.* have proposed the first framework to attack ML-based malware visualisation detection [14]. By introducing some tiny perturbations on the visualised malware samples, the generated AEs can evade malware detectors easily. However, the generated AEs can not preserve the original malicious function. Lucas *et al.*[53] proposed an attack that interweaves binary-diversification techniques and optimization frameworks, which modified the functional part of the binares to mislead DNNs. But they did not propose a defense to enhance the robustness of the model against the attack.

3 The Proposed MRA2

Removing or modifying the content of malware can damage the function of malware. An effective attack strategy must preserve the malicious function in the generated AEs. In the proposed MRA2, two attack strategies, including injection attacks and non-injection attacks, are malicious-function preservable. The proposed MRA2 is implemented in the black-box scenario where adversaries have no access to the model architecture and parameters and hence can not calculate the gradient of loss function for generating AEs [14, 21, 50, 54]. To approximate the gradient, a popular black-box attack usually exploits the substitute model to generate AEs to fool the victim model [55], so the success rate of such attacks depends on the transferability of AEs [56]. However, the transfer rate of AEs among different victim models is low [14, 17]. Moreover, training substitute models would also introduce high computation overheads due to a large variety of queries to the victim model and is easy to be detected by the model supplier.

To address these challenges, this paper uses genetic algorithm (GA) [19] to guide the generation of AEs, which has the following advantages:

- In the black box scenario, the gradient of ML models cannot be directly calculated, while the optimal solution can be obtained without calculating the gradient in the genetic algorithm because it uses a probabilistic mechanism to iterate so that the search process is random. In this case, the perturbation can be added in a random and controllable way.
- The search of GA starts from the population, which has potential parallelism, i.e., multiple individuals can be compared at the same time.
- GA is a modular algorithm, which can be easily modified and adaptable to different problems.

In this paper, GA is improved for our proposed MRA2, which will be discuss in detail in Section 3.2.1.

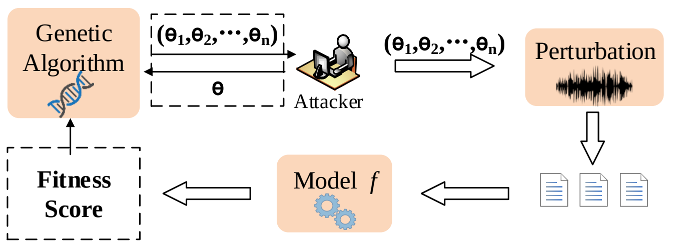

Genetic Algorithm
$(\theta_1,\theta_2,\cdots,\theta_n)$
θ
Attacker
$(\theta_1,\theta_2,\cdots,\theta_n)$
Perturbation
Fitness Score
Model f

Fig. 1: Framework of MRA2. A set of parameters which can guide the injection or non-injection attack are designed as individuals in GA, the optimal parameter population is obtained from GA for AE generation.

3.1 The Framework of MRA2

As shown in Fig. 1, the proposed MRA2 mainly consists of the following five aspects:

1) All of the parameters and corresponding parameter space are sent to GA to start iterations.
2) The parameter population is obtained from GA in each iteration.
3) The malware samples are disturbed with the guide of the parameter population.
4) Disturbed malware samples are sent to the victim model to calculate the fitness score for each individual in GA.
5) A new population is finally generated by eliminating the individuals that have low fitness scores and selecting the individuals that have high fitness scores.

During the iteration, if the optimal parameters can guide the generation of adversarial malware, the attack would be successful, otherwise, the population evolution will be continued in GA until the maximum iteration is reached.

3.2 AE Generation for Black-box Attacks

In this paper, we assume that an attacker has no knowledge of the victim model's details, such as architecture, parameters and exact training dataset, but can obtain the classification outputs (e.g., labels or confidence score) while providing the victim model with random inputs, i.e., black-box setting.

Our MRA2 operates the original executable malware for generating AEs without disassembling the malware, which avoids the access to the source code and the internal secret information of ML models. Therefore, potential attackers in MRA2 can be either a malware developer or a malware owner, which makes MRA2 more threatening and generalized. In addition, our proposed MRA2 can be applied not only to malware visualisation detection, but also to byte sequences-based malware detection because MRA2 is operated on byte-level content malware. The details of injection attacks and non-injection attacks are given below:

3.2.1 Injection Attacks

In the proposed injection attacks, we generate the AEs against ML-based malware detection models by injecting variants of Perlin noise [57] at the end of the malware files. The detailed process of injection attacks is shown in Algorithm 1. We firstly craft the original Perlin noise as follows:

$$N(x,y) = a + (b-a)s_y, \quad (1)$$

where $N(x,y)$ represents the noise value of the corresponding coordinates (x,y), $a = s + (t-s)s_x$, $b = u + (v-u)s_x$,

Algorithm 1 Injection attacks of MRA2

```
Input: Designed parameters $(\theta_1, \theta_2, \theta_3)$, parameter space $\{(1,100),(1,100),(1,100)\}$, an upper bound on the number of iterations $I_{\max}$, a malware $X$ and a CNN-based malware detector $M$.
Output: An executable AE for malware $X'$.
 1: Send the parameters $(\theta_1, \theta_2, \theta_3)$ to GA
 2: while $i \leq I_{\max}$ do
 3:     Get the parameter population
        $\{(\theta_1,\theta_2,\theta_3)_1, (\theta_1,\theta_2,\theta_3)_2, \cdots, (\theta_1,\theta_2,\theta_3)_n\}$
        from each iteration
 4:     for $j \in [1, n]$ do
 5:         Use $(\theta_1,\theta_2,\theta_3)_j$ to generate noise-variation $N_j$
 6:         Inject $N_j$ into the end of $X$ to get $X_j$
 7:         Visualise $X_j$ to get $\mathrm{Img}_j$
 8:         Send $\mathrm{Img}_j$ to $M$ to get Score-good $\mathrm{Sco}_j$
 9:     end for
10:     Send $\{\mathrm{Sco}_1, \mathrm{Sco}_2, \cdots, \mathrm{Sco}_n\}$ to GA
11:     if GA converges then
12:         return $X'$
13:     end if
14: end while
```

$s_x = f(x - x_0)$, $s_y = f(y - y_0)$, $x_0 = \lfloor x \rfloor, y_0 = \lfloor y \rfloor$, the transition curve is $f(x) = 3x^2 - 2x^3$ [57]. Next, the production of the pseudo random vectors and $((x,y)-(x_i,y_j))$ is calculated:

$$\begin{aligned} s &= G(x_0,y_0)((x,y)-(x_0,y_0)), \\ t &= G(x_1,y_0)((x,y)-(x_1,y_0)), \\ u &= G(x_0,y_1)((x,y)-(x_0,y_1)), \\ v &= G(x_1,y_1)((x,y)-(x_1,y_1)), \end{aligned}$$

where $(x_i, y_j),\ i,j \in \{0,1\}$ represents the grid points near the coordinate $N(x,y)$, $x_1 = x_0 + 1$, $y_1 = y_0 + 1$, $G(x_i, y_j)$ represents a pseudo random vector stored in each grid point. With the generated Perlin noise $N(x,y)$, the parameter variant of Perlin noise is designed as:

$$N' = \theta_3 N\left(\frac{\theta_1 x}{\omega}, \frac{\theta_2 y}{\omega}\right), \tag{2}$$

where the parameter set $(\theta_1, \theta_2, \theta_3)$ in GA algorithm is used to control the structures, patterns and textures of variants of Perlin noise, ω is a constant controlling the noise frequency.

In this paper, we assume that a malicious score $D(x)$ is provided by the victim model, where x represents model's input. The fitness function $F(\cdot)$ in GA is designed as $F(x) = 1 - D(x)$. Before running MRA2, the original malware is read and converted to a two-dimensional array. A set of parameters $(\theta_1, \theta_2, \theta_3)$ is designed as an individual in GA to control the structure, patterns and textures of the variant of Perlin noise. At the beginning of GA, the parameter population is initialized in corresponding parameter space, which contains n individuals (e.g., $n \in \{200, 300, 400, 500\}$). In each iteration, the parameter population is obtained from GA to guide the generation of the variants of Perlin noise. Next, all variants are injected to the end of the converted two-dimensional array to generate the candidate samples of adversarial malware. After noise injection, the candidate samples are propagated to the victim model to obtain the malicious score $D(x)$. With the malicious score $D(x)$, the fitness function can be calculated, a new population is then generated by eliminating the individuals that have low fitness scores and selecting the individuals that have high fitness scores. In the new population, the optimal parameter $(\theta_1, \theta_2, \theta_3)_{\max}$ which has maximum fitness score is used to generate malware X'. If $1 - D(X') > d$, the malware AE would be generated successfully, where d is a predefined constant (e.g., 0.5 or 0.7) in MRA2. Otherwise, population evolution will be continued in GA until the maximum iteration is reached.

The settings of genetic algorithm are discussed in detail: For each individual in the population, crossover & mutation operation is used with the probability of p_i to generate descendant, p_i is obtained randomly from $(0,1)$. Given a pre-set constant $p \in (0,1)$, if $p_i < p$, the individual performs mutation & crossover operation. Theoretically, the larger p is, the wider the search scope in the parameter space will be. However, if p is too large, the population can evolve to an unexpected direction. For each $(\theta_1, \theta_2, \theta_3)_i$, the mutation & crossover operator used in this paper is characterised as follows.

$$(\theta_1,\theta_2,\theta_3)_{\max} + m\left((\theta_1,\theta_2,\theta_3)_a - (\theta_1,\theta_2,\theta_3)_b\right) \tag{3}$$

where $a \neq b \neq i$ and $m \in (0,2)$ is the mutation factor. After mutation & crossover operation, individuals are selected according to the fitness score to form a new population. We compare the optimal individual in current population and that in the previous population. One with a higher fitness score is selected as the current optimal individual. In the whole process of GA optimisation, if there is an individual satisfying the convergence condition, the iteration terminates.

In this paper, GA is improved to accelerate its convergence: In each iteration, the neighbourhood of the optimal individual will be calculated and the parameter space is updated with the neighbourhood of the optimal individual. In addition, to guarantee the generated malware AEs to be executable, we control the value of each element of injected noises within $(0, 255)$, which equals to the range of pixel values in grayscale images. With experiments demonstration, injecting legal variant of Perlin noise at the end of .exe files does not affect the function of malware. In practice, such noise is hard to be perceived, since defenders have no information about original malware samples in the real scenario and the size of our injected noises can be flexibly changed. In order to make the original malware and the malware AE as similar as possible, only tens of bytes can be injected to generate AEs.

3.2.2 Non-injection Attacks

Since executable malware files in Windows OS have several aligned sections, there are several automatically filled parts which are independent of malware's function at the end of some unfilled sections. Therefore, these filled parts can be modified to generate adversarial example. In the byte level, the automatically filled parts are consistent and regular, so the proposed non-injection attack employs this property to identify function-irrelevant parts. Without injecting noise, non-injection attack is able to automatically locate function-irrelevant parts in malware and evade the detection by disturbing the functional-irrelevant parts. The detailed description of our non-injection attack is shown in Algorithm 2.

To begin, the function-irrelevant parts are located as shown in Algorithm 3. The original malware is read and convert to a two-dimensional array with the width of W. If W is too small, e.g., W¡8, the result of Algorithm 3 might include some repeated patterns of malware that are function-relevant. If W is too large,

Algorithm 2 Non-injection attack of MRA2

```
Input: Designed parameters $(a, y, n)^{\mathbb{K}}$, parameter space $[(0, \mathrm{len}(A)), (0, W), (0, 255)]^{\mathbb{K}}$, an upper bound on the number of iterations $I_{\max}$, a malware $X$, a trained model $M$, an array $A$ achieved from Algorithm 3, $W$ = 16 and $\mathbb{K} \in (5, 10)$.
Output: A executable AE for malware $X'$.
 1: Send the parameters $(\theta_1, \theta_2, \theta_3)$ to GA
 2: $A = \mathrm{Loc}(X)$
 3: Reshape $X$ into a $X_{2\mathrm{d}}$ with a fixed width $W$.
 4: while $i \le I_{\max}$ do
 5:   Get the parameter population
      $\{((a, y, n)^{\mathbb{K}})_1, ((a, y, n)^{\mathbb{K}})_2, \cdots, ((a, y, n)^{\mathbb{K}})_n\}$
      from each iteration
 6:   for $j \in [1, n]$ do
 7:     $X_i = X_{2\mathrm{d}}$
 8:     for each element of $((a, y, n)^{\mathbb{K}})_j$ do
 9:       $x = \mathrm{A}[a]$
10:       $X_j[x, y] = n$
11:     end for
12:     Visualise $X_j$ to get $\mathrm{Img}_j$
13:     Send $\mathrm{Img}_j$ to $M$ to get Score-good $\mathrm{Sco}_j$
14:   end for
15:   Send $\{\mathrm{Sco}_1, \mathrm{Sco}_2, \cdots, \mathrm{Sco}_n\}$ to GA
16:   if GA converges then
17:     return $X'$
18:   end if
19: end while
```

Algorithm 3 Locating the function-irrelevant segments in executable malware—$\mathrm{Loc}(X)$

```
Input: The malware binary file $X$.
Output: An array $A$ that records the function-irrelevant segments.
1: $A$ = []
2: Reshape $X$ into $X_{2\mathrm{d}}$ with a fixed width $W$
3: for $i \in [0, \mathrm{len}(X_{2\mathrm{d}})]$ do
4:   if $X[i] == X[i+1]$ then
5:     $A$.append($i$)
6:   end if
7: end for
```

e.g., W¿48, the result of Algorithm 3 might include few function-irrelevant parts. Then the converted two-dimensional array is traversed row by row to find adjacent rows where the contents stored are the same. Then, the index of function-irrelevant rows are stored in an array A.

Further, a set of parameters $(a, y, n)^{\mathbb{K}}$ are designed as an individual in GA. The first variable in the parameter is $a \in (0, \mathrm{len(A)})$, which represents the index of function-irrelevant rows in array A, len(A) represents the length of the array that records the function-irrelevant rows. The second variable is $y \in (0, W)$ that represents the column index of the two-dimensional array, where W represents the width of the two-dimensional array. The third variable n is used to replace content in the corresponding position (x, y) of X_{2d}. For example, $X_{2d}[x, y] = n$, where $x = A[a]$, X_{2d} is the malware to be modified. $\mathbb{K}$ represents the number of elements in the parameter space. To control the perturbation as small as possible, $\mathbb{K}$ is within the range of $(5, 10)$ in this paper. Within non-injection attack, GA is used to guide the generation of the optimal parameter $\{(a, y, n)^{\mathbb{K}}\}_{\max}$. The process of parameter optimization in non-injection attack is similar to that in injection attack. Note that non-injection attack is not required to access the malware source code, and can identify the function-irrelevant segments automatically by operating original malware, thereby facilitating black-box adversarial attacks.

4 Evaluation

4.1 Experimental Setup

Dataset used in the experiment include Microsoft kaggle dataset [20], executable software from VirusShare and Windows OS. Microsoft kaggle dataset contains nine categories of malware families, such as Ramnit, Lollipop and Gatak. Executable malware samples are downloaded from VirusShare via python-spider. The entire dataset is separated into training set and testing set with an ratio of 7:3. We follow the convention in the field and use 10% of instances in dataset to generate malware adversarial examples. Inception-v3, support vector machine (SVM) based and random forest (RF) based models are used in this paper. Inception-based detector is trained on grayscale images converted from software in dataset. SVM-based and RF-based detectors are trained on the byte-level feature set extracted from software.

4.2 Success Rate

Table 1 gives the success rate of MRA2 on Inception-, SVM- and RF-based malware detectors. In order to achieve a good trade-off between the time cost of AE's generation and the success rate of MRA2, the maximum number of iterations is limited to 20 in injection attack, 25 in non-injection attack respectively, the influences of more relevant parameters on MRA2 will be presented in Section 4.7.

As shown in Table 1, our MRA2 injection attack achieves an 95.7% success rate on Inception-based detector and 100% success rate on SVM- and RF-based detectors respectively. Versus injection attack, the success rate of MRA2 non-injection attack is lower, especially on the SVM-based detector. The average reduction is around 10%. The reason of lower success rate in non-injection attack is as follows. The proportion of function-irrelevant segments located in non-injection attack is small (for some samples, the proportion of function-irrelevant segments is even lower than 1%), which is much less than the proportion of function irrelevant parts used in injection attack. Although the success rate of non-injection attack is lower than injection attack, the non-injection attack modifies only 5 to 10 bytes in a malware file, which is more imperceptible than injection attack. Besides, non-injection attack need not the process of generating noise. Finally, the results show that MRA2 can attack all ML-based detectors including CNN-based, SVM based and RF based detectors. This is because MRA2 only needs the output information of the victim model and the optimization process is ML-algorithmically independent. The transfer rate of MRA2 is not considered in this paper because MRA2 does not depend on the transferability of AEs and it attacks the victim model without training a substitute model.

4.3 Evaluation of Function Reservation

To evaluate whether the generated AEs can preserve malicious function after evading ML-based models, all AEs are sent to be

TABLE 1: Success rate of MRA2.

Attack methods	CNN with visulisation	SVM without visulisation	RF without visulisation
Injection MRA2	95.7%	100%	100%
Non-injection MRA2	88.7%	79.3%	96.0%

TABLE 2: Function Reservation rate of MRA2.

Attack methods	CNN with visulisation	SVM without visulisation	RF without visulisation
Injection MRA2	100%	100%	100%
Non-injection MRA2	100%	100%	100%

executed in online sandbox to record their behaviors. In the experiment, the VirusTotal online sandbox [58] was used to analyze the behavior of the malware AEs and further verify their malicious function. During the running time of each AE, if its behaviors are detected by more than 10 anti-malware engines, the malicious function of generated AE is preserved. Results from Table 2 demonstrate that all malware AEs generated by MRA2 preserve their malicious function, the malicious function reservation rate reaches 100%.

4.4 MRA2 with defense Strategies

There are several countermeasures to make ML-based models more robust against adversarial examples. Three representative defense strategies—adversarial training [21], ensemble adversarial training [22] and SafetyNet [23]— are summarised. During the adversarial training, the adversarial examples are generated in every step of training and these adversarial examples are injected into the training set. The result in [21] showed that adversarial training improved the robustness of DNNs. Ensemble adversarial training method augments the ML model's training data with adversarial examples crafted on the pre-trained models [22]. Since adversarial examples can transfer between different models, perturbations crafted on an external model are good approximations for the maximization problem in the optimization problem of adversarial training [21]. Another popular defence strategy we used is SafetyNet [23], where the binary threshold of each rectified linear unit (ReLU) layer's output is extracted as the features of the adversarial detector. SafetyNet detects adversarial examples by an radial basis function (RBF)-SVM classifier. Moreover, we construct a RF-based SafetyNet to evaluate MRA2. Since our MRA2 does not require model parameters to calculate the gradient, defence methods based on gradient masking are not discussed in this section. The above representative defence strategies are deployed on Inception-based malware detector to further evaluate MRA2. Table 3 shows how the these defence strategies affects MRA2 on Inception-v3. AE ratio indicates the proportion of AEs in the training set of adversarial training and ensemble training. Since SafetyNet does not require AEs for training, AE ratio is not considered in SafetyNet.

TABLE 3: The success rate of MRA2 with defence strategies.

Defence strategy	AE ratio	Injection MRA2	Non-injection MRA2
Adversarial training	10%	94.6%	88.4%
	20%	95.7%	85.9%
	30%	93.3%	85.3%
Ensemble training	10%	95.1%	63.5%
	20%	93.6%	45.1%
	30%	90.4%	40.2%
SVM-detect	N/A	95.2%	88.1%
RF-detect	N/A	96.0%	88.3%

In Table 3, as the proportion of AEs ranges from 10% to 30%, the success rate of injection attack and non-injection attack is almost unchanged on adversarial training deployed model. The largest reduction of success rate of injection attack and non-injection attack is 2.4% and 3.1% respectively. Therefore, MRA2 is robust to adversarial training. On the SafetyNet deployed model, MRA2 also performs well. The largest reduction of success rate of injection attack and non-injection attack is 0.5% and 0.6% respectively. However, the success rate of non-injection attack on ensemble training deployed model drops 40%. Some possible reasons include: The small proportion of function-irrelevant segments make it difficult to find the optimal perturbation; Non-injection attack only modifies 5 to 10 bytes in a malware with about 50,000 to 300,000 bytes, which is unable to evade the model deployed with ensemble training; Ensemble adversarial training is proposed to improve robustness of models to black-box attacks, it has unique advantages for black box attacks.

Compared with non-injection attacks, injection attack does not have the first two shortcomings. Therefore, ensemble training slightly effects the success rate of injection attack.

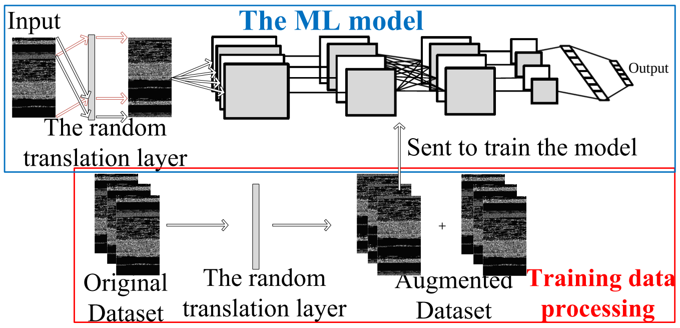


Fig. 2: Framework of the defense.

4.5 MRA2 with the proposed defense strategy

Since MRA2 needs to request the output information of the target model continually for optimization, the current optimal malware sample is related to the optimal malware sample in the last iteration. To mitigate MRA2, a novel defense strategy is proposed. The core idea is to translate the initial malware sample randomly in each iteration to interfere with the process of AE generation.

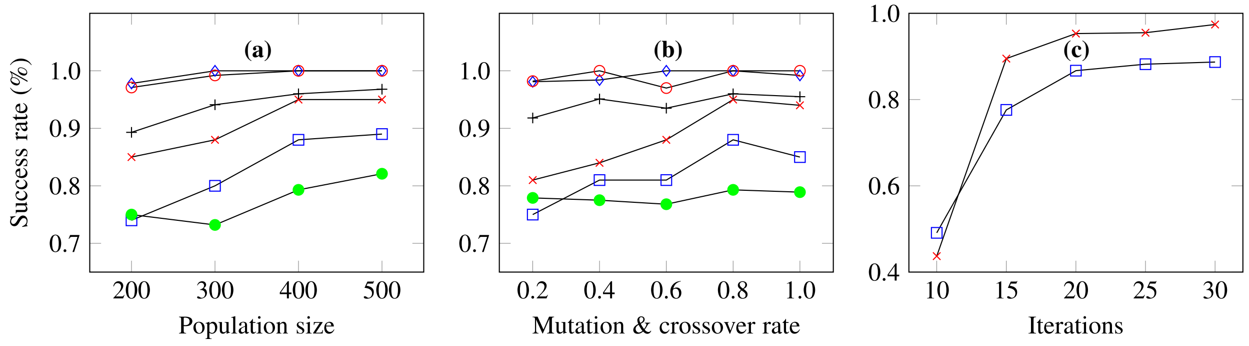


Fig. 3: Analysis of attack rate with different parameters including population size, mutation & crossover rate and iterations. (—×—) and (—□—) denote MRA2 injection and non-injection attack on the CNN-based malware detector. (—◇—) and (—●—) denote MRA2 injection and non-injection attack on the SVM-based malware detector. (—○—) and (—+—) denote MRA2 injection and non-injection attack on the RF-based malware detector.

The framework of the defense is shown in Fig. 2. In the phase of designing model, a well-designed layer called random translation layer is added behind the input layer to improve the robustness of the target model. The detailed workflow of the random translation layer is given below: To begin, a set of demarcation points are preset such as 0,1/4,1/2,3/4. we assume that a input with size of $h \times w$, the above demarcation points respectively represent reading from the input data at 0*h, 1/4*h, 1/2*h and 3/4*h. Then a random candidate τ from the above demarcation points are generated. After obtaining the value of τ, the random translation layer read the input from Row $\tau \times h$ to Row $h-1$ as the first part of the translated data, Row 0 to Row $\tau \times h - 1$ as the second part of the translated data. Finally, with the help of the random translation layer, the input is translated randomly to interfere with the process of AE generation. Adding this random translation layer can improve the robustness of the ML model, while the accuracy of the ML model will be reduced. To solve this problem, in the training phase, we utilize the random translation layer to generate all possible samples for each sample in the training set. The generated samples are supplemented to the training set. The augmented dataset will improve the accuracy of the ML model. Therefore, the ML model with our defence strategy deployed can exhibit good robustness and accuracy.

TABLE 4: The success rate of MRA2 with the proposed defence strategy.

Demarcation points	Injection MRA2	Non-injection MRA2
4	35.6%	18.8%
8	36.2%	15.5%
12	32.7%	8.5%

To evaluate whether the proposed defence strategy can defend against MRA2, we deploy it on Inception-v3 and then use MRA2 to attack the deployed model. Table 4 shows how the proposed defence strategy affects MRA2 on Inception-v3 in terms of the success rate. Demarcation points presents the number of the demarcation points we used, the larger the value of Demarcation points, the more translated samples may be generated.

From the results of Table 4, we can see the success rates of MRA2 decline sharply. Both of the injection attack and non-injection attack are invalid. For injection attack, even in the best case, only 36.2% of the generated adversarial examples can bypass the deployed model within 20 iterations. For non-injection attack, even in the best case, only 18.8% of the generated adversarial examples can bypass the deployed model within 25 iterations.

4.6 Comparisons with State-of-the-art Adversarial Attacks

In the experimental stage, we compare our MRA2 with the state-of-the-art adversarial attacks for malware detection [14, 15, 50]. We evaluate these attacks on three essential indicators, including function reservation rate, end-to-end attack pattern and accessing malware source code. The function reservation rate presents the proportion of the malware AEs that reserve the malicious function in all generated malware AEs. In this article, any adversarial attack with the following attributes will be considered as an end-to-end attack.

- *The original sample is a malware executable file.*
- *The generated adversarial examples can be executed directly without operations such as repackage, compiling and disassembly.*

Accessing the source code malware indicates whether the attacker require the access to the source code of the malware for the AE generation. In order to make a fair comparison, we implement all of these attacks in black-box scenario. We adopt the method in [17] to train DNN-based substitute models to fit the victim models. The generated AEs are then transferred to the victim model finally.

The experimental results are shown in Table 5. We can see that our MRA2 is more powerful and generalized than other attacks. The AEs generated by the other attacks are not executed in a controlled environment. Extensive theoretical analysis are included in Demontis's work [50] and Rosenberg's work [15], while the function of AE is not verified in a practical environment. Besides, the function reservation rate is not evaluated in their experiments. Compared with our MRA2, ATMPA [14] damages the function of AEs.

Moreover, adversarial examples generated by these state-of-the-art adversarial attacks can not be executed directly while AEs generated by MRA2 can be executed directly. In [50] and [20], operations such as repacking and recompiling are needed to construct the final executable malware. However, malware AEs generated by MRA2 can be executed directly which obviously improve the efficiency and the threat of the attack. The attack in [14] modifies all the content of malware samples to generate AEs which can not be executed directly. In addition, Demontiset al. [50] and Rosenberget al. [20] require the access to the source code of the malware for extracting features such as API calls, permissions and string, which needs extra tools (e.g. Drebin).

In this article, we also compare our MRA2 with the state-of-the-art attacks against Inception-, SVM- and RF-based detectors in black-box setting. To implement these state-of-the-art attacks in black box scenario, corresponding substitute models are trained on samples from VirusShare and kaggle [20].

In Rosenberg's work [15], the input of the ML-model is an API call sequence, so they used the popular sequence based malware classifiers, LSTM. To reimplement their work, LSTM model is also trained. For the state-of-the-art attacks, all of the generated AEs on DNN models are transfered to the SVM- and RF-based detectors to evaluate the success rate.

From Table 6, we can see the success rates of all the tested adversarial attacks are at the same level (all of these attacks can achieve at least 95% success rate) while targeting Inception- and LSTM-based malware detection models. As compared with the state-of-the-art attacks on SVM- and RF-based detectors, the MRA2 achieves the best performance even in the black-box scenario. MRA2 can reach 100% success rate while the success rates of state-of-the-art attacks are much lower than MRA2. Since MRA2 does not depend on the transferability of adversarial examples, it achieves a higher success rate on SVM- and RF-based detectors. Therefore, this result shows that adversarial examples generated by previous works for substitute models are not always transferable to victim models.

TABLE 5: Comparisons with different adversarial attacks against ML-based malware detection models.

Methods	Function reservation rate	End-to-end attack	Accessing malware source code
MRA2	100%	✔	✘
ATMPA [14]	0%	✘	✘
Demontis *et al.* [50]	unkown	✘	✔
Rosenberg *et al.* [15]	unkown	✘	✔

TABLE 6: Success rate of different adversarial attacks against ML-based malware detection models.

Attackers	Inception	SVM	RF
MRA2-inject	95.7%	100%	100%
MRA2-non-inject	88.7%	79.3%	96.0%
ATMPA [14]	96.5%	65.3%	71.8%
Demontis *et al.* [50]	98.0%	68.1%	85.4%
Rosenberg *et al.* [15]	97.3% (LSTM)	68.5%	68.2%

4.7 Influence of Model Parameters on MRA2

4.7.1 *Population size in GA*

The population size in GA is related to the search ability. The larger the population size is, the easier is to find the global optimal solution [19], with a higher solve time cost though.

Fig. 3a shows how the population size affect the success rate of MRA2. As the population size growing, the success rate of MRA2 injection and non-injection attacks are increasing accordingly. When the population size approaches to the largest factor tested, *i.e.* 500, the success rates of MRA2 injection and non-injection attacks become stable at 95.9% and 89.7% on the CNN-based malware detector. Success rate of MRA2 injection attacks on SVM- and RF-based malware detectors are 100%. Factors of MRA2 non-injection attacks on SVM- and RF-based detectors are 82.1% and 96.8%, respectively. The ecaluated three ML-based detectors are vulnerable to MRA2.

4.7.2 *Mutation & crossover rate*

Mutation & crossover operation can not only maintain the good genes in the previous generation, but also make the population rich in diversity and expand the search direction [59]. The research of [60] pointed out that the success rate of attack could not be improved by using single crossover operation.

We combined crossover and mutation operations in GA, which are controlled by mutation & crossover rate. Fig. 3b investigate how mutation & crossover rates affect the success rate of MRA2 on different ML-based malware detectors.

For all mutation & crossover rates used, the success rates of MRA2 ranges from 75% to 100%. The success rate of injection attack is higher than that of non-injection attack. The peak value is achieved when the mutation & crossover rate is 0.8.

4.7.3 *Maximum number of iterations*

The maximum number of iterations does not depend on the ML model, so we only consider attacks under Inception here. Since population size affects the time cost of each iteration, the maximum number of iterations affects the overall cost of MRA2. For our MRA2 injection and non-injection attacks, most AEs can be generated in 15 to 20 iterations. Use of 15 iterations allows injection attack to achieve 89.5% attack success rate and non-injection attack to achieve 77.6% , while factors related to 20 iterations grow to 95.3% and 86.7%, respectively. When MRA2 iterates more than 20 iterations, the attack success rate converges around 96% but consuming more time to complete. Similar results can be found in MRA2 non-injection attack when iterating over 25 iterations. The corresponding attack success rate converges around 88%.

5 Conclusion

In this paper, we propose an automated end-to-end black-box attack, named MRA2, where malicious function of malware samples are preserved and generated AEs can be executed directly without disassembly, repackage and compiling. A novel defense strategy is proposed to mitigate MRA2. Theoretical analysis on automatically identifying function-irrelevant segments of the executable malware is also presented. The detectors based on three types of machine learning algorithms are attacked by the MRA2. Experiments demonstrate an up-to 100% attack success rate. To evaluate the function reservation, an online sandbox test is used to demonstrate our proposal's 100% malware-function reservation.

In addition, the proposed MRA2 attack is robust against tested defensive strategies such as adversarial training, ensemble training and SafetyNet. However, MRA2 performs poorly under the model deployed with the proposed defense. Even in the best case, only 36.2% of the generated adversarial examples can bypass the deployed model. MRA2 is compared with the state-of-the-art adversarial attacks on ML-based malware detectors. MRA2 is superior to its competitors, since the generated AEs are function-reserved and can be executed directly without disassembly, repackage and compiling.

References

[1] AV-TEST, "https://www.av-test.org/en/statistics/malware/," 2020.

[2] Abrams, https://www.bleepingcomputer.com/news/security/computer-giant-acer-hit-by-50-million-ransomware-attack/, 2021.

[3] R. Moskovitch, C. Feher, and Y. Elovici, "A chronological evaluation of unknown malcode detection," 04 2009, pp. 112–117.

[4] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–40, 2017.

[5] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[6] X. Pei, L. Yu, and S. Tian, "Amalnet: A deep learning framework based on graph convolutional networks for malware detection," *Computers & Security*, vol. 93, p. 101792, 2020.

[7] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, 2021.

[8] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "Bitblaze: A new approach to computer security via binary analysis," in *International Conference on Information Systems Security*. Springer, 2008, pp. 1–25.

[9] J. Zhang and C. Li, "Adversarial Examples: Opportunities and Challenges," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[10] D. Li, Q. Li, Y. Ye, and S. Xu, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 736–750, 2021.

[11] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Black-box adversarial sample generation based on differential evolution," *Journal of Systems and Software*, vol. 170, p. 110767, 08 2020.

[12] *DeepMal: maliciousness-Preserving adversarial instruction learning against static malware detection*, 2021.

[13] A. N. Carey, H. Mai, J. Zhan, and A. Mehmood, "Adversarial attacks against image-based malware detection using autoencoders," in *Pattern Recognition and Tracking XXXII*, M. S. Alam, Ed., vol. 11735, International Society for Optics and Photonics. SPIE, 2021, pp. 48 – 56.

[14] X. Liu, J. Zhang, Y. Lin, and H. Li, "ATMPA: Attacking Machine Learning-based Malware Visualization Detection Methods via Adversarial Examples," *ACM Proceedings of the International Symposium on Quality of Service*, 2019.

[15] 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 *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 490–510.

[16] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial Examples for Malware Detection," in *European Symposium on Research in Computer Security*. Springer, 2017.

[17] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[18] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. Shevade, and V. Ganapathy, "Activethief: Model extraction using active learning and unannotated public data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 865–872.

[19] S. Das and P. N. Suganthan, "Differential Evolution: A survey of The State-of-the-art," *IEEE Transactions on Evolutionary Computation*, 2010.

[20] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, 2014.

[22] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[23] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 446–454.

[24] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu, "Combining file content and file relations for cloud based malware detection," 2011.

[25] X. Hu, "Large-scale malware analysis, detection, and signature generation," 01 2011.

[26] S. Sahay, A. Sharma, and H. Rathore, "Evolution of malware and its detection techniques," 07 2019.

[27] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," 2008.

[28] E. Raff and C. Nicholas, "An alternative to ncd for large sequences, lempel-ziv jaccard distance." New York, NY, USA: Association for Computing Machinery, 2017.

[29] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proceedings of the sixth ACM conference on data and application security and privacy*, 2016, pp. 183–194.

[30] J. Sexton, C. Storlie, and B. Anderson, "Subroutine based detection of apt malware," *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 4, pp. 225–233, 2016.

[31] P. Priyadarshan, P. Sarangi, A. Rath, and G. Panda, "Machine learning based improved malware detection schemes," in *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2021, pp. 925–931.

[32] E. Raff, W. Fleming, R. Zak, H. Anderson, B. Finlayson, C. Nicholas, and M. Mclean, "Kilograms: Very large n-grams for malware classification," 07 2019.

[33] P. Khodamoradi, M. Fazlali, F. Mardukhi, and M. Nosrati, "Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms," in *2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*. IEEE, 2015, pp. 1–6.

[34] Y. Ye, T. Li, Y. Chen, and Q. Jiang, "Automatic malware categorization using cluster ensemble," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 95–104.

[35] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of api sequences," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2014, pp. 2337–2342.

[36] G. Liang, J. Pang, and C. Dai, "A behavior-based malware variant classification technique," *International Journal of Information and Education Technology*, vol. 6, pp. 291–295, 2016.

[37] A. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *computers & security*, vol. 52, pp. 251–266, 2015.

[38] Z. Zhang, P. Qi, and W. Wang, "Dynamic malware analysis with feature engineering and feature learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 1210–1217, Apr. 2020.

[39] M. Graziano, D. Canali, L. Bilge, A. Lanzi, E. Shi, D. Balzarotti, M. van Dijk, M. Bailey, S. Devadas, M. Liu *et al.*, "Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 1057–1072.

[40] C.-T. Lin, N.-J. Wang, H. Xiao, and C. Eckert, "Feature selection and extraction for malware classification." *J. Inf. Sci. Eng.*, vol. 31, no. 3, pp. 965–992, 2015.

[41] W. Mao, Z. Cai, D. Towsley, and X. Guan, "Probabilistic inference on integrity for access behavior based malware detection," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 155–176.

[42] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic Analysis of Malware Behavior Using Machine Learning," *Journal of Computer Security*, 2011.

[43] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast Malware Classification by Automated Behavioral Graph Matching," in *ACM Annual Workshop on Cyber Security and Information Intelligence Research*, 2010.

[44] K. Berlin, D. Slater, and J. Saxe, "Malicious Behavior Detection Using Windows Audit Logs," in *ACM Proceedings of Artificial Intelligence and Security*, 2015.

[45] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware Analysis Using Visualized Images and Entropy Graphs," *International Journal of Information Security*, 2015.

[46] J. H. Go, T. Jan, M. Mohanty, O. P. Patel, D. Puthal, and M. Prasad, "Visualization approach for malware classification with resnext," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1–7.

[47] A. Makandar and A. Patrot, "Malware class recognition using image processing techniques," in *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*. IEEE, 2017, pp. 76–80.

[48] N. Šrndic and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*. Citeseer, 2013, pp. 1–16.

[49] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: color-inspired convolutional neural network (cnn)-based android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2633–2642.

[50] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[51] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables," in *European Signal Processing Conference*, 2018.

[52] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading Machine Learning Malware Detection," *Black Hat*, 2017.

[53] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, "Malware makeover: Breaking ml-based static analysis by modifying executable bytes," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21, 2021, p. 744–758.

[54] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2016.

[55] W. Hu and Y. Tan, "Generating Adversarial Malware Examples for Black-box Attacks Based on GAN," *arXiv preprint arXiv:1702.05983*, 2017.

[56] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," *arXiv preprint arXiv:1312.6199*, 2013.

[57] J. C. Hart, "Perlin noise pixel shaders," in *ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 2001.

[58] VirusTotal, "https://www.virustotal.com/gui/home/upload," 2020.

[59] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[60] J. Su, D. V. Vargas, and K. Sakurai, "Attacking Convolutional Neural Network Using Differential Evolution," *IPSJ Transactions on Computer Vision and Applications*, 2019.

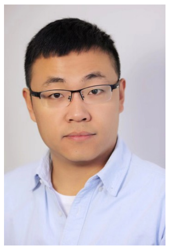
ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 62122023, U20A20202 and 61874042, the Hunan Science and Technology Innovation Leading Talents Project under Grant No. 2021RC4019 and the Hunan Natural Science Foundation for Distinguished Young Scholars under Grant No. 2020JJ2010, and the Natural Science Foundation of Fujian Province under Grant No. 2021J01544.

Jiliang Zhang received the Ph.D. degree in Computer Science and Technology from Hunan University, Changsha, China in 2015. From 2013 to 2014, he worked as a Research Scholar at the Maryland Embedded Systems and Hardware Security Lab, University of Maryland, College Park. From 2015 to 2017, he was an Associate Professor with Northeastern University, China. He is currently a Full Professor at Hunan University. He is the director of Chip Security Institute of Hunan University, and the Secretary-General of CCF Fault-Tolerant Computing Professional Committee. His current research interests include hardware security, hardware-assisted security, and microarchitecture security. He has authored more than 60 technical papers in leading journals and conferences. He is serving as a steering member for Hardware Security Forum of China and a Guest Editor of the IEEE Transactions on Circuits and Systems II: Express Briefs. He is a senior member of IEEE.

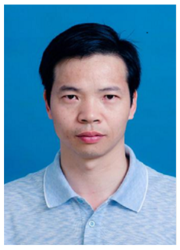
Zihao Luo is a graduate student at the College of Computer Science and Electronic Engineering, Hunan University, China. His current research interests include AI security and malware detection.

Honggang Yu is currently working toward the Ph.D. degree in information and communication engineering with Guilin University of Electronic Technology, Guilin, China. Since 2017, he works as a research scholar at the University of Florida, Gainesville, the United States. His research focuses on security and privacy problems in machine learning and mobile systems. His research work has been published at a range of top-tier conferences, including NDSS, AAAI, etc.



He Li is currently a Research Associate with Computer Lab, University of Cambridge. He got his PhD in the Department of Electrical and Electronic Engineering, Imperial College London. His research work has been published in leading journals and conferences, such as IEEE TC, TVLSI, DAC, ARITH and FPT, etc. He serves on the technical program committees of the top-tier EDA and reconfigurable computing conferences such as ICCAD, ASP-DAC, SOCC and FPT, and severed as a Review Editor for Frontiers in Electronics. Dr. Li is serving as the Publicity Co-chair at IEEE FPT 2020. He is a recipient of the best paper presentation award at FPT 2017.

Xin Zhang is a undergraduate student at the College of Computer Science and Electronic Engineering, Hunan University, China. His current research interests include AI security and malware detection.



Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He is currently a Cheung Kong professor of computer science and technology with Hunan University and the dean of the College of Computer Science and Electronic Engineering, Hunan University. He has authored or coauthored more than 250 research papers in international conferences and journals such as the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Cloud Computing, and the IEEE International Conference on Distributed Computing Systems. His major research interests include parallel and distributed processing, high-performance computing, and big data management. He is currently on the editorial board of the IEEE Transactions on Computers. He is an outstanding member of the CCF.

Yupeng Hu received the MS and PhD degrees in computer science from Hunan University, China, in 2005, and 2008, respectively. He is currently a professor in the College of Computer Science and Electronic Engineering at Hunan University. He is the dean of the Department of Cyberspace Security. He was with the Department of Computer Science and Engineering at UT-Arlington as a visiting scholar from 2015 to 2016. He was also with IBM China Development Laboratory as an academic visitor in 2012. His research interests include storage systems, erasure coding, and network and system security. He is an IEEE and ACM senior member.

Appendix A
Malware Visualization

Malware visualization methods transform the hard to-identify malware information into image. A specific segment of code represents a definite type of malware. In the visualization process, a set of malware samples are read as a vector of 8-bit unsigned integers. After that, these vectors are separated into a two-dimensional array. The corresponding value in the array can be expressed as the gray value of the image in the range from 0 to 255, where 0 indicates black and 255 means white. According to the different characteristics and analytical requirements for different data sets, the width of these transformed images could be appropriately adjusted as needed. Fig. 1 illustrates the flowchart of malware transformation process with a binary file from kaggle [1], where a common Trojan horse download program (Dontovo A) is converted to an 8-bit grayscale image with the width of 64. Code segments of .text, .rdata, .data and .rsrc corresponds to 8-bit grayscale images are shown in Fig. 1.

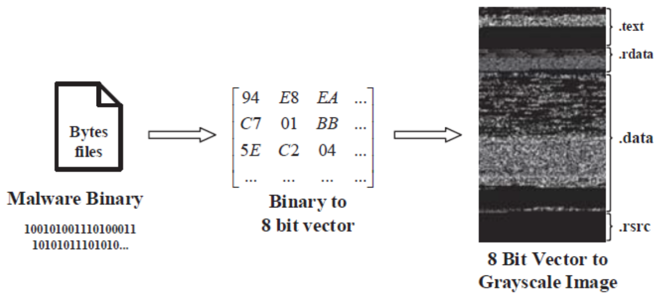
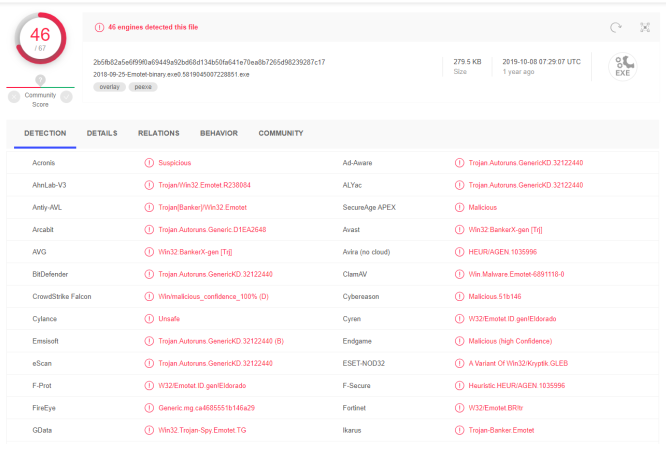


Fig. 1: Malware visualization transformation

Appendix B
Online Sandbox Test

VirusTotal[2] is a website created by the Spanish security company Hispasec Sistemas. VirusTotal aggregates many antivirus products and online scan engines to check for viruses that the user's own antivirus may have missed, or to verify against any false positives. Files up to 550 MB can be uploaded to the website. The report of a malware adversarial example in VirusTotal is shown in Fig. 2, the digit in the report indicates the number of the anti-malware engines that detect the malicious behavior during the running time of an executable sample. In this paper, if the behavior is detected by more than 10 anti-malware engines, the malicious function of generated AE is considered preserved. The function of all generated AEs can be preserved in our experiment.



46 engines detected this file

2b5fb82a5e6f99f0a69449a92bd68d134b50fa641e70ea8b7265d98239287c17
2018-09-25-Emotet-binary.exe0.5819045007228851.exe
overlay peexe

279.5 KB Size

2019-10-08 07:29:07 UTC 1 year ago

Acronis	Suspicious	Ad-Aware	Trojan.Autoruns.GenericKD.32122440
AhnLab-V3	Trojan/Win32.Emotet.R238084	ALYac	Trojan.Autoruns.GenericKD.32122440
Antiy-AVL	Trojan[Banker]/Win32.Emotet	SecureAge APEX	Malicious
Arcabit	Trojan.Autoruns.Generic.D1EA2648	Avast	Win32:BankerX-gen [Trj]
AVG	Win32:BankerX-gen [Trj]	Avira (no cloud)	HEUR/AGEN.1035996
BitDefender	Trojan.Autoruns.GenericKD.32122440	ClamAV	Win.Malware.Emotet-6891118-0
CrowdStrike Falcon	Win/malicious_confidence_100% (D)	Cybereason	Malicious.51b146
Cylance	Unsafe	Cyren	W32/Emotet.ID.gen!Eldorado
Emsisoft	Trojan.Autoruns.GenericKD.32122440 (B)	Endgame	Malicious (high Confidence)
eScan	Trojan.Autoruns.GenericKD.32122440	ESET-NOD32	A Variant Of Win32/Kryptik.GLEB
F-Prot	W32/Emotet.ID.gen!Eldorado	F-Secure	Heuristic.HEUR/AGEN.1035996
FireEye	Generic.mg.ca4685551b146a29	Fortinet	W32/Emotet.BR!tr
GData	Win32.Trojan-Spy.Emotet.TG	Ikarus	Trojan-Banker.Emotet

Fig. 2: Malware visualization transformation

References

[1] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[2] VirusTotal, "https://www.virustotal.com/gui/home/upload," 2020.