Effective Adversarial Malware Generation and Verification Framework based on Heuristic Manipulation of Executable Binaries.

Jui-Lung Hung[1], Chi-Wei Chen[1], Pei-Yu Lo[1],
Chin-Wei Tien[1], and Sy-Yen Kuo[1]

Department of electrical engineering, National Taiwan University, Taiwan
{r08921a02,r08921a28,r08921a29,d99921020,sykuo}@ntu.edu.tw

Abstract. As the application market of Artificial Intelligence has grown rapidly in recent years, application developers started to apply AI techniques in the security field like malware detectors based on Deep Learning. But in general, application developers nowadays usually ignore the security threats to AI models, that is, adversarial attacks. The adversarial attack is an effective way to compromise Artificial Intelligence models by slightly perturbing the original sample with a certain algorithm, and it is very powerful in the computer vision and speech field. However, in the binary field, most of the slightly perturbed binaries are either inexecutable or non-functional. We have searched many recent works of adversarial malware and found that most of them only focused on the process of generating adversarial examples. The adversarial examples they generated only bypassed their own pre-trained malware detector without any executability or functionality verification, and most of them are prone to be detected by traditional antivirus scanning. The lack of a complete framework, functionality verification, and convincing test objects makes it difficult to reproduce and evaluate in the real world. Thus, in this paper, we have developed a practical framework for adversarial malware generation and testing with a weight-based method to manipulate executable binaries, testing malware functions by sandbox system, and using more than 60 commercial and machine-learning-based antivirus for real-world evaluation.

Keywords: Artificial Intelligence · Adversarial Example · Malware Detection.

1 Introduction

Malware attacks are still the main method of hacker attacks. There were at least 7.2 billion attacks and 151.9 million ransomware attacks reported through the first three quarters of 2019. Hence, malware detection is one of the main research directions in the current computer security field.

In order to cope with a large number of new types of malware attacks, traditional signature-based detection is no longer sufficient. Since more and more

researches have shown that machine-learning-based detectors can effectively improve the accuracy of malware detection, more and more antivirus software companies are dedicated to integrating machine learning technologies into their products to prevent new types of malware attacks. However, with adversarial attacks, attackers can easily generate new malware to fool the detectors by adding trivial perturbation on the original malware.

Unfortunately, even if the adversarial attack on malware is seen to be a potentially serious threat in the computer security field, there are only 35 papers focused on binary files among more than 1,200 papers related to adversarial attacks in recent years. In the severe lack of research energy, this field's development is trapped, not only for the attackers but also for the defenders. In the past research, we have doubted some of their results. First, most of the papers only focused on attacking their own pre-trained models for malware detection and did not actually attack commercial antivirus or machine-learning-based antivirus in the real world. Secondly, even if the adversarial sample can pass the static detector, trivial changes to the binary file may significantly affect the functionality of the binary file. That is, it would have a high risk of losing its function. Therefore, an adversarial force attack bypassing the machine learning model without verifying the functionality seems not feasible for real-world applications. Unlike other papers, to make the adversarial attacks on binary files more realistic, we used Virustotal containing about 60 antiviruses, to conduct attacks and used Cuckoo Sandbox[4] to verify its functionality to ensure the virus is real and feasible.

The paper is structured as follows. In Sect. 2 , we discuss related works in this field. In Sect. 3, we proposed an adversarial example generation method and a verification method with the cuckoo sandbox system. In Sect. 4, we demonstrate the experimental results. In Sect. 5, we have a conclusion and future works of this paper. The main contributions of this paper are as follows:

- Design a framework to generate and evaluate adversarial malware example based on [8] with more feasible and real-world settings.
- A score function to evaluate the quality of adversarial malware and adaptive weight-based method to choose optimized binary manipulation method.
- Functionality verification with sandbox system and preliminary definition of valid adversarial example on the binary field.
- Systematic evaluation of the state-of-the-art binary manipulation and the generated adversarial malware.

2 Related Works

2.1 Adversarial Example

Machine Learning models can often achieve high classification accuracy on most datasets nowadays but are easily fooled by small input disturbances. Researchers identified this phenomenon as adversarial examples. The earliest adversarial attack is proposed by Szegedy et al. [9]. Szegedy found that slightly perturbation added on the input lead to significant misclassification of the image recognition models trained on ImageNet architecture. This perturbation is insignificant and cannot be detected by the naked eye, but because AI technology is increasingly integrated into our daily lives, it has become a serious security issue. Therefore, many researchers began to design and implement adversarial attacks to cheat AI examples as white hat tests.

Adversarial attacks can be roughly divided into two types: white box and black box. In the white box setting, the adversary can obtain any information about the training data set distribution or full access to the training model, including architecture and layer information. As [9], authors design a Fast Gradient Sign Method which we only need to disturb the original image with one step gradient sign noise from backpropagation of model. Or in [10], authors proposed a mathematic algorithm DeepFool based on gradient sign which can effectively cheat the model with less image noise addition. In contrast, in a black box setting, adversaries usually have no idea what the target model is or how the model is training. As in [11], authors proposed a concept of substitute network, which is a model trained with the black box model's classification result and the classification boundary of substitute network is assumed to be closed to the black box oracle. Then, we can get the gradient information of the substitute network and transform the white box attack on the black box oracle. Except for substitute network, [12] used gradient estimation method for input optimization, which can effectively estimate the gradient information through only the input-output pairs of the model even if they do not have access to any gradient information of the model. Although there are many useful adversarial attack methods proposed not only in the image field but in the speech field, there are some obstacles and problems that we would like to deal with in the binary field.

2.2 Adversarial Malware

In this section, we will introduce the papers we referenced in this research. Most research have demonstrated that trivial changes have the ability to fool machine-learning-based detectors [13–41]. However, there are some defects that we found in some of those paper. We will introduce them and address the problem in the adversarial malware field that we would like to solve in this paper. We simply conclude three main problems of those paper :

1. **research that did not generate real binary files**

Some of the researches only discuss the manipulation of binary on the abstract layer. In [13], authors investigated methods that can reduce the adversarial blind spots for neural network malware detectors. They approached this as

a saddle-point optimization problem in the binary domain to train DNNs via multiple inner maximization methods that are robust to adversarial malware. Authors of [16] present an attack model (named EvnAttack) to assess the security of the classifier. They collect the real-world sample from Comodo Cloud Security Center, containing 10,000 samples, and then build a practical solution for evasion attacks. As [13, 16, 28] seems to be reasonable for the design of the attack or defend method, they focused on implementing attacks or solving the optimization problem on self-defined feature space and did not actually generate realistic executable binary files. Hence, the functionality and practicality of these adversarial examples cannot be guaranteed.

2. **research that has an unrealistic or infeasible setting for adversarial malware generation**

Some of the researchers designed and conducted the white box attacks on malware. That is, the adversary not only has full information of either training dataset or model architecture but also can access the whole model and training dataset. It is unrealistic since there is no chance for this kind of attack in the real world. In [13, 16, 42, 24, 43, 31, 39], they performed the white box attacks. However, in the real world, there is no chance for a white box attack. Even if [14, 20, 36, 37] performed the black box attacks, some assume that the attackers know the target model, and some assume that attackers can get the feedback or the confidence score from the target model. Either of them is unrealistic. Next, we introduce the contribution of some of these papers in brief:

3. **research that lacks rigorous verification process**

Some of the research only focused on generating the perturbed adversarial malware but did not provide a rigorous verification process. In [14], authors performed a generic black box attack on a static PE malware detector, and their attack is based on reinforcement learning. This paper provides an action list that can effectively manipulate the PE file and these actions used by many other papers. However, even if it provided an excellent action framework to utilize with gym, they lack to verify the generated binaries and discuss the influence of the action on malware samples. In [17], authors use integrated gradients to explain the results achieved by MalConv. They devise an algorithm inspired by other papers and show that change a few bytes is enough to evade MalConv. But they did not discuss and verify the functionality of malware with appending bytes and the feasibility of appending obvious bytes on the file, which only bypassing machine-learning-based MalConv.

The problems we mention above make it hard to reproduce and verify the adversarial malware. This is a severe problem that we did not have any specification or framework in this research field.Therefore, in order to deal with the above-mentioned obstacles that researchers may encounter in the field of adversarial malware, this paper not only proposes a generation framework with an effective scoring function and weighted based action selector but gives a rigorous verification with a preliminary definition of valid adversarial malware as a simple framework for adversarial malware generation. Moreover, we will discuss how perturbation affects malware in section 5.

3 Methodology

3.1 Adversarial Malware Generation

In this section, we will introduce our adversarial example generation method in detail.

The architecture of the adversarial example generation method for executable binaries is shown as Fig. 1. We first define the malware sample in different generation phase as:

- Malware Samples (denoted as MS)
- Evasive Samples (denoted as ES)
- Detected Samples (denoted as DS)
- Adversarial Samples (denoted as AS)

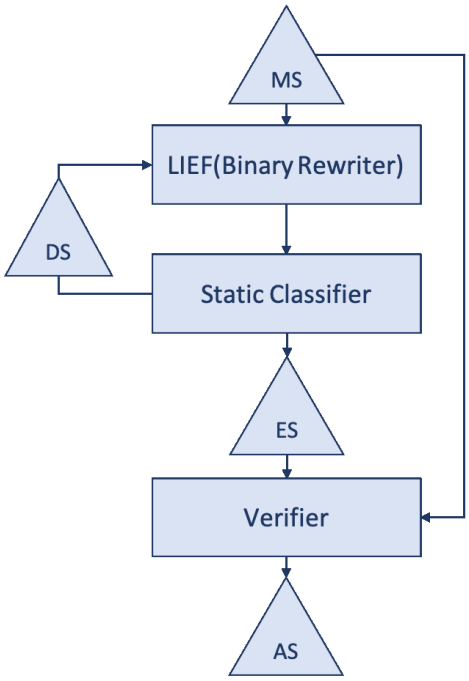

Fig. 1. Architecture of Adversarial Malware Generation System.

The proposed architecture is mainly composed of three components:

- Binaries Manipulation Module
- Static Classifier Module
- Sandbox Verification Module.

The binaries manipulation module consists of a weighted action selector and a LIEF Binary Rewriter. We first choose the manipulation methodologies to manipulate malware samples with a weighted action selector, which we adaptively adjust the weight of each action. If the weight of action is higher, the selector has a higher probability of choosing that better action. As long as the action is selected, we apply the binary manipulation action on the original malware sample with LIEF binary rewriter and generate the intermediate sample.

The static classifier module comprises VirusTotal antivirus scanning engines and a machine-learning-based decision tree that we retrained based on the instruction of [5]. We send the intermediate samples generated by the binaries manipulation module into the static classifier module and design a balanced effectiveness evaluation method to judge if an intermediate sample is an evasive sample or a detected sample. If the intermediate sample is classified as a detected sample, the system would consider the applied action as a bad action to choose. Therefore, we also design a weight adjustment module in this module to adjust the weight of actions in the weighted action selector mentioned above.

The sandbox verification module includes the cuckoo sandbox system and a log comparison module. We use the cuckoo sandbox system as our sandbox verification system in the experiment. By sending the evasive sample and original malware sample into the sandbox system, we can check and verify the malware functionality through dynamic analysis. Moreover, due to the large data to be coped with, we design a log comparison module that can automatically compare the cuckoo log and behavior log of evasive sample and original one and generate the Jaccard similarity and cuckoo score difference as an indicator of the functional similarity of the two. Eventually, we would use a filter to filter out the malfunctioned sample and defined the remained evasive sample as an adversarial sample in malware.

The complete procedure is summarized in the algorithm below:

1. Input Definition: original samples s_0, iteration limit i_0
2. Module Definition: LIEF module L, ViusTotal online classifier V, Machine learning based detector trained by EMBER dataset E, Scoring module S, Weight adjustment module W, Cuckoo Sandbox C
3. Output Definition: adversarial example AS
4. First, choose original malware sample s_0 as evasive sample $s'$to be manipulated and set the iteration i as 0

$$s' \leftarrow s_0 \tag{1}$$

$$i \leftarrow 0 \tag{2}$$

5. Secondly, choose an action a from action list with action weights where weights consists of ($w(a1)$, $w(a2)$ ... $w(a9)$)

$$a \leftarrow random\,choice(action\,list, action\,weights) \tag{3}$$

6. Next, apply the chosen action a on evasive sample with LIEF module

$$s' \leftarrow L(s', a) \tag{4}$$

7. Then, we send the modified evasive sample to VirusTotal online classifier and EMBER decision tree [5] respectively to get the feedbacks f_v, f_e

$$f_v \leftarrow V(s') \tag{5}$$

$$f_e \leftarrow E(s') \tag{6}$$

8. Send both feedbacks f_v and f_e into scoring module S with balance weight b which used for balancing the influences of traditional antivirus scanning and machine learning based malware detection to get *score* for evaluation of the evasive sample s', and then adjust weight by weight module W with *score*.

$$score \leftarrow S(f_v, f_e, b) \tag{7}$$

$$w(a') \leftarrow W(w(a), score) \tag{8}$$

9. If the score is better than original score, increase the iteration and check if the iteration reach the limit i_0. If not, make the generated evasive sample as s' in Step 4. and go back to Step 5. Otherwise, save the valid evasive sample s' for verification.

$$i \leftarrow i + 1 \tag{9}$$

10. Eventually, send the evasive s' and original malware sample s_0 respectively into cuckoo sandbox system for functionality verification.

$$functionality\ verification \leftarrow C(s', s_0) \tag{10}$$

3.2 Definition of Effectiveness Score and Weight Adjustment

We chose the manipulation methodology to be utilized on the original malware sample from the Binary Obfuscation Action List in Weighted Binaries Manipulation Module. The Binary Obfuscation Action List is shown in Table 1. As for each action, a has a weight, says $w(a)$, which indicates its effectiveness. An action with a higher weight would have a higher probability of being chosen.

In this paper, in the static classifier module, we scanned the manipulated samples with sixty-eight mainstream signature-based antivirus engines on VirusTotal, and the machine-learning-based decision tree detector reproduced with the instruction of EMBER[5] endgame team. Based on the detection result, we calculated a score indicating the effectiveness of the applied manipulation methodology. The evaluation formula is as follows:

$$score = 1 - \frac{number\ of\ detected\ engines}{number\ of\ all\ engines} \tag{11}$$

$$score = 1 - (\frac{|detected\ antivirus\ engines|}{|antivirus\ engines|} + b * \frac{|detected\ ML\ detectors|}{|ML\ detectors|}) \tag{12}$$

A higher score means more engines successfully fools by the evasive sample. Moreover, it can be seen as an indicator of how effective the applied manipulation

Table 1. Binary Obfuscation Action List

#	Description
1	Add a function to the import address table that is never used
2	Manipulate existing section names
3	Create a new (unused) sections
4	Append bytes to extra space at the end of sections
5	Create a new entry point which immediately jumps to the original entry point
6	Manipulate (break) signature
7	Manipulate debug info
8	Modify (break) header checksum
9	Append bytes to the overlay (end of PE file)

methodology is. As long as the score of evasive sample is generated, we changed the weight of the selected action as formula 13, where the weight of certain action is denoted as $w(a)$, ϵ means the step size adjusting the weight, and s' and s_0 represent the score of generated evasive sample and original sample respectively. Then, we sent the evasive sample back to the manipulation module for the next iteration. Lastly, we output the evasive sample with the best manipulation methodology after ten iterations.

$$w'(a) = w(a) + \epsilon * (s' - s_0) \tag{13}$$

In the formula 11, we make our score converge between 0 to 1. Therefore, while we adjust the weight of actions as formula 13, the score of the new evasive sample s' could be restricted to 0 to 1. In other words, we could prevent our weights from rapidly being dominated by certain action a' due to drastic score deviation with only a few evasive samples. Also, with the deviation of $s' - s_0$, we could clearly see how the chosen action affects the original sample and the difference of score evaluation on the new evasive sample and the original one. Eventually, we adjust the weight of the action with ϵ as step size and $s' - s_0$ as effectiveness indicator of the generated evasive sample. Moreover, as research to bypass both traditional antivirus engine and machine-learning-based detector, we give our machine-learning-based decision tree more weight as scanning engine and detected engine in formula 11 to balance two significantly different scanning method.

3.3 Functionality Verification

To make the use of adversarial attacks on binary files more realistic, we used Cuckoo Sandbox[4] to verify the functionality of binaries to ensure that the virus is real and feasible. The sandbox performed a dynamic test on the binary to analyze their behavior. First, we need to check the generated evasive sample is still executable in the sandbox system. Therefore, We track the cuckoo behavior logs generated by the Cuckoo Sandbox analysis of the evasive sample and

the corresponding original malware sample. If the log contains some ̈failed to execute ̈message, our system may automatically record the information of the evasive sample and terminate to verify the next sample. If the evasive malware sample and corresponding original one are still runnable in the sandbox system, we would compare the Jaccard similarity of the cuckoo behavior logs. The definition of Jaccard similarity is as follows:

$$J(X,Y) = \frac{X \cap Y}{X \cup Y} \tag{14}$$

We can get the intersection ratio to the union of two logs as an indication of the behavior similarity of the two files. Therefore, if there's a considerable disparity between the two logs, it indicates that the behavior of the evasive sample is further different from the behavior of the original one. Afterward, we would compare the cuckoo score of the two files given by the Cuckoo Sandbox after dynamic analysis of the executable file. Lastly, the designed filter would filter out the evasive samples with low Jaccard similarity, and a large difference of cuckoo score corresponding to the original malware samples as Formula 15 where the Jaccard similarity is bounded to 0 to 1 and Cuckoo score difference is mostly bounded to 0 to 15. Based on our experimental observation, we set those evasive samples in the threshold of Formula 15 as adversarial examples. These generated adversarial examples should be real and feasible in the real world.

$$\{AS\} = \{ES | J(ES, MS) > 0.8 \cap C(ES, MS) < 1.0\} \tag{15}$$

To sum up, in this section, in order to cope with the scarcity of specification or framework in adversarial malware generation research, we have improved the architecture of [8] and proposed a rigorous generation framework consisting of adversarial malware generation module with effective scoring function and weighted based action selector and verification module with two adversarial malware evaluation techniques.

4 Experiment Setting

4.1 Dataset

VirusTotal aggregates many antivirus products and online scan engines to detect and classify viruses. Moreover, users can not only upload suspect URLs or files but download them from the VirusTotal database through VirusTotal API. It is widely used by the research community for data labeling or system evaluation. We collected 20000 PE files from 2017 to 2020 from VirusTotal as our dataset.

4.2 LIEF

To apply the above manipulation functions in Table 1, we use the open-source project named LIEF [2]. This project aims to provide a cross-platform library that can parse, modify, and abstract ELF, PE, and MachO formats. Therefore,

with it, we can parse the original malware sample into a high-level feature layer, modify the content of the abstract file, and rebuild the modified abstract file as a new PE executable binary. We utilize the weighted action selector to choose the action and automatically run the whole generation and verification process with our framework.

4.3 EMBER

EMBER is an open-source dataset that collects features from PE files that serve as a benchmark dataset for researchers. EMBER2017 is composed of features from 1.1 million PE files scanned before 2017, and the EMBER2018 consists of features from 1 million PE files scanned before 2018. We can easily reproduce the benchmark models with the EMBER dataset. In our experiment, we adopt EMBER2018 v2 as the training feature dataset and follow the training instruction provided by [5] to reproduce a decision tree as our machine-learning-based malware detector.

4.4 Cuckoo Sandbox

Cuckoo Sandbox is an open-source automated malware analysis sandbox system. We can throw suspicious files into it, and Cuckoo will provide a detailed report outlining the behavior of the file when executed inside a realistic but isolated environment.

4.5 Experiment Setup

Our experiments are performed on a desktop computer with Intel R CoreTM i7-9700 CPU @ 3.00 GHz, 48.0 GB RAM, 512GB SSD, Nvidia GeForce RTX 2060 GPU, and Ubuntu 20.04 LTS. We roughly divided the experiment into two parts, the generation part and the verification part.

In the generation part, the experiments include binary manipulation with LIEF, machine-learning-based malware detection, uploading and evaluating evasive samples with VirusTotal, scoring module analysis, and weight module adjustment, are all implemented in Python 3.6.8. In this part, to get a more realistic and feasible detection result than [8], we applied VirusTotal API to get fully scanning based on 68 different commercial engines for black box testing. Moreover, we evaluated the scores of antivirus scanning and machine learning detection method. We considered the balance of them with the balance weight in order to prevent the weight adjustment from being dominated by only antivirus scanning results.

In the verification part, we sent the generated evasive samples of the generation part into the cuckoo sandbox system for dynamic analysis combined with VMCloak, a high-level virtual machine manager, which allows us to easily launch, install, and reproduce multiple virtual machines. The virtual machine we adopted is Oracle VirtualBox 6.0 with Windows 7 Service Pack 1 Ultimate

Edition, and we disable the network connection in order to ensure the malware would not infect the host system. After cuckoo sandbox analysis, we track and parse the behavior log, compare the cuckoo score and Jaccard similarity, and filter out the malfunctioned evasive samples.

5 Experimental Result

5.1 Adversarial Example Generation

To make adversarial attacks on binary files more realistic, different from other generation algorithms, we consider the functionality a significant characteristic of the adversaries we generated. Before verifying the functionality of generated evasive samples, we would first check the executability of them. In our experiment, we sampled 1000 malware samples as original malware samples from our collected malware dataset and set the iteration limit as 1 in our algorithm, which can be seen as a one-step perturbation addition. In our work, we first sent the original samples into the binary manipulation module with LIEF, scoring module, and action weight adjustment module. Afterward, we got the perturbed evasive samples and check if they are still executable in the system. The executable result is shown as Fig 2. Among the 1000 manipulated malware samples, we can see that about 784 samples successfully be manipulated as evasive samples and the other 216 samples be filtered out as detected samples. With the executability testing, we found that 82.5% of evasive samples can be executed in the sandbox system, and about 15.7% of them lose executability after perturbation.

After examining the executability of evasive samples, we only remained the evasive samples that both original malware and evasive sample are executable. Then, we sent the remained samples into the log comparison module to get the Jaccard similarity and Cuckoo score difference of the evasive sample and corresponding original malware sample. The amount of valid adversarial examples with different thresholds is indicated in Fig 3 and Fig 4. The Jaccard similarity represents the similarity of dynamic behavior between the evasive sample and the original one. On the other side, the Cuckoo score is the score that Cuckoo Sandbox provided by evaluating a file's malicious behavior according to the virus signature provided by the Cuckoo Community. While the Cuckoo score difference is more closed to zero, the functional similarity of generated perturbed malware would be more closed to the original one. Therefore, through the evaluation of Jaccard similarity and Cuckoo score difference, we design a threshold filter with Formula 15 that filters out the so-called malfunctioned or different-functioned evasive samples. Moreover, for those executable evasive samples with a large score difference, we summarized two possible reasons: they can evade the dynamic detection analyzer and get a low Cuckoo score, or the manipulation caused the program to behave differently from before. Eventually, the proposed system generates 318 adversarial examples. That is, about 49% of the evasive samples sent into last functionality verification can remain the malicious functionality and bypass more static classifiers in the real world.

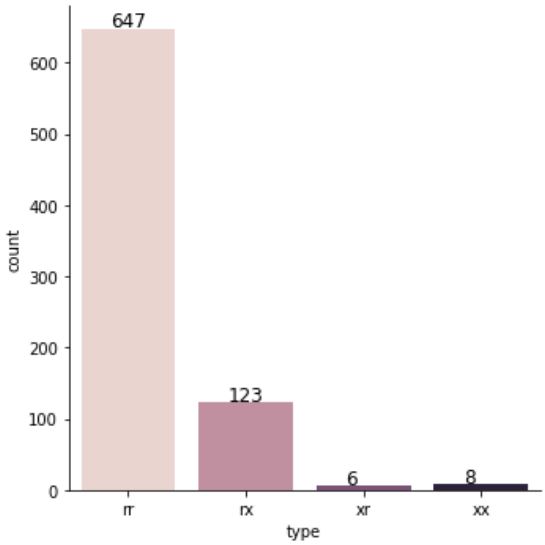

Fig. 2. Executable Amount of Generated Evasive Sample according to Executability of Original and Evasive Malware Sample. Type rr means that both the original and evasive sample is runnable. Type rx means only the original sample is executable. Type xr means only the perturbed evasive sample is runnable. And type xx means neither of the original and the evasive sample is executable.

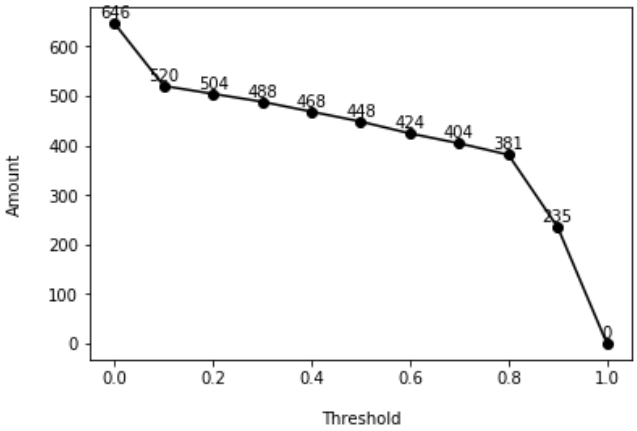

Fig. 3. Amount of Adversarial Example with Different Jaccard Similarity Threshold

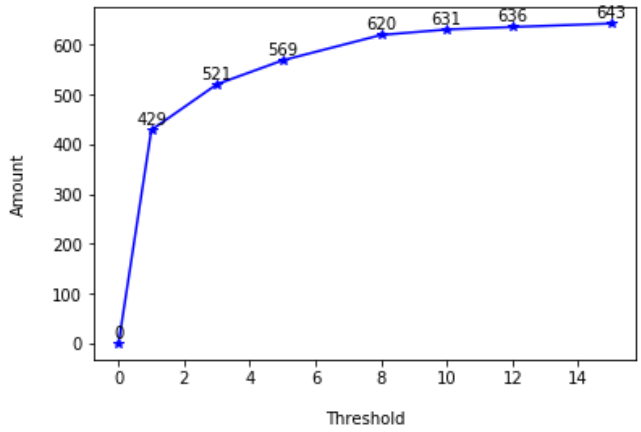

Fig. 4. Amount of Adversarial Example with Different Cuckoo Score Difference Threshold

Furthermore, we observed the average effectiveness score, which we define in Section 3.2 of the evasive executable samples and generated adversarial malware samples. Since we only select the perturbed malware, which has an effectiveness score better than the original malware sample, as an evasive sample, we can ensure that the evasive sample always has the ability to bypass more static classifiers. Moreover, the difference of the scores is an indicator of how strong that the applied action would help the perturbed malware evade the static classifier module. The average effectiveness score difference is 0.1189 and 0.0930, respectively. If we only considered the Formula 11 and did not consider the balanced weight between traditional antivirus scanning and decision tree trained with EMBER, we could assume x engines being bypass more than original malware as Formula 17. Then, we substitute 0.1189 and 0.0930 into effectiveness scores in Formula 17. As a result, the average number of engines that the executable evasive samples and generated adversarial malware examples can bypass more than original's are 8.2 and 6.4, respectively. This is a considerable improvement with only one-step action perturbation. Therefore, we can guarantee that the adversarial examples we generated have not only extremely similar functionality with corresponding original malware but also a more potent ability to bypass more static classifiers. Besides, the completed generation and verification method that we implemented in this system could be an exceptional reference for future research on the adversarial malware field.

$$effectiveness\ score = 1 - \frac{69 - x}{69} \tag{16}$$

$$effectiveness\ score\ diff = \frac{x' - x_0}{69} \tag{17}$$

5.2 Manipulation Action Evaluation

In this section, we focus on the influence of 9 mainstream actions we used to manipulate the malware binaries. First, we discuss the relationship between the applied action and executability with Table 2. In Table 2, we notice that there is a huge difference of *section add* and *create new entry*. As *create new entry* has 119 evasive samples, we get the information that it is a good action to utilize while only bypassing static classifier without any executability and functionality verification. This may be the blind point of the related work on adversarial malware generation. However, with the proposed verification process, we could consider *section add* and *create new entry* as trapped actions to apply during adversarial malware generation.

To find the most effective action to apply, we analyzed the adversarial samples generated by the proposed system. Among the 318 adversarial samples, the proportion of amount and mean effectiveness score difference with different manipulation actions are shown in Table 3. With Table 3, we see that even the valid adversarial samples mostly consist of *section rename* but the average effectiveness score of *section rename* is 0.078058, which is lower than the average effectiveness score of all adversarial samples. On the other side, even if the *create new entry* only accounts for 1.25 % due to the executability verification process, the action has an obvious influence on cheating the static classifier with a 0.262680 score difference. In conclusion, there is no effective actions to apply but only the suitable action like *section rename*, which can slightly perturb the malware without significantly change the functionality.

Table 2. Executable Amount and Amount of Evasive Sample for each Applied Action

Applied Action	Amount of Evasive Samples	Executable Amount of Evasive Samples
overlay append	62	61
imports append	249	246
section rename	189	186
section add	47	7
section append	27	26
create new entry	119	33
remove signature	5	5
remove debug	12	11
break optional header checksum	74	72

6 Conclusion

In this paper, we propose an effective and reliable system to generate the adversarial malware and a feasible evaluation framework to examine and verify the generated sample. Different from other papers, we used LIEF[2] to manipulate the PE files with 68 mainstream antivirus engines, proposed a balanced way

Table 3. Proportion and Average Effectiveness Score Difference of Adversarial Malware Sample with Different Action

Applied Action	Proportion of AS	Avg. Score of AS
overlay append	0.179245	0.053111
imports append	0.154088	0.142038
section rename	0.437107	0.078058
section add	0.006289	0.064356
section append	0.059748	0.072335
create new entry	0.012579	0.262680
remove signature	0.003145	0.234423
remove debug	0.025157	0.094344
break optional header checksum	0.122642	0.132958

to adjust the action weight between antivirus scanning and machine-learning-based malware detection, and used the Cuckoo Sandbox[4] to verify the PE file's malware functionality with our proposed evaluation method and ensure the adversaries are practical and feasible in the real world. Furthermore, we provide a CSV file of our experimental results with full description in [1].

Our future works consist of three directions. First, to further improve the efficiency of the adversarial example generator, we will continue improving the binary obfuscation algorithm and scoring method to generate more adversarial malware examples in the same period of time. Secondly, we would design a more feasible verification and evaluation of adversarial malware by not only use cuckoo sandbox but other dynamic analysis technologies. Then, we plan to develop a new malware detector with adversaries we generated. We will compare the result with the well-known malware detector in related fields to see if it could provide higher reliability than current detectors against adversarial examples. We will compare the result of the state-of-the-art malware detectors with and without this detection mechanism to see to what extent it can improve the model's robustness. We hope that our research can solve real problems and improve the reliability of the marching-learning-based models and reduce the possibility of them being attacked by malware.

7 Acknowledgement

This research was partially supported by VirusTotal. We thank colleagues from VirusTotal who provided partial malware samples and VirusTotal APIs that greatly assisted the research.

References

1. Anonymous Github of the Experimental Result.
https://anonymous.4open.science/r/915575c7-6e2c-4132-8c88-cbc8050734cd/

Table 4. summary of practical adversarial example algorithms

Attack	Target Model	Transformation	Approach
GADGET[38]	Ensemble of custom machine learning and deep learning models	add API calls	gradient-driven
Kolosnjaji et. al.[7]	MalConv	Edit padding bytes	gradient-driven
Kreuk et. al.[27]	MalConv	Edit padding bytes	gradient-driven
Demetrio et. al.[18]	MalConv	Edit PE header bytes	gradient-driven
Park et. al.[42]	Custom CNNs and Gradient boosting decision trees	semantic NOP insertion	gradient-driven
Anderson et. al.[15]	Gradient boosting decision trees	Edit byte features and PE metadata	problem-driven
Song et. al.[9]	Commercial antivirus	Edit byte features and PE metadata	problem-driven

Table 5. summary of practical adversarial example algorithms

Attack	Anderson et. al.[15]	Song et. al.[9]	Ours
Target Model	Gradient boosting decision trees	Commercial Antivirus	Multiple Commercial Antivirus and Gradient boosting decision trees
Transformation	Edit PE metadata	Edit PE metadata	Edit PE metadata
Unique Technique	Reinforcement Learning Agent	Instruction Substitution	Scoring Module and Weighted Selection
Action Select	RL Agent	Apply All Actions	Weighted Selection
Speed	Slow for Training	Slow	Fast
Attack Accuracy	High	High	Medium

2. LIEF. [Online]. Available: https://lief.quarkslab.com/

3. Libinear. [Online]. Available: https://www.csie.ntu.edu.tw/ cjlin/liblinear/

4. Cuckoo Sandbox. [Online]. Available: https://cuckoosandbox.org/

5. Hyrum S Anderson and Phil Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint arXiv:1804.04637

6. Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, Fabio Roli. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. arXiv preprint arXiv:1803.04173, 2018.

7. Octavian Suciu, Scott E. Coull, Jeffrey Johns. Exploring Adversarial Examples in Malware Detection. arXiv preprint arXiv:1810.08280, 2018.

8. Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Automatic Generation of Adversarial Examples for Interpreting Malware Classifiers. arXiv:2003.03100, 2020.

9. Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy. Explaining and Harnessing Adversarial Examples. arXiv preprint arXiv:1412.6572

10. Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard. DeepFool: a simple and accurate method to fool deep neural networks. Conference on Computer Vision and Pattern Recognition(CVPR), 2016

11. Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, Ananthram Swami. Practical black box Attacks against Machine Learning. arXiv preprint arXiv:1602.02697

12. Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, Cho-Jui Hsieh. ZOO: Zeroth Order Optimization Based black box Attacks to Deep Neural Networks without Training Substitute Models. AISec '17: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 2017

13. Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In 2018 IEEE Security and Privacy Workshops (SPW), pages 76–82. IEEE, 2018.

14. Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. arXiv preprint arXiv:1801.08917, 2018.

15. Li Chen. Understanding the efficacy, reliability and resiliency of computer vision techniques for malware detection and future research directions. arXiv preprint arXiv:1904.10504, 2019.

16. Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In 2017 European Intelligence and Security Informatics Conference (EISIC), pages 99–106. IEEE, 2017.

17. Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. arXiv preprint arXiv:1901.03583, 2019.

18. Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing, 2017.

19. Saeed Ehteshamifar, Antonio Barresi, Thomas R Gross, and Michael Pradel. Easy to fool? testing the antievasion capabilities of pdf malware scanners. arXiv preprint arXiv:1901.05674, 2019.

20. Weiwei Hu and Ying Tan. Generating adversarial malware examples for black box attacks based on gan. arXiv preprint arXiv:1702.05983, 2017.

21. Weiwei Hu and Ying Tan. black box attacks against rnn based malware detection algorithms. In Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

22. Alex Huang, Abdullah Al-Dujaili, Erik Hemberg, and Una-May O'Reilly. On visual hallmarks of robustness to adversarial malware. arXiv preprint arXiv:1805.03553, 2018.

23. ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Android malware detection using deep learning on api method sequences. arXiv preprint arXiv:1712.08996, 2017.

24. Aminollah Khormali, Ahmed Abusnaina, Songqing Chen, DaeHun Nyang, and Aziz Mohaisen. Copycat: Practical adversarial attacks on visualization-based malware detection. arXiv preprint arXiv:1909.09735, 2019.

25. Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In 2018 26th European Signal Processing Conference (EUSIPCO), pages 533–537. IEEE, 2018.

26. Alex Kouzemtchenko. Defending malware classification networks against adversarial perturbations with non-negative weight restrictions. arXiv preprint arXiv:1806.09035, 2018.

27. Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv preprint arXiv:1802.04528, 2018.

28. Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. arXiv preprint arXiv:1809.06498, 2018.

29. Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge. arXiv preprint arXiv:1812.08108, 2018.

30. Xiaolei Liu, Xiaojiang Du, Xiaosong Zhang, Qingxin Zhu, Hao Wang, and Mohsen Guizani. Adversarial samples on android malware detection systems for iot systems. Sensors, 19(4):974, 2019.

31. Xinbo Liu, Jiliang Zhang, Yaping Lin, and He Li. Atmpa: Attacking machine learning-based malware visualization detection methods via adversarial examples. 2019.

32. Davide Maiorca, Battista Biggio, and Giorgio Giacinto. Towards robust detection of adversarial infection vectors: Lessons learned in pdf malware. arXiv preprint arXiv:1811.00830, 2018.

33. Marco Melis, Davide Maiorca, Battista Biggio, Giorgio Giacinto, and Fabio Roli. Explaining black box android malware detection. In 2018 26th European Signal Processing Conference (EUSIPCO), pages 524–528. IEEE, 2018.

34. Daniel Park, Haidar Khan, and Bülent Yener. Short paper: Creating adversarial malware examples using code insertion. CoRR, abs/1904.04802, 2019.

35. Robert Podschwadt and Hassan Takabi. Effectiveness of adversarial examples and defenses for malware classification. arXiv preprint arXiv:1909.04778, 2019.

36. Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Query-efficient gan based black box attack against sequence based machine and deep learning classifiers. arXiv preprint arXiv:1804.08778, 2018.

37. shai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval 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, pages 490–510. Springer, 2018.

38. Jack W Stokes, De Wang, Mady Marinescu, Marc Marino, and Brian Bussone. Attack and defense of dynamic analysis-based, adversarial neural malware detection models. In MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM), pages 1–8. IEEE, 2018.

39. Octavian Suciu, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In 2019 IEEE Security and Privacy Workshops (SPW), pages 8–14. IEEE, 2019.

40. Rahim Taheri, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Ali Miri, and Mauro Conti. On defending against label flipping attacks on malware detection systems. arXiv preprint arXiv:1908.04473, 2019.

41. Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In Proceedings of the 2016 network and distributed systems symposium, pages 21–24, 2016.

42. Park Daniel, Khan Haidar, and Yener Bülent. Generation & evaluation of adversarial examples for malware obfuscation. arXiv preprint arXiv:1904.04802, 2019.

43. Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In 2018 26th European Signal Processing Conference (EUSIPCO), pages 533–537. IEEE, 2018.