Journal of Information Security and Applications 68 (2022) 103267

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

MaliCage: A packed malware family classification framework based on DNN and GAN

Xianwei Gao *, Changzhen Hu, Chun Shan *, Weijie Han

Beijing Key Laboratory of Software Security Engineering Technology, School of Computer Science & Technology, Beijing Institute of Technology, Beijing 100081, China

ARTICLE INFO

Index Terms:
Classification
GAN
DNN
Packed malware

ABSTRACT

To evade security detection, hackers always add a deceptive packer outside of the original malicious codes. The coexistence of original unpacked samples and packed samples of same family needs special attention in malware detection. The features of packed malware are changed by the packer, which would disturb the prediction results of malware classifier. The state-of-the-art studies of malware detection mainly focus on whether the malware is packed, or which type of packer is used. However, the ability of detecting the family of packed malware is still insufficient. Motivated by the above challenges, a novel packed malware family classification framework called MaliCage is proposed. The goal of the framework is to classify packed malware accurately. MaliCage consists of three core modules: packer detector, malware classifier, and a packer generative adversarial network (GAN). The packer detector is used as the pre-step of the framework to identify whether malware is packed. After distinguishing the packed samples, the dynamic features extracted from the sandbox are fitted to the malware classifier based on deep neural networks (DNN). The malware classifier can classify unpacked and packed malware simultaneously. Furthermore, the packer GAN generates fake packed samples to alleviate the underfitting of the malware classifiers. We built a single-packer dataset and a multi-packer dataset to evaluate the framework. In the single-packer experiment, 10 classes of malware samples packed by UPX were examined objectively. The accuracy of the malware classifier when using only real packed samples was 91.66%. After introducing fake packed samples generated by packer GAN, the accuracy of the packed malware classifier could reach 97.8%. In the multi-packer scenario, our method can also accurately classify benign programs, unpacked malware and malware packed by several common packers. The validation results show that MaliCage can not only effectively solve the impacts of packed malware on machine learning model, but also improve the detection accuracy.

1. Introduction

With the advance of information and communications technology (ICT), cybersecurity threats are increasing rapidly. Malware become the critical weapons of advanced threat attacks. Large numbers of malware variants are produced every day [3,43]. Machine learning and deep learning are increasingly being studied and applied in malware analysis and detection. The analysis of malware is accomplished by both static and dynamic methods. Static analysis looks at the content of malware without executing it and extracts its features with disassembly tools [42]. However, hackers can obfuscate malware to avoid static detection. Hence, anti-virus companies have long used dynamic analysis to make up for the defects of static analysis. Dynamic analysis studies malware behavior in a virtual sandbox, and it helps to cope with obfuscation issues [11,31]. Although there have been advances in malware detection using artificial intelligence algorithms, attacks still cannot be adequately defended against. Hackers can use packing tools to camouflage malware to evade detection [32]. The detection performance of classification algorithms against malware, especially packed malware, is still insufficient.

Malicious software running on Windows platforms must conform to the Portable Executables (PE) file format designed by Microsoft. The original PE programs can be easily disassembled for analysis, while packer tools can impede this analysis and evade detection. Packers usually compress or encrypt PE files and restore them after they are loaded into memory. Many static and dynamic features of malware will be changed after packing, such as section sizes, opcodes, API calls. Therefore, malware after being packed are very different from the

* Corresponding authors.
E-mail addresses: gaoxw@bit.edu.cn (X. Gao), sherryshan@bit.edu.cn (C. Shan).

https://doi.org/10.1016/j.jisa.2022.103267

Available online 7 July 2022

original malware. Packed malware can bypass the forensic analysis of security experts, which makes it difficult to obtain sufficient packed samples. Most machine learning detection algorithms are based on supervised learning and require enough labeled samples to train. Hence, a machine learning model based on the training dataset of original unpacked samples cannot accurately predict the packed malware's family.

With the increasing of packed malware, the research in this field has already attracted attention [49]. The state-of-the-art studies of malware detection domain mainly focus on whether the malware is packed, or which type of packer is used, but the ability to classify the family of packed malware is still insufficient. According to our investigation, there are four urgent concerns when detecting packed malware:

(1) There is few published research on the classification of packed malware. The analysis of the impact of packed malware on the detection model is also insufficient.
(2) Some studies on packed malware can only predict whether the sample is malicious but cannot classify its family. Alkhateeb et al. [2] utilized a Naive Bayes classifier to distinguish between benign and malicious packed samples, but the authors did not classify the family of malware.
(3) Some detection algorithms require to unpack the packed samples, which makes it difficult to cope with the private packers used by hackers. Maleki et al. [29] presented a method to detect packed malware based on features extracted from the PE header and section table of malware. Their method required unpacking samples through the Quick Unpack tool firstly.
(4) Real packed samples are hard to collect, which has a great impact on the training effect of the packed malware detection model. The purpose of hacker packing is to turn an existing malware into an unknown variant, so that the training dataset of the defender generally does not contain this new variant.

To address these challenges, we propose a packed malware classification framework based on deep learning in this paper. The packed malware must contain a piece of unpacker codes to ensure itself execute properly on target machine. The packing process is reversible, so packer provides important clues for malware detection. Static analysis can be used to judge whether a file is packed. Packed malware must keep functions of the original unpacked malware. Therefore, a sandbox can be used to extract features by dynamic analysis. Static and dynamic analysis could be combined to detect whether a malware is packed and then classify its family. The primary contributions of our work are summarized as follows.

(1) A novel packed malware family classification framework called MaliCage is proposed. It consists of a packer detector and a malware classifier based on deep neural networks (DNN). The packer detector is the pre-step to identify whether a malware is packed, then the classifier is responsible for predicting its family.
(2) We also proposed a packer generative adversarial network (GAN) model. It can generate fake packed samples for classifier training to prevent it from under fitting by learning the characteristics of existing packed samples.
(3) Packed malware datasets from VirusShare and VXheaven are built to experiment and evaluate our model. The accuracy of packed detector reaches 98.20%, and the packed malware classifier reaches 91.66%. After introducing packer GAN, the accuracy of the packed malware classifier was promoted to 97.8%.
(4) We also explored the packing's influence on the features of malware and the performance of malware classifier. The method and experiment results are summarized and compared with other related studies, and our findings can provide reference for other researchers.

The rest of the paper is organized as follows. Section 2 surveys the related work in the field of malware detection. Section 3 discusses the background and motivation of the proposed method. Section 4 describes the framework and implementation of MaliCage. Section 5 evaluates its performance and compares with similar studies. Section 6 concludes our findings in the paper.

2. Related work

In recent years, machine learning and deep learning have made a lot of achievements in cyber security and malware defense [46,54,55,56,57,58,59]. Static and dynamic analysis are widely used in malware detection [19] [48]. Drew et al. [17] applied methods designed for gene sequencing to detect malware and robustly avoid the adaptation of attackers. Zhang et al. [51] proposed a static approach based on text analysis to map ransomware into families. Dynamic analysis is more robust than static analysis and can analyze the malware more comprehensively [9,14,20,44]. It always apply a sandbox to isolate malware, and convert the process, mutexes, files, networks, registries, and other behaviors to features for detection [15,27,39,24]. Wüchner et al. [45] proposed a malware detection approach that used compression-based mining on quantitative data flow graphs to derive highly accurate detection models. Korczynski et al. [26] proposed a Tartarus system that can capture intrinsic characteristics of novel code injection techniques. Bahtiyar et al. [6] proposed a multi-dimensional machine learning approach to predict Stuxnet-like malware using five distinguishing features of advanced malware. Tang et al. [41] used dynamic analysis to extract an API call and generated feature images representing malware behavior according to color mapping rules. An Apriori algorithm was applied to extract the common behaviors of individuals belonging to the same family, and to represent the knowledge of a malware family [16]. Mirza et al. [30] proposed a combination of machine learning techniques applied to a rich set of features extracted from a large dataset of benign and malicious files. These methods often focus on the detection accuracy of simple malware, but do not consider the antagonism of malware. However, malware with advanced evasion techniques brings more challenges to dynamic analysis [5,33]. Malware often determines the presence of an analysis environment and evades detection by performing no malicious activity [1]. Researchers have also studied many ways to prevent evasive malware. Shi et al. [37] proposed a detect-and-hide approach to systematically address anti-virtual machine (VM) techniques in malware. Botacin et al. [8] proposed a framework that relied on modern processors' branch monitor feature to allow researchers to analyze malware while reducing evasion effects. BluePill offered a customizable execution environment that remained stealthy when analysts intervened to alter instructions and data or run third-party tools [13]. In the studies above, the authors often assume that the malware is not packed, so they pay little attention to the problem of packed samples in their experimental dataset. The new packed variants may easily bypass such detection methods.

Packing is a frequently used countermeasure against malware defense. To detect packed malware variants, Zhang et al. [50] proposed a method that extracts a series of system calls that are sensitive to malicious behavior and adopted multi-layer neural networks to classify the features of malware variants. Hua et al. [21] used a deep graph convolutional network (DGCNN) to classify control flow graph data of packed malware. Liu et al. [28] proposed a two-stage packer identification method based on FCG and file attributes. They focused on the detection and identification of packers. But they did not solve the key problem of the classification of packed malware family. Yu et al. [49] proposed a malware spectrum visualization framework to identify malware variants that use evasive techniques such as packer. They treated the packed samples and unpacked samples without differentiation and did not consider the challenge of new packed variants.

The following conclusions can be drawn from the analysis of the above studies. These methods assume that both the training set and the

test set will contain packed samples, but don't consider the gap of lacking packed training samples. Most of the previous studies only looked at the issue of unpacked malware detection and failed to consider the impact of the packed malware on the detection process. Although static analysis methods have high efficiency, they do not analyze and solve the impact of packing. Dynamic analysis needs enough real samples, and the difference between unpacked samples and packed samples is not considered. The impact of malware packing on machine learning model needs more in-depth analysis and attention. The malware detection model must also deal with the problems and challenges caused by packing.

3. Motivation

3.1. Background

To run properly on MS-DOS, Win7 and other versions of Microsoft operating system, a malware must be compiled into a program in PE format. Windows malware is usually a PE program with a DOS header, NT header, section table, and specific sections. The DOS header is compatible with MS-DOS, which indicates the location of the NT header. The NT header defines some native information and important attributes of the file. The section table describes the subsequent sections of the PE file. Windows loads each section according to the description of the section table. Each section is a container with independent memory permission. For example, the code section has read or execute permissions by default, and the section name and number can be defined by themselves. Security experts can easily analyze and detect original (unpacked) malware according to the PE format.

Unfortunately, hackers often use packer tools to camouflage malware to prevent its detection. Packing is an effective tactic of evasion, so we first study the principle of malware packing. A common practice of malware packing is to insert a piece of unpacker code in the binary program, as shown in Fig. 1. The ideal method is to unpack the packed program and extract the original malicious code. Cheng et al. [12] recovered executable malware program from the packed and obfuscated binary code by reconstructing import tables using a hardware-assisted tool, but their method failed to produce an executable PE file from the unpacked code for custom DLLs and OEP Obfuscation.

According to related studies of packer detection, the ultimate packer for executables (UPX) is the most used packer [7,28,40]. Programs and libraries compressed by UPX are completely self-contained and they can run exactly as before, with no runtime or memory penalty for most supported formats. After the packer is added, the original program code generally exists in compressed or encrypted form in the disk file and is only restored in memory during execution. When running, the control power of the program is first obtained by the unpacker code, and then returned to the original code. The packed malware hides its original entry point (OEP) and prevents it from being cracked.

Based on the packer's purpose and behavior, they can be broadly classified into four categories [47]:

(1) Compressor is the simplest way to pack. It shrinks file sizes without considering the tricks to prevent unpacking.
(2) Cryptors encrypt the original malicious payloads and prevent malware from being analyzed without decryption.
(3) Protectors combine functions of compressors and cryptors.
(4) Bundlers combine a package of multiple executable and data files into a single packed executable file, which unpacks and accesses files within the package without extracting them to disk.

3.2. Motivation

Hackers will try to thwart security defense to successfully implement attacks. They frequently employ packers to compress and obfuscate malware variants. The file size, PE attributes, section table, assembly opcodes, and other critical features of packed malware differ significantly from the original malware. The purpose of packing is to make the features of the packed malware closer to benign software. Fig. 2 shows the influence of packing on the malware features and detection model. The new packed variants will lead to false positives in the malware detection model.

To preserve malicious functions, the malware after packing will still retain the original execution code. Therefore, even if malware is packed, we can find some clues to determine its original family by analyzing the packing techniques. Based on the above analysis, we will highlight the following problems and propose a novel malware detection framework in this paper.

(1) What is the impact of malware packing on a malware classification model based on machine learning?
(2) How can we predict whether a suspicious file has been packed?
(3) Can we design a classifier based on deep learning to detect both unpacked and packed samples?
(4) Whether the model classify packed samples only based on unpacked training samples?
(5) In the absence of real packed training samples, can GAN be used to generate simulated packed samples to train a classifier?

These problems have not been adequately addressed in the past. As a branch of machine learning, deep learning is widely applied in many fields and research areas such as speech recognition, image processing, medicine, cyber security, and so on. The draw-back of the traditional machine learning classifiers is that people need to write a complex hypothesis by themselves, while in the deep learning it is generated by the neural networks [38]. Deep learning has become a powerful tool for learning nonlinear relationships effectively. Through the analysis of the process of malware packing, we combine static and dynamic analysis to detect and classify packed malware. Inspired by the success of GAN in imaging, we make use it to automatically generate simulated fake

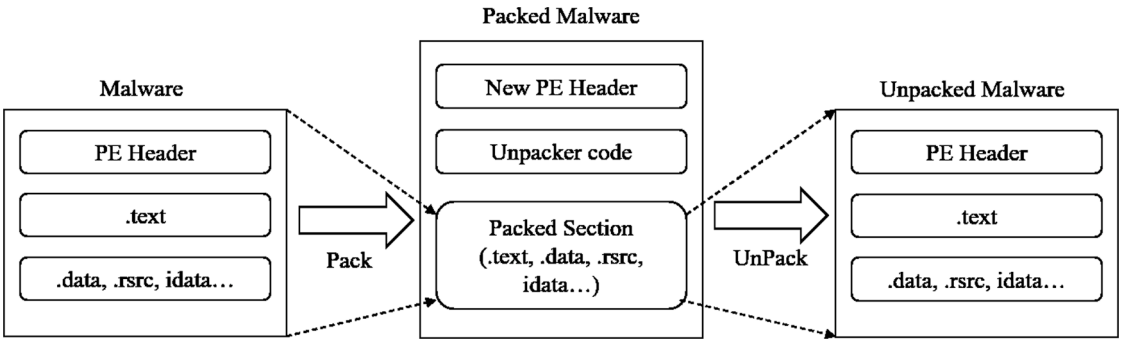


Fig. 1. Process of malware being packed and unpacked.

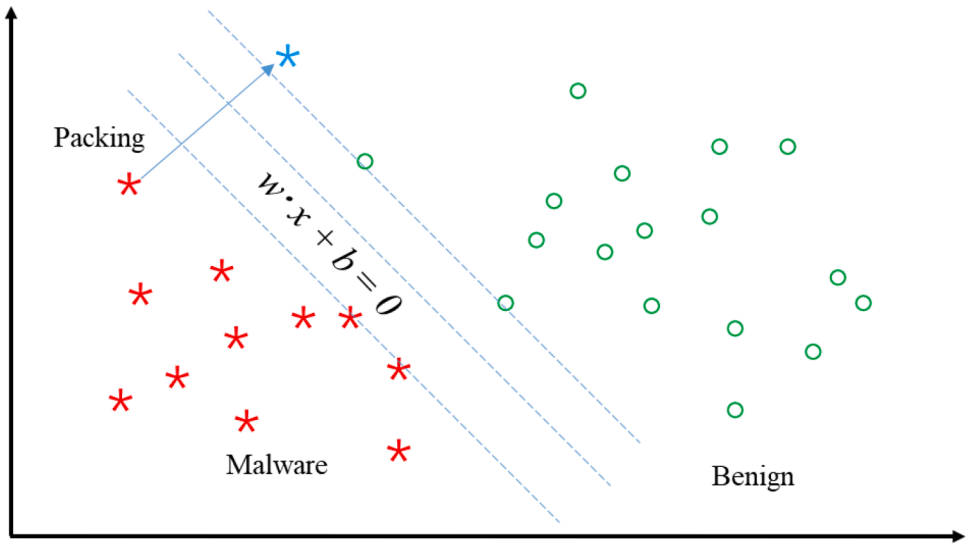


Fig. 2. Influence of packing on malware detection.

packed malware as training data to promote our model. Although packing technology can be used for any executable program, our research object mainly focuses on high-risk backdoor programs such as Conficker. We hope the proposed model can not only judge whether malware is packed but can accurately classify its family.

4. Proposed Approach

4.1. Framework of MaliCage

To tackle the challenges of packed malware, we designed the packed malware classification framework MaliCage shown in Fig. 3. Different shapes of icons in the figure, such as circles, triangles, and pentagrams, represent different malware families. In addition, the white icons represent original unpacked samples, and the black icons represent packed samples.

MaliCage puts the packed malware into a virtual and controlled environment for dynamic analysis, like a mouse cage, and then uses the artificial intelligence technology to classify packed malware. Based on the advantages of deep learning techniques in many application fields, we chose DNN for malware detection and classification. The framework consists of a malware packer detector, a packed malware classifier, and packer GAN.

(1) Malware packer detector: This detector uses static analysis to determine whether malware is packed. It is based on a deep neural network for binary classification detection.
(2) Packed malware family classifier: This classifies the family of packed malware based on sandbox dynamic analysis. It uses deep neural network as the classification algorithm and can classify unpacked and packed samples of the same family.
(3) Packer GAN: Due to the lack of packed samples for training, the detection ability of the packed malware classifier is severely limited. The packer GAN can learn the features of packed samples and automatically generate simulated packed malware samples from the unpacked malware. These fake samples are added to the training dataset, and the packed malware classifier can detect real packed malware variants after training.

There are significant differences between packed and unpacked samples in the same family, so unpacked and packed classifiers are trained separately. MaliCage can detect whether a program is packed

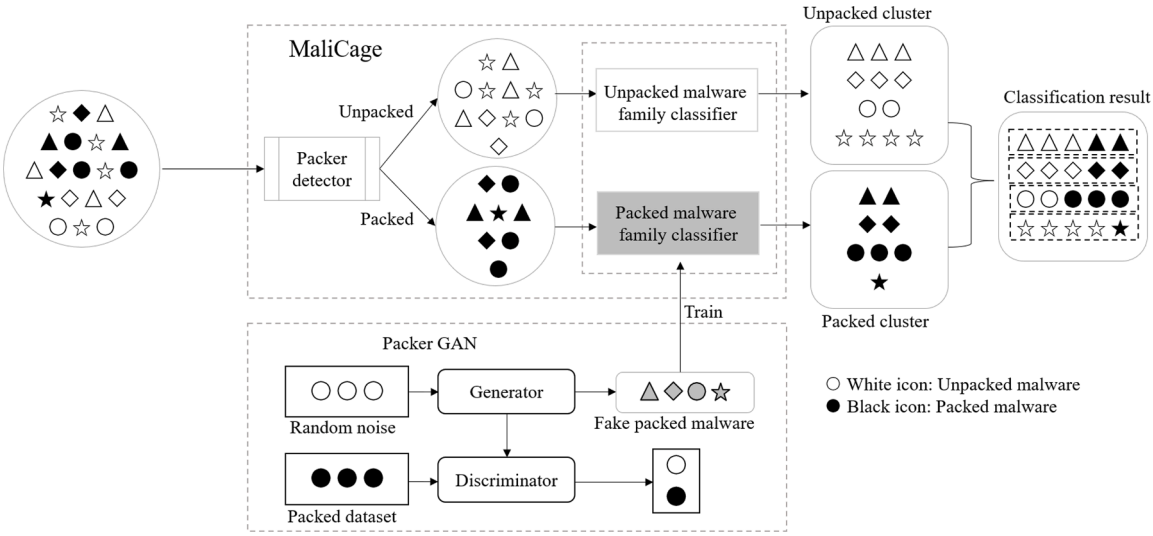


Fig. 3. Packed malware classification framework (MaliCage).

and classify the malware samples into their families.

The processing process of MaliCage shown in in Fig. 4 includes three stages: training, optimization, and testing. The main workflow is as follows.

(1) In the training stage, unpacked and packed samples were collected as training dataset. The training samples are disassembled, and then opcode features are extracted to train packer detector.
(2) The training samples are sent to the sandbox for analysis, and the WinAPI features are extracted to train the malware classifier.
(3) In the optimization stage, the generator and discriminator in the packer GAN are trained with real samples firstly. The discriminator helps generator to produce highly simulated packed samples. Then those fake packed samples are used to promote the malware classifier.
(4) In the testing stage, the static and dynamic features of unknown samples are extracted to detect and classify.

4.2. Malware packer detector

To determine the family of packed malware, we first judge whether it is packed. After the program being packed, the instructions of the unpacker should be executed first when running. Malware camouflaged by same packer will have similar unpacker codes, and these features are difficult to conceal. Based on this hypothesis, we disassemble malware and extract their opcode features for detection. The n-gram is an important concept in natural language processing and can be used to evaluate the difference between two string sequences. In the process of n-gram feature extraction, n is generally a fixed value, such as 1, 2, 3 and so on. To better reflect the logical relationship between program opcodes, we set n as variable x. For example, if x is set to 3, the 1-gram, 2-gram and 3-gram of the program opcodes are calculated first. In algorithm analysis, the number of features has impact on the detection effect. After many tests and comparisons, we found that the top 100 opcode features can achieve better performance. Then, the 100 assembly opcode sequences with the highest frequency are counted from the 3 sets as x-gram features. Algorithm 1 describes the process of a malware packer detector.

Algorithm 1. Detecting malware packer.

Input: training malware dataset_A, test malware dataset_B
Output: detection result of dataset B
1: Opcodes_A ← extract disassembly instructions from dataset_A
2: **for** i ← 1 to n **do**
3:n_grams ← state n-Grams of Opcodes_A from 1 to n
4: **end for**
5:x_grams ← get_top(n_grams, *100*)
6:static_feat_A ← extract(Opcodes_A, x_grams)
7: MPDetector ← Classifier.fit(static_feat_A.unpacked, static_feat_A.packed) *//Using the features of unpacked samples and packed samples in dataset A for training.*
8:static_feat_B ← extract disassembly instructions x_grams from dataset_B
9:value ← MPDetector.predict(static_feat_B)
10: **if** value = 0 **then**
11:result ← "unpacked"

(*continued on next column*)

(*continued*)

12: **else if** value =1 **then**
13: result ← "packed"
14: **end if**
return result

Dataset A contains both unpacked and packed samples, and we can use supervised learning to detect whether malware is packed. Static analysis has the advantage of not needing to run the program, and its operation is simple and fast. It uses special tools to disassemble malware programs into ASM files. Through parsing of the ASM file, the pure code section (UPX1) is located, the top 100 x-gram opcode sets are extracted as features. Fig. 5 shows the disassembly code of demo malware packed by UPX. The code section of the packed malware stores unpacker codes, which are often similar when they come from the same packing algorithm.

We can extract the opcode sequence from the disassembly code from Fig. 5 as follows:

{*push mov push push call or cld add enter push call sar and test ror and xor int add pop and jg mov push*}

1-gram: {push}, {mov}, {call}…
2-gram: {*push mov*}, {*mov push*}, {*push push*}…
3-gram: {*push mov push*}, {*mov push push*}, {*push push call*} …

In the x-gram method, x defines the maximum value of n. We count the values of all n-gram sets from *1* to x and select the top 100 x-gram opcode sequences as the feature fields of the malware. To determine whether a suspicious file is packed, we use the DNN algorithm to detect its opcode x-gram features. DNN uses the sigmoid function to determine whether the malware is packed.

4.3. Malware detection algorithm based on DNN

Our model needs to carry out two tasks: packer detection and malware classification. We design one malware detection model based on deep neural networks to finish them shown in Fig. 6. It switches the two tasks through different input and output layer parameters. The malware detection DNN includes input, hidden, and output layers. After malware is analyzed, the extracted features are submitted to the input layer. Layers between the input and output layers are called hidden layers. The hidden layer does not directly receive or send signals to the outside world. The output layer calculates the detection results by corresponding activation functions.

In neural networks, the activation function of a node defines its output under a given input. Two commonly used activation functions, the rectified linear unit (ReLU) and sigmoid functions, are described in Fig. 7. ReLU changes negative inputs to 0, and other inputs are unchanged. In our model, the features of malware are nonnegative, so ReLU (1) is used as the activation function in the hidden layer. The sigmoid (2) function is used for hidden layer neuron outputs. Its value range is (0,1). It can map a real number to the interval (0,1) and can be used for binary classification. In the malware packer detector, the

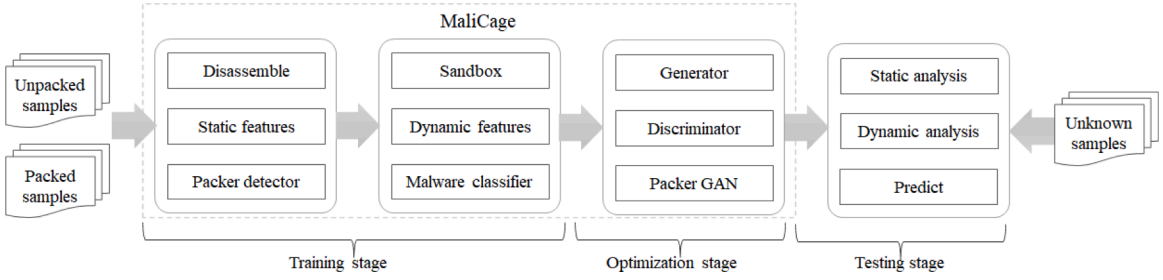


Fig. 4. System architecture and process flow of MaliCage.

```
UPX1:00408000 ; Segment type: Pure code
UPX1:00408000 ; Segment permissions: Read/Write/Execute
UPX1:00408000 UPX1            segment para public 'CODE' use32
UPX1:00408000                 assume cs:UPX1
UPX1:00408000                 ;org 408000h
UPX1:00408000                 assume es:nothing, ss:nothing, ds:UPX0, fs:nothing, gs:nothing
UPX1:00408000
UPX1:00408000 loc_408000:                             ; DATA XREF: start+1↓o
UPX1:00408000                 jmp     ebp
UPX1:00408000 ; ---------------------------------------------------------------------------
UPX1:00408002                 dw 0FFDBh
UPX1:00408004 ; ---------------------------------------------------------------------------
UPX1:00408004                 push    ebp
UPX1:00408005                 mov     ebp, esp
UPX1:00408007                 push    ecx
UPX1:00408008                 push    40110000h
UPX1:0040800D                 call    near ptr 8BBC8016h
UPX1:00408012                 or      eax, 458D3810h
UPX1:00408017                 cld
UPX1:00408018                 add     esp, 4
UPX1:0040801B                 enter   0FFFFCBF7h, 0EEh
UPX1:0040801F                 push    eax
UPX1:00408020                 call    dword ptr ds:858B340Ch
UPX1:00408026                 sar     byte ptr [esi+14h], 50h
UPX1:0040802A                 and     al, 8
UPX1:0040802C                 test    [ebp+edi*8+33081BB7h], ebx
UPX1:00408033                 ror     byte ptr [ebx+79C35DE5h], 15h
UPX1:0040803A                 and     [edx+12BE4193h], dh
UPX1:00408040                 xor     al, 4
UPX1:00408042                 int     3               ; Trap to Debugger
UPX1:00408043                 add     [edi-471227F5h], bh
UPX1:00408049                 pop     ss
UPX1:0040804A                 and     edi, [eax+10h]
UPX1:0040804D                 jg      short loc_4080A5
UPX1:0040804F                 mov     esi, [ebp+8]
UPX1:00408052                 push    1
```

Fig. 5. Disassembly codes of malware packed by UPX.

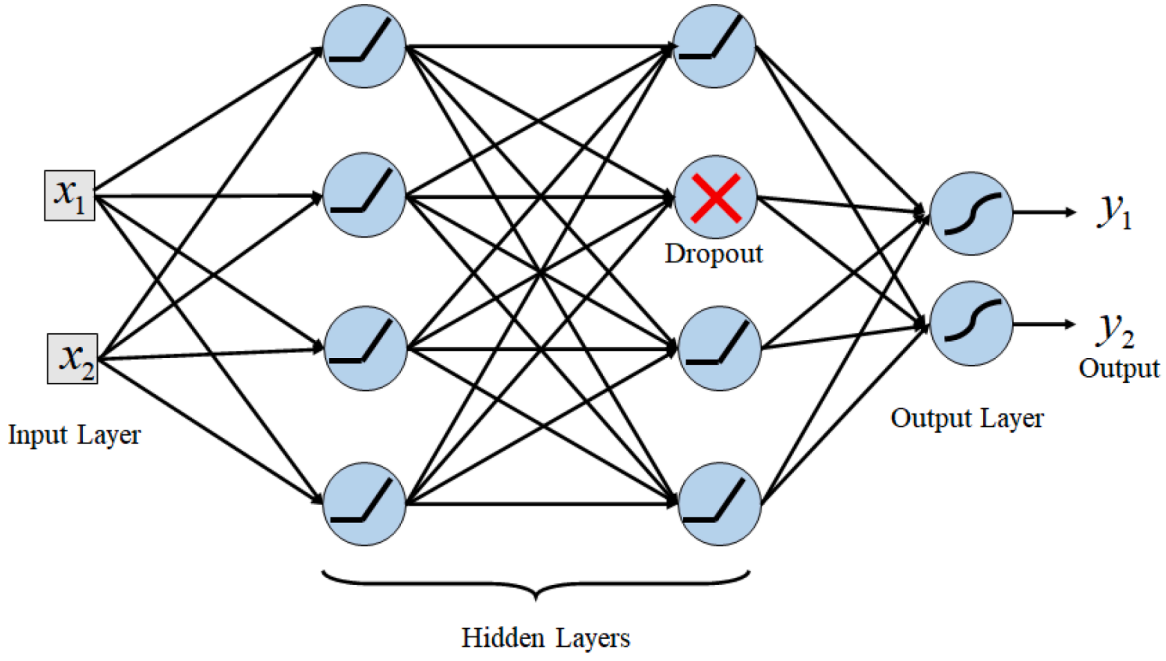


Fig. 6. Malware detection DNN.

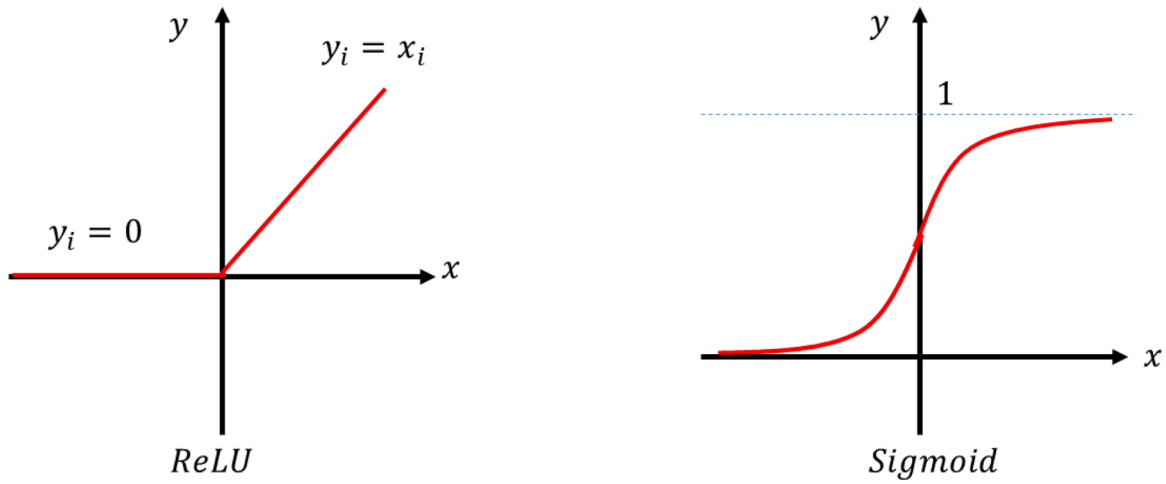


Fig. 7. Activation functions.

sigmoid function is used for activation of the output layer. If the result of sigmoid output is 0, the sample is not packed, otherwise it is packed.

The softmax function is a generalization of sigmoid in multi-classification and shows its results in the form of a probability. Softmax (3) is widely used in machine learning and deep learning specially when dealing with multi classification (c > 2). The final output unit of the classifier needs the numerical processing of softmax function. The *Vi* is the output of the output unit of the front stage of the classifier, *i* represents the category index, and the total number of categories is K. The *Si* represents the ratio of the index of the current element to the sum of the indexes of all elements. Softmax can compress any real vector of length k to a real vector of length k, whose elements sum to 1. Softmax converts the output values of multiple categories into relative probabilities, which is easier to understand and compare. If the result of softmax output is *k*, the sample belongs to the *k*-th family.

$$ReLU: f(x) = max(0, x) \tag{1}$$

$$Sigmoid: f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

$$Softmax: S_i = \frac{e^{V_i}}{\sum_1^K e^{V_k}} \tag{3}$$

Table 1 shows the network structure of the malware classifier DNN, with a total of 77,291 parameters. The structure includes one input layer, two hidden layers, and one output layer. The output layer uses a sigmoid activation function for binary classification, and a softmax activation for multi-classification tasks. After the original vector passes the sigmoid function, the larger elements in the original vector are still larger in the corresponding positions in the output vector. The smaller elements in the original vector will still be small, retaining the size relationship between the original vector elements. In multi-classification, the largest dimension of the output vector is associated with the largest probability of belonging to the class, so it is classified.

Table 1.
Hyper parameters of malware classifier.

	Output Shape	Parameter
Input layer	(None, 32)	224
Hidden layer1	(None, 256)	8448
Activation	-	Relu
Dropout	(None, 256)	0.2
Hidden layer2	(None, 256)	65792
Dropout	(None, 256)	0.2
Out layer	(None, 11)	2827
Activation	-	Softmax
Optimizer	-	Adam
Loss function	-	Categorical crossentropy

To optimize the parameters in this model, we define the categorical crossentropy as the loss function. The categorical crossentropy function is very fit for classification tasks, since one example can be considered to belong to a specific category with probability 1, and to other categories with probability 0. The Adam optimizer is then used to optimize the parameters. It is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. To get an accurate classifier, the neural networks are trained 100 epochs.

4.4. Packed malware classifier

Malware must perform their original malicious functions even after being packed, their dynamic behavior has much relevance to the unpacked samples. They are sent to the Cuckoo sandbox for dynamic analysis to catch their process, network, file, registry, system, and other operations. The submission module adds malware samples requiring analysis to the queue and waits. The management module starts the virtual machine and transmits malware samples to it. The monitoring module in the VM records malware behaviors. The report module receives the analysis report from the virtual machine and saves it as a report file. After the malware is executed, the management module shuts down and initializes the virtual machine to wait for the next analysis task. The dynamic features are extracted from the sandbox reports. Some dynamic analysis methods use APIs called sequences as detection features [35,34,36]. The analysis report of the Cuckoo sandbox mainly includes key information such as target, static and behavior, as shown in Fig. 8. The target part of analysis report mainly includes basic information such as the file name and type of the analyzed program. The static part mainly includes the PE format information of the file, such as section table attributes. The dynamic part mainly includes the behavior of the malware when running in the sandbox, including process information and API call information. When the packed malware runs in the sandbox, it first executes unpacker codes, and then performs its original function. Therefore, the API statistics will include the behavior characteristics of the unpacked samples. In our model, the ApiStats part of the malware is extracted as the dynamic feature to classify packed malware. In the ApiStats part, the number of times the malware calls the windows API during execution is counted.

Algorithm 2 describes the training process of the packed malware classifier. The behavior information in the report includes the list of processes executed by the malware, loaded DLLs, read-write registry, file operation, and network connection. We extract the top 100 WinAPI from sandbox reports as classification features. To reduce the impact of

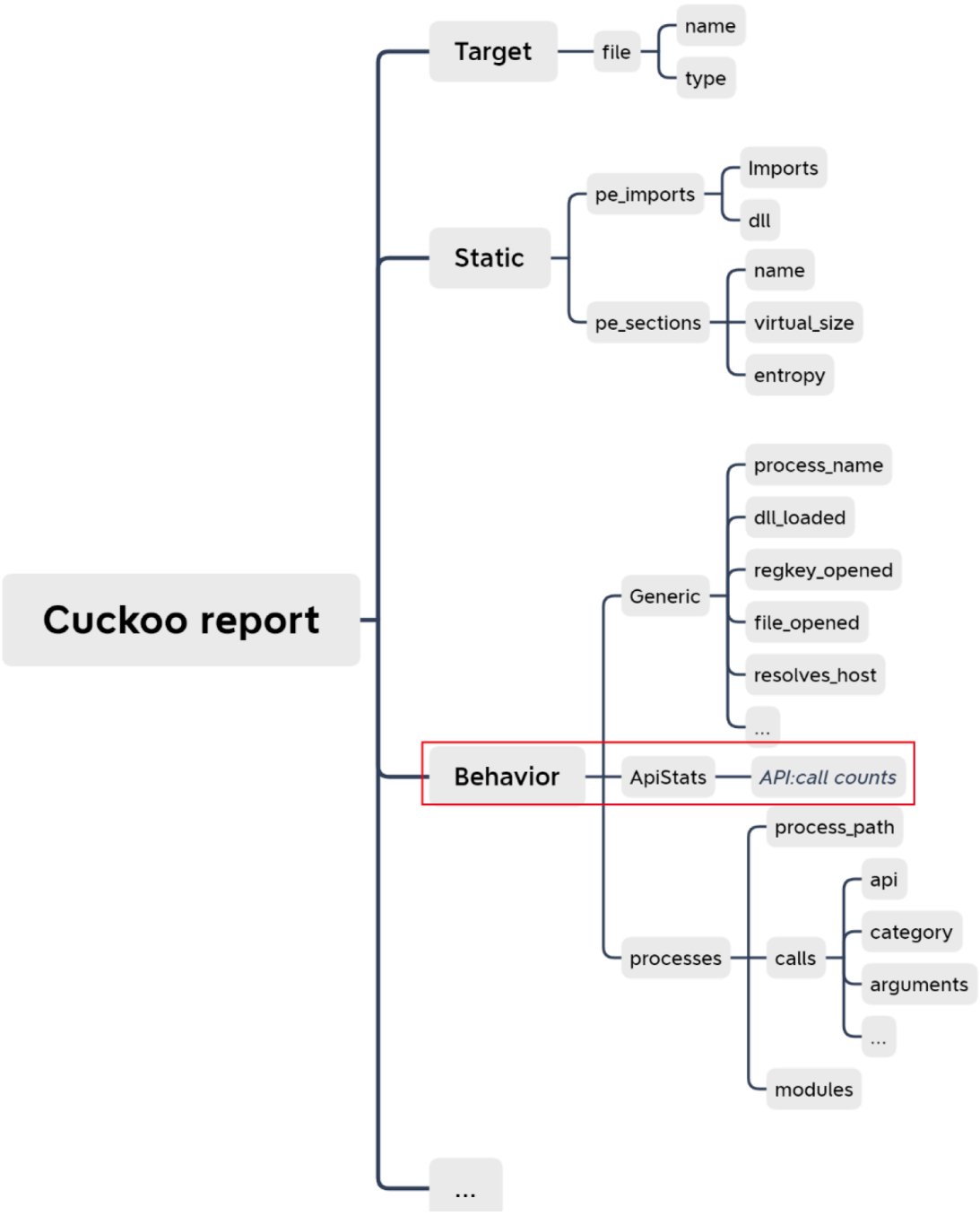


Fig. 8. Sandbox analysis report sample of malware.

packing on the classification effect, we separately train the unpacked and packed samples. After training, the classifier can predict the test data.

Algorithm 2. Training process of packed malware classifier.

Input: training dataset A, test dataset B
Output: classification result of test
1: reports_A, reports_B ← use sandbox to analysis A and B and get reports
2: TopAPI ← Count the 100 most frequently called Windows APIs in training samples
3: feature_A, feature_B ← extract Top API Stats features from reports_A and B
4: PM_classifier ← build a packed malware classifier based on DNN
5: PM_classifier.fit (feature_A) //train classifier
6: result ← PM_classifier.predict (feature_B)
7: **return** result

4.5. Packer GAN

The hackers' purpose of malware packing is to generate new unknown samples, which makes it difficult for security analysts to detect. Therefore, the collection of packed samples is the biggest difficulty in model training. Previous studies on malware detection have not fully considered this problem. Goodfellow et al. [23] proposed an adversarial nets framework that pits the generative model against an adversary, a discriminative model that learns to determine whether a sample is from the model distribution or data distribution. GAN is an important model of adversarial learning, which adversely trains a discriminator and a generator using cross-entropy loss function [4]. For the first time in the

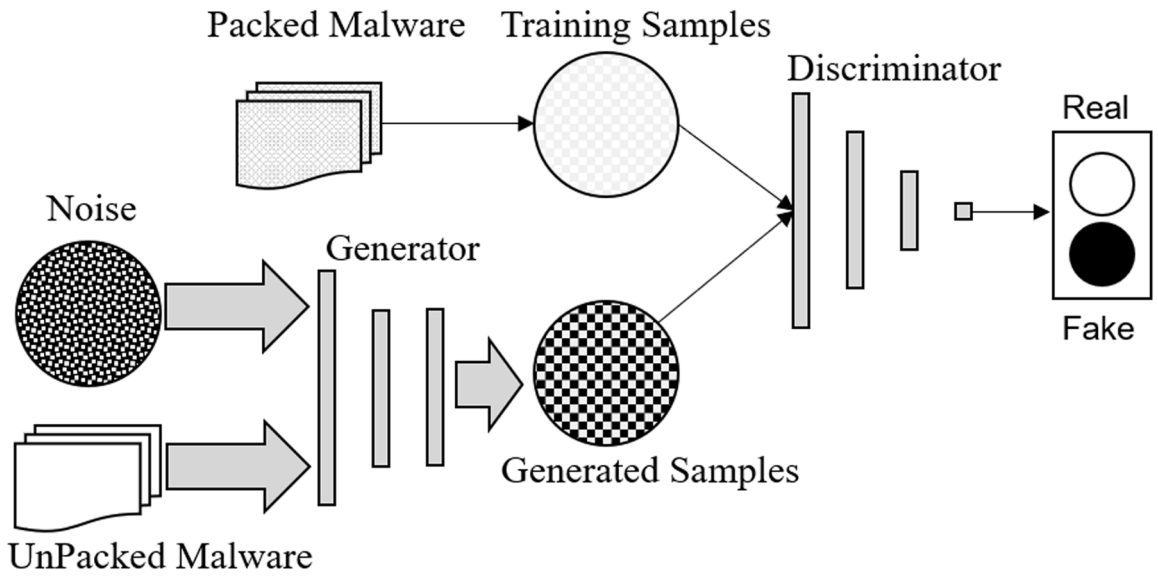


Fig. 9. Architecture of packer GAN.

industry, we designed a packer GAN shown in Fig. 9 to generate fake packed samples for model training by simulating real packed samples automatically.

The packer GAN includes a packer generator and packer discriminator. The discriminator of packer GAN is trained with real unpacked and packed samples, so that it can accurately predict whether a sample is a real packed sample. The generator is trained with real packed samples to generate fake packed samples with noise input that are as realistic as possible.

Algorithm 3. Training process of packer GAN

Input: real packed dataset A, real unpacked dataset B
Output: fake packed samples generated by GAN
1: Feature A, Feature B ← extract WinAPI features from sandbox report of dataset A, B
2: Discriminator, Generator ← build generator and discriminator based on neural network
3: PackerGAN← connect Discriminator, Generator to packer GAN
4: for i ← 0 to epochs do
5: real_packs ← get_real_samples(FeatureA, batch)
6: fake_packs ← Generator.get_fake_samples(noise, batch) //Generate a batch of fake packed malware
7: Discriminator ← train(real_packs, fake_packs)
8: Update(Discriminator, θd) // Update the discriminator's parameter
9: Update(Generator, θg) // Update the generator's parameter
10: end for
11: fake_packed_samples ← PackerGAN.generate(Feature B)
12: return fake_packed_samples

Algorithm 3 describes the working process of packer GAN. The discriminator is a kind of classifier that distinguishes the authenticity of samples. Cross-entropy (4) is always used to distinguish the similarity of distribution.

$$H(p,q) = -\sum_i p_i log q_i \tag{4}$$

The p_i and q_i are the real sample distribution and generated distribution, respectively. The model learns the distribution of the real packed sample X, fixes the generator G, and updates the discriminator D. The discriminator learns to assign high scores to real objects and low scores to generated objects. The generator then uses the input noise Z to generate simulated packed samples. It fixes D and updates G. The generator learns to "fool" the discriminator.

$$\min_G \max_D V(D,G) = E_r[logD(x)] + E_g[log(1 - D(G(z)))] \tag{5}$$

We use loss function (5) to train the model. *V(D, G)* can measure the difference between real and generated samples. *D(x)* is the probability that *x* came from the real *(r)* packed dataset and not the generated *(g)* dataset. *1-D(x)* is the probability that *x* came from the generated samples. The process of max(*V*) is to fix *G*, and let *D* distinguish whether a sample comes from real or generated data. The process of min(*V*) is to fix *D* and obtain *G* such that the difference between the real and generated sample will be minimized. Through this min-max game process, the ideal situation will converge to the generated distribution and fit the real distribution.

Table 2 shows the network structure of packer GAN, with a total of 2,983 parameters, 2,726 trainable and 257 non-trainable. In our model, the generator and discriminator are fully connected networks. The discriminator uses a sigmoid activation function. When the output is less than the threshold value 0.5, the generated sample is considered false, and it is otherwise considered to be packed. After training, packer GAN can use its generator to produce fake packed samples.

5. Evaluation

5.1. Experimental settings

The experimental environment is configured as follows:

(1) Software configuration: Win10 64-bit, Pycharm, IDA Pro, UPX 3.09w, Python 3, Anaconda, Scikit-learn, Keras, pefile, pandas; Ubuntu 18, Cuckoo, VirtualBox, Win7 32-bit.
(2) Malware dataset: We downloaded malware sample compression package from VirusShare website, and 10 classes of malware

Table 2.
Hyper parameters of packer GAN model.

Model	Layer (type)	Output Shape	Parameter
Generator	dense (Dense)	(None, 32)	224
	dropout (Dropout)	(None, 32)	0
	dense_1 (Dense)	(None, 64)	2112
	dropout_1 (Dropout)	(None, 64)	0
	dense_2 (Dense)	(None, 6)	390
Discriminator	dense_3 (Dense)	(None, 32)	224
	dense_4 (Dense)	(None, 1)	33
packer GAN	Generator (Sequential)	(None, 6)	2726
	Discriminator (Sequential)	(None, 1)	257

were extracted from the compression package for training and testing.

Based on the above experimental environment, we developed and evaluated the model designed in this paper.

5.1.1. Evaluation metrics

The accurate measurement of security metrics is a critical research problem [18]. Evaluation metrics were derived from the four basic attributes of the confusion matrix depicting actual and predicted classes:

(1) True positive (*TP*) - Number of samples correctly classified as malicious.
(2) False positive (*FP*) - Number of samples incorrectly classified as malicious.
(3) True negative (*TN*) - Number of samples correctly classified as benign.
(4) False negative (*FN*) - Number of samples incorrectly classified as benign.

We used the following measures to evaluate the performance of our proposed solution.

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

Accuracy measures correct classifications as a proportion of the total.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Precision measures true positive classifications as a proportion of all positive classifications.

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

Recall measures the ratio of correctly predicted positive samples to all samples and reflects the ability of the classifier to detect all the positive samples.

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{9}$$

The *F1-score* measures the harmonic mean of precision and recall and serves as a derived effectiveness measurement.

5.1.2. Labeling the dataset

To prepare the dataset for model training, we need to label the family classes of training samples firstly. VirusTotal provides a security service with more than 70 anti-virus engines which is widely used for malware detecting and labeling [52]. This service can check whether one suspicious file is malicious by MD5 hash and return its family label. If 85% of anti-virus engines detected a sample was positive, it was marked as malware, otherwise, it was marked as benign. The family labels returned by each engine were not consistent. After regularizing the names of various families, the most frequent name was selected as the malware's family label. We calculated MD5 hashes for 1500 real malware samples and invoked the VirusTotal service to obtain the family labels.

5.1.3. Single-packer Dataset

The malware training dataset need include both unpacked and packed samples from multiple families. We chose 1500 common malware downloaded from VirusShare for evaluation. The original samples from ten families are not packed. They are representative in the real-world environment and can objectively reflect the effect of our proposed model. There are 10 classes of 32-bit PE malware in the experimental dataset in Table 3. To build the packed dataset for packer classifiers, 450 (30%) unpacked samples were packed by UPX.

Fig. 10 depicts the composition of dataset in the experiment. Grids with different color backgrounds represent different types of datasets. These are as follows:

Dataset *U*: Real unpacked malware, 1500 samples.
Dataset *P*: Packed malware from 30% of Dataset U, 450 samples.
Dataset *F*: Fake packed samples generated by packer GAN, 500 samples.

All unpacked and packed malware samples were sent to the Cuckoo sandbox for dynamic analysis. The ApiStats features were then extracted from the sandbox analysis reports.

Table 3.
Data of single-packer malware dataset.

No.	Family	Class	Unpacked dataset	Packed dataset	Total
1	Bundlore	1	184	56	240
2	Lydra	2	177	51	228
3	SoftPulse	3	158	30	201
4	Fosniw	4	171	54	217
5	DownloadWare	5	163	48	206
6	Firseria	6	152	42	194
7	Koutodoor	7	145	42	187
8	Rebhip	8	132	37	169
9	ForceStartPage	9	118	40	158
10	Conficker	10	100	50	150
Total			1500	450	1950

5.2. Impact of malware packing

5.2.1. Effect of packing on program properties

Before testing the model, we analyzed the impact of the packing process on the program. The process that compiles malware source code into a PE file is a mapping function, $= f(x, c)$. The x is the malware source code, c is the family of the malware, y is the executable file generated by compiler and f is a function that converts malware source code to a PE file. The family classification of unpacked malware is relatively mature, so we focus on the influence of the packing function p on the detection results. Similar source code should have similar PE structures, so malware can be distinguished by extracting PE features. The packing process is also a mapping function, $z = p(f(x, c))$. When the malware is executed, the unpacking function q is called to release the original malware program, $y = q(z)$.

We provide a very simple C++ program to illustrate the impact of packing on program characteristics. This program determines whether the input number is positive or negative. The logical branch of this program is so simple that its assembly instructions can be clearly seen through reverse analysis of its unpacked file. IDA Pro as a powerful disassembler can generate assembly language source code from machine-executable code and make this complex code more human-readable. The graph view of IDA Pro shows the logical flow of a program's assembly instructions. Fig. 11 shows an example of malware packed by Aspack and UPX. Aspack is an advanced EXE packer created to compress Win32 executable files and provides protection to programs from unprofessional analysis. The program first calls the *JLE* instruction to determine whether the input number is greater than zero. Then the program calls *JNS* to determine whether the number is greater than 0. The graph view (1) of the original demo is simple. Many redundant codes have been added to the packed program, and the execution logic of the original program has been significantly changed. After the executable file is packed by Aspack, its graph view (2) is complex and difficult to analyze. As can be seen in view (3), the logic flow of UPX is more complex than that of Aspack.

(1) Assembly opcode logic flow chart of original demo program. (2) Assembly opcode logic flow chart of demo program packed by Aspack. (3) Assembly opcode logic flow chart of demo program packed by UPX.

To clearly observe the impact of packing on the program, we

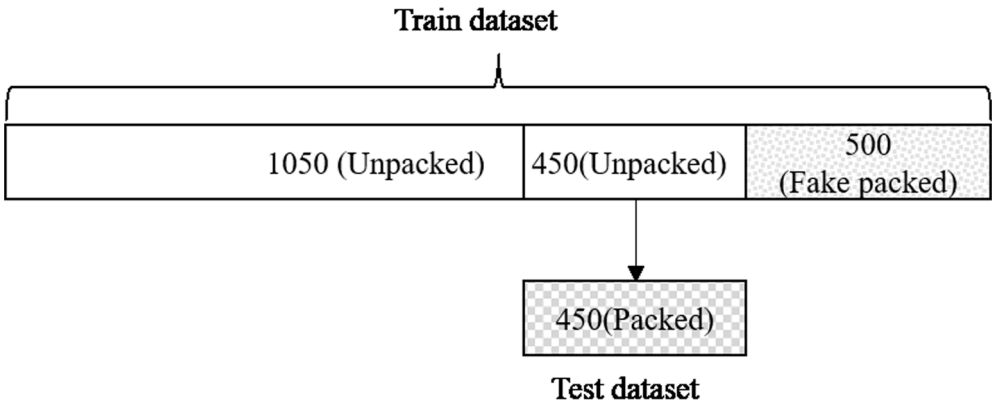


Fig. 10. Composition of experimental dataset.

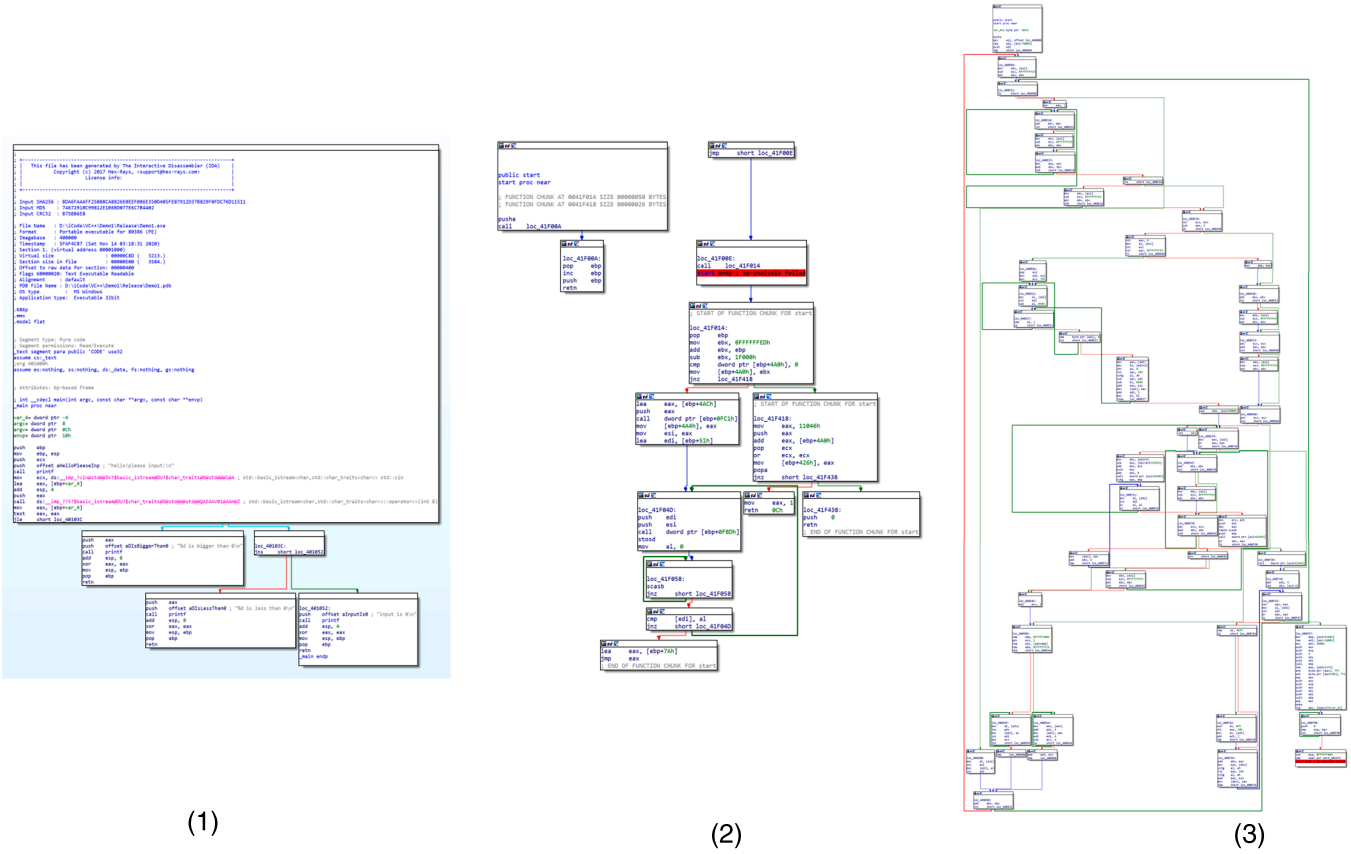


Fig. 11. Assembly instruction graphs of demo program.

compare the PE properties of the packed and unpacked demo program in Table 4. Because the program is compressed by UPX, the file size after packing becomes smaller. The number of sections is reduced, and the *.text*, *.rdata*, and *.data* sections in the original program are hidden in the packed program. The average entropy of the packed program increases, indicating that the code after packing is more complex. In addition, the number of imported APIs is reduced from 44 to 13, which will cause great interference in static analysis. Aspack added two sections (*.aspack and .adata*) after packing the demo program, resulting in a larger file. For security analysts, the more complex the logic flow chart of the program, the more difficult it is to be analyzed. By comparison, UPX has better packing effect than Aspack, and therefore UPX is used in subsequent experiments.

5.2.2. Impact of malware packing on classification

If only the unpacked samples are used for training, what will be the detection effect of the classifier on the packed samples? To get the answer, the classifier was trained with unpacked samples, and was tested with packed samples to determine the impact. In Fig. 12 (1), the classification effect based on opcode static features is very poor, and most packed samples are wrongly grouped into classes SoftPulse and Firseria. The confusion matrix shows that static features are not fit for family classification of packed samples but are more suitable for binary classification. In Fig. 12 (2), the classification effect based on WinAPI

Table 4.
PE property comparison of demo program packed by UPX and Aspack.

PE properties	Demo.exe (unpacked)	packed by UPX)	packed by Aspack)
File size (Byte)	9,728	7,168	12,288
Sections	6(.text, .rdata, . data, .gfids, .rsrc, . reloc)	3(UPX0, UPX1, . rsrc)	8(.text, .rdata, .data, .gfids, .rsrc, .reloc,.aspack,. adata)
Size of code	3584	8192	3584
Mean section entropy	3.4137	3.7675	3.4084
Min section entropy	0.1591	0.0	0.0
Max section Entropy	5.8884 (.text)	7.1761 (UPX1)	7.4193(.rdata)
Mean section rawsize	1450.6667	2048.0	1408.0
Import API	44	13	10
Import DLL	8	8	8
Size of image	28,672	45,056	40,960

dynamic features is slightly better than that of static features, but it is still not ideal.

In Table 5, the accuracy of static feature classification is only 13.77%, indicating that the static features of a sample after being packed are very different from the original static features. The classification accuracy based on dynamic features is 52.44%, indicating that dynamic analysis has a certain detection effect on packed samples. The test results show that the influence of packing on the static analysis method is much greater than on the dynamic analysis method. It also proves that dynamic analysis is more robust and defensive than static analysis.

5.3. Detection results of malware packer detector

According to the above analysis, we can use static analysis method to detect whether the malware is packed. We used IDA to disassemble the malware and generate assembly instructions. Then, we extracted the top 100 n-gram assembly opcode sets as static features. The 1500 unpacked (label:0) samples and 450 packed (label 1) samples are combined into a dataset, which was divided into training and test sets at a ratio of 80:20. The malware packer detector was trained to learn the difference between packed and unpacked samples. Then test data were used to detect and predict whether a test sample is packed.

After the samples were disassembled, the opcode n-gram features were extracted. The best accuracy of the malware packer detector was achieved when dropout set 0.4 is 98.20% in Table 6. The results show that the model based on static features can accurately distinguish between unpacked and packed samples.

5.4. Training of packer GAN

We trained a packer GAN to generate fake packed samples in this section. In the following section, we will compare the improvement effect of packer GAN on malware classifier. The features of each malware family are different, so it is necessary to generate fake samples separately. The training performance of 10 packer GANs is shown in Fig. 13. The red and green lines show the accuracy of the discriminator in detecting real and fake packed samples, respectively. It is obvious from the figure that the accuracy of the discriminator for real samples decreases, while the accuracy of the discriminator for fake samples is improving.

The training accuracy of each packer GAN is listed in Table 7. The average accuracy of discriminators is 33.15% for real samples and 83.17% for fake samples. This shows that the discriminators have difficulty in distinguishing the fake samples generated by the generators from the real samples. Therefore, we generated 50 fake packed samples for each malware family using the trained packer GAN for training.

Principal components analysis (PCA) aims to transform multiple indicators into a few comprehensive indicators by using the idea of dimension reduction. PCA is often used to reduce the dimension of the dataset while maintaining the characteristics of the dataset with the largest contribution to each other's difference. To compare the differences between packer GAN generated samples and real packed samples,

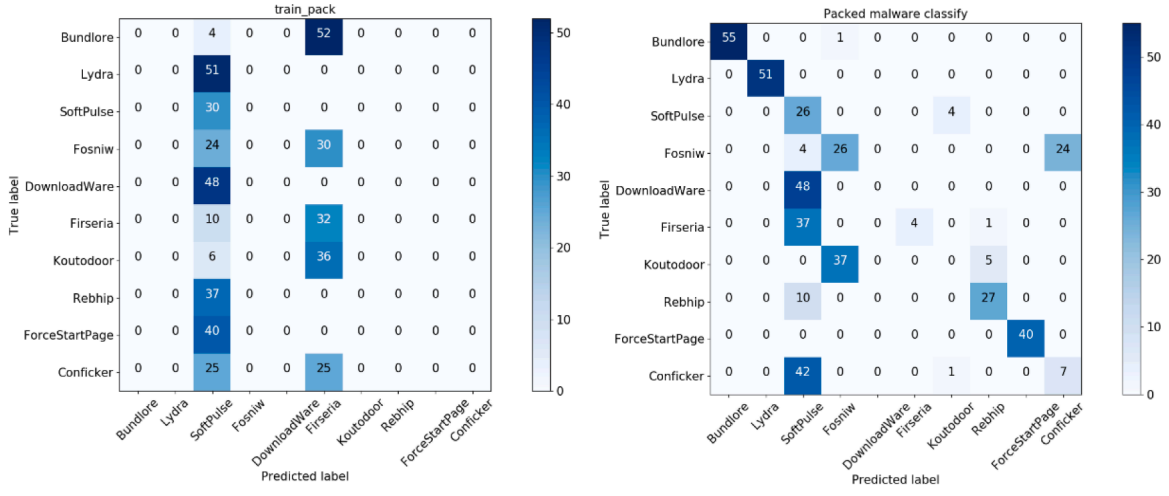


Fig. 12. Unpacked classifier to test packed samples with static and dynamic features.

Table 5.
Classification results of packed samples with unpacked samples (%).

Method	Accuracy	Precision	Recall	F1	Train:Test	Feature
Classification DNN	13.77	0.24	13.77	4.06	1500:450	Opcode
Classification DNN	52.44	57.14	52.44	49.48	1500:450	WinAPI

Table 6.
Test results of malware packer detector (%).

Method	Accuracy	Precision	Recall	F1	Train:Test	Feature	Dropout
Packer detector	95.64	96.27	95.64	95.74	1560: 390	x-gram	0
Packer detector	96.66	97.12	96.66	96.75	1560: 390	x-gram	0.1
Packer detector	97.94	98.11	97.94	97.97	1560: 390	x-gram	0.2
Packer detector	97.69	97.91	97.69	97.73	1560: 390	x-gram	0.3
Packer detector	98.20	98.32	98.20	98.22	1560: 390	x-gram	0.4
Packer detector	96.41	96.87	96.41	96.49	1560: 390	x-gram	0.5

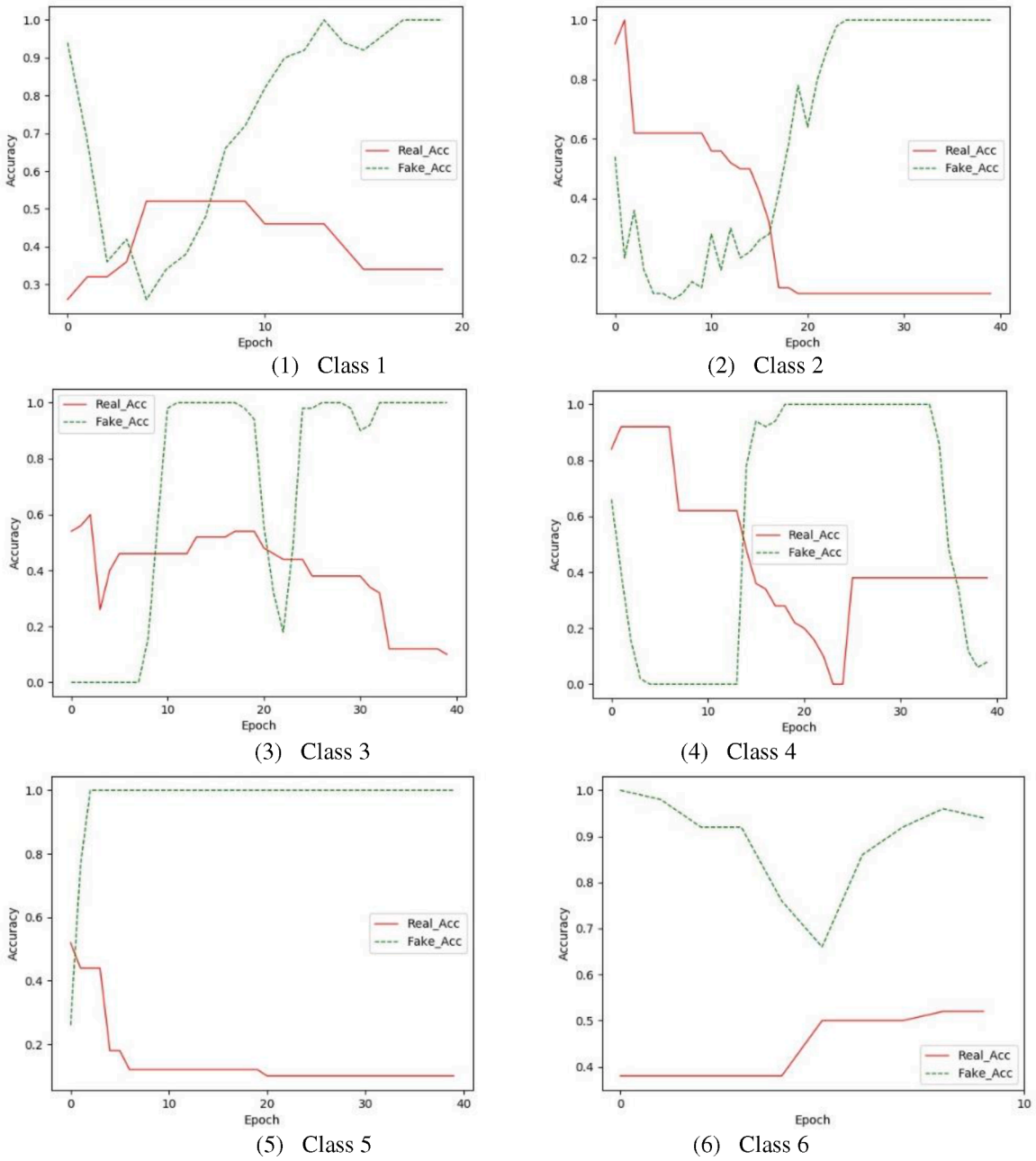


Fig. 13. Training performances of 10 packer GANs

we used PCA to visually compare the two samples in Fig. 14. The figure (1) shows the distribution of unpacked samples and real packed samples, and the figure (2) shows the distribution of unpacked samples and fake packed samples.

The features' mean value of the unpacked samples is -1.7053e-16 after PCA dimensionality reduction. The features' mean value of the real packed samples is 5.2632e-17, and the features' mean value of the fake samples is 8.1820e-17. It can be clearly seen that the average eigenvalues of fake packed samples are more like those of real packed samples. This result shows that the samples generated by packer GAN have

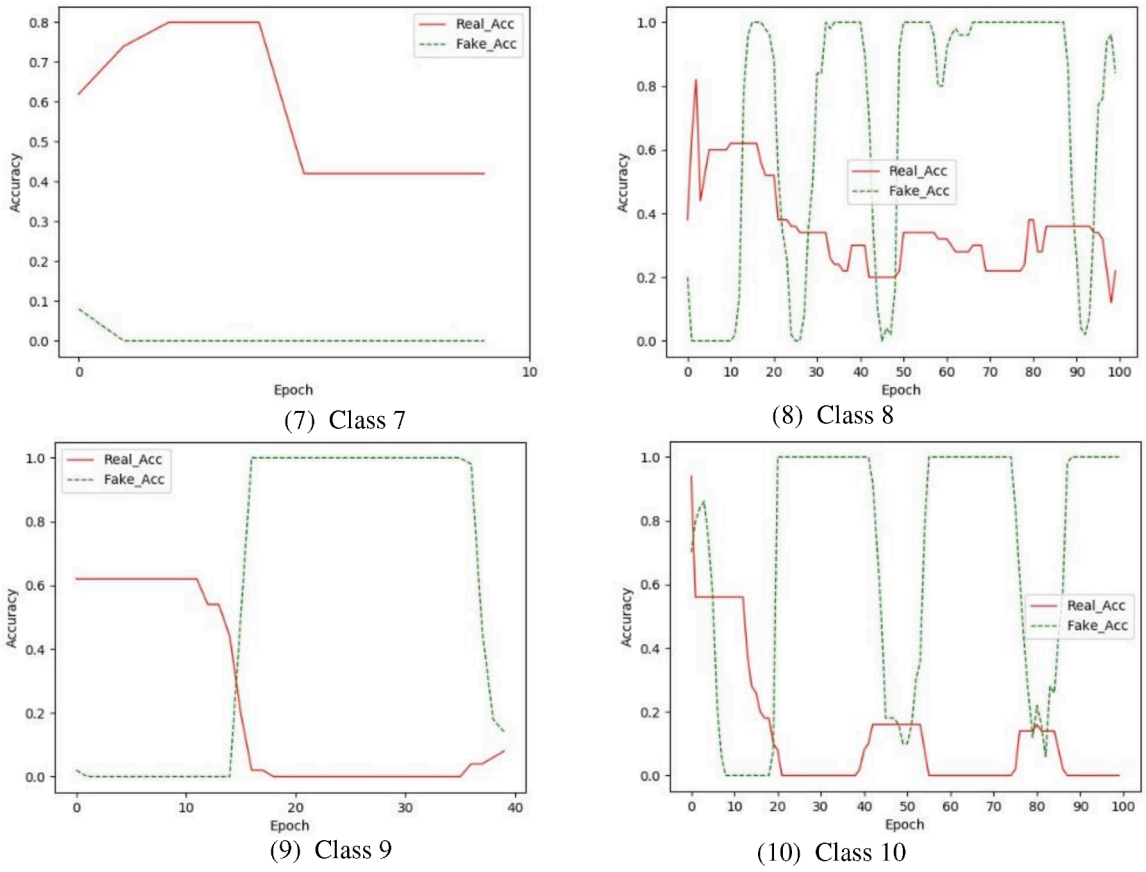


Fig. 13. (*continued*).

Table 7.
Results of Packer GAN training (%).

GAN Class	Real accuracy	Fake accuracy	Generated samples
1	18.49	91.49	50
2	17.49	99.09	50
3	44.90	48.69	50
4	38.09	92.99	50
5	9.39	86.0	50
6	49.0	85.29	50
7	62.99	81.70	50
8	30.19	72.30	50
9	28.89	80.60	50
10	32.09	93.59	50

high similarity with the real samples, and can be used for model training.

5.5. Test of malware classifier

The malware classifier we designed can classify both unpacked and packed samples with dynamic features. To verify whether the classifier meets the design purpose, we only use unpacked samples to verify in the first step. Then we test the classifier with packed samples. Finally, we evaluate the improvement effect of packer GAN on the classifier.

5.5.1. Classify unpacked malware

The 1500 unpacked samples of dataset U are sent to the Cuckoo sandbox for analysis. The top 100 WinAPIs were extracted from the sandbox report as the dynamic features for the classifier. The confusion matrix of the classifier for unpacked malware is listed in Fig. 15. As can be seen from the figure, the test samples of most families are correctly classified.

The test results of the malware classifier for unpacked samples are shown in Table 8. The experiment conducted 5-folds cross-validation to evaluate the effect of the model. All the data are divided into five parts, and each part has 20% data. In each round of experiment, the training data includes 1200 samples from four parts, and the test data includes 300 samples from another part. The cross-validation accuracy of the malware classifier for unpacked samples reached 91.66%. The result proves that WinAPI features are fit for malware classification.

5.5.2. Classify packed malware

To test the detection effect of classifier on packed samples, the experiment was divided into two stages, as shown in Fig. 16. First, we tested the classifier on real packed dataset. Secondly, we used packer GAN to generate fake packed dataset to improve the classifier's performance.

In the first stage, datasets U and P were combined into a new dataset that included 1500 real unpacked samples and 450 real packed samples. A total of 1950 samples were sent to the sandbox for dynamic analysis and feature extraction. They are divided into training set and test set in a ratio of 4:1. The packed malware classifier used these real training samples and test samples to evaluate the classification effect. The test results of this stage are listed in Fig. 17 (1) and Table 9 (1).

In the second stage, we evaluate the effect of packer GAN. After packer GAN training, it generates 50 fake packed samples for each family. As shown in Fig. 14 (2), those fake samples were very similar to real samples. Then they were added to the training dataset and tested again to evaluate the effectiveness of our model. The test results of the second stage are listed in Fig. 17 (2) and Table 9 (2).

According to the data in Table 9, the accuracy of the classifier on the original 1950 dataset reached 94.1%. It can be seen from Fig. 17 (1) that

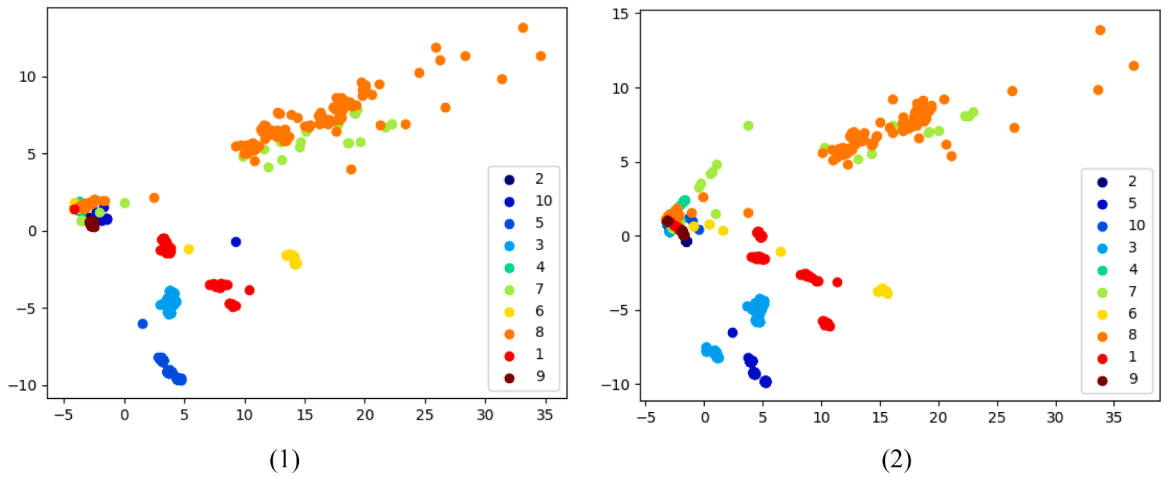


Fig. 14. PCA of malware samples.

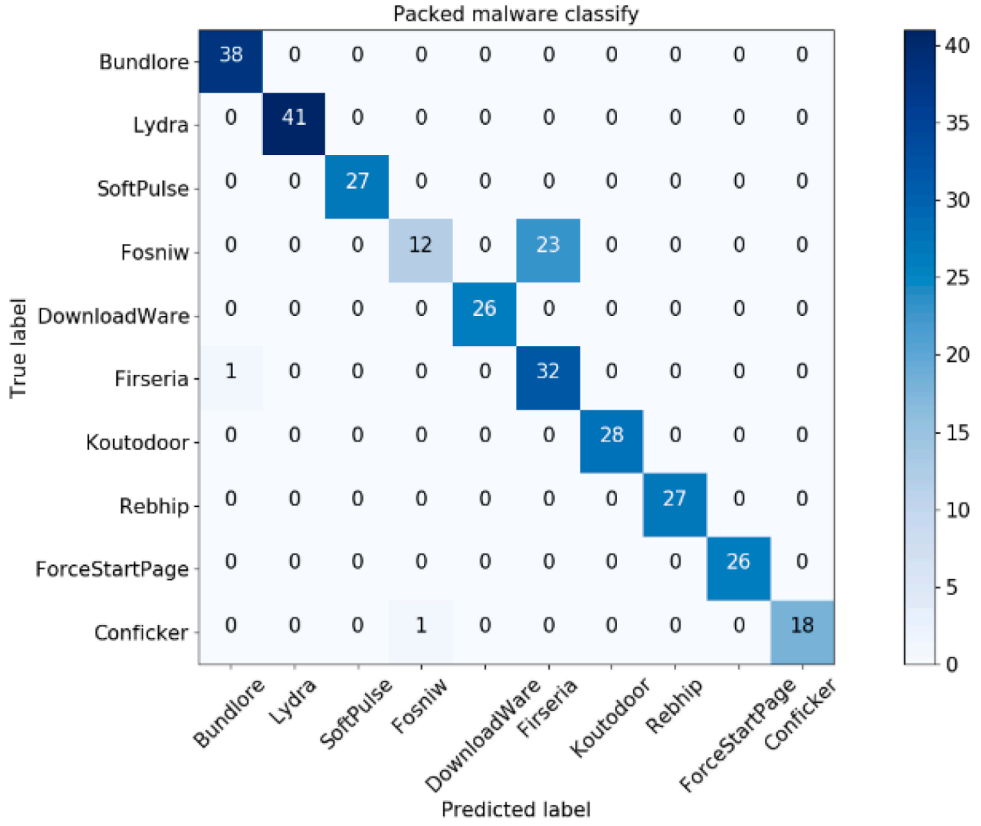


Fig. 15. Confusion matrix of classifying unpacked malware.

Table 8.
Results of malware classifier for unpacked malware (%).

Method	Accuracy	Precision	Recall	F1	Train: Test	Features
Classify Unpacked	91.66	94.17	91.66	90.83	1200:300	WinAPI

there were many false positives in the classification results of class Fosniw. Therefore, packer GAN was used to generate 50 more training samples for class Fosniw. When we added 550 fake samples generated by packer GANs to the training set, the classification accuracy improved to 97.8%, as shown in Fig. 17 (2).

The training and test data of the two test stages are 1560: 390 and 2000:500 respectively. This result shows that increasing the training samples by packer GAN can improve the performance of the classifier.

5.6. Test on multi-packer dataset

5.6.1. Multi-packer dataset

To fully verify the effectiveness of our model, we selected 3233 samples from VXHeaven to build a multi-packer dataset. As shown in the Fig. 18, it includes 361 benign files (A) and 2872 malware (B) from 5 families. To simulate the real packing scene, this data set mainly includes three types of samples: unpacked benign files (A), unpacked malware (C) and packed malware (D) with several packers. These

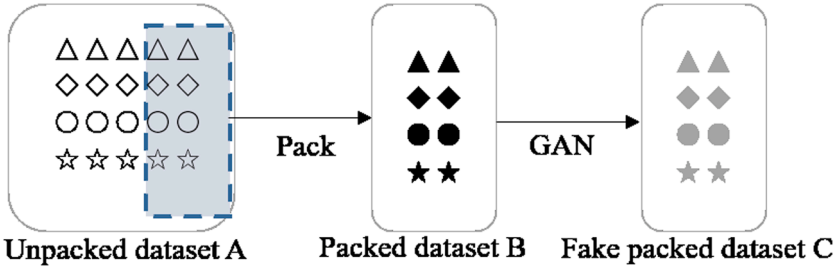
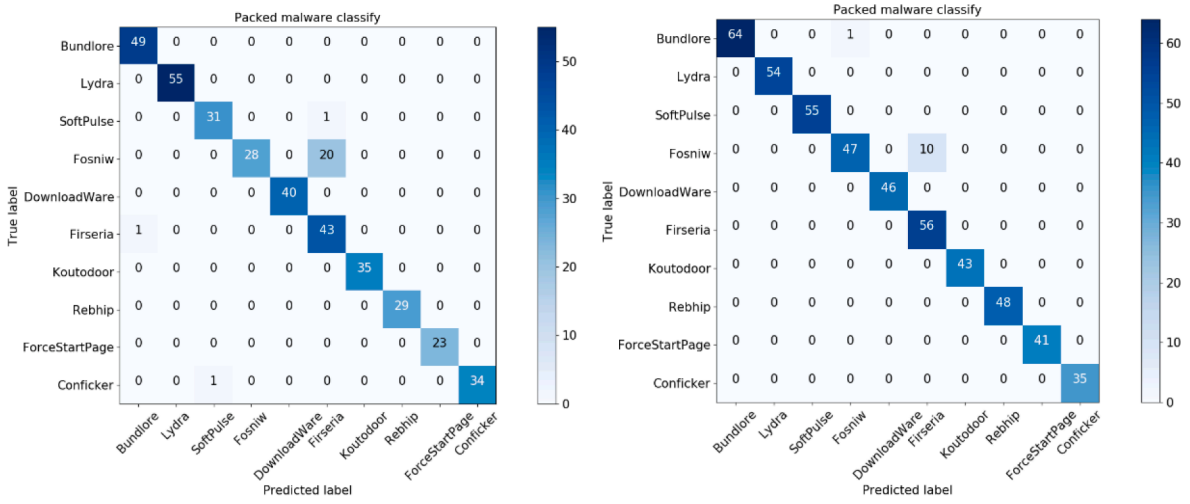


Fig. 16. Packer GAN generated samples.



(1) (2)

Fig. 17. Comparison of classification effect before and after using GAN.

Table 9.
Results of packed malware classifier (%).

Method	Accuracy	Precision	Recall	F1	Train: Test	Features
Classify without GAN(1)	94.10	95.79	94.10	93.94	1560: 390	WinAPI
Classify with GAN (2)	97.80	98.06	97.80	97.78	2000:500	WinAPI

samples were randomly divided into training and test data sets in a ratio of 80%:20%. In addition, we generated 100 fake packed samples(E) for each malware family using packer GAN, a total of 500.

There are 594 malware packed by several packers in the dataset. These packers include Armadillo, ASPack, ASProtect, NsPacK, FSG and UPX. The family classes of malware include: Constructor, Email-Worm, Hoax, Rootkit, Backdoor. The data of samples for each family and the data of packed samples are described in detail in Table 10.

5.6.2. *Test result of multi-packer dataset*

We use this dataset to evaluate the effect of the MaliCage in the simulation environment. First, we send all samples to sandbox for dynamic analysis to extract API call features. Packer GAN is used to generate 500 fake packed samples for model training.

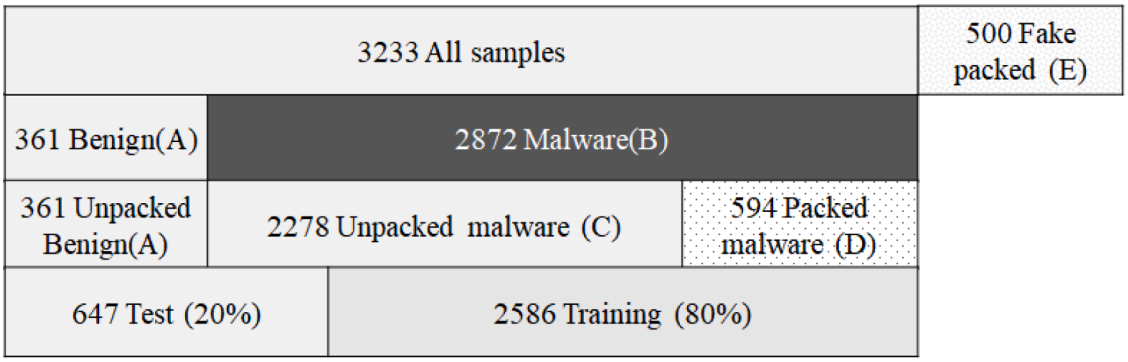


Fig. 18. Components of multi-packer dataset.

Table 10.
Multi-packer dataset.

Family	Benign	Constructor	Email-Worm	Hoax	Rootkit	Backdoor	Total
Label	0	1	2	3	4	5	/
Unpacked	361	433	475	547	380	443	2639
Packed	0	134	131	65	37	227	594
Armadillo	0	21	39	26	22	95	203
ASPack	0	15	7	9	0	19	50
ASProtect	0	9	0	0	10	20	39
FSG	0	8	10	0	0	8	26
NsPacK	0	27	0	0	0	13	40
UPX	0	54	75	30	5	72	236
Samples	361	567	606	612	417	670	**3233**

The nonlinear dimensionality reduction algorithm t-distributed stochastic neighbor embedding (t-SNE) is suitable for reducing high-dimensional data to 2D or 3D for visualization. T-SNE is calculated to observe the distribution of the dataset with dynamic features in Fig. 19. According to figure (1) on the left, the distribution of real packed samples is uneven, which may affect the result of classification. The distribution of fake samples on the right figure (2) is relatively concentrated, so it is helpful for classification.

We first evaluate the detection effect of the model on unpacked samples, then evaluate the impact of the packed samples on the model, and finally evaluate the improvement effect of packer GAN on the model. We divide the experiment into the following six stages for analysis and comparison. The test results are described in detail in Table 11.

(1) Classify unpacked samples

Firstly, we classify the unpacked samples and detect the basic functions of our DNN classifier. The samples come from component A and C. The test accuracy on the unpacked samples reaches 75.21%.

(1) Classify all samples

Next, we put 594 packed malware of component D into the dataset for testing to verify the detection effect of the classifier in the mixed mode. After testing, the accuracy of the model is reduced to 62.28%. It shows that the packed samples have a significant impact on the existing classifiers.

Table 11.
Results of multi-packer datasets (%).

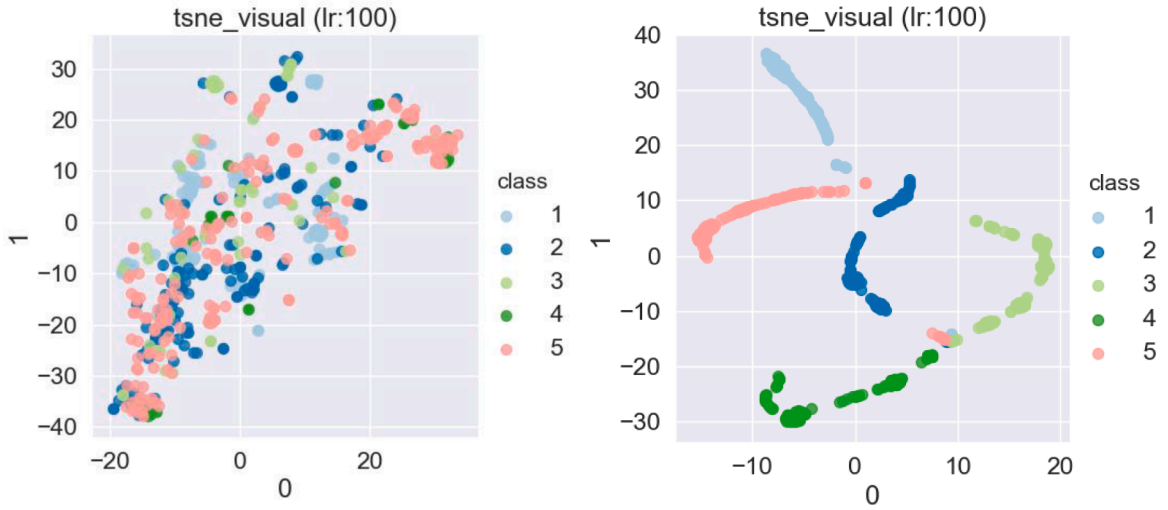
Method	Accuracy	Precision	Recall	F1	Samples	Dataset
Classify all unpacked samples	75.21	77.31	75.21	75.81	2639	2111:528
Classify all samples	62.28	64.96	62.28	62.92	3233	2586:647
Classify malware without GAN	66.60	74.11	66.60	66.93	2872	2297:575
Classify malware with GAN	71.25	74.44	71.25	71.55	3372	2697:675
Classify packed malware with GAN	82.19	82.77	82.19	82.16	1094	875:219

(1) Classify malware without GAN

We are most concerned about the classification effect of the model on malware. Therefore, at this stage, all unpacked and packed samples are combined into component B for testing. The accuracy of the model is 66.60%.

(1) Classify malware with GAN

To improve the accuracy of the classifier, we use packer GAN to



(1) Real packed samples. (2) Fake packed samples.

Fig. 19. T-SNE of malware samples.

generate 100 fake packed samples for each family malware to form component E. Then we combined component E and B for testing, and the accuracy was improved to 71.25%.

(5) Classify packed malware with GAN

Finally, we add the 500 fake samples of component E to the real packed sample of component D to evaluate the promotion effect of packer GAN on packed malware. The accuracy of the test is 82.19%, which proves that packer GAN can significantly improve the training effect of the model.

Through the test on the multi-packer dataset, it is proved that packed samples have a significant impact on malware classifier. The unpacked and packed samples should be analyzed differently in model training. The result also proved that our framework based on DNN, and GAN can effectively solve the challenge of packed malware classification.

5.7. Comparison

Recently, there have been some family detection methods for packed malware. To assess our method's advantages, we make a comparison with similar research in Table 12. Various research methods use different datasets, which makes it difficult to objectively compare the detection accuracy. Therefore, this comparison focuses on the following five key capabilities of these packed malware detection methods.

(1) Direct analysis: Whether the model must rely on the unpacking tools to analysis the packed samples.
(2) Anomaly detection: Whether the model can distinguish between packed malware and normal software.
(3) Packed detection: Whether the model can judge whether a sample is packed.
(4) Packed classification: Whether the model can classify the families of packed malware. It is the most critical capability of packed malware detection.
(5) Extend dataset: Whether the model can expand the packed samples based on unpacked dataset. Because the real packed malware is difficult to obtain, the model must try to solve the problem of insufficient training samples.

The state-of-the-art studies on packed malware usually focus on the basic problem of judging whether malware are packed [2,22,25,53]. Alkhateeb et al. [2] dynamically traced API features from 7000 malware samples obtained from VirusTotal. Their Naive Bayes classifier can only distinguish between benign and malware samples. Hubballi et al. [22] proposed a semi-supervised technique to detect the packed executable file. They randomly selected executable files from Malicia dataset and manually packed those samples with different packers. The method can only find out which packer was used in packing the program but has no ability to classify its family. Kim et al. [25] collected malware from VirusShare and extracted 13 important features to detect them whether packed or not. Because most features are statically extracted from the entropy and header of PE file, this method is easy to be evaded. Zhang et al. [53] compared malware and benign software in a sandbox environment, then adopted the deep belief network to adaptively train a detection model to detect packed malware. Their method can only distinguish packed samples, not family classification.

In recent years, there are a few family classification methods for packed malware. Cesare et al. [10] employed the flowgraph matching algorithm to classify 15,000 packed and polymorphic malware collected from honeypots. They used a fast application-level emulator to reverse the malware packing transformation. However, their method faces a serious challenge, that is, once the reversing process fails, it will not be able to detect those packed malware. Hua et al. [21] analyzed 600 malware from VirusShare dynamically and apply an algorithm to strip the unpacking routine from the graph. They use Deep graph Convolutional Network (DGCNN) to classify packed malware with control flow graph data. Their method must get packed samples for dynamic analysis, and only 600 samples were used for testing. The flexibility and accuracy of their method are worse than ours.

Through comparison, our model has the following advantages in practical scenarios and detection capabilities.

(1) MaliCage detects packed malware by static and dynamic analysis without unpacking in advance. Therefore, our model can detect all kinds of packed samples, including private packers.
(2) Our model can perform anomaly detection and distinguish packed malware from benign software.
(3) The packer detector of MaliCage can accurately distinguish unpacked and packed samples, and provide input for subsequent malware classification.
(4) MaliCage can accurately classify the packed and unpacked samples at the same time.
(5) We innovatively designed packer GAN to generate fake packed samples to improve the accuracy of the detection model.

5.8. Discussion

Due to the full analysis of the theory and influence of malware packing, MaliCage is superior to other methods. Fig. 20 shows the theoretical detection model of MaliCage.

In order to complete the attack task, malware needs to execute a code sequence: $x_o = (a^1, a^2, \ldots, a^n)$. Malware of same family has similar program code: (x_o, y_o). Security analyst can predict whether a software is malware by designing a detection function: $y = f_c(x_o)$, and classify it into families. In order to evade security detection, attackers often use packing tools to hide the original code sequence of malware. However, to ensure the packed malware still execute normally, the packed malware needs to add a piece of unpacking code: $p = (p^1, p^2, \ldots, p^m)$. When the packed malware is executed, it will run the unpacking code sequence firstly, and then execute its original code sequence: $x_p = x_o \oplus p = (p^1, p^2, \ldots, p^m, a^1, a^2, \ldots, a^n)$. The code sequence of the packed malware x_p is obviously different from the original code sequence x_o, which will lead to prediction error of the original classification function: $y_p = f_c(x_p) \neq y_o$. The root cause of model false positives is that the code sequence of the packed software has changed.

The features of the packed and unpacked samples of the same family can be aggregated into two groups, so they need to be detected separately. All the packed samples have similar unpacking code p, so their static features are also very similar. Therefore, it is necessary to design a

Table 12.
Comparison with similar studies.

Methods	Direct analysis	Anomaly detection	Packed detection	Packed classification	Extend dataset	Feature	Algorithm
Alkhateeb et al. [2]	√	√ (97%)	×	×	×	API call	Naive Bayes
Hubballi et al. [22]	√	×	√ (96%)	×	×	PE sections	Cluster
Kim et al. [25]	√	×	√ (96.1%)	×	×	PE header	kNN
Zhang et al. [53]	√	√ (92.6%)	√	×	×	System call	DBN
Cesare et al. [10]	×	×	×	√ (88%)	×	Control graphs	Set similarity
Hua et al. [21]	√	×	×	√ (96.4%)	×	Control graph	DGCNN
MaliCage	√	√	√ (98.2%)	√ (97.8%)	√	Opcode, WinAPI	DNN, GAN

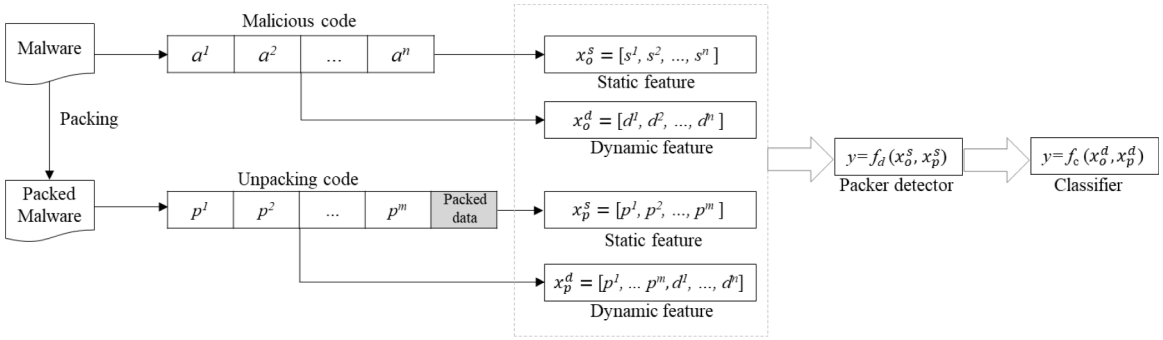


Fig. 20. Theoretical detection model of MaliCage.

function f_d to distinguish the packed samples from the unpacked samples. Then we need to design a packed malware classification function f_c to classify the packed samples. Dynamic analysis can not only extract the features of packed samples: x_o, but also extract the features of packed samples: x_p. However, it is difficult to obtain enough packed samples, so packer GAN is designed to generate fake packed samples for f_c training: $x_{fp} = g(x_o)$. Therefore, our method not only has a variety of detection capabilities, but also has higher detection accuracy.

6. Conclusion

Packed malware has brought significant challenges to malware detection. We analyzed and tested the packer's impact on malware characteristics and detection results. Although static analysis and dynamic analysis are vulnerable to malware packing, they also have unique values in detection packed malware. We tried to combine them to form a robust detection framework for packed malware. To detect packed malware variants, it is necessary to ensure that packed samples are provided in the training dataset. Unfortunately, most malware detection models in the past do not consider the problem of packed training samples. We proposed a novel packed malware family classification framework MaliCage based on DNN and GAN. The packer detector based on assembly opcode features accurately predicted whether a program was packed. Then the packed malware classifier extracted dynamic features from the sandbox reports and classified the packed malware with DNN. The packer GAN was designed to generate fake packed malware training samples for improving the performance of malware classification. Two datasets including single UPX packer and multi-packers are built and used to experiment. The experimental results showed that the influence of packing on static analysis was much greater than on dynamic analysis. The accuracy of packer detector reached 98.2% and the accuracy of packed malware classifier reached 97.8%. Experiments show that our model can well solve the challenge of detection and classification of packed malware, and its detection accuracy is better than the existing methods. In the future, we plan to use more packed samples to optimize MaliCage and apply it to the real environment.

CRediT authorship contribution statement

Xianwei Gao: Conceptualization, Methodology, Software, Validation, Data curation, Writing – original draft, Writing – review & editing. **Changzhen Hu:** Supervision, Project administration. **Chun Shan:** Resources, Funding acquisition. **Weijie Han:** Visualization, Investigation.

Declaration of Competing Interest

The authors declare that they have no competing interests.

Data availability

The authors do not have permission to share data.

Acknowledgment

This work is supported by the National Key R&D Program of China (Grant no. 2016YFB0800700), and the National Natural Science Foundation of China (Grant no. U1636115).

References

[1] Afianian A, Niksefat S, Sadeghiyan B, Baptiste D. Malware dynamic analysis evasion techniques: A survey. ACM Computing Surveys (CSUR) 2019;52(6):1–28.
[2] Alkhateeb E M, Stamp M. A Dynamic Heuristic Method for Detecting Packed Malware Using Naive Bayes[C]. In: 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA); 2019.
[3] Alshamrani A, Myneni S, Chowdhary A, Huang D. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. IEEE Communications Surveys & Tutorials 2019;21(2):1851–77.
[4] Dong S, Xia Y, Peng T. Traffic identification model based on generative adversarial deep convolutional network. Ann Telecommun 2021:1–15.
[5] Arivudainambi D, KA VK, Visu P. Malware traffic classification using principal component analysis and artificial neural network for extreme surveillance. Comput Commun 2019;147:50–7.
[6] Bahtiyar Ş, Yaman MB, Altınığne CY. A multi-dimensional machine learning approach to predict advanced malware. Computer Networks 2019;160:118–29.
[7] Biondi F, Enescu M A, Given-Wilson T, et al. Effective, efficient, and robust packing detection and classification [J] Computers & Security 2019;85:436–51.
[8] Botacin M, Geus PLD, Grégio A. Enhancing branch monitoring for security purposes: From control flow integrity to malware analysis and debugging. ACM Transactions on Privacy and Security (TOPS) 2018;21(1):1–30.
[9] Burnap P, French R, Turner F, Jones K. Malware classification using self organising feature maps and machine activity data. Computers & Security 2018;73:399–410.
[10] Cesare S, Xiang Y, Zhou W. Malwise—an effective and efficient classification system for packed and polymorphic malware. IEEE Trans Comput 2012;62(6):1193–206.
[11] Chakkaravarthy SS, Sangeetha D, Vaidehi V. A Survey on malware analysis and mitigation techniques. Computer Science Review 2019;32:1–23.
[12] Cheng B, Ming J, Leal E A, et al. Obfuscation-Resilient Executable Payload Extraction from Packed Malware [C]/. In: /30th {USENIX} Security Symposium; 2021 ({USENIX} Security 21).
[13] D'Elia DC, Coppa E, Palmaro F, Cavallaro L. On the dissection of evasive malware. IEEE Trans Inf Forensics Secur 2020;15:2750–65.
[14] Dai Y, Li H, Qian Y, Lu X. A malware classification method based on memory dump grayscale image. Digital Investigation 2018;27:30–7.
[15] Ding Y, Xia X, Chen S, Li Y. A malware detection method based on family behavior graph. Computers & Security 2018;73:73–86.
[16] Ding Y, Wu R, Zhang X. Ontology-based knowledge representation for malware individuals and families. Computers & Security 2019;87:101574.
[17] Drew J, Hahsler M, Moore T. Polymorphic malware detection using sequence classification methods and ensembles. EURASIP Journal on Information Security 2017;2017(1):1–12.
[18] Du P, Sun Z, Chen H, Cho JH, Xu S. Statistical estimation of malware detection metrics in the absence of ground truth. IEEE Trans Inf Forensics Secur 2018;13(12):2965–80.
[19] Gibert D, Mateu C, Planes J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J Netw Comput Appl 2020;153:102526.
[20] Guo W, Mu D, Xu J, Su P, Wang G, Xing X. Lemna: Explaining deep learning based security applications. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security; 2018. p. 364–79.

[21] Hua Y, Du Y, He D. Classifying Packed Malware Represented as Control Flow Graphs using Deep Graph Convolutional Neural Network. In: 2020 International Conference on Computer Engineering and Application (ICCEA). IEEE; 2020. p. 254–8.
[22] Hubballi N, Dogra H. Detecting Packed Executable File: Supervised or Anomaly Detection Method?. In: 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE; 2016. p. 638–43.
[23] Goodfellow Ian, Pouget-Abadie Jean, Mirza Mehdi, Xu Bing, Warde-Farley David, Ozair Sherjil, Courville Aaron, Bengio Yoshua. Generative adversarial nets. Advances in neural information processing systems. 2014. p. 2672–80. pages.
[24] Jordaney R, Sharad K, Dash SK, Wang Z, Papini D, Nouretdinov I, Cavallaro L. Transcend: Detecting concept drift in malware classification models. In: 26th {USENIX} Security Symposium ({USENIX} Security 17); 2017. p. 625–42.
[25] Kim JW, Namgung J, Moon YS, Choi MJ. Experimental Comparison of Machine Learning Models in Malware Packing Detection. In: 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE; 2020. p. 377–80.
[26] Korczynski D, Yin H. Capturing malware propagations with code injections and code-reuse attacks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security; 2017. p. 1691–708.
[27] Liang G, Pang J, Shan Z, Yang R, Chen Y. Automatic benchmark generation framework for malware detection. Security and Communication Networks 2018; 2018.
[28] Liu H, Guo C, Cui Y, et al. 2-SPIFF: a 2-stage packer identification method based on function call graph and file attributes [J] Applied Intelligence 2021:1–16.
[29] Maleki N, Bateni M, Rastegari H. An improved method for packed malware detection using PE header and section table information. International Journal of Computer Network and Information Security 2019;11(9):9–17.
[30] Mirza QKA, Awan I, Younas M. CloudIntell: An intelligent malware detection system. Future Generation Computer Systems 2018;86:1042–53.
[31] Nataraj L, Manjunath BS. Spam: Signal processing to analyze malware [applications corner] IEEE Signal Process Mag 2016;33(2):105–17.
[32] Nicho M, Alkhateri M. Modeling Evasive Malware Authoring Techniques. In: 2021 5th Cyber Security in Networking Conference (CSNet). IEEE; 2021. p. 71–5.
[33] Noor M, Abbas H, Shahid WB. Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. J Netw Comput Appl 2018;103:249–61.
[34] Nunes, M., Burnap, P., Rana, O., Reinecke, P., & Lloyd, K. Getting to the root of the problem: a detailed comparison of kernel and user level data for dynamic malware analysis. *Journal of Information Security and Applications,*48, 102365-102365.
[35] Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic malware analysis in the modern era—A state of the art survey. ACM Computing Surveys (CSUR) 2019;52 (5):1–48.
[36] Pektaş A, Acarman T. Malware classification based on API calls and behaviour analysis. IET Inf Secur 2018;12(2):107–17.
[37] Shi H, Mirkovic J, Alwabel A. Handling anti-virtual machine techniques in malicious software. ACM Transactions on Privacy and Security (TOPS) 2017;21(1): 1–31.
[38] Dong S, Wang P, Abbas K. A survey on deep learning and its applications. Computer Science Review 2021;40:100379.
[39] Stiborek J, Pevný T, Rehák M. Multiple instance learning for malware classification. Expert Syst Appl 2018;93:346–57.
[40] Sun L, Versteeg S, Boztaş S, Yann T. Pattern recognition techniques for the classification of malware packers. In: Australasian Conference on Information Security and Privacy. Berlin, Heidelberg: Springer; 2010. p. 370–90.
[41] Tang M, Qian Q. Dynamic API call sequence visualisation for malware classification. IET Inf Secur 2019;13(4):367–77.
[42] Ucci D, Aniello L, Baldoni R. Survey of machine learning techniques for malware analysis. Computers & Security 2019;81:123–47.
[43] U Ugarte-Pedrero X, Graziano M, Balzarotti D. A close look at a daily dataset of malware samples. ACM Transactions on Privacy and Security (TOPS) 2019;22(1): 1–30.
[44] Wagner M, Rind A, Thür N, Aigner W. A knowledge-assisted visual malware analysis system: Design, validation, and reflection of KAMAS. Computers & Security 2017;67:1–15.
[45] Wüchner T, Cisłak A, Ochoa M, Pretschner A. Leveraging compression-based graph mining for behavior-based malware detection. IEEE Trans Dependable Secure Comput 2017;16(1):99–112.
[46] Dong S, Xia Y, Peng T. Network Abnormal Traffic Detection Model Based on Semi-Supervised Deep Reinforcement Learning. IEEE Trans Netw Serv Manage 2021;18 (4):4197–212.
[47] Yan W, Zhang Z, Ansari N. Revealing Packed Malware [J] IEEE Security & Privacy 2008;6(5):65–9.
[48] Ye Y, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR) 2017;50(3):1–40.
[49] Yu J, He Y, Yan Q, et al. SpecView: Malware Spectrum Visualization Framework With Singular Spectrum Transformation [J] IEEE Trans Inf Forensics Secur 2021; 16:5093–107.
[50] Zhang J, Zhang K, Qin Z, et al. Sensitive system calls based packed malware variants detection using principal component initialized MultiLayers neural networks [J] cybersecur 2018;1(1).
[51] Zhang H, Xiao X, Mercaldo F, Ni S, Martinelli F, Sangaiah AK. Classification of ransomware families with machine learning based on N-gram of opcodes. Future Generation Computer Systems 2019;90:211–21.
[52] Zhu S, Shi J, Yang L, Qin B, Zhang Z, Song L, Wang G. Measuring and modeling the label dynamics of online anti-malware engines. In: 29th {USENIX} Security Symposium ({USENIX} Security 20); 2020. p. 2361–78.
[53] Zhang Z, Chang C, Han P, Zhang H. Packed malware variants detection using deep belief networks. In: MATEC Web of Conferences. EDP Sciences; 2020. p. 02002. Vol. 309.
[54] Hussain SJ, Ahmed U, Liaquat H, Mir S, Jhanjhi NZ, Humayun M. IMIAD: intelligent malware identification for android platform. In: 2019 International Conference on Computer and Information Sciences. IEEE; 2019. p. 1–6 (ICCIS).
[55] Imtiaz SI, ur Rehman S, Javed AR, Jalil Z, Liu X, Alnumay WS. DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. Future Generation computer systems 2021;115:844–56.
[56] Imran M, Bashir F, Jafri AR, Rashid M, ul Islam MN. A systematic review of scalable hardware architectures for pattern matching in network security. Computers & Electrical Engineering 2021;92:107169.
[57] Kok S, Abdullah A, Jhanjhi N, Supramaniam M. Ransomware, threat and detection techniques: A review. Int. J. Comput. Sci. Netw. Secur 2019;19(2):136.
[58] Rasool A, Javed AR, Jalil Z. SHA-AMD: sample-efficient hyper-tuned approach for detection and identification of Android malware family and category. Int J Ad Hoc Ubiquitous Comput 2021;38(1–3):172–83.
[59] Rashid M, Imran M, Jafri AR. Exploration of hardware architectures for string matching algorithms in network intrusion detection systems. In: Proceedings of the 11th International Conference on Advances in Information Technology; 2020. p. 1–7.

Xianwei Gao received M.E. degree in Graduate School of Chinese Academy of Sciences in 2005. He is currently pursuing the Ph.D. degree in School of Computer Science and Technology, Beijing Institute of Technology. He has more than ten years of experience in intrusion detection and security incident response. His research interests mainly focus on artificial intelligence and cyber security.

Changzhen Hu received his Ph.D. degree in Information Security from Beijing Institute of Technology in 1996. He is the vice dean, professor and doctoral supervisor of School of Computer Science and Technology, in Beijing Institute of Technology. His research interest is cyberspace security.

Chun Shan received her Ph.D. degree from the Beijing Institute of Technology in 2015. She is currently an associate professor at the School of Computer Science and Technology, Beijing Institute of Technology. And now she is leading the project supported by Natural Science Foundation of China. Her research interests include artificial intelligence, software security.

Weijie Han received B.E. degree in 2003 and M.E. degree in 2006 respectively from Academy of Equipment Command and Technology (now named Space Engineering University). Then he received his Ph.D. degree from Beijing Institute of Technology in 2020. His doctoral dissertation was awarded as the outstanding doctoral dissertation of BIT. He is currently a Lecture in Space Engineering University. His current research interest includes malware detection and APT detection.