Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Malware identification using visualization images and deep learning

Sang Ni[a], *Quan Qian*[b,a,*], *Rui Zhang*[a]

[a] *School of Computer Engineering & Science, Shanghai University, Shanghai 200444, China*
[b] *Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai, China 200444*

ARTICLE INFO

Article history:
Received 12 June 2017
Revised 27 March 2018
Accepted 7 April 2018
Available online 17 April 2018

Keywords:
Network security
Malware
Visual analysis
Deep learning

ABSTRACT

Currently, malware is one of the most serious threats to Internet security. In this paper we propose a malware classification algorithm that uses static features called *MCSC* (Malware Classification using SimHash and CNN) which converts the disassembled malware codes into gray images based on SimHash and then identifies their families by convolutional neural network. During this process, some methods such as multi-hash, major block selection and bilinear interpolation are used to improve the performance. Experimental results show that *MCSC* is very effective for malware family classification, even for those unevenly distributed samples. The classification accuracy can be 99.260% at best and 98.862% at average on a malware dataset of 10,805 samples which is higher than other compared algorithms. Moreover, for *MCSC*, on average, it just takes 1.41 s to recognize a new sample, which can meet the requirements in most of the practical applications.

1. Introduction

Malware is one of today's major Internet security threats. An Internet security threat report from Symantec (2016) shows that more than 430 million unique pieces of malware were discovered in 2015, an increase of 36% from the year before. The amount of new malware has been continuously growing, and its threats are increasing rapidly.

Malware, which is slipped into a victim's computer by hackers (attackers) through security vulnerabilities of the operating system or application software, can influence normal operation, collect sensitive information and steal superuser privileges in order to perform malicious actions. Generally, mainstream malware includes malicious scripts, vulnerability exploits, back doors, worms, trojans, spywares, rootkits, etc., and combinations or variations of the above types as well.

Traditionally, most malware detection systems are based on feature vectors, which contain essential characteristics of malware. Mainstream malware feature extraction can be divided into two categories, static and dynamic. In static analysis, malicious software is analyzed without executing it (Gandotra et al., 2014). The detection patterns used in static analysis include string signature, byte-sequence n-grams, syntactic library call, control flow graph and opcode (operational code) frequency distribution. For static analysis, the executable has to be unpacked and decrypted in advance.However, in dynamic analysis, malicious software is analyzed while being executed in a controlled environment (e.g., virtual machine, simulator, emulator, or sandbox). Before executing the malware samples, appropriate monitoring tools like Process Monitor or Capture BAT are installed and activated. While static analysis is usually vulnerable to code obfuscation, dynamic analysis is time consuming and computationally intensive.

The main contributions of this paper are: (1) Proposing a novel *MCSC* algorithm which combines opcode sequence and *LSH* to extract malware features. During feature extraction,

[*] Corresponding author.
E-mail addresses: qqian@shu.edu.cn, qqian@i.shu.edu.cn (Q. Qian).

https://doi.org/10.1016/j.cose.2018.04.005
0167-4048/© 2018 Elsevier Ltd. All rights reserved.

major block selection is used to extract the main part of malware. The preprocessing can reduce the influence from obfuscation in other parts and reduce calculation in following steps. (2) Using visualization techniques to transform the malware's non-intuitive features into fingerprint images. Compared to the original opcode sequence, a Simhash of the same size can easily be converted into an image. (3) Using a *CNN* to train the malware fingerprint images and get excellent identification results. (4) Improving *MCSC* further using multi-hash, major block selection and bilinear interpolation.

This paper is organized as follows. Malware-related studies are reviewed in Section 2. In Section 3, the *MCSC* algorithm is proposed and discussed in more detail. The experimental results are presented in Section 4, and Section 5 summarizes the whole paper.

2. Related work

Malware detection methods are typically divided into two categories: static analysis and dynamic analysis. In static analysis, the malware binary file is disassembled or decompiled without executing it. Thus, static analysis reveals the malware's behavior while preventing the operating system from malicious damages. However, in most cases static analysis is not a trivial task since attackers use code obfuscation techniques such as binary packers, encryption or self-modifying techniques to evade static analysis. Furthermore, static analysis does not allow a high degree of automation during analysis. In dynamic analysis, the behavior of malware is analyzed during execution in a debugger. Currently, sandbox-based dynamic analysis is one of the most promising techniques. A sandbox executes a malware sample in a controlled environment that can monitor and record information of system calls and behaviors dynamically. The main limitations of dynamic analysis, especially sandbox-based solutions, are the extensive and detailed reports requiring human analysis and interpretation. Furthermore, in contrast to static analysis, dynamic analysis can be automated to a high degree, and it also has high computation complexity.

2.1. *Static analysis*

In static analysis, Schultz et al. (2000) were the first to introduce the concept of data mining for detecting malware. They used three different static features for malware classification: Portable Executable (PE), strings and byte sequences. A rule induction algorithm called Ripper (Cohen, 1995) was applied to find patterns in the DLL. A Naive Bayes algorithm was used to find patterns in the string data, and *n*-grams of byte sequences were used as input data for the Multinomial Naive Bayes algorithm. Tian et al. (2008) used function length frequency to classify Trojans. Zolkipli and Jantan (2011) used variable length instruction sequences along with machine learning for worm detection. They tested their method on a dataset with 1444 worms and 1330 benign files. Kong and Yan (2013) presented a framework for automated malware classification based on structural information (function call graph) of malware. They used an ensemble of classifiers that learn from pairwise malware distances to classify malware into their respective families. Santos et al. (2013a) proposed a method for representing malware that relied on opcode sequences in order to construct a vector representation of the executables. Shankarapani et al. (2011) proposed two general malware detection methods: Static Analyzer for Vicious Executables (SAVE) and Malware Examiner using Disassembled Code (MEDiC). Their experimental results indicate that both of their proposed techniques can provide better detection performance against obfuscated malware. Gu et al. (2015) proposed a malicious document detection method based on wavelet transform and designed a malicious detection system based on this method. Li and Li (2015) proposed a 3-layer description of API-calls and similarity comparison method based on static code structure to determine what code family a malicious Android APK belongs to. This method can be used for evaluation and classification of unknown APKs.

2.2. *Dynamic analysis*

Dynamic analysis generally includes tainting, behavior-based methods, and API call monitoring (Han et al., 2014). Bayer et al. (2009) proposed a system that clusters large sets of malicious binaries based on their behavior effectively and automatically. Zolkipli and Jantan (2011) presented an approach for malware behavior analysis. They used *HoneyClients* and *Amun* as security tools to collect malware. Behavior of these malware were then identified by executing each sample on 2 virtual platforms, *CWSandbox* (Anderson et al., 2011) and *Anubis*. Anderson et al. (2011) presented a malware detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. Imran et al. (2015) proposed a malware classification scheme based on Hidden Markov Models using system calls as observed symbols. Fujino et al. (2015) proposed "API call topics", a kind of API call behavior for a malware family, to identify similar malware samples. They applied an unsupervised non-negative matrix factorization (NMF) clustering analysis to extract API call topics from a large corpus of API calls, which can detect similar malware samples. Lim et al. (2015) proposed a malware classification method based on network flow activity. They used clustering of flow features and a sequence alignment algorithm (generally used in bio-informatics to compare two or more character sequences to obtain their similarities) to analyze the malware traffic flow behavior. The main limitations of dynamic analysis, especially the sandbox-based solutions, are that some malware can detect and change behavior when running in virtual environments. Therefore, dynamic analysis might not always uncover malicious behavior. Additionally, it is difficult to ensure execution path coverage using dynamic analysis.

2.3. *Visualization analysis*

Recently, several visualization techniques have been proposed for malware analysis. Yoo (2004) used Self-Organizing Map to visualize and detect viruses. Quist and Liebrock (2009) presented a VERA framework to visually represent the overall flow of a program, which depends on Ether hypervisor to covertly monitor the program execution based on dynamic analysis. Trinius et al. (2009) visualized the behavior of malware using a treemap and thread graph by collecting information about the API calls and the operations of performed actions in a sandbox. Nataraj et al. (2011) proposed a method for visualizing and classifying malware using image processing techniques, which visualizes malware binaries as gray-scale images. A K-nearest neighbor technique with Euclidean distance method is used for malware classification. Kancherla and Mukkamala (2013) converted executables to a gray-scale image called byteplot and used Support Vector Machines (SVMs) to obtain an accuracy of 95% on a dataset containing 25,000 malwares and 12,000 benign samples. Han et al. (2015) proposed a malware analysis method that uses visualized images and entropy graphs to detect and classify new malware and malware variants. Arefkhani and Soryani (2015) introduced a new malware clustering method based on image processing of local sensitive hashes. Zhang et al. (2017) disassembled binary executables into opcodes sequences that were then converted into images. They used CNN method to recognize if a binary executable is malicious.

2.4. Other methods

Santos et al. (2013b) proposed OPEM, a hybrid unknown malware detector which combines the frequency of occurrence of operational codes (statically obtained) with the information of the execution trace of an executable (dynamically obtained). The article shows that this hybrid approach outperforms static or dynamic analysis. Kolosnjaji et al. (2017) constructed a deep neural network and applied it for the classification of malicious executables and achieved 93% on precision and recall.

Different from the existing work, in this paper we propose a malware classification algorithm that uses static features called *MCSC* (Malware Classification using SimHash and CNN) which combines malware visualization and deep learning techniques. By using *LSH*(locality-sensitive hashing), similar malware code will convert to similar hash values. These similar hash values are then transformed to similar gray images that can be used to train a *CNN*. During this process, some methods such as multi-hash, major block selection and bilinear interpolation are adopted to improve the performance of the algorithm. Experimental results show that *MCSC* is very effective for malware family classification, even at small sample sizes.

3. MCSC algorithm

MCSC algorithm is mainly divided into three parts: feature extraction, malware image generation and CNN training. During feature extraction, we need to extract the part which is recognizable from malware samples and encode it by *SimHash*. Then, these *Simhash* values are converted into malware images. Finally, these images are used to train *CNN* model iteratively aiming to identify their families. The basic procedure of *MCSC* is shown in Fig. 1.

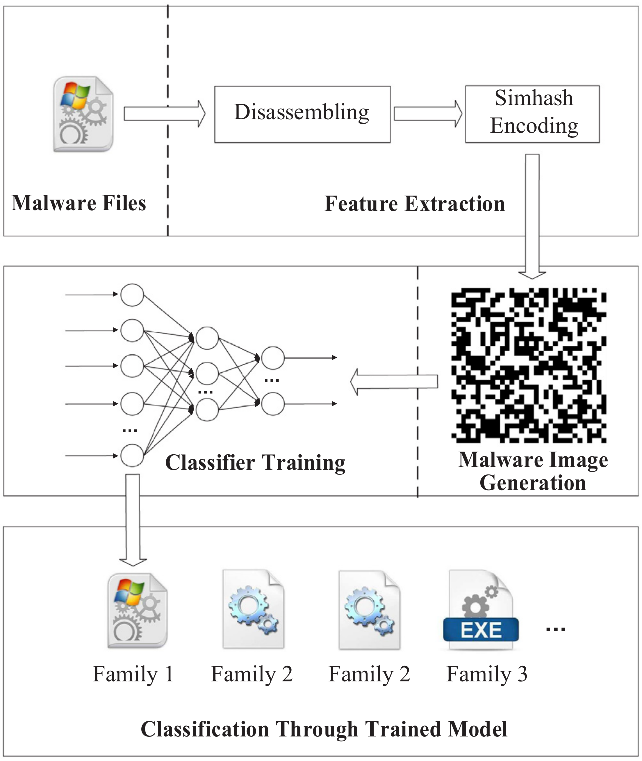


Fig. 1 – Basic procedure of *MCSC* including three parts: feature extraction, malware image generation and *CNN* training.

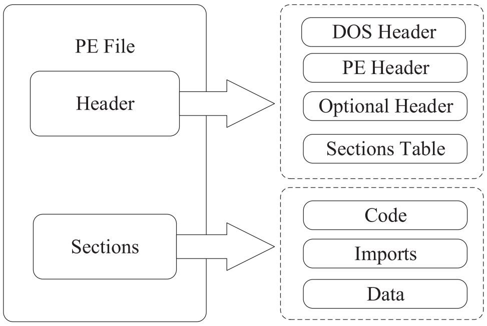


Fig. 2 – PE file structure. *Opcodes* are extracted as a sequence from PE file as the main feature.

3.1. Feature extraction

In Windows OS, PE format is a data structure that encapsulates the necessary information. A PE file consists of a number of headers and sections that tell the dynamic linker how to map the file into memory. An executable image consists of several different regions, each of which requires different memory protection, so the start of each section must be aligned to a page boundary. A simple PE file is shown in Fig. 2. In MCSC algorithm, *opcode* (Operation Code) from code section is chosen as features. Therefore, malware executables must first be disassembled. After getting the disassembled file, all of the *opcodes* are extracted as a sequence split by separators. Unlike IRMD (Zhang et al., 2017), in MCSC, all opcodes are extracted as a single sequence, for example "push, lea, push, mov, call", to pay more attention to the overall features of malware. However, in IRMD, it just selects 2-tuple opcode-sequence, for example, $os_1 = <mov, mov>$, $os_2 = <mov, call>$. 2-tuple opcode-sequence describes the feature of opcode pairs. It should be noted that if obfuscation or packing techniques are applied in malware samples, disassemblers sometime do not working correctly. Then we should use unpacking or other anti-obfuscation techniques.

Since the length of *opcode* sequences for each malware varies, it is hard to compare the similarity of sequences with different length. In this paper, *LSH*, i.e. SimHash, is used for sequence similarity comparison. *SimHash* is a local sensitive hash algorithm proposed by Charikar (2002) and is mainly used for similar text recognition, web pages duplication identification (Manku et al., 2007), etc.

A traditional hashing algorithm ensures that the collision rate is very low. Even for two similar inputs, the hashed output will be quite different. However, the aim of *SimHash* is to make those hashed values comparable. In other words, similar contents will have similar hash values.

The main idea of *SimHash* is using a specified dimension bits vector v to represent a document. The *n-th* bit of v is decided by calculating the hash value of all the keywords in the document. If the number of hash values whose *nth* bit is 1 is greater than the number of hash values whose *nth* bit is 0, then the *nth* bit of the *SimHash* v is set to 1. Otherwise the *nth* bit of the *SimHash* v is set to 0.

For example, we use the first 5 keywords from document *D* in Fig. 3 and a 4 bit hash function *h* to explain the *SimHash* calculation process, which is shown in Fig. 4. Supposing that:

$$D = (w_1 = \text{"}push\text{"}, w_2 = \text{"}lea\text{"}, w_3 = \text{"}push\text{"}, \\ w_4 = \text{"}mov\text{"}, w_5 = \text{"}call\text{"})^T \quad (1)$$

Then, we can get the hash value of each keyword above as follows:

$$h(w_1) = (1, 0, 0, 1)^T \quad (2)$$

$$h(w_2) = (1, 1, 0, 1)^T \quad (3)$$

$$h(w_3) = (1, 0, 0, 1)^T \quad (4)$$

$$h(w_4) = (1, 1, 1, 0)^T \quad (5)$$

$$h(w_5) = (0, 1, 1, 1)^T \quad (6)$$

In *MCSC* algorithm, the *opcode* sequence is regarded as a document and each *opcode* is the keyword referred in *SimHash*. For simplicity, each *opcode* is treated equally, thus all have weight of 1. Then, we can get the weight vector (WV) through weight and hash value as follows:

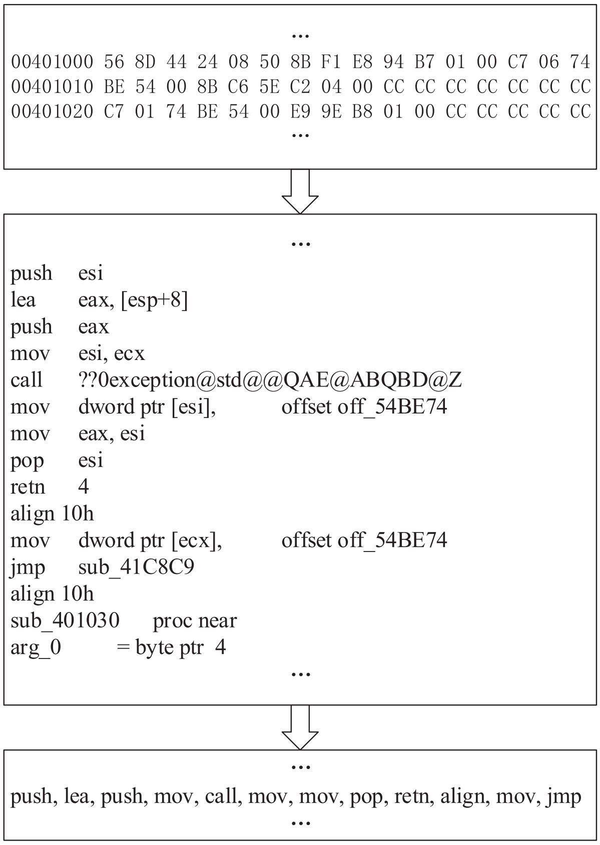
$$WV(w_1) = (1, -1, -1, 1)^T \quad (7)$$

$$WV(w_2) = (1, 1, -1, 1)^T \quad (8)$$

$$WV(w_3) = (1, -1, -1, 1)^T \quad (9)$$

$$WV(w_4) = (1, 1, 1, -1)^T \quad (10)$$

$$WV(w_5) = (-1, 1, 1, 1)^T \quad (11)$$

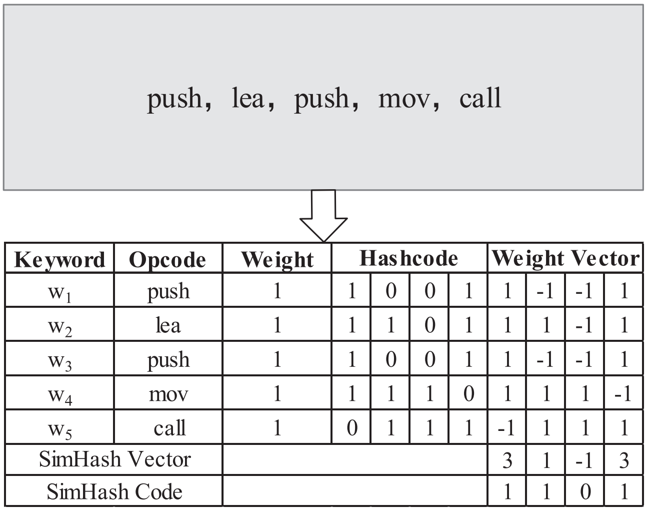


```
                              …
00401000 56 8D 44 24 08 50 8B F1 E8 94 B7 01 00 C7 06 74
00401010 BE 54 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 74 BE 54 00 E9 9E B8 01 00 CC CC CC CC CC
                              …
```

```
                              …
push    esi
lea     eax, [esp+8]
push    eax
mov     esi, ecx
call    ??0exception@std@@QAE@ABQBD@Z
mov     dword ptr [esi],        offset off_54BE74
mov     eax, esi
pop     esi
retn    4
align 10h
mov     dword ptr [ecx],        offset off_54BE74
jmp     sub_41C8C9
align 10h
sub_401030     proc near
arg_0          = byte ptr  4
                              …
```

```
                              …
push, lea, push, mov, call, mov, mov, pop, retn, align, mov, jmp
                              …
```

Fig. 3 – Opcode sequence extraction.



Keyword	Opcode	Weight	Hashcode				Weight Vector			
w_1	push	1	1	0	0	1	1	-1	-1	1
w_2	lea	1	1	1	0	1	1	1	-1	1
w_3	push	1	1	0	0	1	1	-1	-1	1
w_4	mov	1	1	1	1	0	1	1	1	-1
w_5	call	1	0	1	1	1	-1	1	1	1
SimHash Vector							3	1	-1	3
SimHash Code							1	1	0	1

Fig. 4 – Example of SimHash calculation process.

Then, we get a *SimHash* vector by adding up each WV and converting it into binary *SimHash*. In this example, we obtain the *SimHash* vector and *SimHash* value equal to 1101.

$$SimHashVector = (3, 1, -1, 3)^T \tag{12}$$

Finally, each sequence is encoded to a *n-bit SimHash* value which has the same length with the selected hash algorithm. The *SimHash* algorithm is shown in Algorithm 1.

Algorithm 1 SimHash Algorithm.

```
Input:
    opcodes sequence $Seq$, $n$-bit hash algorithm $hash$;
Output:
    $n$-bit SimHash value $s$;
Begin
Initialize a $n$-bit vector $v$ as zero vector;
Initialize a binary number $s$ as zero;
for each opcode in $Seq$ do
    $b$=hash(opcode);
    for $i$=1 to $n$ do
        if $b[i]$==1 then
            $v[i]$+=1;
        else
            v[i]-=1;
        end if
    end for
end for
for $i$=1 to $n$ do
    if $v[i]$ >0 then
        s[i]=1;
    else
        s[i]=0
    end if
end for
End
```

According to the characteristics of the *SimHash* algorithm, similar sequences will hash to similar *SimHash* values. Therefore, these similar binary values (with equal length) can be converted to similar images of equal size.

3.2. *Malware image generation*

After feature extraction, each malware code is encoded into binary *SimHash* value with equal length. Unlike other methods such as IRMD in Zhang et al. (2017), we use SimHash and bilinear interpolation to convert opcode sequence to malware image, while in IRMD, the binary image matrices are reconstructed by opcodes' probabilities and information gains.

We convert each *SimHash* bit to a pixel value ($0 \rightarrow 0$, $1 \rightarrow 255$). That is, if the bit value is 0, then the pixel value is 0; and if the bit value if 1, then the pixel value is 255. By then arranging the *n* pixel dots in a matrix, we convert the *SimHash* bits to a gray scale image. As shown in Fig. 5, images generated by malware variants from the same family have similar fingerprints in some areas. Therefore, malware variants from the same family can be identified by image processing techniques.

Table 1 – Different hash algorithms versus image size.

Algorithm	Image size
MD5 (the first 64bits)	8 × 8
MD5 (128 bits)	8 × 16
SHA-256	16 × 16
SHA-512	16 × 32

The size of image varies with the length of the *SimHash* value. Table 1 shows some corresponding sizes of different hash algorithms.

In order to train the CNN model, we zoom a non-square image by bilinear interpolation algorithm. Suppose the original image size is $(m \times n)$ and needs to be zoomed to $(a \times b)$. Then the gray value of pixel (i, j) in target image corresponds to pixel $(i \times m/a, j \times n/b)$ in the original image. The coordinate value may not be an integer after zooming, so the gray value is decided by the nearest four pixels of the original image. For example, as shown in Fig. 6, $P(x, y)$ is the pixel we need to calculate. $Q_{11}(x_1, y_1)$, $Q_{12}(x_2, y_1)$, $Q_{12}(x_1, y_2)$ and $Q_{22}(x_2, y_2)$ are the nearest four pixels with integer coordinates in the original gray scale image. Then the gray value of $P(x, y)$ is

$$P(x, y) = w_1 \times p_{11} + w_2 \times p_{21} + w_3 \times p_{12} + w_4 \times p_{22} \tag{13}$$

$$Where,\ w_1 = (x_2 - x) \times (y_2 - y) \tag{14}$$

$$w_2 = (x - x_1) \times (y_2 - y) \tag{15}$$

$$w_3 = (x_2 - x) \times (y - y_1) \tag{16}$$

$$w_4 = (x - x_1) \times (y - y_1) \tag{17}$$

Fig. 7 is an example of image zooming from size16 × 32 (rectangle image) to 32 × 32 (square image) by bilinear interpolation.

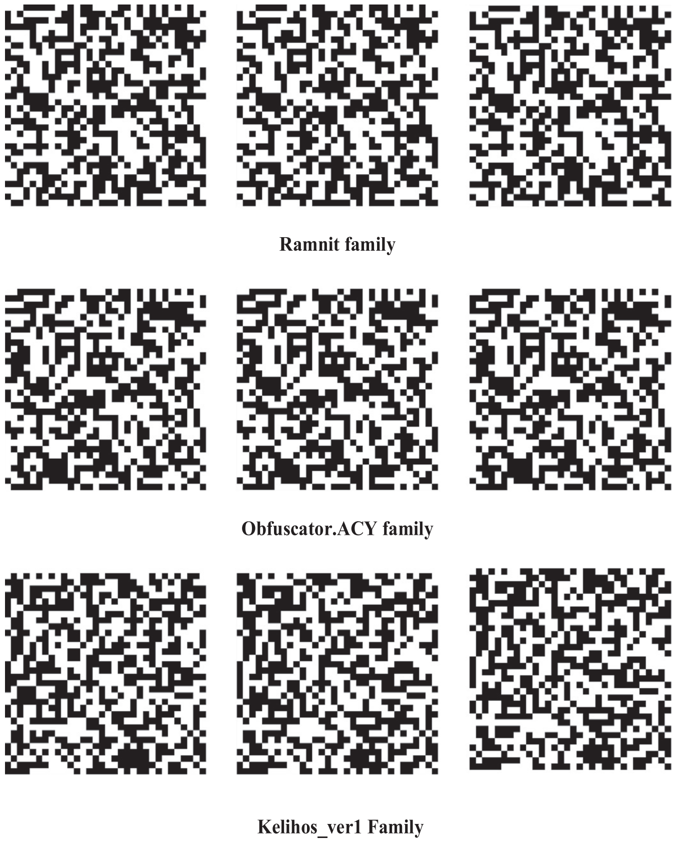
3.3. *Multi-Hash extension*

In *MCSC* algorithm, *SimHash* value is used as features and the length of *SimHash* value affects the image quality to some extent. Therefore, multiple cascading hash functions, instead of a single hash value, can be used to strengthen *SimHash*.

We can combine any of *SHA-512, SHA-384, SHA-256* and *MD5* to generate a longer *SimHash*. The number of bits can be used to distinguish different modes. For example, *SimHash-768* cascades *SHA-512* and *SHA-256*. Similarly, *SimHash-896* cascades *SHA-512*, *SHA-256* and *MD5*. However, it is important to note that multi-hash extension will increase the encoding computation overhead.

3.4. *Major block selection*

The basic *MCSC* algorithm concentrates on all opcodes in the disassembled file of malware. Nevertheless, many opcodes in basic blocks do not have much specificity and appear in most



Ramnit family

Obfuscator.ACY family

Kelihos_ver1 Family

Fig. 5 – Malware images of different malware families, Ramnit, Obfuscator.ACY, Kelihos_ver1 from top to bottom.

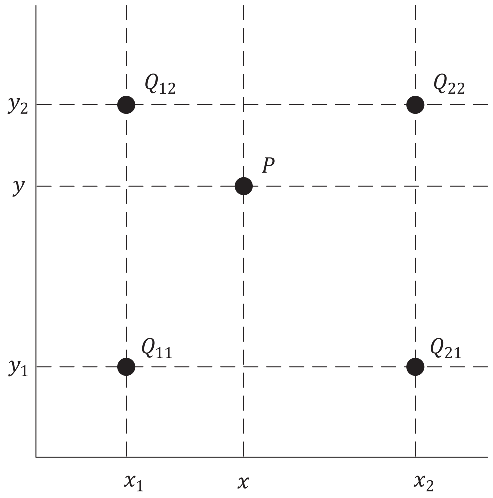


Fig. 6 – Calculating the gray value of pixel *P(x,y)*. *P* is the pixel we need to calculate and the others are the nearest four pixels with integer coordinates.

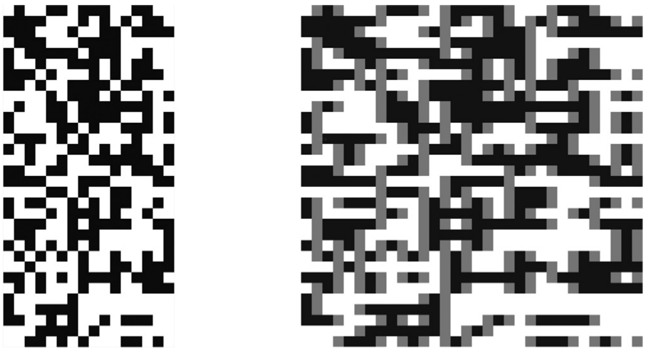
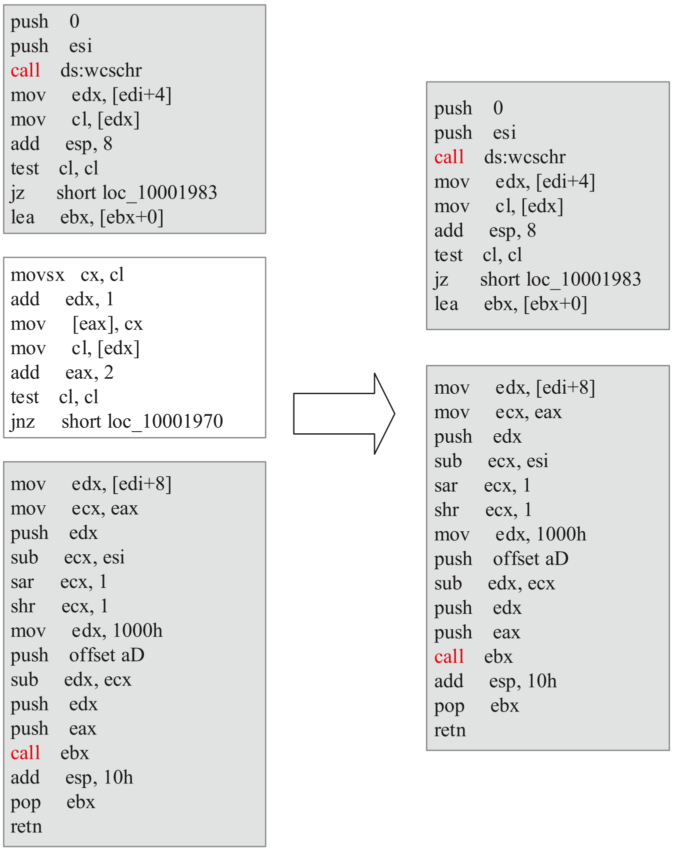


Fig. 7 – Image zooming by bilinear interpolation. (a) before zooming (16 x 32), and (b) after zooming (32 x 32).

malware programs. These opcodes may not contain suitable features for analysis. Additionally, including these shared opcodes will increase the time required to calculate the hash value of each opcode. So if we select some major blocks by extracting opcodes sequences that contain the most representative features, the efficiency of feature extraction can be increased greatly. According to the study of Kang et al. (2012), those blocks with *CALL* instructions should be selected as major blocks. The reason is that the *CALL* instruction, generally, is used to invoke *APIs*, library functions, and other user-defined functions, which are the basic implementation behaviors and functions of most programs. Fig. 8 indicates a procedure of major block selection.



```
push   0
push   esi
call   ds:wcschr
mov    edx, [edi+4]
mov    cl, [edx]
add    esp, 8
test   cl, cl
jz     short loc_10001983
lea    ebx, [ebx+0]
```

```
movsx  cx, cl
add    edx, 1
mov    [eax], cx
mov    cl, [edx]
add    eax, 2
test   cl, cl
jnz    short loc_10001970
```

```
mov    edx, [edi+8]
mov    ecx, eax
push   edx
sub    ecx, esi
sar    ecx, 1
shr    ecx, 1
mov    edx, 1000h
push   offset aD
sub    edx, ecx
push   edx
push   eax
call   ebx
add    esp, 10h
pop    ebx
retn
```

```
push   0
push   esi
call   ds:wcschr
mov    edx, [edi+4]
mov    cl, [edx]
add    esp, 8
test   cl, cl
jz     short loc_10001983
lea    ebx, [ebx+0]
```

```
mov    edx, [edi+8]
mov    ecx, eax
push   edx
sub    ecx, esi
sar    ecx, 1
shr    ecx, 1
mov    edx, 1000h
push   offset aD
sub    edx, ecx
push   edx
push   eax
call   ebx
add    esp, 10h
pop    ebx
retn
```

Fig. 8 – Major block selection for opcodes extraction. Those blocks without *CALL* instructions won't be selected as major blocks.

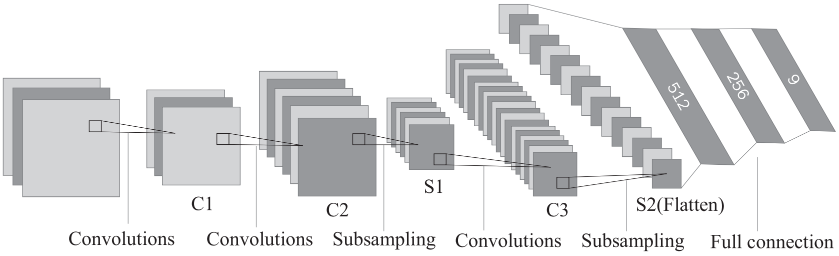


Fig. 9 – CNN structure for malware image training. *Convolutional layers* apply a convolution operation to the input. *Subsampling layers* combine the outputs of neuron clusters at one layer into a single neuron. *Dropout layers* are a type of regularization technique to reduce overfitting. *Fully connected layers* connect every neuron in one layer.

3.5. *CNN training*

Conventional machine learning techniques are limited in their ability to process natural data in their raw form, such as the pixel value of an image. Deep learning allows computational models that are composed of multiple processing layers to learn representations of raw data with multiple levels of abstraction (Lecun et al., 2015).

In *MCSC* algorithm, Convolutional Neural Network (CNN) is adopted to classify the preceding malware images. Fukushima (1980) proposed a calculation model of *CNN* in 1980 firstly based on local connections between neurons and hierarchical transformation. Based on this model, Lecun et al. (1989) proposed *CNN*, the first real multi-layer network structure learning algorithm, and used it for handwritten digit recognition. The model can automatically extract local features of an image and has strong adaptability. Parameter sharing makes *CNN* similar to biological neural networks and reduces the complexity of network model. Because of its strong learning ability in image recognition, *CNN* has achieved a great success in the field of pattern recognition, and *CNN* is chosen here for malware image recognition.

The CNN structure used in this paper is modified from Lenet-5 (Lecun et al., 1998) as shown in Fig. 9. Other similar method, for example the IRMD (Zhang et al., 2017), the architecture of the CNN model has three levels. Both level-1 and level-2 are convolution and pooling levels, and level-3 is full connection level and the activation function is sigmoid. In

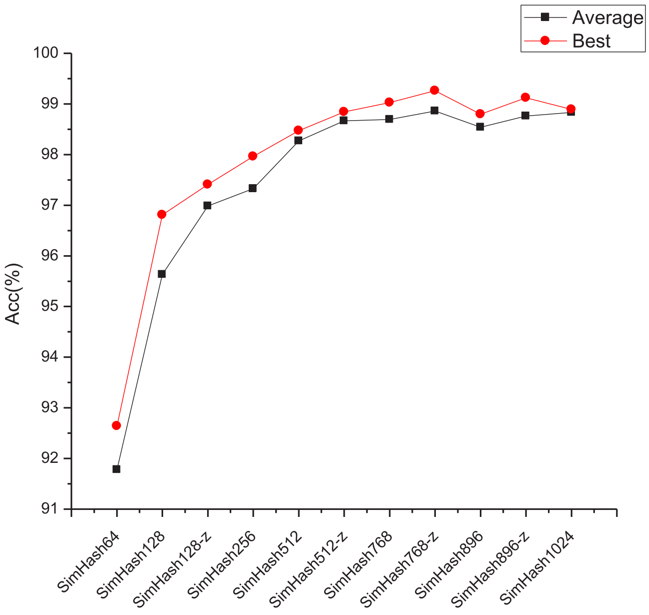


Fig. 10 – Classification accuracy of different *SimHash* modes including SimHash64, SimHash128, SimHash128-z, SimHash512, SimHash512-z, SimHash768, SimHash768-z, SimHash896, SimHash896-z, SimHash1024.

MCSC, each malware image as an input needs to go through two convolution layers, two subsampling layers and three full connection layers. During the process of convolution (C1, C2, C3), each convolutional layer include 32 learnable filters with the size of 2 × 2. If each input feature map is shown as x_i, learnable weight value w_{ij}, then the output feature map is as:

$$y_j = b_j + \sum_i w_{ij} * x_i \tag{18}$$

Table 2 – Distribution malware samples of the experimental dataset.

Malware family	Number of samples
Ramnit	1533
Lollipop	2473
Kelihos_ver3	2938
Vundo	453
Simda	42
Tracur	751
Kelihos_ver1	387
Obfuscator.ACY	1216
Gatak	1012

Table 3 – Cascading mode of different *SimHash* functions.

SimHash type	Cascading mode	Image size
SimHash-64	MD5-64(first 64 bit of MD5)	8 × 8
SimHash-128	MD5	8 × 16
SimHash-256	SHA-256	16 × 16
SimHash-512	SHA-512	16 × 32
SimHash-768	SHA-512+SHA-256	24 × 32
SimHash-896	SHA-512+SHA-256+MD5	28 × 32
SimHash-1024	SHA-512+SHA-384+MD5	32 × 32

Here “*” is a convolution operator, and b is a learnable bias parameter.

The output feature map adopts an activation function $R = h(y)$ for nonlinear mapping. In MCSC, *tanh* is chosen as the activation function.

$$R = \frac{e^y - e^{-y}}{e^y + e^{-y}} \tag{19}$$

During the process of subsampling (S1, S2), max pooling is adopted whose window size is 2 × 2 to reduce training param-

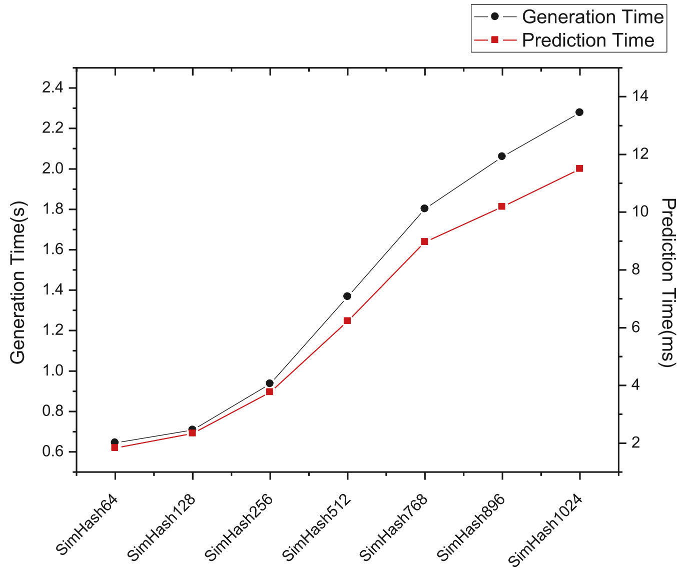


Fig. 11 – For *Kelihos_ver1* family, the average generation and prediction time with different *SimHash* modes including SimHash64, SimHash128, SimHash512, SimHash768, SimHash896, SimHash1024.

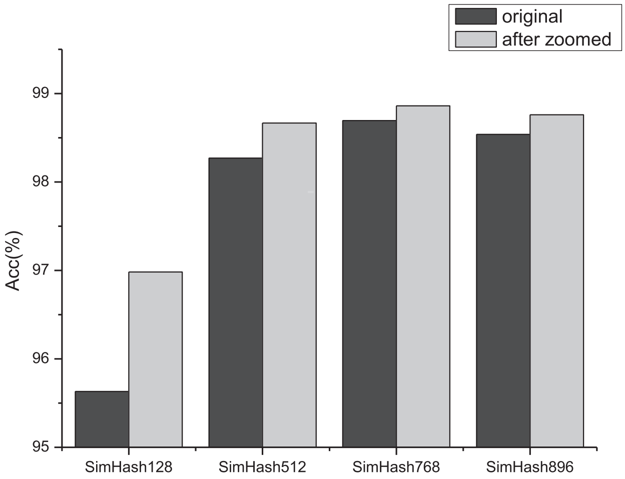


Fig. 12 – Classification accuracy before and after bilinear interpolation with different *SimHash* modes including SimHash128, SimHash512, SimHash768, SimHash896.

eters. After each max pooling there is a dropout layer whose probability is 0.5 to avoid overfitting. After the second subsampling layer (S2), we flatten the output feature map and link it to three fully connected layers whose dimensions are 512, 256, and 9 (number of malware categories) respectively. The first two fully connected layers adopt *tanh* as activation function and the last one uses *softmax*.

$$\sigma(Z)_j = \frac{e^{Z_j}}{\sum_{k=1}^{K} e^{Z_k}} \quad for\ j = 1, \ldots, K. \tag{20}$$

4. Experiment

4.1. *Experimental data and execution environments*

The experimental data in this paper is from the Malware Classification Challenge (Kaggle, 2015) on Kaggle by Microsoft 2015. There are 10,868 labeled malware samples from 9 families in this dataset with binary and disassembly files. After preprocessing, 10,805 samples remain. 80% of them will be used for training and the rest for testing. Distribution of malware samples in the experimental dataset is shown in Table 2. Experimental programs are written in Python and MATLAB, and the hardware environment is an Intel Core i7-3370 processor with 8 GB main memory.

4.2. *Impact of different hash combinations on accuracy and performance*

We adopt different hash combinations to generate malware images and use the same structure CNN model for training. The model whose accuracy is the highest after 100 iterations will be selected for comparison.

Experimental result is shown in Fig. 10. Each *SimHash* mode can be identified by its length. The corresponding relation between *SimHash* mode and length is shown in Table 3. The model that uses bilinear interpolation zooming is marked by -z. It is observed that increasing the length of *SimHash* can improve the accuracy of the model, but we only see small increases by increasing the length past 512. The average accuracy exceeds 92%, and the median is 98.5%.

Fig. 11 shows the average time that different *SimHash* modes spend in generating and predicting a malware image from *Kelihos_ver1* family. It is obvious that generation time increases with the length of *SimHash* and that prediction time depends on the image size. When we classify a new malware sample, the process mainly consists of the above two parts (generating a malware fingerprint image and predicting it by CNN trained model). Experimental results show that identifying a malware sample takes 1.41 seconds in average.

4.3. *Impact of bilinear interpolation zooming on classification accuracy*

Figs. 12 and 13 show the comparison between the original malware images and images after zooming through bilinear interpolation. The results indicate that using bilinear interpolation to zoom an image to a square could achieve better

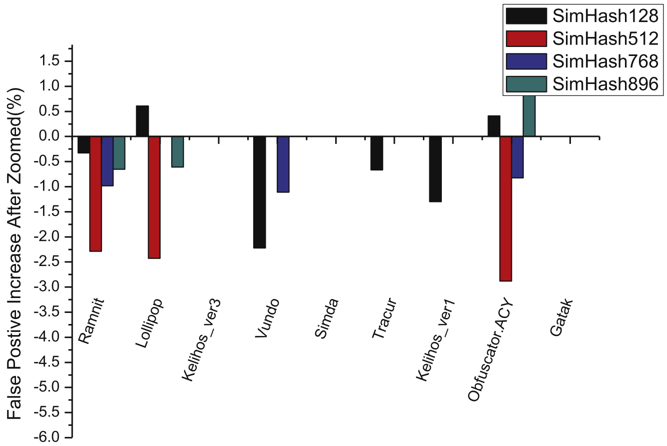


Fig. 13 – False positive rate increase before bilinear interpolation with different *SimHash* modes including SimHash128, SimHash512, SimHash768, SimHash896.

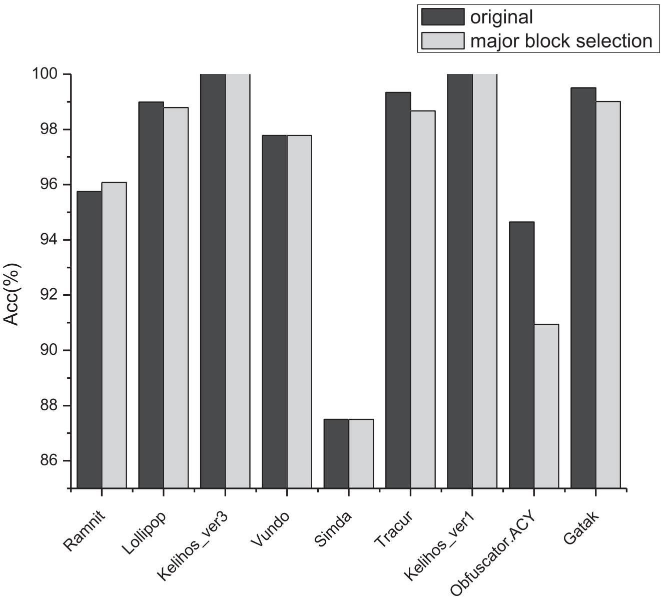


Fig. 14 – Classification accuracy with or without major block selection in 9 malware families: Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak.

Table 4 – Average number of opcodes with or without major block selections.

Malware family	Original	With major block selection	Rate (%)
Ramnit	131,902	89,145	67.58
Lollipop	71,016	49,411	69.58
Kelihos_ver3	3941	3,527	89.50
Vundo	7148	2,469	34.54
Simda	8350	5,409	64.78
Tracur	16,824	10,449	62.11
Kelihos_ver1	10,745	7805	72.64
Obfuscator.ACY	18,645	11,714	62.83
Gatak	22,728	13,892	61.12

classification accuracy. Besides, the false positive rate only increases in a few cases. The reason is that bilinear interpolation magnifies some useful features of the image.

4.4. *Impact of major block selection on the classification accuracy and performance*

Fig. 14 shows a comparison on accuracy and performance between the original mode and the major block selection mode using *SimHash*-512. It shows that after major block selection the classification accuracy of 4 families decreases a little, while the accuracy of *Ramnit* family increases. The accuracy of classification on the whole dataset with or without major block selection is 98.38% and 97.82%, respectively.

Table 4 shows the average number of opcodes with or without major block selection using *SimHash*-512. Fig. 15 shows the average generation time of malware fingerprint images with or without major block selection. The result shows that the generation speed is greatly improved by major block selection. On average, there is a 35.04% decrease in number of opcodes and a 30.62% decrease in generation time.

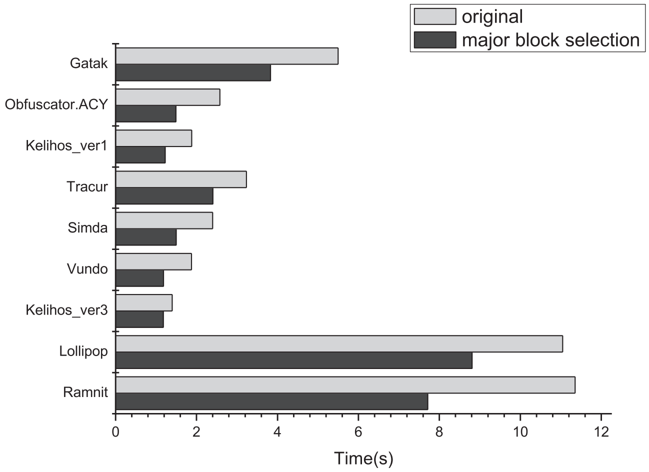


Fig. 15 – Average fingerprint images generation time with or without major block selection in 9 malware families: Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak.

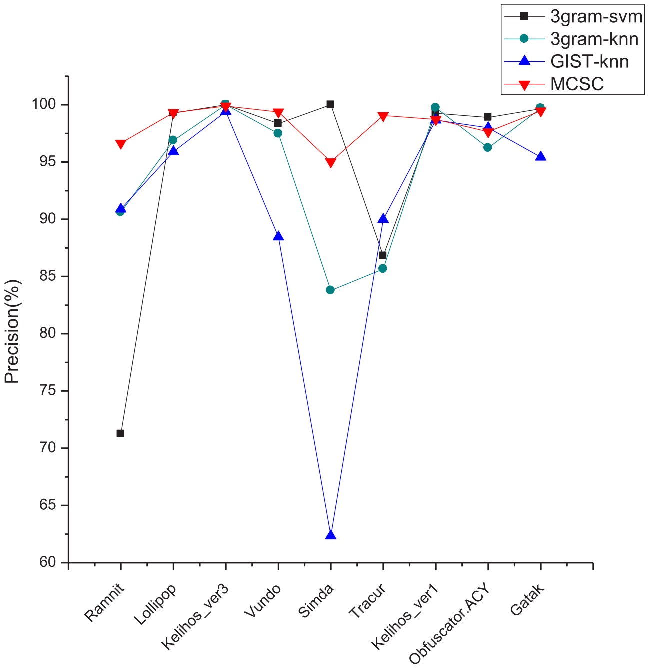


Fig. 16 – Average precision of different malware classification algorithms including 3gram-svm, 3gram-knn, GIST-knn, *MCSC* in 9 malware families.

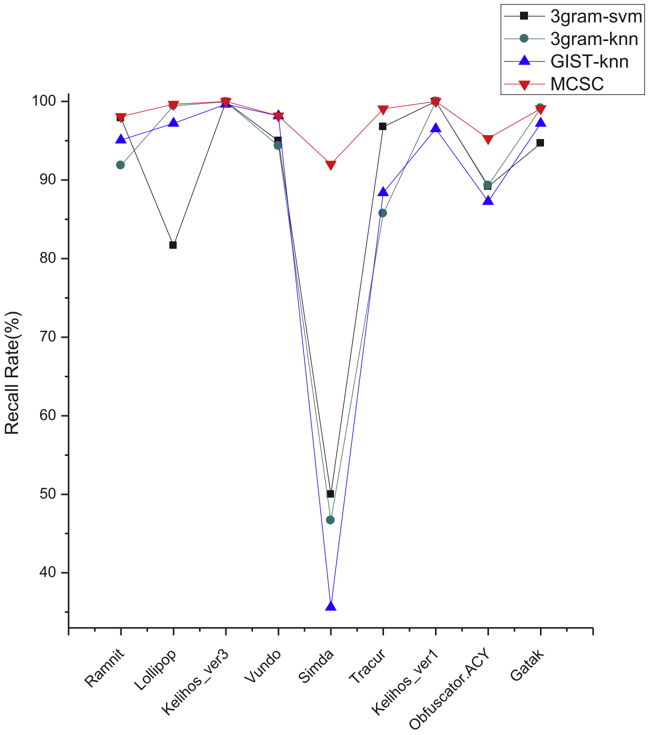


Fig. 17 – Average recall rate of different malware classification algorithms including 3gram-svm, 3gram-knn, GIST-knn, *MCSC* in 9 malware families.

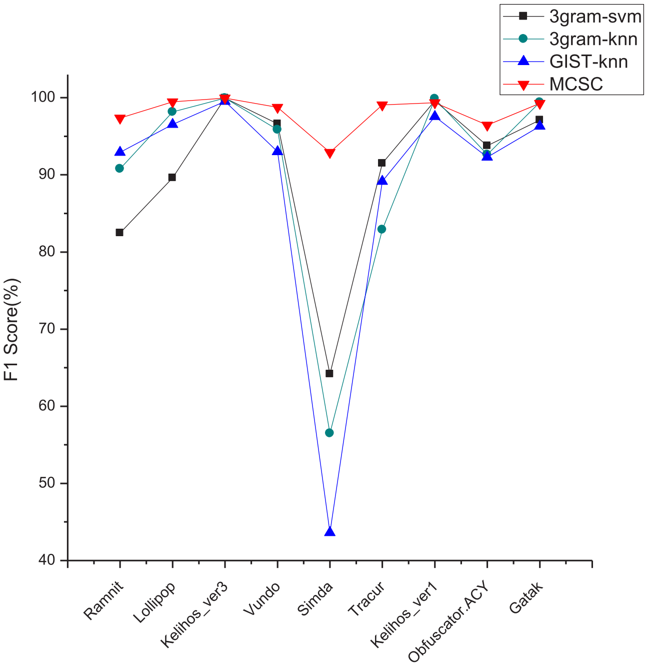


Fig. 18 – Average F1 score of different malware classification algorithms including 3gram-svm, 3gram-knn, GIST-knn, *MCSC* in 9 malware families.

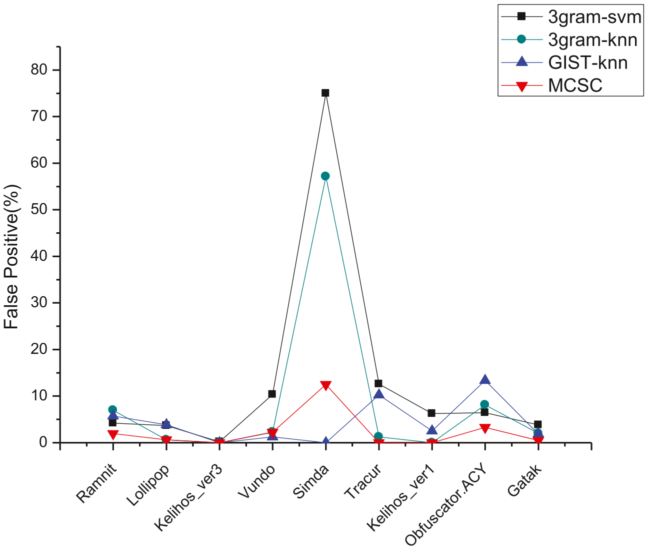


Fig. 19 – Average false positive rate of different malware classification algorithms including 3gram-svm, 3gram-knn, GIST-knn, *MCSC* in 9 malware families.

4.5. *Comparison with other malware classification algorithms*

In this paper, two other algorithms are chosen for comparison with *MCSC*. One is *N-gram* of opcodes (Moskovitch et al., 2008) and the other is the method proposed in Nataraj et al. (2011) which adopts *GIST* features of malware images and SVM classification. In *N-gram* of opcode, we set *N*=3 and select 512

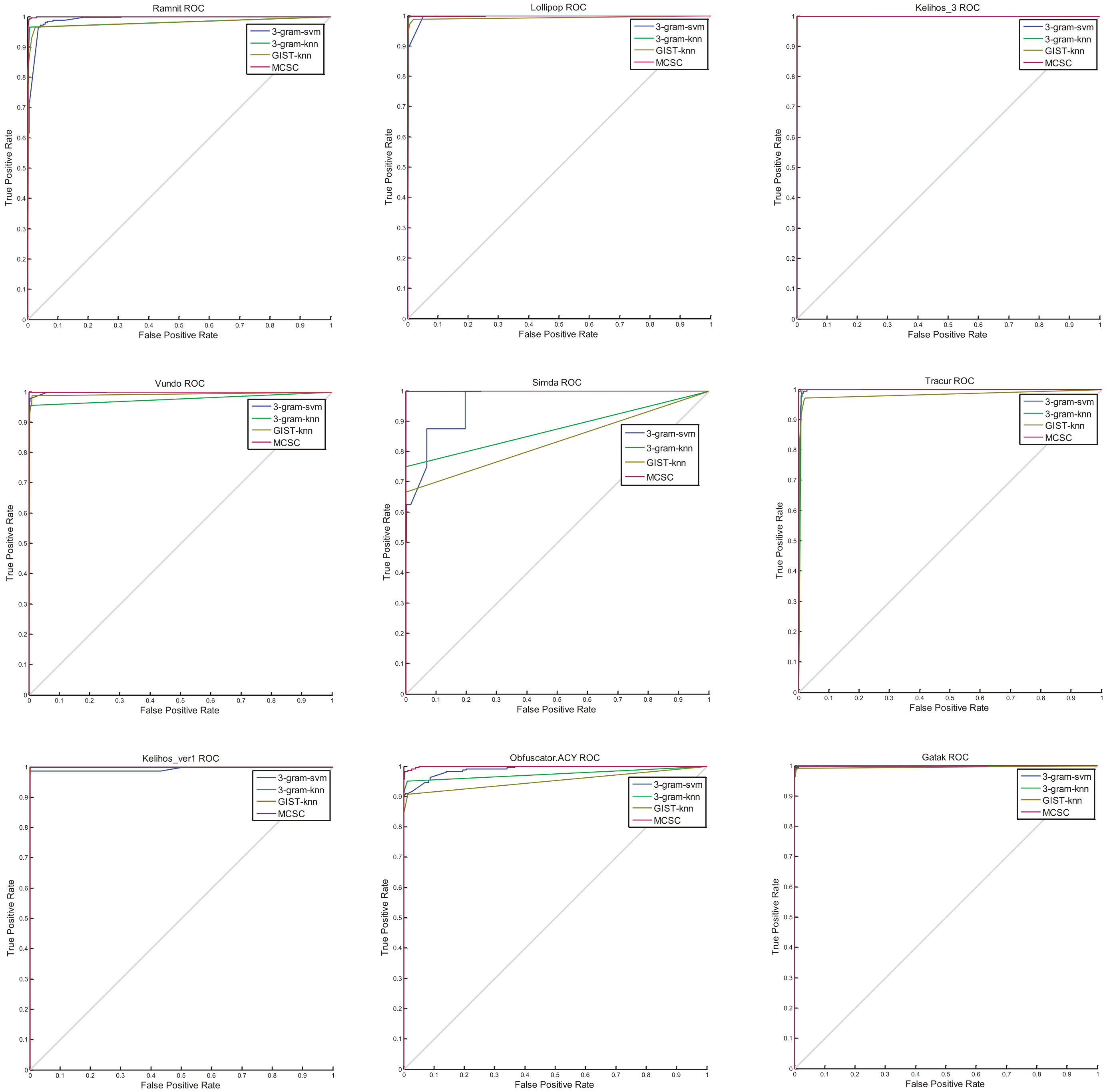


Fig. 20 – ROC curve of different malware families with different algorithms including 3gram-svm, 3gram-knn, GIST-knn, MCSC in 9 malware families.

opcodes combinations which appear most frequently in training set as the malware fingerprint features. Then, Support Vector Machine (SVM) and K-Nearest Neighbor (KNN, $K=3$) are used to classify these samples. In another algorithm, *GIST* feature of 512 dimensions and *KNN* are used to classify these samples. Besides, *SimHash768-z* which performs best in previous experiments is selected for comparison. Each algorithm repeats 5 times for random sampling. In the end, average accuracy of *3-gram-SVM* algorithm on the test set of 9 families is 93.059%, *3-gram-KNN* is 96.076%, *GIST-SVM* is 95.595% and *MCSC* is 98.862%. The best accuracy of *3-gram-SVM* algorithm on the test set of 9 families is 93.383%, *3-gram-KNN* is 97.224%, *GIST-KNN* algorithm gets 95.974% and *MCSC* is 99.260%.

To evaluate the results, we use the precision, recall, F1 metrics and receiver operating characteristic curve (ROC), which have the following definitions:

$$Precision = \frac{TP}{TP+FP} \tag{21}$$

$$Recall = \frac{TP}{TP+FN} \tag{22}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{23}$$

Here, TP is the number of true positive predictions, FP is the number of false positive predictions, and FN is the number of false negative predictions. The F1 score is a weighted average of the precision and recall. The F-measure or balanced F-score(F1 score) is the harmonic mean of precision and recall, which can be defined as Eq. (23).

We regard the multi-classification problem as several binary classification problems of each malware family. In classification problems, precision and recall are common evaluation methods. Precision is the probability that a (randomly selected) retrieved sample is relevant. Recall is the probability that a (randomly selected) relevant sample is retrieved. F-score considers both the precision and the recall. Figs. 16–19 are *Precision*, *Recall rate*, *F1* score and *False positive rate* of the three different algorithms. Furthermore, we can get the probability matrix of each family during prediction and draw the ROC curve for each algorithm as shown in Fig. 20. From the experimental results, it can be seen that *MCSC* has the highest F1 score of 98.07% on average, compared to *3-gram-SVM* algorithm with 90.54%, *3-gram-KNN* with 90.67%, and *GIST-SVM* algorithm with 88.99%. Besides, *MCSCs* false positive is the lowest in 2/3 malware families. *3-gram-SVM* has an average false positive rate of 13.61%, *3-gram-KNN* has 8.70%, *GIST-SVM* has 4.31% and *MCSC* has 2.34%. Since there are only 42 samples in family *Simda*, most of the algorithms cannot recognize *Simda* very well. It is worth noting that *MCSC* algorithm still keeps a good performance. For *ROC* curve, in Fig. 20, we can get a similar conclusion that *MCSC* curve is closer to the upper left corner of the coordinates. Even in family *Simda* and *Obfuscator.ACY* where other algorithms struggle, *MCSC* curve's peek is still very close to 1. Therefore, *MCSC* shows a degree of advantages for those uneven data distributions.

5. Conclusion and future work

In this paper, we propose a malware classification algorithm called *MCSC* which combines malware visualization and deep learning techniques. First of all, we extract the opcode sequences from malware and encode them to equal length with SimHash while preserving the malware's fingerprint features. By taking each SimHash value as a pixel, SimHash bits can be converted to gray images. Then, a convolutional neural network is adopted to train the images and identify the malware families.

Furthermore, during this process optimization methods are adopted to improve the performance of the algorithm. Multi-hash optimization strengthens *SimHash* by improving the classification accuracy. Optimization using bilinear interpolation magnifies useful features of the image for the *CNN*. Major block selection improves the efficiency of image generation, while having a limited impact classification accuracy. The experimental results show the excellent recognition capability of *MCSC*. The classification accuracy has a max of 99.260% and an average of 98.862% on a malware dataset of 10,805 samples, which is higher than other compared algorithms. For *MCSC*, on average, it just takes 1.41s to recognize a new sample. Using major block selection reduces the image generation time by 36% on average and only decreases accuracy by 0.56%. The experimental results show, once again, that malware variants in the same family have some common fingerprints.

Although we have seen some experimental evidence of the viability proposed approach, we need further studies in the following directions. (1) Faster malware detection and classification using high-performance computing based on GPU or other parallelization techniques. (2) Using the proposed method in large-scale application environments. (3) Further study to detect effectively when malware use packing, encryption, anti-debugging and anti-dissembling techniques. (4) Integrating the proposed static method with dynamic analysis to extend the robustness and adaptability of the detection system.

Acknowledgment

This work is partially sponsored by National Key Research and Development Program of China(2016YFB0700504, 2017YFB0701601), Shanghai Municipal Science and Technology Commission (15DZ2260301), Natural Science Foundation of Shanghai (16ZR1411200). The authors gratefully appreciate the anonymous reviewers for their valuable comments.

REFERENCES

Anderson B, Quist D, Neil J, Storlie C, Lane T. Graph-based malware detection using dynamic analysis; 2011. p. 247–58.

Arefkhani M, Soryani M. Malware clustering using image processing hashes. Proceedings of ninth Iranian conference on machine vision and image processing (MVIP). IEEE; 2015. p. 214–18.

Bayer U, Comparetti PM, Hlauschek C, Krgel C, Kirda E. Scalable, behavior-based malware clustering. Proceedings of network and distributed system security symposium, NDSS 2009, San Diego, California, USA, 2009. February -February

Charikar MS. Similarity estimation techniques from rounding algorithms. Proceedings of the thirty-fourth annual ACM symposium on theory of computing, Montreal, Quebec, Canada; 2002. p. 380–8. May

Cohen WW. Fast effective rule induction. Proceedings of the machine learning; 1995. p. 115–23.

Fujino A, Murakami J, Mori T. Discovering similar malware samples using API call topics. Proceedings of the twelfth annual IEEE consumer communications and networking conference (CCNC), Las Vegas, NV, USA; 2015. p. 140–7. January

Fukushima K. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern 1980;36(4):193–202.

Gandotra E, Bansal D, Sofat S. Malware analysis and classification: a survey; 2014. p. 56–64.

Gu B, Fang Y, Jia P, Liu L. A new static detection method of malicious document based on wavelet package analysis. Proceedings of the 2015 international conference on intelligent information hiding and multimedia signal processing (IIH-MSP), Adelaide, SA, Australia; 2015. p. 333–6. September

Han K, Kang B, Im EG. Malware analysis using visualized image matrices, 2014.

Han KS, Lim JH, Kang B, Im EG. Malware analysis using visualized images and entropy graphs. Int J Inf Secur 2015;14(1):1–14.

Imran M, Afzal MT, Qadir MA. Using hidden Markov model for dynamic malware analysis: first impressions. Proceedings of the twelfth international conference on fuzzy systems and knowledge discovery (FSKD), Zhangjiajie, China; 2015. p. 816–21. August

Kaggle (2015). Microsoft malware classification challenge (big 2015). [Online]. Available: https://www.kaggle.com/c/malware-classification/data.

Kancherla K, Mukkamala S. Image visualization based malware detection. Proceedings of the IEEE symposium on computational intelligence in cyber security (CICS), Singapore, Singapore; 2013. p. 40–4. April

Kang B, Kim T, Kwon H, Choi Y, Im EG. Malware classification method via binary content comparison. Proceedings of the ACM research in applied computation symposium; 2012. p. 316–21.

Kolosnjaji B, Eraisha G, Webster G, Zarras A, Eckert C. Empowering convolutional networks for malware classification and analysis. Proceedings of the international joint conference on neural networks (IJCNN), Anchorage, AK, USA; 2017. p. 3838–45. May

Kong D, Yan G. Discriminant malware distance learning on structural information for automated malware classification. Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, Chicago, Illinois, USA; 2013. p. 1357–65. August

Lecun Y, Bengio Y, Hinton G. Deep learning. Nature; 2015. p. 436–44.

Lecun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Backpropagation applied to handwritten zip code recognition. Neural Comput 1989;1(4):541–51.

Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE 1998;86(11):2278–324.

Li Q, Li X. Android malware detection based on static analysis of characteristic tree. Proceedings of the international conference on cyber-enabled distributed computing and knowledge discovery (cyberc), Xi'an, China; 2015. p. 84–91. September

Lim H, Yamaguchi Y, Shimada H, Takakura H. Malware classification method based on sequence of traffic flow. Proceedings of the international conference on information systems security and privacy (ICISSP), Angers, France; 2015. p. 1–8. Febuary

Manku GS, Jain A, Sarma AD. Detecting near-duplicates for web crawling. Proceedings of the international conference on world wide web; 2007. p. 141–50.

Moskovitch R, Feher C, Tzachar N, Berger E, Gitelman M, Dolev S, Elovici Y. Unknown malcode detection using opcode representation. Proceedings of the Intelligence and security informatics, first European conference, euroISI 2008, Esbjerg, Denmark; 2008. p. 204–15. December

Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. Proceedings of the eighth international symposium on visualization for cyber security, Pittsburgh, Pennsylvania, USA; 2011. p. 311–20. July

Quist DA, Liebrock LM. Visualizing compiled executables for malware analysis. Proceedings of the sixth international workshop on visualization for cyber security, 2009,Vizsec 2009. Atlantic City, NJ, USA; 2009. p. 27–32. October

Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection; 2013a. p. 64–82.

Santos I, Devesa J, Brezo F, Nieves J, Bringas PG. OPEM: a static-dynamic approach for machine-learning-based malware detection. Proceedings of the international joint conference CISIS12-ICEUTE 12-SOCO 12 special sessions. Springer; 2013b. p. 271–80.

Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. Proceedings of the IEEE symposium on security and privacy, 2001. S&P 2001. Oakland, CA, USA; 2000. p. 38–49. May

Shankarapani MK, Ramamoorthy S, Movva RS, Mukkamala S. Malware detection using assembly and API call sequences; 2011. p. 107–19.

Symantec (2016). 2016 internet security threat report highlights. [Online]. Available: https://www.symantec.com/security-center/threat-report.

Tian R, Batten LM, Versteeg SC. Function length as a tool for malware classification. Proceedings of the third international conference on malicious and unwanted software, 2008. MALWARE 2008.. VI, USA: Fairfax; 2008. p. 69–79. October

Trinius P, Holz T, Göbel J, Freiling FC. Visual analysis of malware behavior using treemaps and thread graphs. Proceedings of the sixth international workshop on visualization for cyber security, 2009, Vizsec 2009 Atlantic City, NJ, USA; 2009. p. 33–8. October

Yoo I. Visualizing windows executable viruses using self-organizing maps. Proceedings of the workshop on visualization and data mining for computer security, Washington DC, USA; 2004. p. 82–9. October

Zhang J, Qin Z, Yin H, Ou L, Hu Y. IRMD: malware variant detection using opcode image recognition. Proceedings of the IEEE twenty second international conference on parallel and distributed systems (ICPADS), Wuhan, China; 2017. p. 1175–80. December

Zolkipli MF, Jantan A. An approach for malware behavior identification and classification. Proceedings of the third international conference on computer research and development (ICCRD), Shanghai, China; 2011. p. 191–4. March

Sang Ni is a master degree student in the school of computer science, Shanghai University. He got his bachelor degree of Computer Science & Technology from Shanghai University in July, 2015. His research interests include malware detection, intrusion detection, cloud computing, big data analysis, computer and network security.

Quan Qian is a full Professor in School of Computer Engineering & Science, Shanghai University, China. His main research interest concerns computer network and network security, especially in cloud computing, big data analysis and wide scale distributed network environments. He received his computer science Ph.D. degree from University of Science and Technology of China (USTC) in 2003 and conducted postdoc research in USTC from 2003 to 2005. After that, he joined Shanghai University and now he is the lab director of network and information security. So far, he has published journal or conference paper more than 120.

Zhang Rui received her B.E. and Ph.D. degree from Department of Electronic Engineering & Information Science, University of Science and Technology of China, in 2003 and 2008, respectively. After graduation, she works in School of Computer Engineering and Science, Shanghai University. Her main research interests include computer networks, network coding for wireless networks and wireless communication, etc.