Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Using IRP and local alignment method to detect distributed malware☆

Yusheng Dai[a], *Hui Li*[a,*], *Yekui Qian*[b,*], *Yunling Guo*[c], *Ruipeng Yang*[b], *Min Zheng*[d]

[a] *School of electronics and information, Northwestern Polytechnical University, Xi'an, 710129 China*
[b] *Network Security Laboratory, PLA Army Academy of Artillery and Air Defense, Zhengzhou 450000, China*
[c] *Elctronic and Electrical Experiment Teaching Center, Shaanxi University of Technology, Hanzhong, 723000, China*
[d] *Zhongke Yuxin technology development (Xuchang) Co., Ltd, Xuchang 461000, China*

ARTICLE INFO

Article history:
Received 27 July 2020
Revised 20 October 2020
Accepted 5 November 2020
Available online 10 November 2020

Keywords:
Dynamic detection
Distributed malware
IRP sequence
Local alignment
Security

ABSTRACT

Malware seriously threatens national security and personal privacy, but the evasive behavior of malware is becoming more and more covert and difficult to analyze. Thus, effectively detect malware is of great significance. Distributed malware injection is a new type of evasion technique. By dividing the malware into blocks, and then injecting the blocks into multiple benign processes, each block communicates with each other and can perform complete malicious actions in sequence, making the existing of malware detection fails. Currently, commercial anti-virus software cannot effectively detect distributed malware. At the same time, for this type of malware research, we believe that there is still room for improvement in detection performance. For this evasion technique, this paper proposes a detection method using I/O request package (IRP) sequence features combined with local alignment algorithms in bioinformatics. In the detection process, we filter and extract important IRP requests in operating system, and use the local alignment algorithm to compare with the malware's IRP sequence, which can effectively identify the distributed malware hidden in the system. This paper uses real malware to split and perform detection experiments. The results prove that our detection method can effectively detect distributed malware, and the detection accuracy is better than similar research.

1. Introduction

With the rapid development of information technology today, national information security and personal privacy are of paramount importance. As an important carrier of cybercrime, malware is a serious threat to the information security of the country and citizens. How to effectively detect malware is of great significance. However, with the continuous update of malware technology, the evasion technique of malware is also continuously enhanced, making malware more concealed and more difficult to detect. Therefore, how to effectively detect malware with evasive behavior is still a problem faced by current malware detection research.

Malware detection is mainly divided into dynamic detection and static detection. Due to the use of obfuscation, packaging technique and other techniques, static detection of malware has become a difficult task. Dynamic detection can ef-

☆ Fully documented templates are available in the elsarticle package on CTAN.

* Corresponding authors.
E-mail addresses: daiyusheng@mail.nwpu.edu.cn, daiyusheng@126.com (Y. Dai), lh@mail.edu.cn (H. Li), qyk1129@163.com (Y. Qian).

https://doi.org/10.1016/j.cose.2020.102109
0167-4048/

fectively overcome the shortcomings of static detection, but dynamic detection is vulnerable to evasion attacks by malware (Bulazel and Yener, 2017). The common malware evasion techniques include environmental inspection, mimicry attacks (Maiorca et al., 2015; 2013), code reuse (Bletsch et al., 2011; Checkoway et al., 2010; Schuster et al., 2015), and code injection endgame (2017). These evasion techniques have affected the detection of malware to a certain extent. However, the dynamic detection methods based on malware features, including API sequence (Kakisim et al., 2019; microsoft, 2019), malicious behaviors (Ding et al., 2018; Saracino et al., 2018), and ensemble learning (Dai et al., 2019), etc., can all be effectively dealt with the above problems.

Recently, there is a powerful malware evasion technique called malWash (Ispoglou and Payer, 2016). This method splits the malware into multiple blocks and injected them into multiple benign programs that are running on the system. Through fragmented code execution, malicious behavior is minimized to achieve evasion. Because malWash execution requires scheduling code to coordinate, some behaviors may be exposed. D-TIME (Pavithran et al., 2019) is built on the basis of the malWash, using the asynchronous procedure call (APC) function of Windows to use hidden signals for scheduling between each execution block, further enhancing the distributed malware concealed performance. For this malware evasion technique, the general detection methods are no longer effective. Hăjmăsan et al. (2017) proposed to divide the processes in the system into groups according to their relevance. According to the behaviors generated by the execution of these process groups, heuristic methods were used to detect distributed malware. However, the benign processes selected by distributed malware are not necessarily related, and monitoring a large number of processes will burden the monitoring system and affect detection efficiency. Otsuki et al. (2019) proposed a technique for extracting synchronization objects from memory dumps, using stack traces and synchronization objects to find code blocks distributed in different processes. However, because the memory dump is a snapshot of system's memory at a specific moment, it cannot guarantee that all suspicious objects are saved in memory at the current moment.

In view of the characteristics of the above-mentioned distributed injection malware and the problems in the current corresponding research, this paper proposes a method of using I/O request package (IRP) sequence and local alignment in bioinformatics to detect distributed injection malware. When detecting distributed injected malware, drive filtering technique is used to extract all important IRP requests in Windows system as suspicious sequences. The local alignment algorithm is used to find whether there is an IRP sample sequence in the verification sequence, thereby confirming the existence of distributed malware in the system.

This paper mainly contributes in 3 points as follows:

1. We introduced a method for detecting distributed malware, and implemented in Windows system. Our method uses the IRP sequence of malware to compare with the IRP sequence generated by the system, and uses the local alignment algorithm to find the distributed malware in the system.
2. We verified the effectiveness of the IRP sequence as a malware feature, and provided a new idea to detect the evasive behavior of new malware.
3. Through the use of malware samples with distributed injection behavior, the effectiveness of the method proposed in this paper is verified, and the detection performance is verified by comparison experiments to be better than other similar researches.

The remainder of the paper is organized as follows. Section 2 presents the related work. Section 3 introduces the distributed malware model and the detection method proposed in this paper. Section 4 evaluates the detection performance of the IRP features proposed in this paper. Section 5 evaluates the performance of the distributed malware detection method. Finally, Section 6 concludes the paper and proposes further work.

2. Related work

Malware detection and analysis is still a popular research direction. In order to evade detection, malware evasion techniques are constantly being updated. More common technologies such as anti-debugging technology and environmental detection technology are widely used in many types of malware (Afianian et al., 2018). The main methods of these two types of anti-detection technologies are to detect the system environment and key nodes in the environment, such as detecting breakpoints, detecting special functions, and detecting special registers. Maiorca et al. (2015, 2013)proposed a mimicry attack method, which is a method of embedding malware in benign PDF files and using benign PDF as a carrier to carry out attacks. Code reuse is another evasion technique of malware. Checkoway et al. (2010) proposed return-oriented programming (ROP), Bletsch et al. (2011) proposed jump-oriented programming (JOP), and Schuster et al. (2015) proposed counterfeit object-oriented programming (COOP), these methods all make use of vulnerabilities in the system. They use ret instructions, jmp instructions or virtual addresses in the C++ language to destroy the connection of the instruction stream and call other instructions within the system to achieve the purpose of destruction. Hosseini (endgame, 2017) introduced ten common injection attack methods in Windows systems, such as process hollowing, PE injection, DLL injection, etc. The main idea is to inject malware into a benign process to disguise or evade detection. Ispoglou and Payer (2016) proposed malWash method based on software injection. This method splits the malware into blocks, which are executed in the order of the API call sequence of the program, inject code blocks into multiple benign processes, and use shared memory for coordinate between the code blocks. Pavithran et al. (2019) proposed D-TIME method, which improved the malicious behavior that may appear in the malWash method. It uses an inter-process asynchronous call technique called Semaphore based Covert Broadcasting Channel (SCBC) to implement a covert method of malicious behavior called between processes. This method can mask the malicious behavior that may be generated by inter-process communication.

For the above-mentioned malware evasion techniques, researchers have proposed different targeting strategies. In order to solve environmental awareness and anti-debugging techniques used by malware, Kirat et al. (2014) proposed that using bare metal environment can avoid the problem of malware detecting virtual environment. There are already many solutions to mimic attacks and code reuse attacks, the use of software features is a more common detection method. microsoft (2019) used API call sequence combined with hierarchical attention networks to detect malware. Kakisim et al. (2019) obtained more discriminative dynamic feature subsets from multiple API features to improve detection performance. Saracino et al. (2018) recorded various behavior categories and multiple levels of malware and used them as features to detect malware. Ding et al. (2018) used malware behavior to construct a common dependency graph (CDG) of malware families to detect malware. Chakraborty et al. (2020) used ensemble clustering and classification algorithms based on the combination of static features and dynamic behaviors, which can not only effectively detect malware, but also predict potential unknown malware. The above methods can effectively detect regular malware. In addition, because the execution of the malware will leave traces on the hardware, Demme et al. (2013) found that monitoring hardware performance counter of the malware at runtime, through offline analysis can effectively distinguish the malware, confirming the effectiveness of using the hardware performance counter (HPC). Ozsoy et al. (2015, 2016) used a two-level detection method, using low-level hardware features for pre-detection, increasing the weight of suspicious files, and using software detectors for re-detection. Dai et al. (2018) used the HOG extracted from the memory dump grayscale as a feature to verify the effectiveness of the feature detection, and can effectively describe the malware to a certain extent. Zhang and Ma (2016) proposed using IRP combined with n-gram as features, and artificial immune algorithms for malware detection. However, with the above dynamic detection methods, conventional monitoring methods cannot monitor all processes. In this way, when detecting distributed malware, the detection cannot be failed due to comprehensive monitoring of suspicious behavior. And with HPC as features, Zhou et al. (2018) research shows that HPC cannot effectively correspond to upper-layer APIs in describing malicious behaviors, resulting in reduced accuracy. The current commercial AV software cannot effectively detect distributed malware (Pavithran et al., 2019). For the research on distributed malware, Hăjmăsan et al. (2017) proposed to divide the processes in the system into groups according to relevance. According to the behaviors generated by the operations performed by these process groups, heuristic detection is used to detect the distributed malware. However, the benign processes selected by distributed malware are random, and monitoring a large number of processes is prone to a sharp increase in the amount of data, which affects the detection accuracy. Otsuki et al. (2019) proposed a technique for extracting synchronization objects from memory dumps, using stack traces and synchronization objects to find injection code distributed in different processes. Because of the memory dump is a snapshot of the system's memory at a specific moment, the dump file cannot guarantee that all suspicious objects are saved in the memory at the current moment (Table 1).

Table 1 – Comparison between our method and the benchmark method.

Method	Monitor platform	Feature source	Features	Detection algorithm
Hăjmăsan et al. (2017)	BVM	Event	Bytecode signature of event	Heuristic
Otsuki et al. (2019)	PC	Memory dump	Stack, heap, thread synchronization object	Heuristic
Our method	Sandbox	Driver request	Driver, IRP	Local alignment

3. Threats and solutions

3.1. *Distributed injection malware*

In order to enable the malware to hide in an operating system from being detected, Ispoglou and Payer (2016) proposed the malWash method. This method is a distributed injection attack, which is an improved version of code injection. The malWash method splits the malware binary file into code blocks, and injects blocks into other benign programs running in the system at runtime. It uses emulators to coordinate the operation of various code blocks to maintain the same function as the original malware.

Fig. 1 shows the framework and working flow of malWash. In the preparation phase, the malware sample is decompiled into assembly code through IDA pro[1], the assembly code is divided into several blocks, and an executable file is generated, which is responsible for injecting the code block into benign processes. In the injection phase, the executable file generated by malWash injects each code block as a load into the process, and then the emulator controls each code block and executes it in sequence. During the operation, the number of emulators is determined by the malware author. The emulator uses shared memory (or pipes, files, sockets, etc.) for data interaction between the various blocks. Another function of the emulator is to keep the malware running robustly in the system. When a process running the emulator and the code block is closed, the data run by the emulator will be transferred to other emulators. Then the malWash method can regenerate the emulator to ensure that the number of emulators is constant in the system. It means that malware using the malWash method is not only difficult to detect, but also difficult to remove.

Malware can be split in three splitting modes[2] through the malWash method. The first is the basic block split (BBS)

[1] Interactive Disassembler Professional, see https://www.hex-rays.com/products/ida/.

[2] For more information about the split mode, see Ispoglou and Payer (2016) and https://github.com/HexHive/malWASH.

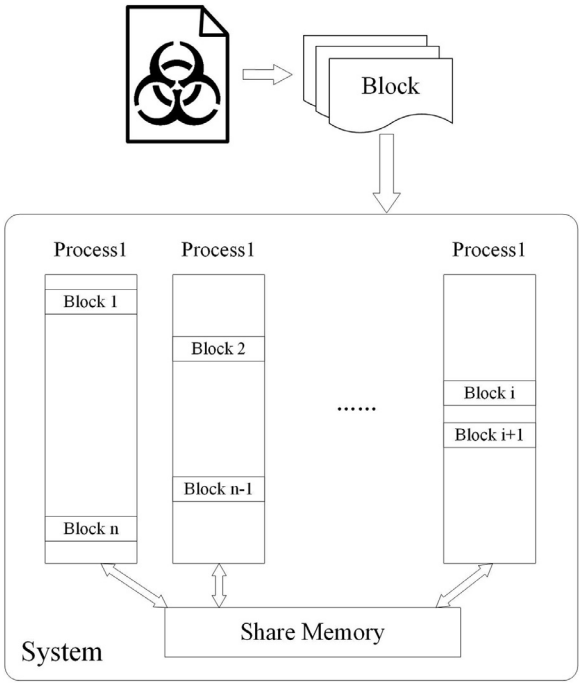


Fig. 1 – malWash running structure.

mode, which splits the malware according to the basic blocks identified in IDA, the basic block is a complete piece of code that ends with a control flow instruction; the second is below the AV signature threshold (BAST) mode, the mode block is smaller than the basic block split mode and larger than the third paranoid mode, the size of the split block can be selected according to the configuration; the third is the paranoid mode, which contains only one instruction per basic block. When the emulator completes a block, it will update the state of the next block, release the object of the current block, and then return to the waiting state.

Based on the malWash method, Pavithran et al. (2019) proposed the D-TIME method. This method can perform more covert code block injection without any new thread preset. And in the inter-process communication, the shared memory method is abandoned, and the Semaphore based Covert Broadcasting Channel (SCBC) is used. And like malWash, the D-TIME method makes the current AV software unable to effectively detect the distributed injected malware.

3.2. *IRP features*

After distributed processing, the traditional dynamic detection methods such as monitoring the malware process and extracting the complete API call sequence cannot detect the malware effectively. However, the behavior of the malware's API calls still exists in operating system and is observable. We use filter drive technique to hook all I/O request packages (IRP) generated in the system to analyze whether there is malware in the current system. IRP is an important data structure in the system. When the upper-layer application communicates with the underlying device or software driver, the application will issue an I/O request. The various IO requests of the driver will be passed to different dispatch functions according to the IRP type.

Generally, different user mode API requests have corresponding kernel mode API. The kernel mode API will issue I/O requests to the hardware device according to the needs of the upper layer. The driver will perform the corresponding hardware operation according to the I/O request issued by the upper API. We use the Windows' CreateFile function to create a file as an example, the entire process is shown in Fig. 2. The CreateFile function is one of the most common functions that appear in malware. When performing a file creation operation, the user first needs to call the CreateFile function of the Win32 API, and then the system pass it to the Kernel32.dll subsystem to call the NtCreateFile function of ntdll.dll. The NtCreateFile function sends an I/O request to the driver, and the data is passed layer by layer to the hardware in the form of IRP.

We can also see from Fig. 2 that the driver in the system is usually divided into multiple layers, called the I/O stack. It will help reduce the internal coupling of the drivers and improve processing efficiency. The driver performs different tasks from the top layer to the bottom layer. When an API function sends a request to the hardware, different IRP data sequences will be obtained when monitoring different drivers. If the monitored driver is on the upper layer, a richer IRP will be obtained, but

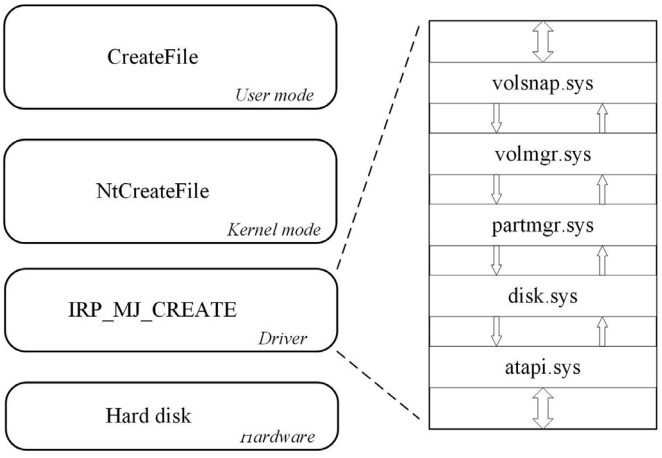


Fig. 2 – A hierarchical model for accessing hardware by an API function in Windows system.

the data may be too redundant; while the driver selected for monitoring is on the lower layer, the obtained IRP data cannot describe more detailed program behavior.

3.3. IRP feature conversion

The IRP sequence used in this paper is similar to MBMAS (Zhang and Ma, 2016), but there are differences. MBMAS only uses IRP for file operations for monitoring. Usually the sequence generated by the API involved in the application software is basically the same, so the IRP sequence generated between the API and the hardware can also be consistent to a considerable extent. In this paper, we not only use the IRP operation of the hard disk reading related to the file, but also use other drivers such as the network and the serial port. We use the driver filtering provided by the open source IRPMon Irpmon, modify it on the basis of it, and conduct selective monitoring of IRP throughout the Windows system. For details on which drivers to monitor and which IRP requests for these drivers, see Section 4.2.

According to the extracted IRP request sequence, and the sequence can be identified and matched by the longest common subsequence, simple character replacement is used in this paper. The choice of drivers is limited, and the IRP request corresponding to each driver is also countable. For example, the top five types of IRP requests most commonly used by the disk.sys driver are IRP_MJ_READ, IRP_MJ_WRITE, IRP_MJ_CREATE, IRP_MJ_CLOSE, IRP_MJ_DEVICE_CONTROL. According to the combination of driver and IRP request type, we use letters instead of each combination of driver and IRP request, for example, set the IRP_MJ_CREATE request of disk.sys to the letter "A". Each malware can obtain a set of letters as its characteristic data through dynamic monitoring.

3.4. Local alignment algorithm

The idea of the local alignment algorithm used in this paper is based on the Smith-Waterman algorithm, which is an algorithm used by bioinformatics knowledge to match protein sequences or DNA sequences. The basic principle is based on the combination of the longest common subsequence and dynamic programming. The algorithm is not to perform a full sequence alignment between two sequences, but to find fragments with a high degree of similarity in the sequences. In this paper, the malware's IRP sequence is used as the sample sequence, and the system extracts the global IRP sequence as a suspicious sequence. The purpose is to find the sample sequence hidden in the suspicious sequence.

In the use of this paper, specific drivers and IRP requests are replaced by letters, visually similar to bioinformatics protein sequences. We set the suspicious sequence $A = \{a_1, a_2, a_3, ..., a_n\}$, and the sample sequence $B = \{b_1, b_2, b_3, ..., b_n\}$, where n and m are the lengths of sequences A and B, and $n > m$. $s(a, b)$ is the similarity score of character a and character b, which is set to 10 in this paper, W_k is the gap penalty, and H is the score matrix.

1. When the comparison starts, the score matrix is initialized, and the value is recorded from 0. The first row and the first column of the matrix are 0, set the row corresponds to the sequence A, the column corresponds to the sequence B, the matrix size is $H_{n+1,m+1}$.
2. Calculate each item in the score matrix H_{ij}:

$$H_{ij} = max\begin{cases} H_{i-1,j-1} + s(a,b) \\ H_{i-1,j} - W_k \\ H_{i,j-1} - W_k \\ 0 \end{cases} \tag{1}$$

$$s(a_i, b_j) = \begin{cases} 10, & a_i = b_j \\ -10, & a_i \neq b_j \end{cases} \tag{2}$$

3. Tracing back the sequence, starting with the largest item H_{ij} found in the matrix H: If $a_i = b_j$, then trace back to cell $H_{i-1,j-1}$; If $a_i \neq b_j$, follow the order of $H_{i-1,j-1}$, $H_{i,j-1}$, $H_{i-1,j}$, and trace back the maximum value in the cell, if the cell with the same maximum value, select the top cell according to the above priority.
4. Match to the locally optimal sequence.

The gap penalty W_k mentioned in the above algorithm uses a finite state machine for penalty. This is to avoid the discontinuity of malware execution due to system scheduling, thereby affecting the IRP sequence matching score. Since the length of the sample sequence B is not fixed and fixed, the state machine needs to dynamically adjust the penalty length during the matching process. There may be a situation where the gap length and the matched characters appear to cross. In this case, the combination of the gap and the consecutive matching characters is a GAP (the GAP length does not exceed the upper limit of the penalty), and the penalties will be counted when they appear continuously. See Algorithm 1 for the complete method of using IRP to detect distributed malware.

Algorithm 1 Distributed malware detection method.

Input: IRP Sequence of sample, $I = \{I_1, I_2, I_3, \cdots, I_m\}$;Malware feature set $S = \{S^1, S^2, S^3, \cdots, S^r\}$;IRP type conversion letter dictionary, D;Threshold, h;
Output: Class label of sample, S_k;

```
Convert IRP type to letter sequence S', length is m;
for S^t ∈ S do:
    Get S^t, length is n;
    function LocalAlignment(S', S^t):
        Initialization matrix M_{n+1,m+1} = {0};
        Calculate each item M_ij in matrix M according to equation(1);
        Find Score_opt = max_{i∈[1,m],j∈[1,n]} M[i][j] then trace back;
        if S_i^t = S'_j then trace back to M_{i-1,j-1};
        else trace back to max{H_{i-1,j-1}, H_{i,j-1}, H_{i-1,j}};
        end if
        Matched to the maximum sequence, score is v', family class is S^t ∈ S_k, k is class label;
    end function
    Record v' to set V;
end for
Find v = max{V};
if v ≥ h then return class label S^t ∈ S_k;
else return unknown type or benign;
end if
```

4. IRP performance evaluation

In order to prove that IRP features can detect distributed malware, we first need to evaluate the performance of IRP features to determine that the feature is suitable for detecting regular malware.

4.1. *Experimental environment and dataset*

The computer environment for the sandbox used in the experiment is the CPU uses Intel(R) Core(TM) i5-6500 @3.20GHz; uses 8GB DDR3 memory. The Cuckoo sandbox host machine is installed in the Ubuntu 16.04 system environment, and the Guest machine uses Windows 7 32-bit operating system with 2GB of memory.

The malware samples used in the experiment in this paper comes from MalwareBenchmark. It is authorized to use a total of about 27k samples of malware, with a collection range of 2013-2015, and a small amount of malware downloaded from theZoo. The number of malware split by ourselves is 100 for distributed malware detection experiments. All samples can be divided into 227 families according to families by VirusTotal. The issues discussed in this paper are mainly distributed malware detection methods, and the classification performance are not the main content of the discussion.

4.2. *IRP features*

In the operating system, the operation of the application program needs to interact with the low-level hardware. The driver is a special program in the operating system, which is responsible for processing the input and output between the API and the hardware, called I/O. In Windows, a complete set of drivers includes a motherboard, serial port, video card, storage device, etc. Each driver contains multiple types of I/O requests. There are hundreds of basic drivers in the Windows 7 system in the experimental environment of this paper, which means that the combination of drivers and I/O requests will reach thousands. If monitoring all the drivers it will generate huge data, which will cause the monitoring data to be stored in the virtual memory (hard disk), resulting in distortion of the monitoring data. Therefore, we are based on the data collected by malware and benign software running in the system, as well as our understanding of malware behavior. Among the hundreds of combinations that can be monitored, 23 IRPs are selected and listed in Table 2.

Compared with the study Zhang and Ma (2016), we used IRPs that belong to a certain driver, instead of using all IRPs without classifying them by driver. In this way, when performing feature vectorization, the IRP sequence discrimination can be made higher. The main reason we choose these IRPs is that these IRPs appear most frequently in statistics, and the number of statistics ranks top. In addition, the driver is layered in operating system. For example, accessing the hard disk usually goes through volsnap.sys, volmgr.sys, pnpmanger.sys, disk.sys, atapi.sys from top to bottom, and finally reaches the hardware. Moreover, there is not much difference between the IRP of the driver of the hardware adjacent layer, and the classification result will not be significantly different.

Table 2 – Drivers and IRPs used in this paper.

Drivers	IRPs
disk	Create
	Read
	Write
	Flush
	DeviceControl
	Cleanup
	Close
	Pnp
kbdclass	Read
mouclass	Read
Ndisproxy	Create
	DeviceControl
	Cleanup
	Close
	Pnp
usbxhci	Create
	DeviceControl
	Close
	InternalDeviceControl
	QueryRelations
	QueryInterface
	QueryCapabilities
	Cleanup

Fig. 3 shows the IRP statistics of different malware. The x-axis unit is seconds, and the y-axis counts the number of IRP requests per second. The figure shows the top four IRP requests for these types of malware. It can be seen from the figure that the ransomware's IRP requests on the hard disk are up to three items: Read, Write and Flush, and network connection requests, which is consistent with the behavior of ransomware. With the keylogger Trojan, the requests that the IRP can record are keyboard reading, network functions, and hard disk reading and writing.

4.3. *IRP classification performance*

To verify the effectiveness of the IRP classification performance we chose, that is we can rely on the IRP feature to properly classify malware, we used sandbox extraction for routine malware IRP validation experiments. The regular malware mentioned here is the malware collected on the network, which is usually run by a single process or injected into one process. Since the IRP contains the process ID and thread ID that initiated the request, these can identify to which process the IRP request originated. IRP sequence classification experiment uses word2vec after converting IRP into characters, so that the IRP character sequence is converted into a vector that can be recognized by the neural network classifier. The IRP sequence vector uses a double-layer GRU neural network as the classifier model. For the parameters of word2vec and double-layer GRU network, refer to the API sequence classifier model in paper (Dai et al., 2019). This experiment will be divided into three control groups: 1. IRP sequences containing driver types, and using word2vec conversion features, using GRU classifier for classification; 2. IRP sequences that do not contain driver types, using word2vec features and GRU classifier for classification; 3. According to the MBMAS method in

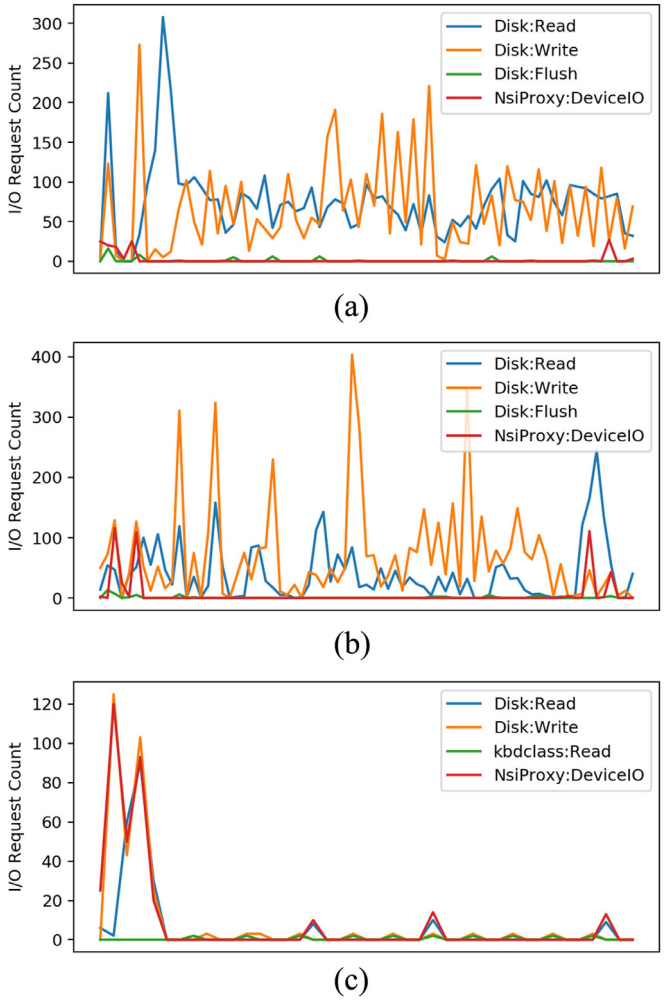


Fig. 3 – Statistics of IRP requests generated by malware.

Table 3 – IRP classification accuracy.

Method	Features	Accuracy
word2vec-GRU	drivers-IRP	93.4%
word2vec-GRU	only IRP	87.2%
4-gram-DBT	only IRP	86.5%

paper (Zhang and Ma, 2016), using IRP features that do not contain the driver types, the boosting decision tree (BDT) with the highest accuracy in Zhang and Ma (2016) is selected for classification, and the vector extracted by 4-gram is used as the input feature. After experiments, the overall accuracy of the three control groups is shown in Table 3, and the ROC curve display is shown in Fig. 4.

We can see from the results that the detection performance of IRP can achieve higher accuracy. The combination of driver types and IRP we proposed can achieve higher accuracy in the overall classification performance than the method using only IRP as feature. It can be seen from Fig. 4 that using the IRP sequence with the driver types as features can also be better than the IRP sequence features without driver types in terms of performance such as precision and recall. Using the driver and the driver's corresponding IRP as features can clearly distinguish which driver and request type the IRP request belongs to. It can clearly explain the behavior of the current program than using the IRP alone. In order to further illustrate the performance of IRP features in classification, we use the confusion matrix to display the classification results in the previous experiment. The classification result of the control group 1 is shown in Fig. 5(a). The results of the MBMAS method of control group 3 are shown in Fig. 5(b).

From Fig. 5, we can see that the IRP sequence feature of our method is higher than the MBMAS method in classification accuracy, but the performance of classification is not the focus of this paper. We can see from the confusion matrix that comparing various types of malware, except for the Downloader, other types of malware can basically achieve an accuracy of over 91%. The downloader's behavior is too simple, and most downloaders require script malware or Powershell to run. And the behavior of some downloaders is compared with the IRP sequence features of this paper, which is similar to Backdoor and Trojan. From the overall classification detection rate, the

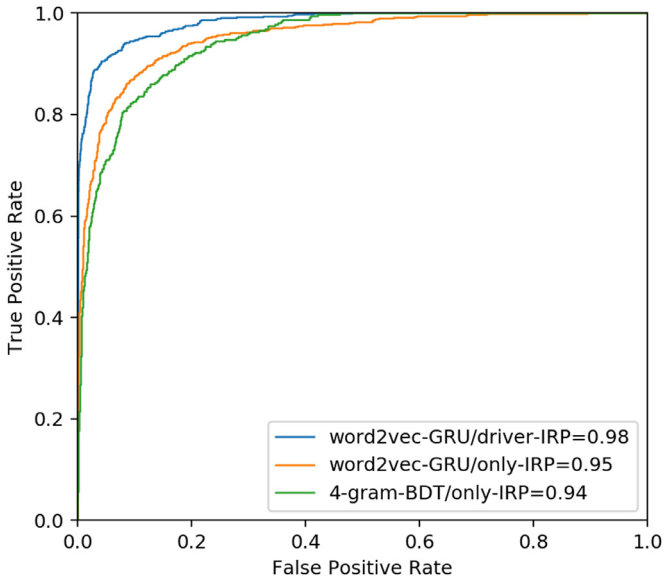


Fig. 4 – ROC curve of IRP feature classification performance

combination of the driver and IRP proposed in this paper is effective for most regular malware classifications, which provides us the next step to verify the feasibility of distributed malware detection.

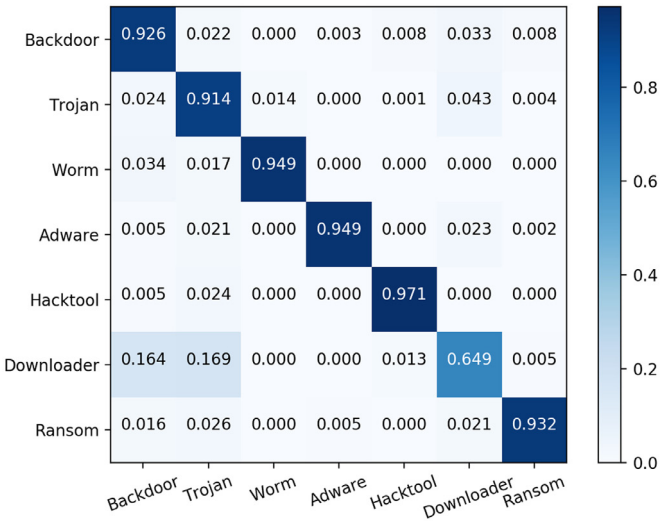
5. Distributed malware detection evaluation

5.1. *Distributed malware detection evaluation*

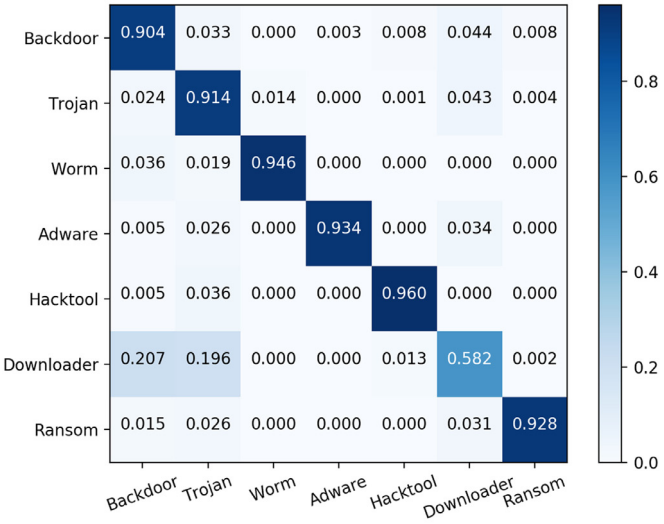
Section 4 introduces the IRP feature extraction method, and extracts all the malware IRP sequences in the dataset as the sample feature set. In this section, all experiments will use local alignment algorithm to compare IRP features with suspicious sequences to detect distributed malware. In this experiment, we default that the process of distributed malware injection is a black box for the detector, so we need to monitor all the key IRPs in the system (Table 2). We set a complete system IRP sequence as a suspicious sequence, and the malware IRP sequence as a sample sequence. We compare the suspicious sequence with the sample sequence. If the sample sequence is in order in the suspicious sequence, and the matched characters exceed 80% of the sample sequence itself, it is marked as "detected". If multiple sample sequences are matched at the same time, the matching sequence with the highest score is selected according to the scoring rules of the local alignment algorithm. Fig. 6 shows the IRP sequence chunk of Trojan.keylogger matched using the local alignment algorithm.

This experiment uses IDA6.5 version, using the split plugin provided by malWash (Ispoglou and Payer, 2016), to set up 100 distributed malware sandbox environments and 15 undistributed malware sandbox environments for IRP test performance evaluation, and compare related research. In this experiment, we used the basic block split mode (BBS mode), set the number of malware injection processes to 3, and the injected malware processes were Opera (version 68.0.3618.173), Adobe (version 19.008.20080), and calculated (Windows 7 version 6.1). We use the method of paper (Hăjmăsan et al., 2017; Otsuki et al., 2019) as a benchmark for comparison. Because the use of HPC proposed in the D-TIME (Pavithran et al., 2019) paper has a high probability of detecting distributed malware, we chose two studies to compare with ours. One is the method for detecting malware using HPC proposed by Ozsoy et al. (2016), and the other is SMASH (Dai et al., 2019) using an integrated algorithm with multiple features (HPC, API sequence and memory dump). The experimental results are shown in Table 4. The second column in the table indicates the number of malware that has been correctly classified. The third column shows the number of malware that was misclassified as benign software. The fourth column is the total number of misclassified. The last column records the number of malware that could not be identified. Since the method in paper (Hăjmăsan et al., 2017; Otsuki et al., 2019) is a detection method rather than a classification, it does not consider misclassification.

From the experiments, we can see that our method and paper (Hăjmăsan et al., 2017; Otsuki et al., 2019) have a high accuracy rate in identifying distributed malware, with an accuracy rate of over 82%. The two methods in Ozsoy et al. (2016), Dai et al. (2019) cannot effectively detect distributed malware, and the accuracy rate is less than 20%. The low accuracy rate makes these two methods unable to effectively detect distributed malware. This paper attempts to reproduces the function of Hăjmăsan et al.'s research on behavior monitoring by process group, but because the processes in which distributed malware is hidden are black boxes for researchers, this will produce a large error in the actual detection environment. Otsuki et al. used memory dumps to extract synchronization objects generated by distributed malware. We have also demonstrated memory dumps to identify malware in previous studies (Dai et al., 2018). According to our research results, memory dumps can identify malware. However, the memory dump file



(a)



(b)

Fig. 5 – Classification results of two groups of experiments.

```
-----AE-AACEAA-AC-E-AAAC-EEAAAC-EEAAACEEAAAC-EG----EAAAACEEAAACG--E----EEAAAC----E--EEAAAC-E--AA-AACG---C--
     AE AACEAA AC E AAAC EEAAAC EEAAACEEAAAC EG    EAAAACEEAAACG  E    EEAAAC    E  EEAAAC E  AA AACG   C
FBAAEAEEAACEAAEACEEAAAACEEEAAACEEEAAACEEAAACEEGGAACEAAAACEEAAACGEEEAGGCEEAAACEEAAEACEEAAACEEAAAACAACGAGACBA
```

Fig. 6 – IRP sequence chunk matched by local alignment algorithm.

Table 4 – Statistics on the ability to identify distributed malware using IRP and other methods.

Method	Malware	Benign	Misclassification	Unidentification
Hăjmăsan et al. (2017)	85/100	7/15	-	8
Otsuki et al. (2019)	82/100	3/15	-	15
Our method	95/100	2/15	3/115	3
HPC Ozsoy et al. (2016)	12/100	0/15	58/115	30
SMASH Dai et al. (2019)	19/100	1/15	64/115	17

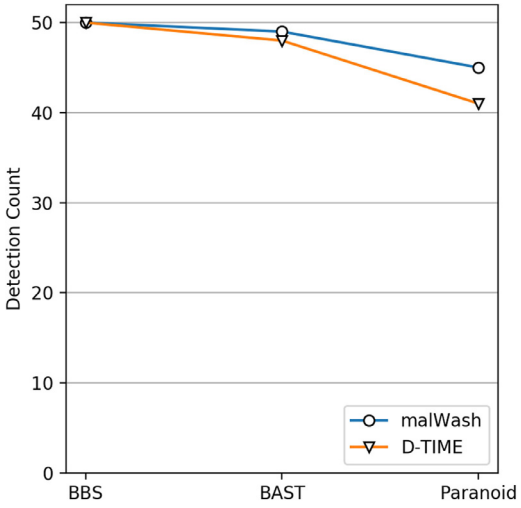


Fig. 7 – The two distributed malware threat models use different split modes and count the number of detected malware.

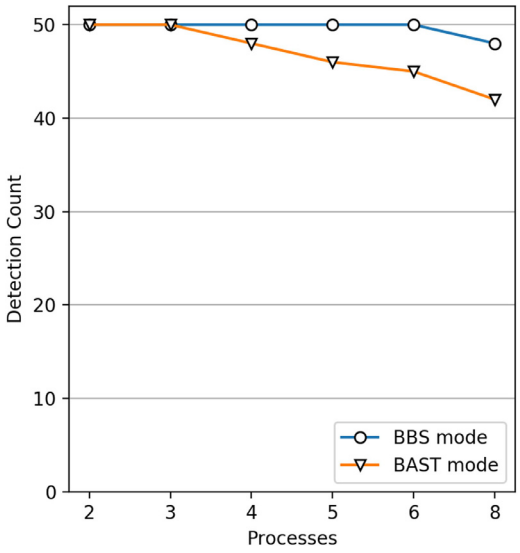


Fig. 8 – Statistics on the detection results of different injection processes in the two block modes.

needs to be constantly updated. Because the memory data is volatile, a considerable part of the data will be released after several accesses, which is also the reason for the low detection rate. Compared with using HPC as the detection method of malware features, because the features of malware are collected in units of processes, when the malware is split into several blocks and injected into several benign processes, the HPC patterns of these malwares will also be split, and the paper (Pavithran et al., 2019) stated that running distributed malware itself would bring performance overhead and cause performance counters to rise. Similarly, our previous research SMASH (Dai et al., 2019) method also has the same problem in dealing with distributed malware.

From the statistics in the last three columns of Table 4, it can be seen that the precision of classification of our method is better than the other two distributed malware detection methods (Hăjmăsan et al., 2017; Otsuki et al., 2019), and the total number of misclassifications and unrecognized is lower than these two methods. The detection method proposed in this paper, although there will be false positives, the classification ability of our method is better than similar research under the same conditions.

5.2. *Split mode evaluation*

Next, we will verify the effectiveness of our IRP detection method based on the malware with different blocking modes mentioned in the malWash (Ispoglou and Payer, 2016). We randomly selected 50 distributed malware that can be correctly classified as samples from the experiment in Section 5.1, and split the malware according to the three modes proposed in malWash. The statistical results according to the number of detected malware are shown in Fig. 7.

It can be seen from the figure that the method of this paper has good performance in detecting BBS mode and BAST mode, and the performance of detecting paranoid mode is slightly reduced. Comparing the two threat models of malWash and D-TIME, D-TIME is inherently more concealed, and more IRP requests are generated in the system, which affects the performance of the detector. Compared with the BAST mode, the paranoid mode generates more code chunks. Whether it is malWash or D-TIME interacts more frequently between different processes, resulting in a decrease in the score of the local alignment, which is the main reason for the decline in detection performance.

5.3. *Injection processes number evaluation*

We use the malWash threat model, as well as two sharding modes: BBS mode and BAST mode, to inject malware into benign processes ranging from 2 to 8 to verify the effectiveness of our IRP sequence detection. We used the 3 benign processes in Section 5.1 and another 5 benign processes, including: notepad.exe, mspaint.exe, Media Player (version 12.0.7600.16385), IE (version 8.0.7600.16385), TeamViewer (version 14.0.12762). The final detection result statistics are shown in Fig. 8.

From the experimental results, in BBS mode, when the number of distributed malware injection processes is not more than 6, our method can detect all distributed malware, when the injected processes continue to increase to 8, the detection performance started to decline. With the BAST mode, when the number of injected processes is more than 4, the detection accuracy rate decreases. However, this is maintained at a relatively high level and the detection rate is above 90%. The main reason for the decrease in detection accuracy is that the interaction between each block generates IRPs, and each benign program normally runs also generates IRPs. When the amount of data is large enough, the score of the local alignment is too low to match completely. In addition, as more processes are started, the success rate of using malWash or D-

TIME to inject benign processes is also lower. During the experiment, we often crash the process due to injection failure.

5.4. *Discussion and limitations*

The distributed malware discussed in this paper runs in a sandbox environment, which is used to extract IRP requests from operating system and generate suspicious sequences for detection. The IRP sequence used for detection and evaluation in this paper, the extraction environment needs to be in a very clean system. The clean here only contains the minimum requirements for a system to use in normal office, such as Office, pdf, simple video player, etc. It does not include entertainment software such as games and music and some large-scale work software such as CAD and EDA. Such a system environment greatly excludes the generation of unnecessary IRP sequences, which affects the detection of distributed malware. This is basically the same as the system environment used for research using sandbox or bare-mental type.

The IRP sequence of a normal downloader is very similar to the IRP sequence of a malware downloader in terms of patterns. In the experiments of this paper, the difference between the downloader and the normal download software cannot be effectively distinguished. The downloader also behaves similarly to some backdoor malware and trojan malware, which is easy to produce false positives during the identification process. Malware that behaves similarly, or some benign software that behaves similarly to malware, can cause false positives, which is also a disadvantage of detection methods based on IRP sequences. In addition, the use of local alignment to detect distributed malware has no resilience. When the behavior of the malware does not match the original observation, or unknown malware appears, it cannot be effectively detected.

6. Conclusion

The distributed injected malware implements a malware concealment technique with its fragmentation and synchronization technique, which can effectively evade the dynamic detection malware technique. Most advanced malware detection techniques cannot effectively detect distributed malware. In response to this problem, this paper proposes a detection method that uses IRP sequence and local alignment algorithm. IRP extracted from malware is used as a sample sequence to compare with suspicious sequence extracted from operating system. Through experimental verification, the IRP sequence features proposed in this paper can be used as the features for malware detection accuracy rate can reach 93.4%. At the same time, the use of sandboxes to detect distributed malware modified by real malware confirmed that the method proposed in this paper can effectively detect most types of distributed malware, and the accuracy of detecting distributed malware can reach 93%. The detection performance is better than recent similar research. Since the detection method proposed in this paper has high false positives for malware with similar behaviors, and the detection accuracy needs to be improved, we will try to solve it in future work.

Declaration of Competing Interest

The authors declare that they do not have any financial or nonfinancial conflict of interests

CRediT authorship contribution statement

Yusheng Dai: Conceptualization, Methodology, Software, Writing - original draft. **Hui Li:** Supervision, Project administration, Funding acquisition. **Yekui Qian:** Formal analysis, Investigation. **Yunling Guo:** Resources, Data curation. **Ruipeng Yang:** Writing - review & editing. **Min Zheng:** Software, Visualization.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61571364), and the Innovation Foundation for Doctoral Dissertation of Northwestern Polytechnical University (CX201952).

REFERENCES

Afianian A, Niksefat S, Sadeghiyan B, Baptiste D. Malware dynamic analysis evasion techniques: a survey. Cryptography and Security 2018. arXiv

Bletsch T, Jiang X, Freeh V, Liang Z. Jump-oriented programming: a new class of code-reuse attack; 2011. p. 30–40.

Bulazel A, Yener B. A survey on automated dynamic malware analysis evasion and counter-evasion: pc, mobile, and web; 2017. p. 2.

Chakraborty T, Pierazzi F, Subrahmanian VS. EC2: Ensemble clustering and classification for predicting android malware families. IEEE Trans. Dependable Secure Comput. 2020;17(2):1–14.

Checkoway S, Davi L, Dmitrienko A, Sadeghi A, Shacham H, Winandy M. Return-oriented programming without returns; 2010. p. 559–72.

Cuckoo sandbox. https://cuckoosandbox.org/.

Dai Y, Li H, Qian Y, Lu X. A malware classification method based on memory dump grayscale image. Digit. Invest. 2018;27:30–7.

Dai Y, Li H, Qian Y, Yang R, Zheng M. SMASH: A malware detection method based on multi-feature ensemble learning. IEEE Access 2019;7:112588–97.

Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo SJ. On the feasibility of online malware detection with performance counters. ACM SIGARCH Comput. Archit. News 2013;41(3):559–70.

Ding Y, Xia X, Chen S, Li Y. A malware detection method based on family behavior graph. Comput. Secur. 2018;73:73–86.

Hăjmăsan G, Mondoc A, Portase R, Cre O. Evasive malware detection using groups of processes; 2017. p. 32–45.

Ispoglou KK, Payer M. In: 10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, USA, August 8-9, 2016. malWASH: Washing malware to evade dynamic analysis. USENIX Association; 2016.

Irpmon. https://github.com/MartinDrab/IRPMon.

Kakisim AG, Nar M, Carkaci N, Sogukpinar I. Analysis and evaluation of dynamic feature-based malware detection methods; 2019. p. 247–58.

Kirat D, Vigna G, Kruegel C. Barecloud: bare-metal analysis-based evasive malware detection; 2014. p. 287–301.

Maiorca D, Ariu D, Corona I, Giacinto G. A structural and content-based approach for a precise and robust detection of malicious pdf files; 2015. p. 27–36.
Maiorca D, Corona I, Giacinto G. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection; 2013. p. 119–30.
malwarebenchmark. http://malwarebenchmark.org/.
Otsuki Y, Kawakoya Y, Iwamura M, Miyoshi J, Faires J, Lillard T. Toward the analysis of distributed code injection in post-mortem forensics; 2019. p. 391–409.
Ozsoy M, Donovick C, Gorelik I, Abughazaleh NB, Ponomarev D. Malware-aware processors: a framework for efficient online malware detection; 2015. p. 651–61.
Ozsoy M, Khasawneh KN, Donovick C, Gorelik I, Abu-Ghazaleh N, Ponomarev D. Hardware-based malware detection using low-level architectural features. IEEE Trans. Comput. 2016;65(11):3332–44.
Pavithran J, Patnaik M, Rebeiro C. D-time: Distributed threadless independent malware execution for runtime obfuscation; 2019.
Saracino A, Sgandurra D, Dini G, Martinelli F. Madam: effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secure Comput. 2018;15(1):83–97.
Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi A, Holz T. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications; 2015. p. 745–62.
Sequence intent classification using hierarchical attention networks,2019. https://www.microsoft.com/developerblog/2018/03/06/sequence-intent-classification/.
thezoo. https://github.com/ytisf/theZoo.
Ten process injection techniques: a technical survey of common and trending process injection techniques. 2017. https://www.endgame.com/blog/technical-blog/ten-process-injection-techniqes-technical-survey-common-and-trending-process.
Virustotal. https://www.virustotal.com/.
Zhang F, Ma Y. Using IRP with a novel artificial immune algorithm for windows malicious executables detection; 2016. p. 610–16.
Zhou B, Gupta A, Jahanshahi R, Egele M, Joshi A. Hardware performance counters can detect malware: Myth or fact?. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ACM; 2018. p. 457–68.

Yusheng Dai received the B.S. degree in computer science and technology from Zhengzhou University, Zhengzhou, China, in 2011, and the M.S. degree in software engineering from Beijing Institute of Technology, Beijing, China, in 2015. He is currently a Ph.D candidate at Northwestern Polytechnical University, Xi'an, China. His research focuses on malware analysis and machine learning.

Hui Li received the B.S. degree in Electrical Engineering, M.S. degree in Circuits and Systems and Ph.D. degree in System Engineering from Northwestern Polytechnical University, Xi'an, China, in 1991, 1996 and 2006, respectively. He joined School of Electronic Information, Northwestern Polytechnical University in 1993 and was promoted to associate professor in 2002. Since 2008, he has been a Professor of School of Electronic Information, Northwestern Polytechnical University. His research interests include communication signal processing, massive MIMO, mmWave communications, avionics integrated system simulation, multi-sensor information fusion.

Yekui Qian received the Ph.D. degree in technology of computer application from University of Science and Technology, Nanjing, China, in 2010. He is a Professor of the PLA Strategic Support Force Information Engineering University. His research interests include cyberspace security, vulnerability mining and exploitation, log mining and analysis, malware analysis and stream monitoring and analysis.

Yunling Guo received the B.S. degree in Department of Electronics from Shaanxi Institute of Technology, Hanzhong, China, in 1993. She was rated as a senior experimenter, Xi'an, China, in 2010. She received the M.S. degree from Xi'an Technological University, Xi'an, China, in 2011. Currently, She works in the Electrical and Electronic Experiment Center of Shaanxi University of science and technology, Hanzhong, China. Her research interests are teaching and research in electrical and electrical electronics, signal and information processing.

Ruipeng Yang received the B.S. degree from Henan Institute of Finance and Economics, Master's degree in technology of computer application and the Ph.D. degree in information and communication engineering from the PLA Strategic Support Force Information Engineering University, Zhengzhou, China, in 2005, 2008 and 2020. Her research focuses on intelligent information processing and network security.

Min Zheng received the B.S. degree in Cryptography Engineering from PLA Information Engineering University, Zhengzhou, China, in 1996. He works for Zhongke Yuxin technology development (Xuchang) Co., Ltd, Xuchang Co.Ltd. He has led many major projects. He is a senior engineer, mainly engaged in cryptography, network security, information security, malware analysis and detection.