Expert Systems With Applications 52 (2016) 16–25

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Malicious sequential pattern mining for automatic malware detection

Yujie Fan[a], Yanfang Ye[b], Lifei Chen[a,c,*]

[a] School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China
[b] Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, USA
[c] Department of Computer Science, University of Sherbrooke, Sherbrooke, Canada

ARTICLE INFO

Keywords:
Malware detection
Instruction sequence
Sequential pattern mining
Classification

ABSTRACT

Due to its damage to Internet security, malware (e.g., virus, worm, trojan) and its detection has caught the attention of both anti-malware industry and researchers for decades. To protect legitimate users from the attacks, the most significant line of defense against malware is anti-malware software products, which mainly use signature-based method for detection. However, this method fails to recognize new, unseen malicious executables. To solve this problem, in this paper, based on the instruction sequences extracted from the file sample set, we propose an effective sequence mining algorithm to discover malicious sequential patterns, and then All-Nearest-Neighbor (ANN) classifier is constructed for malware detection based on the discovered patterns. The developed data mining framework composed of the proposed sequential pattern mining method and ANN classifier can well characterize the malicious patterns from the collected file sample set to effectively detect newly unseen malware samples. A comprehensive experimental study on a real data collection is performed to evaluate our detection framework. Promising experimental results show that our framework outperforms other alternate data mining based detection methods in identifying new malicious executables.

1. Introduction

Malware, short for ***mal***icious soft***ware***, is software that design to damage or destruct computers without owners' permission (Schultz, Eskin, Zadok, & Stolfo, 2001). Due to the rapid development of information technology, malware has posed a serious threat to networks as well as computer systems. For instance, worm has increasingly threaten the hosts and services by exploiting the vulnerabilities of the largely homogeneous deployed software base (Sun & Chen, 2009). In addition, in the application of the online transaction, trojan horses often steal sensitive information from online users through website phishing (Abdelhamid, Ayesh, & Thabtah, 2014). Due to the enormous loss and adverse effect cause by malware, malware detection is one of the cyber security topics that are of great interests.

To protect legitimate users from the attacks, the most significant line of defense against malware is anti-malware software products, which mainly use signature-based method for detection (Griffin, Schneider, Hu, & Chiueh, 2009; Kephart & Arnold, 1994). In these scanning tools, unique signatures (a set of short and unique strings) are extracted from already known malicious files. Then, an executable file is identified as a malicious code if its signature matches with the list of available signatures. Such simple approach is fast to identify known malware with small error rate. However, extracting signature is a tough work which requires a great deal of time, funds and more importantly, the expertise. This is the main disadvantage of this method. The second issue is that signature-based method is restricted to recognize already known malware, and thus it is unreliable and ineffective against the new, unseen malicious codes. In fact, simple obfuscation techniques can easily bypass such signatures-based detection. Besides, driven by the economic benefits, today's malware samples are created at a high speed (thousands per day). For example, Symantec reported that 21.7 million new pieces of malware were created in October 2015 (Symantec, 2015); according to McAfee Labs threat report, there were more than 400 million total malware samples in the first quarter of 2015 (McAfee Labs, 2015).

In order to solve the above-mentioned problems, heuristic-based detection method, which utilizes data mining as well as machine learning techniques, is developed to conduct intelligent malware detection. This approach aims to learn special patterns that capture the characteristics of malware. Generally, its detection process can be divided into two phases: feature extraction and classification. In the first phase, various features are extracted from malware samples via static analysis or dynamic analysis to

* Corresponding author at: School of Mathematics and Computer Science, Fujian Normal University, China. Tel.:+8659122868128.
E-mail addresses: kobefyj@126.com (Y. Fan), yanfang.ye@mail.wvu.edu (Y. Ye), clfei@fjnu.edu.cn (L. Chen).

http://dx.doi.org/10.1016/j.eswa.2016.01.002
0957-4174/

represent the file; based on the extracted features, classification techniques are applied to identify the malware automatically. For instance, Schultz et al. (2001) extracted three different types of features (i.e., system resource information, printable strings and byte sequences) from the files, then used as inputs for Ripper Naive Bayes and Multi-Naive Bayes to classify malware and benign files.

Since Application Programming Interface (API) calls can well represent the actions of an executable, it is one of the most effective features used by the heuristic-based methods. Many researches have been done based on API calls, including Hofmeyr, Forrest, and Somayaji (1998), Ye, Wang, Li, Ye, and Jiang (2008) and so forth. There are some other researchers applying another meaningful feature (i.e., the machine instructions) to detect malware, such as Santos et al. (2010), Shabtai, Moskovitch, Feher, Dolev, and Elovici (2012) and Runwal, Low, and Stamp (2012). Although these works demonstrate desirable detection results, they did not take the order of the features into consideration and thus fail to mine patterns with notable difference between malicious files and benign files.

In this paper, we propose a new sequence mining algorithm to discover malicious sequential patterns based on the machine instruction sequences extracted from the Windows Portable Executable (PE) files, then use it to construct a data mining framework, called MSPMD (short for **M**alicious **S**equential **P**attern based **M**alware **D**etection), to detect new malware samples. The main contributions of this paper can be summarized as follows:

- *Well represented feature for malware detection*: Instruction sequences are extracted from the PE (Portable Executable) files as the preliminary features, based on which the malicious sequential patterns are mined in the next step. The extracted instruction sequences can well indicate the potential malicious patterns at the micro level. In addition, such kind of features can be easily extracted and used to generate signatures for the traditional malware detection systems.
- *Effective malicious sequential pattern mining algorithm*: We propose an effective sequential pattern mining algorithm, called MSPE (**M**alicious **S**equential **P**attern **E**xtraction), to discover malicious sequential patterns from instruction sequence. MSPE introduces the concept of objective-oriented to learn patterns with strong abilities to distinguish malware from benign files. Moreover, we design a filtering criterion in MSPE to filter the redundant patterns in the mining process in order to reduce the costs of processing time and search space. This strategy greatly enhances the efficiency of our algorithm.
- *All-Nearest-Neighbor (ANN) classifier for malware detection*: We propose ANN classifier as detection module to identify malware. Different from the traditional *k*-nearest-neighbor method, ANN chooses *k* automatically during the algorithm process. More importantly, the ANN classifier is well-matched with the discovered sequential patterns, and is able to obtain better results than other classifiers in malware detection.
- *Comprehensive experimental studies*: We conduct a series of experiments to evaluate each part of our framework and the whole system based on real sample collection, containing both malicious and benign PE files. The results show that MSPMD is an effective and efficient solution in detecting new malware samples.

The remainder of this paper is organized as follows: Section 2 introduces the related work. In Section 3, an overview of MSPMD is presented. Section 4 describes the method for instruction sequence feature extraction. Section 5 presents the proposed algorithm for malicious sequential pattern mining. Section 6 describes the ANN classifier for malware prediction based on the mined malicious sequential patterns. Experimental results are presented in Section 7. Finally, Section 8 concludes.

2. Related work

Signature-based method is widely used in anti-malware industry for malware detection (Griffin et al., 2009). However, this classic method always fails to detect variants of known malware or previously unseen malware. The problem lies in the signature extraction and generation process, and in fact these signatures can be easily bypassed (Ye et al., 2008). For example, to evade the widely-used signature-based detection, malware developers can employ techniques such as polymorphism and metamorphism (Jain & Bajaj, 2014). Not only the diversity and sophistication of malware have significantly increased in recent years, driven by economic benefits, today's malware samples are also created at a rate of thousands per day (McAfee Labs, 2015; Symantec, 2015). In order to remain effective, anti-malware industry calls for intelligent malware systems which can automatically detect newly unseen malware from the collected file samples. Many research efforts have already been conducted on developing intelligent malware detection systems applying data mining techniques. Such data-mining-based detection methods require a feature extraction process to mine some features. Actually, the performance of the detection method mainly depends on what the features are extracted from the executables. More specifically, if the extracted features are well representative, it is expected to obtain better result when using these features to detect malware. Over the past few years, API calls and machine instructions are two of the most widely used features (Bazrafshan, Hashemi, Fard, & Hamzeh, 2013). Besides these, there also exists many researches relying on other features for malware detection, such as byte code (Nissim, Moskovitch, Rokach, & Elovici, 2014), data flow graph (Wchner, Ochoa, & Pretschner, 2014), Dynamic Link Libraries (DLLs) (Narouei, Ahmadi, Giacinto, Takabi, & Sami, 2015).

API calls represent the requests of windows executables on operate system. Due to their effectiveness to reflect the actions of executable, API calls are considered to be one of the most attractive features for detecting malware. The first attempt to use API as a feature of program was Hofmeyr et al. (1998). They presented a method for anomaly intrusion detection based on sequences of system calls. In their work, normal behavior was defined in short sequences of system calls executed by program. Then, three measures were used to detect abnormal behavior as deviations from the normal behavior. The representative research on API calls has been done by Ye et al. (2008). They developed an intelligence malware detection system (IMDS): it first extracted the API calls from each sample; then an objective-oriented association (OOA) mining algorithm was employed to generate OOA rules; finally it applied Classification Based on Association (CBA) (Bing, Wynne, & Ma, 1998) to build the classifier for malware detection. The experimental results showed that IMDS outperformed the signature-based methods and other data-mining-based methods in terms of detection rate and classification accuracy. Another interesting work using API calls for malware detection was Ahmadi, Sami, Rahimi, and Yadegari (2013), which was a dynamic malware detection system. They employed the iterative pattern mining method (Lo, Cheng, Han, Khoo, & Sun, 2009) to extract frequent iterative patterns and used Fisher score to conduct feature selection. The experiment results showed that high detection rate with low false alarm can be achieved when applying an iterative pattern mining approach. In very recent, Uppal, Sinha, Mehra, and Jain (2014) utilized the call grams and odds ratio selection to extract the distinct API sequences, then used as inputs to the classifiers to categorize malware and benign samples. Qiao, Yang, He, Tang, and Liu (2014) created a new representation method to transform API call sequences into byte-based sequential data for further

detection. Sundarkumar, Ravi, Nwogu, and Govindaraju (2015) presented an approach to detect malware, which used text mining and topic modeling for feature extraction and feature selection based on the API call sequences.

However, collecting API calls is typically a time-consuming and resource-consuming process, which requires a virtual machine or an emulator (Egele, Scholte, Kirda, & Kruegel, 2012) to analyze the code behaviors during the execution time. On the contrary, the machine instructions can be easily extracted and used to generate signatures for the traditional malware detection systems (Ye, Li, Chen, & Jiang, 2010). Moreover, the subdivision of a machine instruction (i.e., the opcode) implies the operation executed by the executable. These facts make the instructions become an effective feature for malware detection. Our work also focuses on using machine instructions as the preliminary feature for further analyze.

To detect the variants of known malware families, Santos et al. (2010) presented an approach using the frequency of appearance of opcode-sequences to build an information retrieval representation of executables. Shabtai et al. (2012) used the opcodes to detect unknown malicious codes. After extracting the opcode n-gram patterns, they calculated the normalized term frequency (TF) and TF Inverse Document Frequency (TF-IDF) for each opcode patterns in each file. Then, eight classical classification techniques were used to evaluate the proposed feature selection method. The technique presented in Runwal et al. (2012) used the similarity of executables based on opcode graphs for metamorphic malware detection. They extracted the opcode sequences from files and generated a weighted directed graph for each file. After that, a new executable can be predicted as malware or benign file by calculating the similarity of opcode graph obtained from the executable and both file types. Recently, many other techniques have been used for malware detection based on machine instructions. Rad, Masrom, and Ibrahim (2012) used a histogram of instruction opcode frequencies to detect metamorphic malware. They built a histogram for each file and compared against the already built histograms of malware samples to classify the file as malware or benign. Austin, Filiol, Josse, and Stamp (2013) built hidden Markov models (HMMs) for both benign and malware programs. For each program, the probability of the opcode sequence was determined for each of the HMMs. Then, the program was flagged as malware if the HMM with highest probability belonged to malware. Ahmadi, Giacinto, Ulyanov, Semenov, and Trofimov (2015) applied feature fusion technique to combine opcodes with other features as inputs for classifiers to detect malware.

Despite the favorable detection results obtained by the above mentioned works, few methods attempt to mine patterns with a strong ability to distinguish malware from benign files. In this paper, we propose an effective sequential pattern mining algorithm to discover discriminative malicious patterns on the extracted instruction sequences. Based on which, a data mining framework MSPMD is developed for detecting new malware.

3. System architecture

Fig. 1 shows the system architecture of the proposed malware detection framework MSPMD, which consists of three major components: instruction sequence extractor, malicious sequential pattern miner, and ANN (All-Nearest-Neighbor) classifier for malware prediction. We briefly describe each component below.

1. Instruction sequence extractor: MSPMD first extracts instructions from training samples and transforms them into a group of 32-bit global IDs based on their lexicographical order. Then, a subset of instructions is selected using the newly proposed algorithm MIE (Malicious Instruction Extraction), followed by the guiding match method used to generate instruction sequence for each training sample.
2. Malicious sequential pattern miner: In this component, MSPE (Malicious Sequential Pattern Extraction) algorithm is applied to mine discriminating malicious sequential patterns from instruction sequences.
3. ANN classifier: In this module, the input executables (including the training samples and the testing samples) are transformed into vectors based on the mined malicious sequential patterns. Then, the proposed classifier ANN is used to conduct malware prediction.

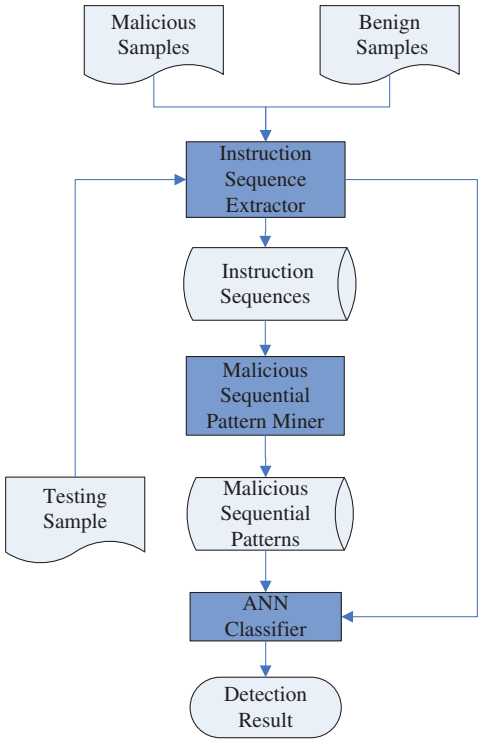

Fig. 1. System architecture of the proposed malware detection framework.

The detail processes and the new methods proposed for the three components will be presented in the following three sections, respectively.

4. Instruction sequence feature extraction

In the first step of MSPMD, each PE file will be transformed into an instruction sequence. These instructions are carefully chosen in order to distinguish malware from benign samples as much as possible; therefore, they can be viewed as the low-level (instruction-level) features representing the executables. In this section, we describe the method used to extract such features from the training sample set, which is implemented in two sub-steps.

4.1. Instruction sequence feature representation

The first sub-step is designed to represent each PE file in a long symbol sequence, where each symbol corresponds to a machine instruction appearing in the executable. This is achieved by disassembling the PE files followed by parsing the operation codes of each instruction, as follows:

Disassembling: A third party disassembler C32Asm (2011) is used to disassemble each sample, creating an assembly representation for the sample. Fig. 2 shows an example, which is a fragment of the disassembly for the Worm PE file named

```
MOV EAX,10011F5C
CALL 10011BF0
SUB ESP,17C
MOV DWORD PTR [EBP-148],ECX
MOV AL,BYTE PTR [EBP-124]
MOV ECX,DWORD PTR [EBP-148]
MOV BYTE PTR [ECX],AL
MOV AL,BYTE PTR [EBP-128]
MOV ECX,DWORD PTR [EBP-148]
MOV BYTE PTR [ECX+1],AL
MOV EAX,DWORD PTR [EBP-148]
AND BYTE PTR [EAX+8],0
MOV ECX,DWORD PTR [EBP-148]
CALL 100037D7
AND DWORD PTR [EBP-4],0
```

Fig. 2. A fragment of the output of disassembled Worm.Win32.AutoRun.aaeu.

Worm.Win32.AutoRun.aaeu. Each line of the assembly corresponds to a machine instruction, composed of an operator and the associated operand. For example, the operator of the first instruction in Fig. 2 is MOV with the CPU register EAX and the hexadecimal number 10011F5C being its operands. Note that for Windows PE files, the number of operators is finite, and for the same operator its operands may vary in different instructions.

Parsing: Based on the assembly instructions generated in the disassembling step, a compact representation is constructed for the samples, making use of the operators but ignoring the operands. This is due to the fact that it is the operator that indicates the behavior (the operation) of an instruction. Moreover, in typical obfuscated malicious codes (Zhang, Chen, & Guo, 2012), the machine instructions may change across different malware variants; however, their operators usually remain the same. For the purpose, we have developed a parser in JAVA to translate the assembly instructions, by discarding the operands and encoding each operator with a unique number (say instruction ID). Fig. 3 gives an example, where 6 malwares (denoted by M) and 4 benign samples (denoted by B) are represented in instruction sequence. For example, the Worm.Win32.AutoRun.aaeu shown in Fig. 2 now is represented in $240 \rightarrow 33 \rightarrow 386 \rightarrow 240 \rightarrow \cdots$, with 240, 33 and 386 being the IDs of MOV, CALL and SUB, respectively. Obviously, the sequences are in variable-length, and the sequence length is dependent on the size of the corresponding PE file.

4.2. Feature selection

In the second sub-step, we propose the MIE method for feature selection in order to reduce the useless information caused by undiscriminating instructions. Since the selected instructions are highly frequent that incline to malicious executables, we introduce the concept "tendency" to measure the extent of an instruction to be malicious.

Definition 1 (tendency)**.** Letting i be an instruction ID, its tendency is defined as:

$$tendency(i) = \begin{cases} \dfrac{f_M(i)}{f_M(i) + f_B(i)}, & f_M(i) \neq 0 \\ 0, & f_M(i) = 0 \end{cases}$$

where $f_M(i)$ and $f_B(i)$ stand for the weighted frequency of the instruction in the malicious and benign samples, respectively.

Intuitively, the frequency of an instruction is similar to that of a keyword in a document collection. Inspired by the term weighting techniques developed in the text mining community (Soucy & Mineau, 2005), we assign each instruction a class-dependent weight according to its coverage in the class (Malware or Benign). Therefore, an instruction will receive a high weight if it widely distributes across the malicious or benign samples. Formally, we calculate the weights for the ith instruction with regard to the malicious and the benign category by

$$w_M(i) = \frac{|N_M(i)|}{|N_M|},$$

$$w_B(i) = \frac{|N_B(i)|}{|N_B|},$$

with $|N_M(i)|$ and $|N_B(i)|$ being the number of malicious and benign samples involving the ith instruction, respectively; $|N_M|$ and $|N_B|$ are the total number of malicious and benign samples. Furthermore, the weighted frequencies of the instruction are formulated as follows:

$$f_M(i) = w_M(i) \times \frac{|U_M(i)|}{|U_M|},$$

$$f_B(i) = w_B(i) \times \frac{|U_B(i)|}{|U_B|},$$

where $|U_M(i)|$ and $|U_B(i)|$ denote the number of times instruction i appearing in the entire malicious and benign samples, $|U_M|$ and $|U_B|$ are the total number of the instructions in the malicious and benign samples.

Based on the definition, the tendency of each instruction can be computed. An instruction i is selected only if $tendency(i) > t$, where t is a user-specified threshold. Then, all selected features are collected to produce variable-length instruction sequences for each sample using the simple guiding match method (Zhang et al., 2012). We can see that each resulting sequence is composed of ordered instructions that have significant tendency to be malicious codes, thus they are able to indicate the potential malicious pattern at the micro level.

5. Malicious sequential pattern mining

In this section, we describe the MSPE algorithm for malicious sequential pattern mining. MSPE aims at discovering the discriminative malicious sequential patterns, which can be viewed as macro-level features to represent the executables.

```
Worm.Win32.AutoRun.aaeu     M     240 33 386 240 240 240 240 240 240 240 ;
Worm.Win32.AutoRun.aam      M     73 11 389 11 291 185 11 33 11 77 185 11
Worm.Win32.AutoRun.aap      M     189 11 11 240 11 11 11 199 11 64 11 11 .
Worm.Win32.AutoRun.aarf     M     33 214 379 16 204 291 11 64 399 230 381
Worm.Win32.AutoRun.aarj     M     11 11 329 11 400 11 210 11 240 11 16 11
Worm.Win32.AutoRun.abk      M     11 185 11 327 327 199 183 185 11 185 11
a2p.exe                     B     327 327 327 240 327 327 240 240 33 327 ;
abcwin.exe                  B     327 240 386 240 327 240 327 327 217 327
ab.exe                      B     386 240 240 240 327 327 327 240 240 400
Accel.exe                   B     240 400 240 240 240 240 240 343 327 240
```

Fig. 3. An example of the sequence representation for Windows PE files.

5.1. Notation and basic definitions

Before mining malicious sequential patterns, we first introduce the related definitions of instruction sequence as follows: let $I = \{I_1, I_2, \ldots, I_m\}$ be the set of instruction items, and m the number of items. An instruction sequence s is an ordered list of the items and is denoted by $s_1 s_2 \ldots s_l$ where each $s_j (1 \le j \le l) \in I$.

Definition 2 (subsequence)**.** A sequence $\alpha = a_1 a_2 \ldots a_n$ is called a subsequence of another sequence $\beta = b_1 b_2 \ldots b_m$, denoted as $\alpha \sqsubseteq \beta$, if there exists integers $1 \le j_1 < j_2 < \cdots < j_n \le m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$.

Definition 3 (support and confident)**.** Letting α be a subsequence of the sequence in S_M or S_B, the support and confidence of α defined as:

$$sup_\alpha\% = \frac{|\{\beta | (\beta \in S_M) \wedge (\alpha \sqsubseteq \beta)\}|}{|S_M|} \times 100\%, \quad (1)$$

$$conf_\alpha\% = \frac{|\{\beta | (\beta \in S_M) \wedge (\alpha \sqsubseteq \beta)\}|}{\sum_{t \in \{M,B\}} |\{s | (\beta \in S_t) \wedge (\alpha \sqsubseteq \beta)\}|} \times 100\%, \quad (2)$$

where $|S_M|$ and $|S_B|$ denote the number of sequences in malicious executables and benign executables set (recall that each executable is represented as an instruction sequence).

Definition 4 (sequential pattern)**.** Let $ms\%$ be a user-specified minimum support. A subsequence α is called a sequential pattern with regard to S_M if $sup_\alpha\% \ge ms\%$.

Definition 5 (malicious sequential pattern)**.** Let $mc\%$ be user-specified minimum confidence. A sequential pattern α is called a malicious sequential pattern if $conf_\alpha\% \ge mc\%$.

5.2. MSPE algorithm

In general, Generalized Sequential Pattern (GSP) algorithm (Srikant & Agrawal, 1996) is a simple and effective method to mine sequential patterns. However, when the minimum support decreases, GSP generates a huge number of candidates, which is time-consuming and resource-consuming. Additionally, when applying GSP to our case directly, it tends to search for the common 常见的 sequential patterns in both malware and benign samples, that is, it is unable to discover the discriminative 有识别力的 sequential patterns that have a strong ability to distinguish malware from benign executables. Therefore, in our work, we extend a modified 改进的 GSP algorithm to mine malicious sequential patterns. This algorithm addresses the above-mentioned shortcomings, and we call it MSPE algorithm.

Similar to GSP algorithm, MSPE algorithm is also an Apriori-like method. But the type of generated patterns and the filtering criterion used to generate them are different from GSP algorithm in the following ways: (1) we introduce the concept of objective-oriented (Shen, Zhang, & Yang, 2002) into MSPE to discover sequential patterns with malicious nature; (2) we also use a kind of "confidence" to filter the sequential patterns such that the costs of processing time and search space will decrease sharply. MSPE contains seven major steps and works as follows:

Step 1. Scans S_M and compute the support and confidence for each item using Eqs. (1) and (2), to generate length-1 sequential patterns, denote as L_1, according to Definition 4.
Step 2. Set the length of pattern $n = 2$.
Step 3. Generate new set of candidates C_n by self-join and prune operation of the sequential patterns found in the $(n-1)$th pass:
  1. Self-join operation: Join L_{n-1} with itself to generate C_n based on the following criterion: l_1 and l_2 are sequential patterns in L_{n-1}, if l_1 with removal of the first item equals to l_2 with removal of the last item, we join l_2 to l_1, by adding the last item of l_2 to l_1.
  2. Prune operation: Remove candidate from C_n if one of its length-$(n-1)$ subsequence is not a sequential pattern found at L_{n-1}.
Step 4. Scan C_n and collect the support and confidence for each $c \in C_n$ to find the new set of sequential patterns L_n according to Definition 4 and Eq. (3). In Eq. (3), c' are all length-$(n-1)$ subsequences of $c \in C_n$.

$$conf_c\% \ge conf_{c'}\% \quad (3)$$

Step 5. $n = n + 1$.
Step 6. Repeat Steps 3–5 until no sequential pattern is found in a pass, or no candidate sequence is generated.
Step 7. Collect malicious sequential patterns from the resulting sequential patterns based on Definition 5.

In our detection framework, the objective is to find out which samples belong to malware, thus the MSPE algorithm is proposed to determine which sequential patterns support this specific objective. This is the reason why MSPE is called of objective-oriented. It is necessary to remark that unlike the existing works, such as Rad et al. (2012) and Ahmadi et al. (2015) which use instruction solely, 唯一地 MSPE takes the order of the instructions into consideration. This also differs from the work in Ye et al. (2008) where the desired itemset patterns were mined based on the unordered Windows API calls. Moreover, since MSPE is objective-oriented, the generated sequential patterns are able to reflect malicious behaviors of malware, and are more discriminative than the iterative patterns in Ahmadi et al. (2013) and the n-gram patterns in Shabtai et al. (2012). In addition, in Step 4, we consider Eq. (3) as a filtering criterion and the minimum support to reduce the number of candidates. More specifically, in Eq. (3), the confidence of length-n sequential pattern must greater than or equal to that of its length-$(n-1)$ subsequence, this is because in our case, the longer the length is, the more discriminative the pattern becomes. In other words, the sequential patterns generated in each iteration should enhance the capacity of malware prediction when comparing with the patterns generated in the last iteration, i.e. $p(M|I) \ge p(M|I')$, where I' is the subsequence of I. Using such new strategy, the cost of running time and memory space can be significantly reduced during the mining process. This makes our algorithm more efficient than the well-known GSP algorithm.

Table 1
Sample database S_M.

ID	Instruction sequence	File type
1	$I_2 \to I_3 \to I_4$	M
2	$I_1 \to I_4 \to I_1 \to I_2$	M
3	$I_1 \to I_2 \to I_1$	M
4	$I_2 \to I_3$	M
5	$I_4 \to I_1 \to I_2$	M

Table 2
Sample database S_B.

ID	Instruction sequence	File type
1	$I_4 \to I_3 \to I_3$	B
2	$I_4 \to I_2$	B
3	$I_1 \to I_2 \to I_1$	B

5.3. Illustrating examples 说明例子

To explain the MSPE algorithm, we illustrate an example using the data shown in Tables 1 and 2, where each row contains three fields: file ID, instruction sequence and file type. Letting

$ms\% = 40\%$, by applying MSPE algorithm the sequential patterns can be obtained as: $< I_1 >$, $< I_2 >$, $< I_3 >$, $< I_4 >$, $< I_1 \rightarrow I_2 >$, $< I_2 \rightarrow I_3 >$, $< I_4 \rightarrow I_1 >$, $< I_4 \rightarrow I_1 \rightarrow I_2 >$. Note that, although the support of pattern $< I_1 \rightarrow I_1 >$ and $< I_4 \rightarrow I_2 >$ meet the condition in Definition 4. However, they still cannot be regarded as sequential pattern, since Eq. (3) is not satisfied. Take $< I_1 \rightarrow I_1 >$ as an example, its confidence 66.7% is less than 75%-the confidence of its subsequence $< I_1 >$. Then, given $mc\% = 80\%$, these sequential patterns are used to mine malicious sequential patterns, and the results are given as:

1. $< I_2 \rightarrow I_3 > \Rightarrow M(40\%, 100\%)$
2. $< I_4 \rightarrow I_1 > \Rightarrow M(40\%, 100\%)$
3. $< I_4 \rightarrow I_1 \rightarrow I_2 > \Rightarrow M(40\%, 100\%)$.

Examining the instruction sequences in Tables 1 and 2, one can see that these three malicious sequential patterns reveal the malicious behaviors hidden in the malware samples set S_M.

In order to demonstrate the effectiveness of the malicious sequential patterns, we show a real example generated by MSPE on the real-world data collection (see Section 7.1 for details). One of the malicious sequential patterns we generated with the condition $t = 0.90$ is:

$$< 182 \rightarrow 351 \rightarrow 351 \rightarrow 184 \rightarrow 184 \rightarrow 184 \rightarrow 184 > \Rightarrow M(sup\% = 93.00\%, conf\% = 97.13\%),$$

where *sup*% and *conf*% denote the support and confidence of this pattern, respectively. The sequence can be rewritten as

$$< idiv \rightarrow scas \rightarrow scas \rightarrow in \rightarrow in \rightarrow in \rightarrow in > \Rightarrow M(sup\% = 93.00\%, conf\% = 97.13\%),$$

by converting the IDs to the corresponding machine instructions.

By analyzing the value of *sup*% and *conf*%, we know that this malicious sequential pattern appears in 1116 malware, while only in 33 benign files. There is a clear difference between malicious and benign executables with regard to this pattern, as it appears in the overwhelm majority of malware but just in few benign executables. It is one of the underlying patterns for determining whether a sample is malware or not.

6. ANN classifier for malware prediction

In this section, we propose ANN classifier for malware detection based on the mined malicious sequential patterns. Different from the traditional *k*-nearest-neighbor method (Han, Kamber, & Pei, 2006), ANN chooses k automatically during the algorithm process.

6.1. Feature representation for testing sample set

Given a new PE sample, before prediction, it will first be transformed into a Boolean vector, where each element indicates the presence of the corresponding sequential pattern. Formally, let $V[x] = < x_1, x_2, \ldots, x_d >$ be a sample described by d numeric attributes, where each $x_j \in \{0, 1\}$, $(j = 1, \ldots, d)$. For intuitive understanding, we present an example in the following, as Table 3 shows.

In Table 3, 10 samples (5 malwares and 5 benign files) are considered, with 5 malicious sequential patterns (p_1 to p_5):

$p_1 :< idiv \rightarrow scas \rightarrow scas \rightarrow in \rightarrow in \rightarrow in \rightarrow in >$
$p_2 :< idiv \rightarrow xchg \rightarrow xchg \rightarrow scas \rightarrow scas \rightarrow in >$
$p_3 :< idiv \rightarrow scas \rightarrow scas \rightarrow in \rightarrow in \rightarrow in >$
$p_4 :< idiv \rightarrow xchg \rightarrow scas \rightarrow scas \rightarrow in >$
$p_5 :< std \rightarrow xchg \rightarrow scas \rightarrow scas \rightarrow in \rightarrow a >$.

Table 3
An example of the feature representation for testing sample set.

File name	Malicous sequential pattern				
	p_1	p_2	p_3	p_4	p_5
Worm.Win32.AutoRun.aap	1	1	1	1	1
Worm.Win32.AutoRun.bhr	1	0	1	0	0
Worm.Win32.AutoRun.byt	1	1	1	1	0
Worm.Win32.AutoRun.cob	0	1	1	1	0
Worm.Win32.AutoRun.zwz	1	1	1	1	1
1KG_su.exe	0	0	0	0	0
360rpt.exe	0	0	0	0	0
ctfmon.exe	0	0	0	0	1
eclipsec.exe	0	1	0	1	0
zoomin.exe	0	0	1	0	0

From Table 3, we can see that the first row of the table is the Boolean vector of a sample named Worm.Win32.AutoRun.aap, which is an Internet worm that contains all of the five malicious sequential patterns, whereas the sixth row shows that none of these patterns belong to the benign sample 1KG_su.exe.

6.2. Malware prediction

After the feature representation, we can easily measure the similarity of different samples according to their containing malicious sequential patterns. Here, similarity is measured by Euclidean distance.

The traditional *k*-nearest-neighbor (*k*NN) (Guo, Wang, Bell, Bi, & Greer, 2003; Han et al., 2006; Zeng, Yang, & Zhao, 2009) is a non-parametric classification method, which is simple but effective in many cases. It first searches for k training samples that are closest to the testing sample. These k training samples are the k "nearest neighbors" of the testing sample. Then, the testing sample is assigned the most common class among its k-nearest neighbors. However, in a sense, the *k*NN method is biased by k, that is, the success of classification is very much dependent on this value.

The proposed detection module ANN is based on *k*NN, but overcomes the issue of "k" inherited in the traditional *k*NN method. It contains three major steps and is outlined in the following.

Step 1. Calculate the Euclidean distance between testing sample y and each training samples t according to $dist(y, t) = ||V[y] - V[t]||_2$.
Step 2. Use $t_s = \text{argmin}_t\, dist(y, t)$ to select training sample t_s whose distance is shortest to y.
Step 3. Assign y to the class (malicious or benign) among t_s using majority vote.

Obviously, the proposed ANN classifier does not need to choose a specific k for final classification: the number of selected training samples (i.e., $|t_s|$) can be seen as an optimal k, which means the k is generated automatically during the algorithm. A real example is illustrated to better understand the difference between traditional *k*NN and ANN classifier. Consider the malicious sample named Worm.Win32.AutoRun.dmv as a testing sample, if we apply *k*NN classifier to recognize the testing sample, different k will generate different classification result, that is when $k = 1$, *k*NN classify it to malware while $k = 9$ it is classified to benign file. However, if we regard ANN classifier as detection module, 997 training samples whose distance to testing sample is shortest are selected, in which 970 training samples belong to malware and the remaining are benign executables. Finally, the testing sample is assigned to malware according to majority voting. Using ANN classifier, the similarity between different samples can be easily computed and the testing sample could be recognized correctly.

Table 4
Coverage of malicious instruction sequence on different t.

t	$cov(M)$	$cov(B)$
0.80	100%	95.13%
0.85	100%	89.75%
0.90	100%	87.50%
0.95	99.25%	79.25%

Table 5
The number of patterns on different $ms\%$ and $mc\%$.

$ms\%$	$mc\%$	Number of patterns
94%	96%	0
	95%	75
93%	96%	659
	95%	2753
92%	96%	9161
	95%	19,815

Table 6
Running time of different sequential pattern mining algorithms (min).

Experiment	1	2	3	4	5
$ms\%$	94%	93%	92%	91%	90%
MIE+GSP	1.85	3.97	19.55	185.9	2368.6
MIE+MSPE	1.77	3.81	16.06	80.39	370.72

7. Experimental results and analysis

In this section, we evaluate each part of our framework and the whole detection system MSPMD through a series of experiment with comparing to a few existing methods. All the experimental studies are conducted under the environment of Windows XP operating system plus Intel T6600 2.20 GHz CPU and 2GB of RAM.

7.1. Data description

Our system is directly suitable for Windows PE file, as PE malware occupy the majority of today's malicious codes. We collect 10,307 Windows PE samples, which consist of 8847 malicious instances and 1460 benign instances. There are no duplicate samples in our dataset. Malware are downloaded from http://vxheaven.org/, while the benign programs are system files coming from a newly installed Windows XP system. However, if a PE file is previously compressed or encrypted by a compress tool such as ASPack and PECompact, we first use unpack tools to decompress the PE code. In each experiment, we sample 2000 records from our dataset, which includes 1200 records of malicious executables and 800 records of benign executables.

7.2. Parameter selection and evaluate criteria

Currently, the principal method to conduct parameter selection is based on experiment results. However, this method is only suitable for specific dataset to some extent, and it may not be generalizable. In our work, we analyze the object influenced by parameter directly to determine the best choice, which reduces the dependency of parameter on dataset.

Different t's correspond to different malicious instructions, which lead to generate different length of malicious instruction sequence for each executable. As malicious instructions indicating the potential malicious patterns at the micro level, the best t should let malicious instruction sequence have full coverage in malicious codes and low coverage in benign codes. Thus, we use $cov(M)$ and $cov(B)$ to denote the coverage of these sequences in malicious and benign codes, respectively, i.e.,

$$cov(M) = \frac{|S_M|}{|N_M|},$$

$$cov(B) = \frac{|S_B|}{|N_B|}.$$

As shown in Table 4, when choose $t = 0.90$, all malware but only 700 benign executables in dataset can be represent as malicious instruction sequence (other 100 benign executables are transformed into empty sequences). This indicates $t = 0.90$ is the best choice.

For $ms\%$ and $mc\%$, malicious sequential pattern with high support and confidence indicates it exists in most malicious codes but appears in few benign codes. It is to say $ms\%$ and $mc\%$ must be set as high as possible in case of there are enough sequential patterns to make sure malicious sequential patterns can distinguish malware from benign executables as much as possible. From Table 5, as $ms\%$ and $mc\%$ decrease, the number of patterns increases. When choose $ms\% = 94\%$, the number of generated malicious sequential patterns is too less to express malicious instruction sequences whatever the value of $mc\%$. Therefore, we set $ms\%$ to 93% and $mc\%$ to 96%, as 659 malicious sequential patterns are just enough.

To evaluate MSPMD, the standard tenfold cross validation is used in the experiments: the original dataset is randomly divided into 10 equal size subsets, where a single subset is retained as testing data, and the remaining 9 subsets are used as training data. This process is repeated 10 times, make sure that each subset used only once as testing data. The 10 results then are averaged to generate estimation. Moreover, the following evaluate measures are used in the results:

- *True positive (TP)*: the number of malicious executables correctly classified
- *True negative (TN)*: the number of benign executables correctly classified
- *False positive (FP)*: the number of benign executables classified as malicious code
- *False negative (FN)*: the number of malicious executables classified as benign code
- *Detection rate (DR)*: $\frac{TP}{TP+FN}$
- *False positive rate (FPR)*: $\frac{FP}{FP+TN}$
- *Accuracy (ACC)*: $\frac{TP+TN}{TP+TN+FP+FN}$

7.3. Evaluation of malicious sequential pattern mining process

The first set of experiments is to evaluate the feature extraction phase in our framework, i.e., the process of mining malicious sequential patterns. We conduct two experiments in this subsection, that is, examining the effectiveness of the proposed sequential pattern mining algorithm MSPE and the mined malicious sequential patterns through the comparison with other methods.

7.3.1. Evaluation of MSPE

We implement MIE, GSP, and MSPE algorithms under Java Development Kit environment. By using different support thresholds, we compare the efficiency of the two sequential pattern mining algorithms. The results are shown in Table 6, where we observe that the running time increases sharply as the minimum support threshold decreases. However, it shows obviously that the MSPE algorithm get much less time with each threshold and it even get 7 times faster than GSP algorithm when set $ms\%$ to 90%. It is also important to say that an Out Of Memory Error will arises if we

Table 7
The comparison of expression ability of different kinds of features.

Feature	Algorithm	Classifier	DR (%)	FPR (%)	ACY (%)
Instruction feature	IG	NB	83.58	4.88	88.20
		SVM	94.33	6.75	93.90
		J4.8	95.00	7.25	94.10
	MR	NB	83.58	4.88	88.20
		SVM	94.25	6.75	93.85
		J4.8	95.67	7.25	94.50
	Chi-square	NB	83.58	4.88	88.20
		SVM	94.25	6.75	93.85
		J4.8	95.67	7.25	94.50
Malicious sequential pattern	MSPE	NB	92.75	4.38	93.90
		SVM	94.75	5.75	94.55
		J4.8	95.67	6.25	94.90

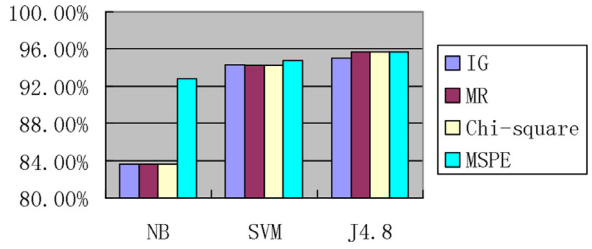

Fig. 4. Detection rate performance of different kinds of features.

use GSP or MSPE to mine sequential patterns directly instead of applying MIE to preprocess instruction first.

Experiment results indicate that MIE is a requisite step in our framework to select a small amount of instruction features which are more inclined to malware, as it reduces the useless information caused by undiscriminating instructions. More importantly, our proposed MSPE algorithm performs much more efficient than traditional sequence mining algorithm. In general, the running time of a sequence mining algorithm mainly depends on the process of seeking patterns that meet some constraints, we improve this in MSPE by using a kind of filtering criterion to reduce the searching space in each iteration. As a result, this strategy greatly enhances the efficiency of MSPE.

7.3.2. Evaluation of malicious sequential pattern

The expression ability of features measures their capability to represent executable. Therefore, in order to evaluate the malicious sequential patterns, we examine their expression ability in comparison with some instruction features among different classifiers. In contrast, we choose three common used algorithms: information gain (IG), max-relevance (MR) and chi-square test (Yang & Pedersen, 1997) to conduct instruction feature selection.

First, we rank each instruction using these three algorithms, and then choose top 100 instructions as the instruction features for classification. For malicious sequential patterns, we use MSPE algorithm to select 10 highest confidence features with the limitation of $sup\% \geq ms\%$ and $conf\% \geq mc\%$. Finally, we apply Naive Bayes (NB), SVM and J4.8 version of Decision Tree these three different classifiers to examine the expression ability of each kind of feature. The results are shown in Table 7, Fig. 4 and Fig. 5. From Table 7, we observe that when using the same classifier, the malicious sequential patterns outperform instruction features in detection rate, false positive rate and accuracy. Particularly on Naive Bayes classifier, they improve detection rate by almost 9% and accuracy by 5.7%. Figs. 4 and 5 present a clearer graphical view of detection rate and accuracy of different features.

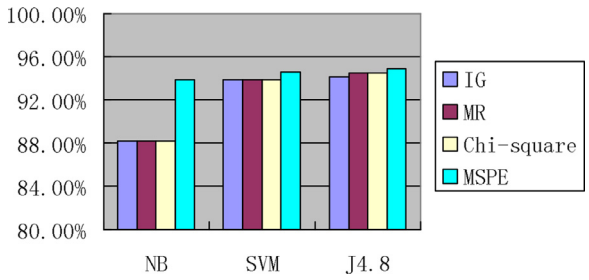

Fig. 5. Accuracy performance of different kinds of features.

Table 8
The comparison of detection results of different classifiers.

Feature	Classifier	DR (%)	FPR (%)	ACY (%)
Malicious sequential pattern	*k*NN	95.18	5.75	94.81
	NB	92.75	4.38	93.90
	SVM	94.75	5.75	94.55
	J4.8	95.67	6.25	94.90
	ANN	96.17	6.13	95.25

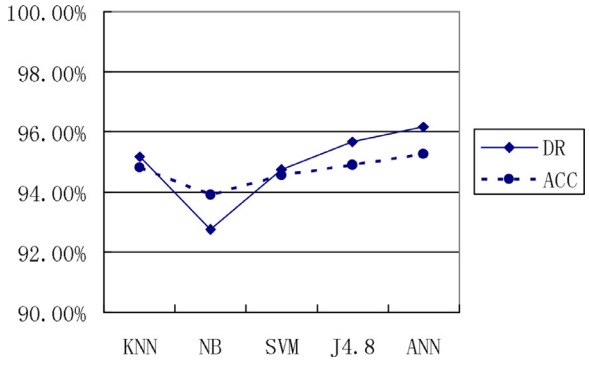

Fig. 6. Detection results of different classifiers.

The good performance achieved by malicious sequential patterns owes to their strong ability to represent malicious executables. As discussed previously, malicious sequential patterns are generated by MSPE algorithm which integrates the concept of objective-oriented. In our case, the objective is to detect malware, thus the MSPE algorithm is tend to find patterns to support this specific objective. Different from other instruction features used in the experiment above, these discriminative patterns capture the notable difference between malware and benign executables and are essential for malware detection whatever the classifiers.

7.4. Evaluation of All-Nearest-Neighbor (ANN) classifier

In the second set of experiments, we consider malicious sequential patterns as classification features to evaluate the proposed ANN classifier in comparison with other common used classification methods, including the classifiers introduced in Section 7.3.2 and *k*NN classifier.

As shown from Table 8, all classification methods take malicious sequential patterns as input and output the detection result. Note that the result of *k*NN in Table 8 is the average accuracies along with the number of neighbors *k* varying from 1 to 9. We can see that ANN outperforms other classifiers in both detection rate and accuracy. Fig. 6 gives a graphic illustration of the detection results of different classifiers.

To further examine the suitability of ANN to malicious sequential patterns, we select different numbers of malicious sequential patterns according to the descending order of the patterns'

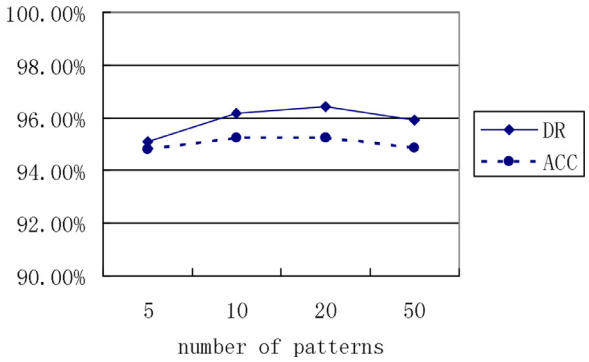

Fig. 7. The comparison of detection rate and accuracy with different number of malicious sequential patterns.

Table 9
The comparison of malware categorization results of different detection systems.

Detection system	TP	TN	FP	FN	DR (%)	FPR (%)	ACY (%)
IMDS	1147	721	79	53	95.58	9.88	93.40
MSPMD	1154	751	49	46	96.17	6.13	95.25

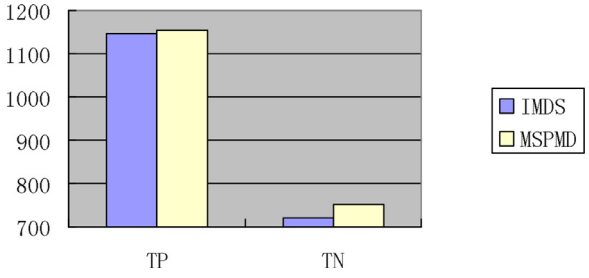

Fig. 8. True positives and true negatives of different malware detection systems.

confidence as inputs to ANN. As shown from Fig. 7, we can see that with different number of patterns, all curves in the figure are stable and both DR and ACC still stay more than 94 percentages.

The better experiment results obtained by ANN demonstrate that the proposed ANN classifier is much more suitable for malicious sequential patterns than other classifiers. This attribute to the success of transforming each executable into a Boolean vector as this representation fits well with the ANN classifier. Moreover, as a distance-based classifier, ANN not only obtains better results than another distance-based classifier *k*NN, but overcomes the issue of "*k*" inherited in the traditional *k*NN method.

7.5. Comparison with other malware detection systems

In the third set of experiments, we compare our MSPMD with IMDS (Ye et al., 2008) which has already been successful used for malware detection to demonstrate the effectiveness of our framework. In IMDS, OOA mining algorithm was applied for frequent patterns mining and then CBA classifier is built for malware detection based on the generated rules. For OOA mining, due to the fact that the number of frequent patterns is much smaller than that of malicious sequential patterns, it is unable to generate frequent patterns satisfied with $sup\% \geq 93\%$ and $conf\% \geq 96\%$, thus we decrease both $ms\%$ and $mc\%$ to 90% and 95%, respectively. To ensure the fairness of the tests, we also select 10 highest confidence patterns with the limitation of $sup\% \geq ms\%$ and $conf\% \geq mc\%$.

Results shown in Table 9 indicate that our MSPMD achieves better results in DR, FPR and ACC when compare with OOA mining and classifier construction method in IMDS, especially for FPR. Figs. 8 and 9 present a clearer graphical view of the results.

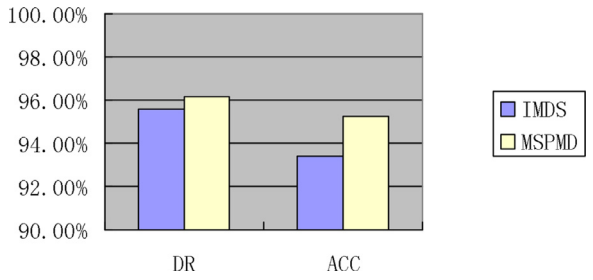

Fig. 9. Detection rate and accuracy of different malware detection systems.

By analyzing, it is the use of sequence mining technique in our framework result in the good performance of MSPMD. This differs greatly from the OOA mining algorithm in Ye et al. (2008), which generated unordered patterns for detection. In conclusion, the MSPE algorithm used in feature extraction phase and the ANN classifier for predicting malware together make our MSPMD become an effectiveness and efficiency solution for malware detection.

8. Conclusion and future work

In this paper, we develop a data-mining-based detection framework called Malicious Sequential Pattern based Malware Detection (MSPMD), which is composed of the proposed sequential pattern mining algorithm (MSPE) and All-Nearest-Neighbor (ANN) classifier. It first extracts instruction sequences from the PE file samples and conducts feature selection before mining; then MSPE is applied to generate malicious sequential patterns. For the testing file samples, after feature representation, ANN classifier is constructed for malware detection. The promising experimental results on real data collection demonstrate that our framework outperforms other alternate data mining based detection methods in identifying new malicious executables.

Unlike the previous researches which are unable to mine discriminative features, we propose to use sequence mining algorithm on instruction sequence to extract well representative features. These features capture the significant difference between malicious files and benign files. Additionally, our proposed algorithm is much more efficient than traditional sequential pattern mining algorithm due to the use of a designed filtering criterion. We also construct a new nearest neighbor classifier as detection module. This specially designed classifier is more suitable than the classic classifiers based on the mined malicious sequential patterns.

Since the framework proposed in this work only focus on malware detection, i.e. whether a sample is malware or not, it is unable to provide malware classification which requires a prediction of the exact types of malware. This weakness would restrain the method from being applied to more extensive applications. For instance, in the field of malicious code analysis, malware detection may not work well in such application as its main task is to classify malware into different groups and analyze the common behaviors in the same category. Therefore, our future efforts will be to extend our framework to predict different types of malware. Another weakness of our method inherits from the traditional *k*NN method, i.e., the lack of an explicit model. Although the proposed ANN classifier overcomes the issue of "*k*", it is still a lazy learning classifier as no model needs to be built, which requires a high cost in classifying new instances. This leads us to continue working on the framework in the future, by combining some strategies such as data reduction in order to enhance the classification efficiency.

Acknowledgments

L. Chen's work was supported by the National Natural Science Foundation of China under Grant no. 61175123, and the Natural Science Foundation of Fujian Province of China under Grant no. 2015J01238.

References

Abdelhamid, N., Ayesh, A., & Thabtah, F. (2014). Phishing detection based associative classification data mining. *Expert Systems with Applications, 41*, 5948–5959.

Ahmadi, M., Giacinto, G., Ulyanov, D., Semenov, S. Trofimov, M. (2015). Novel feature extraction, selection and fusion for effective malware family classification. arXiv: http://arxiv.org/abs/1511.04317.

Ahmadi, M., Sami, A., Rahimi, H., & Yadegari, B. (2013). Malware detection by behavioural sequential patterns. *Computer Fraud & Security, 2013*, 11–19.

Austin, T. H., Filiol, E., Josse, S., & Stamp, M. (2013). Exploring hidden markov models for virus analysis: a semantic approach. In *Proceedings of 46th hawaii international conference on system sciences* (pp. 5039–5048).

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Proceedings of the 5th conference on information and knowledge technology* (pp. 113–120).

Bing, L., Wynne, H., & Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the 4th international conference on knowledge discovery and data mining*.

C32Asm (2011). https://tuts4you.com/download.php?view.1130. Accessed 22.06.14.

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *Computing Surveys, 44*, 6.

Griffin, K., Schneider, S., Hu, X., & Chiueh, T. C. (2009). Automatic generation of string signatures for malware detection. In *Proceedings of the 12th international symposium on recent advances in intrusion detection* (pp. 101–120).

Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003). KNN model-based approach in classification, *Volume 2888 of Lecture notes in computer science* (pp. 986–996). Springer.

Han, J., Kamber, M., & Pei, J. (2006). *Data mining: Concepts and techniques*. Morgan Kaufmann.

Hofmeyr, S. A., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security, 6*, 151–180.

Jain, M., & Bajaj, P. (2014). Techniques in detection and analyzing malware executables: A review. *International Journal of Computer Science and Mobile Computing, 3*, 930–933.

Kephart, J. O., & Arnold, W. C. (1994). Automatic extraction of computer virus signatures. In *Proceedings of 4th virus bulletin international conference* (pp. 178–184).

Lo, D., Cheng, H., Han, J., Khoo, S., & Sun, C. (2009). Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *Proceedings of the 15th international conference on knowledge discovery and data mining* (pp. 557–566).

Narouei, M., Ahmadi, M., Giacinto, G., Takabi, H., & Sami, A. (2015). DLLMiner: Structural mining for malware detection. *Security and Communication Networks, 8*, 3311–3322.

Nissim, N., Moskovitch, R., Rokach, L., & Elovici, Y. (2014). Novel active learning methods for enhanced PC malware detection in windows OS. *Expert Systems with Applications, 41*, 5843–5857.

Qiao, Y., Yang, Y., He, J., Tang, C., & Liu, Z. (2014). CBM: Free, automatic malware analysis framework using API call sequences. In *Knowledge engineering and management* (pp. 225–236). Springer.

Rad, B. B., Masrom, M., & Ibrahim, S. (2012). Opcodes histogram for classifying metamorphic portable executables malware. In *Proceedings of international conference on e-learning and e-technologies in education* (pp. 209–213).

McAfee Labs (2015). McAfee Labs threats report: May 2015. http://www.mcafee.com/us/resources/reports/rpquarterlythreatq12015.pdf. Accessed 17.12.15.

Runwal, N., Low, R. M., & Stamp, M. (2012). Opcode graph similarity and metamorphic detection. *Journal in Computer Virology, 8*, 37–52.

Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., & Bringas, P. G. (2010). Idea: Opcode-sequence-based malware detection. *Engineering secure software and system* (pp. 35–43). Springer.

Schultz, M. G., Eskin, E., Zadok, E., & Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE symposium on security and privacy: 36* (pp. 38–49).

Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics, 1*, 1–22.

Shen, Y., Zhang, Z., & Yang, Q. (2002). Objective-oriented utility-based association mining. In *Proceedings of the international conference on data mining* (pp. 426–433).

Soucy, P., & Mineau, G. W. (2005). Beyond TFIDF weighting for text categorization in the vector space model. In *Proceedings of international joint conference on artificial intelligence: 5* (pp. 1130–1135).

Srikant, R., & Agrawal, R. (1996). *Mining sequential patterns: Generalizations and performance improvements*. Springer.

Sun, W. C., & Chen, Y. M. (2009). A rough set approach for automatic key attributes identification of zero-day polymorphic worms. *Expert Systems with Applications, 36*, 4672–4679.

Sundarkumar, G. G., Ravi, V., Nwogu, I., & Govindaraju, V. (2015). Malware detection via API calls, topic models and machine learning. In *Proceedings of the international conference on automation science and engineering* (pp. 1212–1217).

Symantec (2015). Symantec intelligent report: October 2015. http://www.symantec.com/content/en/us/enterprise/otherresources/b-intelligencereport102015enus.pdf. Accessed 17.12.15.

Uppal, D., Sinha, R., Mehra, V., & Jain, V. (2014). Malware detection and classification based on extraction of API sequences. In *Proceedings of the international conference on advances in computing, communications and informatics* (pp. 2337–2342).

Wchner, T., Ochoa, M., & Pretschner, A. (2014). Malware detection with quantitative data flow graphs. In *Proceedings of the 9th ACM symposium on information, computer and communications security* (pp. 271–282).

Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of international conference on machine learning: 97* (pp. 412–420).

Ye, Y., Li, T., Chen, Y., & Jiang, Q. (2010). Automatic malware categorization using cluster ensemble. In *Proceedings of the 16th international conference on knowledge discovery and data mining* (pp. 95–104).

Ye, Y., Wang, D., Li, T., Ye, D., & Jiang, Q. (2008). An intelligent PE-malware detection system based on association mining. *Journal in computer virology, 4*, 323–334.

Zeng, Y., Yang, Y., & Zhao, L. (2009). Pseudo nearest neighbor rule for pattern classification. *Expert Systems with Applications, 36*, 3587–3595.

Zhang, J. F., Chen, L. F., & Guo, G. D. (2012). Hierarchical feature selection method for detection of obfuscated malicious code. *Journal of Computer Applications, 32*, 2761–2767.