Multiple Trajectories Feature Detection Technology Based on Data Mining

Zheng SHAN1,2,a, Yang WANG1,2,b, Fei XUE1,2,c, Xi MENG1,2,d, Jin HAN1,2,e

1Information Engineering University, Zhengzhou 450001
2State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou Henan HEN371, China

azzzhengming@163.com, bz155340885@qq.com, czkaiwai@126.com, dmengxii0525@163.com, ehanjin1993@outlook.com

Keywords: Behavioral Trajectories, Data Mining, Prefixspan-x, Signature, Threshold.

Abstract. The performance of behavior feature of existing malware is single and it result in detected difficulty. So we use behavioral division to build a multidimensional malicious database which is consist of file operation, net access and memory access. We improved data mining algorithms prefixspan to prefixspan-x in construction process. Prefixspan-x adds function of automatic machine frequent sequence query while discarding the frequent sequences which is not satisfied with condition. And we use this model to dynamic interception and malware detected process. It will overcome the questions that static model cannot detect packers and confusion. Experimental results show that the proposed feature detection technology has high accuracy and low false negative rate.

Introduction

Popularity of the Internet contribute significantly to the development of society, and the malicious programs also showed explosive growth. 360 security company published "2015 Chinese Internet Security Report" that, by the end of 2015, 360 Internet Security Center intercepted a total of new malware samples 356 million, compared with 2014 growth of 9.9%. This is due to technical confrontation escalating between virus and antivirus software, the virus uses a more complex encryption and anti-killing technology, so there are more variants in the Internet. Variants of malicious code and malicious code detection is the focus of prevention [1].

Based on static reverse detection method is hard to break malicious code anti-disassembly techniques and extract features of the malicious code to be detected. Signature-based antivirus software updates its virus database exist a very serious lag. But variants of malicious code and malware behavior is not changed. So feature of malicious software should be extracted from its runtime behavior.

This paper proposed a multiple trajectories feature extraction and detection technology based on data mining(MTFED). First, extract a variety of track information of running software such as operating track of document flow, net flow and memory resources flow. Then using improved prefixspan-x algorithm of data mining sequential pattern to structure malicious feature database. The database is consist of file operation flow characteristics locus vector space, network flow characteristics locus vector space and resources control flow characteristics locus vector space (such as memory). Finally, design a multidimensional characteristic threshold model which is compatible with feature database model. Experiment shows that MTFED has a high accuracy rate and can effectively detect the malicious programs and its variants, but also has a low false positive rate.
Related Work

Malware Detecting

Malware detecting methods can be classified into heuristic based detecting methods and signature based methods [2]. The heuristic based detecting methods, such as RootkitRevealer system, compare the information upper system and the system statement of kernel, in order to identify the hiding files, processes and register information. The heuristic detecting methods can discover unknown malware, while the heuristic rules generated from the experience of analyst, and it is easy to exist high false positive rate and false negative rate in applications. Therefore, it is not generally used in detecting systems, especially in the commercial antivirus. The signature based detecting methods detecting through the signature extracted form malware. Compared to heuristic based detecting methods, signature based detecting methods have higher detecting efficiency and lower rate of false negative. So it is used generally in malware detecting tools, and it is the main method in malware detecting nowadays. Signature based detecting methods are classified into static signature detecting and dynamic signature detecting. Static signature detecting methods get signatures of malware through statically analyzing the PE structure, binary bytecodes, disassemble code of files. Static signature based detecting method need not run malware actually, so it is easy to achieve. Nowadays, there are many researches about static signature based detecting. For example, Matthew G.Schultz and EleazarEskin used the method of data mining technology to detect malware, within three ways which are RIPPER, Naïve Bayes and Multiple classifier systems. And they analyze PE of binary bytecode to extract signatures based on DLL, GNU strings and byte sequence. References [5-8] had researched the malware classifying and detecting method based on API call sequence. However, all of the above-mentioned methods are based on static analysis of PE files. And the advantages of those are that they need not execute malware, so they will not destruction the system. But so far, many malware use pack obfuscation techniques to disturb assembling. If it is not successful to unpack or compress, the static analysis of PE files will not be achieved, so that the detecting will be failed. And, the time information of API sequence can be tampered to avoid detected. Dynamic signature based detecting methods run malware programs in virtual environment, and monitor the behavior of these programs to get their behavior signature. Such as KonradRieck, who supposed collect malicious behaviors by sandbox, and established detecting models by machine learning. The reference [10] used an analysis method, which is based on CPU debug function and used trace-driven, to detect obfuscated malware. However, some malware will detect the virtual environment in virtual antivirus, so they can bypass the detecting programs through add special instructions or construct special structure. It will make antivirus programs be failed to detect malware. In dynamic detection, the program can execute in multipath, malware may execute in different paths because of different input data. Dynamically acquiring method can’t get all of the malicious behaviors, so that there will be false negative in detecting.

Malware Detecting Methods Based on Data Miming

The discovery of sequential pattern, which is finding all of the frequently subsequence in sequence database, is an active research branch in the field of data mining. Reference [11] used the machine code of byte sequence variable length N-gram as the feature extraction method, used weighted information gain as the signature select way, and used decision tree, Support Vector Machine and Naïve Bayes and so on as the classifier to detect malware. In reference [12], Apriori miming algorithm was used to implement detection of malware. However, all of the above methods execute behavior signatures statically, they can’t overcome the shortcoming of static detection. Reference [13] improved the prefixspan algorithm in sequence model, used contracted projection database instead of original database to optimize prefixspan algorithm, and used expert system to obtain the host behaviors to match the rules. While, they didn’t give out the specific accuracy of detection of this method.
Among data miming sequence model algorithms, prefixspan has a better performance to other algorithms. In this paper, authors enhance the traditional prefixspan to get Prefixspan-x algorithm, which has better efficiency, to satisfy a large number of sequential pattern mining to extract features of malicious code sequences.

**Improved Prefixspan-x Algorithm**

**Related Concepts**

Assume all items in each element are arranged in dictionary order. Given sequence 
\[ \alpha = \langle e_1e_2\ldots e_n \rangle, \beta = \langle e'_1e'_2\ldots e'_m \rangle (m \leq n), \gamma = \langle e_m\ldots e_n \rangle, \] if the following conditions are met:
\[ e_i = e_i (i \leq m-1), e'_m \subseteq e_m, \] and all items of \( e_n - e'_m \) are in the rear of items of \( e'_m \), so we called \( \beta \) is the prefix of \( \alpha \);
\[ \alpha' \] is the maximum subsequence of \( \alpha \), \( \beta \) is the subsequence of \( \alpha \), and \( \beta \) is the prefix of \( \alpha' \), so we called \( \alpha' \) is the projection of \( \alpha \) about \( \beta \);
The projection of sequence \( \eta \) about \( \gamma \) is \( \alpha \), so the postfix of sequence \( \eta \) about subsequence \( \gamma \) is \( \langle e'_me_{m+1}\ldots e_n \rangle \), and \( e'_m = (e_m - e'_m) \).
The projection database of \( \alpha \) is all of the sequence whose prefix is \( \alpha \) and relative to the postfix of \( \alpha \), and denoted as \( S|_{\alpha} \).

If \( \alpha \) and \( \beta \) satisfy with condition (1), so the supporting number of \( \beta \) in \( S|_{\alpha} \), which is the projection database of \( \alpha \), is the number of sequence \( \kappa \), while \( \kappa \) satisfy the condition \( \beta \subseteq \alpha \cdot \kappa \) in \( S|_{\alpha} \).

**Prefixspan Algorithm**

Prefixspan algorithm is miming sequence models through prefix projection, and during projecting, not all of the existing frequent subsequence are considered. Instead of it, prefix sequence is found, and the corresponding postfix is projected into a series of projection database. For each projection database, we just need find the local frequent model, and we don’t produce candidate key. The main procedure of this algorithm is following:

a) Scan database for one time, find frequent L2 sequence, and assume the number of it is \( k \);
b) Partition the research space: partition the whole sequence model into \( k \) research space, and make the prefix of them be frequent L2 sequence;
c) Construct the relevant databases, which are the postfix sets of relevant prefix;
d) Find the subsets of frequent model recursively in these postfix sets;

**Prefixspan-x Algorithm**

Candidate sequence is not produced in the miming process of prefixspan algorithm, and relative to the original sequence database, the scaler of projection database is decreasing continually. However, the cost of construct projection database is very large, so the problem of dense data set or large mode miming could not be solved well. The main cost of this algorithm is constructing a number of projection database recursively and scanning databases repeatedly during the executing of algorithm. The main methods to improve prefixspan algorithm are reducing the scale of projection database and optimizing scanning time. In this paper, authors use an improving algorithm, prefixspan-x algorithm, use AC automaton to optimize during searching sequence meet to minimum supporting degree, and
abandon frequent sequence which is not satisfied with minimum length during constructing projection database, so that the cost of time and space during mining will be optimized.

Prefixspan-x algorithm is following:

\[ \text{Prefixspan-x}(a, (\minsup, \minl), S) \]

**prefixspan-x()**:  
1) \( AC(S, \minsup) \)
2) \( S\_{\text{a}} = \text{creatsuffix}(S, a) \)
3) \( b \in S\_{\text{a}} \)
4) \( \text{foreach}(b \neq \text{null}) \)
5) \{  
6) \( a' = a + b; \)
7) if \( l(a') \geq \minl \)
8) get \( a' \)
9) else
10) delete \( a; \)
11)\}
12) \( S |_{a} = \text{creatsuffix}(S |_{a}, a'); \)
13) \( \text{prefixspan} - x(a', l(a) + 1, S |_{a}, \minl); \)
14) \( \text{output}(M1); \)

For example, give following sequence database and assume the minimum support degree is 2, and the minimum length of frequent sequence is 1:

\(< \text{read,write}> \text{read,lseek} >\)
\(< \text{lseek,dup}> \text{dup2,pread,close} >\)
\(< \text{read,lseek}> \text{fsync}(\text{close}) > < \text{fsync} >\), find the frequent items whose support degree is more than 2: read, lseek, fsync, close; then remove all the infrequent items, and generate database

\(< \text{read}> \text{read,lseek} >\)
\(< \text{lseek}> \text{fsync} >\)
\(< \text{read,lseek}> \text{fsync}(\text{close}) >\)

Generate projection database for each frequent item read, lseek, fsync, close, and remove all the frequent sequence which has a smaller length than 2:

\(< \text{read,lseek} > < \text{lseek}> \text{fsync}(\text{close}) > < \text{fsync}(\text{close}) >\)

In the above projection databases, the projection databases of prefix \(< \text{read} >\) also has a frequent item lseek, the projection databases of prefix \(< \text{lseek} >\) also has a frequent item close. Generate frequent
Represent of Software Behavior Trajectory

The interaction between software and computer is implemented by the operating system, and the form of its performance is that the operating system provides a series of interfaces (i.e., system call). It will leave running trajectories after running software on the operating system (i.e., order of the system call after running by time order). It provides a basis for distinguishing normal procedures and malicious programs that studying software running trajectories to find out its behavior features.

Definition 1 Behavior trajectories: Ordered sequence composed by the system calls that are executed according to the time order. The length of the system call sequence l equals to the number of system calls in the sequence.

e.g., in the Linux system, a software behavior trajectories can be expressed as R = (read, mmap, write, close) and the length of the trajectories R can be expressed as l(R)=4.

The software behavior trajectories, the trajectory fragments and the system calls have the following relationship:

We suppose that there are m trajectories in the software (i.e., the software behavior trajectories), and the m trajectories are expressed as T_1, T_2, T_3, ..., T_m, the i-th of which is expressed as T_i = (R_1, R_2, R_3, ..., R_k). R_k is the k-th behavior fragment of the trajectory T_i, R_k = (s_1, s_2, s_3, ..., s_j). R_k is the system call sequence which compose trajectory T_i fragment, and the j-th system call (l ≤ i ≤ m, 1 ≤ j ≤ l) of the time order sequences is expressed as s_j.

Feature Library Construction Based on Prefixspan-x

Feature Element

Definition 2 signature: The specific behavior trajectory fragments in executing can be used to judge whether a program is malicious. Malicious behaviors of malicious programs usually present many aspects. Such as abnormal operation of the document, abnormal access to the network and unauthorized access to resources such as memory, etc. Traditional extraction of malicious program features based on behavior semanteme is the description for the specific and coherent behaviors of the malicious programs, and this method makes the detection fast and accurate. If a malicious program inserts confused program or confused behavior sequence in the behavior trajectories, the detection based on the traditional features will not detect the variants of malicious programs. Therefore, in the paper the elements of malicious behavior features have three dimensions: File flow features, network flow features and resource flow controlling features. The signature is expressed as follow:

\[ \text{signature} = \text{file-sign} \lor \text{net-sign} \lor \text{source-sign} \]

(1)

In which file-sign represents the file flow signature, and net-sign represents the network flow signature, and source-sign represents resource controlling flow signature. The detection for malicious programs is based on this three signature flow.

Signature Library Construction

Sequential pattern mining algorithm can solve the problem of mining software behavior signature in the large sequence database. In this paper, an improved sequential pattern mining algorithm Prefixspan-x is adopted to implement signature sequence mining and to construct signature library. The signature library is consisted by the three-dimensional signature vector space: File flow signature vector space, network flow signature vector space and resource flow signature vector space.
Definition 3 Signature Database: The database consisted by a series of signatures which can represent the malicious programs.

\[ SD = (F_D, N_D, S_D) \]  

In which \( F_D \) represents the file flow signature vector space, \( N_D \) represents the network flow signature vector space, and \( S_D \) represents the resource flow signature vector space.

Learning problem of malicious behavior signature database can be described as:
1. Giving the training set of the malicious programs \( \theta \) and the minimum support \( \text{min}_\text{sup} \).
2. Extracting behavior trajectories of the malicious programs, i.e., file flow behavior trajectory, network flow behavior trajectory and resource control flow behavior trajectory.
3. Adopting Prefixspan-x algorithm mining the frequent sub sequence set \( M_1 \) which can meet the minimum support and the minimum length.
4. Given the behavior trajectory set of the normal program, removing the normal behavior trajectory fragments from \( M_1 \).
5. Generating signature database of the malicious behavior.

The database construction is divided into two phases, while one is the frequent sub sequence mining based on Prefixspan-x algorithm and the other is streamlining the frequent sub sequence set which extracted in phase one.

The specific process of phase one is as follows: For a malicious program given the training set, the paper extracts file flow behavior trajectory, network flow behavior trajectory and resource control flow behavior trajectory of the malicious programs, and respectively initializes training set of the file flow behavior trajectory, the network flow behavior trajectory and the resource control flow behavior trajectory. And then Prefixspan-x algorithm is used to extract the frequent sub sequence set.

Phase two is filtering for the normal behavior trajectory fragments. The frequent sub sequence set extracted in phase one conclude not only the malicious behavior trajectory fragments but also the normal behavior trajectory fragments. The malicious behavior signature vector space can be generated after filtering the normal behavior trajectory fragments. The date set of the normal behavior trajectories uses the behavior trajectories generated after running the normal programs in the initial operating system. The algorithm is as follows (Fig. 1):

In the algorithm 3, there exist the size problem of the normal behavior trajectory fragment set \( D_n \). If the size is too small, the final SD will interfuse the normal behavior trajectory fragments resulting in a higher false alarm rate. If the size is too big, the time expenditure of the extraction algorithm will be too large. The size selection depends on the number of the sample.

**Input**: \( M_1 \)

**Output**: \( M_2 \)

Function: eliminate \((M_1, D_n)\)

1) for \( t \in M_1 \) do
2) for \( s \in D_n \) do
3) if \( s \text{ MATCH } t \) ;
4) Delete \( t \) From \( M_1 \);
5) End for
6) End for
7) Output \( M_2 \);

Figure 1. Algorithm of Removing the Normal Behaviors.
Multi-Trajectories Detection Based on the Behavior Signature

Structural signature require appropriate detection model to highlight its advantages. The running trajectories are different for different properties in running program, and the sequences of the system call are different. In this paper, variable length sequences slice matching algorithm is adopted to detect the system call sequences and extracted signature.

Detection flow is as follows:
The program behavior trajectories are respectively extracted based on the file flow operations, the network operations and the resources operations such as memory within the monitoring time. Supposing that behavior trajectories are represented as \( R_f, R_n, R_s \), \( R_f = (s_1^f, s_2^f, \ldots, s_{r^f}^f) \), \( R_n = (s_1^n, s_2^n, \ldots, s_{r^n}^n) \), \( R_s = (s_1^s, s_2^s, \ldots, s_{r^s}^s) \). In which, \( r, m, q \) respectively represent the length of the system call sequences corresponding to \( R_f, R_n, R_s \). \( s_k^i \) represents the k-th trajectory fragment \( 1 \leq k \leq l(R_i) \) of the i-th behavior trajectory \( (i=f,n,s) \) in the system call sequence. Supposing that \( \text{class}(S) = 0 \) represents the matching weight of the sequence S, and its initial value is 0. The following section only uses the behavior trajectories of the file flow operations as the example. Slice \( R_f \), then N. trajectory fragments are obtained, i.e., \( S_1, S_2, \ldots, S_n \). Ending at \( s_k \left(N < k < l(R_f)\right) \), \( S_i \) are the system call sequences whose length are \( l(1), l(2), l(3), \ldots, l(N) \).

According to the order from N to 1 and the file flow signature of the signature library, these N trajectory fragments \( S_1, S_2, \ldots, S_n \) are compared and calculated as follows: If there is a fragment matching the signature library successfully, there will be \( \text{class}(S_i) = 1 \), otherwise \( \text{class}(S_i) = 0 \).

The weight sequence will be obtained after the above operation:

\[
\left(\text{class}(S_1), \text{class}(S_2), \ldots, \text{class}(S_n)\right)
\]

According to the weight sequence

\[
\left(\text{class}(S_1), \text{class}(S_2), \ldots, \text{class}(S_n)\right)
\]

Calculate the decision value, and make decision according to the decision value and decision threshold \( \lambda \) set in advance.

Calculate judgment component:

\[
\text{sign}_f = \text{class}(S_1) \lor \text{class}(S_2) \lor \ldots \lor \text{class}(S_n)
\]

Seek the judgment value \( \text{sign} = \text{sign}_f + \text{sign}_n + \text{sign}_s \), and judge:

- If \( \text{sign} > \lambda \quad \lambda \neq 0 \), the behavior is malicious.
- If \( \text{sign} \leq \lambda \), the behavior is normal.

\( \lambda \) is a user-set judgment threshold. Different values represent different levels of security, and the smaller the value represents the higher the level of security, the higher the value represents the lower the level of security.

Experimental Design and Results Analysis

Experimental Design

Experimental basic environment: the x86-64 architecture, processor: Intel (R) Core (TM) 2 Quad CPU Q9500, CPU clocked at 2.83GHz, size of memory is 8G, operating system version is centos release 6.5 (Final), Kernel Linux 2.6.32-431.el6.x86_64, GNOME 2.28.2.

Sandbox simulation environment using XEN4.4 virtual machine, install Centos release 6.5 (Final) operating system, install the Java virtual machine, Python interpreter, GCC compilers and other...
software in the system. Without using VMware and Qume but XEN4.4 because the virtual environment can easily be detected by malicious software of behavior-based virtualization software monitoring model. For example, in order to cause difficult to be detected, some malicious software can detect whether VMware communication port contains "VMXH" and hide their own malicious behavior. XEN is a hardware-based virtualization technology can avoid this defect, and virtualization running on high-level privileges can be a good monitor malicious programs.

The experimental set of experimental include 733 malicious samples and 600 normal programs. Malicious samples are downloaded from http://www.VXheaven.org and http://bbs.kafan.cn. Normal programs are from centos 6.5 initial system. Training set contains three types:100 Email-worm, 50 virus and 20 other type of worm programs. Selecting 300 normal programs from the initial system to be normal set. The test set contains two parts: part one (Table 1) is used to testing the capability of model to detect the malicious variants; part two (Table 2) is used to testing the capability of model to detect the whole malicious programs. All selected samples in test set have characteristics of time distribution because antivirus software database update was in February 2015, the samples in test set should be across the time point. So the above two test sets cover 10% malicious programs after April 2016.

Table 1. Test Set 1.

<table>
<thead>
<tr>
<th>Sample type</th>
<th>Count</th>
<th>Sample name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email-wrom</td>
<td>507</td>
<td>Email-Worm.VBS.Lee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email-Worm.VBS.Ypsan.b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>Worm</td>
<td>57</td>
<td>Net-Worm.Perl.Sany.c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRC-Worm.JS.Nepmoon.a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>Normal program</td>
<td>600</td>
<td>easy_install.py</td>
</tr>
<tr>
<td></td>
<td></td>
<td>issue.net</td>
</tr>
</tbody>
</table>

Model learning process constructs signatures for each training set. Using Linux system command “strace” to obtain the behavior trajectory of normal and malicious program.

Performance

Table 2. Test Set 2.

<table>
<thead>
<tr>
<th>Sample type</th>
<th>count</th>
<th>Sample name</th>
</tr>
</thead>
<tbody>
<tr>
<td>worm</td>
<td>564</td>
<td>Email-Worm.JS.Depends</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email-Worm.JS.TheFly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email-Worm.JS.Hatred.a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email-Worm.JS.Nevezed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>Virus</td>
<td>169</td>
<td>Virus.Python.Biennale</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Virus.C.Califax</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Virus.C.LinDataSeg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>Normal progress</td>
<td>600</td>
<td>easy_install.py</td>
</tr>
<tr>
<td></td>
<td></td>
<td>issue.net</td>
</tr>
<tr>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
Performance of this malicious code detection experiment is as follows:

Accuracy Rate (AR): the proportion of the program correctly detected in the sample set, including the correctly detected malicious and normal programs.

Detection rate (DR): the proportion of the malicious program correctly detected in the sample set.

False Positive rate (FPR): the proportion of the program incorrectly detected as malicious programs in the sample set.

**Analysis of Experimental Results**

Experiments using three popular antivirus software on Linux platform (Table 3) to scan respectively and comparing with test results with ri=3,4,5. The result is as follows:

**Table 3. Test Antivirus Software Information.**

<table>
<thead>
<tr>
<th>Antivirus software</th>
<th>version</th>
<th>Database version/sample count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clamav</td>
<td>0.98.4</td>
<td>3719758</td>
</tr>
<tr>
<td>Avast</td>
<td>2015.10.0.2208</td>
<td>150107-1/3002652</td>
</tr>
<tr>
<td>COMODO</td>
<td>7.0.317799.4142</td>
<td>20635</td>
</tr>
</tbody>
</table>

![Figure 2. Result of Test Set 1.](image)

![Figure 3. Result of Test Set 2.](image)
The results can be seen from Fig. 2 and Fig. 3, the model has a great detection results of the malware and its variants and the result closes to the antivirus software test result. Because the selected test samples are from Internet, antivirus software has most of signature of the malicious software and the antivirus software detection of virus database mechanism has a very high accuracy rate. But due to limitation of the number of samples of the virus database, each test result is different. The new virus cannot be detected because the test sample selection has characteristics of time distribution and the antivirus software’s virus database version updating late the new virus cannot be detected. This model is restricted to the size of the training set and the limit of the user-defined security levels, AR and FPR is unstable. When the threshold value is lower, the DR is higher and the FPR will be higher. When the threshold is higher, the FPR is lower and the DR is relatively lower. Model’s AR is close to the selected antivirus software and the false negative rate is relatively lower than antivirus software.

Conclusion

In order to solve the problem of spreading malicious programs and variants, this paper bases on data mining and software behavior trajectory proposed a malicious programs feature extraction and detection method. Using improved Prefixspan-x algorithm of data mining sequential pattern to extract feature. Experiments show that the model has better detection capabilities on the Linux platform and the result is close to the current detection result of the popular antivirus software. The downside is that there is still much room for improving performance of Prefixspan-x algorithm of data mining sequential pattern which is next step needs to be optimized direction.

References


