Mining Distinguishing Patterns Based on Malware Traces

Xiaoyan Sun, Qian Huang, Yuefei Zhu
Zhengzhou Information Science and Technology Institute
Zhengzhou, Henan, China

Ning Guo
Lanzhou University
Lanzhou, Gansu, China

Abstract—The automatic generation of malicious behavior pattern based on system call trace is important to malware detection. This paper studied the existing generation method of malicious behavior specification. In order to reduce the complexity of pattern generation, it constructs graph which vertex label is unique, and uses these graphs to mine the pattern. To address the issue of limitation of the minimal contrast subgraph mining method, it uses multiple positive and negative samples, and proposes a mining method to mine distinguishing patterns based on mutual information. It designs the overall framework of mining process, and gives the mining algorithm. Finally, validation results demonstrate the effectiveness.

Keywords- Malicious behavior; System Call Trace; Subgraph Mining; Distinguishing Pattern; Mutual Information

I. INTRODUCTION

With the development of the Internet, it is difficult to guarantee the security of Internet. A number of malware such as virus, Trojan horse exist in the Internet [1]. Most malware can mutate in the process of spreading. The increasing number of variants of malware will lead to the increasing number of syntax characters such as string or bytes. In order to solve the problem researchers proposed the behavioral detection [2].

Behavioral detection methods include expert system [3], model checking [4] and so on. All of these methods need the patterns that contain semantic information. These patterns can be case-based inference, sequential inference, logic specification, semantic graph and so on. These patterns can resist the malware mutation to a certain extent. So a key problem is the automatic generation of behavioral patterns.

In this paper, we study the methods of generation of behavioral patterns, and design a framework to generate patterns automatically.

This paper is organized as follows:

Section 2 gives the related work in the area of malware behavioral pattern generation. Section 3 introduces some related concepts. Section 4 gives the overall framework of the automatic generation of malicious behavior pattern, mainly presenting the design and implementation of graph constructing and the mining algorithms. Section 5 gives the results of our framework. Finally, we conclude the paper and give the further research directions in Section 6.

II. RELATED WORKS

In the field of malware behavioral pattern generation, a main method is the manual generation by experts. Manual generation is reliable but time consuming. In order to decrease the time of generation, we focus on the automatic generation methods.

Several works have been developed for malware features generation. Most of them use data mining method, such as classifier [5,6,7] or clustering [8]. Most of the features they use are sequences, such as byte n-grams, opcode n-grams, system call n-grams. Various feature selection methods, including document frequency, fisher score, and gain ratio have been commonly used for reducing dimension of features.

In order to generate behavioral pattern that can be used by antivirus or security software on the behavioral detection, Mihai Christodorescu [9] proposed a mining method based on minimal contrast subgraph [10]. This work has significant importance for the malicious behavior pattern (they called them specification) automatic generation. It can find the patterns that appear in one malicious trace but not appear in some non-malicious traces. But the worst time complexity is O(N!). Except these, it only finds the pattern in trace of one malware.

We study the work of Mihai, and propose some method to improve the work of malicious behavior pattern automatic generation, and design a framework for it.

III. SOME CONCEPTS

In the field of malware detection, behavior based detection arouses general interest, for example, we can get the program behavior from the disassemble code, decompile results [11], or run the program in a secure environment and monitor the behavior [12], then detect the existence of malicious behavior. These malicious behaviors can be expressed as inference rules, or the automatic machine model with semantic information. These rules or models can be called distinguishing patterns, because their appearance are different in malicious programs and normal programs.

A. Distinguishing Pattern

We first introduce some concept of distinguishing pattern [13].

Definition 1: Given a database $D$, a pattern $p$, the number of occurrences of $p$ in $D$, denoted by $abs\ sup(p, D)$, is the absolute support of $p$ in $D$. The result of $abs\ sup(p, D)$ divided by the number of total patterns, denoted by $rel\ sup(p, D) = abs\ sup(p, D) / |D|$, is the relative support of $p$ in $D$.

Definition 2: Given two databases $D_1$ and $D_2$, a pattern $p$, the relative support of $p$ in $D_1$ $rel\ sup(p, D_1)$ and the relative support of $p$ in $D_2$ $rel\ sup(p, D_2)$, two support threshold $\alpha$ and $\beta$, ($\alpha, \beta \in [0,1]$), $p$ is a distinguishing pattern, iff $rel\ sup(p, D_1) \geq \alpha$ and $rel\ sup(p, D_2) \leq \beta$.
We can see from the definition that the distinguishing pattern in general sense is a pattern appears more in one class, but less in another class. Because of malware’s complexity and uncertainty, a specific behavior pattern \( p \) needs to have one of the following properties:

1) \( p \) does not appear in normal programs behaviors, but appears in behaviors of malicious programs.

2) The relative support of \( p \) in normal programs behaviors is small, but the relative support of \( p \) in malicious programs behaviors is large.

The general definition of the distinguishing pattern only contains the second property. So we need to redefine the definition. We use the mutual information to measure the degree of distinguishing \( p \) meet the above properties. Therefore, we proposed the definition of Distinguishing Pattern based on Mutual Information (DPMI).

B. Definition of DPMI

Mutual information is an important concept in information theory [14]. It is widely used in statistical language model and correlation analysis to measure the degree of correlation between patterns and classes. Given a pattern \( t \) and a class \( c \), the mutual information \( MI(t,c) \) is calculated as follows:

\[
MI(t,c) = \ln \frac{p(t|c)}{p(t)} = \ln \frac{p(t|c)}{p(t,c)}
\]

\( p(t) \) is the result of the number of samples that contain the pattern \( t \) divided by the number of total samples. \( p(t|c) \) is the probability of \( t \) that appears in class \( c \). The mutual information is positive when the pattern only depends on the class. The mutual information is 0 when the pattern does not depend on the class. The mutual information is negative when the pattern rarely occurs on the class.

We only concern patterns that appear in malware samples, so the distinguishing degree is defined as follows:

\[
EI(t) = MI(t,c_1) - MI(t,c_2) = \ln \frac{p(t|c_1)}{p(t|c_2)} - \ln \frac{p(t|c_1)}{p(t|c_2)} = \ln \frac{p(t|c_1)}{p(t|c_2)}
\]

\( MI(t,c_1) \) is the correlation between pattern \( t \) and class \( c_1 \) (malicious programs), \( MI(t,c_2) \) is the correlation between pattern \( t \) and class \( c_2 \) (normal programs), the distinguishing degree \( EI(t) \) of \( t \) is the result of \( MI(t,c_1) \) minus \( MI(t,c_2) \).

The \( p(t|c) \) will not be 0 (patterns are from the malicious samples). In order to prevent the \( p(t|c) \) to be 0 we introduce a function \( P \), defined as follows:

\[
P(X) = \begin{cases} 
1 & p(x) = 0 \\
p(x) & p(x) \neq 0 
\end{cases}
\]

\( N \) is the number of normal samples. Thus, we can get a new distinguishing degree \( EI(t) \):

\[
EI(t) = \ln \frac{P(t|c_1)}{P(t|c_2)}
\]

Definition 3: Given a malware database \( D_1 \) and a normal programs database \( D_2 \), their corresponding classes are \( c_1 \) and \( c_2 \), a threshold \( \gamma \), \( (\gamma > 0) \), a pattern \( t \) is a malware Distinguishing Pattern based on Mutual Information (DPMI), iff

\[
\ln \frac{P(t|c_1)}{P(t|c_2)} \geq \gamma.
\]

When \( p(t|c_2) \neq 0 \), \( P(t|c_2) \) approximately equal to \( \text{rel}_\text{sup}(p, D) \). When \( p(t|c_2) = 0 \),

\[
P(t|c_2) = \frac{1}{N}.
\]

\( N \) is the number of samples in \( D_2 \).

IV. OVERALL FRAMEWORK

In operating systems, program behaviors need system resources and services. With the emergence of behavior-based detection, the detection of system calls is widely spreading. Each system call is an operation to a resource or service. Using system calls to compose the pattern of malicious behavior is suitable for current security software. In this paper, we mine the malware DPMI based on dynamic system call trace.

The whole framework is given in figure 1. (a) Shows the generation process of dependency graph from the API call traces. It first preprocesses the API traces and makes them to become operations (including object and other information). Then it uses the operations to construct dependency graph. (b) shows the mining process from dependency graphs of malware and normal programs.

Every system call can be mapped to an operation \( op(\alpha, c) \), where \( \alpha \) is the object such as files, registry, network, process / thread and so on, and \( c \) is other information except the operation object. In order to reduce the complexity, we only concerned with two types of parameters: HANDLE and string. Next we focus on the graph constructing and mining process.
A. Construct Dependence Graph

The relationship between parameters is divided into def-use, same-use, and same operation. Def-use relationship is the same as [9]. Same-use relationship refers to string parameters of two operations, and they are the same or part of the same. Same operation refers to two operations and their objects, one operation and its object is the same as another operation and its object.

**Definition 4:** An operation dependency graph is a directed graph \( G = (V, E, \gamma, \sigma) \), its vertex set is \( V \), \( \gamma \) associated the vertex with an operation and it’s object, \( \gamma: V \rightarrow OP(O) \). For example, a sequence:

\[
OP(O, C) = op_1(o_c, c_1)op_2(o_c, c_2)\ldots op_n(o_c, c_n),
\]

\( \gamma(v) = op_j(o) \). Edge set is \( E \), for any two vertices, and \( j>i \), if \( \gamma(v) \) is the first operation that has same-use relationship with \( \gamma(v_i) \), then there is an edge between \( v \) and \( v_i \), denoted by \( <v, v_i> \in E \). It uses \( \rho: E \rightarrow string \) to constrain each edge. The string is “o=o” or “o=c” or “c=o” or “c=c”. “o” represents the operation object. “c” represents other information except the operation object. “=” means they are the same or part of the same.

First of all, if there is a def-use dependency between two operations, it replaces HANDLE parameter with its corresponding object name. If it can’t find the corresponding object name it uses the generic object name to replace the handle. This makes a def-use relationship change into a same-use relationship. Then it uses the definition 4 to construct the graph. After that, it deals with the operation relationship to remove duplicated vertices that have same label. In the end, each vertex in the graph has unique label, and there are not duplicated edges.

We mine malware DPMI using the operation dependency graphs, so the pattern is a subgraph.

**Definition 5:** Given a set of malware dependency graphs \( G_g = \{G_{g_1}, \ldots, G_{g_m}\} \), a set of normal programs dependency graphs \( G_s = \{G_{s_1}, \ldots, G_{s_n}\} \), the corresponding classes are \( c_g \) and \( c_s \), a threshold \( \gamma \),

\[
\gamma \in (0, \ln |G_g|] \text{ or } \gamma \in [\ln \frac{|G_s^g|}{|G_g^s|}, \ln |G_s^g|].
\]

A subgraph \( g \) is a Malicious behavior Distinguishing Subgraph based on Mutual Information (MDSMI), iff

\[
\frac{P(g \mid c_s)}{P(g \mid c_g)} \geq \gamma.
\]

If there is no proper subgraph of \( g \) is a MDSMI, then \( g \) is a minimal MDSMI.

B. MDSMI Mining Algorithm

According to the above definition, we need to study how to obtain minimal MDSMI. We use expanding tree generation method. Because each vertex in the graph has unique label, and there are not duplicated edges, all edges can be sorted to a unique sequence. We use the following order:

1) Vertex labels and edge labels are sorted by Lexicographic order.

2) For edge \( <v, v_i> \), \( \rho(v, v_i) \), \( \gamma(v) \) has a first priority, \( \rho(v, v_i) \) has a second priority, \( \gamma(v) \) has a third priority.

We sort all edges by this order. During expanding, the expansion sequence is as follows:

The order of edge that can be expanded must be less than all of the edges of current graph.

This ensures the uniqueness of the expansion sequence.

In order to improve efficiency, it needs early pruning. We give the following theorems.

**Theorem 1:** If \( g \) is a minimal MDSMI w.r.t \( G_g \) and \( G_s \), then all descendants of \( g \) in the expanding tree will not be minimal MDSMI.

The proof is same as [15].

**Theorem 2:** If \( g \) is not a minimal MDSMI w.r.t \( G_g \) and \( G_s \), then all descendants of \( g \) in the expanding tree will be not be minimal MDSMI.

The proof is same as [15].

**Proof:** Let \( g' \) is any offspring of \( g \), then \( g' \) is a supergraph of \( g \), then \( P(g' \mid c_g) \leq P(g \mid c_g) \), because \( P(g \mid c_g) = \ln |G_s^g| \), then \( P(g \mid c_g) = \ln |G_g^s| \), since \( g \) is a minimal MDSMI,

\[
\ln \frac{P(g \mid c_g)}{P(g \mid c_s)} < \gamma.
\]

Then, \( \ln \frac{P(g' \mid c_g)}{P(g' \mid c_s)} < \ln \frac{P(g \mid c_g)}{P(g \mid c_s)} < \gamma \).

Therefore \( g' \) is not minimal MDSMI. \( \square \)

Zhiping Zeng etc [15] gave a minimal distinguishing subgraph mining algorithms. We considered isolated vertices and modified the expansion order. In the end we get the
isolated vertices set and graphs that have no isolated vertices. The algorithm is as follows:

Algorithm: MDSMI-Mine( \( G, N, \gamma \) )

Input: \( G = \{ G_1, ..., G_n \}, G = \{ G_1, ..., G_n \}, \) a threshold \( \gamma \).
Output: \( rs \) the set of minimal MDSMI.

Begin
1. Scan \( G \) to get vertices set \( V_p \);
2. for each vertex \( v \) in \( V_p \)
3. compute the \( \sup(v, G) / |G| \) and \( \sup(v, G) / |G| \), and get \( P(v | c_j) \) and \( P(v | c_k) \);
4. if \( \ln P(v | c_j) \geq \gamma \), insert \( v \) to \( rs \);
5. Scan \( G \) to get edge set \( E_p \) and sort them;
6. for each edge \( e \) in \( E_p \) (select from small to big)
7. MDSMI-Enum(\( e \))
End

Subalgorithm: MDSMI-Enum(\( d \))

Input: \( d \) a code representing a subgraph.
Output: \( rs \) the set of results w.r.t \( d \).

Begin
8. if \( d \) is a super graph of \( g', g' \in rs \), return;
9. Compute \( \sup(d, G) / |G| \) and \( \sup(d, G) / |G| \), and get \( P(d | c_j) \) and \( P(d | c_k) \);
10. if \( \ln P(d | c_j) \geq \gamma \), insert \( d \) to \( rs \), return;
11. if \( P(d | c_j) = 1/|G| \), return;
12. Get the extensible edge set \( E \) of \( d \);
13. for each edge \( e \) in \( E \)
14. MDSMI-Enum(\( d \) | \( e \));
End

First of all, it judges the vertices to get distinguishing vertices. Then it focuses on the edge set. In line 8, it will judge whether \( d \) is a super graph of \( g', g' \in rs \), if so there is no need for expansion, return. In line 10, it will judge whether it is a MDSMI, if so insert it to result set and return according to theorem 1. In line 11, it will judge whether \( P(d | c_j) = 1/|G| \), if so return according to theorem 2. In line 12-14, it adds an edge \( e \) to \( d \) (\( d \) | \( e \)) to enumerate.

Next, we discuss the meaning of threshold values.

Let \( |G| = N_j \) and \( |G| = N_j \) when \( N_j < N_j \), the value range of \( \ln P(d | c_j) \) has two intervals:

\[
[\ln \frac{1}{N_j}, \ln N_j], [\ln \frac{N_j}{N_j}, \ln N_j]
\]

If \( \gamma \in \left[ \ln \frac{N_j}{N_j}, \ln N_j \right] \), we can find the patterns that satisfy the first property.

If \( \gamma \in \left( \ln N_j, \ln N_j \right] \), the patterns that satisfy the first property must be found, and we can find patterns that satisfy the second property.

Similarly, we can get the meaning in the situation of \( N_j > N_j \). The value of \( \gamma \) can be adjusted according to specific needs.

Now we discuss the complexity of the algorithm. Typically, when the graphs have large number of edges, high vertices degree, and small vertex label set, the graph expansion and isomorphism time complexity is \( O(2^n) \)[16], where \( n \) is the number of edges. But in our case, each vertex in the graph has a unique label, and there are not duplicated edges. The dependence graphs have less edges, low vertices degree (<3), and large vertex label set, the isomorphism time complexity is \( O(n^3) \).

V. RESULTS AND ANALYSIS

We got 30 normal executable files from the clean system of Microsoft Windows xp, and collected 30 viruses and 30 trojans from Internet. We executed them in the virtual machine, and got their execution traces.

First, we constructed dependence graphs from the traces. We give the comparison result of constructed vertex number and original vertex number in the case of 20 samples.

![Figure 2. Vertex Number Comparison](image)

It is clear that the numbers of vertices of constructed graphs are smaller and stabilized.

We got minimal MDSMI from 30 trojans and 30 normal programs, \( \gamma = 4 \), partial results are as follows:

| TABLE I. PARTIAL RESULT OF TROJAN AND NORMAL PROGRAMS |
|-------------------------|-------------------------|-------------------------|
| MDSMI                   | \( P(g | c_j) \)         | \( P(g | c_k) \)         |
|                         | \( P(g | c_j) \)         | \( P(g | c_k) \)         |


We studied the automatically generation method of distinguishing pattern that represents malicious behavior with semantic information, and proposed the definition of DPMI and MDSMI based on mutual information. We designed the overall mining framework, and gave the mining algorithm. In the end, the experimental results showed the effectiveness. In the future we will study the pattern reduction and detection methods based on these patterns.

REFERENCES