IEICE TRANS. INF. & SYST., VOL.E97–D, NO.3 MARCH 2014

PAPER

Research on Software Trust Analysis Based on Behavior

Yingxu LAI[†a)], ***Member***, **Wenwen ZHANG**[†], ***and*** **Zhen YANG**[†], ***Nonmembers***

SUMMARY In this paper, we propose a new trusted modeling approach based on state graphs. We introduce a novel method of deriving state-layer from a system call sequence in terms of probability and statistics theory, and we identify the state sequence with the help of Hidden Markov Model (HMM). We generate state transition graph according to software executing process and pruning rules. Then, we separate local function graphs according to software specific functions by semantic analysis. The state-layer is a bridge between the basic behaviors and the upper layer functions of software to compensate semantic faults. In addition, a pruning strategy of formulating state graphs is designed to precisely describe each piece of software functions. Finally, a detecting system based on our model is proposed, and a case study of RSS software reveals how our system works. The results demonstrate that our trusted model describes software behaviors successfully and can well detect un-trust behaviors, anomaly behaviors, and illegal input behaviors.

key words: *trust, software behavior, system call, state-layer, state graph*

1. Introduction

Along with the rapid development of application software in many important fields, trust and security of software have always been an important issue. Qu considers that it is a vital factor in estimating software trust whether software behavior is legal, permissible and expected [1]. In order to analyze behavior trust, software behavior model should be established for normal behaviors, through which untrusting behaviors can be detected. Even though there are many suggestions of establishing software behaviors models, they are either too complicated to be implemented or lack semantic analysis. Modeling method mainly includes sequence enumeration, machine learning, finite state automaton, etc. [2]–[11]. Short system call sequences are used to represent software behaviors in both sequence enumeration [3] and machine learning [6]. However, the calculation and storage space grow explosively as the length of short sequence increases. Automaton approach [7] establish finite state automaton model in terms of branching or looping structures of program. The automaton usually contains system call, return addresses, virtual stack and other complicated features. In spite of remarkable enhancement in accuracy, it leads to high complexity and enormous storage needs.

All of these modeling approaches, system call is generally used to express software behaviors. Because it is an interface to access OS (operating system), and seldom be changed in different OS editions. But system call sequence as software behavior expression directly will result in an obvious problem: the model has no exact semantic information, which will cause low accuracy. That is because that system call is the basic behavior generated by OS, it cannot describe software behavior thoroughly, and lacks analyzing semantic relationship of system calls. In order to overcome these difficulties, we try to extract a higher layer behavior mode instead of system call sequence. Higher layer behavior mode, called state, is frequent sequence pattern extracted from system call sequence. Essentially, it reveals inner properties hidden in original system call sequence to compensate semantic faults.

In this paper, a new modeling method, software behavior trust model based on state-layer, abbreviated as SBTMS, is presented in Fig. 1. In Fig. 1, the bottom layer is the system call layer, which includes intercepted system call sequence of the running application software in chronological order. The middle layer is the state-layer, which connects basic behaviors with upper functions of software to compensate semantic faults. State is a group of strongly interrelated system calls, associated with software function. The top layer is the software behavior template based on software function, includes a state transition graph and several local function graphs. With the help of pruning rules, state transition graph can depict the overall performance of running software well. On the other hand, the local function graphs, each of which represents one specific software function, present an entrance to analyze semantic relationship of software.

The reminder of this paper is organized as follows. Section 2 contains a review of related research. Section 3 presents implementation of software behavior trust modeling based on state-layer, and state-layer generating method and pruning rules are stated in this section. System architecture and experiment are given in Sect. 4. We summarize the paper and give conclusions in Sect. 5.

2. Related Works

We begin by introducing software behavior model approaches which are based on short system call sequences. Then, we discuss the development of automaton modeling as well as state transition graph in our method. In the end, we introduce current trend of modeling, especially in semantic issues.

A large number of researches based on short system

Manuscript received July 22, 2013.
Manuscript revised October 23, 2013.

[†]The authors are with Beijing University of Technology, China.
a) E-mail: laiyingxu@bjut.edu.cn

DOI: 10.1587/transinf.E97.D.488

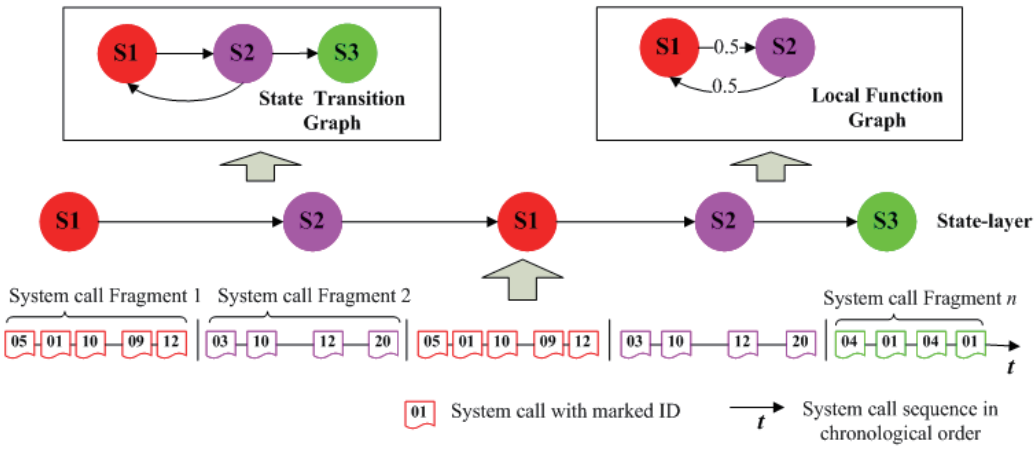


Fig. 1 The process of SB-TMS modeling.

call sequences, such as sequence enumeration, machine learning, were presented in prior works. In sequence enumeration, S. Forrest used fixed length system call sequence to describe normal software behavior in 1988. To overcome inflexibility of fixed length, Wepsi proposed a Variable-length model, called V-gram [4], which introduced variable-length patterns to find sequence patterns of system calls. HMM (Hide Markov Model) is the most representative approach in the field of machine learning. It describes local correlation of system calls by Markov process. In Ref. [13], the quantity of states in HMM was equal to that of system calls (nearly 284), thus lead to higher complexity. To solve the problem, Hoang [6] divided the whole system call sequences into several parts, each of which stood for a state, thus reduced the quantity of states. Yin [14] used feature vector, which is fix length short system call sequences, as states to build a HMM. These improvements reduced complexity to some extent, but it could not change the limitation of HMM algorithm itself. In addition, these reducing state quantity policies are assertive because system calls which express same function may be separated into two states.

In this paper, our state layer generation is similar to short sequence approach. We extract subsequences which correspond to latent functions hidden in the long-term system call sequence. The concept of subsequence has been introduced in [15], which is integral and non-decomposable. Unlike other general ways of pattern mining, our approach employed probability statistics theory to find sequence patterns will be discussed in Sect. 3.1.

Having extracted states, we expect our model describes long-term correlations as automaton does. Automaton exhaustively captures the program structure, and transverses branching or looping structures of the program, which enable us to identify long-term correlations. Sekar [7] used system call together with its program counter to build finite state automaton model (FSA). To resolve impossible path problem of FSA, Feng [8], [9] employed Vt-path model based on abstracting call stack information. Then, he proposed a corresponding static model, called VPStatic, which

Table 1 Main features and characteristics of modern modeling approaches.

Writers	Main features	Characteristics
Tian et al[16-20]	Software behavior and its effects on computer	Use all features to establish model and detect deviation on it. The approach has higher efficiency.
Pao et al [21-23]	IP address and time stamps	Identifies alerts correlation relations automatically by different approaches.
Yanget al [24]	Behavior trace, checkpoint and time-stamp	Propose concept of behavior semantic distance by matching behavior trace, checkpoint and time-stamp.
Fu et al[25]	System objects resolved from parameters of system calls	Define states based on system objects, thus each state has been assigned semantic information.

established automaton by analyzing source code. Contrasted to dynamic models, static models hold zero false alarming and without impossible path. Then static-dynamic hybrid approach, like Dyck [10] and HFA [11], becomes more accuracy. Its drawback is complicated calculation, difficult in practical application. The root of problem is that a single system call has no practical meaning. For the lack of semantic analysis, distribution of automaton is loose and cannot be merged in following process. Therefore, we extract states which are assigned semantic information instead of system calls to address the problem. And then we establish state graphs according to the executed software process to realize excellent description of long-term system calls. In this method, computational complexity is reduced based on our rules, by which low probability edges are reduced and similar states are merged.

The trend of current modeling research focuses on two hot issues. One is using different kinds of features (shown in Table 1) to describe software behaviors, and the other is how to assign semantic information in models. Tian [16]–[20] introduced both software behavior (system call) and its effect on computer environment, that is, what software behavior does to computer system (CPU occupancy, memory occu-

pancy and so on). Pao [21] employed alerts correlation relations to detect attacks from the Internet. Another tendency of modeling is endowing semantic information to software behaviors. Yang [24] proposed concept of behavior semantic distance by matching behavior trace, checkpoint scene and time-stamp. Fu [25] resolved system objects from parameters of system call and assigned semantic information which system objects contain. Although the model precision is improved, it is time-consuming to obtain detail information of executed software. Moreover, the definition of system objects is not very clear. Unlike their system object analysis, our model performs semantic analysis according to the extracted state layer. Each state corresponds to a latent functional fragment. In addition, it is considered to add semantic information in our state graphs at local and global levels. The local function graph established according to local functional template represents sensitive function, such as login. While state transition graph globally reproduces the overall process of execution, we will discuss this part in Sect. 3.2.

3. Software Behavior Trust Model Based on State-Layer

In this paper, we study software behavior under Windows operating system, which provide users with Native APIs to visit operating system. Consequently, 284 system calls in Windows Operating System Services Descriptor Table will be used to analyze software behavior.

SB-TMS modeling process is illustrated in Fig. 2. The establishment includes three main steps: First, system call sequences are intercepted in chronological order. Then, we extract a higher layer, called state-layer, from system calls sequence by pattern extracting. Last, we build state graphs with the help of pruning rules, and extract sensitivity functions.

Then, we introduce some basic concepts and notations in this paper as follows.

Definition 1 State-layer State is sequence pattern derived from system calls, denoted by $S = (sc_1, sc_2, sc_3, \ldots, sc_i)$, $i \in N$. All states form the state-layer set, denoted by $SL = (S_1, S_2, S_3, \ldots, S_i)$, $i \in N$.

Definition 2 State Graph (SG) SG can be classified into two categories: local function graph (LFG) and state transition graph (STG). STG and LFG consist of vertex (V), edge (E) and weight (W), denoted by $\boldsymbol{STG(V, E, W)}$, $\boldsymbol{LFG(V, E, W)}$ respectively, where V represents known state set, denoted by $V = \{v_1, v_2, \ldots, v_n\}$. E is directed edge set of V, which indicates relationship with each adjacent node, denoted by $E = \{< v_i, v_k >, \ldots, < v_k, v_n >\}$. W is a set of occurrence frequency of E, each edge in E corresponds to a weight, denoted by $W = \{w_{ij} | w_{ij} = c_{ij}/c\}$, c_{ij} is occurrence number of $< v_i, v_j >$, c is occurrence number of all edges.

Definition 3 Other notations Definition of weight deviation, In-degree, Out-degree and State distance is shown in Table 2.

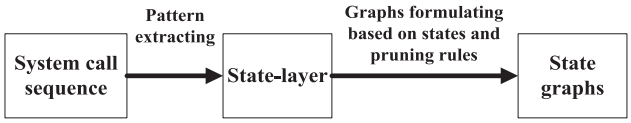


Fig. 2 Construction of SB-TMS.

Table 2 Other notations.

Definition	Description	Notation
Weight deviation	The deviation degree between two *LFG*s, we use Euclidean Distance to describe it	Δw
In-degree	For any v_i in set V, its in-degree is the sum of the corresponding weights of edges whose end vertex is v_i	in^+
Out-degree	For any v_i in set V, its out-degree is the sum of the corresponding weights of edges whose initial vertex is v_i	out^-
State distance	The distance between two different states, we use Euclidean distance to judge the similarity of two states	Δd

3.1 State-Layer Derivation

To derive representative sequence patterns as states to illustrate software behaviors, we present a novel technique to identify optimal sequence patterns using statistics theories. Our State-layer generation includes two parts: representative pattern selection and state identification based on HMM.

3.1.1 Representative Pattern Selection

Representative pattern selection includes three major steps. First, the long sequence is divided into many subsequences according to the correlation each of two system calls. Second, similar parts are classified to one group based on their probability density. Third, we choose optimal subsequences from each group as states. In this step, all subsequences will be matched with the original sequence to find the most optimal one.

Step1 Subsequence generation: Original system call sequence is a long sequence in chronological order. In order to extract optimal sequence pattern, we need to decompose the long sequence into many small fragments. Generally, system calls in a fragment expressed one function are related closely, while system calls in different function fragment share low correlation. In our model, converting original system call sequence into system call vectors, "1" or "0" indicates whether or not the system call is present in system call sequence. We use cross-correlation (shown in Eq. (1)) to measure correlation of two adjacent system call vectors. The lower is cross-correlation, the poorer is correlation between them.

$$\begin{aligned} Cov(X, Y) &= E\{[X - E(X)][Y - E(Y)]\} \\ &= \sum_{i=0}^{L-1} \frac{X_i \times Y_{i+1}}{L} - \sum_{i=0}^{L} \frac{X_i}{L} \times \frac{Y_i}{L} \end{aligned} \quad (1)$$

$$\begin{array}{c}
\{A\ B\ C\ A\ B\ C\ D\ D\ C\ A\ D\ D\ C\ A\ A\ B\ C\} \\
A = \{1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\} \\
B = \{0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\} \\
C = \{0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\} \\
D = \{0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\} \\
(A\ B\ C\ \ A\ B\ C),\ (D\ D\ C\ A),\ (D\ D\ C\ A),\ (A\ B\ C)
\end{array}$$

Step 1: Subsequences generation

	A	B	C	D
1	0.33	0.33	0.33	0.00
2	0.25	0.00	0.25	0.50
3	0.25	0.00	0.25	0.50
4	0.33	0.33	0.33	0.00

Group 1 $(A\ B\ C\ \ A\ B\ C)$, $(A\ B\ C)$

Group 2 $(D\ D\ C\ A)$, $(D\ D\ C\ A)$

Step 2: Similar subsequences grouping

Pattern 1 $(A\ B\ C\ \ A\ B\ C)$ H1 $= -0.16$

Pattern 2 $(D\ D\ C\ A)$ H2 $= -0.15$

Pattern 3 $(A\ B\ C)$ H3 $= -0.15$

State1 $(A\ B\ C)$; state2 $(D\ D\ C\ A)$

Step 3: Optimal subsequence (state) selection

Fig. 3 An original sequence (*ABCABCDDCADDCAABC*). Step 1: In order to decompose the long sequence into many small parts, we calculate $Cov(A, B)$, $Cov(B, C)$ ... by Eq. (1). As we can see $Cov(A, B) = 0.12$, $Cov(B, C) = 0.12$, $Cov(C, A) = 0.09$, $Cov(C, D) = -0.01$ ... therefore, we break at the place of C and D, A and D, A and A. In this way, the original sequence set is divided into four subsequences. Step 2: we calculate probability density matrix according to step 1, get a $4 * 4$ matrix. Then we calculate Euclidean distance each other rows. As a result, we get two groups. Step 3: the result of IE is: $H(ABCABC) = -0.16$, $H(ABC) = -0.15$, $H(DDCA) = -0.15$, therefore, we choose ABC as the optimal pattern for group1 and $DDCA$ for group 2.

where, X and Y are two adjacent system call vectors, $E(X)$ and $E(Y)$ are their expected value respectively, and L is the length of system call vector.

Consequently, the long sequence is broken at the place where two adjacent system calls' correlation is negative. Figure 3 (step1) uses an example to demonstrate the process of subsequence generation.

Step2 Similar subsequences grouping: After getting many subsequences, we use probability density matrix to classify similar subsequences into one group instead of traditional method of pattern mining (shown in Fig. 3 (step2)). We calculate probability density of system calls in each subsequence. A probability density matrix between subsequence and system call has been obtained. Then we calculate Euclidean distance each two rows. The shorter is distance, the more similar are the two subsequences. There is a direct correlation between the distance threshold and the quantity of groups. Groups grow as the threshold increases, so does the calculation complexity. For simplification, we set a threshold, which can obtain enough states to stand for running software completely, but also calculation is not too complicated. According to many experiments, the threshold is set as 25% of the maximum distance.

Compared with data mining, it is well suited to our problem for the following three main reasons: (a) Grouping is based on composition of subsequences, thus classification accuracy is guaranteed. (b) It does not change the order of original sequence, more suitable to describe functions. (c) Each group represents a series of similar sequences. Compared with rigid pattern mining, it reduces the quantity of states.

Step3 Optimal state selection: In this step, we choose an appropriate subsequence from each group as a state. The candidate subsequences must have the following two properties: (a) they must describe the group accurately and contains all inherent properties. (b) they can describe the original long sequence accurately or to the greatest extent. In order to describe properties of one group completely, we choose subsequences which near to the average density of each group as candidates. Then we calculate information entropy (IE) of these candidate subsequences in the original sequence by Eq. (2), the subsequences which has highest IE will be the most optimal sequence patterns. The process is shown in Fig. 3 (step 3).

$$H(X) = -P \log(P) \tag{2}$$

Here, P = occurrences number of subsequence in the original sequence / (original sequence length / length of candidate subsequences).

It still remains a problem of similar states, when the

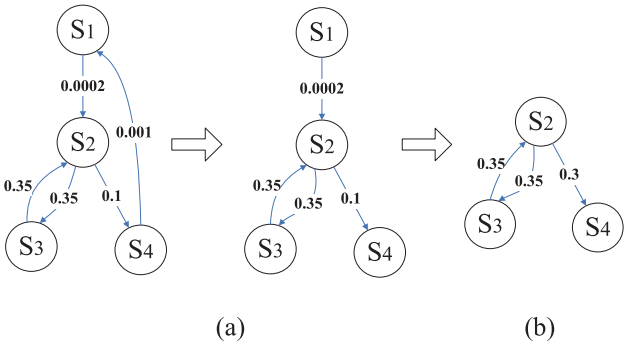


Fig. 4 (a) For any $e_i \in E$, if its corresponding weight $w_i < \sigma$, delete e_i. (b) For any $s_i \in V$, if $in^+ < \sigma$ and $out^- < \sigma$, delete v_i as well as edges.

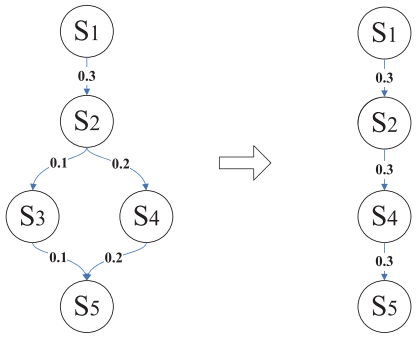


Fig. 5 For any v_i and $v_j \in V$ if $\Delta d_{ij} < \sigma$, and adjacent v is same, v_i and v_j will be merged. (a) Vertex: we choose the state which has a higher weight as the new vertex. For instance, we choose S_4 as the new state. (b) Edge: delete edges between v_i and v_j, if exist; keep edges between v_i, v_j and other vertexes and delete repeat edges. (c) Weight: base on the change of vertex and edge. For instance, $< v_2, v_3 >$, $< v_2, v_4 >$ is merged to $< v_2, v_4 >$ the new weight is $w_{24} = w_{23} + w_{24} = 0.1 + 0.2 = 0.3$.

distance of two states ($\Delta \boldsymbol{d}$) is very small. For instance, *ABC* and *ABCAB*, we cannot merge it without context. We will discuss it in Sect. 3.2.

3.1.2 State Sequence Identification Based on HMM

Next, we need to transfer the long system call sequence into state sequence according to optimal states we have already obtained. Simple sequence matching is rough and cannot tolerate even slight variations. In this paper, we use HMM as a bridge to connect system calls with states. The parameters of HMM include observation vector set, hidden state set and three distribution matrixes. We use three initialized distribution matrixes and each state to build a HMM by Baum-Welch algorithm at first. And then system call sequences are evaluated based on the given models by the forward backward algorithm. At last, the optimal state will be elected by comparison.

3.2 State Graph Generation

In order to build running software model, we will establish STG and LFG. STG is generated by traversing state sequence directly. We scan state sequence $(S_1, S_2, \ldots, S_n)$, if the state has been already in set V, go on scanning the next state, else it is put into set V. If a new edge appears, add the edge in set E and its weight is assigned an initial value, else adjust its weight. Go on scanning next state until the end.

We propose two pruning rules to simplify our graphs for high accuracy, and one separating rule for semantic analysis.

Rule1: delete edges and states with low parameters

For any edge in set E, if its corresponding weight does not over a threshold, we delete it to keep the graph clean and tidy, deleting process is shown in Fig. 4 (a). For any v in set V, neither its out-degree nor its in-degree over a threshold, we delete both the vertex and all the edges connecting to it, shown in Fig.4 (b). We set the threshold as 0.01, that means the edge or state will be deleted because it occurrences only 1 time or less while other edges or states occurrence 100 times. The edge or state with low occurrence frequency should be deleted. These edges or states may be from wrong identification by HMM or unstability of Windows operating system. We delete them for two reasons: one is that they are too instable to describe normal software behaviors, the other is that they cannot describe abnormal because they share nothing with virus and illegal intrusion.

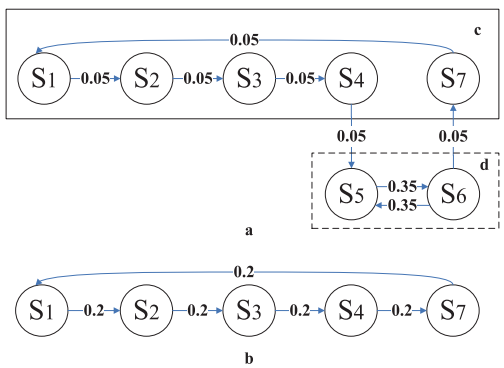


Fig. 6 The IM software contains three parts: connection, communication and disconnection. STG of software is (**a**). For the specific function of connection and disconnection, we establish local function graph (**b**), which is similar with part (**c**) in (**a**). In this way, STG is divided by two LFGs, (**c**) and (**d**). (**d**)corresponds to function of communication.

Rule2: merge equivalent states

For any two states in V, if their distance is less than a threshold and their adjacent states connected by edges are same, we will merge these states. The process of merging is shown in Fig. 5. As we have mentioned in Sect. 3.1, two states with a small distance are equivalent states, which may represent same function fragment, however, we are not sure. Therefore, we solve the problem in the process of establishing STG. If the similar states with same predecessor and successor states, we merge them to simplify graphs. According to many experiments, the threshold is set as 25% of the maximum distance.

Rule3: Separate LFG from STG based on specific function

In this rule, we propose a method of separating LFG

from STG based on specific software function. Each LFG represents one software function. The specific function should be clearly defined and explicitly executed, such as connecting server, user login or download. Clear semantic information of LFG will help to detect loop attack. For instance, the DoS/DDoS attack can be detected by LFG for it always do the same step in circle.

The process of establishing LFG is the same as that of STG. System calls are intercepted when software specific function is executed independently, and a LFG is established by step1 to step3. In Fig. 6, we illustrate an example of the IM software.

4. Experiments

In this section, first we give system architecture for anomaly detection (SB-TMSAD), and then present the detecting process by a case study of RSS reader software. Finally, we discuss detect performance of our model and give results.

4.1 System Architecture

Our framework is comprised of three main templates: State-layer Derivation Builder (SDB), Rule-based State Graphs Constructer (RSGC) and Graphs Detector (GD), as shown in Fig. 7. The dotted line denotes the detecting phase and the solid line denotes the training phase.

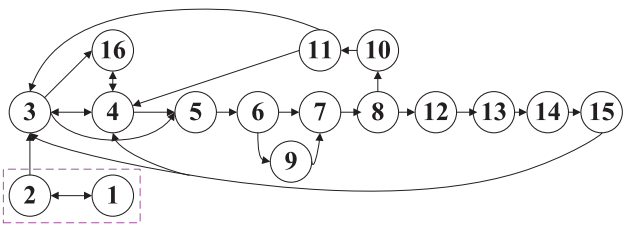
In training phase, SDB transfers intercepted system call sequence to state sequence by Step1 to Step3 in Sect. 3.1.1. Then the state sequence is sent to RSGC to build state graph template, which is according to the software execution process and Rule1 to Rule3. In detecting phase, the structure of detection system is similar to that of training system. System call sequences are collected in real time and transferred to state sequences according to standard state set generated by SDB. Instead of building SG template, they are detected by Graphs Detector (GD), which aims at detecting anomaly via SG template generated by RSGC. In GD, tested SG is compared with the SG template. Based on detection rules, untrusting behavior can be observed.

4.2 Case Study

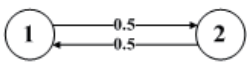
In this paper, we select a widely used RSS reader software, AgileReader, to validate our model. AgileReader allows a user to read news which is subscribed from the Internet. It is composed of user login dialog, reading interface, a database which records user information as well as behaviors and a user recommendation system.

4.2.1 AgileReader Modeling Based on SB-TMS

Given the characteristic of AgileReader, that is, database recording always come along with user reading, no clear boundary between them, AgileReader only can be described as login function and reading function. Finally, we build $STG(V, E)$ which performs the whole process of user using AgileReader and a $LFG(V, E, W)$ for login which is clearly defined. As shown in Fig. 8, the weight (W) should be included in STG, it is helpful to simplify STG by Rule1 and Rule2. In detecting, the tested software is compared with STG by Rule4. Only vertex (V), edge (E) are compared with STG template, the weight is not necessary, so in Figs. 8–10 and Fig. 13, the weight in STG is ignored.



(a) STG for RSS



(b) LFG (login graph) for RSS

Fig. 8 16 states are derived from system calls and STG (a) is built by SDB. The specific function of RSS is login. Therefore, LFG (b) is built and marked in STG (the dotted rectangle in (a)). In this way, STG is naturally comprised of two parts: login part and normal reading part.

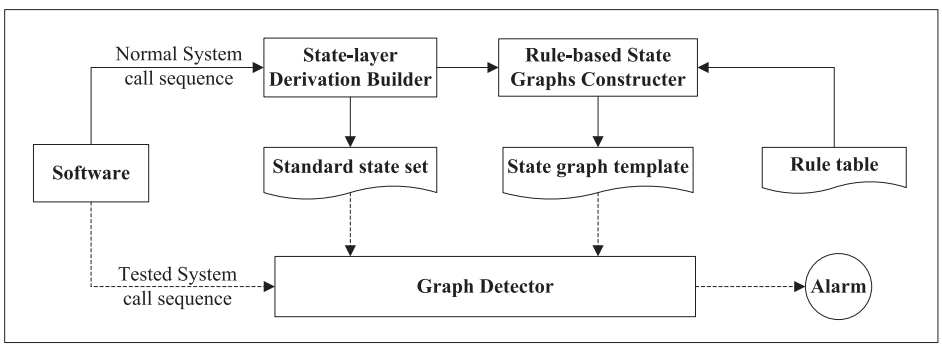


Fig. 7 System architecture of SB-TMS for anomaly detection (SB-TMSAD).

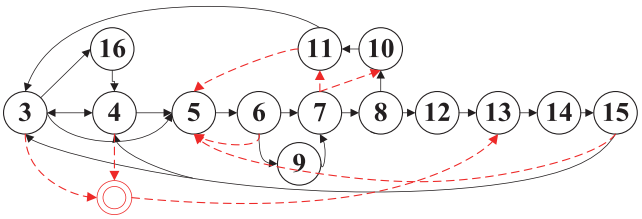


Fig. 9 Add a new function- refreshing, which is not included to RSS SG template, the unexpected behavior is detected by new states and seven new edges. The red hollow node represents new state while the dotted lines in red represent new edges.

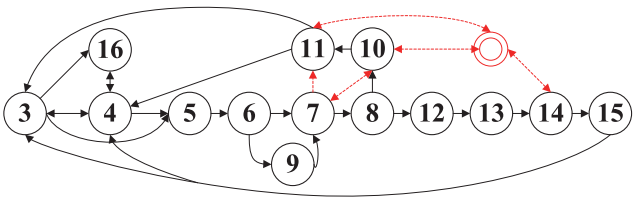


Fig. 10 Strcpy() buffer overflow attack. Software is executing injected code, this attack is detected by new states and five new edges.

4.2.2 AgileReader Detection Based on SB-TMSAD

In detecting, tested SG is contrasted with standard STG and LFG template by the following two rules. Rule4 is for matching states and edges of STG. Rule5 is for matching standard LFG and semantic rules.

Rule4: A tested $STG_t(V_t, E_t)$, $V_t = \{v_1, v_2, \ldots, v_n\}$, $E_t = \{e_1, e_2, \ldots e_n\}$, $n \in N$ is built. Suspicious behavior is detected by

(1) New states $\exists v_i \in V_t$ but $v_i \notin V$

(2) New edges $\exists e_i \in E_t$ but $e_i \notin E$

New behaviors which not include in the STG template are considered as un-trust behaviors, such as code injection attack. But un-trust behaviors may not be malicious. Because based on the definition of trust, any unknown behavior which is not accord with the software behavior template will be detected. The unknown behavior includes malicious behavior, new behaviors caused by software updating (some applications always are updated automatically) and etc. Rule4 cannot distinguish these unknown behaviors. Due to they violate the definition of trust, it should be found early to inform administrator. If the application's version is upgraded, the administrator can decide update the behavior template.

For example, we add a new operation in AgileReader: refresh the news we are reading. It is a new function which is not included in SG template. By Rule4, we can find an unexpected state and 8 edges in Fig. 9.

Code injection attack, which injects a piece of code into the running process firstly, and then try to use control flow transfer instructions (such as function calls) to execute the injected code. So the executing process must include new system call point or relevant instructions, which will result in new states or new edges, and can be detected based on

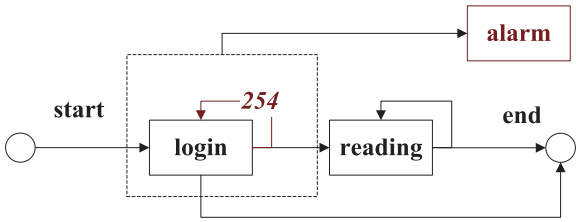


Fig. 11 Repeated login.

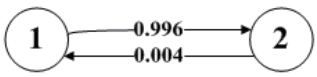


Fig. 12 Tested LFG.

Rule4. For example, the buffer over flow attack executed on AgileReader is shown in Fig. 10.

If no suspicious behaviors are detected by Rule4, the tested software should be detected by Rule5.

Rule5: Tested $LFG_t(V_t, E_t, W_t)$ are built, $V_t = \{v_1, v_2, \ldots, v_n\}$, $E_t = \{e_1, e_2, \ldots, e_n\}$, $W_t = \{w_1, w_2, \ldots, w_n\}$, $n \in N$. Suspicious behavior is detected by $\forall V_t, E_t \in LFG(V, E, W)$ and $\Delta W = W_t - W > \sigma$.

If vertexes and edges in tested $LFG_t(V_t, E_t, W_t)$ are same with $LFG(V, E, W)$ template, but the weight deviation ΔW is larger than threshold, we judge the tested software as untrusting. In this way, loop attack, such as DoS/DDoS attack and user's illegal input attack can be detected. Loop attack is a type of attack which use a plenty of inputs to use up system resources, such as establish a lot of network connections, user repeatedly login or maliciously download. The resource exhausting attack cause great harm because server cannot deal with legal requirements at that time. More importantly, it is hard to be found because it complies with normal states and edges in STG template. By Rule5, we can find this kind of abnormal.

As shown in Fig. 8, AgileReader contains two functions: login and reading. We used a fuzzy testing tool to login for 254 times in 2 minutes. A suspicious semantic relationship graph is shown in Fig. 11. The tested $LFG_t(V_t, E_t, W_t)$ is shown in Fig. 12. Vertexes and edges in tested $LFG_t(V_t, E_t, W_t)$ are the same as $LFG(V, E, W)$ template, but for $\Delta W = W_t - W > \sigma$, we can detect the un-trust behavior by Rule5.

4.3 Detect Performance and Results

While AgileReader is running for 30 minutes, 6821 system calls and 842 states are intercepted. For local function graph, login is repeated 50 times, 837 system calls and 112 states are obtained. State selection reduces the numbers of original data. As we can see in Table 3, the number of state is nearly one eighth of that of system calls, thus save a lot of computing time.

The detection results of our model are shown in Table 4. We use software security tool–Fortify, to find some vulnerabilities of AgileReader. The vulnerabilities are

Table 3 The number of system calls and corresponding states.

AgileReader	Our method		Original sequences	
	System calls number	States number	System calls number	State numbers
STG	284	16	6821	842
LFG	284	2	837	112

Table 4 SB-TMS model attacks detection results based on AgileReader.

vulnerability	Attack description	Result	Performance
New operation	Add refreshing function	√	Figure 9
New operation	Add open-file function	√	Figure A·1(a)
CWE ID 676	Strcpy() buffer overflow	√	Figure 10
CWE ID 253,690	LoadLibraryA() return null pointer overflow	√	Figure A·1(b)
CWE ID 120,131	OnBnClickedOk() backdoor dll injection	√	Figure A·1(c)
Repeated login	Login action appears for a long time	√	Figure 12

marked by Common Weakness Enumeration (CWE), which is community-developed dictionary of software weakness type. Then we take advantage of these security problems to create attacks and detect them by our trusting model. The detail performance is shown in appendix. We can see from Table 4, when the corresponding attacks occur, the new states and edges appear clearly. In Fig. 13, buffer over flow attack which will cause huge damage to the users is detected as well. Every subtle change of system calls will be detected by our model. For repeated login attack, we will detect it by our semantic relationship analysis. This kind of attack can only be detected from a semantic approach. As a result, the traditional way of modeling which using system call information directly, such as V-gram, FSA, HFA, even the modern models, SBTTM [19], SBO [25] are undetectable.

5. Conclusion

The proposed method made the following contributions: (a) It suggested a novel method of abstracting state-layer, which revealed the inner properties of software behaviors and reduced the computation complexity at the same time; (b) It established state graphs, which vividly described the process of executed software in terms of the pruning strategy; (c) It assigned semantic information to local function graphs. With semantic relationship analysis, wrong semantic attack, such as illegal input attack, would be found. The experimental results showed that our trusting model could successfully detect un-trust behavior in the tested software, anomaly behaviors and illegal input behaviors.

Acknowledgments

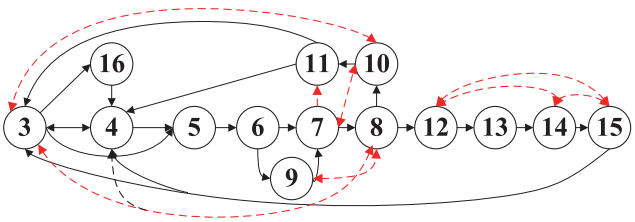
This research was supported by the National Natural Science Foundation of China (61001178), Scientific Research Common Program of Beijing Municipal Commission of Education (KM201210005024), Funding Project for Academic Human Resources Development in Institutions of Higher Learning under the Jurisdiction of Beijing Municipality (PHR201108016).

References

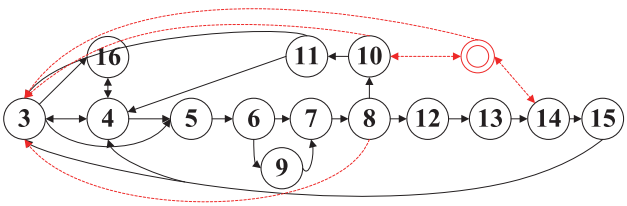
[1] Q. Yanwen, Software behaviouristics, Electronic Industry Publishers, Beijing, 2004.
[2] S. Forrest, "A sense of self for unix processes," Proc. IEEE Symposium on Security and Privacy, pp.120–128, 1996.
[3] S.A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," J. Computer Security, vol.6, pp.151–180, 1998.
[4] A. Wepsi, M. Dacier, and H. Debar, "Intrusion detection using variable-length audit trail patterns," Proc. International Workshop on Recent Advances in Intrusion Detection, pp.110–129, 2000.
[5] X.D. Hoang and J. Hu, "An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls," Proc. IEEE International Conference on Networks, pp.470–474, 2004.
[6] D. Endler, "Intrusion detection: applying machine learning to solar is audit data," Proc. Annual Computer Security Application Conference, pp.268–279, 1998.
[7] R. Sekar, M. Bendre, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," Proc. IEEE Symposium on Security and Privacy, pp.144–155, 2001.
[8] H.H. Feng, O.M. Kolesnikov, and P. Fogla, "Anomaly detection using call stack information," Proc. IEEE Symposium on Security and Privacy, pp.62–75, 2003.
[9] H.H. Feng, J.T. Giffin, and Y. Huang, "Formalizing sensitivity in static analysis for intrusion detection," Proc. IEEE Symposium on Security and Privacy, pp.194–208, 2004.
[10] J.T. Giffin, S. Jha, and B.P. Miller, "Efficient context sensitive intrusion detection," Proc. Symposium on Network and Distributed Systems Security, 2004.
[11] L. Wen and D. Yingxia, "Context sensitive host_based IDS using hybrid automaton," J. Software, vol.1, pp.138–151, 2009.
[12] D. Gao, M.R. Reiter, and D. Song, "Gray-box extraction of executiongraphs for anomaly detection," Proc. ACM Conference on Computer and Communications Security, pp.318–329, 2004.
[13] Y. Qingbo, Z. Rubo, and L. Xueyao, "Research on technology of intrusion detection based on linear prediction and Markov model," J. Computers, vol.5, pp.900–907, 2005.
[14] Y. Qingbo, Z. Rubo, and L. Xueyao, "Research on technology of intrusion detection based on dynamic Markov model," Electronic Sinica, vol.11, pp.1785–1788, 2004.
[15] Z. Yingying, "Theory and applications of sequence-specific mode," Thesis, University of Electronic Science and Technology of China, 2009.
[16] T. Junfeng, H. Jine, D. Ruizhong, and W. Yong, "Creditability evaluation model based on software behavior trace," J. Computer Research and Development, vol.7, pp.1514–1524, 2012.
[17] L. Yuling, D. Ruizhong, F. Jianlei, and T. Junfeng, "Trust model of software behaviors based on check point risk assessment," J. Xidian University, vol.39, pp.179–184, 2012.
[18] L. Zhen, T. Junfeng, and Z. Pengyuan, "A trustworthy behavior model for software monitoring ponit based on classification attributes," J. Electronics & Information Technology, vol.34, pp.1445–

1451, 2012.
[19] T. Junfeng, L. Zhen, and L. Yuling, "A design approach of trustworthy software and its trustworthiness evaluation," J. Computer Research and Development, vol.8, pp.1447–1454, 2011.
[20] M. Fengjun, Research on monitoring and analyzing of software behavior in open network environment, Ph.D. Thesis, Central South University, 2010.
[21] P. Hsingkuo and M. Chinghao, "An intrinsic graphical signature based on alert correlation analysis for intrusion detection," J. Information Science and Engineering, vol.28, pp.243–262, 2012.
[22] W. Wang and T.E. Daniels, "A graph based approach toward network forensics analysis," Proc. ACM Transactions on Information and System Security, vol.12, pp.1–33, 2008.
[23] S. Zhang, J. Li, X. Chen, and L. Fan, "Building network attack graph for alert causal correlation," Computers and Security, vol.27, pp.188–196, 2008.
[24] Y. Xiaohui, Researches on dynamic trusted theories and models of software behavior, Ph.D. Thesis, China University of Science and Technology, 2010.
[25] F. Jianming, T. Fen, W. Dan, and Z. Huanguo, "Software behavior model based on system objects," J. Software, vol.22, pp.2716–2728, 2011.

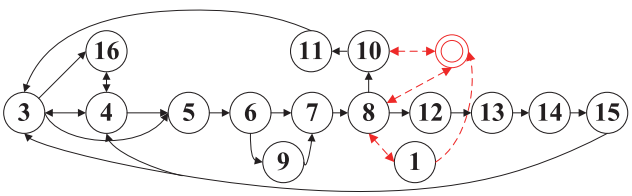
Appendix



(a)Add a new function- openfile, which is not included in RSS, the unexpected behavior is detected by eight new edges



(b) When LoadLibraryA() return null, it lead to program collapse for no return value check. We detect it by new states and edges



(c) OnBnClickedOk() function holds a hidden danger, that is, it may write to memory out of boundary and then execute malware. We inject backdoor dll through it and detect new state and edges

Fig. A· 1 The performance of states when anomaly occurs. ((a) (b) and (c) correspond to the anomaly in Table 4). The red hollow node represents new state while the dotted lines in red represent new edges.

Yingxu Lai received Ph.D from Chinese Academy of Sciences in 2003, M.S. degree from Beijing University of Chemical and Technology in 1997, B.S. degree from Shenyang Institute of Chemical and Technology in 1994. Now she is an associate professor at the College of Computer Science, Beijing University of Technology. Her research interest covers information network security and trusted computing.

Wenwen Zhang received her M.S. degree from Beijing University of Technology in 2013 and B.S. degree from Nanjing University of Posts and Telecommunications in 2010. Her research interests include information security, trusted computing and software behavior analysis.

Zhen Yang is an associate professor at the College of Computer Science, Beijing University of Technology. His research interest covers informationcontent security, public opinion analysis, and trusted computing.