Software Abnormal Behavior Detection Based on Function Semantic Tree

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

SUMMARY Current software behavior models lack the ability to conduct semantic analysis. We propose a new model to detect abnormal behaviors based on a function semantic tree. First, a software behavior model in terms of state graph and software function is developed. Next, anomaly detection based on the model is conducted in two main steps: calculating deviation density of suspicious behaviors by comparison with state graph and detecting function sequence by function semantic rules. Deviation density can well detect control flow attacks by a deviation factor and a period division. In addition, with the help of semantic analysis, function semantic rules can accurately detect application layer attacks that fail in traditional approaches. Finally, a case study of RSS software illustrates how our approach works. Case study and a contrast experiment have shown that our model has strong expressivity and detection ability, which outperforms traditional behavior models.

key words: software behavior, system call, state graph, semantic analysis, deviation density, function semantic rules

1. Introduction

Applications such as instant messengers, Web browsers, and social media software have been the main tools for people to obtain information. However, software defects and vulnerabilities are inevitable, such as collapse, failure, denial of service, and viruses. More and more attention has been given to software security and trust research has been given. So software suspicious behavior detection has become a rapid developing research direction.

Current software behavior models are mainly based on the order or frequency of system calls. These models can detect abnormal situations where the sequence or parameters of system calls are changed directly. Nevertheless, some viruses and attacks, such as DoS (Denial of Service), malicious manipulation, and phishing, are often smarter, in which each step of execution is legal, and thus they can be detected by function semantic analysis only. Therefore, it is important to build models of software behaviors based on multi-step function semantics.

In order to perform semantic analysis, software behaviors can be interpreted as a series of actions executing on a virtual tree. A behavior can be understood as a subject of traversing, selecting, entering into and leaving the behavior tree. Each node of the tree is assigned certain semantic meanings, such as behavior state, type, input and output [1]. Thus, each step of an action is assigned certain semantic meanings by the node that it goes through. As a result, the model of software behavior is developed from the system layer characteristics to the function layer semantic analysis. It thus fills the gap of semantic meaning lacking at the semantic layer.

In this paper, the software behavior model, as well as the two-layer anomaly detection model is developed. One layer is based on Software Behavior, and the other is based on Software Function tree, shown in Fig. 1. The dotted lines denote the detecting phase, and the solid lines denote the training phase. The detection model is abbreviated as SB-SF for Software Behavior-Software Function. In the training step, the system call sequence is transformed into a state sequence (which stands for software behavior) and then into a function sequence (which stands for software function) with the help of software function division. Hence, the behavior model is established in terms of state graph and software function. In the detecting step, at the layer SB, which aims at state sequences, we calculate the deviation density of suspicious behaviors to discover any anomaly. If no anomaly is found, the related behaviors are sent to the SF layer, which uses function semantic rules to detect anomaly against semantic rules. With the help of the two-layer detection, particularly the semantic analysis, SB-SF has the ability to detect both control flow attacks and application layer attacks.

The reminder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the software behavior model based on state graph and software function. Section 4 defines the two-layer detection model, which contains deviation density calculation and function semantic rules. Section 5 presents a case study and analysis to validate our work. We summarize the paper and give conclusions in Sect. 6.

2. Related Works

To model software behaviors, system calls are typically adopted, as they are the interface of accessing system resources for application software, and they are seldom changed in different operating systems and different editions. Current studies are well established in software behavior models for recognizing normal behaviors and for detecting anomaly based on accumulating deviations. Existing approaches of software abnormal behavior detection can be classified into three categories: frequency, control flow, and data flow methods.
System call frequency can be used to identify abnormal behaviors, because frequency of system call is stable when software runs normally, and it changes abruptly when anomaly occurs. For example, T-Stide model [2] recorded frequency of short system call sequences in training, and then estimated those behaviors that have frequency less than 0.001% as abnormal. A vector was used to represent software behaviors by employing text classification technology [3]. The vector’s dimensionality was the number of system calls. Frequency of each item in the vector was to build a KNN (K Nearest Neighbour) classifier, which could be used for detection. This detection approach relied on the frequency stability of system call. It was simple and practical, but maybe failed in more concealed anomaly detection.

The approaches based on control flow are to investigate temporal relations among system calls and to employ various kinds of technologies to simulate or approximate the actual control flow of the programs. These technologies include static analysis, machine learning and automaton.

The static analysis approach detects anomaly by comparing the paths in the detected executable file with normal execution paths established by analyzing source code. Currently, the most widely used static detection model stemmed from call graph model and abstract stack model [4]. It exhaustively captured every piece of functions in the source code, and it has almost zero false alarm. However, static analysis shares a number of problems such as: (i) It is complex for considering all the structures of the program; (ii) It is impossible to analyze library functions as well as dynamic link library; (iii) Source code is not always available for security and proprietary reasons.

Detection based on machine learning extracts normal software behavior patterns and then detects anomaly based on the patterns. For instance, Hidden Markov Model (HMM) was employed to detect anomaly by output state sequence [5]. It has a high accuracy as well as a high complexity. Behavior forecasting was introduced for detection with the help of Bayesian network [6]. Data mining technology was employed to analyze the association of system call sequence and to derive the most representative patterns as behavior expressions [7], which decreased significantly computational expense.

Automaton is another widely-used method for analyzing behaviors. It exhaustively captures the structure of the program, transverses branches, and iterates loops of executables. The approach particularly is useful for identifying long-term correlations. Sekar et al. [8] used system calls together with its program counter to build a finite state automaton (FSA). To resolve the impossible path problem of FSA, the Vt-path model [9] was proposed based on abstracting call stack information.

Data flow detection approach analyzes system call parameters and returns values of normal operating software behaviors as a basis for detecting anomaly. It has been successfully studied to model the parameter’s characteristics of a single system call to produce the rules of behaviors [10]–[13]. A more accurate parameters detection model [14] was proposed based on the idea of clustering. Li et al. [15] also introduced data mining to obtain streamlined sequence patterns before modeling. Data flow detection aims at assisting control flow detection and improving detection ability. It compensates for the weakness of control flow analysis, but still cannot detect the strongly concealed abnormal behaviors.

The trend of the current anomaly detection research is shifting to semantic analysis, which tries to combine the actual intention with the behaviors. An important concept of behavior semantic distance [16] was proposed by matching the behavior function trace, checkpoint scene, and timestamp. System objects were resolved from parameters of system calls and assigned semantic information that the system objects contain [17]. Event, that indicated real user behaviors, was employed to discover invariant constraint rules to restrain behaviors [18]. Since semantics has its own inner logic, which is related to users, it is quite possible to suffer certain subtle attacks from the application layer. As a result, semantic analysis is gradually becoming the hot spot in the current field of software behavior and network behavior.

3. SB-SF Modelling

In this section, we first introduce the basic concepts and notations, and then we present the process of modelling SB-SF.

3.1 Definition

We start with introducing the basic concepts and notations used in this paper.

**Definition 1 State-layer.** A state is a sequence pattern derived from system calls, denoted by $S = (sc_1, sc_2, \ldots, sc_i), i \in N$. All states form the state-layer set, denoted by $SL = (S_1, S_2, \ldots, S_i), i \in N$. 
Definition 2 State Graph (SG). Two categories of SG are defined: local function graph (LFG) and state transition graph (STG). LFG consists of vertex set \( V \), edge set \( E \) and weight set \( W \), denoted by \( LFG(V,E,W) \). For simplification, function graph does not contain weight during the detection step and is denoted by \( LFG(V,E) \). In the definition, \( V \) represents the 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, and is denoted by \( E = \{<v_1,v_2>,\ldots,<v_i,v_j>\} \). \( W \) is the set of occurrence frequencies of \( E \), which are called weights and denoted by \( W = \{w_{ij}\} \), where \( c_{ij} \) is the occurrence number of \( <v_i,v_j> \), and \( c \) is the occurrence number of all edges. The STG consists of vertex set \( V \) and edge set \( E \) and is denoted by \( STG(V,E) \).

Definition 3 Function semantic set FS-Set. FS-Set is an eight-tuple \( (FS_{Id}, FS_{State}, FS_{Input}, FS_{Output}, FS_{Attribute}, FS_{Context}, FS_{Dual}, FS_{Appearance}) \).

\( FS_{Id} \) is an identification of a software function. It is unique in the FS-Set. \( FS_{State} \) describes the state of the function behavior which maybe normal, abnormal, normal termination or abnormal termination. \( FS_{Input} \) is the input of the software function. \( FS_{Output} \) is the output of the software function. \( FS_{Attribute} \) is the attribute of the software function, which is described by \( LFG(V,E,W) \) in this paper. \( FS_{Context} \) is the context of the software. It includes \( FS_{Id} \) and \( FS_{State} \) of the current software function’s precedent and succeed. \( FS_{Dual} \) describes the dual functions in the software, for instance, the dual of opening a file is closing a file. \( FS_{Appearance} \) describes that this function can appear frequently or continuously or not. For instance, reading function can continuously appear in a reading software, but the login function is not allowed.

Definition 4 Function semantic tree (FS-Tree). It describes the semantics of the software. Each software application is composed of a plurality of functions, for instance, a simple communication application consists of four functions: connection, authentication, communication and disconnection. Therefore, FS-Tree describes a kind of tree-like relations among these functions.

To obtain function semantic tree, there are two possible ways: establishing the corresponding tree in the design step and collecting software behavior function sequence during the execution of the application software. The more completely the software is implemented, the more accurate is the FS-Tree. As the source code of software application is usually not available, we use the latter method to establish FS-Tree.

Definition 5 SB-SF model. It is a four-tuple set (system call: SC, state graph: SG, function semantic set: FS-Set, function semantic tree: FS-Tree).

Definition 6 Function: A software function is a single task, such as log in, log off and so on.

3.2 Process of SB-SF Modelling

The process of modelling SB-SF includes two steps: (1) The state sequence is constructed from the system call sequence, and STG and LFG are established with the help of pruning rules. (2) FS-Set and FS-Tree are constructed according to the definitions in Sect. 3.1. Each step consists of a series of steps.

Step 1 Build State Graph

1. Transform the system call sequence into the state sequence by pattern mining and state identification [19].
2. Establish a state graph based on the state sequence.
3. Simplify the state graph by deleting edges with small weight and merging the similar states to generate STG.
4. Reestablish LFG according to the function division of software in the function layer.

Step 2 Build Function Layer

1. Divide software into function based on user behaviors.
2. Transfer the state sequence into the function sequence by boundary states.
3. Establish FS-Set according to definition 3.
4. Establish FS-Tree according to the semantic logic of functions as well as the function sequence.

4. Deviation Density and Semantic Rules

This section describes how we detect software behaviors based on the established model. The detection process consists of a series of steps: calculate the deviation density of suspicious behaviors for state layer detection, detect function sequence, apply function semantic rules to detect any anomaly or violation, and finally, analyze the detection mechanism and ability from the viewpoint of an attacker.

4.1 Deviation Density Decision Based on STG

Deviation accumulation detection implies several difficulties. If the accumulating region is long and the historical accumulation of the detected behaviors shows little abnormal behaviors, the abnormal behaviors in the local area is sparse and can hardly be detected. Moreover, if deviation increases constantly over time, the threshold between normal and abnormal behaviors becomes difficult to determine. To address these problems, we propose a detection approach based on deviation density. This approach considers the changing trend of deviation factors in different situations as well as accumulating deviations in different regions.

4.1.1 Weight Assignment of Suspicious States and Edges

In this paper, suspicious behaviors in state sequence are those unexpected states and paths in STG. The suspicious paths are categorized into four types, as shown in Fig. 2, where, \( S_0 \) represents suspicious state, \( S_N \) represents normal state; \( P_{NO} \) is the path from normal state to anomaly state; \( P_{ON} \) is the path from anomaly state to normal state; \( P_{NN} \) and \( P_{NN} \) is the transition route among anomaly states; \( P_{NN} \) and \( P_{NN} \) is path among normal states.
As intrusion and virus tend to make severe damage to system or replicate themselves to spread in a short period of time, abnormal behaviors are usually local and sudden. In other words, if suspicious behaviors continuously appear, the weight should be increased. If suspicious behaviors seldom appear, the weight should be decreased.

The assignment of suspicious behaviors is based on the situations discussed above. As can be seen from Fig. 2, both $P_{NQ}$ and $P_{QN}$ connect suspicious state and normal state, and thus the weight will be calculated jointly, denoted by $PW_{QNO}$. The specific assignments are shown as follows.

$$SW_Q = SW_0 + 1, \quad \text{if} \quad S_{ucc} = S_Q \tag{1}$$
$$PW_{QQ} = \begin{cases} 1 & \text{Initial value} \\ PW_{QQ} + 1 & P_{ucc} = P_{QQ} \text{ or } P_{ucc} = P_{NQ} \\ PW_{QQ} - 1 & P_{ucc} = P_{QN} \text{ and } PW_{QQ} > 1 \end{cases} \tag{2}$$
$$PW_{QNO} = \begin{cases} 1 & \text{Initial value} \\ PW_{QNO} + 1 & P_{ucc} = P_{NN} \text{ or } P_{QN} \\ PW_{QNO} - 1 & P_{ucc} = P_{NQ} \text{ and } PW_{QNO} > 1 \end{cases} \tag{3}$$

Where, $SW_Q$ is the weight of $S_Q$, and $PW_{QQ}$ is the weight of $P_{QQ}$, $P_{ucc}$ is the successor path of the current path, and $S_{ucc}$ is the successor state of the current state.

Accumulation of deviation is classified into two categories: state deviation accumulation, denoted by $diff_S$, and path deviation accumulation, denoted by $diff_P$. The equations are as follows:

$$diff_S = SW_Q \tag{4}$$
$$diff_P = PW_{QNO} + PW_{QQ} \tag{5}$$

4.1.2 Division of State Sequence Period

Interval method is an effective detection method for local anomaly, which cumulates deviation in the selected interval. Unfortunately, there are no known good ways of determining the interval. The usual way of arbitrary division may put the anomaly behaviors into different sections. To make the interval division clear and reasonable, we extend the interval method by dividing state sequence into periods. Firstly, we divide the long sequence into stationary period (StaP) and suspicious period (SusP). StaP is the period in which no or little suspicious behaviors are observed, while SusP is the period in which are observed. The challenge is to find the threshold that can differentiate the two types of periods. Based on many experiments, we find an empirical threshold weight is 25% of the maximum weight.

SusP is further divided into general suspicious period (GenP) and dangerous period (DanP), depending on the increasing weights of $S_Q$ and $P_{QQ}$. The periods are associated with the abnormal level of detected behaviors. The reasons that we define these different periods are: (i) The way of distinguishing StaP and SusP will not mark anomaly behavior state sequences that belong to one attack. (ii) GenP describes normal behaviors that can tolerate small errors, and it will reduce false positive rate. (iii) DanP is deemed to separate the obvious abnormal to simplify calculations.

Having obtained different periods based on abnormal level of detected behaviors, the next task is to select the clear division standard. In this paper, we introduce two evaluating indicators, Instantaneous Slope (IS) and Cumulative Slope (CS) is defined as follow:

$$IS_{PW_{QQ}} = \frac{d(PW_{QNO})}{dt} \tag{6}$$
$$IS_{PW_{QQ}} = \frac{d(PW_{QQ})}{dt} \tag{7}$$
$$IS_{S_Q} = \frac{d(S_Q)}{dt} \tag{8}$$
$$CS = \sum_{i=0}^{n} diff_S + \sum_{i=0}^{n} diff_P \tag{9}$$

ISs of $PW_{QNO}$, $PW_{QQ}$ and $S_Q$ are defined in Eq. (6)–(8), which can better describe some sudden abnormal behaviors. CS defined in Eq. (9) is the average abnormal for a period of time, which can be a stable data to depict the whole situation of a period of time. The period division is defined as follows: stationary is the period when both IS and CS are under the given threshold. According to the definition of DanP, dangerous period is extracted from suspicious period, which means that IS is rising and CS is over the threshold. The rest of the period is general suspicious period. That is the set of StaP is $\{StaP | IS_{PW_{QQ}} < \sigma_1, IS_{PW_{QQ}} < \sigma_1, IS_{S_Q} < \sigma_1, CS < \sigma_2\}$ and the set of DanP is $\{DanP | IS_{PW_{QQ}} > IS_{S_Q}, IS_{PW_{QQ}} \uparrow, CS > \sigma_3\}$. $\sigma_1$, $\sigma_2$ and $\sigma_3$ is threshold respectively.

Finally, the rate of deviations among the overall behaviors, which is called deviation density in this paper, is defined in Eq. (10).

$$diff_{density} = \frac{\sum_{i=0}^{n} diff_S}{C_S} + \frac{\sum_{i=0}^{n} diff_P}{C_P} \tag{10}$$

where, $C_P$ is the total number of paths, and $C_S$ is the total number of states.

It is obvious that stationary period is normal while dangerous period is abnormal, and thus we can focus on calculating the deviation density of suspicious period between the two clear cases. After the calculation, we judge if the states are abnormal based on threshold and the adjacent periods.

4.1.3 Algorithm Flow of Deviation Density

The process of anomaly detection based on deviation density discussed in the previous section is formalized into an
algorithm shown in Fig. 3. The key step of the algorithm is described as follows.

i. **IS and CS calculation.** Calculate IS and CS in terms of the changeable weight of suspicious behaviors.

ii. **Time division.** Find the StaP and SusP based on IS and CS; Divide SusP into DanP and GenP; StaP is judged as normal while DanP is judged as abnormal.

iii. **Deviation density calculation.** Calculate density deviation of GenP using Eq. (10); if the value is under the threshold and the adjacent periods are not in dangerous periods, it is judged as normal, otherwise as abnormal.

### 4.2 Software Function Behavior Semantic Analysis

To detect application layer attacks of software, function semantic rules are proposed based on FS-Set and FS-Tree. These rules are developed from the view of software behavior’s macro characteristics, which are behavior trustworthiness, effectiveness, duality and timeliness.

**Rule 1 Trustworthiness of function behavior**

Behavior trust refers to that the subject reflected by behavior history is legal, permissible, and within the scope of the statistical characteristics. In other words, if an entity’s behavior is always in the expected way, it is trustworthy.

Since the goals, results of behaviors, as well as the basis for evaluating behaviors are all rely on changing of states, we use function states of software, which is FS_State in FS-Set, to detect trustworthiness of software. If a function state is normal during the execution and normal at termination, the behaviors can be considered trustworthy.

Normal operations of function states indicate that the state sequence, which represents one function of software, is within the expected range. To ensure that all the states at function layer are normal, deviation density has been calculated on the first layer of detection. Hence, we believe that states and paths that have a chance to enter into the second layer detection are performing normal operations.

Normal termination of function states aims at guaranteeing the integrity of each function. To identify it, LFG (local function graph) in the corresponding FS-Set of each function is introduced as a template, that is, for LF(G(V, E)), state sequence should traverse it at least once before going to the next function.

We use Fun_Credibility definition below to describe trustworthiness of function behavior.

**Definition 7**

\[ S = \{s_1, s_2, \ldots, s_n\} \]: as state sequence;

\[ F = \{f_1, f_2, \ldots, f_n\} \]: as function sequence;

\[ R = \{trust, untrust\} \]: as results:
\[
\text{Fun.Credibility} = S^d \times F^l
\]

where, \( S^d = \{d | d : S \to R \} \), \( d \) is FS.State, FS_State = \{normal, abnormal, normal termination, abnormal termination\}; \( F^l = \{l | l : F \to R \} \), \( l \) represents LFG, the credibility of each function is determined by the composition of LFG.

**Rule 2 Validation of function behavior**

Behavior effectiveness refers to whether the change of object state is consistent with what is expected, that is, each function is executed properly and achieved the expected goal. In other words, this rule can be interpreted as the satisfaction of users’ output results. For instance, a user clicks “save” button and the result is saving the file correctly.

However, it is difficult to distinguish the effectiveness of software, as there is no specific standard to verify it. Given the order of function sequence is correct if they are valid, we use FS-Tree, which represents function flow of software, to judge the validity of function behavior. This judgment can be realized by querying FS-Tree. Trajectories of software function behavior should be conformed to FS-Tree and any of trajectories should go along a branch of FS-Tree from the root node to a leaf node.

All the normal function sequence should be in accord with this tree. Fun.Effectiveness is used to describe validity of functional behavior, the formula is as follows:

\[
\text{Fun.Effectiveness} = F^V
\]

where, \( F^V = \{V | V : F \to R \} \), \( V \) is tested function sequence, \( V = \{f_{start}, f_1, \ldots, f_{end}\} \subseteq \text{FS.Tree} \), \( f_{start} \) is start function, \( f_{end} \) is end function.

The rule accuracy depends on how much normal behaviour were collected. In the learning phase, the more completely the software is implemented, the more accurate is the rule. In the test phase, we select a widely used RSS reader software, AgileReader, to evaluate and validate our model. AgileReader allows a user to subscribe and read news through a series of steps: user login, reading interface, and accessing a database that stores user profile and user behaviors, which are used for generating user-specific recommendations. In our experiment, we have recorded the activities of running AgileReader for 3 hours, and collected more than 300,000 system calls, which are transformed into about 4000 states.

**Rule 3 Duality of function behavior**

In category theory, the dual category are widely used, where a category \( C \) is defined by \( C = (C^{\text{op}})^{\text{op}} \). Dual functions are abounded in the actual function behaviors world. For instance, root node and the leaf node is in certain sense of duality. In this paper, duality is even more narrowly defined, like open vs close, connect vs disconnect, which depends on the specific circumstances in the specific software.

Dual function should appear in pairs, or at least in a single cycle. Otherwise, there might be an exception. For instance, a user opens a file, writes the file, and then closes the file. Write operation can be executed repeatedly, but open and close must be executed in pair. It is not acceptable to open twice corresponding to closing once only.

As a result, any function that has a dual function must be followed by the corresponding dual function, as defined below.

\[
\text{Fun.Dual} = F^C
\]

where, \( F^C = \{C | C : F \to R \} \), \( C = \{f_1, f_2, \ldots, f_n\} \), \( f \) represents a function that has duality, and \( f^{-1} \) represents dual function of function \( f \).

In detection phase, if only there is not duality of some a function in \( C \), the behaviour will be judged as abnormal. For example, open and close a file must be executed in pair. It is not acceptable opening twice corresponding to closing once only. If \( C = \{open, close; open, close; \ldots\} \), the behaviour is normal. If \( C = \{open, open, open; \ldots\} \), the behaviour will be judged as abnormal.

**Rule 4 Timeliness of function behavior**

Timeliness of behavior refers to a function or conversion with its dual function cannot stay for a long time. The rules are specifically defined for the function that is “no” in FS.Appearance. For instance, if someone continuously tries to login, it is in all probability that this person does not have the password and tries to crack. The behaviour will be judged as abnormal. If a user continuously executes the operation of opening and closing without performing any other operations, it is necessary to consider whether these are malicious behaviors.

When a function whose FS.Appearance is “no”, we must pay attention to the consecutive times of occurrences. Fun.Timeliness is used to describe the timeliness of function behaviors:

\[
\text{Fun.Timeliness} = F^T
\]

where, \( F^T = \{T | T : F \to R \} \), \( T = \{f_k, f_k^{-1}, \ldots, f_n, f_n^{-1}\} \) \( \&\) \( \{f_k f_{k+1}, f_k f_{k+1}^{-1}, \ldots, f_n f_{n+1}^{-1}\} \) \( (n < \sigma) \), \( f \) represents a function that cannot stay for a long time, and \( f^{-1} \) represents dual function of function \( f \).

4.3 Detection Ability Analysis

In this section, we mainly discuss the detection ability of our model. First, we introduce the detection mechanism of SB-SF, and then we analyze the detection ability from the view of attackers.

4.3.1 Two Layer Detection Mechanism

SB-SF is a behavior model that contains two layers of detection mechanisms. One mechanism is based on state sequence, and the other is based on function sequence. The first layer of detection aims at identifying suspicious states and paths by calculating their deviation densities. It aims
at picking up abnormal behaviors that result in suspicious states and edges. The second layer of detection is based on function semantic rules. It detects application layer attacks that are against semantic rules. The function layer detection is on the basis of the state layer detection. If abnormal behaviors are detected at the lower layer, there is no need to do the upper layer detection at all.

4.3.2 Code Injection Attack

Code injection attack is such an attack in which an attacker takes advantages of the process vulnerability and injects a piece of executable binary code (shellcode) into its space, which then modifies the control flow of the process and gains the control privilege. In order to achieve the expected effect of the attacker, shellcode must contain certain system call points or control transfer instructions. Therefore, this kind of attacks will inevitably lead to changes of system call sequence.

Unfortunately, attackers have developed innovative ways of defeating the detection mechanisms. For example, one possible way is to launch a code-reuse attack, where in existing code is re-purposed to a malicious end. The simplest and most common form of this attack is the return-into-libc technique. This technique allows the attacker to execute an arbitrary sequence of libc functions, with a common example being a call to system("/bin/sh") to launch a shell. While it is powerful, it does not allow arbitrary computation within the context of the exploited application. The new type of attacks use Return-Oriented Programming (ROP). ROP allows for Turing-complete behavior in the target program without the need for injecting any code. However, this attack still builds and chains function gadgets, with each performing certain primitive operations. System call sequences are inevitably changed.

In SB-SF, changing of system call sequence will result in the appearance of the suspicious status and path, and thus this kind of abnormal will be detected by deviation density detection of SB-SF in the first detection layer.

4.3.3 DoS/DDoS Attack

DoS/DDoS attacks can be classified into two types according to the mechanisms used: (i) resource exhaustion attacks and (ii) attack based on the anomaly. In the first type of attacks, the attackers try to use a large number of system resources, such as establish a large number of network connections, and force the process to handle large files and so on. The second type of attacks depends on the specific software defects, such as an integer overflow, which may cause system crash or an endless loop. This kind of attacks generally needs to change the control flow of the process to achieve the purpose. If the first type of attacks cannot trigger an exception of path or parameters, it cannot be detected by most of the existing models. In our approach, rule 4 in SB-SF specially analyzes loop attacks and is able to detect such abnormal behavior. If the second type of attacks changes the control flow of the process, the SB-SF will detect it in the detection layer first. If no change in control flow, it can be detected by rule 4 in the function detection layer.

4.3.4 Malicious Action Attack

Malicious action attack is a type of attacks in which users try to achieve a particular purpose, for example, stealing data, destroying data and programs, and installing and operating Trojan software illegally. Such attacks tend to hide deeply because each step of the operation is legal, but the purpose is malicious. For example, login is legal, but repeatedly login and logout is malicious. Taking payment from a client is a legal operation, but it is illegal if the same operation is performed twice. This kind of attacks cannot be detected by the traditional model, because these models concern the system layer behaviors without any semantic analysis. If each step of an attack is legal, it will bypass the detection based on control flow or data flow. In our approach, the SB-SF is based on function semantic analysis. We define four detection rules at the function layer on credibility, integrity, effectiveness, and timeliness of behaviors. Our model can find the semantic layer attacks that traditional model cannot find. The detail of the detection process will be presented in the next section though a case study.

5. Case Study and Analysis

This case study elaborates the process of attack detection of RSS reader software, studies the detection performance of our model, and presents the experiment results.

5.1 AgileReader Modeling

In this section, we select a widely used RSS reader software, AgileReader, to evaluate and validate our model. AgileReader allows a user to subscribe and read news through a few steps: user login, reading interface, and accessing a database that stores user profile and user behaviors, which are used for generating user-specific recommendations.

In this paper, we study software behavior on Windows operating system, which provide users with Native APIs to

![State transition graph STG and local function graph LFG](image)
visit operating system. Consequently, 284 system calls in Windows Operating System Services Descriptor Table will be used to analyze software behavior.

We first build the STG \((V, E)\), as shown in Fig. 4 (a), which models the entire process of user's using AgileReader, including 16 states of different system calls. The sensitive function of RSS reading is login. Therefore, we build a LFG in Fig. 4 (b), which is also marked it in STG in the dotted box in Fig. 4 (a). In this way, the STG is naturally comprised of two parts: login part and normal reading part.

Considering the characteristics of AgileReader, the database captures the user behaviors simultaneously with users reading. But there is no clear boundary between the behaviors. We extract the functions of software in terms of user behaviors. AgileReader can be described as a sequence of functions of login, reading, and user recommending. By the end of the process, the function semantic sets in Table 1 will be generated, and function semantic tree in Fig. 5 will be created.

### 5.2 AgileReader Anomaly Detection Based on SB-SF

In this section, the process of anomaly detection based on SB-SF is structured in two major steps. First, we contrast STG of AgileReader, detect state sequence according to the deviation. Second, we contrast FS-Set and FS-Tree, and finally, we detect function sequence with the help of function semantic rules.

**Step 1:** State layer detection based on deviation density. Firstly, we find the suspicious states \((\exists v_i \in V \text{ but } v_i \notin V)\) and edges \((\exists e_i \in E \text{ but } e_i \notin E)\) from detected state sequence by comparing and contrasting with STG. Then, we obtain \(PW_{ONQ} \), \(PW_{QQ} \), \(PW_{SQ} \) according to the definition of weight. The process is divided into three periods based on IS and CS. Finally, we calculate the deviation density of general suspicious periods and accomplish the first step of detection.

In this step, we try to find anomaly state and path in detecting unexpected behaviors and code injection attack. Due to the context switches of multitasking and multithreading in operating system and programming environment, a few unknown states may occur in the state sequence [20]. Unknown states that are not included in the STG template are considered un-trust behaviors. Although these behaviors are not malicious, they violate the definition of trust and should be detected. The unknown states will be further determined whether they are malicious or not according to deviation density in the general suspicious periods.

In our experiment, we have recorded the activities of running AgileReader for 3 hours, and collected more than 300,000 system calls, which are transformed into about 4000 states. Due to the non-deterministic feature of multitasking in operating system, a few unknown states occurred in the original state sequence, resulting in the deviation increases constantly over time, as shown in Fig. 6 (a). It is hard to set the threshold. To address these problems, we propose a detection approach based on deviation density. This approach considers the changing trend of deviation factors in different situations as well as accumulating deviations in different regions. With the aid of Matlab, we generate Monte-Carlo simulation of unknown state sequence based on original sequence in order to verify our model.

The result is depicted in Fig. 6 (b). Unknown states occurred in the original state sequence are indicated with pointing-up arrows, malicious states can be identified based on deviation density, as indicated by asterisks. From the figure, our model can tolerate the small errors, reduce false positive rate. In these experiment, our model succeeded on identifying abnormal and simplifying calculation.

**Step 2:** If there are no abnormal behaviors in the first step, then the data will be subject to the next step of detection. First, we transfer the state sequence into function sequence and establish a new local function graph \((LFG_t(V_t, E_t, W_t))\), \(V_t = \{v_1, v_2, \ldots, v_n\}\), \(E_t = \{e_1, e_2, \ldots, e_n\}\), \(W_t = \{w_1, w_2, \ldots, w_n\}\) for each function. Then, we apply the following four function semantic rules to detect possible anomalies.
anomalies. (1) If $\forall V_t, E_t \in LFG(V, E, W)$ and $w < \sigma$ comply with the weight proportion of $LFG(V, E, W)$ or traverse $LFG(V, E, W)$ at least once, detected function sequence conforms to normal function sequence. (2) If $(f_{start}, f_1, ..., f_{end}) \in FS_{-Tree}$, detected function sequence conforms to FS-Tree. (3) If $FS_{\text{Dual}} = \{f_k, f_k^{-1}, ..., f_k\}$ ($n < \sigma$) \&\& $\{f_k^{-1}, f_k, f_k^{-1}, ..., f_k\}$ ($n < \sigma$), detected functions that have dual in a function trajectory should be an empty set. (4) If $f_k$ or $f_k^{-1}$ is not appear continuously, functions which $FS_{\text{Attributes}}$ is ‘No’ with its dual function cannot appear for a long time.

With the help of the four detection rules, SB-SF can detect application layer attacks that violate function semantic rules. Attacks, like Trojan that make the illegal user bypass login and read directly, can be detected by rule 2. Another type of attacks is the loop attacks, such as DoS/DDoS attacks and user’s illegality input attacks. Loop attacks use a large amount of inputs to exhaust system resources, such as establishing many network connections and repeatedly performing logins or maliciously downloads. The resource exhausting attacks cause great harm because server cannot deal with the legitimate requests at that time. More importantly, these attacks are hard to be detected because they keep normal states and edges. Based on rule 4 with semantic relationship analyzing, we can find this kind of anomaly. We generated 100 tested state sequences in order to verify the ability of our semantic detection. We use the receiver operating characteristic (ROC) curve as a performance measure, as shown in Fig. 7. In general, the best possible prediction method would yield a point in the upper left corner or coordinate (0, 1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). We can find from Fig. 7, the curve lies closer to the top left corner, our method have a high detection rate for the analysis of application behavior and a low false alarm rate.

5.3 Comparing SB-SF with Other Models

In this section, we use four existing models as references to compare with our SB-SF model. They are N-gram, which is the most representative model of sequences enumeration, the text processing model based on machine learning, the typical automata model FSA, and Software Behavior model based on system object (SBO) [23], which can detect data semantic attacks, as listed in Table 2. The experiments are

| Classification | Model name | Author | Characteristic |
|----------------|------------|--------|----------------|
| Sequences enumeration | N-gram | Forrest | Use fix-length short sequence set to describe software behavior |
| Machine learning | Text processing based | Liao | Use feature vector to describe software behavior |
| Automata | FSA | Sekar | Use system call and its PC to establish automata |
| Semantic | SBO | Fu | It can detect data semantic attacks, which directly or indirectly modifies system call parameters |
also based on AgileReader used in the previous section. We use software security tool, Fortify, to detect vulnerabilities of AgileReader. It has 5 vulnerabilities, which are marked by Common Weakness Enumeration (CWE) [21], which is a community-developed dictionary of software weakness types. Then we take advantage of these security vulnerabilities to create the attacks. But these attacks are only based on CWE vulnerability, we also implement 5 attacks (2 new operation attacks and 3 application layer attacks) on AgileReader function for sufficient testing. We then use all the five models to detect these attacks.

The results are listed in Table 3. We can observe that the top 7 types of attacks are control flow attacks that will result in new states and edges. They are detected by state layer detection. For example, a buffer overflow attack will cause bad damage to the users is detected as well by FSA and our model. The last 3 types of attacks do not change system call sequence at all. The model that uses system call information cannot detect these attacks. However, they can be detected by our semantic analysis model.

5.4 Threats to Validity

In this sub-section, we discuss the threats to our model validity base on reference [22]. Threats to internal validity arise when a few unknown states occur in the state sequence. It is possible that some unknown states occur in the state sequence. Although these behaviors maybe not malicious, they violate the definition of trust and should be detected. In order to guarantee detection precision, the unknown states will be further determined whether they are malicious or not according to deviation density in the general suspicious periods.

Threats to external validity arise when the results of the experiment are unable to be generalized to other application softwares. Although we do not test our other applications with SB-SF model, we are confident in the accuracy of the results. In this experiment, we intercept system calls while normally running AgileReader for 3 hours, repeating login 50 times, and the results were consistent among them.

Threats to construct validity arise when the attack keeps normal states and edges. In our case, we can find this kind of anomaly with semantic relationship analyzing. We performed evaluation on 100 tested state sequences in order to verify the ability of our semantic detection.

6. Conclusions

In this paper, we proposed a new software behavior detection model based on semantic analysis. This approach established a behavior model and developed a two-layer detection mechanism. The main contributions of the research include: (1) We defined the state graph, which revealed the inner properties of software behaviors and reduced the complexity of computation at the same time. The function semantic set as well as the function semantic tree were assigned semantic information with the help of local function graphs. (2) We developed a novel approach of accumulating deviations from the behavior model, the deviation density

| Vulnerability  | Attack description                             | N-gram | Text processing | FSA | SBO | SB-SF |
|---------------|-----------------------------------------------|--------|----------------|-----|-----|-------|
| CWE ID 676    | strcpy() buffer overflow                      | ✓      | ✓              | ✓   | ✓   | ✓     |
| CWE ID 253    | LoadLibraryA() return null pointer overflow   | ✓      | ✓              | ✓   | ✓   | ✓     |
| CWE ID 690    | LoadLibraryA() return null pointer overflow   | ✓      | ✓              | ✓   | ✓   | ✓     |
| CWE ID 120    | OnBtnClickedOk() backdoor dll injection       | ✗      | ✗              | ✓   | ✓   | ✓     |
| CWE ID 131    | OnBtnClickedOk() backdoor dll injection       | ✗      | ✗              | ✓   | ✓   | ✓     |
| New operation | Add a refreshing function                     | ✓      | ✓              | ✓   | ✓   | ✓     |
| New operation | Add open-file function                        | ✗      | ✓              | ✓   | ✓   | ✓     |
| Phishing software | Break rule 1 and 2                            | ✗      | ✗              | ✗   | ✓   | ✓     |
| Trojan        | Break rule 3                                  | ✗      | ✗              | ✗   | ✓   | ✓     |
| Repeat login  | Login action appears for a long time, break rule 4 | ✗      | ✗              | ✗   | ✗   | ✗     |
calculation model. It addressed the shortcomings of the traditional detection approach. (3) We derived four semantic rules in terms of software behavior’s macro characteristics for function layer detection. Using the semantic relationship analysis, false semantic attacks, such as loop attacks and illegal input attacks, could be detected. (4) A case study with contrast experiments was conducted, and the experiment results showed that our model had a stronger expressivity and detection ability, which demonstrated significant improvement over the traditional behavior models.

Acknowledgements

This research partly sponsored by the Funding Project for Academic Human Resources Development in Institutions of Higher Learning under the Jurisdiction of Beijing Municipality (PHR201108016), Open Research Fund of Beijing Key Laboratory of Trusted Computing.

References

[1] Y. Qu, Software behaviouristics, Electronic Industry Publishers, China, 2004.
[2] S.A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls,” Journal of Computer Security, vol.6, pp.151–180, 1998.
[3] Y. Liao and V.R. Vemuri, “Use of k-nearest neighbour classifier for intrusion detection,” Computers & Security, vol.21, no.5, pp.439–448, 2002.
[4] D. Wanger and D. Dean, “Intrusion detection via static analysis,” Proc. IEEE Symposium on Security and Privacy, pp.156–168, 2001.
[5] Q. Yin, R. Zhang, X. Li, and H. Wang, “Research on Technology of Anomaly Detection Based on Dynamic Markov Model,” Electronic Sinica, vol.32, no.11, pp.1785–1788, 2004.
[6] N. Bouguila, J.H. Wang, and A.B. Hamza, “A Bayesian approach for software quality prediction,” The 4th International IEEE Conference on Intelligent Systems, IEEE, pp.11-49–11-54, 2008.
[7] W. Lee, S.J. Stolfo, and K.W. Mok, “A data mining framework for building intrusion detection models,” Proc. 1999 IEEE Symposium on Security and Privacy, IEEE, pp.120–132, 1999.
[8] R. Sekar, M. Bendre, D. Dharjati, and P. Bollinieni, “A fast automation-based method for detecting anomalous program behaviors,” Processing of IEEE Symposium on Security and Privacy, pp.144–155, 2001.
[9] H.H. Feng, O.M. Kolesnikov, P. Fogla, W. Lee, and W. Gong, “Anomaly detection using call stack information,” Processing of IEEE Symposium on Security and Privacy, pp.62–75, 2003.
[10] C. Kruegel, D. Mutz, F. Valeur, and G. Vigna, “On the detection of anomalous system call arguments,” Proc. Computer Security–ESORICS 2003, Lecture Notes in Computer Science, vol.2808, pp.326–343, Springer, Berlin Heidelberg, 2003.
[11] G. Tandon and P.K. Chan, “Learning rules from system call arguments and sequences for anomaly detection,” ICDM Workshop on Data Mining for Computer Security (DMSEC), pp.20–29, 2003.
[12] G. Tandon and P.K. Chan, “On the learning of system call attributes for host-based anomaly detection,” International Journal on Artificial Intelligence Tools, vol.15, no.6, pp.875–892, 2006.
[13] S. Bhatkar, A. Chaturvedi, and R. Sekar, “Dataflow anomaly detection,” Proc. 2006 IEEE Symposium on Security and Privacy, IEEE, pp.15–62, 2006.
[14] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, “Anomalous system call detection,” ACM Transactions on Information and System Security (TISSEC), vol.9, no.1, pp.61–93, 2006.
[15] P. Li, H. Park, D. Gao, and J. Fu, “Bridging the gap between data-flow and control-flow analysis for anomaly detection,” Proc. 2008 Annual Computer Security Applications Conference, IEEE, pp.392–401, 2008.
[16] X. Yang, “Researches on dynamic trusted theories and models of software behavior,” Ph.D. Thesis, China University of Science and Technology, 2010.
[17] J.-M. Fu, F. Tao, D. Wang, and H.-G. Zhang, “Software behavior model based on system objects,” Journal of Software, vol.22, no.11, pp.2716–2728, 2011.
[18] C. Peng, L. Yang, and J. Man, “Dynamic Modeling of Networked Software Interactive Behavior,” Electronic Sinica, vol.41, no.2, pp.314–320, 2013.
[19] Y. Lai, W. Zhang, and Z. Yang, “Research on Software Trust Analysis Based on Behavior,” IEICE Trans. Inf. & Syst., vol.E97-D, no.3, pp.488–496, 2014.
[20] Y. Chen and W.T. Tsai, Service-Oriented Computing and Web Software Integration, 4th ed., Kendall Hunt Publishing, 2014.
[21] http://cwe.mitre.org/
[22] Y. Yu, J.A. Jones, and M.J. Harrold, “An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization,” Proc. International Conference on Software Engineering, pp.201–210, 2008.
[23] J.-M. Fu, F. Tao, D. Wang Dan, and H.-G. Zhang, “Software Behavior Model Based on System Objects,” Journal of Software, vol.22, no.11, pp.2716–2728, 2011.

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 information content security, public opinion analysis, and trusted computing.