Credibility Evaluation of Software Behavior Based on Behavioral Attribute Distance

Junfeng Tian, Ruling Mu

College of Mathematics and Computer Science, Hebei University, Baoding 071002, China, muruling@126.com
doi:10.4156/ijact.vol3. issue3.3

Abstract

In order to evaluate the credibility of software behavior more accurately, a credibility evaluation of software behavior based on behavioral attribute distance is presented. Software behavior trajectory is described by defining the concept of software attribute. Meanwhile, the attributes of node in the trajectory are made a quantization disposal. Through an evaluation process consists of fixed-point evaluation, vertical evaluation and horizontal evaluation based on behavioral attribute distance, we conducted a consistency evaluation of the actual behavior trajectory and expected behavior trajectory of software, and thus judged the credibility of software behavior. Finally, the simulation experiment has validated the efficiency and accuracy of this evaluation method.

Keywords: Software Attribute, Behavioral Attribute Distance, Fixed-point Evaluation, Vertical Evaluation, Horizontal Evaluation

1. Introduction

In the 1990s, Qu put forward the theory topic of software behaviouristics when he was engaged in computer and network security research. He thought that software behaviouristics found a research approach and method of software behavior gradually. Software behavior will become the emerging application theory of computer science. Meanwhile, it will lay a solid theoretical foundation for some fields, such as behavior control, behavior supervision and behavior credible authentication [1].

Whether the actual behavior that the software shows during a software run is consistent with the expected behavior is a major problem that behavior credibility pays close attention to. Generally, researchers conducted a similarity calculation between the description to actual operation process (actual operation trajectory) and the description to expected operation process of software (expected operation trajectory) through monitoring operation process of software, and then evaluated the credibility of software behavior [2,3,4]. During an actual software run, obtaining software behavior dynamically and describing software behavior accurately become the basic problem of behavior credibility research. From the related concepts of software behavior, this paper mainly researches description methods and credibility evaluation methods of software behavior, helps to judge credibility of software behavior more efficiently and accurately.

The rest of the paper is organized as follows. Section II introduces related work that expounds emphatically the obtainment, description and related evaluation methods of software behavior trajectory. The obtainment and description of software behavior trajectory are introduced in section III. Section IV describes a credibility evaluation of software behavior based on the description to software behavior trajectory. Experimental results and analysis are presented in section V. Finally, summarizes full text and points out the research work in future.

2. Related work

Pavel applied AOP technique to behavior surveillance surveys, and observed the concerned events that occur in the system using Aspect [5]. Wang presented a new model of distributed aspectual middleware called DCAM [6]. Zhang proposed a captured technology of software operation trajectory based on AOP [7]. It mainly makes the sensors that use for capturing trajectory in the AOP space weave into each observation point in the system path. During the system run, each sensor extracts and releases the execution position, execution time and work condition of observation point in the
designated area. This technique can infuse function of capturing operation trajectory and monitoring operation behavior for system without changing the source code, thus improve the loose coupling of monitoring module effectively.

Borut put forward the concept of software attribute [8]. It describes some attribute information during a software run, so as to predict the change of the corresponding attribute information during an actual software run. However, such a qualitative description goes against calculating the similarity between two behavior trajectories intuitively. Steffen proposed the method of finite automata [9]. Yang proposed the concept of outlying reduction by extending attribute reduction in rough set theory [10]. Although it is an intuitive and graphical way of describing behavior, it is easy to turn up the state space explosion when interaction processes that between software entities become more complicated.

Zhang proposed a fuzzy comprehensive evaluation model to calculate software trustworthiness attributes [11]. The model considers synthetically five generally accepted attributes of trustworthiness, and evaluates the credibility of software using the fuzzy comprehensive evaluation method. Chen proposed software credibility measurement model based on multidimensional attributes [12]. The model reflects the credibility of software itself objectively and accurately. However, they do not take into account evaluating the credibility of software according to system behavior. And some intrusion detection systems monitored the underlying system behavior of software, that is system call [13, 14, 15].

Software credibility mainly reflects in the behavior credibility [16]. We conduct a behavioral analysis and then calculate the similarity of software behavior trajectories according to the results of behavior analysis. The similarity calculation of software behavior trajectories needs to calculate the similarity not only between the corresponding system calls but also between the state transition information of corresponding system calls. A semantic similarity calculation model based on distance is simple and intuitive, but depends on a pre-established good concept network [17]. A semantic similarity calculation model based on contents cannot distinguish from the semantic similarity between various concepts more carefully [18]. A semantic similarity calculation model based on attributes calls for a detailed and overall description to each attribute of objective thing [19]. An extensible clustering method can handle large amounts of data sets efficiently, but needs to pay for extracting behavior prefixes and their dependency [20]. A novel algorithm that collects only the essential dynamic information required to be analyzed during regression testing is proposed in [21]. A behavior distance method based on HMM is proposed in [15]. The efficiency and accuracy of this method detects attack are all superior to other behavior distance methods. It is the most reliable method judging behavior deviation at the moment. However, it cannot effectively distinguish from the same system call that contains different attribute information.

This paper will act software attribute as a basic element of judging the credibility of software behavior, thus a credibility evaluation of software behavior based on behavioral attribute distance (BAD) is proposed. BAD method evaluates software behavior step by step using an evaluation process consists of fixed-point evaluation, vertical evaluation and horizontal evaluation, and it can calculate the similarity not only between the same system call that contains different attribute information but also between the state transition information of corresponding system calls. This paper makes the following contribution:

- We get function parameters set through defining Aspect and describe software behavior trajectory through redefining software attribute.
- We evaluate the credibility of software behavior through a credibility evaluation of software behavior based on behavioral attribute distance.

3. The obtainment and description of software behavior trajectory

3.1. Related concept

- **Software**: For completing a certain specific task, programs and data collections scheduling a series of system calls in the expected hardware and software environment and limited time.
- **Software behavior**: The states that system call shows and their evolutionary process during software run. Viewing system call as the unit of observation can reflect running state and
running logic of software more fully and easy to understand the concurrency and coordination of task. Meanwhile, system call is also an effective entry point for evaluation of software behavior [13].

- **Software behavior trajectory**: Software can be viewed as consisting of many software movements, each system call can be regarded as a software movement, and then a series of system calls that are scheduled according to the time sequence in the limited time formed a software behavior trajectory. Each system call is abstracted as a task node, and a software behavior trajectory can be viewed as a digraph that is composed of a series of task nodes.

### 3.2. The obtainment of software behavior trajectory

Function parameters set (FPS) is represented by the five-tuples \( \text{FPS}_i = < \text{Id}_i, \text{Dr}_i, \text{Bt}_i, \text{S}_i, \text{Et}_i > \). \( \text{FPS}_i.\text{Id} \) denotes identification information of system call \( i \), \( \text{FPS}_i.\text{Dr} \) denotes the manual input data and execution results before system call \( i \) begins to execute, \( \text{FPS}_i.\text{Bt} \) denotes the time when system call \( i \) begins to execute, \( \text{FPS}_i.\text{S} \) denotes the hardware and software environment information when system call \( i \) executes, \( \text{FPS}_i.\text{Et} \) denotes the time when system call \( i \) terminates. We get the information of FPS using the trajectory capturing technology [5, 6, 7].

A behavior trajectory is obtained after a software run. After sufficient several trainings, a trajectory graph of more complete behavior trajectory of this software is obtained and called the expected behavior trajectory of software. The expected behavior trajectory is stored in the protected designated memory area. A behavior trajectory that is obtained dynamically and real-time during an actual software run is called the actual behavior trajectory of software.

### 3.3. The description of software behavior trajectory

As long as we can compare the actual behavior trajectory and expected behavior trajectory objectively, whether the software behavior gives rise to a deviation can be judged. So, it needs to characterize the software behavior trajectory properly. Software attribute is represented by the two-tuples: \( \text{SWA} = < D, \text{WPNTG} > \). \( D \) denotes the manual input data or execution results before the monitoring moment during an actual software run, namely \( \text{FPS}_i.\text{Dr} \). \( \text{WPNTG} \) denotes a weighted and parameterized nondeterministic task graph, as shown in Figure 1. The detailed description of it is as follows.
Node: includes task nodes and mark nodes. Compared to task node, mark node only represents a kind of uncertainty relationship, such as branch, loop or concurrency.

Checkpoint: The initiated task node, the previous node of the branch, the previous node of the loop, the previous node of concurrency, the previous node of data flow and the previous node of system call are viewed as specific task nodes and called checkpoints. Because the attack on software occurs generally in the branch, loop and concurrency relationships, so whether there is a potential danger can be informed immediately through monitoring software behavior at the checkpoint [22].

Edge: includes task edges and mark edges. Mark edge only represents a kind of connection.

Edge Weight: Task node i and j that have a previous and subsequent relationship, the time interval between the time when node i terminates and the time when node j begins to execute, namely FPSj.Bt-FPSi.Et.

Attributes of Task Node: Temporal parameter denotes the time interval between the time when task node begins to execute and the time when it terminates, namely FPS,Et-FPS,Bt. Executive parameter denotes when the direct previous node of node terminates, the impact of the manual input data and execution results on whether this node executes, namely the impact of FPS,Dr on whether this system call executes. Environmental parameter denotes the hardware and software environment when task node executes.
Figure 1, different mark nodes represent different uncertainty relationships. The representation of Figure 1 intuitively reflects the dependency between nodes. If we ignore this uncertainty relationship, that is neglecting mark node, we will get Figure 2. Obviously, Figure 2 only represents possible run path of nodes, but doesn’t show the dependencies between nodes. Each task node has temporal parameter, executive parameter and environmental parameter, and each task edge has edge weight. They describe completely the scene information of a certain system call during a software run. Describing software behavior according to the concept of software attribute paves a good foundation for credibility evaluation of software behavior.

3.4. The quantization of attributes of task node

The quantization of temporal parameter: temporal parameter is quantized as the time interval of task node executes $[t_1, t_2]$ and called temporal comprehensive index. $t_1$ denotes the minimum execution time of task node and $t_2$ denotes the maximum execution time of task node. In different hardware and software environments, the execution time of system call is different. Even in the same hardware and software environment, the execution time of system call also has a slight deviation because of the impact of other factors. It is just a time value during an actual software run.

The quantization of executive parameter: executive parameter is quantized as the execution probability of task node executes $p$ and called executive comprehensive index. The impact of the manual input data and execution results when the direct previous node of task node terminates on whether this node executes, and then it is determined.

The quantization of environmental parameter: environmental parameter is quantized as the process number ratio when task node executes $s$ and called environmental comprehensive index. During the system call execution, the process number is a constant unless there are other programs or injected attack. In order to facilitate quantitatively calculation, we only use the process number ratio as the standard of measuring environmental parameter. Given the maximum process number is $M$ after sufficient several trainings, the process number when a certain system call executes is $m$, the process number ratio is defined as $m/M$. Because $m/M$ is a rational number between 0 and 1 inclusive, so $s$ is quantified as a number between 0 and 1 inclusive. If $s > 1$, then the process number when the current system call is greater than $M$, indicating that an attack may be injected into the system.

3.5. The determination of attribute weight

Each attribute of task node describes different information of system call during a software run. The change of an attribute value might affect value of other attributes. Therefore, we need to determine the importance of each attribute describing software behavior, and then determine their weight.

In no injected attack circumstance, environmental parameter is a constant. Different environmental parameters may lead to a wider range of change of temporal parameter, which indicates that temporal
parameter has a less importance. But after the execution results of the previous node are determined, the executive parameter of the subsequent node is fixed.

A new technique for finding the attributes and ranking attributes can identify the key underlying factors behind the contrast behavior [23]. Existing feature ranking techniques such as entropy and statistical measures purely focus on the ability of a single attribute to determine a class label, but this technique considers the interaction between attributes. Attribute weight is obtained through this method.

3.6. The storage of software behavior trajectory

Software behavior trajectory is stored by using adjacency list of the data structure. Through accessing adjacency list, direct previous node and direct subsequent node of a certain node can be obtained quickly and randomly.

The head node of adjacency list is shown in Figure 3. node denotes task node or mark node, data denotes manual input data or the implementation results information, p1 denotes temporal comprehensive index, p2 denotes executive comprehensive index, p3 denotes environmental comprehensive index, firstarc denotes the first node that points to linked list. The table node of adjacency list is shown in Figure 4. adjvex denotes position of nodes that are adjacent to vertex in the graph, timesize denotes edge weight, nextarc denotes position of the next node that is adjacent to vertex in the graph.

<table>
<thead>
<tr>
<th>node</th>
<th>data</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>firstarc</th>
</tr>
</thead>
</table>

Figure 3. Head node

adjvex timesize nextarc

Figure 4. Table node

4. The credibility evaluation of software behavior

The two trajectories mentioned in this section, if no special instruction, refer to the actual behavior trajectory and expected behavior trajectory of software.

The credibility evaluation only on checkpoint in the two trajectories is called the coarse-grained evaluation. And the credibility evaluation on all nodes in the two trajectories is called the fine-grained evaluation. The former can quickly inform of whether there is a potential danger, while the latter can evaluate the credibility of software behavior accurately and comprehensively.

4.1. Related concept

- **Behavioral transfer time**: In a software behavior trajectory, the time interval between the time when one node terminates and the time when another node begins to executes, namely the edge weight. In the two trajectories, we can judge the similarity between trajectories preliminarily through comparing behavioral transfer time of the corresponding edges.

- **Behavioral attribute distance of the corresponding nodes**: In the two trajectories, behavioral attribute distance of one pair of corresponding nodes and behavioral attribute distance of their respective direct previous nodes are weighted. Through this value, we can further judge the similarity between trajectories.

- **Behavioral attribute distance of the two trajectories**: Behavioral attribute distance of all pairs of corresponding nodes in the two trajectories is weighted. Ultimately, we can judge the credibility of software behavior through this value.
4.2. Evaluation methods

According to the related knowledge in trust field, the credibility of software behavior can be divided into three levels. The respective description is as follows:

- **Credible**: refers to that the corresponding information of the two trajectories is very similar, indicating that the actual behavior is consistent with the expected behavior completely.
- **Critical Credible**: refers to that a part of corresponding information of the two trajectories is very similar, but another part is not.
- **Incredible**: refers to that the corresponding information of the two trajectories is extremely dissimilar.

The credibility of software behavior, denoted $T$. The corresponding confidence interval is as follows:

$$ T = \begin{cases} [0,T_1), & \text{incredible} \\ [T_1,T_2), & \text{critical credible} \\ [T_2,\infty), & \text{credible} \end{cases} $$

A credibility evaluation of the two trajectories is conducted according to the process that is shown in Figure 5. Software behavior that is critical credible in fixed-point evaluation is conducted a vertical evaluation and horizontal evaluation step by step, and then a credibility conclusion is drawn. So, it not only saves computing overhead but also reduces the complexity.

![Figure 5. Evaluation process](image)

4.2.1. Fixed-point evaluation

This method is specifically described as follows: Given temporal comprehensive index of task node in the actual behavior trajectory of software is $t'$, and that of the corresponding node in the expected behavior trajectory is $[t_1,t_2]$. Let time offset value be $\Delta t_1, \Delta t_2$, and satisfies $0 \leq \Delta t_1 \leq \Delta t_2$. After sufficient several trainings, $\Delta t_1$ is the maximum time offset value that ensures software behavior credible, and $\Delta t_2$ is the minimum time offset value that ensures software behavior incredible. For all
Credibility Evaluation of Software Behavior Based on Behavioral Attribute Distance  
Junfeng Tian, Ruling Mu  
International Journal of Advancements in Computing Technology, Volume 3, Number 3, April 2011

task nodes in two trajectories, if  
\[ |t' - \frac{f_1 + f_2}{2}| > \Delta t_2, \]  
then \( T \in [0, T_1) \), and if  
\[ |t' - \frac{f_1 + f_2}{2}| \leq \Delta t_1, \]  
then \( T \in [T_2, 1] \), otherwise \( T \in [T_1, T_2) \). As shown in Figure 6.

4.2.2. Vertical evaluation

On the basis of fixed-point evaluation, software behavior that is critical credible is conducted a vertical evaluation. This method is specifically described as follows: Given behavioral transfer time between the corresponding nodes and their respective direct previous node in the two trajectories are \( T_s, T_{s'} \), respectively. Let time offset value be \( \delta_1, \delta_2 \), and satisfies \( 0 \leq \delta_1 \leq \delta_2 \). After sufficient several trainings, \( \delta_1 \) is the maximum time offset value that ensures software behavior credible, and \( \delta_2 \) is the minimum time offset value that ensures software behavior incredible. For all task edges in the two trajectories, if  
\[ |T_{s'} - T_s| > \delta_2, \]  
then \( T \in [0, T_1) \), and if  
\[ |T_{s'} - T_s| \leq \delta_1, \]  
then \( T \in [T_2, 1] \), otherwise \( T \in [T_1, T_2) \). As shown in Figure 7.

4.2.3. Horizontal evaluation

On the basis of fixed-point evaluation and vertical evaluation, software behavior that is still critical credible is conducted a horizontal evaluation. This method is specifically described as follows: Let behavioral attribute distance of the corresponding nodes in the two trajectories be \( d \), and threshold be \( \varphi \). If  
\[ d > \varphi, \]  
then \( T \in [0, T_1) \). For all \( d \), if  
\[ d \leq \varphi \]  
until the software terminates, then calculate the behavioral attribute distance of the two trajectories. Ultimately, we can judge the credibility of software behavior through the behavioral attribute distance of the two trajectories.

The behavioral attribute distance of the corresponding nodes is dynamically calculated, and it is compared with the presetting threshold. As long as its behavior attribute distance is greater than threshold, then we judge that software behavior is incredible. Due to the behavioral attribute distance of a pair of corresponding nodes is greater than other pairs, and finally the behavioral attribute distance of the two trajectories is relatively small. So, the similarity of the two trajectories is relatively large, which affects the results of the credibility evaluation of software behavior.

4.3. A credibility evaluation of software behavior based on behavioral attribute distance

Step 1: After sufficient several trainings, Figure 1 is obtained and viewed as the expected behavior trajectory of software. Figure 8 is obtained during a software run and viewed as the actual behavior trajectory of software.

Figure 6. Fixed-point evaluation

Figure 7. Vertical evaluation
Step 2: Given conditions
a. Let temporal comprehensive index, executive comprehensive index and environmental comprehensive index of a task node in Figure 1 be $t_1, t_2$, $p$, and $s$, respectively.
b. Let temporal comprehensive index, executive comprehensive index and environmental comprehensive index of the corresponding task node in Figure 8 be $t_1', p'$ and $s'$, respectively.
c. Let the weight of the corresponding nodes and their direct previous nodes based on behavioral attribute distance of node be $n_1$, $n_2$, respectively. And they satisfy $n_1 + n_2 = 1$. Because of the impact of behavioral attribute distance of their respective direct previous node, behavioral attribute distance of the corresponding nodes is relatively large, which is logical. In order to weaken the impact, there must be $n_1 < n_2$.
d. Let the weight of temporal parameter, executive parameter and environmental parameter be $w_1$, $w_2$ and $w_3$, respectively. And $w_1 < w_2 < w_3$, $w_1 + w_2 + w_3 = 1$.

Step 3: Calculate the behavioral attribute distance of the corresponding nodes
a. The behavioral attribute distance of the two corresponding nodes is defined as follows, denoted $d_1$. And that of their respective direct previous nodes is also calculated using (1), denoted $d_2$. Equation (1) calculates distance using the weighted average method. It considers not only the impact of each attribute on behavioral attribute distance but also the deviation between attributes.

$$d_1 = \frac{\left(\frac{t_1 + t_2}{2} - t_1'\right)^2 * w_1 + (p - p')^2 * w_2 + (s - s')^2 * w_3}{d_1}$$  \hspace{1cm} (1)

b. The behavioral attribute distance of the corresponding nodes is represented using (2), denoted $d$.

$$d = d_1 * n_1 + d_2 * n_2$$  \hspace{1cm} (2)

Step 4: Calculate the behavioral attribute distance of the two trajectories
The behavioral attribute distance of Figure 8 and Figure 1 is represented using (3), denoted $Dist$.

$$Dist = \sum_{i=1}^{n} d_i * k_i$$  \hspace{1cm} (3)

$n$ denotes the number of nodes in Figure 8. $d_i$ denotes the behavioral attribute distance of the $i$ pair of corresponding nodes. $k_i$ denotes the capacity of node $i$ to judge the credibility of software behavior and is always monotonously decreasing with increase of $i$. It is represented using (4). The decreasing range has something with $n$, but its minimum value is greater than 0.5. In the two trajectories, the behavioral attribute distance of the first pair of corresponding nodes affects the behavioral attribute distance.
distance of their respective direct subsequent node. Similarly, the latter also affects the behavioral attribute distance of their respective direct subsequent node. This kind of impact can be accumulated gradually, which will make the behavioral attribute distance of the last pair of corresponding nodes is relatively large. Equation (4) endows the subsequent node with relatively small weight, so as to weaken the impact, thus ensures the accuracy of the behavioral attribute distance of the two trajectories.

\[ k_i = 1 - (i - 1)^{1 - 0.5} = 1 - \frac{i - 1}{2n} \]  

(4)

**Step 5:** Calculate the similarity of the two trajectories

The similarity of the two trajectories is represented using (5), denoted \( \text{Sim} \):

\[ \text{Sim} = \frac{a}{a + \text{Dist}} \]  

(5)

\( a \) represents regulatory factor. Because the behavioral attribute distance of the two trajectories \( \text{Dist} \) is a rational number between 0 and 1 inclusive, in order to guarantee the similarity of the two trajectories is also a rational number between 0 and 1 inclusive, thus regulatory factor is introduced.

**Step 6:** Evaluate the credibility of software behavior

Let the similarity of the two trajectories be \( \text{Sim} \), the threshold be \( \sigma \). If \( \text{Sim} \geq \sigma \), then \( T \in [T - 1, T) \), otherwise \( T \in [0, T) \).

5. Experimental results and analysis

5.1. Experimental environment

At present, AOP is a more mature programming idea. It can solve well cross concern that is distributed in various modules of the application system. The simulation experiment is conducted in host with Intel Pentium 4 CPU3.06 GHz, the memory 2.0 GB, Linux kernel version 2.4.20. We use AspectJ as a development language, Eclipse and AJDT plug-in as development tools.

5.2. Experimental results and analysis

5.2.1. Define aspect

The information of FPS, namely a system call sequence ordering in time, is dynamically obtained through defining Aspect. We intercept a system call sequence when a program runs under the same external environment. The Aspect is shown in Figure 9.

5.2.2. Obtain the information of FPS

The dynamically and real-time intercepted system call sequence in the experiment is \{open, read, write, close, chmod, rename\}. The information of the corresponding FPS is shown in Table 1.

5.2.3. Obtain software behavior trajectory

After sufficient several trainings, the maximum process number \( M = 31 \). The information in Table 1 are described and quantified. The actual behavior trajectory of software is obtained, as shown in Table 2. The information of the intercepted program is conducted sufficient several trainings, and then the expected behavior trajectory of software is obtained, as shown in Table 3.
Credibility Evaluation of Software Behavior Based on Behavioral Attribute Distance
Junfeng Tian, Ruling Mu
International Journal of Advancements in Computing Technology, Volume 3, Number 3, April 2011

Figure 9. Aspect

Table 1. The data information of FPS

<table>
<thead>
<tr>
<th>System call</th>
<th>Id</th>
<th>Dr</th>
<th>Bt(ms)</th>
<th>S</th>
<th>Et(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>5</td>
<td>input: flex,read</td>
<td>0.000</td>
<td>pnum:31</td>
<td>0.177</td>
</tr>
<tr>
<td>read</td>
<td>3</td>
<td>input:1,buf,2kb</td>
<td>0.278</td>
<td>pnum:31</td>
<td>0.643</td>
</tr>
<tr>
<td>write</td>
<td>4</td>
<td>input:1,buf,4kb</td>
<td>0.732</td>
<td>pnum:31</td>
<td>1.140</td>
</tr>
<tr>
<td>close</td>
<td>6</td>
<td>input:output:1</td>
<td>1.243</td>
<td>pnum:31</td>
<td>1.634</td>
</tr>
<tr>
<td>chmode</td>
<td>15</td>
<td>input:764,1 output:0</td>
<td>1.736</td>
<td>pnum:31</td>
<td>2.021</td>
</tr>
<tr>
<td>rename</td>
<td>38</td>
<td>input:1,flex,filerey</td>
<td>2.117</td>
<td>pnum:31</td>
<td>2.320</td>
</tr>
</tbody>
</table>

Note: pnum denotes the number of processes, and the Bt of the first system call open is timed start from 0.000.

Table 2. The actual behavior trajectory of software

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Id</th>
<th>Ts' (ms)</th>
<th>t' (ms)</th>
<th>p'</th>
<th>s'</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>——</td>
<td>0.177</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.098</td>
<td>0.356</td>
<td>0.333</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.103</td>
<td>0.391</td>
<td>0.333</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.096</td>
<td>0.285</td>
<td>0.200</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.102</td>
<td>0.203</td>
<td>0.250</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>——</td>
<td>[0.161,0.183]</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. The expected behavior trajectory of software

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Id</th>
<th>Ts (ms)</th>
<th>[t1,t2] (ms)</th>
<th>p</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>——</td>
<td>[0.161,0.183]</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.099</td>
<td>[0.339,0.385]</td>
<td>0.333</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.103</td>
<td>[0.393,0.427]</td>
<td>0.500</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.107</td>
<td>[0.386,0.414]</td>
<td>0.333</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.105</td>
<td>[0.263,0.329]</td>
<td>0.200</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0.102</td>
<td>[0.193,0.237]</td>
<td>0.250</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>
5.2.4. Evaluate the credibility of software behavior

In our experiment, \( \Delta t_1 = 0.012, \Delta t_2 = 0.035, \delta_1 = 0.01, \delta_2 = 0.03, (w, w, w) = (0.20, 0.30, 0.50), a = 1, \varphi = 0.05, \sigma = 0.90, (n,n) = (0.35,0.65). \)

a. Table 2 and Table 3 are conducted a fixed-point evaluation. Software behavior is credible because each system call satisfies \( \left| t' - \frac{t_1 + t_2}{2} \right| \leq 0.012 \). Therefore, the credibility conclusion can be drawn without vertical evaluation and horizontal evaluation.

b. The program is modified easily. Suppose that the intercepted system call sequence after modification is consistent with Table 1, but the FPS information of the corresponding system calls may be different. The actual behavior trajectory of software after modification is obtained, as shown in Table 4. Table 4 and Table 3 are conducted a credibility evaluation. In the fixed-point evaluation, some system calls satisfy \( \left| t' - \frac{t_1 + t_2}{2} \right| > 0.035 \), and some satisfy \( 0.012 < \left| t' - \frac{t_1 + t_2}{2} \right| \leq 0.035 \). Therefore, software behavior is critical credible. In the vertical evaluation, some system calls satisfy \( |T_s' - T_d| \leq 0.01 \), and some satisfy \( 0.01 < |T_s' - T_d| \leq 0.03 \). Therefore, software behavior is still critical credible. Finally, in the horizontal evaluation, \( Dist = 0.1221, Sim = 0.891 \). Because of \( Sim = 0.891 < \sigma = 0.90 \), thus software behavior is judged incredible.

### Table 4. The actual behavior trajectory of software after the modification

<table>
<thead>
<tr>
<th>Attributes</th>
<th>( T_s' (\text{ms}) )</th>
<th>( t' (\text{ms}) )</th>
<th>( p' )</th>
<th>( s' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.186</td>
<td>1.000</td>
<td>1.030</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.105</td>
<td>0.400</td>
<td>0.333</td>
<td>1.030</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
<td>0.426</td>
<td>0.500</td>
<td>1.030</td>
</tr>
<tr>
<td>6</td>
<td>0.097</td>
<td>0.421</td>
<td>0.333</td>
<td>1.030</td>
</tr>
<tr>
<td>15</td>
<td>0.086</td>
<td>0.313</td>
<td>0.250</td>
<td>1.030</td>
</tr>
<tr>
<td>38</td>
<td>0.111</td>
<td>0.228</td>
<td>0.333</td>
<td>1.030</td>
</tr>
</tbody>
</table>

5.3. Comparison experiment

Through BAD method and HMM method [15], software behavior trajectories in Table 2 and Table 3, Table 4 and Table 3 are conducted a credibility evaluation. This is the contrast experiment.

5.3.1. Efficiency comparison

Figure 10, the evaluation time of BAD method changes very large before and after modification, but the evaluation time of HMM method has no significant change and is always greater than that of BAD. Before modification, BAD method can judge the credibility of software behavior only through a fixed-point evaluation, so the evaluation time is small. After modification, BAD method can judge the credibility of software behavior through a fixed-point evaluation, vertical evaluation and horizontal evaluation in turn, so the evaluation time is noticeably increased. However, regardless of how the information of system call sequence changes, HMM method evaluates software behavior in accordance with the same pattern, so the evaluation time is almost equal. Therefore, the comprehensive evaluation efficiency of BAD method is higher than that of HMM method.

5.3.2. Accuracy comparison

Figure 11, the fold-line of BAD method is smoother than that of Figure 12 and its change scale is small, indicating that software behavior is credible. Figure 12, the fold-line of BAD method changes dramatically and the distance of each corresponding system call has a large fluctuation range,
indicating that an attack has injected into open system call and software behavior is incredible. Before and after modification, the results of BAD method have a big variation and those of HMM method have no significant change, which can be seen clearly in Figure 11 and Figure 12. This is because HMM method cannot distinguish the same system call with different attribute information, thus it failed to detect the injected attacks. But BAD method can describe the deviation of the two trajectories accurately. Therefore, the evaluation accuracy of BAD method is higher than that of HMM method.

6. Conclusion

BAD method that is proposed by this paper describes software behavior trajectory using the concept of software attribute and conducts a consistency evaluation of the actual behavior trajectory and expected behavior trajectory of software through an evaluation process. This method evaluates gradually the credibility of software behavior using temporal parameter, behavioral transfer time and behavioral attribute distance, which makes the evaluation more efficient and results more accurate. In
addition, this method can be applied to the intrusion detection system, and has very great practical significance. In future work, we will act behavior credibility as the main research topic and dedicate to work out more efficient and accurate credibility evaluation method of software behavior.

7. References