Chance Discovery and Management for Computer Security

Sang-Jun Han, Ja-Min Koo and Sung-Bae Cho
Dept. of Computer Science, Yonsei University,
134 Shinchon-dong, Seodaemun-gu, Seoul 120-749, Korea
{sjhan, icicle, sbcho}@clab.yonsei.ac.kr

Abstract. This paper discusses the role of chance discovery and management in computer security. We address the limitations and requirements of conventional approach in the community of computer security. We then explain how the chance discovery approach overcomes these limitations and propose two novel methods to meet these requirements. The experimental result against the real computer audit data set verifies that the proposed methods based on chance discovery approach can improve the conventional method.

1 Introduction

Due to advances in information communication technology, many software and hardware tools for the security of the computer system have become esses. Especially, intrusion detection systems (IDSs), which detect computer attacks such as unauthorized usage, misuse by a local user and modification of important data by analyzing computer audit data, have been studied by researchers and realized as several commercial products. Pattern recognition techniques have been exploited in order to analyze computer audit data and recognize computer anomalies.

Generally, intrusion detection techniques can be divided into two categories: misuse detection and anomaly detection. Misuse detection uses knowledge about known attacks and attempts to match current behavior against the attack patterns. Most commercial IDSs take this approach because they have the advantage that known attacks can be detected economically and reliably with low positive error. The shortcoming is that it cannot detect unknown attack patterns. However, this problem of misuse detection, anomaly detection, which uses data about normal behaviors and attempts to detect intrusions by noting deviations from normal behaviors, has been studied actively. However, it has some drawbacks. First, it suffers from high false-positive error rate where normal behaviors are considered as attacks. Second, it only recognizes occurrence of a computer anomaly and does not explain what kind of anomaly has been occurred.

Chance discovery is to recognize a chance which is a very rare event with significant impact on decision making or future change [3]. It gives
an awareness of chances but also an explanation about chances. This approach has been applied to various application domains such as predicting earthquake, discovering new topics from WWW, and agent communication [4, 5]. Here, we discuss about how the chance discovery approach works in the domain of computer security systems.

The rest of this paper is organized as follows. In Section 2, we address the limitations of conventional approach and discuss the possible role of chance discovery and management. The detailed description of the proposed methods is presented in Section 3. Experimental results are shown in Section 4. Section 5 is about the conclusion and future works.

2 Computer Anomalies as Chances

Chance discovery focuses on events which are very rare and has significant influence on the future change. Computer anomaly can be viewed as a kind of chance for following reasons. First, computer anomaly is very rare. Computer audit data mostly consists of the traces of normal behavior.

In addition, nowadays many novel types of attack are rapidly coming out. It is very hard to forecast future attack types and make preparation against them. But to defend against unpredictable future anomalies is an essential ability. Second, the impact of computer anomalies is very high. The intruder who succeeds to gain the root privileges can steal the important data file and ruin the whole computer system. Denial of service attack or computer worms can cause network failure and stop the entire local network service.

Anomaly detection approach aims to discover these rare and unpredictable events from the huge amount of computer audit data. It builds model of very frequent and relatively predictable situation i.e. normal system's usage or user's behavior to recognize very rare events i.e. anomalies. However, it lacks the explanation ability. It is very important for computer system's safety of the future to know the type of anomaly which has been occurred because the required preparations are different case by case. The explanation of the occurred anomaly is strongly required for system administrator to maintain computer system's safety. The aspect of chance discovery, emphasis on rareness and explanation, can help to recognize computer anomalies and make computer system more secure and reliable.

3 Chance Discovery Approach for Computer Security

Figure 1 shows an overview of our anomaly detection and interpretation system based on chance discovery approach. The monitored system can consist of one or multiple computer machine(s) or one or multiple computer network(s). The audit facility provides the information which is needed to identify attacks, called audit data, such as command history, system call traces, and network packet dump. Here, we consider the monitored system as a computer machine and the audit facility as a system call tracer because we focus on the host-based security system.
The event modeler builds a model of normal events which are frequently occurred in the monitored system. Anomaly discoverer recognizes computer anomalies using the normal event model and makes alarms. It discovers anomaly by identifying an events deviating from normal and frequent events. If an anomaly is discovered, the anomaly interpreter provides the explanation about the discovered anomaly in order to help the system administrator to understand the vulnerability of the monitored system and to take some actions for preventing the recurrence of an attack and additional damage such as installing some patches, cutting network connections, and shutting down the vulnerable services.

Fig. 1. An overview of our anomaly detection and interpretation system

3.1 Discovering Computer Anomalies

Anomaly discovery with system call traces can be formulated as follows. Let \( P = (s_1, s_2, \ldots, s_N) \) denote the set of all system call records generated by one execution of a program where \( s_t \) denotes a system call event occurred at time \( t \), and \( S_t = (s_{t+1}, s_{t+2}, \ldots, s_{t+L}) \), \( t \leq N - L \) denote the set of sequences made by windowing \( P \) using window length \( L \) at time \( t \) and \( R_t \) denotes the result of sequence evaluation with an evaluation function \( \text{eval} \) given a normal event model \( \lambda \).

\[
R_t = \text{eval}(S_t, \lambda)
\]  

(1)

If \( R_t \) is greater than predefined threshold, given execution of program is recognized as intrusion.

\[
\text{alarm}(R_t) = \begin{cases} 
\text{normal} & \text{if } R_t \geq \text{threshold} \\
\text{attack} & \text{if } R_t < \text{threshold}
\end{cases}
\]

(2)

The performance of anomaly discovery hinges on the technique for building the normal event model \( \lambda \) and the evaluation function \( \text{eval} \). Machine learning techniques which are known for good solutions for this kind of problem such as rule learning, neural network, and HMM have been applied. In this paper, hidden
Markov model (HMM) is used because of its good performance for processing temporal sequence data.

An HMM is a doubly stochastic process with an underlying stochastic process that is not observable, and can only be observed through another set of stochastic processes that produce the sequence of observed symbols [6]. This model can be thought of as a graph with $N$ nodes called 'states' and edges representing transitions between these states. Each state node contains initial state distribution and observation probabilities at which a given symbol is to be observed. An edge maintains a transition probability with which a state transition from one state to another state is made. Figure 2 shows a left-to-right HMM with 3 states.

![Diagram of a left-to-right HMM with 3 states]

**Fig. 2.** An example of left-to-right HMM.

Given an input sequence $O = O_1, O_2, ..., O_T$, HMM can model this with its own probability parameters using Markov process though state transition process cannot be seen outside. Once a model is built, the probability with which a given sequence is generated from the model can be evaluated.

The probability with which the sequence is generated from the model can be calculated by summing the probabilities for all the possible state sequences. In practice, a more efficient method, known as forward-backward procedure, is used. Recognition can be done with sequences of measures of events as its input using well-established HMM procedures. The learning of HMM is to adjust its internal parameters to maximize the probability $Pr(O | \lambda)$. Because no analytic solution is known to do it, an iterative method called Baum-Welch reestimation is used [6].

Conventional host-based anomaly detection technique which uses system call traces has used only system call ID as the measure of system's behavior. System call ID is useful measure to monitor system's behavior. However, it is inadequate for monitoring the whole behavior. For example, we cannot know whether system call is failed or executed successfully. It is also unknown if system call is related to a critical file or an ordinary file. Therefore more information is needed to model user's behavior accurately. Solaris BSM provides additional information on user’s behavior besides the system call ID: such as process ID, system call return value, related file, etc. We also model these additional information using HMMs and combine them to improve the performance.
Because additional system-call-related information is too large and hard to apply to HMM directly, a data reduction technique is needed. We organize the information using a self-organizing map (SOM) to reduce the additional information. System-related information, process-related information, and file access-related information are extracted from each BSM event and reduced using SOM, an unsupervised learning neural network, using Euclidean distance to find the distance between input vector and reference vector as shown in Table 1.

Table 1. Measures extracted from the BSM to detect intrusion.

<table>
<thead>
<tr>
<th>Group</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call</td>
<td>ID, return value, return s</td>
</tr>
<tr>
<td>Process</td>
<td>ID, IPC ID, IPC permission,</td>
</tr>
<tr>
<td></td>
<td>exit value, exit status</td>
</tr>
<tr>
<td>File access</td>
<td>access mode, path, file size</td>
</tr>
<tr>
<td></td>
<td>file name, argument length</td>
</tr>
</tbody>
</table>

Several measures can be extracted from one audit record of BSM that includes an event and the information is normalized for the input of an output of SOM, we can get one representative instead of many inputs that is, one record is converted into one representative. Figure 3 shows the overall flow of reducing audit data. In this paper, we reduce the measure size based on the locality of user action. We observe the range of the measures and some ones used in the system as a table where we find the value of measures used for action. As a result of mapping of measures, we can obtain reduced data.

![Fig. 3. Overall flow of reducing BSM audit data](image)

When one event is evaluated through each model, a vector of evaluated values is generated. A method to combine multiple models is required to decide whether current sequence is an anomaly. Combining multiple detectors is good for improving the performance of detection systems. [8] and [9] combine detectors using ensemble learning. They have applied machine learning techniques such as artificial neural networks and support vector machines to intrusion detection and showed that combining different detectors is superior to just one.
Because additional system-call-related information is too large and various to apply to HMM directly, a data reduction technique is needed. We use self-organizing map (SOM) to reduce these additional information. System call-related information, process-related information and file access-related information are extracted from each BSM event and reduced using SOM which is an unsupervised learning neural network, using Euclidean distance to compute distance between input vector and reference vector as shown in Table 1 [7].

<table>
<thead>
<tr>
<th>Group</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call</td>
<td>ID, return value, return status</td>
</tr>
<tr>
<td>Process</td>
<td>ID, IPC ID, IPC permission</td>
</tr>
<tr>
<td>File access</td>
<td>exit value, exit status</td>
</tr>
<tr>
<td></td>
<td>access mode, path, file system</td>
</tr>
<tr>
<td></td>
<td>file name, argument length</td>
</tr>
</tbody>
</table>

Several measures can be extracted from one audit record of BSM which includes an event and the information is normalized for the input of SOM. As an output of SOM, we can get one representative instead of many measures, that is, one record is converted into one representative. Figure 3 shows the flow of reducing audit data. In this paper, we reduce the measure size based on the locality of user action. We observe the range of the measures and save actual ones used in the system as table where we find the value of measure of current action. As a result of mapping of measures, we can obtain reduced data.

![Reduced Data Diagram](image)

Fig. 3. Overall flow of reducing BSM audit data

When one event is evaluated through each model, a vector of evaluation values is generated. A method to combine multiple models is required to finally decide whether current sequence is an anomaly. Combining multiple detectors is good for improving the performance of detection systems. [8] and [9] combined detectors using ensemble learning. They have applied machine learning techniques such as artificial neural networks and support vector machine to intrusion detection and showed that combining different detectors is superior to individual
approaches. In this paper, we have combined HMM-based detectors that determine if current sequence is abnormal according to the measure it is responsible for: system call-related, process-related and file access-related measures. Each detector participates in the final anomaly decision.

Each detector is given a weight \( W_m \) according to their confidence. Voting method is determined. Typical voting methods include unanimity, majority and OR voting. In OR voting, anomaly is determined if at least one member votes positively. Voting is to determine whether or not the total result \( R \) is greater than or equal to the \( T \) depending on the voting method.

\[
R = \sum W_m \cdot V_m \quad \begin{array}{l}
W_m : \text{model weight} \\
V_m : \text{model voting value}
\end{array}
\]

- \( R = 1 \) (unanimity),
- \( R \geq 0.5 \) (majority),
- \( R > 0 \) (OR voting)

Generally, OR voting enhances a detection rate but it increases an error rate. Unanimity improves an error rate but decreases the detection rate.

3.2 Explanation of Discovered Anomalies

We can provide an explanation of the discovered anomalies in many ways such as noticing the information about what the victim program is, where the intruder is, and the significance of occurred anomaly. In this paper, we attempt to provide the information about what type of anomaly has been occurred because the anomaly type is very important to prevent the future reoccurrence of the same anomaly. To do this, we analyze the state change sequence of system calls \( S_i \) which has been determined as an anomaly.

Even though HMM does not provide the state sequence explicitly, we can estimate the state sequence of the most probable ones using the Viterbi algorithm which finds the most-likely state transition path in a state diagram, given an input sequence [6]. It has been applied to speech and character recognition tasks where the observation symbols are modeled by HMM. The Viterbi algorithm can be easily combined with other information in real-time. The state change sequence \( S \) can be obtained as follows:

- Initialization:
  \[
  \delta_1(i) = \pi_i \cdot b_i(O_1), 1 \leq i \leq N \\
  \psi_0(i) = 0
  \]

- Recursion:
  \[
  \delta_t(j) = \max_{i} [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t), 2 \leq t \leq T, 2 \leq j \leq N \\
  \psi_t(j) = \arg \max_{i} [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t), 2 \leq t \leq T, 2 \leq j \leq N
  \]
• Termination:

\[ P^* = \max_{s \in \mathcal{S}_T} [\delta_T(s)] \]
\[ S^*_T = \arg\max_{s \in \mathcal{S}_T} [\delta_T(s)] \]

• Backtracking:

\[ S^*_t = \gamma_{t+1}(s_{t+1}^*), t = T - 1, T - 2, \ldots, 1 \]

To identify the type of anomalies, we build models of the state change sequences of which anomalies have been previously occurred by simply storing the state change sequences of each anomaly type into the database. We then measure the similarity between the state change sequences of the anomaly which is currently discovered and the stored state change sequences of each anomaly type using edit distance. Edit distance is based on dynamic programming, and the result of edit distance is the minimal cost of sequence of operations that are used to compare strings \( x \) and \( y \).

- \( \delta(\varepsilon, a) \) inserting the letter \( a \) into string \( \varepsilon \)
- \( \delta(a, \varepsilon) \) deleting the letter \( a \) from string \( \varepsilon \)
- \( \delta(a, b) \) replacing \( a \) by \( b \), for \( a \neq b \)

A matrix \( C_{i,j}[|x|, |y|] \) is filled, where \( C_{i,j} \) represents the minimum number of operations needed to match \( x_{1..i} \) to \( y_{1..j} \). This is computed as follows for \( C_{|x|,|y|} = \text{ed}(x, y) \) at the end.

\[ C_{0,0} = 0 \]
\[ C_{i,0} = 0 \]
\[ C_{i,j} = \begin{cases} 0 & \text{if } (x_i = y_j) \\ C_{i-1,j-1} & \text{if } (x_i \neq y_j) \\
\text{else } 1 + \min \left( C_{i,j-1}, C_{i+1,j}, C_{i-1,j-1} \right) & \end{cases} \]

The rationale of the above formula is as follows. First, \( C_{0,j} \) represents the edit distance between a string of length \( i \) or \( j \) and the empty string. Clearly, \( i \) and \( j \) deletions are needed on the nonempty string. For two non-empty strings of length \( i \) and \( j \), we assume inductively that all the edit distances between shorter strings have already been computed, and try to convert \( x_{1..i} \) into \( y_{1..j} \).

The dynamic programming algorithm must fill the matrix in such a way that the upper, left and upper-left neighbors of a cell are computed prior to computing that cell. This is easily achieved by either a row-wise left-to-right traversal or a column-wise top-to-bottom traversal. Table 2 illustrates the algorithm to compute ed(“survey”, “surgery”).

When an anomaly is discovered, it calculates the distances between the standard state sequences and the current state sequence. Afterward, input state sequence is identified with the corresponding intrusion type of which the distance is the smallest.
Table 2. Edit distance between “survey” and “surgery”

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>u</th>
<th>r</th>
<th>g</th>
<th>e</th>
<th>r</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>r</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>v</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

4 Experimental Results

First, we have conducted the experiments to compare the detection methods based on system call ID and that on additional measures reduced by SOM and the combined method. Two models have combined by unanimity voting, majority voting and OR voting methods. Each model is given the same voting weight.

For this experiment, we have used data obtained from one graduate student for one week. He has mainly used text editor, compiler and programs of their own writing. Approximately total 20,000 system call events have been collected and among them 10,000 are used for training and another 10,000 are for testing. 17 cases of u2r intrusion, one of the most typical intrusions, are included in the user’s test data set.

Table 3. The performance of combining multiple models

<table>
<thead>
<tr>
<th></th>
<th>only system call ID</th>
<th>SOM reduced</th>
<th>Voting method</th>
</tr>
</thead>
<tbody>
<tr>
<td>detection rate</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>false-positive error rate</td>
<td>5.33%</td>
<td>23.53%</td>
<td>1.18%</td>
</tr>
</tbody>
</table>

We have used the best result from each model because subthreshold for each model may differ from each other. Detection rate of the combined method does not change because each model’s detection rate is 100%. However, False-positive error rate has enhanced compared to the others as shown in Table 3.

Next, we have evaluated the performance of anomaly interpretation. For this experiment, we have collected the test data set which contains 3 types of anomalies and 30 instances of each anomaly type as shown in Table 4.

In this paper, we use the edit distance for identifying the type of intrusions. To verify the usefulness, we use other 3 kinds of distance measures for comparing the performance: Euclidean distance, Hamming distance and Longest Common Subsequence (LCS). We can observe that the performance of using edit distance is better than those of the others as shown in Figure 4. The performance of
Table 4. The types of anomalies

<table>
<thead>
<tr>
<th>Class</th>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflow</td>
<td>OpenView xlock Heap Overflow</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Lpset -r Buffer Overflow Vulnerability</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Kcms.aparc Configuration Overflow</td>
<td>30</td>
</tr>
</tbody>
</table>

Hamming distance is poor, because it compares the state sequences in equal length one by one. For example, if the input state sequence of kcms attack is [0, 2, 2, 4, 6, 8, 10, 12, 14, 16] and the previously-collected state change sequence of kcms and lpset is [0, 2, 4, 6, 8, 10, 11, 13, 15, 17] and [0, 2, 2, 2, 4, 6, 8, 10, 12, 14] respectively, it identifies with kcms correctly by the edit distance: $E.D($kcms$)=36$, $E.D($lpset$)=42$. However, in case of Hamming distance, it identifies the state sequence with lpset: $H.D($kcms$)=2$ and $H.D($lpset$)=3$.

Fig. 4. Success rates of each distance measure

Another distance measure, LCS, identifies the type of intrusions based on the length of common subsequences. For instance, when the previously-collected state sequences of kcms and xlock are [0, 2, 4, 6, 8, 10, 11, 13, 15, 17] and [0, 2, 4, 6, 8, 10, 12, 14, 16, 17] respectively, the result of xlock is 27 and that of kcms is 18 using the edit distance for given the input state sequence of kcms [0, 2, 4, 6, 8, 10, 12, 14, 16]. Therefore, the input state sequence is identified with kcms correctly, because the distance is smaller than that with xlock. However, in case of LCS, suppose the results of LCS with xlock and lpset as $LCS($xlock$)$ and $LCS($kcms$)$, we can obtain the results as $LCS($xlock$)=9$ and $LCS($kcms$)=6$. Therefore, LCS identifies the input state sequence with xlock incorrectly.
On the other hand, given the input state sequences [0, 2, 3, 5, 7, 9, 11, 13, 15, 17, 18] of lcms. If we identify the type of intrusions with Euclidean distance, the result is xlock (2.8284). However, if we use the edit distance to identify the type of intrusion, we can identify the input state sequence with lcms: \( E.D(\text{xlock})=24 \) and \( E.D(\text{lcms})=21 \). We can reduce the fault rate using the edit distance which performs proper operations.

5 Conclusions

In this paper, we have discussed the role of chance discovery and management for advanced computer security system and proposed a novel computer anomaly discover and interpretation system. The proposed system discovers computer anomalies combining multiple HMM models and interpret the discovered anomalies analyzing the internal state changes. However, it has a drawback: the types of anomalies which are available for interpreting are limited to previously known types. As a future work it is needed to develop the technique to interpret unknown anomalies and provide the explanation about them. We will also evaluate the proposed method with larger data set.

Acknowledgments

This work was supported by Korea Brain Science and Engineering Research Program sponsored by Korean Ministry of Science and Technology.

References