Considering Both Intra-Pattern and Inter-Pattern Anomalies for Intrusion Detection

Ning Jiang *Kien A. Hua*

School of EECS, University of Central Florida
Orlando, FL 32816-2362
{njiang, kienhua} @cs.ucf.edu

Simon Sheu[1]

Department of Computer Science
National Tsing Hua University
101, Section 2, Kuang Fu Road,
Hsin-Chu, Taiwan 30013, R.O.C.
sheu@cs.nthu.edu.tw

Abstract

Various approaches have been proposed to discover patterns from system call trails of UNIX processes to better model application behavior. However, these techniques only consider relationship between system calls (or system audit events). In this paper, we first refine the definition of maximal patterns given in [8] *and provide a pattern extraction algorithm to identify such maximal patterns. We then add one additional dimension to the problem domain by also taking into consideration the overlap relationship between patterns. We argue that an execution path of an application is usually not an arbitrary combination of various patterns; but rather, they overlap each other in some specific order. Such overlap relationship characterizes the normal behavior of the application. Finally, a novel pattern matching module is proposed to detect intrusions based on both intra-pattern and inter-pattern anomalies. We test this idea using the data sets obtained from the University of New Mexico. The experimental results indicate that our scheme detect significantly more anomalies than the scheme presented in* [8] *while maintaining a very low false alarm rate.*

1. Introduction

Intrusion detection using trails of system calls (system audit events) has been studied extensively for many years. In the original works ([2][3][4][10]) of Forrest et al, the application under investigation is executed under various normal scenarios. System calls invoked by the application are captured and recorded in sequences, which are usually referred to as training sequences. Normal behaviors of applications are modeled by registering fixed-length subsequences (i.e. patterns) of these training sequences. Intrusions are reported when there are significant deviations between the newly observed sequences (i.e. intrusion sequences) and the stored fixed-length patterns. The fixed-length pattern approach works well as long as the length of the pattern is properly chosen. Wespi et al proposed several schemes [7][8][9] to produce variable-length patterns. In particular, the technique, presented in [8], showed significant improvement over the fixed-length pattern methods. The main idea of the scheme involves discovering variable length *maximal* patterns and generating "building-blocks" to compose potentially any possible normal system audit sequences.

In this paper, we first refine the definition of maximal patterns in [8] and provide a pattern extraction algorithm to identify such maximal patterns. We then add one more dimension to the problem domain. In addition to examining the relationship between individual system calls (or system audit events), our technique also takes into consideration the relationship between patterns. We argue that an execution path of an application is usually not an arbitrary combination of various patterns; but rather, they overlap each other in some specific order. Such overlap relationship characterizes the normal behavior of the application. Finally, a novel pattern matching module is proposed to detect intrusions based on both intra-pattern and inter-pattern anomalies. We tested our technique using the data sets obtained from the University of New Mexico. The experimental results indicated that with approximately the same space and time efficiency, our scheme detected significantly more anomalies while maintaining a very low false alarm rate.

To the best of our knowledge, the proposed intrusion detection method is the first to exploit overlap relationship between patterns. In [2][3][4][10], overlap relationship between patterns is implicitly recorded by

[1]The author was supported by the MOE Program for Promoting Academic Excellent of Universities, Grant No. 89-E-FA04-1-4, Taiwan, R.O.C.

sliding a window over all the system calls and capturing unique sequences. However, the information is not utilized in the detection of attacks. In [8], the main idea is to identify "building blocks" to model normal application behavior. As a result of the pattern reduction operation, overlapping patterns are "flatted" and overlap relationship between patterns is no longer preserved.

2. System Architecture and Definition

In this section, we first give a brief description of the architecture of our intrusion detection system. Notations and formal definitions are then presented to facilitate further discussion.

2.1. System Architecture

The system consists of two parts: the offline training part and the online detection part. Operations of each component are described as follows:

1. Data collection module: The application under investigation is executed under various normal usage scenarios. During each execution, the application spawns one or many processes. System calls invoked by each of these processes are captured and recorded in a sequence. The output of this module is a number of system call sequences.
2. Data preprocessing module: First, each system call of the training trails is translated into a system call ID according to a predefined translation table. Identical consecutive system call IDs are then aggregated into one. After aggregation, there might be groups of identical system call sequences. For each group, we only keep one representative sequence and remove all other sequences.
3. Pattern extraction module: This module extracts *maximal* patterns from selected set of sequences generated by the data preprocessing module.
4. Pattern overlap relationship identification module: Patterns are organized into adjacency lists in which overlap relationship between patterns is located and maintained.
5. Pattern matching module: Incoming system calls of the application under protection are first subject to the same preprocessing operations as in 2. Pattern adjacency lists are then traversed at real time to identify both intra-pattern and inter-pattern mismatches. Significant deviations from the normal behavior cause the module to raise alerts.

In the following sections of the paper, we focus on the description of the last three components.

2.2. Notations And Definitions

Consider an alphabet Σ of size k, $\Sigma = \{e_1, e_2, ..., e_k\}$. In this paper, each element of Σ is a system call. A sequence s is defined as $s = (a_1, a_2, ..., a_n)$, $(a_i \in \Sigma, 1 \le i \le n)$. The size of a sequence s is represented by $|s|$. The input to the pattern extraction module is a set T of training sequences: $T = \{T_1, T_2, ..., T_m\}$. The ith sequence of T is defined as T_i, and the jth element of T_i is denoted as T_{ij}. Each element e_i of Σ can appear zero or more times in any sequence of T. Each such occurrence is referred to as an *instance* of e_i.

A pattern p is a subsequence of any sequence of T. Hereafter, a pattern with length i is referred to as an i-pattern. A pattern p can appear at various positions in different training sequences. We call each occurrence of a pattern in T an *instance* of that pattern. We use the notation $N(p,T)$ to denote the total number of instances of a pattern p in T. A pattern p is considered *frequent* with respect to the training set T if and only if $N(p,T) \ge 2$ and $|p| \ge 2$. In addition, we use a *location pair* $\langle d, f \rangle$ to record the position of a pattern instance starting at the system call T_{df} in a training sequence T_d. A set $L(p)$ is used to represent all the location pairs of a pattern p. Apparently, $|L(p)| = N(p,T)$.

Definition 1: Given a set T of training sequences and a frequent pattern p of T. Pattern p is *maximal* if and only if there exists at least one instance of p, such that it never appears in T as a subsequence of an instance of any other frequent pattern q. Suppose the length of p is i, p is also referred to as a maximal i-pattern.

Basically, Definition 1 states that a frequent pattern cannot be maximal if it always occurs as a subsequence of some other frequent pattern. For a given training set T, the set $P(T)$ represents all the maximal patterns of T.

We define a Boolean function *overlap* on two *location pairs* $\langle d_1, f_1 \rangle$ and $\langle d_2, f_2 \rangle$ of two maximal pattern p and q, respectively, as follows:

Definition 2: Consider two maximal patterns p and q (p and q can be identical) of a set T of training sequences. $\forall \langle d_1, f_1 \rangle \in L(p), \forall \langle d_2, f_2 \rangle \in L(q)$, a Boolean function is defined on the two location pairs as:

$$overlap(\langle d_1, f_1 \rangle, \langle d_2, f_2 \rangle) = \begin{cases} TRUE & \text{if } \begin{pmatrix} (d_1 = d_2) \wedge (f_1 < f_2) \\ \wedge (f_2 \le f_1 + |p| - 1 < f_2 + |q| - 1) \end{pmatrix} \\ FALSE & \text{otherwise} \end{cases}$$

If $overlap(\langle d_1, f_1 \rangle, \langle d_2, f_2 \rangle) = TRUE$, $p_{f_2 - f_1 + 1}$ is referred to as the *overlapping point*. As an example, suppose we have the following set T of training sequences:

T = {"4 2 66", "4 2 66 105", "2 66 105" }
According to Definition 1, $P(T)$={"4 2 66", "2 66 105"}. The location pairs of both of the maximal patterns in T_2 are <2,1> and <2,2>, respectively. Obviously, *overlap*(<2,1> ,<2,2>)=*TRUE*. The system call "2" in pattern "4 2 66" is referred to as the overlapping point.

Definition 3: A maximal pattern p of a training set T is *terminable* if and only if there exists an instance of p such that it does not overlap with instances of any maximal pattern q. i.e.,

$$\exists loc \in L(p)\left(\forall q \in P(T)\left(\forall loc_1 \in L(q)\begin{pmatrix}\neg overlap(loc_1, loc)\\ \wedge \neg overlap(loc, loc_1)\end{pmatrix}\right)\right)$$

For instance, the maximal pattern "4 2 66", in the last example, is terminable.

3. Pattern Extraction

In this section, we discuss how to compute $P(T)$ given a set T of training sequences. Numerous works [1][5][6] have been proposed to recognize patterns from sequences of events. However, many of them are aimed at discovering patterns that are more general with richer semantic meanings. Although it is feasible to revise some of them to generate maximal patterns as we defined, the performance of the revised algorithms will not be comparable to a specialized one.

We perform a sequential scan of the training sequences. For each never-inspected-before system call e, we determine all maximal patterns including e in an iterative manner. For each iteration, the following operations are performed:

1. We expand each instance of a frequent i-pattern in the "forward" direction of the respective training sequence to form i+1-patterns.
2. Certain i-pattern instances are expanded in the "backward" direction of the respective training sequence to generate maximal patterns.
3. Inspection of both directions is pruned based on a certain observation.

After all the maximal patterns are recognized, system call instances that never participate in any of the maximal patterns are identified and output. Details of the algorithm will be published in the future.

4. Discover Overlap Relationships Between Patterns

In the last section, we discussed how to identify maximal patterns from the training sequences. In the current section, we examine these maximal patterns to discover their relationships. More specifically, for each maximal pattern $p \in P(T)$, we find all the maximal patterns overlapping with p, and determine the corresponding *overlapping points*. We also identify all *terminable* patterns in $P(T)$. Internally, maximal patterns are organized into adjacency lists with each list corresponding to a maximal pattern. Overlapping and terminal information is also recorded for each maximal pattern.

5. Pattern Matching Module

In this section, we present a novel pattern matching module that detects both intra-pattern and inter-pattern mismatches.

During the intrusion detection stage, system calls of various processes of the application under protection are captured at real time. Before being sent to the pattern matching module, system calls are filtered, translated and aggregated, in the same way as in the training phase. The pattern matching module processes one system call at a time.

The pattern matching module verifies the intra-pattern and inter-pattern relationship by traversing the adjacency lists introduced in Section 4. Basically, a Pending Pattern Table (*PPT*) is maintained for each process of the application being monitored. The *PPT* records all legitimate traversing paths. A mismatch counter is incremented when it is impossible to further traverse the adjacency lists (i.e. the *PPT* is empty). An alarm is raised when at least l consecutive mismatches are encountered, where l is a user defined threshold.

6. Experimental Study

To assess the proposed technique, we compare it with an implementation of a building-block-based method similar to [8]. Hereafter, the building-block scheme is referred to as "the reference technique". In our experiments, we applied both techniques to system call trails of the *login* and *sendmail* applications executed under various scenarios. The test data sets can be downloaded from the website of the University of New Mexico (http://www.cs.unm.edu/~immsec/).

We compared both techniques based on their effectiveness, the size of the pattern database, and time efficiency. To measure the effectiveness, we count the number of sequences reported as abnormal. The length of a reported abnormal sequence must be greater than or equal to a predefined threshold, l. The sizes of the pattern databases are measured by counting the internal nodes of the respective data structures (tree for the referenced technique). The time efficiency of both pattern matching modules is determined by their average *PPT* sizes.

The *login* data set was used to test the ability of the proposed technique to detect Trojan horse attacks. We used all 24 normal traces to train the system. Two types of Trojan horse intrusion scripts were employed to attack the target system. One of the two scripts was recovered from an installation of Linux root kit based on a Linux version which is different from the one used to collect normal data. To achieve a stricter test, a second type of intrusion code was "home-grown" by UNM. The experimental result is very promising. With approximately the same time and space cost, our scheme detected at least 100% more anomalies than the referenced technique without raising any false alarm.

The *sendmail* data set has a total of 311 normal sequences, containing approximately 1.5 million system calls. 64 sequences remained after aggregation and duplicate reduction. Of these 64 sequences, 57 were selected to train the system, accounting for approximately 70% of all the system calls. The remaining trails were used to test for false alarm rate. Many intrusion scripts were implemented to generate anomalous behaviors. Again, with approximately the same time and space overhead, our approach detected on average 70% more anomalies than the reference technique. In particular, when the threshold l is set to 12, our scheme detected all the intrusions. The reference technique, on the other hand, failed to raise any alarm for one particular intrusion. However, in terms of false alarm rate, the referenced technique performed slightly better. It did not raise any false alarm while our scheme raised 2 false alarms when l is set to 12.

7. Concluding Remarks

In this paper, we propose a technique that detects intrusions based on both intra-pattern and inter-pattern anomalies. Our contributions are as follows:

1. We refined the definition of *maximal pattern* based on the definition given in [8].
2. An algorithm was proposed to identify maximal patterns in given training sequences.
3. Techniques were developed for identifying and storing overlap relationship between patterns.
4. An efficient pattern matching algorithm was designed.

The proposed technique was tested against a method similar to the one presented in [8] using the popular *sendmail* and *login* data sets. The experimental results indicate that our scheme

1. Detected significantly more anomalies with time and space efficiency similar to the technique proposed in [8],
2. Identified an intrusion missed by the technique proposed in [8], and
3. Achieved a very low false alarm rate.

We, thus, conclude that overlap relationship between patterns is important to intrusion detection. The concept is simple and inexpensive to implement.

8. References

[1] Rakesh Agrawal, Ramakrishnan Srikant. Mining sequential patterns. In *Proc. of the 11th International Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[2] Stephanie Forrest, Steven A. Hofmeyr, Anil Somahaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120-128. IEEE Computer Society, IEEE Computer Society Press, May 1996.

[3] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 202-212. IEEE Computer Society, IEEE Computer Society Press, May 1994.

[4] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. Journal of Computer Security, 6(3):151-180, 1998.

[5] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1997. **1**(3): p. 259-289.

[6] Ramakrishnan Srikant, Rakesh Agrawal. Mining sequential patterns: generalizations and performance improvements. In *International Conference on Extending Database Technology* (*EDBT*'1996).

[7] Andreas Wespi, Marc Dacier, and Hervé Debar. An intrusion-detection system based on the Teiresias pattern-discovery algorithm. In Urs E. Gattiker, Pia Pedersen, and Karsten Petersen, editors, *Proceedings of EICAR '99*, Aalborg, Denmark, February 1999. European Institute for Computer Anti-Virus Research. ISBN 87-987271-0-9.

[8] Andreas Wespi, Marc Dacier, and Hervé Debar. Intrusion detection using variable-length audit trail patterns. In Hervé Debar, Ludovic Mé, S. Felix Wu, editors, *Proceedings of RAID 00, Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, October 2000.

[9] Andreas Wespi, Marc Dacier, Hervé Debar, and Mehdi M. Nassehi. Audit trail pattern analysis for detecting suspicious process behavior. In *Proceedings of RAID 98*, *Workshop on Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium, September 1998.

[10] Christina Warrender, Stephanie Forrest, Barak Pearlmutter. Detecting intrusions using system calls: alternative data models. IEEE Symposium on Security and Privacy. May 1999.