A Real-Time Intrusion Detection and Protection System at System Call Level under the Assistance of a Grid

Fang-Yie Leu, Yi-Ting Hsiao, Kangbin Yim, Ilsun You

A Real-Time Intrusion Detection and Protection System at System Call Level Under the Assistance of a Grid

Fang-Yie Leu[1,*], Yi-Ting Hsiao[1], Kangbin Yim[2] and Ilsun You[3]
[1]Department of Computer Science, Tunghai University, Taichung, Taiwan
{leufy,g98357001}@thu.edu.tw
[2]Soonchunhyang University, South Korea, [3]Korean Bible University, Korea

Abstract. In this paper, we propose a security system, named the Intrusion Detection and Protection System (IDPS for short) at system call level, which creates personal profiles for users to keep track of their usage habits as the forensic features, and determines whether a legally login users is the owner of the account or not by comparing his/her current computer usage behaviors with the user's computer usage habits collected in the account holder's personal profile. The IDPS uses a local computational grid to detect malicious behaviors in a real-time manner. Our experimental results show that the IDPS's user identification accuracy is 93%, the accuracy on detecting its internal malicious attempts is up to 99% and the response time is less than 0.45 sec., implying that it can prevent a protected system from internal attacks effectively and efficiently.

Keywords: Forensic Features, Intrusion Detection and Protection, Data Mining, Identifying Malicious behaviors, Computational Grid

1 Introduction

Currently, most computer systems use user IDs and passwords to authenticate their users. However, many users often share their login information with their coworkers and request them to assist co-tasks, thereby making the login information as one of the weakest points of computer security. Also, internal hackers, the legal users of a system who attack the system internally, are hard to detect. Shan et al. [1] claimed that OS-level system calls are much more helpful in detecting hackers and identifying a malicious internal user. In this paper, we propose a real time security system, named the Intrusion Detection and Protection System (IDPS for short) which detects malicious behaviors at system call level. The IDPS collects users' system-call-usage histories, and uses data mining and forensic techniques to mine typical system calls and their sequences (together named system call sequences (SC-sequences)), as a user's forensic features generated by the activities that the user often performs. The features are a kind of biological characteristics essential in identifying a user. When a user logs in a computer, the IDPS starts monitoring the user's input system calls so as to detect whether he/she is issuing an attack or not, and identify who the account owner is if IDPS discovers that this user is not the account holder. This system collects system call patterns for user operations, and identifies those system call attack patterns that hackers often use. By a long-term observation on user behaviors, user habits can be effectively identified. Further, The Longest Common Subsequence (LCS) algorithm [2] for pattern and

profile mining and computational clustering are also employed to improve the performance of the IDPS.

2 Related Research

Computer Forensics science is one kind of computer security technologies that analyzes what attackers have done, like sending computer viruses, malware and malicious codes, or issuing Distributed Denial-of-Service (DDoS for short) attacks. O'Shaughnessy et al. [3] acquired particular network intrusion and attack patterns from a system log file. The datasets required are extracted from system log files, containing the traces of computer misuse. Therefore, there are obvious potentials for the use of synthetically generated log files that accurately present the traces or patterns of misuse. In this paper, IDPS adopts a similar method to establish attack patterns and evaluate the proposed algorithms. Mahony et al. [4] collected attack patterns and studied on-line systems, including various effective collaborative filtering algorithms, information filtering techniques and expert-system applications. In the previous work [5], we developed a security system to collect forensic features for users on the system command level, rather than on the system call level, by invoking the data mining and forensic techniques. However, if attackers use many sessions to issue attacks, named multi-stage attacks, or DDoS attacks, then due to system processing capability, the intrusion detection system (IDS) cannot thoroughly identify all attack patterns. Giffin et al. [6] provided another example of integrating computer forensics with a knowledge-based system. The system, adopting a predefined model which allows system call sequences to be normally executed, was employed by detection systems to restrict program execution so as to protect an underlying system. This is helpful to detect applications with a series of malicious system calls and automatically identify attack sequences having been collected in knowledge models. Other DoS/DDoS security systems can be found in [7, 8]. The IDPS uses data mining and profiling techniques to respectively analyze and identify user operation characteristics, which as a kind of biological patterns, are essential in identifying a user. This system can analyze and identify attack patterns that hackers often use as well.

3 System Framework

The IDPS as shown in Fig. 1 consists of a system call monitor & filter, mining server, detection server and local computational grid. The system call monitor & filter, as a loadable kernel module embedded in the kernel of the system being considered, collects the system calls submitted to the kernel and stores these system calls in the user's own log file. The mining server analyze users' log data with data mining techniques to identify a user's habit patterns, which are recorded in the user' profiles. Detection server compares users' current inputs with users' computer usage habits collected in their user profiles and attacker profile to, respectively, detect malicious behaviors and identify sources of attacks in a real-time manner. When an intrusion is discovered, the detection server notifies the system call monitor & filter to isolate the user from submitting system calls to the system kernel. Both the detection server and mining server are run on

the local computational grid to respectively accelerate the IDPS's on-line detection and mining speeds, and enhance its detection and mining capability. If a user logs in the system by using other person's login user ID and password, the IDPS can identify who the underlying user is by computing the similarity score between the user's current inputs and the habit patterns collected in the account holders' user profile.

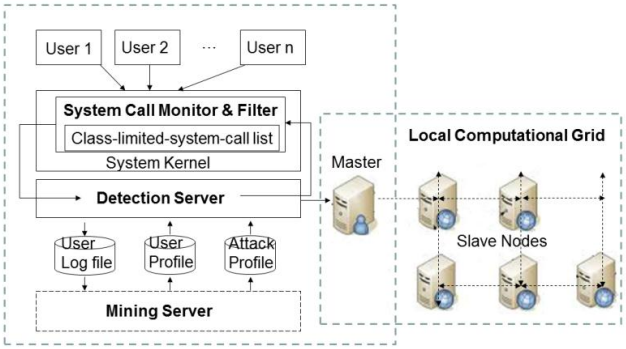

Fig. 1. The IDPS system architecture.

3.1 System Call Monitor & Filter

Due to the possibility of submitting too many system calls to the kernel at the same time, the IDPS may not completely monitor all system calls generated by user-submitted jobs. In this study, we focus on those system calls produced by shell-commands, named shell-command system calls. The Class-limited-system-call list, a component of the system call Monitor & Filter, collects the shell-command system calls that the kernel has received. To know what the typical shell-command system calls are, we use the term frequency-inverse document frequency (TF-IDF, Zhang et al., 2005) algorithm and iData Analyzer [9] tool to analyze the intercepted system calls. Table 1 lists the four commands' representative system calls, which are also the members of the Class-limited-system-call list contained.

Table 1. A part of the Class-limited-system-call list.

Command	**The representative System call**
chmod	fchmodat()
kill	kill()
date	clock_gettime()
rm	unlinkat()

3.2 Mining Server

A mining server extracts those system calls generated by a user from user's log file and counts the time that an SC-sequence appears in this file to produce the user's habit file. The SC-sequences collected in this habit file are then compared with those generated in all other users' habit files in the underlying system to identify the user's user-specific

SC-patterns and those SC-patterns commonly used by all or most users. After that, the user profile is established by attaching a SC-pattern with the corresponding similarity weight. The calculation of the weight will be described later. Fig. 2 illustrates the corresponding control flow.

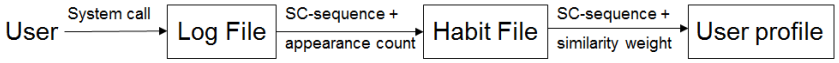

Fig. 2. Control flow of generating a user profile.

Algorithm 1: the algorithm of generating a user habit file

```
Input: a user's log file
Output: the user's habit file
{
  u=|log file| − |sliding window| − 1;
  for(i=0; i < u; i++){1 /* i: L-windows */
    for(j=i+1; j < u; j++){2 /* j: C-windows */
      for (each of ∑_{k=2}^{|L−windows|}(|L − windows| − k + 1) k-grams in cur-
          rent L-window){3
        for (each of ∑_{k'=2}^{|C−window|}(|C − window| − k' + 1) k'-grams in
            current C-window){4
          Compare the k-gram and k'-gram with the LCS
          algorithm;
            if (the identified common pattern already ex-
               ists in the habit file)
               Increase the count of the common pattern
               by one;
            else
               Insert the common pattern into the user's
               habit file with count=1;
        }4}3
      shift C-window one system call right as a new C-
      window;}2
    shift L-window one system call right as a new L-win-
    dow;}1}
```

3.2.1 Mining User and Attack Habits

The IDPS processes the system calls in user log file with a sliding window, named a Log-sliding window (L-window for short), to partition the system calls collected in user's log file along their submitted sequence into k-grams where *k* is the number of a series of consecutive system calls, *k* =2, 3, 4... |sliding window|. In addition, another sliding window of the same size (i.e., the same number of system calls), named Compared-sliding window (C-window for short), is employed for another session in the same user log file. This time, *k'* consecutive system calls, preserving their submitted

sequence, are extracted from a C-window to generate a total of (|sliding window| – k' + 1) k'-grams, k'=2, 3, 4…|sliding window|. Mining server invokes Algorithm 1 to compares each of $\sum_{k=2}^{|L-windows|}(|L-windows|-k+1)$ k-grams with $\sum_{k'=2}^{|C-window|}(|C-window|-k'+1)$ k'-grams by using the LCS algorithm which can reveal the similarity between two strings by skipping noises. After that, the C-window shifts one system call right, and the above mentioned comparison is performed again.

```
execve access getcwd getdents64 = 10
open getdents64 getdents64 exit_group mkdir = 8
execve access rename rename exit_group = 6
access exit_group chdir = 21
```

Fig. 3. A part of a user habit file, in which a line is ended by its appearance count.

Fig. 3 shows an example of a habit file in which a line is a habit, also a SC-sequence or an access pattern, ended by its appearance count. The more frequently a SC-sequence appears, the higher probability the sequence is one of the user's habits. Furthermore, we can apply this algorithm to process an attacker's log file so as to extract his/her usage habits. After legal operations are ripped off, what remains is attack patterns that form a signature file.

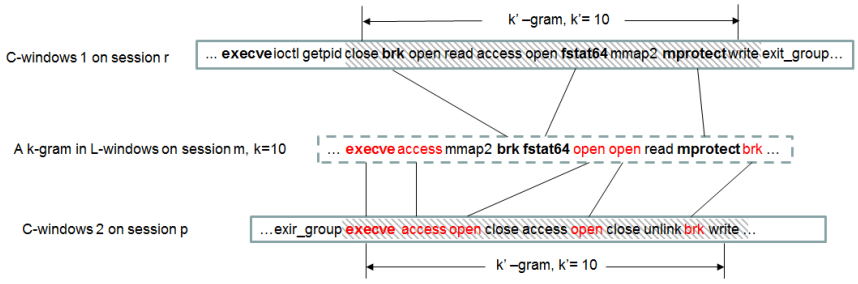

Fig. 4. An example of comparison between an L-window on session m and a C-window of 10 system calls on session p (10 system calls on session r) with the LCS algorithm [9].

Fig. 4 gives two examples. The dash-line rectangle contains an SC-sequence, i.e., a k-gram, extracted from an L-window on session m where k= 10. The solid-line rectangles list two compared sessions, sessions r and p. The shaded areas are C-windows. In session r, system calls that match those in the k'-gram when k' = 10, include brk, fstat64 and mprotect. The remaining system calls, including close, open, read, access, open, mmap2 and write, are noises, and thus are ignored. When k'=k=10, the longest common subsequence between the k-gram in session m and k'-gram in session p, includes execve, access, open, open and brk.

3.2.2 Creating User Profiles

A user profile is a habit file with the appearance counts of SC-patterns being substituted by the patterns' corresponding similarity weights. Given a set of user habit files

D={UP_1,UP_2…UP_N} where N is the number of users and also the number of habit files in the system. Let T={CS_1,CS_2…CS_k} be the set of SC-patterns retrieved from D. Let D_i={UP'_1,UP'_2…UP'_{Mi}}. Each D_i element consists of a set of habit files containing at least one element of T, e.g., CS_i , $CS_i \epsilon T$, and $|D_i|=M_i$. The similarity weight W_{ij} of CS_i in UP_j is defined as

$$W_{ij} = \frac{sf_{ij}}{sf_{ij} + 0.5 + 1.5\,\frac{ns_j}{AVG(ns)}} \times \frac{log\left(\frac{N+0.5}{M_i}\right)}{log\,(N+1)} \tag{1}$$

where i=1,2,3…k, j=1,2,3 …N, sf_{ij} is the appearance count of CS_i in UP_j, ns_j is the total number of SC-patterns in UP_j, $AVG(ns)$ is the average number of SC-patterns an element of D has, and $log((N+0.5)/M_i)/\log(N+1)$ is the inverse characteristics profile frequency (ICPF). We employ Eq. (1), which is commonly used to assign a weight to a term in the information retrieval domain [10], to calculate the similarity weight of CS_i in UP_j.

```
open getdents64 getdents64 exit_group mkdir = 0.198390
execve access getcwd getdents64 = 0.200543
access exit_group chdir = 0.556029
execve access rename rename exit_group = 0.135409
```

Fig. 5. A part of a user profile, in which a line is ended by its similarity weight.

Fig. 5 shows an example of a user profile in which a line is ended by its similarity weight. Once the user profile is created, we send it to all grid nodes.

3.3 Detection Server

Detection server checks to see whether the underlying user is the underlying account holder j or not by calculating the similarity scores between these newly submitted system calls in the user's current session and the usage habits collected in the undering user profile by using the Okapi formula (Robertson et al., 1996), which is commonly used to define the similarity score between documents. Given an unknown user x's current input SC-sequences, denoted by SCSx, the similarity score, e.g., $SimS_{xj}$, between SCSx and user j's user profile UP_j, is defined as

$$SimS_{xj} = \sum_{i=1}^{p} F_{ix} W_{ij} \tag{2}$$

where p is the number of SC-sequences, i.e., a pattern, appearing in both the SCSx and UP_j, F_{ix} is user x's SC-sequence's appearance count in habit file, W_{ij} is the SC-sequence's similarity weight in UP_j. The higher the similarity score, the higher the probability that the user x is the person j who submitted the inputs. The concept of detection server in calculating the user habit is similar to that of the mining server. [5] showed a method to calculate the similarity scores between current session's system-command-sequences and each of the user profiles in the system. The similarity scores between the underlying user and the account holder's profile should be ranked high

within the first x%. If not, the underlying user is recognized as hacker, meaning he/she is not the account holder.

3.3.1 Attack Types

In this study, there are three types of intrusions. Type 0 is defined as the situation where a member of a specific group submits a system call that the group members are prohibited to use. Type I attack is an attack that penetrates a system and submits a sensitive system call that will erase or modify sensitive data or attack the system. A type II attack consists of several attack patterns, each of which is considered as an attack stage. In fact, a hacker mixing specific system calls as noises with an attack pattern can sometimes successfully penetrate a security system. Type 0 and Type I attacks can be detected by the system call monitor & filter by comparing an input system call directly with Class-limited-system-call list. This detection in system call level can protect a system completely. The Type II attack will be identified by detection server.

3.3.2 Detection multi-stage Attack

As stated above, attackers' common attack patterns are presented in the format of a profile. Given current input SC-sequences, we can make sure whether the SC-sequences contain hacker-specific attacks by checking the hacker *j*'s ranking. If the similarity score between the input SC-sequences and hacker profile is within the first x%, the IDPS will issue an alert message, and reply an "unsafe" message accompanied by the user's ID to inform the system call monitor & filter which will isolate the user and prevent him from further use of the system.

4 Experiments

We first install the system call monitor & filter into the main computer, e.g., Redhat ES, of an enterprise system to obtain 10 different categories of user log files as the experimental data for the duration between November 1, 2012 and April 30, 2013. The testbed resources in this study are shown in Table 2.

Table 2. The specifications of the Computational grid (cluster) resources.

Resource	**No.**	**CPU Type**	**No. Core**	**BogoMips /each**	**Mem (GB)**	**Open-Mpi**
Alpha (IDPS)	1	Intel(R) Xeon(R) E5645 @ 2.40GHz	12	4800	25	1.4.1
Beta	1	AMD Opteron(tm) 6174 @ 2.20GHz	48	4400	50	1.4.1
Gamma	1	Intel(R) Xeon(R) E5645 @ 2.40GHz	12	4800	25	1.4.1

4.1 User's similarity score ranking threshold

In this experiment, we define a paragraph size as 3 * |sliding window|. A typical paragraph size is 30 system calls, in which detection server ranks the similarity scores of all user accounts in a system. The purpose is to avoid continuously performing ranking when each system call is input.

```
logid=0, uid=1000, SCS length=30, attacker= 0.1347, account= 0.9578
logid=0, uid=1000, SCS length=60, attacker= 0.3124, account= 0.9634
logid=0, uid=1000, SCS length=90, attacker= 0.7003, account= 0.9284 -> alert
logid=0, uid=1000, SCS length=120, attacker= 0.9501, account= 0.9073 -> alert
```

Fig. 6. Detection server ranks for all user profiles when the paragraph size is 30.

Fig. 6 shows an example, in which the IDPS detects the current input SC-sequence. If the rank of the profile of account holder is lower than the threshold 0.95 or if the rank of an attacker profile is higher than the threshold, the detection server alerts the system manager that the current user is suspected as a hacker.

4.2 Identifying Malicious Programs

A malicious program which is suspected as a Type II attack can be detected by the detection server by checking to see whether the hacker ranking is higher than the per-defined threshold.

```
setreuid32(), setuid32(), setregid32(), setgroups32() = 0.8531
fork(),ptrace(),execve(), ptrace() = 0.9854
socket(), setsockopt(), sendto(), close(), nanosleep() = 0.7638
```

Fig. 7. A part of a hacker profile, in which a line is ended by its similarity weight.

During the experiment, we installed the attack patterns listed in Fig.7 as the hacker injection codes into a running process or issued a DDoS attack to the system. The detection server's attack recognition rate is 99%.

4.3 Detection server accuracy and response time

We use 75% of the user's historical data as the training data to test Algorithm 1 run on the mining server in parallel for creating user profiles. The remaining 25% as the test data is then given to the detection server to simulate the user online inputs. The threshold is set to 0.95. A total of 105400 system calls were collected from 1726 log files. The average length of the SC-sequence sliding window was 10. The user recognition rate of the IDPS is 93%.

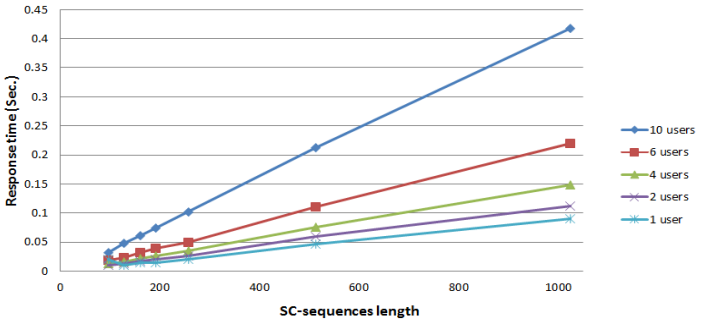

Fig. 8. The experimental response time of the detection server run on 1, 2, 4, 6 and 10 users on 60 processors in parallel.

Fig. 8 shows the experimental result generated by the detection server which employs 60 processors in parallel. The maximum response time is less than 0.45 sec. This means that IDPS can detect malicious behaviors in a real-time manner.

4.4 Comparison with other host-based intrusion detection systems

In this experiment, we compare the IDPS with other host-based intrusion detection systems (HIDS for short). The results are shown in Table 3, in which "✓" means that the system has this function, "×" represents that the system does not have the function, "Δ" shows that the system has the function but not completely equivalent and ART is the average response time.

Table 3. The comparsion of IDPS with other HIDS under attack.

Attack / System	Identify User /ART(sec.)	Type 0 /ART(sec.)	Type I /ART(sec.)	Type II /ART(sec.)	DDoS /ART(sec.)
OSSEC	×	✓ / *60*	✓ / *60*	×	×
SAMHAIN	×	✓ / *60*	✓ / *60*	×	×
McAfee	×	×	×	×	×
Symantec CSP	×	✓ / 2	✓ / 2	Δ / 3	Δ / 3.5
IDPS	✓ / 0.45	✓ / 0	✓ / 0	✓ / 0.45	✓ / 0.45

5 Conclusions and Future Work

In this article, we proposed an approach to find users' habits by employing data mining techniques and profile features. The purpose is to identify the representative SC-sequences for a user. After that, the weight of SC-sequences is computed so that a user's profile can be established. To make sure whether a user is the current account holder or a hacker, the IDPS calculates the similarity scores between the SC-sequences in the current user's input session and each user' usage behaviors. The accuracy is high, making the IDPS a valuable auxiliary subsystem that can assist the system managers to

identify an internal hacker in a closed environment. With this approach, The IDPS can also discover an out-side attacker.

Also, employing a local computational grid environment can shorten the detection server's response time which is less than 0.45 sec. Additionally, to effectively detect an attack and further efficiently reduce the response time, we need a cluster workload monitor and a faster filter and detection algorithm [11, 12]. A mathematical analysis on the IDPS's behaviors will help us to derive its formal performance and cost models so that users can, respectively, determine its performance and cost before using it. They can also increase detection accuracy and improve the decisive rate. Those constitute our future research topics.

References

1. Z. Shan, X. Wang, T. Chiueh and X. Meng, "Safe side effects commitment for OS-level virtualization," The ACM international conference on Autonomic computing, NY, USA, 2011, pp. 111-120.
2. S.J. Shyu and C.Y. Tsai, "Finding the longest common subsequence for multiple biological sequences by ant colony optimization," Computers & Operations Research, vol. 36, no. 1, pp. 73–91, Jan. 2009.
3. S. O'Shaughnessy and G. Gray, "Development and evaluation of a dataset generator tool for generating synthetic log files containing computer attack signatures," International Journal of Ambient Computing and Intelligence, vol. 3, no. 2, pp. 64-76, April 2011.
4. M. O'Mahony, N. Hurley and G. Silvestre, "Promoting recommendations: An attack on collaborative filtering database and expert systems applications," Notes in Computer Science, Springer Berlin, vol. 2453, Aug. 2002, pp. 213-241.
5. F.Y. Leu and K.W. Hu, "Intrusion detection and identification system using data mining and forensic techniques," The Security international conference on Advances in information and computer security, Springer-Verlag Berlin, Heidelberg, 2007, pp. 137-152.
6. J. Giffin, S. Jha and B. Miller, "Automated discovery of mimicry attacks," The international conference on Recent Advances in Intrusion Detection, Springer-Verlag Berlin, Heidelberg, 2006, pp. 41-60.
7. J. Choi, C. Choi, B. Ko, Dongjin Choi, and P. Kim, "Detecting web based DDoS attack using MapReduce operations in cloud computing environment," Journal of Internet Services and Information Security, vol. 3, Issue 3/4, pp. 28-37, November 2013.
8. H.-S. Kang and S.-R. Kim, "A new logging-based IP traceback approach using data mining techniques," Journal of Internet Services and Information Security, vol. 3, Issue 3/4, pp. 72-80, November 2013.
9. R.J. Roger and M.W. Geatz, Data Mining: A tutorial-based primer, Addison Wesley, Addison-Wesley, New York, 2002.
10. D. Zhu and J. Xiao, "R-tfidf, a variety of tf-idf term weighting strategy in document categorization," The IEEE International Conference on Semantics, Knowledge and Grids, Washington, DC, USA, 2011, pp. 83-90.
11. P. Angin and B. Bhargava, "An agent-based optimization framework for mobile-cloud computing," vol. 4, no. 2, pp. 1-17, June 2013.
12. A.P.A. Ling, S. Kokichi, and M. Masao, "Enhancing smart grid system processes via philosophy of Security -*case study based on information security systems,"* Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 3, no. 3, pp. 94-112, Sept. 2012.