# A Grid System Detecting Internal Malicious Behaviors at System Call Level

Fang-Yie Leu* and Yi-Ting Hsiao
Department of Computer Science, Tunghai University, Taichung, Taiwan
{leufy, g98357001}@thu.edu.tw

**Abstract**

In our previous work, we developed a security system which detects malicious behaviors at system-call level. It first creates users' personal profiles for all users of a close environment and an attacker profile for all hackers to keep track of their usage behaviors as the computer forensic features, and then determines whether or not a legally login user $u$ is the account holder or a hacker by comparing $u$'s current computer usage behaviors with the computer forensic features collected in $u$'s personal profiles and the attacker profile. In this study, we implement this security system by using a grid and parallel Message Passing Interface. Experimental results show that the grid system's user identification accuracy is 94%, the accuracy on detecting internal malicious attempts is up to 97% 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**: Computer forensic features, Intrusion detection and protection, Data mining, Computational grid

## 1 Introduction

In the previous work [11], we propose a security system, named Internal Real-Time Intrusion Detection and Protection System (IIDPS for short) which used a single computer to detect malicious behaviors launched to a system at system call level. The IIDPS uses data mining and computer forensic techniques to mine a user's typical system call patterns (SC-patterns for short) as the user's digital forensic features which can be used to identify the user where a SC-pattern is a sequence of system calls that frequently appears in a user's submitted system call sequences (SC-sequences for short). In other words, SC-patterns reflect the activities that the user often performs. When a user logs in to a computer, the IIDPS starts monitoring and checking the user's input system calls to see whether the user is the account holder or not by comparing the behaviors of his/her current inputs with those collected in the user's user profile, and detect whether he/she is issuing an attack by computing the behaviors with the features collected in an attacker profile which gathers malicious behaviors hackers often launched. Once an internal hacker is discovered, the IIDPS isolates the user, alerts system manager, collects digital forensic evidences and analyzes his/her malicious behaviors to improve the IIDPS's future detection capability. In current study, we employ a computational grid [9, 5] as the hardware platform of the IIDPS to speed up the detection of internal malicious behaviors. We call it Parallel IIDPS (P-IIDPS for short), which will be described later. Experiments show that The P-IIDPS can effectively improve performance of the IIDPS.

*Journal of Internet Services and Information Security (JISIS)*, volume: 4, number: 4 (November 2014), pp. 14-24
*Corresponding author: Tunghai University, Taiwan, "9F, No. 187, Sec. 2, Tiding Blvd., Neihu Dist., Taipei City 114, Taiwan, R.O.C." Tel: +886-930069809

## 2 Related Research

Bevel and Gardner [8] addressed that the majority of forensic disciplines focuses on "who" the hacker of a crime is, and the pattern analysis is one of the most important disciplines which deals with "what" having happened. The P-IIDPS uses data mining and computer forensic techniques to analyze user operation characteristics, which as a kind of digital behavior patterns are essential in identifying the corresponding user, i.e., who. This system also analyzes and identifies those attack patterns, i.e., what, frequently used by hackers. Shan et al. [16] claimed that OS-level system calls are much more helpful in detecting hackers and identifying users. However, processing a large volume of system calls, mining malicious behaviors and identifying possible hackers for an instruction will impose a performance overhead. Andalon-Garcia et al. [6] proposed a parallel iterative algorithm for the global alignment of multiple biological sequences with $|n*(n-1)/2|$ comparison pairs. The implementation was developed on a cluster computing through the use of the standard Message Passing Interface (MPI for short) library. Lee et al. [10] presented a fast heuristic algorithm that maps multi-domain applications onto the Coarse-Grained Reconfigurable Architectures (CGRAs for short). The applications implemented on a CGRA show significant improvement of performance.
Although many types of parallel programming models such as shared memory, message passing, threads and data parallel exist, the Symmetric Multi-Processor (SMP for short) cluster with the MPI library [15] has been the mainstream of cluster computing. By employing a grid system, the P-IIDPS returns the detection result within 0.45 sec. The time is shorter if many more SMP nodes are utilized.

## 3 System Framework

The P-IIDPS as shown in Figure 1 consists of a system call monitor & filter, a mining server, a detection server and a computational grid system. The system call monitor & filter, as a loadable module embedded in the kernel of the system being considered, collects those system calls (SCs for short) submitted to the kernel and stores the SCs in the protected system. It also stores the user inputs in the user's log file, which is a file used to keep the SCs submitted by the user. The mining server analyzes log data stored in the protected system with data mining techniques to identify the user's computer usage habits as his/her behavior patterns, which are recorded in the user's user profiles. If a user logs in to the system by using another person's login ID and password, the P-IIDPS identifies who the underlying user is in real-time by computing the similarity score between the user's current inputs, i.e., SCs, and the behavior patterns stored in the account holders' user profile. When an intrusion is discovered, the detection server notifies the system call monitor & filter to isolate the user so as to prevent him/her from continuously using the system.
The P-IIDPS as shown in Figure 2 includes a mining server run on one of the slave nodes of the grid system. The node has a hostfile which defines computation resources for the mining server, excluding the master node of this grid system. The master node is also called the P-IIDPS node. The detection server run on the P-IIDPS node also has a hostfile which defines computation resources for the detection server. The computational grid accelerates the P-IIDPS's mining and on-line detection speeds, and enhances its mining and detection capabilities.

### 3.1 Mining Server

A mining server extracts those system calls generated by a user from the user's log file and counts the time that a SC-pattern appears in this file to produce the user's habit file. A user habit file is generated by invoking the Longest Common Subsequence (LCS for short) algorithm [17] which performs $|n*(n-1)/2|$ times of $<$ L-window, C-window $>$ pair-wise comparison [6] where $n = |SC-sequence| -$

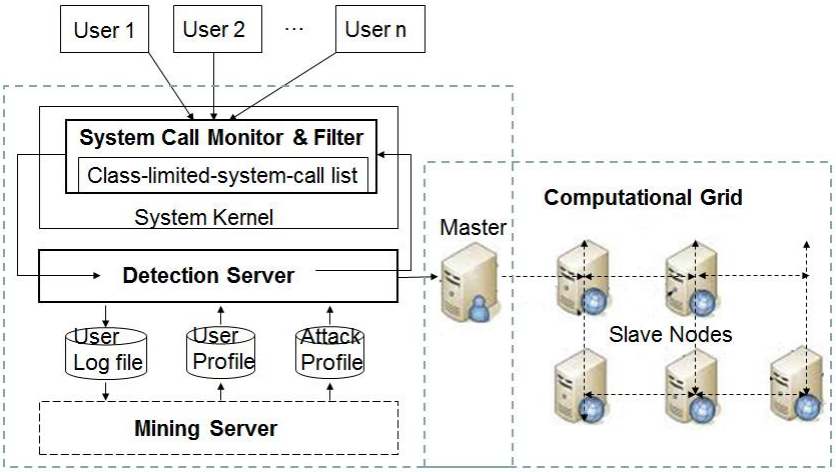


Figure 1: The P-IIDPS system architecture

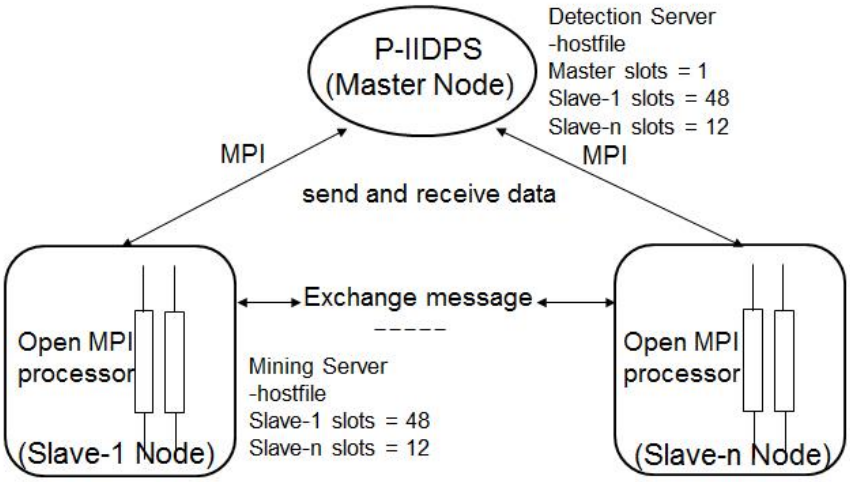


Figure 2: The SMP parallelism architecture of P-IIDPS

$|slidingwindow|+1$. Each pair-wise comparison has $\sum_{k=2}^{|L\text{-}window|}(|L\text{-}window|-k+1)*\sum_{k'=2}^{|C\text{-}window|}(|C\text{-}window|-k'+1)$ times of $<$ k-gram, k'-gram $>$ comparisons.

Algorithm 1 generates a user's profile in parallel. The master node $p_{master}$ computes the total numbers of $<$ k-gram, k'-gram $>$ comparison pairs (see lines 3 and 4) and calculates computational data size for each slave node $p_i$ (see line 5). Next, the master node sends $SCS_u$ and $p_i$'s computational data size to $p_i, 1 \leq i \leq NP$, where $NP$ is the number of slave nodes (see lines 6 and 7). On receiving $SCS_u$ sent by $p_{master}$, $p_i$ calculates its own start point (denoted by *start-pt*) and end point (denoted by *end-pt*) by using the index of current processor's ID, i.e., $i$, and the data size assigned to it (see line 9 to line 11). $p_i$ then compares the k-gram and k'-gram pairs between its *start-pt* and *end-pt*. After that, it returns the identified SC-patterns and their corresponding appearance counts to the $p_{master}$ (see lines 12 and 13). When $p_{master}$ receives the results from any slave node $p_{any}$ by using Recv() function (see line 16) where $p_{any}$ may be $p_0, p_1, p_2, \ldots or p_{NP-1}$, it puts the same SC-pattern received from different slave nodes together and sums up the SC-pattern's accompanied appearance counts to establish the user's habit file (see

**Input**: $N$ users' log files
**Output**: $N$ users' user profiles
1 for (each $u$ of the $N$ users) { /* performed by $p_{master}$ */
2 Retrieve $u$'s SC-sequence as $SCS_u$;
3 $No.of pattern = |SCSu| - |slidingwindow| + 1$;
4 $No.of pair = (No.of pattern * (No.of pattern - 1))/2$; /* total number of < k-gram, k'-gram > comparison pairs */
5 $size = \lceil No.of pair/|NP| \rceil$; /* calculate slave node's computational data size */
6 for (each $p_i$ in $NP$) { /* performed by $p_{master}$ */
7 Send($SCS_u$ and computational data size, $p_i$);}
8 parallel for (each $p_i$ in $NP$) /* each slave node $p_i$ */
9 Recv($SCS_u$ and computational data size, $p_{master}$);
10 $start\text{-}pt = i * size$; /* start point */
11 $end\text{-}pt = start\text{-}pt + size - 1$; /* end point */
12 Compares all k-gram and k'-gram pairs located between $start\text{-}pt$ and $end\text{-}pt$ by invoking the LCS Algorithm;
13 Send(SC-patterns and their appearance counts, $p_{master}$);
14 end parallel
15 for (each slave node $p_{any}$ in $NP$) { /* performed by $p_{master}$ */
16 Recv(SC-patterns and their appearance counts, $p_{any}$);
17 for (each unique SC-pattern received) {
18 Sums up the SC-pattern's appearance counts received from all slave nodes to produce $u$'s habit file; }}}
19 for (each $u$ of the $N$ users) { /* performed by $p_{master}$ */
20 Generates $u$'s user profile;
21 Copies $u$'s user profile to each grid node; }

**Algorithm 1:** generating a user's profile in parallel

line 15 to line 18). After collecting all users' habit files, $p_{master}$ calculates each SC-pattern's similarity weight [11] for each user's user profile in which the appearance count of an SC-pattern is substituted by the SC-pattern's similarity weight. After that all the user profiles are distributed to each grid computer (see line 19 to line 21). With parallel processing, the time required by $p_{master}$ to wait for the completion of identifying of SC-patterns and their appearance counts will be shorten greatly.

To show how L-window and C-window are compared in mining server, an example SC-sequence 1, 2, 3, 4, 5, 6, 7, 8 is given, and the size of a sliding window, i.e., $|slidingwindow|$, is set to 5. The comparison performed by the mining server in Algorithm 1 is as follows.

$< \underline{12345}, 23456 >_1 \Rightarrow < \underline{12345}, 34567 >_2 \Rightarrow < \underline{12345}, 45678 >_3$
$< \underline{23456}, 34567 >_4 \Rightarrow < \underline{23456}, 45678 >_5$
$< \underline{34567}, 45678 >_6$

where the format of $< \underline{X}, Y >_z$ represents the SC-sequence $\underline{X}$ contained in the L-window, SC-sequence $Y$ collected in the C-window, and the subscript $z$ is the sequence number of comparing a k-gram and k'-gram pair.

### 3.2 Detection Server

Given an unknown user $u$'s current input SC-sequence, denoted by $SCS_u$, the similarity score between $SCS_u$ and user $j$'s user profile $UP_j, 1 \leq j \leq N$, where $N$ is the number of users, is defined as

$$Sim(j,u) = \sum_{i=1}^{p} F_{iu} * W_{ij} \tag{1}$$

in which $p$ is the number of SC-patterns appearing in both the $SCS_u$ and $UP_j$, $F_{iu}$ is the appearance count of SC-pattern $i$ identified in $SCS_u$, and $W_{ij}$ is the similarity weight of SC-pattern $i$ collected in $UP_j$. The higher the similarity score, the higher the probability that the user $u$ is the person $j$ who submitted the $SCS_u$.
The detection server adopts the coarse-grained architecture [10] and dynamic process scheduling [18] to control the SMP environment and distribute input system calls to selected slave nodes. The dynamic process scheduling equation [19] is defined as

$$f_{dps}(x,y) = \left\lceil \frac{N}{x} * \frac{1}{1+e^{30-y}} \right\rceil \tag{2}$$

where $x$ is the number of current online users, rather than the total number of users of a system, $y$ is current SC-sequences length, $N$ is the number of available processors and the sigmoid function defined as $\frac{1}{1+e^{30-y}}$ in which the constant 30 represents the $3 * |slidingwindow|$ (i.e., 30 system calls where the size of sliding window is 10).

Given an unknown user $u$'s $SCS_u$, and the similarity score between $SCS_u$ and user $q$'s user profile $UP_q$, $1 \leq q \leq N$, where $N$ is the number of users, the decisive rate of a user $r (1 \leq r \leq N)$, denoted by $X(r)$, among the $N$ users is defined as $X(r) = \frac{(N-rank(r))}{N} * 100\%$, $0 \leq X \leq 100$, where $rank(r)$ is the order of user $r$ from the top after the similarity scores of the $N$ users are sorted where $r$'s account is the one $u$ logs in. Let $X = \frac{1}{N}\sum_{r=1}^{N} X(r)$, i.e., the average decisive rate. The decisive rate of a user profile should be within the top $X\%$, i.e., the threshold. If not, $u$ is considered as an intruder, rather than $r$.
When Algorithm 2 is invoked by the detection server, the sliding window left shifts to identify the SC-sequences on each new input system call. Giving the same example SC-sequence, if user $u$'s input system calls are less than or equal to 5, no action is performed. When the user inputs the $6^{th}$ system call, the L-window contains system calls 2, 3, 4, 5 and 6 and the C-windows covers system calls 1, 2, 3, 4 and 5, i.e.,

$< \underline{23456}, 12345 >_1$
The identified SC-patterns are input to $u$'s new habit file $NHF_u$. After the $7^{th}$ system call is input, the comparison will be
$< \underline{34567}, 23456 >_4 \Rightarrow < \underline{34567}, 12345 >_2$
Note that the subscripts 4 and 2 mean they are the $4^{th}$ and $2^{nd}$ comparisons in the above example (see the section of mining server). The identified SC-patterns are also input to $NHF_u$. On receiving the $8^{th}$ system call, the detection server performs comparisons
$< \underline{45678}, 34567 >_6 \Rightarrow < \underline{45678}, 23456 >_5 \Rightarrow < \underline{45678}, 12345 >_3$
From the above examples, we can see Algorithm 1 and Algorithm 2 compare the same < k-gram, k'-gram > pairs. Due to reverse comparison, the time complexity of Algorithm 2 in detecting malicious behaviors is reduced to O ($n^4$), while that of Algorithm 1 is O ($n^6$).

**Input**: user $u$'s current input system calls (each time only one system call), and all users' user profiles
**Output**: the possible internal intruder or an attacker

1 while ( receiving $u$'s input system call $h$ ) { /*master node */
2 $SCS_u = SCS_u \cup h$;
3 if ($|SCS_u|$) >1024 ) { /* $u$'s input is over 1024 */
4 $SCS_u = Right(SCS_u, |slidingwindow|+1)$; } /* $Right()$ retrieves the last $SCS_u$ */
5 Choose a group of slave nodes, denoted by $SSN$, from the grid where $|SSN| = f_{dps}(currentonlineuser, |SCSu|)$; /* allocate slave nodes by invoking Eq. (2) */
6 $No.ofpair = |SCSu| - |slidingwindow|$; /* total number of < k-gram, k'-gram > comparison pairs*/
7 $size = \lceil No.ofpair/|SSN| \rceil$; /* each slave node's data size where $SSN$ is the set of available processors */
8 for (each $p_i$ in $SSN$) { /* master node */
9 Send($SCS_u$ and computational data size, $p_i$); }
10 parallel for (each $p_i$ in $SSN$) /* performed by $p_i$ */
11 Recv($SCS_u$ and computational data size, $p_{master}$);
12 $start\text{-}pt = i * size$; /* start point */
13 $end\text{-}pt = start\text{-}pt + size - 1$; /* end point */
14 Generate $u$'s current habit file between $start\text{-}pt$ and $end\text{-}pt$ by invoking the LCS Algorithm;
15 for (each user $g, 1 \leq g \leq N$) { /* $N$ is the number of online users in this system */
16 Calculate the similarity score between $u$ and $g$'s user profile by invoking Eq. (1);
17 Send(the part of $u$'s similarity scores, $p_{master}$);
18 end parallel
19 for (each slave node $p_{any}$ in $SSN$) { /* master node */
20 Recv(part of $u$'s similarity scores, $p_{any}$);
21 Accumulate $u$'s similarity scores between $NHF_u$ and each system user;}
22 if (($|SCS_u|$ mod paragraph size ) == 0) { /* master node */
23 if ((the decisive rate of user $u$'s profile <threshold$_1$) or (the decisive rate of a hacker's profile >threshold$_2$)) {
24 Alert system manager that $u$ is a suspected internal intruder or an attacker; }}}

**Algorithm 2:** Detecting an internal intruder or an attacker in parallel

Algorithm 2 detects an internal intruder or an attacker in parallel. The current habit file $NHF_u$ is established by $|SCS_u| - |slidingwindow|$ pair-wised comparison where $|SCS_u|$ is limited ranging between $|slidingwindow|+1$ and 1024. When the length is less than that of a sliding window, the detection server does not compare the k-grams and k'-grams since they are too short to be compared. In this case, the detection server keeps the last input system calls until the length achieves $|slidingwindow|+1$. When the SC-sequence $|SCS_u|$ >1024 (see lines 3 and 4), according to our study, the time required to process the $SCS_u$ is long so the detection server employs the dynamic process scheduling to select the number of slave nodes based on Eq.(2) (see line 5) and then sends $SCS_u$ to the selected slave group, denoted by $SSN$ (see lines 6 and 9). Each member of the group retrieves the portion of $u$'s input SC-sequence to be aligned, calculates appearance counts for the part of $u$'s input SC-sequence assigned to this slave node and returns the calculated similarity scores between the assigned part of $u$'s SC-sequence and all users' profiles in the grid node (see line 10 to line 18).

After receiving the users' similarity scores from all slave nodes, the master node sums up and accumulate all the scores (see line 19 to line 21). Each time when the length of newly received $SCS_u$ meets the pre-defined paragraph size (see line 22), the P-IIDPS judges whether $u$ is an internal intruder by checking to see whether the decisive rate of the account holder is smaller than the pre-defined threshold$_1$, or whether $u$ is an attacker, i.e., dummy user, if the rank of the attacker profile is higher than its pre-defined threshold$_2$ (see line 23). If yes, it concludes that $u$ is an internal intruder or attacker and then alerts the system manager.

# 4 Experiments

To evaluate the P-IIDPS, we first installed the system call monitor & filter into the main computer of an enterprise's production system to obtain 12 user's log files in the duration between November 1, 2013 and April 30, 2014. The testbed configuration and resources in this study are shown in Figure 3 and Table 1. Table 2 shows the measured bandwidth between two arbitrary nodes of our computational grid.

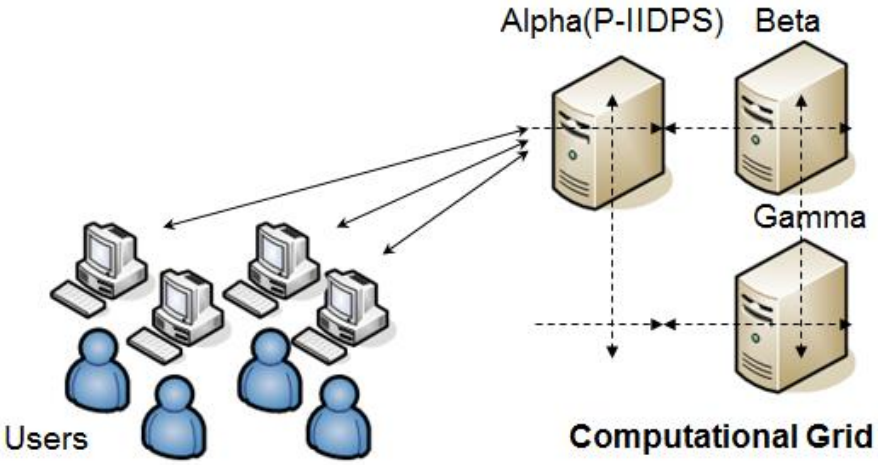


Figure 3: The logical configuration of the tastbed employed in this study

| Resource | CPU Type | No. of Processor | Bogo Mips /each | Mem (GB) | LAN | Open-Mpi |
|---|---|---|---|---|---|---|
| Alpha (P-IIDPS) | Intel(R) Xeon(R) E5645@ 2.40GHz | 12 | 4800 | 25 | Ethernet Fiber | 1.4.1 |
| Beta | AMD Opteron(tm) 6174@ 2.20GHz | 48 | 4400 | 50 | Fiber | 1.4.1 |
| Gamma | Intel(R) Xeon(R) E5645@ 2.40GHz | 12 | 4800 | 25 | Fiber | 1.4.1 |

Table 1: Specification of the members of the grid system

| Point-to- point | | LAN (Real Bandwidth) |
|---|---|---|
| Alpha (P-IIDPS) | Users | Ethernet (82 94 Mbps) |
| Alpha (P-IIDPS) | Beta | Fiber (18705 Mbps) |
| Alpha (P-IIDPS) | Gamma | Fiber (22632 Mbps) |
| Beta | Gamma | Fiber (19412 Mbps) |

Table 2: The bandwidth between two arbitrary nodes of our computational grid

### 4.1 Parallel speedup

The speedup $S(n)$, defined as the ratio of the time consumed by a single processor $T(1)$ over that spent by employing $n$ parallel processors $T(n)$, is expressed as follows.

$$S(n) = \frac{T(1)}{T(n)}$$

Table 3 shows the response times of mining server and detection server on a single processor and 60 processors, when different SC-sequence lengths are given.

| SC-sequences length | 64 | 128 | 192 | 256 | 512 | 640 | 896 | 1024 |
|---|---|---|---|---|---|---|---|---|
| Mining Server $T(1)$ | 16.34 | 123.53 | 413.59 | 635.02 | 2910.7 | 4502.4 | 11104 | 17003 |
| Mining Server $T(60)$ | 0.611 | 1.883 | 4.484 | 5.983 | 23.144 | 30.026 | 63.581 | 97.452 |
| Detection Server $T(1)$ | 0.2221 | 0.5131 | 0.8286 | 1.1126 | 2.3968 | 2.9893 | 4.2574 | 4.8556 |
| Detection Server $T(60)$ | 0.1139 | 0.0106 | 0.0151 | 0.0203 | 0.0471 | 0.0582 | 0.0811 | 0.0905 |

Table 3: The bandwidth between two arbitrary nodes of our computational grid

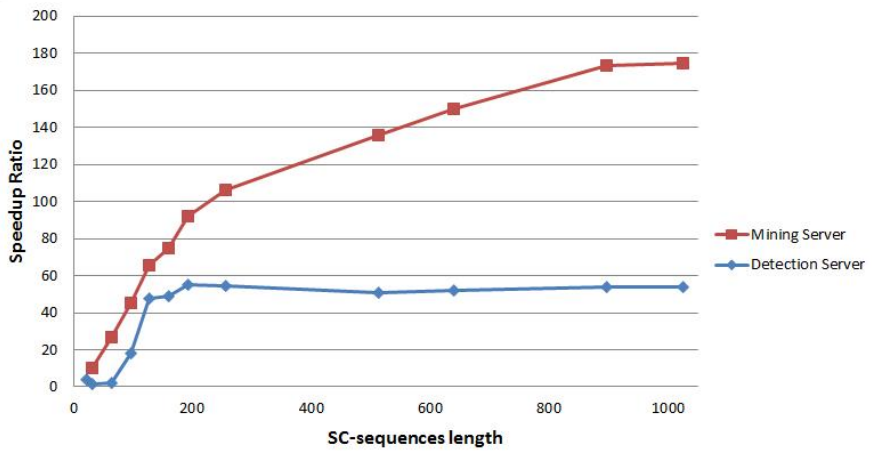


Figure 4: The speedup ratios of the mining server and detection server, both of which consist of 60 parallel processors

The speedup ratios of the mining server and detection server are shown in Figure 4, in which we can see that the testbed grid system with 60 processes can process once a time a SC-sequence of 1024 system calls efficiently. The detection server speedup ratio is $log_2 n$, following the Minsky's Conjecture theorem [13], over thousands of processes, meaning that the detection server has too many sequential parts and a

huge amount of messages are delivered among parallel processors, thus unable to be executed effectively in parallel. That is why the speedup curves do not rise when SC-sequence length is longer than 200.

### 4.2 Detection server accuracy and response time

Given a known user $u$'s current input SC-sequence, i.e., $SCS_u$, if the similarity score between the $SCS_u$ and $u$'s user profile is ranked within the top $X\%$ among the similarity scores between the $SCS_u$ and each of the 12 users' user profiles, as mentioned above, the decisive rate is $X$. The experimental average decisive rate is 92.91% ($= \frac{1}{12}\sum_{i=1}^{12}\frac{12-rank(i)}{12}$, in which i is the node identity of a system user). Therefore, the decisive rate threshold is set to 90%.

This experiment was performed ten times. Each time 75% of users' computer usage history is used as the training data for the mining server (Algorithm 1) to create user profiles. The remaining 25% were test data for detection server (Algorithm 2) to test whether the decisive rate of $u$ is lower higher than the threshold. Table 4 shows the detection accuracy of user recognition.

| User Account | Times of being an account holder | Times of being an attacker (Alarm) | Detection accuracy |
|---|---|---|---|
| root | 100.03 | 5.97 | 94.36% |
| oralce | 110.2 | 6.8 | 94.18% |
| mq | 90.43 | 5.57 | 94.19% |
| ticketing | 191.58 | 11.42 | 94.37% |
| ... | ... | ... | ... |
| cm | 91.02 | 4.98 | 94.81% |
| audit | 105.89 | 2.11 | 98.04% |

Table 4: Detection accuracy of user recognition

Given a $SCS_u$, we can determine whether the $SCS_u$ contains hacker-specific attack patterns or not by employing the process similar to that of judging whether $u$ is the account holder of the account that $u$ logs in or not. If the decisive rate of the hacker profile is higher than threshold 90%, we then suspect that $u$ is a hacker (see line 23 of Algorithm 2) and the P-IIDPS will send a syslog alert message with the user's ID to system manager, record digital forensic evidence and isolate the user from the system. In this experiment, we evaluate 4 internal attacker packages, including Strace [4], GDB [2], TNF (DDoS) [1] and Linux Rootkit [3]. Table 5 lists the detection accuracies of internal attacks.

| Program | True positive | True negative | False positive | False negative | Detection accuracy |
|---|---|---|---|---|---|
| Strace | 49.7% | 48.3% | 1.4% | 0.6% | 98% |
| GDB | 49.2% | 47.5% | 2.5% | 0.8% | 96.7% |
| TFN (DDoS) | 51% | 48.8% | 0.15% | 0.05% | 99.8% |
| Linux Rootkit | 48.5% | 47.8% | 2.2% | 1.5% | 96.3% |

Table 5: Detection accuracy of internal attacks

Figure 5 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 on 12 users.

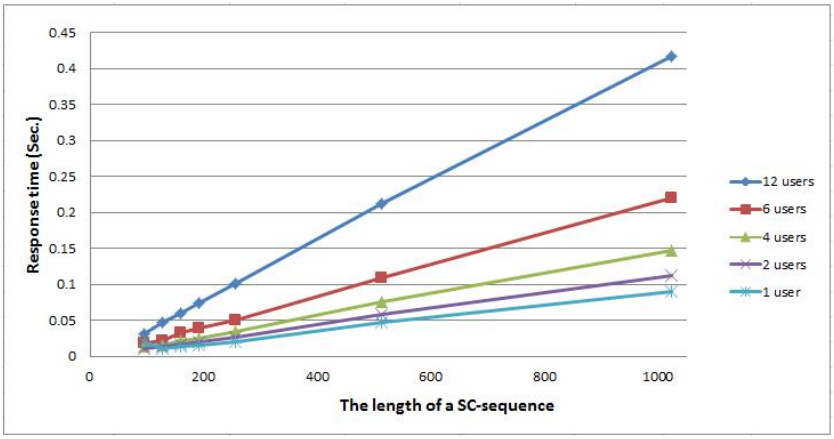


Figure 5: The response time of the detection server on 60 processors in parallel individually given 1, 2, 4, 6 and 12 users' log files

## 5 Conclusions and Future work

In this article, we proposed a grid system which detects malicious behaviors at system-call level. It compares those k-grams and k'-grams retrieved from users' log files and employs Algorithm 1 to generate user profiles in parallel. After that, the P-IIDPS utilizes Algorithm 2 to detect malicious behaviors also in parallel and make sure whether a user is the current account holder in a real-time manner. The attack detection accuracy is high in identifying an internal hacker. Also, employing a grid system can shorten the detection server's response time which is less than 0.45 sec on 12 users.

Additionally, to effectively detect an attack [14] and further efficiently reduce the response time, we need a smart dynamic process workload monitor and faster detection algorithm [7, 12] since it can increase the detection accuracy and improve the decisive rate.

## References

[1] Tribe flood network or tnf. `http://en.wikipedia.org/wiki/TribeFloodNetwork`, December 2012.

[2] Gdb: The gnu project debugger. `http://sources.redhat.com/gdb/`, August 2013.

[3] Recognizing and recovering from rootkit attacks. `http://cecs.wright.edu/~pmateti/Courses/499/Fortification/obrien.html`, September 2013.

[4] Strace. `http://sourceforge.net/projects/strace/`, June 2013.

[5] J. C. Adams and T. H. Brom. Microwulf: a beowulf cluster for every desk. *ACM SIGCSE Bulletin*, 4(1):121–125, March 2008.

[6] I. R. Andalon-Garcia, A. Chavoya, and M. E. M.-C. na. A parallel algorithm for multiple biological sequence alignment. In *Proc. of the 9th International Conference on Information Processing in Cells and Tissues (IPCAT'12), Cambridge, UK, LNCS*, volume 7223, pages 264–276. Springer-Verlag, March-April 2012.

[7] P. Angin and B. Bhargava. An agent-based optimization framework for mobile-cloud computing. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 4(2):1–17, June 2013.

[8] T. Bevel and R. M. Gardner. *Bloodstain pattern analysis with an introduction to crime scene reconstruction with an introduction to crime scene reconstruction, 3rd ed.* New York: CRC Press, 2008.

[9] M. Ku, D. Min, and E. Choi. Scarex: A framework for scalable, reliable, and extendable cluster computing. In *Proc. of the 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT'10), Seoul, Korea*, pages 966–972. IEEE, November-December 2010.

[10] G. Lee, K. Choi, and N. D. Dutt. Mapping multi-domain applications onto coarse-grained reconfigurable architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):637–650, May 2011.

[11] F.-Y. Leu, Y.-T. Hsiao, K. Yim, and I. You. A real-time intrusion detection and protection system at system call level under the assistance of a grid. In *Proc. of the Information & Communication Technology-EurAsia Conference 2014 (ICT-EURASIA'14), Bali, Indonesia, LNCS*, volume 8407, pages 375–385. Springer-Verlag, April 2014.

[12] A. P. A. Ling, S. Kokichi, and M. Masao. Enhancing smart grid system processes via phi-losophy of security -case study based on information security systems. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 3(3):94–112, September 2012.

[13] M. Minsky. Form and content in computer science. *Journal of the ACM*, 17(2):197–215, April 1970.

[14] F. Palmieri, U. Fiore, and A. Castiglione. A distributed approach to network anomaly detection based on independent component analysis. *Concurrency and Computation : Practice and Experience*, 26(5):1113–1129, April 2014.

[15] N. B. R., V. K. K. S., M. C. K., and H. K. G. Mpi based cluster computing for performance evaluation of parallel applications. In *Proc. of the IEEE Conference on Information & communication Technologies (ICT'13), JejuIsland, Korea*, pages 1123–1128. IEEE, April 2013.

[16] Z. Shan, X. Wang, T. cker Chiueh, and X. Meng. Safe side effects commitment for os-level virtualization. In *Proc. of the 8th ACM international conference on Autonomic computing (ICAC'11), Karlsruhe, Germany*, pages 111–120. ACM Press, June 2011.

[17] S. J. Shyua and C.-Y. Tsaib. Finding the longest common subsequence for multiple biological sequences by ant colony optimization. *Computer and Operation Research*, 36(1):73–91, January 2009.

[18] L. M. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, January 2011.

[19] P. C. Yong, S. Nordholm, and H. H. Dam. Optimization and evaluation of sigmoid function with a priori snr estimate for real-time speech enhancement. *Journal of Speech Communication*, 55(2):358–376, February 2013.

---

## Author Biography

**Fang-Yie Leu** received his BS, master and Ph.D. degrees all from National Taiwan University of Science and Technology, Taiwan, in 1983, 1986 and 1991, respectively, and another master degree from Knowledge Systems Institute, USA, in 1990. His research interests include wireless communication, network security, Grid applications and Chinese natural language processing. He is currently a workshop organizer of CWECS and MCNCS workshops, a professor of TungHai University, Taiwan, and director of database and network security laboratory of the University. He is also a member of IEEE Computer Society.

**Yi-Ting Hsiao** received his master degree in department of Computer Science, and BS degree in Department of Industrial Engineering, Tunghai University, Taiwan. He is currently a Project Manager and Chief Engineer of MiTAC Infomation Tech. Corp. Taiwan. His areas of interest include software engineering, Linux embedded system, computer forensic and parallel processing.