An Internal Intrusion Detection and Protection System by Using Data Mining and Forensic Techniques

Fang-Yie Leu, Kun-Lin Tsai, *Member, IEEE*, Yi-Ting Hsiao, and Chao-Tung Yang

***Abstract*—Currently, most computer systems use user IDs and passwords as the login patterns to authenticate users. However, many people share their login patterns with coworkers and request these coworkers to assist co-tasks, thereby making the pattern as one of the weakest points of computer security. Insider attackers, the valid users of a system who attack the system internally, are hard to detect since most intrusion detection systems and firewalls identify and isolate malicious behaviors launched from the outside world of the system only. In addition, some studies claimed that analyzing system calls (SCs) generated by commands can identify these commands, with which to accurately detect attacks, and attack patterns are the features of an attack. Therefore, in this paper, a security system, named the Internal Intrusion Detection and Protection System (IIDPS), is proposed to detect insider attacks at SC level by using data mining and forensic techniques. The IIDPS creates users' personal profiles to keep track of users' usage habits as their forensic features and determines whether a valid login user is the account holder or not by comparing his/her current computer usage behaviors with the patterns collected in the account holder's personal profile. The experimental results demonstrate that the IIDPS's user identification accuracy is 94.29%, whereas the response time is less than 0.45 s, implying that it can prevent a protected system from insider attacks effectively and efficiently.**

***Index Terms*—Data mining, insider attack, intrusion detection and protection, system call (SC), users' behaviors.**

I. Introduction

In the past decades, computer systems have been widely employed to provide users with easier and more convenient lives. However, when people exploit powerful capabilities and processing power of computer systems, security has been one of the serious problems in the computer domain since attackers very usually try to penetrate computer systems and behave maliciously, e.g., stealing critical data of a company, making the systems out of work or even destroying the systems.

Generally, among all well-known attacks such as pharming attack, distributed denial-of-service (DDoS), eavesdropping attack, and spear-phishing attack [1], [2], insider attack is one of the most difficult ones to be detected because firewalls and intrusion detection systems (IDSs) usually defend against outside attacks. To authenticate users, currently, most systems check user ID and password as a login pattern. However, attackers may install Trojans to pilfer victims' login patterns or issue a large scale of trials with the assistance of a dictionary to acquire users' passwords. When successful, they may then log in to the system, access users' private files, or modify or destroy system settings. Fortunately, most current host-based security systems [3] and network-based IDSs [4], [5] can discover a known intrusion in a real-time manner. However, it is very difficult to identify who the attacker is because attack packets are often issued with forged IPs or attackers may enter a system with valid login patterns. Although OS-level system calls (SCs) are much more helpful in detecting attackers and identifying users [6], processing a large volume of SCs, mining malicious behaviors from them, and identifying possible attackers for an intrusion are still engineering challenges.

Therefore, in this paper, we propose a security system, named Internal Intrusion Detection and Protection System (IIDPS), which detects malicious behaviors launched toward a system at SC level. The IIDPS uses data mining and forensic profiling techniques to mine system call patterns (SC-patterns) defined as the longest system call sequence (SC-sequence) that has repeatedly appeared several times in a user's log file for the user. The user's forensic features, defined as an SC-pattern frequently appearing in a user's submitted SC-sequences but rarely being used by other users, are retrieved from the user's computer usage history.

The contributions of this paper are: 1) identify a user's forensic features by analyzing the corresponding SCs to enhance the accuracy of attack detection; 2) able to port the IIDPS to a parallel system to further shorten its detection response time; and 3) effectively resist insider attack.

The remainder of this paper is organized as follows. Section II introduces the related work of this paper. Section III describes the framework and algorithms of the IIDPS. Experimental results are shown and discussed in Sections IV and V, respectively. Section VI concludes this paper.

II. Related Works

Computer forensics science, which views computer systems as crime scenes, aims to identify, preserve, recover, analyze, and present facts and opinions on information collected for a

Manuscript received November 30, 2014; revised February 19, 2015; accepted March 22, 2015. Date of publication April 21, 2015; date of current version June 26, 2017.

F.-Y. Leu is with the Department of Computer Science and Department of Information Management, Tunghai University, Taichung 40704, Taiwan (e-mail: leufy@thu.edu.tw).

K.-L. Tsai is with the Department of Electrical Engineering, Tunghai University, Taichung 40704, Taiwan (e-mail: kltsai@thu.edu.tw).

Y.-T. Hsiao is with MiTAC Information Technology Corporation, Taipei 11493, Taiwan (e-mail: g98357001@thu.edu.tw).

C.-T. Yang is with the Department of Computer Science, Tunghai University, Taichung 40704, Taiwan (e-mail: ctyang@thu.edu.tw).

Digital Object Identifier 10.1109/JSYST.2015.2418434

security event [7]. It analyzes what attackers have done such as spreading computer viruses, malwares, and malicious codes and conducting DDoS attacks [8]. Most intrusion detection techniques focus on how to find malicious network behaviors [9], [10] and acquire the characteristics of attack packets, i.e., attack patterns, based on the histories recorded in log files [11], [12]. Qadeer *et al.* [13] used self-developed packet sniffer to collect network packets with which to discriminate network attacks with the help of network states and packet distribution. O' Shaughnessy and Gray [14] acquired network intrusion and attack patterns from system log files. These files contain traces of computer misuse. It means that, from synthetically generated log files, these traces or patterns of misuse can be more accurately reproduced. Wu and Banzhaf [15] overviewed research progress of applying methods of computational intelligence, including artificial neural networks, fuzzy systems, evolutionary computation, artificial immune systems, and swarm intelligence, to detect malicious behaviors. The authors systematically summarized and compared different intrusion detection methods, thus allowing us to clearly view those existing research challenges.

These aforementioned techniques and applications truly contribute to network security. However, they cannot easily authenticate remote-login users and detect specific types of intrusions, e.g., when an unauthorized user logs in to a system with a valid user ID and password. In our previous work [16], a security system, which collects forensic features for users at command level rather than at SC level, by invoking data mining and forensic techniques, was developed. Moreover, if attackers use many sessions to issue attacks, e.g., multistage attacks, or launch DDoS attacks, then it is not easy for that system to identify attack patterns. Hu *et al.* [17] presented an intelligent lightweight IDS that utilizes a forensic technique to profile user behaviors and a data mining technique to carry out cooperative attacks. The authors claimed that the system could detect intrusions effectively and efficiently in real time. However, they did not mention the SC filter. Giffin *et al.* [18] provided another example of integrating computer forensics with a knowledge-based system. The system adopts a predefined model, which, allowing SC-sequences to be normally executed, is employed by a detection system to restrict program execution to ensure the security of the protected system. This is helpful in detecting applications that issue a series of malicious SCs and identifying attack sequences having been collected in knowledge bases. When an undetected attack is presented, the system frequently finds the attack sequence in 2 s as its computation overhead. Fiore *et al.* [19] explored the effectiveness of a detection approach based on machine learning using the Discriminative Restricted Boltzmann Machine to combine the expressive power of generative models with good classification accuracy capabilities to infer part of its knowledge from incomplete training data so that the network anomaly detection scheme can provide an adequate degree of protection from both external and internal menaces. Faisal *et al.* [20] analyzed the possibility of using data stream mining to enhance the security of advanced metering infrastructure through an IDS. The advanced metering infrastructure, which is one of the most crucial components of smart card, serves as a bridge for providing bidirectional information flow between the user domain and the utility domain. The authors treat an IDS as a second-line security measure after the first line of primary advanced metering infrastructure security techniques such as encryption, authorization, and authentication.

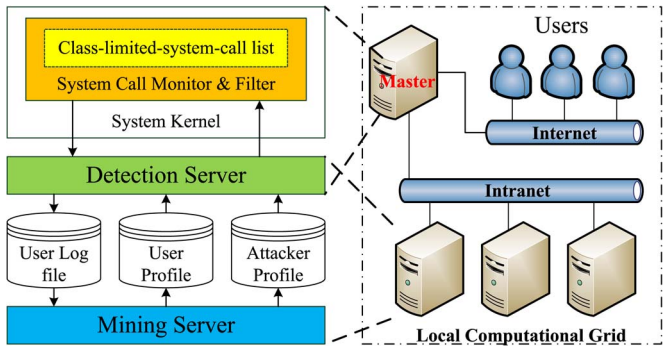

Fig. 1. IIDPS system framework.

III. IIDPS

In this section, we first introduce the IIDPS framework and describe components of the IIDPS in detail. Two algorithms are also presented for generating a user habit file and detecting an internal intruder.

A. System Framework

The IIDPS, as shown in Fig. 1, consists of an SC monitor and filter, a mining server, a detection server, a local computational grid, and three repositories, including user log files, user profiles, and an attacker profile. The SC monitor and filter, as a loadable module embedded in the kernel of the system being considered, collects those SCs submitted to the kernel and stores these SCs in the format of $\langle \text{uid}, \text{pid}, \text{SC} \rangle$ in the protected system where uid, pid, and SC respectively represent the user ID, the process ID, and the SC c submitted by the underlying user, i.e., $c \in$ SCs. It also stores the user inputs in the user's log file, which is a file keeping the SCs submitted by the user following their submitted sequence. The mining server analyzes the log data with data mining techniques to identify the user's computer usage habits as his/her behavior patterns, which are then recorded in the user's user profile. The detection server compares users' behavior patterns with those SC-patterns collected in the attacker profile, called attack patterns, and those in user profiles to respectively detect malicious behaviors and identify who the attacker is in real time. When an intrusion is discovered, the detection server notifies the SC monitor and filter to isolate the user from the protected system. The purpose is to prevent him/her from continuously attacking the system.

Both the detection server and the mining server are run on the local computational grid to accelerate the IIDPS's online detection and mining speeds and enhance its detection and mining capability. If a user logs in to the system by using another person's login pattern, the IIDPS identifies who the underlying user is by computing the similarity scores between the user's current inputs, i.e., SCs, and the behavior patterns

stored in different users' user profiles. In the IIDPS, the SCs collected in the class-limited-SC list, as a key component of the SC monitor and filter, are the SCs prohibited to be used by different groups/classes of users in the underlying system, e.g., a secretary cannot submit some specific privileged SCs. Therefore, commands that generate these SCs will be prohibited to be used by all secretaries.

B. SC Monitor and Filter

An SC in fact is an interface between a user application and services provided by the kernel. Generally, a huge amount of SCs are generated during the execution of a job, i.e., a task or process. For example, when a user changes his/her password by submitting a "passwd" shell command to a Linux operating system, up to 2916 SCs will be generated, including open(), close(), read(), write(), etc. Therefore, it is hard for a system to monitor all SCs at the same time, particularly when many users are running their programs. As a result, we need to filter out some commonly used safe SCs.

To find out what SCs are typical ones generated by a shell command, the statistic model of term frequency-inverse document frequency (TF-IDF) is used to analyze the importance of intercepted SCs collected in a user log file. In the information retrieval domain, the relationship between a term and a document is similar to that between an SC t_i and the command, e.g., j, which generates t_i. The term frequency (TF) employed to measure the weight of the frequency of an SC produced by j is defined as

$$\text{TF}_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^{k=h} n_{k,j}} \tag{1}$$

where $n_{i,j}$ is the number of times that t_i is issued during the execution of j, h is the number of different SCs generated when j is executed, and the denominator $\sum_{k=1}^{k=h} n_{k,j}$ sums up the numbers of times that all these SCs are launched. The inverse document frequency (IDF), the measure of the importance of t_i among all concerned shell commands, is defined as

$$\text{IDF}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|} \tag{2}$$

where $|D|$, the cardinality of D, is the total number of shell commands in the concerned corpus and $\{j : t_i \in d_j\}$ is the set of shell commands d_j, in which each member generates t_i during its execution. The TF-IDF weight of t_i generated by j is defined as

$$(\text{TF-IDF})_{i,j} = \text{TF}_{i,j} \times \text{IDF}_i. \tag{3}$$

In fact, the TF-IDF weight as one of the feature weighting methods in data mining and information retrieval domains increases proportionally to the number of times an SC appears in a user log file, and it can indeed show the importance of a certain SC.

Table I lists four shell commands, including chmod, kill, date, and rm, and the SCs they generate. The SC close(19) appearing in chmod row represents that the SC close() is generated for a total of 19 times when chmod is executed.

TABLE I
SCs AND THEIR GENERATION FREQUENCIES DURING THE EXECUTION OF FOUR SPECIFIC COMMANDS

Command	No. of SCs	System calls generated
chmod	94	close(19), read(3), open(18), execve(1), access(3), brk(3), umask(1), munmap(2), mprotect(4), mmap2(20), stat64(1), fstat64(17), set_thread_area(1), fchmodat(1)
kill	47	read(4), open(5), close(5), execve(1), getpid(1), access(4), kill(1), brk(3), munmap(2), mprotect(5), mmap2(11), fstat64(4), set_thread_area(1)
date	133	read(70), write(1), open(21), close(22), execve(1), clock_gettime(1), …
rm	102	mmap2(20), read(3), open(18), close(20), execve(1), unlinkat(1), …

Thus, the TF weight of close() calculated by using (1) is $0.2021(= 19/94)$ where 94 is the total number of times that SCs are generated during the execution of chmod. The TF weight of the SC kill() during the execution of kill command is $0.02128(= 1/47)$.

According to Table I and (2), the IDF weights for open() and unlinkat() are $0(= \log 4/4)$ and $2(= \log 4/1)$, respectively, since the latter is only generated by the rm command, meaning among the four commands, unlinkat() is a representative SC of rm, and open() is issued by all the four commands, indicating that it cannot represent any one of the four commands. The TF-IDF weight of the remaining SCs can be calculated in the similar manner.

In the IIDPS, all collected data are analyzed by a data mining tool, iData Analyzer [21], in which class predictability and class predictiveness are two parameters utilized to evaluate intraclass and interclass weights, respectively, of an attribute among classified attribute classes. In the following, class predictability is defined in Definition 1, and class predictiveness is defined in Definition 2.

Definition 1 (Class Predictability): Given a class C and a categorical attribute A with $v_1, v_2, v_3, \ldots,$ and v_n as its candidate values, class C's predictability score on $A = v_i$, denoted by $P(A = v_i)$, is defined as the percentage with which A's value in C is v_i. The sum of the predictability scores for attribute A on all its candidate values in C is equal to 1, i.e., $\sum_{\forall T(A=v_i) \in C} P(A = v_i) = 1$ where $T(A = v_i)$ is an instance of C with $A = v_i$. Given an SC t_i and a command j, the predictability score of t_i measures the percentage with which $t_i \in Q$ where Q is the set of SCs produced during the execution of j. (Note that other commands may also generate t_i during their execution.) The predictability score of t_i, denoted by $P(t_i)$, characterizing the execution of j, meets the criterion that $\sum_{\forall t_i \in Q} P(t_i) = 1$.

Definition 2 (Class Predictiveness): Given a class C and a categorical attribute A with $v_1, v_2, v_3, \ldots,$ and v_n as its candidate values, the attribute value v_i's predictiveness score $P(A = v_i)$ is defined as the probability with which an instance T with $A = v_i$ resides in C. The sum of the predictiveness scores for A on all of A's candidate values among classes in $\{C\}$ is equal to 1 where $\{C\}$ is the set of classes, in which each member has

Name	Value	Frequency	Predictability	Predictiveness
Syscall	set_thread_area	1	0.02	0.07
	read	4	0.09	0.02
	open	5	0.11	0.02
	munmap	2	0.04	0.03
	mprotect	5	0.11	0.05
	mmap2	11	0.23	0.03
	kill	1	0.02	1.00
	getpid	1	0.02	0.33
	fstat64	4	0.09	0.01
	execve	1	0.02	0.07
	close	5	0.11	0.01
	brk	3	0.06	0.07
	access	4	0.09	0.05
command	kill	47	1.00	1.00

Fig. 2. Supervised learning results generated by invoking the iData Analyzer Data Mining tool to calculate the class predictability and predictiveness scores for the kill() SC. The high predictiveness score shows that kill() is a representative SC of kill command.

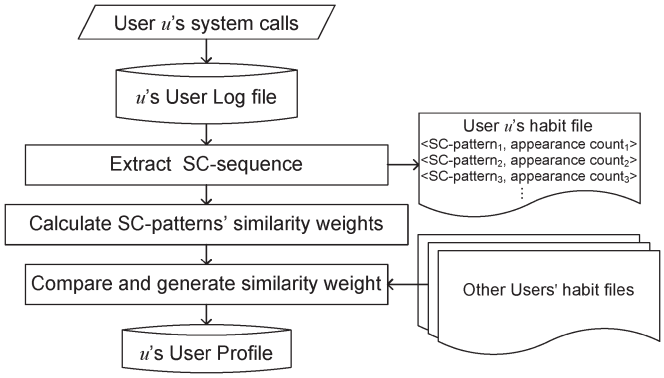

Fig. 3. Control flow of the generation of a user profile.

A as one of its attributes, i.e.,$\sum_{\forall T(A=v_i)\in\{C\}} P(A=v_i)=1$. Let S_j be the set of SCs generated by j. The predictiveness score of an SC t_i, denoted by $P'(t_i)$, is employed to measure the probability with which t_i is a member of S_j among all the shell commands $\{j : t_i \in S_j\}$ generating t_i during their execution. The predictiveness score of t_i among all elements of $\{j : t_i \in S_j\}$ is

$$\sum_{\forall\, t_j\in S_j, j\in\{k:t_i\in S_k\}} P'(t_i)=1. \tag{4}$$

Generally, the training instances of the mining process should contain all the related shell commands and those SCs generated by these commands. With iData Analyzer, the supervised learning output class sheet of the kill command is shown in Fig. 2. The typical SC kill() is one with a small predictability score, i.e., $0.02(=1/47)$, meaning that only one instance of kill() is generated during the execution of kill command, and the SC obtains a high predictiveness score, i.e., $1(=1/1)$, among all class commands, depicting that kill() is one of the representative SCs of kill command.

C. Mining Server

As shown in Fig. 3, a mining server extracts SC-sequence generated by a user u from u's log file, counts the times that a specific SC-pattern appears in the file, and stores the result in $\langle$SC-pattern, appearance counts$\rangle$ format in u's habit file. After this, SC-patterns' similarity weights are calculated to filter out those SC-patterns commonly used by all or most users. Then, the output result is compared with all other users' habit files in the underlying system to further identify u's specific SC-patterns. Finally, the similarity weight is computed to generate u's user profile. The details of these processes will be described later.

Algorithm 1: The algorithm for generating a user habit file
Input: u's log file where u is a user of the underlying system
Output: u's habit file
1. G = |log file| - |Sliding window| ;
   /* |Sliding windows|=|L-window|=|C-window| */
2. for (i=0 ; $i \le G-1$; i++) {
3. for (j=i+1 ; $j \le G$; j++) {
4. for (each of $\sum_{k=2}^{|\text{Sliding window}|}$(|Sliding window| $-k+1$) k-grams in current L-window){
5. for (each of $\sum_{k'=2}^{|\text{Sliding window}|}$(|Sliding window| $-k'+1$) k'-grams in C-window){
6. Compare the k-grams and k'-grams with the longest common subsequence algorithm;
7. if (the identified SC-pattern already exists in the habit file)
8. Increase the count of the SC-pattern by one;
9. else
10. Insert the SC-pattern into the habit file with count=1; }}}}

Fig. 4. Algorithm for generating a user u's habit file.

1) Mining User and Attacker Habits: The IIDPS processes SCs collected in u's log file with a sliding window, named a log-sliding window (L-window for short), which is used to identify consecutive SCs of size |Sliding window| along their submitted sequence and partition the SCs in the window into k-grams where k is the number of consecutive SCs, $k = 2, 3, 4, \ldots,$ |Sliding window|. In this paper, |Sliding window| $=$ 10. In addition, another sliding window of the same size (i.e., the same number of SCs), called compared-sliding window (C-window for short), is employed to identify other SC-patterns also in u's log file. This time, k' consecutive SCs, preserving their submitted sequence, are extracted from a C-window to generate a total of (|Sliding window| $-k'+1$) $k'-$grams, $k' = 2, 3, 4, \ldots,$ |Sliding window|.

The mining server invokes Algorithm 1 shown in Fig. 4 to compare k-grams and k'-grams. At first, all SCs collected in u's log file are treated as a long SC-sequence. When all the $\sum_{k=2}^{|\text{Sliding window}|}$(|Sliding window| $-k+1$) k-grams, derived from the L-window, have been compared with the $\sum_{k'=2}^{|\text{Sliding window}|}$(|Sliding window| $-k'+1$) k'-grams, derived from the C-window, by using the longest common subsequence algorithm [22], which reveals the similarity between two strings by skipping noises, the C-window shifts right one input SC (e.g., originally beginning at position x, and then moves to the position beginning at $x+1$), and the aforementioned comparison is performed again until the C-window covers the last SC-sequence of size |Sliding window|. The next L-window shifts right one SC, and the above comparison continues. The process ends when the comparison, in which the L-window covers the second SC-sequence of size |Sliding window| to the last (i.e., $i = G-1$) and the C-window contains the last SC-sequence of size |Sliding window| (i.e., $j = G$), is completed where $G =$ |log file| $-$ |Sliding window|. Theorem 1 shows that the time complexity of $\langle k$-gram, k'-gram$\rangle$ comparison is $O(n^6)$.

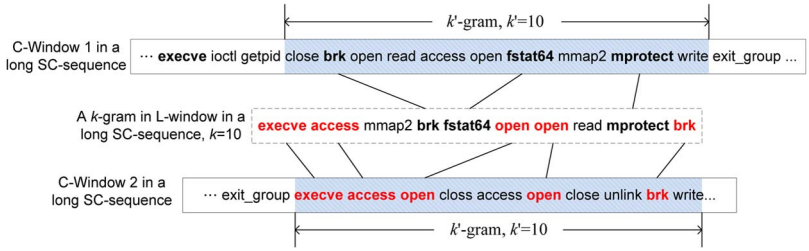

Fig. 5. Example of comparison between k-grams in an L-window and two k'-grams in two C-windows of ten SCs.

Theorem 1: The time complexity of Algorithm 1 is $O(n^6)$ where n is the size of the sliding window.

Proof: Let $m = |\text{SC-sequence}| - (|\text{Sliding window}| - 1)$, which is the number of sliding windows that can be identified in the given SC-sequence. Then, a user profile is generated by invoking $|m^*(m-1)/2|$ times of the $\langle$L-window, C-window$\rangle$ pairwise comparison, and each $\langle$L-window, C-window$\rangle$ pairwise comparison has

$$\sum_{k=2}^{|\text{Sliding window}|} (|\text{Sliding window}| - k + 1) * \sum_{k'=2}^{|\text{Sliding window}|} (|\text{Sliding window}| - k' + 1) \quad (5)$$

times of $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparisons. Let $n = |\text{Sliding window}|$, and let $l = |\text{SC-sequence}|$; the total time of $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparison, denoted by T_{total}, is

$$\begin{aligned} T_{\text{total}} &= \frac{(l-n+1)(l-n)}{2} \times \sum_{k=2}^{n}(n-k+1) \times \sum_{k'=2}^{n}(n-k'+1) \\ &= \frac{(l-n+1)(l-n)}{2} \times \frac{n(n-1)}{2} \times \frac{n(n-1)}{2} \\ &\cong \frac{1}{8}(l-n)^2(n)^4. \end{aligned} \quad (6)$$

This means that the time complexity of $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparison is $O(n^6)$. Of course, if we consider the time complexity on l, it will be $O(l^2)$. □

Fig. 5 gives an example of $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparison. The solid-line rectangles list two compared SC-sequences. A shaded area is a C-window. The dash-line rectangle contains an SC-sequence, i.e., a k-gram, extracted from an L-window where $k = 10$. In the upper rectangle (i.e., marked with C-window 1), SCs that match those in the k'-gram where $k' = 10$ include brk, fstat64, and mprotect (omitting () of an SC for simplicity). The remaining SCs, including close, open, read, access, open, mmap2, and write, are noises and thus ignored. When $k' = k = 10$, the longest common subsequence between the k-gram and the k'-gram in the lower rectangle (marked with C-window 2) includes execve, access, open, open, and brk.

2) Creating User Profiles and Attacker Profiles: In Fig. 3, a user's user profile is a habit file with an SC-pattern's appearance count being substituted by its corresponding similarity weight. We further remove those SC-patterns with similarity weight less than a predefined threshold since they are not the representative ones that can discriminate the user from other users in the underlying intranet. An attack pattern (or a signature), which may be an attacker-specific pattern or a pattern commonly used by attackers, can be identified in the same method. Similarly,

TABLE II
EXAMPLE OF USER HABIT FILES D, SC-PATTERNS T, AND THEIR CORRESPONDING D_i'S

D	T/D_i				
	CS_1/ D_1	CS_2/D_2	CS_3/D_3	...	CS_k/D_k
UH_1	✓	✓			✓
UH_2	✓		✓		✓
UH_3			✓		
...					
UH_{N-1}		✓	✓		✓
UH_N	✓	✓			

an attack pattern that an attacker frequently submits but others have seldom or never submitted will be considered as one of the attacker's representative attack patterns and will obtain a high similarity weight. Hence, signatures collected in an attacker profile (see Fig. 1) can be classified into common signatures and attacker-specific signatures. The latter can be used to identify who the possible attackers are when a protected system is attacked by attacker-specific signatures.

Given a set of user habit files $D = \{UH_1, UH_2, \ldots, UH_N\}$ where N is the number of users, let $T = \{CS_1, CS_2, \ldots, CS_k\}$ be the set of SC-patterns retrieved from elements of D. Let $D_i = \{UH'_1, UH'_2, \ldots, UH'_{Mi}\}$, $1 \le i \le k$, be the set of UH's in which each UH contains the specific element CS_i, $CS_i \epsilon T$, $D_i \subseteq D$, and $|D_i| = M_i$. Examples of D_i are listed in Table II, in which $D_1 = \{UH_1, UH_2, UH_N\}$, $D_2 = \{UH_1, UH_{N-1}, UH_N\}$, $D_3 = \{UH_2, UH_3, UH_{N-1}\}$, and $D_k = \{UH_1, UH_2, UH_{N-1}\}$.

Equation (7), which is commonly used to assign a weight to a term in the information retrieval domain [23], is utilized to give each SC-pattern a similarity weight W_{ij}, which is the weight of CS_i in UH_j where

$$W_{ij} = \frac{f_{ij}}{f_{ij} + 0.5 + \frac{1.5 \times ns_j}{ns_{\text{avg}}}} \times \frac{\log\left(\frac{N+0.5}{M_i}\right)}{\log(N+1)}$$
$$i = 1, 2, 3, \ldots, k, \text{ and } j = 1, 2, 3, \ldots, N \quad (7)$$

in which f_{ij} is the appearance count of CS_i in UH_j, ns_j is the total number of SC-patterns collected in UH_j, ns_{avg} is the average number of SC-patterns that an element of D has, and $\log((N+0.5)/M_i)/\log(N+1)$ is the inverse characteristic profile frequency (ICPF) [23].

D. Detection Server

The detection server captures the SCs sent to the kernel by the underlying user u when u is executing shell commands and

Algorithm 2: Detecting an internal intruder or an attacker
Input: user u's current input SCs, i.e., NSC_u, (each time only one SC is input), and all users' user profiles
Output: u is suspected as an internal intruder or a known attacker
1. $NCS_u = \emptyset$;
2. while (receiving u's input SC, denoted by h) {
3. $NCS_u = NCS_u \cup \{h\}$;
4. if ($|NCS_u|$ > |Sliding window|) {
5. L-window = Right(NCS_u, | Sliding window|); /* Right(x, y) retrieves the last L-window of y from x */
6. for (j= $|NCS_u|$ - |Sliding window| ; j >0 ; j--) {
7. C-window = Mid (NCS_u, j, |Sliding window|);/* Mid (x, y, z) retrieves a sliding window of size z beginning at the position of y from x */
8. Compare k-grams and k'-grams by using the comparison logic employed in Algorithm 1 to generate NHF_u;}
9. for (each user g, $1 \leq g \leq N$)
10. Calculate the similarity score $Sim(u, j)$ between NCS_u and g's user profile by invoking Eq. (8);
11. if (($|NCS_u|$ mod paragraph size) == 0) { /* paragraph size = 30, meaning we judge whether u is an attacker or the account holder for every 30 input SCs */
12. Sort similarity scores for all users;
13. if ((the decisive rate of u's user profile < threshold$_1$) or (the decisive rate of attacker profile > threshold$_2$)){ /* threshold$_1$ is the predefined lower bound of average decisive rate of user u's user profile, while threshold$_2$ is the predefined upper bound of average decisive rate of attacker profile*/
14. Alert system manager that u is a suspected attacker, rather than u himself/herself; }}}}

Fig. 6. Detection server detects whether u is possibly an internal intruder or an attacker.

stores the SCs in the u's log file. After this, the server tries to identify whether u is the underlying account holder or not by calculating the similarity score between the newly generated SCs, denoted by NSC_u, in the u's current inputs (in u's log file) and the usage habits, i.e., forensic signatures (also behavior patterns), stored in u's user profile to verify u. The Okapi model [24], which is utilized to calculate the similarity score between user j's user profile UH_j and an unknown user u's current input SC-sequence, denoted by $Sim(u, j)$, is defined as

$$Sim(u, j) = \sum_{i=1}^{p} F_{iu} \cdot W_{ij} \tag{8}$$

in which p is the number of SC-patterns appearing in both NCS_u and UH_j, F_{iu} is the appearance count of SC-pattern i collected in NCS_u, and W_{ij} produced by invoking (7) is the similarity weight of i in UH_j. The higher the $Sim(u, j)$, the higher the probability, with which u is the person j who submitted NCS_u.

Algorithm 2 shown in Fig. 6 detects an internal intruder. The detection server needs to generate u's temporary habit file by analyzing NCS_u so that F_{iu} is available for later computation. The concept is similar to that of the mining server but different in that the comparison between the L-window and C-window is from the back to the front each time when an SC is input by u, i.e., when the L-window contains the last |Sliding window| SCs, the C-window lefts shift one SC from the L-window and compares the k-grams and k'-grams individually derived from them. After this, for each left shift on the C-window, the two windows compare. The comparison continues until the C-window covers the first |Sliding window| SCs of NCS_u.

In the following, two examples are given to show how the L-window and C-window are compared in Algorithms 1 and 2. Let {1, 2, 3, 4, 5, 6, 7, 8} be the given SC-sequence, and let the size of a sliding window, i.e., |Sliding window|, be 5. The comparison performed by the mining server in Algorithm 1 is shown below. $\langle 12345, 23456\rangle_1 \rightarrow \langle 12345, 34567\rangle_2 \rightarrow \langle 12345, 45678\rangle_3 \rightarrow \langle 23456, 34567\rangle_4 \rightarrow \langle 23456, 45678\rangle_5 \rightarrow \langle 34567, 45678\rangle_6$, where the format of $(X, Y)_z$ represents that the SC-sequence X contained in the L-window and the SC-sequence Y collected in the C-window are compared in the zth comparison in Algorithm 1 where $|X| = |Y| =$ |Sliding window|.

When Algorithm 2 is invoked by the detection server, the sliding window left shifts, i.e., going back, to identify the SC-sequences on each new input SC. If the user's inputs are less than or equal to |Sliding window|, i.e., 5, no action is performed. When the user inputs the sixth SC, the L-window contains SCs 2, 3, 4, 5, and 6 and the C-windows cover SCs 1, 2, 3, 4, and 5, i.e., $\langle 23456, 12345\rangle_1$. After the seventh SC is input, the comparison will be $\langle 34567, 23456\rangle_4 \rightarrow \langle 34567, 12345\rangle_2$. On receiving the eighth SC, the detection server performs comparisons $\langle 45678, 34567\rangle_6 \rightarrow \langle 45678, 23456\rangle_5 \rightarrow \langle 45678, 12345\rangle_3$. In this algorithm, all the identified SC-patterns are input to u's new habit file NHF_u rather than j's habit file where j's account is the one u logs in.

From the examples, we can see that these two algorithms yield the same comparison pairs, i.e., $\langle 12345, 23456\rangle_1$ to $\langle 34567, 45678\rangle_6$ by Algorithm 1 and $\langle 23456, 12345\rangle_1$, $\langle 34567, 23456\rangle_4$, $\langle 34567, 12345\rangle_2$, $\langle 45678, 34567\rangle_6$, $\langle 45678, 23456\rangle_5$, and $\langle 45678, 12345\rangle_3$ by Algorithm 2, meaning that the two algorithms utilize the same comparison contrapositive logic and generate the same NHF_u given the same NCS_u. Theorem 2 shows that the time complexity of detecting an internal intruder or an attacker is O(gn^5).

Theorem 2: The time complexity of detecting an internal intruder or an attacker is O(gn^5) where n is the size of a sliding window, and g is the number of legal users.

Proof: Let n = |Sliding window|, and let g be the number of legal users. Each time when the underlying user u newly inputs an SC and the number of entered SCs is larger than n, the last n SCs will be identified as an L-window. Let $|NCS_u| = l$. Now, in the NCS_u, a total of $l - n + 1$ sliding windows can be found, meaning that the time of sliding window comparison between the L-window and the remaining $l - n$ C-windows is $l - n$. According to Theorem 1, for each pairwise sliding window comparison, there are $(n(n-1)/2) \times (n(n-1)/2)$ $\langle k$-gram, k'-gram$\rangle$ comparisons. Let T_{comp} be the total number of $\langle k$-gram, k'-gram$\rangle$ comparisons in the NCS_u

$$T_{\text{comp}} = (l - n) \times \frac{n(n-1)}{2} \times \frac{n(n-1)}{2}. \tag{9}$$

Since there are g users in the IIDPS, the total time of $\langle k$-gram, k'-gram$\rangle$ comparisons is

$$T'_{\text{total}} = g \times T_{\text{comp}} = g \times (l-n) \times \frac{n(n-1)}{2} \times \frac{n(n-1)}{2}. \tag{10}$$

This indicates that the time complexity of Algorithm 2 is O(gn^5). □

However, in this paper, |Sliding window| is limited to less than 10. Hence, the times consumed for $\langle k$-gram, k'-gram$\rangle$ comparison by Algorithms 1 and 2 are both short.

Definition 3 (Decisive Rate): For the detection server, given an unknown user u's NHF_u and the similarity score between NHF_u and user q's user profile UH_q, $1 \leq q \leq N$, where N is the number of users, the decisive rate of $r(1 \leq r \leq N)$, denoted by x, among the N users is defined as $x = (N - rank(r))/N$, $0 \leq x \leq 1$, 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.

The decisive rate of r's user profile x should be within the top $y * 100\%$, where y is the decision threshold. If not, u is considered as an intruder rather than r.

1) Attack Types: Three types of intrusions are generated in this paper. Type-I attack is defined as the situation where a user of a specific group submits an SC, which the group members are prohibited to use. Type-II attack is an attack that launches a sensitive SC, which is defined as one that may erase or modify sensitive data or system settings, to change the environmental settings of the system or attack the system. A Type-III attack consists of SC-level attack patterns, i.e., SC-patterns, each of which is treated as an attack stage. In fact, an attacker mixing-specific SC can sometimes successfully penetrate a security system.

Generally, SC-patterns that generate a buffer–overflow attack often contain a normal SC with a parameter longer than the buffer's defined length. With this kind of attack, an attacker can penetrate a system by using a Type-III attack and then grant a higher access right to attack the system, e.g., cracking password or acquiring root privilege. The SCs launching a Type-III attack are called a multistage attack pattern. Basically, a single-SC attack pattern is a special category of multistage attack patterns when both the number of stages and the number of SCs are equal to 1.

Type-I and Type-II attacks, defined in the group's class-limited-SC list, can be detected by comparing an input SC directly with the SCs collected in this list. Type-III attack can be identified by invoking Algorithm 2.

As previously stated, all attackers' SC-patterns are also presented in the format of a profile. Given an NCS_u, we can determine whether the NCS_u includes attacker-specific attack patterns or not by employing the process similar to that of judging whether u is the holder of the account that u logs in. After calculating the similarity scores between the corresponding NHF_u and all users' user profiles, if the decisive rate of the attacker profile is higher than $y * 100\%$, we then suspect that u is an attacker and the IIDPS will produce a syslog alert message and reply an "unsafe" message accompanied with the user's ID to inform the SC monitor and filter, which will isolate the user from the system in real time to prevent him/her from continuously attacking the protected system. Of course, if the attacker is not a user of the intranet, a trace-back system [25] or other identification systems [26], [27] are required.

E. Computational Grid

The computational grid consists of a collection of internally connected stand-alone computers working together as a single integrated computing resource. In this paper, a mining server and a detection server are implemented on the grid to speed up data processing. Cluster computing divides a large computational task into smaller subtasks, which are simultaneously performed by employing multiple computer processors. Each processor is responsible for executing some part of the computation, e.g., processing a subset of input data. Typically, the master node coordinates the distributed processing and combines processing results of all processors to complete the entire computation. Fig. 7 shows the symmetric multiprocessing cluster (SMP) with open message passing interface (MPI) architecture. The MPI function provides a standardized and portable environment for processors to mutually exchange messages. The functions of the mining server are accomplished by the grid processors with a hostfile that defines computation resources for the mining server. The detection server is implemented on the master node of the computational grid with the hostfile defining computation resources for the detection server.

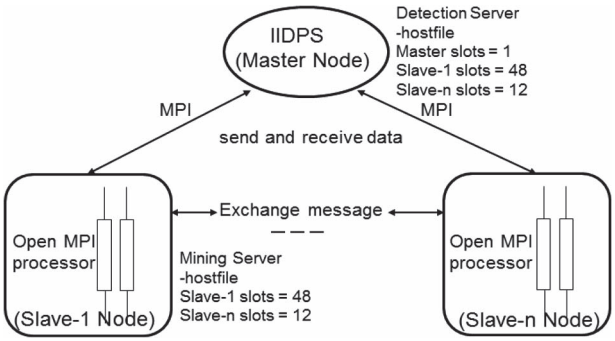

Fig. 7. SMP with open MPI architecture.

F. Parallel Speedup

The speedup S of a parallel system, defined as

$$S = \frac{\text{The time consumed by a single processor to finish a job}}{\text{The time spent by a parallel system to finish a job}} \tag{11}$$

is one of the parameters used to measure the performance of the parallel system with n processors. Generally, speedup of parallel is defined as

$$\mathrm{S} = \frac{1}{f + \max(f'_1, f'_2, \ldots, f'_n)} \tag{12}$$

in which the total time required by a program to accomplish a job is 1, f is the sequential part consumed by a single processor, f'_i is the time spent by processor i to accomplish its assigned task, $\max(f'_1, f'_2, \ldots, f'_n)$ is the response time of the n processors, and $\sum_{j=1}^{n} f'_j = 1 - f$ where $(1 - f)$ is the proportion that can be parallelized (i.e., benefit from parallelization) by employing n parallel processors. In addition, according to Amdahl's law [28], S is limited to

$$S \leq \frac{1}{f + \frac{1}{n}(1 - f)} \tag{13}$$

where $(1 - f)/n$ is the response time if the n processors employed to accomplish the job are identical. In the limit, as n tends to infinity, the maximum speedup approaches to $1/f$. In

TABLE III
RESPONSE TIME OF ALGORITHM 1 BY USING PARALLEL COMPUTING (UNIT: SECONDS)

No. of processing cores	SC-sequence length			
	64	128	256	512
1	16.34	123.53	635.02	2910.70
36	0.45	2.11	8.94	40.45
48	0.43	1.79	6.92	28.19
60	0.61	1.88	5.98	23.14

practice, if f is a small component, performance-to-price ratio falls rapidly as n is increased.

Based on Minsky's conjecture theorem [29], over thousands of processes, the detection server speedup ratios are proportional to $\log_2 n$ where n is the number of processors, meaning that the detection server that has too many sequential parts cannot be executed in parallel. In addition, a huge amount of messages are delivered among parallel processors. When the performance of the employed local computation grid is known, based on the length of the given SC-sequence, we can then determine and employ an appropriate number of grid nodes to serve as the mining server and the online detection server.

In the IIDPS, as shown in Fig. 1, the master node P_m of the local computation grid computes the total numbers of $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparison pairs, calculates computation data size for each slave node P_i, and identifies the log file segment, denoted by $LS(i)$, for P_i, $1 \leq i \leq NP$, where NP is the number of available slave nodes. In each $\langle k\text{-gram}, k'\text{-gram}\rangle$ comparison, the lengths of k-gram and k'-gram are the same, and both k-gram and k'-gram are retrieved from NCS_u, $k = 2, 3, \ldots, |\text{Sliding window}|$ where $|\text{Sliding window}| = |L - \text{window}| = |C - \text{window}|$. Next, the P_m sends $LS(i)$ to P_i. On receiving NCS_u, P_i compares the k-gram and k'-gram pairs identified in $LS(i)$ and returns the identified SC-patterns, and their corresponding appearance counts to P_m. Once P_m receives the results from slave nodes, it sums up the SC-pattern's appearance counts to establish the user's profile. After collecting all users' profiles, P_m calculates each SC-pattern's similarity weight by invoking (7) for each user's user profile and distributes all the user profiles to each parallel computer of local computational grid. With parallel processing, the time required by P_m to wait for the completion of identifying of SC-patterns and their appearance counts will be greatly reduced. Table III lists the response time of Algorithm 1 by using parallel computing.

IV. EXPERIMENTS

To verify the feasibility and accuracy of the IIDPS, three experiments were performed. The first defined the decisive rate threshold between the user profile established for u and each of other users' user profiles. The second studied the accuracy for the online detection server when NCS_u was submitted by u. The third compared the IIDPS with several state-of-the-art host-based IDSs (HIDSs).

Fig. 8 illustrates the configuration of the experimental testbed, which consists of 12 users; one protected computer,

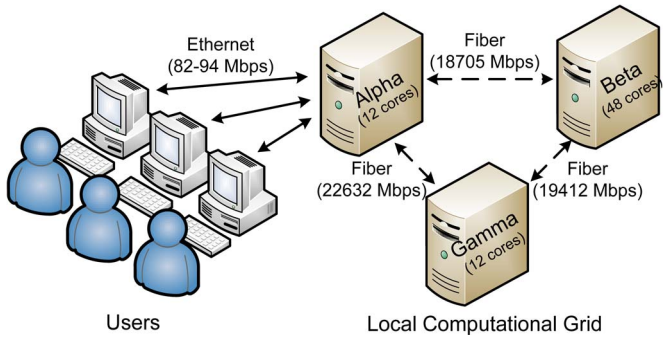

Fig. 8. Logical configuration of the testbed employed in this paper.

named Alpha; and another two members of the computational grid, named Beta with 48 processing cores and Gamma with 12 processing cores. All the computers operate with Linux operating system, and the memory size for each computer is at least 25 GB. The measured bandwidths and network types between two arbitrary nodes of the computational grid are also illustrated in Fig. 8.

In the IIDPS testbed, the SC monitor and filter is first installed into the Alpha computer to obtain each user's log file during the time period between February 1, 2014, and July 31, 2014. The SCs from 12 different categories of users are collected as the experimental data, including the system (root), oracle, message queue (mq), reservation, ticketing, financial, operating strategy (os), backup, configuration management (cm), web application, business rule, and audit users. A total of 193 613 SCs have been collected from 12 log files, in which the length of a sliding window is 10.

A. Decisive Rate Threshold

To determine the decisive rate threshold, 75% of log records are selected as the training data from each of the 12 log files to generate 12 corresponding user profiles, and the remaining 25% are the test data. Ten times of this experiment were performed. Given a known user u's current input SC-sequence, i.e., NCS_u, if the similarity score between NCS_u and u's user profile is ranked within the top $y * 100\%$ among the similarity score between NCS_u and each of the 12 users' user profiles, as aforementioned, the decisive rate threshold is y. The experimental results of decisive rate are shown in Table IV, in which average decisive rate is 0.9312. Therefore, the threshold y is set to 0.9 to evaluate whether the underlying user u of an account is the account holder or not. Besides, the paragraph size is defined as $3 * |\text{Sliding window}|$, i.e., 30 SCs. This means that the detection server calculates the decisive rate for u when the length of u's current input SCs achieves k times the length of a paragraph, where k is a positive integer. The purpose is to avoid continuously computing ranking on each input SC.

Fig. 9 shows the decisive rate between u's current inputs and both the attacker profile and u's user profile. The *logid* is the log sequence ID, uid is the user ID (i.e., u with ID $= 1000$), $ncslength$ is the length of the input SC-sequence (which is $30 * k$, $k = 1, 2, 3, \ldots$), $attacker$ field records the decisive rate for the similarity score between u's current input and attack signatures (attack pattern) in the attacker profile, and $user$ field

TABLE IV
TWELVE USERS' DECISIVE RATES OBTAINED BY CROSS COMPARING USERS' LOG FILES (25% OF TEST DATA)

User log file	Decisive rate	Standard Deviation
oracle.log	0.9333	6.24
cm.log	0.9249	5.83
mq.log	0.9166	5.27
business.log	0.9333	5.0
web.log	0.9249	5.83
reservation.log	0.9416	5.34
os.log	0.9166	5.27
financial.log	0.9083	4.49
ticketing.log	0.9499	5.53
root.log	0.9499	4.09
backup.log	0.9166	3.37
audit.log	0.9583	5.59
Average decisive rate = 0.9312 Average Standard Deviation=5.15		

```
logid=0, uid=1000, ncslength=30, attacker= 0.3211, user= 0.9788
logid=0, uid=1000, ncslength=60, attacker= 0.4003, user= 0.9634
logid=0, uid=1000, ncslength=90, attacker= 0.2387, user= 0.9875
logid=0, uid=1000, ncslength=120, attacker= 0.6541, user= 0.8924 -> alert
logid=0, uid=1000, ncslength=150, attacker= 0.9153, user= 0.8073 -> alert
```

Fig. 9. Detection server calculates decisive rates when the number of u's current input SCs reaches k times the paragraph size, which is 30 SCs, $k = 1, 2, 3, \ldots$.

represents the decisive rate for the similarity score between u's current input and the SC-patterns collected in u's user profile. If the user field is lower than the predefined threshold 90% $(= 0.9)$ or the attacker field is higher than the threshold 90%, the user will be suspected as an attacker.

As shown, when the SC-sequence length is 120, $user = 0.8924$, which is smaller than 0.9. Therefore, the detection server alerts the system manager that the current user may not be this account's holder. When the $ncslength = 150$, $attacker = 0.9153$, which is larger than 0.9, and $user = 0.8073$, which is lower than 0.9. Therefore, the detection server alerts the system manager that the current user is suspected as an insider.

B. Detection Accuracy

In the second experiment, we again randomly chose 75% of users' computer usage history as the training data for creating 12 user profiles, and the remaining 25% are the test data to simulate user u's online inputs. The purpose is to gain the similarity scores between u and all the users so that the IIDPS can judge who the user u is in the intranet.

The statistical information for 12 user profiles generated by the mining server is listed in Table V, in which the "Account ID" shows the user's ID, |Training data| is the number of SCs in the training data, |Habit file| is the number of SC-patterns (rather than SCs) collected in a habit file, and |User profile| is the number of SC-patterns gathered in the user's user profile. About 40% of users' common patterns were removed since their similarity weights are less than the predefined threshold 0.001.

The statistics of user identification accuracy are listed in Table VI, in which "No. of Paragraph" is the number of times

TABLE V
STATISTICS OF 12 USER PROFILES GENERATED BY THE MINING SERVER IN PARALLEL

Account ID	|Training data|	|Habit file|	|User profile|
root	9,531	122,805	73,683
oracle	10,544	126,710	77,293
mq	8,644	109,825	63,698
reservation	16,357	142,155	85,293
ticketing	18,288	145,380	79,959
financial	19,868	153,215	90,396
os	10,202	118,215	72,111
backup	5,678	29,224	16,365
cm	8,643	103,705	64,297
web	11,964	110,203	69,659
business	15,738	139,659	82,398
audit	9,753	105,490	66,458

for calculating decisive rate while evaluating the user's test data where the paragraph size is 30 SCs, "Times of being an account holder" is the average times of its tenfold value that decisive rate is larger than the predefined threshold, and "Times of being an attacker" is the average times of its tenfold value that the decisive rates are smaller than predefined threshold, i.e., the times that the detection server alerts the system manager that the current user is an attacker. The "Detection accuracy" is the accuracy of verifying the user's identity. The database management system utilized to store and process the data is SQLite. There is a total of 100-GB DRAM and 10-TB hard disks' space in the IIDPS, and less than 1 GB is used to store users' profiles.

C. Comparison With Other HIDSs

The IIDPS is a typical HIDS that monitors internal events of a system. A HIDS often gathers and analyzes information issued by users within a system to identify possible threats. To investigate the system's intrusion detection capability, in the third experiment, the IIDPS is compared with four HIDSs, including OSSEC [30], AIDE [31], SAMHAIN [32], and Symantec CSP [33]. The OSSEC analyzes log data, checks file integrity, monitors set policies, detects rootkits, alerts suspected attacks in real time, and responds actively. It has a collaboration learning agent that analyzes log files to identify simple Type-III attacks. AIDE (Tripwire) checks file and directors' integrity for a predefined time interval given by the system administrator. SAMHAIN provides file integrity checking and log file monitoring and analysis. It also detects rootkits, monitors ports, identifies rogue root privilege executables, and figures out hidden processes that issue Type-I and Type-II attacks. Symantec CSP as a superset of the Symantec host IDS can detect part of the Type-III attacks and DDoS attacks launched by a system. It identifies DDoS attacks issued by a system by monitoring the system's outbound traffic. However, this may also trigger false-positive alarms, particularly when users or normal programs upload data to the Internet.

In this paper, an insider attacker may log in to a system by using another user's login ID and password and do something maliciously. As previously stated, a Type-I attack is an attack in which a certain group is prohibited to use. A Type-II attack utilizes rootkits to issue sensitive SCs, i.e., unlinkat() and kill(),

TABLE VI
USER IDENTIFICATION ACCURACY OF THE IIDPS

Account ID	No. of Paragraph	Times of being an account holder	Times of being an attacker (Alarm)	Detection accuracy
root	106	100.03	5.97	94.36%
oracle	117	110.2	6.8	94.18%
mq	96	90.43	5.57	94.19%
reservation	182	171.89	10.11	94.44%
ticketing	203	191.58	11.42	94.37%
financial	220	208.78	11.22	94.90%
os	113	105.67	7.33	93.51%
backup	63	56.68	6.32	89.97%
cm	96	91.02	4.98	94.81%
web	132	124.81	7.91	94.55%
business	174	163.77	10.23	94.12%
audit	108	105.89	2.11	98.04%
Average	-	-	-	**94.29%**

TABLE VII
COMPARISON OF THE IIDPS WITH OTHER HIDSS

Security System	Attack type / response time (second)				
	Identify User	Type-I	Type-II	Type-III	DDoS
OSSEC	× / -	✓ / 60	✓ / 60	× / -	× / -
AIDE	× / -	✓ / 60	✓ / 60	× / -	× / -
SAMHAIN	× / -	✓ / 60	✓ / 60	× / -	× / -
Symantec CSP	× / -	✓ / 2	✓ / 2	Δ / 3	Δ / 3.5
IIDPS	✓ / 0.45	✓ / 0.001	✓ / 0.001	✓ / 0.45	✓/0.45

to modify sensitive data or resource of a system. A Type-III attack employs GDB, i.e., a program debugging tool, to trace a running process and launches a DDoS attack to intrude an outside-world system.

The comparison results of these schemes are shown in Table VII, in which "✓" means that the system has the designated function, "✗" represents that the system does not provide this function, and "Δ" shows that the system has the function, but it does not completely meet that of the IIDPS. All these systems, except the IIDPS, do not have the function of identifying a possible user. The response times of a system for all attack types are also shown in Table VII, in which IIDPS outperforms the other tested systems.

V. DISCUSSION

In this paper, an IIDPS is developed to detect insider attacks at SC level by using data mining and forensic techniques. The experimental results show that the IIDPS can effectively resist several aforementioned attacks. The outcome extends the features of [16], confirming that data mining and forensic techniques used for intrusion detection provide effective attack resistance.

The second experiment indicates that the average detection accuracy is 94.29%. However, in Table VI, the accuracy of user backup is 89.97% since backup's log file has more common SCs than the other users'. It also shows that the IIDPS may detect inaccurately when user's habit suddenly changes. Nevertheless, in most cases, the IIDPS can still identify the legality of a login user.

When a user inputs a command, hundreds or thousands of SCs will be generated. Analyzing a huge number of SCs often takes a long time. As shown in Table VII, the IIDPS spends 0.45 s to identify a user. Although other systems consume longer time for data analysis than the IIDPS does, how to mine SCs in an efficient method should be addressed. Employing a local computational grid can accelerate the processing speed of the miming server and detection server. Generally, users' forensic features retrieved from their basic operations are helpful in detecting the users' malicious behaviors and tell us who the possible attackers are. This can also detect malicious behaviors for systems employing GUI interfaces. However, many third-party shell commands [34], [35] have been developed, including those used in Oracle Database, Oracle WebLogic, IBM WebSphere MQ, and some user-developed applications. We need to study the SCs generated and the SC-patterns produced by these commands so that the IIDPS can detect those malicious behaviors issued by them and then prevent the protected system from being attacked. Additionally, mining user profiles by using an unsupervised cluster approach can also improve the performance of the mining process because processing big data is indeed an engineering challenge. Moreover, to detect an attack and reduce the corresponding response time, we need a cluster workload monitor, a faster filter, an efficient detection algorithm, and a fault-tolerant environment provided by a computational grid. Furthermore, a mathematical analysis on the IIDPS's behaviors is helpful in deriving its formal performance and cost models, with which users can predict performance and cost of the IIDPS before using it. The model proposed in [36] can be further used to increase detection accuracy and improve the decisive rate.

Furthermore, one may ask how a behavior record is created for a new user and how the IIDPS updates a user profile. The answer of the first question is that, when there is a new user k, the IIDPS creates k's log file, habit file, and user profile on his/her first login and then follows the procedure shown in Fig. 3 to generate k's user profile. The answer of the second question is that, each time when k logs in to the system, in addition to the SC k that directly submits, the IIDPS also identifies those SC-sequences generated by submitted commands. These SC-sequences are then used to detect whether k is an attacker or not by invoking algorithm 2. When k is recognized as a legal user,

the IIDPS continues collecting k's SCs and updating his/her profile according to the procedure shown in Fig. 3 on his/her logout. Once k's user profile is updated, it can be used for next-time detection.

VI. Conclusion

In this paper, we have proposed an approach that employs data mining and forensic techniques to identify the representative SC-patterns for a user. The time that a habitual SC-pattern appears in the user's log file is counted, the most commonly used SC-patterns are filtered out, and then a user's profile is established. By identifying a user's SC-patterns as his/her computer usage habits from the user's current input SCs, the IIDPS resists suspected attackers. The experimental results demonstrate that the average detection accuracy is higher than 94% when the decisive rate threshold is 0.9, indicating that the IIDPS can assist system administrators to point out an insider or an attacker in a closed environment. The further study will be done by improving IIDPS's performance and investigating third-party shell commands.

References

[1] S. Gajek, A. Sadeghi, C. Stuble, and M. Winandy, "Compartmented security for browsers—Or how to thwart a phisher with trusted computing," in *Proc. IEEE Int. Conf. Avail., Rel. Security*, Vienna, Austria, Apr. 2007, pp. 120–127.

[2] C. Yue and H. Wang, "BogusBiter: A transparent protection against phishing attacks," *ACM Trans. Int. Technol.*, vol. 10, no. 2, pp. 1–31, May 2010.

[3] Q. Chen, S. Abdelwahed, and A. Erradi, "A model-based approach to self-protection in computing system," in *Proc. ACM Cloud Autonomic Comput. Conf.*, Miami, FL, USA, 2013, pp. 1–10.

[4] F. Y. Leu, M. C. Li, J. C. Lin, and C. T. Yang, "Detection workload in a dynamic grid-based intrusion detection environment," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 427–442, Apr. 2008.

[5] H. Lu, B. Zhao, X. Wang, and J. Su, "DiffSig: Resource differentiation based malware behavioral concise signature generation," *Inf. Commun. Technol.*, vol. 7804, pp. 271–284, 2013.

[6] Z. Shan, X. Wang, T. Chiueh, and X. Meng, "Safe side effects commitment for OS-level virtualization," in *Proc. ACM Int. Conf. Autonomic Comput.*, Karlsruhe, Germany, 2011, pp. 111–120.

[7] M. K. Rogers and K. Seigfried, "The future of computer forensics: A needs analysis survey," *Comput. Security*, vol. 23, no. 1, pp.12–16, Feb. 2004.

[8] J. Choi, C. Choi, B. Ko, D. Choi, and P. Kim, "Detecting web based DDoS attack using MapReduce operations in cloud computing environment," *J. Internet Serv. Inf. Security*, vol. 3, no. 3/4, pp. 28–37, Nov. 2013.

[9] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "MIS: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, 2010, pp. 1–5.

[10] Z. A. Baig, "Pattern recognition for detecting distributed node exhaustion attacks in wireless sensor networks," *Comput. Commun.*, vol. 34, no. 3, pp. 468–484, Mar. 2011.

[11] H. S. Kang and S. R. Kim, "A new logging-based IP traceback approach using data mining techniques," *J. Internet Serv. Inf. Security*, vol. 3, no. 3/4, pp. 72–80, Nov. 2013.

[12] K. A. Garcia, R. Monroy, L. A. Trejo, and C. Mex-Perera, "Analyzing log files for postmortem intrusion detection," *IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev.*, vol. 42, no. 6, pp. 1690–1704, Nov. 2012.

[13] M. A. Qadeer, M. Zahid, A. Iqbal, and M. R. Siddiqui, "Network traffic analysis and intrusion detection using packet sniffer," in *Proc. Int. Conf. Commun. Softw. Netw.*, Singapore, 2010, pp. 313–317.

[14] S. O'Shaughnessy and G. Gray, "Development and evaluation of a data set generator tool for generating synthetic log files containing computer attack signatures," *Int. J. Ambient Comput. Intell.*, vol. 3, no. 2, pp. 64–76, Apr. 2011.

[15] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Appl. Soft Comput.*, vol. 10, no. 1, pp. 1–35, Jan. 2010.

[16] F. Y. Leu, K. W. Hu, and F. C. Jiang "Intrusion detection and identification system using data mining and forensic techniques," *Adv. Inf. Comput. Security*, vol. 4752, pp. 137–152, 2007.

[17] Z. B. Hu, J. Su, and V. P. Shirochin "An intelligent lightweight intrusion detection system with forensics technique," in *Proc. IEEE Workshop Intell. Data Acquisition Adv. Comput. Syst.: Technol. Appl.*, Dortmund, Germany, 2007, pp. 647–651.

[18] J. T. Giffin, S. Jha, and B. P. Miller, "Automated discovery of mimicry attacks," *Recent Adv. Intrusion Detection*, vol. 4219, pp. 41–60, Sep. 2006.

[19] U. Fiore, F. Palmieri, A. Castiglione, and A. D. Santis, "Network anomaly detection with the restricted Boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, Dec. 2013.

[20] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez, "Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study," *IEEE Syst. J.*, vol. 9, no. 1, pp. 1–14, Jan. 2014.

[21] R. J. Roger and M. W. Geatz, *Data Mining: A Tutorial-Based Primer*. Reading, MA, USA: Addison-Wesley, 2002.

[22] S. J. Shyu and C. Y. Tsai, "Finding the longest common subsequence for multiple biological sequences by ant colony optimization," *Comput. & Oper. Res.*, vol. 36, no. 1, pp. 73–91, Jan. 2009.

[23] D. Zhu and J. Xiao, "R-tfidf, a Variety of tf-idf Term Weighting Strategy in Document Categorization," in *Proc. Int. Conf. Semantics, Knowledge Grids*, Beijing, China, Oct. 2011, pp. 83–90.

[24] S. E. Robertson, S. Walker, M. M. Beaulieu, M. Gatford, and A. Payne, "Okapi at TREC-4," in *Proc. 4th text Retrieval Conf.*, 1996, pp. 73–96.

[25] S. Yu, K. Sood, and Y. Xiang, "An effective and feasible traceback scheme in mobile internet environment," *IEEE Commun. Lett.*, vol. 18, no. 11, pp. 1911–1914, Nov. 2014.

[26] B. Sayed, I. Traore, I. Woungang, and M. S. Obaidat, "Biometric authentication using mouse gesture dynamics," *IEEE Syst. J.*, vol. 7, no. 2, pp. 262–274, Jun. 2013.

[27] S. C. Arseni, E. C. Popovici, L. A. Stancu, O. G. Guta, and S. V. Halunga, "Securing an alerting subsystem for a keystroke-based user identification system," in *Proc. Int. Conf. Commun.*, Bucharest, Romania, 2014, pp. 1–4.

[28] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Spring Joint Comput. Conf.*, New Brunswick, NJ, USA, 1967, pp. 1–4.

[29] M. Minsky, "Form and content in computer science," *J. ACM*, vol. 17, no. 2, pp. 197–215, Apr. 1970.

[30] OSSEC. [Online]. Available: http://www.ossec.net/

[31] AIDE. [Online]. Available: http://aide.sourceforge.net/

[32] SAMHAIN. [Online]. Available: http://www.la-samhna.de/samhain/

[33] Symantec CSP. [Online]. Available: http://www.symantec.com/critical-system-protection

[34] Symantec CSP, "Executing operating system commands from PL/SQL," Oracle, Redwood City, CA, USA, White Paper, Jul. 2008.

[35] J. Böhm-Mäder, "The WebSphere MQ for UNIX administration tool," Free Softw. Found., Boston, MA, USA, May 2013.

[36] P. Angin and B. Bhargava, "An agent-based optimization framework for mobile-cloud computing," *J. Wireless Mobile Netw., Ubiquitous Comput., Dependable Appl.*, vol. 4, no. 2, pp. 1–17, Jun. 2013.

Fang-Yie Leu received the B.S., M.S., and Ph.D. degrees from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 1983, 1986, and 1991, respectively, and another M.S. degree from Knowledge Systems Institute, Skokie, IL, USA, in 1990.

He is currently a Professor with Tunghai University, Taichung, Taiwan, where he is also the Director of the Database and Network Security Laboratory and the Chairperson of the Department of Information Management. His research interests include wireless communication, network security, grid applications, and Chinese natural language processing.

Prof. Leu is a member of the IEEE Computer Society and one of the editorial board members of at least six international journals. He is currently a Workshop Organizer of the International Workshop on Cloud, Wireless and e-Commerce Security and the International Workshop on Mobile Commerce, Cloud Computing, Network and Communication Security.

Kun-Lin Tsai (M'00) received the B.S. degree in computer and information science from Tunghai University, Taichung, Taiwan, in 1999 and the M.S. degree in computer science and information engineering and the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2001 and 2006, respectively.

He was a Postdoctoral Researcher with the National Taiwan University of Science and Technology, Taipei, in 2007. He is currently an Assistant Professor with the Department of Electrical Engineering, Tunghai University. His research interests are low-power system design, information security system, and very large-scale integration design.

Yi-Ting Hsiao received the B.S. degree in industrial engineering and the Master's degree in computer science from Tunghai University, Taichung, Taiwan, in 1994 and 2014, respectively.

He is currently a Project Manager and a Chief Engineer with MiTAC Information Technology Corporation, Taipei, Taiwan. His areas of interest include software engineering, Linux embedded system, computer forensics, and parallel processing.

Chao-Tung Yang received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1996.

In 2001, he joined the faculty of the Department of Computer Science, Tunghai University, Taichung, Taiwan, where he is currently a Professor of computer science. He has published more than 250 papers in journals, book chapters, and conference proceedings. His current research interests are in cloud computing and service, grid computing, and multicore programming.

Dr. Yang is a member of the IEEE Computer Society and the Association for Computing Machinery. He is serving in a number of journal editorial boards, including the *International Journal of Communication Systems*, the *Journal of Applied Mathematics*, the *Journal of Cloud Computing*, "Grid Computing, Applications and Technology" Special Issue of the *Journal of Supercomputing*, and "Grid and Cloud Computing" Special Issue of the *International Journal of Ad Hoc and Ubiquitous Computing*.