International Journal of Computer Science and Applications
Vol. 6, No. 3, pp 75 – 91, 2009

A FORENSIC LOGGING SYSTEM BASED ON A SECURE OS

ILKYEUN RA

Dept. of Computer Science and Engineering, University of Colorado Denver
1200 Larimer St. Campus Box 109. Denver, CO 80204, U.S.A.
Ilkyeun.ra@ucdenver.edu
http://carbon.cudenver.edu/~ikra

TAE-KYOU PARK

Dept. of Computer and Information Science, Hanseo University
Seosan, 356-706, South Korea
tkpark@hanseo.ac.kr

This paper describes a forensic logging system that collects fine-grained trace evidence from target servers and networks. To provide a more reliable and efficient forensic logging system, we developed a dedicated network processor that collects network traffic and a hardened TCSEC-B1 level secure operating system. The system also is capable of defending servers from malicious attacks as well as allowing system security managers to obtain forensic evidence from the forensic logging system when the target system is assaulted by malicious attacks from the internet. We present the structure of the system and discuss the benchmark test results of our prototype system.

Keywords: Forensics, logging, secure operating system

1. Introduction

The Internet has created one world with no barriers and has significantly improved our daily life while making a number of tools available to computer criminals. With the Internet, malicious attacks have resulted in a new security science, which is called computer forensics. Computer forensics deals with collecting digital evidence from digital devices, such as personal computers, servers and network-attached devices. Forensics is concerned with the capture, analysis and reconstruction of system activities in order to determine, post-facto, how and whether a machine was compromised. One major goal of network forensics is creating accurate evidence that can be used in a court of law. Forensic analysis also enables system administrators to troubleshoot problems, track down suspicious behaviors, and evaluate the extent of damage to assaulted machines.

Although many organizations and home users adopt general security products such as fire walls, intrusion detection and protection systems, or secure OS for protecting their computer network systems from intrusion, their systems can sometimes be compromised by malicious attackers. Thus, it is essential to gather evidence of activities by adopting

forensic tools to determine post-facto how or whether the system is damaged. Finding the root causes and effects of a network intrusion or attack might require deploying a logging system that collects network and host-based trace evidence to help system administrators who already may be overwhelmed by their routine daily duties. However, current logging systems require an investigator to manually scrutinize log information for clues based on the current state of the system. As such, one of the most costly, time-consuming and human-intensive tasks is the analysis and reconstruction of the compromised system.

Several approaches currently exist [Computer Associates (2006); Jeffris (2002), Goel (2005); Honeynet (2004)] for logging and auditing at the kernel and/or packet level, but these approaches don't provide enough intrusion trace evidence at the operating system kernel and packet level, nor do they support protection mechanisms against attacks.

To reduce the manual labor involved in network intrusion analysis, we developed a forensic logging system based on a security-hardened OS and a network processor (NP). This system supports collecting intrusion evidence; it also supports analyzing and responding to various requests from security administrators.

Our system performs comprehensive monitoring of target servers at the kernel level using a secure OS, and at the packet level using a network processor. It also allows system administrators to: a) view fine-grained and application-independent trace evidence related to all server activities, and b) store kernel and packet data using a machine based on a secure OS. Our system uses a database technology to support high-level queries from the archived logs, and it significantly reduces the human cost of performing the forensic analysis.

The rest of this paper is organized as follows. We begin by presenting the motivation, technical goals and an overview of design approaches in Section II. In Section III, we discuss the implementation of our system in detail and in Section IV we evaluate the attack test results and overhead of our prototype system implementation. Finally, in Section V, we summarize our conclusions and present our future research plans for improvements to our system.

2. Design Overview

This section describes the design rationale and gives an overview of the forensic logging system.

2.1. *Motivation and Technical Goals*

Most system administrators or system security managers manually examine intruders' footprints from system log files to find criminal evidence when the system is assaulted by attackers or the system detects suspicious activities. Manual analysis and scrutiny of log

information to search for evidences is very time-consuming and expensive in terms of human labor costs. Furthermore, with manual forensic operations, it may be difficult to avoid losing some important forensic clue information: the source of attacks, the point of attacks, and the method of attacks, by the intruders. Current forensic systems implement process account management, network traffic traces, and file system checkers; system log files to log and monitor entire users' system usages have been used. However, these approaches do not provide enough evidence to allow reconstruction of the event in terms of what exactly happened to the system because they are implemented separately and fail to find and make correlations from separately collected logging information obtained by each approach. Furthermore, these systems are also vulnerable to malicious attacks because they do not have any extra security features to protect themselves.

To overcome the limitations of current systems, our forensic logging system has been designed to support three principal technical goals. First, the system should monitor server activities at the kernel of the target server OS as well as at the network packet level of the target network. Second, the detailed users' activities information gathered from the target server and the network processor should be transmitted immediately to a separate forensic server (log machine) on which a secure OS has been installed to ensure robust access control. This provides the system with more secure centralized management of logs to support real-time monitoring of intrusions. Third, a database should be available to respond to high-level queries of the log files stored on the forensic server.

In addition, our system has been designed to maintain the following four properties to support comprehensive forensic analysis services: 1) completeness: a logging system should be able to prevent possible forgery of collected logging data by any means and to completely reconstruct the attack event from the collected system log information; 2) authenticity: a logging system should be equipped with strong authentication to protect against message spoofing or intrusion attacks and also should not allow any modification of operations that may be linked to fabricate the collected logging information; 3) reproducibility: a logging system should allow security managers to determine accurately what happened to the system in terms of “who” and “what” for each user's activities to network-attached devices, files, and processes, etc. To support complete reproduction of incidents, a logging system should be able to correlate the attack event with the collected logging information; 4) efficiency: a logging system should intelligently collect logging data efficiently so that it cannot slow down the forensic process without sacrificing the completeness property.

2.2. *Design Overview*

To accomplish our technical goals and preserve the four properties that were discussed in the previous section, we logically designed our forensic logging system as shown in Figure 1.

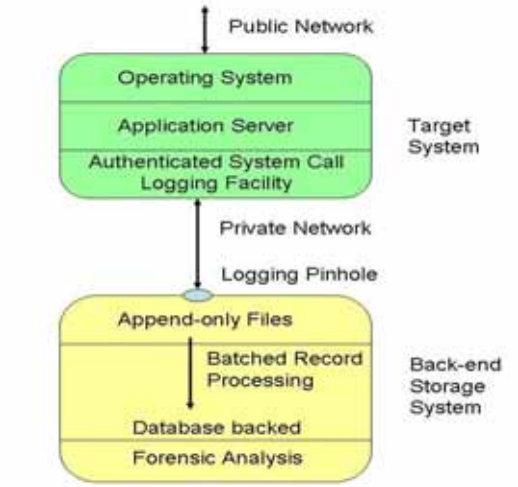


Fig. 1. Logical Organization of the forensic logging system.

In general, when an intruder successfully invades the system, a harmful or malicious operation is executed by a process that is invoked by a system call in the compromised machine. If we collect every system call and its associated activities of users and processes, we can reproduce all attack events or security breaches. Collecting system calls also allows us to achieve completeness of logging information as well as efficiency by recording only necessary information. Our system call logging module is added in target machines and collects entire system call activities and transfers them to the back-end storage system via the internal private network. It also is designed to operate with strict restrictions to avoid the race condition. To prohibit the possible fabrication or modification of logging information, our logging system records all system calls between the internal private network interface and the target system, and requires every access of logging data to authenticate with the target system.

The back-end storage system enforces all access requests including saving logging information to authenticate with it, and monitors and manages all data accesses from its kernel level. One of the drawbacks of implementing forensic systems originates in the tremendous size and complexity of the logging data. Our back-end storage system creates a relational database system by correlating collected trace data (logging information), either periodically or by request from the security manager. It allows the system to provide fast and accurate query results. The back-end storage system should quickly respond to forensic queries, but it is allowed to block all access requests from target machines to protect forensic logging data if it cannot process forensic queries within the recommended time limit.

Figure 2 presents the network environment for the forensic logging system. The logging system gathers information about the detailed activities of users, and stores the information securely in log files on a forensic server. This design provides integrated analysis and centralized management of system logging; also, it facilitates forensic

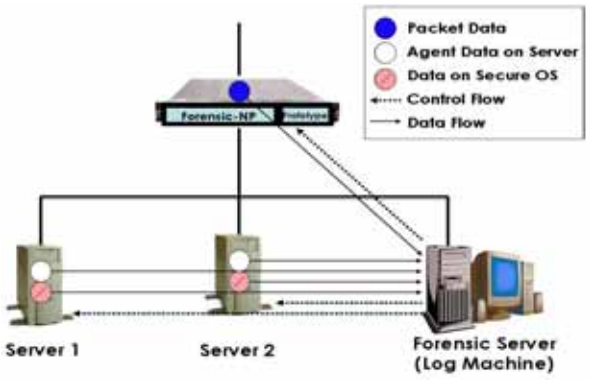


Fig. 2 Basic network Environment

investigations. The system should gather volatile and non-volatile information from the target servers and the networks, and store the information in terms of "who" (users and processes), "when" (in chronological order), and "what" (users' activities). Information capture should conform to the Order of Volatility (OOV) Principle [Brezinski (2002)].

By using an add-on secure OS [TSonNet (2005)] with the logging system, system administrators should be able to protect their servers from attacks, scrutinize suspicious activities, and determine the scope of system compromise. To meet these needs, our design incorporates several forensic logging agents, including a log-extended secure OS module, several server forensic agent modules, and a network agent that is connected through a secured TCP/IP network. The evidence collected by these forensic logging agents is immediately forwarded to the forensic server.

2.3. *System Architecture Overview*

Figure 3 presents the overall system architecture, which includes the server and the network forensic system. The forensic agents consist of four modules including a secure OS, an OS syslog, a volatile-snapper, and an NP. The secure OS module is added on a standard Linux kernel [Beck (1996)] as a Linux Security Module (LSM) [NSA (2008)] and meets TCSEC-B1 [DoD (1985)] requirements through mandatory access controls between all subjects (processes) and all objects (file, directory and device). The OS syslog module collects a substantial amount of information from various kinds of system log files generated by the OS. The volatile-snapper module collects volatile

miscellaneous information according to the OOV principle mentioned above. The NP module is a dedicated hardware device, attached in front of the target servers through an in-line private network (or sub-net), and captures all packet streams transmitted to and from the servers. The logs gathered by the four forensic agent modules are transmitted to the forensic server by the loaders.

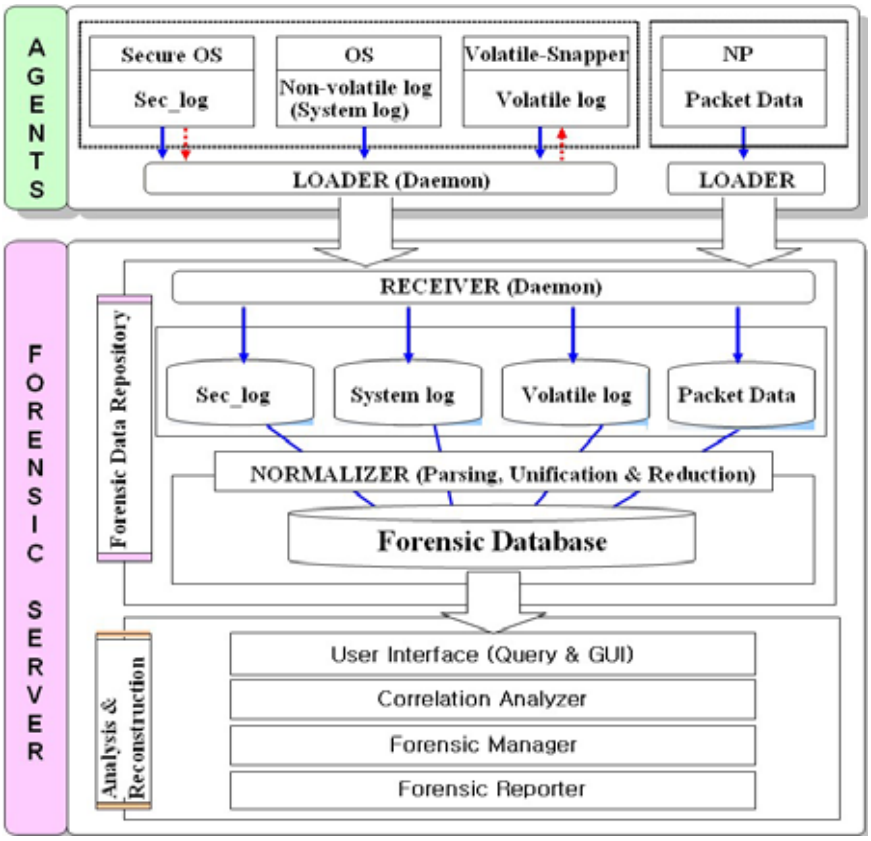


Fig. 3. Forensic Logging System Arachitecture.

The forensic server combines the forensic logs into a forensic data repository (database and files), and analyzes them.

This server is composed of four modules: a receiver, a normalizer, a user an interface component, and a forensic management system (including the forensic analyzer, the manager, and the reporter). The receiver module runs as a daemon on the server, and communicates with the loaders on the target servers and the NP. It also receives the transmitted log information from the loaders. The normalizer module correlates the four received logs (*sec_log, system_log, volatile_log*, and *packet_data*) into a single layer

data format. The forensic management module provides security managers with the resulting information tracing attackers, analyzing logs, reconstructing system files, and defending against attacks via a graphical user interface.

3. Implementation

This section discusses in detail the implementation of the prototype system's major modules: Forensic Agent, Network Processor, Loader and Receiver module, and Forensic Server.

3.1. *Forensic Agents*

The key structures and functions of four forensic agents and the gathering of forensic logs from the servers and the network processor are discussed here.

3.1.1. Secure OS Module

The secure OS meets the TCSEC-B1 rating requirements [DoD (1985)] such as a multi-level security reference monitor using the Bell & Lapadula model [Bell (1973)], labeling all subjects/objects with clearance/category information, and providing advanced security management.

For this project, we have developed our system with a kernel-level secure OS [TSonNet (2005)] because it has distinct advantages:

(a). strengthened access control (i.e., adding kernel-level mandatory access control to the existing discretionary access control).
(b). providing attack protection algorithms (including well-known buffer overflow attacks and unknown root privilege interception attacks at the kernel).
(c). extendable auditing services (adding some attributes for forensics).
(d). supporting a server fire-wall role (i.e., it can block or permit designated user IDs, incoming/outgoing IP addresses, commands and processes from/to the target server).

Using an alarm function, our logging system is designed to capture lots of information when possible. Whenever the secure OS module detects suspicious behaviors, it wakes up the volatile-snapper module (daemon), and starts operations to gather the various volatile logs and post them to its *sec_log* file. The *sec_log* file is read and formatted by the loader (daemon) on the server, and securely transmitted to the receiver (daemon) on the forensic server.

Table 1 shows over fifty formatted fields provided by the loader. The values of these fields are supplied by both the secure OS and the NP. The secure OS module has been implemented with the LSM.

Table 1. Named Fields.

Field Type	Field Name
Where	Serial_No, OS_Type, OS_Version, Local_ID, Agent_MAC_Addr, Agent_IP_Addr, Expand_File-and-Type
When	Event_Data_Time, Event_Millisecond
Who	Source_MAC_Addr, Source_IP_Addr, Source_Port_Addr, Destination_MAC_Addr, Destination_IP_Addr, Desination_Port_Addr, PROTOCOL, I/O, Terminal, Process_ID, Process_Name, Command_Detail, Parent_Process_ID, Parent_Process_Name, Effective_User_ID, Effective_User_Name, Effective_Group_ID, Effective_Group_Name, Real_User_ID, Real_User_Name, Real_Group_ID, Real_Group_Name, Original_User_ID, Original_User_Name, Original_Group_ID, Original_Group_Name, Subject_Atrribute_Clearance, Sibject_Atrribute_Category, Subject_Atrribute_Privilege
What	Inode_Number, Oject_Path-and-Name, Object_Attribute_Clearance, Object_Attribute_Category, Objective_Attribute_Privilege, Event_Msg_Type, Event_Msg_ID, Event_Msg_Detail
Why	Event_Msg_Detect_Rule, Is_Setuid, Is_Setgid, Daemon_Hacking
Etc.	Retry_Count, LINK_1, LINK_2, RESERVED, Indicator, Carriage_Return

3.1.2. OS System Log Module

The syslog is a type of non-volatile log that is accessed during forensic analysis. The system log module of this system collects various information from *syslog* files such as *acct (pacct), aculog, lastlog, loginlog, sulog, messages, utmp, utmp, wtmp, wtmp, vold.log*, and *xferlog*. The OS system log module is always set with a wait state, and accumulates the logging data periodically over specific time intervals, or immediately if it receives an alarm from the secure OS module. The collected logs are hashed, encrypted, and sent to the receiver module of the forensic server via secure socket communications.

3.1.3. Volatile-Snapper Module

According to the OOV principle, the volatile-snapper module has two functions: keystroke logger and volatile information snapper. It stores users' keystrokes on the target server, logs volatile information (using TCT [Jeffris (2002)]), and tracks important environmental files such as *passwd, shadow, groups, hosts.equiv, hosts.allow, hosts.deny, syslog.conf, crontab, xinetd, conf, rc** under */etc* directory, */root/.rhosts*, and */root/.bash_history*.

The Volatile-Snapper module runs as a daemon process, executing *logather.sh* script to gather the volatile information from the kernel periodically over specific time intervals or on demand if the secure OS module indicates that the target server has detected an

attack. This module has been designed to perform data reduction by itself if the size of snapped files becomes very large. The information collected by this module is also preprocessed for integrity and confidentiality, and then sent to the receiver module in the forensic server via secure socket communications. In addition, the *keylogger* monitors all users' keystrokes on the server, and stores them chronologically in the *volatile_log* file, which is similar to the Honeynet architecture [Honeynet (2004)]. This file is very important for a forensic administrator to analyze who logged in when and what commands were entered from the user's keyboard during the session. This *keylogger* is implemented by the LSM and runs by a daemon on the target server.

3.1.4 Network Processor (NP) Module

The NP module gathers all or selective packet data streams (requests or responses). The NP provides extendable services that may be able to preprocess packet data streams at a high speed. For example, to protect a database server from a kind of SQL injection attack, this system may use the NP to scan and filter query contents. Through the user interface of the NP, target server IP addresses can be set for capturing selectively their network packet streams involving particular servers such as databases, web and mail servers. The reason for this is that the captured information can be too large, especially in high-speed networks, to be scrutinized easily. In addition, it can capture the headers and contents of packets separately. After gathering and analyzing the packet data, it formats the data into Table 1 fields, and sends them to the forensic server via the loader daemon. At present, we have modified and implemented *Pcap* a library of Tcpflow[JACOBSON (2008)] to capture packets of TCP/IP applications such as *telnet, rlogin, ftp*, and *http*.

Embedded Linux has been installed as an operating system on a dedicated hardware NP. The current NP consists of Power PC, 512MB, socket support type memory, 64MB Flash, 2 Gigabit Ethernet I/O ports etc. Figure 4 shows the front view of the dedicated hardware NP rack.

Fig. 4. Front View of Network Processor.

3.2. *Loader Module*

The loader module runs on the target servers as a main component of forensic agents and the NP as daemon processes. The loader collects four major logs (*system_log, volatile_log, sec_log,* and *packet_log*) and saves them. The saved logs are hashed into a 128 bit digest by MD5, encrypted by 128bit-SEED cryptographic algorithm [Park (2005)] for integrity and confidentiality, and transmitted directly to the receiver daemon modules via secure socket communications whenever the agents have logs to be transmitted. The detailed components of the loader module and their interactions are shown in Figure 5.

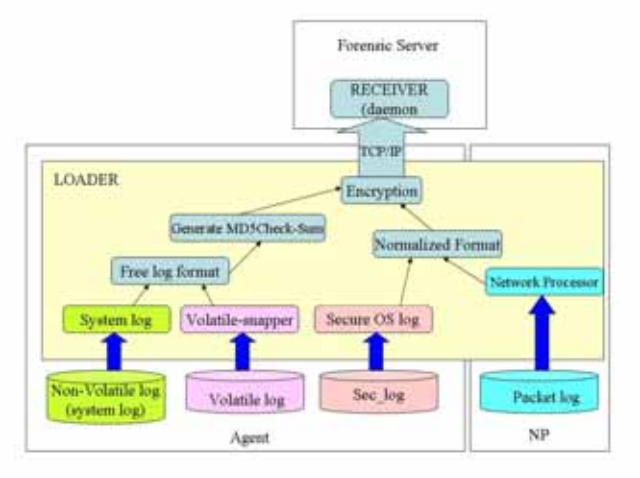


Fig. 5. Loader Module Architecture

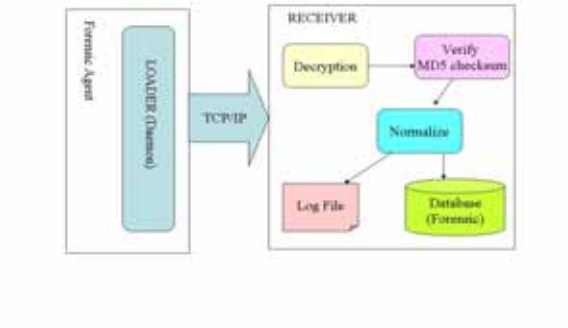


Fig. 6. Receiver Module Architecture.

3.3. *Receiver Module*

The receiver module runs on the forensic server. It receives the collected logs from the target machines by the secure TCP/IP connection, and then it checks and maintains the integrity and confidentiality by comparing MD5 digests and deciphering the logs. If the received data is passed, the receiver module stores the information immediately into the SQLite-relational database on the forensic server in order to correlate and normalize them in the next step. Figure 6 describes the components of the receiver module and the interactions between the components.

3.4. *Forensic Server*

The forensic server consists of a normalizer module and forensic log management module. The main job of the forensic server is to combine the logs obtained from the target servers and NP, and store them in its file system and database. It is very important to protect the forensic server from changes by using a security-hardened OS. All forensic evidence data that the receiver gathers from the target servers must be labeled so only a security manager can access them [TSonNet (2005)].

The system administrators issue commands and queries from their machines. To avoid running out of space on the forensic server, the oldest files are automatically deleted as necessary. Before deleting the dated logs, it is required to back them up to stable storage such as CD-R discs or tape cartridges. As mentioned in the previous subsection, it is necessary to attach security labels to the files and database on the forensic server. This implementation provides the forensic administrator with a user interface to the multi-level secure OS. Thus, the secure OS module protects the forensic server against intrusions and other vulnerabilities.

3.4.1Normalizer Module

In the forensic data repository, there are four different logs: the secure OS *sec_log*, the *system_log*, the *volatile_log*, and the NP *packet_data*. To reconstruct events, it is necessary to correlate these four different logs to a single layer data format ordered by time, IP address and User ID. This task is known as normalization [Forte (2004)]. The filtering is also necessary to extract certain types of log information and rearrange them by time, protocol type, IP address, User ID, MAC address, and etc.

Time synchronization problems across the different platforms can be avoided if all agents in the machines are synchronized within the same time zone. All servers and the NP in this system have been designed to use Network Time Protocol (NTP) embedded in Linux OS conforming to some NTP-related RFCs [Mills (1992)]. This means that the other agents comply with the NP's time for synchronizing by using the NP as a time server. As a result, the information stored on the forensic database by the normalizer module can easily be retrieved via an administrator's query statements.

Some collected information may not be stored in the formatted database because of a different format or non-formatted log files. In this case, this implementation stores only the indices or names of files to the forensic database with links to the original unformatted files. Thus, if there is a need to analyze the forensic information in more detail, the administrator can retrieve the file contents via the links.

3.4.2 Forensic Log Management Module

The forensic log management module provides system administrators with the following services: registering forensic servers, managing forensic agents, examining logs, searching logs, creating statistics, and tracing malicious attacks. It runs on the system administrator's machine, currently PC with MS-Windows, and communicates with the forensic servers via the encrypted TCP/IP connections. It is composed of three subsystems: the user interface subsystem, remote method call subsystem, and forensic management subsystem. Figure 7 shows the architecture of the forensic log management module. The description of each subsystem is as follows:

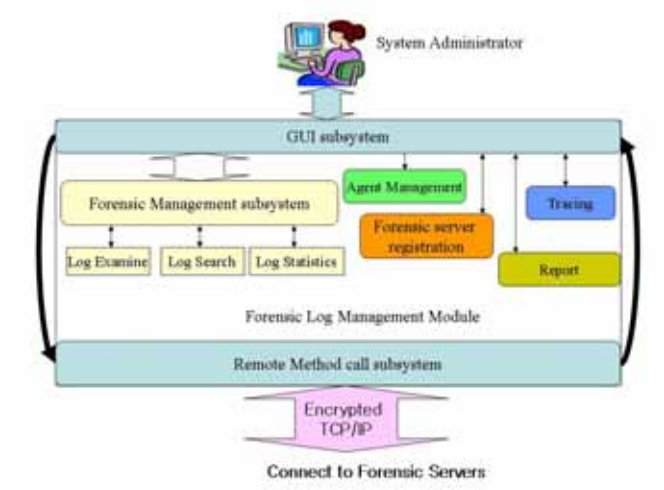


Fig. 7. Forensic Log Management Module Architecture.

(1) User interface(GUI) subsystem

The forensic log management system provides system administrators with a GUI interface so that they can easily and efficiently manage and analyze collected logging data. The GUI subsystem has been developed as a DLL (Dynamic Link Library) to minimize resource consumption of the system administrator's machine and runs on a separate PC over an MS-Windows platform. The implemented user interface DDL has four phases: 1) create session, 2) authorization, 3) invoking remote method call, and 4)

session end. The invoking remote method call subsystem is only allowed if the request passes step (1) and (2) with success.

(2) Remote methods call subsystem

The remote methods call subsystem sends a system administrator's request to the forensic server's receiver daemon via an encrypted TCP/IP connection. When it receives a user's request through the GUI interface, it finds appropriate remote call methods, generates a proper message format, and tries to connect forensic servers. At this time, the remote methods call subsystem creates a session and encrypts the connection links between forensic servers to protect from possible network security breaches such as eavesdropping and middle man attack. It also manages and controls every session between requests and forensic servers.

(3) Forensic management subsystem

The forensic management module, which runs on a Windows PC, is composed of the forensic analyzer, manager, and reporter module. The forensic manager connects to the forensic server via the remote methods call subsystem and sets up the environment for the forensic analyzer. Once the forensic analyzer establishes the secure connection, it retrieves the suspected portion of logs from collected logs at the forensic server, investigates the logs to find evidence of attacks or intrusions, and generates statistical reports upon request from a forensic administrator. It can also invoke the attack tracing service if necessary. The forensic reporter creates a forensic report based on the result of performing forensic analysis, and displays the report in the forensic administrator's window.

4. Evaluation

We installed our prototype forensic logging system on four target servers (RedHat EL4 Linux) that run forensic agents, and installed one NP agent on an embedded Linux Montavista Edition. The forensic manager programs: the receiver, the normalizer, and the user interface module on the forensic server also have been implemented on RedHat EL4 Linux. The forensic management module, however, uses Windows.

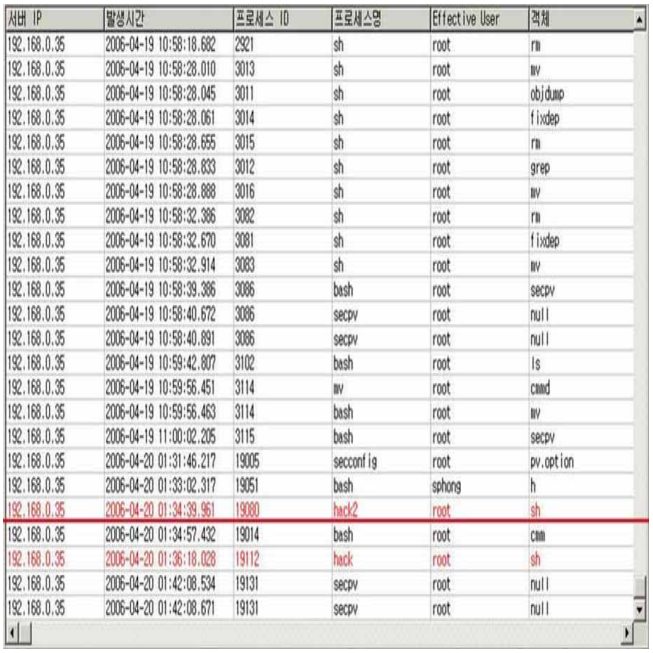
In this section, we present our system test results including attack tests and performance test.

4.1. *Attack Tests*

To see how our prototype forensic system can protect from malicious attacks, we ran the following integrated testing. As soon as we attacked a target server to steal a root privilege, the reference monitor of the secure OS on the target server detected an

unauthorized access. At this time, all activities of the attacking users were captured. The secure OS and volatile-snapper modules (including the keylogger) as well as the NP agent supply the data that is stored on the forensic server by the receiver, loader, and normalizer modules.

Logs are stored in specific files on the forensic server. All these logging files are automatically labeled by the multi-level secure OS kernel. This means that even a user with a root privilege can't access the log files or database on the forensic server.



서버 IP	발생시간	프로세스 ID	프로세스명	Effective User	객체
192.168.0.35	2006-04-19 10:58:18.682	2921	sh	root	rm
192.168.0.35	2006-04-19 10:58:28.010	3013	sh	root	mv
192.168.0.35	2006-04-19 10:58:28.045	3011	sh	root	objdump
192.168.0.35	2006-04-19 10:58:28.061	3014	sh	root	fixdep
192.168.0.35	2006-04-19 10:58:28.655	3015	sh	root	rm
192.168.0.35	2006-04-19 10:58:28.833	3012	sh	root	grep
192.168.0.35	2006-04-19 10:58:28.888	3016	sh	root	mv
192.168.0.35	2006-04-19 10:58:32.386	3082	sh	root	rm
192.168.0.35	2006-04-19 10:58:32.670	3081	sh	root	fixdep
192.168.0.35	2006-04-19 10:58:32.914	3083	sh	root	mv
192.168.0.35	2006-04-19 10:58:39.386	3086	bash	root	secpv
192.168.0.35	2006-04-19 10:58:40.672	3086	secpv	root	null
192.168.0.35	2006-04-19 10:58:40.891	3086	secpv	root	null
192.168.0.35	2006-04-19 10:59:42.807	3102	bash	root	ls
192.168.0.35	2006-04-19 10:59:56.451	3114	mv	root	cawd
192.168.0.35	2006-04-19 10:59:56.463	3114	bash	root	mv
192.168.0.35	2006-04-19 11:00:02.205	3115	bash	root	secpv
192.168.0.35	2006-04-20 01:31:46.217	19005	secconfig	root	pv.option
192.168.0.35	2006-04-20 01:33:02.317	19051	bash	sphong	h
192.168.0.35	2006-04-20 01:34:39.961	19080	hack2	root	sh
192.168.0.35	2006-04-20 01:34:57.432	19014	bash	root	cmm
192.168.0.35	2006-04-20 01:36:18.028	19112	hack	root	sh
192.168.0.35	2006-04-20 01:42:08.534	19131	secpv	root	null
192.168.0.35	2006-04-20 01:42:08.671	19131	secpv	root	null

Fig. 8. Warning Log Messages on the Attempted Server – by selecting attributes [Ser IP, Time of Event, Process ID, Process Name, Effective User and Object] (Korean Language Version).

The attack testing results confirmed that our system is capable of capturing attacks and providing forensic evidence. In addition, the multi-level secure OS shows that it protects the forensic data from unauthorized access attempts.

Figure 8 shows warning messages with two colored lines that have been identified by selecting some attributes from the forensic server database when the secure OS module on the attempted target server (192.168.0.35) protected and collected the attempted illegal activities by the name of ``hack2'' process.

4.2. *Performance Tests*

A practical logging system should have tolerable overhead and meet reasonable storage space requirements. To evaluate our logging system, we performed two experiments to measure the time overhead that the logging system takes to gather the logs from the target server, and the space overhead that is caused by storing all the agent logs on the forensic server. In addition, to measure the average time overhead on file operations, process creations, program executions and communication latencies, we used a micro-benchmark tool – lmbench [McVoy (1996)] - for each operation on the target server.

Table 2 shows that our un-optimized prototype slightly increases process operations, but increases by three times the average for simple file operations (create, delete, stat and open/close). This overhead may be offset by the utility of collecting data and systematically performing logging tasks. As we predicted, almost all of the additional time is spent on the file-related activities. Note, however, the 13% overhead is for *lat_http*.

Table 2. Imbench performance overhead.

System call	Base	Forensics	Overhead
null	0.53 µs	0.53 µs	0 %
stat	4.72 µs	28.20 µs	497 %
open/close	5.92 µs	34.10 µs	476 %
fork	161.00 µs	164.70 µs	2 %
execve	169.30 µs	174.70 µs	3 %
sh	2,956.00 µs	3,553.00 µs	20 %
create	26.10 µs	71.80 µs	175 %
delete	22.80 µs	56.70 µs	148 %
pipe	492.7 Mb/s	493.7 Mb/s	0 %
AF_UNIX	362.0 Mb/s	362.0 Mb/s	0 %
lat_http	645.5 Kb/s	728.2 Kb/s	13 %

This means that the target server is burdened to some extent by transferring the logs. However, we believe that the performance overhead can be reduced significantly by adjusting the frequency and techniques of the log transfer. Table 3 shows the results of the kernel build benchmark that is generally used as a macro-benchmark. The average time overhead in Table 3 is generated by compiling kernel source 2.6.9 to bzImage file. We believe that the average overhead 4% is relatively small. Note that the increase in terms of the space overhead of the forensic server is very dependent on the frequency of possible attacks.

Table 3. Kernel build time.

	Base	Forensics	Overhead
Average	559 s	581 s	4 %

Table 4. Size of logs collected from agents.

Agent	Log Size/Trial
Secure OS Agent	1.5 Kb
Syslog Agent	81.0 Kb
Volatile_Napper Agent	1,500.0 Kb
NP Agent	1.5 Kb
Total	**1,5484 Kb**

As shown in Table 4, the size of files and database grew by a rate of 1.5Mb for one trial, which may be reasonable. We estimate that with periodic log gathering occurring every two minutes for 24 hours each day, 6-months of forensic logs can be stored on a single 200GB disk. Our experiments for the target server agent were performed on a Compaq ML380, with a 864MHz Intel Pentium-III Processor with 1GB of memory and a 6GB HDD running Linux. The forensic server is a Compaq ML380, which has a 1,266MHz Intel Pentium-III Processor with 512MB of memory and a 6GB HDD running Linux.

5. Conclusions

In this paper we presented the structure and development of a forensic logging system. Our system takes advantage of a secure OS module for defending against intrusions, as well as gathering detailed, useful server activity data at the kernel level. It also uses a dedicated network processor for gathering data about the network packets at Giga-bit speed.

By combining both the secure OS and the NP, it is also possible to implement additional security services with the current prototype system. For instance, by filtering server users' malicious code (viruses, worms, etc) or DoS attack streams by the NP, this system may block their accesses to sensitive servers by dropping their packets at the NP.

We plan to improve our prototype system. The list of our future work includes providing more formatted forensic data, optimizing the time and space overhead, and extending the running OS system platforms to Solaris, HP-UX, AIX and Windows in place of the current Linux.

Acknowledgments

This research was supported in part by the Ministry of Information and Communication Republic of Korea, IT R&D Project (A1200-0502-0067).

References

Beck, M.; Bohme, H.; Dziadzka, M; Kunizt, U.; Magnus, R. and Verworner, D. (1996). *Linux Kernel Internals*, Addison-Wesley, Boston, Massachusetts.

Bell, D.; LaPadula, L. (1973). *Secure Computer Systems: A Mathematical Model (Volume II*), MITRE Technical Report **2547**, The MITRE Corporation, Bedford, Massachusetts.

Brezinski, D. and Killalea, T. (2002): *Guidelines for Evidence Collection and Archiving (RFC **3227***), Network Working Group, The Internet Engineering Task Force.

Computer Associates, (2006). *eTrust Network Forensics* (URL: http//www.ca.com/files /FAQs/etrust_network_forensics_faq.pdf).

Forte, D. (2004): The art of log correlation: Tools and techniques for correlating events and log files, Computer Fraud & Security, Vol. **2004**, No. **8**, August, pp. 15--17.

Goel, A.; Feng, W; Maier, D. and Walpole, J. (2005): Forensix: A robust, high-performance reconstruction system, *Proceedings of the Twenty-Fifth IEEE International Workshop on Distributed Computing Systems*}, pp. 155--162.

Jeffris, C. (2002). *The Coroner's Toolkit: Incident Handling and Forensics (GSEC v1.3)*, Information Security Reading Room, SANS Institute, Bethesda, Maryland (URL: http://www.sans.org/reading_room/whitepapers/incident).

McVoy, L. and C. Staelin, C. (1996): **lmbench**: Portable tools for performance analysis, *Proceedings of the USENIX Annual Technical Conference*, January 22-26, San Diego, CA, pp. 279--294.

Mills D. (1992): *Network Time Protocol (Version 3): Specification, Implementation and Analysis (RFC **1305***), Network Working Group, The Internet Engineering Task Force.

National Security Agency (2008): *Security Enhanced Linux*, http://www.nsa.gov/selinux.

Park, J.; Lee, S.; Kim, J. and Lee, J. (2005): The SEED Encryption Algorithm (RFC **4009**), Network Working Group, The Internet Engineering Task Force.

Jacobson, V.; Leres, C.; McCanne, S. (2008): **libpcap**, Lawrence Berkeley Laboratory, (http://www.tcpdump.org).

The Honeynet Project (2004). *Know Your Enemy: Learning about Security Threats*, Addison-Wesley/Pearson Education, Boston, Massachusetts.

TSonNet (2005), *Trusted Systems on the Net*, RedOwl (http://www.tsonnet.co.kr).

Department of Defense (1985). *Trusted Computer System Evaluation Criteria*, DoD Publication **5200.28-STD**, Washington, DC.