

# Design of a Log Server for Distributed and Large-Scale Server Environments

Attila Özgüt (ozgit@metu.edu.tr)

Burak Dayıođlu (dayioglu@metu.edu.tr)

Erhan Anuk (erhan.anuk@ceng.metu.edu.tr)

İnan Kanbur (inan.kanbur@ceng.metu.edu.tr)

Ozan Alptekin (ozan.alptekin@ceng.metu.edu.tr)

Umut Ermiş (umut.ermis@ceng.metu.edu.tr)

**Abstract.** Collection, storage and analysis of multiple hosts' audit trails in a distributed manner are known as a major requirement, as well as a major challenge for enterprise-scale computing environments. To ease these tasks, and to provide a central management facility, a software-suit, named as "Log-Hunter" has been developed. Log-Hunter is a secure distributed log server system which involves log collection and consolidation in a large-scale environment having multiple hosts that keeps at least one audit trail. This architecture also eases the inspection and monitoring of the audit trails generated on multiple hosts. By consolidating all the audit trails on a centralized server, it significantly reduces the manpower requirement, and also provides secure log storage for inspection of log entries as it becomes necessary. This paper presents the functional specifications, architecture and some preliminary performance results of the Log-Hunter.

## 1 Introduction

In large scale computing environments, keeping audit trails on a centralized server is a crucial task. Besides many other aspects, these audit trails are very important in terms of security and recovery related issues [1,2]. However, in an enterprise network accommodating many hosts, keeping track of all audit trails is a difficult and challenging problem. On the other hand, inspection of audit trails on every single host is both inappropriate and time consuming.

Collecting audit trails from their hosts at the time they are generated, and consolidating them into a centralized server in a real-time manner seems to be the best solution. Since audit trails on different hosts are naturally kept in different formats, it is also a good idea to have a common log format for consolidation. Storing these consolidated audit trails on a non-volatile media (such as WORM) increases the

integrity. Finally, all the hosts are desired to communicate with the log consolidation server in a secure way, and time synchronization must be held between all the hosts and the central log consolidation server.

With the motivation of these ideas, a log server have been designed and implemented. We call this software as “Log-Hunter”, which is a secure distributed log server system for large-scale enterprise environments.

## 2 Log Hunter’s Functional Specifications

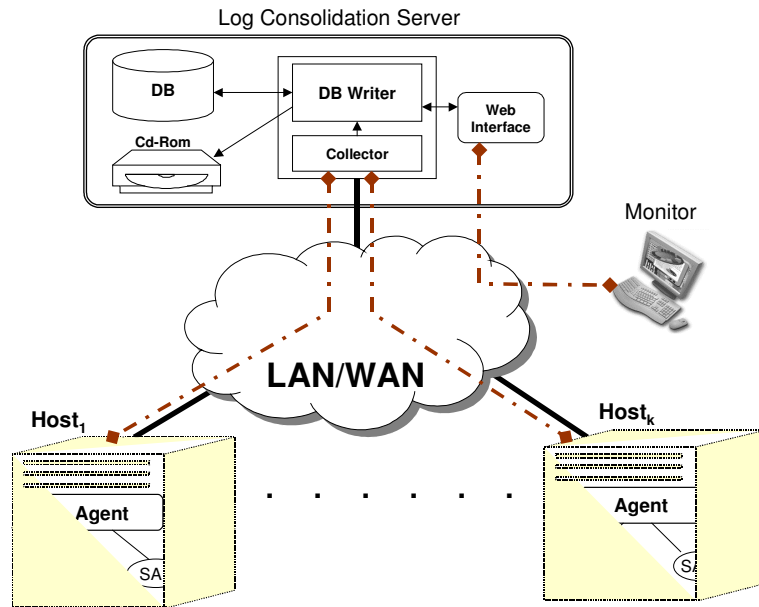
Log-Hunter architecture has the following major features:

- Keeps integrity of audit trails collected from different hosts on a network with a synchronized timestamp.
- Supports platform independence. It runs on all Unix and MS Windows environments. It can also be easily tailored for proprietary platforms.
- Delivers collected audit trails in an efficient and secure way to a centralized log consolidation server.
- Provides secure storage of consolidated data in a hierarchical manner on conventional RDBMS running on disk along with a copy on non-volatile (WORM) media to keep a secure backup copy.
- Provides services to compare secure copies of log entries on non-volatile media with those on RDBMS.
- Provides services to support analysis/correlation of stored log entries.
- Provides a plug-in structure for handling new types of audit trails.

## 3 System Architecture

Log-Hunter consists of three main modules: Agent–subagent Module, Secure Channel Module and Consolidation Server Module. Fig. 1 depicts the general structure of the system.

The overall architecture involves one central log consolidation server and multiple instances of log collection and communication agents that are to be installed on every host that keeps at least one audit trail. The central log consolidation server collects all the audit trails coming from the agents through a secure channel [3]. In order to provide unique and synchronized log timestamps, all software components are time synchronized by using Network Time Protocol (NTP).



**Fig. 1.** General Structure of the Log Hunter

### 3.1 Agent Module

The main duty of Agent-subagent module is to collect audit trails on different hosts of a network and to send these log records to the consolidation server. Since an enterprise network may contain multiple hosts with different operating systems like NT or UNIX, Agent-subagent module is designed to run in both Windows and UNIX environments. On every host in the network there are a single agent and one or more subagents communicating with their superior agent.

Each subagent watches the log file associated with it, and upon arrival of a log record at that file, fetches the newly coming log record. After formatting the log record with the common log format, the subagent sends it to the agent [4].

In UNIX environments, the sub-agents are designed as shared objects consisting of common function interfaces. All the sub-agents are placed in a previously identified directory, and the agent loads all the available sub-agents in that directory at start-up time using UNIX dynamic library functions. On Windows platforms sub-agents are designed as dynamic link libraries. As in UNIX environments they reside in a predefined directory. They are loaded and used by functions in Win-32 API.

The agent collects all the records coming from its subagents, and sends those records to the log consolidation server through a secure channel. Agents are designed in a multi-threaded architecture. In this architecture, agent is the mother process, and subagents run as separate threads of that agent. Every agent is extendible through dynamic loading of subagents. In case of a new type of audit trail to be considered, a new subagent can be installed and dynamically loaded to watch a particular log file. This plug-in approach eliminates the need to install and run multiple different programs to handle every different audit trail format.

### 3.2 Secure Channel Module

Communication between the agents and the log consolidation server is done through SSL (Secure Socket Layer) protocol, which ensures the security of log transfer between an agent and the log consolidation server. SSL encrypts data and ensures its integrity. Whenever an agent is to be installed on some host, a digital certificate is generated for authentication purposes on both sides of the communicating entities, i.e. between an agent and the log consolidation server. Communication between the agents and the log consolidation server is done by two sub-modules, *AgentPort* and *ServerPort*, respectively. There is also one more sub-module, *inserter*, which embeds the digital certificates into the pre-compiled binary code of the both agent and server module. This embedding operation is designed for security reasons; certificates exist as files in the system and therefore they are vulnerable to attacks, modification, replication, etc. Since generation of certificates is done separately, inserter module is designed to embed the certificates into the binary of the agent and server codes just before execution. AgentPort is responsible for establishing the connection to the server, encrypting and sending common formatted log records. Throughout the handshake process, digital certificate of the agent, which is embedded into binary of agent, is used for verification purposes. Agent also authenticates the server it is connecting to. ServerPort verifies connecting agents through their certificates and the list it has in which any new coming agents is introduced to server using web interface. ServerPort is designed in multi-threaded architecture, so for each connection a thread is created and each has its own queue.

### 3.3 Log Consolidation Server Module

The Log Consolidation Server (*LCS*) collects all the audit trails sent by the agents in that environment. Those collected log entries are stored in a relational database, and at the same time, they are written onto non-volatile media to obtain a secure copy. Moreover, for the purposes of displaying and analyzing the stored log entries as well as controlling Log Hunter system configuration, a web interface has been developed.

LCS has two sub-modules. One of them includes the tasks of collecting, inserting into the database, and writing onto non-volatile storage of the collected audit trails (a similar approach can be found in [5]). The other one is the web interface sub-module.

Collecting, inserting and writing processes are done in this order for a single log entry. However, handling incoming log records without any loss, internal multiple queues exist for keeping up with bursty traffic. For security reasons, all of these processing are confined in a single process address space. The mother process creates and uses multiple threads for each of these processing jobs.

### 3.3.1 Log Sub-Module

Log Sub-module is responsible for collecting and inserting log entries into the database, and writing them onto non-volatile storage.

When the Log Sub-Module is started main thread of the program creates three threads for the tasks of database insertion, writing onto CD and accepting new agent connections. The thread that accepts new secure connections is created first, and then database insertion thread is created.

DBWriter thread created by the main process does database insertion. It gets collected audit trails from queues of the threads listening to the agents. DBWriter thread searches all queues for new audit trails and if finds some, inserts them into the database. Before inserting it, DBWriter thread checks the integrity of the information of audit trail by the pre-inserted agent and subagent information in database. In any error situation, audit trails are written into associated log files. The other job that is done by the DBWriter is to calculate the time of writing a cluster of audit trails, and then triggers the thread responsible for writing onto CD.

TCPListener is the thread that deals with agent connections. It listens to a specified port and creates a new thread for any coming requests named as Collector threads. Each connected agent has a separate Collector thread. First of all, a secure communication channel should be established before sending log records. It requires any agent that will be added to the system should be introduced to LCS by using services through the web interface, such as giving new id number and creating new digital certificate for new incoming agent. Any host that has not been introduced to LCS yet would not be able to connect to the system. Any incoming connection request is examined by the ServerPort (*see sec3.2*) inside the Collector thread, which controls the digital certificate of the connecting peer and compares its information with the pre-stored information in the database. By the creation collector thread, a new queue space from memory is allocated for incoming log records. If the authentication and verification checks are successful, Collector thread starts to receive the log records sent by the agent. After fetching log records, Collector thread puts them into its own queue for later manipulations. These collector queues are shared memory used by Collector threads and DBWriter thread. In other words, there is a producer-consumer relationship between Collector threads and DBWriter threads. When agent cuts the connection, associated Collector thread kills itself.

Another thread of the main process is for writing received audit trails into CD, which is CDWriter thread. This thread is created by main thread at the start of execution. DBWriter triggers CDWriter thread according to whether a condition is satisfied or

not which is the number of audit trails in a cluster. First job done by CDWriter thread is to write the audit trails into a file. After CDWriter thread changes this file into "ISO" file and triggers the CD-writing process for writing ISO formatted file into CD. This thread is also change CD related information in the database according to the result of burning CD process.

### 3.3.2 Web Sub-Module

Web interface is designed for the purposes of displaying, analyzing the stored log entries and controlling Log Hunter system configuration. PHP is used for creating connection to database and querying 'LOG' table in the database. In this module, search pages are designed according to selected criteria. Besides these, an instant monitoring page showing the last 2 minutes of inserted audit trails into LOG table is provided. This monitoring page refreshes itself at every n seconds, which is a user-configurable parameter. For implementation of Web Sub-Module, Apache Web Server is installed in the Consolidation Server.

## 4 Summary and Results

The preliminary results obtained from the experiments done on a 10 Mbps Ethernet Network are summarized in the following tables.

**Table-1 Consolidation Server (Pentium III 800 MHz, 512 MB SD-RAM)**

Duration: 60 min. Log Entry Size: 1074 B # Of Agents: 5 # of SAs: 2/Agent	Agent-1 (Windows)		Agent-2 (Windows)		Agent-4 (Linux)		Agent-5 (Linux)		Server (Linux)
	SA-1	SA-2	SA-1	SA-2	SA-3	SA-4	SA-3	SA-4	
Generated/Received Log Entries	69,098	35,884	87,125	35,899	55,328	63,028	54,934	54,435	455,731
# of Log records Generated- Received/sec	19.194	9.968	24.201	9.972	15.369	17.508	15.259	15.121	<b>126.59</b>
Wait Time Lower Bound (sec)	0.9	1	0.9	1	0.9	0.9	0.9	0.9	-
Wait Time Upper Bound (sec)	1.1	1	1.1	1	1.1	1.1	1.1	1.1	-
Burst Record Count	20	10	25	10	15	17	15	15	-
Average Memory Usage	6 MB		6 MB		1.8 MB		1.8 MB		46 MB
Average Processor Usage	1%		1%		1%		1%		71.8%

**Table-2 Consolidation Server (Pentium 4 2.0 GHz, 1 GB RD- RAM)**

Duration: 60 min. Log Entry Size: 1074 B # of Agents: 5 # of SAs: 2/Agent	Agent-1 (Windows)		Agent-2 (Windows)		Agent-4 (Linux)		Agent-5 (Linux)		Server (Linux)
	SA-1	SA-2	SA-1	SA-2	SA-3	SA-4	SA-3	SA-4	
Generated/Received Log Entries	250,248	42,189	280,519	35,255	165,983	128,441	163,802	165,931	1,232,368
# of Log records Generated- Received/sec	69.513	11.719	77.922	9.793	46.106	35.678	45.501	46.092	<b>342.32</b>
Wait Time Lower Bound (sec)	0.9	1	0.9	1	0.9	0.9	0.9	0.9	-
Wait Time Upper Bound (sec)	1.1	1	1.1	1	1.1	1.1	1.1	1.1	-
Burst Record Count	70	12	78	10	45	35	45	46	-
Average Memory Usage	8 MB		8 MB		3.4 MB		3.4 MB		46 MB
Average Processor Usage	1%		1%		1%		1%		73.4%

In these experiments, subagents are synthetically constructed to measure the peak performance. Each sub-agent is programmed in such a way, that they wait for a random amount of time uniformly distributed between *wait-time-lower-bound* and *wait-time-upper-bound*, and then they generate a burst of records specified by the *burst-record-count*. Sub-agents' parameters are chosen to be "aggressive" rather than being realistic, in order for observing the limits of processing.

It has been observed that agent-subagent modules consume very little resource (processing time and memory) on their respective hosts.

The maximum processing rate of logs has been found as 342 logs/sec on second configuration (Table-2). The major bottleneck of the server is observed on I/O subsystem, which can easily be enhanced by using SCSI disks, RAID subsystem, appropriate kernel parameters etc.

As a near-feature work, experiments will be carried out on a resource-rich hardware for the log server. In parallel to experimentation, the software will be enhanced by adding heartbeat and fail-recovery features among agents and the log consolidation server.

## References

- [1] S. E. Hansen and E. T. Atkins, "Automated System Monitoring and Notification With Swatch", Proceedings of the Seventh Systems Administration Conference (LISA VII), 1993.

- [2] S. Axelsson, U. Lindqvist, U. Gustafson and E. Jonsson, "An Approach to UNIX Security Logging", Proc. 21st National Information Systems Security Conference, 1998, pp. 62-75.
- [3] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics", In ACM Transactions on Information and System Security, Vol. 2, No. 2, May 1999, pp. 159-176, USA.
- [4] M. Bishop, "A Standard Audit Trail Format", In Proc. 18th National Information Systems Security Conference", 1995, pp. 136—145.
- [5] C.J. Antonelli, M. Undy and P. Honeyman, "The Packet Vault: Secure Storage of Network Data", In Proceedings of the USENIX Workshop on Intrusion Detection and Network Monitoring, April 1999, USA.
- [6] John Kelsey and Bruce Schneier, "Minimizing Bandwidth for Remote Access to Cryptographically Protected Audit Logs", in Proceedings of Recent Advances in Intrusion Detection, 1999.