SECURE AUDITING FOR LINUX (SAL)
SOFTWARE DESIGN DOCUMENT

Version 1.0
Date: 02/28/03

Security Markings: System documentation is Unclassified.

Table of Contents

1.0 Identification

Secure Auditing for Linux is a research project funded by the Defense Advanced Research Projects Agency (DARPA). The project will develop a kernel level auditing package for Linux (Red Hat distribution) that is compliant with the Common Criteria specifications (DoD 5200.28-STD C2 level equivalency) and provides features to protect logged information from unauthorized modification through the use of encryption techniques. The overall goal of SAL is to produce log files that can be taken into a U.S. court and used as direct evidence in their proceedings.

1.1 Background

According to the Guidance and Policy for the Department of Defense (DoD) Global Information Grid (GIG) Information Assurance (IA) document, it is DoD policy that the DoD defense in depth strategy will provide appropriate degrees of protection to all computing environments (i.e. hosts and applications). Also according to the DoD Guidance and Policy document, GIG information systems will be monitored in order to detect, isolate, and react to intrusions, disruption of services, or other incidents that threaten the security of DoD. It is also required that there be a way to collect and retain audit data to support forensics relating to misuse, penetration, reconstruction, or other investigations. It is well known that the current auditing capabilities of Linux do not satisfy DoD 5200.28-STD C2 specifications (now defined by the Common Criteria).). NSA, developing Security Enhanced Linux, has identified auditing as an area that requires improvement. According to the GIG IA document, all GIG information systems and networks will be certified and accredited in accordance with the Department of Defense Information Technology Security Certification and Accreditation Process (DITSCAP - DoD Instruction 5200.40). During a forensics investigation, law enforcement will often rely on audit and transaction logs as a source of evidence. However, they must also be able to prove that a malicious person has not altered those logs. Section 69 of the Police and Criminal Evidence Act 1984 states "…Logs produced by a computer are not admissible as evidence unless it can be shown that there is no reasonable ground for believing them to be inaccurate and the computer was operating properly during the collection of data". If it can be shown that the logs could and may have been tampered with, they are not admissible as evidence. Forensics investigators can have minimum assurance on logs that maintain date/time stamps and checksums. According to the DoD GIG IA document, systems must "collect and retain audit data to support forensics relating to misuse, penetration reconstruction, or other investigations."

From the DoD and law enforcement perspective, audit logs are not only a necessity, but also a requirement to provide a secure open-source operating environment. The goal of this project has been to create a kernel-level auditing facility that not only monitors all processes and records events, but also provides a way to store the data that would allow it to be admissible in a court of law (i.e. encrypted, cryptographic checksum, exporting to a serial device, etc). We believe this capability would be a benefit not only to law enforcement, but also to all of DoD in support of the GIG information assurance objectives.

As stated earlier, there is a lack of a satisfactory auditing package for the Linux kernel. If an open source operating system is to be seriously considered for use on the Global Information Grid, it must meet basic security standards. Linux does not currently provide C2 level auditing

as defined by the Trusted Computer Systems Evaluation Criteria (TCSEC - DoD 5200.28-STD). One of the fundamental security requirements in the TCSEC is accountability. From the publication: "**Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party.** A trusted system must be able to record the occurrences of security relevant events in an audit log. The capability to select the audit events to be recorded is necessary to minimize the expense of auditing and to allow efficient analysis. Audit data must be protected from modification and unauthorized destruction to permit detection and after-the-fact investigations of security violations." For C2 level auditing, the Trusted Computing Base (TCB) must be able to create, maintain, and protect from modification or unauthorized access or destruction, an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction or objects into a user's address space (e.g., file open, program initiation), deletion of objects, and actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity. To provide some of the basic requirements of TCSEC, it would need to at least report on the following: 1) date/time stamps for all user logons and logoffs, bad password attempts, adding/removing user accounts, remote access to local files. 2) File operations, like open, close, create, delete, and rename. 3) Changes to security privileges (chmod, chown, setuid), information on system load or disk space usage (for covert channels). In addition to the TCSEC, the Common Criteria, the new criteria for evaluating information technology security, will also be a source of information in ensuring basic auditing functionality is provided. Some of the other activities that should be monitored include IPC activity, network activity, some system calls, resource locking activities, and process termination.

Not only can this audit package become a valuable troubleshooting tool for a system administrator, it can also be a tool for computer forensics. Audit logs are one source of information in a computer forensics investigation, but can be a valuable source. Date/time stamps reflecting when a user accessed the system or accessed/modified files can present an accurate picture of events. However, tools exist that allow malicious people to modify audit logs on some systems. With the existence of tools like those, it is easy to understand why there is not a high level of assurance on audit logs. Providing encryption, file monitoring, and multiple locations for file storage (local, remote, serial device), system administrators can have a higher level of confidence that audit logs have not been tampered with.

According to the GIG Information Assurance Guideline and Policy document, all systems must pass the DITSCAP. If a C2 compliant auditing package is not developed for the Linux kernel, then Linux systems will not be allowed to be part of the GIG. It is essential that security packages such as this make it into the Linux kernel and other open source operating system kernels if they are to play a role in the GIG.

2.0 Document Overview

This document is intended to provide a description of the architecture, design and development the Secure Auditing for Linux (SAL). The document will initially describe the high level architecture of SAL, followed by a high-level description of the major components of the system, followed by a detail-level description of the components. The detailed description of the components is actually being generated using Doxygen, a source code documentation extraction tool. Finally, a Requirements Traceability Matrix (RTM) will be provided to that maps all software requirements to a specific component in the design document. Configuration files and message interface descriptions will be included as appendices to this document.

Fonts used in this document –

Italic	*Configuration File / Script File / Book Title*
`Courier (12 pt)`	Name of program that is executed.
`Courier (10 pt)`	Class/Component Name/System Call

References

a. “Secure Auditing for Linux Software Requirements Specification Version 1.02” July 2, 2002
b. DoD CIO Guidance and Policy Memorandum No. 6-8510 “Department of Defense Global Information Grid Information Assurance” June 16, 2000
c. DoD 5200.28-STD, "DoD Trusted Computer Security Evaluation Criteria," December 1985
d. Common Criteria for Information Technology Security Evaluation Version 2.1, August 1999
e. National Security Telecommunications and Information Systems Security Policy (NSTISSP) No. 11, "National Policy Governing the Acquisition of Information Assurance (IA) and IA-enabled Information Technology Products," January 2000
f. DoD Instruction 5200.40, "DoD Information Technology Security Certification and Accreditation (C&A) Process (DITSCAP)," December 30, 1997
g. DoD Directive 8500.1 “Information Assurance” October 24, 2002
h. Network Security with OpenSSL by John Viega, Matt Messier, Pravir Chandra; O'Reilly & Associates; ; June 15, 2002
i. Apache XML documentation
j. PEO encryption documentation
k. SDSC Secure Syslog
l. Syslog
m. Appropriate RFC’s

3.0 High Level Design and Architecture

The Secure Auditing for Linux (SAL) is a kernel level auditing tool used to accurately and securely log audit events. SAL is a client/server-based application that generates audit events on client systems, encrypts the events, and then transmits them to a dedicated log server for secure storage. The SAL application is broken into 3 major functional areas, a SAL Instrumented Client (SIC), a Secure Network Transmission Tunnel (SNTT) and a SAL Log Server (SLS).

We start the architecture discussion with a brief talk about the SIC. A SIC is any computer that contains a special audit generating Linux operating system (OS). The special OS contains static kernel-level source code that monitors system activity. This code monitors a specific set of pre-described system calls (that must be defined at OS kernel building time). As monitored system calls are executed the modified OS records specific state data from the computer and generates an audit event. These audit events are stored in kernel level system buffers until a SAL application daemon, **`auditd,`** has the opportunity to record the kernel level system buffers to disk. From disk the audit events are collected, encrypted and then transmitted to the SLS via the SNTT.

The SNTT is a secure, encrypted network transmission protocol. We have currently implemented this protocol using the Secure Socket Layer (SSL) libraries developed by the OpenSSL project (www.openssl.org). This component of the SAL Architecture is very straight-forward. It is used to establish secure connections between the client and the server. It is further used to provide file-based key management. This SNTT component is integrated into both the SIC and the SLS. On the client end the SNTT is integrated into an application level program called **`auditclient`**. When the **`auditclient`** application is started, the SNTT attempts to establish a connection with the SLS portion of the SNTT. Once a connection is established the **`auditclient`** program transmits the audit entries buffered to disk to the server. On the SLS side the SNTT is integrated with the **`sald`** application. The **`sald`** application is the main program for executing on the server. A `collector` component in the **`sald`** software is responsible for the interface to the SNTT module. On the SLS the SNTT software is further responsible for providing additional hashing and encryption services.

The SLS is the log server for the SAL application. The SLS records audit data to a storage media for later processing. The SLS establishes the **`sald`** application as a server application and waits for connections to be established by SIC. The SLS is a multi-threaded application. As new SIC(s) connect to the server a new thread is created to handle the communications with the server. The server provides a number of new features such as: audit log size thresholds based rotating, email, or syslog capable alerting, dynamic file system support and dynamic server configuration. As audit events are received by the server, server Meta-Data is attached to the event. This Meta-Data provides chain of custody markers, time of receipt indications and based on configuration message checksum values. Once received and processed, audit events are stored to disk in such a way as to prevent the data from later being modified or altered. Additionally as files are closed a certification hash is produced to provide external file authentication. The SLS is also responsible for continuously verifying the integrity of the previously recorded data. This is accomplished by the `archiver` component of the SLS. On a

configurable time interval the `archiver` is requested to verify that all the previously recorded audit events pass verification. If there is an indication that a file has been tampered with alerts are raised to inform the appropriate responsible person. While one of the goals of the SLS is to not allow any modification to previously logged data we are not naïve enough to believe that it can't be. Therefore it is a goal of the SLS to be able to identify when data has been modified or altered and if possible limit the potential corruption to the smallest subset of data possible. Additionally the SLS provides a java-based tool that, assuming a properly authenticated user, will parse audit event logs and produce a clear text (human-readable) data representation. This tool is currently the only mechanism available for a user to get access to the audit data. It provides the ability to search through the audit event logs for specific SIC and then through the specific logs for time intervals. Additionally, a limited filtering capability will be provided. This filtering capability will evolve to allow users of the tools to eventually request a specific set of events to trigger alarms.

4.1 Use of Encryption with SAL

Data privacy, integrity, and authentication are the primary reasons SAL uses data encryption. Date privacy is primarily needed while data is in transit over the network. For this reason all application-level network communication is encrypted. Data integrity is also maintained by the use of encryption algorithms. Here it is used by the server in the generation of a hash for each file that is written. Additional at specified intervals all the audit files stored on the server are rechecked to determine if any modification have occurred. Lastly client authentication is assured by the use of encryption. For the data encryption portion of the SAL program our development team selected the OpenSSL libraries. These libraries have become the de facto standard in open source software. They are complete, well tested and have a thriving development community which maintains the software on multiple platforms. Additionally, the OpenSSL libraries are GPL therefore they are free for our purposes.

4.2 Trade Offs

Client Kernel Patch vs. Kernel Module: We had a lively discussion on this topic. Several of the developers wanted to develop a SAL kernel module for the clients. The module approach is very similar to the approach taken by several other auditing applications. There were also many good reasons for the module approach (no kernel code changes, dynamic installation, no need to rebuild the kernel, more supportability for module changes). But while these good reasons do exist, they were also some of the downfalls for use of a plug-in module. Chiefly the module approach allows for the possibility of the module's removal, thus turning off auditing. We determined that type of vulnerability would make our system not meet our primary goal of C2/CC compliance. We did realize that we could minimize the chances of it being removed, and that only root could remove it but any ability to turn off monitoring was something we wanted to avoid if at all possible. We also considered the performance difference between a plug-in module and static kernel code and determine that static kernel code was going to execute slightly faster because of where the code is being executed (ultimately there are less instructions involved in static kernel code than in a plug-in module. Additionally there was talk about the monitoring capability of a plug-in module and if it could allow the same system call resolution that static kernel code could. As the Linux kernel 2.5 hits the street we will be looking into the use of the Linux Security Module (LSM) however as of this document we have not begun that work.

Log Server on Different Computer: When the SAL program was first discussed we talk about the possibility of allowing the server to run on the same computer that generated the audit events. We figured this implementation would be great for small installations and this seemed highly desirable. If we would allow the SAL server to co-exist with the client software that would be great for testing, great for demos and great for installations with limited system resources. However, it would not be great for our security requirements. We talked at length about the possibility of doing this, but ultimately the question we came up against was "Is this approach really any different then the current logging approach?" We had to answer NO. Most logging systems of today follow the design of allowing logging to be performed on the system that is actually generating the data. According to our interpretation of the C2/CC requirements and discussions with our forensic team and lawyers we determined that the log data "MUST" be removed from the system to ensure any integrity of the data. The need to ensure this integrity over-rides the resource issues that this approach adds for its use. We are still discussing the possibility of allowing the server to execute on the same computer as the client as a possible configuration however this will be worked on as time available basis.

Use of Little Files on client: The use of the little files to buffer data on the client still generates a lot of discussion within our team. The little files are used to buffer audit data from the kernel prior to transit to the server. We use the little files to quickly (at the highest priority that Linux allows) write the data received from the kernel to the file system. When the file is close and a second is opened, the consuming program `auditclient` then reads the data from the closed file and sends it to the server. Then `auditclient,` then is responsible for removing it from the system. In a steady state condition, the files will practically never go to the disk, staying instead in system buffers. If the communication with the server fails (for whatever reason) this approach will automatically spool the files to disk. We have demonstrated that the load on the system can be nearly 100% before the kernel fails to write all of its system calls. After examining other approaches and running benchmarks we determined that we have a valid design. Other approaches (memory, pipes, etc) quickly consumed the buffering resources. We believe that eliminating the little files will cause the integrity of the system to be diminished and that this is unacceptable.

Multi-Threaded Server Application: When we were considering the design of the server we talked about both multi-threaded and multi-process server application approaches. We spent some time coming up with tradeoffs for both approaches because they both had merit. We saw the benefits of the multi-process approach being fault tolerance as in case one of a connection causing the connection application to crash. Additionally a multi-process approach can provide for a better load balancing capability. The problem with the multi-process approach is generally due to its implementation. For multi-process implementation process synchronization and Inter-Process Communication (IPC) issues can often prove more trouble then they are worth. We saw a multi-threaded application being basically the reverse of the above. Multi-threaded applications allow threads to share a common address space so inter-thread communications is very easy, and synchronization between threads is handled very easily with mutexes. This approach also had some pitfalls. The chief pitfall is, if any of the connections with clients cause a crash, the entire application crashes. At this stage of the SAL program it was determine that development time was at a premium so the benefits of easier development outweighed the

pitfalls associated with multi-threaded development effort, so the multi-threaded approach was taken.

XML Server Configuration File: XML was chosen as the configuration file format because of its supportability. Virtually every language we considered using (C/C++, Java) and many of the tools we planned on developing with (Perl, PHP) had XML libraries. The main application for the server uses the Xerces-C++ parser. There are versions of this parser available for Java, as well as Perl and PHP.

4.3 High Level Design of the Components

4.3.1 Instrumented Client

An Instrumented Client is any computer with an interface to a SAL server. Currently only one implementation of this interface is available. That implementation exists on computers running a specially modified Linux operating system/kernel. We currently support the RedHat 7.3 Linux distribution. (A port to the RedHat 8.0 Linux distribution is underway.) That `SAL Kernel` is available as a patch to the standard RedHat Linux kernel. The functionality provided by the patch is system call monitoring. The actual system calls monitored depend on how the kernel is configured, which is defined at kernel compile time. A Perl script (*sal-conf.pl*) is responsible for modifying the kernel based on arguments provided by the kernel builder (HIGH, MEDIUM, LOW). This setting defines the amount of auditing that is performed. (See below for a list of monitored system calls based on the selected option.) Once the kernel is instrumented and rebuilt the system must be rebooted for it to take effect. Once the instrumented kernel is executing all monitored system call are stored into a ping-pong buffer until it's full in which case the monitor stops. During normal operations the buffer will not be allowed to fill so auditing will not be stopped. The ping-pong buffer is emptied by the `auditd` application's call to the `audit` system call which copies the audit data from the buffers into the applications local memory.

Along with the patched kernel two additional application programs are included. These programs are responsible for spooling data from the kernel and transmitting it to the server. The program responsible for spooling data from the kernel is `auditd.` This program requires a temporary file directory which it uses to write spooled audit files ("little files") too. `Auditd` creates files in the temporary directory as data is processed from the kernel via a system call, called `audit`. When a size threshold (configurable via compilation) is exceeded, the `auditd` program closes it and creates another file for the next block of kernel audit data. The format for audit data file name is [suffix].##, where suffix defaults to tmp and ## is a one up counter. See Table 1 for a list of the data that is gathered from the `SAL Kernel.` The auditd process continues this execution cycle until it is stopped by external events.

The `auditclient` application is responsible for establishing a secure connection with the server and then transmitting the spooled audit data files over the network to the SAL server. Upon application startup the `auditclient` application attempts to establish a secure connection with the SAL server using a previously provided encryption key. Currently the

`auditclient` application will wait for the server to become available before any processing will commence. Once the connection is established the application will send all backlogged data (if any) and then begin a steady state processing. Steady state process is the process where auditd creates a little file called "tmp.2323", auditclient transmits "tmp.2322" over to the server for storage and wait for the creation of "tmp.2324" before transmitting "tmp.2323" to the server. While the network connection is available and the server is able to keep up with all the possible clients that are connected the process of waiting for the next file continues. However due to the nature of the SAL application this may not always be the case (server falls behind, network connection is lost, what have you). The spooling of the data to disk by `auditd` will handle the case where the server falls behind or there is a momentary connection problem with the server. However broken connections with the server will cause the `auditclient` program to re-establish a secure connection. For this the `auditclient` application continually verifies that the connection is valid. When these types of interruption occurs the `auditclient` application attempts to re-establish a secure connection with the server in an attempt to regain a steady state.

For the convenience of users of the system a shell script is included in the tarball. This script *audit.sh* is responsible for starting the `auditd` and `auditclient` application in a clean state. Special care must be taken when using this script. It removes the directory where the kernel audit files are stored prior to recreating it. If there are any files that have not been sent over to the server in that directory they will be lost.

Information gathered by the Instrumented Client

Name	**Comments**
serial	A one-up counter representing the sequence number for the system call.
ts_sec	Timestamp: when the syscall occurred represented by the number of seconds since January 1, 1970.
ts_micro	Timestamp (complements ts_sec): the number of microseconds between each second.
syscall	A numerical representation of the system call. These numbers can be linked to a system call name in `arch/i386/kernel/entry.S`.
status	The return status of the system call.
pid	The Process ID: defined in `include/linux/types.h`. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
uid	The Real User ID: defined in `include/linux/types.h`. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
euid	The Effective User ID: defined in `include/linux/types.h`. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
device	A numerical representation of the tty. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`. If no tty is defined the tty_struct of the current task structure is null.
comm	The name of the program that executed the process that called the system call. This information is obtained from the Linux kernel's current task structure

(task_struct) defined in `include/linux/sched.h`.

Table 1.

Instrumented Client Application Programs and Scripts:
Audit.sh: This script removes the temporary directory (TMP_DIR) if it exists. It then calls the audit daemon (auditd) as a background process and the audit client as a foreground process. It calls both the daemon and client using the temporary directory. To kill the audit client, press Ctrl-C. This only kills the client; however, to kill the audit daemon run, '`killall` -s9 auditd'. Killing the auditd daemon will cause the kernel to eventually suspend auditing as the collection buffers cannot be emptied. To unsuspend use '-r'. Below is the audit script included with the tarball.

./audit.sh

Auditd: The daemon gets the audit data out of the kernel. It then writes the data from the kernel to little files that act as buffers.

./`auditd` [-b <base>]
[-c <number>]
[-d <directory>]
[-r]

OPTIONS

-b base	use 'base' as the file base of the temporary files, the default is 'tmp'
-d directory	the directory where the daemon writes the temporary files, the default directory is 'tmp'
-c number	sets the number of collection buffers that are written to a temporary file, the default is 10
-r	resets the kernel auditing if suspended; this option clears the collection buffers by dropping the data and restarts the auditing

---

Auditclient: The audit client reads the tiny files left by the audit daemon and sends the data over the network.

./`auditclient` [-a <IP address>]
[-b <base>]
[-c <directory>]
[-d <directory>]
[-e]
[-n <number>]
[-k]
[-l <file>]
[-p <port>]

[-q]
[-v]

OPTIONS

-a IP address	the IP address of the server that is collecting the data; the IP address to which the client sends the data
-b base	Use 'base' as the file base of the temporary files, the default is 'tmp'
-c directory	the directory from which the client reads the its certificate, the default directory is '/CA'
-d directory	the directory from which the client reads the temporary files, the default directory is 'tmp'
-e	starts with the earliest file in the temporary directory; this option overrides the -n option, so if '-n 10 -e' is given and the earliest file found was 2, it would start at 2
-k	keep the temporary files, do not delete
-l file	logs to a file the last temporary file the client opened for reading
-n X	starts processing at the specified temporary file X, the default is 0
-p	port the port the server is listening on (default 5000)
-q	finish up; read all the available files and then quit, do not wait for the last one. this option starts with the earliest file (just like -e)
-v	verbose

4.3.2 SAL Log Server

The SAL Log Server is responsible for the permanent storage and protection of SAL audit events generated by clients. It performs these tasks by starting the `sald` application at system startup run level 2 or 3. The SAL server can be run headless if properly configured. The `sald` application is a single, multi-threaded application that has one thread handle for each client that connects to the server as well as a thread to support its email and archiving functions. Once started the `sald` program processes the *sald_conf.xml* file and establishes a number of signal handlers. The *sald_conf.xml* file is responsible for configuring the `sald` application (see detailed description of the fields below). Based on the configuration the server establishes connections to the specified networks. The server can support more then one network at a time. Also based on its configuration the server will alert the administrator the application is executing properly. The completion of the server initialization allows the server to enter into its two main processing states, the connection state and the archive monitoring states. These states have their own processing capabilities which will be outline below. Along with these two major execution states a number of utility threads are started to help support the server's operations. These include a main controller thread, a debug output thread, and an email processing thread.

The main controller thread handled by the `controller` class is responsible for the `sald` application startup; it also handles the creation of a configuration file parsing class and the establishment of the signal handlers. While the SAL server requires a configuration file some of the options can be over-ridden by the command line (See Server Command Line Options).

The connection monitoring capability is handled by the `collector` class. It is responsible for establishing and support network connections, receiving client connections, verifying client access, creating a thread to receive the audit data, and creating a client session for processing the client's audit data. When a client initially connects to the server, a verification process is run. It is already known that the client has the appropriate SSL key, so its Internet Protocol (IP) address, its Media Access Control (MAC) address are also checked. The `sessionManager` class is responsible for this verification. The verification is run against what is stored in the server's configuration file. If the client's connection is accepted a `session` is created to support the client and returned to the `collector`. The server, currently, supports two types of sessions, a blob session and text sessions. These sessions support the different data formats defined in appendix B. Initially the `session` creates an `auditDataFacility (ADF)` that is responsible for handling the rotation of the audit files. The `ADF` then waits to process audit data. After a block of audit data has been received and verified by the `session` an `auditEntry` class is created, propagated, and passed off to the `ADF` for storage. The act of storing the `auditEntry` is physically performed by the `auditFile` class. The `auditFile` class is created by the `ADF` to handle the recording of the data to a storage media. As data is written to disk the `ADF` determines if configured thresholds have been exceeded and if so rotates the log file. Once a log file gets rotated the `certificationDB` class is informed and takes ownership of the file and a new auditFile is created to process the next `auditEntry` class that is received.

Archive monitoring is handled by the `archiver` class. It is responsible for maintaining the integrity of the stored audit files. It performs this function by continually verifying the status of the archives. If stored audit files are tampered with alerts are sent to the administrator indicating the problem. The revalidation of the archive is performed every 10 minutes (the default value may be changed via the configuration file). The `sald` application starts an archive monitor thread that is responsible for querying the `certificationDB` for it status. The status of the `certificationDB` is based on computed hash functions. When the `certificaitonDB` takes ownership of an audit log file a SHA hash is computed and stored in the data based. When a request to validate the data base is processed all the hash values are recomputed for the audit log files with entries in the data base. If there are discrepancies with the hash values or there are missing or modified file system stat inconsistencies, an alert is sent to the administrator (configuration dependent).

The server supports additional threads to handle debug and emailing. These threads are created by the `controller` class to provide additional functionality. The `debug` class allows the `sald` program to have a method of dynamically dumping information helpful to the persons working with the application. It can be dynamically changed without stopping the server by modifing the proper value in the configuration file and sending the SIGHUP signal to the `sald` application. A script file *./newconfig.sh* has been included to make that task simpler. There is also a script file named *./salreport.sh* that will produce a report of the running application. This report

provides information about the uptime the individual connections as well as the status of the certification data base. The output for this report is specified in the *sal_config.xml* file.

To make the configuration of the SAL server easier a `Webmin` module has been developed. This module allows the administrator the ability to configure all elements of the server as well as generate encryption key for both client and the server.

4.3.3 SAL Java GUI

The SAL Java GUI is a Java-based application designed for displaying the contents of the audit files stored by the SAL Server. The GUI displays data for each client that has connected to the server and sent audit data. The GUI reads these audit files and displays either a text summary of the data, a pie chart summary of the data, or a selected number of the system calls stored in the selected file. Access to these audit files is defined in either the *config.app* file, or on the `Preferences` tab in the GUI, by setting the `Log Path` field to the location where the server stores it's audit files.

Before starting the GUI, the *gui_config.pl* script must be run to generate a list of system call names in the *syscall_config* file. These system call names are read by the GUI on startup and used to match the system call number stored in the server's audit files with its appropriate name. Since the entries in the *syscall_config* file are generated by examining the appropriate *entry.S* file on the server's machine, there could be an incorrect matching of system call numbers to their names if one of the client systems is running a version of the Linux kernel that does not match the one scanned by the *gui_config.pl* script.

The Java GUI also uses a native method, `auditViewer`, which does the actual reading of the server's audit files. The `auditViewer` native method is defined in the C module *auditviewer.c*, which returns a string containing the data requested by the Java GUI for displaying. The Java GUI can request either a summary of the audit file's data, or a specified number of the system call data stored in the audit file. In order for the native method to work correctly, the user's `LD_LIBRARY_PATH` must contain the path to the dynamic loadable library, *libviewer.so*, which is located in the root directory of the Java GUI.

The GUI consists of a Server module and multiple client modules. The server module contains a client module navigation tree that is used for switching between client modules. New client modules are added to this tree by a thread that is spawned by the server module when the user requests a search for new clients. The client modules contain a `Summary` tab and a `Log File` tab. The `Summary` tab allows the user to select a file for that client from a drop-down menu, and, after pressing the `Refresh Summary` button, displays the total number of system calls stored in the file and a list of all the system call names in the file with the number of times that call was made. For a pie chart showing the percentage of system call usage in that specific file, the user can press the `View Chart` button. The `Log File` tab also allows the user to select

which file to view from a drop-down menu, and also allows the user to select how many system calls to view at a time. Once those choices have been made, and the `Refresh Log` button has been pressed, the `Log File` tab will display a list of all the system call data collected by the client, and stored in the audit file. A list of the data collected by the client, and an explanation of this data can be found in Table 1.

The Java GUI does not automatically detect the addition of new clients to the server, so the user will have to tell the GUI when to re-check the server's logging directory for more clients. This can be done by selecting the `Clients` option from the `Update` menu. This option will re-scan the log directory, and automatically add a new module to the GUI for any new clients detected by the scan. Also, switching between Client modules will automatically refresh the list of files available in the drop-down menus for each client.

---

gui_config.pl

This script file is used to create a listing of all the system call names available for a specified client, and it places this list in a file called syscall_config.

Secure Audit for Linux (SAL) Java GUI: This application reads the audit files stored by the server and displays the requested information from them.

```
./go
```

4.3.4 SAL Audit Viewer

The C-based SAL Audit Viewer is provided for those users that do not wish to install Java on their systems, but still want to examine the contents of the server's log files. The Audit Viewer is passed a number of options, including the name of the file to be read as an option and the path to the directory where this file is stored. It can also be passed options to print out a summary of the file's data, print out the entire contents of the file, or print out only part of the contents of the file.

When reading one of the server's files, the Audit Viewer first checks the file for embedded XML tags that the server places around the client's audit data. The viewer skips over those tags, and then reads in two packets of data, each the size of the Kernel Audit Event structure, as seen in Appendix B. It converts this data from network-byte-order to host-byte-order, checks the serial number stored in each to make sure that they are valid system calls only one number apart, and then begins to read the file until the next set of XML tags is encountered, at which point the process repeats. This continues until the end of the file, or until corrupted data is encountered. If

invalid or corrupted data is encountered, the viewer will attempt to find a place where valid data can be read, but this will only happen for a finite period of time, and will eventually exit with an error message if no valid data is found within the allowed limits.

While reading the system call data, the viewer keeps track of the total number of system calls encountered, as well as a running total of the number of times each system call is in the file. If the user requests a summary of the file, the viewer will print out this total, and a table of system call numbers, with the number of times each system call appeared in the file. If the user passes in the verbose option, the viewer will print out the full contents of the Kernel Audit Event structure for each system call encountered in the file. The user can limit the number of system call's displayed by passing a value to the `-j` option. The number passed to the `-j` option will tell the viewer to skip that number of system calls, and print out the rest. So if the user wants to see 100 system calls, and the total number in the file is 1600, the user should pass `-j 1500` to the viewer. The user can request a summary and a listing of all system calls at the same time if they so desire.

./**auditviewer** [-l <directory>]
[-s]
[-v]
<filename>

OPTIONS

-l directory	the directory where the server logs are found
-s	displays a summary
-v	verbose
-j num_skip	the number of system calls to skip when verbose

`Sal-wbm` is a `Webmin` module for the SAL Server which is used to edit the configuration file as well as create server and client keys. `sal-wbm's` web interface simplifies SAL Server configuration through six different sections/categories; General Options, Log Settings, Networks, Clients, Edit Configuration and Manage Keys.

`General Options`: Basic options may be set such as location of the configuration file. Also the high level variables in the configuration file may be changed.

`Log Settings`: Options for the SAL server logging system are defined; these include Thresholds, Client information, port, etc…

`Networks`: Interfaces for the server can be dynamically added, deleted, and changed as the Network information changes.

`Clients`: Each client requires one entry which defines the clients thresholds, alerts, Mac and IP addresses, etc…

`Edit Configuration`: Allows for manual editing of the configuration file.

`Manage Keys`: Server and Client keys are generated and stored on the servers file system. Server and client keys may be deleted and rebuild, client keys may also be downloaded.

Newconfig.sh
This script file is used to send the `sald` application a HUP signal, which causes `sald` to process the configuration again while it is executing.

Salreport.sh
This script file is used to send the `sald` application a USR1 signal, which causes `sald` to dump a report to the report file.

Secure Audit for Linux (SAL) Server: Executing as a daemon process this application waits for data from clients and stores it to disk.

./**sald** [-d debug setting]
[-l debug value 0...20]
[-c config file]
[-p]
[-k key directory]
[-f]
[-v]

OPTIONS

-d debug file	specifies the name of the file to dump debug to.
-l	specified the debug level.
-c configfile	provides the config file that the SAL server will use.
-v	print the version number of the application.
-p	parse only flag (Not currently implemented)
-k key dir	provides directory where keys are located.
-f	executes sald application in the foreground. [Defaults to background]

(SIGHUP)	Re-process the configuration file.
(SIGUSR1)	Print out a SAL Server Report.

4.3.5 Configuration File Example

```
<Configuration>
```

```
  <HashType>sha</HashType>
  <IntegrityVerificationInterval>15</IntegrityVerificationInterval>
  <BaseStoragePath>/usr/local/sal/clients/storage</BaseStoragePath>
  <DebugLevel>10</DebugLevel>
  <ReportFileName>/usr/local/sal/clients/sal_report</ReportFileName>
  <HashExecutable>/usr/bin/openssl</HashExecutable>
  <ServerKeysDir>/usr/local/sal/keys</ServerKeysDir>
  <DebugFile>/usr/local/sal/tmp/sald.log</DebugFile>
  <CertificationDBLocation>/usr/local/sal/cdb</CertificationDBLocation>
  <Networks>
    <NetworkDefinition1>
      <NetworkIP>198.253.113.243</NetworkIP>
      <NetworkName>eth0</NetworkName>
      <Port>5000</Port>
    </NetworkDefinition1>
    <NumberOf>1</NumberOf>
  </Networks>
  <SALClients>
    <NumberOf>1</NumberOf>
    <System1>
      <ThresholdAlertName>postmaster@localhost</ThresholdAlertName>
      <ClientStorage>/usr/local/sal/clients/storage/</ClientStorage>
      <LogRotateFreq>MID</LogRotateFreq>
      <Enabled>Yes</Enabled>
      <ThresholdAlert>EMAIL</ThresholdAlert>
      <ThresholdMid>40</ThresholdMid>
      <ThresholdHigh>95%</ThresholdHigh>
      <Type>FORMAT1</Type>
      <ClientMAC>"Not Provided"</ClientMAC>
      <StorageType>FLAT</StorageType>
      <Debug>0</Debug>
      <IP>198.253.113.55</IP>
      <ClientKeys>/home/sal/keys/peo_101.keys</ClientKeys>
      <ThresholdLow>20</ThresholdLow>
      <ThresholdMax>DISK_LIMIT</ThresholdMax>
    </System1>
  </SALClients>
</Configuration>
```

Meaning of Each Field.

<Configuration>…</Configuration>

Pair used to start the configuration of SAL Server

<HashType>…</HashType>

Type of hash function to use (passed as argument to HashExecutable)
md5
sha (default)

<HashExecutable>…</HashExecutable>

Name of the executable program used to compute hash values for audit log files (openssl is default).

<IntegrityVerificationInterval>…</IntegrityVerificationInterval>
Time (in minutes) between execution of Integrity verification.

<ServerKeysDir>…</ ServerKeysDir>
Pair used to identify where the SAL Server Keys are located

<DebugLevel>…</ DebugLevel>
The higher the number the more data dump to the debug file.
Range (0..20)

<DebugFile>…</ DebugFile>
File name of the debug file.

<BaseStoragePath>…</BaseStoragePath>
Default storage location if none provided.

<CertificationDBLocation>… </CertificationDBLocation>
Location of certification Database. Default "/usr/local/sal/cdb"

<Networks>…</ Networks>
Pair used to identify where networks that SAL will be listening on are defined.

< NumberOf>…</ NumberOf>
Defines the number of networks SAL will be listening on. (Only the first <NumberOf> are parsed in this file.
<NetworkDefinition#>…</NetworkDefinition#>
Defines the # network that the SAL server will be listening on.
<Port>…</Port>
Defines the Port that the network will use to start communicating.
<NetworkName>… </NetworkName>
Defines the standard name given for the interface.
<NetworkIP>…</NetworkIP>
Defines the IP address of the Network. Currently only Static address are possible.

<BaseStoragePath>…</BaseStoragePath>
Base location for data to be stored on filesystem (Either local or nfs mounted)

<SALClients>…</SALClients>
Pair used to identify where client definition begins.

<NumberOf>1</NumberOf>
Pair used to identify the number of client. (only this number of clients will be parsed in file configuration file.

<System#>…</System#>
Pair used to identify client number #

<IP>…</IP>
IP address of the client. (Only support static IP addresses, for security purposes)
<ClientMAC>…</ClientMAC>
MAC address of the client. If not available an alert will be sent but standard processing will still occur.
<Type>…</Type>
Type of messages to expect “FORMAT1” or “FORMAT2”
<Enabled>…</Enabled>
Indicates if Client is enabled for standard processing. “YES” in all forms is accepted all others are rejected.
<StorageType>…</StorageType>
Defines the type of storage that is used for final data to disk “PEO” for encryption, “STANDARD” for default. Each <TYPE> will have its own default.
<ClientKeys>…</ClientKeys>
Location of Client portion of crypto. Part of the SSL processing.
<ClientStorage>…</ClientStorage>
Location to store Client Data (will over-ride <BaseStoragePath> processing)
<Thresholds>…</Thresholds>
Identifies File threshold to alert on.
<ALERT>…</ALERT>
Identifies how to alert on error. “NONE”, “SYSLOG”, ”EMAIL”, ”CONSOLE” are acceptable answers.
<MAX>DISK_LIMIT</MAX>
<High>95%</High>
<Mid>1000</Mid>
<Low>100</Low>
</Thresholds>
<Debug>0</Debug>

5.0 Detail design of Components

The detail design of the software is available in the code. The doxygen program is used to generate HTML documentation form the source.

6.0 Glossary

ADF	AuditDataFacility
C2	Rainbow Series - Orange Book (level C2)
CC	Common Criteria
IP	Internet Protocol
IPC	Inter Process Communication
LSM	Linux Security Module
MAC	Media Access Control
OS	Operating System
SIC	SAL Instrumented Client
SL	SAL Log Server
SNTT	Secure Network Transfer Tunnel
SSL	Secure Socket Layer

Appendix A – Requirements traceability/System Call List

The following system calls are standard in the Linux operating System as of the 2.4.2-2 release of the kernel. The ones being tracked by SAL include a reference to the appropriate recording requirement from section 2.1.2 of the Requirements Document.

System Call	Description	Requirement Reference
Access	Determines whether a file can be accessed.	REC030
Acct	Switches process accounting on or off.	REC020
Adjtimex	Tunes the kernel clock.	N/A
Alarm	Sets an alarm clock for delivery of a signal.	N/A
Bdflush	Kernel daemon to flush dirty buffers back to disk.	REC110
Brk	Changes the data segment size.	REC140
Capget	Get process capabilities.	REC120
Capset	Set process capabilities.	REC140
Chdir	Chdir changes the current directory to that specified in path.	REC130
Chmod	Changes file access permissions.	REC040
Chown	Changes the user and/or group ownership of each given file.	REC040
chown16	Changes the user and/or group ownership of each given file.	REC040
Chroot	Run command or interactive shell with special root directory.	REC020/REC130
Clone	Creates a new process like fork does.	REC020
Close	Closes a file descriptor, so that it no longer refers to any file and may be reused.	REC030
Creat	Open and possibly create a file or device.	REC030/REC090
create_module	Create a loadable module entry.	REC110
delete_module	Delete a loadable module entry.	REC110
Dup	Duplicates a file descriptor.	REC030
Dup2	Duplicates a file descriptor.	REC030
Execve	Executes the program pointed to by filename.	REC020
Exit	Exit a running process.	REC020
Fchdir	Changes the current directory to that specified in path.	REC130
Fchmod	Changes the mode of the file given by path or referenced by fildes.	REC030
Fchown	Changes the owner of the file specified by path or by fd.	REC040
fchown16	Changes the owner of the file specified by path or by fd.	REC040
Fcntl	Performs one of various miscellaneous operations on fd.	REC030
Fcntl64	Performs one of various miscellaneous operations on fd.	REC030
Fdatasync	Flushes all data buffers of a file to disk (before the system call returns).	REC030
Flock	Apply or remove an advisory lock on an open file. The file is specified by fd.	REC030

System Call	Description	Requirement Reference
Fork	Creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0.	REC010/REC020
Fstat	Gets file status.	REC030/REC120
Fstat64	Gets file status.	REC030/REC120
Fstatfs	Gets file system statistics.	REC030/REC120
Fsync	Synchronize a file's complete in-core state with that on disk.	REC030
Ftruncate	Truncates a file to a specified length.	REC030
ftruncate64	Truncates a file to a specified length.	REC030
get_kernel_syms	Retrieves exported kernel and module symbols.	REC110
Getcwd	Get current working directory.	REC130
Getdents	Gets directory entries.	REC130
getdents64	Gets directory entries.	REC130
Getegid	Gets group identity.	REC120
getegid16	Gets group identity.	REC120
Geteuid	Returns the real user ID of the current process.	REC120
geteuid16	Returns the real user ID of the current process.	REC120
Getgid	Returns the real group ID of the current process.	REC120
getgid16	Returns the real group ID of the current process.	REC120
Getgroups	Get list of supplementary group Ids.	REC120
Getgroups16	Get list of supplementary group Ids.	REC120
Getitimer	Gets value of an interval timer.	REC120
Getpgid	Get process group ID of the process specified by pid to pgid.	REC120
getpgrp	Get process group.	REC120
getpid	Get process identification.	REC120
getppid	Gets process identification.	REC120
getpriority	Gets program scheduling priority.	REC120
getresgid	Gets real, effective and saved group ID.	REC120
getresgid16	Gets real, effective and saved group ID.	REC120
getresuid	Gets real, effective and saved user ID.	REC120
getresuid16	Gets real, effective and saved user ID.	REC120
getrlimit	Gets resource limits and usage.	REC120
getrusage	Gets resource limits and usage.	REC120
getsid	Gets session ID.	REC120
gettimeofday	Gets time.	REC120
getuid	Returns the real user ID of the current process.	REC120
getuid16	Returns the real user ID of the current process.	REC120
init_module	Initializes a loadable module entry.	REC110

System Call	Description	Requirement Reference
Ioctl	Control device.	REC090
ioperm	Sets port input/output permissions.	REC040
Iopl	Changes I/O privilege level.	REC040
Ipc	System V IPC system calls.	N/A
Kill	Terminates a process.	REC020
lchown	Changes ownership of a file.	REC040
lchown16		REC040
link	Makes a new name for a file.	REC030
llseek	Reposition read/write file offset.	REC030
lseek	Repositions read/write file offset.	REC030
lstat	Gets file status.	REC120
lstat64	Gets file status.	REC120
madvise		N/A
mincore		N/A
mkdir	Makes directories.	REC130
mknod	Makes block or character special files.	REC030
mlock	Disables paging for some parts of memory.	REC080/REC140
mlockall	Disables paging for calling process.	REC080/REC140
mmap	Map or unmap files or devices into memory.	REC080
mmap2	Map or unmap files or devices into memory.	REC080
modify_ldt	Gets or sets ldt.	N/A
mount	Mounts a file system.	REC030/REC130
mprotect	Controls allowable accesses to a region of memory.	REC080
mremap	Re-map a virtual memory address.	REC080
msync	Synchronizes a file with a memory map.	REC030/REC080
munlock	Reenables paging for some parts of memory.	REC080/REC140
munlockall	Reenables paging for calling process.	REC080/REC140
munmap	Map or unmap files or devices into memory.	REC080
nanosleep	Pauses execution for a specified time.	N/A
newfstat		N/A
newlstat		N/A
newstat		N/A
newuname		N/A
nfsservctl	Syscall interface to kernel nfs daemon.	N/A
nice	Runs a program with a modified scheduling priority.	REC140
Old_getrlimit	Obsolete system call.	N/A

System Call	Description	Requirement Reference
oldumount	Obsolete system call.q	N/A
olduname	Obsolete system call.	N/A
open	Opens and possibly creates a file or device.	REC030
pause	Waits for signal.	N/A
personality	Sets the process execution domain.	REC140
pipe	Creates a pipe.	REC060
pivot_root	Changes the root file system.	REC030/REC140
poll	Waits for some event on a file descriptor.	REC070
prctl	Operations on a process.	N/A
pread	Reads from or writes to a file descriptor at a given offset.	REC030
ptrace	Process trace.	REC120
pwrite		N/A
query_module	Query the kernel for various bits pertaining to modules.	REC120
quotactl	Manipulates disk quotas.	REC140
read	Read from a file descriptor.	REC030/REC070
readdir	Reads directory entry.	REC130
readlink	Prints contents of symbolic link.	REC030
readv	Reads or writes a vector.	N/A
reboot	Reboots the system.	REC140
rename	Renames files.	REC030
rmdir	Removes an empty directories.	REC140
rt_sigaction	See sigaction.	N/A
rt_sigpending	See sigpending.	N/A
rt_sigprocmask	See sigprocmask.	N/A
rt_sigqueueinfo		N/A
rt_sigreturn	See sigreturn	N/A
rt_sigsuspend	See sigsuspend	N/A
rt_sigtimedwait		N/A
sched_get_priority_max	Gets static priority range.	REC120
sched_get_priority_min	Gets static priority range.	REC120
sched_getparam	Gets scheduling parameters.	REC120
sched_getscheduler	Gets scheduling algorithm/parameters.	REC120
sched_rr_get_interval	Gets the SCHED_RR interval for the named process.	REC120
sched_setparam	Sets scheduling parameters.	REC140
sched_setscheduler	Sets scheduling algorithm/parameters.	REC140
sched_yield	Yields the processor.	N/A

System Call	Description	Requirement Reference
select	Synchronous I/O multiplexing.	N/A
sendfile	Transfers data between file descriptors	REC060/REC070
setdomainname	Sets domain name.	REC070
setfsgid	Sets group identity used for file system checks.	REC140
setfsgid16	Sets group identity used for file system checks.	REC140
setfsuid	Sets user identity used for file system checks.	REC140
setfsuid16	Sets user identity used for file system checks.	REC140
setgid	Sets group identity.	REC140
setgid16	Sets group identity.	REC140
setgroups	Sets list of supplementary group IDs.	REC140
setgroups16	Sets list of supplementary group IDs.	REC140
sethostname	Sets host name.	REC070
setitimer	Sets value of an interval timer.	N/A
setpgid	Sets process group.	REC040/REC140
setpriority	Sets program scheduling priority.	REC140
setregid	Set real and/or effective group ID.	REC040/REC140
setregid16	Set real and/or effective group ID.	REC040/REC140
setresgid	Set real, effective and saved user or group ID.	REC040/REC140
setresgid16	Set real, effective and saved user or group ID.	REC040/REC140
setresuid	Sets real, effective and saved user or group ID.	REC040/REC140
setresuid16	Sets real, effective and saved user or group ID.	REC040/REC140
setreuid	Sets real and/or effective user ID.	REC040/REC140
setreuid16	Sets real and/or effective user ID.	REC040/REC140
setrlimit	Sets resource limits and usage.	REC140
setsid	Runs a program in a new session.	REC020
settimeofday	Sets time.	REC140
setuid	Sets user identity.	REC140
setuid16	Sets user identity.	REC140
sgetmask	ANSI C signal handling.	N/A
sigaction	POSIX signal handling functions.	N/A
sigaltstack	Gets or sets alternate signal stack content.	N/A
signal	ANSI C signal handling.	N/A
sigpending	POSIX signal handling functions.	N/A
sigprocmask	POSIX signal handling functions.	N/A
sigreturn	Returns from signal handler and cleanups stack frame.	N/A
sigsuspend	POSIX signal handling functions.	N/A

System Call	Description	Requirement Reference
socketcall	Socket system calls.	REC070
ssetmask	ANSI C signal handling.	N/A
stat	Displays file or filesystem status.	REC120
stat64	Displays file or filesystem status.	REC120
statfs	Gets file system statistics.	REC120
stime	Sets time.	REC140
swapoff	Disables devices and files for paging and swapping.	REC140
swapon	Enables devices and files for paging and swapping.	REC140
symlink	Make a new name for a file.	REC030
sync	Flushes filesystem buffers.	REC030
sysctl	Configures kernel parameters at runtime.	REC140
sysfs	Gets file system type information.	REC120
sysinfo	Returns information on overall system statistics.	REC120
syslog	Reads and/or clears kernel message ring buffer.	REC110
time	Times a simple command or gives resource usage.	REC120
times	Writes the time used by a current process and its children into a structure.	N/A
truncate	Truncates a file to a specified length.	REC030
truncate64	Truncates a file to a specified length.	REC030
umask	Sets the mask for access rights to a file.	REC030
umount	Unmounts file systems.	REC030
uname	Prints system information.	REC120
unlink	Delete a name and possibly the file it refers to.	REC030
uselib	Selects shared library.	N/A
ustat	Gets file system statistics.	REC120
utime	Changes access and/or modification times of an inode.	REC030/REC140
vfork	Works like fork except for sharing of data segments between the processes.	REC020
vhangup	Virtually hangup the current tty.	N/A
Vm86	Enter virtual 8086 mode.	REC140
Vm86old	See vm86	REC140
Wait4	Waits for process termination, BSD style.	REC020
waitpid	Waits for process termination.	REC020
Write	Sends a message to another user.	N/A
writev	Reads or write a vector.	N/A

Appendix B – Secure Auditing for Linux (SAL): Audit Data Structures

This document describes the content and structure of data transferred between software components on the audit client and from audit clients to the audit server.

Client Data Structures

There are several software components on the client that exchange data. On systems with an instrumented kernel, audited event data is collected by the instrumented kernel and passed to a user-space audit daemon. This audit daemon passes the data on to the audit client process. The audit client process collects data from the audit daemon, and potentially from other sources, passing this data to the audit server. The data formats used on the client systems are described in this section.

Kernel Audit Event

In an instrumented kernel each audited system call generates one audit event. The data for the event is collected in a structure, which is described in Table 1. Multiple audit events are collected into buffers, which are copied out to the audit daemon when the appropriate system call is made. The number of events that may be copied out to the daemon can be adjusted by modifying and compiling several constants in the kernel audit code. All data is converted to network byte order in the kernel.

Table 1 – Kernel Audit Event Structure

Sequence	Size	Type	Name	Comments
1	32 bits	unsigned int	serial	A one-up counter representing the sequence number for the system call.
2	32 bits	unsigned int	ts_sec	Timestamp: when the syscall occurred represented by the number of seconds since January 1, 1970.
3	32 bits	unsigned int	ts_micro	Timestamp (complements ts_sec): the number of microseconds between each second.
4	32 bits	unsigned int	syscall	A numerical representation of the system call. These numbers can be linked to a system call name in `arch/i386/kernel/entry.S`.
5	32 bits	unsigned int	status	The return status of the system call.
6	32 bits	unsigned int	pid	The Process ID: defined in `include/linux/types.h`. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
7	32 bits	unsigned int	uid	The Real User ID: defined in `include/linux/types.h`. This information is obtained from the Linux

				kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
8	32 bits	unsigned int	euid	The Effective User ID: defined in `include/linux/types.h`. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.
9	32 bits	unsigned int	device	A numerical representation of the tty. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`. If no tty is defined the tty_struct of the current task structure is null.
10	128 bits	16 unsigned chars	comm	The name of the program that executed the process that called the system call. This information is obtained from the Linux kernel's current task structure (task_struct) defined in `include/linux/sched.h`.

Audit Daemon Data Structure

The audit daemon collects a buffer of audit event data from the kernel through the audit system call. Each call passes a memory pointer to the kernel which the kernel uses to copy out one buffer of data. The audit daemon collects several of these buffers in memory, then writes them to a temporary file for collection by the audit client process. These temporary audit files are the concatenation of multiple kernel audit buffers. The audit daemon performs no data manipulation or transformation on the buffers.

The daemon is unaware of the kernel audit event structure since it does not modify or interpret them. Each call to the kernel returns the number of bytes copied out, which allows the daemon to concatenate the buffers without padding. The number of buffers collected before the temporary file is created can be adjusted by changing a defined variable and recompiling the audit daemon.

Table 2 – Audit Daemon Data Structure

Sequence	Size	Type	Name	Comments
1	variable	const char	n/a	The first kernel audit buffer.
2	variable	const char	n/a	The second kernel audit buffer.
…	…	…	…	…
n	variable	const char	n/a	The last kernel audit buffer.

Client – Server Data Structures

All data transferred between SAL's Client and Server is encrypted. For the purpose of this document the encryption method used will not be considered when looking at the data formats (consider the encrypted link to be the wire the data travels over.) SAL supports multiple data

transfer formats. The primary formats are: (1) string based and (2) data blob based (a specific example of this format is being used to transmit the kernel audit events from the SAL client). String based formats provide the capability to transfer standard text messages to the server, similar to the existing syslog program. This format can also be extended to support the transmission of XML formatted strings. The concept of the data blob, which is heavily used in data base design, allows for the transmission of arbitrary sized data to the SAL data repository. The SAL Kernel Audit Event is a specific implementation of the blob data format. While one of the primary roles of SAL is to monitor the Linux kernel, a client does not necessarily have to be instrumented. In addition, it is envisioned that the data formats should be able to support a large number of possible applications that require transmission to a secure data repository.

Format 1.0 Audit Package Structure

When a client computer is started and auditing is initiated an SSL connection is attempted with the server. Assuming the connection is granted (which it may not be) the client will transmit configuration/meta data to the server as the first message. All subsequent messages will be the contents of a single temporary file created by the audit daemon. Initially, this initial message consists only of a Client Version Message, which contains a format number and the size of each audit event structure to follow. Future versions may include a copy of the /etc/passwd and /etc/group files, and/or other client configuration data. All of the following messages consist only of audit events. All data is sent in network byte order.

Table 3 - Client Version Message

Sequence	Size	Type	Name	Comments
1	32 bits	unsigned int	size	The number of bytes which follow the size in this initial message.
2	32 bits	unsigned int	major	The major format number of this initial client message, currently the integer value “1”.
3	32 bits	unsigned int	minor	The minor format number of this initial client message, currently the integer value “0”.
4	32 bits	unsigned int	event_size	The number of bytes in each audit event, currently the integer value “52”.

Once the initial message is sent, the client process will block and wait for temporary files to appear in the spool directory. As each new file appears, the client will collect the old file and immediately send the entire file to the server using the established SSL connection. Each file will be transmitted as a single binary object. The client does not process or interpret the data being sent. This data consists of back-to-back audit event messages, of the size specified in the format message, with no additional data or padding interspersed. When the data is received by the server’s session, special archive meta-data is appended before it is permanently and securely stored to disk. At no time during processing by the server is the data manipulated, converted, or otherwise changed.

Format 2.0 Audit Package Structure

The communication connection between the client and the server is encrypted and TCP based. The server waits for connections to be established by clients on the default port of 885. On initialization of the client, the audit client process attempts to connect with the server which involves negotiating an SSL connection. When a connection is established, the server authenticates the connection based on its configuration and creates a session that is responsible for the reception of client audit data. The session is a virtual function that allows for different types of audit data to be received and processed. The selection of a specific class is based on the policy defined for that connection and the type of audit data that is expected to be transmitted. Sessions can be defined to support only one specific type of message and report an error when a different type is received; or to accept any type of audit event that may cross the connection. Each audit message that comes across the connection has a message header. The header is used to define the type of audit message, its size in bytes, a priority, and a facility. The payload of the audit message will follow the header. The header is defined below:.

Table 4 - Audit Event Structure

Sequence	Size	Type	Name	Comments
1	32 bits	unsigned int	size	The number of bytes which follow the size in this initial message.
2	32 bits	unsigned int	major	The major format number of this initial client message, currently the integer value “2”.
3	32 bits	unsigned int	minor	The minor format number of this initial client message, currently the integer value “0”.
4	16 bits	unsigned int	ae_t	Audit event type. 0 = blob 1 = string
5	16 bits	unsigned int	ae_sub	Audit Event subtype 0 = Kernel Audit Event 1 = Client Version Event 2 = syslog 3..N = not used
6	32 bits	unsigned int	size	Number of bytes used to store audit event including header and payload.
7	16 bits	unsigned short	priority	Alert priority 0 = no priority. 1..N = The higher the number the higher the priority. (SAL server does not respond to the priority)
8	16 bits	unsigned short	facility	Indicates where the message came from 0 = Kernel Audit Event (SAL server does not respond to the facility)