

AUDITING AND EVENT
CORRELATION

By

Trevor Norvill

A thesis submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Engineering(Computer
System) Hons

The University Of Queensland

2001

Approved by _____
Chairperson of Supervisory Committee

Program Authorized
to Offer Degree _____

Date _____

T.Norvill
11/38 Durham St,
St.Lucia, Qld. 4067.
October 17, 2001.

Attn.: Professor Simmons
The Dean
Faculty of Engineering
The University of Queensland
St. Lucia, Qld, 4067.

Dear Sir,

In partial fulfilment of the requirements for the degree of the Bachelor of Engineering(Computer System) with honours, I submit for your consideration this thesis entitled:

Auditing and Event Correlation

Your Faithfully,

Trevor Scott Norvill

AUDITING AND EVENT

CORRELATION

Abstract

By Trevor Norvill

Chairperson of the Supervisory Committee: Dr Mark Shultz
Department of Information Technology and Electrical Engineering

This Thesis is focused on Auditing and Event Correlation. A system is developed to help system administrators analyse system attacks in response to alerts from Intrusion Detection Systems or analyse audit data post hoc. The system is aimed at identifying information leakage between files and processes. In addition, the system also aims to identify illegal user command activity and system attacks.

The system is based on Linux and correlates Kernel audit data with shell history data. In addition, File accesses are correlated into a per session audit log. A query engine is written to analyse multiple audit logs. This system is a prototype and further work will evolve from it.

Table of Contents

TABLE OF CONTENTS	I
LIST OF FIGURES	III
ACKNOWLEDGMENTS	IV
INTRODUCTION.....	1
1.1 THE SYSTEM.....	2
<i>Consolidation Tool</i>	2
<i>Analysis Tool</i>	2
1.2 THE VISION.....	3
1.3 INTRODUCTORY CONCEPTS	4
1.4 OUTLINE OF REPORT.....	5
BACKGROUND THEORY.....	7
2.1 INTRUSION DETECTION AND AUDITING.....	7
2.2 THE INTEGRATION OF AUDIT DATA	8
2.3 AUDITING IN UNIX SYSTEMS.....	8
2.4 AUDITING IN LINUX SYSTEMS	10
2.5 STANDARD AUDIT TRAIL RECORDS	10
2.6 CONCLUSION ABOUT AUDITING AND IDS.....	11
REVIEW OF PRODUCTS AND TECHNOLOGY	12
3.1 RELATED PRODUCTS	12
3.2 USER AUDIT DATA	16
3.3 COMMERCIAL AUDITING TOOLS	18
3.4 INTRUSION DETECTION SYSTEMS.....	19
3.5 OTHER ORGANISATIONS.....	19
MOTIVATION FOR DESIGN	20
4.1 MOTIVATION FOR DESIGN	20
4.2 DESIGN GOALS	20
4.3 PROPOSED OPERATION	21
4.4 PROCESS OF DEVELOPMENT	22
PRODUCT SPECIFICATION	25
5.1 MAJOR DESIGN GOALS	25
5.2 MAJOR SOFTWARE COMPONENTS	25
5.3 DATA FOR CONSOLIDATION TOOL.....	27
<i>DATA USED BY THE CONSOLIDATION TOOL</i>	27
<i>ADVANTAGES OF CORRELATING THIS DATA</i>	28
5.4 SPECIFICATION OF CONSOLIDATION TOOL.....	28

Auditing and Event Correlation

<i>INPUTS TO CONSOLIDATION TOOL</i>	28
<i>OUTPUTS TO CONSOLIDATION TOOL</i>	28
<i>TASKS TO BE PERFORMED BY CONSOLIDATION TOOL</i>	28
5.5 SPECIFICATION OF THE QUERY TOOL.....	30
<i>CLASS STRUCTURE</i>	30
<i>METHODS OFFERED</i>	31
5.6 MILESTONES THE PROJECT IS TO ACHIEVE.....	32
IMPLEMENTATION	34
6.1 OBTAINING SHELL HISTORY.....	34
6.2 CONSOLIDATION TOOL FLOW CHART	35
6.3 QUERY ANALYSIS TOOL	38
6.4 OTHER COMPONENTS.....	42
RESULTS.....	43
7.1 UNIT TESTING.....	43
7.2 SYSTEM TESTING.....	44
7.3 FUNCTIONALITY ANALYSIS	45
7.4 PERFORMANCE TESTS	45
7.5 LIMITATIONS OF THE CURRENT MODEL	47
7.6 IMPLEMENTATION ISSUES	47
FUTURE DEVELOPMENTS	49
8.1 IMPROVED COMPONENTS	49
8.2 OBJECT ORIENTATED MODEL FOR CONSOLIDATION TOOL	50
CONCLUSION.....	52
PROJECT OUTCOMES	52
APENDIX 1 CODE LISTING	55

List of figures

Figure 1.1. Consolidation tool.....2
Figure 1.2. Query Analyse tool.3
Figure 3.1. The BSM system.....13
Figure 3.2: BSM RECORD OUTPUT.13
Figure 4.2. Query data organisation.....21
Figure 4.3. Original concept.22
Figure 5.1. The system components relationship.....26
Figure 5.2. Validating commands example.....29
Figure 5.3. Class Structure for query tool.....31
Figure 6.1. Obtaining Shell history.....34
Figure 6.2.....35
Figure 6.3.....37

Acknowledgments

The author wishes to thank Associate Professor George Mohay for academic guidance and direction on this work. In addition, I would like to thank Nathan Carey for also providing valuable advice in the security area. Finally, I would like to thank the Secure Networks Laboratory at QUT for the use of hardware during this project.

Chapter 1

INTRODUCTION

This chapter describes the research area that this Thesis is aimed at, the product that was developed as a result of this research and finally a discussion of the remaining chapters of this thesis is outlined.

Over the last two decades, computer crime has steadily increased with the development of the Internet and Electronic Business. Intrusion Detection and Computer Forensics doesn't seek to prevent these crimes and misuse. Rather, they seek to analyse the aftermath and provide insight on how to prevent future misuse.

The research aim of this project was to examine some commonly available types of host based logging on computer systems and determine how these logs can be used to provide lightweight Intrusion Detection and Computer Forensic Systems. The integration of such audit logs is aimed at enhancing the ability to track system attacks and misuse.

The final system was named Event Correlation Tool tools or E.C.T. ECT's design and development was directed towards analysing data post hoc to track Intrusions and misuse attacks on a computer system. The types of questions this system is aimed at answering are queries relating to principals, actions and objects:

- What did a user do on the system and when did they do it?
- How successful was the user in their attempts?
- What files were accessed and by whom were they accessed?
- Who else accessed the files?
- When were those files accessed in relation to other files and actions?
- What intrusion detection information was logged relative to the above?
- What system information was logged relative to the above?

Auditing and Event Correlation

This project is the start of an ongoing research effort that will continue next year. This thesis will outline background theory, motivation, design and prototype development of the C.A.T system. It will then go on to outline refinements and extensions to this design and recommend redesign for the next phase of the project. The goal is to design a prototype for evaluation so as to facilitate a future production version.

1.1 THE SYSTEM

The CAT system consists of two components, the consolidation tool and the analysis tool. Each of these is outlined below

Consolidation Tool

Uses three sources of audit data from a Linux computer system and consolidates them into one integrated log. Kernel level audit data is combined with application level data from a program that logs user file access history. A method of obtaining user shell history files has been derived and this user level data is used to obtain command line parameters and session data where possible. This is illustrated in figure 1.1 below. The result is an audit log containing correlated information on one user for one session.

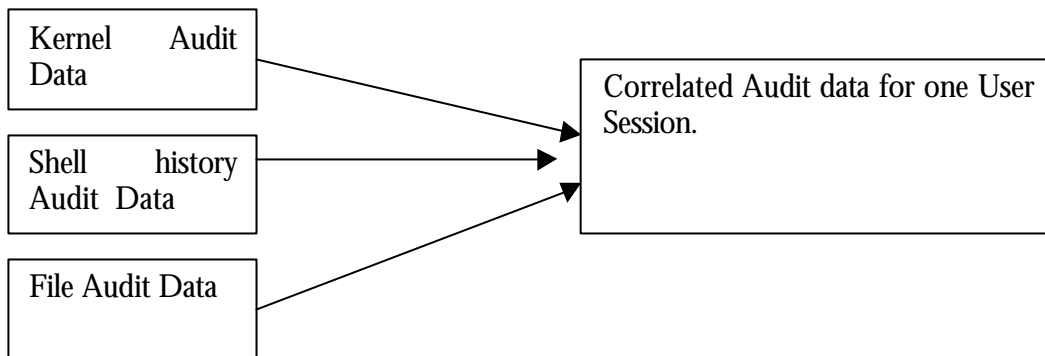


Figure 1.1. Consolidation tool.

Analysis Tool

Once the correlated data is obtained a further tool then queries this integrated log allowing analysis of user activity. Each correlated log that is created by the Consolidation tool represents one session for one user. The analysis tool considers multiple session/user correlated audit logs. The current capabilities of this tool are as follows

Auditing and Event Correlation

1. Find any correlated records that involve File X within a specified time.
2. Find any correlated records that involve user with UID Y
3. Find all records between times T1 and T2.
4. Find any record that involves process ID Z.
5. Find the intersection of any of the above.

These functions will allow a system administrator to track suspicious user activity based on alerts from intrusion detection systems. This project does not establish this link in detail but the usefulness is made clear. It will facilitate the analysis of information leakage on a system. Figure 1.2 below shows how the analyse tool uses the correlated log to provide the user with a front query engine

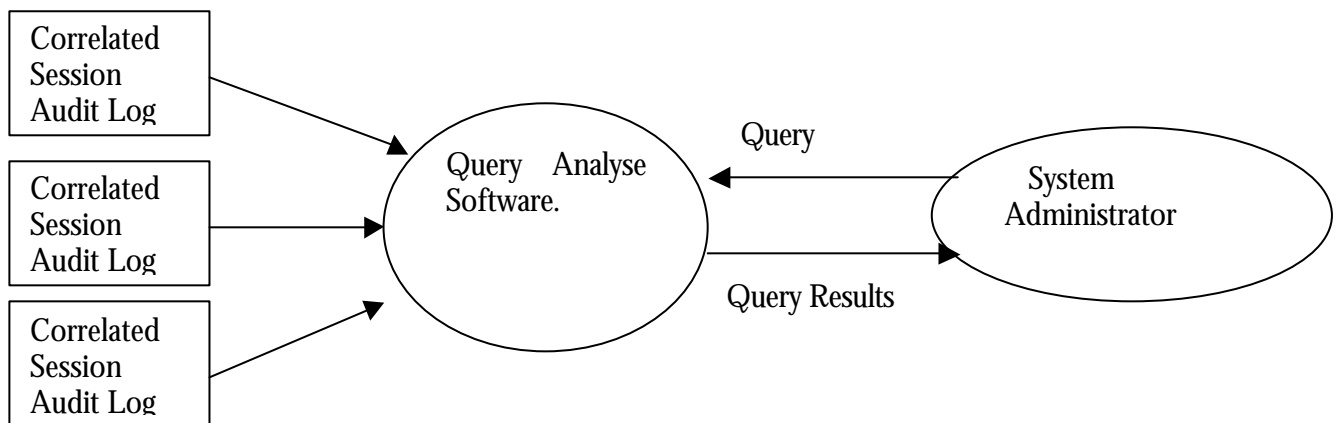


Figure 1.2. Query Analyse tool.

1.2 THE VISION

The vision of this system is aimed at a two-year scope. If taken to its logical conclusion, this system would do the following.

- Integrate many forms of auditing on a system. This would provide a system that a system administrator could query in response to concerns raised by Intrusion detection systems or post hoc Forensic Analysis.
- This system would be available on many different machines. A cross platform solution would provide the needed flexibility that modern computer networks require. A central console would need to moderate all the information provided by all systems on the network and

provide a comprehensive database facilitating a detailed or abstract view of the networks state and history.

1.3 INTRODUCTORY CONCEPTS

SECURITY A term relating to the preservation of confidentiality, integrity and availability. These are the key attributes that define security. Confidentiality is the prevention of unauthorized disclosure of information. Integrity is the prevention of unauthorized modification of information. Availability is the prevention of unauthorized withholding of information [Stef00].

AUDITING Auditing is the process of recording selected events in order to provide a basis for analysis and assurance of system integrity. According to [Price97] Auditing can be defined as “recording and analysing system activity for detection of security problems”.

C2 Level Auditing System security is defined by Trusted Computer System Evaluation Criteria [DOD85]. This is otherwise known as the orange book, put out by the U.S National Computer Security Centre (NCSC) in 1985. It defines criteria that a system must pass to get a certain level of “trust”. The level of security of interest to this thesis is C2.

“Systems in this class (C2) enforce a more finely grained discretionary access control than (C1) systems, making users individually accountable for their actions through login procedures, auditing of security-relevant events, and resource isolation.” [DOD85].

BSM Basic Security Module. The Sun Microsystems operating system Solaris includes an extension called the Basic Security Module. This is aimed at providing enhanced security for Solaris. The BSM is designed to provide a C2 level of security as defined by the Trusted Computer System Evaluation Criteria [DOD85].

INTRUSION DETECTION SYSTEMS Intrusion detection systems (IDS) detect suspicious events and informs the system manager [Gol99]. This is a system, which by whatever means necessary discovers the existence of an attack upon a system [CAR00]

Auditing and Event Correlation

Interestingly, the main focus of these systems is often in the internal misuse of the computer system.

MANDATORY ACCESS CONTROL A policy that tells the system who is allowed to have access to what resource [Gol99].

DISCRETIONARY ACCESS CONTROL A policy in which access control is at the discretion of the user [Gol99].

COVERT CHANNEL is an information flow that is in contravention of stated security policy and not controlled by a security mechanism [Gol99]. A covert channel involves the flow of information from high security levels to low security levels.

RESTRICTED SHELL A shell that restricts the action a user may perform. This is usually done by not allowing a user to cd out of the . directory and not allowing them to change the PATH environment variable.

1.4 OUTLINE OF REPORT

Chapter 1 provides an overview and an outline of the thesis content.

Chapter 2 proceeds to cover background theory including the relationship between Audit information and Intrusion Detection, Auditing on Unix systems and provides the necessary orientation to consider the product.

Chapter 3 reviews the products and technology, commercial auditing and IDS tools and sources of audit data.

Chapter 4 outlines motivation for the design.

Chapter 5 outlines the product specification. This details how the specification was created and transformed into a design.

Chapter 6 explains the software implementation of this product.

Chapter 7 specifies the results of the thesis.

Chapter 8 outlines future developments.

Auditing and Event Correlation

Chapter 9 contains a conclusion on the effectiveness of this work.

Chapter 2

BACKGROUND THEORY

This chapter outlines background theory relevant to the specification and design of the product. The relationship between Intrusion detection and auditing is examined and the various properties of audit data exposed. An analysis of auditing in Unix and specifically Linux is undertaken and the efficiency of correlating various types of logs is examined.

2.1 INTRUSION DETECTION AND AUDITING

Intrusion Detection Systems depend on the amount and quality of audit data available to them [Price97]. There are two basic types of IDS available.

1. Misuse Detection. An IDS that looks for well-defined attacks against known system vulnerabilities is known as a Misuse Detection IDS [CSJ99].
2. Anomaly Detection. IDS that look for deviations from normal system usage [CSJ99].

There are two further key distinctions that distinguish IDS:

1. Network IDS. This approach examines packet headers, packet data and other network information to determine if an attack is occurring.
2. Host Based IDS. This approach involves an Intrusion Detection System in which data from logs on the local machine is analysed in order to detect attacks. These types of systems also look at abnormal resource usage, compromised processes and unauthorized user activity.

Both types of IDS detect possible attacks and alert the system administrator to the possibility of an attack. Once the System Administrator has established that there is a distinct possibility of an attack, he may wish to take some action. However, IDS often do not give enough information to analyse the attack in detail. The System administrators are confronted with an array of separate system logs, kernel level audit data if available and application level audit data. Putting all this information together to provide a big picture and forensic analysis can be difficult at best.

2.2 THE INTEGRATION OF AUDIT DATA

Audit data exists on computer systems in many places and is produced by many pieces of software. Audit data is classified into three categories.

1. System call audit data. Any audit data produced directly by the kernel is called system call level audit data. This information is logged by the kernel and is generally not controllable by the user.
2. Application level audit data. Any data produced by application level programs. These programs run on top of the operating system and are controlled by the user.
3. Network audit data. Network audit data can be classed as application level data but the distinction is made here to distinguish it from host based application level data.

Price [Price97] stated that Application level data has certain advantages over kernel level data. From her work, four main observations of Kernel and Application level data can be stated.

1. Kernel auditing is very difficult to circumvent but usually creates large quantities of data and the data is usually hard to understand due to its low level nature.
2. Application auditing produces audit data that is usually more compact and exact however it is easier to circumvent by operating at a lower level or tunnelling. That is, a user may be able to perform his activities below the application process performing the auditing function and thus prevent it capturing his activity.
3. Kernel or system call audit data does not provide insight into who initiated activities. It often provides little insight as to whether the system performed an activity on behalf of the user or the user himself or herself initiated the action resulting in the audit information.
4. Users may give false information to application level auditors by command and program aliasing. This point is of particular importance to this thesis and will be elaborated on later.

2.3 AUDITING IN UNIX SYSTEMS

Trusted Computer System Evaluation Criteria

Auditing and Event Correlation

Trust is an important and fundamental concept in computer security. A widely accepted means to measure for this “trust” is the U.S Department of Defence (DoD) Trusted Computer Security Evaluation Criteria standard 5200.28-STD [DOD85]. This is otherwise known as the orange book. It defines criteria that a system must meet to achieve a certain level of “trust”. Trust levels range from A to D with A being the most secure. These levels are defined as follows:

- Level D. Minimal Protection
- Level C. Discretionary Protection
- Level B. Mandatory Protection
- Level A. Verified Protection [Dan00]

The level of security we are interested in is C2, the code for controlled access protection. C2 system must:

- Provide system level audit trail
- Audit the use of identification and authentication mechanisms
- Audit file access
- Audit file/object deletion
- Audit administrative actions.

The element of this level of security relevant to this thesis is the auditing feature. The criteria specify the following for each record event:

- Date
- Time of the event
- User
- Type of event
- Success or failure of the event.
- For identification/authentication of events the origin of the request must be present.

Most Unix computer systems log at least some security related information. These are all usually located in the /usr/adm directory. The logs are

- /usr/adm/lastlog Records information on the last time a user logged in.

Auditing and Event Correlation

- /usr/adm/acct Records all executed commands.
- /var/log/utmp Information used by the `who` Unix command
- /var/log/wtmp Records user log in/log out information.

The Unix Syslog facility and its associated syslogd daemon are used to log information. This is a standard part of Unix security facilities.

2.4 AUDITING IN LINUX SYSTEMS

Linux is an open source operating system and is relatively young. It supports the standard Unix style syslog auditing facility. Log files of note are.

- Messages. Contains most of the useful system messages.
- UTMP. Same as general Unix.
- WTMP. Same as general Unix.
- Secure.
- Cron.

2.5 STANDARD AUDIT TRAIL RECORDS

Price [Price97] identified that a standard audit trails is necessary to overcome incompatibility issues in misuse detection. In addition audit record content also needs to be in a standard format. This is mainly due to the distributed nature of modern computer systems networks and the variety of platforms that comprise them. This is relevant to this thesis because misuse detection has common issues to the current work. A post hoc analysis or forensic query system needs the same standardisation of audit records and content to help in interoperability. They have been many attempts at standardized record formats, however, a standardized audit trail format is still a long way off. However, there has been related work with IDS that has attempted to develop and define a common semantics for events and event data.

2.6 CONCLUSION ABOUT AUDITING AND IDS

The above theory highlights that Intrusion Detection Systems alert system administrators to the presence of attacks on their networks. It also contained a discussion of the different types of audit data and their characteristics. If user level data can be correlated with audit data from the kernel level, the advantage would be that it would allow a System administrator to make better use of this information thus improving the security and reliability of this information. The following chapter will look at current products, both commercial and free, that log various sorts of audit data on the Linux and Unix operating systems.

CHAPTER 3

REVIEW OF PRODUCTS AND TECHNOLOGY

This chapter will examine various Auditing and Intrusion Detection products with the aim to understanding the rationale for developing the final product which is the focus of this thesis. Both commercial and open source application and products are examined to give an overall understanding of the field. Also, other sources of audit data are examined including user shell history. A method of obtaining user shell history is developed.

3.1 RELATED PRODUCTS

The following is a description of some audit and security tools that have been developed for Linux. They have been selected for their relevance to the project.

SOLARIS BASIC SECURITY MODULE

The Solaris SHIELD Basic Security Module is an extension to Sun micro System's Solaris Operating system. The extension provides the security features defined as C2 in the trusted Computer System evaluation criteria other than discretionary-access control, identification and authentication features which are already provided by Solaris [Sun95]. The Basic Security Module (BSM) is aimed at providing enhanced auditing abilities on Solaris. It also provides a device allocation mechanism. This feature provides a powerful auditing tool that can serve as the data source to a variety of host and hybrid-based Intrusion Detection Systems. The Solaris BSM record provides at least the following information:

- Header. The Header is in the format
Header, records length in bytes, audit record version number, event description, event description modifier, time and date.
- Subject. The Subject is in the format
Subject, user audits ID, effective user ID, effective group ID, real user ID, real group ID, process ID, session ID, and terminal ID.
- Return Code.

Auditing and Event Correlation

LINUX BASIC SECURITY MODULE

The Linux Basic Security Module or Linux BSM is a project run by the University of California at Davis. The aim is to create a comprehensive auditing package that is fully compliant with the U.S government's Trusted System Evaluation C2 criteria standard. The Linux BSM monitors and records Kernel level events. This is done by a kernel patch. It is directly applied to the Linux kernel. This patch is currently only available for the 2.2.17 kernel. There is an audit daemon called "auditd" that is run to talk to the Kernel patch and retrieve the BSM records. A further piece of software called praudit transforms the data returned by auditd into text that is readable by the user. This is illustrated in figure 3.1. The Linux BSM is currently termed alpha software. This means it is in a stable but incomplete state. The Linux BSM was designed to mirror the Solaris BSM but lacks some of the functionality found in the Solaris version.

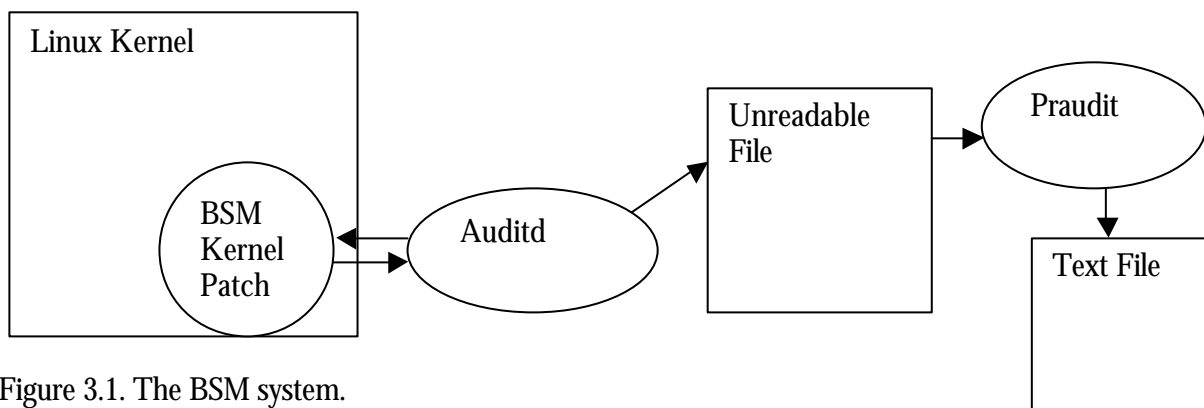


Figure 3.1. The BSM system.

An example of the Linux BSM is shown in figure 3.2

```
Read 29 bytes
Header.event_class=16 header.event_id=23 header.event_size=284
Read 284 bytes
Header, 284, execve(), Thu Jul 5 16:01:48 2001 +418613
Path, /bin/vi,
Subject,0,0,0,0,1227
Return, -13
```

Figure 3.2: BSM RECORD OUTPUT.

Auditing and Event Correlation

ELIOTT

Eliott is a program for discovering insecure file creation/deletion using the dnotify facility of the Linux Kernel. The use of the dnotify facility restricts its use to the 2.4.x Kernel. Eliott is designed to look at one particular directory and its design rationale is based on observing temporary file creation and deletion that may not exist for very long. It is currently designed to log all file creation, deletion and writes. A sample of the Eliott Output is shown in figure 3.4

```
[17:53:19] - New file : [tempfile] (500,100) (s 644)
```

Figure 3.4. Sample output from Eliott.

Important features in the above are the time field, the file accessed and the UID and GID that are represented in the first round brackets. ELIOTT can be used to simulate hard-link exploits in order to find and report vulnerable applications [El01]. Eliott logs its information to standard output, but redirection can be used to log its output to a file. Eliott can be found at [El01].

THE ABACUS PROJECT

The Abacus Project by Psionic Software consists of a range of tools aimed at providing free auditing and Intrusion Detection tools to the community. Components of the Abacus project that will be discussed here are Logcheck, HostSentry and PortSentry. The abacus project can be found at [abac01]. The abacus project will not be used in this thesis but has sufficient relevance to auditing that its components will be discussed here.

LOGCHECK

The Logcheck program was created to help in the processing of UNIX log files and in this case Linux system log files generated by the various Abacus Project tools, system daemons etc. Logcheck also works very well at reporting on other common system security violations and abnormal events. Log check package consists of two components, an auxiliary component called logtail and logcheck itself. Logtail is used in conjunction with logcheck. Logtails job is to remember in a file Logcheck has analysed up to. Logcheck itself contains the functionality described above.

HOSTSENTRY

Auditing and Event Correlation

HostSentry monitors the login accounting files, utmp/wtmp, for unusual activity. The extracted information is stored in a database and the following activity is detected. It produces alerts based on the following observations:

- Detect first log in. This feature alerts an administrator to the event that a user logs in to the system for the first time.
- Detect unusual foreign domain login. If a login occurs from a domain that clearly has no business being on your computer system, it should be treated as suspicious and action should be taken. Host sentry can take the necessary action according to security policy.
- Detect rhost changes. HostSentry logs the event in which a user changes their rhost file. Such an access can have serious security implications.
- Detect history file truncation. HostSentry logs the event in which a user has tried to truncate his shell history file to 0. This is usually a clear sign that someone is trying to cover his or her tracks after attacking a system. This point is of particular relevance to this thesis.
- Detect odd directory names. HostSentry looks for unusual directory names in a users home directory. Hackers usually do this to help hide their activities.
- Detect multiple Logins. HostSentry looks for multiple logins. This is a sure sign that an attack may be occurring.

PORTSENTRY

PortSentry is a port scanner designed to detect stealth port scans PortSentry will detect SYN/half-open, FIN, NULL, X-MAS and oddball packet stealth scans. Four stealth scan operation modes are available for you to choose from [AB01]. Port sentry can also react to scans in real time by blocking offenders. Port sentry consists of an internal state engine that allows PortSentry to remember who has connected. This helps reduce false alerts. Port scans are an indication of an imminent attack and are quite useful.

Auditing and Event Correlation

NABOU AND TRIPWIRE

Nabou is a free product and is classed as a system integrity monitor. It has a database of file properties and each night monitors the files for changes. It is also to look for new user accounts, new Cron jobs, SUID files and strange processes that should not exist [NA00]

Tripwire is an open source tool on Linux. It is a tool that checks to see what has changed on your system. The program monitors key attributes of files that should not change, including binary signature, size, expected change of size, etc. [TRIP01]

Neither Nabou nor Tripwire are used in this project, but any future developments should keep them in mind, especially Tripwire.

LOGSURFER

Logsurfer is a project aimed at producing a tool that monitors host based log files. It has a configuration file that specifies signatures that are used for pattern matching to detect known patterns of attacks that show up in system log file logged from either the syslog facility or from other information services such as FTP and WWW. It is used because of the extremely large amount of data that can appear in system log files. This can often get too much for system administrators. The systems niche is to provide an efficient and effective reduction of this quantity of data. Other advantages include the ability to email results of its analysis and its ability to analyse contexts (Multi-line records). Log check is not used in this thesis but future projects should keep it in mind.

3.2 USER AUDIT DATA

One of the objectives of this project was to investigate if user shell history files could be used in an auditing system. The advantages of this would be that command history often has options and command line parameters. These are usually lost in most auditing systems. This project looked at the bash shell in particular. The user history file in the bash shell is named `.bash_history`. The `.bash_history` file however is not a secure source of data for many reasons.

When a user opens a Linux bash shell, a script in their home directory called `.bashrc` is run. This file runs a file in the `/etc` directory called `bashrc`. This script sets up environment variables and performs

Auditing and Event Correlation

other initialisation functions. A variable that controls the operation of the shell is called an environment variable. These variables can be made read only, so that they can no longer be written to in a given session. The `bashrc` login scripts have been modified to make several of these environment variables read only. These environment variables are `HISTSIZE`, `HISTFILESIZE` and `HISTNAME`. The user cannot be allowed to modify these particular environment variables in this project as it truncates the size of the `.bash_history` file to zero or prevents shell history logging altogether. This means that no data will be logged into the history file. Once the users current session ends, the environment variables are again in a state where they can be written. However, they will always be set to read only when the user logs in or opens a new shell, therefore denying the user the privilege of modifying them.

The ownership permissions on the users history file must be root so that the user cannot remove the write file permission on the `.bash` history file. That is they cannot alter the permission on the history file so as to prevent logging to that particular history file. If they could achieve this, they would have succeeded in preventing the history file from recording the commands the user executed in that session.

One of the major problems with making the bash history file secure is that the user can delete it. The file permissions cannot be made read only because the user then does not have permission to write their history to it in the first place. Therefore, the `.bash_history` file must have user write permissions. This means that the user can delete the file if they desire. A mechanism had been put in place to secure this data so that a user cannot delete it. This can be done by noting that the history from a user session is not written to the history file until that user logs out. That is, the history from a users current session is kept in memory until the user logs out. At the logout point, the command history is written from memory to the `.bash_history` file in the user's home directory. The current method of securing this history is a Suid program written in c. It is called from the `bashrc` log in script and copies the `.bash_history` file from the users home directory to a secure place in a super user owned directory. If a user was to delete the `.bash_history` file just before logging out, they will not have deleted the commands they just completed even though they thought they might have, rather they will have deleted a stale version of it. When they next log to the system the `bashrc` login script will use the Suid

Auditing and Event Correlation

program to copy the history from the previous session to a secure place in the root owned directory. This is obviously out of access range for the user and is therefore secure.

This is obviously not a completely secure source of user command audit data. The user can still recompile a new shell or import their own to change the rules of the game. This could be prevented by the use of a restricted shell. However, it is up to the system administrator as to whether this level of assurance is required.

3.3 COMMERCIAL AUDITING TOOLS

This Thesis did not aim to use a commercial auditing and Intrusion Detection tool because of cost but they will be discussed here for completeness.

INGRES II

Ingress II is a product produced by Computer Associates that came out of research projects in the late 1970's. C. A. claims that Ingress II is a complete solution for eBusiness that enables access to a wide variety of enterprise data, remote and high availability replication. It is designed to run across a variety of platforms such as Windows, Unix and Linux[CA01]. What makes this system worth mentioning here is that it provides a C2 level of Auditing in the supported platforms. It was not considered for this Thesis because it is expensive enterprise software.

SAWMILL

Sawmill is a commercial auditing tool designed to provide a powerful, hierarchical log analysis tool. It is designed to run on many platforms including Linux. Its main functionality includes analysing web server logs, but can process almost any log. The logs supported include apache logs, Cisco logs, Microsoft logs of various sorts, Tivoli Policy Director and other Tivoli related logs and many more. It is mainly used by ISP to quickly and graphically analyse extremely large amounts of data. It is database driven[SAW01]. It is noted here for completeness only.

3.4 INTRUSION DETECTION SYSTEMS

There are several good Intrusion detection system tools on Linux. The one focused on here will be Snort. Snort is a free lightweight Intrusion Detection System for Linux that can monitor small TCP/IP networks. It is designed to detect a wide variety of suspicious network related attacks. It is essentially a libpcap based packet sniffer and logger that can be used for the above mentioned purposes. Some examples of the type of attacks Snort is designed to detect are buffer overflows attacks, stealth port scans, CGI attacks and SMB probes. Snort uses rules based logging to provide content pattern matching.

3.5 OTHER ORGANISATIONS

INTERSECT ALLIANCE

The Intersect alliance is developing a product called S.N.A.R.E. This stands for System Intrusion and Analysis and Reporting Environment. The motivation for the product is that they believe that one of the key factors holding Linux back from use in large organizations is its lack of Host based Intrusion Detection. That is, a system that includes auditing and event logging. S.N.A.R.E aims to provide C2 level audit abilities via a dynamically loaded Kernel module. This means the ability to gain Auditing capabilities without the need to recompile the kernel. This software is currently only in the development phase[SNARE01].

CHAPTER 4

MOTIVATION FOR DESIGN

This chapter will focus on the process of development of the final product in order to orientate the reader to the motivation for the final specification of the product. It will outline the motivation for design including a discussion of how the functionality requirements were set. A discussion of the design goals leads into the proposed operation. This is followed by a discussion of the process that led to the development of the final product.

4.1 MOTIVATION FOR DESIGN

The basic motivation for the product design that drove this project was the desire to see what host based logs could be correlated together to obtain more information than any single log by itself. This goal was a variation of the initial goal which was to see what network analysis tools could be integrated with host based logs to allow forensic analysis.

The correlation goal leads into a second goal. It was desired to see the effectiveness of querying these combined logs to further facilitate forensic analysis. This analysis was aimed at two main points

- Analysing data post hoc once an intrusion detection system has identified an attack or any other means of identifying a suspected attack. This is to more adequately analyse the damage.
- Help identify covert channels and other suspicious flows of data between files and processes.

4.2 DESIGN GOALS

The design goals for this project are defined as follows

1. Construct a prototype that can be evaluated and improved upon as a future project that will be derived from this project

Auditing and Event Correlation

2. Design the system to be extensible so that further functionality can be added. This is an important point as this project will be used as the basis for further work. This further work will incorporate more audit data and the system therefore needs to be extensible.
3. Design the system with efficiency in mind. Due to the large amount of audit data involved, the system must be reasonably efficient.

4.3 PROPOSED OPERATION

The design will consist of two parts that will operate together to provide the services of the system.

- A Consolidation component. The Consolidation component will combine various logs to produce one consolidated per user per session log. These session logs will contain all the needed data for the second component. The output of the Consolidation component will be a single per user/per session text log file.
- The analysis component will take all consolidated logs as inputs and create an object representing consolidated logs that it may then query. Each consolidated session log will consist of several records. Figure 4.2 illustrates this. It shows several consolidated sessions generated by the consolidation component being combined into an object representing several sessions that may then be queried. A user will then be able to query the combined data for desired information.

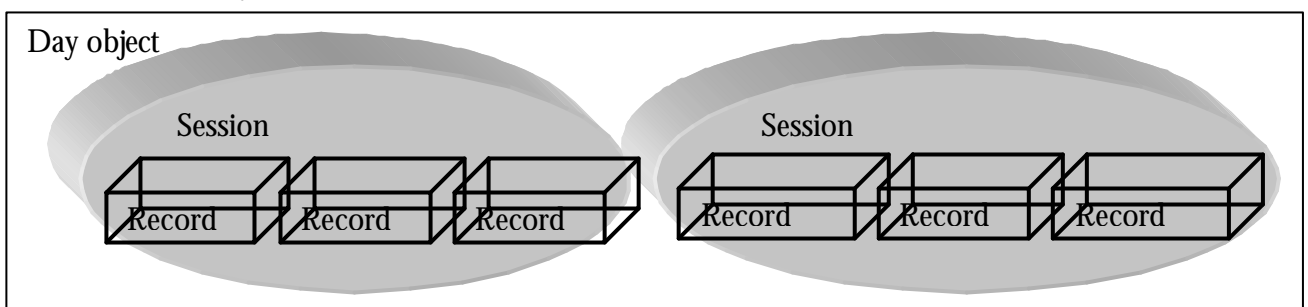


Figure 4.2. Query data organization

4.4 PROCESS OF DEVELOPMENT

This section will briefly outline the process of development. More technical details will be provided in the derivation of the specifications in the next section. The process of development of the E.C.T system started with an analysis of the state of the art of the intrusion detection field. In particular, a review of network analysis tools was undertaken. It was found that these tools along with host based audit logs provided the basis for most Intrusion Detection Systems that in turn provided real time alerts for system administrators. The first model that was developed is shown in figure 4.3. This shows how Intrusion Detection systems use either Network audit data or host based audit data or both to create alerts. The idea was to develop a system that then allowed the system administrator to further analyse the data via intelligent queries post hoc as part of a forensic analysis. Figure 4.3 was the first model concept for the project.

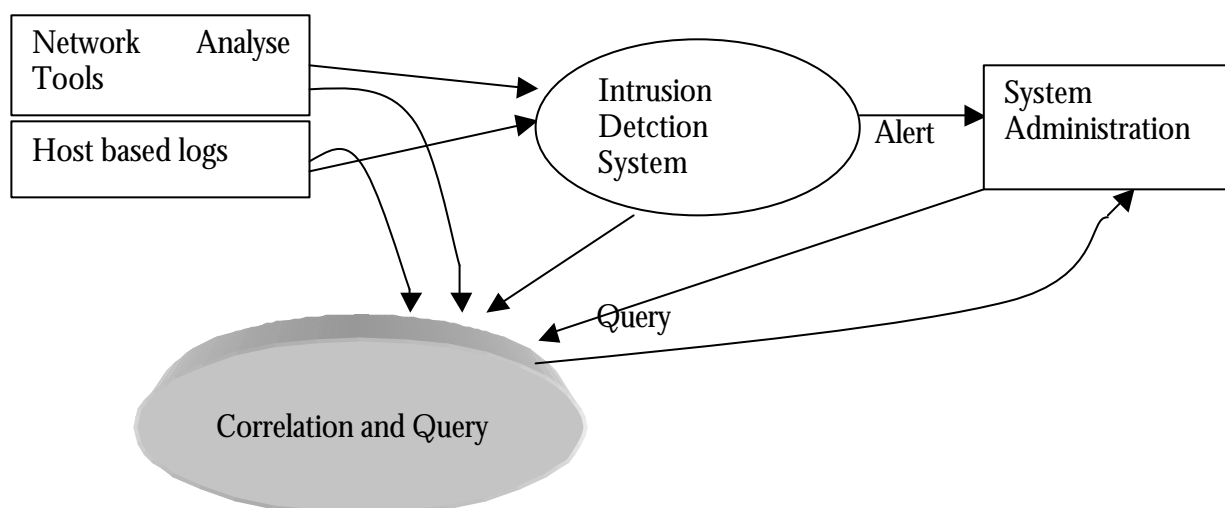


Figure 4.3. Original concept.

This then motivated a search on what sources of audit data there were on various Unix systems. At this point an operating system was chosen. Various operating system platforms were considered. The Linux operating system was eventually chosen because

- It is open source. Linux is open source with many free security products available. This makes it an attractive research platform. Companies such as the Intersect alliance believe that the lack of C2 audit capabilities is holding back Linux from deployment in the corporate environment when substantial security is needed.

Auditing and Event Correlation

- Industry security firms are increasingly supporting Linux. Firms such as IBM Tivoli and Computer Associates are now supporting Linux for their Security products.

After the operating system was chosen, the survey of products that obtain audit data on Linux found the products described in chapter 3. The central and most important piece of this survey was the identification of the Linux Basic Security Module(BSM). The Linux BSM provides a substantial basis for an auditing system and was perfect for our purposes. However, it is only considered alpha software and needs further work before a production system is available. Although it is only experimental software, it is stable and tested on the 2.2.17 Linux kernel. The discovery of the BSM made up the core of the auditing system we wished to design. Two other major needs that were perceived were, data on file access and user shell command line data. File accesses are something not covered by BSM.

This led to the investigation and discovery of the Elliott program. Elliott is not perfect for our purposes but can be changed to meet specifications. It provides analysis of insecure temporary file accesses via examining swap file. In addition, BSM does not include command line parameters. These can be useful in a forensic analysis of audit data because of the range of functionality a single Unix command can be made to perform depending on command line parameters. In addition the BSM can have multiple records for a single command. For instance, the 'vi' editor command shows three entries in the BSM log. It was decided that shell history could be used to make the BSM data more useful and readable.

Once the major components had been identified, a programming language had to be decided on. Due to the nature of text file processing and pattern matching required in the correlation, the Perl programming language was chosen. Perl is unrivalled in its pattern matching abilities and ease of manipulating string and integers. For this reason, it was chosen over a language like C which is extremely difficult to process files and text. At this stage of development Perl was believed to offer an object orientated programming facility. However, after reviewing many books, it was found that Perl was not a truly object orientated has was hoped. Documentation on Perl object orientated facilities is extremely short. The extensibility goal dictated that the query engine be designed using object orientated principles. This would allow the important goal of extensibility to be met. A database was

Auditing and Event Correlation

not chosen at this stage because of the time and learning curve involved and my complete lack of knowledge in this area.

CHAPTER 5

PRODUCT SPECIFICATION

The last chapter defined the scope and rationale of the project. This chapter outlines the exact requirements and specification of the product at this stage of development and, the major design goals will be revisited and elaborated on. The major software components are also introduced. Finally, major tasks and milestones are defined and a conclusion on this specification made.

5.1 MAJOR DESIGN GOALS

1. Produce a system that can be used to analyse data post hoc, possibly in response to an Intrusion Detection System alert. Such functionality is aimed at providing enhanced ability for performing forensic analysis.
2. Correlate several sources of data, both Kernel and application level, to obtain more information from the combined sources than either source by itself could provide. The types of data should included files accessed, kernel audit data and user shell command history providing session information and command parameters. This will fit in seamlessly with point 1.
3. Construct a prototype that can be evaluated and be improved upon as a future project that will be derived from this project.
4. Design the system extensible so that further functionality can be added.
5. Design the system so as to provide scalability.

5.2 MAJOR SOFTWARE COMPONENTS

The system will consist of six major software components. They are

1. The Linux Basic security module. This component is a kernel patch that provides system call level audit data. The Linux BSM consists of three major components.
 - The Kernel patch. The BSM patch is directly applied to the kernel. The kernel then needs to be recompiled. This is one of the disadvantages of the Linux BSM.

Auditing and Event Correlation

- Auditd. Auditd is the audit daemon that communicates with the BSM kernel patch. It maps kernel memory space to users memory space. The Solaris BSM operates in the same way. The output is not legible.
- Praudit. This is the analysis tool that turns the Linux BSM log produced by auditd into a text file that is viewable by a user.

The relationship between these tools is shown in figure 3.1.

2. Eliott. Eliott is a tool designed to log file creation/deletion/writes. Its primary use is to detect insecure file creation.
3. The bash shell collection scripts. This is a collection of scripts designed to collect user bash history. This consists of an especially modified bashrc file that calls an SUID program to copy the users .bash_history file upon login.
4. The consolidation tool. This tool is part of the development for this project. It will produce a correlated log of the bash history, Eliott and BSM log files.
5. The query analysis tool. This tool is the part of the development for this project. It is used to query the combined Bash, BSM and Eliott correlated file. The types of query will be defined later.
6. Linux Operating System. The system all the above are designed to run on is the Linux Operating system.

Figure 5.1 represents how the six components are related

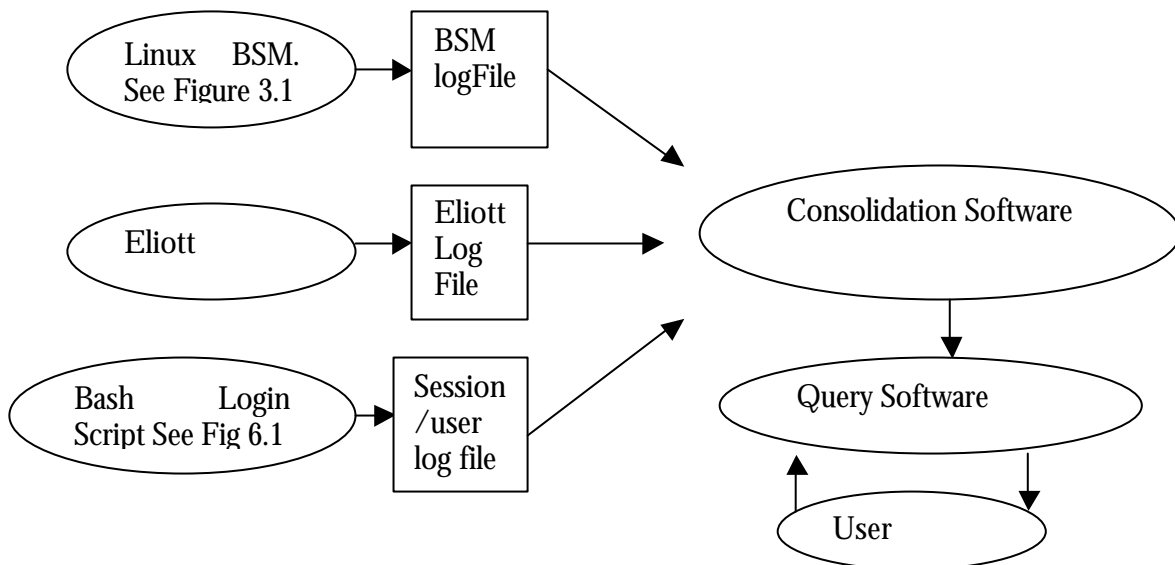


Figure 5.1. The system components relationship

5.3 DATA FOR CONSOLIDATION TOOL

DATA USED BY THE CONSOLIDATION TOOL

The consolidation tool is aimed initially at correlating three sources of data. They are

- BSM audit data
- Shell history audit data
- Elliott file access audit data

In this section, the type of audit data and advantages of each are discussed.

SHELL HISTORY

The Shell history mechanism is provided for users to use shell command line recall. It logs command line commands and associated parameters. It is not a secure source of information because this type of information logging can be avoided, although it requires some effort. However, session information and command line parameters can be achieved from this type of log were possible. This makes it a valuable source of data as command line parameters often make the difference between a valid command and a malicious command [Axe00].

BSM

The BSM output consists of a number of records. An example of a BSM record is shown in figure 5.1. The BSM works in such a way as to allow multiple records for each command entered. The data worth noting in the BSM output is:

1. User ID, Process ID, Effective UID, Group ID, Effective GID.
2. Time
3. Subject
4. Return code.

ELIOTT

The output of the Elliott program is shown in figure 5.2. Properties of this data worth noting are:

1. Time
2. File access (swap file)
3. User ID

Auditing and Event Correlation

ADVANTAGES OF CORRELATING THIS DATA

BASH-BSM

The BSM contains substantial information. However, the output is very hard to read, contains no session information, contains no relationship between commands executed and BSM output and does not have command line parameters. The shell history file on the other hand contains command line parameters. The consolidation of the two files can provide

- A mapping of BSM output to bash commands making the BSM audit data easier to interpret.
- Times for the history file commands.
- Provide command line arguments for the BSM data.

BSM_ELIOTT

Eliott provides information on the files accessed. The advantage of correlating BSM data to Eliott is that BSM provides no details on file accesses. This is one of the systems goals.

5.4 SPECIFICATION OF CONSOLIDATION TOOL

INPUTS TO CONSOLIDATION TOOL

The inputs to the consolidation tool consist of the following at this stage of development.

- 1 session/user log of a bash history file.
- BSM log containing all sessions and users. Each shell history command can map to multiple BSM entries.
- 1 Eliott log file.

OUTPUTS TO CONSOLIDATION TOOL

The outputs of the consolidation tool consist of the following

- 1 session/user log consisting of the correlated BSM, ELIOTT and shell data.

TASKS TO BE PERFORMED BY CONSOLIDATION TOOL

1. Prepare shell history file for consolidation. This consists of the following components

Auditing and Event Correlation

- In order to map a shell history command, a mapping of that shell command to its BSM “template” is needed. In order to achieve this mapping, the first token of the shell history file must be separated from its command line parameters.
- Find out if each command is a valid bash shell command or not. If the command is not a recognized command, then it must be marked invalid. This will prevent the consolidation tool from looking for BSM entries that do not exist in addition to making the system faster. Any commands with invalid parameters will be dealt with later. Figure 5.2 shows a simple example of how this processing occurs.

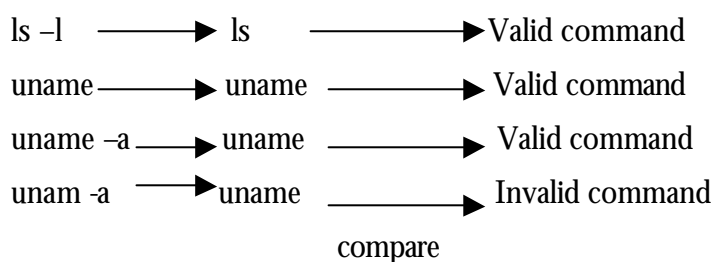


Figure 5.2. Validating commands example

2. The BSM log also needs some pre-processing. The following needs to be done
 - Separate the BSM log file into multiple log file according to User ID. This will allow consolidation with individual shell history file.
 - The BSM entries need to be grouped by Process ID. This will help to group the BSM entries for mapping to each shell command.
 - The BSM entries need to be grouped by Time. Any entries that occur within 200 ms have most likely part of the one shell command. This will indicate possible relationships between the entries and help classify them as valid and invalid.
 - The two groups based on time and Process ID need to be combined to form one group representing the probable grouping of BSM entries. These groups should then almost exactly match the Shell history commands.
3. Pre-process the Elliott Output. The Elliott log is trimmed by extracting records with the UID we are interested in. The times of the individual entries will need to be extracted here too.
4. Do the actual consolidation. This consists of the following
 1. The Bash/BSM mapping presents two cases that must be dealt with.

Auditing and Event Correlation

1. The case in which the bash command is valid. If the shell history command was valid, then a simple search of the BSM log for its corresponding map will provide the required information
 2. The case in which the bash command is invalid. There are two possibilities here. If the command was a program, or shell script that was executed, then a search of the BSM output will find the characteristics of this. If command was indeed not a valid command, then it should be discarded.
2. Invalidation checking. Figure 5.2 shows a situation in which this system cannot predict the correct entry to correlate. This is due to the fact that the system can produce BSM entries on behalf of the user. The simple example in figure 5.2 shows a situation in which the system has logged extra grep commands on behalf of the user. In this situation, the system can make no valid conclusion as to which entry should be the correct one associated with the entry in the shell history file. This system will implement a look back invalidation method that will be explained in chapter 6.

5.5 SPECIFICATION OF THE QUERY TOOL

The query tool is aimed at being able to satisfy the following questions:

1. Find any correlated records that involve File X within a specified time.
2. Find any correlated records that involve user with UID Y
3. Find all records between times T1 and T2.
4. Find any record that involves process ID Z.
5. Find the intersection of any of the above.

The query tool is written in java. This is because extensibility needs to be built into the program thus requiring a good object orientated language.

CLASS STRUCTURE

The nature of the correlated log file leads to a natural object orientated approach. Multiple correlated logs representing one session each will need to be queried. Therefore, a class representing a “day”

Auditing and Event Correlation

object would be appropriate. This day object would consist of multiple sessions. Each session would be represented as a session object. Each session in turn would consist of a number of record objects each representing a record in the correlated log. This leads to the following class structure pictured in figure 5.3. In addition several other classes exist to support these main classes. A ControllingWindows class will control the Graphic user interface. Additionally, a TokenizeTime class will exist to break a time down into its components.

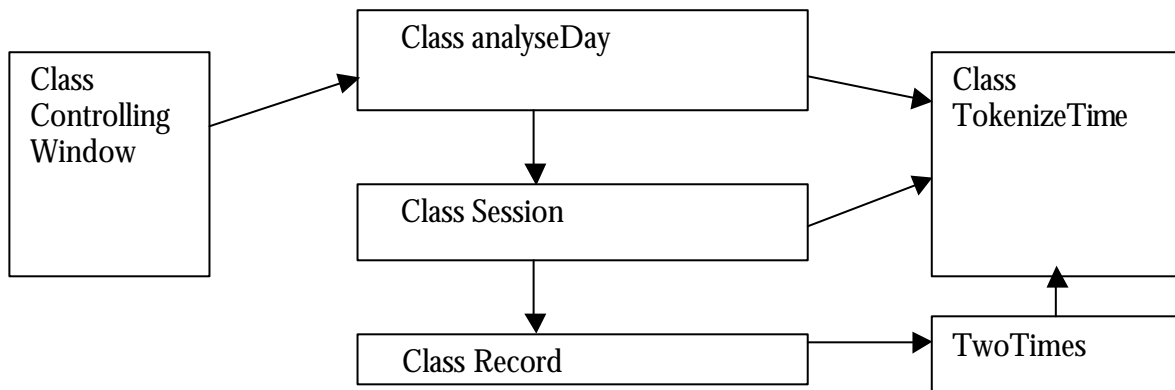


Figure 5.3. Class Structure for query tool.
METHODS OFFERED

The major methods offered by each class of the analysis tool are discussed here.

Class analyseDay

This class should offer methods to do the following.

- Return all records in all session that have the file X in them.
- Return all records in all sessions that have the UID Y in them.
- Return all records that exist between time T1 and time T2.
- Return all records in all sessions that have the PID A in them.

Class Sessions

This class should offer methods to do the following.

- A method to search all records in a session and return records containing file X.
- A method to search all records in a session and return records containing UID Y.

Auditing and Event Correlation

- A method to search all records and return records having a time between Time T1 and time T2.
- A method to search all records in a session and return records containing PID Y.

Class Records

This class should offer the following methods.

- A method that returns a Boolean whose return value depends on whether a particular UID is present in the record or not.
- A method that returns a Boolean whose return value depends on whether a particular File is present in the record or not.
- A method that returns a Boolean whose return value depends on whether a particular record's time is between the two boundary condition T2 and T2 parsed to the method.
- A method that returns a Boolean whose return value depends on whether a particular PID is present in the record or not.

Class TokenizeTime

This class should contain the following methods:

- Three methods to return the Hour, Minute and second of a time string in the format HH:MM:SS.

Class TwoTimes

This class should contain the following methods:

- A Method to return a Boolean subject to the evaluation of the parsed time compared to the boundary times.

5.6 MILESTONES THE PROJECT IS TO ACHIEVE

This project aims to achieve the following milestones:

1. Develop a secure method of obtaining user shell history for use in security audit application.
2. Show that the correlated BSM-bash log is useful in providing information to analyse system attacks. This involves the implementation of the BSM-bash consolidation. This is not a trivial task as there is no direct correlating factor between the two. In fact, a direct correlation cannot always be achieved. This will be outlined further later.

Auditing and Event Correlation

3. Show that the correlation between Elliott and the BSM is useful in providing information to analyse system attacks.
4. Develop a query system that can interrogate the correlated log.

CHAPTER 6

IMPLEMENTATION

This chapter describes the actual software implementation of the E.C.T product. It will start with a look at the consolidation tool followed by a discussion of the implementation of the query tool. A flow chart of the consolidation tool will be discussed and the function implementation explained. The implementation of the query software will then precede a conclusion on the overall software implementation. A complete code listing of both the consolidation tool and the query analyse tool can be found in Appendix 1.

6.1 OBTAINING SHELL HISTORY

As mentioned in chapter 3, a method of obtaining user shell history has been derived. It involves noting that user history is kept in memory and not written to the history file until the user logs out. If a hacker deletes his history file after malicious activity, he may even think he has covered his tracks when all his movements are secured in a super user owned location. This current method of obtaining this information is to make a minor change in the bash shell to force the shell to run the bashrc file in the /etc directory whenever it is invoked. The bashrc file then runs a SUID program and copies the user history file over to a secure root location. Access to other shells is restricted by moving the shell binaries to another location that is not accessible to the user. It must be stressed that the current system does not guarantee shell history as the user can recompile the shell or import a new shell as long as a restricted shell is not in use. However, it makes it much harder for the user to avoid this kind of high-level auditing. In fact, any use of an alternate shell or suspicious use of command line aliasing can then be treated as signs of suspicious activity. The intention here is to grab user history when available to help provide a better picture of what is going on. This is shown in figure 6.1 below.

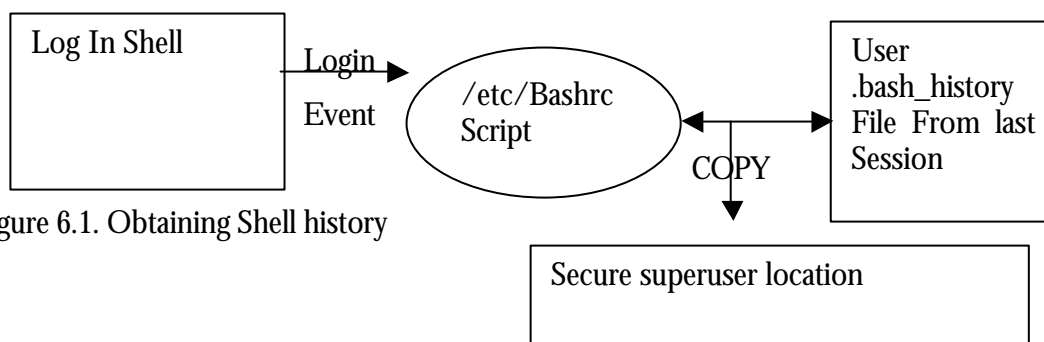


Figure 6.1. Obtaining Shell history

6.2 CONSOLIDATION TOOL FLOW CHART

The following is a main flow chart representing the consolidation tool. It will define significant functions and their functionality will be discussed. A discussion of the main flow chart will follow and then other major functions and their flow charts will be discussed.

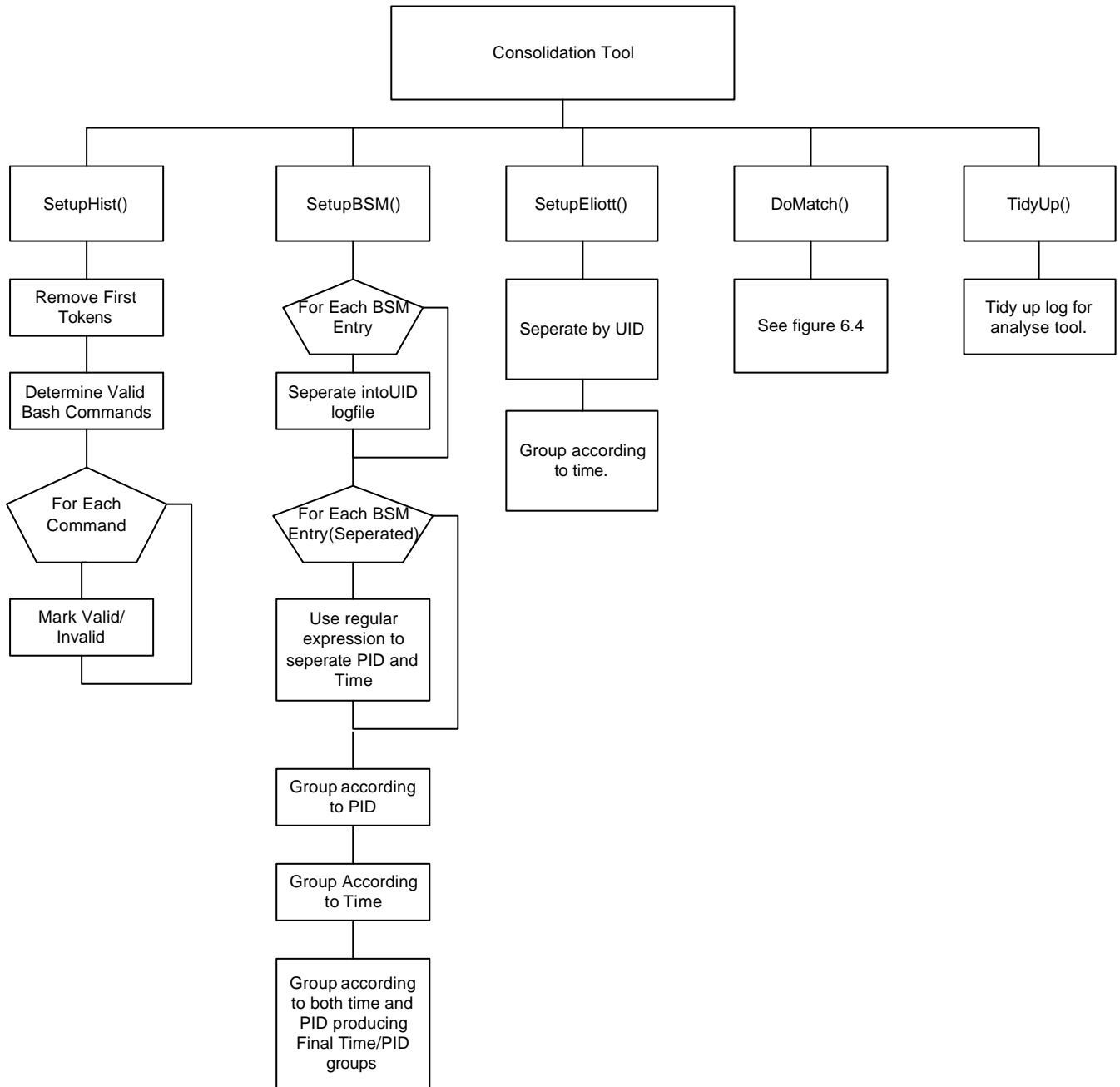


Figure 6.2

Auditing and Event Correlation

The flow chart represents the follow functions

1. SetupHist(). This subroutine is responsible for removing the first tokens of the history file. Once this has been done, each of these commands can be labelled valid or possibly invalid. A label of possibly invalid means that the command could be either a valid shell script or program command or could be garbage. The valid/invalid labels are stored in an array and the 'for each' control statement is used to analyse each in turn.
2. SetUpBSM(). This function is responsible for setting up the framework for Linux BSM log so the matching subroutine has enough information to proceed. The first thing this subroutine does is to separate the file by User ID. It creates a new file for each UID. Regular expressions are then used to group the BSM entries by Process ID. This is stored in an array with each element of the array containing the number of elements in each group. This will be discussed with its own flow chart later. Next, the BSM entries are grouped according to time. A BSM entry is considered a group if the time between it and the next x entries is less than 200 ms. This gives a good representation of the group mappings that the actual BSM software produces.

Finally, neither the time grouping nor the PID grouping by themselves make up an accurate representation of what the actual groups should be. For example, sometimes a process will sporn a new process, creating a new process with a new PID. We wish to group these entries together because they are caused by the one command. This is where time analyse is useful. It allows us to detect sporned processes and group them together. Note however, that the time by itself does not group all relevant BSM entries correctly by itself either. This is because sometimes a shell command can take longer than 200 ms. The solution to this problem is to use both time and Pid groups to create a final grouping. The details of this logic will not be discussed here. This will not get all of the mapping one hundred percent of the time, however, it is very effective in getting most. This system was never designed to get all matches one hundred percent of the time as there are simply some cases that could never be handled due to the lack of the time correlating factor between bash and BSM log files. Once these two groups are created, a final grouping must be created. This will be discussed with its own flowchart.

Auditing and Event Correlation

3. SetupEliott(). This eliminates any Records in the log other than those with the UID we are interested in. The time of the record is extracted.
4. DoMatch(). This is a subroutine that actually performs the mapping between the BSM log and the shell history log. While bash entries have not been matched or accounted for, search for a match for the next entry. The actual match algorithm will be discussed with its own flow chart.

Figure 6.3 is a flow chart representing the GroupPID subroutine.

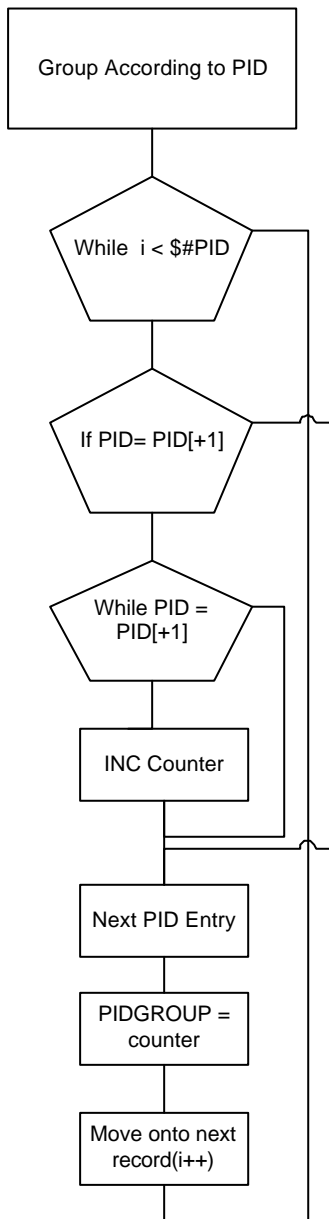


Figure 6.3.

Auditing and Event Correlation

This flow chart shows the process of grouping the BSM log by Process ID for the purpose of mapping the grouped BSM entries to shell history commands. Any contiguous block of entries with the same PID is considered a group. Groups of BSM entries are formed in much the same way. Once groups of PID and times have been formed, a final group based on both of these is formed. The time group takes precedence in deciding what the final grouping of BSM entries should be. This is because a command can consist of many entries, all with varying PID. However, if a PID group consists of several time groups, the PID group will take precedence in deciding what the final grouping is.

The process of mapping the shell history to the BSM output is quite complex. It is done in the DoMatch() subroutine. The Elliott output is also correlated here. In order to map the shell history to the BSM outputs, templates of what the BSM will look like had to be created. These templates have only minimal information and contain no specific information. For example, in a cd command, the return code, time, PID, UID etc would not be included in the template. All this information is removed because this system works by associating an assumed BSM template to each shell history entry. The actual BSM data is grouped by the process already outlined. Once grouped data has been obtained, then the grouped BSM data is stripped of all specific information also. The template associated with each shell history command can then be compared with the stripped BSM data. If a match is obtained, then that match is assumed correct for the moment until the match is checked for invalidation.

An Important thing to note is the method of invalidating entries when no reasonable conclusion about their association with other elements can be made. This problem was discussed in chapter 5 and is illustrated in figure 5.2.

6.3 QUERY ANALYSIS TOOL

The query analysis tool consists of the following classes

1. Analysis
2. ControllingWindow
3. AnalyseDay
4. Session
5. Record

Auditing and Event Correlation

6. TokenizeTime
7. TwoTimes

The constructor for each of these classes do the following

1. Class analysis. This is the class containing the main method. Its job is to set up a new controlling window and set that window to visible.
2. Class Controlling Window. This is a class that controls the graphics and event handling. It consists of three buttons. This class was adapted from code written by Dan Johnson for a comp2500 assignment.
3. Class analyseDay. The constructor for this class accepts a filename. This file name contains the list of session file names to be loaded. The constructed creates a new session object for each line in the input file, using the input file name as a parameter to the session constructor.
4. Class Sessions. The constructor for this class constructs a session object. Again the parameter is the filename of the session log. It then proceeds to split the session up into individual records. This is the most involved piece of code in the program. For each record in the session, a new record object is created. The record class has an overloaded constructor that can take a string of the records to be created, the UID and optionally a string containing the Elliott data. The UID is extracted from the record string by using the String Tokenizer. The string RECORD delimits records.
5. Class Record. As mentioned above, the constructor for the record class consists of 2 overloaded constructors. It creates a records object with UID, records string and optional Elliott string.
6. Class TokenizeTime. This class accepts a string in the form HH:MM:SS that represents a time and converts it to a time object consisting of Hour, Minute and second instance variables.
7. Class TwoTimes. This class constructor creates an object with two boundary times and a time to compare. It uses the TokenizeTime class.

The methods each class offer is:

1. Class Analyse. The class contains only the main method.

Auditing and Event Correlation

2. Class Controlling Windows. This class consists of seven buttons. The open button opens a file containing a list of all the session file names to analyse. The File button makes another dialog box appear and asks for a file to search for. It then displays all records containing that file. The “UID in Record” searches all records for records containing that particular UID. If any records with that particular UID are found, the records containing the UID are returned and displayed. The complex button prompts for two times. It then allows a choice of refining the query by offering to break down the search by the intersection of UID, file, PID and time or any combination. Two buttons related to the syslog facility are present. The syslog button searches the syslog message log for entries between specified times. The Su button searches for entries showing more than 5 su entries between a given time.
3. Class AnalyseDay. This class offers four methods. The NumberOfRecUID(String UIDname) method returns the number of Records containing the string UIDname parsed to it. This is then used by the controllingWindows class to setup an array of the appropriate size. This array is then used by the controllingWindows class to hold an array of records returned by the fileAccessedByUID() method. Note the similarity of these two methods. This is because one returns the number of values so as to initialise an array that will hold the actual results returned by the other. A similar arrangement exists for the other two methods in this class. The NumberOfRec() method returns the number of elements that the DayHasFile() method will return. Again, NumberOfRec() is used by the controlling window to find out how many records DayHasFile() will return and sets up an array of the correct length in preparation for DayHasFile() to be called. DayHasFile(String FileName) searches multiple sessions and returns an array of records containing the parsed String FileName that represents the Elliott data for a record. The fileAccessedByUID(UID) method searches all records in all session for records with file accesses that contain the UID parsed. The TimesBetweenDay(TokenizeTime T1, TokenizeTime T2) method accepts two times. It returns all records in all session that are between these two times. The DayHasPID(PID) method searches multiple sessions and returns an array of records containing the Process ID PID.

Auditing and Event Correlation

4. Class Session. This class consists of 4 methods and works similar to the last class. The analyse day class uses the MatchRecordNumber() and MatchRecordNumberUID to work out the size of the array needed for the sessionHasFile() and sessionFileAccessedByUID methods respectfully. The sessionHasFile(String fileName) method searches the session object for records containing the Elliott data string 'filename'. It returns an array of records that fit the above requirement. The sessionFileAccessedByUID searches the session object looking for records containing the UID passed to the method. It returns an array of records that fit the above requirement. The sessionTimesBetween(TokenizeTime T1, TokenizeTime T2) method accepts two TokenizeTime objects representing the two boundary times and returns all records that fit the constraint. This function is used DayTimesBetween() method from the analyseDay class. The MatchRecordNumberPID(PID) method returns the number records with a Process ID of PID in a session. The SessionHasPID(PID) returns an array of record objects that contain the Process ID PID.
5. Class Record. This class consists of three methods used by the Session class. The hasUID(String UID) method returns a Boolean, true or false, depending on whether the UID is the UID parsed by the caller or not. The hasFile(String fileName) returns a Boolean, true or false, depending on whether or not the string filename exists in the record object or not. The returnRecord() method returns a string containing the String that represents the record. The TimesBetween(TokenizeTime) method again accepts two time objects and compares them to the TokenizeTime object that belongs to the current record object. The code to this is slightly involved and won't be discussed here. HasPID(PID) returns a Boolean depending on the PID present in the current record object. If it is present the method returns true or else it returns false.
6. Class TokenizeTime. This class consists of three methods. Return_Hour returns an integer representing the hour of the object. Return_Min returns an integer representing the minute of the object. Return_Sec returns an integer representing the second of the object.

Auditing and Event Correlation

7. Class TwoTimes. Only one method is present in this class and its job is to return a Boolean depending on whether the time to be compared is between the two boundary times or not.

6.4 OTHER COMPONENTS

Eliott currently only log accesses to one directory at a time. A shell script was developed to append all the Eliott scripts into one complete log file. The Korn shell scripting language was used to do this. The directory to which the Eliott program logs is scanned for files and while there are still files, the script appends the files into a single log file.

The BSM currently only runs on the Linux 2.2.17 Kernel. This Kernel is not available anymore. The BSM was ported to the 2.2.18 Kernel which was available. This was not a major task as only minor code relocation was required. However, later in the project it was discovered that Eliott needed to be run on 2.4.x. This meant porting the BSM up to the 2.4.x kernel. Two weeks were spent on this task. This was much more than originally planned. Large technical changes in the Linux kernel between 2.2.18 and 2.4.x releases meant that there were many technical issues involved. Code relocation was a problem due to the change in size of the kernel. This increase in code size meant that the existing BSM patch did not work on the newer Kernel. The code needed to be applied manually to the fresh 2.4.x Kernel. Additionally, several semaphores were added to the code. This was a new feature in the 2.4.x kernel design that was not present in the old kernel. Several other changes to function names between the kernel versions and the extremely slow development machine meant a painstakingly slow process. The final result was that the BSM worked briefly at run level 3 before crashing due to a memory error. At the time of development, it was decided that no more than two weeks should be spent on this task due to the proof of concept nature of the project. Furthermore, it was decided that more valuable research and development could be achieved given the relatively limited time frame of the project.

Chapter 7

RESULTS

This chapter will outline how well the final implementation of the product meets the specifications outlined in chapter 4 and 5. It will discuss the unit testing procedure and results, outline the procedure and results of system testing. An analysis of the functionality achieved in the final product is undertaken followed by an assessment of the performance of the system. Finally it exposes design shortcomings, discusses implementation issues and makes a conclusion about the results.

7.1 UNIT TESTING

The purpose of the Unit testing was to evaluate the core functionality of the product. The scope of these tests includes the following

- Testing for correlation between shell history and BSM entries in simple cases. Simple cases are defined as those in which each shell command has a directly related BSM command and no multi-record BSM entries are present.
- Testing for correlation between shell history and BSM entries in simple cases that involve extra-simulated BSM entries due to the system. This is designed to test the invalidation mechanism.
- Testing of correlation between shell and BSM entries in complex cases. This involves multi-line BSM records being matched to a single shell history command.
- Testing the correlation of the Elliott output to the correlated shell history and BSM commands.
- Testing the query system for functionality including PID, UID, File and Time searches. The intersection of these are also tested.

The unit testing was carried out by writing shell scripts that ran Unix commands. The BSM and Elliott programs need to run on different kernel versions, so for the purposes of these tests, the shell script was run on both kernel versions and the Elliott times adjusted. The test scripts are not included in this thesis documentation.

Five tests were carried out and are summarized in table 7.1 below.

Auditing and Event Correlation

TEST SET	Description	Purpose	Result
1	Mostly simple cases, a shell script present and no invalidation	Confirm basic functionality and correlation of shell scripts	The output met specifications
2	Mostly simple cases, a shell script present and invalidation needs to be performed(Modified BSM data)	Test the invalidation mechanism.	The output met specifications
3	Test more simple commands, some multi-line, no invalidate	Test simple commands with another data set.	The output met specifications
4	Same as data set 3 with invalidation needed(modified BSM data)	Test data set 3 with invalidations	The output met specifications
5	Test complex multi-line correlation, no invalidation	Test multi-line correlation	Some errors, these were fixed

Table 7.1. Unit testing results.

7.2 SYSTEM TESTING

The aim of system testing is to evaluate the usefulness of the product. This involves creating an attack against the system to show the usefulness to the real world. The attack will be demonstrated on Demonstration day at the Innovation Expo. The demonstration will consist of a simulation of a hacker hacking into the system via a send mail buffer overflow vulnerability. The attacks will then

install a root kit. The functionality of the system will be proved and the usefulness in detecting the attack highlighted.

7.3 FUNCTIONALITY ANALYSIS

The consolidation tool current correlates three sources of data. The BSM, shell history and Elliott data. The consolidation tool produces a per session/user log of these three logs. A method of obtaining user shell history was implemented. However, this data cannot be guaranteed due to the nature of logging mechanism. The major components have all been installed and run. However, one major shortcoming is that the Linux BSM will only run on 2.2.17 Linux Kernel. An extensive attempt to port the BSM to the latest 2.4.2 Linux Kernel was only partly successful. Some BSM functionality was demonstrated but the kernel soon crashed due to a memory mapping. As my skills in the area are limited, it was decided not to waste too much time on this task. Proof of concept was enough for this version of the product.

The Query tool delivers a query engine. This query engine considers multiple session/user logs at once based on the file containing the names of all the session logs. The query engine is capable of the following queries:

1. Find any correlated records that involve File X within a specified time.
2. Find any correlated records that involve user with UID Y
3. Find all records between times T1 and T2.
4. Find any record that involves process ID Z.
5. Find the intersection of any of the above.

It provides this functionality via a graphical user interface written in Java.

7.4 PERFORMANCE TESTS

This product is a lightweight auditing tool. However, some basic performance testing is needed. The testing was performed on a Pentium 166/MMX with a clock speed of 167.047 MHZ and 32 Mbytes of RAM. Table 7.2 represents the time taken to perform each of the Unit tests in table 7.1 for the consolidation tool.

Auditing and Event Correlation

TEST NUMBER	TIME TAKEN
1	1 Second
2	1 Second
3	1 Second
4	2 Seconds
5	1 Second

Table 7.2. Performance of the consolidation tool.

The query tool speed performance was assessed for each of its functions. The parameters used are shown in table 7.3 along with the function tested and the time for execution. For completeness, the tests were run with windowed graphical output and non-graphic output.

Button Tested	Input parameters	Time with graphic output	Time with standard output
UID IN RECORD	0	5 seconds	0.5 seconds
HasPID	2114	1 second	0.5 seconds
HasFile	Testfile	3 seconds	0.5 seconds
Records Between	T1=12:31:43 T2=15:31:47	5 seconds	0.5 seconds
Syslog	T1=12:45:07 T2=12:45:15	4.5 seconds	4 seconds
SuActivity	T1=12:45:00 T2=12:45:59	5.5 seconds	4.5 seconds

These results show that the test machines were extremely short on RAM. There was a lot of hard drive activity when running the windowed output tests. This indicated lots of demand paging due to a lack of ram. The last two Syslog related button showed higher processing times for both standard output and graphic output. This is due to the large message log tested.

7.5 LIMITATIONS OF THE CURRENT MODEL

There are several key limitations in the product. These were either intentionally overlooked because of time frame or have only become apparent while building and testing the product.

PORTABILITY

This product was always designed to be a prototype to provide proof of concept. The end aim of systems such as this would be to operate on a network with many heterogeneous operating system and hardware platforms. It is visioned that a central console be used to contain host based data from all networked consoles. This system was designed exclusively for the Linux Operating system due to the time frame and nature of the prototype. Price's[price97] paper discusses the need for a standardized audit trail.

EXTENSIBILITY

The current consolidation tool uses Perl to perform the correlation. Perl is object orientated but is not as good in this area as languages such as Java. In this project, no object-orientated design was utilized in the correlation. This is seen as a major but planned design flaw when it comes to extensibility because it does not allow as much code reuse. The next chapter will explain the redesign of this component using the JPL programming language. This is a product that allows Perl to be embedded into Java. It is expected that an object-orientated design would facilitate an easier and quicker interface to the existing components. This would allow the correlation of further logs in a more extensible manner.

DATABASE

The query system would be better implemented using a database. This is not seen as a huge change because a lot of the framework for setting up the database is already in place. This would allow new queries to be created quicker and easier. However, for this prototype, there was insufficient time to do this.

7.6 IMPLEMENTATION ISSUES

There were several implementation issues experienced during this project.

BSM ALPHA

Auditing and Event Correlation

The Linux port of the Solaris Basic Security module is considered alpha software and does not offer all the functionality offered by the Solaris version. The BSM is also only available for the 2.2.17 kernel as mentioned before.

ELIOTT SHORTCOMMINGS

The Elliott program was not quite all it was first thought to be. It logs file creation, file deletion but does not log file reads. Some reads are detectable due to the use of swap files creation and deletion. This originally made the author think that reads were also logged, however it does not log file accesses to some graphical editing applications like gedit.

SHELL HISTORY

This project desired to see what benefit shell history could be to an auditing application. No way was found to make shell history obtainable one hundred percent of the time. This is due to the fact that the user can import and compile their own shell binaries, therefore circumventing our security procedures. Also, they can use aliases to mask their real intentions. A recommendation for a future project would be alias analysis. If a user were to alias a command or program, the BSM output would not change. The consolidation tool however would be recognising the correlation between the BSM entries and the aliases command. There would be BSM entries left that had no corresponding correlation to the shell history. Such activities could be considered suspicious.

CHAPTER 8

FUTURE DEVELOPMENTS

This Project will be part of an ongoing research effort. This primary goal of this Thesis was to develop a prototype. This prototype has been evaluated in the previous chapters and the shortcomings of its design were outlined. This chapter will outline future developments that the next version of the product will address. These include an improved Basic Security Module, an improved file-logging program, an object orientated correlation model and the use of a DBMS for the query engine. A basic design recommendation for the object-orientated design of the correlation will be included.

8.1 IMPROVED COMPONENTS

BASIC SECURITY MODULE

The current version of the Linux Basic security Module is only alpha release. It is stable and tested on Linux 2.2.17 however, not as much functionality is included as was hoped. No new work or updates on this product have appeared this year as was hoped. As mentioned in chapter 3, an organization called the Intersect alliance is producing a product called S.N.A.R.E that will provide substantial auditing for Linux. As of this writing, no prototype version has been released to the public unfortunately. During the course of this project, I attempted to port the current Linux BSM to the current 2.4.2 kernel. This was unsuccessful due to my lack of experience in this area. It would be desirable to do two things to the current Linux BSM:

- Port the 2.2.17 version to the 2.4.2 version
- Include more functionality to bring the Linux BSM closer to the Solaris BSM.

ELIOTT

Eliott is designed for insecure file creation detection. As such, it is useful for application in this project but not ideal. It will log file creation, file deletion and detect file accesses via the creation of temporary swap file creation. The disadvantage of this is that graphical application such as gedit do not create swp files when accessing files. This means the file accesses in this manner are unfortunately not detected. It would be desired that an improved program possibly based on Eliott be created to detect all file accesses.

8.2 OBJECT ORIENTATED MODEL FOR CONSOLIDATION

TOOL

The Consolidation Tool was developed using Perl. Perl is an object-orientated language but it is not as effective as Java when it comes to object orientated design and implementation. One of the projects goals was that the consolidation tool be designed extensible. It was found that procedural Perl programming got the job done but will make it difficult to add further functionality. The design of the consolidation tool would have benefited from the use of Object Orientated principles. This section will outline the redesign of the consolidation tool for use with JPL.

JPL or Java Perl Lingo is a programming language that allows Perl to be embedded inside java or java to be embedded inside perl. The former of these is used in practice due to some shortcomings in the language when it comes to embedding java into perl. Its primary use is to take advantage of the radically different features of both languages. Perl has unsurpassed pattern matching and text processing abilities. Java on the other hand is fast becoming the mainstream standard in object-orientated programming. Java profits from vast graphics routines and extremely well supported object orientated functionality. JPL comes as part of the O'Reily toolkit. The use of JPL in this application was obvious. The use of Java's object orientated functionality and graphics support could be combined with Perl use of regular expression to provide a better model than the current procedural implementation.

There are obvious abstractions that make the object orientated approach so attractive in this application. This implementation of the query tools highlighted the advantage of considering records

Auditing and Event Correlation

as objects with methods that may be invoked on them. This would provide more encapsulation and provide a better interface for the various logs to be correlated.

The following is a discussion of the proposed class structure of the object-orientated model of the consolidation tool. It consists of the following classes:

- A controlling class. This would contain the main method etc.
- An ElliottLogGrouped class which would consist of Elliott records grouped by UID.
- Elliott Record Class. Creates an object consisting of an Elliott record.
- BashData. Creates an object consisting of the bash data for one session.
- A BashRecord class. This would create an object representing of one bash record.
- A bashBSM class. This class would represent correlated bash/BSM record.
- A BSMRecord class. This would represent one BSM record.
- A BSMGroup class. This would represent one BSM group that can consist of several BSMRecords.
- A FinalRecord class. This class would represent a final record.
- A FinalLog class. A list of FinalRecords

CHAPTER 9

CONCLUSION

This chapter contains a discussion of the outcomes of the project, both as a research effort and as an undergraduate Engineering thesis. The results are analysed and an assessment of the final product is made.

Project Outcomes

This project designed and developed a prototype tool for correlating, consolidating and analysing different forms of host based audit data on a Linux system. User shell history, specifically command line parameters were used in the audit and event correlation system. Additionally, file access audit data was incorporated and consolidated as well. Finally, the Basic security audit data was also used to add return codes, times and paths.

A method of obtaining user shell history was identified. This shell history was successfully correlated with the Linux port of the Basic security module audit data. This correlation between shell history and BSM output demonstrated two things.

1. The command line parameters from the shell audit data are useful in distinguishing malicious commands from non-malicious commands. Command line parameters are also useful in identifying exactly what a command was aimed at achieving.
2. The correlation Between the BSM and shell history showed that it made the BSM significantly easier to read once it was broken up into groups representing shell commands making it much more useful as a forensic analysis tool.

Audit data from the Elliott file access tool was incorporated into the consolidated log. A query tool was designed and implemented that allowed the analyse of the intersection of all of the above data. The intersection of the file access information and the process information due to the BSM allows the product to be used to identify information leakage between processes and files. In addition, data from

Auditing and Event Correlation

the syslog file messages was integrated within the query tool allowing analysis of system messages in relation to the suspicious user commands and file access activity.

The product performed to the design specification. The product met the testing requirements outlined in chapter 7 and the analysis led to the design alternatives identified in chapter 8.

This prototype system will be useful in detecting information leakage and tracking user activity post hoc possibly in response to Intrusion detection systems. However, it is currently of little practical use due to the fact that BSM and Elliott operate on different versions of the Linux Kernel. However, this is not a huge technical challenge and someone with the appropriate technical knowledge on Linux kernel modules would be able to easily solve this problem. This will hopefully be the people who designed the BSM.

As an undergraduate thesis project in the computer system area, it required significantly different technical skills that had to be obtained during the course of the project. Namely learning Perl, Java and the Linux operating system. My personal motivation for undertaking this type of project was that I'll be working in the security software industry with IBM/Tivoli next year with skills such as these a necessity. As such, the project has exceeded my personal expectations of personal development and contribution to the academic security research area.

Auditing and Event Correlation

BLIOGRAPHY

- [DOD85] <http://www.radium.ncsc.mil/tep/library/rainbow/5200.28-STD.html> Trusted Computer System Evaluation Criteria, DOD standard 5200.28-STD, December 1985
- [RG91] Deborah Russel and G.T. Gangemi Sr, Computer Security Basics. O'Reilly
- [Price97] Host-Based Misuse Detection and Conventional Operating Systems' Audit data collection. Katherine. E. Price. Thesis. Purdue University, 1997.
- [CSJ99] Computer Security Journal. Volume XV, number 2, 1999.
- [Gol99] Computer Security, Dieter Gollmann, John Wiley and sons 1999.
- [CAR00] Interoperability in Intrusion Detection, Nathan Carey, The University of Queensland, 2000.
- [AB01] <http://www.psionic.com/abacus/>
- [NA00] <http://www.nabou.org/>, 2000.
- [Sun95] SunSHIELF Basic Security Module Guide, Sun Microsystems, November 1995.
- [Dan00] <http://home.twmi.rr.com/jayd/Audit.pdf>
- [CA01] <http://www.cai.com/products/ingr.htm>, Computer Associates, 2001.
- [SNARE01] <http://www.intersectalliance.com/projects/Snare/index.html>, Intersect Alliance, 2001.
- [SAW00] <http://www.flowerfire.com/sawmill/>, Flowerfire 2000.
- [El01] <http://www.jedi.claranet.fr/elio/tt/>
- [AXE00] S Axelsson et al. An approach to Unix security logging , Department of Computer Enginerring, Chalmers University of Technology, Sweeden.
- [TRIP01] <http://www.tripwire.org/qanda/faq.php#1>
- [STEF01] Aspects of modelling and performance of Intrusion detection system. Chalmers University of Technology, Sweedon, 2000.

APENDIX 1 Code Listing

```
/**
 * analyse.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */
import java.io.*;
import java.util.*;

/**
 * This class contains the main method
 */
class analyse {

    /**
     * Main method. This method controls the system.
     * It initializes a controlling window
     */
    public static void main(String[] args) {
        ControllingWindow mainWindow = new ControllingWindow();
        mainWindow.setVisible(true);
    }
}

/**
 * analyseDay.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */
import java.io.*;
import java.util.*;

/**
 * This class creates a day object that represents
 * many sessions. Methods to implement queries are
 * provided
 */
class analyseDay {
    private int cnt;
    private int SessionCount;
    private Sessions [] theSession = new Sessions[10000];
    private String UID;
    /**
     * method analyseDay. This method initializes the system.
     * Reads in a list of sessions to be processed. The list
     * of sessions is stored in a file.
     * Each session is an object defined by the session class.
     * Each session consists of record objects defined in the record
     * class
     */
    public analyseDay(String fileName) {
        try {
            final      BufferedReader      DayInputStream      =      new
            BufferedReader(newFileReader(fileName));
            String currentLine = DayInputStream.readLine();
            SessionCount = 0;
        }
    }
}
```

Auditing and Event Correlation

```
        while (currentLine != null) {
            theSession[SessionCount] = new Sessions(currentLine);
            SessionCount++;
            currentLine = DayInputStream.readLine();
        }
        DayInputStream.close();
    } catch (IOException e) {
        System.out.println("Error in reading file");
    }
} // end constructor

/**
 * Number of records in all sessions that have the file
 * fileName in their records
 */
public int NumberOfRec(String FileName) {
    int i, TotalNumber = 0, ArrayPos = 0;
    int [] RecordsInSession = new int[SessionCount];
    for (cnt = 0; cnt < SessionCount; cnt++) {
        RecordsInSession[cnt] = theSession[cnt].MatchRecordNumber(FileName);
        TotalNumber += RecordsInSession[cnt];
    }
    return TotalNumber;
}

/**
 * Returns the number of records that have UserID = UID
 */
public int NumberOfRecUID(String UIDname) {
    int i, TotalNumber = 0, ArrayPos = 0;
    int [] RecordsInSessionUID = new int[SessionCount];
    System.out.println("In NumberOfRecUID");
    for (cnt = 0; cnt < SessionCount; cnt++) {
        System.out.println("Calling MatchRecordNumberUID");
        RecordsInSessionUID[cnt] = theSession[cnt].MatchRecordNumberUID(UIDname);
        System.out.println("Returning From MatchRecordNumberUID");
        TotalNumber += RecordsInSessionUID[cnt];
    }
    return TotalNumber;
}

/**
 * Returns the number of records that have ProcessID = PID
 */
public int NumberOfRecPID(String PIDname) {
    int i, TotalNumber = 0, ArrayPos = 0;
    int [] RecordsInSessionPID = new int[SessionCount];
    System.out.println("Heree");
    for (cnt = 0; cnt < SessionCount; cnt++) {
        RecordsInSessionPID[cnt] = theSession[cnt].MatchRecordNumberPID(PIDname);
        TotalNumber += RecordsInSessionPID[cnt];
    }
    return TotalNumber;
}

/**
 * Returns the number of records that have a time between LeftTime
 * LeftTime and RightTime.
 */
public int NumRecTwoTimes(TokenizeTime LeftTime, TokenizeTime RightTime) {
    int i, TotalNumber = 0, ArrayPos = 0;
    int [] RecordsBetweenTimes = new int[SessionCount];
    for (cnt = 0; cnt < SessionCount; cnt++) {
        RecordsBetweenTimes[cnt] = theSession[cnt].MatchRecordNumberTime(LeftTime,
            RightTime);
        TotalNumber += RecordsBetweenTimes[cnt];
    }
    return TotalNumber;
}
```

Auditing and Event Correlation

```
}

/**
 * Method hasFile. This method searches multiple sessions and returns
 * an array of record containing the passed file.
 */
public Records[] DayHasFile(String FileName) {
    int i,j,TotalNumber = 0, ArrayPos = 0;
    int RecordsInSession;
    /**
     * While the are still sessions to search, keep going
     */
    for (j = 0; j < SessionCount; j++) {
        RecordsInSession = theSession[j].MatchRecordNumber(FileName);
        TotalNumber += RecordsInSession;
    }

    Records [] RecordsToReturn = new Records[TotalNumber];
    for (j = 0; j < SessionCount; j++) {
        RecordsInSession = theSession[j].MatchRecordNumber(FileName);
        Records [] receivedRecords = new Records[RecordsInSession];
        for (cnt = 0; cnt < SessionCount; cnt++) {
            receivedRecords = theSession[cnt].sessionHasFile(FileName);
            for (i=0; i < receivedRecords.length; i++) {
                RecordsToReturn[ArrayPos] = receivedRecords[i];
                ArrayPos++;
            }
        }
    }
    return RecordsToReturn;
} // end hasFile method

/* Method DayHasPID. This method searches multiple sessions and returns
 * an array of record containing the passed PID.
 */
public Records[] DayHasPID(String PID) {
    int i,j,TotalNumber = 0, ArrayPos = 0;
    int RecordsInSession;
    /**
     * While the are still sessions to search, keep going
     */
    for (j = 0; j < SessionCount; j++) {
        RecordsInSession = theSession[j].MatchRecordNumberPID(PID);
        TotalNumber += RecordsInSession;
    }
    System.out.println("Total number of records is" + TotalNumber);
    Records [] RecordsToReturn = new Records[TotalNumber];
    for (j = 0; j < SessionCount; j++) {
        RecordsInSession = theSession[j].MatchRecordNumber(PID);
        System.out.println("Creating receivedRecords array " + RecordsInSession);
        Records [] receivedRecords = new Records[RecordsInSession];
        for (cnt = 0; cnt < SessionCount; cnt++) {
            receivedRecords = theSession[cnt].sessionHasPID(PID);
            for (i=0; i < receivedRecords.length; i++) {
                System.out.println(" " + ArrayPos);
                System.out.println(" " + i);
                RecordsToReturn[ArrayPos] = receivedRecords[i];
                System.out.println("Finsihed");
                ArrayPos++;
            }
        }
    }
    return RecordsToReturn;
} // end hasPID method

/**
 * Method fileAccessedByUID. This method searches all records in
 * all sessions for records with file accesses that contain the UID passed
```

Auditing and Event Correlation

```
* to the method.
*/
public Records[] fileAccessedByUID(String UID) {

    int i,j,TotalNumberUID = 0, ArrayPos = 0;
    int RecordsInSessionUID;
    /**
     * While the are still sessions to search, keep going
     */
    for (j = 0; j < SessionCount; j++) {
        RecordsInSessionUID = theSession[j].MatchRecordNumberUID(UID);
        TotalNumberUID += RecordsInSessionUID;
    }

    Records [] RecordsToReturnUID = new Records[TotalNumberUID];
    for (j = 0; j < SessionCount; j++) {
        RecordsInSessionUID = theSession[j].MatchRecordNumberUID(UID);
        Records [] receivedRecordsUID = new Records[RecordsInSessionUID];
        for (cnt = 0; cnt < SessionCount; cnt++) {
            receivedRecordsUID = theSession[cnt].sessionFileAccessedByUID(UID);
            for (i=0; i < receivedRecordsUID.length; i++) {
                RecordsToReturnUID[ArrayPos] = receivedRecordsUID[i];
                ArrayPos++;
            }
        }
    }
    return RecordsToReturnUID;
} //end fileAccessedByUID

/**
 * Method TimeBetweenSession. This method searches all records in
 * all sessions for records with file accesses that contain a time field
 * between T1 and T2.
 */
public Records[] TwoTimes(TokenizeTime LeftTime, TokenizeTime RightTime) {

    int i,j,TotalNumberTimes = 0, ArrayPos = 0;
    int NumRecordsTimes = 0;
    /**
     * While the are still sessions to search, keep going
     */
    for (j = 0; j < SessionCount; j++) {
        NumRecordsTimes = theSession[j]. MatchRecordNumberTime(LeftTime, RightTime);
        TotalNumberTimes += NumRecordsTimes;
    }

    Records [] RecordsToReturnTimes = new Records[TotalNumberTimes];
    for (j = 0; j < SessionCount; j++) {
        NumRecordsTimes = theSession[j]. MatchRecordNumberTime(LeftTime, RightTime);
        Records [] receivedRecordsBetweenTimes = new Records[NumRecordsTimes];
        for (cnt = 0; cnt < SessionCount; cnt++) {
            receivedRecordsBetweenTimes = theSession[cnt].sessionRecBetweenTime(LeftTime,
            RightTime);
            for (i=0; i < receivedRecordsBetweenTimes.length; i++) {
                RecordsToReturnTimes[ArrayPos] = receivedRecordsBetweenTimes[i];
                ArrayPos++;
            }
        }
    }
    return RecordsToReturnTimes;
} //end fileAccessedByUID
}

/**
 * ControllingWindow.java
 * Created 12 September, 2001.
 * Derived from code written by Dan Johnston
 */
```

Auditing and Event Correlation

```
* Written by Trevor Norvill
*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

/**
 * A window for controlling the operating of the analyse tool
 * This tool analyses correlated data produced by the bashBsmEliott.pl
 * perl script
 */
public class ControllingWindow extends JFrame {

    private analyseDay theDay; //object containing all sessions
    private String fileName; // the file containing the list of sessions
    private Records[] SearchResultsUID; //Search results for UID button
    private syslogRecords[] SearchResultsSyslog; //Search results for syslog
    private String SearchFile; // Search File for hasFile button
    private String SearchUID; // Search file for UID button
    private Records[] SearchResults;
    private String SearchTimeLeft; //First search time
    private String SearchTimeRight; //Second search time

    /**
     * Dimensions of the window
     */
    private static final int TOP_LEFT_X = 250;
    private static final int TOP_LEFT_Y = 100;
    private static final int HEIGHT = 400;
    private static final int WIDTH = 150;

    /**
     * Button for application
     */
    private JButton RecordBetweenTime;
    private JButton openButton;
    private JButton hasFileButton;
    private JButton UIDinRecordButton;
    private JButton hasPIDButton;
    private JButton SyslogActivityButton;
    private JButton SuActivityButton;

    /**
     * Construct a window with buttons to perform all needed operations
     */
    public ControllingWindow() {
        super("analyse");
        setBounds(TOP_LEFT_X, TOP_LEFT_Y, WIDTH, HEIGHT);
        final JPanel content = (JPanel) getContentPane();
        content.setLayout(new FlowLayout(FlowLayout.LEFT));

        /**
         * Set up the open button and action
         */
        openButton = new JButton( "Open" );
        openButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                fileName = JOptionPane.showInputDialog(null, "Enter the name of the file
                containing the list of sessions to be processed");
                if (fileName != null) {
                    fileName = fileName.trim();
                    if (!fileName.equals("")) {
                        theDay = new analyseDay(fileName);
                        enableActionButtons();
                    }
                }
            }
        });
    }
}
```

Auditing and Event Correlation

```
content.add(openButton);
/**
 * create button for hasFile
 */
UIDinRecordButton = new JButton("UID in Record");
UIDinRecordButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        int NumberOfRecords, i;
        String DisplayStringUID = "";
        SearchUID = JOptionPane.showInputDialog(null, "Please enter the name of the
        UID you wish to search for");
        if (SearchUID != null) {
            SearchUID = SearchUID.trim();
            NumberOfRecords = theDay.NumberOfRecUID(SearchUID);
            Records [] SearchResultsUID = new Records[NumberOfRecords];
            SearchResultsUID = theDay.fileAccessedByUID(SearchUID);
            for (i = 0; i < NumberOfRecords; i++) {
                DisplayStringUID = DisplayStringUID + SearchResultsUID[i].returnRecord();
            }
            System.out.println(DisplayStringUID);
            //JOptionPane.showMessageDialog(null,DisplayStringUID);
        }
    }
});
content.add(UIDinRecordButton);

/**
 * create button for hasFile
 */
hasFileButton = new JButton("Has File");
hasFileButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        int NumberOfRecords, i;
        String DisplayString = "";
        SearchFile = JOptionPane.showInputDialog(null, "Please enter the name of the
        file you wish to search for");
        if (SearchFile != null) {
            SearchFile = SearchFile.trim();
            NumberOfRecords = theDay.NumberOfRec(SearchFile);
            Records [] SearchResults = new Records[NumberOfRecords];
            SearchResults = theDay.DayHasFile(SearchFile);
            for (i = 0; i < NumberOfRecords; i++) {
                DisplayString = DisplayString + SearchResults[i].returnRecord();
            }
            System.out.println(DisplayString);
            //JOptionPane.showMessageDialog(null,DisplayString);
        }
    }
});
content.add(hasFileButton);

/**
 * create button for hasPID
 */
hasPIDButton = new JButton("Has PID");
hasPIDButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        int NumberOfRecords, i;
        String DisplayString = "";
        String PID = JOptionPane.showInputDialog(null, "Please enter the name of the
        PID you wish to search for");
        if (PID != null) {
            PID = PID.trim();
            NumberOfRecords = theDay.NumberOfRecPID(PID);
            System.out.println("Got HERE");
            Records [] SearchResults = new Records[NumberOfRecords];
            System.out.println("Got Here 2");
            SearchResults = theDay.DayHasPID(PID);
            for (i = 0; i < NumberOfRecords; i++) {
```

Auditing and Event Correlation

```
        DisplayString = DisplayString + SearchResults[i].returnRecord();
    }
    System.out.println(DisplayString);
    //JOptionPane.showMessageDialog(null,DisplayString);
}
}
});
content.add(hasPIDButton);

/**
 * create button for RecordBetween
 */
RecordBetweenTime = new JButton("Records Between Time");
RecordBetweenTime.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        int NumberOfRecords, i;
        String DisplayString = "", AnswerUID = "", AnswerFile = "", UIDdata = "",
        Filedata = "";
        SearchTimeLeft = JOptionPane.showInputDialog(null, "Please enter the start time
in the format HH:MM:SS");
        SearchTimeRight = JOptionPane.showInputDialog(null, "Please enter the finish t
ime in the form HH:MM:SS");
        AnswerUID = JOptionPane.showInputDialog(null, "Do you wish to refine this
search by UID[y/n]");
        AnswerFile = JOptionPane.showInputDialog(null, "Do you wish to refine this
search by PID[y/n]");
        if (AnswerUID.equals("y")) {
            UIDdata = JOptionPane.showInputDialog(null,"Please enter the UID you wish
to look for");
            UIDdata = UIDdata.trim();
        }
        if (AnswerFile.equals("y") ) {
            Filedata = JOptionPane.showInputDialog(null, "Please enter the file you
wish to look for");
            Filedata = Filedata.trim();
        }
        if ((SearchTimeLeft != null) && (SearchTimeRight != null)) {
            SearchTimeLeft = SearchTimeLeft.trim();
            SearchTimeRight = SearchTimeRight.trim();
            TokenizeTime TimeLeft = new TokenizeTime(SearchTimeLeft);
            TokenizeTime TimeRight = new TokenizeTime(SearchTimeRight);
            NumberOfRecords = theDay.NumRecTwoTimes(TimeLeft, TimeRight);
            Records [] SearchResults = new Records[NumberOfRecords];
            SearchResults = theDay.TwoTimes(TimeLeft, TimeRight);
            for (i = 0; i < NumberOfRecords; i++) {
                if (AnswerUID.equals("y") && (! AnswerFile.equals("y"))) {
                    if (SearchResults[i].hasUID(UIDdata)) {
                        DisplayString = DisplayString + SearchResults[i].returnRecord();
                    }
                }
                if (AnswerFile.equals("y") && (! AnswerUID.equals("y"))) {
                    if (SearchResults[i].hasFile(Filedata)) {
                        DisplayString = DisplayString + SearchResults[i].returnRecord();
                    }
                }
                if (AnswerUID.equals("y") && AnswerFile.equals("y")) {
                    if (SearchResults[i].hasUID(UIDdata) &&
                    SearchResults[i].hasFile(Filedata)) {
                        DisplayString = DisplayString + SearchResults[i].returnRecord();
                    }
                }
                if ((!AnswerUID.equals("y") && (!AnswerFile.equals("y"))) {
                    DisplayString = DisplayString + SearchResults[i].returnRecord();
                }
            }
        }
        System.out.println(DisplayString);
        //JOptionPane.showMessageDialog(null,DisplayString);
    }
}
```

Auditing and Event Correlation

```
    }
  });
  content.add(RecordBetweenTime);
  /**
   * create button for syslog Activity
   */
  SyslogActivityButton = new JButton("Syslog Activity");
  SyslogActivityButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      int NumberOfRecords, i;
      String DisplayString = "", SearchTimeLeft = "", SearchTimeRight = "",
      syslogFile = "messages";
      SearchTimeLeft = JOptionPane.showInputDialog(null, "Please enter the start
      time in the format HH:MM:SS");
      SearchTimeRight = JOptionPane.showInputDialog(null, "Please enter the finish
      time in the form HH:MM:SS");
      if ((SearchTimeLeft != null) && (SearchTimeRight != null)) {
        SearchTimeLeft = SearchTimeLeft.trim();
        SearchTimeRight = SearchTimeRight.trim();
        TokenizeTime TimeLeft = new TokenizeTime(SearchTimeLeft);
        TokenizeTime TimeRight = new TokenizeTime(SearchTimeRight);
        try {
          syslog theSyslog = new syslog();
          NumberOfRecords = theSyslog.NumRecSyslog(TimeLeft, TimeRight);
          syslogRecords [] SearchResults = new syslogRecords[NumberOfRecords];
          SearchResultsSyslog = theSyslog.syslogBetweenTimes(TimeLeft, TimeRight);
          System.out.println("Number OF Records is " + NumberOfRecords);
          for (i = 0; i < NumberOfRecords; i++) {
            DisplayString = DisplayString + SearchResultsSyslog[i].theData + "\n";
          }
          System.out.println(DisplayString);
          //JOptionPane.showMessageDialog(null,DisplayString);
        } catch (IOException w) {}
      }
    }
  });
  content.add(SyslogActivityButton);
  /**
   * create button for SU Activity. Looks for more than 5 Su
   * entries in specified time.
   */
  SuActivityButton = new JButton("SU Activity");
  SuActivityButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) {
      int NumberOfRecords, i;
      String DisplayString = "", SearchTimeLeft = "", SearchTimeRight = "",
      syslogFile = "messages";

      SearchTimeLeft = JOptionPane.showInputDialog(null, "Please enter the start
      time in the format HH:MM:SS");
      SearchTimeRight = JOptionPane.showInputDialog(null, "Please enter the finish
      time in the form HH:MM:SS");
      if ((SearchTimeLeft != null) && (SearchTimeRight != null)) {
        SearchTimeLeft = SearchTimeLeft.trim();
        SearchTimeRight = SearchTimeRight.trim();
        TokenizeTime TimeLeft = new TokenizeTime(SearchTimeLeft);
        TokenizeTime TimeRight = new TokenizeTime(SearchTimeRight);
        try {
          syslog theSyslog = new syslog();
          NumberOfRecords = theSyslog.NumRecSu(TimeLeft, TimeRight);
          syslogRecords [] SearchResults = new syslogRecords[NumberOfRecords];
          SearchResultsSyslog = theSyslog.SuBetweenTimes(TimeLeft, TimeRight);

          for (i = 0; i < NumberOfRecords; i++) {
            DisplayString = DisplayString + SearchResultsSyslog[i].theData + "\n";
          }
          System.out.println(DisplayString);
        }
      }
    }
  });
}
```

Auditing and Event Correlation

```
        //JOptionPane.showMessageDialog(null,DisplayString);
    } catch (IOException e) {}
    }
}
});
content.add(SuActionButton);

}

private void enableActionButtons () {
    hasFileButton.setEnabled(true);
}

}

/**
 * Records.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */

import java.io.*;
import java.util.*;

/**
 * Class Records. This class creates an object representing one record
 * in the consolidated BSM/BASH/ELIOTT log. Various methods are provided
 * to allow other objects to query records values and properties
 */
public class Records {
    private String theRecord;
    private String theUID;
    public String thePID;
    private String theEliottData;
    private int Hour, Min, Sec;
    TokenizeTime RecordsTime;

    /**
     * Constructor for this class. creates a record object given pid,
     * UID, recordString and Time
     */
    public Records (String GivenRecord, String UID, String PID, TokenizeTime theTime) {
        theRecord = GivenRecord;
        theUID = UID;
        thePID = PID;
        RecordsTime = theTime;
        theEliottData="q";
    } // End Constructor

    /**
     * Overloaded constructor Records. This creates a records given record
     * string, UID, PID, Eliott data and time
     */
    public Records(String GivenRecord, String UID, String PID, String EliottData,
    TokenizeTime theTime) {
        theRecord = GivenRecord;
        theUID = UID;
        thePID = PID;
        RecordsTime = theTime;
        theEliottData = EliottData;
    }

    /**
     * Returns true if this record object
     * contains UID
     */
    public boolean hasUID(String UID) {
        System.out.println("In hasUID");
    }
}
```

Auditing and Event Correlation

```
        if (theUID.equals(UID)) {
            return true;
        }else {
            return false;
        }
    }
}

/**
 * Returns ture if this record object contains file fileName
 */
public boolean hasFile(String fileName) {
    if (theEliottData.equals(fileName)) {
        return true;
    }else {
        return false;
    }
}

/**
 * Returns ture if this record object contains Process ID PID
 */
public boolean hasPID(String PID) {
    if (thePID.equals(PID)) {
        return true;
    }else {
        return false;
    }
}

/**
 * Returns the PID of the record
 */
public String returnPID() {
    return thePID;
}

/**
 * Returns ture if this record object contains a time between those specified
 */
public boolean BetweenTimes(TokenizeTime LeftTime, TokenizeTime RightTime) {
    TwoTimes compareTimes = new TwoTimes(LeftTime, RightTime, RecordsTime);
    return compareTimes.betweenTwoTimes();
}

/**
 * returns a string containing the record data for this object
 */
public String returnRecord() {
    return theRecord;
}
}

/**
 * Sessions.java
 * Created 12 September, 2001.
 * Written by Trevor Norvill
 */
import java.io.*;
import java.util.*;
```

Auditing and Event Correlation

```
/**
 * This class creates a session object consisting of many records.
 * It provides methods to query the contents of these sessions.
 */
public class Sessions {
    private int RecordNumber;
    private String RecordToBeCreated;
    private Records [] theRecords = new Records[10000];
    private String UID;
    private String PID;
    private String ElliottData;
    private boolean ElliottFlag;
    private int sessionLineNumber;
    TokenizeTime TheTime;

    /**
     * Constructor for class session. Constructs a session consisting of an array
     * of records. The session name is passed to the constructor as a String.
     * The constructor then breaks down then session into records and creates
     * record objects for each. This currently implements PID as the last PID of
     * the record. In future, it would be nice to have an array of PID for each
     * record because one record can have more than one PID.
     */
    public Sessions (String SessionFileName) throws IOException {
        boolean first_occurence = true;
        System.out.println(SessionFileName);
        final      BufferedReader      SessionInputStream      =      new      BufferedReader(new
        FileReader(SessionFileName));
        int sessionLineNumber = 0;
        RecordNumber = 0;
        boolean FirstTimeInRecord = true;
        String sessionLine = "";
        try {
            sessionLine = SessionInputStream.readLine();
        } catch (IOException z) {
            System.out.println("Can't read file");
        }
    }
    while ( ! sessionLine.equals("END")) {
        try {
            System.out.println(sessionLine);
            if (sessionLine == null) {SessionInputStream.readLine();
                break;
            }
            if ((sessionLine.equals("Record")) && (sessionLineNumber != 0)) {
                if (ElliottFlag == false) {
                    theRecords[RecordNumber] = new Records(RecordToBeCreated, UID, PID,
                    TheTime);
                    FirstTimeInRecord = true;
                } else {
                    theRecords[RecordNumber] = new Records(RecordToBeCreated,UID, PID,
                    ElliottData, TheTime);
                    FirstTimeInRecord = true;
                }
                ElliottData = "";
                RecordToBeCreated = "";
                RecordNumber++;
                first_occurence = true;
            }
            if (sessionLineNumber == 0) {
                RecordNumber = 0;
            }
            String tmp = "";
            tmp = "" + sessionLine.charAt(0);
            if (tmp.equals("[") && (first_occurence == true)) {
                ElliottFlag = true;
                StringTokenizer ElliottReader;
                ElliottReader = new StringTokenizer(sessionLine,"[");
            }
        }
    }
}
```

Auditing and Event Correlation

```

EliottReader.nextToken();
String EliottFileNameTmp = EliottReader.nextToken();
StringTokenizer EliottReader2 = new StringTokenizer(EliottFileNameTmp, "];");
String EliottFileName = EliottReader2.nextToken();
StringTokenizer EliottReader3 = new StringTokenizer(EliottFileName, ".");
EliottFileName = EliottReader3.nextToken();
EliottData = EliottData + EliottFileName;
first_occurence = false;
}
if (("" + sessionLine.charAt(0)).equals("h") && (" " +
sessionLine.charAt(1)).equals("e") && (" " + sessionLine.charAt(6)).equals(","))
{
    if (FirstTimeInRecord == true) {
        StringTokenizer timeToken = new StringTokenizer(sessionLine, ",");
        timeToken.nextToken();
        timeToken.nextToken();
        timeToken.nextToken();
        String TmpToken = timeToken.nextToken();
        StringTokenizer timeToken2 = new StringTokenizer(TmpToken);
        timeToken2.nextToken();
        timeToken2.nextToken();
        timeToken2.nextToken();
        String TotalTime = timeToken2.nextToken();
        TheTime = new TokenizeTime(TotalTime);
        FirstTimeInRecord = false;
    }
}
if (("" + sessionLine.charAt(0)).equals("s") && (" " +
sessionLine.charAt(1)).equals("u") && (" " + sessionLine.charAt(2)).equals("b")) {
    StringTokenizer reader;
    reader = new StringTokenizer(sessionLine, ",");
    reader.nextToken();
    UID = reader.nextToken();
    reader.nextToken();
    reader.nextToken();
    reader.nextToken();
    PID = reader.nextToken();
    System.out.println("UID is" + UID);
}
} catch (Exception b) {System.out.println("This exception");}
try {
    RecordToBeCreated = RecordToBeCreated + sessionLine + "\n";
    sessionLine = SessionInputStream.readLine();
    sessionLineNumber++;
    if (sessionLine.equals("END")) {
        System.out.println("END line");
    }
}
} catch (Exception p) { System.out.println("Found exception");
SessionInputStream.skip(1);
char [] arraychar = {};
}
} // End constructor for session.

/**
 * Searches session for a record containing a file with fileName. If
 * that file is found, return an array of records containing the records
 */
public Records[] sessionHasFile(String fileName) {
    int i, NumberInArray = 0, number = 0;
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasFile(fileName)) {
            NumberInArray++;
        }
    }
    Records [] RecordsReturn = new Records[NumberInArray];
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasFile(fileName)) {

```

Auditing and Event Correlation

```
        RecordsReturn[number] = theRecords[i];
        number++;
    }
}
return RecordsReturn;
}

/**
 * Returns all records that contain PID
 */
public Records[] sessionHasPID(String PID) {
    int i, NumberInArray = 0, number = 0;
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasPID(PID)) {
            NumberInArray++;
        }
    }
    Records [] RecordsReturn = new Records[NumberInArray];
    for (i = 0; i < RecordNumber; i++) {
        System.out.println(theRecords[i].returnPID());
        if (theRecords[i].hasPID(PID)) {
            RecordsReturn[number] = theRecords[i];
            number++;
        }
    }
    return RecordsReturn;
}

/**
 * returns an integer that represents the number of
 * records that have the fileName property fileName
 */
public int MatchRecordNumber(String fileName) {
    int i, NumberInArray = 0;
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasFile(fileName)) {
            NumberInArray++;
        }
    }
    return NumberInArray;
}

/**
 * Returns an integer representing the number of records with
 * a matching user id UID.
 */
public int MatchRecordNumberUID(String UID) {
    int i, NumberInArray = 0;
    System.out.println("In MatchRecordNumberUID [Session.java]");
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasUID(UID)) {
            NumberInArray++;
        }
    }
    return NumberInArray;
}

/**
 * returns an integer representing the number of records with
 * process ID PID.
 */
public int MatchRecordNumberPID(String PID) {
    int i, NumberInArray = 0;
    System.out.println("In MatchRecordNumberPID [Session.java]");
    for (i = 0; i < RecordNumber; i++) {
        if (theRecords[i].hasPID(PID)) {
            NumberInArray++;
        }
    }
}
```

Auditing and Event Correlation

```
        System.out.println("The number of returns is" + NumberInArray);
        return NumberInArray;
    }

    /**
     * Returns an integer representing the number of records that have
     * a time between LeftTime and RightTime
     */
    public int MatchRecordNumberTime(TokenizeTime LeftTime, TokenizeTime RightTime) {
        int i, NumberInArray = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].BetweenTimes(LeftTime, RightTime)) {
                NumberInArray++;
            }
        }
        return NumberInArray;
    }

    /**
     * This method searches the session looking for records containing
     * the UID passed to the method. It returns an array of records
     * that fit the above requirement
     */
    public Records[] sessionFileAccessedByUID (String UID) {
        int i, NumberInArray = 0, number = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].hasUID(UID)) {
                NumberInArray++;
            }
        }
        Records [] RecordsReturn = new Records[NumberInArray];
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].hasUID(UID)) {
                RecordsReturn[number] = theRecords[i];
                number++;
            }
        }
        return RecordsReturn;
    }

    /**
     * This method searches the session looking for records containing
     * a time that is between the times passed to the method. It returns an array of
     * records that fit the above requirement
     */
    public Records[] sessionRecBetweenTime(TokenizeTime LeftTime, TokenizeTime RightTime) {
        int i, NumberInArray = 0, number = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].BetweenTimes(LeftTime, RightTime)) {
                NumberInArray++;
            }
        }
        Records [] RecordsReturn = new Records[NumberInArray];
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].BetweenTimes(LeftTime, RightTime)) {
                RecordsReturn[number] = theRecords[i];
                number++;
            }
        }
        return RecordsReturn;
    }
}

/**
 * syslog.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */
```

Auditing and Event Correlation

```
*/
import java.io.*;

/**
 * This class creates an object representing a syslog record.
 * It provides methods to query the syslogRecords within it.
 */
public class syslog {

    private syslogRecords [] theRecords = new syslogRecords[10000];
    private int RecordNumber;

    /**
     * constructs a syslog log in memory.
     */
    public syslog() throws IOException {
        int cnt = 0;
        String syslogLine = "";
        String syslogFile = "messages";
        final BufferedReader syslogInputFileStream = new BufferedReader(new
        FileReader(syslogFile));
        syslogLine = syslogInputFileStream.readLine();
        while (syslogLine != null) {
            System.out.println("Cnt is" + cnt);
            //System.out.println(syslogLine);
            theRecords[cnt] = new syslogRecords(syslogLine);
            System.out.println(theRecords[cnt].theData);
            syslogLine = syslogInputFileStream.readLine();
            cnt++;
            RecordNumber++;
        }
    } //syslog constructor.

    /**
     * Method NumRecSyslog. This method returns the number of
     * records between TimeLeft and TimeRight.
     */
    public int NumRecSyslog(TokenizeTime LeftTime, TokenizeTime RightTime) {
        int i, NumberInArray = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntryBetweenTimes(LeftTime, RightTime)) {
                NumberInArray++;
            }
        }
        return NumberInArray;
    }

    /**
     * This method searches all available syslog entries and finds out if
     * if they are between time T1 and T2. Return all records that meet
     * that criteria.
     */
    public syslogRecords[] syslogBetweenTimes(TokenizeTime LeftTime, TokenizeTime
    RightTime) {
        int i, NumberInArray = 0, number = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntryBetweenTimes(LeftTime, RightTime)) {
                NumberInArray++;
            }
        }
        syslogRecords [] RecordsReturn = new syslogRecords[NumberInArray];
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntryBetweenTimes(LeftTime, RightTime)) {
                System.out.println("Storing" + theRecords[i].RecordString() + "indexed by " + i);
                RecordsReturn[number] = theRecords[i];
                number++;
            }
        }
        return RecordsReturn;
    }
}
```

Auditing and Event Correlation

```
    }

    /**
     * This method returns the number of Su records where there are
     * more than 5 su records in the time specified
     */
    public int NumRecSu(TokenizeTime Left, TokenizeTime Right) {
        int i, NumberInArray = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntriesShowingSu(Left, Right)) {
                NumberInArray++;
            }
        }
        if (NumberInArray < 5) {
            NumberInArray = 0;
        }
        return NumberInArray;
    }

    public syslogRecords[] SuBetweenTimes(TokenizeTime LeftTime, TokenizeTime RightTime) {
        int i, NumberInArray = 0, number = 0;
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntriesShowingSu(LeftTime, RightTime)) {
                NumberInArray++;
            }
        }
        if (NumberInArray < 5) {
            NumberInArray = 0;
        }
        syslogRecords [] RecordsReturn = new syslogRecords[NumberInArray];
        for (i = 0; i < RecordNumber; i++) {
            if (theRecords[i].EntriesShowingSu(LeftTime, RightTime) && (NumberInArray >= 5)) {
                System.out.println("Storing" + theRecords[i].RecordString() + "indexed by " + i);
                RecordsReturn[number] = theRecords[i];
                number++;
            }
        }
        return RecordsReturn;
    }
}

}
```

```
/**
 * syslogRecords.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */
import java.io.*;
import java.util.*;

/**
 * This class creates an object representing a one line record
 * of the syslog log. It provides methods to query the contents
 * of this log
 */
public class syslogRecords {

    private String theRecord;
    private TokenizeTime theTime;
    public String theData;

    public syslogRecords(String theRecord) {
        StringTokenizer LineReader;
        LineReader = new StringTokenizer(theRecord, " ");
        LineReader.nextToken();
        LineReader.nextToken();
    }
}
```

Auditing and Event Correlation

```
String timeString = LineReader.nextTokent();
theTime = new TokenizeTime(timeString);
theData = theRecord;
} //constructor for syslogRecords

/**
 * Method EntryBetweenTimes. This method accepts two times and returns
 * true or false depending on the result of betweenTwoTimes.
 */
public boolean EntryBetweenTimes(TokenizeTime Left, TokenizeTime Right) {
    TwoTimes compareTimes;
    compareTimes = new TwoTimes(Left,Right,theTime);
    return compareTimes.betweenTwoTimes();
}

/**
 * This method determines if the records is an Su records
 * and returns true or false depending on the outcome
 */
public boolean EntriesShowingSu(TokenizeTime LeftTime, TokenizeTime RightTime) {
    boolean returnValue;
    TwoTimes compareTimes;
    compareTimes = new TwoTimes(LeftTime ,RightTime,theTime);
    if (compareTimes.betweenTwoTimes()) {
        //Check to see if the record is an su entry
        StringTokenizer SuReader;
        SuReader = new StringTokenizer(theData," ");
        SuReader.nextToken();
        SuReader.nextToken();
        SuReader.nextToken();
        SuReader.nextToken();
        String possibleSu = SuReader.nextToken();
        if ((" + possibleSu.charAt(0)).equals("s") && (" + possibleSu.charAt(1)).equals("u"))
        {
            returnValue = true;
        } else {
            returnValue = false;
        }
    } else {
        returnValue = false;
    }
    return returnValue;
}

/**
 * returns the records sting
 */
public String RecordString() {
    return theRecord;
}
}

/**
 * TokenizeTime.java
 * Created 12 September, 2001.
 * Written by Trevor Norvill
 */

import java.io.*;
import java.util.*;

/**
 * This class represents a time value. It provides methods
 * to extract Hour, Min and second.
 */
public class TokenizeTime {
    private String HourStringValue;
```

Auditing and Event Correlation

```
private int HourIntValue;
private String MinuteStringValue;
private int MinuteIntValue;
private String SecStringValue;
private int SecIntValue;
public TokenizeTime(String Time) {
    StringTokenizer TimeToken;
    TimeToken = new StringTokenizer(Time,":");
    HourStringValue = TimeToken.nextToken();
    MinuteStringValue = TimeToken.nextToken();
    SecStringValue = TimeToken.nextToken();
    HourIntValue = Integer.parseInt(HourStringValue);
    MinuteIntValue = Integer.parseInt(MinuteStringValue);
    SecIntValue = Integer.parseInt(SecStringValue);
}
public int Return_Hour() {
    return HourIntValue;
}
public int Return_Min() {
    return MinuteIntValue;
}
public int Return_Sec() {
    return SecIntValue;
}
}
/**
 * TwoTimes.java
 * Created 12 Septemer, 2001.
 * Written by Trevor Norvill
 */
import java.io.*;

/**
 * This is a class that forms an object consisting of two TokenizeTimes. It purpose
 * is to compare provide a method to these objects for comparing the two times with
 * a specified time. It is used by syslogRecord.java and Record.java
 */
public class TwoTimes {

    private TokenizeTime LeftTime;
    private TokenizeTime RightTime;
    private int Hour, Min, Sec;
    private TokenizeTime RecordTime;

    /**
     * Contructor TwoTimes. This constructor creates a Left and Right TokenizeTimes
     * instance variables. It also uses the Return_ methods to extra Hour, Min and Sec
     * from the time to be tested.
     */
    public TwoTimes (TokenizeTime Left, TokenizeTime Right, TokenizeTime theTime) {
LeftTime = Left;
        RightTime = Right;
        Hour = theTime.Return_Hour();
        Min = theTime.Return_Min();
        Sec = theTime.Return_Sec();
    } // End Constructor

    /**
     * This method compares a times against two boundary conditions. It returns true
     * the time exists in the middle of the two times else it returns false
     */
    public boolean betweenTwoTimes() {
        int testHourL = LeftTime.Return_Hour();
        int testMinL = LeftTime.Return_Min();
        int testSecL = LeftTime.Return_Sec();
    }
}
```

Auditing and Event Correlation

```
int testHourR = RightTime.Return_Hour();
int testMinR = RightTime.Return_Min();
int testSecR = RightTime.Return_Sec();
boolean returnValue = false;
if ((testHourL < Hour) && (testHourR > Hour)) {
    returnValue = true;
} else {
    if ((testHourL == Hour) || (testHourR == Hour)) {
    if ((testHourL == Hour) && (testHourR != Hour)) {
    if (testMinL < Min) {
        returnValue = true;
    } else {
        if (testMinL == Min) {
            if (testSecL <= Sec) {
                returnValue = true;
            } else {
                returnValue = false;
            }
        } else {
            returnValue = false;
        }
    }
    }
}

    if ((testHourL != Hour) && (testHourR == Hour)) {
    if (testMinR > Min) {
        returnValue = true;
    } else {
        if (testMinR == Min) {
            if (testSecR >= Sec) {
                returnValue = true;
            } else {
                returnValue = false;
            }
        } else {
            returnValue = false;
        }
    }
}

    if ((testHourR == Hour) && (testHourL == Hour)) {
    if ((testMinL < Min) && (testMinR > Min)) {
returnValue = true;
    } else {
        if ((testMinL == Min) && (testMinR == Min)) {
            if ((testSecL <= Sec) && (testSecR >= Sec)) {
                returnValue = true;
            } else {
                returnValue = false;
            }
        }
    }
}
}
} else {
    returnValue = false;
}
}
return returnValue;
}
}

use Getopt::Std;
use English;
#####
```

Auditing and Event Correlation

```
#####
# Title: bashBsmEliott.pl
# Programmer: Trevor Norvill
# Created: 23/6/01
# Discription: This is a script to merge the Linux Port of the Solaris Basic
# Security Module log files with individual .bash_history log
# file and the Elliott file access log.
#####
#####

#####
# At the moment this will not handle pipes
#####

#####
# Open the .bash_history file for input and get command line args.
#####
getopt('ubn');
open BASHFILE, "$opt_b" or die "could not open $opt_b($OS_ERROR)\n";
open BASHHISTTOKEN, "+<bash_history_token" or die "Could not open file for
writing($OS_ERROR)\n";
open BSMDATA, "bsmData" or die "Could not open bsmData file($OS_ERROR)\n";

# #####
# This is a function that formats the .bash_history file. It removes
# the fist tokens from every entry and places them in the BASHHISTTOKEN
# file for later processing. It also sets up an array with all commands
# that constitute valid bash commands. Each .bash_history command is then
# assigned a value of valid or possibly not valid. This is stored in another
# array.
#####
sub SetupHist {
    $i = 0;
    $count = 0;
    while (<BASHFILE>)
    {
        $basharray[$count] = $ARG;
        print "bash array", $basharray[$count];
        s/((\w+) [^\s]*)/$2\n/;
        print BASHHISTTOKEN;
        ++$count;
    }
    #
    # Close and reopen the list of bash history 1st tokens
    #
    close BASHHISTTOKEN;
    open BASHHISTTOKEN, "+<bash_history_token" or die "Could not open file
bash_history_token";
    $i = 0;
    while (<BASHHISTTOKEN>)
    {
        $first_tokens[$i] = $ARG;
        ++$i;
    }
    #
    # Read valid bash cmds into array valid_bash_cmd from end of file
    #
    $list_num = 0;
    while(<DATA>)
    {
        $valid_bash_cmd[$list_num] = $ARG;
        ++$list_num;
    }
    #
    # For each command in the bash history, assign valid or maybe not valid to each
```

Auditing and Event Correlation

```
# command based on the data sections valid bash commands.
#
$cmd_counter = 0;
foreach $cmd (@first_tokens)
{
    $j = 0;
    while (($cmd ne $valid_bash_cmd[$j]) and ($j <= ($#valid_bash_cmd)))
    {
        ++$j;
    }
    if ($j <= $#valid_bash_cmd) {
        @cmd_status[$cmd_counter] = "Valid";
    }
    else {
        @cmd_status[$cmd_counter] = "Maybe_not_Valid";
        print $cmd, $cmd_status[$cmd_counter], "\n";
    }
}
++$cmd_counter;
}

#####
# Setup a hash table with mapping of templates of bsm->bash
#####

#
# Take a record at a time
#
$INPUT_RECORD_SEPARATOR = '';
open ASSUMEDMAP, "assumedMAP.map" or die "Could not open file assumedMAP.map\n";
$count = 0;
while (<ASSUMEDMAP>)
{
    chomp;
    $map{$valid_bash_cmd[$count]} = $ARG;
    ++$count;
}
}

#####
# Seperate BSM into uid groups
#####
sub Seperate
{
    $INPUT_RECORD_SEPARATOR = '';
    print "Separating records";
    $count = 0;
    while (<BSMDATA>)
    {
        /(.*\n(.*)\n(.*)\n(.*)\n(.*)\n(subject,(\\d+),\\d+,\\d+,\\d+,\\d+)\n(.*)/;
        $uid[$count] = $7;
        ++$count;
    }
    $count = 0;
    close BSMDATA;
}

#####
# Setup the BSM for correlation. This requires the following,
# Group entries by pid
# Group entries by time
# Mark groups with common PID and time
# Mark any groups with first entry as /sh as invalid
#####
sub SetupBSM
{
    #
    # Seperate bsm log into uid groups
    #
}
```

Auditing and Event Correlation

```
Seperate();

#
# Out put entry with uid x to BSMx. Do this for all
# entries.
#
open BSMDATA,"bsmData";
while (<BSMDATA>)
{
  if (exists $knownuid{$uid[$count]})
  {
    open FILEH, ">>$knownuid{$uid[$count]}";
    print FILEH;
    close FILEH;

  }
  else
  {
    $knownuid{$uid[$count]} = "BSM${uid[$count]}";
    open FILEH, ">$knownuid{$uid[$count]}";
    print FILEH;
    close FILEH;

  }
  ++$count
}
#
# Seperate out PID from each BSM entry
#
$count = 0;

$SEARCHuid = $opt_u;
# hardwire to BSM0 file for the moment
close BSMDATA;
open BSMDATA, "BSM${SEARCHuid}";
while (<BSMDATA>)
{

  $bsmEntry[$count] = $ARG;

#/(.*)\n(.*)\n(.*)\n(header,284,[^,]*,([^\`]*))\n(.*)\n(subject,\d+,\d+,\d+,\d+,( \d+))\n(
.*)/;

/(.*)\n(.*)\n(.*)\n((header,284,[^,]*,([^\`]*))\n(.*)\n(subject,\d+,\d+,\d+,\d+,( \d+))\n
(.*)/;

  #was $9 and $6 $5$7
  $pathTokens[$count] = $7;
  $pids[$count] = $9;
  $times[$count] = $6;
  if ($5 eq "header\,284\,chdir()\,") {
    $formattedBSM[$count] = "$5";
  } else {
    $formattedBSM[$count] = "$5\n$7";
  }
  ++$count;
}

#
# Group according to PD. Ignore last 2 record because they
# contain no data. Produce an array containing the number of
# entries in a record. A record is a group of entries and an entry
# is one entry in the bsm data.
#
$RECORD = 0;
$loopcnt = 0;
while ($loopcnt <= ($#pids - 2))
{
```

Auditing and Event Correlation

```
if ($pids[$loopcnt] == $pids[$loopcnt+1])
{
    $loop = $loopcnt;
    $setflag = 0;
    while (($pids[$loop] == $pids[$loop + 1]) and ($loop <= ($#pids - 3)))
    {
#
# Chdir are not allocated new pid when 2 consecutive chdir occur
#
if ($formattedBSM[$loop] eq "header\,284\,chdir()\," and $formattedBSM[$loop + 1] eq
"header\,284\,chdir()\,") {
    $setflag = 1;
    break;
}
        ++$loop;
    }
    if ($setflag == 1) {
        --$loop;
    }
    $group[$RECORD] = ($loop - $loopcnt + 1);
    $loopcnt = ($loop + 1);
}
else
{
    $group[$RECORD] = 1;
    ++$loopcnt
}
++$RECORD
}
$cnt = 0;
while ($cnt <= $#group)
{
    ++$cnt
}

#
# Next, analysis time field in BSM output to find common time related
# groups. This will indicate possible relationships between the entries
# and help to classify them as valid or invalid entries.
#

#
# The following strips out the time fields from each BSM entry
# and places each time field in its own array.
#
$cnt = 0;
while ($cnt <= ($#times - 2))
{
    $times[$cnt] =~ /(\w+) (\w+) (\d+) (\d+):(\d+):(\d+) (\d+) ([^`]*)/;
    $month[$cnt] = $2;
    $day[$cnt] = $3;
    $hour[$cnt] = $4;
    $min[$cnt] = $5;
    $sec[$cnt] = $6;
    $year[$cnt] = $7;
    $frac[$cnt] = $8;
    $frac[$cnt] =~ s/\+([`]*)/$1/;
    ++$cnt;
}
$i = 0;
$cnt = 0;
$loop = 0;
$eliottCnt = 0;
while ($cnt <= ($#times - 2))
{
    if (($year[$cnt] == $year[$cnt + 1]) and ($month[$cnt] == $month[$cnt + 1])){
        if (($day[$cnt] == $day[$cnt + 1]) and ($hour[$cnt] == $hour[$cnt + 1])) {
            if ($min[$cnt] == $min[$cnt + 1]) {
                $loop = $cnt + 1;
            }
        }
    }
}
```


Auditing and Event Correlation

```
$cnter = 0;
while ($cnter <= $#times) {
    ++$cnter;
}
$i = 0;
$cnt = 0;
$pidValue = 0;
$nextTime = 0;
$GroupSel = 0;
while ($cnt <= $#group)
{
    $pidValue = $group[$cnt];
    $totalpid += $group[$cnt];
    ++$cnt;
    $nextTime = $time_record[$i];
    $totaltime += $time_record[$i];
    ++$i;
    while ($pidValue < $nextTime)
    {
        $pidValue += $group[$cnt];
        $totalpid += $group[$cnt];
        ++$cnt;
        $flag = pid;
    }
    while ($nextTime < $pidValue) {
        print $cnt;
        $nextTime += $time_record[$i];
        $totaltime += $time_record[$i];
        ++$i;
        $flag = time;
    }
    while ($totaltime != $totalpid)
    {
        if ($totaltime < $totalpid) {
            $finalGroup[$GroupSel] = $pidValue - ($totalpid - $totaltime);
            $group[( $cnt - 1)] -= ($group[( $cnt - 1)] - ($totalpid - $totaltime));
            $totalpid -= ($totalpid - $totaltime);
            $cnt = $cnt - 1;
            $flag = false;
        }
        if ($totalpid < $totaltime) {
            $finalGroup[$GroupSel] = $nextTime - ($totaltime - $totalpid);
            $time_record[( $i - 1)] -= ($time_record[( $i - 1)] - ($totaltime - $totalpid));
            $totaltime -= ($totaltime - $totalpid);
            $i = $i - 1;
            $flag = false;
        }
    }
}

if ($nextTime == $pidValue) {
    $finalGroup[$GroupSel] = $pidValue;
}
elseif ($flag == pid) {
    $finalGroup[$GroupSel] = $pidValue;
}
elseif ($flag == time) {
    $finalGroup[$GroupSel] = $nextTime;
}
++$GroupSel
}
#
# The following code concatenates the eliott log file with the BSM where appropriate
# ElliottMapping[$entry] contains the starting position of eliott output for BSM record
$entry.
# $entry. IF ElliottMapping[$entry] != 50000, this means that there is a valid eliott
# map for that entry. Since multiple eliott entries can map to one BSM entry, the
# eliott output to groupedBSM which is later used to produce the main output log file.
```

Auditing and Event Correlation

```
# while loop iterates appending the
#
$EliottBsmEntry = 0;
$i = 0;
while ($EliottBsmEntry < $#bsmEntry) {
  if ($EliottMapping[$EliottBsmEntry] != 50000) {
    $j=0;
    print "j is $j and eliottMapping is $EliottMapping[$EliottBsmEntry] and
    EliottBsmEntry is $EliottBsmEntry EliottAtNumberEntires is
    $EliottEntriesAtNumber{$EliottMapping[$EliottBsmEntry]} \n";
    $EliottPosition = $EliottMapping[$EliottBsmEntry];
    while ($j < $EliottEntriesAtNumber{$EliottMapping[$EliottBsmEntry]}){
      #print "Adding $eliott_entry[$EliottPosition + $j -1] to groupedBSM at Entry
$EliottBsmEntry ";
      $bsmEntry[$EliottBsmEntry] .= "$eliott_entry[($EliottPosition + $j)]";
      ++$j;
    }
    $EliottPosition += $EliottMapping[$EliottBsmEntry];
  }
  ++$EliottBsmEntry;
}

$cnt = 0;
$entry = 0;
$EliottPosition = 0;
while ($cnt <= $#finalGroup)
{
  $counter = $finalGroup[$cnt];
  while ($counter > 0)
  {
    $groupedBSM[$cnt] .= "$bsmEntry[$entry]\n";
    $formattedGroupBSM[$cnt] .= "$formattedBSM[$entry]\n";
    substr($formattedGroupBSM[$cnt], -1) = "";
    $pathTokensGrouped[$cnt] .= "$pathTokens[$entry]\n";
    --$counter;
    ++$entry
  }
  ++$cnt
}

#
# Subroutine setupEliott. This function pre processes the eliott log file
# and extracts the data needed later on in the script such as time etc.
#
sub SetupEliott
{
  open ELIOTT, "Eliott_file" or die "Could not open file for writing($OS_ERROR)\n";
  $total = 0;
  while (<ELIOTT>) {

    $eliott_entry[$i] = $ARG;
    #/\[(\d+):(\d+):(\d+)\](.*)\n/;
    /\[(\d+):(\d+):(\d+)\](.*)\((\d+),(\d+)\)(.*)\n/;
    $UidEliott = $5;
    if ($UidEliott == $SEARCHuid) {
      $EliottHour[$i] = $1;
      $EliottMin[$i] = $2;
      $EliottSec[$i] = $3;
      ++$i;
    }
  }
  $i = 0;
  $Cnt = 0;

  while ($Cnt <= $#eliott_entry) {
```

Auditing and Event Correlation

```

    if (($EliottHour[$Cnt] == $EliottHour[$Cnt +1]) and ($EliottMin[$Cnt] ==
    $EliottMin[$Cnt +1]) and ($EliottSec[$Cnt] == $EliottSec[$Cnt + 1])) {
    $loop = $Cnt +1;
    while (($EliottHour[$Cnt] == $EliottHour[$loop]) and ($EliottMin[$Cnt] ==
    $EliottMin[$loop]) and ($EliottSec[$Cnt] == $EliottSec[$loop]) and (($loop) <
    $#eliott_entry))
    {
        ++$loop;
    }
    $eliottTimes[$i] = $loop - $Cnt;
    $Cnt = $Cnt + ($loop - $Cnt);
}
else {
    $eliottTimes[$i] = 1;
    ++$Cnt;
}
$EliottEntriesAtNumber{$total} = $eliottTimes[$i];
$total += $eliottTimes[$i];

    ++$i;
}
}

#
# This subroutine matches the bash entries to its bsm entries.
#
sub DoMatch
{
    open BASHBSMELIOTT, "+>$opt_n" or die " could not open .bash_history($OS_ERROR)\n";
    $cnt = 0;
    $bashCount = 0;
    $place = 0;
    #
    #While there are still bash history entries, keep going
    #
    while ($bashCount < $#first_tokens) {
        $stry = 0;
        $valid = "true";
        $match = "false";
        $place = $cnt;
        while (($match eq "false") and ($valid eq "true"))
        {
            #
            # $cmd_status hold the value Valid or Maybe_not_valid
            # . If the command is a valid bash command, then use the bsm-bash
            # mapping to help find the correct entry.
            #
            if ($cmd_status[$bashCount] eq "Valid")
            {

                $testvar = "$map{${first_tokens[$bashCount]}} cmp "$formattedGroupBSM[$place]";
                if ($testvar == 0) {
                    print "Record\n", " ", $basharray[$bashCount], " ", "$groupedBSM[$place]",
                    "\n";
                    $FinalRecordArray[$bashCount] .= "Record\n";
                    $FinalRecordArray[$bashCount] .= $basharray[$bashCount];
                    $FinalRecordArray[$bashCount] .= $groupedBSM[$place];
                    $FinalRecordArray[$bashCount] .= "\n";
                    ++$bashCount;
                    $match = true;
                    ++$place;
                    $numberToInvalidate = 2;
                    $NumberBeforeMatch = $place - $cnt -1;
                    while ($NumberBeforeMatch > 0) {
                        $stest = "$map{${first_tokens[$bashCount] - 2}} cmp
                        "$formattedGroupBSM[$place - $NumberBeforeMatch - 1]";
                        if ($stest == 0) {
                            $lastBashEntry = 0;

```

Auditing and Event Correlation

```

$numberOfSameBash = 0;
$previous = 0;
while ($lastBashEntry == 0) {
    $compare = "$map{$first_tokens[$bashCount - 2 - $previous]}" cmp
"$map{$first_tokens[$bashCount - 3 - previous]}";
    if ($compare == 0) {
        $lastBashEntry = 0;
        ++$numberOfSameBash;
    } else {
        if ($previous == 0) {
            $numberOfSameBash = 1;
        }
        $lastBashEntry = 1;
    }
    ++$previous;
}
++$numberToInvalidate;

} else {
    # $InvalidArray[$bashCount] = 0;
}
--$NumberBeforeMatch;
if ($numberOfSameBash != numberToInvalidate) {
    $InvalidArray[$bashCount - 1] = $numberOfSameBash
}
}

$cnt = $place;
}

if (($try == 40) or ($try == ($#first_tokens - $bashCount))) {
    print BASHBSMELIOTT "Could not find match \n";
    $valid = "false";
    ++$bashCount;
    $try = 0;
}
++$try;
++$place;
}
elseif ($cmd_status[$bashCount] eq "Maybe_not_Valid")
{
    #
    # find out if command is a shell script or program being
    # run. look for a "." in the path. If it is after a /, then this is
    # a program.
    #
    @chars = unpack("A1" x length($pathTokensGrouped[$place]),
    $pathTokensGrouped[$place]);
    $slashFlag = 0;
    $times = 0;
    foreach $character (@chars) {
        if ($character eq ".") {
            $slashFlag = 1;
            $times = 1;
        }
    }
    if ($character eq "/" and $slashFlag == 1) {
        print "RECORD\n, $basharray[$bashCount], $groupedBSM[$place]\n";

        $FinalRecordArray[$bashCount] .= "Record\n";
        $FinalRecordArray[$bashCount] .= $basharray[$bashCount];
        $FinalRecordArray[$bashCount] .= $groupedBSM[$place];
        $FinalRecordArray[$bashCount] .= "\n";
    }
    if ($times >= 2) {
        $slashFlag = 0;
        $times = 0;
    }
}
}

```

Auditing and Event Correlation

```
    }
    ++$bashCount;
    ++$place;
    $cnt = $place;
    $match = "true";

    if ($try == 15) {
        #
        #assume the bash command was an invalid command
        #move onto next bash command
        #
        print "not a known command or program";
        $valid = "false";
        ++$bashCount;
        $try = 0;
    }
    ++$try;
}
else
{
    ++$bashCount;
    ++$try;
}
} #end elsif

} #end While loop
$j = 0;
$cnt = 0;
$around = 1;
$innerLoop = 1;
while ($cnt <= $#FinalRecordArray) {
    while (($j <= $#InvalidArray) and ($around == 1)) {
        if ($InvalidArray[$j] > 0) {
            $loop = 1;
            while ($InvalidArray[$j] > 0) {
                $validInvalid[$j - $loop] = 1;
                $validInvalid[$j] = 0;
                --$InvalidArray[$j];
                ++$loop;
            }
        } else {
            $validInvalid[$j] = 0;
        }
        ++$j;
    }
    $around = 0;
    $i = 0;

    while (($i <= $#validInvalid) and ($innerLoop == 1)) {
        print "The Value of validInvalid is $validInvalid[$i] \n";
        ++$i;
    }
    $innerLoop = 0;
    if ($validInvalid[$cnt] == 0) {
        print BASHBSMELIOTT $FinalRecordArray[$cnt];
    }
    ++$cnt;
}
print BASHBSMELIOTT "END";
close BASHBSMELIOTT;
} #End Subroutine DoMatch

#
# Function to clean up the output log file BASHBSMELIOTT
# So that java programm can read it
#
sub TidyUp {
    my $NamesOfSessionFile = "Record";
```

Auditing and Event Correlation

```
$INPUT_RECORD_SEPARATOR = "\n";
open SessionNamesFile, ">> $NamesOfSessionFile" or die "Could not open
$NamesOfSessionFile";
print SessionNamesFile "$opt_n";
}

#####
# Main Program
#####
SetupEliott();
SetupHist();
SetupBSM();
DoMatch();
TidyUp()
#####
# Data for telling valid commands apart from invalid commands
#####
__END__
chgrp
chmod
chown
cksum
cp
file
install
ln
lockfile
ls
lsattr
mv
pathchk
rm
sum
touch
fuser
killall
nohup
pidof
ps
pstree
renice
top
apropos
info
locate
makewhatis
man
whatis
whereis
cd
mkdir
pwd
rmdir
finger
groupmod
grpck
id
passwd
su
useradd
userdel
usermod
users
who
captainfo
clear
dumpkeys
getkeycodes
infocmp
```

Auditing and Event Correlation

login
setterm
stty
tic
tput
tset
badblocks
cfdisk
du
fdisk
quota
debugfs
df
dumpe2fs
fsck
fsck.minix
fuser
lsattr
mkfs
mount
rdev
swapoff
sync
umount
lpc
lpd
lpq
lpr
lprm
pr
tunelp
identd
klogd
lpd
rlogin
rwhod
rwho
syslogd
talk
tcpd
arch
hostname
hwclock
uname
depmod
insmod
ksyms
lsmod
modprobe
rmmod
cat
head
less
lock
tac
tail
cmp
comm
diff3
colrm
vi
column
cut
expand
fmt
fold
merge
paste
sort

Auditing and Event Correlation

```
split
tr
unexpand
uniq
colcrt
eqn
grog
tbl
troff
compress
shar
dmesg
init
lilo
rdev
runlevel
sync
telinit
X
xf86config
sleep
#xmseconfig
#xvidtune
#at
#atq
#atrm
#batch
#crontab
#usleep
#egrep
#fgrep
#find
#finger
#grep
#locate
#updatedd
#which
#df
#dmesg
#free
#ipcs
#ps
#pstree
#runlevel
#tload
#top
#vmstat
#who
#ipchains
#cal
#date
#ispell
#rpm
#arp
#bootpgw
#bootptab
#bootptest
#dip
#hostname
#ifconfig
#in.identd
#netstat
#nslookup
#rmail
#route
#routed
#rusers
#tcpdchk
#tcpdmatch
```

Auditing and Event Correlation

```
#traceroute
#ftp
#hostname
#netstat
#ping
#rcp
#rdate
#rdist
#rlogin
#rsh
#rwall
#telnet
#nisdomainname
#ypbind
#ypcat
#ypdomainname
#ypinit
#yppasswd
#yppoll
#yppush
#ypserv
#ypset
#ypwhich
#ypxfer
#mattrib
#mbadblocks
#mcd
#del
#mdeltree
#mdir
#mdu
#mformat
#mlabel\mmd
#mmove
#mrd
#mtype
#nmbd
#smbclient
#smbd
#smbmount
#smbstatus
#smbumount
#testparm
```

Auditing and Event Correlation

Auditing and Event Correlation