Preliminary report on Distributed ASAX

Abdelaziz Mounji   Baudouin Le Charlier   Denis Zampunieris
Naji Habra,
Institut D'Informatique,
FUNDP,
rue Grangagnage 21,
5000 Namur
E-mail: {amo, ble, dza, nha}@info.fundp.ac.be

May 27, 1994

Contents

.

Chapter 1

Introduction

The purpose of this report is to present a distributed on-line system capable of performing efficient, intelligent and network-level analysis of security audit trails in a network of SUN workstations. The distributed system is in fact an extension of ASAX ([1], [2], [3]) whose main features can be summarized by the following:

Universality: by supporting any kind of native audit trail format. This is achieved by means of user-provided Format Adaptors translating the native format to the NADF (Normalized Audit Data Format) format;

Power: by devoting a brand new rule-based language especially designed for formulating arbitrary complex queries on the audit trail;

Efficiency: by allowing to resolve multiple queries in a single pass analysis of the audit trail using a bottom-up method.

However, ASAX was originally designed for analyzing a single audit trail generated on a single machine.
In effect, the present and future integration of computer systems and networks suggest more elaborate analysis that should be carried out at the network level. This stems from the fact that (from security analysis standpoint at least) some pattern of actions may appear legitimate if considered at the host level but could reveal malicious behaviour if related to actions occurring on other hosts of the network. On the other hand, such a distributed system potentially provides a coherent and integrated control of the network auditing/analysis system.

The distributed analysis should be ideally applied to an heterogeneous network but in this paper, we suggest an experimental system restricted to a network of Sun workstations. However, the paper presents high level, user-oriented functionalities general enough to be applicable to an heterogeneous network. In this regard, the experimental distributed system will serve a twofold goal. First, it will demonstrate the usefulness of these functionalities in achieving network security monitoring and intrusion/anomaly detection. Second, it will prove the feasibility of these functionalities by implementing a functioning prototype version on a network of SUN workstations.

Distributed Analysis should not be restricted to security audit trails. In this first attempt, we concentrate on the security aspects but we hope that many design choices are applicable to the analysis of distributed systems in general.

The rest of the paper is organized as follows. The next chapter describes the functionalities a distributed system for audit trail analysis should provide. Chapter 3 introduces some basic features of the SunOS 4.1 security level C2. Chapter 4 depicts a general architecture of a distributed system for audit trail analysis and explains the interactions between the various components (machines, processes, ... etc). Chapter 5 describes the detailed functionalities implemented by the experimental system and shows how the user-oriented requirements of chapter 2 can all be fulfilled using the detailed functionalities. Chapter 6 is a specification of the Security Administrator command interface for the distributed system. Chapter 7 addresses the implementation design issue.

Chapter 2

Desirable functionalities for distributed network monitoring

2.1 Introduction

In this chapter we will examine the high level functionalities of a distributed on-line system for audit trail analysis. We claim that the provided functionalities are essential for the Security Administrator to accomplish his task in the most effective and convenient way.

2.2 Single Point Administration

In a network of computers and in the context of auditing/analysis of security related events, it is desirable that the security administrator could gain control over the auditing/analysis system of the whole network from a single machine. To be accomplished, none of his (security-related) duties should require multiple (physical or remote) login to numerous machines. Rather, he must be provided with an (command/graphical) interface which allows management and control of the auditing/analysis resources, programs and parameters in a transparent way. For instance, remote hosts should be accessed using logical names without need of multiple connections to remote hosts. This is especially useful in case the security administrator wants to apply the same administration tasks (set up auditing parameters such as audit directories, level of granularity, analysis, ... etc) on a group of hosts in the network.

That is, the single point administration requirement means that remote audited nodes should be considered as logical objects on which security administration operations could be applied as if they were directly available on the local machine.

2.3 Local Analysis (Filtering)

The distributed auditing/analysis system is able to perform analysis on any node taking part in the network of audited nodes. The analysis is considered local in the sense that the audit data subject to analysis represents events taking place at the audited host. No assumption is otherwise made about which node is actually performing the analysis.

Local analysis requires that the auditing mechanism is activated. It can be carried out in two different modes. In the *on-line* (or real-time) mode, analysis is applied to audit records as soon as they are generated by the auditing subsystem. In *off-line* mode, the system analyzes audit trails previously stored by the auditing subsystem. The outcome of a local analysis is a sequence of audit records from the local audit trail which satisfy the analysis. Local analysis is also called *filtering* since at the network level, it serves as a pre-selection of audit records. The various audit records obtained at each node will subsequently be subject to a more elaborate analysis.

2.4 Global Analysis

The distributed auditing/analysis system is able to perform more intelligent analysis of a global audit trail gathering audit records that result from the various local analyses. Global analysis aims at identifying correlations between events occurring at different nodes. The overall benefit is reducing the complexity of malicious behaviour detection. In effect, a malicious behaviour can be viewed as an elaborate sequence of actions involving many nodes. On the other hand, these actions could in turn be as complex rather than mere pre-selection of audit records on record type basis for instance.

As a result, concerted local filtering and global analysis leads to a more intelligent monitoring and intrusion/anomaly detection system for computer networks. Moreover, this combination yields a better balancing of the processing resources since the total amount of cpu time is distributed over the network.

If the current global analysis indicates that a security violation is suspected, it can trigger a finer granularity of generated events on certain nodes. It can also affect the current local analysis by asking such nodes to apply a more appropriate filter in order to capture more audit records.

In the following, the node performing the global analysis is called the *central* (or master) machine while filtering takes place at *slave* machines.
Note that the master machine can also analyze its own local audit trail and then be considered slave at the same time.

2.5 Availability

In the case where any host goes down, auditing must continue on any other machines and the global analysis of the rest must continue. If the central machine breaks down, analysis can be restarted from another machine and resumed from the time of the crash (i.e. the global analysis will still be completely performed albeit delayed).

2.6 Audit Trail Control

Appropriate audit trail control is driven by 2 factors:

Huge volume of audit data at the finest level of granularity, audit data for a single user can exceed 10 MB of data per day;

The network context audit data could span over numerous hosts and hence introduces the difficulty of coherent maintenance.

Consequently, audit trail control requirement should include:

At the host level common methods for reducing the required storage i.e:

- a tradeoff between a finer granularity (in order to achieve detailed analysis) and the resulting big amount of data;
- archiving policy to removable media;
- data compression.

At the network level since audit data is scattered over many hosts, a centralized management is necessary in order to avoid inconsistencies between audit trail controls applied at each host.

2.7 Security

Audit data must be protected from corruption and/or unauthorized access. The whole effort devoted to audit trail analysis is meaningless if appropriate access control to audit data is not enforced. Multi-user operating systems support access control mechanism to ordinary files that must be used for this purpose. Only a very few number of users (user **audit** in Unix systems) should be allowed access to audit trails.

Note that audit trails analysis could also be used as the last line of defense against unauthorized access to audit trails themselves.
Again, in a network audit context, audit record transmission between a slave machine and the master machine must be secured against users listening to the transmission media.

2.8 Multi Point Administration

It is desirable that the audit files could be queried by several persons with different interests and privileges. In this perspective, the system must be able to manage access control to audit trails at the record or at the audit data (audit record fields) level. For example, the accounting department executive should be able to access audit data relevant to accounting while he must be denied access to security related audit data. At the network level, different persons (accounting auditor, security officer, ... etc) can perform different kinds of analysis on different machines depending on their domain of investigation.

2.9 Portability

The distributed on-line analysis system should be highly independent of any machine or network architecture in order to allow monitoring/analysis of networks as heterogeneous as possible. Provisions should be made for most system and network architectures which can be supported with a minimum reprogramming effort.

2.10 Proposed solution

The solution that we describe in the rest of this paper provides a complete treatment of requirement 2.2, 2.3, 2.4 and 2.5 (Single Point Administration, Local Analysis, Global Analysis and Availability). Partial solutions are provided to requirements 2.6 and 2.7 (Audit Trail Control and Security). They amount to reusing mechanisms which are provided by the SunOS 4.1. Security level C2. Requirement 2.8 (Multiple Point Administration) is not addressed formally because its satisfactory treatment requires defining the notion of shared data with access privileges defined at the data field level. Nevertheless, our solution could be naturally extended to handle Multiple Point Administration. Requirement 2.9 (Portability) is addressed at the architecture and language level, but the implementation and the audit trail generation control are specific. However the implementation will isolate the system dependent parts in a few modules to allow "portability with minimum reprogramming".

Chapter 3

The SunOS 4.1 Security level C2

3.1 Installation

Under the security level C2, the SunOS 4.1 operating system provides auditing of security-related events in the so-called audit trails. For a host (namely a Sun workstation) to run in this security level, the `Security` option must be selected when installing SunOS. The system kernel must be configured with `SYSAUDIT` option and the machine must be rebooted.

3.2 Process audit state

Security-related events could be audited at different levels of granularity. For less granular auditing, an event can be considered atomic and represented by a single audit record while it could be decomposed into a sequence of events (represented by several audit records) in a more granular level of auditing. The *process audit state* for a given process determines the set of event types to be audited for it. The *system audit value* characterizes which events should be audited for an arbitrary process running on a host while a *user audit value* for a given user specifies what events should be audited for an arbitrary process initiated by this user on this host. As a result, the audit state of an arbitrary process is completely determined by the system audit value and the associated user audit value.

The system audit value is represented in the file *audit_control* while the user audit value is represented in the file *passwd.adjunct*. The latter file is an adjunct to the file /etc/passwd and contains encrypted passwords. Users are denied access to /etc/passwd.adjunct in order to avoid insider password attacks.

3.3 Naming conventions

Audit trails are stored in the directory `/etc/security/audit/`*server*`/files` where *server* is the name of the audited host. This directory is automatically created when the system is booted C2. Furthermore, some naming conventions are used for audit files: when auditing starts, a file designated $date_0$*.not_terminated* (where $date_0$ is made up of the year, month, day, hour minutes and seconds) is created. When this file is closed, its name is changed to $date_0.date_1$ (where $date_1$ is the date of the close operation), at which time, auditing continues on a the file named $date_1$*.not_terminated* and so on. The audit deamon writes audit records to the current file unless it receives a signal forcing it to switch to an other file. When audit trails quota is exceeded, the audit daemon (*auditd*) informs the security administrator of such event and suspends auditing until some arrangements are made; at that time, auditing can resume.

3.4 Command interface

The audit system interface *audit(8)* command can be used to

- signal audit daemon to close the current audit file and open a new audit file in the current audit directory;
- signal audit daemon to read audit control file. The audit daemon stores the information internally;
- signal audit daemon to disable auditing and die;
- change the process audit state of all processes owned by a user by changing his user audit value and/or the system audit value. These two values are contained in the file audit_control and passwd.adjunct respectively.

See *audit(8)* man page and *Administering C2 Security* of the *System and Network Administration* for more details.

Chapter 4

General Architecture

In order to understand clearly the architecture of our proposed solution, it is worth considering two different levels.: the local level (or filtering) and the global level. In the following, we outline each level in turn. The current chapter only aims at giving an intuitive view of the solution. The exact functionalities are detailed in chapter 5.

4.1 Filtering

The system is built on a network of Sun workstations. At the host level, there are 4 main components (see Figure 4.1) :

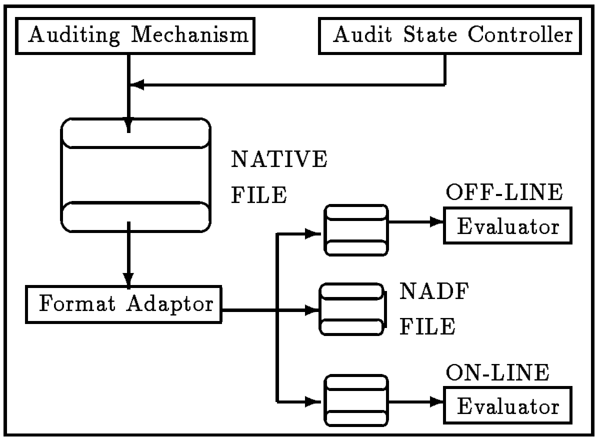

Figure 4.1: Filtering

Audit State Controller: it is able to alter the granularity level used by the auditing mechanism. This granularity can be controlled for all processes running on this host or on a user basis by controlling the granularity of processes belonging to that user. The **Auditing Mechanism** is the original mechanism provided by the SunOS 4.1 security level C2 and there is no possibility to modify it. It runs independently of our distributed system. The *audit state controller* is a separated process of our distributed system which receives commands from the central console (see 4.2) and sends messages to the auditing mechanism by means of the (SunOS security level C2) *audit(8)* interface;

Format Adaptor: this is the module responsible for translating the O.S. generated audit trail to a canonical format. The Format Adaptor is linked to the SunOS4.1 Auditing Mechanism in a pipeline fashion. The native file is erased automatically after translation. NADF files are kept until the auditor decides to erase them. Precise naming conventions are used for NADF files. They allow to select files on a "time of generation" basis.
Keeping converted files instead of native files has several advantages: the files are converted only once and can be reanalyzed several times without requiring a new conversion. Moreover, in the context of an heterogeneous network, they provide a standard and unique format;

Evaluator: performs analysis on the file (in canonical format) previously generated by the Format Adaptor. Note that several evaluators can be active at the same time. They perform different analysis on different NADF files or possibly on the same file. Off-line and on-line analyses are implemented in the same way. The only difference is that on-line evaluation applies to the currently generated NADF file.

Audit records resulting from local analysis on slave machines are sent to the central machine for further analysis.

4.2 Global Analysis

At the network level, the system consists of one central or master machine and one or more slave machines. Slave machines analyze their local audit trails and send the filtered audit records to the master machine which then performs a more intelligent analysis. Two processes are run on the central machine (see Figure 4.2):

Central Evaluator performs global analysis on a global audit trail containing the audit records resulting from local analysis and sent by the slave machines. The result of the central evaluation can be network wide reports, alarms and statistics;

Console: is an interactive command interface used by the Security Administrator to control the network monitoring system. It can be used to affect the level of auditing granularity on host or user basis. It can also be used to apply a given analysis on any host. More generally, the console is used by the auditor to specify any change to the system current behaviour. The set of available commands are described in chapter 6.

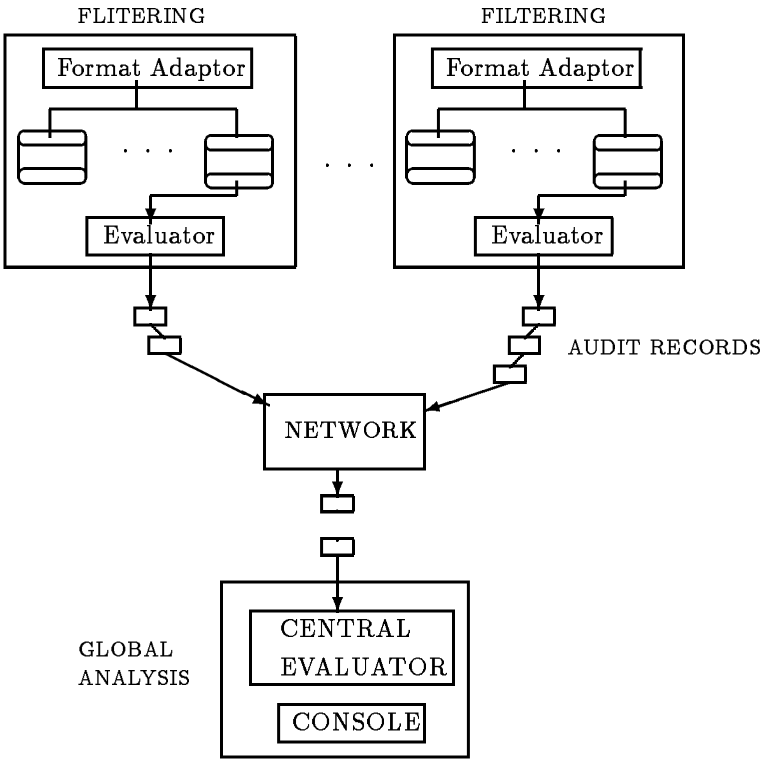

Figure 4.2: Global Analysis

Chapter 5

Detailed Functionalities

5.1 Introduction

In chapter 2 we described high level functionalities and user-oriented requirements. The present chapter deals in detail with the functionalities supported by the proposed solution of a distributed on-line system for audit trail analysis. It aims also to show how the following detailed functionalities address the user-oriented requirements.

5.2 Initialization

We distinguish 2 kinds of initialization:

- installation of the security feature by rebooting under the security level C2. Refer to *Administering C2 Security* of the *System and Network Administration* for more details;
- Initialization of the analysis system for all hosts attached to the network to be monitored.

At completion of the initialization procedure, the distributed system must be ready for starting network analysis. This initialization consists of the following steps:

- starting of the Format Adaptor module on each host so that all local native audit trails are translated to NADF files. This translation must be on-line;
- activation of an evaluator on each host so that it is ready for processing local NADF files;
- activation of the interactive console process on one of the hosts. This host is chosen by the Security Administrator and is considered to be the master machine.

At this point, and using the interactive console, the Security Administrator is able to launch any analysis he wants on any host. This is done by specifying a host name and a rule module that implement this analysis. Chapter 6 lists the command language used by the console. Note that the initialization procedure automatically starts the Format Adaptors and evaluators on the various hosts and then activates the Security Administrator console that prompts the Security Administrator for commands.

5.3 Audit Trail Control

Audit trail control involves elimination of native audit trails, login control, and management of audit trails. These points are described in the following:

- native audit trails must be systematically removed since they are redundant with the corresponding NADF files. When the native file reaches a given size (say 1MB), it is removed and the format adaptor resumes the translation of a new native file;

- naming conventions must be chosen so that audit trails are automatically assigned standard names as they are generated by the format adaptor. An NADF file name has the following general format:

  crtime_cltime.NADF

  where *crtime* is the date of the NADF file creation and *cltime* is the date at which the file was closed. If an NADF file is still been generated by the Format Adaptor, *cltime* is set to the conventional value *not_terminated.*

In order to allow easier management of NADF files, it is necessary to specify a maximum size that these files may grow before a new NADF file is created. Choosing a small value for this maximum size results in excessive file switches and leads to the proliferation of small files on the host. Conversely, choosing a too large value yields too large and unmanageable files especially for back-up and restore operations. The system uses a default value for an NADF file (say 1MB). However, this value can be set up to a different one at the initialization procedure if needed.

5.4 Login control

The login control requirement aims at tuning the auditing granularity level. Under the SunOS 4.1, this is achieved by setting up the system audit value and the user audit value for each user. It is desirable to allow a dynamic login control which can change the audit state of current processes either temporarily or permanently. Under SunOS level C2, there is a certain number of C library interface routines that can be used to read the audit control file and alter process audit states.

5.5 NADF file management

The NADF file management is left to the Security Administrator who bears the responsibility of choosing:

- a back-up/restore policy;
- secure the audit data contained in the NADF file by choosing the right access control available in the O.S;
- organize the NADF file system so that the NADF files are well classified.

5.6 Distributed Analysis

The Security Administrator is able to apply local analysis on any host attached to the audited network by specifying a rule module implementing such an analysis and the host (Sun workstation) this analysis must be applied to. He can also activate a central analysis on the central machine. The result of the central analysis could be reports, alarms or statistical measures on network, host and user basis.

In order to choose one or more NADF file, the Security Administrator must supply at the interactive console a time stamp or a time interval which will completely determine the sequence of NADF files to be analyzed by the local evaluators. If a time interval is specified, one or more NADF file is selected for analysis. This analysis terminates when it encounters the first record with a time stamp greater than the upper bound of this interval. Otherwise, if a time stamp is given, the analysis is applied to the NADF file whose name reflects the time interval the time stamp falls into. The analysis is started from the first record whose time stamp is greater or equal to the given time stamp. In this case, the analysis continues until the end of the last generated NADF file and then becomes on-line. When no time stamp is given, the analysis is performed on-line i.e, starting from the next record generated by the audit subsystem and converted to NADF format. This is the default option.

On the central machine side, the central evaluator analyzes only audit records incoming from the other slave machines. As mentioned before, there is (at least) two versions running on the central machine: the central (or master) version which performs the network level analysis and a slave version which performs a local analysis. These two versions differ only by the origin of the NADF records stream that is analyzed. However, the master evaluator makes no special distinction between audit records incoming from the local evaluator and the other audit records sent by the remote workstations.

The above functionalities clearly fulfill the high-level global and local analysis presented in chapter 2. On-line (local or global) analysis is realized by activating the desired rule module on the chosen host (slave or master). Off-line analysis (global or local) analysis is also fulfilled by specifying a time stamp or a time interval.

In addition, nothing precludes the possibility of running more than one evaluator process on a single workstation. Moreover, this lends itself to the very interesting possibility of having scalable network analysis by considering many different global analysis as local analysis that produces records for a second level global analysis and so on. This feature can be implemented without any special additional arrangements.

The availability requirement is also adequately addressed thanks to the NADF files produced by the Format Adaptors. On the one hand, suppose that a given slave workstation goes down. The rest of the slave machines as well as the central machine are not affected and the analysis (local and global) can continue. This analysis is not incomplete since no audit records are lost as long as the slave machine is down. Moreover, when it is up again, the format adaptor attached to it is restarted immediately to store audit records in NADF files. This host can subsequently resume analysis from the time just before the crash and the analysis is recovered.

On the other hand, if the master machine breaks down during analysis, any other (up) host can be chosen to perform the global analysis. This is achieved by stopping all the local analysis still running on the other hosts and resuming them from the time just before the central machine crashed. All the records sent by the slave machines to the central machine while the latter was down are not lost since they are retrieved from the various local NADF files.

It is interesting to note that node crash recovery is treated as an expected and normal behaviour rather than an exception condition. Node crash recovery amounts to the application of another rule module (the same as before the crash in fact) yet with an earlier starting time. The only additional arrangement is rebooting the workstation under the security level C2 and activating the initialization procedure.

5.7 Portability

Two levels of portability can be identified:

At the design level: the distributed nature of the monitoring/analysis system makes it reusable for other network and machine architectures since no assumption is made about the underlying communication protocol, host architecture and operating system;

At the code level: this derives from the universality feature of the ASAX evaluator (hiding of the native format by using NADF files). In addition, the evaluator code proved highly portable to many machine architectures (IBM PC, SINIX, BS2000 SunOS, Ultrix, and currently under work VAX VMS).

Chapter 6

Command Interface

6.1 Introduction

This chapter presents the Security Administrator Console Interface. It describes the command interpreter for the command language by giving a specification of each command. These commands implement the detailed functionalities given in the previous chapter and support concepts such as master and slave machine. A *distributed evaluator* is a set of evaluators performing filtering and an evaluator (the master) responsible for the global analysis. In order to be able to check the validity of certain commands, the state[1] of a particular host is described by 3 variables:

isAlive = 1 if the host is alive; = 0 if the host is down;

isC2 = 1 if the host is C2 security level; = 0 otherwise. This easily checked by verifying the presence of the file `/etc/security/passwd.adjunct`;

haveFA = 1 if a Format Adaptor process is running on the host; = 0 otherwise.

For concise specifications, the variables corresponding to a host *hostname* are referred to as:

- *hostname*.isAlive;
- *hostname*.isC2;
- *hostname*.haveFA.

In the following specifications, *precondition* corresponds to a set of conditions to be fulfilled in order that the specified command operates consistently. In case one or more of these conditions does not hold, an error message is generated on the standard output of the console process to indicate the kind of the error.

[1] In fact, additional items must be included in describing a host state such as the fact that a host is attached to a network and that all requirements for communicating with other monitored hosts are fulfilled. However, these issues are considered irrelevant as far as the interactive command interface description is concerned.

6.2 Initialization

The Distributed Asax Console is invoked by the command **console**. It aims at starting the command interpreter. The command language is fully described by the specification of the commands found in the rest of this chapter. .

6.2.1 Activation of Format Adaptor

purpose executes the Format Adaptor on a given host, a specified list of hosts or all known hosts;

synopsis
fa -b
fa -h *hostname*
fa -f *hostfile*

precondition

- *hostname*.isAlive = 1;
- *hostname*.C2 = 1;
- *hostname*.haveFA = 0;

postcondition

- *hostname*.isAlive = 1;
- *hostname*.C2 = 1;
- *hostname*.haveFA = 1;
- a Format Adaptor process is started on the host *hostname* using the file with the standard extension *not_terminated* as input file.

The command **fa** -b is equivalent to the sequence of commands:

$$\textbf{fa}\ hostname_1\ \ldots\ \textbf{fa}\ hostname_n$$

where $hostname_1$, ..., $hostname_n$ is the sequence of the known hosts. Similarly, the command **fa** -f *hostfile* is equivalent to the sequence of commands:

$$\textbf{fa}\ hostname_1\ \ldots\ \textbf{fa}\ hostname_q$$

where $hostname_1$, ..., $hostname_q$ is the sequence of host names contained in the file *hostfile*.
This command is not actually implemented since the distributed FAs are automatically started at boot time from `/etc/rc.local`.

6.3 Analysis Control

6.3.1 Distributed Evaluator Description File

All evaluator processes making up a distributed evaluator are assigned a unique identification number. A distributed evaluator is fully described by giving the following items:

- the set of slave machines and the master machine making up the distributed evaluator;
- the module name used by each slave machine to perform its own filtering;
- the module name used by the master machine to perform the global analysis;
- and optionally a time stamp or a time interval which determines the NADF files involved in the distributed analysis.

Commands used to activate a distributed analysis are supplied in a text file (Distributed Evaluator Description File) describing the distributed evaluator to be activated. We choose the following BNF syntax for such files:

```
⟨description file ⟩ ::= ⟨slave description ⟩ ;
     ⟨master description ⟩ .

⟨slave description ⟩ ::=
     slaves ⟨slave group ⟩ ; ... ; ⟨slave group ⟩

⟨master description ⟩ ::=
     master ⟨host name ⟩: ⟨module name ⟩ [: ⟨time spec ⟩]

⟨slave group ⟩ ::= ⟨hosts list ⟩ : ⟨module name ⟩

⟨hosts list ⟩ ::= ⟨host name ⟩ , ... , ⟨host name ⟩

⟨time spec ⟩ ::= ⟨time stamp ⟩
     | ⟨time interval ⟩

⟨time stamp ⟩ ::=
     ⟨year ⟩ ⟨month ⟩ ⟨day ⟩ ⟨hour ⟩ ⟨minutes ⟩ ⟨seconds ⟩

⟨time interval ⟩ ::= [ ⟨time stamp ⟩ , ⟨time stamp ⟩ ]

⟨year ⟩, ⟨month ⟩, ⟨day ⟩ ::= 2 decimal digits

⟨hour ⟩, ⟨minutes ⟩, ⟨seconds ⟩ ::= 2 decimal digits

⟨host name ⟩ ::= identifier

⟨module name ⟩ ::= identifier
```

The system performs some checks on the given file before creating a new distributed evaluator. These checks include valid hosts (alive, in the security level C2 and having a Format Adaptor) and time specifications.
Figures 6.1, 6.2, 6.3 are examples of distributed analysis description files. The first one is on-line, the second analyses all records generated since 19940215060000 while the third one analyses records with a time stamp falling in the time interval [19940215060000, 19940216000000].

master `patate: global;`
slaves `patate, salade, poireau, endive: filter.`

Figure 6.1: Distributed Analysis Description File: Example1.

master `patate: global: 19940215060000;`
slaves `patate, salade, poireau, endive: filter.`

Figure 6.2: Distributed Analysis Description File: Example2.

master `patate: global:[19940215060000, 19940216000000];`
slaves `patate, salade, poireau, endive: filter.`

Figure 6.3: Distributed Analysis Description File: Example3.

6.3.2 Distributed Evaluator execution

purpose receives a Distributed Evaluator Description file and initiates filtering on slave machines and global analysis on the master machine according to this description. The created distributed evaluator receives an identification number.

synopsis run *descr_file*

precondition

- *descr_file* is valid *wrt* the above syntax;
- for each host *hostname* listed in the file *descr_file*

we have:

- *hostname*.isAlive = 1;
- *hostname*.C2 = 1;
- *hostname*.haveFA = 1;

postcondition

- the state of the hosts is not changed;
- for each group of slaves, an evaluator process is initiated on each host of the group with the same module name;

- an evaluator process is initiated on the master host with associated module name;
- if a time stamp is given, it is used to determine the NADF files used for slaves as well as for the master machine. Otherwise, all analyses are performed on-line.

6.3.3 Changing current filtering

purpose receives an evaluator instance number and replaces the corresponding analysis by the one supplied as argument to the command.

synopsis **rerun** *instanceNumber* -m *module_name*

precondition

- *module_name* is a valid path name;
- *instanceNumber* is a valid evaluator instance number;

postcondition let *cur* be the current NADF record being analyzed at the time this command is invoked.

- the current analysis performed by the evaluator identified by *instanceNumber* is suspended and the completion rules are executed;
- the analysis is resumed from the NADF record *cur* using the module name *module_name*.

6.3.4 Changing the time interval of the analysis

purpose receives a distributed evaluator instance number and a time specification and replaces the corresponding global analysis by the one supplied as argument to the command.

synopsis **rerun** *instNum* [-t *low*] | [-i *low high*]

precondition *instNum* is a valid **master** evaluator instance number;

postcondition

- all evaluators making up the distributed evaluator whose master evaluator is the one identified by *instNum* are stopped;
- with -t option, the analyses on slave machines are resumed from the NADF record having the time stamp *low* or greater and using the same module name as before this command is invoked;
- with -i option, the analyses on slave machines are resumed on the NADF records falling in the time interval [*low*, *high*] and using the same module name as before this command is invoked;
- on the master machine, the analysis is resumed from the NADF record next received by the master evaluator and using the same module name as before this command is invoked;

6.4 Login Control

6.4.1 Notations and definitions

The following definitions and notations are necessary for an effective use and understanding of the **logcntl** command described in the next sections. Most of the notations and definitions are excerpts form [5] where auditing installation and administration are fully described.

Event class An *event class* defines a set of occurrences to be audited. The classes defined to date are listed in the following table:

short name	long name	short description
dr	*data_read*	Read of data, open for reading. etc.
dw	*data_write*	write or modification of data
dc	*data_create*	creation or deletion of objects
da	*data_access_change*	change in object access (modes, owner)
lo	*login_logout*	Login, Logout, creation by `at`
ad	*administrative*	Normal administrative operation
p0	*minor_privilege*	Privileged operation
p1	*major_privilege*	Unusual privileged operation

Audit Flag An audit flag describes a particular audit class in an audit state definition. An audit flag is an indication of what to do with an event. The format is,

$$< option >< class >$$

where *option* is either +, - or not present, and *class* is any event class. A plus means to audit successful events. A minus means to audit failed events. The absence of either means to audit both successful and failed events.

Audit Value Definition *An audit value definition* is a comma-separated list of audit flags. Here is a sample definition:

+dr,-dw,lo,p0,p1

This means: audit successful data read, failed data writes, all new login, and both kinds of privileged operations.

System Audit Value Definition Is the audit value definition contained in the `flags:` line in the `/etc/security/audit/audit_control` file. As said in chapter 3, it determines what event to be audited for all users on the system. Here is a sample content of an *audit_control* file:

```
dir:/etc/security/audit/poireau/files
flags:+dr,-dw,lo,p0,p1
minfree:20
```

The `dir:` line identifies the directory where audit trails are generated. The audit threshold line `minfree:` specifies the percentage of free space that must be present in the file system containing the current audit file. If free space falls below this threshold, the audit daemon *auditd(8)* informs the auditing administrator to take appropriate actions. The `flags:` entry was described above. The following table gives examples of what programs or system -or library- calls each of the flags audits. A more complete list can be obtained from Appendix E, *Formats of Audit Records* in [5].

flag	*examples*		
dr	*stat*(2)	*statfs*(2)	*access*(2)
dw	*ftruncate*(2)	*kill*(2)	*utimes*(2)
dc	*link*(2)	*mkdir*(2)	*rmdir*(2)
da	*chmod*(2)	*chown*(2)	*fchmod*(2)
lo	*login*(1)	*rexd*(8)	*rexecd*(8)
ad	*su*(1)	*passwd*(1)	*clri*(8)
p0	*chroot*(2)	*quota*(1)	*quotaon*(8)
p1	*reboot*(2)	*setdomainname*(2)	*sethostname*(2)

In addition, the special flag *all* (not listed in the above table) indicates that all events should be audited; *-all* indicates that failed attempts are to be audited, and *+all* indicates that successful attempts are to be audited. The prefixes ^, ^- and ^+ turn off flags specified earlier in the audit value definition. They are typically useful for updating a user audit value in `/etc/security/passwd.adjunct`.

User Audit Value the *user audit value* for a particular user is determined by two audit value definitions. The first one corresponds to what event classes must always be audited. The second one refer to what event classes must never be audited. These two audit value definitions are contained in the file `/etc/security/passwd.adjunct`. For each user, an entry in this file is a colon-separated list of 7 fields the sixth and seventh of which determines the user audit value for this particular user. Here is a sample entry in this file:

`amo:`<*encrypted password*>`:::-dw:-dr`

These flags mean *never* to audit successful data reads and *always* audit failed data writes.

Process Audit State as described in chapter 3, every process has an associated *audit state* that determines which events are to be audited for that process. The audit state is set at login time. It is the result of a combination of the *system audit value* and the *user audit value* according to the following rules:

- If the system audit state defines auditing for an event and the user audit state has nothing to say, the event will be audited;

- If the system audit state defines auditing for an event and the user audit state says to ignore this event, the event will not be audited;
- If the user audit state defines auditing for an event, it will be audited regardless of the system audit state.

Since a process inherits its audit state from its parent process, all children have the same audit state as the login shell. The process audit state can be manipulated with the **logcntl** command described below.

6.4.2 Temporary change of user audit state

purpose receives audit flags and modify temporarily process audit states on a user basis.

synopsis **logcntl** -t *host user auditflags*

precondition *user* is a valid user name, *host* is a valid host name and *auditflags* are valid audit flags;

postcondition set the processes audit state of all processes running on the host *host* and owned by the user *user* using *auditflags*. This done following the rules:

- if the previous audit state defines auditing for an event and the *auditflags* have nothing to say, the event will be audited;
- if the previous audit state defines auditing for an event and this event is turned off (using the flag ^) in *auditflags* this event will not be audited;
- if *auditflags* defines auditing for an event, this event will be audited, regardless of the previous audit state.

A new login session for *user* on *host* reconstructs the process audit state from the audit flags in the audit_control and passwd.adjunct files.

6.4.3 Permanent change of user audit state

purpose receives audit flags and modify permanently process audit states on a user basis.

synopsis **logcntl** -p *host user always never*

precondition *user* is a valid user name, *host* is a valid host name, *always* and *never* are valid audit values;

postcondition The process audit state for processes owned by *user* on *host* is reconstructed using the system audit value and the flags *always* and *never* instead of those found in the passwd.adjunct file on the host *host*. This is done following the combination rules described in 6.4.1. This change in audit state process takes effect immediately (i.e, all current processes of *user* are affected on *host*) and lasts for all subsequent sessions of this user.

6.4.4 Host login control

purpose receives audit flags and modify process audit states on a host basis.

synopsis **logcntl** -h *host auditflags*

precondition *host* is a valid host name and *auditflags* are valid audit flags;

postcondition replaces the system audited value (as represented by the `flags:` line of the *audit_control* file) by the audit flags *auditflags* on the host *host*. All *current* user sessions are not affected by this change, but all new login sessions will apply this new flags. .

6.4.5 Network login control

purpose receives audit flags and modify process audit states on a network basis.

synopsis **logcntl** -f *hostfile*

precondition *hostfile* is a text file where each line is composed of a host name and (possibly enclosed in quotes) an audit value to be applied for that host. These two items must be separated by at least one white space;

postcondition The effect of this command is the same as successively executing the previous command with each pair of host name and audit value contained in the file *hostfile*.

6.5 Miscellaneous Commands

6.5.1 Stopping an evaluator

purpose receives an evaluator instance number and stops the corresponding analysis

synopsis **kill** *instanceNumber*

precondition *instanceNumber* is a valid evaluator instance number;

postcondition

- if *instanceNumber* is a *slave* evaluator instance number, this evaluator is terminated;
- if *instanceNumber* is a *master* evaluator instance number, this evaluator and all slave evaluators associated with it are terminated.

6.5.2 Reinitializing the system

purpose stop all current distributed evaluators

synopsis **reset**

precondition Let H be the set of known hosts. For each element *hostname* in H we have:

- *hostname*.isAlive = 1;
- *hostname*.C2 = 1;
- *hostname*.haveFA = 1;

postcondition all evaluators running on any of the hosts in H are terminated.

6.5.3 Current active evaluators

purpose show characteristics of current active evaluators on the various hosts.

synopsis **ps**

precondition

postcondition For each active evaluator, the following characteristics are shown on the standard output:

- instance number;
- instance number of the corresponding master evaluator. If this is a master evaluator, it is considered to be its own master.
- name of the host on which it is running;
- module name used by the evaluator;
- starting and ending time of the distributed analysis. If a colon is given as a time stamp, the analysis is on line.

6.5.4 Audit trail size control

purpose change default maximum size for native and NADF files.

synopsis **dflsize** *host* [-i *adf_size*] [-o *nadf_size*]

precondition *host* is a valid host name

postcondition with option -i (resp. -o) change the default maximum size for native (resp. NADF) files to *adf_size* (resp. *nadf_size*) on *host*.

6.5.5 The configuration

purpose show the list of known hosts and their state.

synopsis conf

precondition

postcondition For each known host, the following characteristics are shown on the standard output:

- host name;
- is it up or down;
- architecture name;
- internet address;
- is it C2 Security level;
- does it have a Format Adaptor.

This command is not yet implemented.

6.5.6 Help command

purpose gives a short description of commands.

synopsis help [command_name]

precondition

postcondition If a command name is given, a short description of its purpose as well as its syntax is given. If no argument is given, it prints a short description of the purpose of all commands.

Chapter 7

The implementation

7.1 Introduction

This chapter examines the implementation issue of the detailed functionalities addressed in chapter 5. This prototype version uses PVM (Parallel Virtual Machine) as inter-process communication support. This choice is justified for two reasons: the prototyping phase could not afford the cost of implementing process communication built on IPC (interprocess communication) and sockets. On the other hand, PVM is architecture and network independent and is also a freely distributed software.
The next section is a brief overview of PVM. Section 3 describes the processes living in each host and their interaction. Section 4 specifies all the messages exchanged between processes. Section 5 contains a description of a typical session which illustrates the main ideas.

7.2 PVM

PVM is a software system that allows the utilization of an heterogeneous network of parallel and serial computers as a single computation resource. These computing elements may be interconnected by one or more networks, which may themselves be different (e.g. Ethernet, the Internet, and fiber optic networks). These computing resources are accessed by applications via a library of standard interface routines. These routines allow the initiation and termination of processes across the network as well as communication and synchronization between processes.

Application programs are composed of components that are subtasks at a moderately large level of granularity. During execution, multiple instances of each component may be initiated. Refer to [4] for further informations.

7.3 Format Adaptors

On each host participating in the virtual machine, a single Format Adaptor is running on-line. This is implemented by piping the Format Adaptor with audit record generator of the SunOS 4.1 so that audit records are caught on the fly and translated to NADF format and then written to the current NADF file. The Format Adaptor reads an audit record from the current native audit file. If the end of file is reached, it sleeps for 1 second and then looks for possible subsequent audit records. Observation of the target system shows that, in the average, an audit record is written to the native file in about the same period of time.

When the current NADF file exceeds the size limit, the following operations are performed:

- suppose date0 is the date at which this NADF file was open. Then the current NADF file has the standard name date0_not_terminated.NADF. Then this file is closed and its name is changed to date0_date1.NADF where date1 is the date of the close operation;
- a new current NADF file named date1_not_terminated.NADF is created;
- the Format Adaptor signals audit deamon to close the current audit file and to open a new audit file;
- the Format Adaptor erases the old native file and then resumes the translation process using the newly created files.

Native and NADF files reside under the same directory which is

/etc/security/audit/*server*/files

where *server* is the name of the workstation the Format Adaptor is running on. Format Adaptors are not part of the PVM system so that they are not affected when the virtual machine is lost for some reason. This is mandatory in order to ensure the Distributed system availability requirement.

7.4 The Parallel Virtual Machine Configuration

Using PVM it is possible to view the network of audited Sun workstations as a single computing resource that may be analyzed using the ASAX evaluators. Each workstation runs SunOS 4.1 under the security level C2. The virtual machine consists of a certain number of concurrent ASAX evaluators capable of performing local analysis and exchanging messages. A detailed description of these processes follows.

7.4.1 The audit state controller processes

The audit state controller process is responsible for maintaining the process audit state of all processes running on the same host. It is always blocked waiting for control messages to arrive. A control message that can be received by such a process contains the audit flags representing the new audit state to be applied. The audit state controller process put the host in the state represented in the control message and sleeps again waiting for further control messages.

It uses the C library which offer a relatively complete set of routines to control the audit state. It also uses the PVM system library routines for receiving the above control messages. There is also one such a process on each host of the PVM system.

7.4.2 Evaluator processes

An evaluator process is responsible for performing audit trail analysis proper. At a given time, it has a memory area containing the internal code of rule module to be applied and a set of active rules. The effect of the evaluator process is to apply the set of all active rules to the NADF record previously stored (by some body) in its receive buffer and then to wait for the next received audit record. The evaluator process can also receive a module name. In this case, it first executes all rules active at completion, release all memory areas for rule module internal code and active rules then it compiles this rule module yielding internal code and a list of init actions. It next executes init actions and waits for the next received record. If one is received, the current active rules are applied to it and so on. Note that many evaluator processes can coexist in the same host.

Evaluator processes use a predefined C routine *send_current* to send the current record contained in the receive buffer to the (unique) central evaluator process.

7.4.3 The supplier processes

On each host taking part in the PVM system, there is one or more supplier processes running on that host. A supplier process reads audit records from one of the NADF files stored in the local host and send them in sequence to an evaluator process. The receiving evaluator could be the local evaluator or any other one belonging to the PVM system. It continues sending records in sequence unless it is interrupted. It can subsequently receive an other signal in order to read another NADF file and send its records to a perhaps different evaluator. The supplier process uses PVM library routines for sending audit records and receiving control messages.

7.4.4 The console process

In addition to the above component processes, a console process can be activated on a specified machine in order to offer an interactive Security Administrator interface to the PVM system. This process implements the set of commands described in the previous chapter. It parses the commands entered by the Security Administrator, constructs the corresponding message and then send it to the specified instance.

7.5 Specification of the exchanged messages

In the following, the set of all exchanged messages are described by specifying the sending and receiving processes, the content of the message and the actions performed by the receiver upon reception. All messages are tagged so that a process can distinguish easily the format of the rest of the message.

Sending of an NADF record: au_rec_msg this message may originate from an evaluator or a supplier process and can only be received by an evaluator process. It contains the audit record exchanged. When an evaluator process receives such a message, it applies the set of active rules to it and then blocks waiting for a further record;

Modification of a set of analysis rules: mod_name_msg this message may originate from a console process or an evaluator process and can only be received by an evaluator process. It contains a string representing the path name of a rule module. When an evaluator process receives such a message, it returns to the sender an error code if the path name is incorrect, it otherwise compiles the named rule module and restarts analysis using this module and starting from the *next* received audit record;

Request to send an NADF record: supply_msg this message may originate from a console process or an evaluator process and can only be received by a supplier process. It contains optionally a time stamp or a time specification. Upon reception of such a message, a supplier process starts sending audit records to its associated evaluator process. If the message does not contain a time specification, the supplier process starts sending on-line. Otherwise, the specified time stamp or time interval is used to determine the first NADF record that must be sent. All audit records are appended a new audit data *host_name* specifying the sending host name. This continues until the next *idle_msg* or *supply_msg* message is received;

Request to stop sending NADF records: idle_msg this message may originate from a console process or an evaluator process and can only be received by a supplier process. It only contains its message tag. When receiving such a message, a supplier process stops reading and sending of audit records from its local audit trail until the next (if any) *supply_msg* message is received;

Modification of the granularity level: au_state_msg this message may originate from a console process or an evaluator process and is only received by an audit state process. It contains a string representing a system or a user audit state. When an audit state process receives such a message, it changes the audit state to the one specified by the message.

7.6 A typical session

7.6.1 Initialization

Suppose that all participating hosts are C2 Security level. The Security Administrator can specify a list of hosts he wants to monitor by editing a text file (ex. *hostfile*) containing this list. He invokes the Security Administrator Console using the command **dasax**. He can start a Format Adaptor process on each of these hosts using the command **fa** -f *hostfile*. This causes an NADF file to be generated on each of the hosts and the analysis can be started.

7.6.2 Distributed Analysis

The Security Administrator prepares a text file (ex. *descr_file*) describing the distributed analysis he wants to launch from his console. Figure 7.1 shows a sample file representing a distributed analysis description. The dis-

```
master patate:  global;
slaves patate, salade, poireau, endive:  filter.
```

Figure 7.1: Distributed Analysis Description File.

tributed (on-line) analysis is triggered with the command **run** *descr_file*. At this point he can expect report messages resulting from this analysis. Suppose that a message indicates a suspicious behaviour on a given host, the Security Administrator can first change the audit state of that user to a more granular one. This is done by informing the corresponding audit state instance using the command **logcntl** and supplying the user name and the audit flags as arguments. He can subsequently activate an appropriate analysis on the master host (for example a record show to screen up the user actions). This is done using the command **rerun** *instanceNumber* show.asa where *instanceNumber* is the instance number of the master evaluator.

Suppose that the master host (`patate`) suddenly goes down at 4pm and is up again at 16:17, he can first reactivate the Format Adaptor on `patate` (**fa** -h `patate`) and then the analysis is resumed by adding a time stamp to the file *descr_file* as shown in Figure 7.2.

```
master patate:  show:  19931202155900;
slaves patate, salade, poireau, endive:  filter.
```

Figure 7.2: Distributed Analysis Description File.

The analysis is automatically recovered and the monitoring can continue. Similarly he can trigger other analysis on certain group of hosts and he can use the command **ps** to see what analysis are currently running.

7.6.3 Termination

A distributed analysis can be stopped by sending a signal to the corresponding master evaluator. This is done by the command **kill** *instanceNumber* where *instanceNumber* is the instance number of the master evaluator. All slave evaluators depending on this master evaluator are stopped automatically. The only thing to do in order to terminate analysis on all hosts is to invoke the command **reset** which keeps only Format Adaptor processes and put the system in the state just after initialization.

Bibliography

[1] N.Habra, B. Le Charlier, A. Mounji. *Preliminary report on Advanced Security Audit Trail Analysis on Unix* 15.12.91. 34 pages

[2] N.Habra, B. Le Charlier, A. Mounji. *Advanced Security Audit Trail Analysis on Unix. Implementation design of the NADF Evaluator* Mar 93. 62 pages

[3] N.Habra, B. Le Charlier, I. Mathieu, A. Mounji. *ASAX: Software Architecture and Rule-based Language for Universal Audit Trail Analysis.* Proceedings of the Second European Symposium on Research in Computer Security (ESORICS). Toulouse, France, November 1992.

[4] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, V. Sunderam. *A User Guide to PVM (Parallel Virtual Machine).* ORNL/TM-11826. July, 1991. 13 pages

[5] Sun Microsystems. *System and Network Administration* Part Number 800-3850-10 Revision A of 27 March, 1990.