

Log Centralization for prelude NIDS  
Prelude Log Monitoring Lackey

Alexandre LAUNAY    Pierre-Jean TURPEAU

ENSEIRB

*Computer Science - Third year*

March 7, 2002

## **Abstract**

This report describes the work done on an open source Hybrid Intrusion Detection System called Prelude. It mainly deals with the problem of log centralization in an heterogenous environment.

As a third year project, we have implemented this fonctionnality for the Prelude IDS. The underlying idea is to create a free intrusion detection system capable of analyzing both logs from the windows and unix world (and any other type of network equipment capable of sending logs using the BSD Syslog protocol).

This document contains informations on Syslog and the BSD Syslog protocol, logs in Windows, the problem of log centralization and the Prelude centralization module.

# Contents

<b>List of Figures</b>	<b>iii</b>
<b>1 Introduction to syslog</b>	<b>3</b>
1.1 The syslog program . . . . .	3
1.1.1 Presentation . . . . .	3
1.1.2 Facilities and Severity . . . . .	3
1.1.3 Configuring syslog . . . . .	5
1.2 The syslog protocol . . . . .	6
1.2.1 Transport Layer Protocol . . . . .	6
1.2.2 Message format . . . . .	7
1.2.3 Architecture . . . . .	8
1.2.4 Relaying syslog packets . . . . .	9
<b>2 Logs in windows environment</b>	<b>11</b>
2.1 Managing logs with windows . . . . .	11
2.2 The Event Log Architecture . . . . .	12
2.2.1 Applications logs . . . . .	13
2.2.2 Security logs . . . . .	13
2.2.3 System logs . . . . .	13
2.3 The Event Log Entry Structure . . . . .	13
2.4 NT Auditing events . . . . .	15
2.5 Windows 2000 enhancements . . . . .	16
<b>3 Log centralization architecture</b>	<b>17</b>
3.1 Issues . . . . .	17
3.1.1 Centralization Issues . . . . .	17
3.1.2 security Issues . . . . .	17
3.2 Integrating Windows in a syslog architecture . . . . .	18

3.2.1	Windows weakness . . . . .	18
3.2.2	Open source Tools . . . . .	18
3.2.3	Freeware tools . . . . .	19
<b>4</b>	<b>Work logs for security purpose</b>	<b>20</b>
4.1	Securing log files . . . . .	20
4.2	Detecting attacks . . . . .	22
4.3	Intrusion Detection System . . . . .	24
4.3.1	Presentation . . . . .	24
4.3.2	General architecture . . . . .	24
<b>5</b>	<b>Prelude IDS</b>	<b>27</b>
5.1	What is Prelude IDS ? . . . . .	27
5.2	Prelude-LML . . . . .	29
5.2.1	How Log Monitoring Lackey works . . . . .	29
5.2.2	Interaction with plugins . . . . .	35
5.3	Future extensions . . . . .	37
5.3.1	Simple plugin implementation . . . . .	37
5.3.2	Local file monitoring . . . . .	38
5.3.3	SSL integration . . . . .	38
5.4	Compiling Prelude-LML . . . . .	39
<b>6</b>	<b>Conclusion</b>	<b>41</b>
<b>A</b>	<b>Windows events</b>	<b>42</b>
A.1	Audit events . . . . .	42
A.2	Interesting events to look for . . . . .	46
<b>B</b>	<b>First proposal</b>	<b>49</b>
B.1	Goals . . . . .	49
B.2	Solution . . . . .	51
B.3	Details . . . . .	53
<b>C</b>	<b>Glossary</b>	<b>54</b>
	<b>Bibliography</b>	<b>55</b>

# List of Figures

1.1	Syslog Message Facilities . . . . .	4
1.2	Syslog Message Severities . . . . .	5
1.3	Syslog message format . . . . .	7
4.1	PEO Protocol . . . . .	22
4.2	Typical IDS architecture . . . . .	26
5.1	Overview of the Prelude architecture . . . . .	28
5.2	Overview of the Prelude-LML architecture . . . . .	30
5.3	Data structures . . . . .	33
5.4	Data structures . . . . .	34
5.5	The logic behind Prelude SSL servers . . . . .	40
B.1	Simple overview of the Prelude architecture . . . . .	50
B.2	NTSyslog actual architecture . . . . .	51
B.3	Modified NTSyslog architecture . . . . .	52

# Introduction

Nowadays, network architectures within organizations are more and more complexes and often composed of various services running on several remote servers such as mail servers, web servers, database servers, print servers, etc. plus a wide number of users accessing these services through an intranet or an extranet from various types of desktops (UNIX-Like or Windows for example).

For system administrators, it's then a daily challenge to secure the data and protect the network to prevent the loss of functionality. Intrusion detection systems are one of the key tools available to detect malicious attacks and intrusions from a remote host or misuse of the network from a local user.

In the open source community there are few IDS. One of them is Prelude an Hybrid IDS. The characteristic of this kind of tool is to act as a Network IDS and as an Host based IDS. It is based on sensors and managers and each sensor has the capability to detect intrusion attempts within protocols (RPC, HTTP, telnet) and network attacks (spoofing, scanning, etc...).

Unfortunately, this tool is developed under Linux and works only on UNIX-Like environments. To increase its domain of use, we want to add the capability to centralize information from other types of systems. Indeed, we now often find Microsoft Windows servers in network architectures and we still have network equipments that use the BSD syslog protocol to report errors to syslog servers. Our goal is to develop a particular sensor which makes behaviour analysis on logs. These logs can be fetched from local files or from remote machines.

To introduce our work, you will find an overview of Syslog and the BSD Syslog protocol, the logs in a Windows environment and how to retrieve

them from a remote machine, some solutions for the logs centralization in an heterogenous environment and the description of logs centralization in Prelude.

# Chapter 1

## Introduction to syslog

### 1.1 The syslog program

#### 1.1.1 Presentation

The syslog program provides a standardized framework under which programs (both operating system and applications) can issue messages to be handled by none, any, or all of the following actions based upon the configuration of syslog:

- recorded to a file (i.e. `/var/adm/messages`) or device (i.e. `/dev/console`)
- sent directly to a user or users if currently logged in (i.e. `root`)
- forwarded to another machine (i.e. `@loghost`)

#### 1.1.2 Facilities and Severity

Each message is a single line of text with an associated facility and severity. The facility can be thought of as a category that depends upon the program from which the message originates as shown in table 1.1. The developer of a program decides which facility a program will utilize. In some cases, it may be configurable by the end user. Severities are hierarchical and range from `emerg` being the most important down to `debug` as the least significant as shown in table 1.2.

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages (note 1)
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon (note 2)
10	security/authorization messages (note 1)
11	FTP daemon
12	NTP subsystem
13	log audit (note 1)
14	log alert (note 1)
15	clock daemon (note 2)
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Figure 1.1: Syslog Message Facilities

Numerical Code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Figure 1.2: Syslog Message Severities

### 1.1.3 Configuring syslog

The syslog program is designed for Linux/Unix systems. It is actually a daemon : syslogd. The configuration file for syslog (/etc/syslog.conf) is where the administrator can define rules based upon the facility and severity of each arriving message. Each rule consists of a set of facility/severity pairings and an associated action separated by one or more tab characters (notspace characters!).

The components of a pairing are delimited by a period with the facility being followed by the severity (i.e. kernel.info). The facility of a message must match exactly and the severity of the message must be an equal or more important value before the corresponding action is taken. Severities in the file are essentially thresholds.

For example there is hereunder four rules concerning the kern facility:

```
# Kernel messages are first, stored in the kernel
# file, critical messages and higher ones also go
# to another host and to the console
#
kern.*                /var/adm/kernel
kern.crit             @finlandia
kern.crit             /dev/console
kern.info;kern.!err  /var/adm/kernel-info
```

The behaviour concerning these rules is described hereunder :

- the first rule direct any message that has the kernel facility to the file /var/adm/kernel.
- the second statement directs all kernel messages of the priority crit and higher to the remote host finlandia.
- the third rule directs these messages to the actual console.
- the fourth line tells the syslogd to save all kernel messages that come with priorities from info up to warning in the file /var/adm/kernel-info. Everything from err and higher is excluded.

It is to be pointed out that To redirect syslog messages to another machine, you must at a minimum modify the configuration of syslog on the originating system and probably the recipient system as well. On the originating system, you add a new line with an action consisting of an "at" sign immediately followed by the name of the recipient system to receive the message (in this case "finlandia"). The hostname used above must be resolvable and accessible by the originating system. Messages can be redirected to more than one remote loghost by using multiple lines with the same pairing, but a different action.

## 1.2 The syslog protocol

The communication between two syslog daemons (i.e. when logs are redirected to remote machines) is based on the specific protocol : the *BSD syslog protocol*.

The following informations are taken from the RFC 3164 BSD syslog Protocol.

### 1.2.1 Transport Layer Protocol

The syslog protocol relies on the user datagram protocol (UDP) as its underlying transport layer mechanism. The UDP port that has been assigned to syslog is 514. Furthermore the payload of any IP packet that has a UDP destination port of 514 MUST be treated as a syslog message.

## 1.2.2 Message format

### Description

A correct syslog message is constituted of three discernable parts :

- PRI : it is a number contained within angle brackets: <>. This number is known as the *Priority* value and represents both the Facility and Severity. The Priority value is calculated by first multiplying the Facility number by 8 and then adding the numerical value of the Severity ;

$$PRI = 8 * facility + severity$$

- HEADER : The HEADER contains two fields called the TIMESTAMP and the HOSTNAME. The HOSTNAME will contain the hostname or the IP address of the machine sending the message. The TIMESTAMP has the following format : “Mmm dd hh:mm:ss”:

Jan 03 06:35:11

- MSG : The MSG part has two fields known as the TAG field and the CONTENT field. The value in the TAG field will be the name of the program or process that generated the message. The CONTENT contains the details of the message.

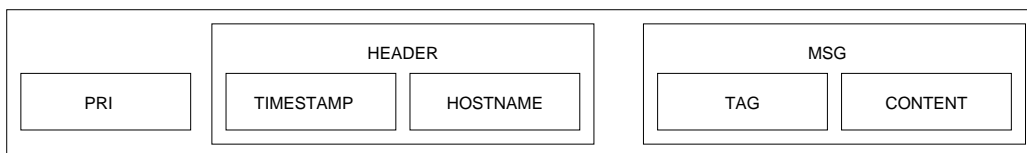


Figure 1.3: Syslog message format

### Example

```
<34>Oct 11 22:14:15 mymachine su: 'su root' failed for lonvick on /dev/pts/8
```

The different parts are identified as follows:

- PRI : < 34 >
- HEADER :
  - TIMESTAMP : OCT 11 22:14:15
  - HOSTNAME : mymachine
- MSG :
  - TAG : su
  - CONTENT : 'su root' failed for lonvick on /dev/pts/8

### 1.2.3 Architecture

The following definitions describe the entities involved in the syslog protocol:

- **device:** machine that can generate a message ;
- **relay:** machine that can receive the message and forward it to another machine ;
- **collector:** machine that receives the message and does not relay it to any other machines. This has been commonly known as a "syslog server" ;
- **Sender:** Any device or relay will be known as the "sender" when it sends a message ;
- **Receiver:** Any relay or collector will be known as the "receiver" when it receives the message.

The architecture of the devices may be summarized as follows:

- Senders send messages to relays or collectors with no knowledge of whether it is a collector or relay.
- Senders may be configured to send the same message to multiple receivers.
- Relays may send all or some of the messages that they receive to a subsequent relay or collector. In the case where they do not forward all of their messages, they are acting as both a collector and a relay. In the following diagram, these devices will be designated as relays.

- Relays may also generate their own messages and send them on to subsequent relays or collectors. In that case it is acting as a device. These devices will also be designated as a relay in the following diagram.

#### 1.2.4 Relaying syslog packets

When a relay receives a packet, it will check for a valid PRI and a valid TIMESTAMP in the HEADER part. From these rules there will be three general cases of received messages. They are described in the following subsections.

##### Valid PRI and TIMESTAMP

If the relay does find a valid PRI and a valid TIMESTAMP, then it will check its internal configuration. Relays **MUST** be configured to forward syslog packets on the basis of their Priority value. If the relay finds that it is configured to forward the received packet, then it **MUST** do so without making any changes to the packet.

##### Valid PRI and invalid TIMESTAMP

If a relay does not find a valid TIMESTAMP in a received syslog packet, then it **MUST** add a TIMESTAMP. It **SHOULD** additionally add a HOSTNAME. The remainder of the received packet **MUST** be treated as the CONTENT field of the MSG and appended.

The TIMESTAMP will be the current local time of the relay.

The HOSTNAME will be the name of the device, as it is known by the relay. If the name cannot be determined, the IP address of the device will be used.

If the relay adds a TIMESTAMP, or a TIMESTAMP and HOSTNAME, after the PRI part, then it **MUST** check that the total length of the packet is still 1024 bytes or less. If the packet has been expanded beyond 1024 bytes, then the relay **MUST** truncate the packet to be 1024 bytes. although this may cause the loss of vital information from the end of the original packet.

##### No PRI

If the relay receives a syslog message without a PRI, or with an unidentifiable PRI, then it **MUST** insert a PRI with a Priority value of 13 as well as

a `TIMESTAMP` and `HOSTNAME` as described in Section above.

### **Example of packet Modification**

If a syslog relay receives the following packet :

`Use the BFG!`

It should modify the packet by adding the `PRI`, the `TIMESTAMP` and his `HOSTNAME`. The original message would then be placed in the `MSG` part of the new syslog packet :

```
<13>Feb  5 17:32:18 10.0.0.99 Use the BFG!
```

# Chapter 2

## Logs in windows environment

### 2.1 Managing logs with windows

#### Event Viewer

In the windows world logs are managed by the Event Log service, which runs automatically in default workstation and server configurations, and listens for registered events from application providers. Configuration options for the Event Log service are controlled in the Event Viewer application. It is accessible as a standalone program or through the MMC. In the configuration options, you can increase or decrease the available space for the logfiles and set retention policies. The Event Log is accessible to remote machines via Remote Procedure Calls (RPC) via applications that utilize the Event Log API, such as the Event Viewer.

#### Task Scheduler

In both releases, there is a service that manages scheduled processing of executables and batch scripts. In Windows NT 4 without Internet Explorer 5, it is called the Schedule service, and in Windows 2000 or Windows NT 4 with Internet Explorer 5, it is called the Task Scheduler service. In either case, the scheduling service needs to be set to run at startup under the local system account in order to ensure proper functionality.

In Windows NT, the AT command is used to list, add, and delete jobs in the schedule. To get a list of scheduled jobs, just type AT in a command prompt window. To take a simple example on how to add a job, let's say

you have a script called runme.bat that you want to run once a day at noon. The correct AT syntax for submitting this job would be :

```
at 12:00 /every:Su,M,T,W,Th,F,S runme.bat.
```

The same procedure works in Windows 2000, but there is also a Scheduled Tasks folder in the Control Panel that allows you to submit jobs using the "Add Scheduled Task" wizard.

What happens when you want to run a command more frequently than once a day? Unfortunately, that's one of the shortcomings of AT. There is no way to schedule a single job to run once an hour, let alone once every 5 minutes. If you want to run a script or executable with a greater frequency, then you should utilize the Windows NT/2000 Server Resource Kit utility called "soon". You can create a script that launches your event and then schedules the next time for the system to run the job using the soon command. For example, if I wanted to run the C:\ runme.bat script every 5 minutes, I would make a C:\ runme5.bat script with the following contents:

```
@echo off

REM -----
REM RUNM5.BAT
REM -----

SOON 300 C:\RUNME5.BAT
C:\RUNME.BAT
[EOF]
```

In order to get this whole scheduling/rescheduling process started, I would suggest using the AutoExecNT service to kick it off during system boot. This useful service is part of the Windows NT/2000 Server Resource Kit, and you can get more information about it in the Resource Kit helpfile.

## 2.2 The Event Log Architecture

In windows NT there are three logs that are managed by the Event Log service. They are described in the following sections.

### 2.2.1 Applications logs

Any application that registers itself to the Event Log service can log to the Application category. You can find a list of all registered applications to the Event Log in the registry key :

```
HKLM\System\CurrentControlSet\Services\Eventlog\Application
```

Another important component to this registry entry is the location of the Event Message file, which translates message IDs to the appropriate human readable explanation for the error.

### 2.2.2 Security logs

The Security log is used exclusively for the NT auditing system, whether it be NTFS, share access, Winlogon, printing, BackOffice, or other Microsoft approved auditing subsystems for applications or services. Most people use the Security log to monitor unauthorized attempts to access data or resources on the NT system.

### 2.2.3 System logs

The System log registers events from any system-level component, such as the printing subsystem or the networking subsystem, for example. It also registers events from any installed service that is integral to the function of the system, such as the DHCP or Server service, for example. Some of the most important system events will be found in this log and it should be monitored closely.

## 2.3 The Event Log Entry Structure

The Event Log Entry Structure is constituted of 8 fields :

- **Date:** The date the event occurred
- **Time:** The (local) time the event occurred
- **User:** The username of the user on whose behalf the event occurred. This is the client ID if the event was actually caused by a server process,

or the primary ID if impersonation is not taking place. Where applicable, a security log entry contains both the primary and impersonation IDs

- **Computer:** The name of the computer where the event occurred
- **Event ID:** A unique number identifying the particular event type. The first line of the description usually contains the name of the event type. For example, 529 is the ID of the event that occurs when a logon failure occurs, and so the first line of the description of such an event is "Logon Failure"
- **Source:** The name of the system component that recorded the event in the security log. For the example, this is Security, indicating that it is the result of Windows NT security auditing. Applications can also define their own auditable events that can be recorded in the security log
- **Type:** Information, Error, Warning, Success Audit or Failure Audit
- **Category:** A classification of the event by the event source

N.B.: the human readable description of the event is not stored in Events Logs but in Message Files specific to each application.

### Example

```
3/14/2001 5:37:20 PM CORP\JDOE KANT 529 Security Failure  
Audit N/A
```

A dissection of an event log entry is helpful to determine what information can be extracted from the Event Log facility. Events are classified by event type (severity), with "information" at the low end, "warning" at the middle, and "error" at the highest severity. Note that these classifications are completely arbitrary ? the impact of an error for one service may be less than an informational error for another. The log entry is timestamped and the registered source is recorded. The event ID and user ID that generated the event, along with the computer name, are also recorded.

## 2.4 NT Auditing events

Windows NT has interesting auditing functions for security purpose. More than the general audit that is applied to all resources it is possible to configure specific audits on the access to file for example.

The security event logs are exclusively produced by windows auditing system, as we wrote before. Actually these audit events are a little bit more complex and have 10 fields :

- 1.Date
- 2.Time
- 3.User
- 4.Computer
- 5.Event ID
- 6.Source
- 7.Type
- 8.Category
- 9.Description
- 10.Data - event specific data.

In addition, the event category, which is event-specific and primarily used in the Security audit log, is stored with the event. The event description, which is tied to the event ID, is not stored in the event log entry, but any specific data to that entry is stored with the event.

A list of Audit Events specific structure is available in the section A.1 page 42.

## 2.5 Windows 2000 enhancements

Windows 2000 Servers configured with Active Directory or just DNS has 3 additional logs:

- Directory Service - Contains events reported by Active Directory

File: %SystemRoot%\System32\Config\Director.evt

- DNS Server - Contains events reported by Microsoft Windows 2000 DNS Server.

File: %SystemRoot%\System32\Config\DNSEvent.evt

- File Replication Service - Contains events reported by Microsoft FRS Service.

File: %SystemRoot%\System32\Config\NTFrs.evt

# Chapter 3

## Log centralization architecture

The aim of this chapter is to describe how to integrate windows event logs into a syslog environment.

### 3.1 Issues

#### 3.1.1 Centralization Issues

Unfortunately, a failed logon to a domain from an NT workstation will only log a security event to the workstation (if auditing for logon events is enabled) attempting to connect, rather than to a domain controller. For that reason alone, it is necessary to audit failed logons on every workstation that is on your domain. Therefore, you can see the importance of collecting event logs from multiple locations and collating them into a single source in order to perform proper analysis of unwanted attempts to access your systems.

#### 3.1.2 security Issues

There are known security issues with the design of the standard syslog service :

- Syslog uses the UDP protocol, which provides no inherent assurance or feedback to the sender whether the message arrived safely at its destination.

- Messages are sent in unencrypted plain text over the network, so it would not be particularly difficult for someone to intercept logging traffic and have insight into the installation.
- Anyone can direct messages of a misleading nature or significant quantity to the syslog daemon with no authentication. This might obscure the tracks of an intruder, overwhelm the network, or fill any available disk space allocated for logs.

## 3.2 Integrating Windows in a syslog architecture

### 3.2.1 Windows weakness

In one hand Windows 2000 has some log centralizing capabilities. In fact it is possible to monitor logs from a remote machine. On the other hand Windows NT doesn't provide this functionality and is still installed on many machines. Consequently we can not build a reliable log centralization architecture without third-party products which may be very expensive.

It should rather be interesting to use a architecture that has already been probed. Syslog is a standard protocol stemming from Unix. Nowadays, it is supported by nearly all major devices. For example, most routers and network printers are able to provide diagnostic information via syslog. So a syslog based monitoring solution is able to gather data from a variety of sources. Using syslog gives additional flexibility as needs may grow.

### 3.2.2 Open source Tools

We propose here a list of tools that can be useful when centralizing logs:

- *Zebedee* : Zebedee is a simple program to establish an encrypted, compressed "tunnel" for TCP/IP or UDP traffic between two systems. This allows data from, for example, telnet, ftp and X sessions to be protected from snooping. You can also use compression, either with or without data encryption, to gain performance over low-bandwidth networks.

- *NTsyslog* : This program runs as a service under Windows NT 4.0 and Windows 2000. It formats all System, Security, and Application events into a single line and sends them to a syslog(3) host.

### **3.2.3 Freeware tools**

- *Kiwi's Daemon Syslog* : a freeware windows and NT that receives logs, displays and forward Unix type syslog messages PC hosts.
- *Kiwi's syslog generator* : a windows and NT message generator which sends Unix type syslog to any Unix or PC syslog daemon.

# Chapter 4

## Work logs for security purpose

In this section we'll present some principles and examples about preventing detecting malicious attacks.

### 4.1 Securing log files

When a somebody takes control over a machine, he usually begins by altering log files to cover his intrusion. Log files are worthless if you cannot trust the integrity of them. Consequently the first step to reviewing logs is securing logs.

There are some simple measures to protect logs :

#### Using a secure log server

First you have to log traffic both locally and to a remote log server. It is interesting to make your log server a dedicated system, ie. the only thing it should be doing is collecting logs from other systems. It can be as easy as building a linux box to act as your log server. This server should be highly secured, with all services shut off, allowing only console access. Also, it is necessary that port 514 UDP is blocked or firewalled at your Internet connection. This protects your log server from receiving bad or un-authorized logging information from the Internet.

## Changing names

It is also possible to recompile syslogd to read a different configuration file, such as `/var/tmp/.conf`. This way the black-hat does not realize where the real configuration file is. This is simply done by changing the entry `"/etc/syslog.conf"` in the source code to whatever file you want. Even though the original configuration file is now useless, this will throw off the black-hat from realizing the true destination of our remote logging.

## Secured Transport protocol

Another option for your systems is to use a secure method of logging. This can be done by:

- replacing syslogd binary with something that has integrity checking and a greater breadth of options: `syslog-ng` for example.
- securing NTsyslog by integrating SSL export of logs.

## Cryptographic log protection

When log files aren't secured it can rather be interesting to know whether these files have been altered during a malicious attack. This can be done by using the PEO protocol like in modular syslog.

The PEO (from the Spanish "Primer Estado Oculto" which means "Hidden Initial State") algorithm consists of three stages. Stage one is the initial key (K0) generation; stage two calculates a new key consisting of the hash of the previous one concatenated with the new log line; on stage three, the auditor repeats all the second stage operations for the whole log set. If the integrity was compromised, the last stage will not match.

The PEO protocol can be simply modified to get stronger. It is possible to add to each line of source a MAC<sup>1</sup>. Furthermore there are other protocols which work on the same principles like VCD<sup>2</sup> [9] but we won't give details here.

---

<sup>1</sup>Message Authentication Code

<sup>2</sup>Remounting Key Vectors

INSTANT	ACTION	DESCRIPTION
0	INIT $K_0 = \text{Random}()$	The auditor generates a random $K_0$ and stores it in a secure place
i	STORE $K_i = H(K_{i-1}, D_i)$	The source generates $D_i$ , the system stores it and computes $K_i$ using $D_i$ and $K_{i-1}$ . $K_{i-1}$ is destroyed.
n	VERIFY $V_0 = K_0$	The auditor verifies $K_n$ computing as a function of $K_0$ and the $D_j$ s

Figure 4.1: PEO Protocol

## Conclusion

Most of the logs we will use are the ones stored on the remote log server. As mentioned earlier, we can be fairly confident of the integrity of these logs since they are on a remote and secured system. Also, since all systems are logging to a single source, it is much easier to identify patterns in these logs. The only time you would want to review logs stored locally on a system is to compare them to what the log server has. You can determine if the local logs have been altered by comparing them to the remote logs.

## 4.2 Detecting attacks

To explain the importance of archiving logs we shall give an simple example of log processing. At the beginning of a malicious attack, there is often scan to find a hole in a system. A common tool to realize this is *nmap*. This tool scans every port on a machine to find which services are open. A scan with nmap would leave the following traces in your log files :

```

/var/log/secure
Apr 14 19:18:56 mozart in.telnetd[11634]: connect from 192.168.11.200
Apr 14 19:18:56 mozart imapd[11635]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.fingerd[11637]: connect from 192.168.11.200

```

```
Apr 14 19:18:56 mozart ipop3d[11638]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.telnetd[11639]: connect from 192.168.11.200
Apr 14 19:18:56 mozart in.ftpd[11640]: connect from 192.168.11.200
Apr 14 19:19:03 mozart ipop3d[11642]: connect from 192.168.11.200
Apr 14 19:19:03 mozart imapd[11643]: connect from 192.168.11.200
Apr 14 19:19:04 mozart in.fingerd[11646]: connect from 192.168.11.200
Apr 14 19:19:05 mozart in.fingerd[11648]: connect from 192.168.11.200
```

/var/log/maillog

```
Apr 14 21:01:58 mozart imapd[11667]: command stream end of file, while
reading line user=??? host=[192.168.11.200]
Apr 14 21:01:58 mozart ipop3d[11668]: No such file or directory while
reading line user=??? host=[192.168.11.200]
Apr 14 21:02:05 mozart sendmail[11675]: NOQUEUE: [192.168.11.200]:
expn root
```

/var/log/messages

```
Apr 14 21:03:09 mozart telnetd[11682]: ttloop: peer died: Invalid or
incomplete multibyte or wide character
Apr 14 21:03:12 mozart ftpd[11688]: FTP session closed
```

By processing these logs you can identify patterns in your log files. With the precedent example it is easy to undersand that the machine is beeing scanned by the IP 192.168.11.200. Based on these patterns and log entries, you can determine what the attacker is looking for, and potentially what tools they are using. Based of this knowledge, you can better secure and protect your systems.

Unfortunately there are scanning methods which wouldn't leave any logs in your files (steath mode for nmap), or the attacker's IP could be spoofed. Moreover are there much more complex methods to get inside a network and therefore the security of a system should rather be managed with an IDS.

## 4.3 Intrusion Detection System

### 4.3.1 Presentation

Intrusion detection systems (IDS) tools aim to detect computer attacks and/or computer misuse, and to alert the proper individuals upon detection. They serve three essential security functions: they monitor, detect, and respond to unauthorized activity by company insiders and outsider intrusion. Intrusion detection systems use policies to define certain events that, if detected will issue an alert. In other words, if a particular event is considered to constitute a security incident, an alert will be issued if that event is detected. Certain intrusion detection systems have the capability of sending out alerts, so that the administrator of the IDS will receive a notification of a possible security incident in the form of a page, email, or SNMP trap. Many intrusion detection systems not only recognize a particular incident and issue an appropriate alert, they also respond automatically to the event. Such a response might include logging off a user, disabling a user account, and launching of scripts.

There are nowadays two types of IDS:

- Host-based Intrusion Detection System (HIDS): these systems collect and analyze data that originate on a computer that hosts a service, such as a Web server. Once this data is aggregated for a given computer, it can either be analyzed locally or sent to a separate/central analysis machine.
- Network-based Intrusion Detection System (NIDS): analyzes data packets that travel over the actual network. These systems are best at detecting unauthorized outsider access and bandwidth theft/denial of service. In fact the packets that initiate/carry these attacks can best be detected before they arrive to there destination.

### 4.3.2 General architecture

To give an general idea on how IDS work we shall define here some typical terms about IDSs. The diagram 4.2 illustrates the relationships of some of the terms defined herein.

## **Activity**

Elements of the data source or occurrences within the data source that are identified by the sensor or analyzer as being of interest to the operator. Examples of this include (but are not limited to) network session showing unexpected telnet activity, operating system log file entries showing a user attempting to access files to which he is not authorized to have access, application log files showing persistent login failures, etc.

Activity can range from extremely serious occurrences (such as an unequivocally malicious attack) to less serious occurrences (such as unusual user activity that's worth a further look) to neutral activity (such as user login).

## **Administrator**

The human with overall responsibility for setting the security policy of the organization, and, thus, for decisions about deploying and configuring the IDS. This may or may not be the same person as the operator of the IDS. In some organizations, the administrator is associated with the network or systems administration groups. In other organizations, it's an independent position.

## **Alert**

A message from an analyzer to a manager that an event of interest has been detected. An alert typically contains information about the unusual activity that was detected, as well as the specifics of the occurrence.

## **Analyzer**

The ID component or process that analyzes the data collected by the sensor for signs of unauthorized or undesired activity or for events that might be of interest to the security administrator. In many existing IDSs, the sensor and the analyzer are part of the same component. In this document, the term analyzer is used generically to refer to the sender of the IDMEF message.

## Data Source

The raw information that an intrusion detection system uses to detect unauthorized or undesired activity. Common data sources include (but are not limited to) raw network packets, operating system audit logs, application audit logs, and system-generated checksum data.

## Event

The occurrence in the data source that is detected by the sensor and which may result in an IDMEF alert being transmitted. For example, 'N' failed logins in 'T' seconds might indicate a brute-force login attack.

## IDS

Intrusion detection system. Some combination of one or more of the following components: sensor, analyzer, manager.

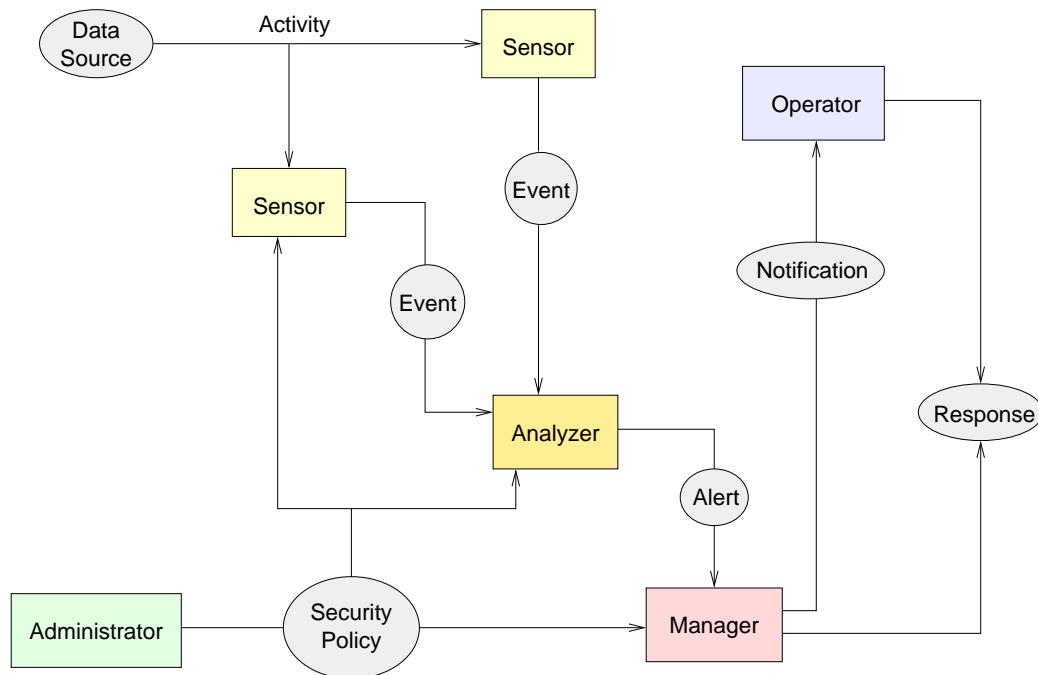


Figure 4.2: Typical IDS architecture

# Chapter 5

## Prelude IDS

### 5.1 What is Prelude IDS ?

Prelude is an hybrid intrusion detection system which has two major capabilities: it is both a Network IDS and a host based IDS. Intrusion detection is done by sensors which have complex packet analysis capability. It looks for abnormal behaviours and known signatures. If any malicious attack is found, an alert is raised and sent to the manager which centralize information over the prelude network. The central manager is in charge of the reporting and does also correlation in conjunction with a specific agent.

As the source of the problem can be inside “the apple”, remote connections between two prelude entities are done with secured links using OpenSSL protocol and certificates. If a sensor is running on the manager host, no secured connection is used.

Sensors play an important role in the Prelude architecture. They are capable of real time packet analysis and can provide fonctionnalities for both Network and Host Based IDS. They complete the low level work done by the core prelude packet analyzer (IP fragments reassembly) :

1. An evolved signature engine which permits to track specific datas within IP datagrams, UDP packets or TCP streams.
2. A set of protocol plugins which can detect specific protocols such as RPC, HTTP, FTP or TELNET.

3. A set of detection plugins that check for intrusion signatures within protocols.

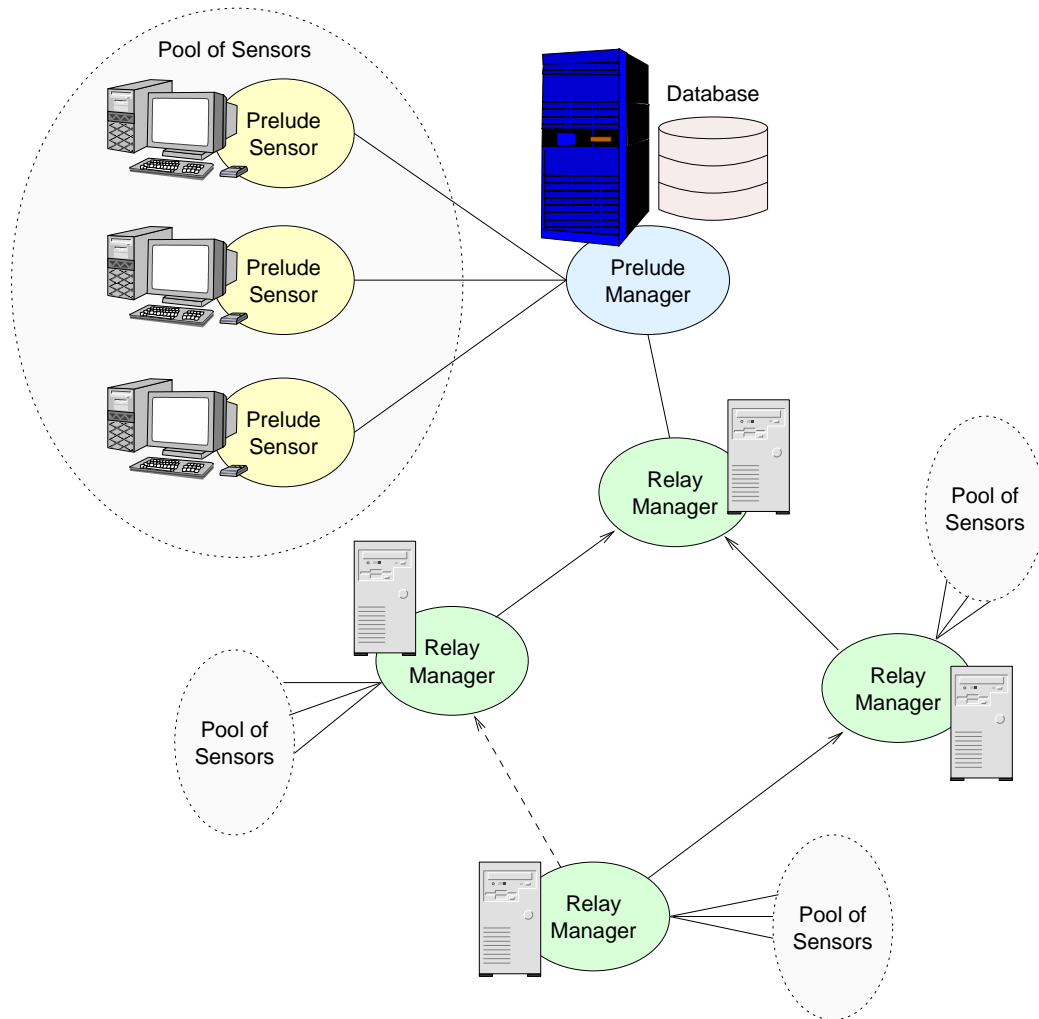


Figure 5.1: Overview of the Prelude architecture

## 5.2 Prelude-LML

Prelude-LML stands for Prelude Log Monitoring Lackey. It is a particular sensor which role is to add the capability to monitor any type of logs from any type of source to the Prelude IDS. At its actual state, it is only capable of receiving logs through UDP packets but it should be capable in a near future to monitor local files and receive logs through a secure connection using OpenSSL.

### 5.2.1 How Log Monitoring Lackey works

Figure 5.2 shows the internal organization of Prelude-LML. It has external interfaces (UDP, SSL and Local Files), dispatcher, plugins and internal communication facilities. Each module will be described longer in next sections.

#### UDP generic server

The UDP generic server is the only external interface available at that time. The Local Files interface should be done in a short amount of time followed by the integration of the SSL connection.

This simple module creates an UDP socket used within a thread to receive UDP packets. Each time a packet is received, its content is transmitted to a message reader function provided by the user.

Here is the interface of our UDP server implementation:

- `udp_server_t *udp_server_new( uint16_t port, udp_server_msg_reader_t *mreader, queue_t *queue );`

Create a UDP socket binded to the specified port. It use the mreader function to manage received packet through a queue if necessary.

- `void udp_server_start( udp_server_t *server );`

Start the server within a thread. When a UDP packet is received, it's transmited to the `udp_server_msg_reader_t` function provided during creation of the server. Strings given to the message reader function are statically allocated string buffer within the UDP server.

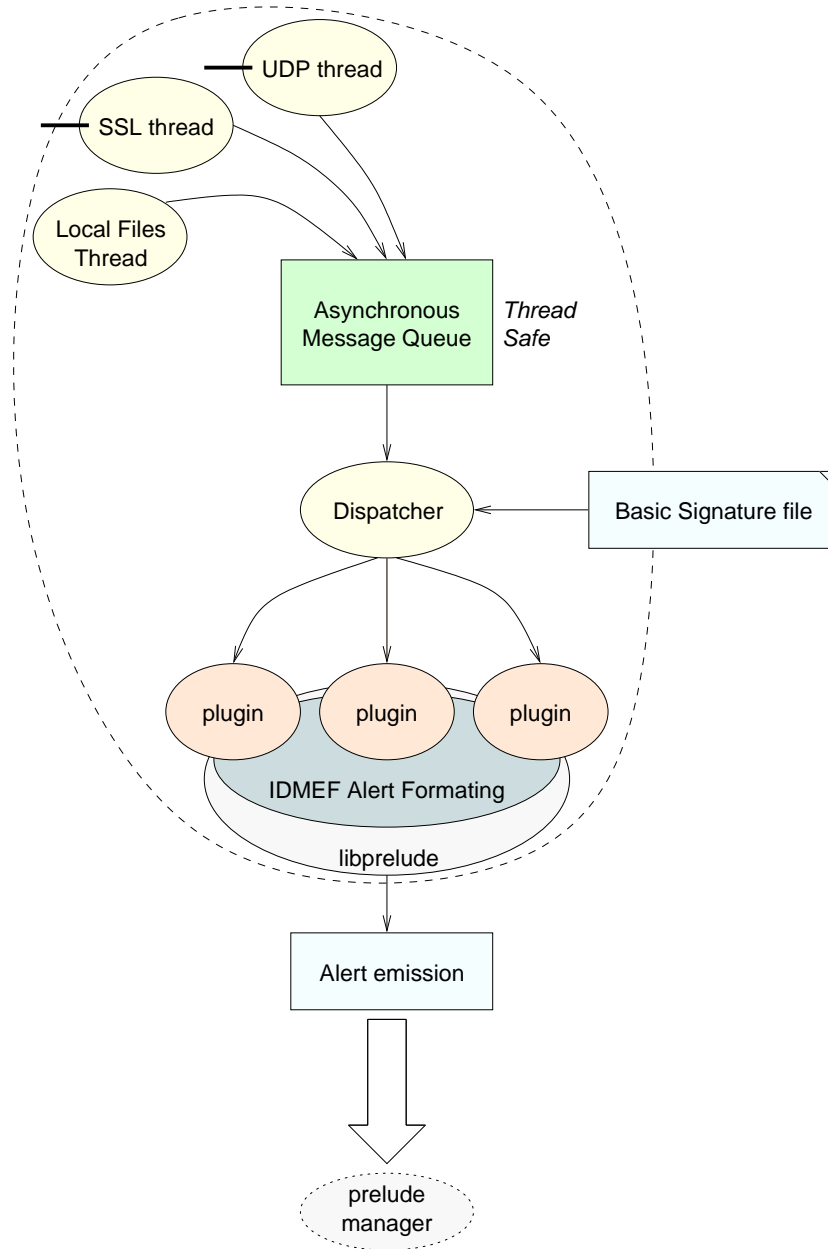


Figure 5.2: Overview of the Prelude-LML architecture

- `void udp_server_close( udp_server_t *server );`
- `typedef void (udp_server_msg_reader_t)( queue_t *queue, const char *str, const char *from );`

Type definition of a message reader function. The `str` argument is the raw UDP datas and the `from` argument is the IP address of the sender.

The length of data passed to the message reader has a maximum limit. We fixed this limit base on the fact that we are on an ethernet network, but can be changed at compilation time. Here is how we calculated it:

```
#define ETHERNET_LEN 1500
#define ETHERNET_HEADER_LEN 14
#define IP_HEADER_LEN 32 /* if no options in the IP datagram */
#define UDP_HEADER_LEN 8

/* we suppose we are on an ethernet network */
#define MAX_UDP_DATA_SIZE ( ETHERNET_LEN          \
                            - ETHERNET_HEADER_LEN \
                            - IP_HEADER_LEN        \
                            - UDP_HEADER_LEN )
```

### Thread safe polymorphic queue

The communication between external interfaces and the dispatcher is very important. As kernel buffers are limited in the UDP and TCP stacks, we need to provided an extensible queue that can handle a large number of entries with no loss of data.

We have decided to implement a polymorphic thread safe queue. We use a single linked list for that. It is polymorphic, because it manages only pointers and data structures referenced by theses pointers could be heterogenous (unless you want to delete the queue and objects stored in it). It is thread safe, because we use mutex to secure the pointers modifications during the push and pop operations. The pop operation has the particularity to be blocking if the queue is empty.

Here is the interface of the queue implementation:

- `queue_t *queue_new( queue_object_fun_t *delete_fun );`  
Create a queue. The argument `delete_fun` is a function used by `queue_delete()` to delete queued object. If `NULL`, objects are not freed by `queue_delete()`.
- `void queue_delete( queue_t *queue );`
- `void queue_dump( queue_t *queue, queue_object_fun_t *dump_object );`  
Print on `stdout` the content of the queue. The argument `dump_object` is a function used to dump each object.
- `int queue_empty( queue_t *queue );`  
return 1 if the queue is empty. return 0 if not.
- `void queue_push( queue_t *queue, void *object );`  
Push the object pointer in the tail of the queue.
- `void *queue_pop( queue_t *queue );`  
Pop the pointer at the head of the queue. It's a blocking operation if the queue is empty.

### Use of PCRE for dispatching

To dispatch logs to the proper plugins, we use a signature system based on regular expressions. The pattern matching system use the PCRE library (Perl-compatible regular expressions<sup>1</sup>) which has the same syntax and semantics than Perl 5.

Each time a log arrive to the dispatcher, its content is matched to regular expressions. When a valid pattern is found, the corresponding plugin (as specified in configuration files) is called with the log as argument to analyze it.

---

<sup>1</sup><http://www.pcre.org/>

Our actual implementation of this part of the dispatcher is not really satisfying (figure 5.3). The regex are implemented as a circular double linked list and each entry contains the compiled regex, the name of the plugin and the pattern matching options. First, options are not implemented yet. Second, plugins associated to the same regular expression are not grouped within the same entry (in a list for example). And finally, regular expressions and plugins management is not combined. We manage the plugin's pointers with a hashtable which is used to retrieve them fastly with their names. It means that for a particular log, we check all the regex, when a valid pattern is found, we call the corresponding plugin through the hashtable (we use plugin names as keys).

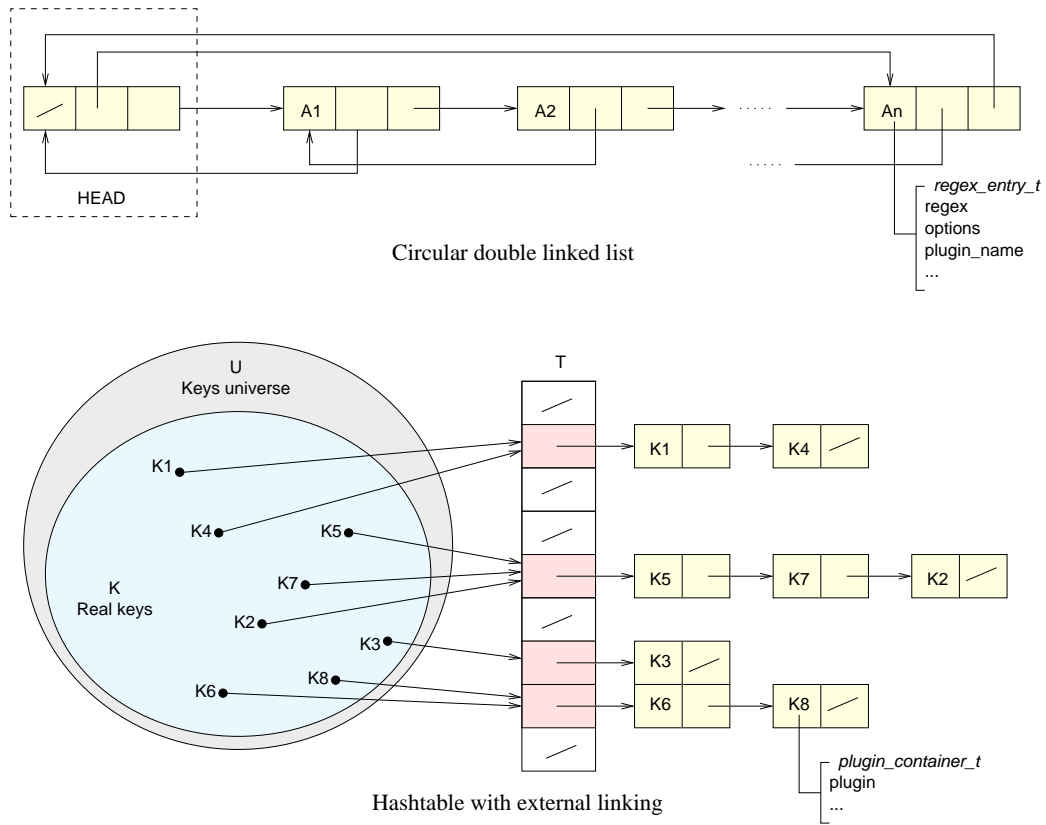


Figure 5.3: Data structures

The actual worthless implementation is both time and memory consuming in some cases. It would be a good thing to store the registered plugins in a list, and have references to cells of this list from each regex entry (depending on configured associations). These references could be stored in a list, so that we could group plugins that are associated to the same regex (figure 5.4). Using grouping (different plugins associated to the same regex) is less time consuming and avoiding the use of hashtable by a direct references to plugins is less memory consuming.

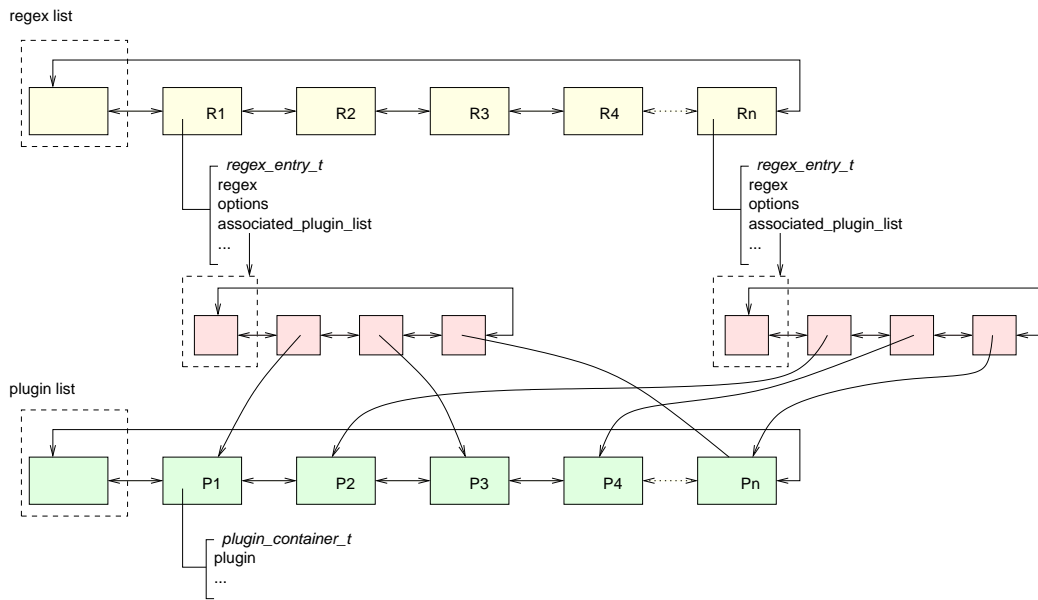


Figure 5.4: Data structures

The regular expressions used by the dispatcher are defined in a specific configuration file found in `$prefix/etc/prelude-lml/plugins.rules` which have the following syntax:

```
# plugin-name-list      pcre-options          regex
#
Pax                     -                      .* PAX:.*
Debug                   -                      .*
```

```
# pcre-options: (not implemented yet)
#
# x          exclusive
# i          case insensitive (default is case sensitive)
# -          no options
```

## 5.2.2 Interaction with plugins

### Philosophy

The philosophy in the Prelude-LML development is to keep it away from overload of specificities that can fit in plugins. In our first ideas of what Prelude-LML should be and do, we often thought about log format (BSD Syslog protocol, multi-lines logs, etc...) before the plugin level. Now we consider that Prelude-LML is just a log relay to the plugins through the dispatcher. To increase its genericity in log treatment, it doesn't need to parse the log content before transmit it to the plugins. It is certainly more memory consuming, but it will keep the core of Prelude-LML clean and light. As plugins can do behaviour analysis on logs, stateless plugins can't be mandatory and they have to manage the memory by themselves if they need it.

### Interfaces

The plugin mechanisms (loading, configuring and function calls) used by Prelude-LML are the same ones found in sensors and are mostly provided by libprelude. As the purpose of Prelude-LML is to gather any type of logs from various type of sources (from local files, from hosts and equipments using the BSD Syslog protocols, etc...) we have decided to use a very simple interface between the dispatcher and the plugins. It is defined in *log-common.h*:

```
typedef struct {
    char      *source;
    struct tm  time_received;
    char      *log;
} log_container_t;
```

It's a very simple structure. The *source* field contains the origin of the log. It could be a remote host, we then put its IP address retrieved when

receiving the UDP packet, or a file name if it came from a local file. The *time\_received* field contains the time at which the log have been processed by the message reader, it's then often different from the timestamp contained in the log itself. And finally, we have the *log* field which contains the original log message with no modification.

For information purpose, here is the definition of *struct tm* provided by the libc:

```
struct tm {
    int    tm_sec;    /* seconds 0-59 */
    int    tm_min;    /* minutes 0-59 */
    int    tm_hour;   /* hours 0-23 */
    int    tm_mday;   /* day of the month 1-31 */
    int    tm_mon;    /* month 0-11 */
    int    tm_year;   /* year since 1900 */
    int    tm_wday;   /* day of the week 0-6 since sunday */
    int    tm_yday;   /* day in the year 0-365 */
    int    tm_isdst;  /* daylight saving time */
};
```

## Available plugins

Plugins are located in `$prefix/lib/prelude-lml/` and are automatically loaded by Prelude-LML.

**Debug plugin:** It has been ripped from the Debug plugins found in the detection directory of Prelude Sensor. It's a very simple plugin which send an alert to the Prelude Manager for each log it received.

**PAX plugin:** PaX<sup>2</sup> is an implementation of non-executable pages for IA-32 processors (i.e. pages which user mode code can read or write, but cannot execute code in). Since the processor's native page table/directory entry format has no provision for such a feature, it is a non-trivial task.

The project was designed to provide Linux with protection from buffer overflows. Making parts of the memory pages read/write access enabled, but not executable provides the protection.

---

<sup>2</sup><http://pageexec.virtualave.net/>

Of course the PaX kernel patch generate logs when a buffer overflow is detected. Some guys had wrote a PaX log parser which is able to generate IDMEF alerts and send them to the Prelude Manager. We have re-used this code and integrated it in the Prelude-LML code as a plugin.

## 5.3 Future extensions

The actual implementation of Prelude-LML is far from complete. As we started it from scratch, we often had to reconsider our choices and this process slow down the development. Many things have to be done. Here is a non exhaustive list.

### 5.3.1 Simple plugin implementation

The purpose of this plugin is to provide a simple way to send alerts based on pattern matching. It could be used when no suitable plugin have been found by the dispatcher. We then just have to declare it at the end of the *plugins.rules* file, associated with every kind of logs.

```
# plugin-name-list      pcre-options          regex
#
Pax                     x                     .* PAX:.*
NTSyslog                x                     .* NTSyslog:.*
Simple                  -                     .*
```

The simple plugin could have his own configuration file in `$prefix/etc/prelude-lml/simple.conf` containing a list of regex to match and informations to put in generated alerts. Something like this could be interesting:

```
# /usr/local/etc/prelude-lml/simple.conf

rule1 {
    regex = '((<\d{1,3}??>)([A-Z][a-z]{2}? \d{1,2}??)';
    alert.analyzer.model = 'False timestamp';
    alert.analyzer.class = 'Generic class';
    alert.assessment.impact.severity = 'impact_medium';
    alert.assessment.impact.completion = 'failed';
    alert.assessment.impact.type = 'other';
```

```
    alert.assessment.impact.description = 'medium impact';
    alert.action.category = 'notification_sent';
    alert.action.description = 'no action';
    alert.assessment.confidence.rating = 'high';
    alert.classification.url = 'http://www.hijacking.net/';
};
```

The major problem is to define a suitable Lex/Yacc grammar in conformance with IDMEF specifications to ease the write of IDMEF alerts.

### 5.3.2 Local file monitoring

While fetching datas from remote logs, one of the future interesting capability of Prelude-LML is to be able to gather informations from local files. For some specific logs, which are not managed by the syslog daemon, it will be possible to have real-time analysis with alert centralization.

Each tracked file will have a read-only file descriptor opened. Using the *poll()* system call we will monitor events on several file descriptors and dispatch them to plugins. We are thinking about having a dedicated thread to track a limited number of files (100? 200?).

The list of files to monitor will be defined in the standard Prelude-LML configuration file (`$prefix/etc/prelude-lml/prelude-lml.conf`) as a simple space separated list.

### 5.3.3 SSL integration

The integration of the SSL protocol is not a trivial process. It's not just ripping the code from the Prelude Manager without understanding it. In order to make it happen it's worth spending time on the underlying logic.

#### Server Logic/Server Generic

The concept of server as used in the Prelude Manager is composed of two major entities:

- server-logic

Run within a thread to handle events on a pool of hundreds sockets.

- `server-generic`

Wait for new connections and manage both plain-text and SSL authentications.

We have built a diagram (figure 5.5) of the *server logic/server generic* system used by the Prelude Manager. This figure contains only essential notions which have a relation to the connection management.

To use such system, we have to implement only the high level functions and use the facilities offered by *server-generic.c*:

- `xx_server_accept_connection()`;  
Called when a connection authentication has succeed.
- `xx_server_read_connection()`;  
Called when new datas have been retrieved on the connection.

## How to integrate SSL in Prelude-LML

As we know have a pretty good knowledge of the *server generic/server logic* system, the integration of SSL connections in Prelude-LML is not so difficult.

What we want is to implement *log-server.c* wich will be our log server over an SSL connection. First, the dispatcher have to be moved from the main thread to a new thread so that *server-generic.c* can manage new connections in the main thread. Then, we implement a special *log\_server\_read\_connection()* which receive logs through messages and push them on our thread safe queue.

That's it! Everything (certificate, SSL transactions, etc...) is now managed by *libprelude*.

## 5.4 Compiling Prelude-LML

You first need to build/install *libprelude* and *libpcre*. Then, it is pretty straight forward. After unpacking the archive, launch the *autogen.sh* script, launch the *configure* script and use *make* followed by *make install* to build and install everything.

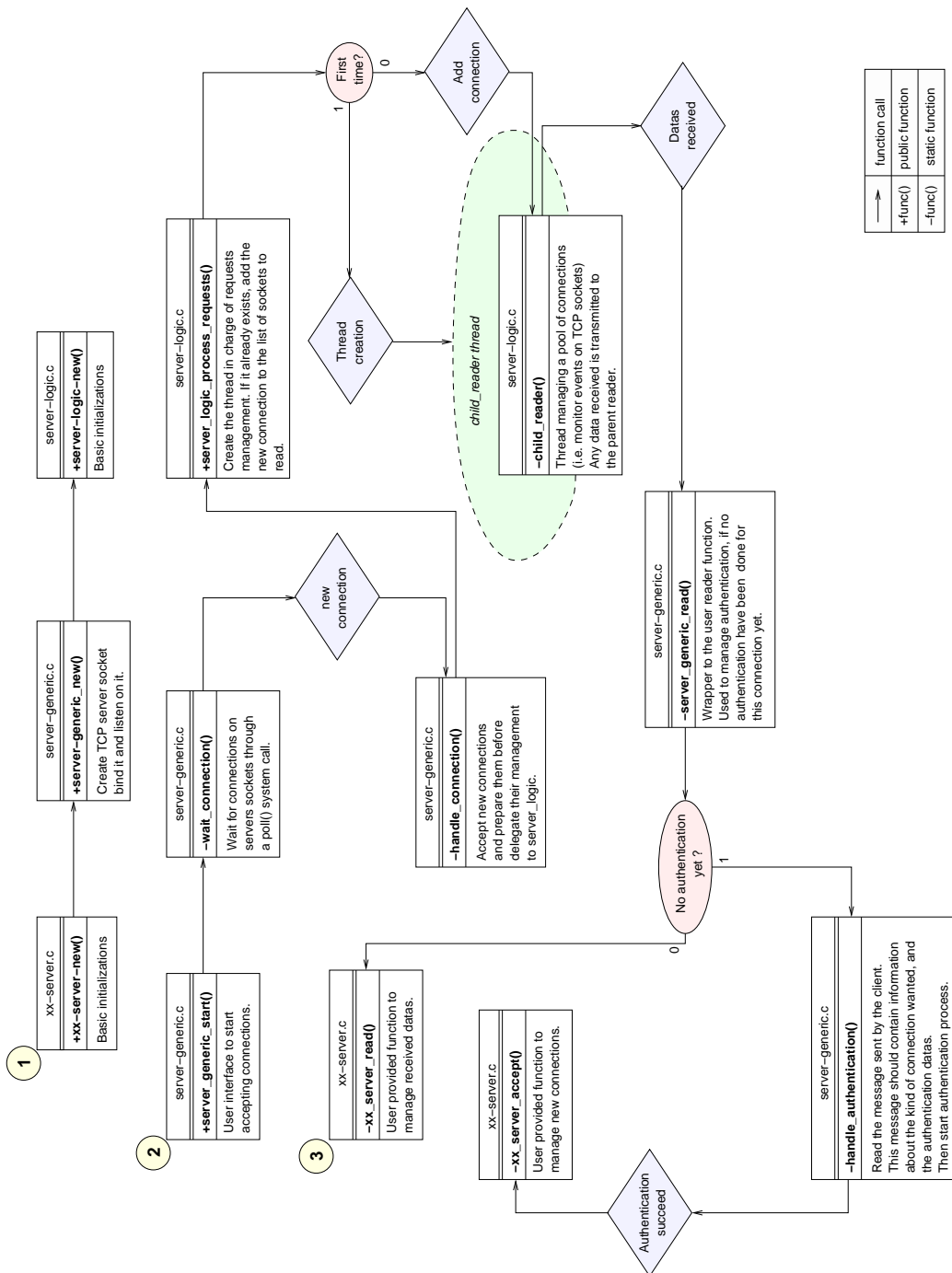


Figure 5.5: The logic behind Prelude SSL servers

# Chapter 6

## Conclusion

As we have explained all along this report log centralization is the first step to manage and secure information systems. There is still a lot to do about integrating heterogeneous environments such as Windows and Unix.

As a matter of fact have we in the first time interested ourselves to the brand new IDMEF<sup>1</sup> format. The purpose of the Intrusion Detection Message Exchange Format is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them. Unfortunately IDMEF is under development and we haven't found much documentation about it except the working draft<sup>2</sup>.

Prelude LML already relies on a IDMEF structure for the communication with the Prelude Manager. Its role is to host plugins. At the moment there are only two plugins : a debug plugin and a PAX plugin. We haven't got a enough time to create a windows plugin but all the information about Windows events could give a good basis.

---

<sup>1</sup>Intrusion Detection Message Exchange Format

<sup>2</sup><http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt>

# Appendix A

## Windows events

### A.1 Audit events

Additional fields are located within the Description and are based on the category of the event<sup>1</sup>:

```
Audit category
  event number - event name
  audit fields
System Event
  512 - System Restart
  513 - System Shutdown
  514 - Authentication Package Load
    Authentication Package
  515 - Logon Process Registered
    Logon Process Name
  516 - Some Audit Event Records Discarded
    Number of audit messages discarded
  517 - Audit Log Cleared
Logon/Logoff
  Information recorded
    1.Reason
    2.User Name
    3.Domain
```

---

<sup>1</sup><http://www.eventid.net/docs/ntaudit.asp>

4.Logon ID

5.Logon Type - the method of logon.  
(i.e. 2 for normal logon, 3 for remote)

6.Logon Process

7.Authentication Package

528 - Successful Logon (2-7)

529 - Unknown Username or Bad Password (1-3, 5-7)

531 - Account Currently Disabled (1-3, 5-7)

534 - Logon Type Restricted (1-3, 5-7)

535 - Password Expired (1-3, 5-7)

537 - Unsuccessful Logon (1-3, 5-7)

538 - User Logoff (2-5)

Object Access

560 - Object Open

Object Server

Object Type

Object Name

New Handle ID - if its a kernel object,  
it is unique only to that process.  
If it is a server object, it is unique  
across all processes.

Operation ID - this is used to group events that are  
part of the same operation. It is unique  
only to the process performing the operation.

Process ID

Primary User Name

Primary Domain

Primary Login ID

Client User Name

Client Domain

Client Logon ID

Accesses

Privileges

562 - Handle Closed

Object Server

Handle ID

Process ID

Privilege Use

576 - Special Privilege Assigned  
User Name  
Domain  
Logon ID  
Assigned

577 - Privileged Service Called  
Service  
Server  
Process ID  
Primary User Name  
Primary Domain  
Primary Logon ID  
Client Use Name  
Client Domain  
Client Logon ID  
Privileges

578 - Privilege Object Operation  
Object Type  
Object Name  
Object Server  
Process ID  
Primary User Name  
Primary Domain  
Primary Logon ID  
Client Use Name  
Client Domain  
Client Logon ID  
Privileges

Account Management  
Information recorded

- 1.Member
- 2.New/Target Account Name
- 3.New/Target Domain
- 4.New/Target Account ID
- 5.Caller User Name
- 6.Caller Domain
- 7.Caller Logon ID
- 8.Privileges

- 624 - User Account Created (2-8)
- 642 - User Account Changed (2-7)
- 630 - User Account Deleted (2-7)
- 632 - Global Group Member Added (1-7)
- 633 - Global Group Member Removed (1-7)
- 635 - Local Group Created (2-7)
- 636 - Local Group Member Added (1-8)
- 637 - Local Group Member Removed (1-8)
- 639 - Local Group Changed (2-7)
- 638 - Local Group Deleted (2-7)

Policy Change

- 608 - User Right Assigned
  - User Right
  - Assigned To
  - User Name
  - Domain
  - Logon ID
- 609 - User Right Removed
  - User Right
  - Removed From
  - User Name
  - Domain
  - Logon ID
- 612 - Audit Policy Change
  - New Policy
  - User Name
  - Domain
  - Logon ID

Detailed Tracking

- 592 - New Process Has Been Created
  - New Process ID
  - Image File Name
  - Creator Process ID
  - User Name
  - Domain
  - Logon ID
- 593 - Process Has Exited
  - Process ID

User Name  
Domain  
Logon ID  
594 - Handle Duplicated  
Source Handle ID  
Source Process ID  
New Handle ID  
Target Process ID  
595 - Indirect Access to Object  
Object Type  
Object Name  
Via Handle ID  
Object Server  
Process ID  
Primary User Name  
Primary Domain  
Primary Logon ID  
Client User Name  
Client Domain  
Client Logon ID  
Accesses Granted

## A.2 Interesting events to look for

### Logon/Logoff

- Unknown Username or Bad Password - Security Event 529 - Failure Audit
- Unsuccessful Logon - Security Event 537 - Failure Audit

These are simple bad logins. Make sure you are auditing these at each workstation and send these messages upstream via a log dump batch process or via syslog.

### Violations of Account Policies

- Account Logon Time Restriction Violation - Security Event 530 - Failure Audit

- Account Currently Disabled - Security Event 531 - Failure Audit
- Account Has Expired - Security Event 532 - Failure Audit
- User Not Allowed to Log on - Security Event 533 - Failure Audit
- Logon Type Restricted - Security Event 534 - Failure Audit
- Password Expired - Security Event 535 - Failure Audit

### **System Events**

- System Restart - System Event ID 512  
 The event occurs when the system has been rebooted. Perhaps you will want to look for system reboots outside of business hours.
- Some Audit Event Records Discarded - System Event ID 516  
 If you get this message, it could mean several things: your log file retention settings aren't holding the load that you thought they would, you are logging too much in your security audit settings, or someone's flooding your event log legitimately or illegitimately.
- Audit Log Cleared - System Event 517  
 If you get this message, someone or something (ie. a script or program) has cleared the security log. If this message catches you unaware, you may have a problem.

### **User and Group Managemen**

There are many account level auditing events, but I am only listing a few here:

- User Account Created / User Account Deleted - Security Event 624 / 630
- Change Password Attempt - Security Event 627
- Local Group Member Added / Global Group Member Added - Security Event 636 / 632

I would watch for any events of this type that included the Administrators / Domain Administrator group or any other trusted group on the system.

- User Account Changed - Security Event 642

This event could be registered frequently depending on how much user management you need to do.

- Domain Policy Changed - Security Event 643

### **Policy Change**

- User Right Assigned / Removed - Security Event 608/609

‘A change in the user rights policy (which is typically found in User Manager) has occurred. Some installation programs will do this - and some viruses and trojans do, too!

- Audit Policy Change - Security Event 612

Someone has changed the audit policy. You will see this when you start to refine your auditing on the system, but after that, you should never see this event.

- New Trusted Domain / Removing Trusted Domain - Security Event 610/611

These messages are rather self-explanatory.

# Appendix B

## First proposal

*In the beginning of the project we were more focused on the interaction between the Windows and the UNIX environments. Based on that, you will find our first proposal in this section. Then it appeared that the need of a specialized Windows tool was not a primary requirement, and the lack of log centralization in Prelude was more important and should be addressed first.*

### B.1 Goals

At the moment, the Prelude [14] system is only working on some UNIX-like operating systems. It's okay for the Prelude Manager which centralize the information but it's a limitation if you have other types of computers running on your local network and if you want to enhance the security of it.

According to this problem, the main goal of our project is to centralize as most as possible all types of logs from any computer connected to a LAN. By all types of computers it means that any host, with a modified version of Syslog running on it, should be able to contact the Prelude Manager in order to emit a security warning or to simply archive the logs. Of course, such host could be a NT/2k system or other UNIX-like system not supported by Prelude.

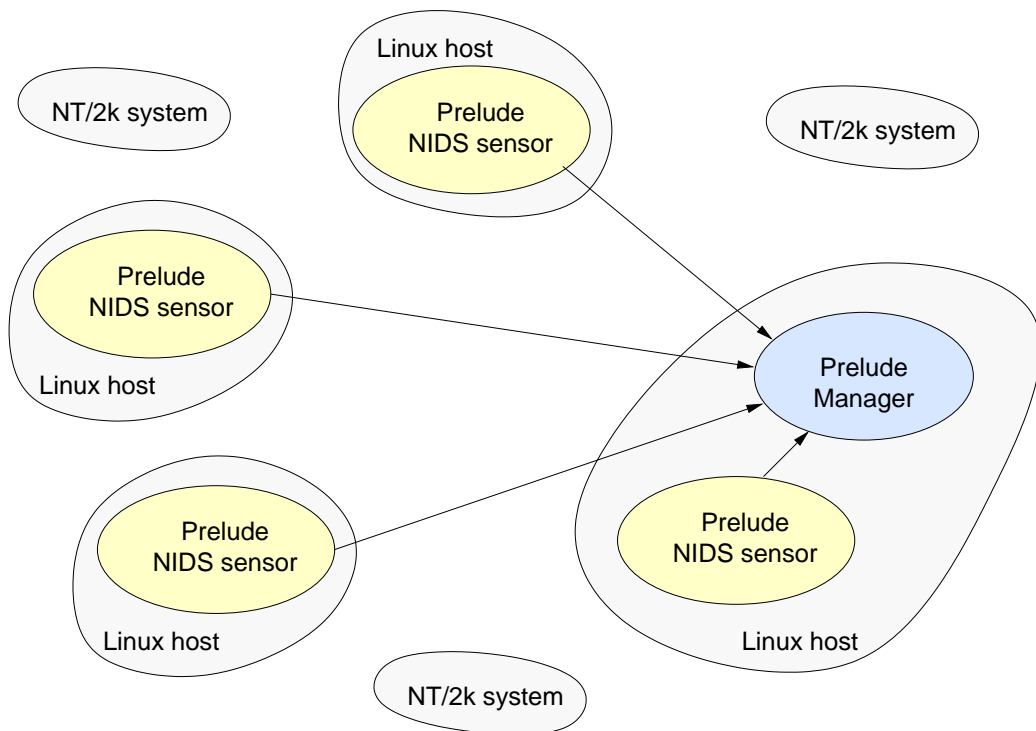


Figure B.1: Simple overview of the Prelude architecture

## B.2 Solution

Our solution is mainly focused on the Win NT/2k architecture by modifying an existing tool called NTSyslog [11] available for free under the GPL license. In the standard distribution of NTSyslog (figure B.2), the logs are transmitted from a WinNT/2k system to a syslog host using the UDP protocol which is known to not be secure.

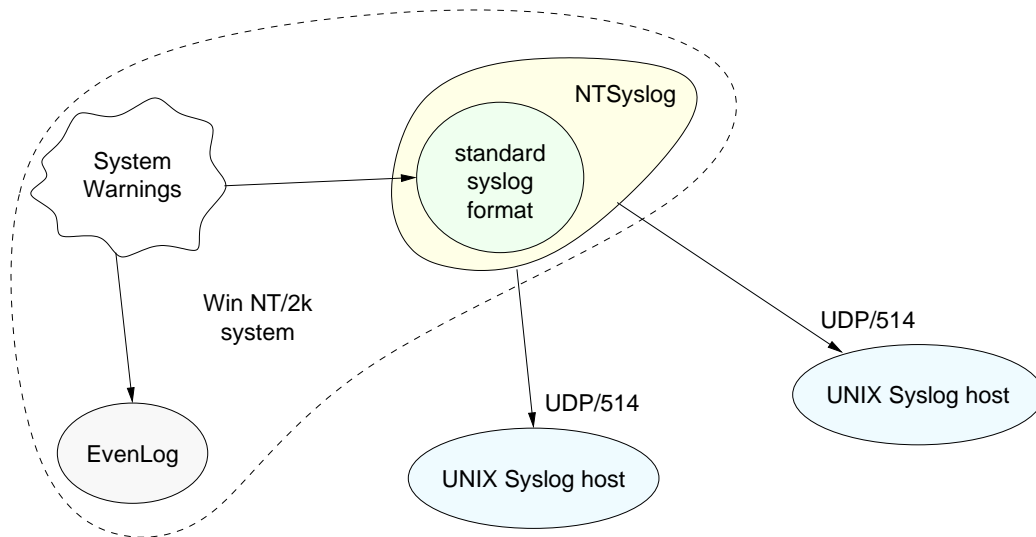


Figure B.2: NTSyslog actual architecture

What we want is to add some features to NTSyslog so that it will be able to securely transmit data to a Prelude Manager with a simple pre-filtering system with the ability to remove useless system warnings (less warnings == less bandwidth used). Other features are also illustrated in our architecture proposal (figure B.3). One of them is the IDMEF-binary converter. It will convert the filtered logs into the IDMEF specification in a binary form (C structure) instead of a text form like standard XML documents. It will save network bandwidth during transmission and ease the decoding process on the Prelude Manager side.

There are other minor changes we can add to NTSyslog to improve its efficiency. First, it could be interesting to save logs in a text file on each

WinNT/2k machine. If the connection to the Prelude Manager is broken we still have a trace of what happened on this machine. We can also secure this local log file by using the PEO protocol [9] as used in Modular Syslog [2]. That prevents any hidden modification from an intruder which has gained the “root” account (or the “Administrator” account) on the attacked host.

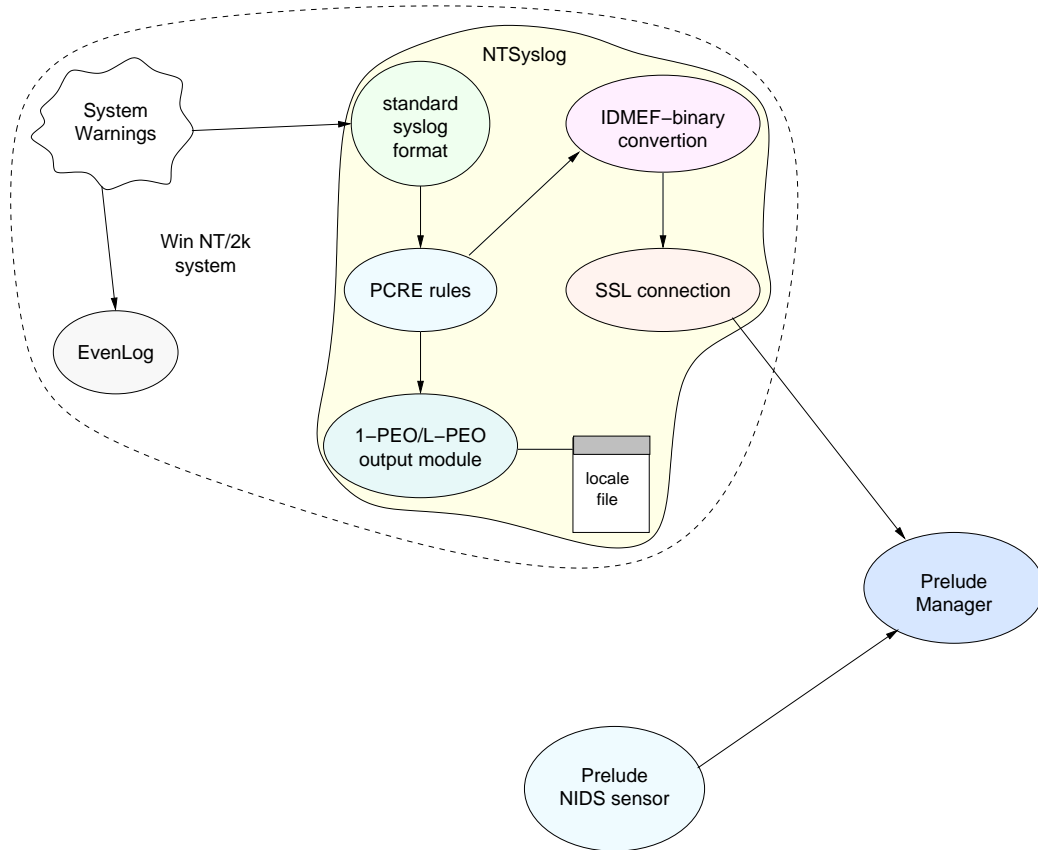


Figure B.3: Modified NTSyslog architecture

If we have enough time we would like to propose some interesting rules for the regexp filter. A sort of mixed-up between good security and less log interference.

Another nice thing would be to test how NTSyslog react with advanced scanners [6] and IDS testers [3], so that we could tune the filtering rules.

## **B.3 Details**

The command line part of NTSyslog have been successfully compiled with cygwin [10] (using the MingW-environment package) under Win2k. We will use this kind of environment to implement the new features on the Windows platform.

In particular, we will try to use libpcre for the regexp filter and OpenSSL for the secure connection between NTSyslog and a Prelude Manager. They are both provided as a package in the CygWin environment.

# Appendix C

## Glossary

**Host-based IDS :** Looks at only packets addressed to the machine.

**Hybrid IDS :** It consists of a misuse detection component detecting mostly anomalies based on a statistical approach wrapping events as intrusive if they are deviant from the expected behavior. It combines Network IDS and Host-based IDS.

**IDMEF :** Intrusion Detection Message Exchange Format which define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them.

**Network IDS :** Sniffing at all packets in each network segment.

**PEO :** Primere Estado Oculto (Hidden First State).

**Secure logging :** Ability to record a given amount of information on a given storage media and be able to check the authenticity of that record later.

**VCR :** Vector de Claves Remontante (Remounting Key Vector).

# Bibliography

- [1] Intrusion detection working group. <http://www.semper.org/idwg-public/>.
- [2] Ariel Aizenberg, Alejo Sanchez, and Claudio Castiglia. Modular syslog, 2001. <http://msyslog.sourceforge.net/>.
- [3] Stephane Aubert. Idswakeup, 2000. <http://www.hsc.fr/ressources/outils/idswakeup/>.
- [4] Inc. B. Feinstein/Guardent, G. Matthews/CSC/NASA Ames Research Center, and J. White/MITRE Corporation. The intrusion detection exchange protocol, november 2001. <http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-03.txt>.
- [5] D. Curry, H. Debar, and Merryl Lynch/France Telecom. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition, december 2001. <http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt>.
- [6] Renaud Deraison. Nessus. <http://www.nessus.org/>.
- [7] Paul Innella. The evolution of ids. *Security Focus*, november 2001. Tetrad Digital Integrity, LLC. <http://www.securityfocus.com/infocus/1514>.
- [8] Paul Innella and Oba McMillan. An introduction to intrusion detection systems. *Security Focus*, december 2001. Tetrad Digital Integrity, LLC. <http://www.securityfocus.com/infocus/1520>.
- [9] Emiliano Kargieman and Ariel Futoransky. Vcr and peo revised. Technical report, CORE SDI S.A., october 1998. <http://www.corest.com/files/files/11/PE0.pdf>.

- [10] Inc. RedHat. Cygwin : Unix environment for windows. <http://sources.redhat.com/cygwin/>.
- [11] Sabernet.net. Windows nt syslog service, 2001. <http://ntsyslog.sourceforge.net/>.
- [12] Alejo Sanchez. Do you trust your system logs ? *Daemon News*, december 2001. [http://ezine.daemonnews.org/200112/log\\_protection.html](http://ezine.daemonnews.org/200112/log_protection.html).
- [13] Matthew Tanase. The future of ids. *Security Focus*, december 2001. <http://www.securityfocus.com/infocus/1518>.
- [14] Yoann Vandoorselaere. Prelude before the tempest: Hybrid intrusion detection system. <http://www.prelude-ids.org/>.