An intrusion detection system based in the gathering of Linux Syslog Logs from Linux, Windows NT and Snort

Jose Antonio Fernandes Salvador

September 13, 2003

MSc Computer Systems Security
CS4T01 MSc. Project
Enrolment No. 02029685
Course 2002-2003
Supervisor: Andrew Blyth
Second assessor: Gaius Mulley

Contents

1 Acknowledgement

In first place thanks to my family that has supported me all my life in all the decisions I had done, specially my mother Amalia Ferreiro Martins. Thanks to my sister Monica Fernandes Salvador for her economic help.

Thanks for the company EDEX that keep me working abroad and that allows me to finance this master and for their support in general. Special thanks to Andoni Eguiluz Data center director and teacher in the Universidad de Deusto that has help me and give me support since I know him some years ago.

My gratitude to the members of the Open Source Users Group E-GHOST for their help in technical problems that I faced during the project and that they give me solutions very fast through their mailing list.

My special thanks to all the friends that I did during this master from different nationalities that help me in this year and made me to enjoy this year. The list would be really big.

My thanks to all the friends in Spain that help me and have support me, special thanks to my mates and teachers of Penkat Silat, my sensei of JuJitsu and my friend and confident Maria Garcia Rebollar.

My thanks to the La Haceria and all the people I meet there that give something special that helped me to get back home.

2 Introduction

[Pro02] [Wad00] There are different layers to assure the security of the computer systems in a company. There is the perimeter defences and gateways like firewalls and routers, the network security with Network Intrusion Detection Systems and host security with personal firewalls, Intrusion Detection Systems, System policies, Local accounts, etc.

However all of them defends from known attacks or techniques. But how there is it known that they are configured properly and that they works well?

The answer is in the logging capabilities of all the systems. But who is going to analyse all those different logs that are dispersed and each one with a different format? And not only that, there is a huge amount of them. Therefore there are many times when the information is needed from different devices or systems at the same time to create a picture of what has happened with a particular form of strange behaviour.

The answer is a centralized log that stores the information from all the systems in a suitable format for later analysis or real time analysis. However, actually there is no universal way of logging information in a centralized manner from different kinds of systems. For this reason the most universal one that is Syslog is going to be taken and complemented to be able to receive events from several systems.

But actual Syslog stores the events in a file, and extracting the information from that file is the most time consuming and challenging part of work. There are several scripts using regular expressions that allow for analysis, but are slow and tedious. The store in a database increases the velocity of searches and allows the use of SQL for doing queries and built programs to analyse or show the information, like graphic visualization.

This centralization allows easier protection of the logs, in a secure server, and easier access to the information because it is all together.

The main advantages of storing the information in a database is a much faster log storing, even on slow boxes and better searching methods.

3 Project Aims and Objectives

The aim of this project is to develop an event logging architecture that is able to centralize the logging capabilities of the different systems like Linux, Windows NT and Snort. To store it in a proper format in a database that does later analysis easier. Using, improving or complementing technologies and mechanisms that actually exist in the Syslog architecture.

This aim is going to be driven by the following objectives:

- A study of the actual events logging capabilities in Linux, Windows and Snort.
- A study of how the capabilities can inter-operate with a Syslogd server.
- Implement an architecture that centralizes the events logging from the different sources.
- Evaluation of the architecture.

4 Project Method

[Lev00] [Cha98] The Rapid Prototyping Model is going to be followed because, there is little knowledge in this area and there are a lot of unresolved questions at this time, therefore the project can be considered of high risk.

The approach is going to be to develop different prototypes, ranging from easy to difficult, so the knowledge gained in the previous prototypes will help in the actual one. Once one is finished then the next one will begin, and look if it is possible to do all of them or not. The following procedures will be attempted:

- Gather information from Syslog.
- Gather information from NT.
- Gather information from Snort.

5 Project Plan

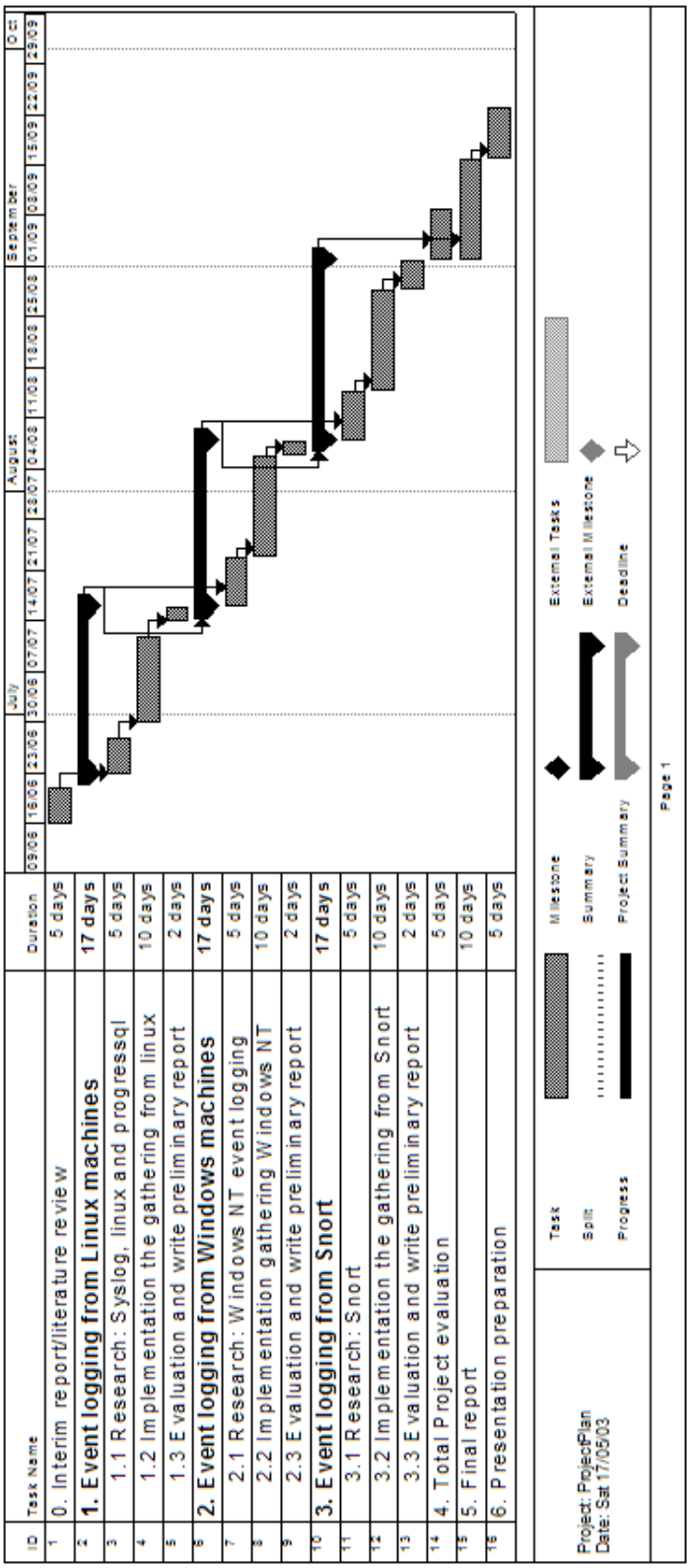

Figure 1: Project Plan

This project has been divided into three parts:

1. Research and literature review: Research into what has been already done, and what the existing solutions are. How the approach in this project can contribute . Research in the logging capabilities that exist in the different systems. And how they can be integrated and stored in an unique database. The time estimated for research is 4 weeks.

2. Implementation: Three prototypes are going to be developed, starting from the easiest one. There is going to be a lot of work in configuring systems too. The time estimated for implementation is 6 weeks.

3. Evaluation: The work done, objectives reached and comparison with other approaches will be evaluated. The time estimated for evaluation is 4 weeks.

But there is not going to be a block of 4 and 6 weeks. There is going to be a week of research followed with one or two weeks of implementation. Because the prototyping model is going to be used and because the research and implementation of one prototype will give a lot of knowledge that can be used in following prototypes.

To avoid times of inactivity and delays in the project plan, the implementation and research sometimes will be done at same time. This is because there is much uncertainty since it is the first time the author has undertaken a project of this kind.

6 Detailed project plan

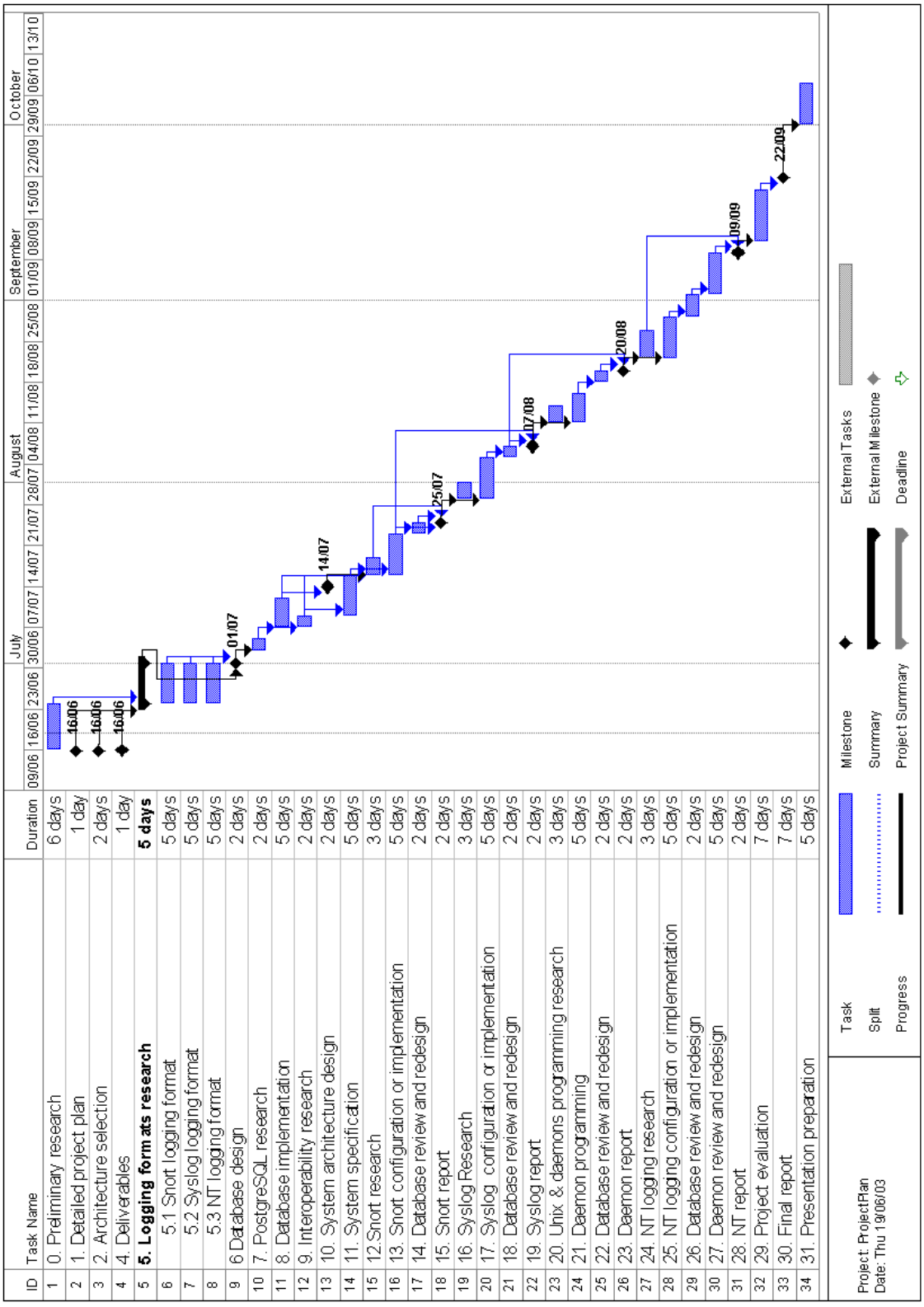

Figure 2: Detailed Project Plan

7 System Architecture

7.1 Free Software

Free Software has been used in all the system for two reasons:

- Technical and educational advantages of open source well proved.
- The author identifies himself with Free Software philosophy.

Free with the meaning of freedom not "no money".

7.2 Operating System (OS)

The central Linux machine is using the Debian distribution because it is becoming the most commonly chosen one for security and because it is the author's favourite.

The reason why Debian is getting his popularity for security is the process that any package must follow in order to be in the stable distribution [DEB] and the proactive security conscience of Debian community. There are only one or two stable releases per year, with packages that are aged and very well tested. In addition, security patches usually are back ported from more recent versions, and there is an apt source specifically for security updates for packages in stable, something the other distributions do not have. A version has to move from experimental, through unstable and testing to stable [DIS].

Some other advantages of this distribution are:

- His dependable and efficient software packaging system (apt) and excellent documentation.
- Debian packages integrate very well into the entire system. For all packages of the distribution the whole source is available. The entire distribution is free which means that everybody can improve packages and still distribute them.
- There are no secret pieces of code that are held back letting users worry about secret back doors or secret functionality.
- The APIs are completely available to help developers make better applications.
- The Debian Project maintains an open bug tracking system where everybody is able to report and check on bugs. Bugs are normally be fixed within a few days.

7.3 Programming language

The chosen one is C. It is the high level programming language that is more close to the machine. For this reason it has some of the advantages of optimisation of assembler, but at the same time it does not so hard to understand, read and remember.

It has the disadvantage or advantage of allowing a total control of the machine, so the programmer can do anything but must be very careful.

Finally is the authors favourite programming language, he likes the way it can be done a lot in a few space. This can be a problem if the program is not well structured because can be difficult to read, but on the other hand if well written u can see more easily what the code does without getting lost in details.

7.4 Logging server

Syslog it is the standard logging server. It is very simple and it lack a lot of features. That is the reason of appearing some replacements like Ng-Syslog [Sch] that provide security. But it has been chosen because his simplicity and the fact that is available in most of the Unix Operating Systems and devices.

There is a tendency in Unix Systems to create simple programs that performs a task very well. If we want a logging server, we will have a program that does logging. If we want security, we will use OpenSSH [Ope], the free version of Secure Shell protocol. So, Syslog is used and instead of modifying it we will try to write programs that complements it. The only problem to this approach is how the different programs will communicate, but interprocess communication has a lot of possibilities and is very well documented.

7.5 Database

The chosen database is PostgreSQL [Pos] that is the most powerful Free Software database. The source code is available freely and it can be accessed easily through the library libpq-fe.h.

PostgreSQL was originally developed in the UC Berkeley Computer Science Department. It has been a pioneer in many of the relational and object concepts that later were incorporated in many commercial databases. It has support for the language SQL92/SQL3, transaction integrity, and type extensibility. It is a public domain and free software of the Berkeley's database.

7.6 Windows client logger

The following Windows client logger are going to be tested:

- NTsyslog: It is Free Software. Can be used in Windows NT/2000/XP [NTS02].
- EventReporter: Can be used in Windows NT/2000/XP [EVE].
- BackLog: GNU Public Licence [BAC].

7.7 Snort

[Tol00] Reason to choose this as Network Intrusion Detection System (NDIS):

- It is that is a lightweight program.
- The fact that to write rules is quite easy to do.
- Most hacker use it so many times rules for new attacks can be found faster than in some other NDIS (this depends of how well one is connected).

8 Deliverables and milestones

The following deliverables have been identified, they are produced during the project and they set several milestones:

- Detailed project plan.
- Specification of the chosen architecture: Operating System, programming language, database, etc.
- Deliverables specification.
- Database design. The more complex part of the project is the design of the database because has to integrate different sources of information.
- System architecture and specification.
- Program source code and detailed configuration specification.
- Final report.

9 Design

9.1 Architecture Design

The approach has always been to try to avoid modification of the way systems work and to produce the least interference possible with the systems.

There is two possibilities:

- The first approach is to send all the information to the Syslog server and then store it in the database. This includes the snort information. The database design is oriented to Syslog format.

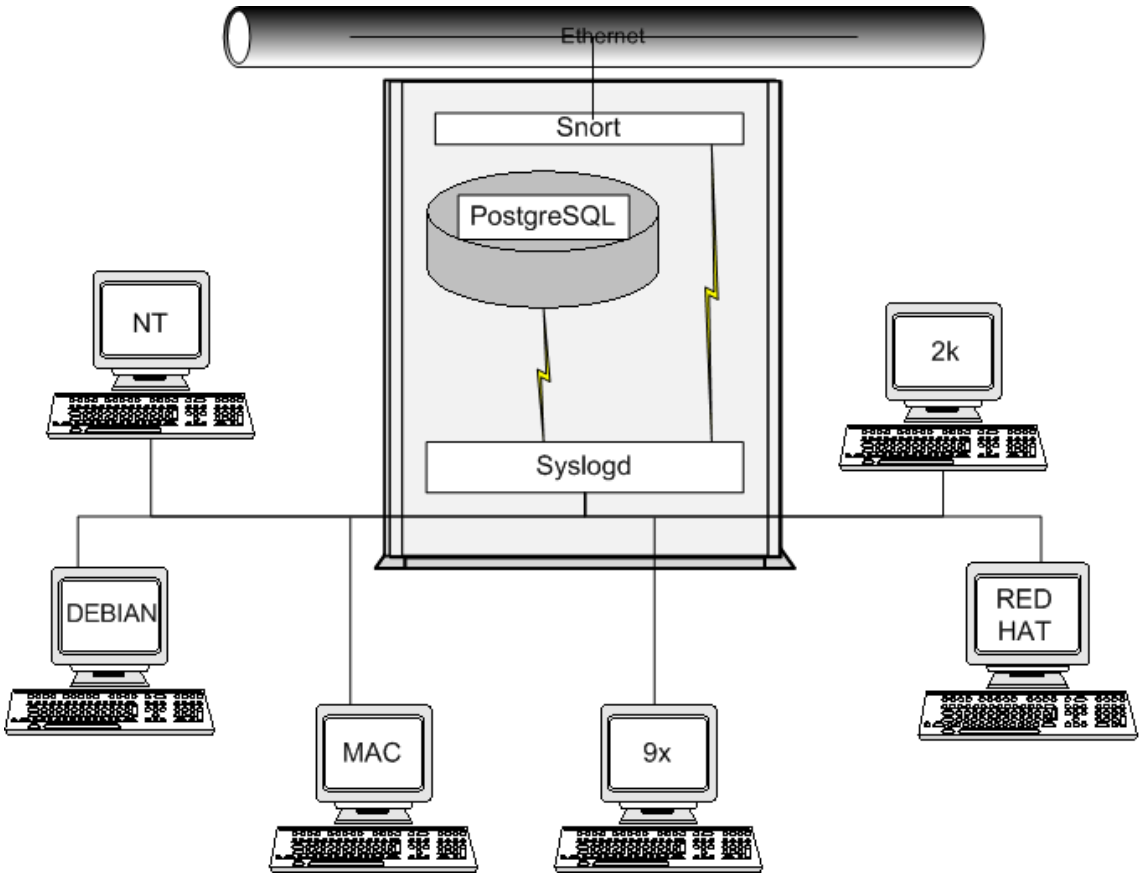

Figure 3: Architecture design one

- The second approach is to send all the information to the Syslog server except the snort information, so that each one stores the information in the same database separetely. The database design is oriented to snort format.

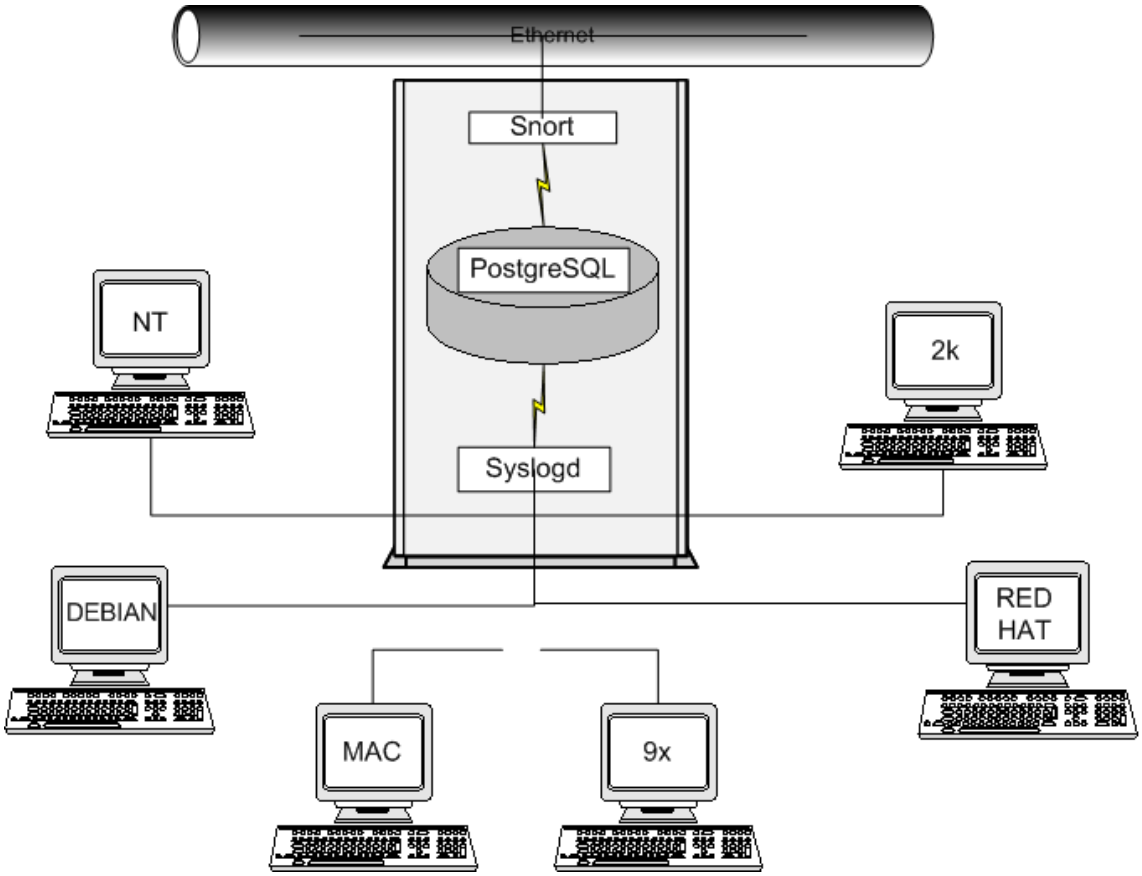

Figure 4: Architecture design two

This second one is the best because the information stored is much more detailed. There is already a module that stores the Snort information in a PostgreSQL database. Therefore, a program is needed to store the Syslog information in this database.

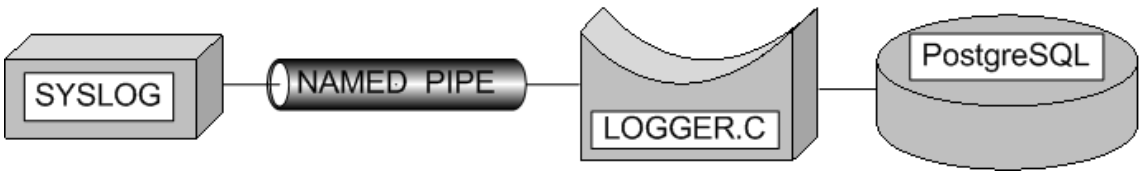

Figure 5: Logging from Syslog to PostgreSQL

The snort database is going to be used without modifications, so that already existent analysis software can be employed. Looking at the Syslog message format. There are four parts that can easily integrate with the snort database:

- Time. To put be in the table event in the field timestamp.

- Host. To put be in the table sensor. Each machine logging information would be a sensor.
- Application. be To put in the table reference, each program would be a reference source.
- Message. To be put in the table signature.

The only modification might be to add new indexes because the program must look at existing sensors, references and signatures. There is already a index for signature. One is necessary for the host name in the sensor and for the ref_tag in reference. See appendix A to see the modified script for the database creation. In addition a signature class entry must be inserted, one called application which is used to classify all the new references.

9.2 Development design

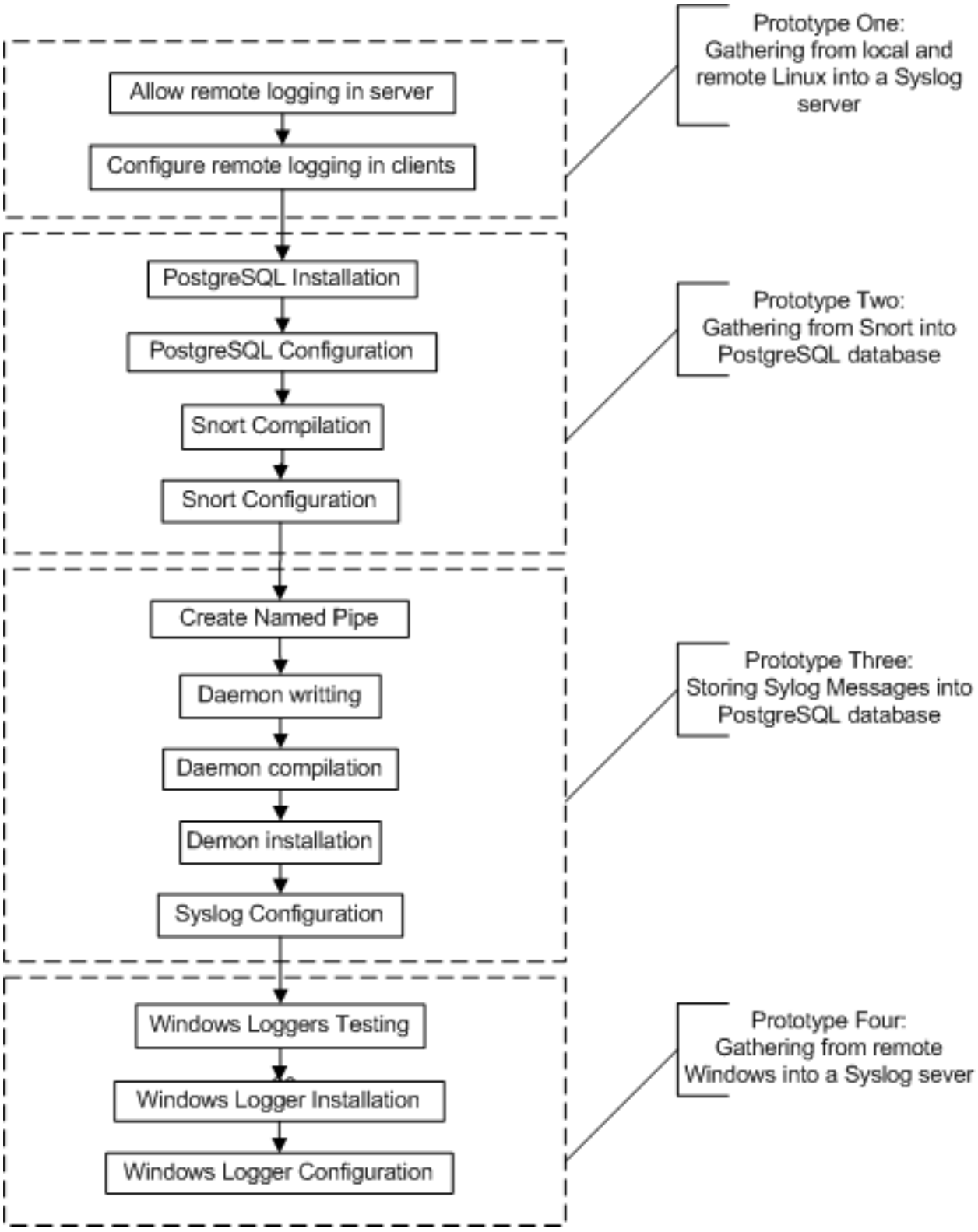

Figure 6: Development design

9.3 Database Design

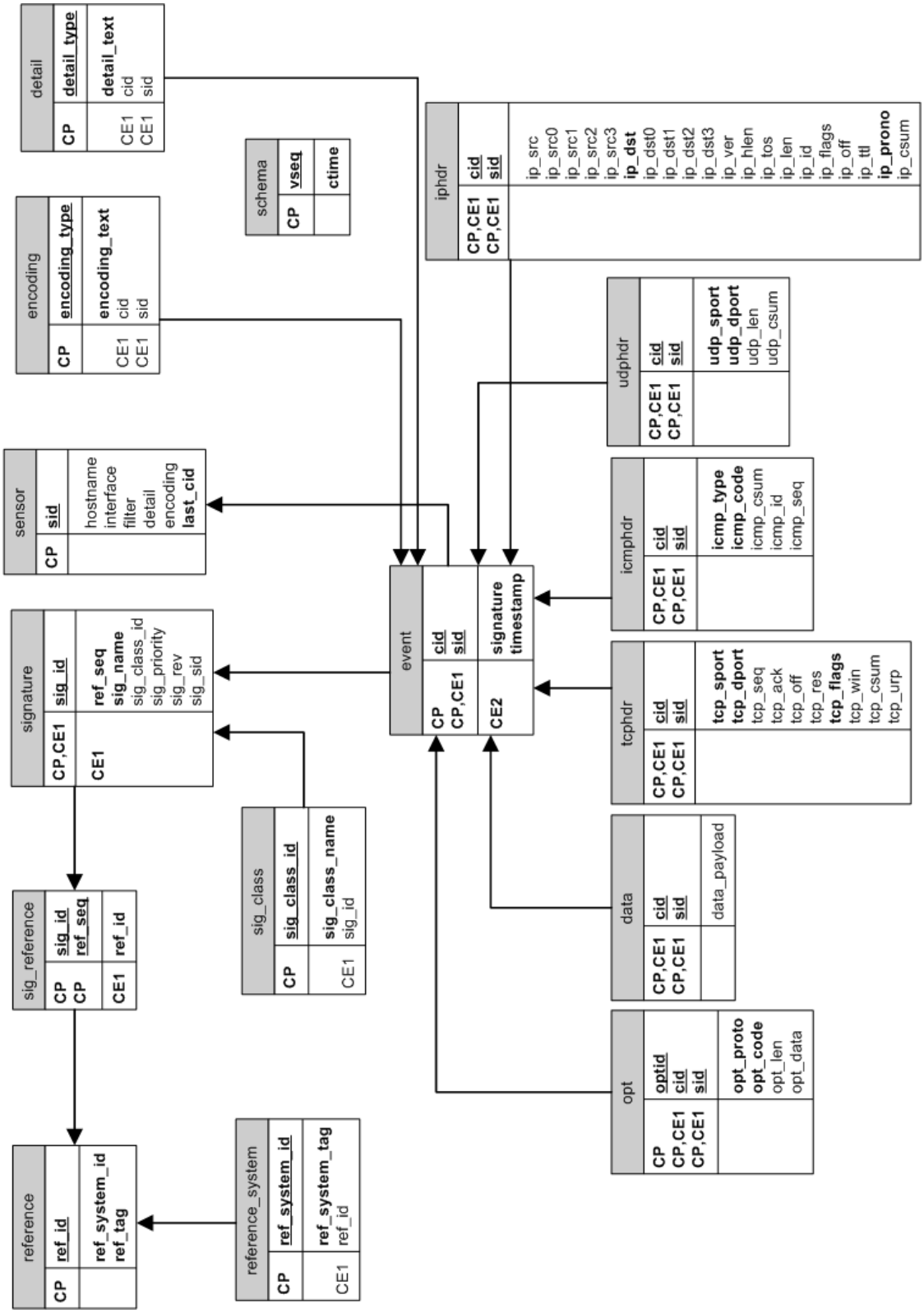

Figure 7: Database design

The program would work with the original script. The proposed script allows improved performance through indexes. In addition, it inserts some information that helps one to understand the information that has been stored in the database, so if someone searches in the database without knowing the architecture, they would understand.

9.4 Program Design

9.4.1 Message parsing

The following is the main program structure. There is some logic not included, such as how the message parsing and database probations and storing are going to be done. It is left to implementation time.

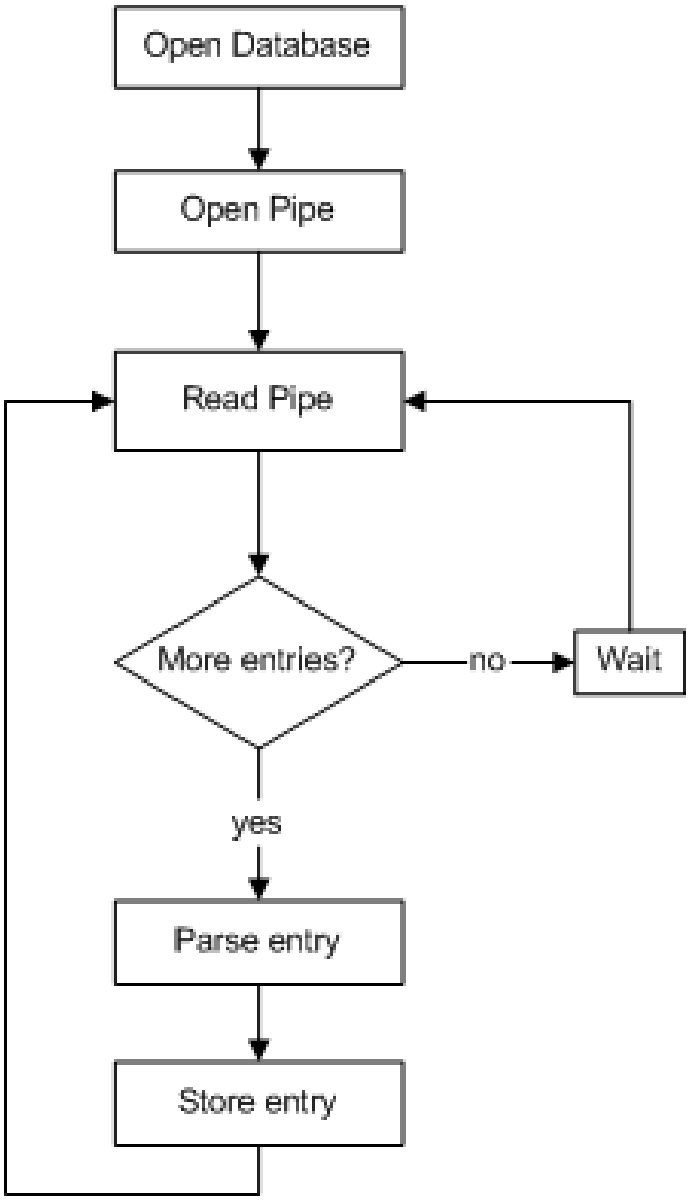

Figure 8: Program design

9.4.2 Database storing

The next problem to solve is the database storing of the Syslog entries. By following the study done in the database design the information is integrated into a Snort database without modifying it and using the same rules as snort.

In order to make sure that the programming was safe and that it was following the same procedures as Snort, the source code of snort database storing plugging was looked into.

The above would be the logical design and would work appropriately, but a modification was made. If the sensor does not exist, nothing is logged in the entry. This has one disadvantage:

- That the hosts to be monitored must be added to the table manually.

On the other hand, it has two advantages:

- There will be no incorrect information in the database.
- It allows a certain degree of security, so no one can attach a host and start sending incorrect information if the name of a host that is being monitored is not known.

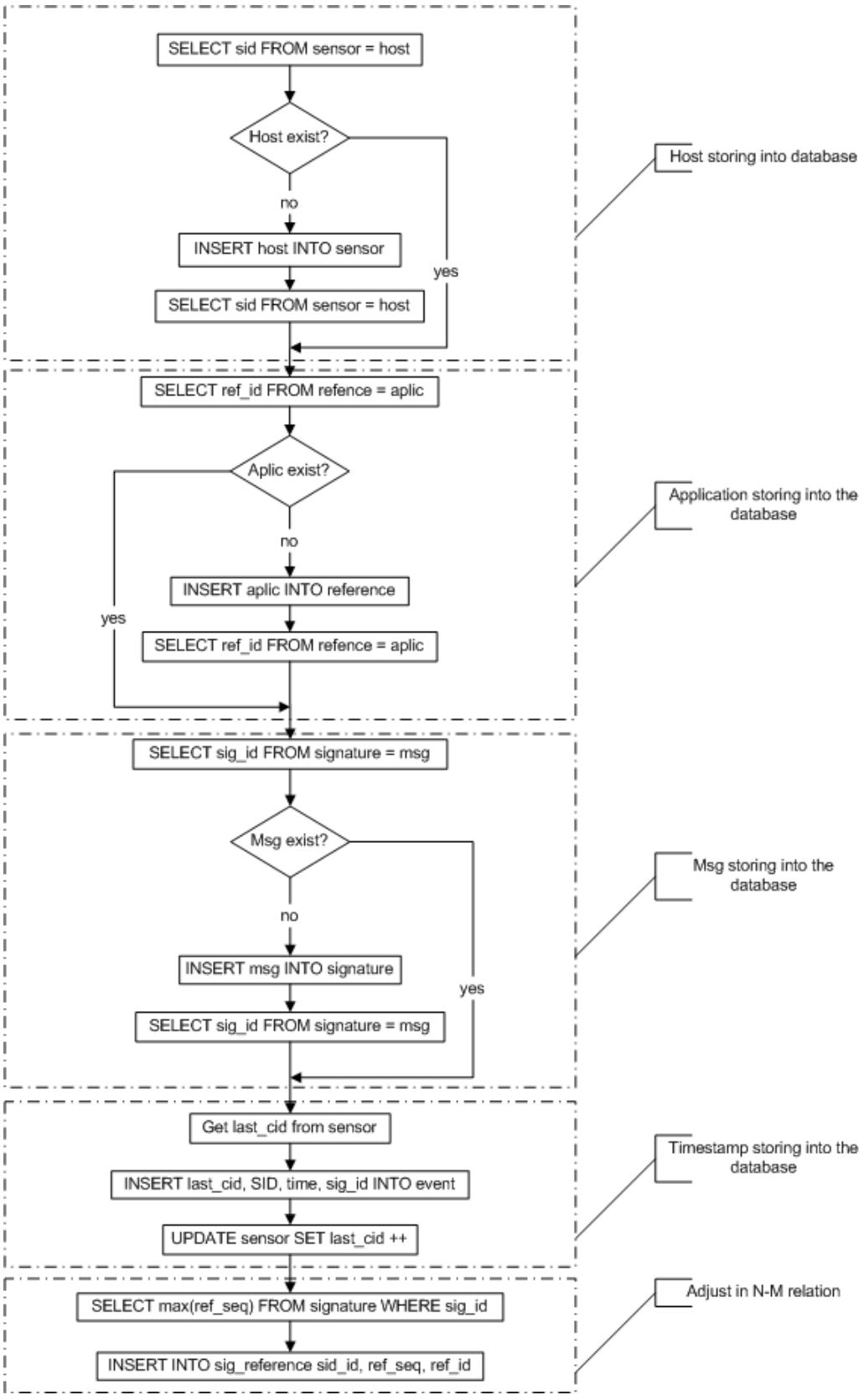

Figure 9: Database Manipulation

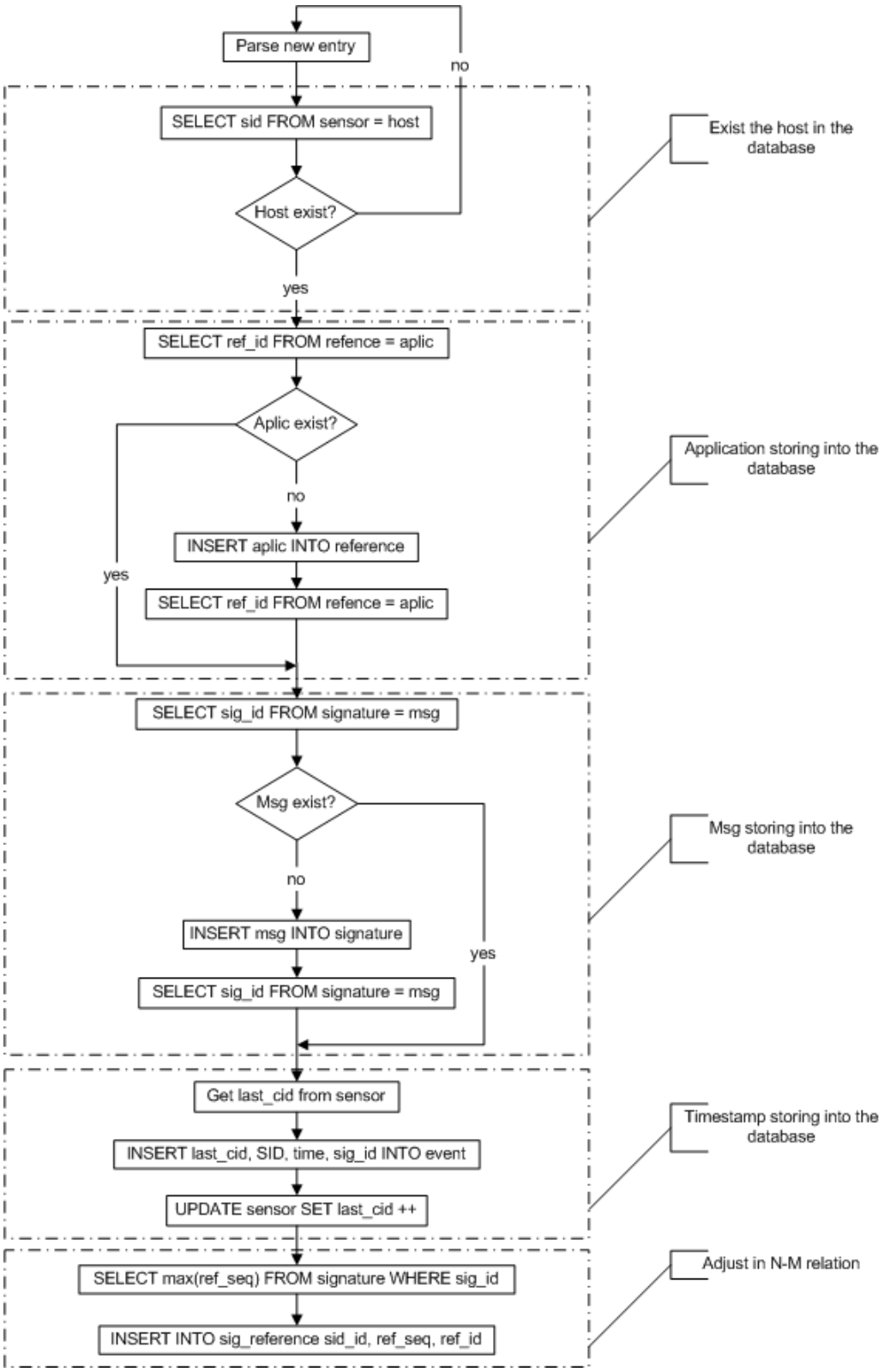

Figure 10: Database Manipulation Improved

10 Literature Review

The purpose of the project is simple, to centralize the logs of different systems in an easy to use repository. But it implies putting together information in different formats. Nobody who has documented all of the has been found. Firstly, all the parts involved will be presented, and secondly, the work already done in the field will be examined.

The objectives are:

- To understand current event logging capabilities in Linux, Windows and Snort.
- Store Syslog messages in a database.
- Send Windows events to Syslog.
- Send Snort logs to Syslog.

10.1 Events logging

10.1.1 Snort

[All01] Alerting is a part of the Alerting and Logging subsystem, wichc is activated at run-time through the use of command-line options.

When Snort inspects a network packet and detects a match between a rule and the network packet, Snort sends an alerting message to the user-defined facility and logs the packets causing the rule violation. The alerts may either be sent to Syslog, logged to an alert text file in two different formats, or sent as Win Popup messages using the Samba smb client program.

There are two options for sending the alerts to a plain text file; full and fast alerting. Full alerting writes the alert message and the packet header information through the transport layer protocol. The fast alert option writes a condensed subset of the header information to the alert file, allowing greater performance under load than full mode. There is a fifth option to completely disable alerting.

Similarly, logging can be set up to log packets in their decoded, human-readable format to an IP-based directory structure, or in tcpdump binary format to a single log file. The decoded format logging allows fast analysis of data collected by the system. The tcpdump format is much faster to record to the disk and should be used in instances where high performance is required. Logging can also be turned off completely.

More interesting is to look at the database Schema and Entity Relation diagram to see how alerts are stored. See appendix H. [Dan02]

The table sensor stores the host or device that generates the alert. In event is the date and time when the alert was generated and is relationated with a signature and reference that explains what is the alert and classifies it. This will be better explained in the database design.

10.1.2 Linux

[Lon01] The priority for each event log type controls the service and facility that the Syslog message is sent to. Each log type has a separate priority. If the priority for a particular key does not exist the default is 9, user.alert.

Usually, Syslog refers to a "facility" and "severity". These are combined in to a single value called "priority".

Standard facility and severity values are:

- (0) kernel
- (1) user
- (2) mail
- (3) system
- (4) security/auth 1
- (5) syslog
- (6) line printer
- (7) news
- (8) uucp
- (9) clock 1
- (10) security/auth 2
- (11) ftp
- (12) ntp
- (13) log audit
- (14) log alert
- (15) clock 2
- (16) local 0
- (17) local 1
- (18) local 2
- (19) local 3
- (20) local 4
- (21) local 5
- (22) local 6
- (23) local 7

The importance of an event determines the severity level of the log entry. Important events should be investigated before they affect availability. Messages with the following severity levels can be logged, listed in order of severity level:

- emerg: The cluster system is unusable.
- alert: Action must be taken immediately to address the problem.
- crit: A critical condition has occurred.
- err: An error has occurred.
- warning: A significant event that may require attention has occurred.
- notice: An event that does not affect system operation has occurred.
- info: An normal cluster operation has occurred.
- debug: Diagnostic output detailing normal cluster operations.

The Priority value is calculated by first multiplying the Facility number by 8 and then adding the numerical value of the Severity. In the PRI part of a Syslog message, these values would be placed between the angle brackets as (0) and (165) respectively.

Each entry in the log file contains the following information:

1 Timestamp The TIMESTAMP field is the local time and is in the format of "Mm dd hh:mm:ss" where:

Mmm is the English language abbreviation for the month of the year with the first character in uppercase and the other two characters in lowercase. The following are the only acceptable values:

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec dd is the day of the month. If the day of the month is less than 10, then it must be represented as a space and then the number. For example, the 7th day of August would be represented as "Aug 7", with two spaces between the "g" and the "7". hh:mm:ss is the local time. The hour 'hh' is represented in a 24-hour format. Valid entries are between 00 and 23, inclusive. The minute 'mm' and second 'ss' entries are between 00 and 59 inclusive.

A single space character follows the timestamp field.

2 Hostname: The hostname field will contain only the hostname. Without any embedded spaces.

A single space character follows the hostname field.

3 Message:

The msg part will fill the remainder of the Syslog packet. This will usually contain some additional information of the process that generated the message, and then the text of the message. There is no ending delimiter to this part.

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. This has traditionally been a freeform message that gives some detailed information

of the event. The tag is a string that must not exceed 32 characters. Any non-alphanumeric character will terminate the tag field and will be assumed to be the starting character of the content field. Most commonly, the first character of the contnt field that signifies the conclusion of the TAG field has been seen to be the left square bracket character "[", a colon character ":", or a space character.

10.1.3 Windows

[Sta00] [Bra00] [Mur98] The Event Logging mechanism was first implemented on Windows NT. It is also a component of Windows 2000 and XP. It provides a way for applications and Operating System to record events. The event logging is a Windows service.

In order for an application to create an entry in the event log, the application sends the appropriate information to the Event Logging Service using a function called ReportEvent. Then the Event Logging Service examines this information and creates an entry in the appropriate log file.

The different types of events can be viewed using the Windows administration utility called Event Viewer. Most Windows applications, including the operating system itself, log events to one of the three logs: Application, System and Security:

- Application Log: Records events reported by Applications. Application errors such as failing to load a DLL can also appear in the application log.
- Security Log: Records events that have been set for auditing with group policies. These events can include log-on and log-off, changes to access rights, system start-up and system shut-down.
- System Log: Records events concerning the operating system or one of its components, such as a failure of a device or a service to start or stop.

The event logs are stored in the directory:

```
\<<winroot>>\system32\config
```

Where "winroot" is the directory that hosts windows. The three log files are "AppEvent.evt", "SecEvent.evt" and "SysEvent.evt" and cannot be viewed using an ordinary text editor.

In addition, these files do not reflect the latest changes, which the log only writes at specific intervals. The Event Viewer enables users to see the contents of these three files, including any recent information.

For each event, the Event Viewer shows the following attributes:

- Type: There are five different types of events that can be logged:
  - Information: Indicate infrequent but significant successful operations.
  - Warning: Indicate problems that are not immediately significant, but which may indicate conditions that could cause future problems. If an application can recover from an event without loss of functionality or data, it can generally classify the event as a warning event.

  - Error: Indicate errors, such as the failure of a service to start. This type of event indicates that a loss of functionality or data has occurred.
  - Success Audit: This is an event related to the successful execution of an action. Such as a successful log-on.
  - Failure Audit: This is an event related to the failed execution of an action. Such as a failed log-on.

- Event Date and Time: These two attributes inform about the precise date and local timetable of when a specific event occurred.
- Event Source: Each log file in the Windows Registry contains a sub-key called event sources. This sub-key describes the application, service or hardware component that was logged.
- Event Category: This is a category of the event that can help administrators to organize and categorize events. Sometimes it can be used to describe the related action in greater detail.
- Event ID: Is a unique identifier for a specific event.
- Event Computer: This attribute contains the name of the computer where the event occurred.
- Event User: This is the user account that was logged on when the event occurred. When an event was caused by a server process the impersonation is taking place.
- Event Description: This is a detailed description of a specific event.

10.2 Syslog into a database

The default storage in Syslog is to store the events in files. This makes it very difficult to analyse the information.

The main approaches are to modify or hack the Syslog [R.01] or totally replace Syslog [Ale03] so as to store the information directly in the database.

Another approach is to redirect the events to a named pipe that communicates with another daemon. This method is very frequently discussed in forums and mailing lists [Voy99] [Aac03], but there is no published implementation of it. It is the method that is going to be used in this project because it does not modify Syslog and is the most efficient solution.

The advantages of not modifying the Syslog daemon are that it can be updated automatically or replaced for another without having to hack it again. In addition, it is easier to maintain because all the code that does one thing is together, and this suits the Linux philosophy on simple programs that perform a task well.

There is a similar project that stores the messages without modifying the daemon, but is not based on the standard Syslog. This project uses a replacement called Ng-Syslog that has many features not available in Syslog [Ear02]. One of these, is the use of templates in the syslog.conf, which allows the system to convert the message format into sql format. This feature is not available in Syslog.

Another similar approach is a wrapper that stores messages in mysql [Fra01]. The problem with it is that it is written as a common program instead of having a daemon and therefore is very inefficient.

Another weakness found in all of them is that they store information in Mysql. Since the potential use of this project can imply the use in networks where the number of messages can be huge, it is advisable to use a better database, such as PostgreSQL, which is considered the best open source database available.

All Syslog messages have an associated logging facility and level. The logging facility can be thought of as "where" that information must be sent and the level as "what" kind of information is sent.

The logging facilities used by Syslog are local0 through local7. Furthermore, there are different degrees of importance attached to each incoming message. These levels, starting from the one of the highest to the one of the lowest importance, are: emergency (0), alert (1), critical (2), error (3), warning (4), notification (5), informational (6), and debug (7). The lower the number, the greater the importance.

10.3 Daemon

10.3.1 Reasons for choosing a daemon

There are programs that perform a task without users needing to know what it is doing. The term 'daemon' is used for processes that perform a service in the background. A server is a process that usually begins execution at start-up, runs forever, neither dies nor gets restarted, operates in the background, waits for requests to arrive and responds to them and frequently spawns other processes to handle these requests.

In this case, the program that passes the messages from Syslog into a database does not need user interaction, it must be initiate automatically at system-starting after Syslog server.

10.3.2 How a daemon works

[SSS] [Kar01] Daemons are programs that run as a background process that does not belong to a terminal session. Usually, system services such as network services or printing.

Simply invoking a program in the background is not adequate for long-running programs; this does not correctly detach the process from the terminal session that started it.

Here are the steps to be taken when making a daemon:

1. Daemonizing: fork() so the parent can exit, this returns control to the command line or shell invoking the program. This step is required so that the new process is guaranteed not to be a process group leader. The next step, setsid(), fails if the daemon is a process group leader.

2. Process Independency: setsid() to become a process group and session group leader. Since a controlling terminal is associated with a session, and this new session has not yet acquired a controlling terminal, our process now has no controlling terminal.

3. fork() again so the parent can exit. This means that the daemon, as a non-session group leader, can never regain a controlling terminal.

4. Running Directory: chdir("/") to ensure that the daemon process does not keep any directory in use. Failure to do this could mean that an administrator could not unmount a filesystem, because it was the current directory.

5. File Creation Mask: umask(0) so that complete control is had over anything the daemon writes. It is not known which umask may have been inherited. Optional.

6. close() fds 0, 1, and 2. This releases the standard In, Out, and Error that the daemon inherited from the parent process. There is no way of knowing where these fds might have been redirected to. Note that many daemons use sysconf() to determine the limit _SC_OPEN_MAX. _SC_OPEN_MAX tells them the maximum number of open files. Then in a loop, the daemon can close all possible file descriptors.

7. Establish new open descriptors for Stdin, Stdout and Stderr. Even if it is not planned to use them, it is still a good idea to have them open. The precise handling of these is a matter of taste; if there is a logfile, for example, it might be wished to open it as Stdout or Stderr, and open '/dev/null' as Stdin; alternatively, '/dev/console' could be opened as Stderr or Stdout, and '/dev/null' as Stdin, or any other combination that makes sense for each particular daemon.

Almost none of this is necessary (or advisable) if the daemon is being started by inetd. In that case, stdin, stdout and stderr are all set up to refer to the network connection, and the fork()s and session manipulation should not be done (to avoid confusing inetd). Only the chdir() and umask() steps remain useful.

10.4 Why a named pipe

A pipe is designed to interconnect processes. The programmer does not have to worry about blocking and unblocking it or anything else. It only has to be read as if it were a file.

10.5 Why is better to store in a database

Although files and folders are a natural way to organize and access data, it is useful for a small number of items or those which are not very often accessed. They definitely fail when cross-referencing and processing of the information is required. And other problems arise such as duplication of data and inconsistency.

11 Project development

The project has four parts that correspond to the prototypes proposed:

- Gathering messages from remote Linux boxes into a central Syslog server
- Snort logging into PostgreSQL.
- Syslog central server storing into a snort PostgreSQL database.
- Gathering messages from remote windows boxes into a central Syslog server.

The different prototypes did not add many modifications because the goal is not to modify, but only to add new features. So each one added something, particularly configuration. The automatic installation of everything is out of the scope of the project.

In the project development, the main information sources have been the man pages, and the explanation in the configuration files in different software packages. The information given by free software packages is amazing, as nothing else i necessary. Everything is just waiting to be consulted.

11.1 Gathering from Linux boxes

Configuration must be done in the server to allow remote logging and in each client so that the messages are sent by the network.

11.1.1 How to allow remote logging

The Syslog server by default does not allow remote logging, so it must be enabled: Edit the file

```
/etc/rc2.d/S10sysklogd
```

, setting:

```
SYSLOGD="-r"
```

Put the following entry in the file

```
/etc/services
```

like:

```
syslog 514/udp
```

11.1.2 Logging remotely

Configure systems to send logs to a central Linux machine via Syslog. The following configuration must be done in all hosts that are wanted to be logged into the central Syslog server.

Put the following entry in the file

```
/etc/services
```

like:

```
syslog 514/udp
```

And in the configuration file file

```
/etc/syslog.conf
```

like:

```
*.* @132.132.132.129
```

11.2 Gathering from Snort

[BF03] Snort will store the information in its PostgreSQL database, which is going to be the same where the Syslog server stores the logs so that all logs are together.

11.2.1 PostgreSQL configuriguration

Installation in Debian is considerably easy, many systems already have it installed:

```
apt-get install PostgreSQL
```

[SNO] [PDE] [WD01] The first step is to configure the database. There is an explanation in the file README.database:

1. Install PostgreSQL (apt-get).
2. Create a database for snort. In Debian, when PostgreSQL is installed for first time, there is only one user 'postgres' that can be only accessed form the Debian user 'postgres'. This user is the root user inside PostgreSQL. Therefore, a 'postgres' user has to be created in the Operating System (OS) prompt:

   ```
   adduser postgres
   passwd ***********
   ```

   And create the database from the OS prompt:

   ```
   createdb snort
   ```

3. Create a user that has privileges to INSERT and SELECT on the database: From the Operating System prompt with "postgres" user:

   ```
   createuser josean
   ```

4. Build the structure of the database according to files supplied with snort in the "contrib" directory. This should be done by means of the user previously created and also from the OS prompt.

   ```
   psql snort < ./contrib/create_postgresql
   ```

5. Snort installation. In Debian "apt-get install" can be used but this will not install the plugging for database logging. So, snort sources must be downloaded and compiled. If snort has already been installed:

   ```
   apt-get remove snort
   ```

   Depending on the system it was be necessary to install the libpq library. And for installation [Har]:

   ```
   1. Untar: tar -xvzf snort-2.0.1.tar.gz
   2. Up: cd snort-2.0.1
   3. ./configure --with-postgresql
   4. make
   5. make install
   6. cd * /etc/snort
   7. cd ../etc
   8. cp snort.conf /etc/snort
   9. cp *.config /etc/snort
   10. cp ./rules /etc/rules
   ```

   Point 3 can be different depending on the system configuration. In case of a failed compilation before compiling again:

   ```
   1. Uninstall: make uninstall
   2. Remove files: rm -r snort-2.0.1
   ```

6. Configure the plugging in the file /etc/snort/snort.conf

11.3 Store in database

Central Linux machine stores the logs in a database. The central Linux machine is using the Debian distribution because it is becoming the most commonly chosen one for security and because it is the author's favourite. The PostgreSQL database was chosen because it is the best open source database available and because it has some object oriented features.

11.3.1 Named pipe creation

The pipe creation must be carried out before starting Syslogd. So, there is a script in

```
/etc/init.d
```

called logtdb that creates the pipe:

```
mkfifo /tmp/myLog.pipe
```

To check the pipe has been created and works properly:

- In one console type:

```
cat /tmp/myLog.pipe
```

- In another being root:

```
echo test /tmp/myLog.pipe
```

11.3.2 Daemon

Basic Daemon compilation:

```
cc -o LogToDb LogToDb.c
```

With database access Daemon compilation: cc -I/usr/local/pgsql/include -o LogToDb

```
LogToDb.c -L/usr/local/pgsql/lib -lpq
```

Add to the file

```
 /etc/init.d/logtdb
```

:

```
/home/josean/project/LogToDb
```

Give execution permission:

```
chmod +x logtdb
```

Then do a link to that script to

```
 /etc/rc2.d/S05logtodb
```

for example:

```
ln -s /etc/init.d/logtdb /etc/rc2.d/S05logtodb
```

The S05 is due to the fact that, in that directory, all are links to scripts in /etc/init.d. It is the run level that Debian starts by default. The Syslog starts with priority "10", as its name shows (S10syslogd), so in order to be able to execute a script prior to that one, the script must have a lower number.

If there is something wrong and the process crashes, and it does not allow logging, reboot and type:

```
boot:Linux init 3
```

Where Linux is the name that was given to image in multi-boot and init 3, indicates to enter in run level 3, instead of run level 2, that is the default one and the one that can have been damaged.

To see if the daemon is running:

```
ps -efgrep LogToDb
```

Database insertion: The first thing that must be done is to check if the sensor exists in the "sensor" table. If it does not exist, then it must be inserted in the "hostname" field. The rest of fields can be left empty except "last_cid", which has to be 0.

Secondly, the application's presence in the "reference" table must be checked. If it does not exist then it must be inserted in the field "ref_tag". The "ref_system_id" must be populated with the id 1 that was inserted into the database at creation time.

The third step is to insert the message in the "signature" table and "sig_name" field if it does not exist. The "sig_rev" field is the revision number that does not make sense in this program, so it is ignored. The same happens with the internal signature id in the "sig_sid" field.

There is a sig_priotity field that is ignored but which could be used to store the facility. This will imply the use of multiple pipes, so Syslog will send the message to one or another depending on the facility. But this will work against performance.

Finally in the "event" table the occurrence of the message is stored and the time is stored in the "timestamp" field. But first of all the last cid must be checked in the "sensor" table and then it has to be updated.

One more insertion is necessary, not to store information. It is just to relate the message with the application because it is a n-m relation. Therefore, an entry must be stored in the "sig_reference" table.

11.3.3 Modify Syslog configuration file

Modify Syslog configuration file (syslog.conf) to send messages to a named pipe.

```
Add the line: *.* /tmp/myLog.pipe
```

To see if it works properly type in one console:

```
cat /tmp/myLog.pipe
```

11.3.4 Storing daemon

Write a daemon that reads the named pipe and stores the information in the database. The Levent Karakas guide to implement a daemon has been used [Kar01]. Add code to read data from a named pipe and store it in a database.

Below an explanation is given of what the program does. Some details and logic have been left out to simplify it. Only the main code has been included: to see all the program, look at Appendix B.

The first thing that the program does is to transforms itself into a daemon. All the necessary steps are in a function called daemonize (fork, setsid, chdir, umask, etc):

```
if(getppid()==1) return; /* already a daemon */
i=fork();
if (i<0) exit(1); /* fork error */
if (i>0) exit(0); /* parent exits */
/* child (daemon) continues */
setsid(); /* obtain a new process group */
```

```
for (i=getdtablesize();i>=0;--i) close(i); /* close all descriptors */
i=open("/dev/null",O_RDWR); dup(i); dup(i); /* handle standart I/O */
umask(027); /* set newly created file permissions */
chdir(RUNNING_DIR); /* change running directory */
lfp=open(LOCK_FILE,O_RDWR|O_CREAT,0640);
if (lfp<0) exit(1); /* can not open */
if (lockf(lfp,F_TLOCK,0)<0) exit(0); /* can not lock */
/* first instance continues */
sprintf(str,"%d\n",getpid());
write(lfp,str,strlen(str)); /* record pid to lockfile */
signal(SIGCHLD,SIG_IGN); /* ignore child */
signal(SIGTSTP,SIG_IGN); /* ignore tty signals */
signal(SIGTTOU,SIG_IGN);
signal(SIGTTIN,SIG_IGN);
signal(SIGHUP,signal_handler); /* catch hangup signal */
signal(SIGTERM,signal_handler); /* catch kill signal */
```

Secondly it opens the pipe. It is opened to write mode to avoid to receivinf an EOF if the writing process has not started yet:

```
fdPipe = open(cPipe, O_RDWR)
```

The third step is to open the database:

```
conn = PQconnectdb("dbname=snort3");
```

After that it enters into a infinite loop:

```
while (1) {
```

Then all the parsing work starts.
The date:

```
BDEntry.time = cpBufferOffset;
*(BDEntry.time + posFin)= '\0';
```

The host:

```
BDEntry.host = cpBegin;
*(BDEntry.host + posFin)= '\0';
```

The application:

```
BDEntry.aplic = cpBegin;
*(BDEntry.aplic + posFin)= '\0';
```

The message:

```
BDEntry.mess = cpBegin;
*(BDEntry.mess + posFin)= '\0';
```

Of special interest is a buffer of constant size to read the pipe. The important thing is that the buffer can contain more than one message and that the last one is usually not finished. Several pointers are used to manage that buffer.

- cBuffer: An array that is the real buffer.
- cpBufferOffset: To point to the beginning of a Syslog entry.
- cpBegin: To point to the beginning of a part of a Syslog entry (date, host, application or message).

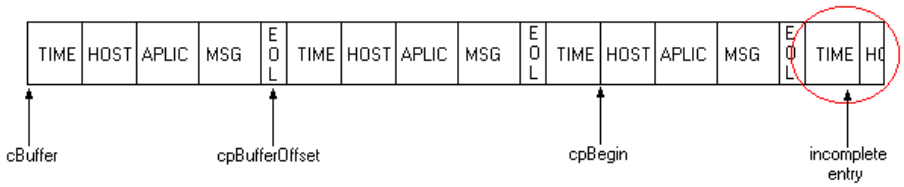

Figure 11: Buffer Manipulation

When the last message is reached, because it is not finished, it must be copied to the beginning of the buffer and the rest of the buffer must be completed. For that reason, the number of bytes read form the pipe is variable. To control how many bytes are in the buffer, there is the "iBytesLeftBuffer" variable.

The only thing left is the storing into the database.

The host must already exist:

```
sprintf(query,"SELECT sid FROM sensor
               WHERE hostname = '%s'",BDEntry.host);
cpResult = PQexec(conn,query);
resultH = atoi(PQgetvalue(cpResult,0,0));
```

The application. If it is not already in the database, it is added:

```
sprintf(query,"SELECT ref_id FROM reference
               WHERE ref_tag = '%s'",BDEntry.aplic);
pResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  if (PQntuples(cpResult) < 1) {
    PQclear(cpResult);
    sprintf(query, "INSERT INTO reference (ref_system_id, ref_tag) VALUES
             (1,'%s')",BDEntry.aplic);
    PQexec(conn,query);
    sprintf(query,"SELECT ref_id FROM reference
                   WHERE ref_tag = '%s'",BDEntry.aplic);
    cpResult = PQexec(conn,query);
  }
  resultA = atoi(PQgetvalue(cpResult,0,0));
}
PQclear(cpResult);
```

The message. If it is not already in the database, it is added:

```
sprintf(query,"SELECT sig_id FROM signature
                 WHERE sig_name = '%s'",BDEntry.mess);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  if (PQntuples(cpResult) < 1) {
    PQclear(cpResult);
    sprintf(query, "INSERT INTO signature (sig_name, sig_class_id) VALUES
            ('%s',1)",BDEntry.mess);
    PQexec(conn,query);
    sprintf(query,"SELECT sig_id FROM signature
                WHERE sig_name = '%s'",BDEntry.mess);
    cpResult = PQexec(conn,query);
  }
  resultA = atoi(PQgetvalue(cpResult,0,0));
}
PQclear(cpResult);
```

The date. It is stored into the main table, the event. Many adjustments must be made first. To maintain the last_cid and relation with other tables:

```
//Time to event
sprintf(query,"SELECT last_cid FROM sensor WHERE sid = %d",resultH);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  resultC = atoi(PQgetvalue(cpResult,0,0));
}
PQclear(cpResult);

sprintf(query, "INSERT INTO event (sid, cid, signature, timestamp) VALUES
        (%d, %d, %d, '%s')",resultH, resultC +1, resultA ,BDEntry.time);
PQexec(conn,query);

//Update last sid
sprintf(query, "UPDATE sensor SET last_cid =%d
           WHERE sid =%d",resultC + 1, resultH);
PQexec(conn,query);

//Relation between messages and aplications, signatures and references
sprintf(query,"SELECT max(ref_seq) FROM sig_reference
               WHERE sig_id = %d",resultA);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  resultS = atoi(PQgetvalue(cpResult,0,0));
} else {
  resultS = 0;
}
PQclear(cpResult);

sprintf(query, "INSERT INTO sig_reference (sig_id, ref_seq, ref_id) VALUES
```

```
       (%d, %d, %d)",resultA, resultS+1, 1);
PQexec(conn,query);

//Store event
sprintf(query,"SELECT max(cid) FROM event WHERE sid = %d",resultH);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  resultS = atoi(PQgetvalue(cpResult,0,0));
} else {
  resultS = 0;
}
PQclear(cpResult);

sprintf(query, "INSERT INTO event (sid, cid, signature, timestamp) VALUES
       (%d, %d, %d, TO_TIMESTAMP('%s','YYYY Mon DD HH24:MI:SS'))"
        ,resultH, resultF+1, resultA, BDEntry.time);
PQexec(conn,query);
```

Transactions are used so snort will not interfere. Start of a transaction:

```
//Start transaction
cpResult = PQexec(conn,"BEGIN");
PQclear(cpResult);
```

To finish the storing the transaction must be carried out so that the changes in the database are saved:

```
//End of transaction
cpResult = PQexec(conn,"COMMIT");
PQclear(cpResult);
```

11.4 Gathering from Windows boxes

Windows machines must be able to send the logs to a Linux machine.

Possibilities analysed:

- Scrambler [CPL]: It was discarded early on because it is a commercial tool and because of its complexity. It is a very complete tool with many features such as event monitoring and alerting. But only logging capabilities are needed. Nowadays it is very common for a workstation to be heavily overweighted with applications that are not directly productive, such as anti-virus or backup programs, plus some constant running applications like email and fax clients. All this apart from the normal business applications. So it is important to use the simplest and most efficient program.

- BackLog [BAC]: It has the advantage of being designed to work without having to modify the normal operation of the client machine. It works like a service. It has the additional advantage of using notification to get the messages. Instead of scaning the Event logs on time intervals, it uses the operating system to be

notified of new messages. Therefore, it provides real time notification and less overhead in the system. Under GNU Public License.

All three NT event logs (Application, System and Security) are monitored, and event information is converted to tab delimited text format, then delivered over UDP to a remote server.

BackLog is currently configured to deliver audit information to a SYSLOG server running on a remote or local machine. A configuration utility allows you to set the appropriate Syslog target and priority, as well as the target DNS or IP address of the server that should receive the audit information.

The service can be configured easily with a program called Audit Service Configuration, the only problem being that all messages can be sent with a single Category, which is always the same.

Figure 12: Backlog Audit Service Configuration

Memory usage: 3288 kb.

Program size: 32 kb.

Total software size: 141 kb.

- NTsyslog: It is Free Software that can be used in Windows NT/2000/XP [NTS02]. It formats all System, Security, and Application events into a single line and sends them to a Syslog host.

The service will be started automatically by the service control manager during system start-up. You can start and stop the service manually from the Services Control Panel, or from the command line, with the following command:

Start: net start ntsyslog Stop: net stop ntsyslog

The service can be configured to run as a local user with the following rights:

- Log on as a service.
- Manage auditing and security log.

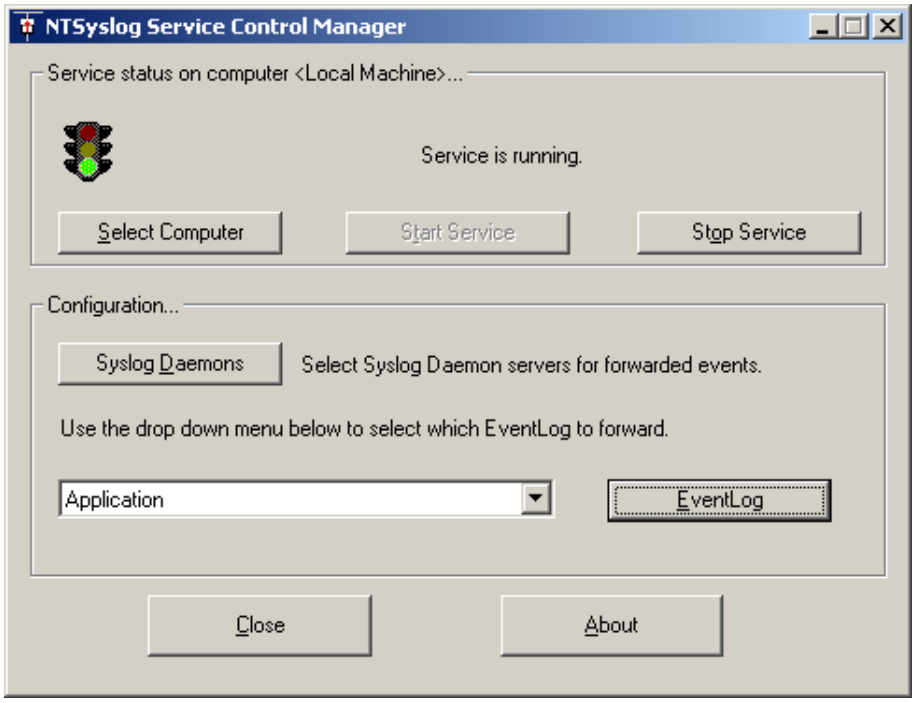

Figure 13: NTSyslog Service Control Manager

NTSyslog can log information to two Syslog servers at the same time:

It allows a much finer configuration on how to log. It is possible to direct any of the three windows categories to any facility and severity.

To remove the service run: ntsyslog -remove

Configure it by editing the registry keys properly, or by using the NtSyslogCtrl app.

Memory usage: 1128 kb.

Program size: 64.5 kb.

Total sw size: 450 kb.

- EventReporter: Can be used in Windows NT/2000/XP [EVE].

  The EventReporter Service runs as an NT Service:

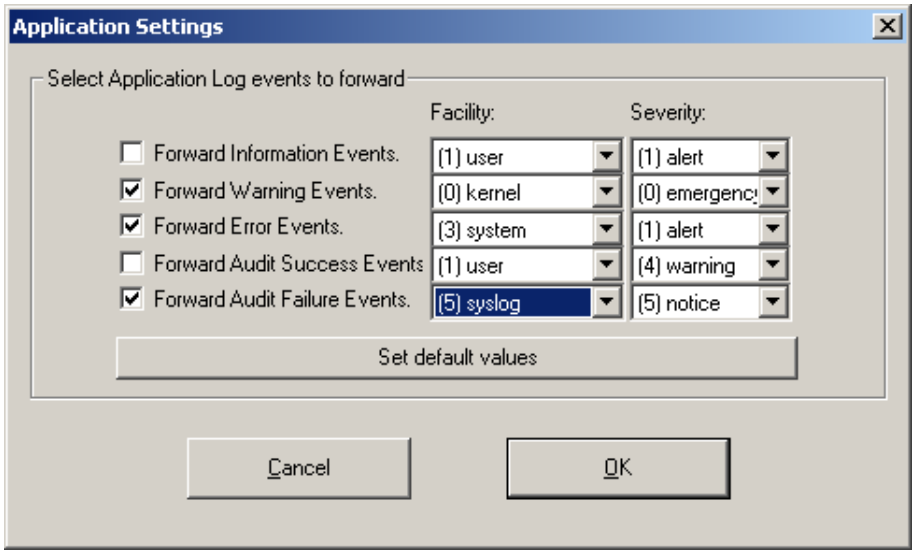

Figure 14: NTSyslog allows any facility and severity

- Centralized Logging: EventReporter allows consolidation of multiple NT event logs and forwards them automatically to either a Syslog server or to be emailed.
- Syslog Support: NT Event Messages can be forwarded using Standard Syslog Protocol. NT severity classes are mapped to the corresponding Syslog classes. Syslog facility codes are fully supported.
- Remote Administration: The client can be used to remotely manage EventReporter instances.
- Full NT Event Log Decoding: EventReporter can fully decode all types of NT event log entries. It has the same capabilities as the Event Viewer.

The service can be configured easily with a program called EventReport Client:

It has many extra possibilities. One of interest for the project is the fact that it allows facility selection, but not severity, which is managed automatically.

Memory usage: 3408 kb.

Program size: 180 kb.

Total sw size: 996 kb.

11.5 NTSyslog choice

NTSyslog was chosen because it supports all the standard facilities and severities, it is the one that needs less resources and it does not have any features that are more, or less, than necessary. It is as if it was designed for this project.

11.5.1 Installation

Install the service by executing the following command:
NTsyslog -install

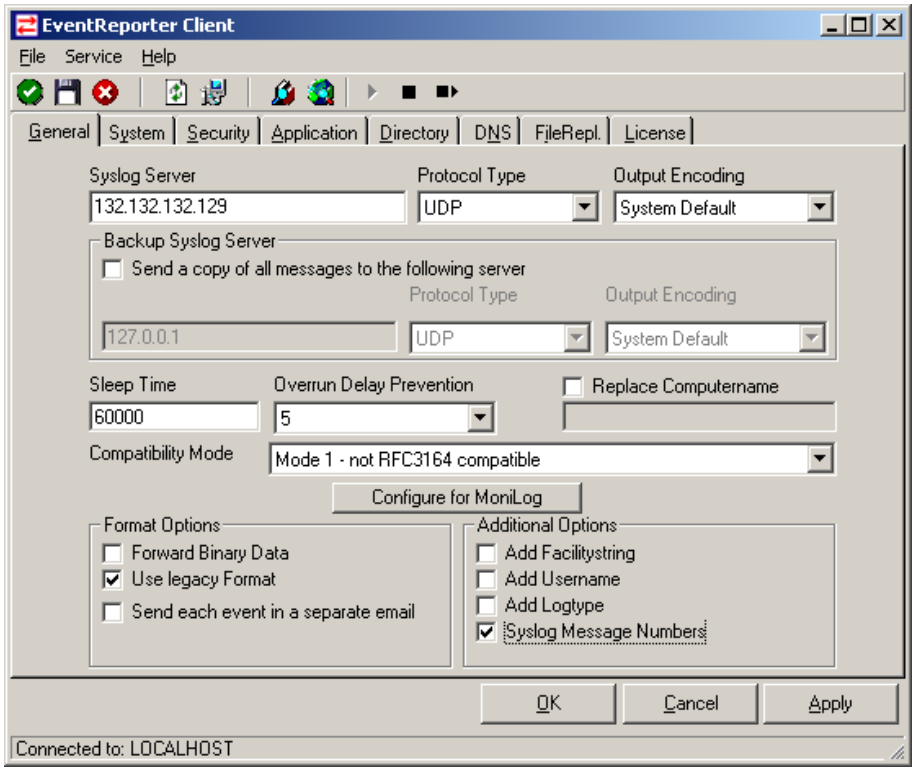

Figure 15: EventReport Client

11.5.2 Configuration

Input the host destination for the messages, it must have Syslog and remote logging enabled:

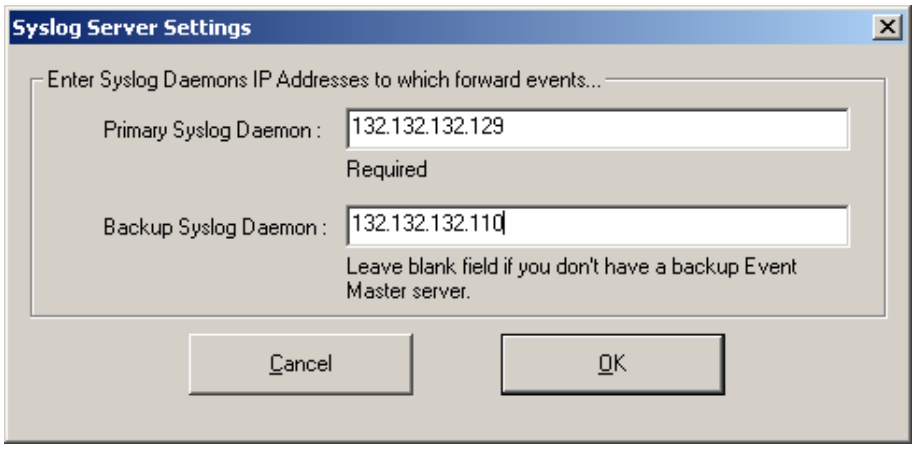

Figure 16: NTSyslog can log to multiple Syslog servers

Select what kind of events are to be send to the central Syslog server and what facility and severity is requiered:

First, auditing in the Operating System must be enabled. It may be interesting to enable more local logging and the central does not have to be the same. Windows

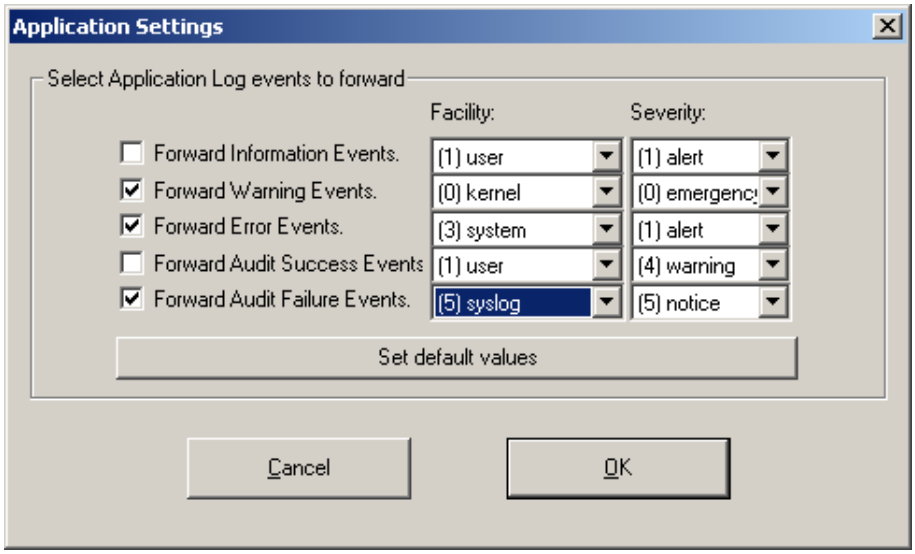

Figure 17: NTSyslog allows any facility and severity

allows the logging any and everything, much more than can be processed. Therefore what is to be logged must be chosen carefully.

12 Possible improvements

- Use configuration file for pipe name, database name, etc.
- Create the pipe from the program and manage it.
- Create installation script.
- The program and all the documentation has been prepared for a new installation and configuration. It is not prepared to afford an existing one.

  For example: there is a script to configure a new database but not to modify an existing one. The program is not designed to afford a modified database, it uses fixed ids. Note that by not using fixed ids, the program would be more complex, which would affect reliability and performance.

  The program has the clear goal of storing the message in a database. It would be worth putting all that logic into it. So, it would be better to write a script or program that will modify the database in a way that the same fixed ids can be used and the database will not lose consistency.
- Something important in a centralized system that has not been studied is the time synchronization between all the systems.
- Rewriting the program to support the standard in Syslog. It has been developed to support Debian machines and Windows messages sent by NTSyslog; other message formats that are correct might not be supported. If the message is logged using another variation of the standard, the program does not crash. It just does not log the message. Neither is there a registration of the discarded message.
- There is no registration of discarded messages.
- Implement a start, stop and start procedure for the daemon.
- To perform tests with analysing tools.
- To rewrite the program to allow easy expansion to use other databases.

13 Testing

In this project where the architecture building is more important and takes more time than the coding, the most important tests are performance, interoperability and error detection.

The first set of testing has been done during the end of each part's development. Testing was carried out to check functionality, correct errors and check performance.

The second set has been to check the addition of each prototype to the full architecture that has been developed until then, to find incompatibilities and to check performance.

The last set took place at the end of the project to check mainly the performance and the limits of the architecture.

13.1 Machines used

Several machines were used for testing, the role of each was selected by availability. For example, the server is the author's laptop, which is more or less always with him. The main test machine is his desktop computer. The other machines are geographically dispersed productive machines in a real organization.

- Satu: The server machine is a laptop of a unknown branch called Creature. The characteristics are:
  - Processor: Pentium III 1900 MHz.
  - HD: 30 GB IDE. 10 GB for the Operating System.
  - RAM Memory: 256 MB.
  - Operating System: Linux Debian Kernel 2.2.20.
- Dwa: Main Linux client for the second and third set of tests. Unknown branch.
  - Processor: Pentium II 266 MHz.
  - HD: 2 GB.
  - RAM Memory: 64 MB.
  - Operating System: Linux Debian Kernel 2.2.20.
- Tiga: Main Windows client for the second and third set of tests. Unknown branch.
  - Processor: AMD Athlon 900 MHz
  - HD: 40 GB IDE. 10 GB for each Operating System.
  - RAM Memory: 256 MB.
  - Operating System: Windows 98 Second Edition 4.10.2222 A and Windows 2000 Server 5.00.2195.
- Empat: Secondary client for the second and third set of tests is a DELL machine.
  - Processor: Pentium III 2000 MHz.

  - HD: 15 GB.
  - RAM Memory: 256 MB.
  - Operating System: Windows 2000 Server 5.00.2195.

- Lima: Another client for the third set of tests is an Intel machine.
  - Processor: Pentium III 800 MHz.
  - HD: 40 GB.
  - RAM Memory: 256 MB.
  - Operating System: Windows XP Professional 2002.

- Enam: Another client for the third set of tests. Unknown branch.
  - Processor: Pentium III 866 MHz.
  - HD: 40 GB.
  - RAM Memory: 256 MB
  - Operating System: Windows XP Professional 2002.

- Tujuh: Another client for the third set of tests. Unknown branch.
  - Processor: Pentium III 800 MHz.
  - HD: 20 GB.
  - RAM Memory: 196 MB.
  - Operating System: Windows 2000 Server 5.00.2195

- Delapan: Another client for the third set of tests. Unknown branch.
  - Processor: AMD Athlon XP 1500 MHz.
  - HD: 40 GB.
  - RAM Memory: 256 MB.
  - Operating System: Windows XP Professional 2002.

- Sembilan: Another client for the third set of tests. Unknown branch.
  - Processor: Pentium III 1800 MHz.
  - HD: 8 GB SCSI.
  - RAM Memory: 512 MB.
  - Operating System: Windows 2000 Server 5.00.2195.

A name has been given for further reference.

13.2 Daemon Testing

The daemon was the first part of coding. For testing, it was put in Satu for three month to run automatically at Operating System start up. It was always there during other testing. It never gave any problems. Neither was any change in the behaviour of the computer detected.

13.3 Parsing Testing

To test if the daemon is able to process all the messages sent by Syslog.

A first test was performed with a logger program that does not store into database. There was a Windows 2000 machine logging everything to the Syslog server. The logger was left for six hours, because it was reporting all the information to the console and the amount of information was huge; it was not possible to process all the information in real time. The pipe was filling up faster than being read. But neither the program, pipe nor system crashed. It is not normal to log everything, it was just to try the program. So the test was considered successful.

A second test was done adding two more machines (one Linux and another Windows NT) with the same result.

13.4 Database Store Testing

13.4.1 Snort Standalone Testing

The first test has been to check the snort alert storing in the database. To generate attacks that would produce alerts in Snort the vulnerabilities scanner Web Hack Control Center [Bar03] was used from a Windows 98 machine.

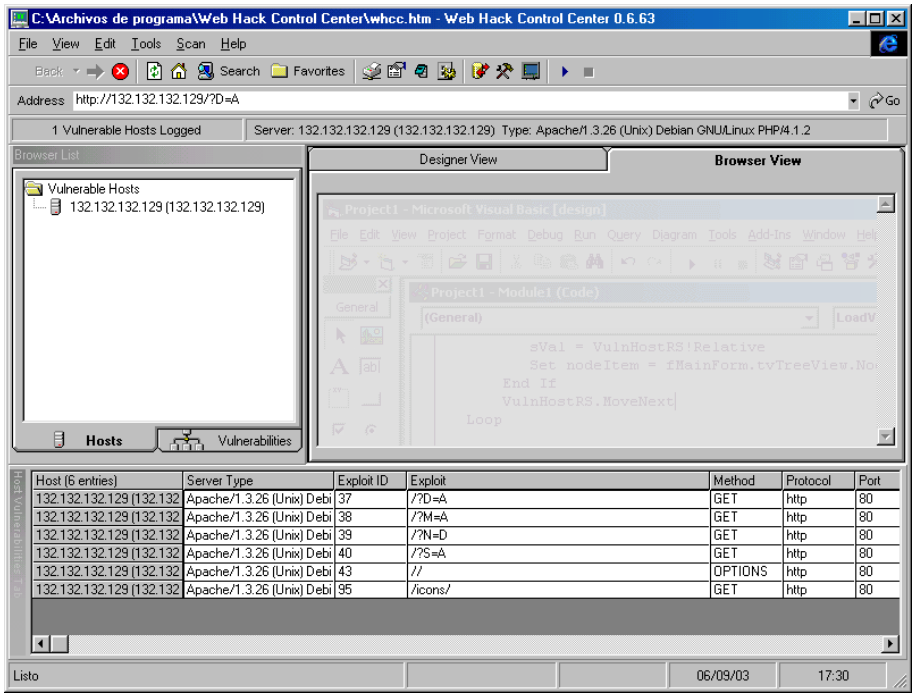

Figure 18: Web Hack Control Center

Snort detects the attacks and stores them in the database:

The test was successful, the snort was able to generate and store the alerts.

13.4.2 Syslog Standalone Testing

The first test has been to only store messages from the local machine.

The second has been done, adding a remote linux machine.

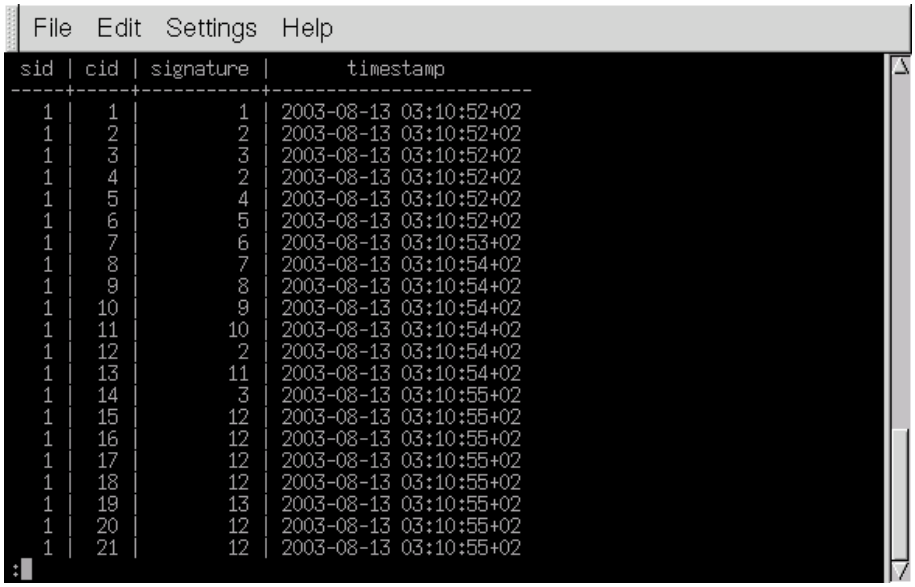

Figure 19: Alerts generated and stored by snort

The third has been done, adding a remote windows machine.
The fourth has been done, adding a linux and a windows machine.

13.4.3 Syslog and Snort Testing

The first test carried out on the storage of messages from the local machine and snort. This test revealed some problems because sometimes snort was written the last_sid in the 'sensor' table of the database, between the reading and update of that value by the logger. The solution is to use a transaction so the reading and the writing is taken as a unique and indivisible operation. That means that other operations are blocked until they are performed and that both or none are done. It was soon discovered because the author suspected that it may happen.

The same could happen in the other way, the daemon interfering with Snort, but it was not revealed in the test. To make sure the source code of the output plugging was consulted. The transactions are implemented and turned on by default.

In the second test, the storage of messages from snort and remote windows and linux was looked at.

13.4.4 Syslog and Snort final Testing

In the final test all the machines were logging into Satu:

- One local linux to syslog.
- One remote linux to syslog.
- Four remote windows to syslog.
- Two remote windows to snort.

There was no masive logging because there will always be a limit of how many logs can be procesed. It is more important to choose the logs carefully. The importance of this test was to check the possibility of interoperability.

14 Evaluation

14.1 Evaluation criteria

For all the prototypes there are explanations of:

1. How it has been fully developed. In the design and development there is a detailed explanation that would allow anyone to reproduce the project or expand it. Nothing is taken for granted. It was written at the same time as it was being done, so nothing could be left out.

2. If not, there is a detailed analysis of what has been accomplished and what not. There is a section of possible improvements, but the project's goals have been fully achieved. It can work perfectly without any of those improvements.

3. The knowledge earned. This report is the proof of the knowledge accomplished in logging, Windows, Linux, PostgreSQL, Snort, system configuration and programming.

4. How to extend the prototypes to gather information from other systems or other programs easily. This architecture can be extended without modification, there would be some cases where it is not that way, but they are explained in the possible improvements section.

There are going to be three kinds of project evaluation:

- The aim and objectives reached. In particularly, there is a database in a server where all the events generated from the different systems are stored.

- A comparison with other approaches to detect the strengths and weaknesses.

- Testing.

14.2 Aim and Objectives

The aim of the project was to develop event logging architecture that was able to centralize the logging capabilities of Linux, Windows NT and Snort. It has been accomplished.

Part of the aim was to store the messages in a database to ease later analysis. This has been accomplished: all the messages had been stored in a database, using the snort database. THis not only allows the utilization of the advantages of a database search, but also the use of already existing tools and methodologies designed for search in the snort database.

In order to achieve this aim the following objectives were identified and reached:

- A study of the current event logging capabilities in Linux, Windows and Snort. A study of the logging capabilities was done. Especially the event message format and meaning. That information was very useful for the later development of the project.

- A study of how the capabilities can inter-operate with a Syslogd server. Two architectures that were able to centralize the messages were designed and the best one for the main aim was chosen. In the case of windows, where no mechanism was already embedded to allow the logging, there was a study and evaluation of existing programs that allow it, and one that seems to be perfect for the purpose was chosen.

- Implementation of an architecture that centralizes the events logging from the different sources.

- Evaluation of the architecture. Finally there was an evaluation of the architecture, with special emphasis on the performance and error tolerance. It passed the performance and error tolerance tests successfully. The only thing missing was a error notification.

The final result is a centralized system that can store information in a unique database from:

- Snort.

- Local Syslog information.

- Remote Syslog information.

- Windows event information.

The aims were successfully achieved. Because of the way the different systems communicate, it would be very easy to add another source. It would depend more on the nature of the source than on the architecture created. That is so because there is no modification of the systems, just configuration making use of what there is already. And when necessary, the creation of bridges, as in the case of the program that logs the information sent to a pipe by Syslog. There are some corrections that should be made in the program to support the standard of Syslog, but these are trivial modifications.

14.3 Comparison with other approaches

There is a comparison and study in the literature review. It shows how this approach has been discussed and suggested in forums, but there is no public implementation available. There are explanations of why this approach is better, the most relevant of which are:

- Centralization for better analysis.

- Centralization in a database to ease analysis.

- Avoiding interference and modifications in systems when possible

All were taken into account during the project and were decisive in all the decisions on architectures and solutions. All were successfully achieved.

14.4 Testing

Testing was carried out during all parts of the project and has already been explained. It found some problems that were corrected.

15 Conclusions

Event Logging is a very important security and maintenance mechanism and it is vital for any machine. It gives system administrators the ability to determine any software or hardware error, even a security break. The centralized logging architecture provides a easy way of getting all the information, search and analysing it.

It has some drawbacks because the remotely logging, can create important security threats. Another disadvantage is that what to log must be chosen carefully or the system will be overwhelmed, and even if it does not crash, important messages could be discarded.

The use of open source software was a good choice. It is better developed in security and very well documented. The possibility of being able to check the snort source code allowed the author to check in coding time if there were any incompatibilities between the logger and snort output plugging, instead of having to use trial and error or reverse engineering, which would have taken much longer.

16 Appendix A: Database creation script

```
-- This script is a modification of the original.
-- Done By Jose Antonio Fernandes Salvador: varelse@euskalnet.net
--
-- Copyright (C) 2000-2002 Carnegie Mellon University
--
-- Maintainer: Roman Danyliw <rdd@cert.org>, <roman@danyliw.com>
--
-- Original Author(s): Jed Pickel <jed@pickel.net>    (2000-2001)
--                     Roman Danyliw <rdd@cert.org>
--                     Todd Schrubb <tls@cert.org>
--
-- This program is free software; you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

CREATE TABLE schema ( vseq        INT4     NOT NULL,
                      ctime       TIMESTAMP with time zone NOT NULL,
                      PRIMARY KEY (vseq));
INSERT INTO schema  (vseq, ctime) VALUES ('106', now());

CREATE TABLE signature ( sig_id       SERIAL NOT NULL,
                         sig_name     TEXT NOT NULL,
                         sig_class_id INT8,
                         sig_priority INT8,
                         sig_rev      INT8,
                         sig_sid      INT8,
                         PRIMARY KEY (sig_id));
CREATE INDEX sig_name_idx ON signature (sig_name);
CREATE INDEX sig_class_idx ON signature (sig_class_id);

CREATE TABLE sig_reference (sig_id  INT4  NOT NULL,
                            ref_seq INT4  NOT NULL,
                            ref_id  INT4  NOT NULL,
                            PRIMARY KEY(sig_id, ref_seq));

CREATE TABLE reference (  ref_id        SERIAL,
```

```
                        ref_system_id INT4 NOT NULL,
                        ref_tag       TEXT NOT NULL,
                        PRIMARY KEY (ref_id));
CREATE INDEX ref_tag_idx ON reference (ref_tag);

CREATE TABLE reference_system ( ref_system_id   SERIAL,
                                ref_system_name TEXT,
                                PRIMARY KEY (ref_system_id));

CREATE TABLE sig_class ( sig_class_id        SERIAL,
                         sig_class_name      TEXT NOT NULL,
                         PRIMARY KEY (sig_class_id) );
CREATE INDEX sig_class_name_idx ON sig_class (sig_class_name);
INSERT INTO sig_class (sig_class_id, sig_class_name) VALUES (1,aplication);

CREATE TABLE event  ( sid    INT4 NOT NULL,
                      cid    INT8 NOT NULL,
                      signature   INT4 NOT NULL,
                      timestamp   timestamp with time zone NOT NULL,
                      PRIMARY KEY (sid,cid));
CREATE INDEX signature_idx ON event (signature);
CREATE INDEX timestamp_idx ON event (timestamp);

-- store info about the sensor supplying data CREATE TABLE sensor ( sid
SERIAL,
                      hostname    TEXT,
                      interface   TEXT,
                      filter   TEXT,
                      detail   INT2,
                      encoding   INT2,
                      last_cid    INT8 NOT NULL,
                      PRIMARY KEY (sid));
CREATE INDEX hostname_idx ON sensor (hostname);

-- All of the fields of an ip header
CREATE TABLE iphdr  ( sid    INT4 NOT NULL,
                      cid    INT8 NOT NULL,
                      ip_src      INT8 NOT NULL,
                      ip_dst      INT8 NOT NULL,
                      ip_ver      INT2,
                      ip_hlen     INT2,
                      ip_tos     INT2,
                      ip_len    INT4,
                      ip_id       INT4,
                      ip_flags    INT2,
                      ip_off      INT4,
                      ip_ttl      INT2,
                      ip_proto    INT2 NOT NULL,
```

```
                    ip_csum    INT4,
                    PRIMARY KEY (sid,cid));
CREATE INDEX ip_src_idx ON iphdr (ip_src);
CREATE INDEX ip_dst_idx ON iphdr (ip_dst);

-- All of the fields of a tcp header
CREATE TABLE tcphdr(  sid    INT4 NOT NULL,
                    cid    INT8 NOT NULL,
                    tcp_sport   INT4 NOT NULL,
                    tcp_dport   INT4 NOT NULL,
                    tcp_seq     INT8,
                    tcp_ack     INT8,
                    tcp_off     INT2,
                    tcp_res     INT2,
                    tcp_flags   INT2 NOT NULL,
                    tcp_win     INT4,
                    tcp_csum    INT4,
                    tcp_urp     INT4,
                    PRIMARY KEY (sid,cid));
CREATE INDEX tcp_sport_idx ON tcphdr (tcp_sport);
CREATE INDEX tcp_dport_idx ON tcphdr (tcp_dport);
CREATE INDEX tcp_flags_idx ON tcphdr (tcp_flags);

-- All of the fields of a udp header
CREATE TABLE udphdr(  sid    INT4 NOT NULL,
                    cid    INT8 NOT NULL,
                    udp_sport   INT4 NOT NULL,
                    udp_dport   INT4 NOT NULL,
                    udp_len     INT4,
                    udp_csum    INT4,
                    PRIMARY KEY (sid,cid));
CREATE INDEX udp_sport_idx ON udphdr (udp_sport);
CREATE INDEX udp_dport_idx ON udphdr (udp_dport);

-- All of the fields of an icmp header
CREATE TABLE icmphdr( sid    INT4 NOT NULL,
                    cid    INT8 NOT NULL,
                    icmp_type   INT2 NOT NULL,
                    icmp_code   INT2 NOT NULL,
                    icmp_csum   INT4,
                    icmp_id     INT4,
                    icmp_seq    INT4,
                    PRIMARY KEY (sid,cid));
CREATE INDEX icmp_type_idx ON icmphdr (icmp_type);

-- Protocol options
CREATE TABLE opt    ( sid         INT4 NOT NULL,
                    cid         INT8 NOT NULL,
```

```
                      optid       INT2 NOT NULL,
                      opt_proto   INT2 NOT NULL,
                      opt_code    INT2 NOT NULL,
                      opt_len     INT4,
                      opt_data    TEXT,
                      PRIMARY KEY (sid,cid,optid));

-- Packet payload
CREATE TABLE data   ( sid          INT4 NOT NULL,
                      cid          INT8 NOT NULL,
                      data_payload TEXT,
                      PRIMARY KEY (sid,cid));

-- encoding is a lookup table for storing encoding types
CREATE TABLE encoding(encoding_type INT2 NOT NULL,
                      encoding_text TEXT NOT NULL,
                      PRIMARY KEY (encoding_type));
INSERT INTO encoding (encoding_type, encoding_text) VALUES (0, 'hex');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (1, 'base64');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (2, 'ascii');

-- detail is a lookup table for storing different detail levels
CREATE TABLE detail  (detail_type INT2 NOT NULL,
                      detail_text TEXT NOT NULL,
                      PRIMARY KEY (detail_type));
INSERT INTO detail (detail_type, detail_text) VALUES (0, 'fast');
INSERT INTO detail (detail_type, detail_text) VALUES (1, 'full');

-- be sure to also use the snortdb-extra tables if you want
-- mappings for tcp flags, protocols, and ports
```

17 Appendix B: Program source code

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include "/usr/include/postgresql/libpq-fe.h"

//For daemon
#include <stdio.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>

#define BUFFERSIZE 256
#define LF '\n'


void signal_handler(sig)
int sig;
{
  switch(sig) {
  case SIGHUP:
    log_message(LOG_FILE,"hangup signal catched");
    break;
  case SIGTERM:
    log_message(LOG_FILE,"terminate signal catched");
    exit(0);
    break;
  }
}

void daemonize()
{
int i,lfp;
char str[10];
  if(getppid()==1) return; /* already a daemon */
  i=fork();
  if (i<0) exit(1); /* fork error */
  if (i>0) exit(0); /* parent exits */
```

```
  /* child (daemon) continues */
  setsid(); /* obtain a new process group */
  for (i=getdtablesize();i>=0;--i) close(i); /* close all descriptors */
  i=open("/dev/null",O_RDWR); dup(i); dup(i); /* handle standart I/O */
  umask(027); /* set newly created file permissions */
  chdir(RUNNING_DIR); /* change running directory */
  lfp=open(LOCK_FILE,O_RDWR|O_CREAT,0640);
  if (lfp<0) exit(1); /* can not open */
  if (lockf(lfp,F_TLOCK,0)<0) exit(0); /* can not lock */
  /* first instance continues */
  sprintf(str,"%d\n",getpid());
  write(lfp,str,strlen(str)); /* record pid to lockfile */
  signal(SIGCHLD,SIG_IGN); /* ignore child */
  signal(SIGTSTP,SIG_IGN); /* ignore tty signals */
  signal(SIGTTOU,SIG_IGN);
  signal(SIGTTIN,SIG_IGN);
  signal(SIGHUP,signal_handler); /* catch hangup signal */
  signal(SIGTERM,signal_handler); /* catch kill signal */
}

main()
{
  daemonize();
int main(int argc, char **argv)
{

  struct message {    //To store pointers to the entry parts
    char *time;
    char *host;
    char *aplic;
    char *mess;
  } BDEntry;

  // Named Pipe management.
  int fdPipe;              // Pipe descriptor
  char cPipe[]= "/tmp/myLog.pipe";  // Pipe location

  //Aux variables to manage the buffer
  size_t iBytesLeftBuffer=0;      // Extra bytes read of the next message
  char cBuffer[BUFFERSIZE];      // Read buffer
  char *cpBufferOffset = cBuffer;    // End of data in the buffer
  int iReadedBytes;          // Number of bytes read from the pipe

  char *cpEOL;            // Mark for the end of the line

  //Aux variables to parse the message into his parts
  int pos;          // Beggining of a entry part
  int posFin;          // Size until the end of a entry part
```

```
 int sizeEOL;        // Size of a full entry
char *cpBegin;        // Beggining of the actual entry
char *espacio = " ";    // Space character
char *dosPuntos = ":";    // Colon character
char *finLinea = "/n";    // End of line character

//DB variables
PGconn *conn;        // For connection
PGresult *cpResult;      // For get the results
char query[BUFFERSIZE];    // To built the query
int resultH = 0;       // To store the sensor id
int resultA = 0;       // To store the signature id
int resultC = 0;       // To store the last sensor id
int resultS = 0;       // To store the reference id
int resultF = 0;      // To store the maximum cid

// Time management
time_t t;
struct tm* petm;
int anio;

// To get the actual year
time(&t);
petm= localtime(&t);
anio = petm->tm_year + 1900;

//Open the Named Pipe. It is open to write too to avoid to receive a EOF
                                  // if the writing process has not started yet
if ( (fdPipe = open(cPipe, O_RDWR)) < 0 ) {
  printf("The Named Pipe [%s] could not be opened.\n", cPipe);
} else {
  printf("Pipe opened:\n", cPipe);

  //Connect to the database
  conn = PQconnectdb("dbname=snort3");
  if (PQstatus(conn) == CONNECTION_BAD) {
    exit(-1);
  } else {
    while (1) {
    // Read the pipe
      while ((iReadedBytes =
               read(fdPipe,cpBufferOffset+iBytesLeftBuffer * sizeof(char),
                                  (BUFFERSIZE - iBytesLeftBuffer -1)))==0) {
        sleep(5);
      }
      iBytesLeftBuffer = iBytesLeftBuffer + iReadedBytes;

      while ((cpEOL = memchr(cpBufferOffset,LF,(size_t)iBytesLeftBuffer))
```

```
                                !=NULL) {
    // All entry size calculation
    sizeEOL = cpEOL - cpBufferOffset;

// Parse the message
    // Parse date
    pos = 0;
    posFin=15;
    pos=pos+posFin+1;

    //Avoid incorrect entries
    if (sizeEOL > pos) {
      BDEntry.time = cpBufferOffset;
      *(BDEntry.time + posFin)= '\0';

      // Parse machine or sensor or host
      cpBegin = cpBufferOffset + pos;
      posFin = strcspn(cpBegin,espacio);
      pos=posFin+pos+1;
      //Avoid incorrect entries
      if (sizeEOL > pos) {
        BDEntry.host = cpBegin;
        *(BDEntry.host + posFin)= '\0';

        // Parse aplication
        cpBegin=cpBufferOffset+ pos;
        posFin = strcspn(cpBegin, dosPuntos);
        pos=posFin+pos+2;
        //Avoid incorrect entries
        if (sizeEOL > pos) {
          BDEntry.aplic = cpBegin;
          *(BDEntry.aplic + posFin)= '\0';

          // Parse message
          cpBegin=cpBufferOffset+pos;
          posFin = cpEOL -cpBegin;
          pos=pos + posFin;
          //Avoid incorrect entries
          if (sizeEOL == pos) {
            BDEntry.mess = cpBegin;
            *(BDEntry.mess + posFin)= '\0';

      //Database insertion
            //Host to sensor
            sprintf(query,"SELECT sid FROM sensor
                        WHERE hostname = '%s'",BDEntry.host);
            cpResult = PQexec(conn,query);
```

```
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {

if (PQntuples(cpResult)) {
  resultH = atoi(PQgetvalue(cpResult,0,0));
  fprintf(stdout,"Encuentra el sensor\n");
  PQclear(cpResult);

  //Aplication to reference
  sprintf(query,"SELECT ref_id FROM reference
           WHERE ref_tag = '%s'",BDEntry.aplic);
  cpResult = PQexec(conn,query);
  if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
    fprintf(stdout,"Query Ok para Aplic");
      if (PQntuples(cpResult) < 1) {
        PQclear(cpResult);
        sprintf(query, "INSERT INTO reference
                    (ref_system_id, ref_tag)
                          VALUES (1,'%s')",BDEntry.aplic);
        PQexec(conn,query);
        sprintf(query,"SELECT ref_id FROM reference
                   WHERE ref_tag = '%s'",BDEntry.aplic);
        cpResult = PQexec(conn,query);
      }
      resultA = atoi(PQgetvalue(cpResult,0,0));
  }
  PQclear(cpResult);


  //Message to signature
  sprintf(query,"SELECT sig_id FROM signature
                 WHERE sig_name = '%s'",BDEntry.mess);
  cpResult = PQexec(conn,query);
  if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
      if (PQntuples(cpResult) < 1) {
        PQclear(cpResult);
        sprintf(query, "INSERT INTO signature
             (sig_name, sig_class_id)
                    VALUES ('%s',1)",BDEntry.mess);
        PQexec(conn,query);
        sprintf(query,"SELECT sig_id FROM signature
                    WHERE sig_name = '%s'",BDEntry.mess);
        cpResult = PQexec(conn,query);
      }
      resultA = atoi(PQgetvalue(cpResult,0,0));
  }
  PQclear(cpResult);
  //Start transaction
  cpResult = PQexec(conn,"BEGIN");
```

```
PQclear(cpResult);

//Time to event
sprintf(query,"SELECT last_cid FROM sensor
            WHERE sid = %d",resultH);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
    fprintf(stdout,"Encuentra el last_cid\n");
    resultC = atoi(PQgetvalue(cpResult,0,0));
}
PQclear(cpResult);

//Update last sid
sprintf(query, "UPDATE sensor SET last_cid =%d
                           WHERE sid =%d",resultC + 1, resultH);
PQexec(conn,query);
fprintf(stdout,"Query: %s\n", query);

//End transaction
cpResult = PQexec(conn,"COMMIT");
PQclear(cpResult);

//Relation between messages, aplications, signatures and referenc
sprintf(query,"SELECT max(ref_seq) FROM sig_reference
                  WHERE sig_id = %d",resultA);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  resultS = atoi(PQgetvalue(cpResult,0,0));
} else {
  resultS = 0;
}
PQclear(cpResult);

sprintf(query, "INSERT INTO sig_reference
      (sig_id, ref_seq, ref_id)
            VALUES (%d, %d, %d)",resultA, resultS+1, 1);
PQexec(conn,query);

//Store event
sprintf(query,"SELECT max(cid) FROM event
                WHERE sid = %d",resultH);
cpResult = PQexec(conn,query);
if ((PQresultStatus(cpResult) == PGRES_TUPLES_OK)) {
  resultF = atoi(PQgetvalue(cpResult,0,0));
} else {
  resultF = 0;
}
PQclear(cpResult);
```

```
                    if (*(BDEntry.time + 4) == ' ') {
                      *(BDEntry.time + 4) ='0';
                    }

                    sprintf(query, "INSERT INTO event
                        (sid, cid, signature, timestamp)
                           VALUES (%d, %d, %d,
                              TO_TIMESTAMP('%d %s','YYYY Mon DD HH24:MI:SS'))",
                                 resultH, resultF+1, resultA, anio, BDEntry.time);
                    PQexec(conn,query);

                  }
                }

                  //All this allows fail safe (tolerancia a fallos).
            };

            //Move to the next entry in the buffer
            cpBufferOffset = BDEntry.mess + posFin + 1;
            iBytesLeftBuffer=iBytesLeftBuffer - sizeEOL - 1;

            BDEntry.time = NULL;
            BDEntry.host = NULL;
            BDEntry.aplic = NULL;
            BDEntry.mess = NULL;
          }


          //When there is no full entries anymore it is possible that there is
          //the beggining of a entry, so it must be moved to the beggining
          //of the buffer before reading it
          strncpy(cBuffer, cpBufferOffset, iBytesLeftBuffer);
          cpBufferOffset = cBuffer;
        }
      }
    };

    close(fdPipe);

    return(1);
    }
}
```

18 Appendix C: Starting and Stopping Snort script

```
#!/bin/bash
# $Id: S99snort,v 1.1 2001/12/18 22:14:37 cazz Exp $
# /etc/init.d/snort : start or stop the SNORT Intrusion Database System
#
# Written by Lukasz Szmit <ptashek@scg.gliwice.pl>
#
# Configuration

# set config file & path to snort executable
SNORT_PATH=/usr/local/bin
CONFIG=/usr/snort/snort.conf

# set interface
IFACE=eth0

# set GID/Group Name
SNORT_GID=nogroup

# other options
OPTIONS="-D"

# End of configuration


test -x $SNORT_PATH/snort || exit 0

case "$1" in
     start)
echo "Starting Intrusion Database System: SNORT"
$SNORT_PATH/snort -c $CONFIG -i $IFACE $OPTIONS
if [ "`pidof $SNORT_PATH/snort`" ]; then
echo "SNORT is up and running!"
else
exit 0
fi
echo -n "."
;;

     stop)
echo "Stoping Intrusion Database System: SNORT"
if [ "`pidof $SNORT_PATH/snort`" ] ; then

    kill -TERM `pidof $SNORT_PATH/snort`

    # Wait until the timeout
    count=120
```

```
    numdots=0
    while ([ $count != 0 ]) do
let count=$count-1

if [ "‘pidof $SNORT_PATH/snort‘" ] ; then
    echo -n .
    let numdots=$numdots+1
    sleep 1
else
    count=0
fi
    done

    # If it’s not dead yet, kill it.

    if [ "‘pidof $SNORT_PATH/snort‘" ] ; then
echo " TIMEOUT!"
kill -KILL ‘$SNORT_PATH/snort‘
    else
case $numdots in
  0) echo "." ;;
  1) echo ;;
  *) echo " done." ;;
esac
    fi
else
    echo "SNORT is  not running!";
fi
;;
     restart)
$0 stop
$0 start
;;
     *)
echo ’Usage: /etc/init.d/snort {start|stop|restart}’
exit 1
;;
esac
exit 0
;;
```

19 Appendix D: Snort configuration file

```
#--------------------------------------------------
#   http://www.snort.org     Snort 2.0.0 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# $Id: snort.conf,v 1.124 2003/05/16 02:52:41 cazz Exp $
#
##################################################
# This file contains a sample snort configuration.
# You can take the following steps to create your
# own custom configuration:
#
#  1) Set the network variables for your network
#  2) Configure preprocessors
#  3) Configure output plugins
#  4) Customize your rule set
#
##################################################
# Step #1: Set the network variables:
#
# You must change the following variables to reflect
# your local network. The variable is currently
# setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
# var HOME_NET 10.1.1.0/24
#
# or use global variable $<interfacename>_ADDRESS
# which will be always initialized to IP address and
# netmask of the network interface which you run
# snort at.  Under Windows, this must be specified
# as $(<interfacename>_ADDRESS), such as:
# $(\Device\Packet_{12345678-90AB-CDEF-1234567890AB}_ADDRESS)
#
# var HOME_NET $eth0_ADDRESS
#
# You can specify lists of IP addresses for HOME_NET
# by separating the IPs with commas like this:
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:
```

```
var HOME_NET any

# Set up the external network addresses as well.

# A good start may be "any"

var EXTERNAL_NET any

# Configure your server lists.  This allows snort to only look for attacks
# to systems that have a service up.  Why look for HTTP attacks if you are
# not running a web server?  This allows quick filtering based on IP addresses
# These configurations MUST follow the same configuration scheme as defined
# above for $HOME_NET.

# List of DNS servers on your network
var DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
var SMTP_SERVERS $HOME_NET

# List of web servers on your network
var HTTP_SERVERS $HOME_NET

# List of sql servers on your network
var SQL_SERVERS $HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS $HOME_NET

# Configure your service ports.  This allows snort to look for attacks
# destined to a specific application only on the ports that application
# runs on.  For example, if you run a web server on port 8081, set your
# HTTP_PORTS variable like this:
#
# var HTTP_PORTS 8081
#
# Port lists must either be continuous [eg 80:8080], or a single port [eg 80].
# We will adding support for a real list of ports in the future.

# Ports you run web servers on
var HTTP_PORTS 80

# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80

# Ports you do oracle attacks on
var ORACLE_PORTS 1521
```

```
# other variables
#
# AIM servers.  AOL has a habit of adding new AIM servers, so instead of
# modifying the signatures when they do, we add them to this list of
# servers.
var AIM_SERVERS [64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.

# Path to your rules files (this can be a relative path)
var RULE_PATH ../rules

# Configure the snort decoder:
# ============================
#
# Stop generic decode events:
#
# config disable_decode_alerts
#
# Stop Alerts on experimental TCP options
#
# config disable_tcpopt_experimental_alerts
#
# Stop Alerts on obsolete TCP options
#
# config disable_tcpopt_obsolete_alerts
#
# Stop Alerts on T/TCP alerts
#
# config disable_ttcp_alerts
#
# Stop Alerts on all other TCPOption type events:
#
# config disable_tcpopt_alerts
#
# Stop Alerts on invalid ip options
#
# config disable_ipopt_alerts


# Configure the detection engine
# ==============================
#
# Use a different pattern matcher in case you have a machine with very
# limited resources:
#
# config detection: search-method lowmem


##################################################
```

```
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# frag2: IP defragmentation support
# ------------------------------
# This preprocessor performs IP defragmentation.  This plugin will also detect
# people launching fragmentation attacks (usually DoS) against hosts.  No
# arguments loads the default configuration of the preprocessor, which is a
# 60 second timeout and a 4MB fragment buffer.

# The following (comma delimited) options are available for frag2
#    timeout [seconds] - sets the number of [seconds] than an unfinished
#                        fragment will be kept around waiting for completion,
#                        if this time expires the fragment will be flushed
#    memcap [bytes] - limit frag2 memory usage to [number] bytes
#                       (default:  4194304)
#
#    min_ttl [number] - minimum ttl to accept
#
#    ttl_limit [number] - difference of ttl to accept without alerting
#                         will cause false positves with router flap
#
# Frag2 uses Generator ID 113 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Oversized fragment (reassembled frag > 64k bytes)
#   2       Teardrop-type attack

preprocessor frag2

# stream4: stateful inspection/stream reassembly for Snort
#--------------------------------------------------------------------
# Use in concert with the -z [all|est] command line switch to defeat
# stick/snot against TCP rules.  Also performs full TCP stream
# reassembly, stateful inspection of TCP streams, etc.  Can statefully
# detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
#   detect_scans - stream4 will detect stealth portscans and generate alerts
#                  when it sees them when this option is set
#   detect_state_problems - detect TCP state problems, this tends to be very
#                           noisy because there are a lot of crappy ip stack
```

```
#                           implementations out there
#
#   disable_evasion_alerts - turn off the possibly noisy mitigation of
#                            overlapping sequences.
#
#
#   min_ttl [number]       - set a minium ttl that snort will accept to
#                            stream reassembly
#
#   ttl_limit [number]     - differential of the initial ttl on a session versus
#                             the normal that someone may be playing games.
#                             Routing flap may cause lots of false positives.
#
#   keepstats [machine|binary] - keep session statistics, add "machine" to
#                         get them in a flat format for machine reading, add
#                         "binary" to get them in a unified binary output
#                         format

#   noinspect - turn off stateful inspection only
#   timeout [number] - set the session timeout counter to [number] seconds,
#                       default is 30 seconds
#   memcap [number] - limit stream4 memory usage to [number] bytes
#   log_flushed_streams - if an event is detected on a stream this option will
#                         cause all packets that are stored in the stream4
#                         packet buffers to be flushed to disk.  This only
#                         works when logging in pcap mode!
#
# Stream4 uses Generator ID 111 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Stealth activity
#   2       Evasive RST packet
#   3       Evasive TCP packet retransmission
#   4       TCP Window violation
#   5       Data on SYN packet
#   6       Stealth scan: full XMAS
#   7       Stealth scan: SYN-ACK-PSH-URG
#   8       Stealth scan: FIN scan
#   9       Stealth scan: NULL scan
#   10      Stealth scan: NMAP XMAS scan
#   11      Stealth scan: Vecna scan
#   12      Stealth scan: NMAP fingerprint scan stateful detect
#   13      Stealth scan: SYN-FIN scan
#   14      TCP forward overlap

preprocessor stream4: detect_scans, disable_evasion_alerts
```

```
# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#   clientonly - reassemble traffic for the client side of a connection only
#   serveronly - reassemble traffic for the server side of a connection only
#   both - reassemble both sides of a session
#   noalerts - turn off alerts from the stream reassembly stage of stream4
#   ports [list] - use the space separated list of ports in [list], "all"
#                  will turn on reassembly for all ports, "default" will turn
#                  on reassembly for ports 21, 23, 25, 53, 80, 143, 110, 111
#                  and 513

preprocessor stream4_reassemble

# http_decode: normalize HTTP requests
# ------------------------------------
# http_decode normalizes HTTP requests from remote
# machines by converting any %XX character
# substitutions to their ASCII equivalent. This is
# very useful for doing things like defeating hostile
# attackers trying to stealth themselves from IDSs by
# mixing these substitutions in with the request.
# Specify the port numbers you want it to analyze as arguments.
#
# Major code cleanups thanks to rfp
#
# unicode          - normalize unicode
# iis_alt_unicode  - %u encoding from iis
# double_encode    - alert on possible double encodings
# iis_flip_slash   - normalize \ as /
# full_whitespace  - treat \t as whitespace ( for apache )
#
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       UNICODE attack
#   2       NULL byte attack

preprocessor http_decode: 80 unicode iis_alt_unicode double_encode iis_flip_slash f

# rpc_decode: normalize RPC traffic
# ---------------------------------
# RPC may be sent in alternate encodings besides the usual
# 4-byte encoding that is used by default.  This preprocessor
```

```
# normalized RPC traffic in much the same way as the http_decode
# preprocessor.  This plugin takes the ports numbers that RPC
# services are running on as arguments.
# The RPC decode preprocessor uses generator ID 106
#
# arguments: space separated list
# alert_fragments - alert on any rpc fragmented TCP data
# no_alert_multiple_requests - don't alert when >1 rpc query is in a packet
# no_alert_large_fragments - don't alert when the fragmented
#                            sizes exceed the current packet size
# no_alert_incomplete - don't alert when a single segment
#                       exceeds the current packet size

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
# -------------------------
# Detects Back Orifice traffic on the network.  Takes no arguments in 2.0.
#
# The Back Orifice detector uses Generator ID 105 and uses the
# following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Back Orifice traffic detected

preprocessor bo

# telnet_decode: Telnet negotiation string normalizer
# ---------------------------------------------------
# This preprocessor "normalizes" telnet negotiation strings from
# telnet and ftp traffic.  It works in much the same way as the
# http_decode preprocessor, searching for traffic that breaks up
# the normal data stream of a protocol and replacing it with
# a normalized representation of that traffic so that the "content"
# pattern matching keyword can work without requiring modifications.
# This preprocessor requires no arguments.
# Portscan uses Generator ID 109 and does not generate any SID currently.

preprocessor telnet_decode

# Portscan: detect a variety of portscans
# ---------------------------------------
# portscan preprocessor by Patrick Mullen <p_mullen@linuxrc.net>
# This preprocessor detects UDP packets or TCP SYN packets going to
# four different ports in less than three seconds. "Stealth" TCP
# packets are always detected, regardless of these settings.
# Portscan uses Generator ID 100 and uses the following SIDS for that GID:
#  SID     Event description
```

```
# -----   -------------------
#   1       Portscan detect
#   2       Inter-scan info
#   3       Portscan End

# preprocessor portscan: $HOME_NET 4 3 portscan.log

# Use portscan-ignorehosts to ignore TCP SYN and UDP "scans" from
# specific networks or hosts to reduce false alerts. It is typical
# to see many false alerts from DNS servers so you may want to
# add your DNS servers here. You can all multiple hosts/networks
# in a whitespace-delimited list.
#
#preprocessor portscan-ignorehosts: 0.0.0.0

# arpspoof
#---------------------------------------
# Experimental ARP detection code from Jeff Nathan, detects ARP attacks,
# unicast ARP requests, and specific ARP mapping monitoring.  To make use
# of this preprocessor you must specify the IP and hardware address of hosts on # t
# Also takes a "-unicast" option to turn on unicast ARP request detection.
# Arpspoof uses Generator ID 112 and uses the following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Unicast ARP request
#   2       Etherframe ARP mismatch (src)
#   3       Etherframe ARP mismatch (dst)
#   4       ARP cache overwrite attack

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

# Conversation
#---------------------------------------
# This preprocessor tracks conversations for tcp, udp and icmp traffic.  It
# is a prerequisite for running portscan2.
#
# allowed_ip_protcols 1 6 17
#      list of allowed ip protcols ( defaults to any )
#
# timeout [num]
#      conversation timeout ( defaults to 60 )
#
#
# max_conversations [num]
#      number of conversations to support at once (defaults to 65335)
#
#
```

```
# alert_odd_protocols
#      alert on protocols not listed in allowed_ip_protocols
#
# preprocessor conversation: allowed_ip_protocols all, timeout 60, max_conversation
#
# Portscan2
#-----------------------------------------
# Portscan 2, detect portscans in a new and exciting way.  You must enable
# spp_conversation in order to use this preprocessor.
#
# Available options:
#       scanners_max [num]
#       targets_max [num]
#       target_limit [num]
#       port_limit [num]
#       timeout [num]
#       log [logdir]
#
#preprocessor portscan2: scanners_max 256, targets_max 1024, target_limit 5, port_l

# Too many false alerts from portscan2? Tone it down with
# portscan2-ignorehosts!
#
# A space delimited list of addresses in CIDR notation to ignore
#
# preprocessor portscan2-ignorehosts: 10.0.0.0/8 192.168.24.0/24
#

# Experimental Perf stats
# -----------------------
# No docs. Highly subject to change.
#
# preprocessor perfmonitor: console flow events time 10

##################################################################
# Step #3: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.
# General configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
# alert_syslog: log alerts to syslog
# ----------------------------------
# Use one or more syslog facilities as arguments.  Win32 can also
# optionally specify a particular hostname/port.  Under Win32, the
# default hostname is '127.0.0.1', and the default port is 514.
#
```

```
# [Unix flavours should use this format...]
# output alert_syslog: LOG_AUTH LOG_ALERT
#
# [Win32 can use any of these formats...]
# output alert_syslog: LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname, LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
# output log_tcpdump: tcpdump.log

# database: log to a variety of databases
# ---------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test dbname=db host=localhost
output database: alert, postgresql, user=root password=12094216 dbname=snort
# output database: log, unixodbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test

# unified: Snort unified binary format alerting and logging
# -------------------------------------------------------------
# The unified output plugin provides two new formats for logging
# and generating alerts from Snort, the "unified" format.  The
# unified format is a straight binary format for logging data
# out of Snort that is designed to be fast and efficient.  Used
# with barnyard (the new alert/log processor), most of the overhead
# for logging and alerting to various slow storage mechanisms
# such as databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#    filename - base filename to write to (current time_t is appended)
#    limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128

# You can optionally define new rule types and associate one or
# more output plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
```

```
# {
#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC Server";)
#
# This example will create a rule type that will log to syslog
# and a mysql database.
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#   output database: log, mysql, user=snort dbname=snort host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE
# redalert tcp $HOME_NET any -> $EXTERNAL_NET 31337 \
#   (msg:"Someone is being LEET"; flags:A+;)

# # Include classification & priority settings
#

include classification.config

#
# Include reference systems
#

include reference.config

#####################################################################
# Step #4: Customize your rule set
#
# Up to date snort rules are available at http://www.snort.org
#
# The snort web site has documentation about how to write your own
# custom snort rules.
#
# The rules included with this distribution generate alerts based on
# on suspicious activity. Depending on your network environment, your
# security policies, and what you consider to be suspicious, some of
# these rules may either generate false positives ore may be detecting
# activity you consider to be acceptable; therefore, you are
# encouraged to comment out rules that are not applicable in your
# environment.
#
```

```
# Note that using all of the rules at the same time may lead to
# serious packet loss on slower machines. YMMV, use with caution,
# standard disclaimers apply. :)
#
# The following individuals contributed many of rules in this
# distribution.
#
# Credits:
#   Ron Gula <rgula@securitywizards.com> of Network Security Wizards
#   Max Vision <vision@whitehats.com>
#   Martin Markgraf <martin@mail.du.gtn.com>
#   Fyodor Yarochkin <fygrave@tigerteam.net>
#   Nick Rogness <nick@rapidnet.com>
#   Jim Forster <jforster@rapidnet.com>
#   Scott McIntyre <scott@whoi.edu>
#   Tom Vandepoel <Tom.Vandepoel@ubizen.com>
#   Brian Caswell <bmc@snort.org>
#   Zeno <admin@cgisecurity.com>
#   Ryan Russell <ryan@securityfocus.com>
#
#==========================================
# Include all relevant rulesets here
#
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default.  These require tuning and maintance.
# Please read the included specific file for more information.
#==========================================

include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules

include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
```

```
include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/snmp.rules

include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/pop3.rules

include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules
# include $RULE_PATH/web-attacks.rules
# include $RULE_PATH/backdoor.rules
# include $RULE_PATH/shellcode.rules
# include $RULE_PATH/policy.rules
# include $RULE_PATH/porn.rules
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/virus.rules
# include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/local.rules
```

20 Appendix E: Starting and Stopping Syslog script

```
#! /bin/sh
# /etc/init.d/sysklogd: start the system log daemon.

PATH=/bin:/usr/bin:/sbin:/usr/sbin

pidfile=/var/run/syslogd.pid
binpath=/sbin/syslogd

test -x $binpath || exit 0

# Options for start/restart the daemons
#   For remote UDP logging use SYSLOGD="-r"
#
SYSLOGD="-r"

create_xconsole()
{
    if [ ! -e /dev/xconsole ]; then
mknod -m 640 /dev/xconsole p
    else
chmod 0640 /dev/xconsole
    fi
    chown root.adm /dev/xconsole
}

running()
{
    # No pidfile, probably no daemon present
    #
    if [ ! -f $pidfile ]
    then
return 1
    fi

    pid=`cat $pidfile`

    # No pid, probably no daemon present
    #
    if [ -z "$pid" ]
    then
return 1
    fi

    cmd=`cat /proc/$pid/cmdline | tr "\000" "\n"|head -1`

    # No syslogd?
```

```
    #
    if [ "$cmd" != "$binpath" ]
    then
return 1
    fi

    return 0
}

case "$1" in
  start)
    echo -n "Starting system log daemon: syslogd"
    create_xconsole
    start-stop-daemon --start --quiet --exec $binpath -- $SYSLOGD
    echo "."
    ;;
  stop)
    echo -n "Stopping system log daemon: syslogd"
    start-stop-daemon --stop --quiet --exec $binpath --pidfile $pidfile
    echo "."
    ;;
  reload|force-reload)
    start-stop-daemon --stop --quiet --signal 1 --exec $binpath --pidfile $pidfile
    ;;
  restart)
    echo -n "Stopping system log daemon: syslogd"
    start-stop-daemon --stop --quiet --exec $binpath --pidfile $pidfile
    echo "."
    sleep 1
    echo -n "Starting system log daemon: syslogd"
    start-stop-daemon --start --quiet --exec $binpath -- $SYSLOGD
    echo "."
    ;;
  reload-or-restart)
    if running
    then
start-stop-daemon --stop --quiet --signal 1 --exec $binpath --pidfile $pidfile
    else
start-stop-daemon --start --quiet --exec $binpath -- $SYSLOGD
    fi
    ;;
  *)
    echo "Usage: /etc/init.d/sysklogd {start|stop|reload|restart|force-reload|reloa
    exit 1
esac

exit 0
```

21 Appendix F: Syslog configuration file

```
#  /etc/syslog.conf Configuration file for syslogd.
#
# For more information see syslog.conf(5)
# manpage.

#
# First some standard logfiles.  Log by facility.
##

auth,authpriv.* /var/log/auth.log
*.*;auth,authpriv.none -/var/log/syslog
#cron.* /var/log/cron.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
lpr.* -/var/log/lpr.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
uucp.* /var/log/uucp.log
*.*  |/tmp/myLog.pipe

#
# Logging for the mail system.  Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info -/var/log/mail.info
mail.warn -/var/log/mail.warn
mail.err /var/log/mail.err

# Logging for INN news system
#
news.crit /var/log/news/news.crit
news.err /var/log/news/news.err
news.notice -/var/log/news/news.notice

#
# Some 'catch-all' logfiles.
#
*.=debug;\
auth,authpriv.none;\
news.none;mail.none -/var/log/debug
*.=info;*.=notice;*.=warn;\
auth,authpriv.none;\
cron,daemon.none;\
mail,news.none -/var/log/messages

#
```

```
# Emergencies are sent to everybody logged in.
#
*.emerg *

#
# I like to have messages displayed on the console, but only on a virtual
# console I usually leave idle.
#
#daemon,mail.*;\
# news.=crit;news.=err;news.=notice;\
# *.=debug;*.=info;\
# *.=notice;*.=warn /dev/tty8

# The named pipe /dev/xconsole is for the `xconsole' utility.  To use it,
# you must invoke `xconsole' with the `-file' option:
#
#    $ xconsole -file /dev/xconsole [...]
#
# NOTE: adjust the list below, or you'll go crazy if you have a reasonably
#      busy site..
#
daemon.*;mail.*;\
news.crit;news.err;news.notice;\
*.=debug;*.=info;\
*.=notice;*.=warn |/dev/xconsole
```

22 Appendix G: Pipe creation and Logger launching script

```
#!/bin/sh
mkfifo /tmp/myLog.pipe
/home/josean/project/LogToDb
```

23 Appendix H: Snort Database Schema

ACID: Database (v100-103) ER Diagram

Snort (and other devices) log to database with the following schema:

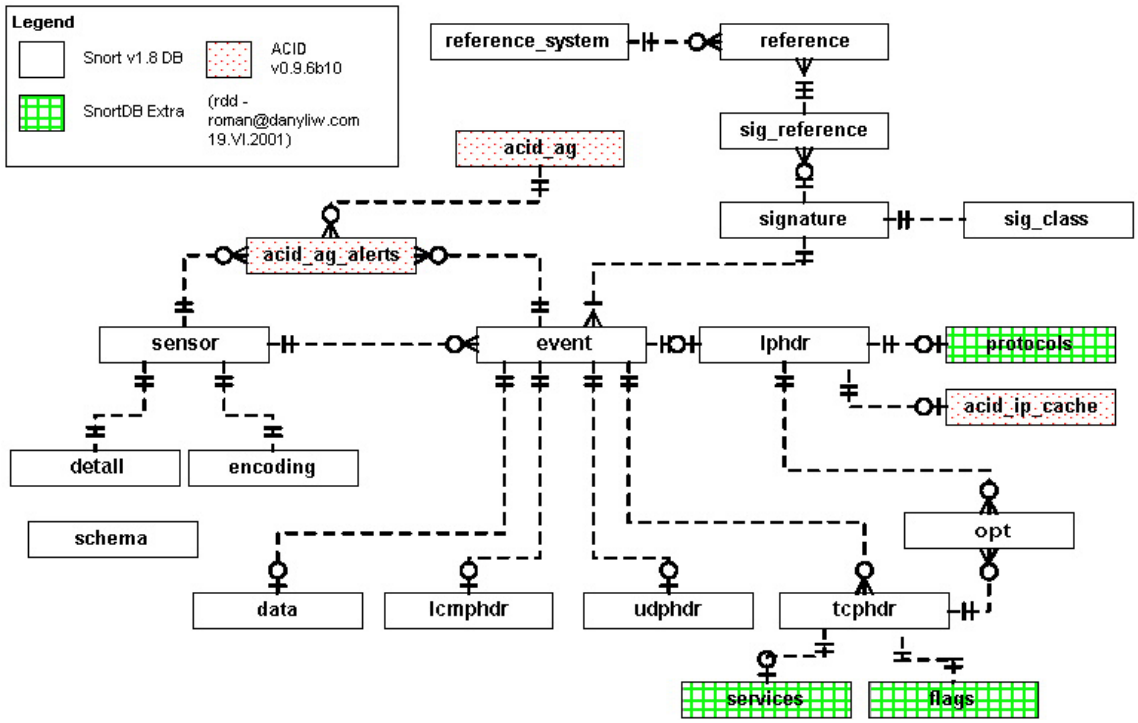

Figure 20: Snort Schema

Table	Component	Description
schema	Snort	Self-documented information about the database
sensor	Snort	Sensor name
event	Snort	Meta-data about the detected alert
signature	Snort	Normalized listing of alert/signature names, prior
sig_reference	Snort	Reference information for a signature
reference	Snort	Reference IDs for a signature
reference_system	Snort	(lookup table) Reference system list
sig_class	Snort	Normalized listing of alert/signature classificati
data	Snort	Contents of packet payload
iphdr	Snort	IP protocol fields
tcphdr	Snort	TCP protocol fields
udphdr	Snort	UDP protocol fields
icmphdr	Snort	ICMP protocol fields

```
| opt              | Snort     | IP and TCP options
| detail           | Snort     | (lookup table) Level of detail with which a sensor
| encoding         | Snort     | (lookup table) Type of encoding used for the packe
| protocols        | SnortDB   | extra (lookup table) Layer-4 (IP encoded) protocol
| services         | SnortDB   | extra (lookup table) TCP and UDP service list
| flags            | SnortDB   | extra (lookup table) TCP flag list
| acid_ag          | ACID      | Meta-data for alert groups
| acid_ag_alert    | ACID      | Alerts in each alert group
| acid_ip_cache    | ACID      | Cached DNS and whois information
+------------------+-----------+-------------------------------------------------
```

```
------------------------------------------------------------------------------
schema
------------------------------------------------------------------------------

+-------+------------------+------+-----+---------------------+-----------------
| Field | Type             | Null | Key | Default             | Description
+-------+------------------+------+-----+---------------------+-----------------
| vseq  | int(10) unsigned |      | PRI | 0                   | Database schema ID
| ctime | datetime         |      |     | 0000-00-00 00:00:00 | Timestamp of databa
+-------+------------------+------+-----+---------------------+-----------------
```

```
------------------------------------------------------------------------------
sensor
------------------------------------------------------------------------------

+-----------+------------------+------+-----+---------+----------------------------
| Field     | Type             | Null | Key | Default | Description
+-----------+------------------+------+-----+---------+----------------------------
| sid       | int(10) unsigned |      | PRI | NULL    | Sensor ID
| hostname  | text             | YES  |     | NULL    | Hostname of the sensor (IP
| interface | text             | YES  |     | NULL    | Network interface (e.g. eth
| filter    | text             | YES  |     | NULL    | BPF filter
| detail    | tinyint(4)       | YES  |     | NULL    | Detail level of the logging
| encoding  | tinyint(4)       | YES  |     | NULL    | Encoding format of the payl
+-----------+------------------+------+-----+---------+----------------------------
```

```
--------------------------------------------------------------------------------
event
--------------------------------------------------------------------------------

+-----------+------------------+------+-----+---------------------+----------------
| Field     | Type             | Null | Key | Default             | Description
+-----------+------------------+------+-----+---------------------+----------------
| sid       | int(10) unsigned |      | PRI | 0                   | Sensor ID
| cid       | int(10) unsigned |      | PRI | 0                   | Event ID
| signature | int(10) unsigned |      | MUL | 0                   | Signature ID
| timestamp | datetime         |      | MUL | 0000-00-00 00:00:00 | Timestamp of wh
+-----------+------------------+------+-----+---------------------+----------------

--------------------------------------------------------------------------------
signature
--------------------------------------------------------------------------------

+--------------+------------------+------+-----+---------+----------------------+
| Field        | Type             | Null | Key | Default | Description          |
+--------------+------------------+------+-----+---------+----------------------+
| sig_id       | int(10) unsigned |      | PRI | NULL    | Signature ID         |
| sig_name     | varchar(255)     |      | MUL |         | Signature Name       |
| sig_class_id | int(10) unsigned | YES  | MUL | NULL    | Classification ID    |
| sig_priority | int(10) unsigned | YES  |     | NULL    | Priority             |
| sig_rev      | int(10) unsigned | YES  |     | NULL    | Revision number      |
| sig_sid      | int(10) unsigned | YES  |     | NULL    | Internal signature ID |
+--------------+------------------+------+-----+---------+----------------------+

--------------------------------------------------------------------------------
sig_reference
--------------------------------------------------------------------------------

+---------+------------------+------+-----+---------+-----------------------------
| Field   | Type             | Null | Key | Default | Description
+---------+------------------+------+-----+---------+-----------------------------
| sig_id  | int(10) unsigned |      | PRI | 0       | Signature ID
| ref_seq | int(10) unsigned |      | PRI | 0       | Reference sequence number (mu
| ref_id  | int(10) unsigned |      |     | 0       | Reference ID
+---------+------------------+------+-----+---------+-----------------------------
```

reference

Field	Type	Null	Key	Default	Description
ref_id	int(10) unsigned		PRI	NULL	Reference ID
ref_system_id	int(10) unsigned			0	Reference system ID
ref_tag	varchar(20)				Reference tag (e.g. CVE

reference_system

Field	Type	Null	Key	Default	Description
ref_system_id	int(10) unsigned		PRI	NULL	Reference system ID
ref_system_name	varchar(20)	YES		NULL	Reference system name

sig_class

Field	Type	Null	Key	Default	Description
sig_class_id	int(10) unsigned		PRI	NULL	Signature classificati
sig_class_name	varchar(60)		MUL		Classification name (e

```
--------------------------------------------------------------------------------
data
--------------------------------------------------------------------------------

+--------------+------------------+------+-----+---------+-------------------------
| Field        | Type             | Null | Key | Default | Description
+--------------+------------------+------+-----+---------+-------------------------
| sid          | int(10) unsigned |      | PRI | 0       | Sensor ID
| cid          | int(10) unsigned |      | PRI | 0       | Event ID
| data_payload | text             | YES  |     | NULL    | Packet payload encoded a
+--------------+------------------+------+-----+---------+-------------------------



--------------------------------------------------------------------------------
iphdr
--------------------------------------------------------------------------------

+----------+----------------------+------+-----+---------+-------------------------
| Field    | Type                 | Null | Key | Default | Description
+----------+----------------------+------+-----+---------+-------------------------
| sid      | int(10) unsigned     |      | PRI | 0       | Sensor ID
| cid      | int(10) unsigned     |      | PRI | 0       | Event ID
| ip_src   | int(10) unsigned     |      | MUL | 0       | Source IP address (32-bi
| ip_dst   | int(10) unsigned     |      | MUL | 0       | Destination IP address (
| ip_ver   | tinyint(3) unsigned  | YES  |     | NULL    | IP version
| ip_hlen  | tinyint(3) unsigned  | YES  |     | NULL    | IP Header length
| ip_tos   | tinyint(3) unsigned  | YES  |     | NULL    | IP type-of-service
| ip_len   | smallint(5) unsigned | YES  |     | NULL    | IP datagram length
| ip_id    | smallint(5) unsigned | YES  |     | NULL    | IP ID
| ip_flags | tinyint(3) unsigned  | YES  |     | NULL    | IP flags
| ip_off   | smallint(5) unsigned | YES  |     | NULL    | IP fragment offset
| ip_ttl   | tinyint(3) unsigned  | YES  |     | NULL    | IP time-to-live
| ip_proto | tinyint(3) unsigned  |      |     | 0       | IP protocol
| ip_csum  | smallint(5) unsigned | YES  |     | NULL    | IP checksum
+----------+----------------------+------+-----+---------+-------------------------



--------------------------------------------------------------------------------
tcphdr
--------------------------------------------------------------------------------
```

Field	Type	Null	Key	Default	Description
sid	int(10) unsigned		PRI	0	Sensor ID
cid	int(10) unsigned		PRI	0	Event ID
tcp_sport	smallint(5) unsigned		MUL	0	TCP source port
tcp_dport	smallint(5) unsigned		MUL	0	TCP destination port
tcp_seq	int(10) unsigned	YES		NULL	TCP sequence number
tcp_ack	int(10) unsigned	YES		NULL	TCP ACK number
tcp_off	tinyint(3) unsigned	YES		NULL	TCP offset
tcp_res	tinyint(3) unsigned	YES		NULL	TCP reserved
tcp_flags	tinyint(3) unsigned		MUL	0	TCP flags
tcp_win	smallint(5) unsigned	YES		NULL	TCP window
tcp_csum	smallint(5) unsigned	YES		NULL	TCP checksum
tcp_urp	smallint(5) unsigned	YES		NULL	TCP urgent pointer

---

udphdr

Field	Type	Null	Key	Default	Description
sid	int(10) unsigned		PRI	0	Sensor ID
cid	int(10) unsigned		PRI	0	Event ID
udp_sport	smallint(5) unsigned		MUL	0	UDP soure port
udp_dport	smallint(5) unsigned		MUL	0	UDP destination port
udp_len	smallint(5) unsigned	YES		NULL	UDP length
udp_csum	smallint(5) unsigned	YES		NULL	UDP checksum

---

icmphdr

Field	Type	Null	Key	Default	Description
sid	int(10) unsigned		PRI	0	Sensor ID
cid	int(10) unsigned		PRI	0	Event ID

icmp_type	tinyint(3) unsigned		MUL	0	ICMP type
icmp_code	tinyint(3) unsigned			0	ICMP code
icmp_csum	smallint(5) unsigned	YES		NULL	ICMP checksum
icmp_id	smallint(5) unsigned	YES		NULL	ICMP ID
icmp_seq	smallint(5) unsigned	YES		NULL	ICMP sequence number

---

opt

Field	Type	Null	Key	Default	Description
sid	int(10) unsigned		PRI	0	Sensor ID
cid	int(10) unsigned		PRI	0	Event ID
optid	int(10) unsigned		PRI	0	Option ID (multiple opti
opt_proto	tinyint(3) unsigned			0	Option protocol (IP, TCP
opt_code	tinyint(3) unsigned			0	Option code
opt_len	smallint(6)	YES		NULL	Option length
opt_data	text	YES		NULL	Option data

---

acid_ag

Field	Type	Null	Key	Default	Description
ag_id	int(10) unsigned		PRI	NULL	Alert Group (AG) ID
ag_name	varchar(40)	YES		NULL	AG name
ag_desc	text	YES		NULL	AG description
ag_ctime	datetime	YES		NULL	Timestamp of AG creation tim
ag_ltime	datetime	YES		NULL	Timestamp of last AG modific

---

acid_ag_alert

Field	Type	Null	Key	Default	Description
ag_id	int(10) unsigned		PRI	0	Alert Group (AG) ID
ag_sid	int(10) unsigned		PRI	0	Sensor ID
ag_cid	int(10) unsigned		PRI	0	Event ID

acid_ip_cache

Field	Type	Null	Key	Default	Description
ipc_ip	int(10) unsigned		PRI	0	IP address (32-bi
ipc_fqdn	varchar(50)	YES	MUL	NULL	FQDN
ipc_dns_timestamp	datetime	YES		NULL	DNS lookup timest
ipc_whois	text	YES		NULL	whois information
ipc_whois_timestamp	datetime	YES		NULL	whois lookup time

References

[Aac03] Aacosta. *Using named pipes with syslog*, 2003. http://www.experts-exchange.com/Programming/Programming_Platforms/Linux_Programming/Q_20502703.html. (Accessed 11th May 2003).

[Ale03] Alejo. *Modular Syslog*, 2003. http://sourceforge.net/projects/msyslog/. (Accessed 11th May 2003).

[All01] Julia Allen. *The CERT Guide To System and Network Security Practices*. Addison-Wesley, 2001.

[BAC] *BackLog - Windows NT Event Redirection*. http://www.intersectalliance.com. (Accessed 8th May 2003).

[Bar03] J. Barber. *Web Hack*, 2003. http://www.ussysadmin.com/whcc/default.php. (Accessed 10th August 2003).

[BF03] Jay Beale and James C. Foster. *Snort 2.0, Intrusion detection*. Syngress, 2003.

[Bra00] Roberta Bragg. *Windows 2000 Security*. New Riders Publishing, 2000.

[Cha98] Ining Tracy Chao. *The Rapid Prototyping Model*. Concordia University, 1998. http://www.arts.ualberta.ca/ tchao/rpwebsite/. (Accessed 20th March 2003).

[CPL] CPL Systems. *Scrambler Alert*. http://www.Scrambler.net/. (Accessed 27th August 2003).

[Dan02] Roman Danyliw. *ACID: Database (v100-103) ER Diagram*, September 2002. http://www.andrew.cmu.edu/ rdanyliw/snort/acid_db_er_v102.html. (Accessed 15th July 2003).

[DEB] http://www.debian.org. (Accessed 12th June 2003).

[DIS] *Debian GNU/Linux Distributions*. http://debian.math.lsu.edu/distributions.html. (Accessed 12th June 2003).

[Ear02] Michael Earls. *Centralized syslog-ng to mysql database*, 2002. http://www.vermeer.org/syslog/. (Accessed 11th May 2003).

[EVE] *EventReporter*. http://www.eventreporter.com/en/. (Accessed 12th May 2003).

[Fra01] Przemyslaw Frasunek. *SQLSyslogd 0.1 – syslogd to MySQL wrapper*, 2001. http://www.frasunek.com/sources/security/sqlsyslogd/. (Accessed 11th May 2003).

[Har] Patrick S. Harper. *Snort Installation Manual: Snort 2.0.0, Apache 2.0.45, PHP 4.3.1, MySQL 4.0.12 and Acid 0.9.6b23*. http://www.internetsecurityguru.com/documents/snort_acid_rh9.pdf. (Accessed 11th August 2003).

[Kar01] Levent Karakas. *Unix Daemon Server Programming*, 2001. http://www.enderunix.org/docs/eng/daemon.php. (Accessed 11th May 2003).

[Lev00] LevelA.com. *Software Life Cycle Models*, 2000. http://www.levela.com/software_life_cycles_swdoc.htm. (Accessed 20th March 2003).

[Lon01] C. Lonvick. *The BSD syslog Protocol*. Network Working Group, August 2001. http://www.ietf.org/rfc/rfc3164.txt. (Accessed 11th July 2003).

[Mur98] James D. Murray. *Windows NT Event Logging*. O'Reilly, 1998.

[NTS02] *NTsyslog*, 2002. http://ntsyslog.sourceforge.net/. (Accessed 12th May 2003).

[Ope] OpenBSD. *OpenSSH*. http://www.openssh.com/. (Accessed 27th August 2003).

[PDE] *Postgresql for Debian*. /usr/share/doc/postgreSQL/README.Debian.

[Pos] PostgreSQL Global Development Group. *PostgreSQL*. http://www.postgresql.org/. (Accessed 27th August 2003).

[Pro02] The Honeynet Project. *Know Your Enemy*. Addison-Wesley, 2002.

[R.01] Eran R. *Syslogd+mysql - Patched FreeBSD syslogd*, 2001. http://freshmeat.net/projects/syslogdmysql/?topic_id=148. (Accessed 12th May 2003).

[Sch] Balazs Scheidler. *Syslog_ng*. BalaBit. http://www.balabit.com/products/syslog_ng/. (Accessed 27th August 2003).

[SNO] *README.database*. Snort: README.database.

[SSS] *Unix Programming Frequently Asked Questions*. http://www.erlenstar.demon.co.uk/unix/. (Accessed 12th May 2003).

[Sta00] Willian R. Stanek. *Microsoft Windows 2000*. Microsoft Publications, 2000.

[Tol00] Mark D. Tollison. *An Analysis of theSnort Network Intrusion Detection System*, December 2000. http://www.sans.org/rr/intrusion/snort2.php. (Accessed 15th June 2003).

[Voy99] Voytek. *Tubular Bells and Circular Named Pipes, or, how to syslog to a pipe ?*, 1999. http://w3.hethmon.com/os2isp/1999/Dec/Msgs/l2w93273.html. (Accessed 11th May 2003).

[Wad00] Thomas A. Wadlow. *The Process of Network Security*. Addison-Wesley, 2000.

[WD01] John Worsley and Joshua Drake. *Practical PostgreSQL*. O'Reilly, 2001.