Distributed Intrusion Detection Platform

Flexible, scalable and secure

Analytical Network Project
Masters programme on System and Network Administration
University of Amsterdam, The Netherlands

http://www.os3.nl/

Jeroen van Beek
jvb@os3.nl

Tjerk Nan
nan03@os3.nl

June 29, 2004

Abstract

SURFnet[1] is the national computer network for higher education and research in The Netherlands. It connects the networks of universities, colleges, research centers, academic hospitals and scientific libraries to one another and to other networks in Europe and the rest of the world.

Like other networks, the SURFnet network is exposed to attacks. The origin of the attacks performed on SURFnet connected networks can be found at local internal networks, networks of other SURFnet connected organizations and the Internet. SURFnet wants to gather representative data concerning malicious activities on the networks of connected parties. With the information extracted from the collected data, security advisories can be given to administrators of affected systems or networks.

To acquire the desired information, a number of sensors need to be placed in customer administrated networks. Since time and expertise desired for adequate "Intrusion Detection" might not be available at all sites, deployment and maintenance need to be easy. Local sensor maintenance should not be necessary or be kept to an absolute minimum. Therefore, sensor management and data storage should be centralized. This setup also enables central data analysis.

The goal of this project is to design a flexible, scalable and secure platform, which meets the requirement mentioned above. A proof of concept setup might be build to demonstrate functionality.

This document will discuss the design of a platform for Distributed Intrusion Detection and the proof of concept we have implemented.

Table of Contents

1 Introduction

In this chapter, the background, project goal and additional requirements are discussed.

1.1 Background

To monitor network activity an Intrusion Detection System might be used. An Intrusion Detection System listens to the network traffic, searching for malicious patterns that could indicate abuse. Large networks often need more than one Intrusion Detection System to watch multiple locations.

A Distributed Intrusion Detection System provides a central data storage for sensors placed on more than one location. The goal of this project is designing a platform for Distributed Intrusion Detection that focuses on flexibility, scalability and security. The solution should be component based. Changing a component should not affect other parts of the system, thus providing flexibility.

The necessary maintenance effort should not be affected by the number of sensors, which provides scalability. Adequate security should be implemented to guarantee data confidentiality and data integrity.

1.2 Project goal

The goal of this project is to design a flexible, scalable and secure platform for deployment of a Distributed Intrusion Detection System. A proof of concept setup will be build to demonstrate that the design realizes this goal.

1.3 Requirements

Since time and expertise desired for adequate intrusion detection might not be available at all sites, deployment should be simple. Local sensor maintenance should not be necessary or be kept to an absolute minimum.

1.4 Project focus

Due to the limited amount of time available for this project, priorities are defined.

Do

- Design a flexible and scalable platform for Distributed Intrusion Detection Systems, considering all elements and properties of such a system, which also meets additional security and acceptance requirements;
- A proof of concept implementation.

Don't

- Implementing new Intrusion Detection Systems;
- Implementing all technologies and solutions mentioned in this document.

2 Distributed Intrusion Detection System

This chapter will provide background information about Intrusion Detection Systems and discusses the concept of a Distributed Intrusion Detection System.

2.1 Intrusion Detection System

An Intrusion Detection System inspects all inbound and outbound network traffic. It matches traffic with known malicious patterns, which may indicate that viruses or hackers are active on the network.

An Intrusion Detection System consists of three components:

1. A sensor component, comparing network traffic with known malicious patterns;
2. A data storage component used for storing collected malicious patterns, usually a database;
3. A user interface for representation of the collected results, which we will call an "alert dashboard".

Most Intrusion Detection Systems integrate the three components in one physical device. However, it is possible to use an individual system for each component.

2.2 Distributed Intrusion Detection System

In a large and geographically divided network, multiple Intrusion Detection Systems might show trends in hacking and virus related malicious traffic. As a result, security advisories and warnings can be sent to involved parties. With the obtained information, security policies can be updated and quality of service can be improved.

Deploying multiple individual Intrusion Detection Systems will not be a desired solution. Every system will have its own data storage and information representation system. Therefore, information is fragmented. It is difficult to get a complete picture. This solution also requires a relatively large maintenance effort.

A Distributed Intrusion Detection Systems is a more desirable solution. A Distributed Intrusion Detection System consists of several individual sensors placed in representative local networks. Unlike a collection of conventional Intrusion Detection Systems, all sensor management, data storage and systems for result representation are located on a central location.

2.3 Elements of a Distributed Intrusion Detection System

A Distributed Intrusion Detection System shares the three components of a conventional Intrusion Detection System and adds a fourth: sensor management. The system usually consists of one or more sensor components. Sensor components can be placed in different networks.

Central site

- Sensor management software to enable and configure sensors;
- Data storage software to store data collected by one or more sensors;
- Alert dashboard software to query the data storage system;
- Hardware to run these three software components.

Sensor site

- Sensor software which runs on the sensor to collect suspicious traffic;
- Hardware to run this software.

Network

- To connect the sensors to the central site, a connection is needed.

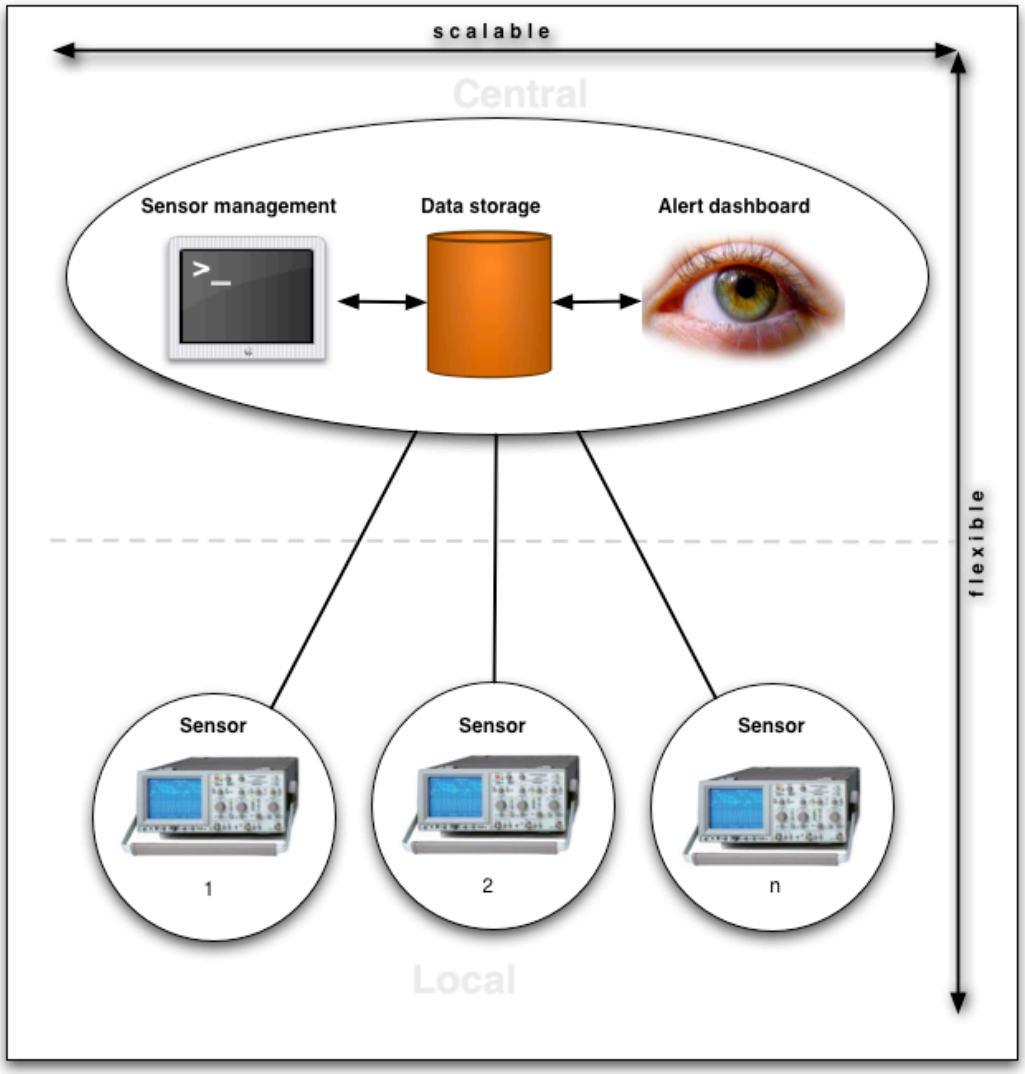

Figure 1: global system setup

3 Architecture

SURFnet would like to gather representative data concerning malicious activity on the networks of connected parties. Therefore, they need a Distributed Intrusion Detection System. This implicates basic elements and properties as described in paragraph 2.3.

Additionally, SURFnet indicates that required effort and expertise needed for local sensor maintenance must be limited as much as possible to increase acceptability. The ideal situation would be a sensor system without any local maintenance, except for initial deployment.

There are three major aspects in designing a platform for Distributed Intrusion Detection Systems: flexibility, scalability and security. These aspects are discussed in the following paragraphs.

3.1 Flexibility

The Distributed Intrusion Detection platform should be Intrusion Detection Software independent.

There should be a mechanism that makes it possible to update or replace applications, without having the need to redesign and change significant parts of the system: the system should be component based. There should not be any dependency between components.

A Distributed Intrusion Detection System like the eCSIRT system[2] does not meet this flexibility requirement. Changing to another Intrusion Detection System would mean that eCSIRT would have to design an entirely new system since the individual components share a number of important dependencies.

3.2 Scalability

Scalability means how well a system will function if its size increases. In this context, an increasing size means that the numbers of sensors grows. It should not make a difference if the system consists out of one, ten or a thousand sensors.

There are two main scalability aspects: performance and maintainability. Those two aspects need to be addressed because the platform for Distributed Intrusion Detection might be rolled out SURFnet wide and might consist out of many sensors. Good scalability will only be possible if the platform for Distributed Intrusion Detection is designed with this idea in mind.

3.2.1 Performance

The system should be able to handle an increasing number of sensors in terms of performance. There are two components of the system that will have an impact on the performance of the system as a whole.

- The data storage system: the system must be able to handle an adequate number of transactions;
- The connection between sensors and servers: it must be able to handle the amount of traffic that will be generated.

Unconsidered choices considering performance may result in a system that is not fully scalable.

3.2.2 Maintainability

Manual sensor configuration is not desirable, since necessary maintenance effort will rise with each sensor added. To accomplish scalability, all sensors should be the same. Though, all sensors do need a unique identification. This identity is used to relate events to a sensor and thus location.

3.3 Security

The information gathered by sensors might be confidential. Therefore, data confidentiality must be guaranteed for all components in the system.

4 Design

Chapter 3 discussed the desired architecture of a platform for Distributed Intrusion Detection. This chapter will discuss the technical design, using the architecture as a guideline.

4.1 Flexibility

Paragraph 3.1 discussed the need for an independent sensor environment to reach a high level of flexibility. There is a technical solution for this requirement: netbooting.

Netbooting

Most computer systems load the operating system from the local hard drive. With netbooting, the operating system of the system is stored on a central server in a so called netboot image. The system boots the operating system directly from the server, using the network.

This solution offers the desired flexibility, because there is no software installed on the sensor. If software should be replaced or updated, the central netboot image should be changed only.

Layer 2 network: single broadcast domain

A netboot environment would be the ideal solution although it has some additional requirements. It would require a single broadcast domain, provided by a layer 2 network. Using a layer 2 network has an advantage: it simplifies network design, see Figure 2

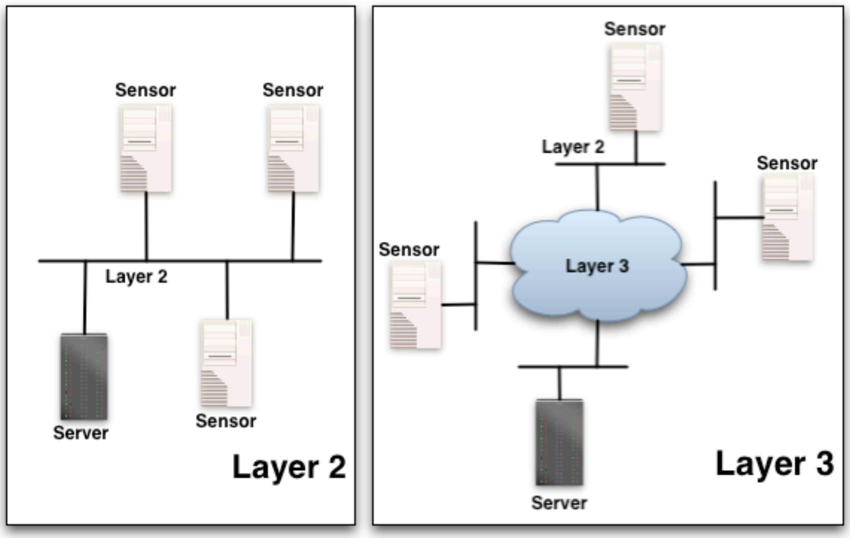

Figure 2: Layer 2 versus Layer 3

The only way to connect sensors to the central system is the Internet, which is a layer 3 network. Using technology to transport layer 2 network traffic over a layer 3 network link, solves this problem.

To create a virtual layer 2 network, tunnel technology is necessary. It is used to create a tunnel through the layer 3 network to the sensor sites.

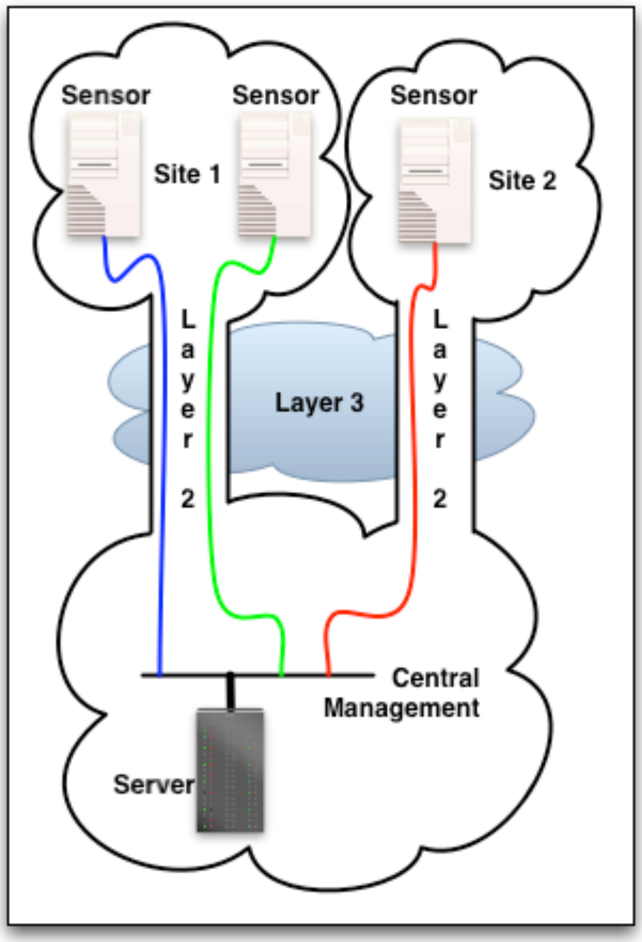

Figure 3: Layer 2 over layer 3

Figure 3 visualizes the concept of layer 2 tunneling over a layer 3 link.

Tunnel software is installed on separate devices: one tunnel device is necessary for each sensor site. Since the tunnel devices provides layer 2 connectivity, remote netbooting is possible for sensor devices behind a tunnel device, see Figure 4.

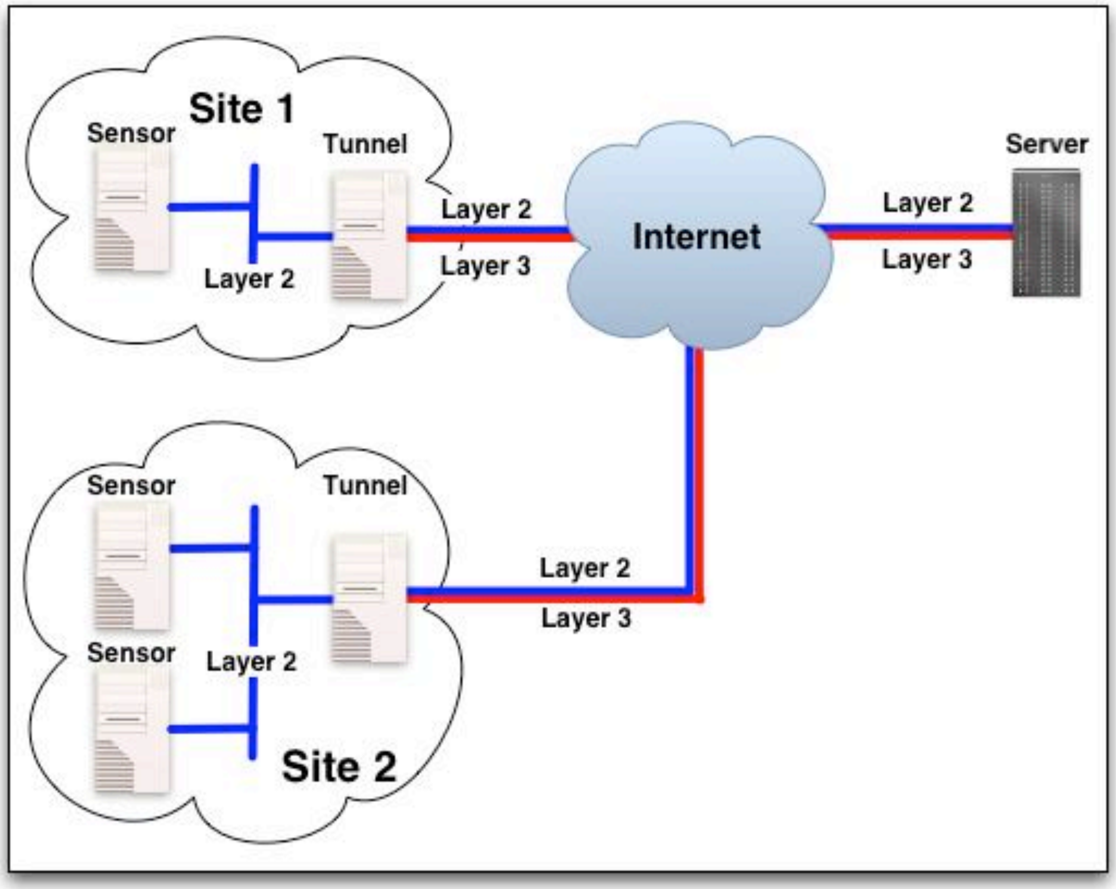

Figure 4: Layer 2 over layer 3 using tunnel devices

Even if just one sensor is required at a site, two machines must be installed: a tunnel device and a sensor. Generally, the number of machines per sensor required to deploy one sensor is (number of sensors + 1) / number of sensors. This creates a larger footprint than a system like eCSIRT: eCSIRT requires exactly one machine per sensor. The extra device in our setup provides an extreme high level of flexibility. Unlike eCSIRT, local maintenance is not needed at all after initial installation.

Central network share
The necessary applications should be run from a central location. This way software maintenance is centralized. Updating software at the central location affects all sensors: there is no need to manually update each and every individual sensor. The central storage location on a server should be accessible for all sensors. A network file system is needed.

4.2 Scalability

4.2.1 Performance

There are two main performance aspects: the network connection between sensors and the server and the central data storage system.

Network
The tunnel solution provides connectivity from the sensors to the central system. Theoretically, the tunneling solution is fully scalable as long as the maximal bandwidth of all remote tunnel devices accumulated does not exceed the available bandwidth of the central server interface. If it does and actual remotely generated traffic (nearly) equals the maximum remote bandwidth available, an additional local interface is needed. Note that most real life situations show that just a fraction of the available bandwidth is used.

Central data storage
Intrusion Detection Systems usually log their data to a database. With multiple sensors involved, the database system will have to endure a high load. The choice for a specific database system will often depend on the chosen Intrusion Detection System: some systems depend on a specific database engine. When deciding which database system to use, the aspect of performance should be taken into account.

4.2.2 Maintainability

The system should be able to maintain a large sensor network with a minimal management effort. Local maintenance of sensors should not be necessary, implicating central management. A central configuration change - like a software update - shall affect all sensors, without the need to reconfigure each individual sensor manually. Using this concept, an infinite number of sensors can be served.

Site management consists of the following tasks:

- Adding and removing sensors and tunnel devices;
- Configuring sensor settings;
- Sensor software management.

The layer 2 network plays a key role in the design of such a system. It enables the use of central services for automatic configuration of the sensors, like netbooting.

4.3 Security

Data confidentiality needs to be a design criterion. Extra attention is given to the following four points:

1. Physical access to the layer 2 network to which sensors and the tunnel device are connected;
2. Physical access to the sensor;
3. Access to the tunneled data, transmitted from the sensors to the central site;
4. Access to the alert dashboard.

The weakest link of the security chain is access to the layer 2 management network. If an attacker can connect to this network, the complete system might be compromised. To avoid this situation, the sensor needs to be located in a physically safe place. The tunneled data should be encrypted to preserve privacy. Access to the alert dashboard is restricted and transferred data is encrypted.

5 Implementation

5.1 The server

For the proof of concept setup a single server is used which provides the following services:

1. A central NFS share to give the sensors access to applications;
2. Central database storage system using MySQL[7] for storing alerts;
3. A DHCP service to configure the sensor' connections;
4. The alert dashboard software – see Appendix C – Alert dashboard - for querying alerts generated by sensors;
5. Sensor and tunnel device management tools implemented by using shell scripts.

Note on NFS
Although NFS poses a security threat, it is transported over the encrypted layer 2 network, which provides the necessary security. See also paragraph 5.2.

5.2 The network

OpenVPN[3] is used to provide tunnel functionality. OpenVPN features easy configuration and fast deployment. However, it is not necessary to use this specific application; all applications that support layer 2 tunneling can be used.

Endpoints of the tunnel are the tunnel devices that are placed on sensor locations and a server on the central site.

Securing the layer 2 link is accomplished by using the BlowFish algorithm with one secret key used on all individual tunnel devices. Besides this symmetric algorithm, it is also possible to use asymmetric cryptography.

5.3 The sensor

Bootable CD instead of netbooting
As described in chapter 4 netbooting is the desired solution for loading an operating system on a sensor. Because of the limited amount of time available for this project, a netboot implementation is not used. A conceptually similar solution is used: a bootable CD environment. The bootable CD is implemented using a customized KNOPPIX[4] distribution.

Using a bootable CD environment instead of a netboot environment introduces a dependency. Interaction of local administrators is needed if the operating system needs to be replaced. Since the system uses a layered concept, other components of the system are not affected by this change.

The startup sequence of a sensor:

1. Loading the operating system;
2. Initializing and configuring the network interfaces;
3. Mounting the central NFS share;
4. Executing the `start.sh` script on the central NFS share.

5.4 Sensor software

Services

SSHD

The sensors are running a secure shell service to enable remote administration. Using the secure shell, services or the sensor can be restarted.

NTP client

An NTP client is used on sensor devices to keep all internal clocks synchronized. Incorrect time settings render collected data useless since the events have to be linked with the moment of occurrence.

Applications

Intrusion Detection Systems

The NFS share provides two well-known Intrusion Detection Systems – Prelude[6] and SNORT[5] - to demonstrate that our solution is Intrusion Detection System independent.

Remote packet capturing

If necessary, it is possible to capture network traffic using Tethereal[10] and store results on an NFS share for further examination. The implementation of tethereal proves that the sensor platform is flexible: it can run any kind of software.

5.5 Sensor management tools

For the proof of concept, sensor management tools were developed. The tools consist of shell scripts, which provide the following functionality:

- Adding and removing sensors;
- Adding and removing tunnel devices;
- Restarting sensors.

See Appendix B – Sensor management for more information.

6 Conclusion

6.1 Results

During the four weeks of the project, we were able to realize the goals discussed in this document. We successfully designed a flexible, scalable and secure platform for Distributed Network Intrusion Detection. We also managed to implement a proof of concept, which demonstrates that our design actually is flexible, scalable and secure.

Flexibility
Adding, changing and removing sensor software affects only a single component and not the entire system. Our proof of concept can run any Intrusion Detection System with no changes to the platform, proving flexibility. The platform is component based. Each component can be changed or replaced without affecting other components.

Scalability
As long as the traffic of all sensors accumulated does not exceed the available bandwidth available to the central system, the design will be fully scalable. An increasing number of sensors do not have an impact on sensor maintenance activities. This goal is achieved by centralizing sensor management.

Security
The proof of concept implements the security aspects mentioned in our design. Avoiding physical access to the layer 2 network is accomplished by placing the server and tunnel devices in a secure environment. Confidentiality of the tunneled data is guaranteed by using cryptographic technology. User authentication restricts access to the alert dashboard. Data transmitted from the alert dashboard to an authenticated user is encrypted to prevent eavesdropping.

Additional requirements
SURFnet required that local sensor maintenance should be limited as much as possible. We managed to design a platform for Distributed Intrusion Detection that does not require any local sensor maintenance at all. The proof of concept implementation requires a minimal local maintenance effort only if the sensor operating system should be updated or replaced.

We conclude that our design and implementation of a Distributed Intrusion Detection platform provides a solid base for deploying Distributed Intrusion Detection Systems on a large scale.

6.2 Future work

Due to the limited time available for this project, some functionality is not implemented, though a real life system do needs it:

Server environment
All server services - sensor management, storage software and alert dashboard - are installed on one physical machine. For scalability issues a machine per component is desirable.

Netboot environment
To reach the highest possible level of independency for the sensor machines, a netboot environment must replace the bootable compact disc environment. Also mention that the KNOPPIX image contains more applications than strictly necessary and booting this image will result in a local root shell without logging in.

Sensor management error handling
At this time, sensor management does not include adequate error handling. Faulty entered parameters might cause unforeseen consequences. For a production system, proper error handling needs to be added.

Database abstraction layer
Some research on database abstraction layers might be useful. If a database independent interface – like unixodbc - can be configured to cooperate with SQL-picky programs like Prelude, portability of the system to different database systems can be ensured.

Tweaking NFS
The NFS server is running on a Linux host using default settings. Tweaking the NFS server service or even migrate the services to another platform can increase performance.

Graphical User Interface for sensor and tunnel management
In the proof of concept setup, sensor management is console based. A graphical variant – like a web interface - might increase the ease of use.

6.3 Personal notes

We would like to end this document with a personal note. Working on this project has been a pleasurable experience. Most challenging part of the project was finding a mechanism for minimizing local maintenance effort. Hopefully, our proof of concept setup is just the beginning, not the end, of a large scale Intrusion Detection System. While we believe that we have designed and created something useful in the four weeks of the project, we feel that many interesting topics relevant to this project could be researched and developed, given more time.

Appendix A – Customizing a KNOPPIX CD image

A customized KNOPPIX image is created using the "Knoppix Remastering Howto"[8]. To create an image that is small enough to fit on a recordable CD, most X-Windows related packages are removed. Other modifications that are not described in the referred how-to:

- Changing runlevel from graphical to console in `/etc/inittab`:
  `id:2:initdefault`

- Adding a startup script – `S90ids` – to `/etc/rc2.d`:

  ```
      /bin/sleep 10
      /sbin/portmap
      /bin/mount <server>:/<share><mountpoint>
      <mountpoint><folder>/<global script>
  ```

  `sleep` is used to wait for the DHCP-client that is running in the background (KNOPPIX default). This sleep command can be removed if the DHCP-client is not running in the background. Since this project was not about customizing KNOPPIX, this quick and dirty work-around is used.

  `portmap` is needed to mount an NFS share.

  `mount` mounts the NFS share.

  `<global script>` starts software, which is stored on the central share.

- Enable remote rebooting of a sensor: disable ejecting the tray of the CD drive in `/etc/init.d/knoppisx-reboot`: Change `NOEJECT=""` to `NOEJECT="yes".`

Appendix B – Sensor management

Almost all software running on the sensor is ran from a central share. This share has following directory tree layout:

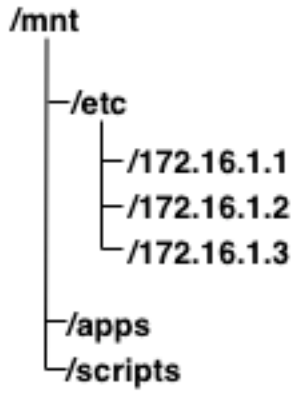

Figure 5: Directory tree layout

etc
The etc directory contains the specific configuration parameters for the sensors like sensor names. These settings are stored in a directory – one for each sensor - with the sensors' IP-address as its name.

apps
The apps directory contains all applications that can be run on the sensor. The proof of concept uses NTPD for time synchronization with the server, SSHD for remote sensor administration, snort for storing Snort binaries and libraries, and prelude for storing Prelude related software. It also contains tethereal for capturing network traffic.

scripts
The scripts directory contains the following scripts:

- `start.sh;`
- `start-prelude.sh;`
- `start-snort.sh.`

`start.sh` is executed after the sensor mounted the central share. First it identifies the sensor by its IP-address. Next, it starts the necessary services and finally, it runs the application specific scripts. Arguments used in these scripts:

- `NFS_SERVER` (IP-address);
- `SENSOR_IP` (IP-address);
- `INTERNAL_INTERFACE` (like `eth0`);
- `EXTERNAL_INTERFACE` (like `eth1`).

Arguments are all hardcoded except for `SENSOR_IP`, which is determined in runtime.

`start-<ids>.sh`
All Intrusion Detection System software is started by its individual script. For example, the proof of concept setup runs Snort and Prelude. This implicates that there are two scripts that are executed by the `start.sh` script: `start-prelude.sh` and `start-snort.sh`. These scripts are called with the arguments discussed above. New sensor functionality can be added by following these steps:

- Compile needed software, for example `a.out`;
- Check dependencies by executing `ldd  a.out`. Copy possibly found libraries to `<a.out-path>\lib`;
- Copy the binaries - and if needed libraries - to the central share;
- Create a shell script which can be used to start the application;
- Add the line export `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<a.out-path>\lib` if libraries need to be added to the library path;
- Call the new script from the global script – `start.sh` – with the appropriate arguments;
- if needed, create a shell script to create application specific configuration files in the sensor configuration directory – `etc/<ip-address>/<a.out>` - and call this shell script from `register-sensor.sh`.

Appendix C – Alert dashboard

The alert dashboard uses the following software:

- Analysis Console for Intrusion Databases[9], a PHP front-end for Snort;
- PIWI – see [6] for more information – a Perl frond-end for Prelude;
- Shell scripts for centrally started manual packet capture sessions using tethereal[10] – in remote sensor networks.

Appendix D – Setting up a new sensor site

This chapter provides a step-by-step guide for setting up a new sensor location. Keep in mind that:

- A sensor machine has two Ethernet interfaces; one with a public IP-address for sniffing purposes, the other one with a private IP-address for management purposes;

- Each sensor location needs one tunnel device which must be up and running before a sensor is installed;

- Each sensor site can host one or more sensors.

Setting up a tunnel machine

Preparing a device as a tunnel machine

- Install an operating system supported by OpenVPN;
- Install OpenVPN.

Registering a tunnel:

```
./register-tunnel.sh  <tunnel-ip> <tunnel-name> <port>
```

- `<tunnel-ip>` the public IP-address of the tunnel machine;
- `<tunnel-name>` the freely chosen unique name for this tunnel machine, let's say the location plus institute;
- `<port>` each OpenVPN connection needs a dedicated UDP port.

Starting a tunnel:

```
./start-vpn <tunnel-name>
```

- `<tunnel-name>` The name of the tunnel machine;
- If `all` is used for <name>, all existing tunnels will be disconnected, followed by starting all registered tunnels.

Registering a sensor:

```
./register-sensor.sh <sensor-ip> <sensor-mac> <sensor-name> <id>
```

- `<sensor-ip>` the private IP-address of the sensor machine;
- `<sensor-mac>` the MAC-address of the interface to which to <sensor-ip> is bound;
- `<sensor-name>` freely chosen name used as hostname in the DHCP configuration file and as the sensor name used by the Snort Intrusion Detection System;
- `<id>` unique ID used by the Prelude Intrusion Detection System. This ID must me numeric.

Setting up a sensor machine

For the first sensor on a location only:

- Register a tunnel machine;
- Activate a tunnel to this tunnel machine.

For all sensors:

- Make sure the BIOS booting order is set to CD first;
- Place a sensor CD in a sensor PC and boot.

Maintenance

If sensor application changes, changes can be applied to the central share. After successful testing, the global sensor start script must be updated. The new settings can be applied by rebooting all sensors.

Rebooting sensors:

```
./reboot-sensor <sensor-ip>
```

- `<sensor-ip>` Use `all` to reboot all sensors. To reboot a single sensor - perhaps for testing new software - a specific sensor can be selected by entering its private IP-address.

Remove sensor machine:
If a sensor needs to be disabled, it can be removed from the distributed system.

```
./remove-sensor <sensor-ip>
```

- `<sensor-ip>` the private IP-address of the sensor.

If all sensors on a site are disabled and the location itself will not be used any longer for distributed Intrusion Detection System activities, the tunnel machine can be removed from the distributed system.

Removing a tunnel machine:

./remove-tunnel <tunnel-name>

- `<tunnel-name>` is the name of the tunnel machine entered when the tunnel machine was registered.

Appendix E – Deliverables

A short list of files and documents that are part of the project:

- This document, containing background information about the project;
- An ISO image containing the bootable KNOPPIX sensor CD;
- Software package with management and start-up scripts.

Appendix F – List of figures

Appendix G – References

[1] “Webpage of SURFnet,” http://www.surfnet.nl/

[2] “The European CSIRT Network,” http://www.ecsirt.org/

[3] “OpenVPN Home Page,” http://openvpn.sourceforge.net/

[4] “KNOPPIX Home Page.” http://www.knoppix.org/

[5] “Snort Home Page,” http://www.snort.org/

[6] “Prelude Home Page,” http://www.prelude-ids.org/

[7] “MySQL Home Page," http://www.mysql.com/

[8] “Knoppix Remastering Howto,” by knoppix.net,
http://www.knoppix.net/docs/index.php/KnoppixRemasteringHowto

[9] “ACID Home Page,” http://acidlab.sourceforge.net/

[10] “Tethereal A Network Protocol Analyzer,” http://www.ethereal.com/