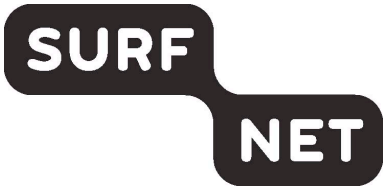
Implementing Snort into SURFids

ing. Sander Keemink
ing. Michael van Kleij
February 4, 2008

UNIVERSITEIT VAN AMSTERDAM
Masteropleiding System and Network Engineering



Abstract

SURFnet is *the* scientific and academic internet service provider in the Netherlands. Customers of SURFnet are schools, universities and research facilities.

SURFnet's SURFids is a unique Intrusion Detection System (IDS). It differs from other Intrusion Detection Systems in the way it is set up. In SURFids all detection functionalities are centralized. The sensors only send network traffic to the centralized processing servers. At the moment this setup consists of Argos and Nepenthes. SURFnet wishes to further improve this system. Snort might be a valuable addition to this system.

The primary goal of implementing Snort into SURFids is to be able to recognize known attacks from the attacks reported by Argos. Argos in itself isn't able to recognize known attacks. Because of this SURFnet doesn't know whether Argos reported a new attack, or an existing one.

> *Which implementation of Snort into SURFids gives the most added value to the customer while not degrading performance in a noticeable way.*

Quote 1: Research Question

In this research we tried to answer the question of how to implement Snort in the best way. Quote 1 shows the research question. Subquestions of this were:

- *Where in the SURFids setup should Snort be placed to add the most value to the system?*
- *How do we get the data from Snort into the SURFids database?*

During the research it became more and more apparent that Snort could be a system on its own, whereas the initial idea was to combine Snort with Argos. Snort detects some attacks that aren't detected by both Argos and Nepenthes and for that reason might be a valuable addition on its own. Snort does need a honeypot behind it to be able to detect attacks based on bidirectional traffic.

Snort will supply the system with a lot of information. There are various ways to insert this information in the SURFids database. Depending on the time available we advise the following:

- Develop or alter the Barnyard database plugin.
- Use the Barnyard csv output plugin and develop a program/script to parse these values and put them in a database.

In a situation with little time we suggest the second advice. Otherwise the first advice is better because it has one step less.

Acknowledgements

We would like to offer our gratitude to the following people and organizations:

- *SURFnet*: for providing us with the means and the opportunity to conduct this research project.
- *All the people of the SURFids group*: For offering help and support.
- *All those who gave their feedback on this document*

Contents

List of Figures

Introduction

This document describes the project: "Implementing Snort into SURFids" executed by Sander Keemink and Michael van Kleij on behalf of SURFnet. The project description can be found in quote 2. In our project plan we defined our own research goal: *Which implementation of Snort into SURFids gives the most added value to the customer while not degrading performance in a noticable way.* To be able to answer this question we have conducted two experiments. These experiments can be found in chapter 4. Before this however, we explain a bit about SURFnet (chapter 1), Intrusion Detection Systems (chapter 2) and Snort (chapter 3) first.

> SURFids is an open source project which is created by SURFnet with help of students of the SNE master. At this moment the project is supported by organisations in the US, Germany, Sweden, Norway, Japan and Australia. The current version of SURFids is in fact a distributed Honeypot solution. SURFnet offers this system as a service to her customers. At this moment about 50 sensors are active within the system. SURFnet adds functionality to the SURFids system on a regular basis. We want to find out how Snort can be integrated into the system

Quote 2: Project description

In the experiments we discuss two options. One in which the Snort is placed before Argos, and one in which Snort is placed before both Nepenthes and Argos. Based on the results of these experiments we will give an advice about which of the two options is the best. The primary reason to determine which of the two solutions is best is based on the amount of added information to SURFids. All this can be read in the chapters 4 and 5. A conclusion will be given in the end. Another chapter, chapter 6, will cover the possibilities of inserting the Snort data into the SURFids database.

Chapter 1

About SURFnet

SURFnet is *the* scientific and academic internet provider in the Netherlands. SURFnet aims to provide a high speed, high availability network for its customers (Universities, Research facilities etc.) SURFnets mission can be read in quote 3.

> It is SURFnets mission to facilitate groundbreaking education and research through innovative network services.
>
> SURFnet combines the demand of the institutions connected to SURFnet. In doing so we create advantages of scale, innovation and collaboration from which they benefit.
>
> The SURFnet network services comprise five focus areas: Network infrastructure, Security, Authentication & authorisation, Group communication and Content Delivery.[8]

Quote 3: SURFnets mission

Innovation of the current SURFnet services, and the development of new services is a key element to realize this mission. In pursuit of these goals services like SURFids are continuously improved.

Chapter 2

SURFids

To know what SURFids is, one must first have an understanding of what an IDS is. In this chapter an IDS is explained before SURFids is covered.

2.1 What is an IDS?

An IDS is an Intrusion Detection System. It is, as the name implies a system to detect intrusions. It is *not* a system to prevent intrusions. To prevent malicious traffic one needs to take a look at an IPS[1]. An IPS is sometimes called an Active Intrusion Detection System where the other systems are called a Passive Intrusion Detection Systems.

An IDS is able to analyze network traffic it receives to see whether it contains a possible attack on a host in the network. When an IDS detects such traffic it will only report this traffic. The IDS won't interfere with the traffic itself, so potentially dangerous traffic will arrive at the intended host.

There are various ways to implement Intrusion Detection Systems. The most common Intrusion Detection Systems systems are Network Intrusion Detection Systems and Host-based Intrusion Detection Systems. The difference between the two is that a Network IDS looks at *all* the network traffic, including traffic which isn't destined for that host while a Host-based IDS only looks at attacks destined for that system. In this research the focus is only on the Network based IDS Snort. This research aims to find a way to implement Snort in the existing SURFids system. More information about Snort can be found in chapter 3.

2.2 What is SURFids?

SURFids is the IDS implementation of SURFnet. It is unique in many ways. Like many IDS systems SURFids consists of sensors. Unlike many IDS systems these sensors don't do anything with the traffic except forwarding this through a VPN tunnel. These sensors are installed at the customers site and they make a tunnel to a central tunnel server. All traffic is processed by two honeypot systems[2]. The used honeypot systems are Nepenthes and Argos. On the tunnel

[1] IPS: Intrusion Prevention System

[2] These are Host-based Intrusion Detection Systems

server a decision is made as to which honeypot the traffic will be sent. The default honeypot system is Nepenthes.

This setup is unique in the way that the sensors are "maintenance free", all intelligence and maintenance are stored on the tunnel server and honeypot systems. Adding sensors is as easy as pushing an USB-stick into a computer. A schematic view of SURFids can be seen in figure 2.1. In this schematic view Nepenthes is shown as a separate system, but at this moment Nepenthes still runs on the tunnel server itself.

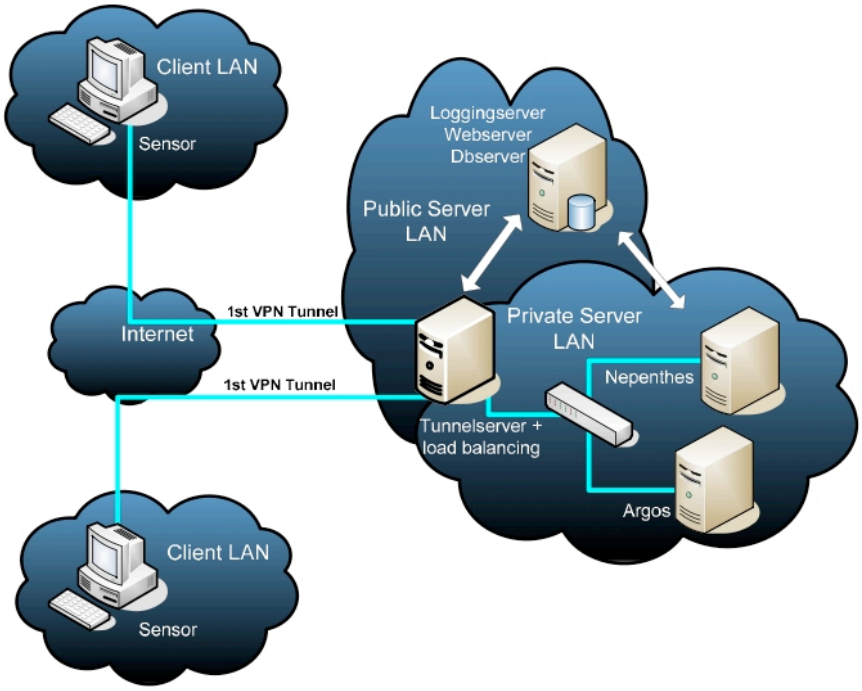


Figure 2.1: SURFids

2.2.1 Nepenthes

Nepenthes is one of the honeypot programs used by SURFids. It is a low interaction honeypot, which means it simulates vulnerabilities instead of running the service that has the vulnerability [11]. A low interaction honeypot can only detect *known* attacks. It is modular of setup which means it is easy to expand.

Nepenthes registers every connection on ports on which it emulates an exploit. All connections on these ports are possible malicious attacks until an exploit is, in fact, executed.

2.2.2 Argos

Argos is a high interaction honeypot which means that it is running a full operating system. All actions which are a response to network traffic are monitored

for potential illegal activity. In Argos network packets are marked as potential threats. The data from these packets is always marked and when the system tries to execute this data on an illegal way –like using it as a jump target– an interrupt is thrown and Argos shellcode is inserted to analyze the attack. This method does not allow Argos to recognize known attacks. Every attack is new for Argos. More information about Argos can be found on the Argos website [2].

Chapter 3

Snort

Snort is a Network Intrusion Detection System (NIDS) it describes itself as follows [15]:

> "Snort™is an open source network intrusion prevention and detection system utilising a rule-driven language, which combines the benefits of signature, protocol and anomaly based inspection methods."

Snort [18] is based on libpcap [9] and uses its functionality to capture all network packets on an interface, even those destined for other hosts. Snort uses a rule-driven language to detect malicious packets and gives information on detected malicious packets. With this rule language Snort is able to detect attack types like: buffer overflows, port scans, ddos attacks and many more.

Snort is being developed by Sourcefire [19], a commercial company founded by Martin Roesch, the creator of Snort. Sourcefire is not only active in the development of Snort but also in the creation of Snort rules.

3.1 Snort modes

Snort offers four modes in which it can operate [14]. These modes determine the functionality Snort provides.

- *Sniffer mode*, displays all IP packets in a continuous mode on screen.
- *Packet Logger mode*, logs all network packets to disk.
- *Network Intrusion Detection System mode*, analyzes network traffic by matching it against rules and outputs alert or log information based on those rules.
- *Inline mode*, obtains packets from iptables instead of libpcap and then use inline-specific rule types to help iptables pass or drop packets.

For this project we will be working with the network intrusion detection mode. This is the mode that is able to analyze traffic intended for other hosts, in our case Argos and Nepenthes.

3.2 Snort Components

Snort can be logically divided into components [1] [10]. These components work together to decode and analyze each IP packet and output information based on the results. Snort consists of the following components:

- Packet Decoder
- Preprocessor
- Logging and Alerting System
- Output Modules

Figure 3.1 shows how these components operate. A packet enters Snort at the packet decoder and passes through the other components where it will either generate an alert or gets dropped. When Snort drops a packet it means Snort hasn't found a match and moves on to the next packet.

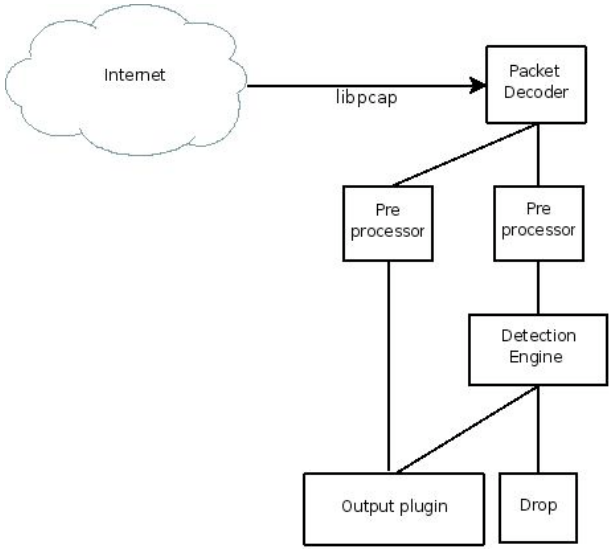


Figure 3.1: Schematic overview of the Snort components and data flow

3.2.1 Packet Decoder

The packet decoder decodes the specific protocol elements of a captured packet. It starts at the link layer protocol and moves up to the network layer protocol and finally the transport layer protocol. From this it stores information like ports and addresses. Snort will generate an alert if a malformed header, unusual length TCP options or other anomalies are detected.

3.2.2 Preprocessor

Snort knows two types of preprocessors, those that examine packets for suspicious activity and those that modify packets so that the detection engine can

properly interpret them. Not all attacks can be detected by rule matching using the detection engine so preprocessors have been setup to detect suspicious activity. A packet modification example is when an attacker wants to elude the NIDS with the use of IP fragmentation [17]. Another example with the use of IP fragmentation is a Denial of Service attack [10]. The *frag3* preprocessor can be used to detect ddos attacks but also to reassemble fragmentated packets so the detection engine can match them against rules.

3.2.3 The Detection Engine

The detection engine reads Snort rule files line by line and loads them into an internal data structure. Rules are split into two functional sections; the rule header, see example 1, and the rule option, see example 2.

Example 1 A rule header for port 135 based traffic

```
alert tcp any any -> any 135
```

To match an IP packet to a rule the detection engine follows its internal data structure until a match is made or the packet is dropped in the case of no match. First a packet will be referenced against the rule header. What protocol is used? TCP, go to the TCP structure. What destination port? Port 135, go to the subset of TCP for destination port 135. This structure is followed until a match is made against a rule option or in the case of no match the packet will be dropped.

Example 2 A rule option for a DCOM exploit

```
(msg:"DCOM Exploit (MS03-026) targeting Windows2000 SP0";
content:"|74 16 e8 77 cc e0 fd 7f cc e0 fd 7f|";
classtype:attempted-admin;
sid:1100001;
reference:URL,
www.microsoft.com/security/security_bulletins/ms03-026.asp;
reference:URL,
jackhammer.org/rules/1100001;
rev:1;)
```

The rule option contains the message that is displayed in logs and alerts, the payload against which will be matched, some reference data and additional options to specify where to look for the payload. Detection plugins can be used for special payload matching operations in data structures for example, to get a value from ASN.1 coding or a Remote Procedure Call (RPC).

3.2.4 Output Plugins

The output plugins provide the means to get the Snort output to the user. Snort has a modular setup and provides the means to use multiple output plugins.

For more information on the output plugins and it's reference to SURFids see chapter 6.

3.3 Snort rules

The value of Snort as an NIDS depends on the rules used. The rules determine what you can and can not detect. There are several different resources available from where one can obtain those rules:

- *Subscribers*, for an annual fee of $499 per sensor, rules become available before they are released to the public when the Sourcefire Vulnerability Research Team has finished them.
- *Registered*, a registered user can obtain the commercial Sourcefire rules 30 days after they are released to the subscribed users. Registration is free.
- *Unregistered*, the rules that come with each major Snort release.
- *Bleeding Edge Threats* [7], a community that provides open source rules.

Snort rules are created by the Sourcefire Vulnerability Research Team (VRT) [21]. Sourcefire VRT reacts on vulnerabilities by analyzing these and aim at making rules that match against all possible attacks that use this vulnerability. Rules are created on the vulnerability, not on a known attack so that one rule matches several attacks.

Bleeding Edge rules are not tested as thoroughly as the Sourcefire rules. Before the rules are released they are checked if the syntax is correct and they don't crash Snort. Bleeding Edge rules are more focussed on a known attack and respond faster to new attacks. Over time they try to make the rules more general and vulnerability specific instead of attack specific. Some of these rules make it into the official Snort rule set.

Automatic rule management can be done by Oinkmaster [12]. Oinkmaster is a BSD licenced perl script to update Snort rules. Sourcefire provides means for subscribers and registered users to use Oinkmaster to obtain their rules.

3.4 Snort performance

Snort performance in Mbits isn't stated very clearly. There are reports of throughput of 100Mbit without package loss. Recent research [13] on improving performance for libpcap based application achieved a throughput of 125Mbit without packet loss. This was done with a standard Snort configuration with all preprocessors and rules and the original libpcap.

Snort performance can be improved by using a modified version of libpcap that implements a shared memory ring buffer [26]. This implementation of libpcap improves Snorts performance by limiting the number of times an IP-packet is copied before Snort can perform its operations on it. For more information on this see the Snort manual [14]. Unfortunately no information is available on how much performance gain can be achieved.

Offloading Snort by transferring the logging to another application is another way to improve performance [4]. Especially when one wants to log all

information to a database. Snort has to contact the database, perform an operation and wait for an acknowledgement which can be time consuming on a busy network. Snort offers the unified output plugin which outputs all information in a unified binary format, which decouples the output stage from Snort. This method is used by Barnyard [3] a program which sole purpose is to read the Snort unified binary format and output it to another format like a database.

3.4.1 Barnyard

Barnyard reads unified binary format and writes it to a resource. The advantage of using Barnyard is evident in situations when one is using a database. When the database server becomes temporarily unavailable Barnyard waits until it is up again and continues where it stopped. Snort doesn't provide this service if the database is unavailable Snort output is lost and a restart is required once the database is available again.

Chapter 4

Experiments

During this research project three experiments were defined of which two were executed. The second and third experiment were considered logically the same, so only one of these experiments has been executed due to time and hardware limitations. The experiments that were executed are experiments 1 (see [22]) and 3 (see [23]). In this chapter the experiments are explained. For a detailed experiment description please read the experiment setup documents.

4.1 Experiment 1: Snort before Argos

In the first experiment the Snort machine was placed before Argos. This setup was chosen to determine if Snort has an added value to Argos. In this setup Snort is invisible to the outside world but Snort is able to analyse all traffic intended for Argos.

To implement this setup a separate server was used. This server was configured to act as a bridge. Snort was configured to listen on the bridge interface and analyze all data. Furthermore Snort was configured to log all data as unified (binary) files and Barnyard was used to write these logs to a database.

In figure 4.1 a schematic view of the experiment setup is shown. We expected to run this experiment for 1 day, but in the end this experiment ran for 7 days. The following results were expected from this experiment:

- Degrade performance with a small amount, adding a few ms RTT per packet.
- Able to cope with the load, now and in the future.
- Difficult to manage and extend.

It was immediately assumed that this setup would function with Snort and that Snort would be able to recognize most, if not all attacks reported by Argos.

The results of this experiment can be found in section 5.

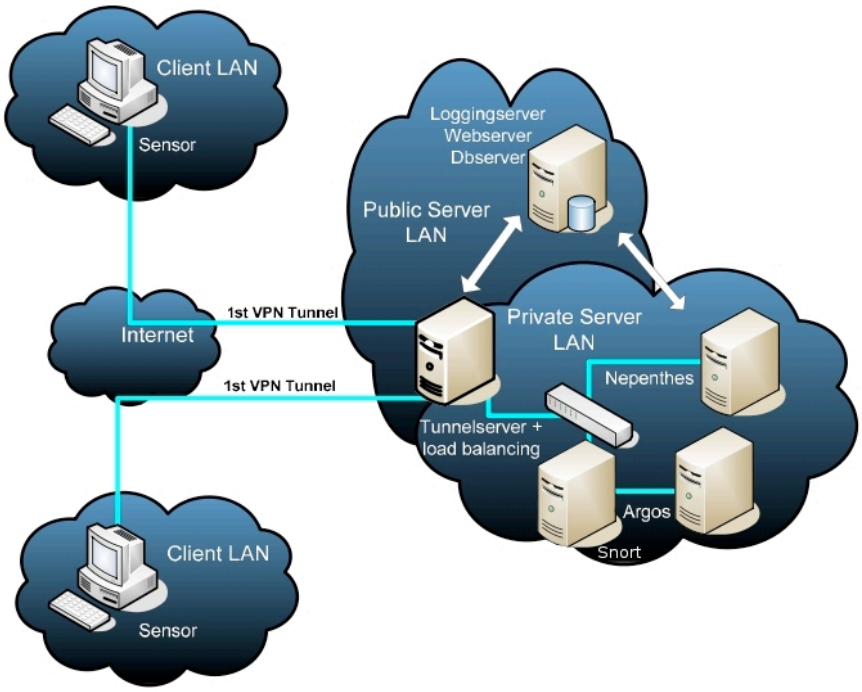


Figure 4.1: Logical view of experiment 1 setup

4.2 Experiment 2: Snort next to Argos and Nepenthes

In experiment 2 we placed Snort on a separate server and connected this to the switch on which Argos and Nepenthes were also connected. See figure 4.2. With this setup Snort is able to analyse all traffic intended for both Argos and Nepenthes without impact on the current logical setup. This also provides the least performance impact because all systems run independently of each other.

It is important in this setup that the switchports on which Nepenthes and Argos are connected are mirrored to the port on which Snort is connected.

For this setup to work Nepenthes has to be installed on a separate server, and not as is currently the case on the tunnel server.

This experiment hasn't been conducted because of time and hardware limitations. The expected results are the same as experiment 3, which could be conducted without changing the complete setup of SURFids.

4.3 Experiment 3: Snort on the tunnel server

In experiment 3 we placed Snort on the tunnel server. See figure4.3. We choose this setup because Snort is able to analyse all traffic for both Argos and Nepenthes and doesn't require any change of the physical setup.

Because of time and hardware limitations experiment 2 wasn't executed and

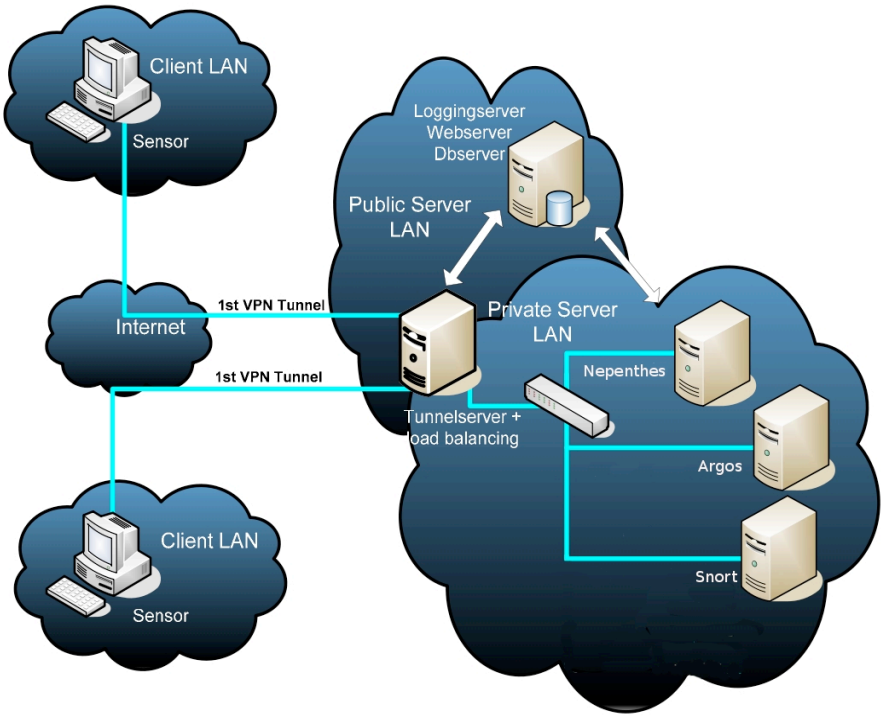


Figure 4.2: Logical view of experiment 2 setup

instead experiment 3 was conducted. Because of the fact that experiment 3 is logically the same[1] as experiment 2 this should not matter.

We expected that this experiment setup would give different performance results from experiment 2. This because Snort is in the path of the data, where Snort would be outside the datapath in the case of experiment 2. In that case all the data would be mirrored to the Snort server. Because of this we expected that experiment 3 would be somewhat slower than experiment 2, but equally as fast as experiment 1.

[1] With logically the same we mean that the detected attacks should be exactly the same

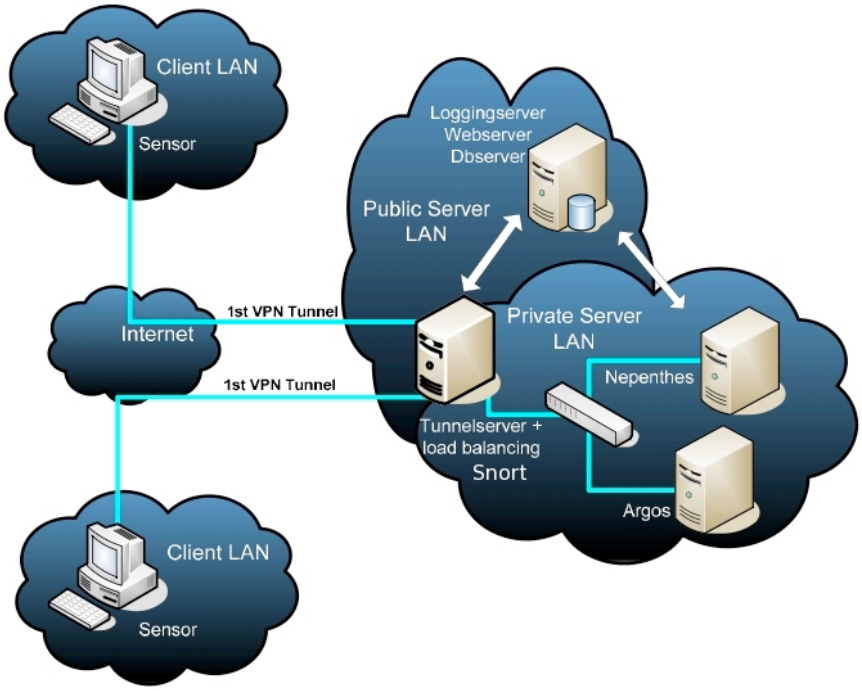


Figure 4.3: Logical view of experiment 3 setup

Chapter 5

Results

The experiments produced various results as well as various problems. Some results were curious and require more attention in future work. During the experiments we created specific experiment reports which contain all information about the experiment results. These results were documented in the appendixes to try to keep this report from becoming 'bloated' with unnecessary information. Because of this the sections covering the results of the experiments will only contain the most important information. Interested readers are referred to [24] and [25].

5.1 Experiment 1

During experiment 1 some problems were encountered. The database wasn't configured correctly and the rulesets used were incomplete. When the database was configured correctly and the rulesets were added the Snort setup functioned perfectly. This can be seen in table 5.1. As can be seen in the table the problems with the rulesets were solved on the 14^{th} of January.

Date	Argos	Snort	Overlap	Perct.
12-01-2008	20	3	2	10%
13-01-2008	20	10	5	25%
14-01-2008	32	97	20	62,5%
15-01-2008	36	178	34	94%
16-01-2008	26	139	25	96%

Table 5.1: Results from Snort and Argos

5.1.1 Points of interest

During the experiment some interesting issues were noticed:

- Time skew.
- Multiple entries per attack.
- Missing payload data.

- Port difference between Snort and Argos.
- Attacks with source IP 192.168.25.75 (The Argos machine).

These issues will be discussed in more detail with some ideas about how they could be solved.

Time skew

When looking at the logs from both Snort and Argos one can see a difference in the reported time. An example of this is in log 1. The log shows that the Argos machine logs an entry at 15:42 and the Snort machine logs the same entry on 15:51. The solution to prevent these timeskews is to make sure all systems have the same time. The Network Time Protocol is ideal for this. As long as all servers use the same, trustworthy, timeserver on a regular basis –say about every hour or every day– the timeskew will not become to big.

Log 1 Time skew between Argos and Snort

```
A: 12-01-2008 15:42:07  Malicious attack - Argos  212.147.79.166
65225  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-12 15:51:29.
663 212.147.79.166:65225     192.168.8.129:445   TCP
```

Multiple entries per attack

Snort often registers multiple entries per detected attack. According to us [24] this is due to the fact that many attacks consist of multiple network packets. Snort doesn't keep track of all packets and therefore it happens that attacks that are sent multiple times are registered as separate attacks. The Argos machine, however, does not register all these attacks as individual attacks. This creates a problem because it's undesirable to register any given attack more than once.

Another reason for multiple entries is that it's possible that multiple rules trigger on a given attack. This results in entries that differ from each other in the name of the attack. This complicates the possibility to compare entries to each other because the name might differ. However, even though the name might differ the source IP, source port, destination IP and destination port are the same. The double entries are always close to each other in time, so when there are multiple attacks from the same IP and port to the same IP and port within a few seconds one can conclude this is the same attack.

Another way to make sure whether an attack is registered multiple times is to compare the Argos log with the Snort log. When Argos reports only one attack at a given time and Snort sees various attacks from that IP and port one could conclude that only one attack has happened.

Missing payload data

When using Barnyard with the default configuration instead of letting Snort write directly in the database the payload data of packets is lost. This might

not be a problem for SURFnet when it doesn't need the payload data. However, when SURFnet does want the payload data it's necessary to enable logging as wel as alerts. When this is done Barnyard will extract the payload data from the logfiles and insert them into the database.

Port difference between Argos and Snort

On January 17 2008 an attack was logged on both Snort and Argos, but with different ports. On Argos this attack was logged with port 6129 and on Snort this attack was registered on port 445. This is shown in log 2. As is shown in this log the time is correct (mind the timeskew). Furthermore the attack originates from the same IP. In Argos there is no log of an attack around this time from that IP on port 445. It is possible that there were multiple packets of which Argos logged the one on port 6129 and Snort the one on port 445. Unfortunately we don't have a tcpdump of that time.

Log 2 Port difference between Argos and Snort

```
A: 16-01-2008 07:44:42 Malicious attack
- Argos 89.139.111.141 3421
192.168.8.129 6129 TEST

S: SHELLCODE x86 NOOP 2008-01-16 07:34:33
89.139.111.141:3306   192.168.8.129:445
TCP
```

Attacks with source IP 192.168.25.75

The IP address mentioned above is the IP address of the Argos machine itself. During this experiment this machine has been registered 130 times. All entries contain a reference to the Allaple worm. In log 3 a log of this attack is visible. We suspect that the Argos image of Windows 2000 has been infected with this virus. It is possible that this happened while creating the image. The Argos images should be reloaded occasionally, so we expect that infection has to be in the image because if not, the virus would disappear after a reload.

We reported this to SURFnet which conducted a research on this. SURFnet did not detect a virus on the image. They did however find out that the Argos image didn't reload after every attack. It is possible that the machine got infected but didn't reload after this. After is was reloaded it could be enfected again with again the problem that it didn't reload. This is an explanation to why Snort discoverd these attacks.

Log 3 Allaple worm

```
 BLEEDING-EDGE WORM Allaple ICMP Sweep Reply Inbound   2008-01-17
14:17:51   192.168.25.75   58.13.12.123   ICMP
```

Date	Nep.	Snort	Overlap	Perct.
22-01-2008	55	52	24	43.6%
23-01-2008	88	230	62	70%
24-01-2008	52	840	52	100%
25-01-2008	28	175	28	100%

Table 5.2: Results of experiment 3

5.1.2 Results of experiment 1

Our goal with these experiments was to find out which setup of Snort in SURFids provides the most added value to the system while not degrading performance in a noticable way.

During the experiment the performance of the system was tested on various occasions and the result was an added average delay of 1.354ms, which is negligible. Because the Snort machine sits in between the tunnel server and the Argos machine and Snort is a CPU intensive process it can be expected that this delay will grow with the load of the server. But based on other research [13] we don't expect it to become a problem

In this setup Snort provides a lot of added information. It is able to detect more than 90% of all attacks (see table 5.1). Because of this the system is able to give a name to the registered attacks. Moreover, Snort detects attacks which aren't detected by Argos.

5.2 Experiment 2

This experiment has not been conducted due to time and hardware limitations. Therefore there are no results of this experiment. We expect the results of experiment 2 to be similar to the results of experiment 3.

5.3 Experiment 3

The same as with experiment 1, problems were encountered while conducting this experiment. There were some problems with the rulesets and with the used Snort versions. It appears that the version of Snort included in Ubuntu 7.10 which was used in experiment 1 is version 2.7 while Debian stable uses Snort 2.3. Snort couldn't read the rulesets because they were in a different format than the format used by version 2.7. Because we used the configuration file we also used in experiment 1 there were also some problems with the preprocessors. The solution to this all was to download the Snort 2.7 sourcecode and compile this.

Later-on a configuration fault was discovered in the configuration file. We solved this error and since then Snort has recognized 100% of the attacks reported by Nepenthes as malicious attack. This is visible in table 5.2

5.3.1 Results of experiment 3

Because Nepenthes is different from Argos and knows which attacks it detects it is less interesting to look at the overlapping results. More interesting in this

Date	Possible malicious	Verified as malicious by Snort	perct.
22-01-2008	15	1	7 %
23-01-2008	13	1	8 %
24-01-2008	14	2	14 %
25-01-2008	10	1	10 %

Table 5.3: Possible malicious attacks recognized by Snort

	Total	*Possible malicious attack.*	*Malicious attack.*	*Snort*
Closed port	10	0	0	3
Open port	10	9	1	5

Table 5.4: Summary of the Metasploit attack results

case are the results reported by Nepenthes as possible malicious attacks. These are shown in table 5.3

Because Snort detects more than 90% of the malicious attacks reported by Nepenthes we assume Snort will detect about 90% of the malicious attacks which for some reason were detected by Nepenthes as possible malicious attacks. Because of this assumption we can say for example that the 14 attacks that Snort didn't detect on 22-01-2008 were not malicious attacks. Only one possible malicious attack of that day was also detected by Snort as an attack. Therefore we say that that attack is a real malicious attack.

MetaSploit

To figure out whether Snort really *does* detect more attacks than Nepenthes we used MetaSploit to attack the IDS with some less-common attacks. The results of this are visible in table 5.4. The complete results are listed in [25].

Snort detects 3 of the attacks aimed at a closed port. Nepenthes detects none of these attacks. On the other hand Nepenthes detects all 10 of the attacks aimed at open ports while Snort detects only 5 of those. It is not strange for Nepenthes to register all attacks on its open ports, as can be seen in the table 9 of the 10 attacks are listed as possible malicious attack. Nepenthes registers *every* connection attempt on an open port as possible malicious. It is possible that an attacker launches an attack and that the attack itself aborts after the initial connection because the attack is not able to attack the running service. For example, an IIS attack is launched on an Apache server. The attack recognizes the Apache server and aborts the attack. Until this moment the traffic has been legitimate, so Snort does not detect the attack. Snort recognizes more attacks than Nepenthes. Nepenthes does report 9 possible malicious attacks, but only 1 malicious attack where Snort reports 8 malicious attacks. Snort only detects malicious attacks and because of this won't report possible malicious attacks.

Chapter 6

Integrating Snort output into SURFids

For the integration of Snort into SURFids it is important that the Snort output can be integrated into the current SURFids database. To determine whether this is possible this question has been divided into three steps:

1. Determine if the Snort output is compatible with the SURFids database. Could these values be inserted directly into the database, is conversion required or are extra tables needed.
2. How can SURFids use the Snort data?
3. Look into the possibilities to insert Snort output into the SURFids database.

6.1 Snort output compatibility with SURFids

In order to determine if Snort output is compatible with the SURFids database the Snort output values [14] were compared to the SURFids database schema [20]. The SURFids database has multiple tables, the ones used to compare to the Snort output are: *attacks* and *details*. In these tables SURFids stores the attack information and extra details on those attacks.

Attack table fields [20]:

- *id*: Unique identifier.
- *timestamp*: Timestamp in epoch format.
- *severity*: Severity of the attack (Nepenthes defines possible values: 0, 1, 16, 32). Possible malicious attack, Malicious attack, Malware offered or Malware downloaded.
- *source*: Source IP address.
- *sport*: Source port.
- *dest*: Destination IP address.
- *dport*: Destination port.

- *sensorid*: Identifier of the sensor that logged the attack.
- *src_mac*: The MAC address of the attacker if known.
- *dst_mac*: The MAC address of the destination.
- *atype*: Type of malicious attack: Nepenthes, Argos, ARP poisoning or Rogue DHCP.

Detail table fields [20]:

- *id*: Unique identifier.
- *attackid*: Identifier of the related attack.
- *sensorid*: Identifier of the sensor that logged the related attack.
- *type*: Type of attack (Nepenthes defines possible values: 1, 2, 4, 8). Type of detailed info, one attack can have multiple types.
- *text*: Detailed info.

The fields of the SURFids tables have been compared to all possible Snort output values. The overlap of the two can be found in table 6.1. Some of the SURFids fields are not found in the Snort output, these are: *severity*, *atype*, *sensorid* and *type*. These values are SURFids specific values and can be added to the Snort data.

The Snort output values that are non overlapping with the SURFids database fields are the *IP packet headers* and *IP packet payload*.

SURFids field name	*Snort field name*	*Additional information*
timestamp	timestamp	Conversion may be required depending on the Snort output module
source	src	
sport	srcport	
dest	dst	
dport	dstport	
src_mac	ethsrc	
dst_mac	ethdst	
text	msg, sig_id and url	

Table 6.1: Overlapping Snort and SURFids values

6.2 Using the Snort data in SURFids

The main reason for using Snort within the SURFids project is to give a more detailed view into Argos information. It is not desirable to log all the attacks twice in the *attack* table. To give more insight in the Argos attacks the extra Snort information can be logged into the *detail* table. In order to log the information there will have to be *type* values defined for the Snort information. *Type* values will have to be defined for the following Snort output fields:

- *msg*: The type of attack that Snort logged.
- *sig_id*: The signature identifier of the Snort rule, can be used to reference attack information on the Snort website. Enter the *sig_id* at the end of the following URL: http://www.snort.org/pub-bin/sigs.cgi?sid=
- *url*[1]: Some rules come with a URL to extra information on the attack.

SURFids defines a *type* value for every specific attack detail. To add the Snort information three extra *type* values will have to be defined for the values *msg*, *sig_id* and *url*.

In the experimental setup Snort detected attacks that Nepenthes and Argos did not. This information can be used in SURFids in the same way the Nepenthes and Argos data is presented today. It is important to check if an attack already has been detected by Nepenthes or Argos before adding the data to the database. An attack type or *atype* id has to be defined to use the Snort data independently.

6.2.1 Severity of an attack

SURFids registers attacks with a *severity* value, this can be: possible malicious attack, malicious attack, malware offered or malware downloaded. Snort doesn't distinguish between these levels, all registered attacks can be seen as malicious attacks. When a Snort rule is triggered by an IP packet it is considered malicious, if an IP packet is not malicious it shouldn't trigger a rule.

Snort does not provide rules that register if malware is offered or downloaded. During experiment three we did register *ATTACK-RESPONSES Microsoft cmd.exe banner* messages in Snort when Nepenthes registered: *malware downloaded*. Snort only confirms what Nepenthes registers.

6.3 Possibilities to integrate data

We have defined three options to get the Snort output from the Snort machine into the SURFids database:

- Import Snort tables into the SURFids database.
- Develop a Snort or Barnyard output plugin.
- Use a Snort or Barnyard output plugin.

We will discuss three options for integrating Snort data and points that have to be considered when chosing an integration method.

6.3.1 Import Snort tables into the SURFids database

Using the Snort database schema [6] all the Snort tables can be added into the SURFids database. The advantage of this implementation is that one can use the standard Snort database output plugin. It could also be useful if SURFnet

[1] Not all plugins provide this information but it can be extracted from the Snort rules using the *sig_id*

is interested in the data that doesn't correlate with the SURFids database like IP headers and packet payload, which could be useful for analyzing attacks.

The downside of this option is that parts of the SURFids web engine will have to be rewritten in order to process the data.

6.3.2 Develop a Snort output plugin

Snort has a modular setup which allows developers to write custom output plugins, this has been confirmed by the Snort 2.0 Intrusion Detection book [4]. Snort is open source software which enables a developer to use an existing output plugin as a template. The C sources and header files are all included in the */src/output-plugins* directory. Snort already comes with its own database module (spo_database) for the Snort database schema. This plugin could be a perfect candidate for a template.

The Snort complementary program Barnyard has a similar modular setup which enables an user to develop custom plugins.

6.3.3 Using a Snort output plugin

Snort provides a number of plugins to write output to a specific format. It is possible to develop a program or script to read this data and write it, conform the SURFids schema, to the SURFids database. This has also been confirmed by the Snort 2.0 Intrusion Detection book [4].

Possible Snort modules to use:

- *alert_unixsock*: writes the data to an UNIX socket.
- *alert_fastlog*: output minimal data in ASCII format to a file. Also supported by Barnyard
- *csv*: output the data in comma separated value ASCII format to a file. Also supported by Barnyard.
- *unified*: writes the data in unified binary format.

Besides the standard Snort output plugins there is also a third party plugin, created by CERT.org [5], which outputs the Snort data in a XML format. The plugin can write the data to a XML file or send it, with the HTTP POST option, to a script on a web server.

6.3.4 Multiple Snort output by one attack

Snort logs all attacks it registers. In practice this means that if an attacker sends multiple IP packets which trigger a Snort rule all the IP packets will trigger a separate log/alert action. With a custom output plugin or processing script/program this can be handled and logged to the SURFids database only once.

6.3.5 Snort logs and alerts

Snort knows two output options, log and alert [16]. The alert facility provides information on the events that happened, basically the information in the SURFids *attack* and *detail* tables. The log facility outputs the full IP packets. With the use of output plugins one can use separate plugins for both types or use one plugin that can handle both. Plugins that can handle both alert and log information are the database and XML plugin. It's important to consider these output options if SURFnet intends to use IP payload information for analyzing attacks.

6.4 Conclusion

Snort output can be inserted in the SURFids database with a few minor changes. There will have to be *type* values defined for the Snort attack information. In order to use Snort data independent from Argos and Nepenthes an *atype* will also have to be defined for Snort.

The most practical option to insert the Snort output into the SURFids database is to develop an output plugin or process data from a standard Snort output plugin. Both options provide the same benefits:

- Snort output can be inserted conform the SURFids database schema.
- Checks can be done before inserting the data. To determine if the data is to be used complementary to Nepenthes or Argos or as an individual attack. Make sure an attack is not logged to the database multiple times.

The Snort output values *IP packet headers* and *IP packet payload* can be integrated into the SURFids database by adding a table to the database or adding *type* values for the *detail* table. To be able to use full IP packet and payload information one has to use log information. If SURFids chooses to use this information they can modify the Snort database plugin or extract information from an other log format, like XML, with a program or script.

The *severity* of an attack registered by Snort can be registered as a malicious attack. Snort does not provide enough information to detect malware offers or downloads. The registered *ATTACK-RESPONSES Microsoft cmd.exe banner* message can also be detected for other reasons for example after a successful DCOM attack.

Chapter 7

Conclusion

Implementing Snort in SURFids is not only possible, it will add very interesting data into the system, while it won't degrade performance in a noticeable way. We strongly advise implementing Snort into SURFids. The following benefits will be achieved:

- It will be possible to classify possible malicious attacks as reported by Nepenthes as malicious attacks.
- More information about Argos attacks.
- A whole range of previously undetected attacks.

The only way to get all these benefits is to implement Snort in the way described in experiment 2 or 3. Our preference would be experiment 2. The reason for this is that in experiment 2 the Snort machine is placed completely outside the normal traffic flow, therefore the Snort machine won't delay the traffic at all. This also improves the manageability of the system because all servers have one specific task. This setup would require moving the Nepenthes and database installation from the tunnel server to separate servers. This is shown in image 2.1.

To be able to work with Snort in SURFids it must first be possible to insert the Snort data into the SURFids database before it is of any use. To be able to do this some issues need to be solved:

- Snort often logs attacks more than once. This needs to be detected to prevent double attacks from being recorded in the database.
- The output from Snort needs to be adapted so it can be inserted into the SURFids database (see chapter 6).

We advise SURFnet to use Snort unified binary output and to use Barnyard to processes that output. With the use of Barnyard Snort is offloaded and database connection problems can be dealt with in an appropriate way. There are two possibilities with Barnyard; develop a custom database output plugin or use the csv output plugin.

The Barnyard csv output plugin provides a format that can be parsed easily and the format is well documented in the Barnyard configuration file. This

solution does not deal with payload information which is not used in the current SURFids setup. Developing a program or script to deal with csv provides an extra step in the data processing process and an extra point of failure. It is however relatively easy to develop.

Developing a custom Barnyard database plugin is the shortest output process but the development process is longer and relatively more difficult than developing a csv parser program or script. A custom database plugin also provides the ability to deal with IP payload data should SURFnet be interested in this, now or in the future.

Based on the conclusions made we can determine that Snort can offer a real added value to SURFids.

7.1 Future work

Before Snort can be integrated into SURFids an output plugin or parser has to be developed for Barnyard. This plugin needs to be able to detect double entries from Snort and remove them. Furthermore it needs to be able to recognize Argos and Nepenthes registrations so it can update them, or drop a Snort alert in order to prevent double registrations of attacks.

We also advice to implement the logical setup in a physical way. This means changing the current setup of SURFids to the setup as shown in figure 4.2.

Bibliography

[1] Andrés Felipe Arboleda and Charles Edwared Bedón. Snort™ diagrams for developers. *Universidad del Cauca - Colombia*, 2005. http://afrodita.unicauca.edu.co/~cbedon/snort/snortdevdiagrams.html.

[2] Argos. Argos official website. http://www.few.vu.nl/argos/.

[3] Andrew Baker. Barnyard. http://sourceforge.net/projects/barnyard.

[4] Brian Caswell, Jay Beale, James C. Foster, and Jeremy Faircloth. *Snort 2.0 Intrusion Detection*. Syngress, 2003.

[5] CERT.org. Snort XML output plugin. http://www.cert.org/kb/snortxml/.

[6] Roman Danyliw. Snort and ACID database ER diagram. http://www.andrew.cmu.edu/user/rdanyliw/snort/acid_db_er_v102.html.

[7] Bleeding Edge Threats. Bleeding Edge Threats website. http://www.bleedingthreats.net/.

[8] SURFnet homepage. http://www.surfnet.nl.

[9] Van Jacobson, Craig Leres, and Steven McCanne. libpcap. *Lawrence Berkeley National Laboratory*, 1994. http://www-nrg.ee.lbl.gov/.

[10] Jack Koziol. *Intrusion Detection with Snort*. Sams, 2003.

[11] Nepenthes. Nepenthes official website. http://nepenthes.mwcollect.org.

[12] Andreas Östling. Oinkmaster. http://oinkmaster.sourceforge.net/.

[13] Antonis Papadogiannakis, Demetres Antoniades, Michalis Polychronakis, and Evangelos P. Markatos. Improving the Performance of Passive Network Monitoring Applications using Locality Buffering. *Institute of Computer Science Foundation for Research & Technology - Hellas*, 2007. http://dcs.ics.forth.gr/Activities/papers/pcapLB-paper.pdf.

[14] The Snort™ Project. Snort users manual. 2007. http://www.snort.org/docs/snort_manual/2.8.0/snort_manual.pdf.

[15] The Snort™ Project. The official Snort website. http://www.snort.org/.

[16] The Snort™ Project. What is the difference between "Alerting" and "Logging"? http://snort.org/docs/faq/1Q05/node72.html.

[17] Thomas H. Ptacek and Timoth N. Newsham. Insertion, Evasion, and Denial of Service: Elluding Network Intrusion Detection. *Secure Networks, Inc.*, 1998. http://www.snort.org/docs/idspaper/.

[18] Martin Roesch. Snort - Lightweigth Intrusion Detection for Networks. *LISA*, 1999. http://www.usenix.org/event/lisa99/full_papers/roesch/roesch_html/.

[19] Sourcefire. Sourcefire official website. http://www.sourcefire.com/company/.

[20] SURFnet. SURFids database tables. http://ids.surfnet.nl/wiki/doku.php?id=docs:2.00:debug.

[21] Sourcefire Vunerability Research Team. Snort™ Rules. http://www.sourcefire.com/products/snort/rules/.

[22] Michael van Kleij. Experiment 1: Snort before argos. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[23] Michael van Kleij. Experiment 3: Snort on the tunnelserver. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[24] Michael van Kleij & Sander Keemink. Results of experiment 1. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[25] Michael van Kleij & Sander Keemink. Results of experiment 3. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[26] Phil Woods. MMAPed pcap. http://public.lanl.gov/cpw/.

Appendix A

List of Appendixes

This section contains a list of the appendixes included with this report.

A.1 Project plan

The project plan contains the project guideline as well as the project definition.

A.2 Experiment setup 1

Experiment 1 is focussed on analyzing the added value of Snort in addition to Argos. This is done by placing Snort in between the SURFids tunnel server and the Argos machine.

A.3 Experiment setup 2

Experiment 2 focusses on analyzing both Nepenthes and Argos data by attaching Snort to the switch where Argos and Nepenthes are connected to and placing the port in monitoring mode.

A.4 Experiment setup 3

Experiment 3 has the same focus as experiment number two but in a different setup. Snort is placed on the tunnel server and analyzes all the data coming through the tunnels to Nepenthes or Argos.

A.5 Experiment results 1

The results of experiment 1 show that Snort is able to recognize over 90% of the attacks made to Argos and that Snort is able to recognize attacks that Argos does not recognize.

A.6 Experiment results 3

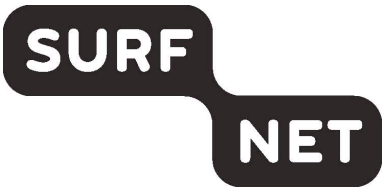
Experiment 3 shows that Snort is able to recognize over 90% of the reported malicious attacks by Nepenthes and a minority of the possible malicious attacks. Further more we explain why Snort only detects a minority of the possible malicious attacks and why we consider the remaining possible malicious attacks false positives.

Implementing Snort into SURFids

Sander Keemink, Michael van Kleij
January 9, 2008

UNIVERSITEIT VAN AMSTERDAM

Masteropleiding System and Network Engineering



Contents

Introduction

This document is meant to be a guideline in our research of the integration of Snort into SURFids. In this research we try to find the best way to implement Snort into SURFids. At this moment multiple questions arise as to which implementation is the best.

We've got four weeks to complete this research. We start on 07-01-2008 and we should be finished on 01-02-2008. These four weeks are divided as follows:

1. Preliminary tasks
2. Research
3. Research
4. Concluding research and writing paper

This document is organised in the same way as the available time. First we'll describe the preliminary tasks, then the research itself followed by the concluding tasks.

1 Preliminary tasks

The preliminary tasks consist of the following tasks:

- Reading into subject matter
- Writing this document
- Preparing the experiments

- Installing and configuring the test environment

We'll further specify these tasks below.

1.1 Reading into subject matter

We're going to read the available documentation of SURFids, Snort and Argos. Nepenthes will be touched, but only on the surface. The goal of this task is to get a better understanding of how these programs work and how we might be able to let these programs cooperate with each other.

1.2 Writing this document

As stated in the introduction this document is meant to serve as a sort of guideline for us in this project. This document makes it clear what we will do and, more importantly what we won't do.

1.3 Preparing the experiments

For each experiment we will make a detailed overview of what we will do during this experiment. We will give all the points of interest and the way we will test these points of interest. Part of preparing the experiments is designing the test setup for each experiment.

1.4 Installing and configuring the test environment

This task might be repeated multiple times if the test environment differs between tests. In that case only the first test environment will be installed in the preliminary tasks period. The other test environments will be installed when necessary.

2 Research

During the research period we will try to answer all the questions we've asked ourselves during the preparation of these experiments. The research goal is to be able to answer the following question:

Which implementation of Snort into SURFids gives the most added value to the customer while not degrading performance in a noticable way.

We will research the following possible implementations:

1. Placing Snort on the tunnel server
2. Placing Snort before Argos
3. Implementing Snort as a stand-alone honeypot (much like the way Argos and Nepenthes are deployed)

To find out which of the above implementations is the best we'll look at the following characteristics of the system:

- Performance
- Informational value
- Added value to the existing system
- Ease of administration

Because of the time limitation of one month we won't do the following:

- Implementing Snort into the live SURFids
- Creating a step-by-step manual to implement Snort

When we're done with the project SURFnet should be able to use our results and the documentation about how we've set up our test environment to start a project to implement Snort into the live SURFids.

3 Concluding research

During this final fase of our research we'll do the following tasks:

- Finishing the research
- Writing the report
- Making a presentation of our results

3.1 Finishing the research

It's possible that the research period has not been fully completed at the end of the third week. Therefore we have some time scheduled to finish the research.

3.2 Writing the report

When all experiments are finished we can write the report containing the results of the experiments and our conclusion.

3.3 Making a presentation of our results

We will make a presentation containing our results and conclusion for SURFnet and the UvA.

Experiment 1: Snort before Argos

Goal

The goal of this experiment is to determine whether the setup as displayed in figure 2 will work and wheter it adds any value to the existing setup.

Setup

For this experiment we need a Snort server which is configured like a transparant proxy. The Snort machine will be placed in between the Tunnelserver and the Argos server. In figure 1 on page 1 the current situation is depicted. In figure 2 on page 2 the situation during the experiment is depicted.

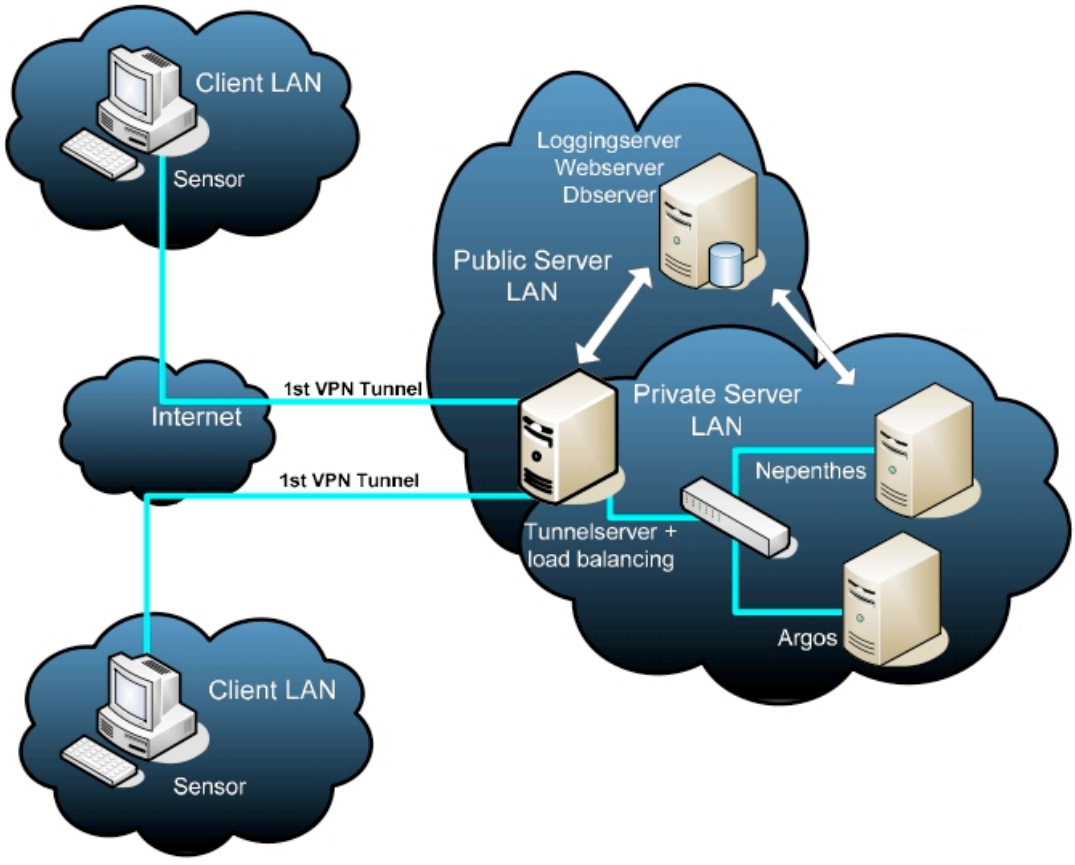


Figure 1: Schematic view of the current situation of SURFids

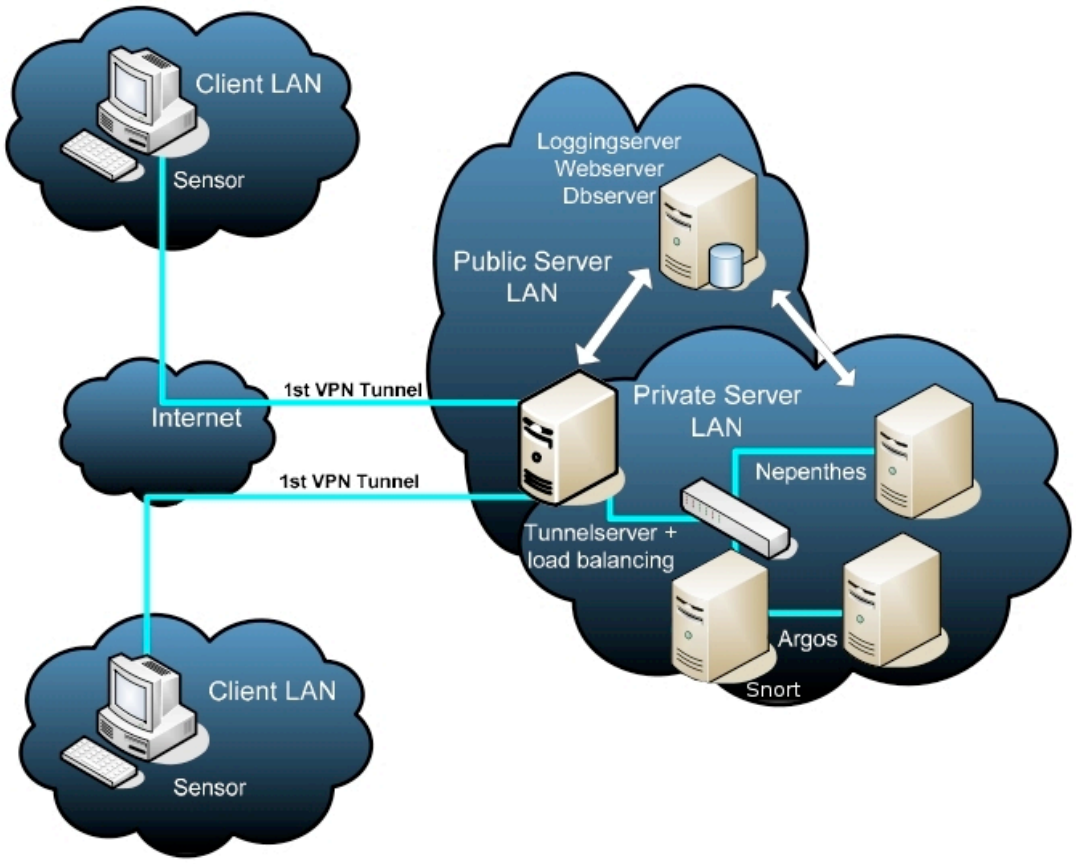


Figure 2: Schematic view of SURFids during the experiment

Requirements

For this experiment the following is required:

- A server with at least 3 network interface cards
- A Debian based OS
- Snort with PostgreSQL support
- Real data
- An extra UTP cable
- A SURFids implementation

Setting up the Snort server

1. Install a Debian based OS on a server with at least 3 network interface cards.

2. Install Snort and PostgreSQL. Make sure you select the eth0 interface as the interface on which Snort will listen. [2] [3]

```
apt-get install snort snort-pgsql postgresql-8.1
```

3. Configure the database

```
su postgres
createdb snort
zcat /usr/share/doc/snort-pgsql/create_postgresql.gz | psql snort
createuser -P snort

Enter password for new user: snort-password
Enter it again: snort-password
Shall the new user be a superuser? (y/n) n
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

4. Log into the database

```
psql snort
```

5. Give the snort user the correct privileges

```
grant all privileges on database snort to snort;
```

6. Edit the Snort configuration. Find the line that says #output plugins and insert the following line:

```
output database: alert, postgresql, user=snort
password=snort-password dbname=snort host=postgresql-host-ip
```

7. Make sure PostgreSQL is correctly configured to accept connections from Snort

8. Create a bridge interface [1]

```
/usr/sbin/brctl addbr br0
```

9. Add the real network interfaces to the bridge

```
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1
```

10. Start the network interfaces

```
/sbin/ifconfig eth0 0.0.0.0
/sbin/ifconfig eth1 0.0.0.0
```

We will use the default Snort ruleset and some of the bleeding edge Snort rulesets for this experiment, the reason for this is that we don't have access to the paid ruleset.

Running the experiment

Snort must now run for about a day to gather information about the incoming traffic. We expect to gather information of about 100 attacks during this day. If we gather less than this amount we might need to extend the test period to 2 or 3 days. To make sure that we gather enough information it might be possible to configure the SURFids system to send all possible attacks to the Argos machine. This should cause more known attacks to be sent to Argos. Because Argos doesn't recognise known attacks and Snort does, we can compare the alerts that both systems give.

Expected results

We expect to gather at least 100 alarms from Snort. These results must be compared to alarms from Argos. We expect that we will be able to recognise all known attacks and hope to provide more information in attacks that are as yet unknown.

Furthermore we expect this setup of Snort to:

- Degrade performance with a small amount, adding a few ms RTT[1] per packet.
- Be able to cope with the load very well, now and in the future.
- Difficult to manage and extend.

[1] RTT: Round Trip Time

Degrade performance Because of the added (but invisible) hop in the network about 5ms of RTT latency will be added.

Load Because only the traffic destined for Argos is passed through Snort we don't expect a heavy load on the Snort server. As long as the average network throughput to the Argos machine doesn't become to high[2] Snort should be very well able to manage the load. Especially in a Snort - Barnyard setup – which we won't test –

Difficult to manage and extend Because the Snort machine is a bridge between the tunnel server and Argos it's a high risk machine. When this machine fails the Argos machine won't be available. It will also be difficult to extend the Snort server when the load for this machine becomes to high. Adding a second bridge should be possible, but brings a whole range of other (management) problems with it.

References

[1] Francois Bayart. Setting up a bridge firewall. http://www.debian.org/doc/manuals/securing-debian-howto/ap-bridge-fw.en.html, 2008.

[2] Anton Chuvakin and Vladislav V. Myasnyankin. Complete snort-based ids architecture. http://www.securityfocus.com/infocus/1640, 2002.

[3] unknown. Perfect setup of snort + base + postgresql on ubuntu 6.06 lts. http://www.howtoforge.com/intrusion_detection_snort_base_postgresql_ubuntu6.06, 2007.

[2] We expect that to high is around 50Mbit+, but that's only a guess.

Experiment 2: Snort next to Argos and Nepenthes

Goal

The goal of this experiment is to give insight in the data Snort provides in comparison to Nepenthes and Argos. The focus of this experiment is to give insight if Snort provides an added value to the data Argos provides. With this setup one can also compare Nepenthes and Snort data, does Snort give more information, does Snort detect more/less intrusions?

Setup

In this experiment Snort will be implemented on a stand alone machine. Snort will listen to all traffic on it's interface. Configure the switch, to which the Argos and Nepenthes machines are connected, to echo all it's data to the Snort machine. This way Snort will be able to analyse all data coming through the switch. This will not effect the running configuration of both systems and gives insight in the data Snort collects in comparison to the data Argos and Nepenthes provide. In the current situation the tunnel server sends the data to either Nepenthes or Argos see figure 1.

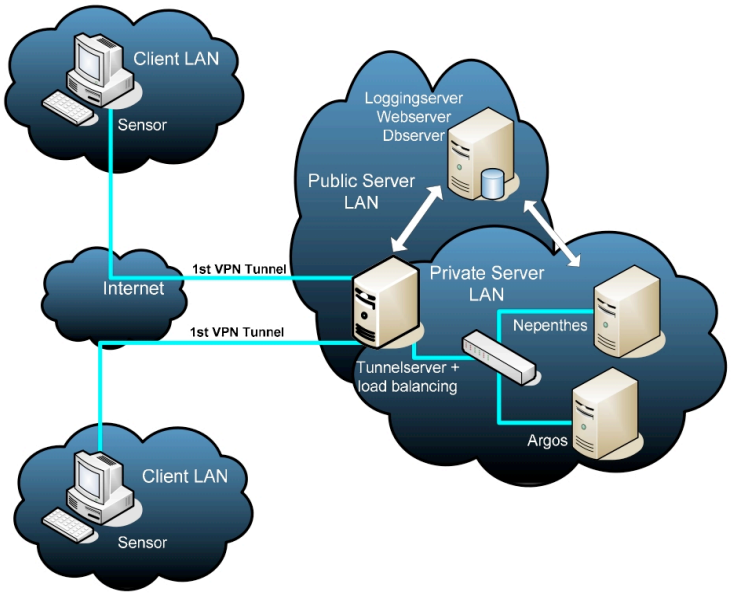


Figure 1: Schematic view of the current SURFids setup

The new situation as seen in figure 2 shows Snort being placed next to Argos and Nepenthes. *Unfortunately this picture is the logical situation. Nepenthes is physically located on the tunnel server. Because of this setup Snort won't be able*

to analyse Nepenthes data. In a future setup where Nepenthes is located next to Argos this data can also be analysed by Snort.

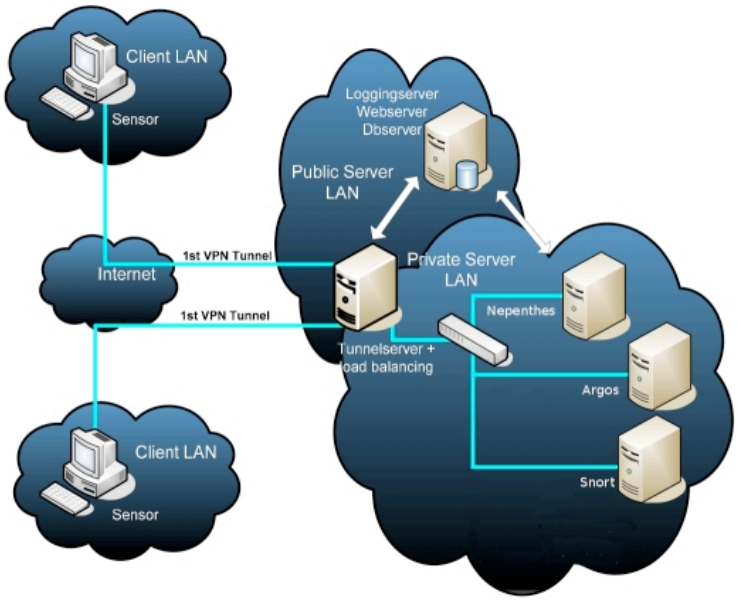


Figure 2: Schematic view of SURFids during the experiment

Requirements

For this experiment the following is required:

- A server with at least two network interface card
- A Debian based OS
- Snort with PostgresSQL support
- Real data
- A SURFids implementation

Setting up the Snort server

Setting up Snort and PostgreSQL will not be handled thoroughly for more information see the description of experiment 1 [1]

1. Configure the switch to echo all traffic to the port on which the Snort machine is connected. On most switches this is also referred to as monitoring data.

2. Install a Debian based OS on a server with at least tow network interface cards.
3. Install Snort and PostgreSQL
4. Configure the database

Running the experiments

Based on the information from the SURFids environment where we will conduct our experiment we expect around 100 possible malicious attacks on the SURFids sensors per day. If a possible hostile source tries to conduct multiple malicious attacks those attacks will be send to Argos. At this moment there aren't many (any) Argos results, this could influence the length of the experiment.

Expected results

Depending on the amount of positive attacks on Argos (that will log data) we will be able to conclude whether Snort provides added value to the current setup. Running the experiment for one day we expect to get around 100 alerts which can be compared to the Argos data. If one day doesn't provide enough data we will have to run the experiment for two or three.

Expected results from Snort:

- Snort will log all known attacks.
- No impact on the running setup.
- The Snort machine will be able to cope with the load very well, now and in the future [1].

References

[1] M. van Kleij. Expirement 1: Snort before argos. http://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbookl, 2008.

Experiment 3: Snort on the Tunnel server

Goal

The goal of this experiment is to determine whether it is possible to install Snort on the tunnel server and if this setup has any added value to the current system as depicted in figure 1.

Setup

The current SURFids situation is depicted in figure 1. In the experiment this situation will not be changed. Snort can be installed on the tunnel server and configured to listen on the interface which connects to the honeypot systems.

Unfortunately image 1 is only a schematic view. In reality Nepenthes is installed on the tunnel server. For this experiment to be succesfull there are several options. Easiest of which would be installing Nepenthes on a seperate machine. In which case this setup would logically be the same as the setup in experiment 2 [1].

Requirements

For this experiment the following is required:

- full access to a SURFids setup and thus the tunnel server
- Real data
- Snort with PostgreSQL support

Setting up the Snort server

Install Snort on the tunnelserver as described in experiment 1 [2]

Running the experiment

This experiment must run for about one day to gather enough information. because this experiment will capture both data from Argos and Nepenthes the gathering of data will go faster than in experiment 1.

Expected results

We expect this experiment to gather exactly the same information as experiment 2. We expect the management, expandability and performance of this setup to be less than experiment 2.

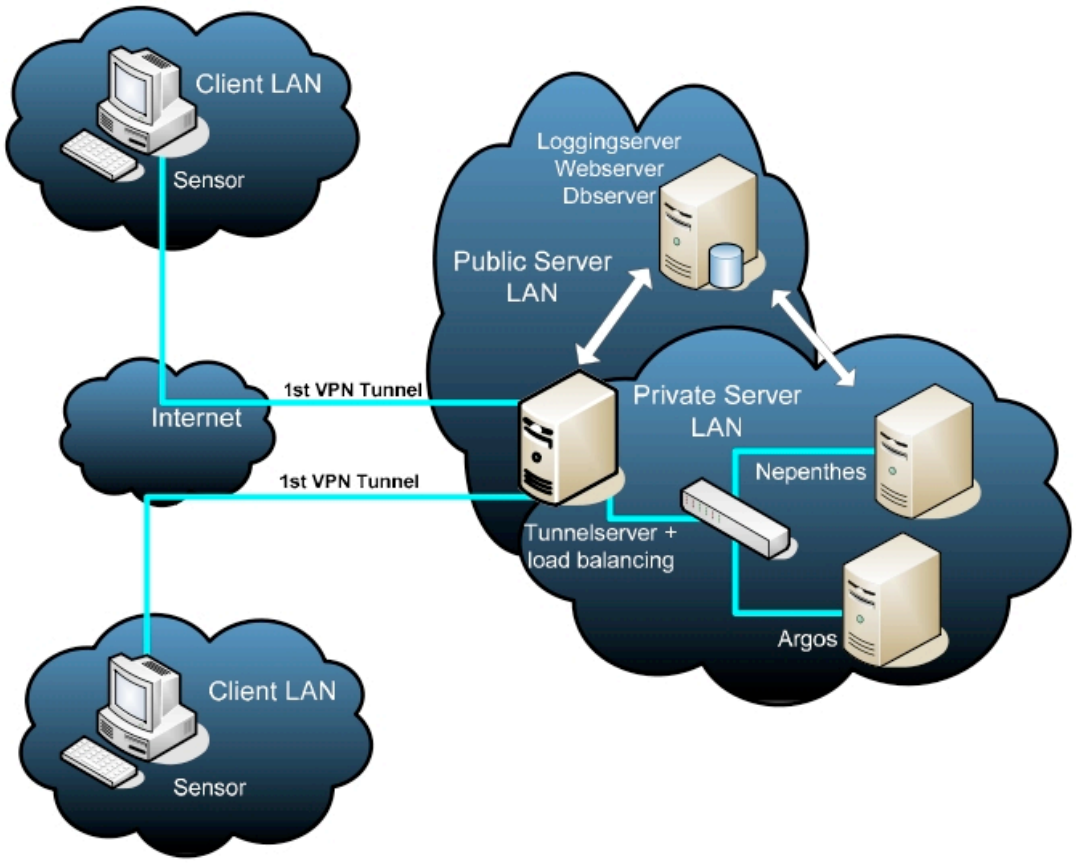


Figure 1: Schematic view of the current situation of SURFids

References

[1] Sander Keemink. Experiment 2: Snort next to argos and nepenthes. https://www.os3.nl/2007-2008/students/sander_keemink/rp1, 2008.

[2] Michael van Kleij. Experiment 1: Snort before argos. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

Results of Experiment 1

Sander Keemink, Michael van Kleij

January 30, 2008

1 Introduction

This document describes the results of the first experiment executed for SURFnet in the "Implementing Snort into SURFids" project. The experiment itself is described in [4]. During this experiment we've seen various results and problems. These are described in this document.

2 Running the experiment

We began working on this experiment on the 11th of January 2008. We configured the server to be able to function as a bridge. For this we needed to add two network interface cards into the server. When the bridge was fully functional we inserted this server in between the tunnel server and the Argos server.

2.1 Snort

During the installation of Snort we encountered some problems. The first one was a database problem. Later on we had some problems with the rulesets. These problems have all been solved. In the following paragraphs we describe the problems and their solutions.

Database As database we installed PostgreSQL version 8.2. In Ubuntu this server is configured to run on port 5433. Snort however, is configured to connect to the PostgreSQL database on port 5432. This gave us some problems because we couldn't get Snort to connect on port 5433. When we configured the PostgreSQL database to listen on port 5432 instead of port 5433 this problem was solved.

The next database related problem was a rights problem. Even though we granted the user Snort all privileges as described in [4] Snort couldn't write in the database. Using phppgadmin the rights were changed[1] to let the Snort user write to the database. This solved some of the problems, but eventually we changed the owner of the database to the Snort user and hereby solving all the remaining problems with writing to the database.

Rulesets The rulesets are the core of Snort. These rulesets are created to catch the exploits and report these to the user. In our case with Snort before Argos we wanted to see at least the attacks Argos reported, but preferrably more.

When we analysed the results for the first time something caught our attention. Not all of the attacks reported by Argos were reported by Snort, but other attacks that were not reported by Argos were reported by Snort. We wanted to make sure that Snort recognised at least the attacks Argos saw so we added some rules like the following rule: 1. These rules were specific for DCOM attacks on port 135 [2]. We also enabled the shellcode ruleset as included in Snort to be able to see shellcode attacks.

Example 1 Snort rule

```
alert tcp any any -> any 135 (msg:"DCOM
Exploit (MS03-026) targeting Windows 2000
SP0"; content:"|74 16 e8 77 cc e0 fd 7f
cc e0 fd 7f|";
classtype:attempted-admin; sid:1100001;
reference:URL,www.microsoft.com/security/
security_bulletins/ms03-026.asp;
reference:URL,jackhammer.org/rules/1100001;
rev:1;)
```

[1] Thanks to Kees who changed the rights in the database

2.2 Barnyard

Even though we didn't mention Barnyard in the experiment setup we thought Barnyard might have some added value in this setup. Barnyard is a program which analyses the unified logs Snort supplies and handles these. In our case Barnyard writes the data into the database instead of Snort. The advantage of this is that Barnyard takes a part of the load off Snort. Especially in a setup where the data needs to be written to a database on another host. This insures Snort registers all attack attempts. Barnyard is also able to recognise wheter the database is available, and if it isn't it won't send updates to the database untill the database is available again [1].

Because we use Bleeding Edge Snort rules and some special rules for dcom attacks Barnyard wasn't able to recognise all attacks. Barnyard uses the files *gen-msg.map* and *sid-msg.map* to find the attack names from the attack signature in the logfile we had to use a script [5] which is able to generate these maps.

3 Results

During the days we've run this experiment we've gathered a lot of information. As of 16 January 2008 we've collected almost 300 'attacks', whereas Argos has collected only 108 attacks. Of these 108 attacks 61 were also discovered by Snort. Since 15 January however, we detect more than 90% of all attacks discovered by Argos. In table 1 you can see the amount of attacks discovered by Argos, Snort and their overlap. On 16 January we saw an attack registered in Argos and Snort of which we're sure it's the same attack. Argos reports this attack on different ports than Snort, so we didn't count this attack as an overlapping attack. See section 3.1 and log 1 for more information.

Date	Argos	Snort	Overlap	Perct.
12-01-2008	20	3	2	10%
13-01-2008	20	10	5	25%
14-01-2008	32	97	20	62,5%
15-01-2008	36	178	34	94%
16-01-2008	26	139	25	96%

Table 1: Results from Snort and Argos

A lot of the results from Snort in the table are attacks which don't register at Argos because of Argos not running those services – like MS-SQL –. Another reason of the high amount of results from Snort is that Snort is a network ids which inspects packets. Some attacks are executed in a way that multiple packets with the same payload are sent to the attacked host. Snort sees each of these packets as an individual attack where Argos sees this as a single attack. In Appendix A you can find all the overlapping attacks and some of the attacks on which only Snort triggered.

3.1 Points of interest

During this experiment we found some things which were strange. Here we specify these strange results:

- *Time skew:* There is a clock skew between Argos and Snort. Argos is about 10 minutes ahead of our Snort machine. The time in the Snort machine is synced with ntp.ubuntu.com. So our guess would be the Argos machine being 10 minutes ahead. At the moment of writing this time skew flipped over and now Argos is behind.
- *Multiple entries per attack* As we've already said in section 2 Snort enters multiple entries in the database in some attacks. To be able to reliably implement Snort into SURFids it would be best to be able to recognise these double entries and filter them out of the results. We can recognise these results using the timestamp, source port, destination port and name of the attack. When the timestamp is within 1 or 2 seconds from earlier entries this entry can be seen as a duplicate. The names however might differ because it's possible that multiple rules trigger on an attack.
- *Missing payload data* When using Barnyard the payload data is lost. This data might not be interesting for most of the customers, and at the moment SURFids might not even be able to store this data, but in analysis of attacks this data might come in handy.
- *Port difference between Argos and Snort* On the 16th of January we saw an attack reported by both Snort and Argos. Argos reported this

attack on port 6129 while Snort reported this attack on port 445. The attack logs of this attack can be seen in log 1. Keep in mind that the timestamps between Argos and Snort differ with aproximately 10 minutes. For an explanation of the logformat you can look at table 2.

- *Attacks with source ip 192.168.8.75* This ip address is the ip address of the Argos machine. The attacks are almost always directed outward to the internet. it appears that an Argos image might be infected with a virus.

Log 1 Port difference between Argos and Snort

```
A: 16-01-2008 07:44:42 Malicious attack
- Argos 89.139.111.141 3421
192.168.8.129 6129 TEST

S: SHELLCODE x86 NOOP 2008-01-16 07:34:33
89.139.111.141:3306   192.168.8.129:445
TCP
```

3.2 Performance

During the experiment we've looked closely to the performance of the system. Our main focus was on the performance of the complete SURFids system and not so much of the system running Snort. However, we do expect the overall SURFids system will experience degradation of performance when the Snort machine runs with a heavy load. At this moment performance loss because of the load on the server is neglible, therefore we've only looked to the ping averages of the system with the Snort machine and the system wthout the Snort machine. You can see these results in log 2 and log 3. As can be seen in these results there is a small added delay caused by the Snort machine. The average added delay is 1.354ms, which is a neglible difference. When the Snort machine has a higher load we expect this difference to grow, because the load on Snort influences the speed with which the machine can handle the packets for the Argos machine.

Log 2 Ping averages with Snort

```
30 packets transmitted, 30 received,
0% packet loss, time 29001ms
rtt min/avg/max/mdev =
5.911/8.391/34.333/5.171 ms

30 packets transmitted, 30 received,
0% packet loss, time 29000ms
rtt min/avg/max/mdev =
5.845/9.220/44.890/7.453 ms

30 packets transmitted, 30 received,
0% packet loss, time 28997ms
rtt min/avg/max/mdev =
5.839/9.394/33.939/6.313 ms

30 packets transmitted, 30 received,
0% packet loss, time 29021ms
rtt min/avg/max/mdev =
5.654/7.842/28.170/4.413 ms
```

3.3 Informational and Added Value

Snort in the current situation will add some value to the current SURFids system. It registers more than 90% of the attacks which are reported by Argos. In these situations Snort is often able to tell which attack it is, as can be seen in appendix A. Because of this we can see when an attack appears to be a new attack. It is possible that an existing attack isn't recognised by Snort, but with the current rulesets we don't expect this to be a common issue. One of the ways to be more sure that relatively new attacks will be recognised is buying the paid rulesets. These contain rules that are newer than the public rulesets. Updating these rules can be done automatically using Oinkmaster [3].

When using Barnyard the amount of information gathered from Snort decreases somewhat as it doesn't have the payload data anymore. This might or might not be an issue depending wheter SURFnet wants this information.

3.4 Ease of administration

As we've described in the experiment setup [4] we expected this setup to be difficult to manage and extend. There are some reasons for this.

Log 3 Ping averages without Snort

```
30 packets transmitted, 30 received,
0% packet loss, time 29001ms
rtt min/avg/max/mdev =
5.808/7.354/30.453/4.411 ms

30 packets transmitted, 30 received,
0% packet loss, time 29002ms
rtt min/avg/max/mdev =
5.880/7.220/12.821/2.060 ms

30 packets transmitted, 30 received,
0% packet loss, time 29000ms
rtt min/avg/max/mdev =
5.885/8.518/36.331/5.862 ms

30 packets transmitted, 30 received,
0% packet loss, time 28998ms
rtt min/avg/max/mdev =
5.902/6.339/8.575/0.608 ms
```

1. The system can only be reached using a special management network interface card.
2. When the Snort machine's load becomes to high it needs to be replaced, or there must be a second Snort machine added. When this is the case an Argos machine must also be added. Another possibility is using a loadbalancer before Snort and converging the traffic to one Argos machine.

The management issue won't be such a problem as we thought at first. As long as it's possible to have 3 network interface cards in the server it is possible to create a bridge and a network interface. Because the interfaces within the bridge don't have an ip address they're in no way directly reachable. The interface with an ip adres will be heavily firewalled. See example 2 for the used firewall rules.

Extending the setup is possible, but must be thought through extensively before being done. One of the questions that need to be answered before extending the setup is whether it is necessary to add an extra Argos machine to the setup or if it's enough to just add another Snort machine. When it's necessary to add an Argos machine it might also be necessary to add a Snort machine, or add a loadbalancer behind the Snort machine.

Example 2 iptables rules

```
iptables -A INPUT -i eth0 -s 192.168.7.0/24
-p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i eth0 -s 192.168.7.0/24
-p tcp --dport 80 -j ACCEPT
iptables -A INPUT -i eth0 -m state --state
ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -j REJECT
```

Conclusion

We believe this setup of Snort is able to bring interesting data into SURFids. When this data is stored in a good, intelligent way it is possible to add relevant information to the Argos reports. Because Snort registers more attacks than Argos does we think it might be even better to add this information to the database as well. With this information SURFids is better able to detect various attacks that might previously have gone unnoticed. However, there are some issues that need to be resolved before Snort can be integrated in SURFids. There needs to be a way to recognise double alerts for one attack, and the time skew needs to be solved.

References

[1] Snort core team. Snort faq. http://www.snort.org/docs/faq/1Q05/node86.html.

[2] BT Counterpane. Security alert: Microsoft rpc dcom remote shell vulnerability. http://www.counterpane.com/alert-v20030801-001.html.

[3] Andreas Östling. Oinkmaster. http://oinkmaster.sourceforge.net/.

[4] Michael van Kleij. Experiment 1: Snort before argos. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[5] Phil Wood. One liner to generate map files from rules. http://www.mcabee.org/lists/snort-users/Aug-02/msg00758.html.

A Results

This appendix contains a subsection of the results gathered from Snort. We will show all the overlapping results and some of the Snort results of which there was no Argos result.

A.1 Overlapping results

The layout of the results is as displayed in table 2

Argos:	date	attack	source ip	source port	dest. ip	dest. port	Sensor	info
Snort:	attack	date	source ip	source port	dest. ip	dest. port	protocol	

Table 2: Results layout

```
A: 12-01-2008 15:42:07  Malicious attack - Argos  212.147.79.166
65225  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-12 15:51:29.663
212.147.79.166:65225     192.168.8.129:445   TCP

A: 12-01-2008 18:35:55  Malicious attack - Argos  91.35.236.128
2875  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-12 18:46:10.804
91.35.236.128:2875   192.168.8.129:445   TCP

A: 13-01-2008 04:43:40  Malicious attack - Argos  194.203.40.52
38590  192.168.8.129 445  TEST  services.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-13 04:53:57.23
194.203.40.52:38590   192.168.8.129:445   TCP

A: 13-01-2008 11:55:28  Malicious attack - Argos  220.145.105.163
1990  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-13 12:05:43.888
220.145.105.163:1990   192.168.8.129:445   TCP

A: 13-01-2008 17:51:49  Malicious attack - Argos  82.233.196.6
1689  192.168.8.129 445  TEST  lsass.exe
S:  NETBIOS SMB-DS IPC$ unicode share access   2008-01-13 18:02:06.951
82.233.196.6:1689   192.168.8.129:445   TCP

A: 13-01-2008 19:06:55  Malicious attack - Argos  86.147.233.54
62512  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-13 19:17:12.952
86.147.233.54:62512   192.168.8.129:445   TCP

A: 13-01-2008 20:55:13  Malicious attack - Argos  89.61.252.93
3446  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-13 21:05:31.80
89.61.252.93:3446   192.168.8.129:445   TCP

A: 14-01-2008 04:23:40  Malicious attack - Argos  71.49.152.204
```

```
22130  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-14 04:33:56.73
71.49.152.204:22130   192.168.8.129:445   TCP

A: 14-01-2008 04:30:27  Malicious attack - Argos  72.89.186.108
3931  192.168.8.129 445  TEST  services.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-14 04:40:45.86
72.89.186.108:3931   192.168.8.129:445   TCP

A: 14-01-2008 06:53:39  Malicious attack - Argos  124.87.104.6
1402  192.168.8.129 445  TEST  lsass.exe
S: NETBIOS SMB-DS IPC$ unicode share access   2008-01-14 07:03:56.62
124.87.104.6:1402   192.168.8.129:445   TCP

A: 14-01-2008 14:29:13  Malicious attack - Argos  84.151.122.120
1807  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-14 14:39:33.513   84.151.122.120:1807
192.168.8.129:445   TCP

A: 14-01-2008 15:07:31  Malicious attack - Argos  192.168.7.149
48270  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 15:17:50.335
192.168.7.149:48270   192.168.8.129:135   TCP

A: 14-01-2008 15:23:47  Malicious attack - Argos  116.4.34.112
17665  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 15:34:03.19
116.4.34.112:17665   192.168.8.129:135   TCP

A: 14-01-2008 15:32:23  Malicious attack - Argos  66.228.18.20
4631  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-14 15:42:42.474   66.228.18.20:4631
192.168.8.129:445   TCP

A: 14-01-2008 16:01:17  Malicious attack - Argos  83.71.89.114
63825  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-14 16:11:36.92   83.71.89.114:63825   192.168.8.129:445
TCP

A: 14-01-2008 16:34:15  Malicious attack - Argos  192.168.7.149
60077  192.168.8.129 135  TEST  svchost.exe
S: DCOM Exploit (MS03-026) targeting Windows2000 SP0   2008-01-14
16:44:34.48   192.168.7.149:60077   192.168.8.129:135   TCP

A: 14-01-2008 16:50:40  Malicious attack - Argos  192.168.7.149
46596  192.168.8.129 135  TEST  svchost.exe
S: DCOM Exploit (MS03-026) targeting WindowsXP SP1   2008-01-14
```

```
17:00:59.277   192.168.7.149:46596   192.168.8.129:135   TCP

A: 14-01-2008 18:20:24  Malicious attack - Argos  89.244.255.237
3514  192.168.8.129 135  TEST
S:  Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14
18:30:41.427   89.244.255.237:3514   192.168.8.129:135   TCP

A: 14-01-2008 18:31:40  Malicious attack - Argos  41.221.16.208
62437  192.168.8.129 445  TEST
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-14 18:42:00.163   41.221.16.208:62437
192.168.8.129:445   TCP

A: 14-01-2008 19:01:46  Malicious attack - Argos  85.104.253.108
3763  192.168.8.129 445  TEST
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-14 19:12:05.93   85.104.253.108:3763   192.168.8.129:445
TCP

A: 14-01-2008 19:16:53  Malicious attack - Argos  81.152.77.3
3192  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 19:27:13.538
81.152.77.3:3192   192.168.8.129:135   TCP

A: 14-01-2008 19:37:29  Malicious attack - Argos  75.42.232.84
63835  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 19:47:48.579
75.42.232.84:63835   192.168.8.129:135   TCP

A: 14-01-2008 20:00:58  Malicious attack - Argos  213.151.234.234
2421  192.168.8.129 445  TEST
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-14 20:11:18.714   213.151.234.234:2421
192.168.8.129:445   TCP

A: 14-01-2008 20:38:17  Malicious attack - Argos  89.244.255.237
3633  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 20:48:37.404
89.244.255.237:3633   192.168.8.129:135   TCP

A: 14-01-2008 21:14:39  Malicious attack - Argos  64.223.62.105
50498  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-14 21:24:59.185   64.223.62.105:50498
192.168.8.129:445   TCP

A: 14-01-2008 21:59:37  Malicious attack - Argos  86.155.151.246
61855  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-14 22:09:58.273   86.155.151.246:61855
```

```
192.168.8.129:445   TCP

A: 14-01-2008 23:18:52  Malicious attack - Argos  212.84.122.247
1400  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-14 23:29:12.135
212.84.122.247:1400   192.168.8.129:135   TCP

A: 15-01-2008 00:45:23  Malicious attack - Argos  212.80.64.90
14855  192.168.8.129 445  TEST
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-15 00:55:42.983   212.80.64.90:14855
192.168.8.129:445   TCP

A: 15-01-2008 01:41:27  Malicious attack - Argos  96.229.5.225
1919  192.168.8.129 135  TEST  svchost.exe
Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 01:51:47.408
96.229.5.225:1919   192.168.8.129:135   TCP

A: 15-01-2008 06:14:21  Malicious attack - Argos  89.244.255.237
4563  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 06:24:42.399
89.244.255.237:4563   192.168.8.129:135   TCP

A: 15-01-2008 09:49:12  Malicious attack - Argos  79.187.16.50
3312  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-15 09:59:33.416   79.187.16.50:3312
192.168.8.129:445   TCP

A: 15-01-2008 10:34:44  Malicious attack - Argos  89.244.255.237
3343  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 10:45:04.854
89.244.255.237:3343   192.168.8.129:135   TCP

A: 15-01-2008 10:45:44  Malicious attack - Argos  91.154.29.68
1717  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-15 10:56:05.794
91.154.29.68:1717   192.168.8.129:139   TCP

A: 15-01-2008 10:53:19  Malicious attack - Argos  192.168.7.149
42957  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 11:03:40.094
192.168.7.149:42957   192.168.8.129:135   TCP

A: 15-01-2008 10:56:59  Malicious attack - Argos  192.168.7.149
60687  192.168.8.129 135  TEST  svchost.exe
S:  Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15
11:06:46.582   192.168.7.149:60687   192.168.8.129:135   TCP

A: 15-01-2008 11:41:33  Malicious attack - Argos  192.168.7.222
```

```
50553  192.168.8.129 135  TEST  svchost.exe
S: DCOM Exploit (MS03-026) targeting Windows2000 SP0   2008-01-15
11:51:54.54   192.168.7.222:50553   192.168.8.129:135   TCP

A: 15-01-2008 11:55:24  Malicious attack - Argos  192.168.7.149
54076  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 12:05:46.211
192.168.7.149:54076   192.168.8.129:135   TCP

A: 15-01-2008 12:01:38  Malicious attack - Argos  79.193.240.108
1382  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 12:12:00.07
79.193.240.108:1382   192.168.8.129:135   TCP

A: 15-01-2008 12:19:42  Malicious attack - Argos  79.15.87.183
2298  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-15 12:30:03.466
79.15.87.183:2298   192.168.8.129:139   TCP

A: 15-01-2008 12:28:41  Malicious attack - Argos  79.193.240.108
2897  192.168.8.129 135  TEST  svchost.exe
S:Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 12:39:02.827
79.193.240.108:2897   192.168.8.129:135   TCP

A: 15-01-2008 12:54:06  Malicious attack - Argos  87.161.212.83
2449  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 13:04:27.787
87.161.212.83:2449   192.168.8.129:135   TCP

A: 15-01-2008 13:06:54  Malicious attack - Argos  91.154.29.68
3911  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-15 13:17:15.822
91.154.29.68:3911   192.168.8.129:139   TCP

A: 15-01-2008 14:10:29  Malicious attack - Argos  192.168.7.222
59395  192.168.8.129 135  TEST  svchost.exe
S: Snort Alert [1:1100001:0]   2008-01-15 14:21:03
192.168.7.222:59395i   192.168.8.129:135   TCP
A: 15-01-2008 14:30:59  Malicious attack - Argos  192.168.7.222
42386  192.168.8.129 135  TEST  svchost.exe
S: Snort Alert [1:1101000:0]   2008-01-15 14:41:11
192.168.7.222:42386   192.168.8.129:135   TCP

A: 15-01-2008 14:35:15  Malicious attack - Argos  192.168.7.149
55489  192.168.8.129 135  TEST  svchost.exe
S: Snort Alert [1:1101000:0]   2008-01-15 14:45:35
192.168.7.149:55489   192.168.8.129:135   TCP

A: 15-01-2008 14:39:12  Malicious attack - Argos  192.168.7.222
46461  192.168.8.129 135  TEST  svchost.exe
```

```
S:  Snort Alert [1:1100001:0]   2008-01-15 14:49:33
192.168.7.222:46461   192.168.8.129:135   TCP

A: 15-01-2008 14:42:51  Malicious attack - Argos  192.168.7.222
33764  192.168.8.129 135  TEST  svchost.exe
S: Snort Alert [1:1100001:0]   2008-01-15 14:53:11
192.168.7.222:33764   192.168.8.129:135   TCP

A: 15-01-2008 14:47:43  Malicious attack - Argos  192.168.7.222
52576  192.168.8.129 135  TEST  svchost.exe
S:  Snort Alert [1:1100001:0]   2008-01-15 14:58:03
192.168.7.222:52576   192.168.8.129:135   TCP

A: 15-01-2008 14:51:15  Malicious attack - Argos  192.168.7.222
58949  192.168.8.129 135  TEST  svchost.exe
S: Snort Alert [1:1100001:0]   2008-01-15 15:01:37
192.168.7.222:58949   192.168.8.129:135   TCP


A: 15-01-2008 16:15:42  Malicious attack - Argos  192.168.7.149
49507  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 16:26:04
192.168.7.149:49507   192.168.8.129:135   TCP


A: 15-01-2008 16:28:08  Malicious attack - Argos  192.168.7.222
35195  192.168.8.129 135  TEST  svchost.exe
S: DCOM Exploit (MS03-026) targeting Windows 2000 SP0   2008-01-15
16:38:29   192.168.7.222:35195   192.168.8.129:135   TCP

A: 15-01-2008 16:48:35  Malicious attack - Argos  89.122.56.4
61590  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 16:58:57
89.122.56.4:61590   192.168.8.129:135   TCP

A: 15-01-2008 16:58:49  Malicious attack - Argos  121.119.76.193
4778  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-15 17:09:11
121.119.76.193:4778   192.168.8.129:139   TCP

A: 15-01-2008 17:34:14  Malicious attack - Argos  116.30.147.128
4367  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 17:44:34
116.30.147.128:4367   192.168.8.129:135   TCP

A: 15-01-2008 17:47:27  Malicious attack - Argos  83.36.149.69
1191  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 17:57:49
83.36.149.69:1191   192.168.8.129:135   TCP
```

```
A: 15-01-2008 19:02:38  Malicious attack - Argos  88.122.11.130
3045  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-15 19:12:59   88.122.11.130:3045   192.168.8.129:445
TCP

A: 15-01-2008 19:06:57  Malicious attack - Argos  87.210.214.249
1893  192.168.8.129 445  TEST  lsass.exe
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (Win2k)
2008-01-15 19:17:19   87.210.214.249:1893   192.168.8.129:445
TCP

A: 15-01-2008 19:16:49  Malicious attack - Argos  87.209.193.86
1437  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 19:27:11
87.209.193.86:1437   192.168.8.129:135   TCP

A: 15-01-2008 19:28:49  Malicious attack - Argos  83.36.149.69
4286  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 19:39:11
83.36.149.69:4286   192.168.8.129:135   TCP

A: 15-01-2008 22:53:42  Malicious attack - Argos  83.135.140.187
4935  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 23:04:04
83.135.140.187:4935   192.168.8.129:135   TCP

A: 15-01-2008 23:32:35  Malicious attack - Argos  83.135.140.187
2480  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-15 23:42:54
83.135.140.187:2480   192.168.8.129:135   TCP

A: 16-01-2008 03:28:20  Malicious attack - Argos  70.226.173.146
56667  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 03:38:43
70.226.173.146:56667   192.168.8.129:139   TCP

A: 16-01-2008 04:34:30  Malicious attack - Argos  60.40.195.117
1847  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 04:44:53
60.40.195.117:1847   192.168.8.129:135   TCP

A: 16-01-2008 04:45:56  Malicious attack - Argos  70.226.173.146
49737  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 04:56:18
70.226.173.146:49737   192.168.8.129:139   TCP

A: 16-01-2008 06:18:12  Malicious attack - Argos  70.226.173.146
62193  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 06:28:35
```

```
70.226.173.146:62193   192.168.8.129:139   TCP

A: 16-01-2008 06:45:45  Malicious attack - Argos  70.226.173.146
63991  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 06:56:07
70.226.173.146:63991   192.168.8.129:139   TCP

A: 16-01-2008 07:12:43  Malicious attack - Argos  70.226.173.146
64250  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 07:23:05
70.226.173.146:64250   192.168.8.129:139   TCP

A: 16-01-2008 07:25:31  Malicious attack - Argos  89.139.111.141
2815  192.168.8.129 445  TEST  dwDrvInst.exe
S: SHELLCODE x86 NOOP   2008-01-16 07:29:11   89.139.111.141:2815
192.168.8.129:445   TCP

A: 16-01-2008 08:00:53  Malicious attack - Argos  75.171.136.47
2595  192.168.8.129 445  TEST  services.exe
S: BLEEDING-EDGE EXPLOIT x86 PexFnstenvMov/Sub Encoder   2008-01-16
08:11:16   75.171.136.47:2595   192.168.8.129:445   TCP

A: 16-01-2008 10:26:06  Malicious attack - Argos  91.11.247.92
2250  192.168.8.129 445  TEST  lsass.exe
S: SHELLCODE x86 0x90 unicode NOOP   2008-01-16 10:36:30
91.11.247.92:2250   192.168.8.129:445   TCP

A: 16-01-2008 10:54:31  Malicious attack - Argos  211.59.110.72
61661  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 11:04:54
211.59.110.72:61661   192.168.8.129:139   TCP

A: 16-01-2008 11:52:48  Malicious attack - Argos  89.243.236.180
53407  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 12:03:11
89.243.236.180:53407   192.168.8.129:135   TCP

A: 16-01-2008 12:44:33  Malicious attack - Argos  12.25.184.29
14508  192.168.8.129 135  TEST  svchost.exe
S: SHELLCODE x86 NOOP   2008-01-16 12:54:56   12.25.184.29:14508
192.168.8.129:135   TCP

A: 16-01-2008 13:36:53  Malicious attack - Argos  70.226.173.146
53609  192.168.8.129 139  TEST
S: SHELLCODE x86 inc ebx NOOP   2008-01-16 13:47:16
70.226.173.146:53609   192.168.8.129:139   TCP

A: 16-01-2008 13:57:05  Malicious attack - Argos  89.139.111.141
4435  192.168.8.129 445  TEST  DWRCS.EXE
S: SHELLCODE x86 NOOP   2008-01-16 14:07:05   89.139.111.141:4435
```

```
192.168.8.129:445   TCP

A: 16-01-2008 14:24:20  Malicious attack - Argos  217.91.64.121
36985  192.168.8.129 135  TEST  svchost.exe
S: SHELLCODE x86 NOOP   2008-01-16 14:35:04   217.91.64.121:36985
192.168.8.129:135   TCP

A: 16-01-2008 14:43:17  Malicious attack - Argos  217.91.64.121
37000  192.168.8.129 135  TEST  svchost.exe
S: SHELLCODE x86 NOOP   2008-01-16 14:53:58   217.91.64.121:37000
192.168.8.129:135   TCP

A: 16-01-2008 15:03:17  Malicious attack - Argos  217.91.64.121
39390  192.168.8.129 135  TEST  svchost.exe
S: SHELLCODE x86 NOOP   2008-01-16 15:14:01   217.91.64.121:39390
192.168.8.129:135   TCP

A: 16-01-2008 16:25:35  Malicious attack - Argos  216.198.166.72
2301  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 16:36:19
216.198.166.72:2301   192.168.8.129:135   TCP

A: 16-01-2008 16:50:38  Malicious attack - Argos  86.68.240.14
2869  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 17:01:22
86.68.240.14:2869   192.168.8.129:135   TCP

A: 16-01-2008 19:24:35  Malicious attack - Argos  201.151.199.49
7041  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 19:35:19
201.151.199.49:7041   192.168.8.129:135   TCP

A: 16-01-2008 20:02:37  Malicious attack - Argos  61.218.45.185
63079  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 20:13:20
61.218.45.185:63079   192.168.8.129:135   TCP

A: 16-01-2008 20:15:55  Malicious attack - Argos  200.83.129.218
61813  192.168.8.129 445  TEST
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-16 20:26:39   200.83.129.218:61813   192.168.8.129:445
TCP

A: 16-01-2008 21:58:27  Malicious attack - Argos  87.20.162.120
1284  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 22:09:11
87.20.162.120:1284   192.168.8.129:135   TCP

A: 16-01-2008 22:31:19  Malicious attack - Argos  62.197.168.78
3450  192.168.8.129 445  TEST  lsass.exe
```

```
S: BLEEDING-EDGE EXPLOIT LSA exploit   2008-01-16 22:42:03
62.197.168.78:3450   192.168.8.129:445   TCP

A: 16-01-2008 22:45:01  Malicious attack - Argos  91.11.230.170
2211  192.168.8.129 445  TEST  lsass.exe
S: SHELLCODE x86 0x90 unicode NOOP   2008-01-16 22:55:47
91.11.230.170:2211   192.168.8.129:445   TCP

A: 16-01-2008 23:20:54  Malicious attack - Argos  201.151.199.49
7809  192.168.8.129 135  TEST  svchost.exe
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-16 23:31:37
201.151.199.49:7809   192.168.8.129:135   TCP
```

A.2 Snort attacks

As Snort logs a lot more attacks than Argos does we list some of these attacks here. These attacks are the ones we find the most interesting for SURFnet. Snort is able to detect a lot more attacks than these listed here, but it's impossible to list all those attacks.

```
MS-SQL Worm propagation attempt   2008-01-17 14:32:44
218.64.237.219:1358   192.168.8.129:1434   UDP

BLEEDING-EDGE WORM Allaple ICMP Sweep Reply Inbound   2008-01-17
14:17:51   195.169.125.75   58.13.12.123   ICMP

BLEEDING-EDGE WORM Potential MySQL bot scanning for SQL server
2008-01-16 22:43:32   125.24.222.67:2320   192.168.8.129:3306
TCP

BLEEDING-EDGE SCAN Potential VNC Scan 5900-5920   2008-01-15
16:38:54   87.53.231.189:62439   192.168.8.129:5900   TCP

SNMP trap tcp   2008-01-14 11:53:32.27   192.168.7.222:54165
192.168.8.129:162   TCP
```

Results of Experiment 3

Sander Keemink, Michael van Kleij

January 29, 2008

1 Introduction

This document describes the results of the second experiment conducted for SURFnet in the "Implementing Snort into SURFids" project. The experiment itself is described in the following experiment setup document: [1]. We've chosen to conduct experiment 3 instead of experiment 2. The reason for this was that Nepenthes is installed on the tunnelserver and it would take to much time to set up a separate Nepenthes and Snort server.

2 Running the experiment

The work on this experiment began on Monday the 21^{st} of January. On this day we set up the experiment. During this setup a problem was encountered, the tunnel-server, on which Snort was supposed to be installed, ran Debian stable instead of Ubuntu 7.10 which we used in our previous experiment. Debian stable containes Snort version 2.3 and Ubuntu 7.10 containes Snort 2.7. These versions use another rule format and the version installed on the Debian machine wasn't able to use the preprocessors we had configured. The solution to this problem was downloading the source code of version 2.7 and compiling this. Because we use Barnyard it isn't necessary to give any configure options to the configure program. When Snort was compiled and ready to run Barnyard had to be installed. This program was also compiled from source, with the *–enable-postgres* option.

Because external connections to the database were blocked Barnyard wasn't able to connect to the database, but when the iptables rules were modified Barnyard was able to run.

2.1 Snort Rulesets

Soon after Snort was running it became clear that Snort didn't recognize all attacks. Specifically the NetDDE and Symantec AV attacks. Even though there are rules included in the default Snort rulesets Snort doesn't trigger on these attacks.

On the 23^{rd} of January we discovered that in the Snort configuration file the HOME_NET variable wasn't configured correctly. Because many rules only trigger when an attack is aimed at the homenet these rules didn't trigger. This solved the problems we have had with the NetDDE and Symantec AV attacks. See example 1 for the correct definition of the HOME_NET variable. It is important to specify *all* protected[1] IP ranges as the HOME_NET. Otherwise the problems as mentioned above will arise.

Example 1 snort.conf

```
var HOME_NET [192.168.3.0/24,192.168.8.0/24]
```

3 Results

During this experiment we've seen that the amount of attacks registered by Nepenthes is much higher than the amount of attacks registered by Argos. On top of that Nepenthes knows which type of attack it is when it classifies an attack as a malicious attack. Snort detects almost all the attacks that are reported by Nepenthes as malicious attack. Of the possible malicious attacks it doesn't register that much. This is not a problem because Nepenthes

[1] Protected IP ranges are the ranges that you want to protect. In most cases these will be all the IP ranges on your network

Date	Nep.	Snort	Overlap	Perct.
22-01-2008	55	52	24	43.6%
23-01-2008	88	230	62	70%
24-01-2008	52	840	52	100%
25-01-2008	28	175	28	100%

Table 1: Results experiment 3

registers every connection on an open port as possible malicious. More about this is discussed in section 3.1 and section 3.3. In table 1 you can see the attacks registered by Nepenthes and Snort.

Since we've corrected the configuration error the overlap has grown significantly. At this moment we've got an overlap of above 90%. The same as the Argos situation. We don't expect it to be 100% on a long interval, but on occasion a 100% overlap must be possible on a given day.

The same as with experiment 1[2] Snort registers some attacks that aren't registered by Nepenthes. These attacks are primarily SQL attacks, but we've also seen some attacks on port 137 which contained shellcode. The name of this attack is not known, however, when you read the references Snort gives you can find out more about this attack. Log 1 shows an alarm from Snort on this attack.

Log 1 Attack on port 137

```
SHELLCODE x86 NOOP   2008-01-23 20:09:22
192.168.98.225:3768  192.168.8.129:137
UDP
```

We've also seen some attacks being registered by Nepenthes as a possible malicious attack. While most of these possible malicious attacks are not registered by Snort, some are.

Snort registered the following attacks. These attacks were either registered by Nepenthes as possible malicious, or not at all.

1. NETBIOS DCERPC NCACN-IP-TCP IActivation remoteactivation overflow attempt on port 135
2. MS-SQL Worm propagation attempt on port 1434
3. IMAP MDaemon authentication overflow single packet attempt on port 143
4. WEB-IIS w3who.dll buffer overflow attempt on port 80
5. FTP USER overflow attempt on port 21
6. SHELLCODE x86 NOOP on UDP port 137
7. Back Orifice Snort Buffer Attack on port 9080
8. WEB-IIS nsiislog.dll access on port 80
9. WEB-MISC PCT Client_Hello overflow attempt on port 443
10. XML-RPC for PHP Remote Code Injection on port 80
11. Sentinel LM attack on UDP port 5093

Of these attacks numbers 1, 3, 4, 5, 8, 9 and 10 are recognized by Nepenthes as possible malicious attacks. The other attacks are not registered by Nepenthes.

3.1 Metasploit

SURFnet asked us if Snort is able to detect attacks that Nepenthes doesn't register. Nepenthes registers all incoming connections on an open port as possible malicious attacks (see example 2). This has been confirmed by making telnet connections to those open ports, all these connections are registered as possible malicious. If an attack is made on an open port and Nepenthes is able to recognize this attack the attack is registered as a malicious attack. On closed ports Nepenthes never registers anything because it doesn't simulate a known vulnerability for it.

Snort is able to detect and analyze attacks to closed ports, however in some cases bidirectional traffic is required to detect an attack. Bidirectional traffic only occurs when the attacked host answers on the attack, and thus only when a service runs on the correct port. Therefore Snort won't detect all attacks on closed ports.

To get a better insight if Snort is complementary to Nepenthes we used Metasploit [2] [?] to run twenty known attacks. These are mostly attacks that we didn't register during the experimental period with either Snort or Nepenthes.

[2] Metasploit is a framework for running known attacks.

	Total	*Poss.*	*Mal.*	*Snort*
Closed port	10	0	0	3
Open port	10	9	1	5

Table 2: Summary of the Metasploit attack results

The results of the experiment can be seen in table 5 in appendix B, a summary of this table is table 2. For the complete Snort and Nepenthes logs on the attacks see appendix B.1. The table consists of the Metasploit name for the attack, whether the attack was on an open or closed port, the registration of Nepenthes and Snort. Nepenthes always registers a connection on an open port as malicious. Only when Nepenthes recognizes the attack will the attack be registered as a malicious attack. When an attack is made on a closed port Nepenthes won't register this attack. Snort registers an attack only if that attack triggered a rule, sometimes an attack can trigger multiple rules. In table 5 we show the amount different rules triggered.

Log 2 Ping averages with Snort

```
30 packets transmitted, 30 received,
0% packet loss, time 29001ms
rtt min/avg/max/mdev =
5.842/7.911/20.058/3.242 ms

30 packets transmitted, 30 received,
0% packet loss, time 29001ms
rtt min/avg/max/mdev =
5.890/8.675/42.962/6.605 ms

30 packets transmitted, 30 received,
0% packet loss, time 28998ms
rtt min/avg/max/mdev =
5.797/8.178/16.167/3.005 ms

30 packets transmitted, 30 received,
0% packet loss, time 28999ms
rtt min/avg/max/mdev =
5.840/7.352/16.813/2.303 ms
```

3.2 Performance

Due to circumstances we have conducted experiment 3 instead of experiment 2. In our project setup of experiment 3 [1] we described that we expect the performance of experiment 3 to be about the same as experiment 1. This because Snort sits in between the honeypot and the tunnelserver even though we installed Snort on the tunnelserver on which Nepenthes is also installed. The performance in ping replies can be seen in log 2.

As can be seen the average ping reply in this setup is 8.029ms. In experiment 1 this was 8.711ms. The difference between the two experiments is 0.682ms. The difference between these two is so small that it could also be a measurement error. On the other hand however, this difference is almost half of the difference between SURFids with Snort and SURFids without Snort.

3.3 Informational and Added Value

One of the big questions when conducting these experiments is whether this setup provides additional information to the current setup and if this information adds any value to it. This question is answered by looking at the results and comparing these results to the setup normally used.

As we've already said in section 3 Snort detects some attacks which aren't recognized by Nepenthes. This information is, in our opinion, valuable to the system. Attacks which are normally not registered, or registered as possible malicious attack by Nepenthes can be detected by Snort. This causes the system to detect more attacks than it would normally do.

Unlike experiment 1 [2] the results on which both Snort and Nepenthes trigger don't give any added value to the system. Nepenthes provides sufficient information when it detects an attack. The information Snort provides in these attacks is the same as the information Nepenthes provides. Therefore only attacks that are recognized as possible malicious or attacks that aren't registered at all are interesting. Snort is able to verify whether a possible malicious attack is in fact a real malicious attack. The same goes for attacks that aren't recognized at all by Nepenthes. Table 3 shows the amount of possible malicious attacks that are recognized by Nepenthes and the amount of those that are verified by Snort.

Nepenthes reports every connection on it's ports as a possible malicious attack. If Snort triggers on

Date	Nep.	Snort	perct.
22-01-2008	15	1	6.66%
23-01-2008	13	1	7.69%
24-01-2008	14	2	14.28%
25-01-2008	10	1	10%

Table 3: Possible malicious attacks recognized by Snort

one of these attacks we're sure it really is a malicious attack. An example of this *sensitivity* for attacks by Nepenthes is demonstrated in example 2. Snort analyzes the network packets and matches them to the used rulesets. These rulesets are designed to only trigger on real exploits. Network packets which are not extraordinary won't trigger an alarm. The example in example 2 won't trigger a Snort alarm because it's 'normal' network traffic. When Snort *does* trigger on a possible malicious attack, it is most likely that this possible malicious attack is a real malicious attack.

Example 2 Sensitivity of Nepenthes

```
tempest@marath:~$ telnet 192.168.8.129 80
Trying 192.168.8.129...
Connected to 192.168.8.129.
Escape character is '^]'.
GET index.html
Connection closed by foreign host.

25-01-2008 11:32:36 Possible malicious attack
192.168.7.222 40398 192.168.8.129 80  TEST
```

4 Conclusion

We believe this setup, whether it be with Snort installed on the tunnelserver or it being installed on a separate server with mirroring mode enabled on it's switchport, *will* have an added value to SURFids. This setup would be able to recognize attacks reported by Argos, and attacks which are reported by Nepenthes as possible malicious. Furthermore it is able to recognize attacks which aren't discovered by either Argos or Nepenthes.

References

[1] Michael van Kleij. Experiment 3: Snort on the tunnelserver. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

[2] Michael van Kleij & Sander Keemink. Results of experiment 1. https://www.os3.nl/2007-2008/students/michael_van_kleij/rp1-logbook, 2008.

A Results

As there are to many results to include in this document we've selected a subsection of all the results. There is an SQL dump available of all the results. This dump can be requested via e-mail. E-mail to sander.keemink@os3.nl or michael.vankleij@os3.nl.

A.1 Overlapping results

This section contains the attacks that were registered by Nepenthes as a malicious attack and of which Snort had a corresponding log. In table 4 the layout of these logs is explained.

Nepenthes:	date	attack	source IP	source port	dest. IP	dest. port	Sensor	info
Snort:	attack	date	source IP	source port	dest. IP	dest. port	protocol	

Table 4: Results layout

```
N: 22-01-2008 00:01:43  Malicious attack - Nepenthes  77.4.148.39
1237  192.168.8.129 445  TEST  LSASS
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (WinXP)
2008-01-22 00:01:43   77.4.148.39:1237   192.168.8.129:445
TCP

N: 22-01-2008 01:38:13  Malicious attack - Nepenthes  76.66.57.229
25870  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 01:38:13
76.66.57.229:25870   192.168.8.129:135   TCP

N: 22-01-2008 02:39:36  Malicious attack - Nepenthes  70.131.129.167
53670  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-22 02:39:45   70.131.129.167:53670   192.168.8.129:445
TCP

N: 22-01-2008 04:06:27  Malicious attack - Nepenthes  70.131.129.167
59343  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-22 04:06:25   70.131.129.167:58855   192.168.8.129:445
TCP

N: 22-01-2008 09:14:38  Malicious attack - Nepenthes  77.133.16.33
1239  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 09:14:38
77.133.16.33:1239   192.168.8.129:135   TCP

N: 22-01-2008 09:43:59  Malicious attack - Nepenthes  86.68.28.27
2843  192.168.8.129 445  TEST  LSASS
S: BLEEDING-EDGE EXPLOIT MS04011 Lsasrv.dll RPC exploit (WinXP)
2008-01-22 09:43:59   86.68.28.27:2843   192.168.8.129:445
TCP
```

```
N: 22-01-2008 10:41:38  Malicious attack - Nepenthes  220.97.211.153
4090  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 10:41:38
220.97.211.153:4090   192.168.8.129:135   TCP

N: 22-01-2008 18:21:55  Malicious attack - Nepenthes  85.60.158.104
2358  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 18:21:54
85.60.158.104:2358   192.168.8.129:135   TCP

N: 22-01-2008 18:25:14  Malicious attack - Nepenthes  77.45.194.213
2066  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 18:25:14
77.45.194.213:2066   192.168.8.129:135   TCP

N: 22-01-2008 18:31:50  Malicious attack - Nepenthes  90.194.6.167
1699  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 18:31:50
90.194.6.167:1699   192.168.8.129:135   TCP

N: 22-01-2008 18:33:19  Malicious attack - Nepenthes  91.155.203.228
2964  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-22 18:33:19
91.155.203.228:2964   192.168.8.129:135   TCP

N: 23-01-2008 06:09:15  Malicious attack - Nepenthes  70.131.129.167
54795  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-23 06:09:25   70.131.129.167:54795   192.168.8.129:445
TCP

N: 23-01-2008 06:37:54  Malicious attack - Nepenthes  80.53.15.251
3476  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-23 06:37:53
80.53.15.251:3476   192.168.8.129:135   TCP

N: 23-01-2008 10:39:33  Malicious attack - Nepenthes  70.131.129.167
55181  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-23 10:39:42   70.131.129.167:55181   192.168.8.129:445
TCP

N: 23-01-2008 10:49:57  Malicious attack - Nepenthes  91.195.96.229
61259  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-23 10:49:57
91.195.96.229:61259   192.168.8.129:135   TCP

N: 23-01-2008 16:57:21  Malicious attack - Nepenthes  66.153.237.204
50314  192.168.8.129 2967  TEST  Symantec AV
```

```
S: EXPLOIT symantec antivirus realtime virusscan overflow attempt
2008-01-23 16:57:21   66.153.237.204:50314   192.168.8.129:2967
TCP

N: 23-01-2008 17:02:53  Malicious attack - Nepenthes  70.131.129.167
50585  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-23 17:03:02   70.131.129.167:50585   192.168.8.129:445
TCP

N: 23-01-2008 17:19:49  Malicious attack - Nepenthes  217.231.191.29
3722  192.168.8.129 135  TEST  DCOM
S:  NETBIOS DCERPC NCACN-IP-TCP ISystemActivator RemoteCreateInstance
little endian attempt   2008-01-23 17:19:48   217.231.191.29:3722
192.168.8.129:135   TCP

N: 23-01-2008 17:32:47  Malicious attack - Nepenthes  70.131.129.167
57015  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-23 17:32:56   70.131.129.167:57015   192.168.8.129:445
TCP

N: 23-01-2008 20:43:52  Malicious attack - Nepenthes  66.153.237.204
62709  192.168.8.129 2967  TEST  Symantec AV
S: EXPLOIT symantec antivirus realtime virusscan overflow attempt
2008-01-23 20:43:52   66.153.237.204:62709   192.168.8.129:2967
TCP

N: 23-01-2008 20:47:12  Malicious attack - Nepenthes  190.128.124.208
3810  192.168.8.129 135  TEST  DCOM
S: Possible dcom*.c EXPLOIT ATTEMPT to 135-139   2008-01-23 20:47:11
190.128.124.208:3810   192.168.8.129:135   TCP

N: 23-01-2008 20:59:22  Malicious attack - Nepenthes  79.22.244.181
1410  192.168.8.129 139  TEST  NetDDE
S: SHELLCODE x86 inc ebx NOOP   2008-01-23 20:59:28
79.22.244.181:1410   192.168.8.129:139   TCP

N: 23-01-2008 21:10:00  Malicious attack - Nepenthes  190.67.236.122
3302  192.168.8.129 2967  TEST  Symantec AV
S: EXPLOIT symantec antivirus realtime virusscan overflow attempt
2008-01-23 21:10:00   190.67.236.122:3302   192.168.8.129:2967
TCP

N: 23-01-2008 21:35:27  Malicious attack - Nepenthes  70.131.129.167
52062  192.168.8.129 445  TEST  ASN1
S: BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
2008-01-23 21:35:36   70.131.129.167:52062   192.168.8.129:445
TCP
```

```
N: 23-01-2008 21:36:37  Malicious attack - Nepenthes  190.67.236.122
3836  192.168.8.129 2967  TEST  Symantec AV
S: EXPLOIT symantec antivirus realtime virusscan overflow attempt
2008-01-23 21:36:36   190.67.236.122:3836   192.168.8.129:2967
TCP

N: 23-01-2008 23:12:26  Malicious attack - Nepenthes  79.22.244.181
3634  192.168.8.129 139  TEST  NetDDE
S: SHELLCODE x86 inc ebx NOOP   2008-01-23 23:12:27
79.22.244.181:3634   192.168.8.129:139   TCP

N: 23-01-2008 23:24:52  Malicious attack - Nepenthes  200.40.49.222
3980  192.168.8.129 2967  TEST  Symantec AV
S: EXPLOIT symantec antivirus realtime virusscan overflow attempt
2008-01-23 23:24:52   200.40.49.222:3980   192.168.8.129:2967
TCP
```

B Metasploit

Attack name	*Open port*	*Nepenthes*	*Snort*
Microsoft RPC DCOM attack	Yes	Possible	1
Microsoft Message Queueing Service Path Overflow	Yes	Possible	0
Microsoft IIS 5.0 Printer Host Header Overflow	Yes	Malicious	4
Microsoft IIS ISAPI nsiislog.dll ISAPI POST Overflow	Yes	Possible	1
Microsoft SQL Server Resolution Overflow	No	0	1
Microsoft Plug and Play Service Overflow	Yes	Possible	0
Microsoft Private Communications Transport Overflow	Yes	Possible	1
HP OpenView Omniback II Command Execution	No	0	0
PHP XML-RPC Arbitrary Code Execution	Yes	Possible	2
Serv-U FTPD MDTM Overflow	Yes	Possible	1
Novell eDirectory NDS Server Host Header Overflow	No	0	0
Novell NetMail IMAP APPEND Buffer Overflow	Yes	Possible	0
Solaris sadmind Command Execution	No	0	2
AppleFileServer LoginExt PathName Overflow	No	0	0
Veritas Backup Exec Name Service Overflow	No	0	0
Hummingbird Connectivity LPD Buffer Overflow	No	0	0
Microsoft WINS Service Memory Overwrite	Yes	Possible	0
SentinelLM UDP Buffer Overflow	No	0	1
Sun Solaris Telnet Remote Auth. Bypass Vulnerability	No	0	0
RealServer Describe Buffer Overflow	No	0	0

Table 5: The results of the Metasploit attacks

B.1 Metasploit experiment results

Microsoft RPC DCOM attack
24-01-2008 13:25:59 Possible malicious attack 192.168.7.149 35370 192.168.8.129 135 TEST
NETBIOS DCERPC NCACN-IP-TCP IActivation remoteactivation overflow attempt 2008-01-24 13:25:58 192.168.7.149:35370 192.168.8.129:135 TCP

Microsoft Message Queueing Service Path Overflow
24-01-2008 13:31:17 Possible malicious attack 192.168.7.149 40923 192.168.8.129 2103 TEST

Microsoft IIS 5.0 Printer Host Header Overflow
24-01-2008 13:39:05 Malicious attack - Nepenthes 192.168.7.149 35336 192.168.8.129 80 TEST IIS
BLEEDING-EDGE WEB Proxy GET Request 2008-01-24 13:39:05 192.168.7.149:35336 192.168.8.129:80 TCP
BLEEDING-EDGE EXPLOIT x86 PexFnstenvMov/Sub Encoder 2008-01-24 13:39:05 192.168.7.149:35336 192.168.8.129:80 TCP
WEB-IIS ISAPI .printer access 2008-01-24 13:39:05 192.168.7.149:35336 192.168.8.129:80 TCP
COMMUNITY WEB-MISC mod_jrun overflow attempt 2008-01-24 13:39:05 192.168.7.149:35336 192.168.8.129:80 TCP

Microsoft IIS ISAPI nsiislog.dll ISAPI POST Overflow
24-01-2008 13:42:42 Possible malicious attack 192.168.7.149 51134 192.168.8.129 80 TEST
WEB-IIS nsiislog.dll access 2008-01-24 13:42:41 192.168.7.149:51134 192.168.8.129:80 TCP
http_inspect: BARE BYTE UNICODE ENCODING 2008-01-24 13:42:41 192.168.7.149:51134 192.168.8.129:80 TCP

Microsoft SQL Server Resolution Overflow
MS-SQL version overflow attempt 2008-01-24 13:47:57 192.168.7.149:33967 192.168.8.129:1434 UDP

Microsoft Plug and Play Service Overflow
24-01-2008 13:51:07 Possible malicious attack 192.168.7.149 53317 192.168.8.129 445 TEST

Microsoft Private Communications Transport Overflow
24-01-2008 13:55:21 Possible malicious attack 192.168.7.149 51935 192.168.8.129 443 TEST
WEB-MISC PCT Client_Hello overflow attempt 2008-01-24 13:55:21 192.168.7.149:51935 192.168.8.129:443 TCP

HP OpenView Omniback II Command Execution

PHP XML-RPC Arbitrary Code Execution
24-01-2008 14:02:55 Possible malicious attack 192.168.7.149 59008 192.168.8.129 80 TEST
WEB-PHP xmlrpc.php post attempt 2008-01-24 14:02:55 192.168.7.149:59008 192.168.8.129:80 TCP
BLEEDING-EDGE EXPLOIT XML-RPC for PHP Remote Code Injection 2008-01-24 14:02:55 192.168.7.149:59008 192.168.8.129:80 TCP
WEB-PHP xmlrpc.php post attempt 2008-01-24 14:02:55 192.168.7.149:59008 192.168.8.129:80 TCP
BLEEDING-EDGE EXPLOIT XML-RPC for PHP Remote Code Injection 2008-01-24 14:02:55 192.168.7.149:59008 192.168.8.129:80 TCP

Serv-U FTPD MDTM Overflow
24-01-2008 14:06:13 Possible malicious attack 192.168.7.149 50289 192.168.8.129 21 TEST

ftp_pp: FTP command channel encrypted 2008-01-24 14:06:12 192.168.7.149:50289 192.168.8.129:21 TCP

Novell eDirectory NDS Server Host Header Overflow

Novell NetMail IMAP APPEND Buffer Overflow
24-01-2008 14:09:31 Possible malicious attack 192.168.7.149 53663 192.168.8.129 143 TEST

Solaris sadmind Command Execution RPC portmap sadmind request UDP 2008-01-24 14:27:47 192.168.7.149:34021 192.168.8.129:111 UDP
RPC portmap Solaris sadmin port query udp request 2008-01-24 14:27:47 192.168.7.149:34021 192.168.8.129:111 UDP

AppleFileServer LoginExt PathName Overflow

Veritas Backup Exec Name Service Overflow

Hummingbird Connectivity 10 SP5 LPD Buffer Overflow

Microsoft WINS Service Memory Overwrite
24-01-2008 16:16:34 Possible malicious attack 192.168.7.149 50057 192.168.8.129 42 TEST

SentinelLM UDP Buffer Overflow
COMMUNITY EXPLOIT Sentinel LM exploit 2008-01-24 16:19:21 192.168.7.149:34028 192.168.8.129:5093 UDP

Sun Solaris Telnet Remote Authenticataion Bypass Vulnerability

RealServer Describe Buffer Overflow