Diploma Thesis

IDS on Raspberry Pi

A Performance Evaluation

Author: Andreas ASPERNÄS
Author: Thommy SIMONSSON
Supervisor: Oskar PETTERSSON
Examiner: Jacob LINDEHOFF
Term: VT2015
Subject: Computer Science
Level: G1E
Course code: 1DV41E

Abstract

This is a report on the possibility of using a Raspberry Pi as an intrusion detection system in a home environment to increase network security. The focus of this study was on how well two different generations of Raspberry Pi would be able to handle network traffic while acting as an intrusion detection system. To examine this a testing environment was set up containing two workstation computers connected to a Raspberry Pi, each computer hosting a virtual machine. Tests measuring the network throughput as well as the CPU and memory usage were performed on each of the Raspberry Pi devices. Two models of Raspberry Pis were used; Raspberry Pi model B+ and Raspberry Pi 2 model B; each of them running the operating system Arch Linux ARM. The results of these tests were that both of the Raspberry Pis could be used as an intrusion detection system but has some limitations that could impede usage depending on the requirements of the user. Raspberry Pi 2 model B show benefits of its updated hardware by suffering lower throughput degradation than Raspberry Pi model B+, while using less of it's total CPU and memory capacity.

Keywords: IDS, intrusion detection system, Raspberry Pi model B+, Raspberry Pi 2 model B, Arch Linux ARM, Snort, network, security, throughput.

Sammanfattning

Den här rapporten behandlar möjligheten att använda en Raspberry Pi som ett intrångdetekteringssystem i en hemma miljö för att öka nätverkssäkerheten. Fokusen i den här studien ligger på hur väl de två senaste generationerna av Raspberry Pi skulle kunna hantera nätverkstrafik samtidigt som den undersöker nätverkstrafiken och söker efter hot. För att kontrollera hur väl en Raspberry Pi kan fungera som ett intrångdetekteringssystem har en laborationsmiljö upprättats bestående av två fysiska maskiner som vardera används för att virtualisera en virtuell maskin. Tester för att mäta datagenomströmning, processor och minnesbelastning utfördes på var och en av Raspberry Pi. Två modeller av Raspberry Pi användes; Raspberry Pi model b+ och Raspberry Pi 2 model b, både körde operativsystemet Arch Linux ARM. Resultatet av testerna visade att det går att använda båda enheterna för att upprätta ett intrångdetekteringssystem, men det finns vissa begränsningar i enheterna vilket kan begränsa implementationsmöjligheterna. Raspberry Pi 2 model B uppvisade bättre resultat i form av att den är lägre belastad och har en högre datagenomströmning till skillnad från Raspberry Pi model B+. Raspberry Pi 2 model B har nyare och snabbare hårdvara vilket är den troliga orsaken till att den presterar bättre.

Nyckelord: IDS, intrångdetekteringssystem, Raspberry Pi model B+, Raspberry Pi 2 model B, Arch Linux ARM, Snort, nätverk, säkerhet, datagenomströmning.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Presentation of the problem . . . 1
- 1.3 Purpose and questions . . . 2
  - 1.3.1 Expected results . . . 2
- 1.4 Limitations . . . 2
- 1.5 Target audience . . . 2

2 Theory and technical background **4**
- 2.1 Intrusion detection system . . . 4
  - 2.1.1 Statistical Anomaly Detection . . . 4
  - 2.1.2 Rule Based Detection . . . 4
- 2.2 Different types of IDS . . . 4
- 2.3 Snort . . . 5
  - 2.3.1 Signatures . . . 5
  - 2.3.2 Rules . . . 5
- 2.4 Raspberry Pi . . . 5
  - 2.4.1 Raspberry Pi model B+ . . . 6
  - 2.4.2 Raspberry Pi 2 model B . . . 6
- 2.5 Arch Linux ARM . . . 6
- 2.6 Iperf . . . 7
- 2.7 Sar . . . 7

3 Method **8**
- 3.1 Scientific Approach . . . 8
- 3.2 Topology . . . 8
- 3.3 Measuring tools . . . 9
  - 3.3.1 Iperf . . . 9
  - 3.3.2 Sar . . . 9
- 3.4 Pre-study . . . 10
  - 3.4.1 Results . . . 10
  - 3.4.2 Analysis . . . 12
- 3.5 Experiments . . . 13
- 3.6 Method discussion . . . 13

4 Results **15**
- 4.1 Network Throughput . . . 15
- 4.2 CPU Load . . . 16
- 4.3 Memory Load . . . 17
- 4.4 Additional tests . . . 18

5 Analysis **19**
- 5.1 Observations . . . 19
- 5.2 Throughput . . . 19
- 5.3 Performance . . . 20
- 5.4 Raspberry Pi model B+ . . . 20
- 5.5 Raspberry Pi 2 model B . . . 20

6 Discussion **22**

7 Conclusion **23**
7.1 Recommendations for further research 24

A Snort installation on Arch Linux ARM **A**

B Snort rules from the thesis "IDS För Alla" **B**

C All available Snort rules **D**

D Hardware configuration **H**

E Software specifications **I**

1 Introduction

This chapter brings up the presentation of the problem, its background as well as the questions and hypothesis we seek to examine. It will further explain the purpose of this study, the confinements of that we are limited by and present the target audience for whom it is intended.

1.1 Background

The Internet is becoming increasingly dangerous; not only for large companies, but home networks are also being targeted by malicious activity [1]. As more devices in our homes are being connected to the Internet and more of our private data, such as photos, videos and financial information are stored digitally, our vulnerability against intrusions is increased[2]. If home network security is not taken seriously, we may become unnecessary exposed to malicious users that seek to destroy or exploit our digital lifestyle. Therefore it is necessary to take a closer look at how home users can become better protected. Home users may not be aware of the dangers on the Internet and therefore can not actively protect themselves against these dangers.

In order to effectively be able to protect a network from intrusion the user needs to be aware of that the network is under attack. An Intrusion Detection System (IDS) is a system that monitors the network in real time and detects malicious activity. An IDS could therefore be used to help home users protect their network from malicious activity. [3]. So why are not more people taking advantage of this useful tool? We believe the reason is because an IDS is often company grade and not designed to be used in home networks, it may be too difficult for home users to implement it as a security solution, as well as be too expensive [4].

Home users and small size companies could benefit from a small scale solution with a low implementation and maintenance cost. One possible solution is to run an intrusion detection software, for example Snort [4], on a Raspberry Pi. The device is affordable and flexible as it can run a number of operating systems and might therefore be a very suitable device to provide an entry level upgrade in network protection.

1.2 Presentation of the problem

In 2013 a study[5] was made on the possibility on using a Raspberry as an IDS in a home network. They performance tested a Raspberry Pi model B+ running the operating system IPFire and and the intrusion detection software Snort. The results from this study showed that a Raspberry Pi could be used as an IDS, but with the following problems:

- A limit on how many Snort-rules could be used due to limitations in memory.
- A noticeable degradation in throughput when Snort was active.

We want to examine the possibility of implementing an intrusion detection system, on a Raspberry Pi model B+, that does not suffer from such limitations, by using the operating system Arch Linux ARM which is lightweight and uses low system resources [6]. We also want to investigate if the Raspberry Pi 2 model B, together with Arch Linux ARM can even further improve the results compared to the Raspberry Pi model B+.

1.3 Purpose and questions

The main purpose of this study is to examine the effect on network performance when using a Raspberry Pi as an intrusion detection device. We wish to see how Raspberry Pi 2 model B compares to Raspberry Pi model B+ with its updated hardware and measure their capabilities to handle the network traffic while examining the traffic for intrusion. For the throughput to be within acceptable limits we set a requirement of no loss greater than 30%. This is intended as a point of reference used to evaluate and analyse the data gathered from our experiments. We will also measure the CPU and memory usage on the devices so that we may be able to detect potential limitations in the hardware.

The questions we are asking are:

- Can a Raspberry Pi be used as an intrusion detection system in a home network without lowering the network performance outside of acceptable limits?
- How would a Raspberry Pi 2 model B compare to Raspberry Pi model B+ in terms of throughput, CPU and memory performance?
- Is it possible to avoid the limitations mentioned in the previous study from 2013 [5] by implementing Arch Linux ARM?

1.3.1 Expected results

Arch Linux ARM is a minimalistic operating system with only the minimum required software, drivers and modules pre-installed [6], leaving the users to install only the software that is needed for their implementation. We expect that by using Arch Linux ARM on a Raspberry Pi model B+ more system resources can be used to forward traffic and by allocated by Snort. A Raspberry Pi 2 model B might yield even greater results because it has improved hardware in form of a faster CPU and twice the amount of memory, which could eliminate any bottlenecks found when using the the Raspberry Pi model B+.

1.4 Limitations

This study will only focus on the prospect of using a Raspberry Pi as an intrusion detection system using the software Snort. Neither the Snort software or any of its rules will be optimized to find out which rules impacts the throughput performance the most. We will not examine how effectively Snort detects malicious activity with different rule-sets, nor which kinds of traffic is the most difficult to process by Snort.

1.5 Target audience

This thesis is written for intermediate home users, with a deeper understanding for network traffic and security, who are looking for methods to increase their awareness regarding the traffic that is passing through their network and want to increase their network security. It could also be applicable for small companies or sole traders with their own business who want to increase their network security in order to protect their systems, but cannot afford to invest in an enterprise grade solution.

These two groups of users might not have access to a high speed Internet connection that may be limited by the throughput capacity of a Raspberry Pi, or require an IDS that does not suffer any downtime. The latter part is important to take in consideration when deciding on the placement of the IDS. If it is connected directly between the Internet service provider and the home network, any downtime might be problematic as the connection to the Internet will go down if the Raspberry Pi goes offline.

2 Theory and technical background

In this chapter we will introduce the software and hardware used in our study. This chapter can be read to get a better understanding of the technologies used and applied in order to better comprehend this study.

2.1 Intrusion detection system

An intrusion detection system (IDS) is a system that is used to detect malicious traffic and attacks against a network or a single host system. An IDS work by the theory that there is a difference between legitimate users and their network traffic from malicious users and their network traffic. An IDS uses two different methods to search for malicious activity; Statistical Anomaly Detection and Rule Based Detection [7, pp. 312-313].

2.1.1 Statistical Anomaly Detection

Statistical Anomaly Detection is based on user behavior and network traffic patterns, which is used to form an apprehension regarding the patterns of the network or a host system. This category can be further divided into two subcategories; Threshold detection and Profile Based detection [8]. Threshold detection defines a threshold for how many times a user may execute a certain action or how many times a certain type of network traffic traverse the network, if the users goes above this threshold an alert is sent. Profile based detection creates a baseline for normal behaviour and if a user or the network traffic diverges from this baseline to a high enough degree an alert is sent.[8]

2.1.2 Rule Based Detection

Rule based detection uses a set of predefined rules to identify an intruder or attack. This category can be divided further into two subcategories; Anomaly detection and Penetration identification [7, pp. 313]. Anomaly detection uses rules that are produced by looking at previous attack patterns or signatures of malicious traffic. Penetration identification uses an expert system containing rules written by security experts, that are used when searching for suspicious behavior in a network or on a host system. [7, pp. 313]

2.2 Different types of IDS

An IDS can be positioned in different locations within the network. It can be placed directly on an network device such as a router, a switch, a host system or it can be placed as its own dedicated device. When placed in the physical network it is known as a network based IDS. When it is placed on a host system, it is known as a host based IDS. The optimal placement for an IDS depends on the environment it is going to protect. In some cases there are more than one place that is optimal and in these special cases it is possible to implement more than one IDS to provide higher security. [8]

The network based IDS is used to find malicious traffic in a physical network. It can be placed to force data to flow through it or on a span port. The IDS will then process the

captured data that traverse the network. If it finds any data packets that is suspicious it will send an alert. The alert is written to a log file that can be read by administrators. This is often paired with a software that will scan the log-file for new alerts and report them to the administrator. [9] [10]

2.3 Snort

Snort is an network IDS. It is an open source software created by Martin Roesch. As Snort is an open source software it is supported on a number of operating systems. Snort is a versatile software as it is capable of analyzing traffic in real-time and can be used to perform a number of different functions such as traffic analyzing and packet sniffing. These functions can be use to detect and prevent attacks against the network [11]. Snort applies signatures and rules to find malicious network traffic.

2.3.1 Signatures

A packets signature is based on the content of the packet. There are three different kinds of signatures analyzed when using signature detection; string signature, port signature and header condition signature[10]. The signatures are then compare against signatures collected from previous attacks and malicious scenarios [12].

2.3.2 Rules

The rules used to detect malicious traffic or users does not require any previous knowledge about the attack as it can be defined by an administrator. Rules work by specifying certain actions as illegitimate and any action detected that matches any of those rules is flagged as an intrusion or attack. The foremost benefit of using rules over signatures is that rules require no data collected from previous attacks[13].

2.4 Raspberry Pi

The minimal computer known as the Raspberry Pi is an invention of four professors at the University of Cambridge in England. The four professors were; Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft. They noticed a decreasing skill and number of A Level students who wanted to study Computer Science. The four men thought that this was because home computers had become expensive and therefore parents would not let their children experiment on them. So they sought to find a solution that would allow the younger generation to experiment with computers at home. [14] [15]

In 2006 these four men began designing what would later become the very first Raspberry Pi. Together with Pete Lomas, MD of hardware design and manufacturing at Norcott Technologies, and David Braben, the co-author of BBC Micro Game Elite they formed the educational foundation named The Raspberry Pi foundation[14]. By 2011 the first version of the Raspberry Pi was available to public and in the first two years over 2 million units were sold [14]. In order to create a product with low cost they chose to use ARM processors for the Raspberry Pi rather than a x86 variant[16].

2.4.1 Raspberry Pi model B+

The Raspberry Pi model B+ is the fourth iteration and the latest version of the original Raspberry Pi. It was released for sale in July 2014, and was in February 2015 replaced by the new Raspberry Pi 2 b [17] [18]. The Raspberry Pi model B+ has the following specifications:

- A single-core ARM1176 processor running at 700MHz
- 512 Megabytes of memory
- Four USB-ports
- One 10/100 Megabit/s Ethernet-port
- One micro-sd card slot for storage

Compared to the earlier models, the Raspberry Pi model B+ has more USB-ports and they have replaced the old SD-card slot for a micro-SD card slot, which has a push-push locking mechanism. It also has a lower power consumptions compared to the earlier models [17] [19].

2.4.2 Raspberry Pi 2 model B

The Raspberry Pi 2 model B was released in February 2015 as the latest Raspberry Pi model with updated hardware. The Raspberry Pi 2 model B have the following specifications:

- A quad-core ARM Cortex-A7 CPU processor running at 900MHz
- 1024 Megabytes of memory
- Four USB-ports
- One 10/100 Megabit/s Ethernet-port
- One micro-SD card slot for storage

The Raspberry Pi 2 model B is faster and has twice the amount of memory compared to its predecessor, the Raspberry Pi model B+. It now has a quad-core processor which is speculated to make the Raspberry Pi 2 model B up to six time faster than the previous models [18] [20].

2.5 Arch Linux ARM

Arch Linux ARM is an ARM based Linux distribution[21] ported from the x86 based Linux distribution Arch Linux [22]. The Arch Linux philosophy is that users should be in total control over the operating system[6], which allows the users to implement it in any way they like. Therefore Arch Linux can be used for simple tasks and as well as more advanced scenarios. Arch Linux ARM is based directly on Arch Linux and they share almost all the code which makes Arch Linux ARM a very fast, Unix-like and flexible Linux distribution [6].

Arch Linux ARM has adapted the rolling-release update function from the x86-version. This means that small iterations is made available to the users as soon as they are ready, instead of the releasing larger updates every few months[21].

2.6 Iperf

Iperf is a tool used to measure the performance of a network. The developers NLANR and DAST created Iperf as a more modern alternative for measuring a networks functionality and throughput capacity [23]. Iperf can be used to measure the maximum bandwidth with both TCP and UDP traffic. It does also report delay jitter and datagram loss. Iperf is available for most modern operating system such as Windows, MacOS X and Linux/Unix[24].

Iperf is a client-server software where one side is a server and the other side is a client [24]. The two parts connects and network traffic is sent between them to measure the desired data. The method can be configured in a number of ways which provides great flexibility for the users [24].

2.7 Sar

Sar is a software for Linux that is used to gather information about the system by collecting information from different activity counters in the operating system. Sar gathers the information based on a count and an interval parameter. The count variable determines how often the information should be gathered. The interval parameter determines the duration. The data is written to a file, Sar adds the data collected and calculates an average from the data points collected [25].

3 Method

In this section of the thesis we will explain our scientific approach, the topology used in our experiments and the tools we used to collect the data. Furthermore, it will explain the execution of the experiments and discuss of the chosen method.

3.1 Scientific Approach

The approach for this report is inductive and empiric data will be collected from our tests. Data will be gathered by performing a number of experiments and it will be used as the foundation for this thesis. The data that will be measured is the CPU usage, memory usage and bandwidth throughput. An observation of the results will be made and it will then result in an analysis from which we will draw our conclusions.

3.2 Topology

The environment used in the experiments contains two Dell Precision T3500 machines, each one hosting a virtual machine. The virtual machines ran the operating system Ubuntu and had the throughput measuring software Iperf installed. For hardware specifications see appendix; D.

For each test a Raspberry Pi was connected between the physical machines with an Ethernet connection. See figure 1. As the Raspberry Pis' only have one physical Ethernet port, a Nintendo Wii USB-To-Ethernet adapter was used to enable them to connect to one of the virtual machines through its USB-port. Both Raspberry Pis ran the operating system Arch Linux ARM and had the intrusion detection software Snort installed. The configuration for Snort was identical on both Raspberry Pi and both had IPv4-forwarding enabled in the kernel [26, pp. 421-423]. See appendix; E for software specifications.

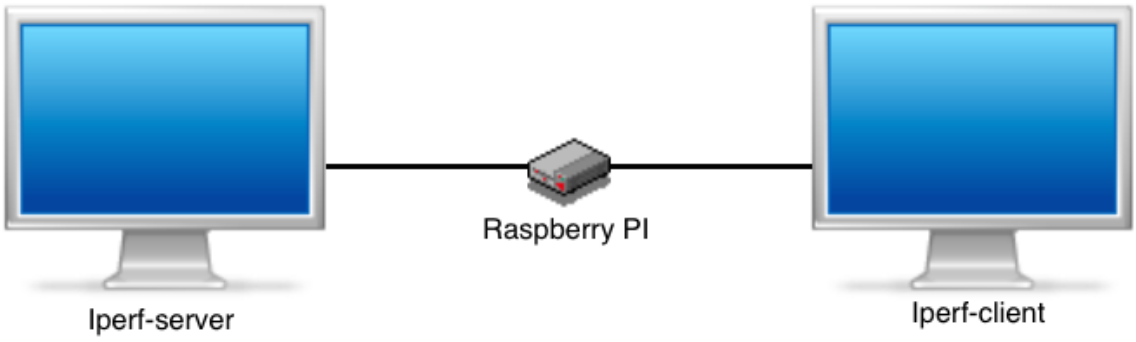

Figure 1: Topology

Images: [27], [28].

3.3 Measuring tools

In this section the software and measuring tools used in the experiments will be shown and explained. Furthermore the commands used to control the software and the tools will also be shown and explained.

3.3.1 Iperf

Iperf was the tool used to generate traffic to measure bandwidth capacity and throughput. We chose to use this tool because it gives us the ability to generate the traffic we need at will and provides it consistently. Iperf can be used to generate both TCP and UDP traffic between an Iperf server and an Iperf client.

On the server side the following commando was used:

Iperf -s

The command runs Iperf in server mode with port 5001 open. This makes it possible for an Iperf client to connect to the Iperf server on port 5001 and sends TCP-packets to the server [29].

On the client side the following command was used:

Iperf -c 192.168.1.10 -t 300

The command runs Iperf in client mode and thereby connects to the Iperf server on the default port 5001. It then sends TCP-packets for 300 seconds. We chose to use the default setting for how much data to be transferred at a time. The default is to send 85KB sized TCP-packets a number of times. As we chose 300 seconds it will send 85KB sized TCP-packets as many times as is possible for 300 seconds [29].

3.3.2 Sar

While performing the Iperf tests Sar will be used to gather information about the system load on the Raspberry Pis. IPerf will be running for 300 seconds and the gathering of the CPU-load will commence after 120 seconds.

Sar -u 1 150 > CPU_load.txt

The command above will collect information about the CPU-load once a second for 150 seconds and write the data to the file CPU_load.txt. The flag -u indicates that data regarding all processes will be gathered [25].

Sar -r 1 150 > MEM_usage.txt

The above command will collect information about the memory usage once a second for 150 seconds and write the data to the file MEM_usage.txt. The flag -r indicates that data regarding the internal memory usage shall be gathered. The memory usage measurement will start 120 seconds after Iperf is started [25].

Both the CPU and memory usage will be measured at the same time as the Iperf test is running. By starting the CPU and memory measuring 120 seconds after Iperf is started we will be guaranteed that the Raspberry Pis have reach the maximum system load.

3.4 Pre-study

In the beginning of this project we performed a pre-study with the goal of answering a number of questions. The purpose of these questions was not only to see if it was theoretically possible to conduct our experiments, but also if it would be practically feasible. The questions we wanted answers to were the following;

- Would it be possible to install Arch Linux ARM on a Raspberry Pi 1 and 2?
- Would it be possible to install Snort on Arch Linux ARM?
- Will Snort work with different rule-sets?
- Would it be possible to use an USB-to-Ethernet adapter with the Raspberry Pis?
- What is the bandwidth capacity of the internal NICs and the USB-to-Ethernet adapter?
- Is it possible to use Iperf to measure the throughput in our experiments?
- Would the data be reliable and can we draw any conclusions from them?

The study was conducted in three parts; the first part was to investigate the hardware and software, to install and configure all the necessary parts. The second part was to conduct tests and analyze them to see how to get more accurate results. The third part was to measure the bandwidth capacity of the USB-to-Ethernet adapter when connected to the Raspberry Pis and their internal NICs.

3.4.1 Results

These are our results from the pre-study. We conducted several throughput tests to try out different settings on Iperf. With Snort disabled we ran Iperf for 2, 5 and 10 minutes to see which would be best used in the final experiments and give the most reliable data. We also performed measurements of the USB-to-Ethernet adapter connected to the Raspberry Pis and their internal NICs.

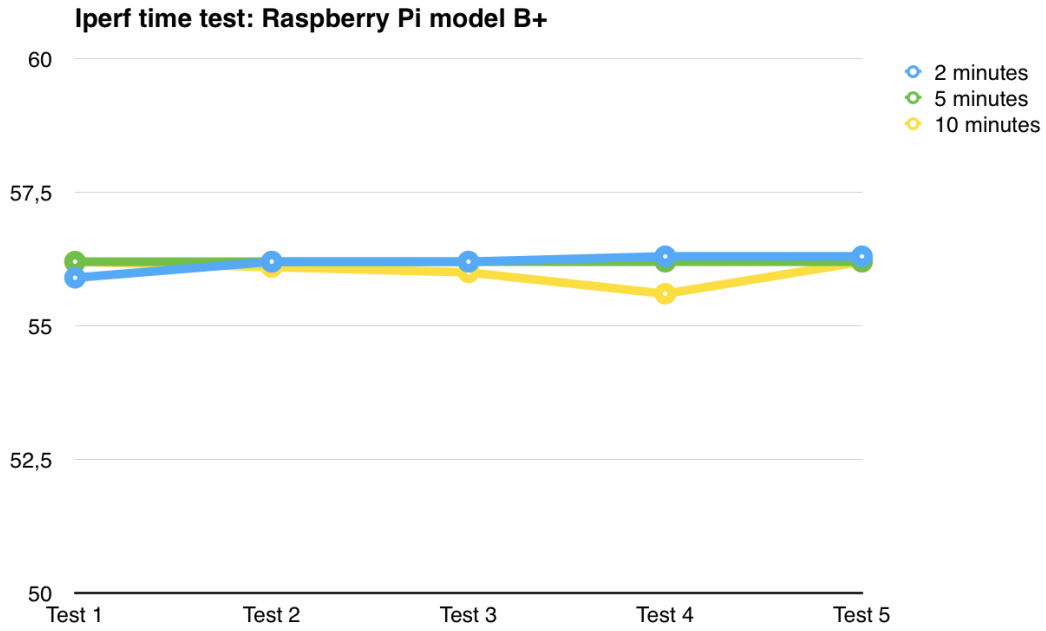

Figure 2: Raspberry Pi model B+

Figure 2 shows the measured throughput in our tests to determine for how long Iperf should be running in each final test. The Y-axis shows the throughput in Mbit/s and the X-axis shows which of the test that gave the specific result. For the 2 minute-tests the Raspberry Pi model B+ measured 55,9 Mbit/s on test 1, 56,2 Mbit/s on test 2, 56,2 Mbit/s on test 3, 56,3 Mbit/s on test 4 and 56,3 Mbit/s on test 5. For the 5 minute-tests the Raspberry Pi model B+ measured 56,2 Mbit/s on test 1, 56,2 Mbit/s on test 2, 56,2 Mbit/s on test 3, 56,2 Mbit/s on test 4 and 56,2 Mbit/s on test 5. For the 10 minute-tests the Raspberry Pi model B+ measured 56,2 Mbit/s on test 1, 56,1 Mbit/s on test 2, 56,0 Mbit/s on test 3, 55,6 Mbit/s on test 4 and 56,2 Mbit/s on test 5.

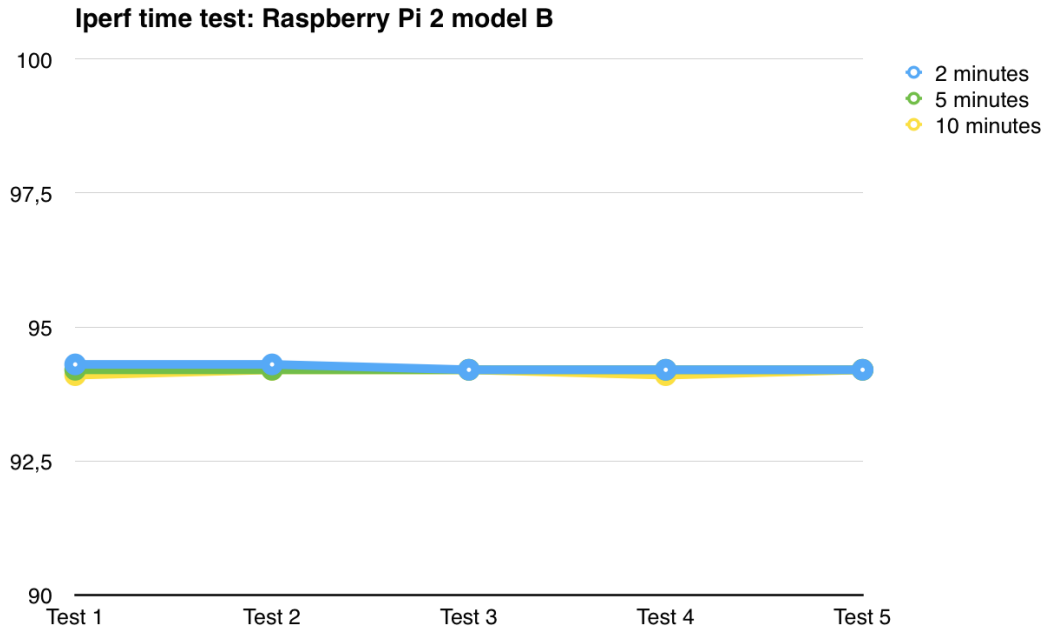

Figure 3: Raspberry Pi 2 model B

Figure 3 shows the measured throughput in our tests to determine for how long Iperf should be running in each final test. The Y-axis shows the throughput in Mbit/s and the X-axis shows which of the test that gave the specific result. For the 2 minute-tests the Raspberry Pi 2 model B measured 94,3 Mbit/s on test 1, 94,3 Mbit/s on test 2, 94,2 Mbit/s on test 3, 94,2 Mbit/s on test 4 and 94,2 Mbit/s on test 5. For the 5 minute-tests the Raspberry Pi 2 model B measured 94,2 Mbit/s on test 1, 94,2 Mbit/s on test 2, 94,2 Mbit/s on test 3, 94,2 Mbit/s on test 4 and 94,2 Mbit/s on test 5. For the 10 minute-tests the Raspberry Pi 2 model B measured 94,1 Mbit/s on test 1, 94,2 Mbit/s on test 2, 94,2 Mbit/s on test 3, 94,1 Mbit/s on test 4 and 94,2 Mbit/s on test 5.

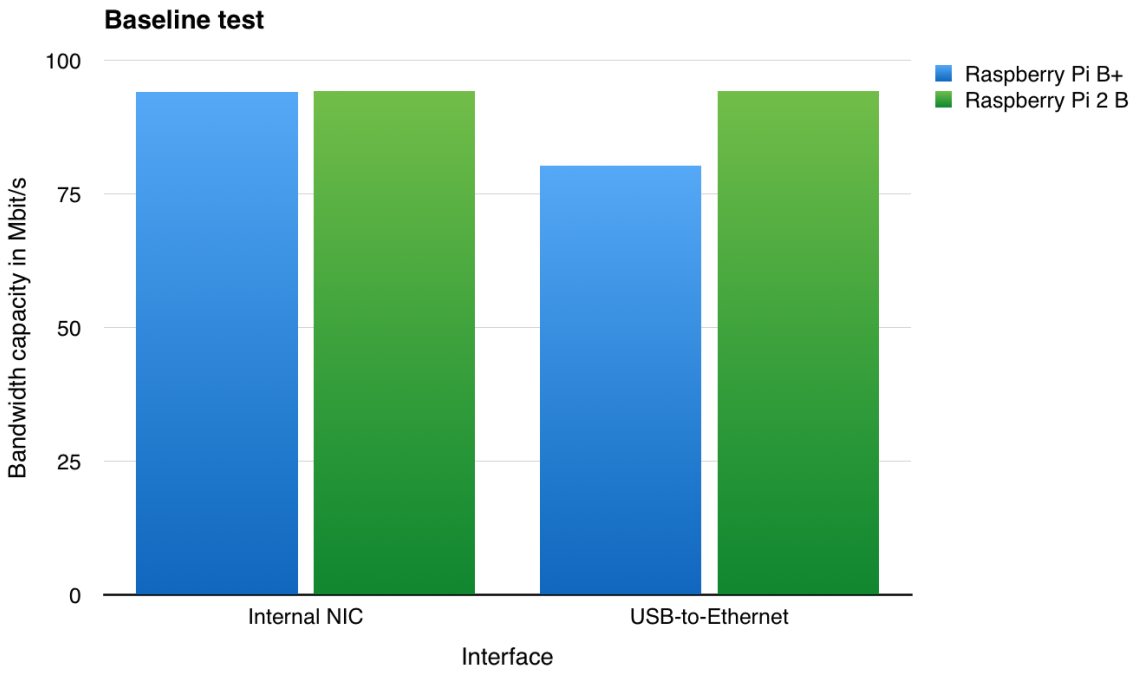

Figure 4: Bandwidth baseline

Figure 4 shows the measured bandwidth capacity in our tests to determine what the maximum bandwidth capacity for each internal NIC/USB-To-Ethernet is. The Y-axis shows the bandwidth capacity measured in Mbit/s and the X-axis shows which of the interfaces was tested. The Raspberry Pi model B+ internal NIC showed an average bandwidth capacity of 94,1 Mbit/s and the USB-to-Ethernet measured 80,28 Mbit/s of bandwidth. For the Raspberry Pi 2 model B the internal NIC measured an average bandwidth capacity of 94,2 Mbit/s and the USB-to-Ethernet showed a bandwidth capacity of 94,2 Mbit/s.

3.4.2 Analysis

The pre-study showed that we would be able to use the method we designed. It was possible to install Arch Linux ARM on a Raspberry Pi and to install Snort on Arch Linux ARM, even though there is no official package for Arch Linux ARM. Furthermore the chosen USB-To-Ethernet adapter worked with Arch Linux ARM, without the need to install any additional packages to get full functionality.

From the collected data we concluded that a 5 minute-test would be the most appropriate as the data showed that longer tests did not yield better results as can be seen in figure 2 and 3. The largest deviation between the overall slowest and fastest throughput being 0.3% for the Raspberry Pi model B+ and 0.2% for the Raspberry Pi 2 model B. In the 5 minute-tests the deviation was 0% for both the Raspberry Pi model B+ and the Raspberry Pi 2 model B.

To see if any of the interfaces could be a potential bottleneck during the throughput tests, we established a baseline for the maximum bandwidth capacity for each of the interfaces. The results show that no throughput could be higher than 80,28 Mbit/s on the Raspberry Pi model B+, as this was the lowest bandwidth capacity measured. The highest possible throughout on the Raspberry Pi 2 model B is 94,2 Mbit/s.

3.5 Experiments

The experiments is performed in six steps. During each step Iperf is used to measure the throughput and Sar is used to measure CPU and memory usage. Snort will be configured to use two different rule-sets, as well as being complete turned off. The first rule-set, rule set: A, will contain all the standard rules available from Snorts official website[30]. The second rule-set, rule-set: B, contains the rules used in the previous study[5] from 2013. The rule-sets for Snort are:

- Rule-set A: 24 204 rules, see appendix C.
- Rule-set B: 10 908 rules, see appendix B.

The experiments will be done in multiple steps. To improve the quality of the results, during each step the test will be conducted five times. Iperf will be running for 300 seconds each test. The performance measurements with Sar will start after 120 seconds and it will be running for 150 seconds. Each step in the experiment represents a given scenario that we wish to test. The experiments are going to be performed in the following steps:

- Step 1: test the Raspberry Pi model B+ using Snort with rule-set: A
- Step 2: test the Raspberry Pi model B+ using Snort with rule-set: B
- Step 3: test the Raspberry Pi model B+ with Snort disabled
- Step 4: test the Raspberry Pi 2 model B using Snort with rule-set: A
- Step 5: test the Raspberry Pi 2 model B using Snort with rule-set: B
- Step 6: test the Raspberry Pi 2 model B with Snort disabled

3.6 Method discussion

The reason for using two physical machines to host one virtual machine each is to ensure that the performance degradation are minimal. When running them on a dedicated physical machine they have access to two of the machines physical cores, 4 GB memory and a 1 Gbit/s network interface card. By using two identical physical machines to host the virtual machines we could eliminate any performance difference related to the physical hardware, the hyper-visor or the host operating system.

All the software and the operating systems used are stable versions. By avoiding beta versions we hope to eliminate any abnormal results that might be caused by errors in the software. Software versions that have been available for some time usually are better documented and therefore reduces the risk of misconfiguration.

Arch Linux ARM was chosen because the earlier study from 2013 [5] showed that memory usage was an issue while using the operating system IPFire. By using Arch Linux ARM, less memory is used by the operating system and thereby making more memory available for Snort. Arch Linux ARM applies an rolling-release cycle which means that it can be updated daily. It was important to make sure that once the installation of Arch Linux ARM was completed we made sure that it had no access to the Internet so that we could guarantee that no package would be updated by accident. The same principals

was used for the virtual Ubuntu clients, once Iperf was installed their connection to the Internet was removed.

As for installation of software on Arch Linux ARM, the packet manager Pacman had all of the software we needed except Snort. As there is no official Snort package for Arch Linux ARM we had to download, compile and install it manually. Therefore it is possible that a official Snort package for Arch Linux ARM could improve the performance of Snort. For Snort installation see appendix; A.

4 Results

In this section our results will be presented and explained. The data from our tests will be displayed along with graphs to give the reader a summarized overview and make it easier to correlate the data collected from the tests.

4.1 Network Throughput

This section will show the results of our throughput test between the Raspberry Pi model B+ and Raspberry Pi 2 model B.

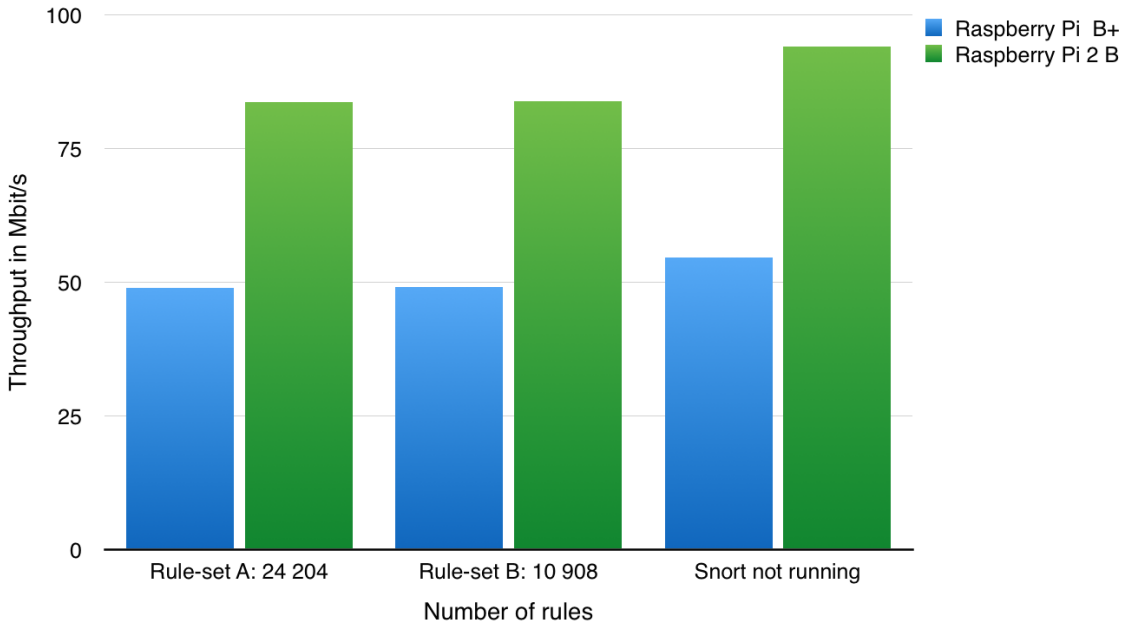

Figure 5: Throughput

The graph in figure 5 shows the average network throughput through each of the Raspberry Pi devices during different set of conditions.The Y-axis shows the throughput measured in Mbit/s and the X-axis displays how many Snort-rules was implemented as a set of conditions. With rule-set A implemented, the throughput of Raspberry Pi model B+ was measured to 48,86 Mbit/s and Raspberry Pi 2 model B to 83,58 Mbit/s. With rule-set B the Raspberry Pi model B+ measured to 49,1 Mbit/s and Raspberry Pi 2 model B to 83,8 Mbit/s. With Snort offline the throughput was measured to 54,7 Mbit/s for the Raspberry Pi model B+ and 94,02 Mbit/s for Raspberry Pi 2 model B.

4.2 CPU Load

This section will show the results of our CPU performance measurements while testing the throughput capacity on the Raspberry Pi 2 model B. The results from Raspberry Pi model B+ is not included as they were not reliable.

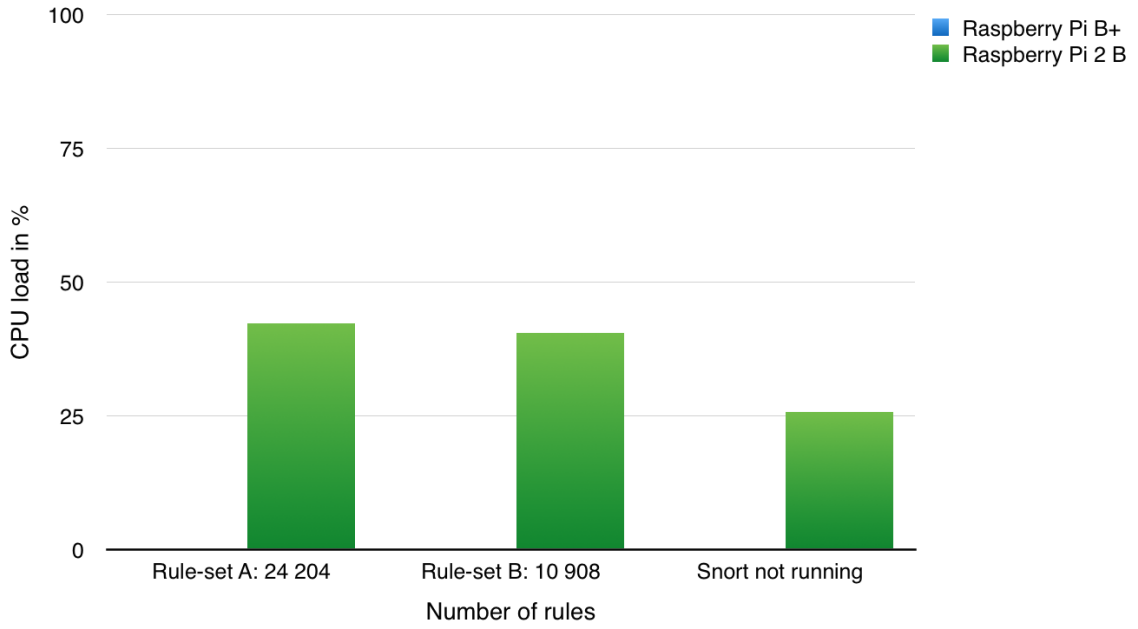

Figure 6: CPU

The graph in figure 6 shows the average CPU load during the throughput-test. Raspberry Pi model B+ has been excluded because of unreliable data. The Y-axis shows the CPU load of the Raspberry Pi device measured in percent and the X-axis shows the number of Snort-rules implemented during the tests. During throughput-test with rule-set A implemented on the Raspberry Pi 2 model B measured to 42,206% CPU load. With rule-set B implemented the CPU load measured to 40,45% on the Raspberry Pi 2 model B. With Snort offline and no rules implemented the CPU load on the Raspberry Pi 2 model B measured to 25,886%.

4.3 Memory Load

This section will show the results of our memory performance measurements while testing the throughput capacity on Raspberry Pi model B+ and Raspberry Pi 2 model B.

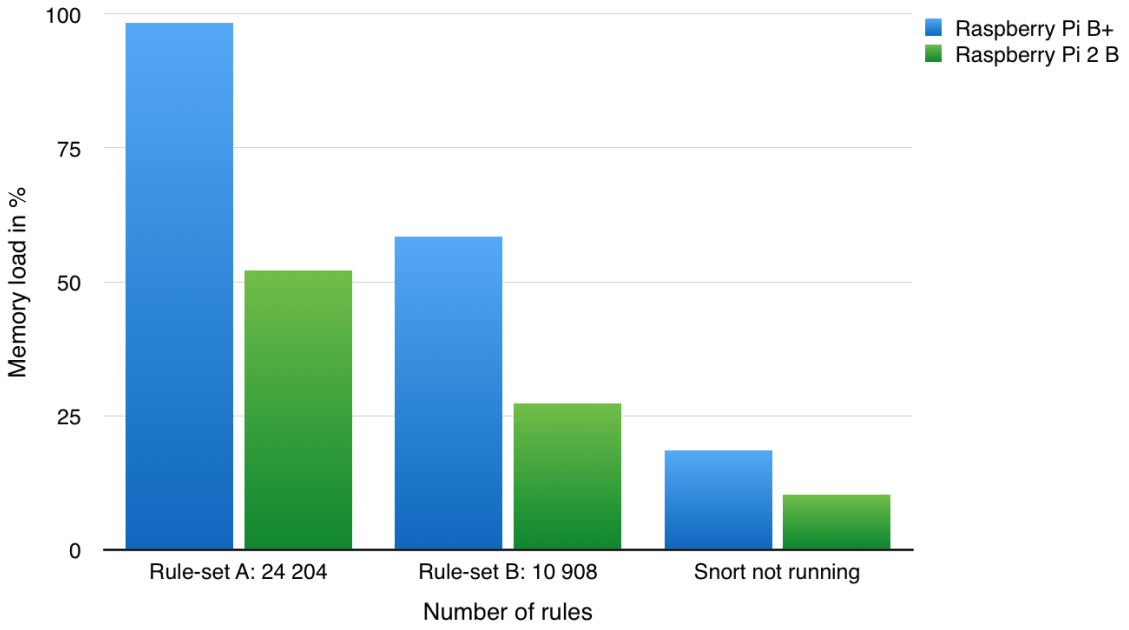

Figure 7: Memory

The graph in figure 7 displays the average memory load of each of the Raspberry Pi devices during the testing of the throughput, under three different sets of conditions. The Y-axis shows the memory load measured in percent and the X-axis shows the set of conditions with how many rules are implemented. With rule-set A implemented the memory load on Raspberry Pi model B+ was measured to 98,334% and 52,08% on the Raspberry Pi 2 model B. With rule-set B implemented the memory load was measured to 58,416% on the Raspberry Pi model B+ and 27,268% on the Raspberry Pi 2 model B. With no rules implemented and Snort offline the memory load on the Raspberry Pi model B+ was 18,5% and 10,172% on the Raspberry Pi 2 model B.

4.4 Additional tests

This section show the results of a further tests, made after the main experiments were concluded. These results show the memory usage on the Raspberry Pi model B+ with Iperf not running, compared to the results from the memory usage of Raspberry Pi model B+ with Iperf running, from figure 7.

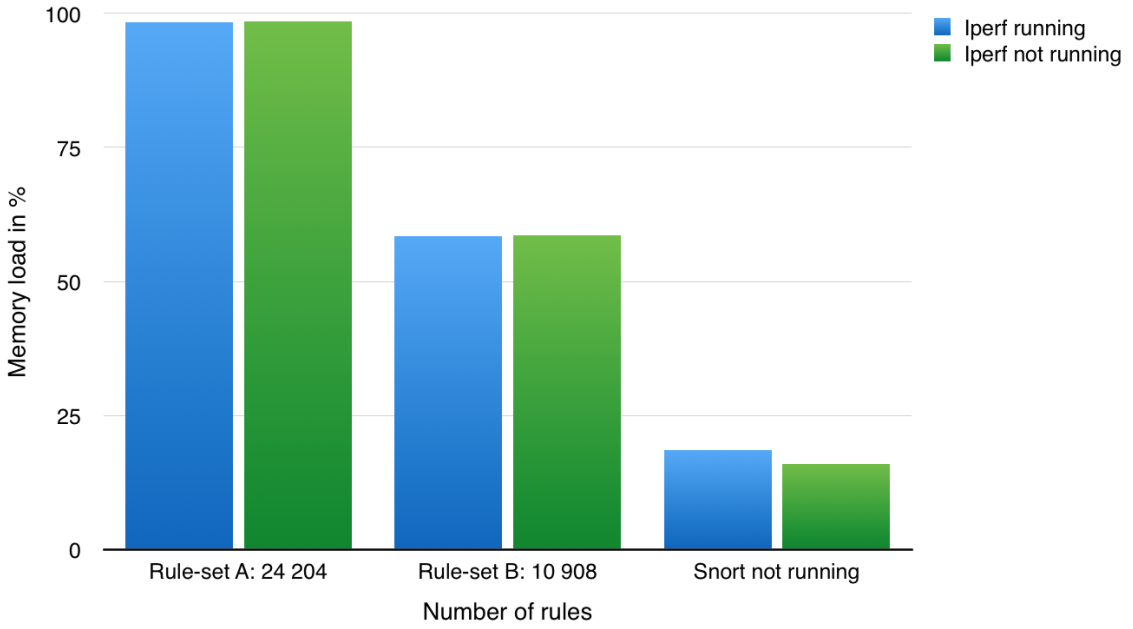

Figure 8: Memory on Raspberry Pi model B+

The graph in figure 8 show a memory comparison on the Raspberry Pi model B+ between memory used during and not during a network throughput test. The tests were under different conditions with two sets of Snort-rules. Y-axis shows the memory load measured in percent and the X-axis shows the different conditions the tests were made. During rule-set A the memory load was 98,344% with Iperf online and 98,568% with Iperf offline. With rule-set B the Raspberry Pi model B+ measured 58,416% with Iperf online and 58,632% with Iperf offline. With Snort offline the memory load on the Raspberry Pi model B+ was 18,5 with Iperf online and 15,876% with Iperf offline.

5 Analysis

In this chapter the data from our tests will be analyzed. The data can be divided into two parts; network throughput and system performance. Each of these will be analyzed in different sections, ending with an examination of the Raspberry Pi devices.

5.1 Observations

First an observation on the performance results. No CPU results could be presented from Raspberry Pi model B+ because it could not provide any viable data. The measuring tool is suppose to check the activity counter once every second, but the time stamps reveal it could be delayed up to 30 seconds or more, even continuing to measure after the throughput test has been completed. The collected data within the time frame does however show up to 99-100% CPU usage.

We see three possible reasons behind this. One is for the CPU to have been so busy forwarding packages that very little resources was given to check the CPU and memory load. Only after the throughout test was completed did the performance test check the CPU and memory load at regular intervals. Another reason might be because the Raspberry Pi model B+ has a single-core processor it can only handle one process at a time [31, pp. 396] [32, pp. 44-51]. The Raspberry Pi 2 model B on the other hand has a quad-core processor and can therefore handle four separate processes at the same time. One process could be used to forward packages, one to run Snort and another to run Sar. A third possibility is a combination of both of the above given reasons.

The same problem did occur when the measuring of memory on the Raspberry Pi model B+. However, some additional tests were made with a comparison of memory usage during a network test with no network traffic were being forwarded. The results showed that forwarding network traffic did almost not affect how much memory was used. Therefore, we believe the earlier presented data in section 4.3 to be valid.

It should also be noted that it is possible to see the impact Sar has on the throughput performance. Our tests in the pre-study showed that the throughput on the Raspberry Pi model B+ was 56,2 Mbit/s and in our final experiments, with Sar active we measured a throughput of 54,7 Mbit/s, a 1,5 Mbit/s drop. The pre-study results for the Raspberry Pi 2 model B measured 94,2 Mbit/s and in the final test with Sar active measured 94,02 Mbit/s, a 1,8 Mbit/s drop.

5.2 Throughput

The results from our baseline throughput tests shows that there is a difference in throughput between the Raspberry Pi model B+ and the Raspberry Pi 2 model B. The tests also shows that there is a throughput degradation when activating Snort, but once activated the number of Snort rules did little to change the network performance for any of the devices. It is unknown to us if Snort could be optimized to process traffic faster and therefore grant higher throughput as it it outside our confines, but we see it as a possibility.

5.3 Performance

The results from the performance testing show there is a difference in CPU and memory usage between the two Raspberry Pies. It seems that both the packet forwarding as well Snort have a impact on the performance of the device. Without definitive results we can not confirm the CPU usage on the Raspberry Pi model B+, but we can speculate that it would have been close to 100%. The CPU usage on the Raspberry Pi 2 model B was less than 50%, showing the difference in performance between the 700 MHz single-core processor on Raspberry Pi model B+ and the 900 MHz quad-core processor on the Raspberry Pi 2 model B.

Regarding the memory usage, when applying rule-set A the Raspberry Pi model B+ used almost 100% of its available memory compared to the Raspberry Pi 2 model B. The Raspberry Pi 2 model B used almost the same amount of memory as the Raspberry Pi model B+, but because it has access to twice the amount of memory compared to the Raspberry Pi model B+, it only uses about 50% of its available memory. The pattern continues when using rule-set B and when Snort is offline, which indicates that both devices consumed almost the same amount memory for each of the three rule-sets.

5.4 Raspberry Pi model B+

On the Raspberry Pi model B+ we noticed a drop in throughput, the lowest average throughout measured to 48,86 Mbit/s, thereby getting a 52,14% in drop in performance on a theoretical 100 Mbit/s connection. However, our baseline tests show that the maximum capacity is only 80,28 Mbit/s, therefore the drop is only 39%. Even with Snort disabled there was a degradation in throughput performance. This indicates that the limitations in hardware has the single biggest affect on performance. The attempts to measure the system load failed as the Raspberry Pi model B+ was not able to both forward traffic and execute the commands for Sar. This indicates that the forwarding of traffic, which is done by a kernel module[26, pp. 421-423], gets a higher priority than the system load measurements done by Sar. However, in an attempt to validate the memory usage data, additional tests were made. The purpose of these tests were to measure the memory load of Raspberry Pi model B+ when Iperf was offline, and therefore not overloading the processor so that valid data could be collected. These data could then be compared to the previous results in section 4.3. The results indicates that the throughput tests does little to affect the memory usage of the device. Although all of the rules in rule-set A could be applied, the memory usage measured over 98%, leaving less then 2% for anything else.

5.5 Raspberry Pi 2 model B

The throughput test with Snort disabled shows that the Raspberry Pi 2 model B is not able to forward traffic beyond 94,02 Mbit/s. The reason for this could be because the CPU not being fully utilized to forward traffic[26, pp. 421-423], or Sar allocating system resources while measuring system performance. However the maximum bandwidth capacity is 94,2 Mbit/s according to our baseline tests and cannot be exceeded without faster a NIC and USB-to-Ethernet. When activating Snort using rule-set A, we noticed a throughput degradation of 12,2% compared to the maximum throughput with Snort disabled. When applying rule-set B the throughput was 12,1% lower than the maximum

throughout. In total there was a 0.1% difference between the rule-sets tested. Neither of the tests with Snort activated generated a system load of 100%. At most the CPU utilization was at 42.2% and the maximum memory usage was 52.1%, which indicates that it would be possible to implement more Snort-rules without risking a larger throughput loss.

6 Discussion

In our analysis we saw that there was a noticeable difference between the Raspberry Pi model B+ and the Raspberry Pi 2 model B, in network throughput as well as memory and CPU usage. Generally the Raspberry Pi 2 model B has almost twice the throughput performance compared to the Raspberry Pi model B+. When comparing the difference between rule-sets for throughput, CPU and memory, we see only a small difference in throughput and CPU performance compared to the memory usage. The memory usage followed a more linear increase as more rules were applied, indicating that while network throughput is more dependent on CPU than memory, the number of rules implemented relies more on memory than on CPU.

When comparing our results to the previous study from 2013 [5] we noticed some interesting differences. The study measured the throughput on the Raspberry Pi model B+ running the operating system IPfire; their study did not include the Raspberry Pi 2 model B. We tested the throughput performance on both the Raspberry Pi model B+ and the Raspberry Pi 2 model B running the operating system Arch Linux ARM. Both of our Raspberry Pis produced better results. The Raspberry Pi 2 model B is expected to do so because of its updated hardware, but even our Raspberry Pi model B+ measured twice as high as theirs in throughput. It is however important to point out that our methods are not the same as they used TPTest instead of Iperf to measure throughput. Furthermore we were able to implement more than twice the number of Snort-rules and still get a higher throughput. They noted in their report[5] it was not possible to implement all available Snort-rules as there was not enough available memory.

While our studies can not be fully compared because of our different methods, it is possible to point out that our solution did not suffer the same problems they reported. We believe the main reason we could implement more Snort-rules is because we used a different operating system. While we choose Arch Linux ARM because of its resource efficiency and compatibility with both the Raspberry Pi model B+ and the Raspberry Pi 2 model B, they choose IPFire for its graphical interface which they reasoned would be easier to use for users with limited technical expertise and who are not used to a command line interface. As we had no requirement on ease of use we were able to choose what we believe is a more efficient operating system. The change in operating system increased the efficiency, combined with the absence of a graphical user interface allowed more resources to be used to handle the network traffic, and therefore increase the throughput performance.

In the study from 2013[5] they also measured how well Snort could detect malicious activity. With the rules they were able to implement, Snort could only detect 54% of the attacks. We speculate that we would be able to detect even more attacks if we did the same intrusion detection tests, because we were able to implement more rules. If our theory is true our implementation should grant higher security.

As a final note, it is important to mention that we can not be sure that all Snort-rules implemented are used when analyzing the network traffic created by Iperf. In our tests Iperf only sent TCP-packages, not carrying any particular data, therefore it might be possible that Snort did compare the data packages to all of the rules available, and thus giving us a greater result than would be archived in a real environment. In order to better study the impact Snort might have on performance while analyzing traffic might be to use regular traffic from many users going through the device.

7 Conclusion

In this study we examined the possibilities of using a Raspberry Pi as an IDS in a home network. Our premiss was to measure the performance of the two latest models of Raspberry Pis and see if they could satisfy our requirements of not reducing network throughput by more then 30%. We also wished to compare our results to a previous study [5] on the subject in order to see if we could avoid the limitations they discovered.

Our conclusion is then that a Raspberry Pi can indeed be used as an IDS but only under certain circumstances, the first of which is the users throughput demands. Using a Raspberry Pi as an IDS did have an negative effect on the network throughput. All though the effect was less so on the Raspberry Pi 2 model B, because of its improved hardware over the previous generation it is evident that the capacity of both the Raspberry Pis suffers from their limited hardware. Even with the IDS software disabled the best throughput performance measured in our tests was 94 Mbit/s with the Raspberry Pi 2 model B. We believe that on a 100 Mbit/s connection a loss of only 6 Mbit/s could be acceptable for most users, but not on a 1000 Mbit/s connection as the loss would be over 90%.

The Raspberry Pi model B+ suffered a throughput drop which exceeds our limit of 30% regarding throughput performance, but we believe it could still be used as an IDS if it fulfills the users requirements. One scenario could be the following: the IDS is connected between the Internet and the users network. If the user has an Internet connection of 14 Mbit/s, there would be no need for any greater throughput capacity of the IDS device. The Raspberry Pi model B+ would then be a better choice because of the lower implementation costs compared to the Raspberry Pi 2 model B.

When comparing our study with the previous study from 2013 [5] we were able to avoid the the limitations they mentioned, while getting throughput results that fulfill our own requirements. However, while they had a requirement that the implementation should be as easy as possible which they had to take consideration when choosing operating system. Our solution does not include a graphical user interface and therefore allocates less system resources, but instead requires the user to know how to use a command line interface in Linux and have some knowledge in networking. However, we do not believe this has to be a downside, instead we see a potential market for companies wishing to sell this kind of implementation as a low cost service. The company could prepare a memory card with all the necessary configurations and updates. The user would then only need to connect the device to the network, insert the memory card and power adapter. This solution could even be used by larger companies wishing to improve the network security of users working from their homes.

As a closing statement we believe the price is one of the strongest arguments for this kind of implementation. The low cost of the Raspberry Pis means that it can be a more affordable IDS compared to what would otherwise be an expensive enterprise grade solution. Our solution opens up possibilities for users in increasing their home network security from intrusions. Looking into the future we believe that Raspberry Pi devices has many possibilities to explore and that the last word its role in network security has yet to be said.

7.1 Recommendations for further research

This is our recommendations for studies that could be researched further. These were some of subjects that we chose not to examine and therefore excluded in our confinements for this study.

It could be of interest to compare different Linux operating systems made for the Raspberry Pi and see if there is any differences between them when using the Raspberry Pi as an network intrusion detection system.

Investigations could be made to see if the rules were able to implement with rule-set A does provide a higher grade of security over rule-set B. More rules could in theory result in more attacks being detected by an intrusion detection device. It is also possible to research which rules are the most useful.

It could also be possible to further research if different types of network traffic would affect how Snort analyze network traffic. There is a tool called httperf, normally used to measure the performance of web servers, which could be used to simulate web traffic passing through the IDS device. However, in order to fully investigate how Snort affects network performance, it would need to be tested in a real network environment.

References

[1] M. Garnaeva, V. Chebyshev, D. Makrushin, R. Unuchek, and A. Ivanov. Online threats (web-based attacks). Kaspersky Lab ZAO. [Online]. Available: https://securelist.com/analysis/kaspersky-security-bulletin/68010/kaspersky-security-bulletin-2014-overall-statistics-for-2014/ [Kontrollerad: 2015-05-16]

[2] N. Kho, "Content and the connected home," *EContent*, vol. 38, 2015, no.3, pp. 12-17.

[3] P. Robichaux. Noticing and responding to network-borne attacks. Microsoft. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc723457.aspx [Kontrollerad: 2015-05-16]

[4] ——. Protecting yourself before an attack. Microsoft. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc723457.aspx [Kontrollerad: 2015-05-16]

[5] F. Johansson, M. Johansson, and J. Johansson. Ids för alla : Intrångsdetekteringssystem för hemmaanvändare. Diva. [Online]. Available: http://hh.diva-portal.org/smash/record.jsf?pid=diva2%3A635365&dswid=6482 [Kontrollerad: 2015-05-19]

[6] A. L. ARM. Why arch linux arm? Arch Linux ARM. [Online]. Available: http://archlinuxarm.org/support/faq [Kontrollerad: 2015-05-09]

[7] W. Starlings, *Network Security Essentials Applications and Standards*, 4th ed. Prentice Hall, 1 Lake Street, Upper Saddle River, NJ 07458: Pearson Education, Inc., 2011.

[8] M. Berge. Intrusion detection faq: What is intrusion detection? SANS. [Online]. Available: https://www.sans.org/securityresources/idfaq/what_is_id.php [Kontrollerad: 2015-05-09]

[9] E. Carter. (2002, 2) Intrusion detection systems. Cisco. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=25334 [Kontrollerad: 2015-05-09]

[10] S. Northcutt. Intrusion detection faq: What is network based intrusion detection? Sans. [Online]. Available: http://www.sans.org/security-resources/idfaq/network_based.php [Kontrollerad: 2015-05-09]

[11] Snort. What is snort? Sourcefire. [Online]. Available: https://www.snort.org/faq/what-is-snort [Kontrollerad: 2015-05-09]

[12] ——. What is a signature? Sourcefire. [Online]. Available: https://www.snort.org/faq/what-is-a-signature [Kontrollerad: 2015-05-09]

[13] ——. What is a snort rule? Sourcefire. [Online]. Available: https://www.snort.org/faq/what-is-a-snort-rule [Kontrollerad: 2015-05-09]

[14] R. P. FOUNDATION. About us. RASPBERRY PI FOUNDATION. [Online]. Available: https://www.raspberrypi.org/about/ [Kontrollerad: 2015-05-09]

[15] C. Severance, "Eben upton: Raspberry pi." *Computer*, vol. 46, 2013, issue 10, p14-16.

[16] R. P. FOUNDATION. What is a raspberry pi? RASPBERRY PI FOUNDATION. [Online]. Available: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/ [Kontrollerad: 2015-05-09]

[17] ——. Raspberry pi 1 model b+. RASPBERRY PI FOUNDATION. [Online]. Available: https://www.raspberrypi.org/products/model-b-plus/ [Kontrollerad: 2015-05-09]

[18] ——. Raspberry pi 2 model b. RASPBERRY PI FOUNDATION. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ [Kontrollerad: 2015-05-09]

[19] ——. Raspberry pi hardware. RASPBERRY PI FOUNDATION. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/ [Kontrollerad: 2015-05-09]

[20] T. PULTAROVA, "Raspberry pi 2 aims to give pc makers run for money." *Engineering & Technology*, vol. 10, 2015, issue 2, p21-21.

[21] A. L. ARM. Arch linux arm. Arch Linux ARM. [Online]. Available: http://archlinuxarm.org [Kontrollerad: 2015-05-09]

[22] ——. What is arch linux arm? Arch Linux ARM. [Online]. Available: http://archlinuxarm.org/support/faq [Kontrollerad: 2015-05-09]

[23] M. Gates, A. Tirumala, J. Dugan, and K. Gibbs. The Board of Trustees of the University of Illinois, title = What is Iperf?, url = https://iperf.fr/, urldate = 2015-05-09.

[24] ——. The Board of Trustees of the University of Illinois, title = Iperf features, url = https://iperf.fr/, urldate = 2015-05-09.

[25] S. Godard. Description. [Online]. Available: http://www.unix.com/man-page/All/1/sar/ [Kontrollerad: 2015-05-09]

[26] E. Nemeth, G. Snyder, T. R Hein, B. Whaley, and [et al.], *UNIX AND LINUX SYSTEM ADMINISTRATION HANDBOOK*, 4th ed. Upper Saddle River, New Jersey: Pearson Education, Inc., 2014.

[27] Pixelpress. Dynatek jaz drive icon. IconArchive.com. [Online]. Available: http://www.iconarchive.com/show/apples-icons-by-fasticon/display-icon.html [Kontrollerad: 2015-05-17]

[28] F. I. Design. Display icon. IconArchive.com. [Online]. Available: http://www.iconarchive.com/show/apples-icons-by-fasticon/display-icon.html [Kontrollerad: 2015-05-17]

[29] D. Kirkland. Description. Canonical Ltd. [Online]. Available: http://manpages.ubuntu.com/manpages/lucid/man1/iperf.1.html [Kontrollerad: 2015-05-09]

[30] Snort. Rules. Sourcefire. [Online]. Available: https://www.snort.org/downloads/#ruledownloads [Kontrollerad: 2015-05-21]

[31] W. Stallings, *OPERATING SYSTEM Internals and Design Principles*, 7th ed. Upper Saddle River, New Jersey: Prentice HallÙ, 2012.

[32] R. Love, *Linux Kernel Development*, 3rd ed. RR Donnelley, Crawfordsville, Indiana: Pearson Education, Inc., 2010.

A Snort installation on Arch Linux ARM

- Download Snort from Arch Linux AUR.
- Download and install the Data Acquisition Library(DAQ) from Snorts official website.
- Edit the PKGBUILD-file(shell-script file that compiles the Snort installation package).

  Configure it to compile Snort for an ARM-based system.

  Remove configuration for static DAQ.
- An updated rule set needed to be downloaded from Snorts official website.
- Create the following files and directories as Snort expect them to exist:

  /etc/snort/rules/white_list.rules

  /etc/snort/rules/black_list.rules

  /etc/snort/rules/local.rules

  /usr/lib/snort_dynamicrules
- Edit /etc/snort/snort.conf, rows edited:

  File path to the rules.

  Set the home network.

  Rules to be applied.

B Snort rules from the thesis "IDS För Alla"

Snort rules from the thesis "IDS För Alla" [5].

include $RULE_PATH/app-detect.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/blacklist.rules
include $RULE_PATH/botnet-cnc.rules
include $RULE_PATH/browser-chrome.rules
include $RULE_PATH/browser-firefox.rules
include $RULE_PATH/browser-ie.rules
include $RULE_PATH/chat.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/emerging-ciarmy.rules
include $RULE_PATH/emerging-compromised.rules
include $RULE_PATH/emerging-current_events.rules
include $RULE_PATH/emerging-deleted.rules
include $RULE_PATH/emerging-dns.rules
include $RULE_PATH/emerging-dos.rules
include $RULE_PATH/emerging-malware.rules
include $RULE_PATH/emerging-misc.rules
include $RULE_PATH/emerging-mobile_malware.rules
include $RULE_PATH/emerging-rpc.rules
include $RULE_PATH/emerging-scan.rules
include $RULE_PATH/emerging-shellcode.rules
include $RULE_PATH/emerging-snmp.rules
include $RULE_PATH/emerging-trojan.rules
include $RULE_PATH/emerging-worm.rules
include $RULE_PATH/file-pdf.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/indicator-obfuscation.rules
include $RULE_PATH/indicator-shellcode.rules
include $RULE_PATH/malware-backdoor.rules
include $RULE_PATH/malware-cnc.rules
include $RULE_PATH/malware-other.rules
include $RULE_PATH/malware-tools.rules
include $RULE_PATH/os-windows.rules
include $RULE_PATH/phishing-spam.rules
include $RULE_PATH/policy-multimedia.rules
include $RULE_PATH/policy-other.rules
include $RULE_PATH/policy-social.rules
include $RULE_PATH/policy-spam.rules
include $RULE_PATH/protocol-finger.rules
include $RULE_PATH/protocol-ftp.rules
include $RULE_PATH/protocol-icmp.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/specific-threats.rules

```
include $RULE_PATH/spyware-put.rules
include $RULE_PATH/web-activex.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-php.rules
```

C All available Snort rules

include $RULE_PATH/local.rules
include $RULE_PATH/app-detect.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/blacklist.rules
include $RULE_PATH/botnet-cnc.rules
include $RULE_PATH/browser-chrome.rules
include $RULE_PATH/browser-firefox.rules
include $RULE_PATH/browser-ie.rules
include $RULE_PATH/browser-other.rules
include $RULE_PATH/browser-plugins.rules
include $RULE_PATH/browser-webkit.rules
include $RULE_PATH/chat.rules
include $RULE_PATH/content-replace.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/exploit-kit.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/file-executable.rules
include $RULE_PATH/file-flash.rules
include $RULE_PATH/file-identify.rules
include $RULE_PATH/file-image.rules
include $RULE_PATH/file-multimedia.rules
include $RULE_PATH/file-office.rules
include $RULE_PATH/file-other.rules
include $RULE_PATH/file-pdf.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/icmp-info.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/indicator-compromise.rules
include $RULE_PATH/indicator-obfuscation.rules
include $RULE_PATH/indicator-shellcode.rules
include $RULE_PATH/info.rules
include $RULE_PATH/malware-backdoor.rules
include $RULE_PATH/malware-cnc.rules
include $RULE_PATH/malware-other.rules
include $RULE_PATH/malware-tools.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/multimedia.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/nntp.rules

include $RULE_PATH/oracle.rules
include $RULE_PATH/os-linux.rules
include $RULE_PATH/os-other.rules
include $RULE_PATH/os-solaris.rules
include $RULE_PATH/os-windows.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/p2p.rules
include $RULE_PATH/phishing-spam.rules
include $RULE_PATH/policy-multimedia.rules
include $RULE_PATH/policy-other.rules
include $RULE_PATH/policy.rules
include $RULE_PATH/policy-social.rules
include $RULE_PATH/policy-spam.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/protocol-finger.rules
include $RULE_PATH/protocol-ftp.rules
include $RULE_PATH/protocol-icmp.rules
include $RULE_PATH/protocol-imap.rules
include $RULE_PATH/protocol-pop.rules
include $RULE_PATH/protocol-services.rules
include $RULE_PATH/protocol-voip.rules
include $RULE_PATH/pua-adware.rules
include $RULE_PATH/pua-other.rules
include $RULE_PATH/pua-p2p.rules
include $RULE_PATH/pua-toolbars.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/scada.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/server-apache.rules
include $RULE_PATH/server-iis.rules
include $RULE_PATH/server-mail.rules
include $RULE_PATH/server-mssql.rules
include $RULE_PATH/server-mysql.rules
include $RULE_PATH/server-oracle.rules
include $RULE_PATH/server-other.rules
include $RULE_PATH/server-webapp.rules
include $RULE_PATH/shellcode.rules
include $RULE_PATH/smtp.rules
include $RULE_PATH/snmp.rules
include $RULE_PATH/specific-threats.rules
include $RULE_PATH/spyware-put.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/voip.rules
include $RULE_PATH/web-activex.rules

include $RULE_PATH/web-attacks.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/black_list.rules
include $RULE_PATH/content-replace.rules
include $RULE_PATH/deleted.rules
include $RULE_PATH/emerging-activex.rules
include $RULE_PATH/emerging-botcc.portgrouped.rules
include $RULE_PATH/emerging-botcc.rules
include $RULE_PATH/emerging-chat.rules
include $RULE_PATH/emerging-ciarmy.rules
include $RULE_PATH/emerging-compromised.rules
include $RULE_PATH/emerging-current_events.rules
include $RULE_PATH/emerging-deleted.rules
include $RULE_PATH/emerging-dns.rules
include $RULE_PATH/emerging-dos.rules
include $RULE_PATH/emerging-drop.rules
include $RULE_PATH/emerging-dshield.rules
include $RULE_PATH/emerging-exploit.rules
include $RULE_PATH/emerging-ftp.rules
include $RULE_PATH/emerging-games.rules
include $RULE_PATH/emerging-icmp.rules
include $RULE_PATH/emerging-icmp_info.rules
include $RULE_PATH/emerging-imap.rules
include $RULE_PATH/emerging-inappropriate.rules
include $RULE_PATH/emerging-info.rules
include $RULE_PATH/emerging-malware.rules
include $RULE_PATH/emerging-misc.rules
include $RULE_PATH/emerging-mobile_malware.rules
include $RULE_PATH/emerging-netbios.rules
include $RULE_PATH/emerging-p2p.rules
include $RULE_PATH/emerging-policy.rules
include $RULE_PATH/emerging-pop3.rules
include $RULE_PATH/emerging-rbn-malvertisers.rules
include $RULE_PATH/emerging-rbn.rules
include $RULE_PATH/emerging-rpc.rules
include $RULE_PATH/emerging-scada.rules
include $RULE_PATH/emerging-scan.rules
include $RULE_PATH/emerging-shellcode.rules
include $RULE_PATH/emerging-smtp.rules
include $RULE_PATH/emerging-snmp.rules
include $RULE_PATH/emerging-sql.rules
include $RULE_PATH/emerging-telnet.rules

include $RULE_PATH/emerging-tftp.rules
include $RULE_PATH/emerging-tor.rules
include $RULE_PATH/emerging-trojan.rules
include $RULE_PATH/emerging-user_agents.rules
include $RULE_PATH/emerging-voip.rules
include $RULE_PATH/emerging-web_client.rules
include $RULE_PATH/emerging-web_server.rules
include $RULE_PATH/emerging-web_specific_apps.rules
include $RULE_PATH/emerging-worm.rules
include $RULE_PATH/emerging.conf
include $RULE_PATH/file-java.rules
include $RULE_PATH/indicator-scan.rules
include $RULE_PATH/local.rules
include $RULE_PATH/os-mobile.rules
include $RULE_PATH/protocol-dns.rules
include $RULE_PATH/protocol-nntp.rules
include $RULE_PATH/protocol-other.rules
include $RULE_PATH/protocol-rpc.rules
include $RULE_PATH/protocol-snmp.rules
include $RULE_PATH/protocol-telnet.rules
include $RULE_PATH/protocol-tftp.rules
include $RULE_PATH/protocol-voip.rules
include $RULE_PATH/reference.config
include $RULE_PATH/server-samba.rules

D Hardware configuration

Machine	Processor	Description
Dell Precision T3500	Intel Xeon W3530, 2,8ghz	Host for the virtual Ubuntu client
Virtual machine	Intel Xeon W3530, 2,8ghz	Virtual Ubuntu client

E Software specifications

Software	Version	Description
Windows 7 Professional	Service Pack 1	Operating system
VMware Workstation	11.0.0 build-2305329	Hypervisor
Ubuntu	14.10	Operating system
Arch Linux ARM	3.18.11-2-ARCH	Operating system
Iperf	2.0.5	Throughput measuring software
Snort	2.9.7.2 GRE (Build 177)	Intrusion detection software
Sysstat	11.1.3-1	Package containing Sar