A study of Centralized Network Intrusion Detection System using low end single board computers
Abstract:

The use of Intrusion Detection Systems is a normal thing today in bigger companies, but the solutions that are to be found in market is often too expensive for the smaller company. Therefore, we saw the need in investigating if there is a more affordable solution. In this report, we will show that it is possible to use low cost single board computers as part of a bigger centralized Intrusion Detection System. To investigate this, we set up a test system including 2 Raspberry Pi 3 Model B, a cloud server and the use of two home networks, one with port mirroring implemented in firmware and the other with dedicated span port. The report will show how we set up the environment and the testing we have done to prove that this is a working solution.

Keywords: IDS, NIDS, Network Intrusion Detection System, Raspberry Pi, Distributed NIDS, Single Board Computer
1 Table of Content

2 Terminology ................................................................................................................. 4
3 Introduction ..................................................................................................................... 4
3.1 Background .................................................................................................................. 4
3.2 Previous surveys in this area ....................................................................................... 5
3.3 IDS Intrusion Detection System .................................................................................... 5
  3.3.1 Intrusion detection ................................................................................................. 5
  3.3.2 Types of Intrusion Detection ............................................................................... 6
  3.3.3 Market analysis ...................................................................................................... 6
3.4 Problem definition ...................................................................................................... 7
3.5 Scope and goals .......................................................................................................... 7
3.6 Research methodology ............................................................................................... 7
4 Implementation of proposed system ................................................................................. 7
  4.1 Hardware .................................................................................................................... 7
  4.2 Management server .................................................................................................. 8
  4.3 Software .................................................................................................................... 10
    4.3.1 Snort ................................................................................................................... 10
    4.3.2 Barnyard2 .......................................................................................................... 11
    4.3.3 Pulledpork ......................................................................................................... 11
    4.3.4 Raspbian Lite .................................................................................................. 11
    4.3.5 BASE (Basic Analysis and Security Engine) ...................................................... 11
  4.4 VPN ........................................................................................................................... 12
  4.5 Hardware scenarios .................................................................................................. 14
    4.5.1 Build Scenario 1 (fig 11) .................................................................................. 15
    4.5.2 Build Scenario 2 (fig 12) .................................................................................. 16
    4.5.3 Build Scenario 3 (fig 13) .................................................................................. 17
  4.6 Snort Installation on the Raspberry Pi ....................................................................... 17
5 Results of testing .......................................................................................................... 18
6 Discussion ..................................................................................................................... 24
7 Conclusions .................................................................................................................. 24
8 Future work .................................................................................................................. 25
9 References .................................................................................................................... 26
10 Appendices .................................................................................................................. 27
  10.1 Appendix A: Install of Raspbian Lite ................................................................. 27
  10.2 Appendix B: Installing Snort .................................................................................. 29
    10.2.1 Dependencies ................................................................................................. 29
    10.2.2 Install DAQ (Data Acquisition Library) ......................................................... 30
    10.2.3 Basic Configuration of Snort .......................................................................... 31
  10.3 Appendix C: Installing Barnyard2 .......................................................................... 35
  10.4 Appendix D: Install of PulledPork .......................................................................... 38
    10.4.1 Pulledpork configuration ................................................................................. 38
  10.5 Appendix E: Make Snort and Barnyard2 run as service and install of BASE ... 41
  10.6 Appendix F: List of work done by who ................................................................. 44
2 Terminology

- IPS (intrusion prevention system): A system to prevent computer network attacks
- IDS (Intrusion detection system): A system to detect computer network attacks
- NIDS (Network Intrusion Detection System)
- IOT (Internet of Things): A new age of internet where everything is connected to a computer network
- Single Board Computer: A cheap, single board computer that contains all parts for a functional computer
- IP (internet protocol): Protocol used on computer networks
- Centralized: Design with central management.
- IPSEC: encrypted network traffic using according to RFC 4301
- NAT (network address translation): procedure to mask IP addresses behind a firewall and show another external IP
- Internal network: network where you usually put your internal resources like user workstations, printers, and internal servers
- External network: usually untrusted network like internet or network where
- DMZ: demilitarized Zone: A fully firewalled network segment used for hosting internet facing servers with very minimal firewall openings to prevent unauthorized access to the servers
- GUI: Graphical User Interface
- OS: Operative System
- SOC: System on a Chip
- CIA: Confidentiality, Integrity and Availability
- SOHO: small off and Home Office
- BASE (Basic Analysis and Security Engine) a program to present the Snort data saved in database

3 Introduction

3.1 Background

With the evolution of IOT and devices, the number of networks and the need to identify threats early will increase. Introduction of IPv6 will allow IP addresses to be assigned a vast number of devices. A typical historical setup is an intrusion detection system on the border router which listens to traffic going in and out from the internet. With the increased number of devices there is a need to segment the networks both logically and physically. With this physical segmentation the ability to monitor the different segments will be...
needed. Since current hardware to do this is very expensive we will investigate if this can be done with standard low end single board computers with centralized architectures.

### 3.2 Previous surveys in this area

Since 2013 it is already known that a Raspberry Pi can work as a cheap home NIDS. Three students at Halmstad University made a study (Johansson, Johansson, and Johansson 2013) where they used a Raspberry Pi Model B+ to run their tests. They used a Linux distribution called IPFire (“Www.ipfire.org - Welcome to IPFire” 2017) which have Snort included in the OS. It comes with a GUI that makes it more user friendly but uses more system resources. Their setup was a little different compared to what is going to be presented in this paper. Instead of mirroring the traffic they used a second USB network interface so they forced all the traffic through the Raspberry Pi which could limit the throughput and, they only tested if it could work on a small home network. What they wanted to evaluate was the possibility of a valuable and user-friendly IDS for an average home user. They tested how good the system was on detecting attacks against the network and if there was any loss of performance on network speed. This report written in Swedish explains very well the function of an IDS and the technique behind it. They concluded that Raspberry Pi with Snort was useful as an extra protection on the home network but they also saw limitations in hardware. The internal memory was not enough to load all the Snort rules which limited the detection rate.

It is also tested in 2015 (Aspernäs and Simonsson 2015) on the Raspberry Pi 2 Model B in another survey. This test was made with a lightweight Linux distribution called Arch Linux. This is a barebone OS with no GUI which is not that user friendly but that results in more free system resources for the task. They also tested the difference between the older Raspberry Pi Model B and the latter Raspberry Pi 2 Model B. These two tests were made with focus on home networks and on older hardware/software but still showed that the Raspberry Pi is a potential product suitable for our tests.

### 3.3 IDS Intrusion Detection System

#### 3.3.1 Intrusion detection

The definition of intrusion detection is described by NIST (Bace and Mell 2001) as the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, where intrusion is defined as attempts of attacks on CIA or bypassing the security mechanisms of a computer or network. IDS are software or hardware that automates the monitoring and analysis process of networks and computers.
3.3.2 Types of Intrusion Detection

- Anomaly based

Anomaly based detection or behavior based detection monitors network connections, host and users over time to record normal behavior and then alert when unusual traffic patterns is discovered.

- Signature based

In signature based detection the system uses predetermined rules and compares them with the ongoing network traffic. When the system finds a match, an alert is raised.

3.3.3 Market analysis

The current market is turning to an integrated solution for antivirus, host based intrusion based detection and firewalls. While these systems are very advanced and efficient, the price increase can be steep when all features are added.

The price for a typical entry model like the MacAfee NS3100 intrusion detection device (fig 1) is 9000 dollar (“McAfee Network Security IPS NS3100 - Security Appliance - Associate - IPS-NS3100A - Intrusion Detection/Prevention” 2017). This model supports up to eight connections but all must be within ethernet distance (copper 100m) of the device. For the multisite company with several smaller firewalls there might be a need to deploy one device per site if local internet access is used.

![NS3100](Ns3100IDS.png)

The NS3100 is supported by a company with many years of experience in the area so that will be a good investment but probably too high for most smaller companies especially when compared to the router cost of for example the Meraki MX60 in this paper (800dollar may2017)
One possible solution is to use host based solutions built into the clients and servers which lowers the price. But that takes away some of the features of the early warning system of listening on the outside of the firewall. And also, if a system is compromised, the host based feature is usually turned off by the malware infection. In such a case, an independent network based solution will continue to monitor the network for signs of the malware infection.

3.4 Problem definition

The problem of scaling current IDS solutions into bigger networks with physical segmentation with many new device types with possible vulnerabilities. This includes IOT devices and a still growing number of handheld devices.

3.5 Scope and goals

Examine efficiency of single board computers to determine if they can be used as IDS systems in distributed or centralized solution and find if its scalable. We will design a signature distribution system.

We will not harden devices or create fully functional solutions ready for production. This is work that can be done later whereas the scope of this paper is to determine if the primary features like the actual IDS sensor with secure VPN connectivity can run on the single board computers.

3.6 Research methodology

By observations of the test system we will evaluate if Raspberry Pi 3 Model B is able to collect the data and deliver the data to the central server. By measuring hit rate when exposing the network to different bandwidth loads we can determine when the hardware reaches its maximum reliable performance.

4 Implementation of proposed system

4.1 Hardware

The Raspberry Pi 3 Model B (fig 2) is a single board computer, not much larger than a credit card (85x56x17mm) but still quite powerful. It is equipped with network card, USB- and HDMI port and that have made it very popular around the world as for example media center solutions. Aside of Raspbian (“Download Raspbian for Raspberry Pi” 2017),
a Debian-based Linux distribution, the Raspberry Pi is compatible with several different OS such as Ubuntu, Win10 IOT and Android Things.

The Raspberry Pi 3 Model B is the latest edition. It comes with a Broadcom SOC BCM2837 that have a quad-core 64-bit ARM Cortex-A53 running at 1.2 GHz, a VideoCore IV GPU running at 400MHz and a 1 GB DDR2 Memory that is shared with the GPU. It is equipped with 4 USB 2.0 ports, 1 micro USB 2.0 port (for powering the unit), 1 HDMI port, audio out jack and a RJ45 Ethernet 10/100. To connect to wireless Bluetooth 4.0 and Wi-Fi 802.11n are incorporated.

4.2 Management server

The distributed design is where a central management server receives all logs from the nodes and presents the logs to the administrator in a subtle way. This can either be done using a on premise server or hosted in a cloud offering like Amazon or similar. For this environment, we choose an offering from Linode (“SSD Cloud Hosting & Linux Servers - Linode” 2017) where we could setup a Linux virtual server in the cloud. The system makes it possible to install various Linux distributions and we choose Ubuntu 17.04 which was easily installed from the dashboard in Linode Manager (fig 3). This server will receive a fixed public IP address which makes it very suitable for hosting this solution where IPSEC over NAT is needed. It’s much harder to create IPSEC encrypted traffic when both sides of the communication are behind a NAT firewall.
Fig 3 Control panel of Linode cloud server
4.3 Software

4.3.1 Snort

Snort (“Snort - Network Intrusion Detection & Prevention System” 2017) ids is the most interesting software for this implementation since it is open source. It was created by Martin Roesch in 1998 who founded the company Sourcefire and was recently acquired by Cisco to be used in their enterprise IDS systems. It can achieve real-time traffic analysis and packet logging on IP networks in many different scenarios ranging from simple packet analyzing and reporting to full active intrusion prevention with capabilities to tear down TCP sessions to stop attacks. The current version of the software is Snort 2.9.9.0.

The software runs on both Windows and Unix based systems can even run on smaller platforms like the Raspberry Pi. There are many Add-ons and variants of the software that can be used like daemon logger, Pulled Pork and many other. There are also full Linux distributions with the main purpose of being IDS installations like for example Security Onion.

The software is maintained under GPL license and rules are updated via three levels: common, registered and subscribed. The two first are free of charge and is suitable for private use and smaller companies. The Subscribed version could be used on enterprise installations.

The commercial version of the software will run on Cisco isr4000 and Asr1000 series routers which leverages the Linux based IOS XE operating system (“Cisco 4000 Series Integrated Services Routers” 2017).

Rules distribution can be automated via some third-party tools achieving a fully automatic system.

The installation of the software can be done either from the source packages or via the software distribution tools available in Linux/UNIX or via Software download in Windows. There are some requirements for the installation but there are good tutorials available on the internet to get going since the software have been developed for almost 20years. The challenging thing is to understand what the different attacks are so that you
look for the correct signatures. An understanding of networking in general is needed to do correct placement of probes depending on what the purpose of the IDS/IPS is.

4.3.2 Barnyard2
When Snort receives a lot of data to process, a low-end system could cause data loss. And that is because the lack of processing power. A solution to that is Barnyard2 ("Barnyard2 - ForensicsWiki" 2017), an open source interpreter that help Snort with some of the task. We set Snort to just put the analyzed data in a binary format called unified2 to a log file where Barnyard then collects and parses the data to a readable format. Barnyard2 also sends the processed data to several types of databases. All this is done in an asymmetrical way when there are resources not used. In this study, we are going to use Barnyard2 to parse the data and send it over the encrypted tunnel to a MySQL database on the central server.

4.3.3 Pulledpork
To automate the updates of the Snort rules we are using Pulledpork ("Pulled Pork - Suricata & Snort Rule Management - Darknet" 2016). This is a Perl script that can determine the version of Snort installed and download and install the appropriate ruleset. We are implementing this so we know that we always have the latest rules.

4.3.4 Raspbian Lite
The choice of OS was important since we need one which is less resource hungry. Snort does not have a graphical interface so we do not need to support that. The choice landed on Raspbian Lite ("Download Raspbian for Raspberry Pi" 2017), a lite version on the official Raspberry OS built on the Linux Distribution Debian 6. This to not have any compatibility problems on the System side.

4.3.5 BASE (Basic Analysis and Security Engine)
To be able to analyze the data on the server we need a tool to read the saved data from the database. An easy, lightweight and opensource interface to do the task is BASE ("BASE" 2017). On a live system, there are paid solutions that are better for the task but for our purpose this is a good choice. BASE works in a web browser and presents the Snort alerts in a clean way (fig 4).
In a centralized solution, we need to securely send all that data to the central server for investigation and since this data is telling a potential attacker of vulnerabilities in a network it’s very important to secure the traffic between the node and the central server. This can be done via dedicated networks but in a widely-dispersed environment like the proposed design we need to be more flexible and create a packet encryption that can traverse any network. Typically, the IPSEC protocols have been hard to implement over for example network address translated network. New features in router have made this simpler and the routers transport the IPSEC traffic over the network. Another problem historically has been that encryption needs processing power which usually isn’t a problem with computer or routers. However, we saw a need to implement a VPN tunnel into the small computers used for this research so that a benchmark could be done. We chose Strongswan as the software to create the tunnel.

The design chosen is point-point IPsec meaning that the traffic will go from the node directly to the centralized machine.

4.4 VPN
To install strongswan we use the Debian packages: (fig 5)

![Installed version of Strongswan](image1)

You install it with the command:

```
sudo apt-get install strongswan
```

This is done on both Raspberry Pi node and on management server.

On the remote node, you use this config (fig 6) in the /etc/ipsec.conf file.

```
config setup
conn %default
    ike lifet ime=60m
    key lif etime=20m
    rekey margin=3m
    keyingtries=1
    # key exchange=ikev2
    authby=secret

conn nat-t
    right=172.104.131.205
    left firewall=yes
    type=transport
    auto=add
```

![Config setup IPsec for remote node (the Raspberry Pi)](image2)

And on the server, you use this file: (fig 7)

```
config setup
conn %default
    ike lifet ime=60m
    key lif etime=20m
    rekey margin=3m
    keyingtries=1
    # key exchange=ikev2
    authby=secret

conn nat-t
    right=%any
    type=transport
    auto=add
```

![Config setup IPsec for server](image3)
This will allow any connection to the server using IPsec to be encrypted. To stop unencrypted traffic, we can add rules in the firewall to only accept the encryption protocol ports.

This encryption is based on shared key so an encryption key is needed. This is put into the file /etc/ipsec.secrets on each machine: (fig 8)

```
172.104.131.205 : PSK Osv+NlxY9LLZwvj4qCC2o/gGrWDF2d21jL
pi@raspberrypi:~ $  
```

*Fig 8 shared key used for IPsec authentication between server and probe*

The packet capture shows only ESP traffic between the server at 172.104.131.205 and one of the probes at 31.209.56.123: (fig 9)

```
tcpdump: listening on wan0_sniff, link-type EN10MB (Ethernet), capture size 262144 bytes
06:41:13.330743 IP (tos 0x0, ttl 64, id 5293, offset 0, flags [DF], proto UDP (17), length 144)
  31.209.56.123.4500 > 172.104.131.205.4500: [no cksum] UDP-encap: ESP(spi=0xc3a27a1a,seq=0x22), length 116
06:41:13.339460 IP (tos 0x0, ttl 55, id 13944, offset 0, flags [none], proto UDP (17), length 144)
  172.104.131.205.4500 > 31.209.56.123.4500: [no cksum] UDP-encap: ESP(spi=0xc3b4071c,seq=0x22), length 116
06:41:14.332221 IP (tos 0x0, ttl 64, id 56375, offset 0, flags [DF], proto UDP (17), length 144)
```

*Fig 9 Screen capture showing tcpdump with traffic*

![Diagram showing path of encrypted traffic]

*Fig 10 Path of encrypted traffic*

### 4.5 Hardware scenarios

We built three sensors with different hardware to test how it’s possible to deploy them. We have chosen these three scenarios as standard setup that could be used in a SOHO environment. The solution is totally scalable and will work just as well with 100 probes as with only a few. The only limit will be the management traffic and the SQL database compared with number of incidents logged. The solution is called a centralized with distributed probes. This means that we only transfer log data and not actual copies of data streams even though Snort itself can add samples of packets into the log database.
4.5.1 Build Scenario 1 (fig 11)

Raspberry Pi 3 Model B with ethernet connected to a Cisco 3560 Gigabit router with a span port configured to mirror all traffic on the fiber outside a Meraki Firewall. The Meraki does not have a mirror function directly and therefore a switch needs to be added to add this functionality.

This scenario listens to the external interface of the firewall as opposed to the other two scenarios. This means we can detect attacks that is not penetrating the firewall.

The management traffic will flow from the Raspberry Pi over the Wi-Fi through the router and then lastly over the internet to the management server. The traffic is encrypted with IPsec as described in the IPsec chapter.

This scenario would suit a smaller company that will use a standalone firewall and smart switches can be segmented into VLAN’s so that one VLAN can span the external traffic. You could even have a separate switch outside the firewall. Since a few more device which also cost a little bit more it might not be suitable for your home environment.

![Fig 11 Build Scenario 1 (small office solution)](image-url)
4.5.2 Build Scenario 2 (fig 12)

Raspberry Pi 3 Model B with an Asus RT-AC68U with Asus WRT-merlin firmware supporting port mirroring. When analyzing the traffic, it is clear that this scenario only listening on the inside of the firewall which might be a good scenario that only warns about real threats that have penetrated the firewall. With a bigger solution like with scenario 1 you can easily add a second or third Raspberry Pi in several places in the solution and get data from for example internal, external and DMZ. Since this solution uses only one ethernet interface both the packet capture and the management traffic will happen on the same interface. This is generally no problem except for the fact that when the link is saturated with mirror traffic the management traffic might be lost since it shares the same interface. But in reality, this will never happen because of the usual bandwidth usage that very rarely hits 100%.

This scenario will work well with the home office solution based on LTE connectivity.
4.5.3 Build Scenario 3 (fig 13)

Almost the same as scenario 2 but connected to ADSL 8Mbit with asymmetrical bandwidth.

![Diagram of Build Scenario 3](image.png)

**Fig 13 Build Scenario 3**

4.6 Snort Installation on the Raspberry Pi

The installation of Snort, Barnyard2 and Pulledpork is described in various tutorials linked to on the Snort website ([https://www.snort.org/documents](https://www.snort.org/documents)) for numerous different platforms. None of them includes the Raspbian OS but since it is built on Debian, a tutorial made for Ubuntu (Dietrich 2015) was a great help to manage through the installation process. (Debian and Ubuntu is similar)

To begin with we installed the Raspberry Pi OS Raspbian Lite (see Appendix A) on a new or formatted micro SD card. The Raspbian Lite comes as an image file so this was done with an image burner.

The install of Snort was done via compiling and installing source code from Snort website ([www.snort.org](http://www.snort.org)). A detailed instruction of how to install and configure Snort 2.9.9.0 is found in (Appendix B).

When Snort is installed we continued with the installation of Barnyard2 (see Appendix C). This is also done from source code and for this install to work we also needed to configure the MySQL database on the server which is also described in Appendix C.
The last install before we get Snort up and running with automated updates is Pulled pork (see Appendix D). The use of Snort without Pulled pork can result in lack of new rulesets, both because it can be complicated updating and one could forget doing it. The Snort rules is updated daily with new threats so to stay protected Pulled pork is one great script for the automated task.

To get Snort and Barnyard2 to start automatically when system start it is a good idea to make them run as service and how to do that is described in Appendix E. We are also using BASE to present the data from the database and how to install is also in appendix E.

5 Results of testing

After setting up the system with three different sensors that connected to the central server and uploaded its findings in the database we started testing if it really worked. The different scenarios had different throughput and the results show that the Pie is loaded to 100% very fast when pushing traffic to it.

Scenario 1 (Gigabit handover and max throughput over fiber of 106mbit) (fig 14) gave a load of the Raspberry Pi of 105% almost instantly after initiating the data stream (fig 15). But even though it was fully loaded we discovered that it catches the anomalies anyway.

Fig 14 Build scenario 1, throughput
This scenario is quite unfair to the 100mbit Raspberry Pi since the Gigabit link can push more than the port can even handle, this must mean that packet drops will occur and thereby missed by the Raspberry Pi. Future versions of Raspberry Pi will probably be equipped with Gigabit port. The Asus Tinker Board (“Tinker Board | Single-Board Computer” 2017) (a similar board with better specifications) have Gigabit ethernet in its current version and might be more suitable now.

The second scenario with the 40mbit LTE connection gives the same result (fig 16-17).
The last Scenario with the 8mbit ADSL gives the same result (fig 18-19) when it comes to processor load and we have not been able to figure out the reason for the load on the processor even at low throughput.

![Image](image1.png)

**Fig 17 Build scenario 2, performance with htop**

**Fig 18 Build scenario 3, throughput**
To be able to test if we really can use it we used the scenario 1 infrastructure and the MX60’s capabilities to limit bandwidth (fig 20).

Global bandwidth limits

<table>
<thead>
<tr>
<th>Per-client limit</th>
<th>down (Kb/s)</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>up (Gb/s)</td>
<td>10000</td>
<td></td>
</tr>
</tbody>
</table>

Fig 20 MX60 bandwidth limiter

The test then was conducted but first creating a local rule in the local.rules Snort file.

```
alert tcp any any -r any 23 (msg:"Telnet";sid:10000003;rev:1;)
```

This rule pretty much looks for traffic from any source to any source on port TCP 23 which is the common telnet port.

The test was conducted by filling the allocated bandwidth with a big download, checking that the correct throughput was achieved and then starting a test script (fig 21) from a computer on the inside of the firewall directed outbound.

The script tries to open a TCP session on port 23 and since the server chosen (log server) actively refuses this session it fails. Then it waits for 5 seconds and tries again. It it does this for 10times and after it was finished we checked how many hits we got in the Base interface.
This was captured in the diagram with 10mbit steps up to 106mbit. We checked the actual throughput in both the download (fig 22) in windows and in the Meraki interface before starting the script.

The base interface easily shows the hits (fig 23). Some other hits were discovered during the run so it does take other hits also.
Up to 30mbit the hit rate is 100% but then it starts to degrade. It takes quite a lot of hits even at 60mbit however. At 70mbit we can see almost 50% degradation and that is consistent up to 100mbit (fig 25-26).

<table>
<thead>
<tr>
<th>Bandwidth mbit</th>
<th>Successful hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>8</td>
</tr>
<tr>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>70</td>
<td>6</td>
</tr>
<tr>
<td>80</td>
<td>6</td>
</tr>
<tr>
<td>90</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>106</td>
<td>6</td>
</tr>
</tbody>
</table>

Fig 25 results of the bandwidth tests

These tests might be too limited to really show the actual performance of the sensor but we also determined that it is not possible to compare a real production environment with the test environment because of the non-deterministic way a production network behave. A suitable test would be to measure these statistics on a heavily used production link where we inject our tests during a time span of weeks and count how many attacks were sent and corresponding hits. The average bandwidth utilization can be presented together with the hits. We did not have any possibility to do this in the short timeframe the project lasted.
The traffic from the Raspberry Pi to the management server is measured in kilobits/s and occasionally reaches over 10kbits/s during database maintenance. The approximate number of sensors that can be attached based on this number is very high and there shouldn’t be any pragmatic limitation to how many sensors that can be connected (fig 27).

![Bandwidth usage over time](image)

**Fig 27 Bandwidth usage over time**

### 6 Discussion

The solution to use many devices like the Raspberry Pi with any real added cost is very attractive and an extensive network of probes can be installed. It could also be possible that low end routers will incorporate for example Snort in the core routing software like Cisco is doing with their ASR 4000 series routers. This would add another dimension to the flexibility where for example a company could give all employees a preconfigured router with the IDS built-in. The raspberries will be harder to deploy since the need for a mirroring port is hard to do with some hardware. But as shown the Asus router in scenario 2 can do this with a simple configuration change. One could also argue that host based IDS built into the anti-virus software is one solution but that will not protect the vast amount of IOT devices where for example one big malware attack was spreading via hacked firmware of network connected cameras.

### 7 Conclusions

When performing the tests, we found a probable limit for the bandwidth that the current Raspberry Pi can handle to be somewhere around 40mbit and potentially future upgrades can handle even more traffic. The conclusion is then that for most small office and home network that have a maximum of 40mbit the Raspberry Pi will do just fine as an IDS solution. We however think that having the same system running on a 100mbit link is not a waste since the hit rate is quite ok even with high load. And a link is typically not loaded a very high percent of the time giving the Raspberry Pi a chance to reach the higher hit rates even if max bandwidth is bigger than what the Raspberry Pi can handle.
When future upgrades to the Raspberry Pi comes, the test can be repeated using the same OS image since it would most probably be backward compatible.

The tests we have done was made in a test environment with artificial traffic at given bandwidth steps. To get a more real-life experience it must be installed in a real-life environment with many users using the network. Such scenario will however not give a constant bandwidth and might even prove that it works better since a smaller company might have an average usage of their bandwidth of around 10-20% of the total. Only a longtime continuous test where test attacks are performed and outcome registered to compare hit rate in the real environment can tell if the system would work in production. One note on intrusion detection is that it should only be one part of the total defense so if it fails there should still be other parts that handle the intrusion.

8 Future work

Since we discovered some degraded performance when pushing the faster bandwidth through the firewall we believe that this work can be redone on other hardware like the Asus Tinker Board or on future versions of Raspberry Pi where potentially a Gigabit Ethernet port is available. One could also test cheaper mini-ITX boards with intel Celeron processors to add computing power. The work can be ported to pretty much any Linux distribution without too much adaptation.

Since the scope was limited to setting up a test scenario the machines have not been hardened and the robustness of the automated scripts is not 100%. This can be done by limiting ports in the server that is open to only allow IPSEC traffic to it.

The Snort suite have IPS features that easily can be turned on and turning the same system into a fully featured network threat protection system which actively can tear down active hostile TCP sessions.
9 References


10 Appendices

10.1 Appendix A: Install of Raspbian Lite

The Raspbian Lite install is an easy task. On a windows computer just load the micro-SD card with proper adapter or card reader and start Etcher (“Etcher by Resin.io” 2017), a SD card imager recommended by The Raspberry Pi Foundation (“Raspberry Pi Foundation - About Us” 2017). This program comes in two versions, one portable and one that needs to be installed. Any one of them will do. The Raspbian Lite image comes zipped so it needs to be unzipped first. When that is done just load the image in Etcher and control that the right drive is selected and then start the image burn process with a press on the flash button (fig 28) and then click ok on the windows prompt.

![Etcher SD card imager](image1)

Fig 28 Etcher SD card imager

When the image is ready it’s almost time to insert the micro-SD card in the Raspberry Pi card slot but before that, we need to do one more thing. Previous versions of Raspbian had SSH enabled by default but due to security concerns this is not the case anymore. Therefore, we need to make a file named ssh with no extension and place it in the boot partition of the SD card. This file can contain anything or be empty. (Fig 29)

![An empty file named ssh](image2)

Fig 29 An empty file named ssh
On first boot the system see this file and enable SSH, deletes the file and then automatically resizes the filesystem to adapt to the size of the SD-card. When this is done it’s time to login. For that we use Putty ("Download PuTTY - a Free SSH and Telnet Client for Windows" 2017) or any other terminal that supports SSH. To get the IP address of the Raspberry Pi you can in windows cmd type arp -a and rule out which one belongs to the Pi or enter the router config and find it there, and at the same time set the IP address as static so it will keep the same address all the time. The default login for Raspbian is user: pi and password: raspberry. (Fig 30)

For security reasons, it’s a clever idea to change the default password. First change the password on the default user.
Type passwd and then enter and follow instructions. You be prompted with current password and then asked for new password. (Fig 31)

To be sure to have all latest OS updates on the Raspberry Pi run the following:

```bash
sudo apt-get update && sudo apt-get dist-upgrade -y
```
and then wait until everything is updated. After this we are ready to start the Snort installation.
10.2 Appendix B: Installing Snort

There is no ready packet to install Snort so we must make our own from source code. Before download we need to create a directory on the Raspberry Pi where we can store the downloads.

mkdir -p ~/source

10.2.1 Dependencies

Snort needs a few things before we can install including, pcap (Packet Capture), pcre (Perl Compatible Regular Expressions) and libdnet (a low level portable interface for networking routines) so we need to install these first and they exist in Raspbian Jessie Repository. To install these just run:

```
sudo apt-get install libpcap-dev -y
sudo apt-get install libpcre3-dev -y
sudo apt-get install libdumbnet-dev -y
```

To install Snort DAQ (Data Acquisition Library) it is also needed to install bison (a general-purpose parser generator(“Bison - GNU Project - Free Software Foundation” 2017)) and flex (Fast Lexical analyzer(Estes [2013] 2017)) first. These are also found in the repository.

```
sudo apt-get install bison -y
sudo apt-get install flex -y
```

The source code for Snort is available at Snort webpage. (Fig 32)

<table>
<thead>
<tr>
<th>Source</th>
<th>Fedora</th>
<th>Contao</th>
<th>FreeBSD</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget <a href="https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz">https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wget <a href="https://www.snort.org/downloads/snort/snort-2.9.9.0.tar.gz">https://www.snort.org/downloads/snort/snort-2.9.9.0.tar.gz</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tar xvfz daq-2.0.6.tar.gz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd daq-2.0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>./configure &amp; make &amp; &amp; sudo make install</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tar xvfz snort-2.9.9.0.tar.gz</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd snort-2.9.9.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>./configure --enable-sourcefire &amp; &amp; make &amp; &amp; sudo make install</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Fig 32 The source code needed to install Snort*
10.2.2 Install DAQ (Data Acquisition Library)

First, we need to download the daq from Snort and save it in the directory we made earlier. With following commands, we move to that directory and download the source code.

```
cd -p ~/source
wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
```

This file is compressed so we need to decompress it before we change into that new directory where we can start the installation.

```
tar xvfz daq-2.0.6.tar.gz
```
```
cd daq-2.0.6/
```

Inside the directory daq-2.0.6 we run the following to install the package:

```
./configure
make
sudo make install
```

Finally, all the dependencies are installed and we can install the main program Snort. We go back to the directory /source and download the Snort source code from their website and decompress the file.

```
cd ~/source
wget https://www.snort.org/downloads/snort/snort-2.9.9.0.tar.gz
```
```
tar xvfz snort-2.9.9.0.tar.gz
```

Then we move into the decompressed directory /snort-2.9.9.0

```
cd snort-2.9.9.0
```

and compile the package almost in the same way as before with the DAQ.

```
./configure --enable-sourcefire --enable-reload
make
sudo make install
```

A critical step to succeed with the installation is to update the shared library cache, this is done with the command:

```
sudo ldconfig
```

To get Snort to work for all users we need to add a symlink:

```
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```
Test if Snort is installed and working by verifying the version with command:

```
snort --version
```

The result should look like this: (fig 33)

![Snort Version Check](image)

**Fig 33 Successful installation of Snort**

10.2.3 Basic Configuration of Snort

Now that Snort is installed we need to do some configurations before we can start the program. First, we need to add a Snort user and a Snort user group. This is because it’s a bad idea to run Snort as root. We also need to create the directories for rules, logfiles and configuration. Snort need `/etc/snort` for config and rule files, `/var/log/snort` for alert files and `/usr/local/lib/snort_dynamicrules` for compiled rules. (Fig 34)

**Creation of user and group:**

```
sudo groupadd snort
```
```
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort
```

**Creation of Needed directories:**

```
sudo mkdir /etc/snort
```
```
sudo mkdir /etc/snort/rules
```
```
sudo mkdir /etc/snort/rules/iplists
```
```
sudo mkdir /etc/snort/preproc_rules
```
```
sudo mkdir /usr/local/lib/snort_dynamicrules
```
```
sudo mkdir /etc/snort/so_rules
```
```
sudo mkdir /var/log/snort
```
```
sudo mkdir /var/log/snort/archived_logs
```

We also need to create some empty rule files for storing of rules and iplists:

```
sudo touch /etc/snort/rules/iplists/black_list.rules
```
```
sudo touch /etc/snort/rules/iplists/white_list.rules
```
```
sudo touch /etc/snort/rules/local.rules
```
```
sudo touch /etc/snort/sid-msg.map
```

31
For the Snort user and group to be able to read and write we need to change some
permissions and change the ownership of the folders:

```bash
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archived_logs
sudo chmod -R 5775 /etc/snort.so_rules
sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort
```

One more thing to do before we can start Snort is to copy configuration files from Snort
source. We need:

1. classification.config
2. file_magic.conf
3. reference.config
4. snort.config
5. threshold.conf
6. attribute_table.dtd
7. gen-msg.map
8. Unicode.map

All these files come in the Snort source download so we need to go to that directory and
copy the files over to the right directory.

```bash
cd ~/source/snort-2.9.9.0/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort
```

```bash
cd ~/source/snort-2.9.9.0/src/dynamic-preprocessors/build/usr/local/lib/snort_dynamicpreprocessor/
sudo cp * /usr/local/lib/snort_dynamicpreprocessor
```

Now Snort is installed and the Snort directory should look like this: (Fig 35)
Lastly, we need to do some changes in the snort.conf file. We need to comment out all the rulesets that is included in this file. We do this because we are going to use pulledpork for downloading and installing rules automatically.

For this we use one command that comment out all of them at the same time.

```
sudo sed -i "s/include \$RULE\_PATH/#include \$RULE\_PATH/" /etc/snort/snort.conf
```

After this is done we open the snort.conf file in a texteditor to make some manual edits, I use nano but any texteditor will do. (Fig36)

```
sudo nano -c /etc/snort/snort.conf
```

The first to change is on Line 45 that default looks like: `ipvar HOME_NET any`

change the “any” to your own home network IP subnet so the line looks like: `ipvar HOME_NET 192.168.2.0/24`
If you don’t know your IP subnet you can check that with:

```
ifconfig | grep “inet add”
```

Next on lines 104-106 set the file paths as follows:

```bash
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
```

on line 113-114:

```bash
var WHITE_LIST_PATH /etc/snort/rules/iplists
var BLACK_LIST_PATH /etc/snort/rules/iplists
```

Last thing to edit in the snort.conf file is to uncomment the line 546. This to be able to use the local.rules file later if we want to test Snort with our own rules. So just remove the hash symbol from the beginning of the line so the line look like this:

```bash
include $RULE_PATH/local.rules
```

Now we can save and close the file. In nano it’s done with ctrl+o then “enter” then ctrl+x

Now we can test if Snort is installed and configured correctly. This is done with the command:

```
sudo snort -T -i eth0 -c /etc/snort/snort.conf
```

-T is for Testing, -i is for interface and -c is for the config.

If all is installed and configured correctly, it should look like (Fig 37) and we can move on to install Barnyard2.

![Completed installation of Snort](image-url)
10.3 Appendix C: Installing Barnyard2

Barnyard2 take care of Snort binary output files and converts them into human readable data and saves it to a database. If Snort did all this processing by itself it could be significant risk of data loss. So, installing Barnyard2 is highly recommended. What we do now is to configure Snort to output the events in the format unified2 to a folder where Barnyard2 then asynchronously reads and places the events into a database. Barnyard need a database so the first thing we do is to install mysql. Run this command and choose a password for root user when asked for:

```
sudo apt-get install -y mysql-server libmysqlclient-dev mysql-client autoconf libtool
```

Then we must edit the snort.conf file to get Snort to output the alerts in the unified2 format. So, open the config file in a texteditor and add the following to the empty line 522. Here I use nano:

```
sudo nano -c /etc/snort/snort.conf
```

add on line 522:

```
output unified2: filename snort.u2, limit 128
```

this make Snort save the output to a file named snort.u2.timestamp and that when the file size reach 128 MB it starts a new file.

Now it's time to download and install Barnyard2 and we get the latest version from github with the wget command. We go first to the source directory so we find the downloaded package later.

```
cd ~/source
```

Then we download the newest barnyard2 package and decompress.

```
wget https://github.com/firnsy/barnyard2/archive/master.tar.gz -O barnyard2-Master.tar.gz
```

```
tar zxfv barnyard2-Master.tar.gz
```

After that enter the barnyard2-master directory and begin the install.

```
cd barnyard2-master
autoreconf -fvi -I ./m4
```

We need to create a link to the dumbnet library that we installed earlier cause Barnyard2 use another name for it.

```
sudo ln -s /usr/include/dumbnet.h /usr/include/dnet.h
```

```
sudo ldconfig
```

we also need to point the installation to the right mysql library and since we are on a Raspberry Pi with an arm processor we need the arm-linux-gnueabihf library.

`.configure --with-mysql --with-mysql-libraries=/usr/lib/arm-linux-gnueabihf`

And then the final steps to install:

```
make
sudo make install
```

Make sure you test that Barnyard2 was installed correctly. (Fig 38)

![Fig 38 A successful install of Barnyard2](image)

Next step is to copy the barnyard2 configuration file from the downloaded source to the Snort directory and then create a barnyard2 directory in `/var/log/` and a barnyard2.waldo file in `/var/log/snort/`. We also need to set Snort user and group as owner of these.

```
sudo cp ~/source/barnyard2-master/etc/barnyard2.conf /etc/snort/
sudo mkdir /var/log/barnyard2
sudo chown snort.snort /var/log/barnyard2
sudo touch /var/log/snort/barnyard2.waldo
sudo chown snort.snort /var/log/snort/barnyard2.waldo
```

To make barnyard2 work we need to create a database on the central server for it to save the alerts to. So, on the server machine start and login to mysql as root and use the barnyard2 schema located in the source directory to create the database.

```
mysql> CREATE database snort;
mysql> USE snort;
mysql> source ~/source/barnyard2-master/schemas/create_mysql
mysql> CREATE USER 'snortuser'@'IP pie' IDENTIFIED BY 'InsertYourPasswordHere';
mysql> GRANT create, insert, select, delete, update on snort.* to 'snortuser'@'IP pie';
mysql> EXIT
```

Mysql have the default behavior to only listen to local host (127.0.0.1) so to get the server database to listen to the connected Pies we need to change that in the mysql configuration file. So still on the server open the file and comment line 47:

```
sudo nano /etc/mysql/my.cnf
Line 47: #bind-address = 127.0.0.1
```
This make the server vulnerable for attacks and should be secured on a real scenario for example by limiting host with IP tables.

To get Barnyard2, back on the Raspberry Pi, to connect to the database we need to add the database credentials in the configuration file. So, open the barnyard2.conf in the text editor and add following at the end of the file:

```
output database: log, mysql, user=sno rt password=InsertYourPasswordHere
dbname=sno rt host=ip SERVER sensor_name=sensor(nr)
```

The config file stores the password in plain text so for security reasons we should restrict who can read the file. Use the following command:

```
sudo chmod o-r /etc/snort/barnyard2.conf
```
10.4 Appendix D: Install of PulledPork

Pulledpork is a script that manages the Snort rules. It can be used for automated downloads, parsing and modifications for all the Snort rulesets. To get it to install we need some dependencies installed first. It adds support for openSSL in the LWP (perl www library).

```bash
sudo apt-get install -y libcrypt-ssleay-perl liblwp-useragent-determined-perl
```

Then we can download the pulledpork source from github to the source directory and decompress.

```bash
cd ~/source
wget https://github.com/shirkdog/pulledpork/archive/master.tar.gz -O pulledpork-master.tar.gz
tar xzvf pulledpork-master.tar.gz
```

After that we move to the pulledpork-master directory and copy the perl script to the `/usr/local/bin` directory and the pulledpork config files to the Snort directory.

```bash
cd pulledpork-master
sudo cp pulledpork.pl /usr/local/bin
sudo chmod +x /usr/local/bin/pulledpork.pl
sudo cp etc/*.conf /etc/snort
```

Finally check that pulledpork is installed correctly. (Fig 39)

![Fig 39 A successful install of Pulledpork](image)

10.4.1 Pulledpork configuration

Now that everything is installed we need to configure pulledpork to do the automatic updates of the Snort rules. To download the official rulesets from www.snort.org we need to register. There are old rules one can get without registration and there is also a paid version to get the absolute latest ruleset but for home use the registered free alternative is sufficient. To download this, we need an “oinkcode” and we get that by register a free account at Snort webpage. This “oinkcode” we need to enter in the pulledpork configuration file together with some other changes. Therefore, open the config file at `/etc/snort/pulledpork.conf` in a texteditor and the following lines should look like this:

```
Line 54: DELETE local.rules
Line 74: rule_path=/etc/snort/rules/snort.rules
Line 89: local_rules=/etc/snort/rules/local.rules
```
Now we can try if pulledpork works by doing a manual run:

```bash
sudo /usr/local/bin/pulledpork.pl -c /etc/snort/pulledpork.conf -l
```

the -c tells the path to the pulledpork config file and the -l make pulledpork write detailed log. The result of the above should output something like this: (Fig 40)

![Fig 40 Pulledpork with installed ruleset](image)

To make the pulledpork snort.rules file active in Snort we need to edit the snort config file to include the path to the file. Open snort.conf in a texteditor and add following line at the end:

```bash
include $RULE_PATH/snort.rules
```

To check that Snort is working and all the rules are loaded run following: (Ignore duplicate warnings and flowbits warnings)

```bash
sudo snort -T -c /etc/snort/snort.conf -i eth0
```

the result: (fig 41)
Now we need pulledpork to run once every day and for that we add the script to crontab and tell when to do the update. Here is needed to choose a random time so the Snort servers don’t get overloaded at certain times.

```bash
sudo crontab -e
```

this asks you to choose your preferred text editor. Crontab list time in a bit of reversed order starting with minutes. (minute (0-59), hour (0-23), day (1-31), month (1-12) and week day (0-6) where Sunday is 0). In the text editor enter following at the end of the file and save and close:

```bash
49 07 * * * /usr/local/bin/pulledpork.pl -c /etc/snort/pulledpork.conf -l
```

This tells pulledpork to update Snort rules every morning at 07:49. (Should pick a different time with every new configuration). If Snort was compiled with “--enable-reload” then Snort will automatically reload itself to load the new ruleset. Otherwise a restart of Snort is needed every day after the update to load the updated ruleset.
10.5 Appendix E: Make Snort and Barnyard2 run as service and install of BASE

To make Snort and barnyard start at system start and then run all the time in the background we need to make some scripts. We start with Snort and open a text editor to create a service file:

```
sudo nano /lib/systemd/system/snort.service
```

The file should look like following text:

```
[Unit]
Description=Snort NIDS Daemon
After=syslog.target network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/snort -q -u snort -g snort -c /etc/snort/snort.conf -l eth0

[Install]
WantedBy=multi-user.target
```

To get Snort service to start at boot we need to enable it:

```
sudo systemctl enable snort
```

now we can start the service and then check that it is running. (Fig 42)

```
sudo systemctl start snort
systemctl status snort
```

Next, we do the same with Barnyard2. First create the service file:

```
sudo nano /lib/systemd/system/barnyard2.service
```

```
[Unit]
Description=Barnyard2 Daemon
After=syslog.target network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -q -w /var/log/snort/barnyard2.waldo -g snort -u snort -D -a /var/log/snort/archived_logs   (Note: From ExecStart to here is one line)

[Install]
WantedBy=multi-user.target
```

Enable service to start at boot:
sudo systemctl enable barnyard2

Start the service and check that it is running: (Fig 43)

```
sudo systemctl start barnyard2
```
```
sudo systemctl status barnyard2
```

![Fig 43 Barnyard2 running as service]

To install BASE which is done on the server we need an apache server, PEAR image graph and Adodb(a database abstraction layer for PHP).

```
sudo apt-get install -y apache2 libapache2-mod-php5 php5 php5-mysql php5-common php5-gd php5-cli php-pea
```
```
sudo pear install --alldeps Image_Graph
```

Adodb we need to download from sourceforge:

```
cd ~/source
wget https://sourceforge.net/projects/adodb/files/adodb-php5-only/adodb-5.20.8/adodb-5.20.8.tar.gz
tar -xzvf adodb-5.20.8.tar.gz
sudo mv adodb5 /var/adodb
sudo chmod -R 755 /var/adodb
```

and we find BASE at sourceforge too so we download and copy it to the apache root:

```
cd ~/source
wget https://sourceforge.net/projects/secureideas/files/BASE/base-1.4.5.tar.gz
tar -xzvf base-1.4.5.tar.gz
sudo mv base-1.4.5 /var/www/html/base/
```

Then we move to the base directory, create the config file based on a premade template and open it up in a text editor:

```
cd /var/www/html/base
sudo cp base_conf.php.dist base_conf.php
sudo nano /var/www/html/base/base_conf.php
```

The lines we need to edit are as follows:

```
Line 50:   $BASE_urlpath = '/base';
Line 80:   $DBlib_path = '/var/adodb/';
Line 102:  $alert_dbname = 'snort';
Line 103:  $alert_host = 'localhost';
Line 104:  $alert_port = '';
```
Line 105: $alert_user = 'snort';
Line 106: $alert_password = 'InsertYourMySQLPasswordHere';

Save the file and close. We also need to set some permissions on the BASE directory and hide the cleartext password.

```bash
sudo chown -R www-data:www-data /var/www/html/base
sudo chmod o-r /var/www/html/base/base_conf.php
```

After a restart of the apache server the BASE should now be working. The last configurations are done from the browser.

From your browser connect to your Raspberry Pi with this address:

http://IPaddressOfServer/base/index.php

Then click on setup page link. On the upper right corner is a button named “Create BASE AG”, click on that and then on main page. Now all is set and done.
10.6 Appendix F: List of work done by who

1. Terminology: Michael
2. Introduction: Michael/Andreas
3. Background: Michael/Andreas
4. Previous surveys in this area: Andreas
5. IDS Intrusion Detection System: Michael
6. Intrusion detection: Michael
7. Types of Intrusion Detection: Michael
8. Market analysis: Michael
9. Problem definition: Michael
10. Scope and goals: Michael
11. Research methodology: Michael
12. Implementation of proposed system: Michael
13. Hardware: Michael
14. Management server: Michael
15. Software: Michael/Andreas
16. Snort: Michael
17. Barnyard2: Andreas
18. Pulledpork: Andreas
19. Raspbian Lite: Andreas
20. BASE (Basic Analysis and Security Engine): Andreas
21. VPN: Michael
22. Hardware scenarios: Michael
23. Snort Installation on the Raspberry: Andreas
24. Results of testing: Michael
25. Discussion: Michael/Andreas
26. Conclusions: Michael/Andreas
27. Future work: Michael/Andreas
28. References: Andreas
29. Appendices: Andreas
30. Appendix A: Install of Raspbian Lite: Andreas
31. Appendix B: Installing Snort: Andreas
32. Dependencies: Andreas
33. Install DAQ (Data Acquisition Library): Andreas
34. Basic Configuration of Snort: Andreas
35. Appendix C: Installing Barnyard2: Andreas
36. Appendix D: Install of PulledPork: Andreas
37. Pulledpork configuration: Andreas
38. Appendix E: Make Snort and Barnyard2…: Andreas

Lot of work have been done together where Michael put more effort in hardware and server implementations and Andreas put more effort in the software implementations.