



***Investigation of Intrusion Detection and
Prevention Systems***

Jack Wilson

White Paper

Abertay University

BSc Ethical Hacking

2016/2017

ABSTRACT

An Intrusion Detection System (IDS) is a piece of software which monitors an individual computer (or an entire network) to detect malicious activity. Attacks sent by hackers are becoming increasingly harder to detect, so an IDS plays a crucial role within a network to keep data safe and prevent unwanted access from a malicious attacker.

This paper aims to look at the history of IDS's, explain the setup process for SecurityOnion (a blue-team focused Linux distribution), analyse potential ways to bypass an intrusion detection system and suggest counter-measures to the bypass.

While SecurityOnion was found to be generally simple to set up, it required some tweaking to the rules to properly secure the network from certain attacks and scans.

TABLE OF CONTENTS

Abstract	i
1. Introduction	1
1.1 Signature-Based Intrusion Detection	1
1.2 Anomaly-Based Intrusion Detection	2
1.3 Host-Based Intrusion Detection	2
1.4 Intrusion Prevention	3
2. Procedure	4
2.1 IDS Setup	4
2.2 Bypassing IDS	8
2.3 Writing Custom Snort Rules	11
2.4 Intrusion Prevention Configuration	13
3. Discussion and Conclusions	14
3.1 Counter-Measures	14
3.2 Discussion	14
3.3 Conclusion	16
3.4 Future Work	17
4. References	18
5. Appendices	20
5.1 Appendix 1: Classifications.	20
5.2 Appendix 2: DDoS Prevention Ruleset	22

TABLE OF FIGURES

Figure 1: VMware Guest Operating System Installation.....	4
Figure 2: SecurityOnion Installation Preparation.....	5
Figure 3: SecurityOnion Disk Erase Screen.....	6
Figure 4: User Account Setup.....	6
Figure 5: Evaluation or Production Mode Options.....	7
Figure 6: Snort Detecting Nmap Scan.....	8
Figure 7: Nmap Decoy Scan Results.....	9
Figure 8: Denial of Service Attack Against IDS.....	10
Figure 9: Tshark Monitoring of DoS Attack.....	11
Figure 10: Squert Web Interface.....	15
Figure 11: In-Depth Traffic Analysis.....	16

1. INTRODUCTION

Intrusion detection systems are a critical tool in a network to mitigate the risk of an attack. There are a few variants and sub-variants of intrusion detection systems. Simply, IDS's analyse and monitor computer activity and network traffic looking for malicious activity or anomalies to alert the system administrator to a potential attack.

The early variants of intrusion detection systems were first seen in the 1960's and were very simplistic. They would involve a network administrator sitting in front of a computer and monitoring for unusual activity such as a computer being more active than usual. This was the beginning of anomaly-based IDS where the network administrator would have an understanding of what network traffic typically looks like and look for significant abnormalities, however it was inefficient and not scalable for a large business.

In the 1970's, IDS's developed slightly. Logs of network and computer activity would be printed out (often thousands of pages per week depending on the size of the network) and a network administrator would have to analyse the logs to detect anomalies. This may have been marginally more efficient than someone monitoring a network constantly, however it was still time consuming and could result in an attack going unnoticed for some time depending on the frequency of log analysis. (SANS, 2002).

Even 40 years ago signature-based IDS still faced the same problem it faces today of only recognising known attacks. It would not prevent an unknown attack or a zero-day exploit. (Threat Stack, 2015).

Intrusion detection systems as they are known today with simple monitoring from a web interface only began to develop in the 1990's. They allowed for real-time monitoring of network activity to allow near-instantaneous threat response.

The first, and most common, IDS is called Network-Based Intrusion Detection. This system analyses the inbound traffic of a network and attempts to detect suspicious activity while additionally preventing wasted company time (i.e. monitoring if a user is spending hours streaming Netflix instead of working). There are two types of Network Intrusion Detection Systems:

1.1 SIGNATURE-BASED INTRUSION DETECTION

Signature-based IDS works very similar to most anti-virus software. It queries a database of known malicious signatures (e.g. signatures for port scans, attacks from programs like Metasploit and downloaded malware) and compares that to the network traffic. Although this system is very efficient at detecting known traffic, it is only as powerful as the database it is querying.

For example; an attacker may use a new payload to send an attack to a network. If this payload has not been used before (A zero-day exploit), or if the signature is tweaked slightly, it will not be in the database or have been flagged as being malicious, so the IDS would fail in preventing the attack.

Another potential issue with the signature-based IDS comes with the size of the database. If the database being queried is very large, then this may add some latency to the network. (SC Magazine, 2002). A potential workaround to this would be to implement more powerful hardware for the IDS (e.g. a more powerful CPU and extra RAM) that can execute queries faster and handle more queries at a time. A sensible estimate of the server requirements would need to be made by analysing the system requirements and the size of the network and configuring a server to run the IDS appropriately while leaving room for future expandability if required.

1.2 ANOMALY-BASED INTRUSION DETECTION

The second type of Network-Based IDS is called Anomaly-Based Intrusion Detection. The way this works is more complex than signature-based IDS. Simply put, this IDS takes a baseline measurement of what traffic typically looks like on a network, analyses all traffic passing through the network and detects abnormalities.

The IDS then works by capturing all HTTP headers entering the network and it can be set to filter out known traffic (for example web traffic to the web server, mail traffic going to the mail server and inbound/outbound traffic to a DNS server) based on user-defined rules. Intrusion Detection Systems typically come with common rules included, however additional rules can be added based on the requirements of the network.

In the event of an anomaly taking place, the IDS would alert the network administrator who would assess whether there was a legitimate threat or a false-positive and respond appropriately. (Alien Vault, 2014). For example, an attacker running an NMAP scan to find open ports on a server would be flagged as an anomaly and trigger the intrusion detection system.

1.3 HOST-BASED INTRUSION DETECTION

The alternative type of IDS is called Host-Based Intrusion Detection. This is installed locally on a PC/Workstation and it attempts to identify unusual behaviour on said device. HIDS works by monitoring the activity of the operating system and installed applications. It then works using a combination of anomaly and signature-based intrusion detection, along with some pre-defined rules to detect malicious activity.

The key point to note is that intrusion detection systems only detect and alert network administrators to malicious activity. An intrusion prevention system is required to prevent the malicious activity. (SANS, [no date]).

1.4 INTRUSION PREVENTION

An Intrusion Prevention System (IPS) works in a similar way to a Network-Based IDS in the fact it sits behind the network firewall (as opposed to being installed locally on workstations). It allows for traffic monitoring the same as an IDS would, however, it has a lot more power.

As well as alerting a network administrator to potential threats, the IPS also has the ability to drop or reject packets it deems to be malicious and even block traffic from the source IP of any malicious activity (similar to how a firewall would function).

An IPS will work using signatures, however there is a couple of different ways it will utilise the signatures: the first is through exploit-facing signature detection which looks for a specific signature of a known exploit in the network traffic stream to determine malicious activity. The second detection method is called vulnerability-facing signature detection. This method focusses more on the system's known vulnerabilities rather than the exploits themselves which allows for better detection of unknown variants of exploits, at the risk of an increase in false positive detections. (Palo Alto Networks, [no date]).

2. PROCEDURE

2.1 IDS SETUP

SecurityOnion is a Linux distribution based on Ubuntu 14.04. The setup process was designed to be as simple as possible, with the tools pre-configured and just requiring some basic network information such as IP address, default gateway, subnet address and DNS servers. It came pre-installed with some useful tools including:

- Snort and Suricata (network-based IDS's).
- OSSEC (a host-based IDS).
- netsniff (a packet capturing tool).
- Bro (transactional data logging for HTTP, FTP, DNS and SSL). (SecurityOnion, [no date])

The tool came as a bootable .iso image and was installed in VMware Workstation 12 for the purpose of research and investigation. In VMware select File>New Virtual Machine and choose the “Typical (Recommended)” installation type.

Select install from a .iso file and browse the PC for the bootable image.

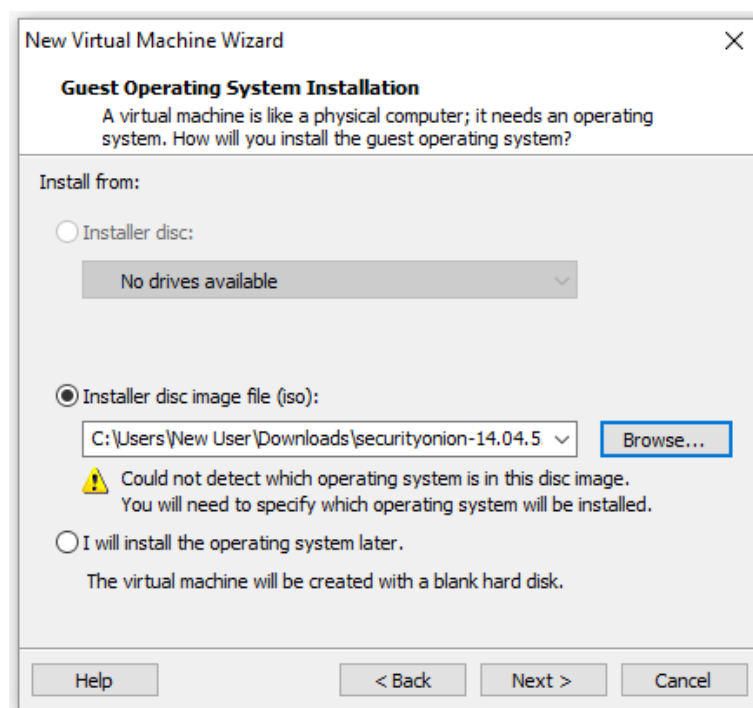


Figure 1: VMware Guest Operating System Installation

After clicking next, select the “Linux” radio button and choose “Ubuntu 64-bit” from the drop-down list. Name the virtual machine and select the installation folder. Click next and provide the

virtual machine with disk space (20GB is sufficient) and choose “Split virtual disk into multiple files”.

For testing purposes, I provided the virtual machine with two virtual CPU cores and 4GB of RAM, however in a production environment this will vary largely depending on the services running and the traffic throughput. The memory recommendation, for example, ranges from 8GB to 256GB depending on the size of the network. (GitHub, 2016). Select finish and power on the virtual machine.

Once the operating system has booted, choose the installation language and click continue. Leave the options as they are shown in *Figure 2* and click continue.

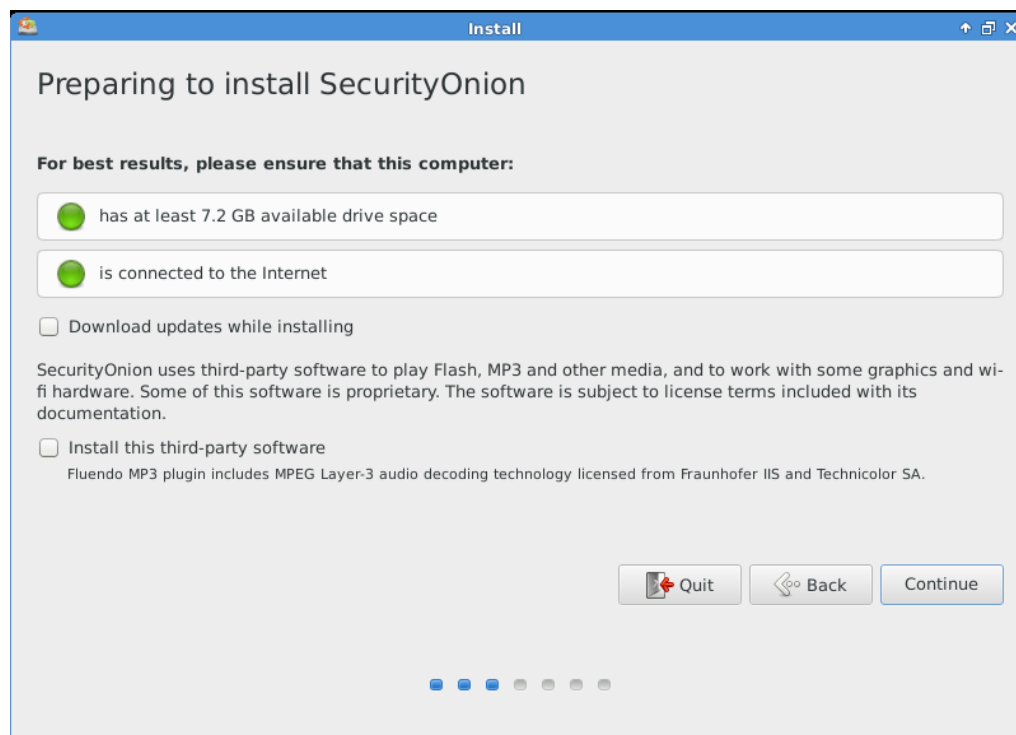


Figure 2: SecurityOnion Install Preparation

Choose the “Erase disk and install SecurityOnion” option and click “Install Now” as shown in *Figure 3*. Confirm the partitions to be erased.

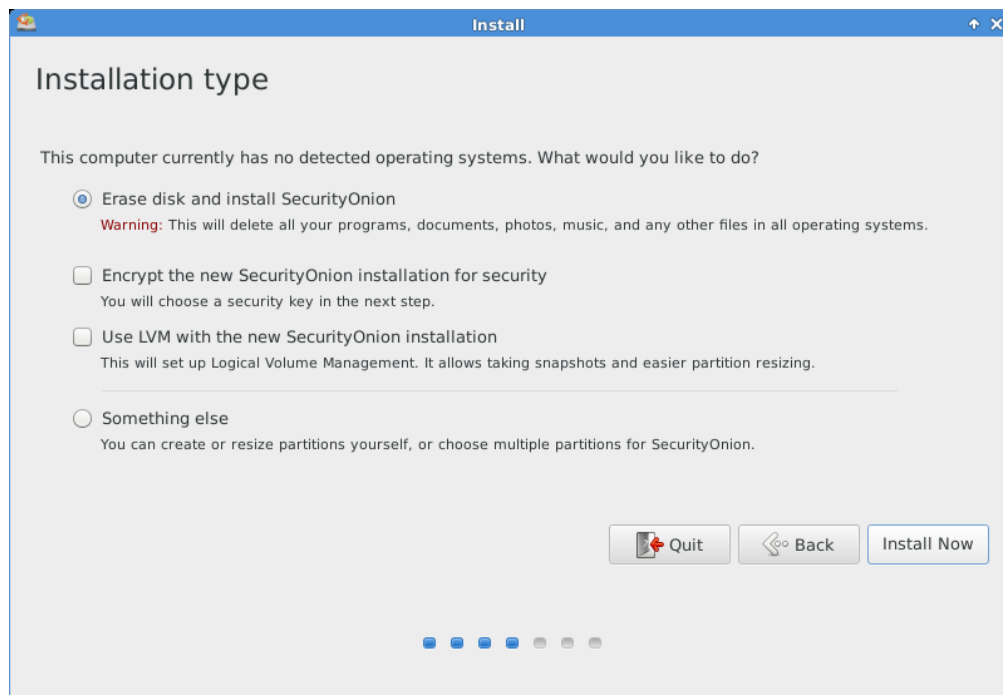


Figure 3: SecurityOnion Disk Erase Screen

Continue through the following steps selecting the appropriate time zone and keyboard layout. Fill in the user account details such as name, username and password. Ensure the “Encrypt my home folder” check box is **NOT** selected as shown in *Figure 4*. Click continue and the installation will finalise.

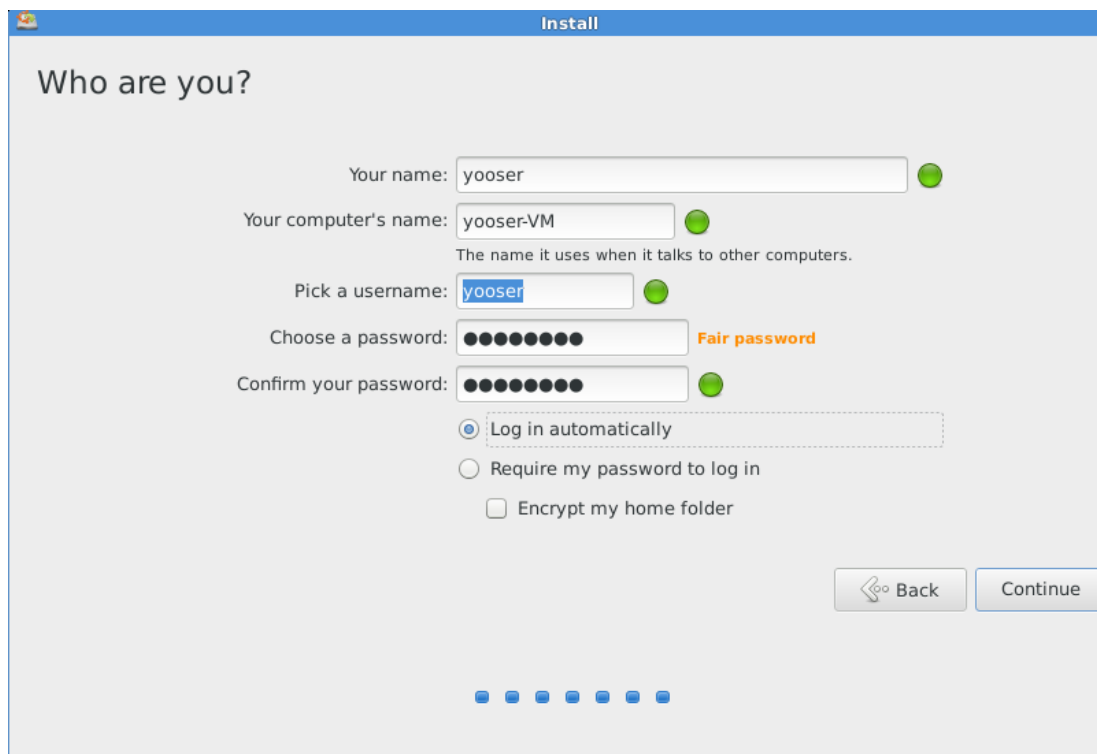


Figure 4: User Account Setup

After restarting the virtual machine, boot into the operating system and log in. Install system updates using the *sudo soup* command and reboot. (GitHub, 2016).

Double-click the *Setup* icon on the desktop. Click continue, followed by “Yes, configure /etc/network/interfaces”. Set the IP address to static and enter the IP address, subnet and default gateway specific to the local network. For example, the IP address in the example was 192.168.1.110, the subnet address was 255.255.255.0 and the gateway was 192.168.1.254. Enter DNS server(s), for example 8.8.8.8 8.8.4.4 which are Google’s DNS servers. Set the domain to 127.0.0.1 (localhost), confirm the changes and reboot the system.

After rebooting, double-click the *Setup* icon again, and continue with the rest of the setup. Select *Continue*, followed by *Yes, skip network configuration*. At this point two options are presented: evaluation mode and production mode. Evaluation mode is intended to be set up quickly and easily for beginner users, using Snort and Bro for network monitoring. Production mode is intended for a production environment and allows more control (such as a choice of Snort or Suricata for the IDS). This option can be viewed in *Figure 5* below.

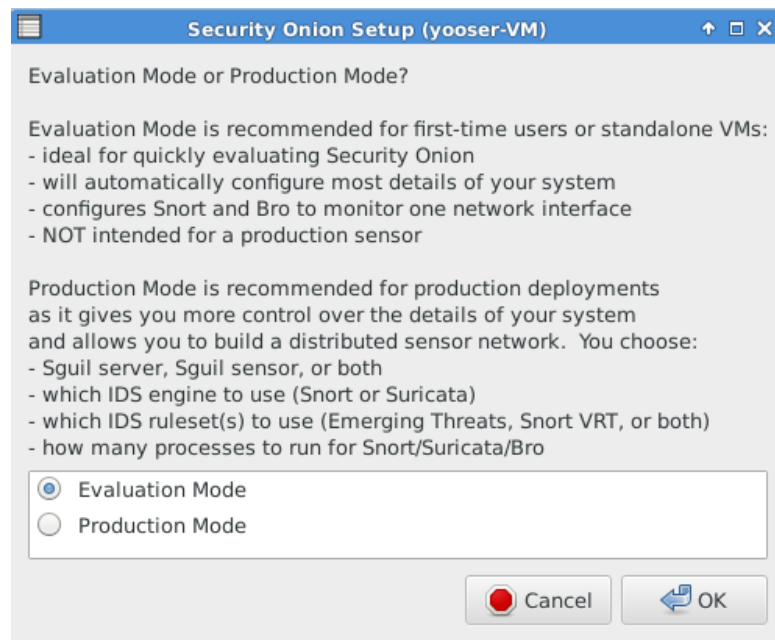


Figure 5: Evaluation of Production Mode Options

For experimentation purposes, evaluation mode was selected. Select the username for the various services, set a password and confirm the settings. The IDS is now fully set up and running. Clicking the *Squert* desktop icon launches the browser interface for the IDS. There will be a warning that the site may not be safe which is caused by the SSL certificate being self-signed. Select advanced and “Proceed to localhost (unsafe)”. The intrusion detection system is now set up and can be monitored through the web interface.

2.2 BYPASSING IDS

To test the configuration of the intrusion detection system an “attacker” machine running Kali Linux was set up in VMware. Some basic enumeration scans were ran using nmap that would indicate open ports, suggest possible services that were running and guess the operating system the IDS was running.

The first scan was a simple port scan of the first 4000 ports:

```
nmap -p 1-4000 192.168.45.140
```

Snort detected the scan, alerting that there was suspicious activity on certain ports (mostly ports associated with databases). An example can be seen in *Figure 6* below, showing suspicious activity on port 1521 and classifying the activity as “Attempted access”.

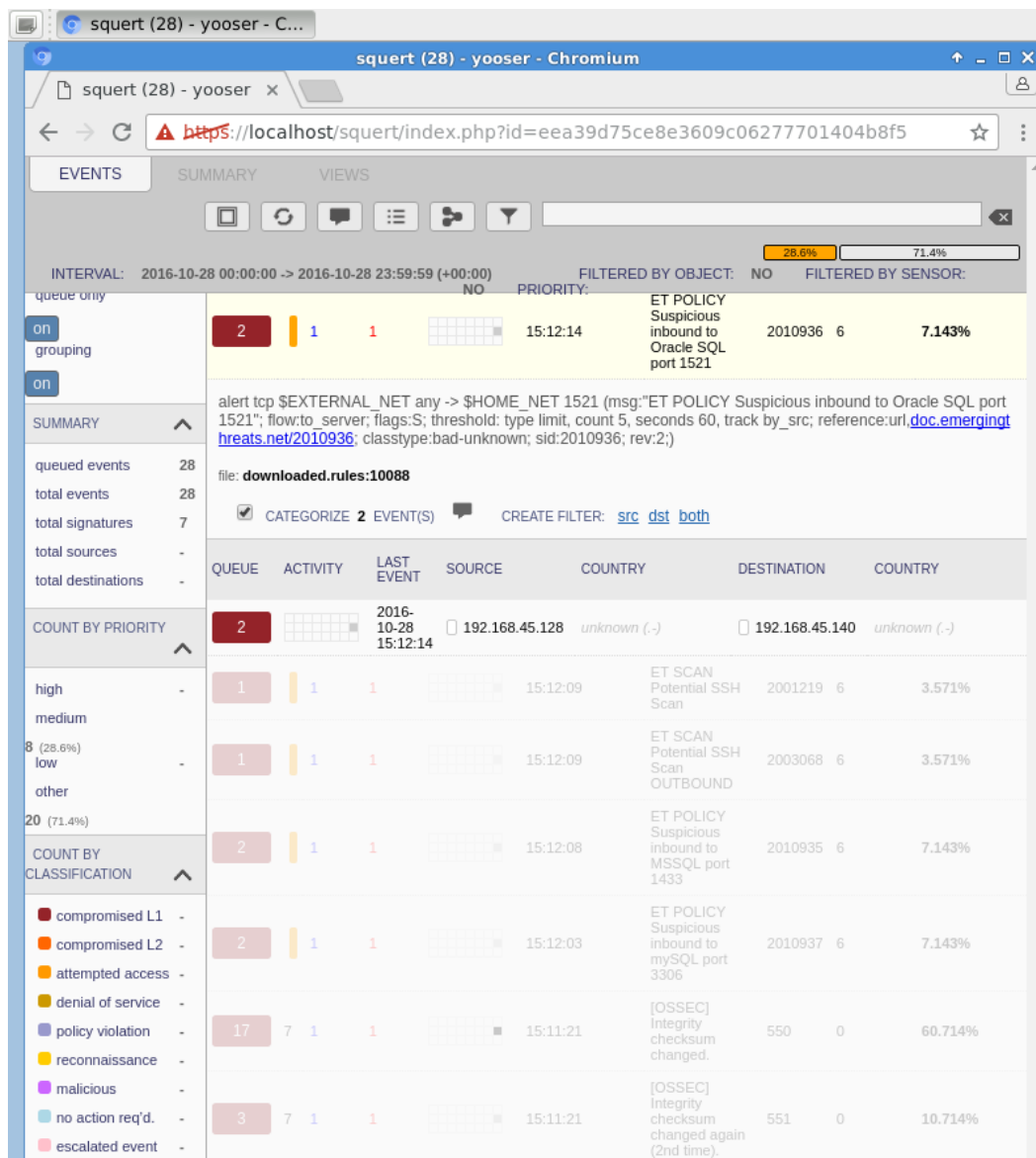


Figure 6: Snort Detecting Nmap Scan

The next scan ran was another port scan of the first 4000 ports, but with decoy IP addresses added. Analysing the scan in the web interface showed the scan was still detected and classified as attempted access, however Snort detected the scan coming from four different IP addresses: the attacker machine, google.com, microsoft.com and yahoo.com. The result can be viewed in *Figure 7* below, and the command used for the scan was:

```
nmap -p 1-4000 -D google.com,Microsoft.com,yahoo.com 192.168.45.140
```

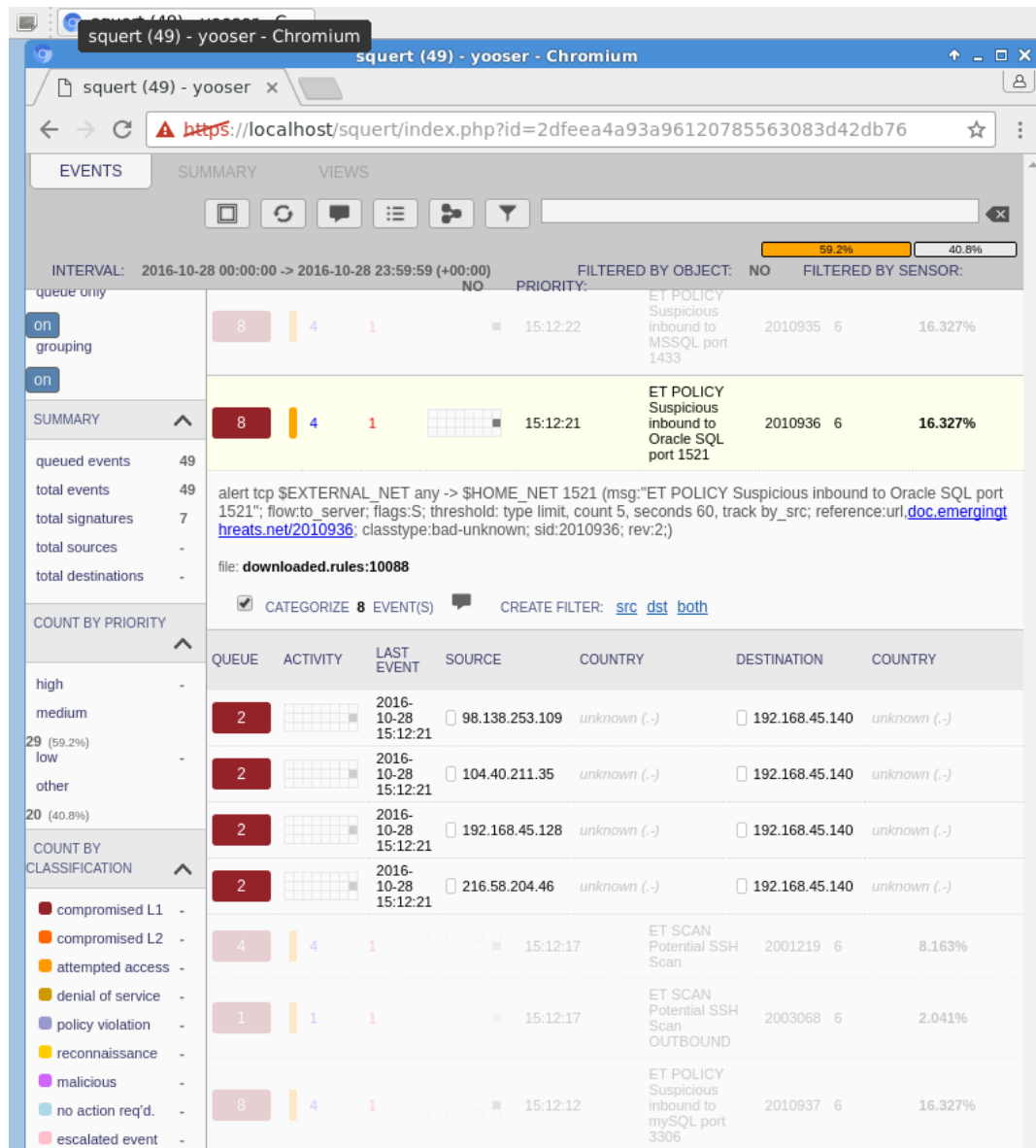


Figure 7: Nmap Decoy Scan Results

The next scan was, again, an nmap port scan, however the packets being sent were fragmented (sending only 8 bits at a time) to try and avoid detection. Once again, the scan was detected by Snort. The command used to fragment was:

```
nmap -mtu 8 -p 1-4000 192.168.45.140
```

Next, a TCP SYN scan was attempted. Once again, this was detected.

```
nmap -sS -p 1-4000 192.168.45.140
```

A service and OS detection scan was ran against SecurityOnion to try and guess which operating system was running. This scan can be seen below and was picked up by nmap.

```
nmap -sV -o 192.168.45.140
```

An “Xmas” scan was the next scan attempted. The scan sent a TCP frame that contains URG, PUSH and FIN flags. This scan managed to successfully evade the IDS, with no record of it happening on the IDS. The command that successfully evaded Snort was:

```
nmap -sX -p 1-4000 192.168.45.140
```

Not just the Xmas scan evaded Snort; FIN, NULL and ACK scans all bypassed the IDS. This is due to the type of traffic being uncommon on a network so no rules are present in Snort by default. (Jammes and Papadaki, [no date]).

The next stage of testing (outside of basic enumeration) was a denial of service (DoS) attack. The resource allocation for the IDS was lowered from two virtual CPU cores to one and the RAM reduced from 4GB to 1GB. The command *hping3 -flood 192.168.45.140* was then ran in multiple terminal windows simultaneously, overloading the IDS and causing a “denial of service” attack. This attack can be viewed in *Figure 8* and *Figure 9*, below.

The attack was running for 60 seconds and roughly 10,500,000 packets were sent – meaning the IDS had to process roughly 175,000 packets per second. Despite this, no alerts were shown in the IDS web interface.

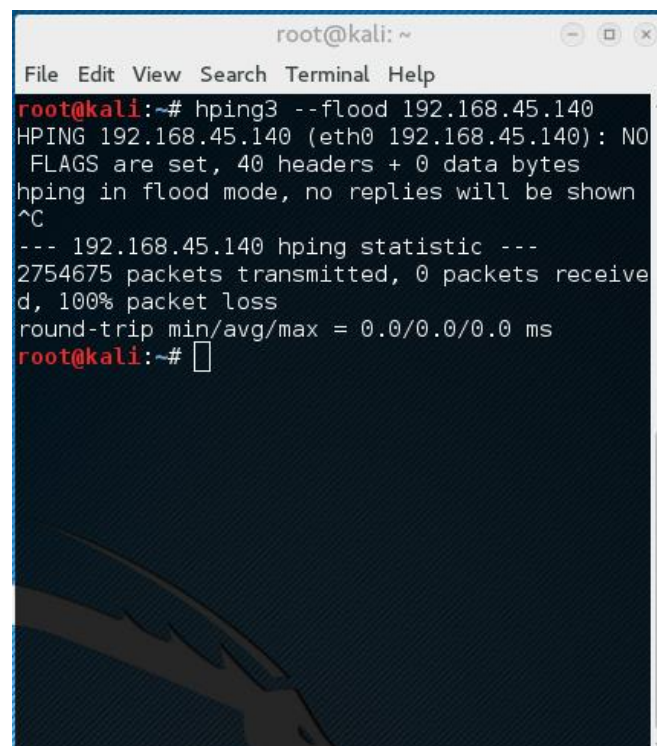
A screenshot of a terminal window titled 'root@kali: ~'. The terminal shows the execution of the command 'hping3 --flood 192.168.45.140'. The output indicates that the flood is in progress, with 2754675 packets transmitted and 0 received, resulting in 100% packet loss. The round-trip time is shown as 0.0/0.0/0.0 ms. The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The background of the terminal is dark blue with a subtle pattern.

Figure 8: Denial of Service Attack Against IDS


```

root@kali: ~
File Edit View Search Terminal Help
1817488 62.954428714 192.168.45.143 -> 192.168.45.140 TCP 54 34478 -> 0 [<None>] Seq=504930204 Win=512 Len=0
1817489 62.954440387 192.168.45.143 -> 192.168.45.140 TCP 54 34479 -> 0 [<None>] Seq=3648564690 Win=512 Len=0
1817490 62.954449917 192.168.45.143 -> 192.168.45.140 TCP 54 34480 -> 0 [<None>] Seq=64075342 Win=512 Len=0
1817491 62.954462067 192.168.45.143 -> 192.168.45.140 TCP 54 34481 -> 0 [<None>] Seq=316781513 Win=512 Len=0
1817492 62.954471400 192.168.45.143 -> 192.168.45.140 TCP 54 34482 -> 0 [<None>] Seq=763414118 Win=512 Len=0
1817493 62.954483475 192.168.45.143 -> 192.168.45.140 TCP 54 34483 -> 0 [<None>] Seq=364165286 Win=512 Len=0
1817494 62.954492839 192.168.45.143 -> 192.168.45.140 TCP 54 34484 -> 0 [<None>] Seq=1345721501 Win=512 Len=0
1817495 62.954504967 192.168.45.143 -> 192.168.45.140 TCP 54 34485 -> 0 [<None>] Seq=1786250701 Win=512 Len=0
1817496 62.954514683 192.168.45.143 -> 192.168.45.140 TCP 54 34486 -> 0 [<None>] Seq=2457269299 Win=512 Len=0
1817497 62.954526844 192.168.45.143 -> 192.168.45.140 TCP 54 34487 -> 0 [<None>] Seq=253168552 Win=512 Len=0
1817498 62.954536340 192.168.45.143 -> 192.168.45.140 TCP 54 34488 -> 0 [<None>] Seq=3596384335 Win=512 Len=0
1817499 62.954548505 192.168.45.143 -> 192.168.45.140 TCP 54 34489 -> 0 [<None>] Seq=1971439803 Win=512 Len=0
1817500 62.954557803 192.168.45.143 -> 192.168.45.140 TCP 54 34490 -> 0 [<None>] Seq=1146165282 Win=512 Len=0
1817501 62.954569829 192.168.45.143 -> 192.168.45.140 TCP 54 34491 -> 0 [<None>] Seq=3931295685 Win=512 Len=0
1817502 62.954679422 192.168.45.143 -> 192.168.45.140 TCP 54 34492 -> 0 [<None>] Seq=3506226005 Win=512 Len=0
1817503 62.954700723 192.168.45.143 -> 192.168.45.140 TCP 54 34493 -> 0 [<None>] Seq=142430540 Win=512 Len=0
1817504 62.954710283 192.168.45.143 -> 192.168.45.140 TCP 54 34494 -> 0 [<None>] Seq=1736642944 Win=512 Len=0
1817505 62.954722627 192.168.45.143 -> 192.168.45.140 TCP 54 34495 -> 0 [<None>] Seq=3384675878 Win=512 Len=0
1817506 62.954732063 192.168.45.143 -> 192.168.45.140 TCP 54 34496 -> 0 [<None>] Seq=1571585450 Win=512 Len=0
1817507 64.792450916 192.168.45.1 -> 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
1817508 65.793385866 192.168.45.1 -> 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
1817509 66.803877205 192.168.45.1 -> 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
1817510 67.793606438 192.168.45.1 -> 239.255.255.250 SSDP 215 M-SEARCH * HTTP/1.1
1817511 73.294626539 192.168.45.140 -> 216.58.214.14 TCP 60 [TCP ACKed unseen segment] [TCP Previous segment not captured] 53130 -> 443 [ACK] Seq=1854 Ack=2227 Win=65535 Len=0
1817512 73.294631962 216.58.214.14 -> 192.168.45.140 TCP 60 [TCP ACKed unseen segment] [TCP Previous segment not captured] 443 -> 53130 [ACK] Seq=2227 Ack=1855 Win=64240 Len=0
1817513 73.393180006 192.168.45.140 -> 216.58.214.3 TCP 60 [TCP Keep-Alive] 56292 -> 80 [ACK] Seq=302 Ack=159287 Win=134215680 Len=0
1817514 73.393186633 216.58.214.3 -> 192.168.45.140 TCP 60 [TCP Keep-Alive ACK] 80 -> 56292 [ACK] Seq=159287 Ack=303 Win=64240 Len=0
1817515 77.900809689 Vmware_bd:06:13 -> Vmware_e2:2e:f8 ARP 60 Who has 192.168.45.2? Tell 192.168.45.140
1817516 77.900841559 Vmware_e2:2e:f8 -> Vmware_bd:06:13 ARP 60 192.168.45.2 is at 00:50:56:e2:2e:f8
1817517 95.135834675 172.217.23.14 -> 192.168.45.140 TCP 60 443 -> 59360 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1817518 95.393550704 216.58.214.14 -> 192.168.45.140 TCP 60 443 -> 54264 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
^C7114064 packets dropped
1817518 packets captured
root@kali:~#

```

Figure 9: Tshark Monitoring of DoS Attack

Once it was established that a DoS attack was undetectable, a basic port scan using *nmap -p 1-4000 192.168.45.140* was attempted during a second DoS attack. The port scan was detected in the exact same manner as shown previously, so the DoS attack was unsuccessful in assisting reconnaissance scans to evade the IDS.

2.3 WRITING CUSTOM SNORT RULES

Custom rules can also be written by anyone and added to the local.rules file. (O' Reilly, 2007). Rules must follow the format:

action protocol source_IP source_port direction destination_IP destination_port (options)

Action

There are eight options for action:

- Alert – Cause an alert on the IDS and create a log for the packet
- Log – Only log the packet
- Pass – Ignore the packet
- Activate – Alert as above and then activate another dynamic rule
- Dynamic – Do not activate automatically unless triggered by an activate rule
- Drop – Block and log the packet
- Reject – Block and log the packet. If the protocol is TCP then send a TCP reset. If the protocol is UDP send an ICMP port unreachable message.
- Sdrop – Block the packet and do not log

Protocol

Tell the rule which protocol to look for – either TCP or UDP.

Source IP

Specify which source IP address will trigger the rule. Either a specific address in the dot decimal notation (e.g. 192.168.1.1) can be used or the word “any” can be used which will trigger the rule for any IP address passing through the network.

Source Port

Specify which port triggers the rule (e.g. port 22 would trigger a rule for an SSH connection). This can be set to a specific port or use “any” for traffic travelling through any port. A range of ports can also be used using a colon (e.g. 1:1024 for all ports in that range or 1024: for every port greater than 1024).

Direction

Specify the direction in which the packets are travelling. For example -> would represent outbound traffic, <- would represent inbound traffic and <> would represent traffic in any direction.

Destination IP

Allow the rule to be triggered if traffic is travelling to/from the specified IP address. (This could, for example, be used to block a malicious website).

Destination Port

Allow the rule to be triggered if traffic is travelling to/from the specified port. As with the source port this can be a single port or a range of ports.

Options

There are several options that can be added to the rule that have various functions.

- msg – This option tells the IDS what message to print alongside an alert. It would be formatted as: *msg: “Error”;*
- reference – This option allows rules to include references to external systems to identify attacks such as Microsoft’s Technet website, Nessus, CVE and McAfee. Formatted as *reference:mcafee,<link>;*
- gid – This part of the rule is used to tell the rule where in Snort the rule was triggered from. Format: *gid:<number>;*
- sid – A unique identifier for each Snort rule. Format: *sid:<number>;*

- **rev** – Another unique identifier for tracking revisions of Snort rules. Format: *rev:<number>;*.
- **classtype** – Used to classify a Snort event if the rule is triggered. For example, an nmap scan would be classified as *attempted recon*. The most common class types are *compromised*, *attempted access*, *denial of service*, *policy violation*, *reconnaissance* and *malicious*, however, there are almost forty different classifications that can be found in *Appendix 1*. The format of the classification option is: *classtype:<class name>;*. Classifications can be added or updated by editing the *classification.config* file.
- **priority** – This option will assign a level of priority to a rule that would override a priority level set with the *classtype* option. Usage: *priority:<integer>;*.
- **metadata** – This can be miscellaneous extra information associated with the rule in the format: *metadata:key1 value1, key2 value2;;* (Snort, [no date]).

2.4 INTRUSION PREVENTION CONFIGURATION

As well as offering intrusion detection Snort also offers intrusion prevention. This can include features such as dropping packets that are known to be malicious or entirely blocking malicious IP addresses (similar to how a firewall would function). Like the detection, the IPS is configured through rules. If, for example, the administrator wanted all XMAS scans to be dropped entirely instead of causing an alert then the rule would need to be edited from:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN XMAS"; flow:stateless;  
flags:SRAFPU,12; reference:arachnids,144; classtype:attempted-recon; sid:625; rev:7;)
```

to:

```
drop tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN XMAS"; flow:stateless;  
flags:SRAFPU,12; reference:arachnids,144; classtype:attempted-recon; sid:625; rev:7;)
```

The snort service would need to be restarted for the change(s) to take effect, but after the changes are effective Snort would immediately drop all packets relating to an XMAS scan.

Another task a network administrator may want to do is to block access to specific websites (such as Facebook to prevent wasted time) or websites that are known to be malicious. Based on the formatting of rules shown above, the rule to block an IP address would be:

```
Drop tcp any any -> 193.60.168.13 any (msg:"Blocked!"; sid:9999111; classtype:policy-  
violation;)
```

3. DISCUSSION AND CONCLUSIONS

3.1 COUNTER-MEASURES

Snort is generally well maintained by both the development team and the community. However, some rules may be left out of releases either by accident or for performance improvements (less rules to be checked against will result in better performance).

It is important to find a balance between security and performance when configuring Snort, so it is recommended to do some research and decide which rules may be beneficial to add. For Snort to alert for NULL, FIN or XMAS scans that went undetected above the following rules would need to be added to the local.rules file located in the /etc/nsm/rules directory:

- *alert tcp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"SCAN NULL"; flow:stateless; ack:0; flags:0; seq:0; reference:arachnids,4; classtype:attempted-recon; sid:623; rev:6;)*
- *alert tcp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"SCAN SYN FIN"; flow:stateless; flags:SF,12; reference:arachnids,198; classtype:attempted-recon; sid:624; rev:7;)*
- *alert tcp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"SCAN XMAS"; flow:stateless; flags:SRAFP,12; reference:arachnids,144; classtype:attempted-recon; sid:625; rev:7;)*

The above rules came from a community member on GitHub who has collated hundreds of rules relating to bad traffic, denial of service, DNS, FTP, POP2/3 and many other services. (GitHub, 2011).

To allow for a denial of service attack to be detected would require more rules to be added to the intrusion detection system. The following ruleset has a total of thirty rules which should cover most bases for a DDoS attack. An example rule can be viewed below and the entire ruleset can be viewed in *Appendix 2*. (GitHub, 2011).

Alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS TFN Probe"; icmp_id:678; itype:8; content:"1234"; reference:arachnids,443; classtype:attempted-recon; sid:221; rev:4;)

3.2 DISCUSSION

The initial testing showed Snort to be mostly effective at monitoring attacks. The tests included a port scan using Nmap (A network discovery and security auditing tool (Nmap, [no date])). A scan from an attacker machine to determine open ports and guess which operating system was running on the intrusion detection system and a denial of service attack.

Not only did Snort detect the port scan and OS detection scanning (both under normal traffic and under a denial of service attack), it classified scans into different threat levels, depending on the

risk posed to the system. A preview of the Squert web interface detecting and classifying the risks can be viewed in *Figure 10*.

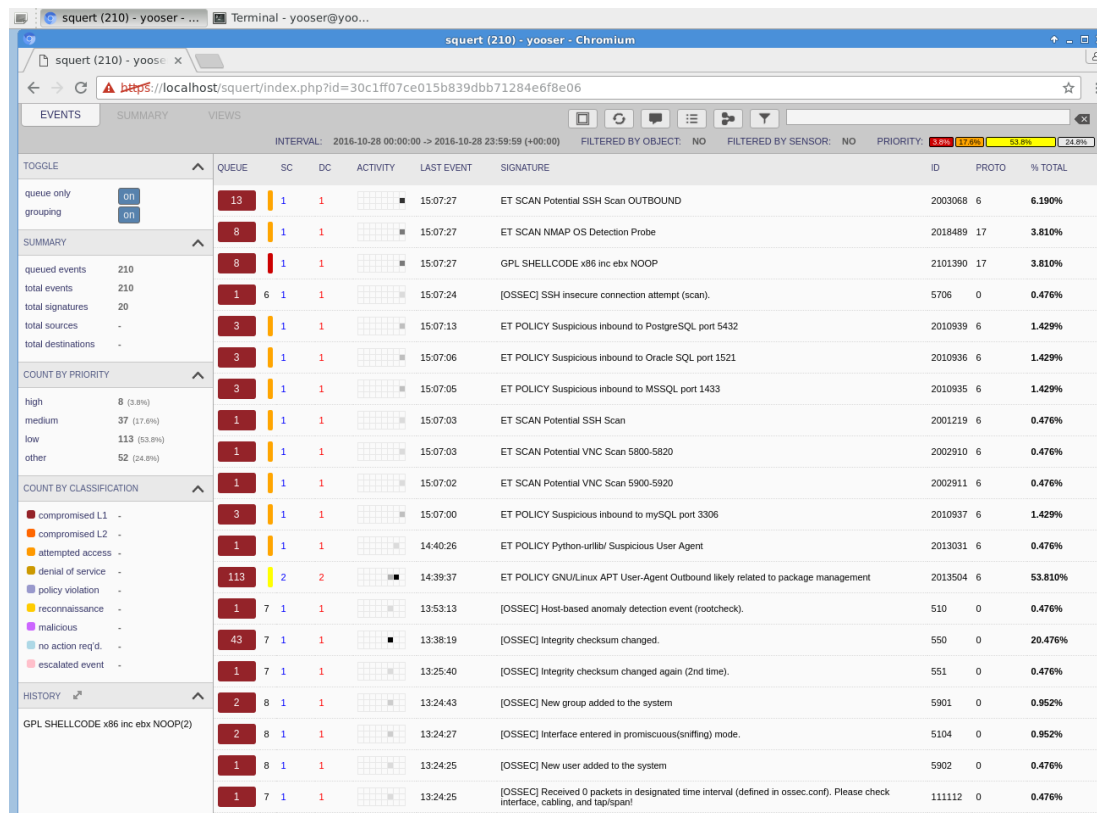


Figure 10: Squert Web Interface

The web interface and database combination also allowed for in-depth analysis of the traffic including the priority level of the traffic, the protocol used, source and destination IP addresses and ports the traffic was being transmitted on, as shown in *Figure 11* below:

Query: "2018489" program="snort"

From: 2016-10-26 15:11:55 To: UTC Add Term Report On Index Reuse current tab Grid display

2018489 groupby:program (1) [Grouped by program] X "2018489" program="snort" (8) X

Result Options... Field Summary

Records: 8 / 8 26 ms 2 << first < prev 1 next > last >> 15 ▾

Timestamp	Fields
Fri Oct 28 15:07:24	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:38126 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=38126 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:24	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:38126 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=38126 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:24	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:38126 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=38126 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:24	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:38126 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=38126 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:26	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:32853 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=32853 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:27	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:32853 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=32853 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:27	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:32853 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=32853 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=
Fri Oct 28 15:07:28	[1:2018489:3] ET SCAN NMAP OS Detection Probe [Classification: Attempted Information Leak] [Priority: 2]: (UDP) 192.168.45.146:47961 -> 192.168.45.140:32853 host=127.0.0.1 program=snort class=SNORT sig_priority=2 proto=UDP srcip=192.168.45.146 srcport=47961 dsnip=192.168.45.140 dstport=32853 sig_sid=1:2018489:3 sig_msg=ET SCAN NMAP OS Detection Probe sig_classification=Attempted Information Leak interface=

Records: 8 / 8 26 ms 2 << first < prev 1 next > last >> 15 ▾

Figure 11: In-Depth Traffic Analysis

The DoS attack against the intrusion detection system went undetected, however there is no certainty the attack was even successful. Pinging a web server from the IDS showed no additional latency while under DoS attack compared to normal operating circumstances.

Furthermore, adding the rules found in *Appendix 2* made no difference, and the DoS caused no alert in the IDS web interface. It may be that a traditional ICMP flood attack that was demonstrated was ineffective against modern systems with gigabit bandwidth capabilities. A more modern, complex DoS (perhaps even a distributed denial of service attack) may be required to saturate the connection enough to cause problems with the IDS.

3.3 CONCLUSION

Snort has proven to be a very successful security tool, and it could be very beneficial in a home, small business or large enterprise network. There were a few different options for the installation. Firstly, Snort could be setup from scratch in a pre-existing environment such as Ubuntu, CentOS or even Windows. This method required the most time and knowledge to set up.

The second option was a package called Autosnort. This was a community-developed series of bash scripts that would semi-automate the installation by installing Snort, Barnyard and several other required programs and repositories automatically, just requiring the user to edit a configuration file (this method would only work on a Linux-based system). (GitHub, 2016). This option was attempted and found to be fairly simple to get up-and-running. However, the web interface was being viewed over HTTPS and the SSL certificate provided by the Autosnort

creator was expired at the time of writing so the web interface could not be viewed, essentially rendering the IDS useless.

The third option (that was used) was Security Onion. This is a defensive security-focused Linux distribution built on Ubuntu 14.04 and was by far the simplest of the three to set up. It was simply a case of setting up the virtual machine and entering the network configuration and user account details into a GUI. This process took no longer than fifteen minutes to have Snort fully functional.

Granted, some more work was required to test, investigate, tweak and optimise the rules after setup, however this would be true regardless of which of the three Snort setup processes was used.

3.4 FUTURE WORK

Given more time, some clients and servers could have been added to the virtual network along with some services (such as FTP or email) to test how effectively the previously mentioned community rules for those services work using multiple network interfaces.

Some more enumeration and exploits could have been ran against the intrusion detection system using Windows Powershell (using modules such as Powersploit) to test Snort's effectiveness against that system.

Finally, more time could be spent investigating denial of service attacks to create a complex enough DoS attack to cause disruption and be detectable by Snort.

4. REFERENCES

SC Magazine. Signature-Based or Anomaly-Based Intrusion Detection: The Practice and Pitfalls. 1 May 2002. Accessed 21 October 2016 at <http://www.scmagazine.com/signature-based-or-anomaly-based-intrusion-detection-the-practice-and-pitfalls/article/30471/>

Alien Vault. Open Source Intrusion Detection Tools: A Quick Overview. 13 January 2014. Accessed 21 October at <https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview>

SANS. IDFAQ: What is Intrusion Detection? [no date]. Accessed 24 October 2016 at <https://www.sans.org/security-resources/idfaq/what-is-intrusion-detection/1/1>

Palo Alto Networks. What is an intrusion prevention system? [no date]. Accessed 24 October 2016 at <https://www.paloaltonetworks.com/documentation/glossary/what-is-an-intrusion-prevention-system-ips>

SecurityOnion. About SecurityOnion. [no date]. Accessed 28 October 2016 at <https://securityonion.net/#about>

GitHub. Hardware. 21 September 2016. Accessed 28 October 2016 at <https://github.com/Security-Onion-Solutions/security-onion/wiki/Hardware>

GitHub. Upgrade. 21 September 2016. Accessed 28 October 2016 at <https://github.com/Security-Onion-Solutions/security-onion/wiki/Upgrade>

SANS. Intrusion Detection: A Brief History and Overview. April 2002. Accessed 29 October 2016 at <https://www.computer.org/csdl/mags/co/2002/04/r4s27.pdf>

Threat Stack. The History of Intrusion Detection Systems (IDS) – Part 1. 9 September 2015. Accessed 29 October 2016 at <http://blog.threatstack.com/the-history-of-intrusion-detection-systems-ids-part-1>

Nmap. About Nmap. [no date]. Accessed 29 October 2016 at <https://nmap.org>

Jammes, Z and Papadaki, M. Snort IDS Ability to Detect Nmap and Metasploit Framework Evasion Techniques. [no date]. Accessed 6 November 2016 at <https://www.cscan.org/download/?id=918>

GitHub. eldondev / Snort / rules. 1 June 2011. Accessed 8 November 2016 at <https://github.com/eldondev/Snort/tree/master/rules>

O' Reilly. Write Your Own Snort Rules. 2007. Accessed 9 November 2016 at <http://archive.oreilly.com/pub/h/1393>

Snort. User's Manual 2.9.8.3. [no date]. Accessed 9 November 2016 at <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node31.html>

GitHub. eldondev / Snort / rules / ddos.rules. 1 June 2011. Accessed 9 November 2016 at <https://github.com/eldondev/Snort/blob/master/rules/ddos.rules>

GitHub. da667 / Autosnort. 8 November 2016. Accessed 10 November 2016 at
<https://github.com/da667/Autosnort>

5. APPENDICES

5.1 APPENDIX 1: CLASSIFICATIONS.

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
inappropriate-content	Inappropriate Content was Detected	high
policy-violation	Potential Corporate Privacy Violation	high
shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high
attempted-dos	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
default-login-attempt	Attempt to login by a default username and password	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium
rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium

successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-detect	A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious username was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	Access to a potentially vulnerable web application	medium
icmp-event	Generic ICMP event	low
misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low
tcp-connection	A TCP connection was detected	very low

5.2 APPENDIX 2: DDOS PREVENTION RULESET

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS TFN Probe";
icmp_id:678; itype:8; content:"1234"; reference:arachnids,443; classtype:attempted-recon;
sid:221; rev:4;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS tfn2k icmp possible
communication"; icmp_id:0; itype:0; content:"AAAAAAAAAAAA"; reference:arachnids,425;
classtype:attempted-dos; sid:222; rev:2;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 31335 (msg:"DDOS Trin00 Daemon to
Master PONG message detected"; content:"PONG"; reference:arachnids,187;
classtype:attempted-recon; sid:223; rev:3;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS TFN client command
BE"; icmp_id:456; icmp_seq:0; itype:0; reference:arachnids,184; classtype:attempted-dos;
sid:228; rev:3;)

alert tcp \$HOME_NET 20432 -> \$EXTERNAL_NET any (msg:"DDOS shaft client login to
handler"; flow:from_server,established; content:"login|3A|"; reference:arachnids,254;
reference:url,security.royans.net/info/posts/bugtraq_ddos3.shtml; classtype:attempted-dos;
sid:230; rev:5;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 18753 (msg:"DDOS shaft handler to agent";
content:"alive tijgu"; reference:arachnids,255; classtype:attempted-dos; sid:239; rev:2;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 20433 (msg:"DDOS shaft agent to handler";
content:"alive"; reference:arachnids,256; classtype:attempted-dos; sid:240; rev:2;)

alert tcp \$HOME_NET any <> \$EXTERNAL_NET any (msg:"DDOS shaft synflood";
flow:stateless; flags:S,12; seq:674711609; reference:arachnids,253; reference:cve,2000-0138;
classtype:attempted-dos; sid:241; rev:10;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 31335 (msg:"DDOS Trin00 Daemon to
Master message detected"; content:"l44"; reference:arachnids,186; classtype:attempted-dos;
sid:231; rev:3;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 31335 (msg:"DDOS Trin00 Daemon to
Master *HELLO* message detected"; content:"*HELLO*"; reference:arachnids,185;
reference:url,www.sans.org/newlook/resources/IDFAQ/trinoo.htm; classtype:attempted-dos;
sid:232; rev:5;)

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 27665 (msg:"DDOS Trin00 Attacker to
Master default startup password"; flow:established,to_server; content:"betaalmostdone";
reference:arachnids,197; classtype:attempted-dos; sid:233; rev:3;)

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 27665 (msg:"DDOS Trin00 Attacker to Master default password"; flow:established,to_server; content:"gOrave"; classtype:attempted-dos; sid:234; rev:2;)

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 27665 (msg:"DDOS Trin00 Attacker to Master default mdie password"; flow:established,to_server; content:"killme"; classtype:bad-unknown; sid:235; rev:2;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 27444 (msg:"DDOS Trin00 Master to Daemon default password attempt"; content:"l44adsl"; reference:arachnids,197; classtype:attempted-dos; sid:237; rev:2;)

alert icmp \$HOME_NET any -> \$EXTERNAL_NET any (msg:"DDOS TFN server response"; icmp_id:123; icmp_seq:0; itype:0; content:"shell bound to port"; reference:arachnids,182; classtype:attempted-dos; sid:238; rev:6;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 6838 (msg:"DDOS mstream agent to handler"; content:"newserver"; classtype:attempted-dos; sid:243; rev:2;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 10498 (msg:"DDOS mstream handler to agent"; content:"stream/"; reference:cve,2000-0138; classtype:attempted-dos; sid:244; rev:3;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 10498 (msg:"DDOS mstream handler ping to agent"; content:"ping"; reference:cve,2000-0138; classtype:attempted-dos; sid:245; rev:3;)

alert udp \$EXTERNAL_NET any -> \$HOME_NET 10498 (msg:"DDOS mstream agent pong to handler"; content:"pong"; classtype:attempted-dos; sid:246; rev:2;)

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 12754 (msg:"DDOS mstream client to handler"; flow:to_server,established; content:">"; reference:cve,2000-0138; classtype:attempted-dos; sid:247; rev:4;)

alert tcp \$HOME_NET 12754 -> \$EXTERNAL_NET any (msg:"DDOS mstream handler to client"; flow:to_client,established; content:">"; reference:cve,2000-0138; classtype:attempted-dos; sid:248; rev:4;)

alert tcp \$EXTERNAL_NET any -> \$HOME_NET 15104 (msg:"DDOS mstream client to handler"; flow:stateless; flags:S,12; reference:arachnids,111; reference:cve,2000-0138; classtype:attempted-dos; sid:249; rev:8;)

alert tcp \$HOME_NET 15104 -> \$EXTERNAL_NET any (msg:"DDOS mstream handler to client"; flow:from_server,established; content:">"; reference:cve,2000-0138; classtype:attempted-dos; sid:250; rev:4;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS - TFN client command LE"; icmp_id:51201; icmp_seq:0; itype:0; reference:arachnids,183; classtype:attempted-dos; sid:251; rev:3;)

alert icmp 3.3.3.3/32 any -> \$EXTERNAL_NET any (msg:"DDOS Stacheldraht server spoof"; icmp_id:666; itype:0; reference:arachnids,193; classtype:attempted-dos; sid:224; rev:3;)

alert icmp \$HOME_NET any -> \$EXTERNAL_NET any (msg:"DDOS Stacheldraht gag server response"; icmp_id:669; itype:0; content:"sicken"; reference:arachnids,195; classtype:attempted-dos; sid:225; rev:6;)

alert icmp \$HOME_NET any -> \$EXTERNAL_NET any (msg:"DDOS Stacheldraht server response"; icmp_id:667; itype:0; content:"ficken"; reference:arachnids,191; classtype:attempted-dos; sid:226; rev:6;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS Stacheldraht client spoofworks"; icmp_id:1000; itype:0; content:"spoofworks"; reference:arachnids,192; classtype:attempted-dos; sid:227; rev:6;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS Stacheldraht client check gag"; icmp_id:668; itype:0; content:"gesundheit!"; reference:arachnids,194; classtype:attempted-dos; sid:236; rev:6;)

alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"DDOS Stacheldraht client check skillz"; icmp_id:666; itype:0; content:"skillz"; reference:arachnids,190; classtype:attempted-dos; sid:229; rev:5;)

alert icmp \$EXTERNAL_NET any <> \$HOME_NET any (msg:"DDOS Stacheldraht handler->agent niggahbitch"; icmp_id:9015; itype:0; content:"niggahbitch"; reference:url,staff.washington.edu/dittrich/misc/stacheldraht.analysis; classtype:attempted-dos; sid:1854; rev:7;)

alert icmp \$EXTERNAL_NET any <> \$HOME_NET any (msg:"DDOS Stacheldraht agent->handler skillz"; icmp_id:6666; itype:0; content:"skillz"; reference:url,staff.washington.edu/dittrich/misc/stacheldraht.analysis; classtype:attempted-dos; sid:1855; rev:7;)

alert icmp \$EXTERNAL_NET any <> \$HOME_NET any (msg:"DDOS Stacheldraht handler->agent ficken"; icmp_id:6667; itype:0; content:"ficken"; reference:url,staff.washington.edu/dittrich/misc/stacheldraht.analysis; classtype:attempted-dos; sid:1856; rev:7;)