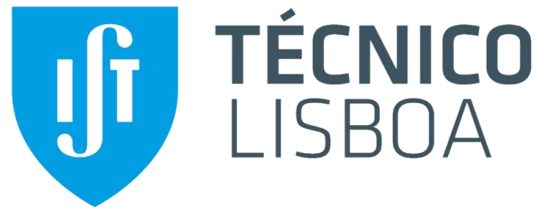


Instituto Superior Técnico



SNORT

Master's in computer science and Engineering
Network Advanced Security and Architecture



98678, Bruno Freitas

Table of Contents

Introduction.....	4
Snort.....	5
Network topology	6
DoS based attacks	7
ICMP flood.....	7
Snort Rule	7
Attack demonstration.....	7
SYN flood	8
Snort Rule	8
Attack demonstration.....	8
Ping of Death.....	9
Snort Rule	9
Attack demonstration.....	9
Land Attack.....	10
Snort Rule	10
Attack demonstration.....	10
HTTP flooding.....	11
Snort Rule	11
Attack demonstration.....	11
TCP reset attack	12
Snort Rule	12
Attack demonstration.....	13
Christmas tree attack.....	14
Snort Rule	14
Attack demonstration.....	14
UDP flood	15
Snort Rule	15
Attack demonstration.....	15
DNS flood	16
Snort Rule	16
Attack demonstration.....	16
Smurf attack	17
Snort Rule	18
Attack demonstration.....	18
References	20

Annex	21
Snort	21

Table of Figures

Figure 1 Snort configuration file overview	5
Figure 2 Snort configuration file DAQ configuration	5
Figure 3 Snort configuration file import rules	6
Figure 4 Network topology used for testing	6
Figure 5 ICMP flood attack command execution	7
Figure 6 ICMP flood Wireshark capture	7
Figure 7 ICMP flood snort alert	8
Figure 8 SYN flood attack command execution.....	8
Figure 9 SYN flood Wireshark capture	9
Figure 10 SYN flood Snort alert	9
Figure 11 Land attack command execution.....	10
Figure 12 Land attack Wireshark capture.....	11
Figure 13 Land attack Snort alert.....	11
Figure 14 HTTP flooding attack command execution	12
Figure 15 HTTP flood Wireshark capture	12
Figure 16 HTTP flood Snort alert	12
Figure 17 TCP reset attack command execution.....	13
Figure 18 TCP reset attack Wireshark capture.....	13
Figure 19 TCP reset attack Snort rule.....	13
Figure 20 XMAS tree attack command execution	14
Figure 21 XMAS tree attack Wireshark capture	14
Figure 22 XMAS tree attack Snort alert.....	15
Figure 23 UDP flood attack command execution	15
Figure 24 UDP flood Wireshark capture.....	16
Figure 25 UDP flood Snort alert	16
Figure 26 DNS flood attack command execution	16
Figure 27 DNS flood Wireshark capture.....	17
Figure 28 DNS flood Snort alert	17
Figure 29 Smurf attack network topology	17
Figure 30 Smurf attack VPC1 Wireshark capture	18
Figure 31 Smurf attack Snort machine Wireshark capture	19
Figure 32 Smurf attack Snort alert	19

Introduction

This report will study an important concept regarding network security. This concept is a NIDS, called Snort. (Note: Snort is an IPS but can be configured as a IDS which is the case for this study).

An Intrusion Detection System (IDS) analyse network traffic for signatures matching known attacks. On the other hand, an Intrusion Prevention System (IPS) also analyse the network traffic, but it can stop the packets from being delivered ("prevents"). Why is it important? It is important because it does monitoring of the network traffic, alerting such attacks as Denial of Service and imposing a small impact on performance.

There are two types of IDS, a NIDS and a HIDS. A HIDS is a Host-based intrusion detection system and as the name says, it is used to analyse events on a computing device rather than the traffic that goes through the network. On the other hand, NIDS is a network-based intrusion detection system and is used to analyse the network traffic.

In this study we will configure a NIDS, called Snort, to alert several types of Denial-of-Service attacks. For each attack, will be briefly explained what it is and will be demonstrated the attack running and the alert given by Snort. It is also given a simple Scapy script to replicate the attack as also as the Snort rule to alert the attack.

All attacks were targeted at the specific snort machine, but all the rules implemented were made general enough for snort to behave like a real NIDS. A network topology where snort would be an NIDS of an organization was not used for reasons of time and because the objective of this work was to understand how to make rules, and how to protect from some general attacks.

Snort

So, to understand a little more let's analyse some basic files about Snort. Firstly, there is a configuration file called *snort.conf* which states the main steps to make our own custom configuration:

```
#####
# This file contains a sample snort configuration.
# You should take the following steps to create your own custom configuration:
#
# 1) Set the network variables.
# 2) Configure the decoder
# 3) Configure the base detection engine
# 4) Configure dynamic loaded libraries
# 5) Configure preprocessors
# 6) Configure output plugins
# 7) Customize your rule set
# 8) Customize preprocessor and decoder rule set
# 9) Customize shared object rule set
#####
```

Figure 1 Snort configuration file overview

By the Figure 1 is possible to see the structure of this configuration file. In the first step, which is “set the network variables”, we setup the HOME_NET variable to the subnet of our organization, in this case is 192.168.1.0/24. Then, in the second step there is also an important configuration if we need to configure snort as inline mode that can be used to run snort as an IPS.

```
# Configure ports to ignore
# config ignore_ports: tcp 21 6667:6671 1356
# config ignore_ports: udp 1:17 53

# Configure active response for non inline operation. For more information, see README.active
# config response: eth0 attempts 2

# Configure DAQ related options for inline operation. For more information, see README.daq
#
# config daq: <type>
# config daq_dir: <dir>
# config daq_mode: <mode>
# config daq_var: <var>
```

Figure 2 Snort configuration file DAQ configuration

In the Figure 2, we see what would be needed to configure the inline mode. For example:

```
config daq: afpacket
config daq_mode: inline
```

where *afpacket* tell us it is in inline on Linux using two bridged interfaces. To check the available *daq* types just type “*snort --daq-list*”. Let's see some *daq* types very generally:

- Pcap: is the default mode, used for sniffer and IDS modes
- Afpacket: inline on Linux using two bridged interfaces
- Ipq: old way to process iptables packets
- Nfq: new improved way to process iptables packet
- Ipfw: inline mode in BSD systems
- Dump: allows you to test the various inline mode features

After this, another important step is related with the rules of snort. In this configuration file we can include or exclude (comment out) which rules we want to and do not want to use, as we can see in Figure 3.

```
#####
# Step #7: Customize your rule set
# For more information, see Snort Manual, Writing Snort Rules
#
# NOTE: All categories are enabled in this conf file
#####

# site specific rules
include $RULE_PATH/local.rules

include $RULE_PATH/app-detect.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
```

Figure 3 Snort configuration file import rules

But what is this Snort rule? Is a methodology for performing detection and has the following structure:

➤ Rule Header

```
<action>    <protocol>    <IP_source>    <source_port>    <action>
<IP_destination> <destination_port>
```

➤ Rule Options

```
<Option_keyword> <arguments>
```

For example, let's see this structure in the following rule:

```
alert tcp any !21 -> 10.10.10.10 400: (msg: "TCP connection");
```

The action is an **alert** so it will generate an alert (it can also be **log** which after generating the alert, it then logs the packet, or **pass** which ignores the packet and drops it). Related to the protocol it can be **TCP**, **UDP** or **ICMP** and in this case is TCP. The source IP address is any (it can also be a subnet like 10.10.10.0/24) and source port is any except 21. After this we have “->” which indicates the direction of the connection (it can also be “<=>” which indicates it is in both directions). The destination IP address is 10.10.10.10 with any port greater than or equal to 400. So, in translating it into English it means that will alert all TCP traffic that comes from any IP address with any source port except 21 to the destination 10.10.10.10 to any port greater than or equal to 400. Then, if this happens it will print the message specified in the Rule Options which is “TCP connection”. There are several rule options available that will be used and discussed further in this report.

Network topology

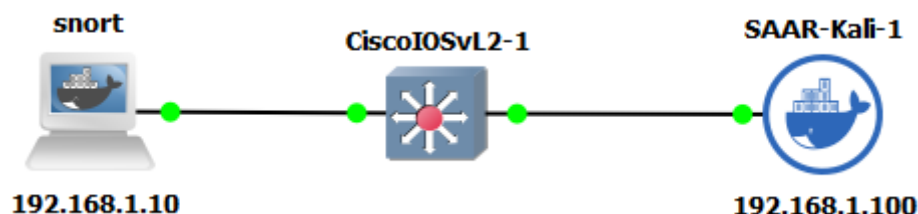


Figure 4 Network topology used for testing

The Figure 4 represents the topology used for testing the snort rules and see the attack taking into play. The topology used was simple because the purpose of this report is to understand the attacks and which rules can be implemented for detecting them.

- Command to test if the snort configurations (rules inclusive) are OK:
snort -T -i eth0 -c /etc/snort/snort.conf
- Command to run snort (in alert mode, in quit mode and with the specified snort file configuration):
snort -A console -q -c /etc/snort/snort.conf -i eth0

DoS based attacks

ICMP flood

ICMP flood or Ping flood is a Denial of Service (DoS) attack where the attacker overwhelms the victim with ICMP echo requests.

Snort Rule

Adding the following rule in local.rules will alert if an ICMP flood is happening:

```
alert icmp any any -> $HOME_NET any (msg:"ICMP flood"; sid:1000001; rev:1;
classtype:icmp-event; detection_filter:track by_dst,count 100, seconds 3;)
```

So, analysing the rule, it will keep a track by destination of icmp packets with any source IP and any source port to \$HOME_NET to any destination port and if it reaches 100 in 3 seconds will raise an alert with the message ICMP flood and type ICMP Event.

Attack demonstration

To test this, an attacker will do an ICMP flood with the command "hping3 -C --faster --flood -V -p 80 192.168.1.10" as can be seen in the Figure 5.

```
root@SAAR-Kali-1:/usr/attacks# hping3 -C --faster --flood -V -p 80 192.168.1.10
using eth0, addr: 192.168.1.100, MTU: 1500
HPING 192.168.1.10 (eth0 192.168.1.10): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.1.10 hping statistic ---
903231 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@SAAR-Kali-1:/usr/attacks#
```

Figure 5 ICMP flood attack command execution

By the Figure 6 is possible to see what packets were sent, ICMP packets:

No.	Time	Source	Destination	Protocol	Length	Info
2749	25.785717	192.168.1.100	192.168.1.10	ICMP	60	Echo (ping) reply id=0x4d00, seq=8633/47393, ttl=64
2750	25.806660	192.168.1.100	192.168.1.10	ICMP	60	Echo (ping) reply id=0x4d00, seq=8889/47394, ttl=64
2751	25.806967	192.168.1.100	192.168.1.10	ICMP	60	Echo (ping) reply id=0x4d00, seq=9145/47395, ttl=64
2752	25.808980	192.168.1.100	192.168.1.10	ICMP	60	Echo (ping) reply id=0x4d00, seq=9401/47396, ttl=64

```
> Frame 2750: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
> Ethernet II, Src: 82:3d:de:74:71:7f (82:3d:de:74:71:7f), Dst: 02:68:7d:4b:94:ec (02:68:7d:4b:94:ec)
> Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.10
> Internet Control Message Protocol
```

Figure 6 ICMP flood Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 7.

```

5/31-19:26:42.320348  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10
5/31-19:26:42.320631  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10
5/31-19:26:42.322646  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10
5/31-19:26:42.322947  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10
5/31-19:26:42.324470  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10
5/31-19:26:42.324764  [**] [1:1000001:1] ICMP flood [**] [Classification: Generic ICMP event] [Priority: 3] {ICMP} 192.168.
.100 -> 192.168.1.10

```

Figure 7 ICMP flood snort alert

➤ Possible Scapy Script:

```

from scapy.all import *

packet=IP(src=RandIP(), dst="192.168.1.10")/ICMP()
send(packet, loop=1)

```

SYN flood

SYN flood is a Denial of Service (DoS) attack where the attacker overwhelms the victim by making him consume resources by exploiting part of the normal TCP three-way handshake. This handshake consists in

1. Client requests the connection by sending a TCP packet with the SYN flag on
2. Server responds with a TCP packet with SYN and ACK flags on
3. Client responds with a TCP packet with ACK flag on

After these 3 steps the connections is stablished and the client-server can communicate. Now, an attacker exploits this by executing the step 1, the victim does the second step, but the third step is not done by the attacker which leads to the victim waiting for the ACK packet wasting resources.

Snort Rule

Adding the following rule in local.rules will alert if an SYN flood is happening:

```

alert tcp any any -> $HOME_NET 80 (flags: S; msg:"Possible SYNflood"; flow:stateless;
sid: 1000002; rev:1; detection_filter:track by_dst, count 20, seconds 10;)

```

So, analysing the rule, it will keep a track by destination of tcp packets with any source IP and any source port to \$HOME_NET to port 80 and if it reaches 20 in 10 seconds will raise an alert with the message Possible SYNflood. Note that the flow is stateless which means that the alert will be triggered regardless of the state of the stream processor.

Attack demonstration

To test this, an attacker will do an SYN flood with the command " hping3 -S --faster --flood -V -p 80 192.168.1.10" as can be seen in the Figure 8.

```

root@SAAR-Kali-1:/usr/attacks/dos_attacks# hping3 -S --faster --flood -V -p 80 192.168.1.10
using eth0, addr: 192.168.1.100, MTU: 1500
HPING 192.168.1.10 (eth0 192.168.1.10): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.1.10 hping statistic ---
62107 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@SAAR-Kali-1:/usr/attacks/dos_attacks#

```

Figure 8 SYN flood attack command execution

By the Figure 9 is possible to see what packets were sent which are the SYN packets and their response SYN-ACK but the connection is not established:

No.	Time	Source	Destination	Protocol	Length	Info
7	10.341180	192.168.1.100	192.168.1.10	TCP	60	1063 → 80 [SYN] Seq=0 Win=512 Len=0
8	10.341213	192.168.1.10	192.168.1.100	TCP	54	80 → 1063 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	10.342522	192.168.1.100	192.168.1.10	TCP	60	1064 → 80 [SYN] Seq=0 Win=512 Len=0
10	10.342534	192.168.1.10	192.168.1.100	TCP	54	80 → 1064 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	10.370491	192.168.1.100	192.168.1.10	TCP	60	1065 → 80 [SYN] Seq=0 Win=512 Len=0
12	10.370512	192.168.1.10	192.168.1.100	TCP	54	80 → 1065 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	10.373130	192.168.1.100	192.168.1.10	TCP	60	1066 → 80 [SYN] Seq=0 Win=512 Len=0

> Frame 8: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface -, id 0
 > Ethernet II, Src: 02:68:7d:4b:94:ec (02:68:7d:4b:94:ec), Dst: 82:3d:de:74:71:7f (82:3d:de:74:71:7f)
 > Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.100
 > Transmission Control Protocol, Src Port: 80, Dst Port: 1063, Seq: 1, Ack: 1, Len: 0

Figure 9 SYN flood Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 10.

05/31-20:05:18.360668	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22448 -> 192.168.1.10:80
05/31-20:05:18.360984	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22449 -> 192.168.1.10:80
05/31-20:05:18.363023	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22450 -> 192.168.1.10:80
05/31-20:05:18.363391	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22451 -> 192.168.1.10:80
05/31-20:05:18.365443	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22452 -> 192.168.1.10:80
05/31-20:05:18.365857	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22453 -> 192.168.1.10:80
05/31-20:05:18.367839	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22454 -> 192.168.1.10:80
05/31-20:05:18.368203	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22455 -> 192.168.1.10:80
05/31-20:05:18.370231	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22456 -> 192.168.1.10:80
05/31-20:05:18.370617	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22457 -> 192.168.1.10:80
05/31-20:05:18.372582	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22458 -> 192.168.1.10:80
05/31-20:05:18.372950	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22459 -> 192.168.1.10:80
05/31-20:05:18.374999	[**]	[1:1000002:1]	Possible SYNflood	[**]	[Priority: 0]	{TCP} 192.168.1.100:22460 -> 192.168.1.10:80

Figure 10 SYN flood Snort alert

➤ Possible Scapy Script:

```
from scapy.all import *
```

```
packet=IP(src=RandIP(), dst="192.168.1.10")/TCP(dport=80, flags="S")
send(packet, loop=1)
```

Ping of Death

Ping of death is a Denial of Service (DoS) attack where the attacker overwhelms the victim by sending an IPv4 packet larger than 65,535 bytes in fragments because the maximum accepted size for an IP packet is 65,535 bytes. Old devices attempting to reassemble the fragments could end up with an oversized packet and crashes could occur (because a memory overflow could occur, for example).

Snort Rule

Adding the following rule in local.rules will alert if a Ping of Death is happening:

```
alert icmp any any -> $HOME_NET any (msg: "Ping of Death"; sid:1000003; rev:1; dsize:
>60000;)
```

So, analysing the rule, it will see the icmp packets with any source IP and any source port to \$HOME_NET to any destination port and if the packet size is greater than 65535 will raise an alert with the message Ping of Death.

Attack demonstration

To test this, an attacker will do a Ping of Death with the Scapy script:

```
from scapy.all import *
```

```
packet=IP(src=RandIP(), dst="192.168.1.10")/ICMP()/("X"*60000)
send(fragment(packet), loop=1)
```

No demonstrations are given because the student could not put the attack to work (maybe because it is a simulated environment, but further investigation would be needed?!)

Land Attack

Land attack is a Denial of Service (DoS) attack where the attacker sets the source and destination of a TCP packet to be the same and possibly crashing the victim due to the packet being repeatedly processed by the TCP stack.

Snort Rule

Adding the following rule in local.rules will alert if a Land Attack is happening:

```
alert tcp any any -> $HOME_NET any (sameip; msg:"LAND attack"; sid:10000004; rev:1;)
```

So, analysing the rule, it will see the tcp packets with any source IP and any source port to \$HOME_NET to any destination port and if the source and destination IP are the same will raise an alert with the message LAND attack.

Attack demonstration

To test this, an attacker will do a Land attack with the Scapy script:

```
from scapy.all import *
```

```
packet=IP(src="192.168.1.10", dst="192.168.1.10")/TCP(dport=RandShort())
send(packet, loop=1)
```

```
root@SAAR-Kali-1:/usr/attacks/dos_attacks# python3 land attack.py
WARNING: No route found for IPv6 destination :: (no default route?). This affects only IPv6

.....^C
Sent 2214 packets.
root@SAAR-Kali-1:/usr/attacks/dos_attacks#
```

Figure 11 Land attack command execution

By the Figure 12 is possible to see what packets were sent, with the same source and destination addresses:

No.	Time	Source	Destination	Protocol	Length	Info
1967	120.522668	192.168.1.10	192.168.1.10	TCP	60	20 → 20106 [SYN] Seq=0 Win=8192 Len=0
1968	120.529732	192.168.1.10	192.168.1.10	TCP	60	20 → 31419 [SYN] Seq=0 Win=8192 Len=0
1969	120.530138	192.168.1.10	192.168.1.10	TCP	60	20 → 62387 [SYN] Seq=0 Win=8192 Len=0
1970	120.536254	192.168.1.10	192.168.1.10	TCP	60	20 → 33487 [SYN] Seq=0 Win=8192 Len=0
1971	120.536608	192.168.1.10	192.168.1.10	TCP	60	20 → 63858 [SYN] Seq=0 Win=8192 Len=0

> Frame 1967: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
 > Ethernet II, Src: de:2c:c7:bd:eb:99 (de:2c:c7:bd:eb:99), Dst: ce:41:bd:d8:15:e3 (ce:41:bd:d8:15:e3)
 > Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.10
 > Transmission Control Protocol, Src Port: 20, Dst Port: 20106, Seq: 0, Len: 0

Figure 12 Land attack Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 13.

```
06/01-11:53:14.895942 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:55849
06/01-11:53:14.897318 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:39735
06/01-11:53:14.897645 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:22335
06/01-11:53:14.899102 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:26505
06/01-11:53:14.899511 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:12842
06/01-11:53:14.900984 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:60103
06/01-11:53:14.901273 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:38343
06/01-11:53:14.902684 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:50693
06/01-11:53:14.902975 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:30082
06/01-11:53:14.904461 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:9733
06/01-11:53:14.904748 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:43283
06/01-11:53:14.906132 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:7090
06/01-11:53:14.906441 00000000:1 LAND attack [Priority: 0] {TCP} 192.168.1.10:20 -> 192.168.1.10:28446
```

Figure 13 Land attack Snort alert

HTTP flooding

HTTP flooding is a Denial of Service (DoS) attack where the attacker overwhelms the victim with HTTP requests. To understand the potential of this attack let's assume we have a HTTP server. The attacker will send a numerous GET requests trying to obtain an object (for example a big image) wasting server resources.

Snort Rule

Adding the following rule in local.rules will alert if a HTTP flood is happening:

```
alert tcp any any -> $HOME_NET 80 (msg: "Possible http flood attack"; flow:established;
content:"GET"; nocase; http_method; detection_filter:track by_dst, count 1000, seconds
10; sid: 10000005; rev:1;)
```

So, analysing the rule, it will keep a track by destination of tcp packets with any source IP and any source port to \$HOME_NET port 80 and that are with an established connection. If the HTTP method is GET (no case which is not case sensitive) and if it reaches 1000 in 10 seconds will raise an alert with the message Possible http flood attack. //FALAR do estabelecimento no mundo real e na simulação!!!

NOTE: in this report only a simulation has been done so the attacker will not open a TCP connection to send the HTTP requests and that's why the above rule does not work. To see the alert running with the script given in this report the following rule must be used:

```
alert tcp any any -> $HOME_NET 80 (msg: "Possible http flood attack"; content:"GET";
nocase; http_method; detection_filter:track by_dst, count 1000, seconds 10; sid:
10000005; rev:1;)
```

The only difference is that the parameter "flow:established" is not used because the script will send only the HTTP packets and will not open a TCP connection.

Attack demonstration

To test this, an attacker will do a HTTP flooding with the Scapy script:

```
from scapy.all import *

packet=IP(src=RandIP(), dst="192.168.1.10")/TCP(dport=80)/ "GET /HTTP/1.0\r\n\r\n"
send(packet, loop=1)
```

```
root@SAAR-Kali-1:/usr/attacks/crashes_attacks# python3 http_flooding.py
WARNING: No route found for IPv6 destination :: (no default route?). This affects only IPv6
.....
.....
.....
.....
.....
.....
.....
```

Figure 14 HTTP flooding attack command execution

By the Figure 15 is possible to see what packets were sent, simple HTTP packets:

No.	Time	Source	Destination	Protocol	Length	Info
17315	2296.228693	58.216.109.193	192.168.1.10	HTTP	71	GET /HTTP/1.0
17316	2296.230167	30.151.41.60	192.168.1.10	HTTP	71	GET /HTTP/1.0
17317	2296.230465	126.241.60.193	192.168.1.10	HTTP	71	GET /HTTP/1.0
17318	2296.231879	152.70.13.227	192.168.1.10	HTTP	71	GET /HTTP/1.0

> Frame 17315: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface -, id 0
 > Ethernet II, Src: 1e:ff:11:b4:b5:0c (1e:ff:11:b4:b5:0c), Dst: aa:d0:48:fd:58:e3 (aa:d0:48:fd:58:e3)
 > Internet Protocol Version 4, Src: 58.216.109.193, Dst: 192.168.1.10
 > Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 0, Len: 17
 ✓ Hypertext Transfer Protocol
 > GET /HTTP/1.0\r\n
 \r\n
 [HTTP request 1/1]

Figure 15 HTTP flood Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 16.

```
06/01-14:58:45.005751 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 255.255.00.255:20 -> 192.168.1.10:80
06/01-14:58:45.004026 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 58.216.109.193:20 -> 192.168.1.10:80
06/01-14:58:45.005501 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 30.151.41.60:20 -> 192.168.1.10:80
06/01-14:58:45.005797 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 126.241.60.193:20 -> 192.168.1.10:80
06/01-14:58:45.007212 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 152.70.13.227:20 -> 192.168.1.10:80
06/01-14:58:45.007508 1:10000005:1 Possible http flood attack [**] [Priority: 0] {TCP} 83.89.113.128:20 -> 192.168.1.10:80
```

Figure 16 HTTP flood Snort alert

TCP reset attack

TCP reset attack is a Denial of Service (DoS) attack where the attacker overwhelms the victim by sending a TCP packet with RST flag on. To understand it better let's see assume a simple scenario where the victim has a TCP connection to an organization's private server. So, an attacker starts sending RST packets to the victim to close the connections it has (if a connection exists closes it and a RST packet is sent when it is received a packet in a closed port)

Snort Rule

Adding the following rule in local.rules will alert if a TCP reset attack is happening:

```
alert tcp any any -> $HOME_NET any (flags:R; msg: "Possible TCP reset attack";
flow:stateless; sid: 10000006; rev:1;)
```

So, analysing the rule, it will see tcp packets with any source IP and any source port to \$HOME_NET to any destination port and if the flags is R (RST) will raise an alert with the message Possible TCP reset attack.

Attack demonstration

To test this, an attacker will do a TCP reset attack with the Scapy script (to simplify only the port 80 was used but we could send to all ports to close all connections):

```
from scapy.all import *
```

```
packet=IP(src=RandIP(), dst="192.168.1.10")/TCP(dport=80, flags="R")
send(packet, loop=1)
```

```
root@SAAR-Kali-1:/usr/attacks/dos_attacks# python tcp_reset.py
bash: python: command not found
root@SAAR-Kali-1:/usr/attacks/dos_attacks# python3 tcp_reset.py
WARNING: No route found for IPv6 destination :: (no default route?). This affects only IPv6
.....^C
Sent 2045 packets.
root@SAAR-Kali-1:/usr/attacks/dos_attacks#
```

Figure 17 TCP reset attack command execution

By the Figure 18 is possible to see what packets were sent, a TCP packet with the flag RST on, from random sources:

No.	Time	Source	Destination	Protocol	Length	Info
3667	264.871891	44.222.166.14	192.168.1.10	TCP	60	20 → 80 [RST] Seq=1 Win=8192 Len=0
3668	264.875772	24.85.100.112	192.168.1.10	TCP	60	20 → 80 [RST] Seq=1 Win=8192 Len=0
3669	264.876077	15.233.31.184	192.168.1.10	TCP	60	20 → 80 [RST] Seq=1 Win=8192 Len=0
3670	264.880302	120.149.98.49	192.168.1.10	TCP	60	20 → 80 [RST] Seq=1 Win=8192 Len=0
3671	264.880603	11.23.44.246	192.168.1.10	TCP	60	20 → 80 [RST] Seq=1 Win=8192 Len=0


```
> Frame 3667: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
> Ethernet II, Src: de:2c:c7:bd:eb:99 (de:2c:c7:bd:eb:99), Dst: ce:41:bd:d8:15:e3 (ce:41:bd:d8:15:e3)
> Internet Protocol Version 4, Src: 44.222.166.14, Dst: 192.168.1.10
✓ Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 1, Len: 0
  Source Port: 20
  Destination Port: 80
  [Stream index: 3510]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  Sequence number (raw): 0
  [Next sequence number: 1 (relative sequence number)]
  Acknowledgment number: 0
  Acknowledgment number (raw): 0
  0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x004 (RST)
  Window size value: 8192
  [Calculated window size: 8192]
```

Figure 18 TCP reset attack Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 19.

```
06/01-13:11:29.790741 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 183.196.137.210:20 -> 192.168.1.10:80
06/01-13:11:29.791031 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 51.203.203.48:20 -> 192.168.1.10:80
06/01-13:11:29.792464 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 173.139.135.225:20 -> 192.168.1.10:80
06/01-13:11:29.792755 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 205.140.75.217:20 -> 192.168.1.10:80
06/01-13:11:29.812951 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 48.45.115.180:20 -> 192.168.1.10:80
06/01-13:11:29.813222 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 255.168.29.36:20 -> 192.168.1.10:80
06/01-13:11:29.814634 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 123.137.178.165:20 -> 192.168.1.10:80
06/01-13:11:29.814946 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 52.219.237.174:20 -> 192.168.1.10:80
06/01-13:11:29.816396 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 197.185.140.143:20 -> 192.168.1.10:80
06/01-13:11:29.816686 [**] [1:10000006:1] Possible TCP reset attack [**] [Priority: 0] {TCP} 206.37.24.152:20 -> 192.168.1.10:80
```

Figure 19 TCP reset attack Snort rule

Christmas tree attack

Christmas tree attack is a Denial of Service (DoS) attack where the attacker overwhelms the victim with special crafted TCP packets, where the URG, PSH and FIN flags are on. The problem is that some devices do not know what to do with such packets, for example, some devices may reboot, others respond and allow recognition (for example, to see which operation a host is running). The DoS comes when an attacker overwhelms the victim with these packets because they require much more processing than a “normal” one.

Snort Rule

Adding the following rule in local.rules will alert if a Christmas tree attack is happening:

```
alert tcp any any -> $HOME_NET any (flags:FPU; msg: "Possible X-MAS tree attack";
flow:stateless; sid: 10000007; rev:1;)
```

So, analysing the rule, it will see tcp packets with any source IP and any source port to \$HOME_NET to any destination port and if the flags F, P and U (FIN, PSH and URG respectively) will raise an alert with the message Possible X-MAS tree attack.

Attack demonstration

To test this, an attacker will do a Christmas tree attack with the Scapy script:

```
from scapy.all import *

packet=IP(src=RandIP(), dst="192.168.1.10")/TCP(dport=80, flags="FPU")
send(packet, loop=1)
```

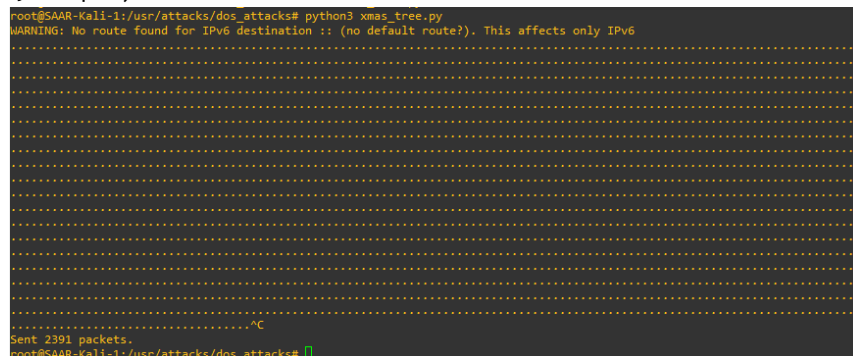


Figure 20 XMAS tree attack command execution

By the Figure 21 is possible to see the packets sent, with FIN, PSH and URG flags on:

No.	Time	Source	Destination	Protocol	Length	Info
6458	1536.509820	23.0.233.253	192.168.1.10	TCP	60	20 → 80 [FIN, PSH, URG] Seq=1 Win=8192 Urg=0 Len=0
6459	1536.514346	145.59.54.102	192.168.1.10	TCP	60	20 → 80 [FIN, PSH, URG] Seq=1 Win=8192 Urg=0 Len=0
6460	1536.514648	14.11.255.189	192.168.1.10	TCP	60	20 → 80 [FIN, PSH, URG] Seq=1 Win=8192 Urg=0 Len=0
6461	1536.518602	172.149.107.85	192.168.1.10	TCP	60	20 → 80 [FIN, PSH, URG] Seq=1 Win=8192 Urg=0 Len=0
6462	1536.518805	92.169.96.100	192.168.1.10	TCP	60	20 → 80 [FIN, PSH, URG] Seq=1 Win=8192 Urg=0 Len=0

```

> Frame 6458: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
> Ethernet II, Src: de:2c:c7:bd:eb:99 (de:2c:c7:bd:eb:99), Dst: ce:41:bd:d8:15:e3 (ce:41:bd:d8:15:e3)
> Internet Protocol Version 4, Src: 23.0.233.253, Dst: 192.168.1.10
▼ Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 1, Len: 0
  Source Port: 20
  Destination Port: 80
  [Stream index: 5556]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  Sequence number (raw): 0
  [Next sequence number: 2 (relative sequence number)]
  Acknowledgment number: 0
  Acknowledgment number (raw): 0
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x029 (FIN, PSH, URG)
  Window size value: 8192

```

Figure 21 XMAS tree attack Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 22.

```
06/01-13:32:42.148532 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 185.118.64.96:20 -> 192.168.1.10:80
06/01-13:32:42.148802 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 199.18.120.159:20 -> 192.168.1.10:80
06/01-13:32:42.150253 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 217.36.58.21:20 -> 192.168.1.10:80
06/01-13:32:42.150546 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 100.154.175.0:20 -> 192.168.1.10:80
06/01-13:32:42.152029 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 140.224.69.216:20 -> 192.168.1.10:80
06/01-13:32:42.152323 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 168.68.226.101:20 -> 192.168.1.10:80
06/01-13:32:42.153831 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 173.25.31.61:20 -> 192.168.1.10:80
06/01-13:32:42.154140 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 146.13.197.92:20 -> 192.168.1.10:80
06/01-13:32:42.155580 [**] [1:1000007:1] Possible X-MAS tree attack [**] [Priority: 0] {TCP} 223.129.185.130:20 -> 192.168.1.10:80
```

Figure 22 XMAS tree attack Snort alert

UDP flood

UDP flood is a Denial of Service (DoS) attack where the attacker overwhelms the victim with UDP datagrams.

Snort Rule

Adding the following rule in local.rules will alert if a UDP flood is happening:

```
alert udp any any -> $HOME_NET !53 (msg: "UDP flood detected"; flow:stateless;
detection filter:track by dst, count 1000, seconds 10; sid:10000008; rev:1;)
```

So, analysing the rule, it will keep track by destination of udp packets with any source IP and any source port to \$HOME_NET to any destination port different than 53 and if it reaches 1000 in 10 seconds will raise an alert with the message UDP flood detected.

Attack demonstration

To test this, an attacker will do a UDP flood attack to random ports with the Scapy script:

```
from scapy.all import *

packet=IP(src=RandIP(), dst="192.168.1.10")/UDP(dport=RandShort())
send(packet, loop=1)
```

[illegible]

Figure 23 UDP flood attack command execution

By the Figure 24 is possible to see what packets were sent, UDP datagrams:

No.	Time	Source	Destination	Protocol	Length	Info
12606	2458.123702	162.228.172.61	192.168.1.10	UDP	60	53 → 26756 Len=0
12607	2458.125172	192.7.48.172	192.168.1.10	UDP	60	53 → 8135 Len=0
12608	2458.125462	162.202.59.218	192.168.1.10	UDP	60	53 → 50245 Len=0
12609	2458.126864	44.195.90.26	192.168.1.10	UDP	60	53 → 22599 Len=0
12610	2458.127158	9.227.233.80	192.168.1.10	UDP	60	53 → 53947 Len=0

> Frame 12606: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
 > Ethernet II, Src: de:2c:c7:bd:eb:99 (de:2c:c7:bd:eb:99), Dst: ce:41:bd:d8:15:e3 (ce:41:bd:d8:15:e3)
 > Internet Protocol Version 4, Src: 162.228.172.61, Dst: 192.168.1.10
 > User Datagram Protocol, Src Port: 53, Dst Port: 26756

Figure 24 UDP flood Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 25.

```
06/01-13:47:59.353538    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 91.229.238.204:53 -> 192.168.1.10:421  
58  
06/01-13:47:59.353830    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 162.228.172.61:53 -> 192.168.1.10:267  
56  
06/01-13:47:59.355302    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 192.7.48.172:53 -> 192.168.1.10:8135  
06/01-13:47:59.355591    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 162.202.59.218:53 -> 192.168.1.10:502  
45  
06/01-13:47:59.356993    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 44.195.90.26:53 -> 192.168.1.10:22599  
06/01-13:47:59.357287    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 9.227.233.80:53 -> 192.168.1.10:52947  
06/01-13:47:59.358634    *** [1:10000008:1] UDP flood detected *** [Priority: 0] {UDP} 108.66.169.136:53 -> 192.168.1.10:496  
13
```

Figure 25 UDP flood Snort alert

DNS flood

DNS flood is a Denial of Service (DoS) attack where the attacker overwhelms the victim, DNS server, to resolve some resource records wasting their resources to emulated requests (and because it is hard to differentiate the real requests from these ones, this attack is hard to prevent).

Snort Rule

Adding the following rule in local.rules will alert if a DNS flood is happening:

```
alert udp any any -> $HOME_NET 53 (msg: "DNS flood detected"; detection_filter:track
by_dst, count 1000, seconds 10; sid:10000009; rev:1;)
```

So, analysing the rule, it will keep track by destination of udp packets with any source IP and any source port to \$HOME_NET to port 53 which is DNS, and if it reaches 1000 in 10 seconds will raise an alert with the message DNS flood detected.

Attack demonstration

To test this, an attacker will do a DNS flood attack with the Scapy script:

```
from scrapy.all import *
```

```
packet=IP(src=RandIP(), dst="192.168.1.10")/UDP()/DNS(rd=1,
qd=DNSQR(qname="www.attack.com"))
send(packet, loop=1)
```

[illegible]

Figure 26 DNS flood attack command execution

By the Figure 27 is possible to see what packets were sent:

No.	Time	Source	Destination	Protocol	Length	Info
16987	2936.068277	182.108.83.7	192.168.1.10	DNS	74	Standard query 0x0000 A www.attack.com
16988	2936.068549	130.58.9.34	192.168.1.10	DNS	74	Standard query 0x0000 A www.attack.com
16989	2936.069960	245.86.234.247	192.168.1.10	DNS	74	Standard query 0x0000 A www.attack.com
16990	2936.070251	192.243.125.190	192.168.1.10	DNS	74	Standard query 0x0000 A www.attack.com
16991	2936.071710	183.117.229.248	192.168.1.10	DNS	74	Standard query 0x0000 A www.attack.com

```

> Frame 13094: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface -, id 0
> Ethernet II, Src: de:2c:c7:bd:eb:99 (de:2c:c7:bd:eb:99), Dst: ce:41:bd:d8:15:e3 (ce:41:bd:d8:15:e3)
> Internet Protocol Version 4, Src: 61.17.195.146, Dst: 192.168.1.10
> User Datagram Protocol, Src Port: 53, Dst Port: 53
> Domain Name System (query)

```

Figure 27 DNS flood Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 28.

06/01-13:55:57.298411	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	182.108.83.7:53 -> 192.168.1.10:53
06/01-13:55:57.298678	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	130.58.9.34:53 -> 192.168.1.10:53
06/01-13:55:57.300090	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	245.86.234.247:53 -> 192.168.1.10:53
06/01-13:55:57.300380	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	192.243.125.190:53 -> 192.168.1.10:53
06/01-13:55:57.301849	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	183.117.229.248:53 -> 192.168.1.10:53
06/01-13:55:57.302151	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	228.143.174.193:53 -> 192.168.1.10:53
06/01-13:55:57.303640	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	42.62.190.166:53 -> 192.168.1.10:53
06/01-13:55:57.303936	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	234.218.201.200:53 -> 192.168.1.10:53
06/01-13:55:57.305348	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	25.55.22:53 -> 192.168.1.10:53
06/01-13:55:57.305643	[**]	[1:10000009:1]	DNS flood detected	[**]	[Priority: 0]	{UDP}	199.12.67.35:53 -> 192.168.1.10:53

Figure 28 DNS flood Snort alert

Smurf attack

Smurf attack is a Denial of Service (DoS) attack where the attacker overwhelms the victim by sending ICMP packets to the broadcast IP address with the source as the victim IP address. So, with this, all hosts in the network will reply to that request answering back to the victim (because it was his spoofed IP address).

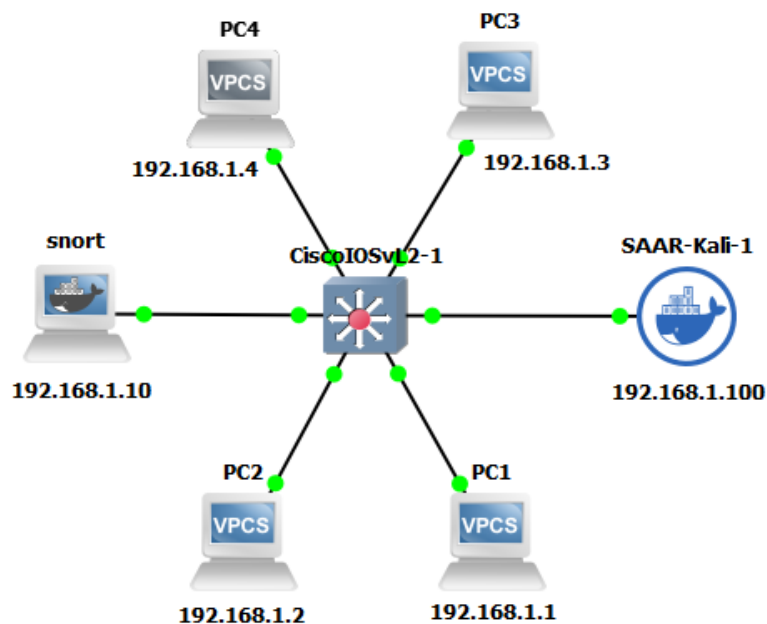


Figure 29 Smurf attack network topology

For this demonstration, a new network topology was made to easily see the attack, Figure 29. This topology is the one in the Figure 29. For the new VPCs the command `ip 192.168.1.X 255.255.255.0` where “X” was the VPC number, was used to give an IP address to each VPC. For “snort” and “SAAR-kali-1” machines configuration, see the annex chapter.

Snort Rule

Adding the following rule in local.rules will alert if a smurf attack is happening:

```
alert icmp any any -> $HOME_NET any (msg: "SMURF attack detected"; itype:0;
detection_filter:track by_dst, count 1000, seconds 10; sid:10000010; rev:1;)
```

So, analysing the rule, it will keep track by destination of icmp packets type 0 which represent echo replies with any source IP and any source port to \$HOME_NET to any destination port and if it reaches 1000 in 10 seconds will raise an alert with the message SMURF attack detected.

Attack demonstration

To test this, an attacker will do a smurf attack with the Scapy script:

```
from scapy.all import *

packet1=IP(src="192.168.1.10", dst="192.168.1.1")/ICMP()
packet2=IP(src="192.168.1.10", dst="192.168.1.2")/ICMP()
packet3=IP(src="192.168.1.10", dst="192.168.1.3")/ICMP()
packet4=IP(src="192.168.1.10", dst="192.168.1.4")/ICMP()
packets = [packet1, packet2, packet3, packet4]

while True:
    for packet in packets:
        send(packet)
```

Note: the script is done to send to each VPC because when used the broadcast address which in this case is 192.168.1.255 the VPCs were receiving but not responding, maybe it is a limitation of the GNS3 (but this requires more investigating to say it). The script to execute a smurf attack would be:

```
from scapy.all import *

packet=IP(src="192.168.1.10", dst="192.168.1.255")/ICMP()
send(packet, loop=1)
```

By the Figure 30 which is a Wireshark capture, is possible to see that was sent ICMP packets from 192.168.1.10 (which is the victim) for the VPCs, in this case for the VPC1 who will reply to that ICMP packet:

No.	Time	Source	Destination	Protocol	Length	Info
1663	38.471807	192.168.1.10	192.168.1.1	ICMP	60	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (repl...
1664	38.471839	192.168.1.1	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (requ...
1665	38.481332	192.168.1.10	192.168.1.1	ICMP	60	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (repl...
1666	38.481365	192.168.1.1	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (requ...
1667	38.495023	192.168.1.10	192.168.1.1	ICMP	60	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (repl...
1668	38.495040	192.168.1.1	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (requ...

> Frame 20: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
 > Ethernet II, Src: 36:a8:14:c0:70:89 (36:a8:14:c0:70:89), Dst: Private_66:68:00 (00:50:79:66:68:00)
 > Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.1
 > Internet Control Message Protocol

Figure 30 Smurf attack VPC1 Wireshark capture

The Wireshark capture in Figure 31, shows the reply packets arrived at the victim PC:

No.	Time	Source	Destination	Protocol	Length	Info
863	42.473536	192.168.1.4	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
864	42.481792	192.168.1.2	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
865	42.497571	192.168.1.3	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
866	42.512645	192.168.1.1	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
867	42.516067	192.168.1.4	192.168.1.10	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=63

> Frame 27: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface -, id 0
 > Ethernet II, Src: 36:a8:14:c0:70:89 (36:a8:14:c0:70:89), Dst: aa:d0:48:fd:58:e3 (aa:d0:48:fd:58:e3)
 > Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.10
 > Internet Control Message Protocol

Figure 31 Smurf attack Snort machine Wireshark capture

Snort with the rule specified above will raise an enormous number of alerts in the console as can be seen in the Figure 32.

```

06/01-16:24:20.486875 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.4 -> 192.168.1.10
06/01-16:24:20.506354 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.1 -> 192.168.1.10
06/01-16:24:20.534892 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.2 -> 192.168.1.10
06/01-16:24:20.539388 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.3 -> 192.168.1.10
06/01-16:24:20.548932 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.1 -> 192.168.1.10
06/01-16:24:20.560531 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.4 -> 192.168.1.10
06/01-16:24:20.568783 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.2 -> 192.168.1.10
06/01-16:24:20.584565 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.3 -> 192.168.1.10
06/01-16:24:20.599638 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.1 -> 192.168.1.10
06/01-16:24:20.603948 [**] [1:10000010:1] SMURF attack detected [**] [Priority: 0] {ICMP} 192.168.1.4 -> 192.168.1.10
  
```

Figure 32 Smurf attack Snort alert

References

Network Intrusion Detection Systems (SNORT)

[Network Intrusion Detection Systems \(SNORT\) - YouTube](#)

Snort

<https://pt.wikipedia.org/wiki/Snort>

hping3 flood ddos

<https://linuxhint.com/hping3/>

Writing Effective Snort Rules with Examples [Best Practices]

<https://coralogix.com/blog/writing-effective-snort-rules-for-the-sta/>

Basic snort rules syntax and usage [updated 2021]

<https://resources.infosecinstitute.com/topic/snort-rules-workshop-part-one/>

Snort Intrusion Prevention System (IPS) Configuration and Rule Creation

https://www.youtube.com/watch?v=enSll_9Bjag

Detecting DDos attack using Snort

https://www.researchgate.net/publication/338660054_DETECTING_DDOS_ATTACK_USING_Snort

Ping of Death

<https://www.cloudflare.com/learning/ddos/ping-of-death-ddos-attack/>

Mitigation of DoS and Port Scan Attacks Using Snort

https://www.researchgate.net/publication/335803817_Mitigation_of_DoS_and_Port_Scan_Attacks_Using_Snort

Annex

Snort

- PC1:
`ip 192.168.1.1 255.255.255.0 192.168.1.254`
- PC2:
`ip 192.168.1.2 255.255.255.0 192.168.1.254`
- SAAR-Kali-1:
Edit config:
`auto eth0`
`iface eth0 inet static`
 `address 192.168.1.100`
 `netmask 255.255.255.0`
- Snort:
Edit config:
`auto eth0`
`iface eth0 inet static`
 `address 192.168.1.10`
 `netmask 255.255.255.0`

Edit file /etc/snort/snort.conf and change the line:
Setup the network addresses you are protecting
`ipvar HOME_NET 192.168.1.0/24`

==== EXTRA (not used) ====
 - Setup a port mirroring in CiscoIOSvL2-1
`en`
`conf t`
`monitor session 1 source interface g0/1`
`monitor session 1 destination interface g0/0`
`end`