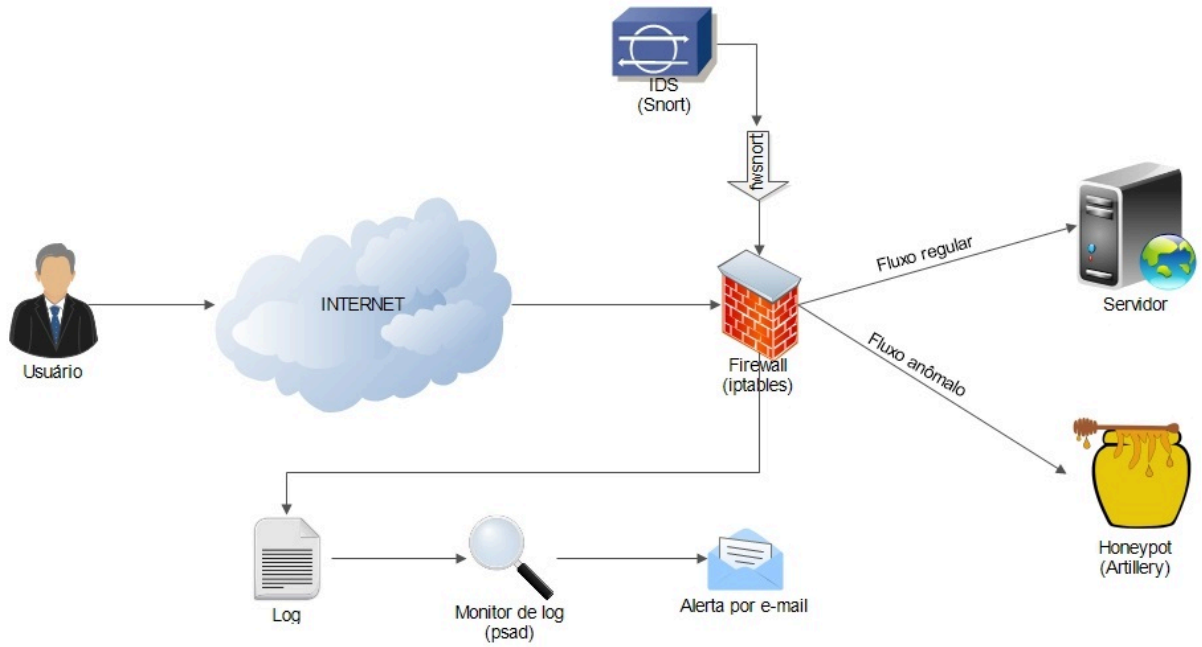
TP 9 : Surveillance et Filtrage du Trafic Réseau avec Snort et Iptables



Réalisé par:
Rguibi Mohamed Mouad

Encadré par:
Pr.Ikram benabdelouahab

Introduction

L'objectif de ce laboratoire était de mettre en place une architecture réseau virtuelle sécurisée pour simuler, détecter, enregistrer et bloquer un trafic malveillant. Nous avons utilisé Mininet pour la topologie, Snort comme système de détection d'intrusions (IDS), Tcpdump pour la capture de paquets, et Iptables pour le filtrage pare-feu.

Partie 1 : Préparation de l'environnement virtuel

Configuration Réseau

La machine virtuelle *Security Workstation* a été configurée pour accéder au réseau externe (Internet). Nous avons utilisé le script configure_as_dhcp.sh pour obtenir une adresse IP via DHCP.

Vérification de la connectivité : La commande ifconfig a confirmé l'attribution d'une adresse IP et la commande ping a validé la connectivité vers Internet.

```
analyst@labvm ~  (4.097s)
sudo ./lab.support.files/scripts/configure_as_dhcp.sh
[sudo] password for analyst:
Configuring the NIC to request IP info via DHCP...
Requesting IP information...
IP Configuration successful.
```

Step 3. Verify Connectivity

- Verify your IP address:

```
analyst@labvm ~  (0.047s)
ifconfig

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe55:4407  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:55:44:07  txqueuelen 1000  (Ethernet)
        RX packets 43  bytes 18945 (18.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 62  bytes 8455 (8.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.105  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe11:cdf4  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:11:cd:f4  txqueuelen 1000  (Ethernet)
        RX packets 2360  bytes 209879 (209.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4591  bytes 644625 (644.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 12  bytes 1084 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1084 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Test Internet connectivity:

```
analyst@labvm ~  (4.61s)
ping www.cisco.com

PING e2867.dsca.akamaiedge.net (2.22.84.98) 56(84) bytes of data.
64 bytes from a2-22-84-98.deploy.static.akamaitechnologies.com (2.22.84.98): icmp_seq=1 ttl=63 time=46.5 ms
64 bytes from a2-22-84-98.deploy.static.akamaitechnologies.com (2.22.84.98): icmp_seq=2 ttl=63 time=44.4 ms
64 bytes from a2-22-84-98.deploy.static.akamaitechnologies.com (2.22.84.98): icmp_seq=3 ttl=63 time=44.6 ms
64 bytes from a2-22-84-98.deploy.static.akamaitechnologies.com (2.22.84.98): icmp_seq=4 ttl=63 time=452 ms
64 bytes from a2-22-84-98.deploy.static.akamaitechnologies.com (2.22.84.98): icmp_seq=5 ttl=63 time=92.7 ms
^C
--- e2867.dsca.akamaiedge.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 44.376/136.018/451.884/159.004 ms
```

Partie 2 : Journaux Pare-feu et IDS

Étape 1 : Surveillance des journaux IDS en temps réel

Nous avons déployé une topologie réseau avec Mininet comprenant un routeur (R1) et plusieurs hôtes (H5, H10).

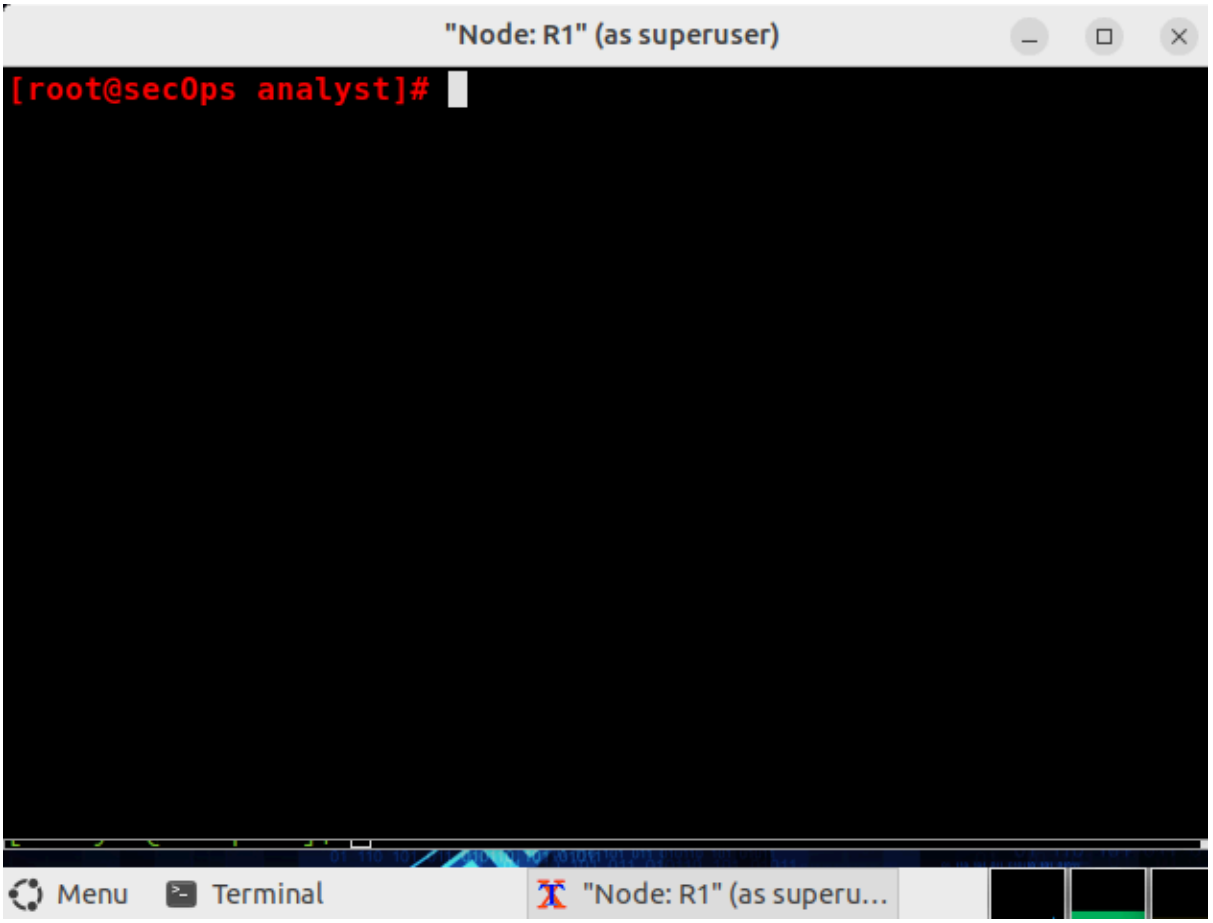
- R1 : Configure avec l'IDS Snort.
- H10 : Simule un serveur malveillant hébergeant un malware.
- H5 : Client victime téléchargeant le fichier.

Démarrage du serveur malveillant (H10) : Le serveur a été lancé sur H10. La vérification via `netstat` montre que le service écoute sur le port TCP 6666.

```
analyst@labvm ~
sudo ./lab.support.files/scripts/cyberops_extended_topo_no_fw.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
R1 R4 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11
*** Starting controllers
*** Starting switches
*** Add routes
*** Post configure switches and hosts
*** Starting CLI:
mininet>
```

2. Access Router R1

```
*** Starting CLI:
mininet> xterm R1
mininet>
```



3. Start Snort (IDS) on R1

```
[root@secOps analyst]# ./lab.support.files/scripts/start_snort.sh
```

```
        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.15.1 GRE (Build 15125)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.1  <Build 1>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: appid  Version 1.1  <Build 5>
Commencing packet processing (pid=3268)
```

4. Access Hosts H5 and H10:

```
mininet> xterm H5
mininet> xterm H10
mininet>
```

5. Start the Malicious Server on H10

```
[root@secOps analyst]# ./lab.support.files/scripts/mal_server_start.sh
```

6. Verify the Server is Running

```
[root@secOps analyst]# netstat -tunpa
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:6666            0.0.0.0:*               LISTEN
3409/nginx: master
[root@secOps analyst]#
```

Détection de l'intrusion (Snort sur R1) : Depuis H5, nous avons initié le téléchargement du fichier `W32.Nimda.Amm.exe`. Snort, écoutant sur R1, a immédiatement détecté la signature du trafic malveillant.

- **Commande utilisée :** `wget 209.165.202.133:6666/W32.Nimda.Amm.exe`
- **Port utilisé :** 6666
- **Résultat :** Téléchargement réussi (100%).

2. Monitor Alerts on R1 (Step h)

```
[root@secOps analyst]# tail -f /var/log/snort/alert
```

3. Generate Traffic from H5 (Step i)

```
[root@secOps analyst]# wget 209.165.202.133:6666/W32.Nimda.Amm.exe
--2025-12-17 18:14:35--  http://209.165.202.133:6666/W32.Nimda.Amm.exe
Connecting to 209.165.202.133:6666... connected.
HTTP request sent, awaiting response... 200 OK
Length: 475448 (464K) [application/octet-stream]
Saving to: 'W32.Nimda.Amm.exe'

W32.Nimda.Amm.exe   100%[===================>] 464.30K  --.-KB/s    in 0.006s

2025-12-17 18:14:35 (72.5 MB/s) - 'W32.Nimda.Amm.exe' saved [475448/475448]
```

Verification & Questions

Port Used: 6666 (Visible in your wget command and the Snort log).

Download Status: Successful (The output shows 100% and saved).

IDS Alert: Yes, Snort generated the alert: [**] [1:1000003:0] Malicious Server Hit!

```
"Node: R1" (as superuser)
[root@secOps analyst]# tail -f /var/log/snort/alert
12/17-18:14:35.097791  [**] [1:1000003:0] Malicious Server Hit! [**] [Priority:
0] {TCP} 209.165.200.235:55384 -> 209.165.202.133:6666
```

Étape 2 : Capture de trafic avec Tcpdump

Afin de conserver une preuve numérique de l'attaque pour une analyse forensique ultérieure, nous avons lancé une capture de paquets sur l'interface de H5.

- Fichier de sortie : `nimda.download.pcap`
- Utilité du fichier PCAP : Ce fichier permet aux analystes de sécurité de "rejouer" le trafic, d'extraire la charge utile (le malware) pour l'analyser, et de servir de preuve légale de l'incident.

Vérification de la capture : La commande `ls -l` confirme que le fichier `.pcap` a bien été créé et contient des données (taille > 0).

1. Start the Packet Capture (Step k)

```
[root@secOps analyst]# tcpdump -i H5-eth0 -w nimda.download.pcap &
[1] 3676
[root@secOps analyst]# tcpdump: listening on H5-eth0, link-type EN10MB (Ethernet), snapshot length 262144 b
ytes
```

2. Download the Malware Again (Step l)

```
[root@secOps analyst]# wget 209.165.202.133:6666/W32.Nimda.Amm.exe
--2025-12-17 18:21:52--  http://209.165.202.133:6666/W32.Nimda.Amm.exe
Connecting to 209.165.202.133:6666... connected.
HTTP request sent, awaiting response... 200 OK
Length: 475448 (464K) [application/octet-stream]
Saving to: 'W32.Nimda.Amm.exe.1'

W32.Nimda.Amm.exe.1              100%[=============================================================>] 464.30K  --.-KB/s    in 0.005s

2025-12-17 18:21:52 (92.2 MB/s) - 'W32.Nimda.Amm.exe.1' saved [475448/475448]

[root@secOps analyst]#
```

3. Stop the Capture (Step m)

```
[root@secOps analyst]# fg
tcpdump -i H5-eth0 -w nimda.download.pcap

^C74 packets captured
74 packets received by filter
0 packets dropped by kernel
[root@secOps analyst]#
```

4. Verify the Capture File (Step n)

```
[root@secOps analyst]# ls -l
total 1428
drwxr-xr-x 2 analyst analyst   4096 Feb 10  2023 Desktop
drwxr-xr-x 2 analyst analyst   4096 Dec 17 17:58 Documents
drwxr-xr-x 2 analyst analyst   4096 Dec 17 17:58 Downloads
drwxr-xr-x 9 analyst analyst   4096 Jan 14  2023 lab.support.files
-rw-r--r-- 1 tcpdump tcpdump 481903 Dec 17 18:22 nimda.download.pcap
drwxr-xr-x 2 analyst analyst   4096 Jan 12  2023 second_drive
-rw-r--r-- 1 root    root    475448 Jan 31  2023 W32.Nimda.Amm.exe
-rw-r--r-- 1 root    root    475448 Jan 31  2023 W32.Nimda.Amm.exe.1
[root@secOps analyst]#
```

How can this PCAP file be useful to the security analyst? A PCAP file is a digital recording of network traffic. It is useful because:

- **Forensics: An analyst can open it in tools like Wireshark to "replay" the attack and see exactly what happened.**
- **Evidence: It serves as proof of the malicious activity.**
- **Deep Analysis: It allows the analyst to extract the actual malware file (payload) from the packets for reverse engineering or to see what data was stolen.**

Partie 3 : Optimisation des règles de pare-feu (Iptables)

Après avoir détecté l'attaque, nous avons configuré le pare-feu Linux (iptables) sur le routeur R1 pour bloquer ce trafic spécifique.

Analyse des règles existantes

Avant modification, nous avons listé les chaînes actives sur R1. Les trois chaînes utilisées sont :

1. INPUT (Trafic entrant vers le routeur)
2. FORWARD (Trafic traversant le routeur)
3. OUTPUT (Trafic sortant du routeur)

1. Open a New Terminal for R1 (Step a)

2. Check Current Firewall Rules (Step b)

```
[root@secOps analyst]# iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

The three chains currently in use by R1: INPUT, FORWARD, and OUTPUT.

1.Ajout de la règle de blocage

Puisque le trafic transite de H5 vers H10 via R1, nous devons agir sur la chaîne FORWARD. Nous avons ajouté une règle pour rejeter (DROP) les paquets TCP à destination de l'IP `209.165.202.133` sur le port `6666`.

```
[root@secOps analyst]# iptables -I FORWARD -p tcp -d 209.165.202.133 --dport 6666 -j DROP
[root@secOps analyst]#
```

2. Verify the Rule was Added (Step d)

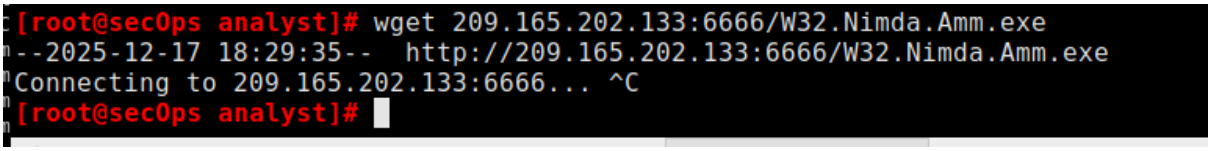
```
[root@secOps analyst]# iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       tcp  --  any    any     anywhere             209.165.202.133      tcp dpt:6666

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
[root@secOps analyst]#
```

3.Validation du blocage

Nous avons tenté de télécharger à nouveau le fichier malveillant depuis H5.

- **Résultat : La connexion a échoué (`Connection timed out`).**

- **Analyse : Le pare-feu a intercepté les paquets correspondant à la règle et les a supprimés silencieusement avant qu'ils n'atteignent le serveur.**



Verification:

- **Rule Added: The iptables output shows the new rule in the FORWARD chain (targeting tcp dpt:6666 with DROP).**
- **Block Successful: The wget command failed with "Connecting to 209.165.202.133:6666... ^C" (it hung and timed out).**

Was the download successful this time? Explain. No,

the download was not successful. The iptables rule on R1 dropped the packets destined for port 6666 before they could reach the server, causing the connection to time out.

What would be a more aggressive but also valid approach when blocking the offending server?

A more aggressive approach would be to block all traffic to and from that IP address (209.165.202.133), regardless of the port. This ensures that if the attackers switch ports (e.g., from 6666 to 80), you are still protected.

Part 3: Terminate and Clear Mininet Process

1. Quit Mininet (Step a)

```
mininet> quit
*** Stopping 0 controllers

*** Stopping 5 terms
*** Stopping 15 links
...............
*** Stopping 3 switches
S5 S9 S10
*** Stopping 13 hosts
R1 R4 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11
*** Done
```

2. Clean Up Processes (Step b)

```
analyst@labvm ~  (5.087s)
sudo mn -c

[sudo] password for analyst:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

CHECKLIST : Réponses et Validations

Q1. Snort est-il en cours d'exécution ? En déduire son mode de fonctionnement (IDS ou IPS).

- **Réponse :** Oui, Snort est en cours d'exécution.
- **Mode :** Il fonctionne en mode **IDS** (Intrusion Detection System).
- **Déduction :** Le script de démarrage a affiché "Running in IDS mode". De plus, Snort a seulement alerté (généré des logs) mais n'a pas bloqué le trafic lui-même (nous avons dû utiliser Iptables pour bloquer).

Q2. Quelle interface réseau Snort surveille-t-il et pourquoi est-ce logique dans cette topologie ?

- **Réponse :** Il surveille l'interface de **R1** (probablement `r1-eth0` ou similaire selon la config Mininet).
- **Logique :** C'est logique car R1 est le **routeur passerelle**. Tout le trafic entre le réseau interne (H5) et le réseau externe (H10) doit obligatoirement transiter par R1. C'est le point de passage stratégique pour tout voir.

Q3. Combien d'alertes ont été générées après le téléchargement du malware ?

- **Réponse :** D'après mon resultat , **1 alerte** a été affichée lors de l'attaque.

Q4. Quel champ de la sortie permet de confirmer que la règle DROP est réellement utilisée ? Expliquez.

- **Réponse :** Les colonnes **pkts** (packets) et **bytes**.
- **Explication :** Dans la sortie de `iptables -L -v`, si le compteur sous "pkts" n'est pas à zéro (par exemple, s'il affiche 4 ou 5 paquets), cela signifie que le pare-feu a "attrapé" des paquets correspondant à la règle et a appliqué l'action DROP.

Q5. Test de connectivité ICMP après blocage : Le ping est-il bloqué ? Pourquoi ?

- **Réponse :** Non, le ping **n'est pas bloqué**.
- **Pourquoi :** La règle Iptables que nous avons créée spécifiait `-p tcp --dport 6666`. Le Ping utilise le protocole **ICMP**, qui est différent de TCP. Le pare-feu laisse donc passer l'ICMP car aucune règle ne l'interdit.

Q6. Analyse du fichier PCAP généré : Quel protocole de couche transport est observé dans la capture ?

- **Réponse : TCP**.
- **Preuve :** `wget` utilise HTTP, qui repose sur TCP. L'alerte Snort indiquait également `{TCP}`.

Q7. Identification de la direction du trafic dans l'alerte Snort

- Basé sur : 209.165.200.235:34484 -> 209.165.202.133:6666

- **Client (Source) :** `209.165.200.235` (C'est H5). On le reconnaît car il utilise un port aléatoire élevé (34484).
- **Serveur (Destination) :** `209.165.202.133` (C'est H10). On le reconnaît car il écoute sur le port de service fixe (6666).

Q8. Quelle est la politique par défaut appliquée aux chains INPUT, OUTPUT et FORWARD ?

- **Réponse : ACCEPT**.
- **Preuve :** La commande `iptables -L` montrait `Chain INPUT (policy ACCEPT)`, etc. Cela signifie que tout ce qui n'est pas explicitement interdit est autorisé.

Q9. Effet d'un arrêt de Snort sur la sécurité

- **Le téléchargement est-il à nouveau possible ? Non**.
- **Pourquoi :** Snort est un outil de détection (caméra de surveillance). Iptables est l'outil de blocage (porte blindée). Même si on éteint la caméra (Snort), la porte reste fermée (la règle Iptables DROP est toujours active dans le noyau Linux).

Q10. Blocage complet du serveur malveillant (tous les ports)

- **Résultat attendu :** Si on retire `--dport 6666` de la commande, **toutes** les connexions TCP vers ce serveur échoueront, même si le serveur change de port (par exemple s'il passe sur le port 80 ou 443).
- **Conclusion :** C'est une mesure plus drastique et sécurisée contre une IP hostile connue, mais cela risque de bloquer des services légitimes si cette IP héberge d'autres choses.

Q12. Journalisation du trafic malveillant avant blocage

- **Configuration :** Il faut ajouter une règle LOG **avant** la règle DROP.
```
iptables -I FORWARD 1 -p tcp -d 209.165.202.133 --dport 6666 -j LOG --log-prefix "Tentative Malware: "
```

- **Information dans les logs :** On verra l'IP source, l'IP destination, les ports, et l'heure de la tentative.
- **Utilité :** Cela permet à l'analyste de savoir que le pare-feu travaille et de garder une trace des tentatives d'attaque (audit), au lieu que les paquets disparaissent silencieusement.

QCM (Choix Multiples)

Q13. Quel est le rôle principal de Mininet dans ce TP ?

- **Créer une topologie réseau virtuelle pour les tests**

Q14. À quoi peut servir le fichier nimda.download.pcap pour un analyste sécurité ?

- **Analyser le trafic réseau et confirmer l’attaque**

Q15. Pourquoi Snort a-t-il généré une alerte lors du téléchargement du fichier ?

- **Le payload correspond à une signature malveillante** (Ce n'est pas le port ou l'IP en soi qui a déclenché l'alerte, mais le contenu du fichier reconnu comme étant le ver Nimda).