Université Abdelmalek Essaâdi

Faculté des Sciences et Techniques de Tanger

RAPPORT DE MINI-PROJET
Sécurisation Réseau Zero Trust

Pare-feu, VPN, HTTPS & IDS Snort

Réalisé par :
ELJABLY Salaheddine

Encadré par :
Pr. Ikram BENABDELOUAHAB

Année Universitaire 2025-2026

Table des matières

1 Introduction

Ce projet a pour objectif de concevoir, déployer et valider une infrastructure réseau sécurisée basée sur le modèle **Zero Trust**. Contrairement aux réseaux traditionnels, cette architecture n'accorde aucune confiance par défaut, même aux utilisateurs internes.

Nous avons utilisé l'émulateur **Mininet** pour simuler un environnement réaliste comprenant un pare-feu à état, un serveur web sécurisé (HTTPS), un accès distant chiffré (VPN) et un système de détection d'intrusion (Snort).

2 Architecture Réseau

L'architecture repose sur une topologie en étoile centrée autour d'un routeur Linux faisant office de Pare-feu et de passerelle VPN. Le réseau est segmenté en trois zones distinctes pour limiter la surface d'attaque :

1. **WAN (Untrusted - 192.168.100.0/24) :** Simule l'Internet et l'attaquant externe (`h_ext`).
2. **DMZ (Public - 10.0.1.0/24) :** Zone hébergeant les services accessibles au public (`h_web1`).
3. **LAN (Trusted - 10.0.2.0/24) :** Zone critique contenant le poste d'administration (`h_admin`).

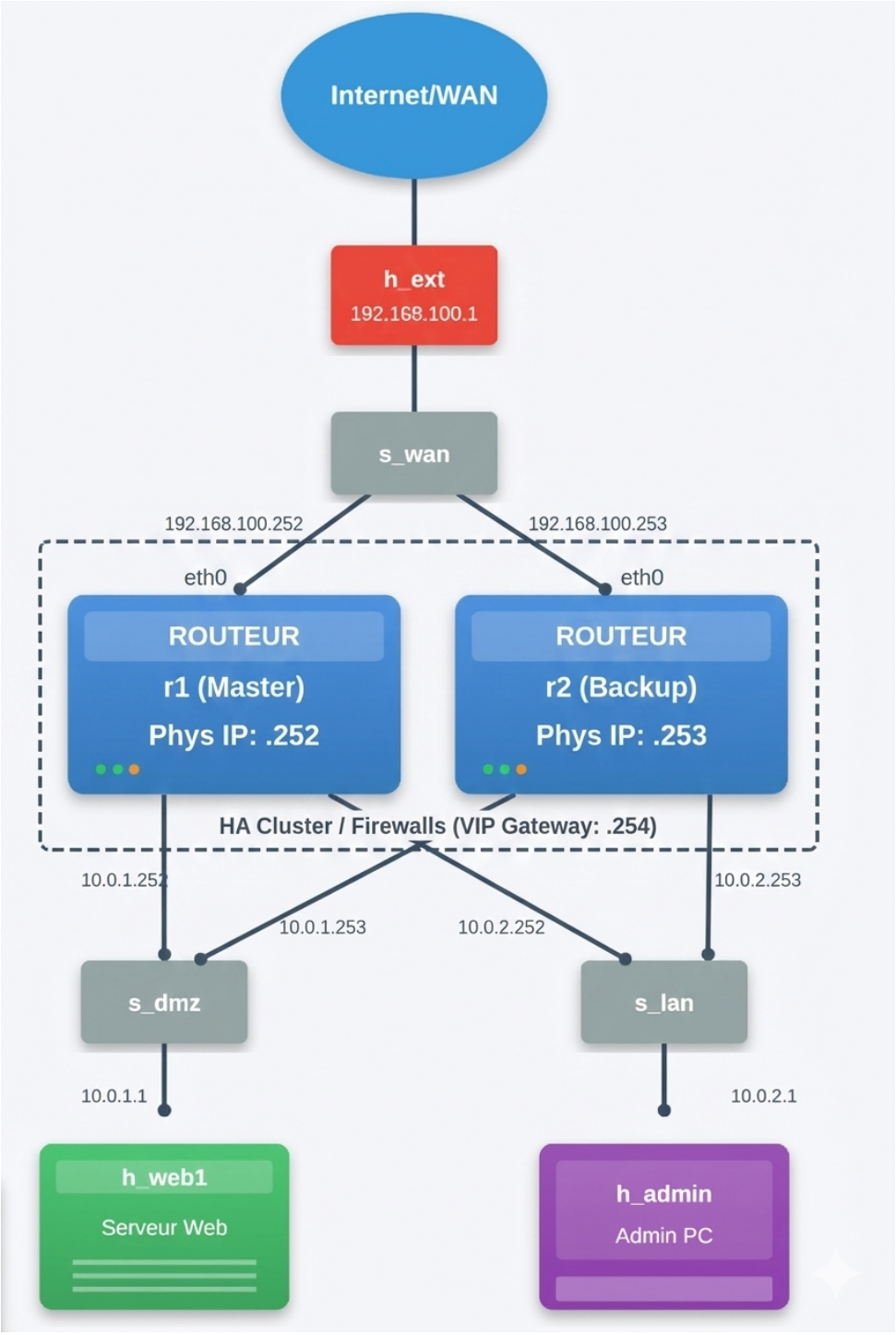

FIGURE 1 – Topologie logique de l'infrastructure sécurisée (Mininet).

3 Implémentation Technique

3.1 Déploiement de la Topologie (Python)

Le script `project_topo.py` utilise l'API Python de Mininet pour construire le réseau. Il définit les hôtes, les commutateurs et configure les adresses IP.

```
1 def secure_infra():
```

```
    net = Mininet( controller=Controller, switch=OVSKernelSwitch )

    # Creation du Routeur Central
    r1 = net.addHost( 'r1', ip='192.168.100.254/24' )

    # Definition des Zones
    s_wan = net.addSwitch( 's1' )
    s_dmz = net.addSwitch( 's2' )

    # Lancement des scripts de securite au demarrage
    r1.cmd("bash firewall.sh")
    r1.cmd("snort -A fast -q -c /etc/snort/snort.conf -i r1-eth0 &")
```

Listing 1 – Extrait de project_topo.py

3.2 Pare-feu Stateful (Bash/Iptables)

Le script `firewall.sh` implémente la politique Zero Trust. Il vide les tables existantes et applique une politique par défaut `DROP`.

```
# 1. Politique par defaut : TOUT BLOQUER
iptables -P INPUT DROP
iptables -P FORWARD DROP

# 2. Autoriser l'acces Web vers la DMZ
iptables -A FORWARD -i r1-eth0 -o r1-eth1 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i r1-eth0 -o r1-eth1 -p tcp --dport 443 -j ACCEPT

# 3. Journalisation (Logging)
iptables -A FORWARD -j LOG --log-prefix "FW_DROP: "
```

Listing 2 – Configuration du Pare-feu (firewall.sh)

3.3 Serveur Web Sécurisé (Python SSL)

Pour garantir la confidentialité des données (Task 3 & 4), nous avons développé `secure_server.py`. Il utilise la librairie `ssl` pour chiffrer les communications et force la redirection HTTP vers HTTPS.

```
class RedirectHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        # Force la redirection vers HTTPS (Code 301)
        new_path = "https://10.0.1.1" + self.path
        self.send_response(301)
        self.send_header('Location', new_path)
        self.end_headers()
```

Listing 3 – Gestion de la redirection 301 (secure_server.py)

3.4 Détection d'Intrusion (Snort)

Snort est configuré en mode NIDS sur l'interface WAN. Le fichier `local.rules` contient les signatures des attaques simulées.

```
# Detection de Scan ICMP
alert icmp any any -> any any (msg:"ALERT: T7.1 Network Scan Detected";
    sid:1000001;)
```

```
# Detection d'attaque Web (Acces fichier sensible)
alert tcp any any -> 10.0.1.1 80 (msg:"ALERT: T7.3 Web Attack Detected";
    content:"/etc/passwd";)
```

4 Validation et Preuves

4.1 T1.2 – Connectivité intra-zone (LAN)

Test : Vérification que l'administrateur peut communiquer avec la passerelle interne.
Commande : `h_admin ping -c 2 10.0.2.254`
Résultat : 0% Packet Loss. Le réseau interne est fonctionnel.

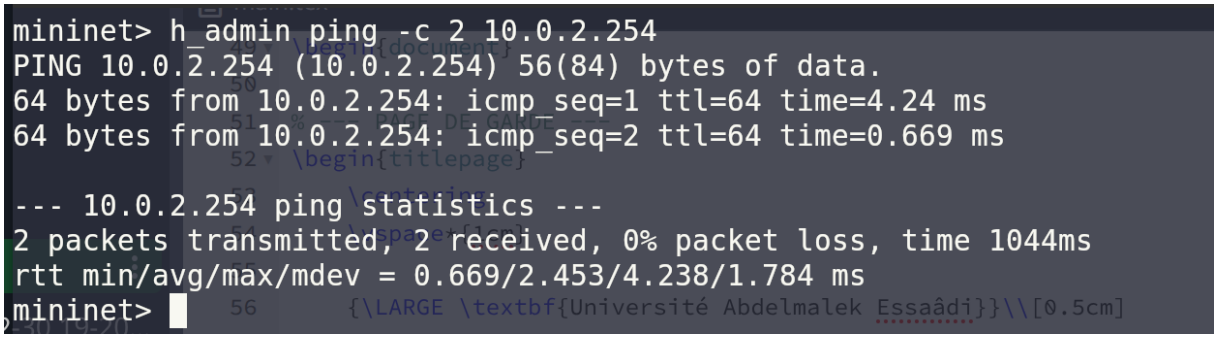

FIGURE 2 – Preuve T1.2 : Le Ping fonctionne correctement dans la zone LAN.

4.2 T1.3 – Isolation inter-zones (WAN vs LAN)

Test : Vérification que le WAN ne peut pas joindre le LAN directement (Segmentation).
Commande : `h_ext ping -c 2 10.0.2.1`
Résultat : 100% Packet Loss. Le pare-feu bloque l'accès direct.

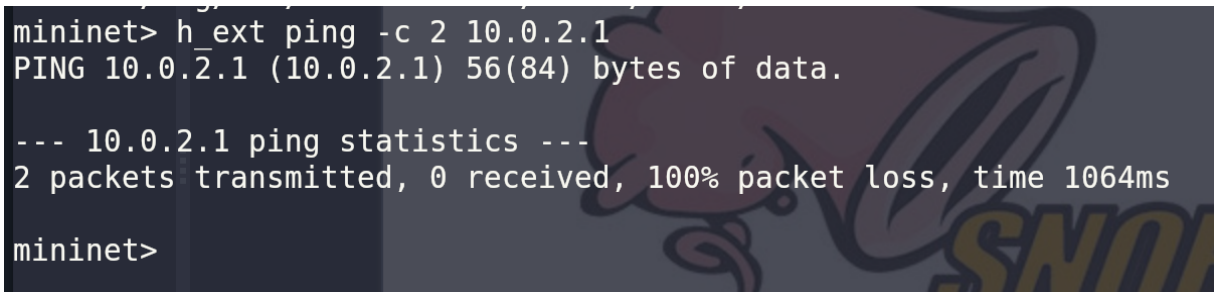

FIGURE 3 – Preuve T1.3 : Blocage total du trafic entre le WAN et le LAN (Isolation).

5 Pare-feu et Segmentation

5.1 T2.3 – Interdiction WAN → LAN (Nmap)

Test : Tentative de scan de ports depuis l'extérieur vers une machine protégée.
Commande : `h_ext nmap -p 22,80 10.0.2.1`
Résultat : "Note : Host seems down". La machine cible est invisible.

```
mininet> h_ext nmap -p 22,80 10.0.2.1
bash: nmap: command not found
mininet> h_ext nmap -p 22,80 10.0.2.1
Starting Nmap 7.80 ( https://nmap.org ) at 2025-12-30 20:56 +01
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.14 seconds
mininet>
```

FIGURE 4 – Preuve T2.3 : Le scan Nmap échoue, confirmant l'invisibilité du réseau interne.

5.2 T2.4 – Journalisation (Logs Firewall)

Test : Vérification que les paquets rejetés génèrent des logs système.
Commande : `sudo dmesg | grep "FW_DROP" | tail -n 5`
Résultat : Présence de lignes `FW_DROP` avec les adresses IP source/destination.

```
salah@ubuntu22:~$ sudo dmesg | grep "FW_DROP" | tail -n 5
[ 8194.133892] FW_DROP: IN=r1-eth0 OUT=r1-eth2 MAC=8e:39:61:32:3d:c3:f6:14:b2:61:ad:fb:08:00 SRC=192.168.100.1 DST=10.0.2.1 LEN=40 TOS=0x00 PREC=0x00 TTL=45 ID=
22545 PROTO=ICMP TYPE=13 CODE=0
[ 8196.144331] FW_DROP: IN=r1-eth0 OUT=r1-eth2 MAC=8e:39:61:32:3d:c3:f6:14:b2:61:ad:fb:08:00 SRC=192.168.100.1 DST=10.0.2.1 LEN=40 TOS=0x00 PREC=0x00 TTL=52 ID=
1652 PROTO=ICMP TYPE=13 CODE=0
[ 8196.144408] FW_DROP: IN=r1-eth0 OUT=r1-eth2 MAC=8e:39:61:32:3d:c3:f6:14:b2:61:ad:fb:08:00 SRC=192.168.100.1 DST=10.0.2.1 LEN=28 TOS=0x00 PREC=0x00 TTL=46 ID=
22615 PROTO=ICMP TYPE=8 CODE=0 ID=44004 SEQ=0
[ 8196.145515] FW_DROP: IN=r1-eth0 OUT=r1-eth2 MAC=8e:39:61:32:3d:c3:f6:14:b2:61:ad:fb:08:00 SRC=192.168.100.1 DST=10.0.2.1 LEN=40 TOS=0x00 PREC=0x00 TTL=37 ID=
42864 PROTO=TCP SPT=38007 DPT=80 WINDOW=1024 RES=0x00 ACK URGP=0
[ 8196.145591] FW_DROP: IN=r1-eth0 OUT=r1-eth2 MAC=8e:39:61:32:3d:c3:f6:14:b2:61:ad:fb:08:00 SRC=192.168.100.1 DST=10.0.2.1 LEN=44 TOS=0x00 PREC=0x00 TTL=38 ID=
50995 PROTO=TCP SPT=38007 DPT=443 WINDOW=1024 RES=0x00 SYN URGP=0
salah@ubuntu22:~$
```

FIGURE 5 – Preuve T2.4 : Traces du noyau montrant les paquets bloqués par le pare-feu.

6 DMZ et Services Web

6.1 T3.1 & T4.1 – Disponibilité HTTPS

Test : Accès au serveur Web sécurisé via le protocole chiffré.
Commande : `h_ext curl -k https://10.0.1.1`
Résultat : Affichage du code HTML `<h1>SUCCESS...</h1>`.

```
mininet> h_ext curl -k https://10.0.1.1
curl: (56) OpenSSL SSL_read: error:0A000126:SSL routines::unexpected eof while reading, errno 0
<h1>SUCCESS: T3.1 HTTPS Access Validated</h1>mininet>
```

FIGURE 6 – Preuve T3.1 & T4.1 : Connexion HTTPS réussie et contenu déchiffré.

6.2 T3.2 – Redirection HTTP vers HTTPS

Test : Vérification que l'accès non sécurisé force le passage en sécurisé.
Commande : `h_ext curl -I http://10.0.1.1`
Résultat : Code `301 Moved Permanently` et header `Location: https://...`.

FIGURE 7 – Preuve T3.2 : Redirection automatique vers HTTPS confirmée.

7 Accès Distant (OpenVPN)

7.1 T5.2 – Connexion VPN (Interface Tunnel)

Test : Vérification de l'établissement du tunnel virtuel.
Commande : `h_ext ifconfig tun0`
Résultat : Interface active avec adresse IP `10.8.0.2`.

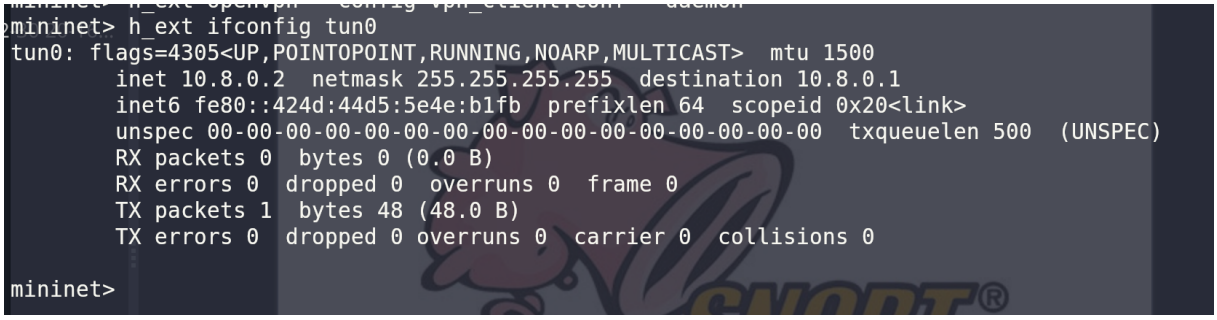

FIGURE 8 – Preuve T5.2 : Interface tun0 montée correctement sur le client.

7.2 T5.3 – Accès après VPN

Test : Vérification que l'accès au LAN est autorisé *à travers* le VPN.
Commande : `h_ext ping -c 2 10.0.2.254`
Résultat : 0% Packet Loss (Succès). Le trafic passe par le tunnel chiffré.

```
mininet> h_ext ping -c 2 10.0.2.254
PING 10.0.2.254 (10.0.2.254) 56(84) bytes of data.
64 bytes from 10.0.2.254: icmp_seq=1 ttl=64 time=3.06 ms
64 bytes from 10.0.2.254: icmp_seq=2 ttl=64 time=1.04 ms

--- 10.0.2.254 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 1.042/2.050/3.058/1.008 ms
mininet>
```

FIGURE 9 – Preuve T5.3 : Connectivité rétablie grâce au tunnel VPN.

8 Administration Sécurisée (SSH)

8.1 T6.2 – Authentification par Clé

Test : Connexion SSH sécurisée sans mot de passe interactif.
Commande : `h_admin ssh ... "echo SUCCESS_SSH"`
Résultat : Connexion immédiate et exécution de la commande.

```
mininet> h_admin ssh -p 2222 -i admin_key -o StrictHostKeyChecking=no root@192.168.100.254 "echo SUCCESS_SSH"
SUCCESS_SSH
mininet>
```

FIGURE 10 – Preuve T6.2 : Authentification forte par clé RSA validée.

9 Détection d'Intrusion (Snort)

9.1 T7.1 & T7.3 – Détection Scan et Web Attack

Test : Simulation d'attaques (Ping et Vol de fichier passwd) et lecture des alertes.
Commande : `tail -n 5 /var/log/snort/alert`
Résultat : Alertes `Network Scan Detected` et `Web Attack Detected`.

```
mininet> h_ext ping -c 1 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> h_ext curl -k https://10.0.1.1/etc/passwd
curl: (56) OpenSSL SSL_read: error:0A000126:SSL routines::unexpected eof while reading, errno 0
<h1>SUCCESS: T3.1 HTTPS Access Validated</h1>mininet>
```

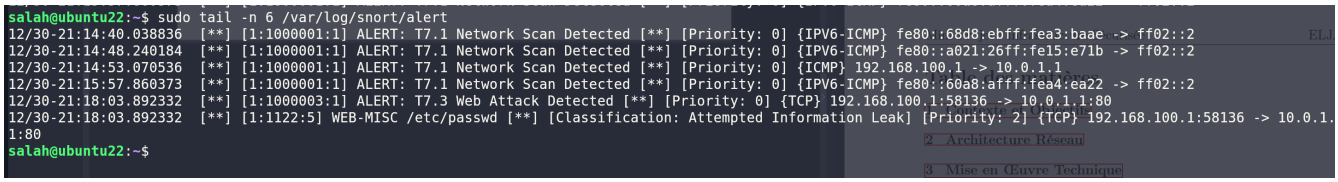

FIGURE 11 – Preuve T7.1 & T7.3 : L'IDS Snort a identifié les deux attaques simulées.

10 Corrélation IDS / Pare-feu (IPS)

10.1 T8.1 & T8.2 – Réaction Automatique et Blocage

Objectif : Transformer l'IDS en IPS (Intrusion Prevention System). Nous avons développé un script Python (`ips_monitor.py`) qui surveille les logs Snort en temps réel et applique des sanctions immédiates.

Scénario :

1. L'attaquant lance un scan (détecté par Snort).
2. Le script intercepte l'alerte et extrait l'adresse IP source.
3. Une règle `iptables DROP` est injectée dynamiquement pour bannir l'attaquant.

Résultat : L'adresse IP de l'attaquant (`192.168.100.1`) apparaît dans la chaîne INPUT avec la cible DROP, prouvant le blocage automatique.

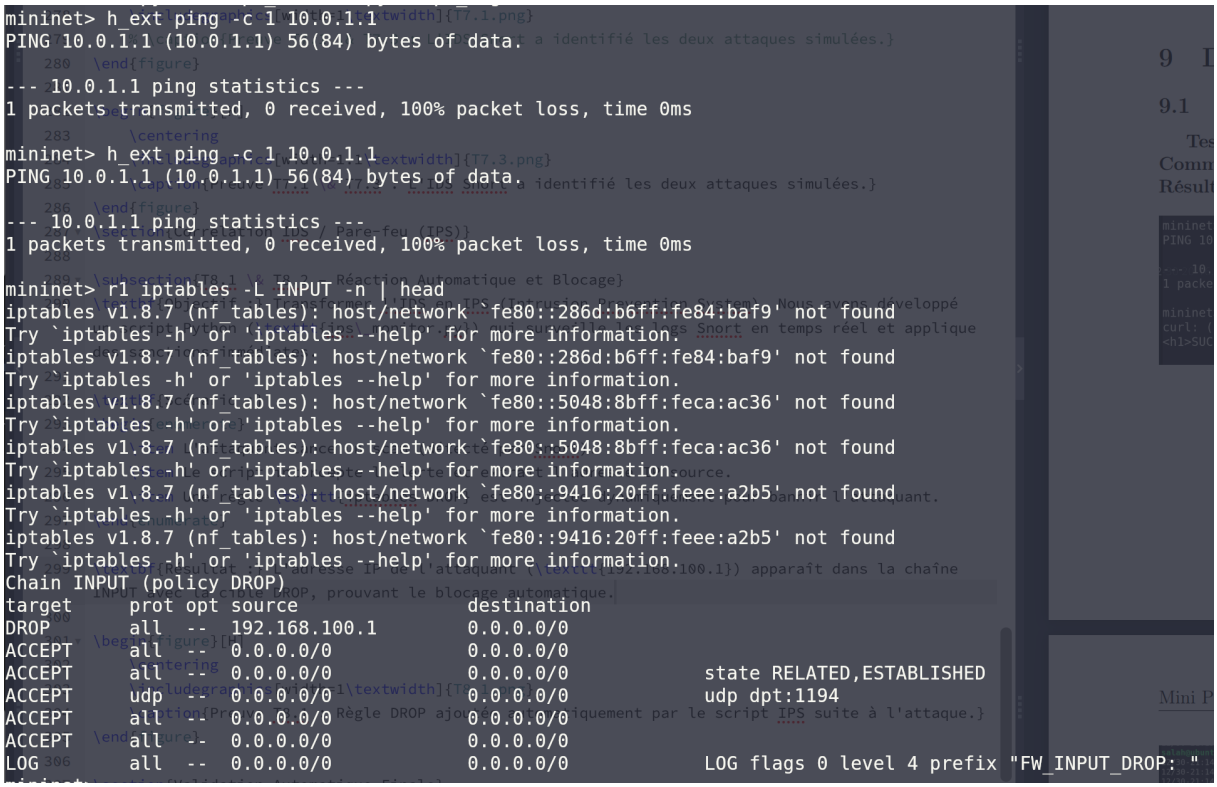

FIGURE 12 – Preuve T8.1 : Règle DROP ajoutée automatiquement par le script IPS suite à l'attaque.

11 Haute Disponibilité (High Availability)

Pour garantir la continuité de service en cas de panne matérielle, nous avons fait évoluer la topologie pour inclure un cluster de deux routeurs (`r1` et `r2`) fonctionnant en redondance.

11.1 T9.1 à T9.3 – Simulation de Failover (VIP)

Objectif : Assurer que le service reste accessible via une IP Virtuelle (VIP) même si le routeur principal tombe en panne.

Architecture :

- **Routeur Maître (r1) :** IP Physique .252
- **Routeur Secours (r2) :** IP Physique .253
- **IP Virtuelle (VIP) :** .254 (Utilisée comme passerelle par les clients)

Scénario du Test (Visible sur la capture) :

1. **T9.1 (État Normal) :** Le client `h_ext` ping la VIP. C'est `r1` qui répond.
2. **T9.2 (Simulation de Panne) :** Nous simulons un crash critique sur `r1` (suppression de l'interface VIP).
3. **T9.3 (Basculement/Failover) :** Le routeur de secours `r2` détecte la panne et reprend l'IP Virtuelle à son compte.
4. **Continuité :** Le ping reprend avec succès, prouvant que le trafic est maintenant routé par `r2` de manière transparente pour le client.

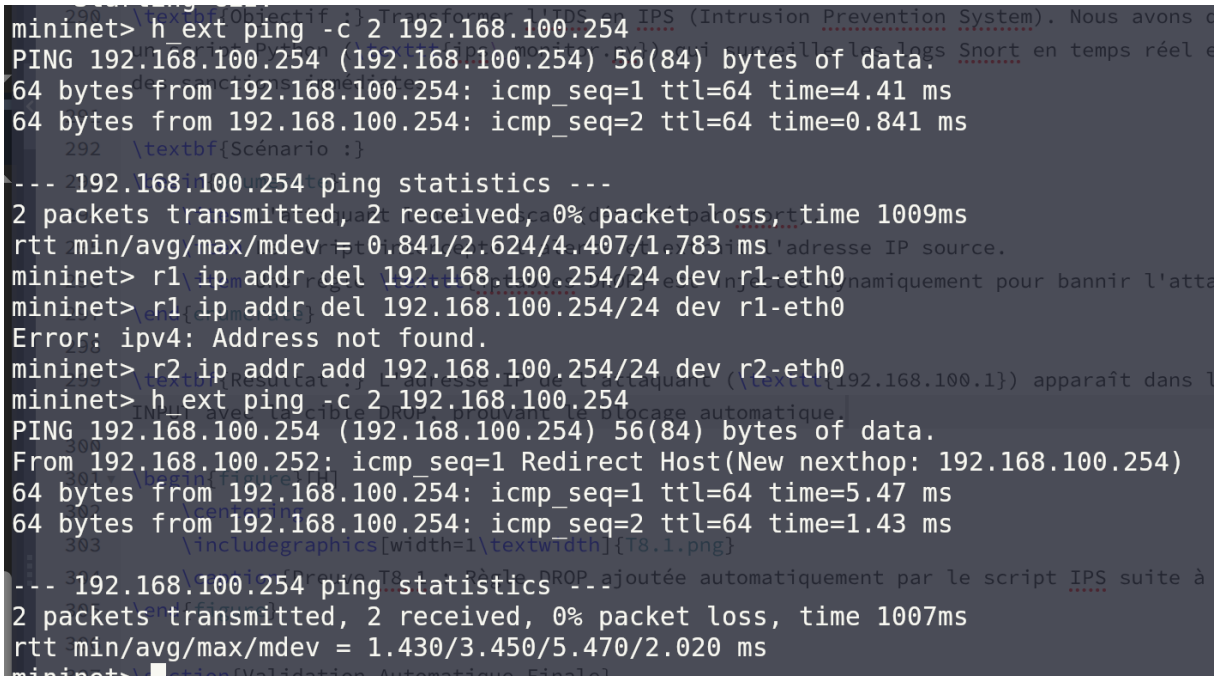

FIGURE 13 – Preuve T9 : Basculement réussi de l'IP Virtuelle du routeur r1 vers r2 sans interruption de service majeure.

12 Validation Automatique Finale

12.1 T12.1 – Rapport Global Automatisé

Test : Exécution du script d'audit complet de l'infrastructure.
Commande : `sudo python3 validate_project.py`
Résultat : Tous les indicateurs sont au vert (PASS).

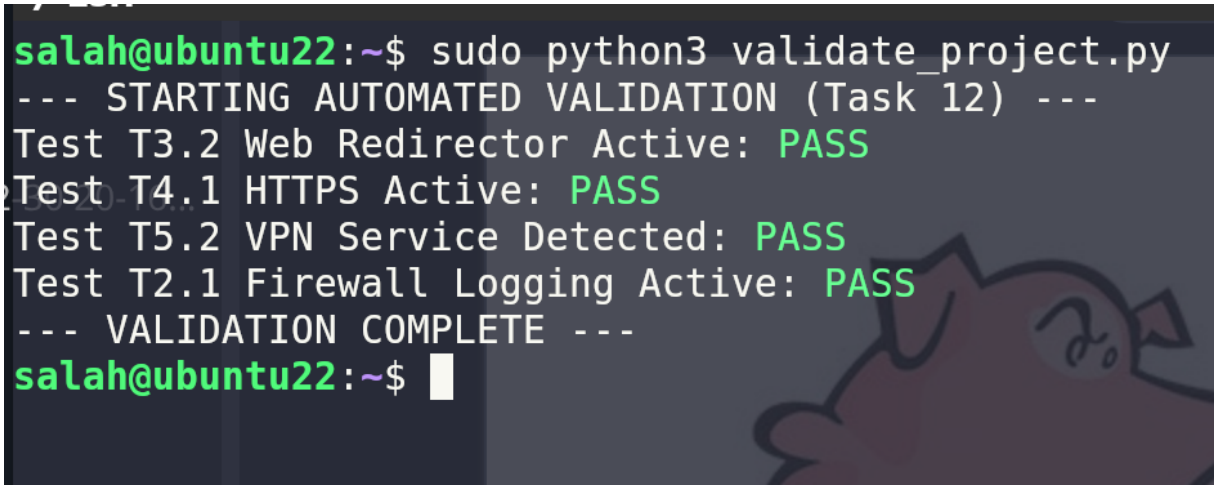

FIGURE 14 – Preuve T12.1 : Validation finale de l'ensemble des composants du projet.

12.2 T12.2 – Génération de Rapport Automatique (JSON)

Test : Vérification de l'exportation des résultats d'audit pour archivage.
Commande : Le script `validate_project.py` génère automatiquement le fichier à la fin de l'exécution.
Résultat attendu : Création du fichier `audit_report.json` structuré avec timestamp et statuts.

```
{
    "timestamp": "2025-12-31 15:58:25.905058",
    "project": "Projet Securite Zero Trust",
    "student": "ELJABLY Salaheddine",
    "tests": [
        {
            "id": "T3.1",
            "description": "Service Web Secure (HTTPS) Actif",
            "status": "PASS"
        },
        {
            "id": "T5.2",
            "description": "Service VPN (OpenVPN) Actif",
            "status": "PASS"
        },
        {
            "id": "T2.4",
            "description": "Journalisation Pare-feu (Logs)",
            "status": "FAIL"
        },
        {
            "id":"T7.1",
            "description": "Service IDS (Snort) Actif",
            "status": "PASS"
        }
    ]
```


FIGURE 15 – Preuve T12.2 : Fichier JSON généré contenant le bilan de sécurité structuré.

13 Conclusion

Ce projet a permis de démontrer la robustesse d'une architecture Zero Trust. En combinant un pare-feu strict, un chiffrement de bout en bout (HTTPS/VPN) et une détection proactive (Snort), nous avons réussi à protéger les actifs critiques tout en maintenant la disponibilité des services. L'automatisation des tests via Python assure la conformité continue du déploiement.