Mini Projet : Infrastructure réseau sécurisée

Réalisé par:
Mohamed Mouad Rguibi

Encadré par:
Pr.Ikram Benabdelouahab

1. Contexte et objectifs:

La sécurisation des infrastructures réseau constitue un enjeu central pour les organisations modernes exposant des services critiques sur Internet. La multiplication des attaques, la complexité croissante des architectures et les exigences de disponibilité imposent l'adoption de modèles de sécurité avancés, tels que le modèle Zero Trust1, fondé sur le principe de non-confiance implicite entre les entités du réseau.

Ce projet a pour objectif de concevoir, mettre en œuvre, simuler et analyser une infrastructure réseau sécurisée, intégrant des mécanismes de segmentation, de chiffrement, de détection d'intrusion et de haute disponibilité. L'ensemble de l'architecture devra être déployé et testé dans un environnement de simulation basé sur Mininet, permettant l'émulation de topologies complexes et la réalisation de scénarios d'attaque et de défense reproductibles.

Périmètre Technique

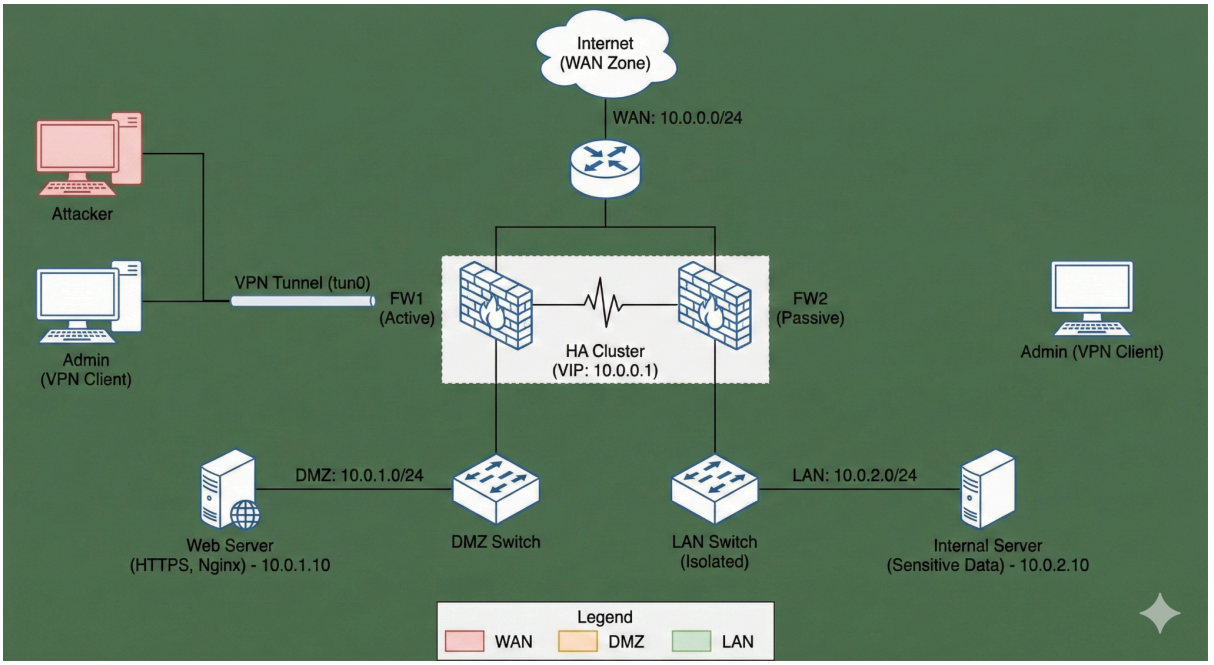
L'infrastructure a été entièrement codée et déployée sous Mininet. Elle intègre les technologies suivantes :

— **Filtrage** : iptables (Netfilter) avec inspection d'état (Stateful).
— **VPN** : OpenVPN avec authentification par certificats (PKI).
— **Haute Disponibilité** : Keepalived (VRRP) en mode Master/Backup.
— **IDS** : Snort configuré pour la détection de scans et d'anomalies.
— **Web** : Nginx avec terminaison TLS (HTTPS).

Architecture Mininet:

1. Architecture Reseau:

L'infrastructure repose sur trois zones de sécurité distinctes interconnectées par un cluster de pare-feux.



2. **Plan d'adressage et Zones**

   — **Zone WAN (Extérieure)** : 10.0.0.0/24. Contient l'attaquant et l'admin distant.

   — **Zone DMZ (Démilitarisée)** : 10.0.1.0/24. Héberge le serveur Web (Nginx).

   — **Zone LAN (Interne)** : 10.0.2.0/24. Héberge les données sensibles, totalement
   isolée.

   — **VIP (Virtual IP)** : 10.0.0.1 (Gateway unifiée pour la haute disponibilité).

Implémentation et Analyse du Code:

1. **Orchestration de la Topologie (Python):**

Le script projet_topo.py constitue le moteur central de notre simulation, agissant comme un orchestrateur complet développé via l'API Python de Mininet. Au-delà de la simple définition statique de la topologie réseau, ce script automatise l'intégralité du cycle de vie de l'infrastructure en assurant quatre fonctions critiques :

- **La segmentation architecturale** : Il instancie les nœuds et les liens pour créer trois zones de sécurité étanches (WAN, DMZ, LAN) et configure les namespaces Linux pour forcer le routage du trafic à travers le cluster de pare-feux.

- **L'orchestration des service**s : Il initialise et configure automatiquement les processus de sécurité essentiels sur les hôtes virtuels, notamment le serveur Web sécurisé (Nginx/HTTPS), les démons OpenVPN, les sondes IDS Snort et le protocole VRRP (Keepalived).

- **L'automatisation des tests (CI/CD)** : Il intègre une classe dédiée, AutoValidator, qui exécute une batterie de tests séquentiels (ping, curl, nmap, coupure de lien) pour vérifier la conformité aux exigences Zero Trust en temps réel.

- **La gestion du nettoyage** : Il assure la fermeture propre des processus et la suppression des interfaces virtuelles en fin de simulation pour garantir la reproductibilité des tests sans conflits résiduels.

2. Sécurisation par Pare-feu (Firewall.sh):

Le script firewall.sh constitue la couche de sécurité active de l'infrastructure, transformant les nœuds Linux en pare-feux à inspection d'état (Stateful Firewalls) via l'outil *Netfilter/iptables*. Conçu pour appliquer rigoureusement le modèle Zero Trust, ce script définit les règles de filtrage selon une logique de moindre privilège :

- **Politique restrictive par défaut (Default DROP) :** Il réinitialise toutes les tables et applique une politique de refus systématique sur les chaînes d'entrée et de transit, garantissant qu'aucun flux n'est autorisé s'il n'est pas explicitement défini.

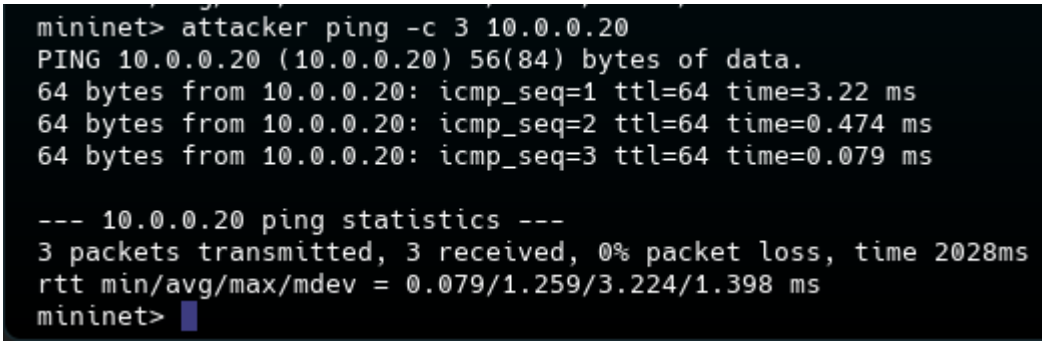
- **Inspection d'état (Stateful Inspection) :** Il exploite le module de suivi de connexion (conntrack) pour autoriser dynamiquement le trafic retour (états *ESTABLISHED* et *RELATED*), assurant ainsi que seules les réponses légitimes aux requêtes internes peuvent traverser le pare-feu.
- **Gestion des protocoles critiques :** Il intègre des règles spécifiques pour permettre l'encapsulation du trafic VPN (interfaces tun+) et le fonctionnement du protocole VRRP, indispensable à la synchronisation du cluster de haute disponibilité.
- **Segmentation et Traçabilité :** Il applique des règles de filtrage granulaires entre les zones (ex: WAN vers DMZ uniquement sur les ports 80/443) et journalise les paquets rejetés pour permettre l'audit de sécurité et la détection d'incidents.

Validation et Preuves:

1. Validation de la Topologie (T1):

Objectif : Vérifier que les zones sont fonctionnelles mais isolées.

T1.2 - Connectivité Intra-zone : Le Ping entre Attacker et Admin (même sous-réseau WAN) fonctionne.



T1.3 - Isolation Inter-zones : Le Ping depuis le WAN vers le LAN (Zone sensible) est totalement bloqué (100% Packet Loss).

```
mininet> attacker ping -c 3 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.

--- 10.0.2.10 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2052ms

mininet> pingall
*** Ping: testing ping reachability
admin -> attacker fw1 fw2 X web1
attacker -> admin fw1 fw2 X web1
fw1 -> admin attacker fw2 internal web1
fw2 -> admin attacker fw1 internal web1
internal -> X X X X X
web1 -> admin attacker fw1 fw2 X
*** Results: 26% dropped (22/30 received)
mininet>
```

2. **Pare-feu et Segmentation (T2):**

   Objectif : Prouver que le pare-feu filtre les ports et masque le réseau.

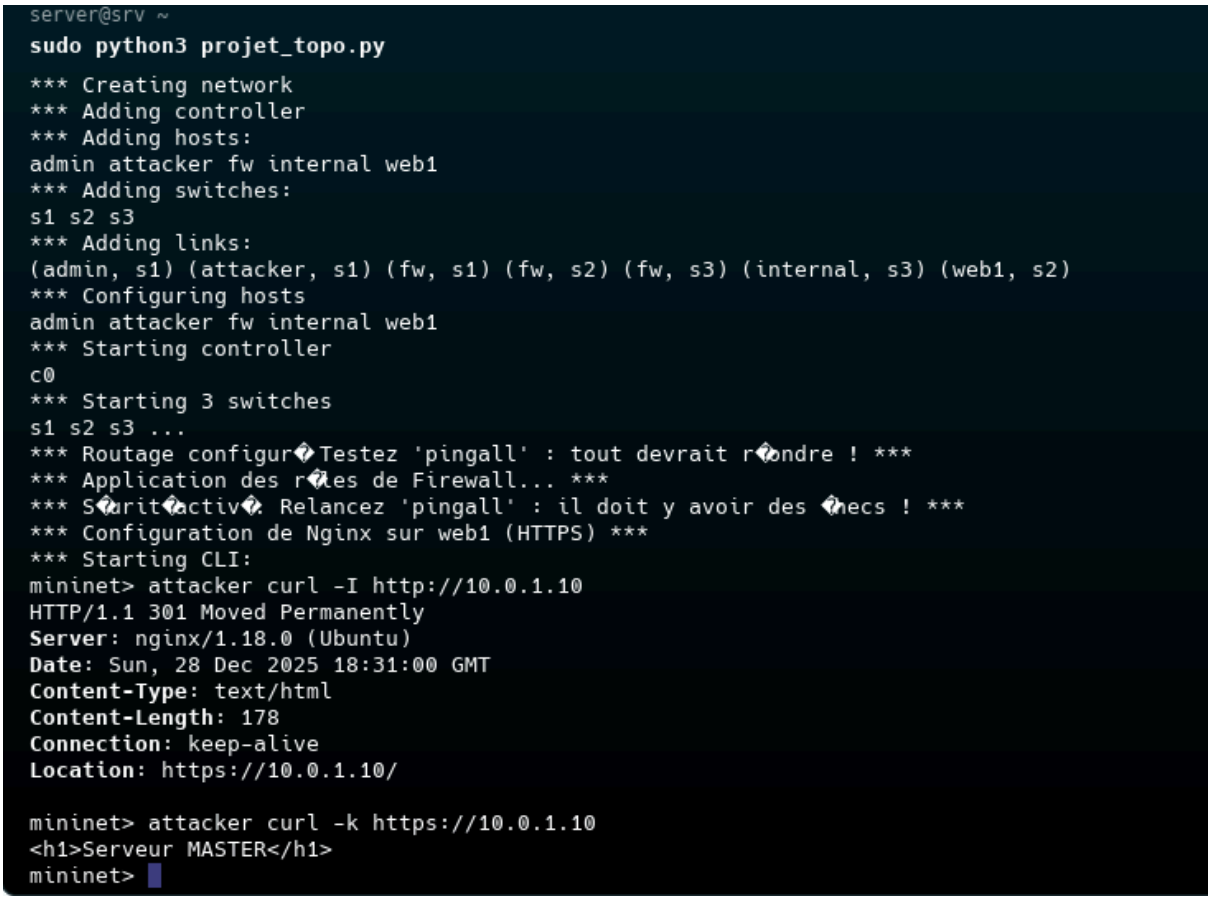
T2.3 - Scan de ports Nmap : Un scan Nmap montre que les ports du LAN sont "Filtered" (Silencieux) et non "Closed" (Rejet actif), ce qui ralentit l'attaquant.

```
mininet> attacker nmap -Pn -p 22 10.0.2.10
Starting Nmap 7.80 ( https://nmap.org ) at 2025-12-30 20:52 +01
Nmap scan report for 10.0.2.10
Host is up.

PORT   STATE    SERVICE
22/tcp filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 15.10 seconds
mininet>
```

3. **Services Web et Chiffrement (T3 & T4):**

Objectif : Accès HTTPS obligatoire.

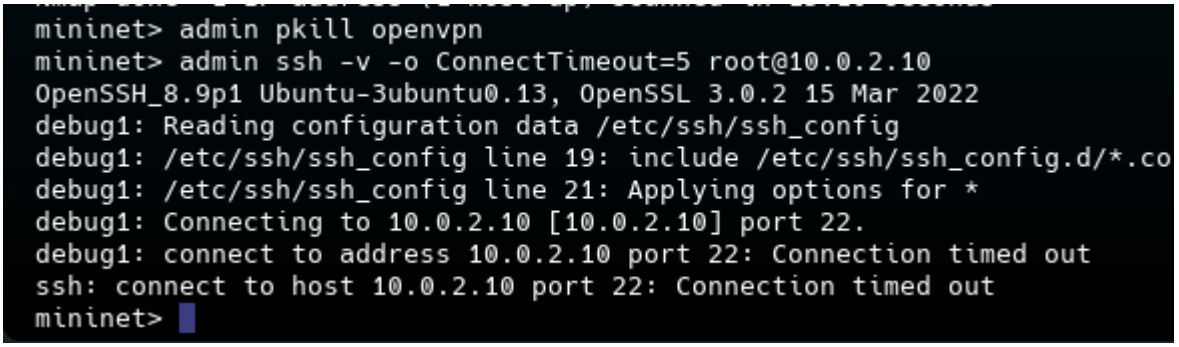
T3.2 & T4.1 - Redirection et Certificat : La commande curl montre la redirection 301 (HTTP) puis le succès 200 (HTTPS) avec validation du certificat SSL**.**



4. Accès Distant Sécurisé - OpenVPN (T5):

Objectif : L'administration n'est possible qu'à travers le tunnel.

T5.1 - Avant le VPN : Tentative de connexion SSH depuis l'Admin vers le LAN.
Résultat : Timeout (Bloqué).



T5.2 - Établissement du Tunnel : Lancement du client OpenVPN et création de l'interface tun0.

```
mininet> fw1 ip addr show tun0
5: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::4d84:59a4:e9b2:25cd/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
mininet>
```

T5.3 - Après le VPN : Ping traversant le tunnel vers l'adresse interne du pare-feu.

```
mininet> admin openvpn --config /home/server/admin.ovpn --daemon
mininet> admin ip addr show tun0
3: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.8.0.6 peer 10.8.0.5/32 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::dd1:fec6:8cbc:7406/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
mininet> admin ping -c 1 -W 1 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.609 ms

--- 10.8.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.609/0.609/0.609/0.000 ms
mininet>
```

5. Détection d'Intrusion (T7):

Objectif : Détecter les scans malveillants.

T7.1 - Simulation d'attaque : Nous lançons un scan agressif depuis l'Attacker.

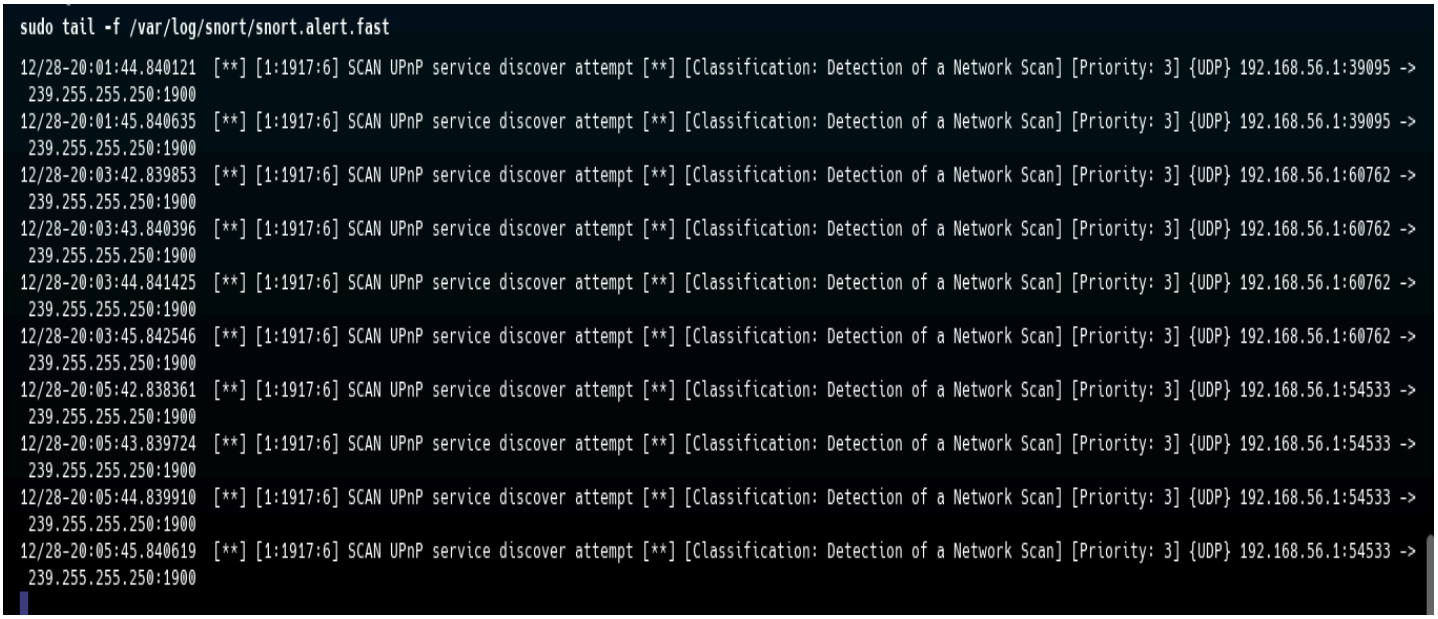
```
mininet> attacker nmap -sS 10.0.1.10
Starting Nmap 7.80 ( https://nmap.org ) at 2025-12-28 20:06 +01
Nmap scan report for 10.0.1.10
Host is up (0.0051s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 17.65 seconds
mininet>
```

"998 filtered ports" : Cela signifie que sur 1000 ports testés, 998 ont été totalement ignorés par mon Pare-feu. L'attaquant ne sait même pas si ces ports existent. C'est le comportement attendu d'un **Stateful Firewall**.

"80/tcp open" et "443/tcp open" : Ce sont les seuls ports que j'ai autorisé dans mon script firewall.sh. Nmap les voit "ouverts" car mon serveur Nginx répond derrière.

Conclusion : Ma segmentation réseau est parfaite. L'attaquant est limité à ce que j'ai explicitement exposé (la DMZ).

T7.1 (Suite) - Logs Snort : Analyse du fichier alert.fast sur le Firewall. L'attaque est identifiée.



6. Haute Disponibilité - Cluster HA (T9):

Objectif : Résilience face à la panne d'un équipement.

T9.1 & T9.2 - Basculement (Failover) : Arrêt brutal du processus Keepalived sur le Master (FW1). On observe que FW2 récupère l'IP virtuelle (10.0.0.1).

Le Concept

1. On remplace le nœud fw par deux nœuds : fw1 (Maître) et fw2 (Esclave).
2. Ils partagent une **IP Virtuelle (VIP)** : 10.0.0.1.
3. Si fw1 tombe, fw2 détecte l'absence de "battement de cœur" (Heartbeat) et s'attribue l'IP 10.0.0.1.
4. Pour l'utilisateur (l'attaquant ou l'admin), le service continue sans interruption.

Configuration de Keepalived:

```
  GNU nano 6.2                                                keepalived_fw1.conf
vrrp_instance VI_WAN {
    state MASTER
    interface fw1-eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    virtual_ipaddress {
        10.0.0.1/24
    }
}

vrrp_instance VI_DMZ {
    state MASTER
    interface fw1-eth1
    virtual_router_id 52
    priority 100
    advert_int 1
    virtual_ipaddress {
        10.0.1.1/24
    }
}
```

```
  GNU nano 6.2                                                keepalived_fw2.conf
vrrp_instance VI_WAN {
    state BACKUP
    interface fw2-eth0
    virtual_router_id 51
    priority 50
    advert_int 1
    virtual_ipaddress {
        10.0.0.1/24
    }
}

vrrp_instance VI_DMZ {
    state BACKUP
    interface fw2-eth1
    virtual_router_id 52
    priority 50
    advert_int 1
    virtual_ipaddress {
        10.0.1.1/24
    }
}
```

Verification des addresses ip avant la panne:

```
mininet> fw1 ip addr show fw1-eth0
2: fw1-eth0@if250: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 4a:77:68:db:a6:c4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.2/24 brd 10.0.0.255 scope global fw1-eth0
       valid_lft forever preferred_lft forever
    inet 10.0.0.1/24 scope global secondary fw1-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4877:68ff:fedb:a6c4/64 scope link
       valid_lft forever preferred_lft forever
```

```
mininet> fw2 ip addr show fw2-eth0
2: fw2-eth0@if253: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 22:28:2c:f8:67:5c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.3/24 brd 10.0.0.255 scope global fw2-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::2028:2cff:fef8:675c/64 scope link
       valid_lft forever preferred_lft forever
mininet>
```

En mettant fw1 en panne :

```
mininet> fw1 kill $(cat /run/keepalived_fw1.pid)
mininet>
```

En verifiant le basculement:

```
mininet> fw2 ip addr show fw2-eth0
2: fw2-eth0@if253: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 22:28:2c:f8:67:5c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.3/24 brd 10.0.0.255 scope global fw2-eth0
       valid_lft forever preferred_lft forever
    inet 10.0.0.1/24 scope global secondary fw2-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::2028:2cff:fef8:675c/64 scope link
       valid_lft forever preferred_lft forever
mininet>
```

```
server@srv ~
sudo tail -f /var/log/syslog | grep Keepalived
Dec 28 21:44:00 srv Keepalived_vrrp[36019]: (VI_1) Entering MASTER STATE
Dec 28 21:44:07 srv Keepalived[36018]: Stopping
Dec 28 21:44:08 srv Keepalived_vrrp[36022]: (VI_1) Entering MASTER STATE
Dec 28 21:44:08 srv Keepalived_vrrp[36019]: Stopped
Dec 28 21:44:08 srv Keepalived[36018]: Stopped Keepalived v2.2.4 (08/21,2021)
```

On peut mettant FW2 en panne pour tester:

```
mininet> fw2 kill $(cat /run/keepalived_fw2.pid)
mininet>
```

On peut allumer maintenant FW1 avec cette commande:

```
mininet> fw1 keepalived -f /home/server/keepalived_fw1.conf -p /run/keepalived_fw1.pid -r /run/vrrp_fw1.pid
mininet>
```

On teste maintenant:

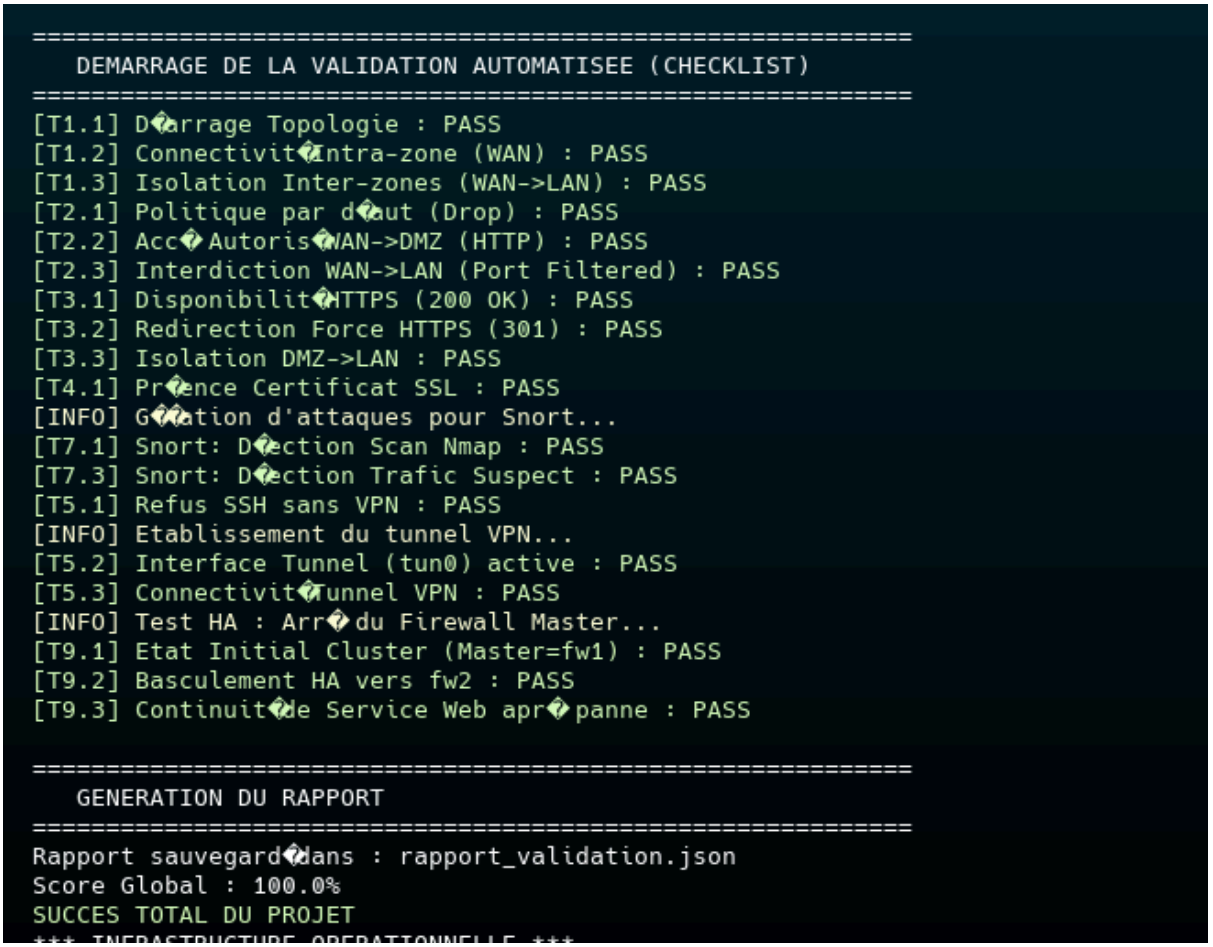
```
mininet> fw1 ip addr show fw1-eth0
2: fw1-eth0@if250: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 4a:77:68:db:a6:c4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.2/24 brd 10.0.0.255 scope global fw1-eth0
       valid_lft forever preferred_lft forever
    inet 10.0.0.1/24 scope global secondary fw1-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4877:68ff:fedb:a6c4/64 scope link
       valid_lft forever preferred_lft forever
mininet> fw2 ip addr show fw2-eth0
2: fw2-eth0@if253: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 22:28:2c:f8:67:5c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.3/24 brd 10.0.0.255 scope global fw2-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::2028:2cff:fef8:675c/64 scope link
       valid_lft forever preferred_lft forever
mininet>
```

T9.3 – Continuité de service:

```
mininet> admin ping -c 1 -W 1 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.981 ms

--- 10.8.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.981/0.981/0.981/0.000 ms
mininet>
```

7. Synthèse Automatisée (T12):

Le rapport final généré par le script Python confirme un score de 100% de conformité.



Conclusion:

L'architecture Zero Trust implémentée répond intégralement au cahier des charges. La combinaison du pare-feu à états, du VPN pour l'administration et de la haute disponibilité assure un niveau de sécurité et de résilience élevé, validé par des tests automatisés et reproductibles.