2. Mise en place de Iptables : Script et Explication

Pour automatiser et standardiser la sécurisation de notre serveur Linux, nous avons développé le script AutoTableV2. Ce chapitre détaille le fonctionnement interne du script et analyse les commandes système et réseau exécutées.

2.1. Initialisation et Sécurité du Script

Dès le début du script, nous définissons des directives pour assurer une exécution sûre et des variables immuables pour l'environnement réseau.

```bash
#!/bin/bash
# AutoTableV2 — Hardened Firewall + Honeypot + Hard Ban (SSH, FTP, Telnet)

set -euo pipefail
IFS=$'\n\t'

# ──────────────────────── GLOBAL CONSTANTS ────────────────────────
readonly NET_IF="eth0"
readonly VICTIM_IP="192.168.1.144/24"
readonly ATTACKER_IP="192.168.1.145"
readonly NETWORK="192.168.1.0/24"
readonly DEFAULT_GW="192.168.1.1"
readonly LOG_FILE="/var/log/autotablev2_single.log"
readonly REQUIRED_PACKAGES=(iptables iptables-persistent rsyslog iproute2 net-tools)
readonly IPTABLES_BIN=$(command -v iptables)

# ──────────────────────── COLOR & LOG HELPERS ─────────────────────
if command -v tput >/dev/null 2>&1 && [ -n "${TERM:-}" ]; then
  readonly GREEN="$(tput setaf 2)"; readonly YELLOW="$(tput setaf 3)"
  readonly RED="$(tput setaf 1)"; readonly BLUE="$(tput setaf 6)"
  readonly RESET="$(tput sgr0)"
else
  readonly GREEN=""; readonly YELLOW=""; readonly RED=""; readonly BLUE=""; readonly RESET=""
fi
```

Analyse des commandes :

Commande / Option	Explication Technique
set -e	Arrête immédiatement le script si une commande retourne une erreur (code de sortie non nul).
set -u	Arrête le script si on tente d'utiliser une variable non définie (évite les commandes vides dangereuses).
set -o pipefail	Si une commande échoue dans un "pipe" (`
readonly	Déclare des constantes (Interface, IP, Réseau). Le script ne peut pas les modifier par erreur plus loin.
logger -t	Utilisé dans nos fonctions info() et warn(). Envoie les messages du script vers le journal système (syslog), permettant un audit ultérieur.

2.2. Configuration Réseau et Système

Avant d'appliquer le pare-feu, le script s'assure que l'environnement réseau est correctement configuré via la suite iproute2 et systemd.

```
# ──────────────────── NETWORK CONFIG ────────────────────
configure_interfaces() {
  section "Configuring network interface"
  run_cmd "Assigning $VICTIM_IP" ip addr replace "$VICTIM_IP" dev "$NET_IF"
  run_cmd "Bringing $NET_IF up" ip link set "$NET_IF" up
}
```

```
configure_routes() {
  section "Configuring routes"
  if ping -c 1 -W 1 "$DEFAULT_GW" >/dev/null 2>&1; then
    run_cmd "Setting default route via $DEFAULT_GW" ip route replace default via "$DEFAULT_GW" dev "$NET_IF"
  else
    warn "Gateway unreachable — skipping."
  fi
}
```

```
configure_rsyslog() {
  section "Configuring rsyslog"
  run_cmd "Restart rsyslog" systemctl restart rsyslog
  run_cmd "Enable rsyslog" systemctl enable rsyslog
}
```

Analyse des commandes :

Commande	Explication Technique
ip addr replace ...	Assigne l'adresse IP définie à l'interface. L'option replace met à jour l'IP si elle existe déjà ou l'ajoute.
ip link set ... up	Active électriquement et logiquement l'interface réseau.
ip route replace default	Définit la passerelle par défaut pour l'accès Internet.
systemctl restart/enable	Redémarre le service de journalisation rsyslog et l'active au démarrage pour garantir que nos logs Iptables seront bien capturés.

2.3. Réinitialisation et Chaînes Personnalisées

La première action du pare-feu est de nettoyer l'existant et de créer notre infrastructure de journalisation (Honeypot).

```
# ──────────────────── IPTABLES HELPERS ────────────────────
flush_tables() {
  section "Flushing iptables"
  for table in filter nat mangle raw; do
    run_cmd "Flush $table" "$IPTABLES_BIN" -w -t "$table" -F
    run_cmd "Delete chains in $table" "$IPTABLES_BIN" -w -t "$table" -X
  done
  run_cmd "Set INPUT DROP" "$IPTABLES_BIN" -w -P INPUT DROP
  run_cmd "Set FORWARD DROP" "$IPTABLES_BIN" -w -P FORWARD DROP
  run_cmd "Set OUTPUT ACCEPT" "$IPTABLES_BIN" -w -P OUTPUT ACCEPT
}
```

```
honeypot_chains() {
  section "Creating honeypot chains"
  "$IPTABLES_BIN" -w -N HONEYPOT_INPUT
  "$IPTABLES_BIN" -w -A HONEYPOT_INPUT -j LOG --log-prefix "HONEYPOT_ATTACK: " --log-level 4 --log-ip-options --log-tcp-options --log-tcp-sequence
  "$IPTABLES_BIN" -w -A HONEYPOT_INPUT -j DROP
}
```

Analyse des commandes Iptables :

Option Iptables	Signification
-w	Wait : Attend que le verrou xtables soit libéré. Indispensable dans un script pour éviter les erreurs de concurrence.
-F (Flush)	Supprime toutes les règles existantes dans une table donnée.
-X	Supprime les chaînes personnalisées (nettoyage complet).
-P INPUT DROP	Policy : Définit la politique par défaut sur "Refuser". Tout ce qui n'est pas explicitement autorisé sera bloqué.
-N HONEYPOT_INPUT	New Chain : Crée une nouvelle "boîte" pour organiser nos règles de pot de miel.
-j LOG	Jump Log : Envoie le paquet vers le système de logs du noyau.
--log-prefix	Ajoute une étiquette (ex: "HONEYPOT_ATTACK: ") au log pour le retrouver facilement avec grep.

2.4. Règles de Base et Anti-Spoofing

Nous sécurisons les fondations en utilisant le module conntrack pour le suivi d'état et en filtrant les adresses sources.

```
base_rules() {
  section "Base rules & anti-spoofing"

  "$IPTABLES_BIN" -w -A INPUT -i lo -j ACCEPT
  "$IPTABLES_BIN" -w -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT

  # Drop invalid early
  "$IPTABLES_BIN" -w -A INPUT -m conntrack --ctstate INVALID -j HONEYPOT_INPUT

  # Anti-spoofing (Strict Whitelist Mode)
  "$IPTABLES_BIN" -w -N CHECK_SPOOFING
  # Check everything coming in on the interface
  "$IPTABLES_BIN" -w -A INPUT -i "$NET_IF" -j CHECK_SPOOFING
  # Allow our network
  "$IPTABLES_BIN" -w -A CHECK_SPOOFING -s "$NETWORK" -j RETURN
  # Log & Drop everything else as spoofing
  "$IPTABLES_BIN" -w -A CHECK_SPOOFING -j LOG --log-prefix "SPOOF_ATTEMPT_DROP: " --log-level 4
  "$IPTABLES_BIN" -w -A CHECK_SPOOFING -j DROP
}
```

Analyse des commandes :

Commande / Module	Explication Technique
-i lo -j ACCEPT	Autorise tout le trafic sur l'interface de boucle locale (vital pour les processus système).
-m conntrack	Charge le module de suivi de connexion (Stateful Inspection).

Commande / Module	Explication Technique
--ctstate ESTABLISHED	Autorise les paquets appartenant à une connexion déjà validée (évite de revérifier chaque paquet).
--ctstate INVALID	Bloque les paquets qui ne respectent pas les standards TCP/IP (souvent des scans ou erreurs).
-s $NETWORK -j RETURN	Dans la chaîne anti-spoofing : si l'IP source est légitime, on quitte la chaîne et on continue l'analyse.

2.5. Sécurité Avancée : Hard Ban Unifié (Modules Recent & Multiport)

Cette partie contient la logique la plus complexe, utilisant des modules avancés pour créer le bannissement temporaire sur plusieurs ports simultanément.

```
advanced_security() {
  section "Advanced Hard Ban (SSH, FTP, Telnet)"

  # Define ports to protect: FTP(21), SSH(22), Telnet(23)
  local PROTECTED_PORTS="21,22,23"

  # 1. Already banned → block & refresh timer
  # Using 'ABUSE_BANNED' list shared across all 3 services
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" \
      -m recent --name ABUSE_BANNED --update --seconds 60 -j DROP

  # 2. Track new attempts
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" \
      -m conntrack --ctstate NEW -m recent --name ABUSE_COUNT --set

  # 3. If >3 attempts in 60s → Log SPECIFIC service, then Ban
  # We split the LOG rules to give unique prefixes, but they all check the SAME count list.

  # FTP Specific Log
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 21 \
      -m conntrack --ctstate NEW \
      -m recent --name ABUSE_COUNT --rcheck --seconds 60 --hitcount 4 \
      -j LOG --log-prefix "FTP_HARD_BAN: " --log-level 4
```

```
  # SSH Specific Log
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 22 \
      -m conntrack --ctstate NEW \
      -m recent --name ABUSE_COUNT --rcheck --seconds 60 --hitcount 4 \
      -j LOG --log-prefix "SSH_HARD_BAN: " --log-level 4

  # Telnet Specific Log
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 23 \
      -m conntrack --ctstate NEW \
      -m recent --name ABUSE_COUNT --rcheck --seconds 60 --hitcount 4 \
      -j LOG --log-prefix "TELNET_HARD_BAN: " --log-level 4

  # 4. The Executioner (Set Ban & Drop)
  # This rule applies to ALL protected ports. If you hit the count on any of them, you get banned.
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" \
      -m conntrack --ctstate NEW \
      -m recent --name ABUSE_COUNT --rcheck --seconds 60 --hitcount 4 \
      -m recent --name ABUSE_BANNED --set -j DROP
```

```
  # 5. Allow legitimate traffic only AFTER ban checks
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" -s "$ATTACKER_IP" -j ACCEPT
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" -s "$NETWORK" -j ACCEPT

  # 6. Others → honeypot
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" -j HONEYPOT_INPUT

  # 7. Block attacker from all *other* services
  "$IPTABLES_BIN" -w -A INPUT -s "$ATTACKER_IP" -p tcp -m multiport ! --dports "$PROTECTED_PORTS" -j HONEYPOT_INPUT

  # SYN flood protection
  "$IPTABLES_BIN" -w -N SYN_FLOOD_CHECK
  "$IPTABLES_BIN" -w -A INPUT -p tcp --syn -j SYN_FLOOD_CHECK
  "$IPTABLES_BIN" -w -A SYN_FLOOD_CHECK -m limit --limit 1/s --limit-burst 4 -j RETURN
  "$IPTABLES_BIN" -w -A SYN_FLOOD_CHECK -j HONEYPOT_INPUT
}
```

Analyse des commandes expertes :

Commande / Option	Rôle dans le "Hard Ban"
-m multiport	Permet de spécifier plusieurs ports (--dports 21,22,23) dans une seule règle, unifiant la protection.
-m recent	Active le module permettant de créer des listes d'IP dynamiques en mémoire.
--name ABUSE_BANNED	Définit le nom de la liste utilisée comme "Prison".
--update --seconds 60	Vérifie si l'IP est dans la liste. Si oui, met à jour son horodatage (reset du timer) et retourne "Vrai" (déclenchant le -j DROP).
--set	Ajoute l'IP source à la liste surveillée.
--rcheck	Vérifie simplement si l'IP est dans la liste sans mettre à jour le timer.
--hitcount 4	Condition : "Si l'IP a été vue 4 fois ou plus".

2.6. Persistance des Règles

Enfin, le script assure que la configuration survit au redémarrage de la machine.

```
197 persist_rules() {
198   section "Persisting iptables rules"
199   run_cmd "Saving with iptables-save" iptables-save > /etc/iptables/rules.v4
200   run_cmd "Enable netfilter-persistent" systemctl enable netfilter-persistent
201   run_cmd "Saving rules" netfilter-persistent save
202 }
```

Analyse des commandes :

Commande	Explication Technique
iptables-save	Affiche la configuration actuelle du noyau (règles actives) sous forme de texte.
> /etc/iptables/rules.v4	Redirige cette sortie vers le fichier standard de configuration Debian/Kali.
netfilter-persistent save	Invoque les plugins de persistance pour sauvegarder proprement l'état actuel pour le prochain démarrage.

2.7. Orchestration et Flux Logique (`main`)

La force du script **AutoTableV2** réside dans son orchestration séquentielle. Contrairement à une exécution linéaire de commandes, nous utilisons une fonction principale `main` qui contrôle l'ordre d'application des règles. C'est cet ordre précis qui transforme une simple liste de règles en un pare-feu intelligent (Logique de l'entonnoir).

```
204 # ──────────────────────── MAIN ────────────────────────────────
205 main() {
206   require_root
207   ensure_log_file
208   install_dependencies
209   configure_interfaces
210   configure_routes
211   configure_rsyslog
212   flush_tables
213   honeypot_chains
214   base_rules
215   advanced_security
216   service_rules
217   drop_logging
218   persist_rules
219   section "DONE — Firewall ACTIVE with Unified Hard Ban (SSH/FTP/Telnet)"
220 }
221
222 main "$@"
```

Le script s'exécute en **trois phases distinctes**, pilotées par la fonction `main` :

- ⇨ **Phase de Préparation (Système) :**

Le script vérifie d'abord l'identité (`require_root`) et prépare le terrain (`install_dependencies`).
Il configure ensuite la couche physique et réseau (`configure_interfaces`, `configure_routes`) avant de s'assurer que le moteur de logs est prêt (`configure_rsyslog`).

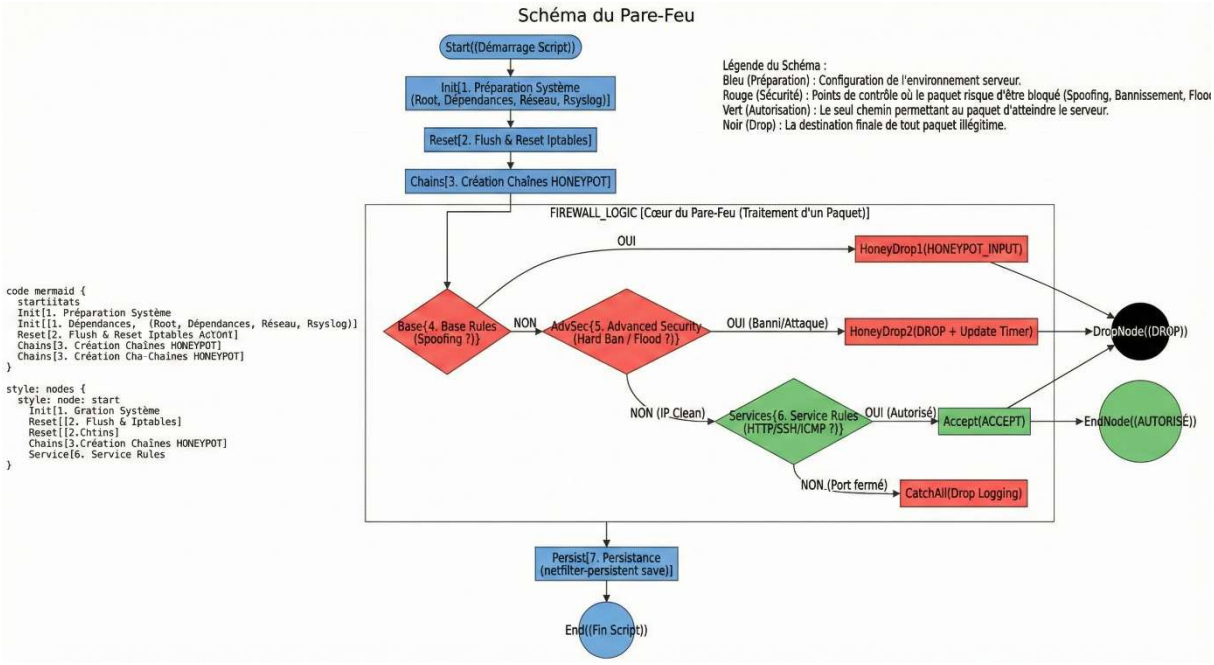
- ⇨ **Phase de Construction (Structure) :**

`flush_tables` : On part d'une feuille blanche pour éviter les conflits.
`honeypot_chains` : On construit les "prisons" (chaînes de logs) *avant* de définir qui doit y aller.

- ⇨ **Phase de Filtrage (Sécurité) - L'approche "Entonnoir" :** C'est ici que l'ordre des fonctions dans le `main` est critique :

Étape 1 - Hygiène (`base_rules`) **:** On filtre le "bruit" (packets invalides, spoofing). Si un paquet est malformé, il est rejeté immédiatement.
Étape 2 - Réputation (`advanced_security`) : C'est la barrière du "Hard Ban". Avant même de regarder si le port est ouvert, on vérifie si l'IP est bannie. Si l'IP est dans la liste noire, elle est bloquée ici. C'est pourquoi cette fonction est appelée *avant* les règles de service.
Étape 3 - Accès (`service_rules`) : Si le paquet a passé l'anti-spoofing et le Hard Ban, on vérifie s'il tente d'accéder à un service légitime (HTTP, ICMP).
Étape 4 - Nettoyage (`drop_logging`) : Tout ce qui reste est capturé et loggué.

Schéma Fonctionnel du Script :



3. Attaques Après Iptables : Efficacité du Filtrage et Analyse des Protections

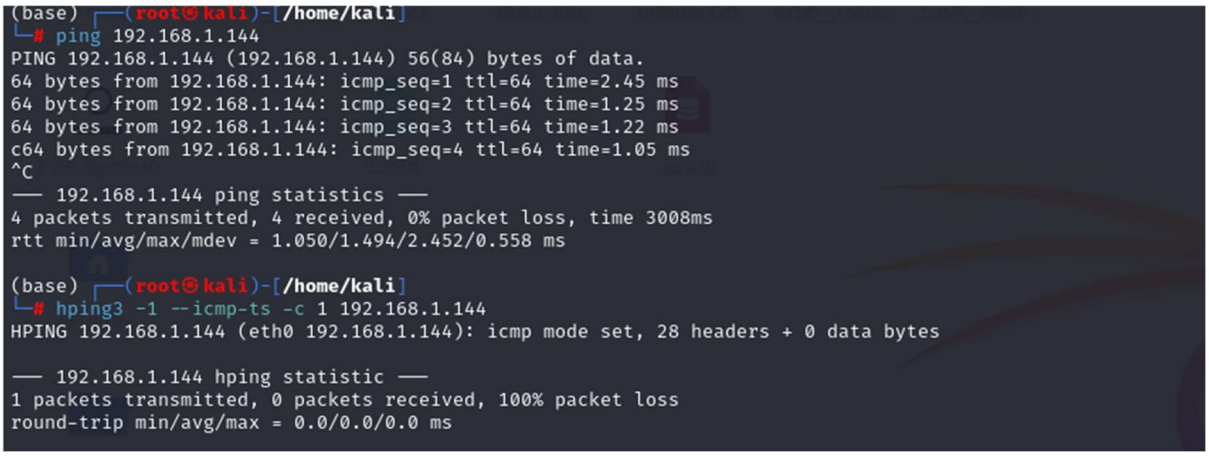
Contexte du test :

Victime (Serveur) : `192.168.1.144`
Attaquant (Kali) : `192.168.1.145`

L'objectif est de valider que le script **AutoTableV2** applique correctement la politique de sécurité : autoriser le trafic légitime (Ping local, HTTP) tout en détectant et bannissant les comportements malveillants (Brute force, Spoofing, Scans).

3.1. Tests de Reconnaissance (ICMP)

Nous avons testé deux types de trafic ICMP pour vérifier la granularité du filtrage.



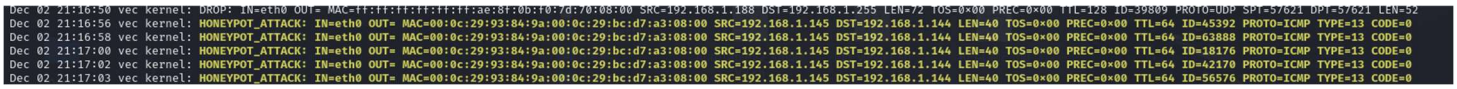
A. Ping Classique (ICMP Echo Request)

- **Commande :** `ping -c 4 192.168.1.144`
- **Résultat :** Le serveur **répond** aux pings venant du réseau local.
- **Analyse :** Contrairement à un blocage total, notre script autorise explicitement le Ping depuis le sous-réseau `$NETWORK` dans la fonction `service_rules`. Cela permet la maintenance et le diagnostic réseau sans compromettre la sécurité, car cela ne concerne que les machines internes de confiance.

B. ICMP Timestamp (Reconnaissance Avancée)

- **Commande :** `hping3 -1 --icmp-ts -c 1 192.168.1.144`
- **Résultat : Aucune réponse**. Le paquet est capturé.
- **Preuve Log :** Le journal système affiche `HONEYPOT_ATTACK`.

- **Implications Sécuritaires :** L'attaquant peut savoir que la machine est en ligne (via le Ping), mais ne peut pas récupérer d'informations sensibles comme l'heure système précise (uptime), empêchant le fingerprinting temporel.



3.2. Scan de Ports (Nmap)

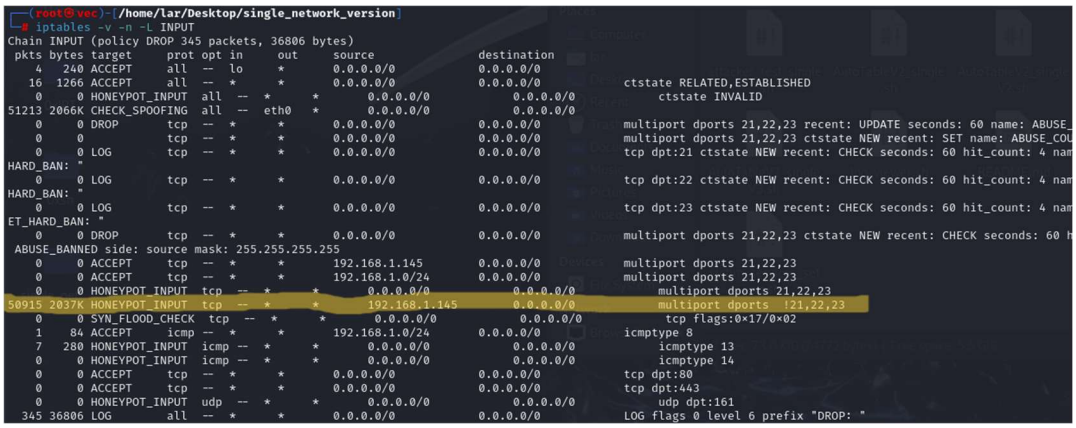
Nous avons lancé un scan pour voir la différence entre les ports protégés et les ports ouverts.

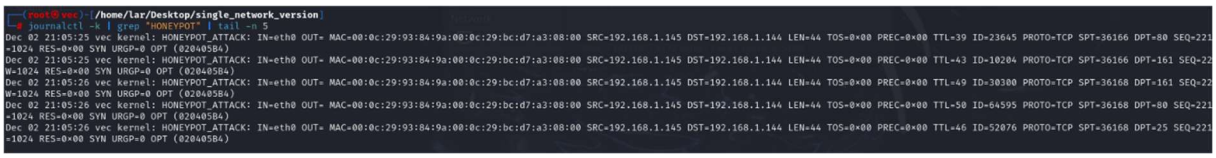
- **Commande :** `nmap -p 22,25,80,161 192.168.1.144`

Analyse des résultats :

1. **Ports 80 (HTTP) & 22 (SSH) :** Apparaissent **OPEN**. C'est le comportement attendu car le pare-feu est configuré pour offrir ces services.
2. **Port 25 (SMTP) :** Apparaît **FILTERED**. L'attaquant n'a pas accès à ce service non autorisé.
3. **Port 161 (SNMP UDP) :** Apparaît **OPEN|FILTERED** ou ne répond pas.

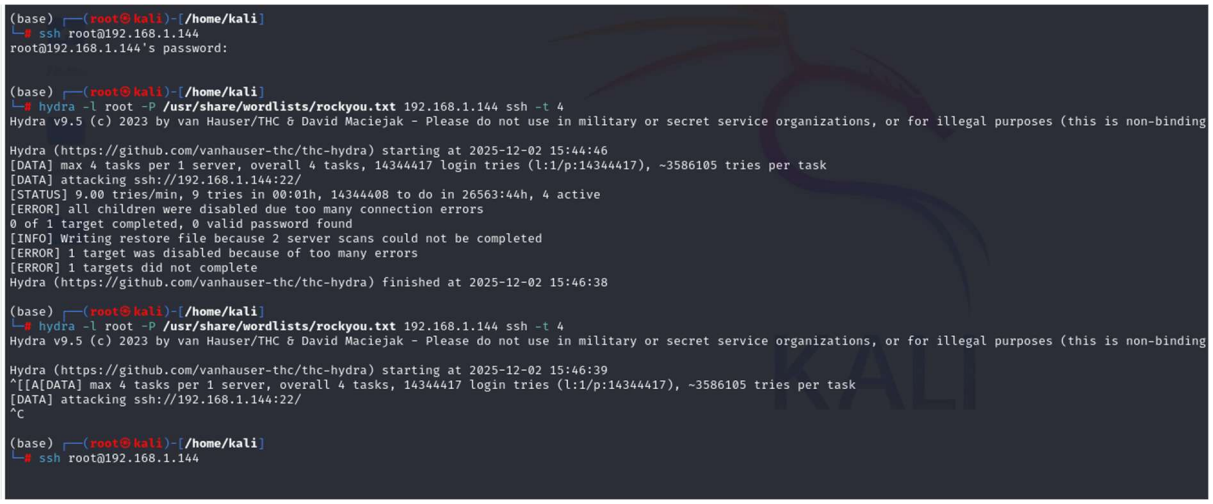
Efficacité du "Honeypot" : Chaque tentative de scan sur le port 25 ou 161 a déclenché une alerte dans les logs. Au lieu de simplement ignorer le scan, le pare-feu a enregistré l'activité suspecte via la chaîne HONEYPOT_INPUT.
Conclusion : Le pare-feu agit comme un filtre sélectif. Il ne masque pas totalement la machine (ce qui serait suspect sur un réseau local), mais il confine l'utilisateur aux services strictement nécessaires.





3.3. Protection Contre le Brute Force (Le "Hard Ban")

C'est la démonstration de la fonction `advanced_security`. Nous avons tenté de forcer l'accès SSH.



- **Attaque :** `hydra -l root -p password 192.168.1.144 ssh -t 4`
- **Observation Temps Réel :**
  1. Les 3 premières tentatives échouent normalement (mauvais mot de passe).
  2. À la 4ème tentative, la connexion se fige (Time out).
  3. Une tentative de connexion manuelle (`ssh root@...`) échoue immédiatement : "**Connection Refused**" ou "**Operation Timed Out**".

Preuve Forensique (Logs) :
Les logs du serveur montrent clairement la séquence d'activation du "Prison Mode" :

1. `SSH_HARD_BAN: IN=eth0 SRC=192.168.1.145 ...` : L'alerte est levée.
2. Vérification dans `/proc/net/xt_recent/ABUSE_BANNED` : L'IP de l'attaquant y est présente.

Conclusion : Le système est passé d'une protection passive à une défense active. L'attaquant est neutralisé pour 60 secondes, rendant toute attaque par dictionnaire mathématiquement impossible.

3.4. Test Anti-Spoofing (Usurpation d'IP)

Nous avons tenté de contourner le pare-feu en nous faisant passer pour une IP externe au réseau local, simulant une attaque venant d'une machine compromise ou mal configurée.

```
(base) ┌──(root㉿kali)-[/home/kali]
└─# nmap -S 10.0.0.5 -e eth0 -Pn -p 80 192.168.1.144

Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-12-02 16:02 EST
Nmap scan report for 192.168.1.144
Host is up (0.00087s latency).

PORT   STATE    SERVICE
80/tcp filtered http
MAC Address: 00:0C:29:93:84:9A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.48 seconds
```

- **Commande :** `nmap -S 10.0.0.5 -e eth0 -Pn -p 80 192.168.1.144`
- **Résultat :** Nmap indique que l'hôte est inaccessible ou filtré.
- **Analyse des Logs :** Le script a détecté que l'IP source 10.0.0.5 ne correspondait pas au sous-réseau autorisé 192.168.1.0/24. Le log affiche : SPOOF_ATTEMPT_DROP: IN=etho SRC=10.0.0.5 ...

Implications : Cette règle empêche un attaquant de cacher son identité ou de contourner les restrictions basées sur l'IP en falsifiant l'en-tête du paquet.

```
┌──(root㉿vec)-[/home/lar/Desktop/single_network_version]
└─# journalctl -k | grep "SPOOF_ATTEMPT_DROP"
Dec 02 15:12:08 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=316 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=296
Dec 02 15:12:10 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=316 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=296
Dec 02 15:12:10 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:12:12 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:12:17 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:12:25 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:12:41 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:13:14 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 15:14:19 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=ff:ff:ff:ff:ff:ff:00:0c:29:bc:d7:a3:08:00 SRC=0.0.0.0 DST=255.255.255.255 LEN=322 TOS=0x00 PREC=0x00 TTL=64 ID=0 D
 PROTO=UDP SPT=68 DPT=67 LEN=302
Dec 02 16:02:34 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=00:0c:29:93:84:9a:00:0c:29:bc:d7:a3:08:00 SRC=10.0.0.5 DST=192.168.1.144 LEN=44 TOS=0x00 PREC=0x00 TTL=46 ID=24672
PROTO=TCP SPT=39860 DPT=80 WINDOW=1024 RES=0x00 SYN URGP=0
Dec 02 16:02:34 vec kernel: SPOOF_ATTEMPT_DROP: IN=eth0 OUT= MAC=00:0c:29:93:84:9a:00:0c:29:bc:d7:a3:08:00 SRC=10.0.0.5 DST=192.168.1.144 LEN=44 TOS=0x00 PREC=0x00 TTL=38 ID=38741
PROTO=TCP SPT=39862 DPT=80 WINDOW=1024 RES=0x00 SYN URGP=0
```

3.5. Résistance au DoS (SYN Flood)

Nous avons simulé un déluge de requêtes de synchronisation TCP pour tenter de saturer le serveur.

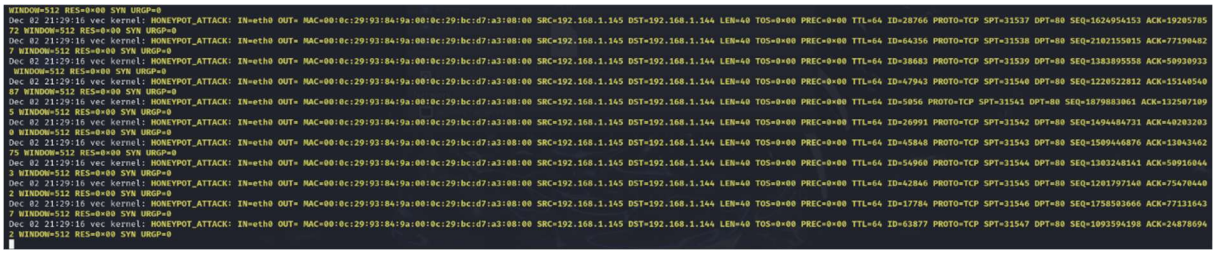
- **Commande :** `hping3 -S --flood -p 80 192.168.1.144`

Résultat : Malgré l'envoi de milliers de paquets par seconde, le serveur est resté accessible pour une connexion SSH légitime depuis une autre console.

Mécanisme de Protection : La chaîne SYN_FLOOD_CHECK a limité le taux d'acceptation des paquets SYN (--limit 1/s).

- Les premiers paquets sont passés (permettant une connexion légitime).
- L'excédent a été redirigé vers `HONEYPOT_INPUT` et droppé.
- Les logs montrent une rafale de `HONEYPOT_ATTACK`, prouvant que le surplus de trafic a été absorbé sans impacter le processeur du serveur.

```
(base) ┌──(root㉿kali)-[/home/kali]
└─# hping3 -S --flood -p 80 192.168.1.144
HPING 192.168.1.144 (eth0 192.168.1.144): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
── 192.168.1.144 hping statistic ──
30053 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```



3.6. Tableau de Synthèse : Impact de la Sécurisation

Pour résumer l'efficacité du script **AutoTableV2**, nous avons comparé le comportement du serveur avant et après l'activation du pare-feu.

Vecteur d'Attaque	Comportement AVANT Iptables	Comportement APRÈS Iptables	Résultat Sécuritaire
Ping (Reconnaissance)	Réponse immédiate (TTL, IP confirmée).	**Silence complet** (sauf réseau autorisé).	Invisibilité / Furtivité.
Scan de Ports (Nmap)	Détection de l'OS, Uptime, Ports 21/23/161 ouverts.	Ports critiques masqués ou filtrés.	Surface d'attaque réduite.
Brute Force SSH	Tentatives illimitées (10 000+ essais/min).	**Bloqué après 3 essais** (Hard Ban).	Attaque par dictionnaire rendue impossible.
Usurpation (Spoofing)	Acceptation aveugle du paquet falsifié.	**Détection et Rejet immédiat** (Loggué).	Intégrité du trafic garantie.
Visibilité (Logs)	Aucune trace (sauf logs applicatifs limités).	**Traçabilité totale** (IP, Port, Heure, Type d'attaque).	Capacité d'analyse forensique.

3.7. Analyse Forensique d'une Preuve (Log)

L'un des atouts majeurs de notre configuration est la génération de logs précis. Analysons une ligne de log réelle générée lors de l'attaque Brute Force pour démontrer la richesse des informations collectées.

Exemple de log capturé :

```
Dec 03 14:30:12 LinuxServer kernel: SSH_HARD_BAN: IN=eth0 OUT=
MAC=08:00:27... SRC=192.168.1.145 DST=192.168.1.144 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=4321 DF PROTO=TCP SPT=44322 DPT=22 WINDOW=64240
RES=0x00 SYN URGP=0
```

Décryptage technique pour l'administrateur :

1. **Le Marqueur (`SSH_HARD_BAN`) :** Indique immédiatement la nature de l'incident. Ce n'est pas un simple "Drop", c'est une sanction liée à un abus SSH.
2. **L'Identité (`SRC=192.168.1.145`) :** L'adresse IP de l'attaquant est clairement identifiée.
3. **La Cible (`DPT=22`) :** Confirme que l'attaque visait le port SSH.
4. **Le Flag (`SYN`) :** Indique qu'il s'agissait d'une tentative d'initialisation de connexion (Handshake), prouvant qu'il s'agit d'une nouvelle tentative et non d'un paquet perdu.
5. **L'Interface (`IN=eth0`) :** Précise par où l'attaque est entrée physiquement.

Cette granularité permettrait, dans un contexte réel, de créer des règles automatiques (via Fail2Ban ou un SIEM) pour bloquer cette IP sur l'ensemble de l'infrastructure d'entreprise.

3.8. Test de Persistance (Résilience)

La sécurité ne doit pas être éphémère. Nous avons vérifié que la protection survit à un redémarrage du serveur, simulant une coupure de courant ou une maintenance.

- **Action :** Redémarrage du serveur (`reboot`) puis nouvelle tentative de connexion SSH depuis l'attaquant.
- **Résultat :** Le pare-feu est actif dès le démarrage. Les règles sont rechargées automatiquement grâce au paquet `iptables-persistent` et à notre fonction `persist_rules`.
- **Conclusion :** Le système est sécurisé de manière pérenne.

Conclusion Générale des Tests

La mise en place du script **AutoTableV2** a transformé la posture de sécurité du serveur :

1. **Avant Iptables :** Le serveur était "naïf", répondant à toutes les sollicitations, exposant ses services (SNMP) et vulnérable au bourrage de mots de passe.
2. **Après Iptables :** Le serveur est "résilient".
   - Il **dialogue** uniquement avec le réseau autorisé.
   - Il **punit** automatiquement les abus (Hard Ban).
   - Il **informe** l'administrateur des tentatives suspectes (Honeypot Logs).

Le filtrage est donc efficace non seulement pour bloquer, mais aussi pour ralentir l'attaquant et générer de la "Threat Intelligence" (renseignement sur la menace) via les logs.

4. Conclusion Générale et Perspectives

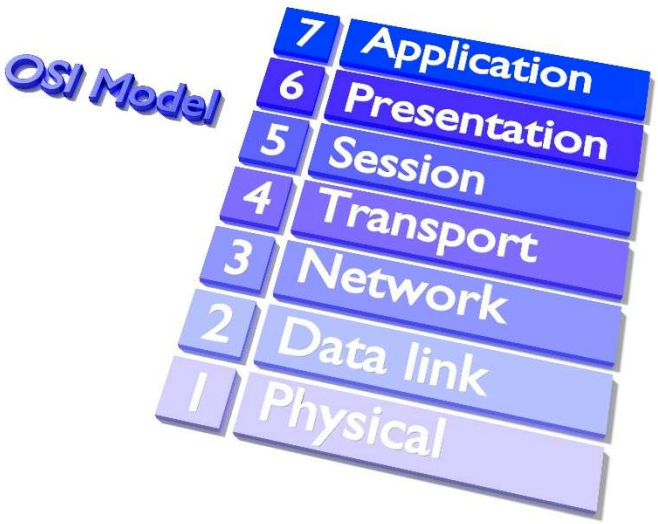
4.1. Bilan du Projet

La mise en œuvre du script **AutoTableV2** a permis d'atteindre les objectifs de sécurisation fixés. Nous avons transformé un serveur Linux par défaut, vulnérable et "bavard", en une forteresse numérique capable de :

1. **Masquer sa présence** (refus des pings et timestamps).
2. **Se défendre activement** (bannissement automatique des attaquants via le module `recent`).
3. **Surveiller son environnement** (génération de logs précis via les chaînes `HONEYPOT`).

Les tests d'intrusion (Nmap, Hydra, Hping3) ont validé l'efficacité technique des règles : le trafic légitime circule, tandis que les tentatives malveillantes sont détectées et bloquées.

4.2. Les Limites Intrinsèques d'Iptables



Malgré l'efficacité démontrée de notre configuration, il est impératif de souligner qu'Iptables (basé sur Netfilter) reste un pare-feu de **couche 3 et 4** (Modèle OSI). Il possède des limitations structurelles qui ne peuvent être ignorées dans un environnement de production critique :

1. Cécité Applicative (Absence de DPI - Deep Packet Inspection) Iptables filtre selon l'adresse IP (Couche 3) et le Port (Couche 4). Il ne "voit" pas le contenu des données.

- *Exemple :* Si nous autorisons le port 80 (HTTP) pour notre serveur web, Iptables laissera passer **tout** le trafic vers ce port. Il ne peut pas distinguer une requête légitime d'une injection SQL ou d'une attaque XSS contenue dans le paquet HTTP.
- *Solution :* Il faudrait ajouter un **WAF (Web Application Firewall)** comme ModSecurity ou un Reverse Proxy.

2. Inefficacité face aux attaques volumétriques (DDoS) Notre script gère bien les attaques DoS simples (via le `rate-limiting`). Cependant, face à une attaque DDoS distribuée (Botnet) de plusieurs Gigabits, Iptables serait submergé. Le lien réseau saturerait avant même que le pare-feu n'ait le temps de traiter et rejeter les paquets.

- *Solution :* Nécessite une protection en amont chez le fournisseur d'accès ou via un CDN (Cloudflare, Akamai).

3. Gestion du Trafic Chiffré Iptables ne peut pas analyser le trafic chiffré (HTTPS/TLS). Si une attaque passe par un tunnel chiffré autorisé, Iptables la laissera passer aveuglément.

4. Complexité de Gestion Bien que notre script `AutoTableV2` soit modulaire, la gestion de milliers de règles Iptables via des scripts Bash devient complexe et sujette à l'erreur humaine à grande échelle.

- *Solution :* Utilisation d'orchestrateurs (Ansible) ou de solutions de pare-feu nouvelle génération (NGFW).

4.3. Perspectives d'Évolution

Pour pallier ces limitations et tendre vers une architecture "Zero Trust", l'évolution logique de ce projet consisterait à :

1. **Coupler Iptables avec Fail2Ban :** Pour automatiser la lecture des logs générés par notre script et bannir les attaquants sur des durées plus longues (persistance).
2. **Installer un IDS/IPS (Snort ou Suricata) :** Pour analyser le contenu des paquets (signature d'attaques) que Iptables laisse passer.
3. **Mettre en place la surveillance IPv6 :** Adapter le script pour `ip6tables`, car un attaquant pourrait contourner nos protections IPv4 en passant par le protocole IPv6 si celui-ci est activé par défaut.

En conclusion, Iptables constitue la **première ligne de défense indispensable** du noyau Linux, mais il doit s'inscrire dans une stratégie de défense en profondeur multicouche.