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, analyse les commandes exécutées et justifie l'ordre séquentiel des opérations.

2.1. Initialisation, Sécurité et Audit (`require_root`, `ensure_log_file`)

Dès le lancement, le script sécurise son environnement d'exécution. Il vérifie les privilèges et met en place un mécanisme de journalisation locale pour que toute l'exécution du script soit enregistrée dans un fichier d'audit.

```
1   #!/bin/bash
2   # Use bash interpreter for script execution.
3
4   # AutoTableV2 - Hardening & firewall tool for the LinuxServer gateway. # Describe tool purpose.
5   # Features include dependency checks, interface configuration, firewall structuring, rsyslog tuning, and auditing. # Summarize capabilities.
6
7   set -euo pipefail # Exit on first error, undefined variable, or failed pipeline.
8   IFS=$'\n\t' # Limit word splitting to newline and tab for safer parsing.
9
10  # ------------------------- GLOBAL CONSTANTS -------------------------
11  readonly EXT_IF="enp0s3" # External-facing interface toward DMZ/router.
12  readonly LAN_IF="enp0s8" # Internal LAN1 interface toward PCs.
13  readonly EXT_IP="10.10.0.10/24" # IP/mask assigned to EXT_IF.
14  readonly LAN_IP="10.10.10.1/24" # IP/mask assigned to LAN_IF.
15  readonly DEFAULT_GW="10.10.0.1" # Default gateway via router R2.
16  readonly ROUTE_LAN2_NET="10.10.20.0/24" # Network for LAN2 behind BSD server.
17  readonly ROUTE_LAN2_GW="10.10.0.20" # Next hop toward LAN2 (BSD server IP).
18  readonly LOG_SERVER="10.10.0.30" # Remote log collector IP.
19  readonly LOG_FILE="/var/log/autotablev2.log" # Local audit log file.
20  readonly REQUIRED_PACKAGES=(iptables iptables-persistent rsyslog iproute2 net-tools) # Dependency list to install.
21  readonly IPTABLES_BIN=$(command -v iptables) # Resolve absolute iptables binary path.
```

```
83  # ------------------------- PRE-FLIGHT CHECKS -------------------------
84  require_root() { # Enforce root execution to manage networking and firewall.
85    if [ "$EUID" -ne 0 ]; then # Compare effective user ID against zero (root).
86      error "Ce script doit être lancé en root." # Display localized error message.
87      exit 1 # Exit immediately without running configuration.
88    fi # End privilege check.
89  }
```

```
57  ensure_log_file() { # Create log file and tee stdout/stderr into it.
58    sudo touch "$LOG_FILE" # Ensure log file exists with elevated privileges.
59    sudo chmod 640 "$LOG_FILE" # Restrict log file permissions to root and group.
60    exec > >(tee -a "$LOG_FILE") 2>&1 # Mirror all subsequent output into log file.
61  }
```

Analyse des commandes :

Commande / Option	Explication Technique
`set -euo pipefail`	Mode strict : arrête le script à la moindre erreur ou variable manquante.
`readonly`	Déclare des constants réseaux immuables pour éviter toute modification accidentelle.
`if [ "$EUID" -ne 0 ]`	Vérifie si l'utilisateur est **Root** (ID 0). Indispensable pour modifier Iptables.
`touch "$LOG_FILE"`	Crée le fichier de log `/var/log/autotablev2.log`.
`chmod 640`	Sécurise le fichier de log (lecture uniquement pour Root et le groupe).

Commande / Option	Explication Technique
`exec > >(tee ...)`	**Redirection Avancée :** Redirige toute la sortie standard (stdout) et d'erreur (stderr) du script à la fois vers l'écran et vers le fichier de log.

2.2. Documentation et Dépendances (`describe_network`, `install_dependencies`)

Le script est auto-documenté : il affiche la topologie réseau au lancement. Ensuite, il s'assure que les outils nécessaires sont présents.

```
63  describe_network() { # Output and log the current lab network architecture.
64    section "Network architecture overview" # Print section heading for clarity.
65    cat <<EOF # Emit descriptive block summarizing topology.
66  Topology Summary:
67    - External network 192.168.100.0/24 hosts router R1 (192.168.100.1) and attacker Kali (192.168.100.10).
68    - Router R2 at 10.10.0.1 interconnects the DMZ 10.10.0.0/24 with upstream cloud and downstream gateways.
69    - LinuxServer (this host) uses $EXT_IF with $EXT_IP toward R2 and $LAN_IF with $LAN_IP toward LAN1.
70    - LAN1 10.10.10.0/24 sits behind LinuxServer acting as gateway for PCs such as PC1 (10.10.10.11) and PC2 (10.10.10.12).
71    - BSDServer at 10.10.0.20 extends connectivity to LAN2 10.10.20.0/24 with gateway 10.10.20.1.
72    - LogServer at $LOG_SERVER collects rsyslog events over UDP/TCP 514 from all infrastructure nodes.
73    - Known attacker source 192.168.100.10 executed reconnaissance, brute force, spoofing, and SYN flood attacks pre-firewall.
74
75  Honeypot Security Model:
76    - Firewall configured with transparent shield/honeypot effect: attacks are logged first, then blocked.
77    - Suspicious traffic (attacker IP, vulnerable ports, spoofing, rate limit violations) sent to HONEYPOT chains.
78    - HONEYPOT chains log detailed attack information (IP options, TCP sequences, UID) before dropping packets.
79    - This allows verification of attack patterns while maintaining security through blocking.
80  EOF
81  }

91  install_dependencies() { # Install or verify required packages.
92    section "Installing/Verifying packages" # Announce dependency phase.
93    export DEBIAN_FRONTEND=noninteractive # Suppress interactive apt prompts.
94    run_cmd "Updating apt cache" apt-get update -qq # Refresh package metadata quietly.
95    run_cmd "Installing required packages: ${REQUIRED_PACKAGES[*]}" apt-get install -y "${REQUIRED_PACKAGES[@]}" # Install dependencies
96  }
```

Analyse des commandes :

Commande	Explication Technique
`cat <<EOF`	**Heredoc :** Affiche un bloc de texte descriptif de l'architecture réseau et du modèle de sécurité (Honeypot) directement dans le terminal.
`export DEBIAN_FRONTEND=noninteractive`	Empêche `apt` de poser des questions (Oui/Non) bloquantes lors de l'installation.
`apt-get install -y`	Installe les paquets requis (`iptables-persistent`, `rsyslog`) sans demande de confirmation.

2.3. Configuration Système et Réseau (`configure_interfaces`, `routes`, `rsyslog`)

Avant d'activer le pare-feu, le script configure la couche réseau (OSI 2 et 3) et le service de logs.

[INSÉRER CAPTURE D'ÉCRAN ICI : Les fonctions "configure_interfaces", "configure_routes" et "configure_rsyslog"]

```
98  # ------------------------- NETWORK CONFIG -------------------------
99  configure_interfaces() { # Configure NIC addresses and enable links.
100   section "Configuring interfaces" # Announce interface configuration stage.
101   run_cmd "Bringing $EXT_IF up with $EXT_IP" ip addr replace "$EXT_IP" dev "$EXT_IF" # Assign EXT interface address.
102   run_cmd "Bringing $LAN_IF up with $LAN_IP" ip addr replace "$LAN_IP" dev "$LAN_IF" # Assign LAN interface address.
103   run_cmd "Enabling interface $EXT_IF" ip link set "$EXT_IF" up # Bring external interface up.
104   run_cmd "Enabling interface $LAN_IF" ip link set "$LAN_IF" up # Bring LAN interface up.
105 }
```

```
configure_routes() { # Configure routing table entries.
  section "Configuring routes" # Announce routing configuration stage.
  run_cmd "Setting default route via $DEFAULT_GW" ip route replace default via "$DEFAULT_GW" dev "$EXT_IF" # Install/replace default route.
  run_cmd "Ensuring route to LAN2 $ROUTE_LAN2_NET via $ROUTE_LAN2_GW" ip route replace "$ROUTE_LAN2_NET" via "$ROUTE_LAN2_GW" dev "$EXT_IF"
}
```

```
# ------------------------- RSYSLOG CONFIG -------------------------
configure_rsyslog() { # Configure rsyslog forwarding to centralized collector.
  section "Ensuring rsyslog remote forwarding" # Announce rsyslog configuration stage.
  local conf_file="/etc/rsyslog.d/99-remote.conf" # Define config snippet path.
  cat <<EOF >"$conf_file" # Write forwarding directives to rsyslog snippet.
# Generated by AutoTableV2
\$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
*.* @$LOG_SERVER:514
*.* @@$LOG_SERVER:514
EOF
  run_cmd "Restarting rsyslog" systemctl restart rsyslog # Apply new rsyslog configuration.
  run_cmd "Enabling rsyslog service" systemctl enable rsyslog # Ensure rsyslog starts on boot.
}
```

Analyse des commandes :

Commande	Explication Technique
`ip addr replace`	Assigne l'adresse IP de manière idempotente (écrase si existe déjà).
`ip link set up`	Active l'interface réseau.
`ip route replace`	Définit la passerelle par défaut.
`systemctl restart rsyslog`	Redémarre le démon de journalisation pour s'assurer qu'il prendra en compte les logs du noyau (Iptables).

2.4. Nettoyage et Structure Honeypot (`flush_tables`, `honeypot_chains`)

C'est la fondation du pare-feu. On supprime les anciennes règles et on prépare les chaînes de journalisation.

```
# ------------------------- IPTABLES HELPERS ------------------------
flush_tables() { # Reset firewall state to a clean slate.
  section "Resetting iptables" # Announce flushing stage.
  for table in filter nat mangle raw; do # Iterate through relevant netfilter tables.
    run_cmd "Flushing table $table" "$IPTABLES_BIN" -w -t "$table" -F # Flush built-in chains in table.
    run_cmd "Deleting custom chains in $table" "$IPTABLES_BIN" -w -t "$table" -X # Delete user-defined chains in table.
  done # End table loop.
  run_cmd "Setting default policy INPUT DROP" "$IPTABLES_BIN" -w -P INPUT DROP # Default-drop inbound traffic.
  run_cmd "Setting default policy FORWARD DROP" "$IPTABLES_BIN" -w -P FORWARD DROP # Default-drop forwarded traffic.
  run_cmd "Setting default policy OUTPUT ACCEPT" "$IPTABLES_BIN" -w -P OUTPUT ACCEPT # Allow local outbound conversations.
}
```

```
honeypot_chains() { # Create honeypot chains for attack detection and logging.
  section "Creating honeypot detection chains" # Announce honeypot setup stage.
  "$IPTABLES_BIN" -w -N HONEYPOT_INPUT # Create dedicated chain for INPUT honeypot logging.
  "$IPTABLES_BIN" -w -N HONEYPOT_FORWARD # Create dedicated chain for FORWARD honeypot logging.
  "$IPTABLES_BIN" -w -A HONEYPOT_INPUT -j LOG --log-prefix "HONEYPOT_ATTACK_INPUT: " --log-level 4 --log-ip-options --log-tcp-sequence
  --log-tcp-options --log-uid # Log attack details with full packet info.
  "$IPTABLES_BIN" -w -A HONEYPOT_INPUT -j DROP # Drop after logging to complete honeypot effect.
  "$IPTABLES_BIN" -w -A HONEYPOT_FORWARD -j LOG --log-prefix "HONEYPOT_ATTACK_FORWARD: " --log-level 4 --log-ip-options
  --log-tcp-sequence --log-tcp-options --log-uid # Log forwarded attack details.
  "$IPTABLES_BIN" -w -A HONEYPOT_FORWARD -j DROP # Drop after logging to complete honeypot effect.
}
```

Analyse des commandes Iptables :

Option	Signification
`-F` / `-X`	**Flush/Delete :** Vide toutes les tables et chaînes pour partir d'un état sain.

Option	Signification
`-P INPUT DROP`	**Politique par Défaut :** Tout ce qui n'est pas autorisé est interdit (Whitelist).
`-N HONEYPOT_INPUT`	**Nouvelle Chaîne :** Crée une "boîte" dédiée pour traiter les paquets suspects.
`--log-prefix`	Ajoute le tag `"HONEYPOT_ATTACK:"` pour faciliter la recherche dans les logs.

2.5. Règles de Base et Hygiène (`base_rules`)

Cette fonction gère le trafic de bas niveau et la protection anti-usurpation (Anti-Spoofing).

```
149  base_rules() { # Apply baseline hygiene and anti-spoofing rules (with honeypot logging).
150    section "Base traffic hygiene" # Announce baseline rule stage.
151    "$IPTABLES_BIN" -w -A INPUT -i lo -j ACCEPT # Permit loopback traffic inbound.
152    "$IPTABLES_BIN" -w -A OUTPUT -o lo -j ACCEPT # Permit loopback traffic outbound.
153    "$IPTABLES_BIN" -w -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT # Allow established inbound responses.
154    "$IPTABLES_BIN" -w -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT # Allow established forwarded flows.
155    "$IPTABLES_BIN" -w -A INPUT -m conntrack --ctstate INVALID -j HONEYPOT_INPUT # Send invalid packets to honeypot for logging before drop.
156    "$IPTABLES_BIN" -w -A FORWARD -m conntrack --ctstate INVALID -j HONEYPOT_FORWARD # Send invalid forwarded packets to honeypot.
157    "$IPTABLES_BIN" -w -A INPUT -i "$LAN_IF" -s 10.10.0.0/24 -j HONEYPOT_INPUT # Log DMZ spoofing attempts from LAN side.
158    "$IPTABLES_BIN" -w -A FORWARD -i "$LAN_IF" -s 10.10.0.0/24 -j HONEYPOT_FORWARD # Log spoofed DMZ traffic traversing forward path.
159    "$IPTABLES_BIN" -w -A INPUT -i "$EXT_IF" -s 10.10.10.0/24 -j HONEYPOT_INPUT # Log LAN1 spoofing attempts entering external interface.
160    # Block RFC1918 private addresses on external interface (spoofing detection), but exclude legitimate external network 192.168.100.0/24.
161    for net in 10.0.0.0/8 172.16.0.0/12; do # Loop through RFC1918 ranges (excluding 192.168.0.0/16 to allow external network).
162      "$IPTABLES_BIN" -w -A INPUT -i "$EXT_IF" -s "$net" -j HONEYPOT_INPUT # Log private source addresses arriving on external interface (spoof
163    done # End anti-spoof loop.
164    # Note: 192.168.0.0/16 is intentionally NOT blocked in base_rules to allow external network 192.168.100.0/24 traffic.
165    # This enables honeypot/rate limiting detection. Spoofing from other 192.168.x.x ranges will be caught by service/advanced rules if needed.
166  }
```

Analyse des commandes :

Commande	Explication Technique
`-m conntrack --ctstate ESTABLISHED`	Autorise le trafic des connexions déjà acceptées (Optimisation).
`--ctstate INVALID`	Bloque les paquets techniques incorrects (Scan NULL/XMAS).
`-N CHECK_SPOOFING`	Chaîne personnalisée vérifiant si l'IP source appartient bien au réseau local légitime.

2.6. Sécurité Avancée : Hard Ban Unifié (`advanced_security`)

Le cœur du système de défense active. Si un attaquant force le SSH, il est banni de tous les services (FTP, Telnet) pour 60 secondes.

```
advanced_security() { # Apply rate-limiting and SYN flood controls with honeypot logging.
  section "Advanced protections" # Announce advanced security stage.
  # Unified hard-ban using iptables 'recent' for FTP(21), SSH(22), Telnet(23)
  local PROTECTED_PORTS="21,22,23"

  # 1) Already banned → drop and refresh timer
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" \
    -m recent --name ABUSE_BANNED --update --seconds 60 -j DROP

  # 2) Track all NEW attempts to protected ports
  "$IPTABLES_BIN" -w -A INPUT -p tcp -m multiport --dports "$PROTECTED_PORTS" \
    -m conntrack --ctstate NEW -m recent --name ABUSE_COUNT --set

  # 3) Log per-service when threshold exceeded (4 hits in 60s)
  "$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

  "$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

  "$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
```

```
advanced_security() { # Apply rate-limiting and SYN flood controls with honeypot logging.
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 23 \
  # 4) On threshold → add to ban list and drop
  "$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 SSH from the known attacker IP after ban checks (observability allowed, abuse gets banned)
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 22 -s 192.168.100.10 -j ACCEPT

  # 6) For all other SSH sources not DMZ or attacker → honeypot (DMZ SSH allowed later in service_rules)
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 22 ! -s 10.10.0.0/24 ! -s 192.168.100.10 -m conntrack --ctstate NEW -j HONEYPOT_INPUT
```

```
advanced_security() { # Apply rate-limiting and SYN flood controls with honeypot logging.
  # 6) For all other SSH sources not DMZ or attacker → honeypot (DMZ SSH allowed later in service_rules)
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 22 ! -s 10.10.0.0/24 ! -s 192.168.100.10 -m conntrack --ctstate NEW -j HONEYPOT_INPUT

  # 7) FTP/Telnet attempts (21,23) → honeypot (no open access), after ban checks
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 21 -j HONEYPOT_INPUT
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 23 -j HONEYPOT_INPUT

  # 8) Block attacker IP for all other services (non-SSH) - honeypot effect (preserve V2 behavior)
  "$IPTABLES_BIN" -w -A INPUT -s 192.168.100.10 ! -p tcp --dport 22 -j HONEYPOT_INPUT
  "$IPTABLES_BIN" -w -A FORWARD -s 192.168.100.10 -j HONEYPOT_FORWARD

  # 9) SYN flood protection (unchanged)
  "$IPTABLES_BIN" -w -N SYN_FLOOD_CHECK # Create dedicated chain for SYN flood detection.
  "$IPTABLES_BIN" -w -A INPUT -p tcp --syn -j SYN_FLOOD_CHECK # Send all SYN packets through protection chain.
  "$IPTABLES_BIN" -w -A SYN_FLOOD_CHECK -m limit --limit 1/s --limit-burst 4 -j RETURN # Allow limited burst of SYNs (legitimate traffic).
  "$IPTABLES_BIN" -w -A SYN_FLOOD_CHECK -j HONEYPOT_INPUT # Send excess SYN traffic to honeypot for logging (DoS attack detection).
}
```

Analyse des commandes (Modules Recent & Multiport) :

Option	Rôle
`-m multiport --dports 21,22,23`	Applique la règle simultanément aux ports FTP, SSH et Telnet.
`-m recent --update --seconds 60`	Vérifie si l'IP est déjà bannie ("Prison"). Si oui, prolonge la peine.
`--hitcount 4`	Déclenche le bannissement après 3 tentatives ratées.
`--set`	Ajoute l'IP coupable à la liste noire dynamique `ABUSE_BANNED`.

2.7. Services et Routage (`service_rules`, `forwarding_rules`)

Si le paquet n'est pas malveillant, on vérifie s'il est autorisé à accéder à un service ou à traverser le routeur.

```
service_rules() { # Define application-layer access policies with honeypot logging.
  section "Service accessibility rules" # Announce service control stage.
  "$IPTABLES_BIN" -w -A INPUT -p icmp --icmp-type echo-request -s 10.10.0.0/24 -j ACCEPT # Allow DMZ hosts to ping gateway.
  "$IPTABLES_BIN" -w -A INPUT -p icmp --icmp-type echo-request -s 10.10.10.0/24 -j ACCEPT # Allow LAN1 hosts to ping gateway.
  "$IPTABLES_BIN" -w -A INPUT -p icmp --icmp-type timestamp-request -j HONEYPOT_INPUT # Log ICMP timestamp requests (reconnaissance atte
  "$IPTABLES_BIN" -w -A INPUT -p icmp --icmp-type timestamp-reply -j HONEYPOT_INPUT # Log outbound timestamp replies (reconnaissance att
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 22 -s 10.10.0.0/24 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT # Allow SSH from DMZ ne
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 80 -j ACCEPT # Allow HTTP service for web testing.
  "$IPTABLES_BIN" -w -A INPUT -p tcp --dport 443 -j ACCEPT # Allow HTTPS service for secure web testing.
  "$IPTABLES_BIN" -w -A INPUT -p udp --dport 161 -j HONEYPOT_INPUT # Log SNMP access attempts (information disclosure risk).
  # Note: Attacker IP (192.168.100.10) handling moved to advanced_security to allow SSH rate limiting before blocking.
}

forwarding_rules() { # Permit legitimate routed paths between segments with honeypot logging.
  section "Forwarding & routing policies" # Announce forwarding policy stage.
  "$IPTABLES_BIN" -w -A FORWARD -i "$LAN_IF" -o "$EXT_IF" -s 10.10.10.0/24 -d 10.10.0.0/24 -j ACCEPT # Allow LAN1 traffic toward DMZ ser
  "$IPTABLES_BIN" -w -A FORWARD -i "$EXT_IF" -o "$LAN_IF" -s 10.10.0.0/24 -d 10.10.10.0/24 -j ACCEPT # Allow DMZ servers to reply/initia
  "$IPTABLES_BIN" -w -A FORWARD -i "$LAN_IF" -o "$EXT_IF" -s 10.10.10.0/24 -d 192.168.100.0/24 -j ACCEPT # Allow LAN1 to reach external
  # Note: Suspicious forwarding attempts (attacker IP, spoofing) are already caught by base_rules and sent to HONEYPOT_FORWARD.
}
```

Analyse des commandes :

Commande	Explication Technique
`-p tcp --dport 80`	Autorise le trafic Web (HTTP).
`icmp --icmp-type echo-request`	Autorise le Ping uniquement (pas les timestamps).
`-A FORWARD`	**Routage :** Autorise le trafic à traverser le serveur (du LAN vers le WAN) si le serveur agit comme routeur.

2.8. Journalisation Finale et Persistance (`drop_logging`, `persist_rules`)

Le filet de sécurité final et la sauvegarde.

```
drop_logging() { # Add catch-all logging for unmatched packets (honeypot chains handle most attacks).
  section "Catch-all drop logging" # Announce catch-all logging configuration stage.
  "$IPTABLES_BIN" -w -A INPUT -j LOG --log-prefix "IPTABLES_INPUT_DROP: " --log-level 6 # Log unmatched INPUT packets (should be rare with h
  "$IPTABLES_BIN" -w -A FORWARD -j LOG --log-prefix "IPTABLES_FORWARD_DROP: " --log-level 6 # Log unmatched FORWARD packets (should be rare
}

persist_rules() { # Save firewall state for persistence across reboots.
  section "Persisting firewall state" # Announce persistence stage.
  run_cmd "Saving active rules to /etc/iptables/rules.v4" /sbin/iptables-save >/etc/iptables/rules.v4 # Dump ruleset to persistence file.
  run_cmd "Enabling netfilter-persistent" systemctl enable netfilter-persistent # Enable netfilter persistence service.
  run_cmd "Saving via netfilter-persistent" netfilter-persistent save # Trigger persistence save action.
}
```

Analyse des commandes :

Commande	Explication Technique
`-A INPUT -j LOG`	**Log Catch-all :** Tout paquet qui n'a pas été accepté ou traité par le Honeypot arrive ici. Il est loggué avant d'être tué par la politique par défaut.
`netfilter-persistent save`	Sauvegarde les règles actives pour qu'elles soient rechargées au prochain démarrage.

2.9. 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).

```
258  # ------------------------- MAIN -----------------------------------
259  main() { # Primary orchestration function.
260    require_root # Verify script is executed as root.
261    ensure_log_file # Start logging early for full audit trail.
262    describe_network # Document network architecture in logs/output.
263    install_dependencies # Ensure required packages are installed.
264    configure_interfaces # Configure interface IP settings.
265    configure_routes # Program routing table entries.
266    configure_rsyslog # Configure rsyslog forwarding behavior.
267    flush_tables # Reset firewall state to a clean baseline.
268    honeypot_chains # Create honeypot chains for attack detection and logging (must be before rules that use them).
269    base_rules # Apply baseline hygiene policies (uses honeypot chains for suspicious traffic).
270    advanced_security # Enable advanced protections FIRST (rate limiting, attacker IP handling) - must be before service_rules.
271    service_rules # Configure service-specific rules (uses honeypot chains for attack logging) - DMZ SSH allowed after rate limiting.
272    forwarding_rules # Configure routing/forwarding policies.
273    drop_logging # Enable catch-all drop logging for any unmatched traffic.
274    persist_rules # Persist firewall configuration across reboots.
275    section "Completed. Firewall & logging hardened with honeypot attack detection." # Print completion message.
276  }
277
278  main "$@" # Execute main function with provided CLI arguments.
```

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`).

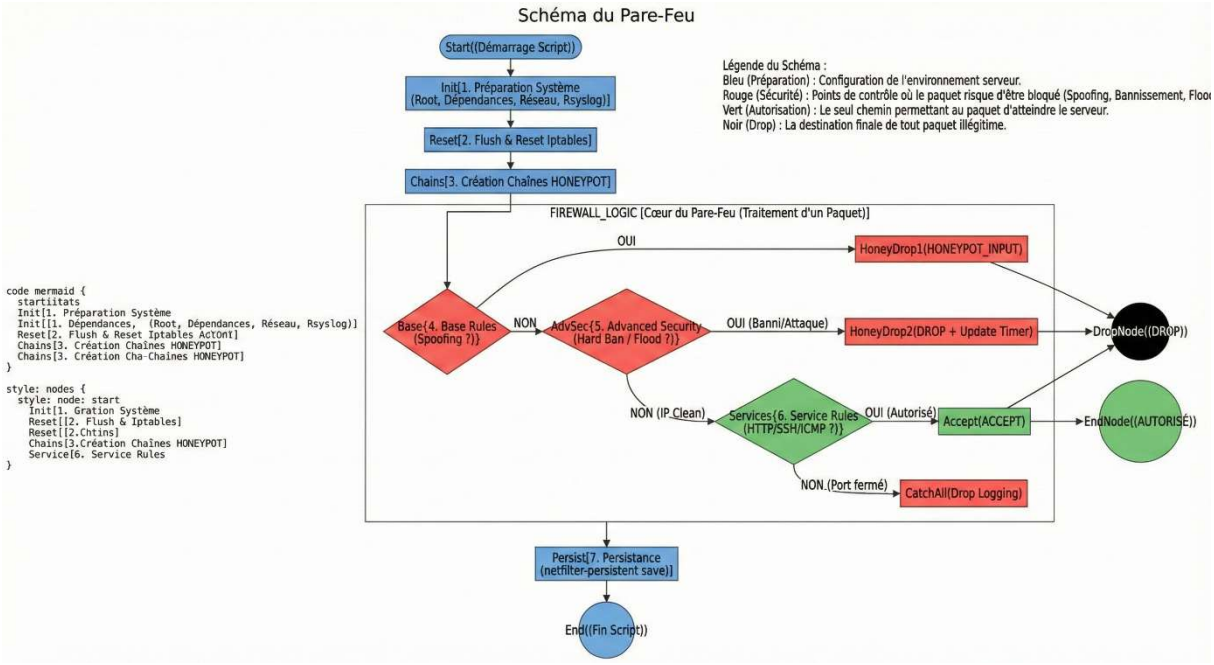
- ⇨ **Phase de Construction (Structure) :**

`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

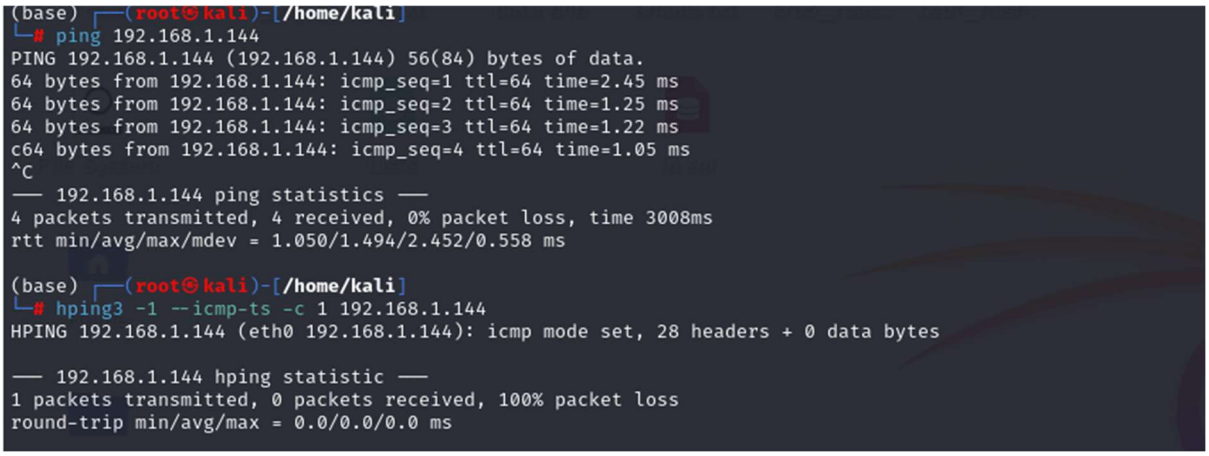
Contexte du test :

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.



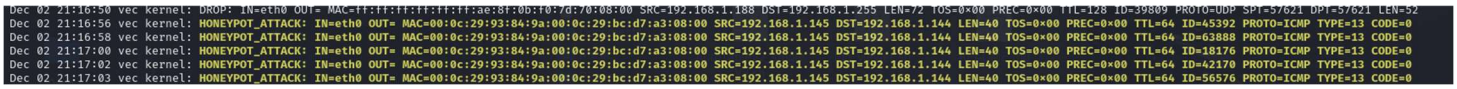
A. Ping Classique (ICMP Echo Request)

- **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)

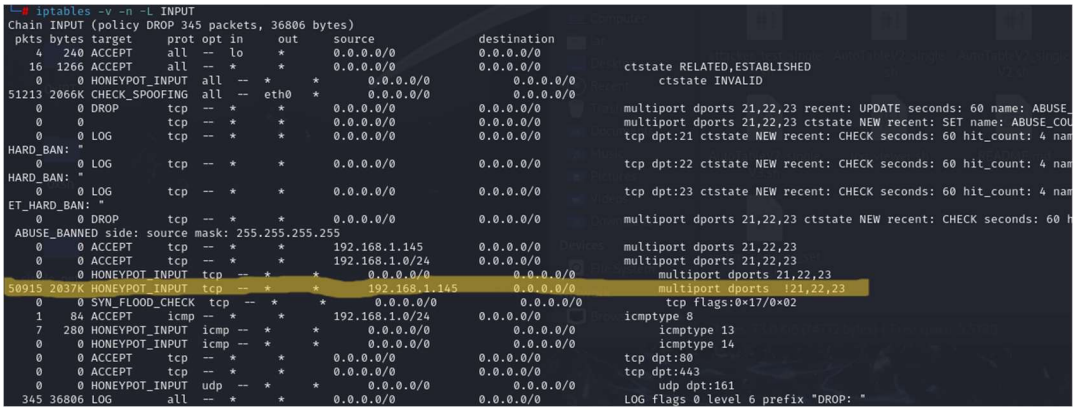
Nous avons lancé un scan pour voir la différence entre les ports protégés et les ports ouverts.

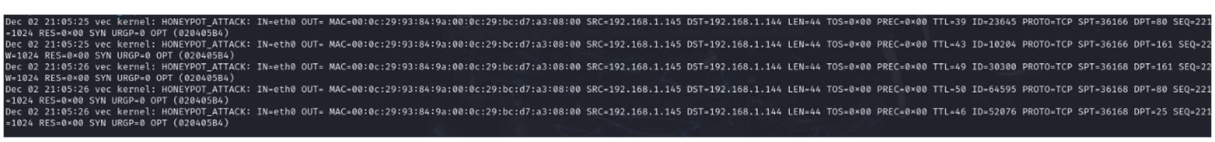
- **Commande :** `nmap -p 22,25,80,161 192.168.1.144`

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.

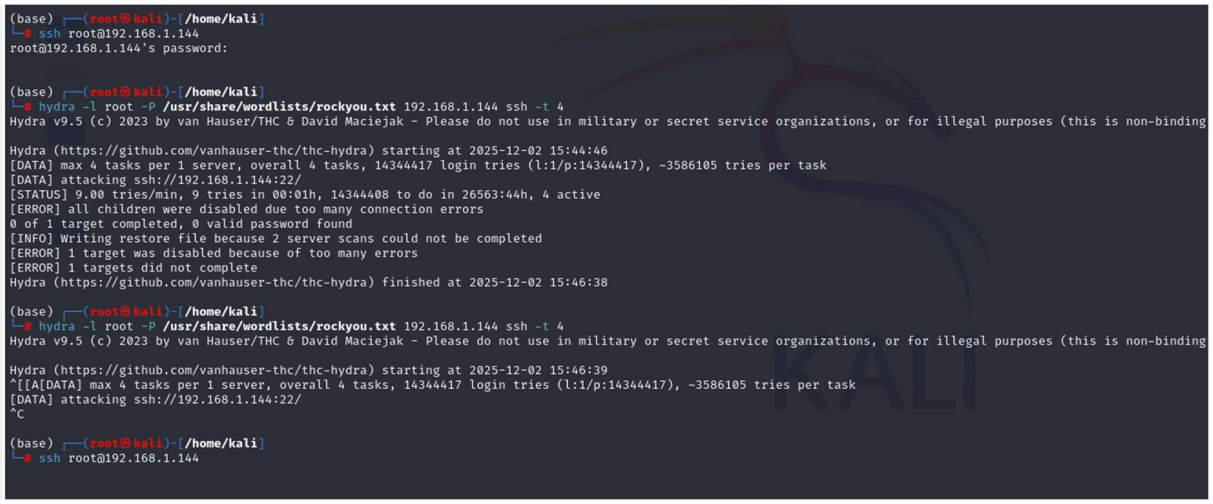
Efficacité du "Honeypot" : Chaque tentative de scan sur le port 25 ou 161 a déclenché une alerte dans les logs. Au lieu de simplement ignorer le scan, le pare-feu a enregistré l'activité suspecte via la chaîne HONEYPOT_INPUT.
Conclusion : Le pare-feu agit comme un filtre sélectif. Il ne masque pas totalement 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.

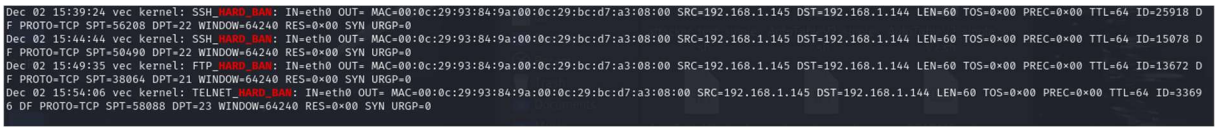


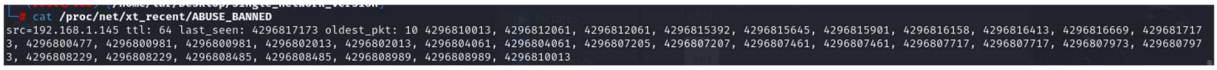
- **Attaque :** `hydra -l root -p password 192.168.1.144 ssh -t 4`
- **Observation Temps Réel :**
  1. Les 3 premières tentatives échouent normalement (mauvais mot de passe).
  2. À la 4ème tentative, la connexion se fige (Time out).
  3. Une tentative de connexion manuelle (`ssh root@...`) échoue immédiatement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

```
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.

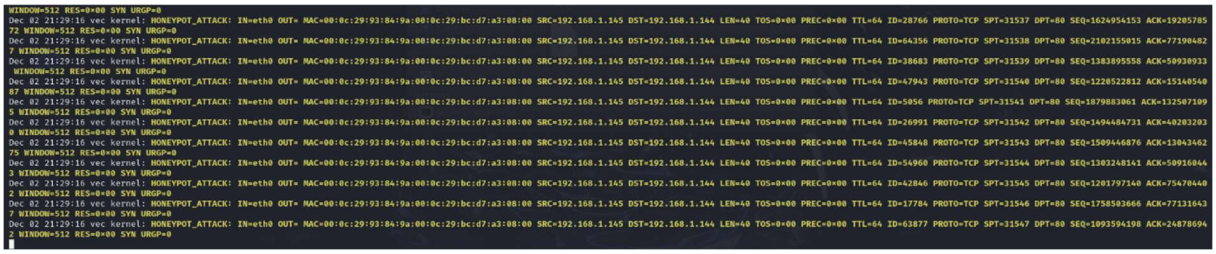
- **Commande :** `hping3 -S --flood -p 80 192.168.1.144`

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

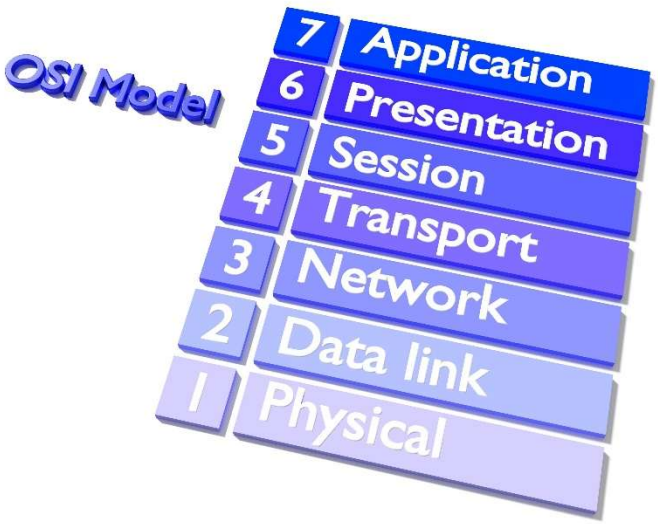
4.1. Bilan du Projet

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.