Sécurisation de l’accès TVbox/Linux Node
SSH-UFW-Fail2ban

Contexte et objectifs

Une fois le système Armbian fonctionnel et accessible à distance, la sécurisation de l’accès SSH devient une étape indispensable.
Par défaut, l’image Armbian autorise une connexion SSH directe en tant que `root` avec un mot de passe initial connu, ce qui constitue un risque évident dans un contexte réseau, même restreint à un homelab.

L’objectif de cette phase est donc de :

- supprimer tout accès SSH direct au compte `root`,
- remplacer l’authentification par mot de passe par une authentification par clé SSH,
- utiliser un utilisateur non privilégié (`dome`) pour l’administration, via `sudo`.

Cette approche est conforme aux bonnes pratiques classiques en administration système et permet de réduire significativement la surface d’attaque.

Création et préparation de l’utilisateur

Un utilisateur dédié à l’administration du système est utilisé : `dome`.

Si celui-ci n’existe pas encore, il est créé et ajouté au groupe `sudo` afin de permettre l’élévation de privilèges contrôlée :

```
adduser dome
usermod -aG sudo dome
```

L’administration du système s’effectuera désormais exclusivement via cet utilisateur.

Mise en place de l’authentification par clé SSH

La génération de la clé SSH est effectuée sur la machine cliente (poste d’administration), et non directement sur la box, conformément aux bonnes pratiques.

La commande suivante est utilisée :

```
ssh-keygen -t ed25519 -a 64 -C "dome@mxq-pro"
```

Le choix de l'algorithme **Ed25519** est motivé par :

- sa robustesse cryptographique,
- sa rapidité,
- sa compatibilité native avec OpenSSH moderne.

La clé publique générée est ensuite copiée sur la box dans le compte `dome`, soit via `ssh-copy-id`, soit manuellement dans le fichier :

```
/home/dome/.ssh/authorized_keys
```

Les permissions sont vérifiées afin d'éviter tout refus de connexion par OpenSSH :

```
chmod 700 /home/dome/.ssh
chmod 600 /home/dome/.ssh/authorized_keys
chown -R dome:dome /home/dome/.ssh
```

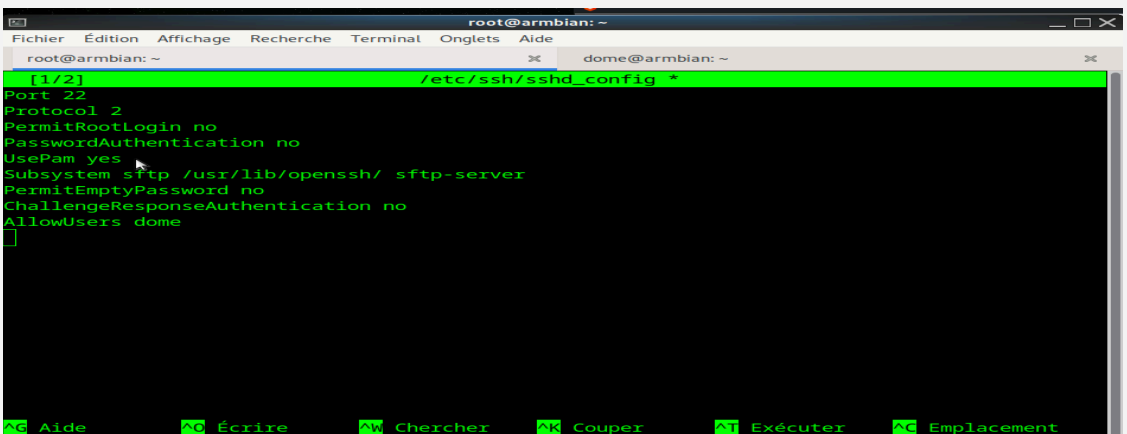
Désactivation de l'accès SSH root et des mots de passe

La configuration du service SSH est ensuite durcie via l'édition du fichier :

```
/etc/ssh/sshd_config
```

Les directives suivantes sont explicitement définies :

```
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
PermitEmptyPasswords no
ChallengeResponseAuthentication no
UsePAM yes
```



Optionnellement, l’accès SSH peut être limité à un utilisateur précis :

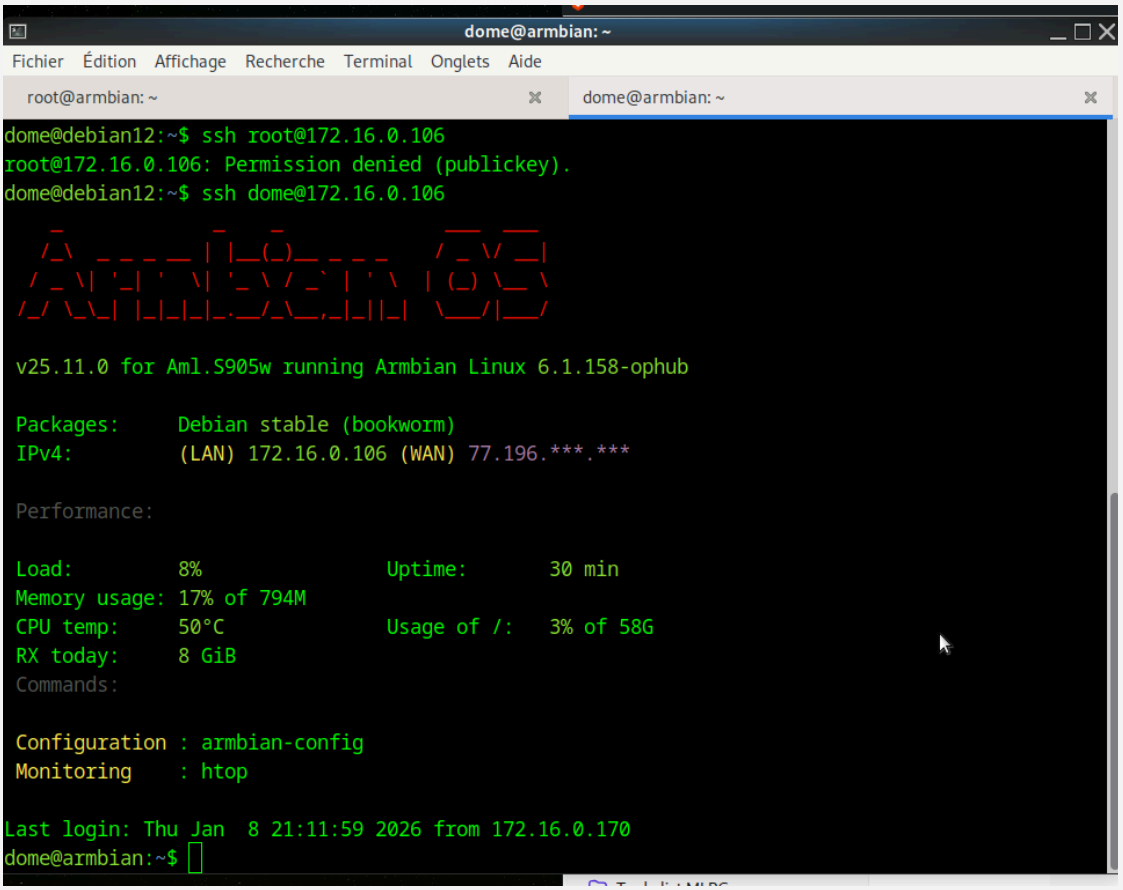
```
AllowUsers dome
```

Avant redémarrage du service, la configuration est validée afin d’éviter toute erreur bloquante :

```
sshd -t
```

Puis le service est rechargé :

```
systemctl restart ssh
```



Validation et tests

La validation s’effectue en plusieurs étapes :

- connexion SSH réussie avec l’utilisateur dome via clé,
- refus explicite de toute tentative de connexion SSH en tant que root,
- absence de demande de mot de passe lors de l’authentification.

Cette séquence garantit que l’accès distant repose exclusivement sur des clés cryptographiques et un compte non privilégié.

Le système dispose désormais d'un accès distant **fonctionnel, minimal et correctement cloisonné**. En d'autres termes : le SSH est passé du mode "portes ouvertes, bienvenue" à "badge, badge encore, et on se parle" — exactement ce qu'on attend d'un nœud réseau un minimum sérieux.

5. Renforcement de la sécurité du système (mesures complémentaires)

Une fois l'accès SSH sécurisé par clé et l'authentification root désactivée, il est recommandé de mettre en place des mesures supplémentaires afin de renforcer la posture de sécurité globale du système.
Ces actions visent à limiter l'exposition réseau, à réduire l'impact des tentatives d'attaque automatisées et à encadrer strictement les privilèges administrateur.

5.1 Mise en place d'un pare-feu simple avec UFW

Contexte

Par défaut, un système Linux fraîchement installé accepte les connexions entrantes sur tous les ports ouverts par les services actifs.
Même dans un environnement de type homelab, cette configuration est considérée comme trop permissive.

Le pare-feu **UFW (Uncomplicated Firewall)** est retenu pour ce projet en raison de :

- sa simplicité de configuration,
- son intégration native avec iptables,
- son adéquation avec un nœud réseau léger et headless.

Installation et configuration

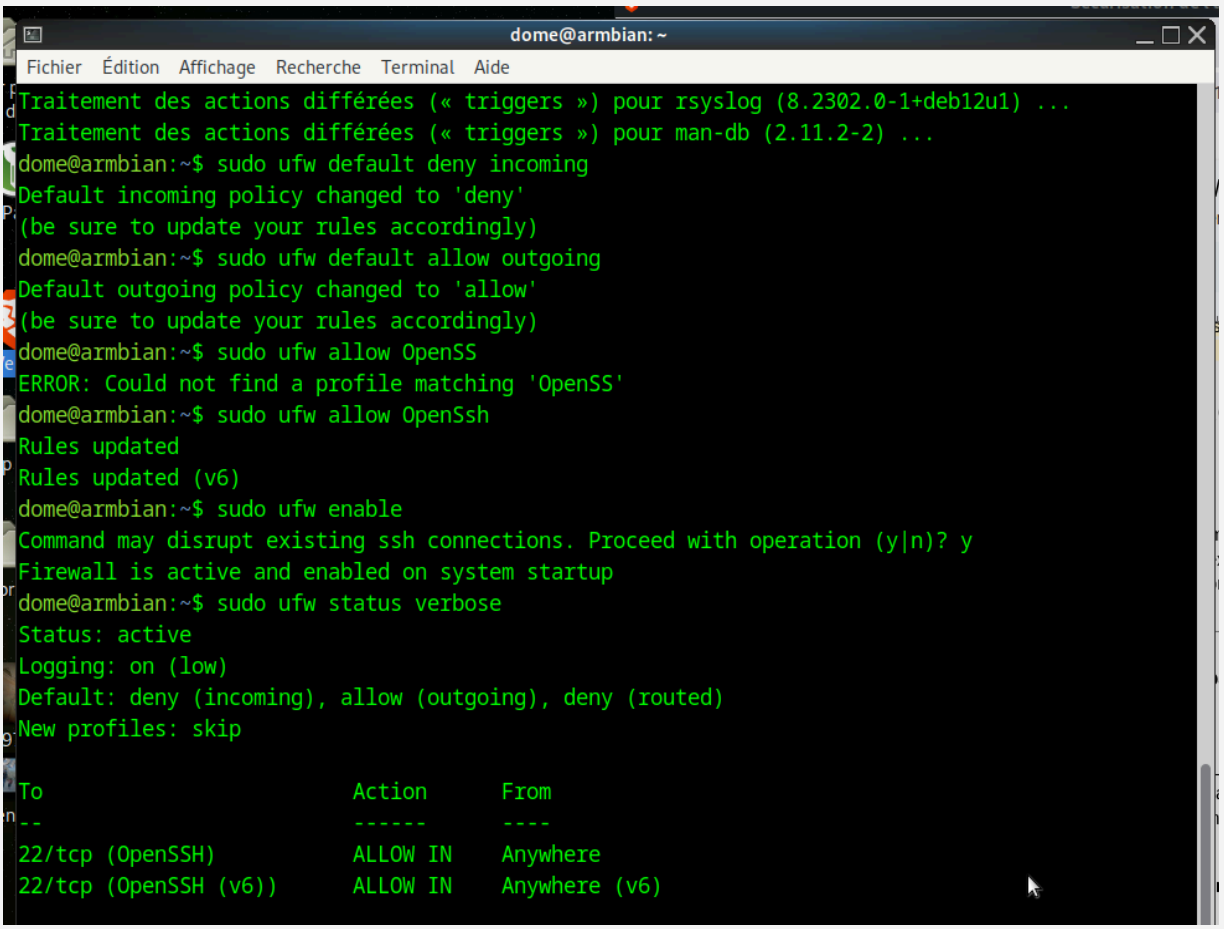
Le pare-feu est installé puis configuré avec une politique restrictive par défaut :

```
sudo apt install -y ufw
```

Définition des règles de base :

```
sudo ufw default deny incoming
```

```
sudo ufw default allow outgoing
```



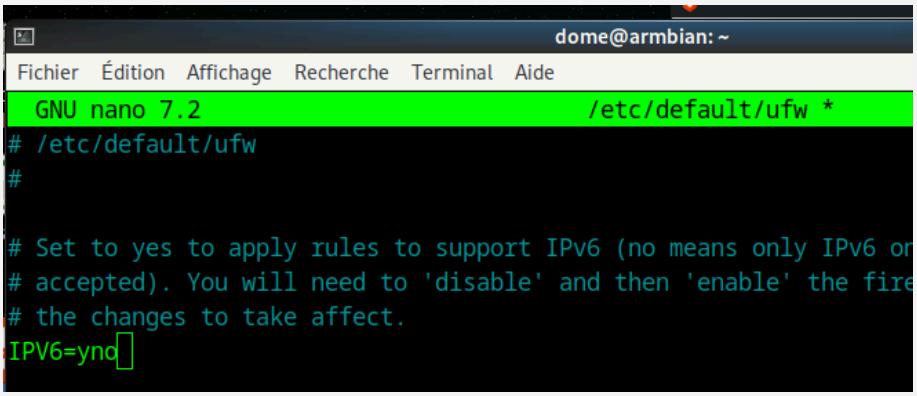
Modification de la configuration UFW

Le comportement IPv6 de UFW est contrôlé via le fichier de configuration principal :

```
sudo nano /etc/default/ufw
```

La directive suivante est modifiée :

```
IPV6=no
```



Par défaut, cette valeur est généralement définie à `yes`.

Autorisation explicite du service SSH (port 22 par défaut) :

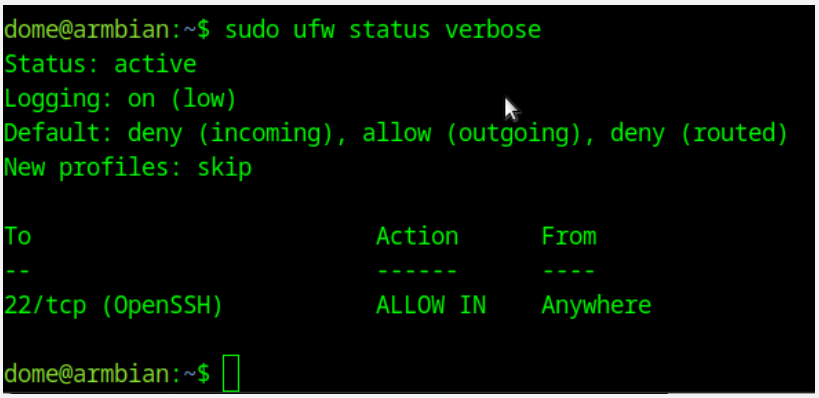
```
sudo ufw allow OpenSSH
```

Activation du pare-feu :

```
sudo ufw enable
```

Vérification de l'état et des règles appliquées :

```
sudo ufw status verbose
```



Résultat attendu

- Toute connexion entrante non explicitement autorisée est bloquée.
- L'accès SSH reste fonctionnel.
- Le système adopte une posture *deny by default*, conforme aux bonnes pratiques de sécurité.

5.2 Protection contre les attaques par force brute avec Fail2ban

Contexte

Même avec une authentification par clé, un service SSH exposé sur un réseau peut faire l'objet de tentatives répétées de connexion automatisées.
Bien que ces tentatives soient inefficaces, elles génèrent du bruit inutile dans les journaux et consomment des ressources.

Fail2ban permet de surveiller les logs système et de bannir temporairement les adresses IP présentant un comportement suspect.

Installation et activation

```
sudo apt install -y fail2ban
```

```
sudo systemctl enable --now fail2ban
```

```
dome@armbian:~$ sudo systemctl enable --now fail2ban
Synchronizing state of fail2ban.service with SysV service script with /lib/systemd/systemd-sysv
-install.
Executing: /lib/systemd/systemd-sysv-install enable fail2ban
dome@armbian:~$ sudo fail2ban-client status
Status
|- Number of jail:      1
`- Jail list:   sshd
```

Dans sa configuration par défaut, Fail2ban :

- surveille les tentatives d'authentification SSH,
- bannit automatiquement les adresses IP après plusieurs échecs consécutifs,
- applique un bannissement temporaire via iptables.

Cette configuration est jugée suffisante pour les besoins du projet.

Vérification du fonctionnement

```
sudo fail2ban-client status
```

```
sudo fail2ban-client status sshd
```

```
dome@armbian:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- File list:        /var/log/auth.log
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
dome@armbian:~$
```

Résultat attendu

- Réduction drastique des tentatives répétées depuis une même adresse IP.
- Journaux système plus lisibles.
- Protection passive efficace, sans impact sur l'usage normal.

Notifications Fail2ban lors du bannissement d'une adresse IP

Principe général

Fail2ban repose sur un mécanisme d'actions.
Lorsqu'une adresse IP est bannie, une ou plusieurs actions peuvent être déclenchées simultanément, par exemple :

- ajout d'une règle firewall,
- écriture dans les logs,
- envoi d'une notification (email, script personnalisé, webhook).

Par défaut, sur Debian / Armbian, Fail2ban **bannit sans notifier**.
L'envoi de notifications doit donc être explicitement configuré.

Méthode 1 – Notification par email (solution native et robuste)

Pré-requis

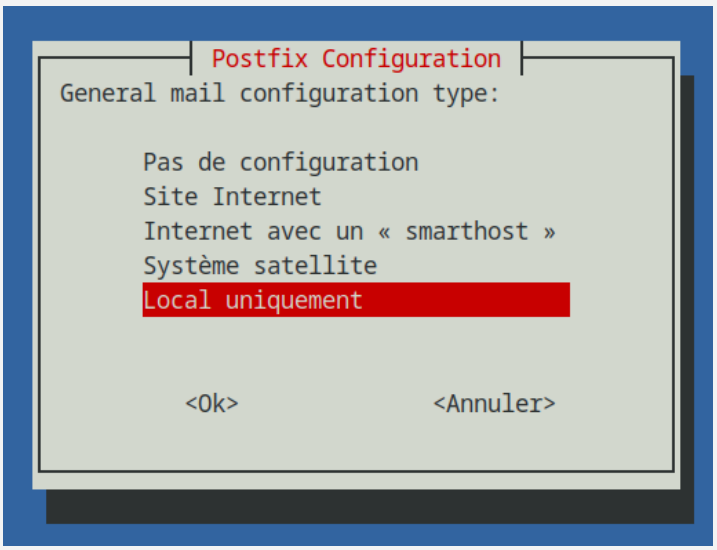
- un MTA léger installé (ex. `postfix` en mode local-only ou `ssmtp`),
- une adresse email de destination.

Installation d'un MTA simple (exemple avec Postfix)

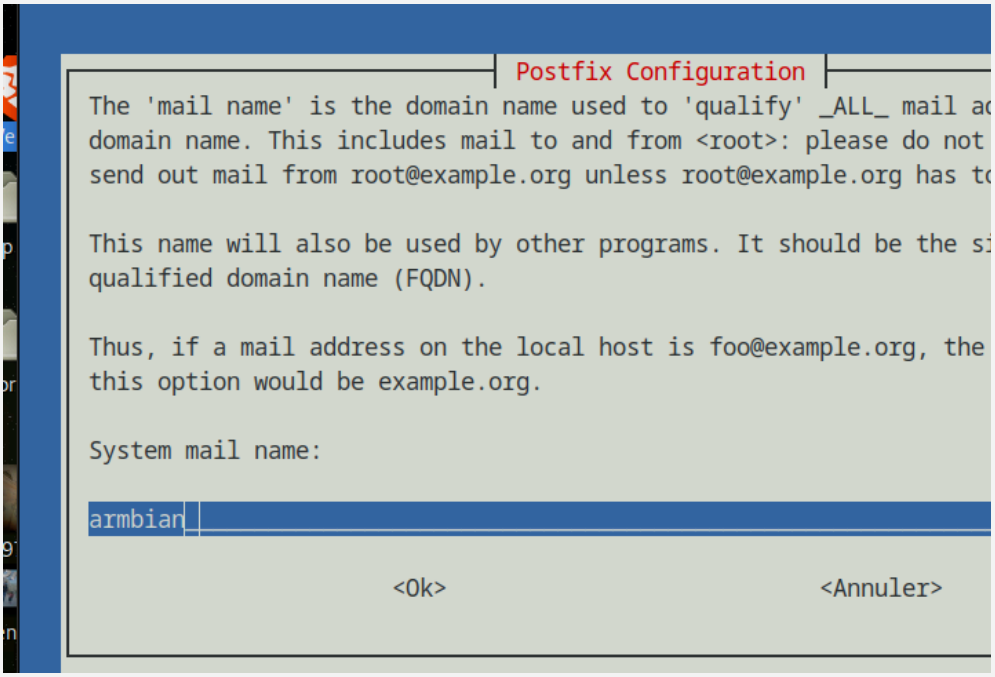
```
sudo apt install -y postfix mailutils
```

Lors de l'installation :

- choisir **"Local only"**,



- utiliser le nom d'hôte de la machine.



Configuration Fail2ban

La configuration se fait **dans un fichier override**, jamais dans les fichiers par défaut.

Créer ou éditer :

```
sudo nano /etc/fail2ban/jail.local
```

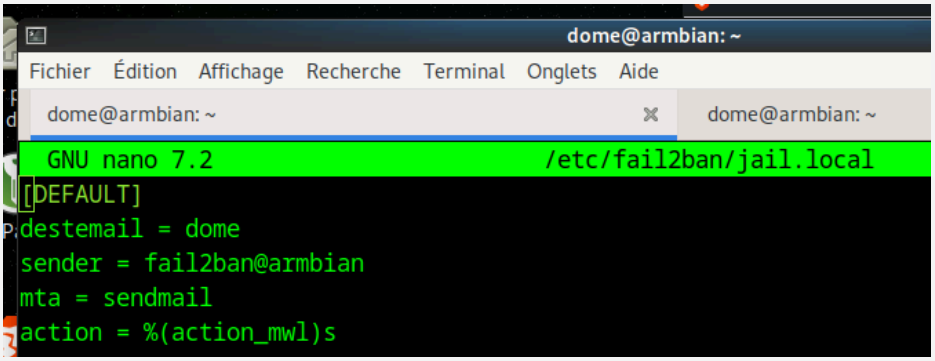
Ajouter :

```
[DEFAULT]

destemail = admin@exemple.local

sender = fail2ban@armbian

mta = sendmail

action = %(action_mwl)s
```

Explication de l'action utilisée :

- `action_mwl` :
  - **m** → mail
  - **w** → whois (infos sur l'IP)
  - **l** → logs pertinents

C'est l'option la plus complète pour un usage d'administration.



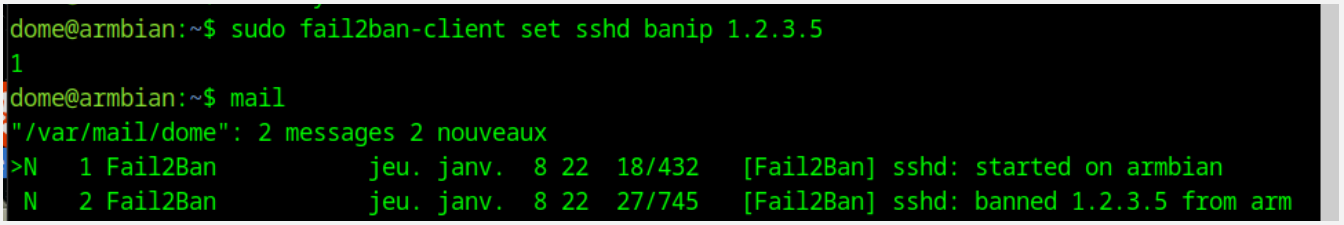
Application de la configuration

```
sudo systemctl restart fail2ban
```

Test de fonctionnement

Simulation d'un bannissement (à des fins de test uniquement) :

```
sudo fail2ban-client set sshd banip 1.2.3.5
```



Si la configuration est correcte :

- l'IP est bannie,
- un email de notification est envoyé.

Méthode 2 – Script personnalisé (Slack, Discord, webhook, etc.)

Principe

Fail2ban permet d'exécuter un script à chaque bannissement via une **action personnalisée**.
Cette approche est particulièrement adaptée à :

- Slack
- Discord
- Gotify
- Grafana OnCall
- tout système exposant un webhook HTTP

Exemple conceptuel

1. Création d'un script :

```
/usr/local/bin/fail2ban-notify.sh
```

2. Le script reçoit automatiquement des variables telles que :

- IP bannie
- jail concerné
- durée du bannissement
- hôte

3. Le script envoie une requête HTTP ou un message formaté.

Fail2ban peut alors utiliser une action combinée :

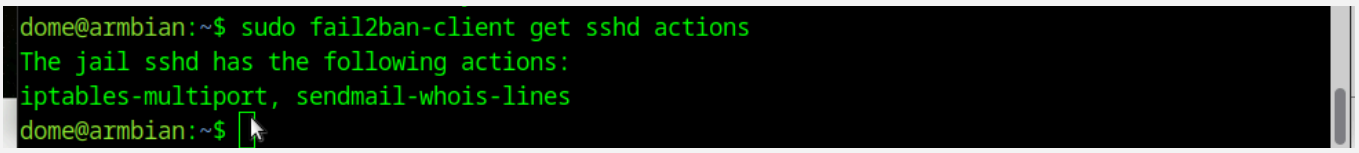
```
action = iptables[name=SSH, port=ssh, protocol=tcp]
         notify-script
```

Cette méthode dépasse volontairement le cadre minimal du projet, mais elle est parfaitement compatible avec une montée en gamme ultérieure.

Vérification des notifications

Pour lister les actions actives sur un jail :

```
sudo fail2ban-client get sshd actions
```



Pour consulter les logs Fail2ban :

```
sudo journalctl -u fail2ban
```

Bilan

Fail2ban :

- **sait notifier nativement** via email,
- **s'intègre facilement** à des systèmes modernes via scripts,
- permet une visibilité immédiate sur les tentatives d'intrusion.

Dans le cadre de ce projet, l'email constitue une solution **simple, fiable et suffisante**.
Les notifications temps réel (webhooks) pourront être envisagées ultérieurement lors de l'intégration VPN / supervision.

En résumé :
Fail2ban ne se contente pas de bloquer — il **prévient**, et c'est souvent là que commence une vraie hygiène de sécurité.

5.3 Gestion propre des privilèges administrateur (sudo)

Principe

Dans le cadre de cette configuration, le compte `root` n'est plus utilisé pour les connexions distantes. L'administration du système repose exclusivement sur l'utilisateur `dome`, avec élévation de privilèges ponctuelle via `sudo`.

Cette approche permet :

- une traçabilité claire des commandes sensibles,
- une réduction des erreurs destructrices involontaires,
- un cloisonnement net entre usage courant et administration.

Bonnes pratiques appliquées

- Pas de connexion SSH directe en tant que `root`.
- Utilisation de `sudo` uniquement lorsque nécessaire.
- Session utilisateur standard par défaut.

Exemple :

```
sudo apt update

sudo systemctl restart ssh
```

Remarque liée au contexte Armbian / TV Box

Dans le cadre de ce projet, un incident antérieur avait empêché le démarrage du service SSH en raison de l'absence de clés hôte (`no hostkeys available`).
La résolution avait nécessité la génération manuelle des clés système via :

```
ssh-keygen -A
```

Cette étape reste à garder en tête lors de toute reconfiguration SSH sur des images Armbian destinées aux TV box Amlogic, comme documenté précédemment

Android TV Box to Linux Node

Modification du port SSH

Objectif

Réduire significativement le bruit lié aux scans automatisés ciblant le port SSH par défaut (22), sans modifier le modèle de sécurité existant.

Cette mesure :

- n'a pas vocation à renforcer l'authentification,
- complète les protections déjà en place (clé SSH, Fail2ban, UFW),
- améliore la lisibilité des journaux et la pertinence des alertes.

Ex. Choix du port

Un port non standard, élevé et non utilisé est retenu.

Exemple utilisé ci-dessous :

```
22022
```

Tout port >1024 conviendrait, à condition qu'il ne soit pas déjà occupé.

Étape 1 – Modification de la configuration SSH

Le fichier de configuration du démon SSH est édité :

```
sudo nano /etc/ssh/sshd_config
```

La directive suivante est ajoutée ou modifiée :

```
Port 22022
```

Si la ligne `Port 22` existe, elle peut :

- soit être remplacée,
- soit être laissée en commentaire.

À ce stade, **un seul port est conservé** afin d'éviter toute ambiguïté.

Validation de la configuration

Avant redémarrage du service, la syntaxe est vérifiée :

```
sudo sshd -t
```

Aucune sortie signifie que la configuration est valide.

Étape 2 – Mise à jour du pare-feu UFW

Le nouveau port SSH doit être explicitement autorisé.

```
sudo ufw allow 22022/tcp
```

L'ancienne règle OpenSSH (port 22) peut être supprimée :

```
sudo ufw delete allow OpenSSH
```

Vérification :

```
sudo ufw status verbose
```

Étape 3 – Redémarrage du service SSH

```
sudo systemctl restart ssh
```

Étape 4 – Test impératif (avant toute déconnexion)

⚠️ **Étape critique**

Dans un **nouveau terminal**, tester la connexion :

```
ssh -p 22022 dome@IP_DE_LA_BOX
```

La connexion doit :

- fonctionner sans mot de passe,
- utiliser la clé SSH existante.

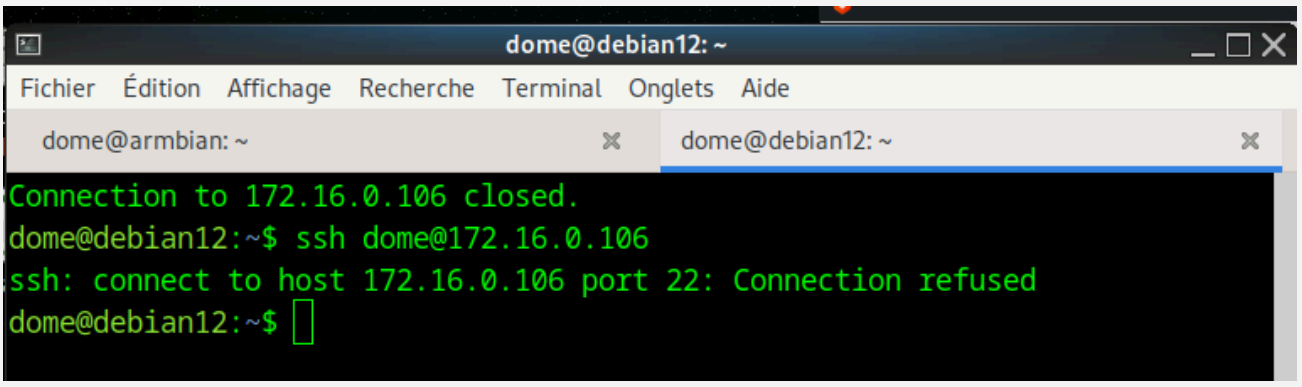
Vérification complémentaire

Tentative volontaire sur l'ancien port :

```
ssh dome@IP_DE_LA_BOX
```

Résultat attendu :

- refus de connexion,
- timeout ou "connection refused".



Étape 5 – Adaptation des outils existants

Fail2ban

Aucune modification n'est nécessaire tant que le jail `sshd` est utilisé.
Fail2ban surveille le service SSH indépendamment du port.

Commandes futures

Pour plus de confort, un alias peut être défini sur la machine cliente :

```
alias ssh-armbian='ssh -p 22022 dome@IP_DE_LA_BOX'
```

Méthode recommandée

Créer le fichier `.bash_aliases` (il n'existe pas par défaut) :

Ajouter :

```
alias ssh-armbian='ssh -p 22022 dome@IP_DE_LA_BOX'
```

Sauvegarder, puis recharger :

```
sudo source ~/.bashrc
```

Ou simplement ouvrir une nouvelle session SSH.

Évaluation du niveau de sécurité actuel

Contexte

Le système est un nœud Linux **headless**, issu d'une TV box Amlogic S905W recyclée, intégré dans un homelab.
L'objectif n'est pas une certification ISO 27001, mais un **niveau de sécurité cohérent, maîtrisé et proportionné**.

Mesures effectivement en place

Accès et authentification

- Connexion SSH en tant que `root` désactivée
- Accès SSH via utilisateur non privilégié (`dome`)
- Authentification par clé SSH (Ed25519)
- Authentification par mot de passe SSH désactivée
- SSH accessible uniquement sur un port non standard

Exposition réseau

- Pare-feu UFW actif
- Politique *deny incoming / allow outgoing*
- Seul SSH explicitement autorisé
- IPv6 désactivé au niveau UFW

Protection active

- Fail2ban actif
- Jail `sshd` opérationnel
- Bannis automatiques fonctionnels
- Notifications locales configurées (Postfix local-only)

Notifications et audit

- MTA local-only (Postfix)
- Alertes Fail2ban stockées localement
- Aucune dépendance externe
- Aucun secret stocké en clair

En résumé :
le système ne se contente plus de démarrer correctement — il sait désormais **dire non**, et c'est souvent la compétence la plus sous-estimée en sécurité.

Ce qui reste améliorable

Rien de critique.
Uniquement des optimisations de confort ou de maturité :

- **changer le port SSH (bruit ↓, sécurité ≈)**
- **restreindre SSH à une plage IP LAN**
- **journalisation centralisée**
- **accès uniquement via VPN (WireGuard)**
- **supervision (Prometheus, alertes)**