Sécuriser Votre Infrastructure

Table des matières

Module 1 : Application des recommandations de l'ANSSI sur GNU/Linux

Objectifs pédagogiques :

- Comprendre les recommandations de l'ANSSI sur la configuration sécurisée d'un système GNU/Linux.
- Savoir auditer, corriger et maintenir un système selon ces recommandations.
- Appliquer une démarche proactive de sécurisation.

Références principales :

- Guide ANSSI "Configuration sécurisée d'un système GNU/Linux"
- CIS Benchmarks Linux
- Normes ISO/IEC 27001 liées aux bonnes pratiques de sécurité

Contenu du module :

1.1. Hygiène de base du système

- **Principe** : Réduire la surface d'attaque dès l'installation.

Actions :

- Choisir une distribution supportée (Debian, RHEL, Ubuntu LTS).
- Utiliser le partitionnement sécurisé :
  - `/home, /var, /tmp` sur des partitions séparées avec options `noexec, nodev, nosuid`.

Exemples :

```
# Afficher les options de montage
mount | grep -E "/var|/tmp|/home"
```

1.2. Durcissement du système

- **Principe** : Modifier les paramètres du noyau et désactiver les services inutiles.

Actions :

- Modifier `/etc/sysctl.conf` pour sécuriser le noyau :

```
# Désactiver la redirection IP
net.ipv4.ip_forward = 0
net.ipv4.conf.all.send_redirects = 0
```

- Interdire les paquets réseau suspects :

```
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.log_martians = 1
```

- Supprimer les services non nécessaires :

```
systemctl disable bluetooth
systemctl disable avahi-daemon
```

1.3. Gestion des comptes et des droits

- **Principe** : Appliquer le principe du moindre privilège.

Actions :

- Supprimer les comptes inutilisés.
- Interdire la connexion directe du compte `root` par SSH.
- Utiliser `sudo` pour les actions administratives.

Exemple :

```
# Interdire root en SSH
echo "PermitRootLogin no" >> /etc/ssh/sshd_config
systemctl restart sshd
```

- Restreindre les permissions sur les fichiers critiques :

```
chmod 600 /etc/shadow
chmod 644 /etc/passwd
```

1.4. Surveillance et audit

- **Principe** : Assurer une traçabilité complète des événements.

Outils recommandés :

- `auditd` : journalisation des accès aux fichiers, changements de droits, etc.
- `logwatch, logrotate, rsyslog.`

Exemple :

```
# Installation et activation de auditd
sudo apt install auditd
sudo systemctl enable auditd
```

1.5. Protection contre les attaques par force brute

- **Principe** : Détecter et bloquer les comportements suspects.

Outils recommandés :

- `fail2ban` : bloque les IP ayant échoué plusieurs tentatives de connexion.

Exemple :

```
sudo apt install fail2ban
sudo systemctl enable fail2ban
```

Configurer `/etc/fail2ban/jail.local` :

```
ini
```

```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 5
```

1.6. Contrôle d'intégrité

- **Principe** : Détecter les modifications non autorisées.

Outils :

- `AIDE` (Advanced Intrusion Detection Environment)

Exemple :

```
sudo apt install aide
sudo aideinit
```

1.7. Sécurité renforcée avec SELinux ou AppArmor

- **Principe** : Appliquer un contrôle d'accès obligatoire (MAC).

Choix :

- AppArmor (Ubuntu, Debian)
- SELinux (CentOS, RHEL)

Exemple AppArmor :

```
sudo apt install apparmor apparmor-utils
sudo aa-status
sudo aa-enforce /etc/apparmor.d/usr.sbin.sshd
```

1.8. Mise à jour régulière du système

- **Principe** : Corriger les vulnérabilités connues.

Commandes utiles :

```
sudo apt update && sudo apt upgrade
```

- Planification avec `unattended-upgrades` ou `dnf-automatic`.

📌 Cas pratique :

Contexte : Une machine Debian utilisée comme serveur web doit être durcie pour être exposée à Internet.

1. Désactivation des services inutiles.
2. Application des réglages de sécurité sysctl.
3. Installation et configuration de fail2ban et auditd.
4. Application de permissions strictes sur `/var/www/`.
5. Vérification du tout avec `lynis` ou `OpenSCAP`.

🛠 Outils d'audit recommandés :

- `Lynis` : outil de diagnostic automatisé.

```
sudo apt install lynis
sudo lynis audit system
```

- `OpenSCAP` : pour une analyse conforme aux normes de sécurité.

Objectif :

Appliquer les recommandations de l'ANSSI pour sécuriser un serveur Debian qui héberge un site web (ex. Apache ou Nginx) en production.

❓ Prérequis :

- Une machine Debian (10 ou 11)
- Droits `root` ou accès via `sudo`
- Serveur web installé (`apache2` ou `nginx`)

🔐 Étapes de durcissement :

---

1. Mise à jour complète du système

```
sudo apt update && sudo apt full-upgrade -y
```

Installez les mises à jour de sécurité et corrigez les vulnérabilités connues.

---

2. Désactivation des services inutiles

Identifiez les services actifs :

```
sudo systemctl list-units --type=service --state=running
```

Désactivez ceux non nécessaires :

```
sudo systemctl disable bluetooth.service
sudo systemctl disable avahi-daemon.service
sudo systemctl stop bluetooth.service
```

---

3. Configuration du pare-feu `ufw`

```
sudo apt install ufw
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 'OpenSSH'
sudo ufw allow 'WWW Full'  # autorise HTTP (80) et HTTPS (443)
sudo ufw enable
```

---

4. Sécurisation du SSH

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

Modifiez les lignes suivantes :

```
nginx
```

```
PermitRootLogin no
PasswordAuthentication no
AllowUsers monutilisateuradmin
```

Redémarrez le service :

```
sudo systemctl restart sshd
```

Utilisez l'authentification par **clé SSH uniquement**.

---

5. Mise en place de Fail2Ban

```
sudo apt install fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

Configurez `/etc/fail2ban/jail.local` :

```
ini
```

```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 5
```

Redémarrez : `sudo systemctl restart fail2ban`

---

6. Durcissement du noyau avec sysctl

```
sudo nano /etc/sysctl.d/99-sysctl.conf
```

Ajoutez :

```
conf
```

```
net.ipv4.ip_forward = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.all.rp_filter = 1
```

Appliquez : `sudo sysctl -p /etc/sysctl.d/99-sysctl.conf`

---

7. Installation et configuration de auditd

```
sudo apt install auditd
sudo systemctl enable auditd
sudo auditctl -w /etc/passwd -p wa -k passwd_changes
```

---

8. Contrôle d'intégrité avec AIDE

```
sudo apt install aide
sudo aideinit
sudo mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db
sudo aide --check
```

---

9. Droits et permissions sur le répertoire web

```
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 750 /var/www/html
```

---

10. Vérification de la sécurité avec Lynis

```
sudo apt install lynis
sudo lynis audit system
```

Notez les recommandations et corrigez selon le rapport.

✓ Résultat attendu :

- Seuls les ports nécessaires (22, 80, 443) sont ouverts.
- Aucun compte root n'est accessible à distance.
- Le système est à jour et surveillé.
- Des logs sont enregistrés, protégés, et analysés.
- Le serveur est conforme aux bonnes pratiques de l'ANSSI.

Module 2 : Gestion sécurisée des identités

2.1. Principes fondamentaux

✓ Principes clés :

- **Principe du moindre privilège** : chaque utilisateur ne possède que les droits nécessaires à ses fonctions.
- **Séparation des responsabilités** : les droits administratifs sont accordés via des comptes distincts.
- **Traçabilité** : chaque action doit être identifiable (journalisation et audit).
- **Révocation rapide** : suppression immédiate des comptes inactifs ou obsolètes.

2.2. Création, gestion et suppression des comptes

🔧 Bonnes pratiques :

- Comptes nominatifs uniquement (éviter les comptes partagés).
- Définition claire du cycle de vie du compte (création → usage → désactivation).
- Utilisation d'un outil de gestion centralisée si possible (LDAP, FreeIPA, Active Directory…).

Exemple :

```
# Création d'un utilisateur sans accès root
sudo adduser alice
sudo usermod -G www-data alice
```

2.3. Gestion des mots de passe

Recommandations ANSSI :

- Longueur minimale : 12 caractères.
- Complexité : majuscules, minuscules, chiffres, caractères spéciaux.
- Expiration : tous les 90 jours (optionnel selon contexte).
- Historique : empêcher la réutilisation d'anciens mots de passe.

Implémentation avec PAM :

Modifiez `/etc/security/pwquality.conf` :

```ini
minlen = 12
dcredit = -1
ucredit = -1
ocredit = -1
lcredit = -1
```

Activez via `/etc/pam.d/common-password` :

```ruby
password requisite pam_pwquality.so retry=3
```

Exemple :

```
# Forcer la modification du mot de passe à la prochaine connexion
sudo chage -d 0 alice
```

2.4. Authentification forte (MFA / 2FA)

Objectif :

Ajouter une couche de sécurité supplémentaire via un second facteur (code, token, biométrie…).

Outil recommandé :

- **Google Authenticator** pour Linux SSH
- **Yubikey / TOTP / WebAuthn** pour les environnements professionnels

Installation de Google Authenticator :

```
sudo apt install libpam-google-authenticator
google-authenticator
```

Ajoutez à `/etc/pam.d/sshd` :

swift

```
auth required pam_google_authenticator.so
```

Et dans `/etc/ssh/sshd_config` :

nginx

```
ChallengeResponseAuthentication yes
```

2.5. Gestion des privilèges administrateurs

🔐 Objectif :

Contrôler l'utilisation des droits root.

Bonnes pratiques :

- Ne jamais utiliser le compte `root` en direct.
- Utiliser `sudo` avec journalisation des commandes.
- Attribuer des commandes spécifiques à des groupes.

🛠 Exemple :

```
# Ajouter un utilisateur au groupe sudo
sudo usermod -aG sudo alice

# Fichier sudoers personnalisé :
sudo visudo
alice ALL=(ALL:ALL) /bin/systemctl restart nginx.service
```

2.6. Désactivation ou verrouillage de comptes inactifs

🛠 Commandes utiles :

```
# Verrouiller un compte :
sudo usermod -L bob

# Supprimer un compte et son dossier personnel :
sudo deluser --remove-home bob

# Trouver les comptes inactifs depuis 30 jours :
sudo lastlog -b 30
```

2.7. Centralisation de l'authentification

🔐 Objectif :

Unifier et tracer l'authentification via :

- **LDAP** (ex : OpenLDAP, FreeIPA)
- **Kerberos**
- **SSO** (Keycloak, Azure AD, SAML)

🔐 Avantages :

- Meilleure gestion des droits et des groupes.
- Authentification unique (SSO).
- Révocation immédiate centralisée.

2.8. Journalisation et traçabilité

🧐 Objectif :

Savoir qui a fait quoi, quand, et comment.

Outils :

- `auditd`, `journald`
- Fichier `/var/log/auth.log`
- `sudo` avec options `log_output`

🧐 Exemple :

```
# Voir les connexions SSH
grep "sshd" /var/log/auth.log
```

Module 3 : Déploiement d'un pare-feu pfSense sous VirtualBox

3.1. Introduction à pfSense

🔎 Qu'est-ce que pfSense ?

- **pfSense** est une distribution **FreeBSD** dédiée à la sécurité réseau.
- Fonctionnalités principales :
  - Filtrage de paquets (pare-feu stateful)
  - NAT, VPN (IPSec, OpenVPN, WireGuard)
  - DHCP, DNS, proxy, etc.
  - Interface Web d'administration

🎁 Versions :

- pfSense CE (Community Edition)
- pfSense Plus (commerciale)

3.2. Préparation de l'environnement VirtualBox

🧐 Prérequis :

- Installer **VirtualBox**
- Télécharger l'**ISO de pfSense CE** depuis https://www.pfsense.org/download/
- Avoir une **image ISO FreeBSD x64**

💼 Réseau à simuler :

- **WAN** : vers Internet
- **LAN** : réseau interne vers un poste client

3.3. Création de la machine virtuelle pfSense

🛠️ Étapes dans VirtualBox :

1. **Créer une VM** :
   - Nom : `pfSense`
   - Type : `BSD`, Version : `FreeBSD (64-bit)`
   - RAM : 1024 à 2048 Mo
   - Disque : 10 Go (dynamique)
2. **Ajouter 2 cartes réseau** :
   - **Carte 1** (WAN) : mode `NAT` ou `Bridged`
   - **Carte 2** (LAN) : mode `Réseau interne` (nom : `LAN`)
3. **Monter l'ISO pfSense** comme disque d'amorçage.

3.4. Installation de pfSense

❓ Étapes de l'installation :

- Boot sur l'ISO → Installation automatique (`Install pfSense`)
- Sélection du disque → `Auto (UFS)`
- Accepter la configuration automatique des interfaces
- Attribuer :
  - WAN → `em0`
  - LAN → `em1`
- Rebooter (en retirant l'ISO)

3.5. Configuration initiale de pfSense

💼 IP statique sur l'interface LAN :

- Depuis l'interface texte de pfSense :

```
Option 2) Set interface IP address
Interface: LAN (em1)
IP: 192.168.1.1
Netmask: /24
No DHCP
```

💻 Depuis le navigateur du poste client connecté sur LAN :

- URL : http://192.168.1.1
- Login : `admin` / `pfsense`
- Suivre le **Setup Wizard** :
  - Définir mot de passe admin
  - Configurer NTP, DNS
  - Vérifier interfaces WAN/LAN

3.6. Tests de connectivité réseau

🌐 Tester l'accès Internet :

- Depuis pfSense (Shell) : `ping 8.8.8.8`

🖥️ Tester du poste client :

- IP manuelle sur le client : `192.168.1.10/24`, GW : `192.168.1.1`
- Tester accès à Internet si NAT activé sur WAN

3.7. Configuration du pare-feu (firewall rules)

🔒 Par défaut :

- **Sortie LAN vers WAN** autorisée
- **Entrée WAN vers LAN** interdite

🛠️ Ajouter des règles :

1. Allez dans : **Firewall > Rules > LAN**
2. Ajouter :
    - Source : LAN net
    - Destination : any
    - Action : Pass
3. Bloquer un port spécifique (ex : port 22) :
    - **Firewall > Rules > WAN**
    - Source : any, Port : 22
    - Action : Block

3.8. Surveillance et logs

📊 Monitoring intégré :

- **Status > System Logs**
- **Status > Traffic Graph**
- **Diagnostics > Packet Capture**

🔔 Notifications :

- Envoi d'emails en cas d'alerte (SMTP)
- Intégration possible avec des SIEM via **syslog**

3.9. Sauvegarde et restauration

💾 Sauvegarder :

- **Diagnostics > Backup & Restore**
- Exporter un fichier `.xml`

🔁 Restaurer :

- Depuis la même interface Web
- Très utile pour tests, mises à jour, déploiement

❓ Étude de cas pratique :

Contexte : Simuler un accès distant sur le port 80 depuis le WAN vers un serveur web sur le LAN.

Étapes :

1. Créer une VM cliente dans le réseau LAN avec un serveur Apache (192.168.1.100).
2. Dans pfSense, aller dans **Firewall > NAT > Port Forward**
    - Interface : WAN
    - Port externe : 80
    - IP interne : 192.168.1.100
    - Port : 80
    - Créer automatiquement la règle de pare-feu
3. Depuis une autre VM sur le WAN, tester : `curl http://<IP-WAN-de-pfSense>`

Module 4 : Déploiement des certificats

4.1. Introduction aux certificats numériques

🔑 Définitions clés :

- **Certificat** : fichier numérique qui lie une clé publique à une identité.
- **CA (Certification Authority)** : entité de confiance qui délivre les certificats.
- **PKI (Public Key Infrastructure)** : ensemble des services permettant la gestion des certificats.

🔒 Usages :

- Chiffrement des communications (HTTPS, VPN, SMTP…)
- Authentification (clients, serveurs)
- Signature électronique

4.2. Structure d'un certificat X.509

Contenu typique :

- Nom du sujet (CN)
- Clé publique
- Validité (dates de début et fin)
- Nom de l'émetteur (CA)
- Signature de la CA

Exemple : `openssl x509 -in moncert.pem -text -noout`

4.3. Génération d'un certificat auto-signé (Self-signed)

 Commande OpenSSL :

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout monsite.key -out monsite.crt \
-subj "/C=FR/ST=Paris/L=Paris/O=Entreprise/CN=monsite.local"
```

Utilisé pour les tests ou les environnements internes.

4.4. Création d'une autorité de certification (CA) interne

 Étapes :

1. Créer la clé privée :

```
openssl genrsa -out ca.key 4096
```

2. Générer le certificat CA :

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt
```

3. Signer des CSR (Certificate Signing Requests) de vos serveurs :

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key \
-CAcreateserial -out server.crt -days 825 -sha256
```

4.5. Génération d'un CSR pour un serveur web

```
openssl req -new -newkey rsa:2048 -nodes -keyout site.key -out site.csr
```

Paramètres à remplir (CN = FQDN du site).

4.6. Déploiement dans un serveur web (Apache ou Nginx)

Apache :

```
Apacheconf

<VirtualHost *:443>
  ServerName monsite.local
  SSLEngine on
  SSLCertificateFile /etc/ssl/certs/site.crt
  SSLCertificateKeyFile /etc/ssl/private/site.key
</VirtualHost>
```

Activation :

```
sudo a2enmod ssl
sudo systemctl reload apache2
```

Nginx :

```
server {
  listen 443 ssl;
  server_name monsite.local;

  ssl_certificate /etc/ssl/certs/site.crt;
  ssl_certificate_key /etc/ssl/private/site.key;
```

```
  location / {
    root /var/www/html;
  }
}
```

4.7. Obtenir des certificats gratuits avec Let's Encrypt

Installation de Certbot :

```
sudo apt install certbot python3-certbot-apache  # ou python3-certbot-nginx
```

Génération automatique :

```
sudo certbot --apache -d monsite.com
```

Renouvellement automatique :

```
sudo certbot renew --dry-run
```

4.8. Vérification des certificats

Vérifier les détails :

```
openssl x509 -in moncert.crt -noout -text
```

Tester depuis un client :

```
openssl s_client -connect monsite.com:443
```

4.9. Gestion du cycle de vie des certificats

🗓️ Bonnes pratiques :

- Surveiller la date d'expiration (`cron`, `Nagios`, `Zabbix`)
- Utiliser des durées de vie courtes pour limiter le risque
- Révoquer en cas de compromission (CRL ou OCSP)
- Protéger strictement les clés privées

❓ Cas pratique suggéré :

Contexte : Vous êtes en charge d'un site intranet `intranet.local` non exposé à Internet. Vous devez le sécuriser avec HTTPS via un certificat signé par votre propre CA.

Étapes :

1. Créer votre autorité de certification (CA)
2. Générer une CSR pour `intranet.local`
3. Signer le certificat avec votre CA
4. Déployer le certificat et la clé sur le serveur Apache local
5. Ajouter le certificat racine de la CA au navigateur pour le marquer comme « de confiance »

🔹 Étape 1 : Créer l'autorité de certification (CA)

📁 Créer un dossier de travail :

```
mkdir -p ~/certs/CA
cd ~/certs/CA
```

🔑 Générer la clé privée de la CA :

```
openssl genrsa -out ca.key 4096
```

🔹 Créer le certificat de la CA :

```
openssl req -x509 -new -key ca.key -sha256 -days 3650 -out ca.crt \
-subj "/C=FR/ST=Ile-de-France/L=Paris/O=Entreprise/CN=CA-MonIntranet"
```

✅ Vous avez maintenant une **CA prête à signer des certificats**.

📄 Étape 2 : Générer le CSR pour votre serveur

🔑 Générer la clé privée du serveur :

```
openssl genrsa -out serveur.key 2048
```

📄 Créer le fichier de requête (CSR) :

```
openssl req -new -key serveur.key -out serveur.csr \
-subj "/C=FR/ST=Ile-de-France/L=Paris/O=MonSite/CN=intranet.local"
```

✍️ Étape 3 : Signer le certificat avec la CA

```
openssl x509 -req -in serveur.csr -CA ca.crt -CAkey ca.key \
-CAcreateserial -out serveur.crt -days 825 -sha256
```

✅ Vous avez maintenant :

- `serveur.crt` (certificat signé)
- `serveur.key` (clé privée)
- `ca.crt` (certificat racine)

🔹 Étape 4 : Déployer sur Apache

📁 Placer les fichiers :

```
sudo cp serveur.crt /etc/ssl/certs/
sudo cp serveur.key /etc/ssl/private/
sudo cp ca.crt /usr/local/share/ca-certificates/ca.crt
sudo update-ca-certificates
```

⚙️ Modifier le vhost Apache :

```
apacheconf
```

```
<VirtualHost *:443>
    ServerName intranet.local
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/serveur.crt
```

```
    SSLCertificateKeyFile /etc/ssl/private/serveur.key
</VirtualHost>
```

Activer HTTPS :

```
sudo a2enmod ssl
sudo systemctl reload apache2
```

Étape 5 : Ajouter le certificat CA dans votre navigateur (manuellement)

Pour Firefox / Chrome :

- Aller dans Paramètres > Sécurité > Gérer les certificats > Importer
- Ajouter `ca.crt` comme certificat de confiance pour les sites Web

⚠ Sans cette étape, votre navigateur marquera le site comme non sécurisé.

Tests à faire

```
# Tester la validité du certificat
openssl x509 -in /etc/ssl/certs/serveur.crt -noout -text

# Tester la connexion SSL
openssl s_client -connect intranet.local:443
```

Module 5 : Mise en place des environnements de test et de pré-production

---

Objectifs pédagogiques :

- Comprendre les différences entre les environnements **de développement**, **de test**, **de pré-production** et **de production**.
- Apprendre à concevoir, déployer et configurer des environnements **sécurisés** pour les phases de test et de validation.
- Réduire les risques lors du déploiement en production grâce à une **simulation fidèle** des conditions réelles.

Contenu détaillé :

5.1. Environnements : définitions et rôles

Environnement	Usage principal	Accès
Développement	Écriture de code, tests unitaires	Équipe Dev
Test	Tests fonctionnels, intégration	Équipe QA/Test
Pré-production	Validation finale avant production	Équipe entière
Production	Environnement utilisé par les clients	Utilisateurs finaux

---

5.2. Bonnes pratiques de séparation des environnements

- 🔒 **Isolation réseau** : Chaque environnement est sur un VLAN ou une VM différente.
- 🔐 **Accès contrôlé** : VPN, pare-feu, rôles IAM distincts.
- 🎁 **Clonage de l'environnement prod** : pour simuler au plus près les conditions réelles.
- 🚫 **Aucune donnée réelle** dans les environnements de test (RGPD, sécurité).

5.3. Mise en place de l'environnement de test

❓ Outils recommandés :

- Machines virtuelles (VirtualBox, Proxmox, VMware)
- Conteneurs (Docker)
- Infrastructure as Code (Terraform, Ansible)

🏗️ Exemple : créer un environnement test d'une application web

1. Cloner le dépôt Git sur une machine virtuelle
2. Déployer le backend (ex : Flask/Django) + base de données (MariaDB/PostgreSQL)
3. Déployer le frontend (React, Angular)
4. Utiliser des données de test générées
5. Simuler des charges (Apache Benchmark, JMeter)

5.4. Mise en place de l'environnement de pré-production

🎯 Objectif : miroir fidèle de la production

- Même architecture que la prod (serveurs, OS, versions de services)
- Données **anonymisées ou synthétiques** de production
- Surveillance et alertes configurées
- Tests de montée en charge, de sécurité, de compatibilité

🏗️ Exemple : simulation d'une infra web sécurisée

- Reverse proxy avec Nginx
- Certificats SSL Let's Encrypt ou internes
- Base de données restaurée à partir d'un dump réel anonymisé
- Load-balancer simulé (HAProxy)
- Firewall (pfSense, iptables)

5.5. Sécurisation des environnements

- 🔑 Comptes distincts pour chaque environnement
- 🔐 Mots de passe générés automatiquement
- 🔒 Limitation des connexions SSH aux IPs autorisées
- ❓ Révocation des certificats expirés
- ❓ Nettoyage régulier des ressources inutiles (volumes, conteneurs, images obsolètes)

5.6. Stratégie de déploiement entre environnements

🔁 Étapes type :

1. **Build** dans Dev
2. **Test automatique (CI/CD)** dans Test
3. **Validation manuelle ou semi-automatisée** dans Pré-prod
4. **Déploiement progressif** vers Production

🎁 Outils :

- GitLab CI/CD, Jenkins, GitHub Actions
- Docker Compose, Kubernetes, Helm Charts

5.7. Documentation des environnements

- Schéma réseau
- Liste des services déployés
- Variables d'environnement
- Accès SSH, ports ouverts, certificats en usage
- Procédures de mise à jour/test

❓ Cas pratique :

Contexte : Vous gérez une application métier critique. L'objectif est de créer une pré-production pour tester les mises à jour majeures avant déploiement en prod.

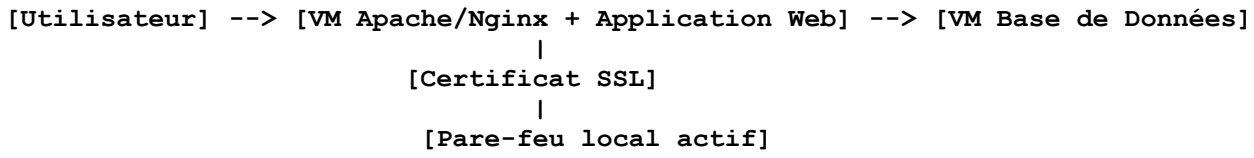
Tâches à réaliser :

1. Cloner l'application depuis GitHub
2. Déployer sur 2 machines virtuelles : serveur applicatif et serveur BDD
3. Ajouter un pare-feu avec des règles minimales
4. Simuler une montée en charge avec `ab` ou `wrk`
5. Tester un certificat SSL interne
6. Documenter les services, IPs, ports utilisés

🎯 Contexte :

Vous devez mettre en place un **environnement de pré-production** pour tester une **application métier critique** avant son déploiement en production. L'environnement doit simuler une architecture réelle avec deux machines virtuelles : un serveur applicatif et un serveur base de données. L'ensemble doit être **sécurisé, documenté et prêt à accueillir des tests de charge.**

🏗️ Architecture ciblée



🛠 Étapes de mise en place

1. Création de deux VM VirtualBox

- VM1 : `app-vm` (Ubuntu Server ou Debian)
- VM2 : `db-vm` (MariaDB/PostgreSQL)
- Réseau : mode **réseau interne** ou **réseau privé hôte**

📌 *Exemple VirtualBox :*

```
VBoxManage createvm --name app-vm --register
VBoxManage createvm --name db-vm -register

# Configurer mémoire, disque, carte réseau, ISO, etc.
```

2. Déploiement de l'application (VM1)

a. Cloner l'application :

```
git clone https://github.com/monorganisation/monapp.git
cd monapp
```

b. Installer les dépendances (Python / Node / etc.) :

```
sudo apt install python3-venv nginx -y
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

c. Configurer le serveur web :

```
sudo cp monapp.conf /etc/nginx/sites-available/
sudo ln -s /etc/nginx/sites-available/monapp.conf /etc/nginx/sites-enabled/
sudo systemctl reload nginx
```

3. Déploiement de la base de données (VM2)

a. Installer PostgreSQL :

```
sudo apt install postgresql -y
```

b. Créer la base et l'utilisateur :

```
sudo -u postgres psql
CREATE DATABASE monapp;
CREATE USER monuser WITH ENCRYPTED PASSWORD 'password123';
GRANT ALL PRIVILEGES ON DATABASE monapp TO monuser;
```

c. Modifier `pg_hba.conf` et `postgresql.conf` pour autoriser l'IP de `app-vm`.

4. Mise en place d'un pare-feu (ufw sur chaque VM)

VM1 :

```
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

VM2 :

```
sudo ufw allow from <IP_VM1> to any port 5432
sudo ufw enable
```

5. Déploiement d'un certificat SSL auto-signé

```
openssl req -x509 -newkey rsa:2048 -nodes -keyout monapp.key \
-out monapp.crt -days 365 -subj "/C=FR/ST=Paris/L=Paris/O=Test/CN=intranet.local"

sudo cp monapp.crt /etc/ssl/certs/
sudo cp monapp.key /etc/ssl/private/
```

Configurer Nginx pour l'utiliser :

```
server {
  listen 443 ssl;
  ssl_certificate /etc/ssl/certs/monapp.crt;
  ssl_certificate_key /etc/ssl/private/monapp.key;
  ...
}
```

6. Test de montée en charge

Avec `ab` (Apache Benchmark) :

```
ab -n 1000 -c 10 https://intranet.local/
```

Avec `wrk` (plus performant) :

```
wrk -t4 -c100 -d30s https://intranet.local/
```

7. Documentation technique

- Adresse IP des VM
- Versions installées (OS, BDD, serveur web)
- Accès SSH
- Ports ouverts
- Commandes de déploiement utilisées
- Fichier `README_PREPROD.md`

✓ Résultat attendu

Élément	Statut
2 VM configurées	✓ OK
Application déployée	✓ OK
Base de données accessible	✓ OK
Pare-feu actif et restrictif	✓ OK
Certificat SSL fonctionnel	✓ OK
Test de charge effectué	✓ OK
Documentation produite	✓ OK

Module 6 : Utilisation et validation d'un environnement de pré-production

6.1. Rôle stratégique de la pré-production

La pré-production est un **miroir de l'environnement de production**, permettant :

Objectif	Détail
✅ Valider les nouvelles fonctionnalités	Avant d'impacter des utilisateurs réels
✅ Tester les montées de version	OS, bases de données, packages applicatifs
✅ Évaluer les performances	Sous charge réelle ou simulée
✅ Contrôler la sécurité	Scanner de vulnérabilités, essais d'intrusion
✅ Réduire les erreurs en prod	En détectant les bugs ou problèmes d'intégration

6.2. Étapes de validation en pré-production

🧪 1. Tests fonctionnels

- Exécution des cas d'usage standards
- Vérification des routes, de la base de données, des fichiers logs
- Outils : Postman, curl, navigateurs, scripts de test

📈 2. Tests de montée en charge

- Simuler des accès simultanés
- Observer le comportement mémoire / CPU
- Outils : `ab`, `wrk`, `JMeter`, `Locust`

🛡️ 3. Tests de sécurité

- Vérifier la configuration SSL
- Utiliser des scanners de vulnérabilités : `Lynis`, `OpenVAS`, `Nessus`
- Vérification des permissions et des ports ouverts : `nmap`, `netstat`, `ufw`

🔁 4. Tests de déploiement

- Simulation d'un rollback (revenir à la version précédente)
- Tests d'installation / mise à jour de paquets
- Validation de scripts de migration de base de données

🧾 5. Validation documentaire

- Vérifier que toute action est consignée dans un guide clair
- Existence d'un plan de retour arrière (backout plan)

6.3. Conditions d'une bonne pré-production

Critère	Bonnes pratiques
Isolation réseau	Pas d'accès direct à la prod ou internet libre
Comptes utilisateurs dédiés	Pas de comptes admin partagés ou root global
Donnees fictives	Jamais de données client ou personnelles
Monitoring activé	Logs, alertes, performances surveillées
SSL/TLS activé	Même sécurité que la prod
Pare-feu actif	Règles similaires à la production

6.4. Validation : grille d'évaluation

Test	Résultat attendu	État
Accès à l'application	Réponse HTTP 200 / HTTPS OK	✅
Authentification fonctionnelle	Accès contrôlé	✅
Base de données répond	Requête SQL correcte	✅
Stress test à 200 connexions	Réponse en < 2 secondes	✅
Port non utilisés fermés	`nmap` montre ports filtrés	✅
Certificat SSL valide	Pas d'erreur navigateur	✅

6.5. Cas pratique – Utilisation et validation

🛠️ Contexte :

Vous avez un environnement pré-production avec une app web, base de données, SSL et pare-feu. Vous devez tester avant un déploiement en production.

✅ À valider :

1. Connexion via HTTPS avec certificat valide
2. Réponse correcte des endpoints de l'API
3. Temps de réponse < 1s à 50 utilisateurs
4. Tous les ports inutiles fermés
5. Fichier `README` et `CHANGELOG` à jour
6. Requête SQL simulée : pas de lenteur
7. Aucune erreur critique dans les logs

📄 Livrables attendus

- Rapport de test avec captures d'écran ou sortie terminal
- Fichier de validation avec :
  - ✅ Test réussi
  - ❌ Test échoué
  - 📌 Recommandations
- Documentation technique mise à jour

Cas Pratique – Validation d'un environnement de pré-production

🎯 Contexte :

Vous avez déployé une application métier sur un environnement de pré-production avec :

- Un serveur web (Nginx ou Apache) avec certificat SSL auto-signé
- Une base de données (PostgreSQL ou MariaDB)
- Un pare-feu actif (UFW ou pfSense)
- Un système de monitoring/logging simple
  Votre mission est de **valider la stabilité, la sécurité et la conformité** avant passage en production.

✅ Étapes de validation

🔹 1. Vérification de l'accessibilité web en HTTPS

✅Test :

```
curl -k https://intranet.local/
```

- **Résultat attendu** : Code 200 OK
- **Test SSL (local)** :

```
openssl s_client -connect intranet.local:443
```

✓ **Certificat détecté**, chiffrement actif, expiration vérifiée

---

🔹 2. Test fonctionnel de l'application

✅Méthodes :

- Naviguer dans l'application
- Utiliser Postman pour tester les endpoints d'API
- Vérifier les fonctionnalités de login, requêtes, enregistrement

Exemple :

```
curl -k -X POST https://intranet.local/api/login -d '{"user":"test","pass":"123"}'
```

✓ **Connexion OK**, réponse HTTP 200, session établie

🔺 3. Test de montée en charge simple

Outils :

- `ab` (Apache Benchmark)
- `wrk`

Commande :

```
ab -n 1000 -c 20 https://intranet.local/
```

- Temps de réponse moyen < 500ms
- Aucun plantage serveur
- RAM et CPU surveillés avec `htop`, `vmstat`

✓ Serveur **stable**, pas de saturation mémoire

🔺 4. Analyse de la base de données

Requête de performance :

`sql`

```
EXPLAIN ANALYZE SELECT * FROM clients WHERE email='contact@test.com';
```

✓ Temps de requête < 50ms

Vérification de la connectivité sécurisée :

```
telnet db-server 5432
```

✓ Port ouvert uniquement pour IP autorisée

🔺 5. Scan de sécurité réseau

Outil :

```
nmap -sS -T4 -Pn intranet.local
```

✓ Ports ouverts uniquement : 22, 80, 443
⊘ Ports non utilisés : fermés ou filtrés

Pare-feu :

```
sudo ufw status verbose
```

✓ Règles bien en place (whitelist IP, rejet par défaut)

🔺 6. Contrôle des journaux et erreurs

Nginx/Apache :

```
sudo tail -n 100 /var/log/nginx/error.log
```

Application :

- Logs applicatifs sans erreurs critiques
- Surveillance de la RAM / CPU (`htop`, `glances`)

✓ Pas de stacktrace, erreurs 5xx, ou fuites mémoire visibles

🔻 7. Vérification documentaire

- Présence d'un fichier `README_PREPROD.md` :
  - Liste des services
  - Accès, IP, ports, commandes
- Présence d'un `CHANGELOG.md` pour les mises à jour
- Plan de retour arrière (`rollback.md`)

✓ **Documentation conforme** et à jour

📝 Résumé dans un tableau de validation

Test	Statut	Commentaire
HTTPS accessible avec certificat valide	✅	Certificat auto-signé accepté
Fonctionnalités principales testées	✅	API, login, enregistrement
Montée en charge à 20 connexions	✅	Aucun crash ou timeout
Requête SQL rapide et indexée	✅	< 50ms pour SELECT
Ports non utilisés fermés (NMAP)	✅	Pare-feu UFW actif
Logs sans erreurs critiques	✅	Application stable
Documentation présente et lisible	✅	README et plan rollback fournis

Module 7 : Tests de mises à jour de sécurité

7.1. Pourquoi tester les mises à jour de sécurité ?

Risques sans test préalable	Conséquences possibles
Incompatibilité logicielle	L'application peut cesser de fonctionner
Problème de dépendances	Paquets cassés, instabilité du système
Régression fonctionnelle	Pertes de fonctionnalités
Redémarrage de services critiques	Rupture de service

✅ Tester les mises à jour **réduit le risque de panne** lors du déploiement en production.

7.2. Types de mises à jour de sécurité

Type	Exemples
Système	`linux-image`, `glibc`, `openssl`, `sudo`
Services critiques	`nginx`, `apache2`, `ssh`, `postgresql`
Bibliothèques	`libssl`, `python3-requests`, `libxml2`
Kernel	Mises à jour nécessitant redémarrage complet

7.3. Procédure de test d'une mise à jour

Étape 1 : Liste des mises à jour disponibles

```
sudo apt update
apt list –upgradable
```

Étape 2 : Identifier celles marquées "security"

```
apt-get upgrade -s | grep security
```

Étape 3 : Sauvegarder le système (snapshot ou backup)

- VM : créer un **snapshot VirtualBox**
- Serveur : sauvegarde avec `rsync`, `Borg`, ou `tar`

Étape 4 : Appliquer les mises à jour dans l'environnement de test

```
sudo apt install --only-upgrade <paquet>
```

Étape 5 : Redémarrer les services concernés

```
sudo systemctl restart nginx
sudo systemctl status nginx
```

7.4. Validation post-mise à jour

Vérification	Commande ou méthode
Fonctionnement de l'application	Tests manuels ou automatisés
Statut des services	`systemctl status`, `journalctl`
Intégrité des logs	Aucun nouveau message d'erreur
Sécurité	Scan `Lynis`, `nmap`, `chkrootkit`
Kernel (si mis à jour)	`uname -r` + redémarrage + test complet

7.5. Automatisation des tests de mise à jour

- Utilisation de **playbooks Ansible** pour automatiser :
  - L'installation
  - Les tests de service
  - Le rollback si nécessaire

Exemple avec Ansible :

`yaml`

```yaml
- name: Apply security updates
  apt:
    upgrade: safe
    update_cache: yes
```

7.6. Gestion du rollback

Avant toute mise à jour :

- Créer une **sauvegarde** ou un **snapshot**
- Conserver les anciennes versions dans le cache (`/var/cache/apt/archives`)
- En cas de panne : revenir à la version précédente

Cas pratique – Test d'une mise à jour

Contexte :

Un correctif critique de `OpenSSL` vient de sortir. Il doit être appliqué sur les serveurs. Vous devez le **tester en pré-production avant la production.**

Étapes :

1. Snapshot de la VM
2. Mise à jour :

```
sudo apt install --only-upgrade openssl
```

3. Redémarrage de `apache2` ou `nginx`
4. Test de chiffrement TLS :

```
openssl s_client -connect localhost:443
```

5. Vérifier les journaux : `journalctl -xe, /var/log/nginx/error.log`

✓ Si tout est conforme, vous pouvez planifier le déploiement en production avec la même procédure.

Module 8 : Mise à jour de la documentation technique après test

8.1. Pourquoi mettre à jour la documentation technique après un test ?

Une documentation **obsolète** peut causer :

❌ Des erreurs lors des interventions techniques.

❌ Une mauvaise gestion des mises à jour et des configurations.

❌ Des retards en cas d'incident.

Une documentation **à jour** assure :

✅ Une **répétition fidèle** des opérations testées.

✅ Une **meilleure collaboration** entre les équipes.

✅ Un **historique clair** des changements et tests réalisés.

8.2. Types de documentation technique à mettre à jour

Type de documentation	Contenu
Documentation système	OS, versions, configurations réseau
Procédures opérationnelles	Déploiements, mises à jour, rollback
Tests et validations	Rapports de tests, checklist de validation
Documentation sécurité	Patches appliqués, analyse des vulnérabilités

Journal des modifications (Changelog) Liste des changements apportés

8.3. Structurer une documentation technique après un test

✅Exemple de structure :

```
🗁 Documentation technique
  ├── 01_Architecture.md
  ├── 02_Configuration_Systeme.md
  ├── 03_Mise_a_jour_Securite.md
  ├── 04_Tests_Validation.md
  ├── 05_Procedures_Opérationnelles.md
  ├── 06_Changelog.md
```

✅Contenu minimal d'une fiche de mise à jour

Section	Détails
Date et auteur	Qui a réalisé la mise à jour ?
Contexte	Pourquoi la mise à jour a été effectuée ?
Tests réalisés	Quelle méthode a été utilisée ?
Résultats des tests	Fonctionnement validé ou erreurs détectées ?
Rollback possible ?	Oui/Non + procédure de retour arrière
Impact sur l'environnement	Services redémarrés, dépendances mises à jour
Lien vers logs/screenshots	Preuves des tests réalisés

8.4. Outils pour gérer la documentation technique

Type	Outils recommandés
Fichiers Markdown	GitHub Wiki, MkDocs, Hugo
Wikis collaboratifs	Confluence, MediaWiki
Systèmes de versioning	Git, GitLab Docs
Gestion documentaire	SharePoint, Nextcloud
Automatisation des mises à jour	Ansible Docs, Redoc

📌 **Markdown** est souvent utilisé pour stocker la documentation dans Git.
📌 **Confluence** est idéal pour la collaboration dans les entreprises.

8.5. Intégrer la mise à jour documentaire dans le workflow DevOps

✅ **Automatiser la documentation avec Git**

1. Modifier la documentation en Markdown.
2. Ajouter et commit dans le dépôt Git :

```
git add Documentation/
git commit -m "Mise à jour de la doc après test de mise à jour de sécurité"
git push origin main
```

3. Lancer une **review** pour validation avant intégration.

✅ **Mise à jour automatique avec Ansible**

- Utiliser Ansible pour générer une documentation à partir des configurations actuelles.
- Exemple de tâche Ansible :

```
- name: Générer la documentation système
  command: ansible-inventory --list > docs/inventaire.json
```

🔧 Cas pratique – Mise à jour d'une documentation après un test

Contexte : Vous venez de réaliser un test de mise à jour de sécurité sur un serveur Ubuntu. Vous devez mettre à jour la documentation technique pour que l'équipe puisse reproduire cette opération à l'avenir.

✅ Étapes à suivre :

1️⃣ Créer un fichier de mise à jour

```
nano Documentation/03_Mise_a_jour_Securite.md
```

2️⃣ Ajouter ces informations :

```
markdown
```

```
# Mise à jour de sécurité – Serveur Web Apache

- **Date :** 05/2025
- **Auteur :** Admin DevOps
- **Contexte :** Mise à jour de sécurité critique Apache
- **Paquets concernés :** apache2 (v2.4.57)
- **Commandes exécutées :**
  ```
  sudo apt update && sudo apt upgrade apache2
  ```
- **Tests réalisés :**
  - ✅ Vérification de la version avec `apache2 -v`
  - ✅ Test du site web après redémarrage
- **Problèmes rencontrés :** Aucun
- **Rollback possible :** Oui, via snapshot VM
- **Lien vers logs :** `/var/log/apt/history.log`
```

3️⃣ Commit et push vers Git :

```
git add Documentation/03_Mise_a_jour_Securite.md
git commit -m "Ajout de la procédure de mise à jour Apache"
git push origin main
```

4️⃣ Partager sur le wiki de l'entreprise

Module 9 : Lecture de documentation technique en anglais

🔹 9.1. Types de documentation technique en anglais

Type de document	Exemple
Rapports de vulnérabilité	CVE reports (ex: NIST NVD, Mitre.org)
Documentation de logiciels	Guides pfSense, OpenSSH, Apache, Nginx
Logs et messages systèmes	Messages de journal système, erreurs, warnings
Manuels de commandes (man pages)	`man ssh`, `man iptables`
Forums/Stack Overflow	Solutions communautaires aux problèmes techniques

🔹 9.2. Vocabulaire de base de la sécurité informatique (EN/FR)

Anglais	Français
vulnerability	vulnérabilité
patch	correctif
exploit	code exploitant une faille
disclosure	divulgation

Anglais	Français
buffer overflow	débordement de mémoire tampon
privilege escalation	élévation de privilèges
authentication	authentification
firewall	pare-feu
misconfiguration	mauvaise configuration

🔹 9.3. Méthode de lecture efficace

Étapes de compréhension :

1. **Identifier le contexte** (What is the software? What is the issue?)
2. **Lire les mots-clés techniques** (port, privilege, access, authentication, config file…)
3. **Repérer les commandes, exemples, outputs** à tester
4. **Traduire les paragraphes complexes** uniquement si nécessaire
5. **Mettre en pratique** la solution décrite

🔹 9.4. Exemple concret – Lecture d'un rapport CVE

Exemple : CVE-2024-12345

Extrait en anglais :

"A vulnerability in OpenSSH 8.9 allows a remote attacker to perform unauthorized code execution due to improper input validation in the scp module. The issue affects systems where scp is enabled and accessible remotely."

Interprétation :

- Vulnérabilité dans **OpenSSH 8.9**
- Attaque distante possible (**remote attacker**)
- Exécution de code sans autorisation (**unauthorized code execution**)
- Module concerné : **scp**
- Condition : scp activé et exposé

Action :

- Vérifier la version locale : `ssh -V`
- Désactiver temporairement `scp`
- Appliquer le patch ou mettre à jour : `apt upgrade openssh-client`

🔹 9.5. Recherches techniques en anglais

🔍 Exemple :

Problème : Erreur `Permission denied (publickey)` lors d'une connexion SSH.

Bonne requête Google :

`openssh permission denied (publickey) fix ubuntu`

 Avantages :

- Réponses sur forums (AskUbuntu, StackOverflow)
- Solutions testées, souvent avec exemples

Cas pratique – Exploiter un guide de mise à jour en anglais

Objectif :

Lire et appliquer les instructions d'un article de blog/documentation pour patcher une faille critique.

Extrait en anglais :

"To fix CVE-2023-9999, you must update OpenSSL to 1.1.1w. After updating, restart the service using: `systemctl restart apache2`*. Verify the new version with* `openssl version`*."*

Instructions dérivées :

1. **`sudo apt update && sudo apt install openssl`**
2. **`sudo systemctl restart apache2`**
3. **`openssl version`** → **vérifier** **`1.1.1w`**

✓ Mise à jour réussie, vulnérabilité corrigée.

Bonnes pratiques

- Toujours **vérifier la date** du document
- Favoriser les sources officielles : vendor, GitHub, NIST, Debian Security
- Garder un **glossaire anglais-français** à jour
- S'exercer à lire **une section par jour** (guide, patch, forum)

Module 10 : Veille technologique sur les accès sécurisés aux serveurs

10.1. Pourquoi faire une veille sur les accès sécurisés ?

Raisons clés	Conséquences en cas de négligence
Nouvelles vulnérabilités fréquentes	Exploitation de failles non corrigées
Nouveaux outils et techniques (Zero Trust)	Retard dans l'adoption des bonnes pratiques
Évolution des attaques (Brute force, MFA)	Compromission des accès
Dépréciation de protocoles obsolètes	Incompatibilité ou exposition accrue

🔹 10.2. Protocoles concernés par la veille

Protocole	Usage courant	Risques associés
SSH	Administration à distance	Attaques par dictionnaire, faille OpenSSH
RDP	Accès distant Windows	Exploits, vulnérabilités RDP BlueKeep
VPN (OpenVPN, WireGuard)	Connexions sécurisées	Vol d'identifiants, mauvaise config
Telnet (obsolète)	Ancien accès distant	Données en clair, à proscrire

🔹 10.3. Outils et plateformes de veille recommandés

📰 Sites officiels & bases de vulnérabilités

- CVE
- NIST NVD
- CERT-FR (ANSSI)
- Debian Security
- Red Hat Security Advisories

🛠️ Suivi logiciel spécifique

- GitHub Releases : `openssh`, `wireguard`, `fail2ban`
- Blogs techniques : Rapid7, Tenable, Qualys

🔔 Services d'alertes / abonnements

- **RSS feed CVE** (ex : CVE Details)
- **Email alertes CERT-FR**
- **Security Weekly Newsletter**

🔹 10.4. Méthodologie de veille hebdomadaire

Étape	Outil / Méthode
1. Parcourir les alertes	CERT-FR, CVE RSS, GitHub Watch
2. Identifier les impacts	Vérifier si le service utilisé est concerné
3. Noter et classer	Utiliser un tableau Excel, Notion, ou SIEM
4. Tester en pré-prod	Réaliser les patchs dans un environnement dédié
5. Diffuser en interne	Mail sécurité, ticket Jira, documentation

🔹 10.5. Sujets clés à surveiller en 2025

Sujet	Pourquoi c'est important
📌 Authentification sans mot de passe (FIDO2)	Réduction des risques liés au phishing
📌 Protocoles quantiques et post-quantiques	Préparation à la crypto post-RSA
📌 MFA obligatoire pour SSH	Devenu standard pour les infrastructures sensibles
📌 Vulnérabilités OpenSSH	Nombreuses CVE annuelles
📌 VPN minimalistes (WireGuard)	Plus rapide et sécurisé qu'OpenVPN

🔹 10.6. Exploitation des informations de veille

Exemple :

Découverte de la CVE-2024-29845 sur OpenSSH (vulnérabilité d'élévation de privilège sur Ubuntu 22.04).

✅ Étapes :

1. Lire le **bulletin CVE**.
2. Identifier l'environnement concerné.
3. Rechercher la version locale : `ssh -V`
4. Planifier le patch en pré-production.
5. Mettre à jour et documenter.

🧪 Cas pratique : Mettre en place une veille technologique sur SSH

Objectif :

Créer une **routine hebdomadaire** de veille et rédiger un **bulletin de synthèse**.

Étapes :

1. S'abonner aux alertes CVE contenant `OpenSSH`.
2. Utiliser un lecteur RSS (Feedly) pour suivre :
   - NVD
   - CERT-FR
   - GitHub OpenSSH
3. Classer les alertes importantes.
4. Rédiger une fiche comme suit :

```
🛡 Bulletin de veille – Semaine 18/2025

📌 Protocole concerné : OpenSSH
📄 Vulnérabilité : CVE-2025-12345
🗓 Date de publication : 29 avril 2025
📝 Description : Faille de contournement d'authentification via une mauvaise validation d'identité sur les versions <8.9
✅ Impact : Serveurs exposés à distance sans MFA
🛠 Recommandation : Upgrade vers OpenSSH 8.9p2 + activer MFA
```

🔹 11.1. Pourquoi sécuriser les échanges de données ?

Menace	Risque associé
Interception (sniffing)	Vol de données sensibles
Altération de contenu	Modification des informations
Usurpation d'identité	Accès non autorisé à des systèmes
Non-conformité réglementaire	Sanctions légales, réputation entachée

🔐 Les échanges doivent être :

- **Chiffrés**
- **Authentifiés**
- **Traçables**

🔹 11.2. Les normes de sécurité de référence

Norme	Objectif principal
ISO/IEC 27001	Système de management de la sécurité de l'information
ISO/IEC 27002	Bonnes pratiques de sécurité (contrôles techniques)
RFC 5280	Norme des certificats X.509 pour SSL/TLS
RFC 8446	Spécification du protocole TLS 1.3
eIDAS (UE)	Réglementation sur l'identification électronique
RGPD (UE)	Obligation de protection des données personnelles
PCI-DSS	Sécurité des paiements (obligatoire pour les banques)
ANSSI – RGS	Référentiel Général de Sécurité (France)

🔹 11.3. Protocoles d'échange sécurisé

Protocole	Utilisation	Sécurisé ?	Recommandé ?
HTTPS	Web sécurisé	✅ Oui	✅ Oui
FTPS / SFTP	Transfert de fichiers	✅ Oui	✅ Oui
SMTP + TLS	Email sécurisé	✅ Partiel	✅ Oui
SSH	Accès distant chiffré	✅ Oui	✅ Oui
IPsec / VPN	Tunnel réseau chiffré	✅ Oui	✅ Oui
Telnet / FTP	Ancien protocole non chiffré	❌ Non	❌ À éviter

🔹 11.4. Gestion des certificats pour sécuriser les échanges

- **Certificats X.509** : utilisés pour HTTPS, VPN, signatures numériques.
- Éléments clés :
  - 🔐 Clé publique / privée
  - 🏢 Autorité de Certification (CA)
  - 📅 Date de validité
  - ✍ Signature numérique
- **Outils associés** :
  - OpenSSL, Let's Encrypt, Certbot
  - Autorités : DigiCert, GlobalSign, Sectigo

🔹 11.5. Intégrité, chiffrement et authentification

Mécanisme	Description	Exemple
Chiffrement	Protéger les données pendant le transport	TLS 1.3
Hashing	Vérifier l'intégrité	SHA-256
Signature numérique	Garantir l'authenticité du message	GPG, X.509
Authentification mutuelle	Vérifier client et serveur	Mutual TLS

🔹 11.6. Bonnes pratiques pour les échanges sécurisés

✍ Utiliser **TLS 1.3 ou supérieur**
✍ Forcer le **HTTPS** avec redirection
✍ Activer le **HSTS** (HTTP Strict Transport Security)
✍ Automatiser le renouvellement des certificats
✍ Vérifier la **chaîne de certification complète**
✍ Mettre en place **l'authentification forte (MFA)**
✍ Bloquer les protocoles obsolètes (SSL 3, TLS 1.0/1.1)

🧪 Cas pratique : sécurisation d'un service web avec HTTPS + conformité TLS

Contexte : Vous devez sécuriser un site web sous Apache sur Debian en appliquant les standards TLS modernes et une politique de sécurité conforme à l'ANSSI.

Étapes :

1. **Générer un certificat avec Let's Encrypt :**

   ```
   sudo apt install certbot python3-certbot-apache
   sudo certbot --apache
   ```

2. **Forcer TLS 1.3 uniquement :**
   Dans `/etc/apache2/mods-enabled/ssl.conf` :

   ```
   SSLProtocol -all +TLSv1.3
   SSLCipherSuite TLS_AES_256_GCM_SHA384
   ```

3. **Activer HSTS :**
   Dans le vhost :

   ```
   Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
   ```

4. **Redémarrer Apache :**

   ```
   sudo systemctl restart apache2
   ```

5. **Vérifier la conformité via SSL Labs** :
   https://www.ssllabs.com/ssltest/

Module 12 : Révision des règles d'authentification

🔹 12.1. Pourquoi réviser les règles d'authentification ?

Risque constaté	Impact potentiel
Mots de passe faibles / réutilisés	Intrusion, compromission des comptes
Partage de comptes	Pertes de traçabilité
Absence de MFA	Accès facilité même après vol d'identifiants
Absence de rotation	Persistance d'un accès en cas de fuite

☞ La révision des règles vise à **adapter la sécurité aux menaces actuelles**, tout en garantissant **l'expérience utilisateur**.

🔹 12.2. Principes d'une authentification sécurisée

Principe	Exemples concrets
🔐 **Facteurs d'authentification**	Mot de passe + code SMS + empreinte digitale
🔄 **Rotation périodique**	Changement tous les 90 jours (selon contexte)
❌ **Blocage après échecs**	5 tentatives → compte temporairement verrouillé
✅ **Authentification multifactorielle (MFA)**	OTP, clé FIDO2, app mobile
📄 **Auditabilité**	Journalisation des connexions

🔻 12.3. Recommandations officielles (ANSSI, NIST, ISO)

Référence	Recommandation clé
ANSSI – Guide d'hygiène informatique	Ne pas utiliser le mot de passe seul
NIST SP 800-63B	Éviter les règles de complexité inutiles
ISO/IEC 27001	Mise en place de politiques d'accès cohérentes
CIS Controls v8	MFA pour tous les accès externes et admins

🔻 12.4. Politique de mot de passe recommandée (exemple)

Élément	Valeur conseillée
Longueur minimale	12 caractères
Complexité	Facultative, mais passphrase recommandée
Vérification de fuite	Oui (ex. via HaveIBeenPwned, Cracklib)
Expiration automatique	Non (sauf exposition ou incident)
Interdiction des doublons	Historique sur les 5 derniers mots de passe

---

🔻 12.5. Mise en œuvre de MFA

Types de MFA :

- **OTP par email/SMS** (basique, vulnérable aux interceptions)
- **Application mobile (TOTP)** : Google Authenticator, Authy
- **U2F / FIDO2** : Clé physique (ex : YubiKey)
- **Certificat utilisateur** : Authentification forte par clé privée

Où l'appliquer :

Système	MFA recommandé ?
Accès VPN	✅ Obligatoire
Portails administrateurs	✅ Obligatoire
Serveurs SSH	✅ Recommandé
Applications internes	⚠ Selon criticité

🔻 12.6. Audit des accès et des règles d'authentification

Outils pour Linux :

```
# Vérifier les comptes avec shell actif
cat /etc/passwd | grep '/bin/bash'
```

```
# Vérifier les mots de passe expirés
chage -l <nom_utilisateur>

# Lister les connexions SSH récentes
last -a | grep ssh
```

Outils complémentaires :

- **Lynis**
- **Fail2ban**
- **auditd**
- **Active Directory** : rapport d'accès et de MFA

Cas pratique : Révision des règles d'authentification sur un serveur Debian

Contexte : Votre entreprise souhaite renforcer la sécurité des comptes utilisateurs sur un serveur Debian utilisé en production.

Étapes proposées :

1. **Désactiver les comptes inutilisés :**

   ```
   sudo usermod -L <nom_utilisateur>
   ```

2. **Exiger un mot de passe fort à la création :**
   Modifier `/etc/login.defs` :

   ```
   nginx
   PASS_MIN_LEN 12
   ```

3. **Activer la vérification de mots de passe compromis :**

   ```
   sudo apt install libpam-pwquality
   ```

   Modifier `/etc/pam.d/common-password` :

   ```
   ruby
   password requisite pam_pwquality.so retry=3 minlen=12 ucredit=-1 lcredit=-1
   dcredit=-1
   ```

4. **Installer MFA via Google Authenticator :**

   ```
   sudo apt install libpam-google-authenticator
   google-authenticator
   ```

   Activer dans `/etc/pam.d/sshd` et `/etc/ssh/sshd_config` :

   ```
   nginx
   ChallengeResponseAuthentication yes
   AuthenticationMethods publickey,keyboard-interactive
   ```

5. **Redémarrer SSH et tester.**

Module 13 : Risques liés aux systèmes déployés

🔹 13.1. Introduction aux risques liés aux systèmes déployés

Catégorie de risque	Exemples	Impact potentiel
Risque technique	Vulnérabilités logiciels, mauvaise configuration	Exploitation des failles, intrusion dans le système
Risque humain	Erreurs de configuration, mauvaise gestion des accès	Compromission des comptes, accès non autorisés
Risque opérationnel	Mises à jour manquantes, absence de tests	Dégradation des services, non-respect des SLA
Risque organisationnel	Absence de procédure de gestion des incidents	Perte de contrôle, erreurs dans la gestion des incidents

---

🔹 13.2. Risques associés à l'architecture des systèmes

Les choix architecturaux ont un **impact majeur** sur la sécurité des systèmes.

Risque d'architecture réseau mal sécurisée :

- **Exemples** : absence de segmentation réseau, pare-feu mal configuré, ouverture excessive des ports.
- **Conséquences** : accès non contrôlé entre systèmes, propagation rapide d'une attaque.

Risque lié à l'infrastructure Cloud :

- **Exemples** : mauvaise gestion des privilèges d'accès, absence de chiffrement des données au repos, configuration incorrecte des groupes de sécurité.
- **Conséquences** : fuite de données, élévation de privilèges, attaques par escalade des droits.

Risque lié aux services tiers :

- **Exemples** : API non sécurisées, logiciels externes mal configurés, services cloud non surveillés.
- **Conséquences** : compromission de l'intégrité du système, attaques par supply chain (exemple : SolarWinds).

🔹 13.3. Risques associés aux vulnérabilités logicielles

Vulnérabilités de logiciels non patchés :

- **Exemples** : Serveurs web non mis à jour, logiciels anciens (PHP, Apache), systèmes non patchés.
- **Conséquences** : Exploitation des failles de sécurité, vol de données, prise de contrôle des systèmes.

Risque lié aux logiciels obsolètes :

- **Exemples** : utilisation de versions anciennes de logiciels (ex. OpenSSL, SSH).
- **Conséquences** : exposition aux failles anciennes qui ne sont plus corrigées.

Risque des erreurs de configuration :

- **Exemples** : permissions trop larges sur des fichiers ou répertoires sensibles, services mal configurés.
- **Conséquences** : accès non autorisé à des ressources sensibles, corruption de données.

🔹 13.4. Risques liés à la gestion des identités et des accès

Risque d'accès non autorisé :

- **Exemples** : comptes partagés, mots de passe faibles ou réutilisés, absence de gestion des comptes utilisateurs.
- **Conséquences** : accès illégitime aux systèmes, compromission des informations sensibles.

Risque d'élévation de privilèges :

- **Exemples** : mauvaise gestion des permissions, erreurs dans les configurations de rôle (RBAC).
- **Conséquences** : prise de contrôle de services critiques, élévation de privilèges pour des utilisateurs malveillants.

Risque lié à l'authentification faible :

- **Exemples** : absence de MFA, gestion de mots de passe inadéquate.
- **Conséquences** : vol de credentials, compromission des comptes utilisateurs.

🔹 13.5. Risques liés aux mises à jour et à la gestion des versions

Risque d'interruption de service :

- **Exemples** : installation d'un patch non testé, mises à jour automatiques sans vérification.
- **Conséquences** : incompatibilité de version, panne de service, indisponibilité de l'application.

Risque d'introduction de nouvelles vulnérabilités :

- **Exemples** : ajout de nouvelles fonctionnalités sans tests de sécurité approfondis.
- **Conséquences** : introduction de failles de sécurité dans un système autrement sécurisé.

Risque d'incompatibilité entre versions :

- **Exemples** : mise à jour d'un serveur web incompatible avec la version d'une application ou d'une bibliothèque.
- **Conséquences** : pannes de services, instabilité du système.

🔹 13.6. Risques physiques et environnementaux

Risque d'attaque physique :

- **Exemples** : vol de matériel, accès physique non autorisé au serveur.
- **Conséquences** : perte de données, modification malveillante du matériel ou du logiciel.

Risque lié à la défaillance de l'infrastructure physique :

- **Exemples** : panne de serveur, défaillance de l'alimentation ou du refroidissement.
- **Conséquences** : perte de données, interruption de service prolongée.

🔹 13.7. Méthodologie de gestion des risques sur les systèmes déployés

1. **Identification des risques** :
   - Utiliser des outils d'audit de sécurité comme **Lynis**, **OpenVAS**, **Nessus** pour identifier les vulnérabilités.
   - Mener des **tests de pénétration** pour détecter les failles exploitables.
2. **Évaluation des risques** :
   - Analyser la **probabilité** et **l'impact** des risques.
   - Utiliser des matrices de risque pour déterminer les priorités.
3. **Mise en place de mesures correctives** :
   - Appliquer des **patches de sécurité** immédiatement après leur publication.
   - Réaliser des **tests de sécurité** pour valider la solidité du système.
4. **Suivi continu** :
   - Mettre en place des outils de **monitoring** (ex. : **Nagios**, **Zabbix**, **Prometheus**).
   - Analyser les **logs** pour détecter des comportements suspects.

🧪 Cas pratique : Analyse des risques d'un système déployé en production

Contexte : Vous gérez un serveur web Debian exposé à Internet, et vous devez réaliser une analyse des risques.

Étapes proposées :

1. **Identification des risques** :
   - Scan de vulnérabilités avec **OpenVAS** et **Lynis**.
   - Vérification des configurations via **sslyze** (analyse de la configuration SSL/TLS).
2. **Évaluation des risques** :
   - Identifier les vulnérabilités critiques (ex. OpenSSH exposé sans mise à jour, ancien PHP).
   - Prioriser les risques en fonction de leur impact potentiel sur la confidentialité, l'intégrité et la disponibilité des services.
3. **Mesures correctives** :
   - Mettre à jour PHP et OpenSSH.
   - Configurer un pare-feu avec **ufw** pour limiter les accès externes.
   - Implémenter une politique de **MFA** sur l'accès SSH.
4. **Suivi** :
   - Configurer **Fail2ban** pour bloquer les tentatives d'accès bruteforce.
   - Mettre en place des alertes de sécurité pour surveiller les tentatives de connexion.

Module 14 : Intégration des règles de sécurité dans les systèmes à déployer

---

🔹 14.1. Introduction à l'intégration de la sécurité dans les systèmes à déployer

Intégrer des règles de sécurité dans les systèmes à déployer est essentiel pour éviter les vulnérabilités qui pourraient compromettre l'intégrité, la confidentialité et la disponibilité du système.

- **Objectif** : Minimiser les risques de sécurité dès le début du cycle de développement, afin de garantir une protection continue.
- **Principe fondamental** : La **sécurité dès la conception** (Security by Design) doit être appliquée à tous les niveaux du système, depuis l'architecture réseau jusqu'à la gestion des utilisateurs et des accès.

🔷 14.2. Phases de l'intégration de la sécurité dans les systèmes

1. Phase de conception et d'architecture

- **Objectif** : Assurer la sécurité des systèmes dès leur conception.
- **Actions** :
  - Choisir une architecture **sécurisée** : utilisation de **zones démilitarisées (DMZ)**, séparation des services critiques, segmentation réseau.
  - Implémenter des mécanismes de **contrôle d'accès**, de **chiffrement des communications** (SSL/TLS).
  - Définir une **gestion des identités** claire et intégrer des **politiques de mot de passe sécurisé** et de **MFA**.

2. Phase de développement

- **Objectif** : Intégrer des règles de sécurité dans le code dès le développement.
- **Actions** :
  - Effectuer une **revue de code sécurisée** pour identifier des vulnérabilités comme des injections SQL, des XSS, ou des failles dans la gestion des sessions.
  - Utiliser des outils d'analyse de sécurité du code comme **SonarQube**, **Checkmarx**, ou **Fortify**.
  - **Chiffrer les données sensibles** dans le code source (ex. : chiffrement des clés API, des mots de passe).
  - Pratiquer le **développement sécurisé (Secure Coding)** en respectant les principes de la **OWASP** (Open Web Application Security Project).

3. Phase de test et pré-production

- **Objectif** : Tester le système dans un environnement de pré-production pour identifier les risques de sécurité.
- **Actions** :
  - Mettre en place des **tests de pénétration** sur l'application, l'infrastructure et les interfaces utilisateur.
  - Utiliser des outils comme **OWASP ZAP**, **Burp Suite**, ou **Nessus** pour identifier les vulnérabilités courantes.
  - Vérifier la conformité avec les **règles de sécurité** : audit des accès, des permissions, et des configurations des systèmes.
  - Tester la **résilience** du système contre les attaques DDoS, les injections, et autres menaces communes.

4. Phase de déploiement

- **Objectif** : Déployer le système tout en respectant les règles de sécurité.
- **Actions** :
  - Appliquer des **politiques de configuration sécurisée** (pare-feu, gestion des mises à jour automatiques, paramètres de sécurité des applications).
  - Utiliser des outils d'**infrastructure as code (IaC)** comme **Ansible**, **Terraform**, ou **Puppet** pour automatiser le déploiement sécurisé.
  - Configurer des mécanismes de **contrôle d'accès** (pare-feu, VPN, règles d'accès réseau).
  - S'assurer que les systèmes sont configurés pour **loguer** toutes les actions de sécurité (audit logs) pour les analyser régulièrement.

5. Phase de maintenance

- **Objectif** : Maintenir la sécurité du système tout au long de son cycle de vie.
- **Actions** :

- Mettre en place un **plan de gestion des incidents de sécurité** : suivi des alertes, gestion des accès en cas de détection d'une brèche.
- Appliquer régulièrement les **patches de sécurité** (mise à jour des systèmes et applications).
- Implémenter des **tests de sécurité continus** dans l'environnement de production pour garantir que les nouveaux changements n'introduisent pas de vulnérabilités.

🔹 14.3. Bonnes pratiques pour l'intégration de la sécurité

Principes clés à respecter :

1. **Moindre privilège** : Ne donner aux utilisateurs et services que les permissions strictement nécessaires.
2. **Séparation des responsabilités** : Différencier les rôles et responsabilités pour minimiser les erreurs humaines.
3. **Audits réguliers** : Vérifier périodiquement la conformité des systèmes avec les standards de sécurité (ex. : audits de sécurité, scans de vulnérabilités).
4. **Chiffrement des données sensibles** : Utiliser des technologies de chiffrement pour les données sensibles stockées et en transit (ex. : SSL/TLS, AES-256).
5. **Sécurisation de l'accès réseau** : Mettre en œuvre des pare-feu, des VPN, et des contrôles d'accès granulaires.
6. **Formation des utilisateurs** : Former les utilisateurs à la sécurité pour éviter les erreurs humaines telles que la mauvaise gestion des mots de passe.

🔹 14.4. Outils pour l'intégration de la sécurité

Outil	Utilisation
Ansible / Puppet	Automatisation des configurations de sécurité sur les serveurs
Docker	Sécurisation des conteneurs, intégration des règles de sécurité au niveau de l'image
OWASP ZAP	Scanner de vulnérabilités pour les applications web
Fail2ban	Protection contre les attaques par force brute
Snort	Système de détection des intrusions (IDS)
Kali Linux	Plateforme pour les tests de pénétration et l'audit de sécurité
OpenSCAP	Outil d'audit de sécurité basé sur les standards de sécurité (ex. : DISA STIG)

🔹 14.5. Cas pratique : Intégration des règles de sécurité dans un système de gestion de contenu (CMS)

Contexte : Vous devez déployer un système de gestion de contenu (CMS) comme **WordPress** sur un serveur Debian. Vous devez intégrer les règles de sécurité suivantes avant de mettre le CMS en production.

Étapes proposées :

1. **Phase de conception** :
   - Sélectionner une architecture sécurisée pour le CMS : utilisation de **TLS** pour chiffrer les communications et segmentation du réseau (ex. : base de données sur un sous-réseau séparé).

- Mettre en place une politique d'**authentification forte (MFA)** pour les administrateurs du CMS.

2. **Phase de développement** :
   - Auditer le code du CMS et ajouter des règles de **protection contre les injections SQL** et les attaques XSS.
   - Ajouter des contrôles sur les permissions des utilisateurs, en appliquant le principe du **moindre privilège**.
3. **Phase de test** :
   - Tester le CMS en utilisant des outils comme **OWASP ZAP** pour analyser les vulnérabilités d'application.
   - Vérifier que tous les plugins installés sont à jour et sécurisés.
4. **Phase de déploiement** :
   - Automatiser le déploiement avec **Ansible** et configurer le serveur web pour restreindre l'accès aux fichiers sensibles.
   - Activer **Fail2ban** pour bloquer les tentatives d'attaques par brute force sur les connexions administratives.
5. **Phase de maintenance** :
   - Mettre en place une **surveillance continue** du CMS pour détecter les comportements suspects (ex. : modification non autorisée de fichiers).
   - Planifier les mises à jour régulières du CMS et de ses plugins.

Module 15 : Pare-feux du système

🔹 15.1. Introduction aux pare-feux du système

Un pare-feu (firewall) est une **barrière de sécurité** qui contrôle le flux de données entre deux réseaux, ou entre un réseau et un système, en fonction de règles de sécurité prédéfinies.

- **Objectif** : Le pare-feu a pour objectif de **protéger** le système contre les accès non autorisés tout en permettant la communication légitime.
- **Fonctionnement** : Il peut être déployé à différents niveaux :
  - **Pare-feu de réseau** (ex. : matériel, dispositif séparé entre les réseaux).
  - **Pare-feu de système** (ex. : logiciel, intégré directement dans l'OS).

🔹 15.2. Types de pare-feux du système

1. Pare-feu logiciel

- **Description** : Le pare-feu logiciel s'exécute sur le système d'exploitation de la machine (ex. : iptables sur Linux, Windows Firewall sur Windows).
- **Exemples** :
  - **Linux** : **iptables**, **nftables**, **ufw**.
  - **Windows** : **Windows Defender Firewall**.
- **Caractéristiques** :
  - **Filtrage de paquets** : Examine les paquets entrants et sortants selon des règles définies.
  - **Inspection approfondie (DPI)** : Peut inspecter les paquets pour déterminer le contenu de la communication.
  - **Application des politiques de sécurité** sur les connexions entrantes et sortantes.

2. Pare-feu matériel

- **Description** : Placé entre le système et le réseau pour filtrer les connexions avant qu'elles n'atteignent les systèmes internes.
- **Exemples** :
  - **Firewall appliance** : Cisco ASA, pfSense.
  - **Firewalls d'entreprise** : Fortinet, Palo Alto Networks.
- **Caractéristiques** :
  - Permet un filtrage réseau à grande échelle.
  - Inclut souvent des fonctionnalités avancées telles que la détection des intrusions (IDS), le filtrage d'URL, etc.

3. Pare-feu de type "Next-Generation Firewall" (NGFW)

- **Description** : Pare-feu évolué offrant des fonctionnalités supplémentaires par rapport aux pare-feux traditionnels, telles que l'inspection SSL, l'identification des applications, et la gestion des menaces.
- **Exemples** : Palo Alto, Cisco Firepower.
- **Caractéristiques** :
  - Inspection des **protocoles avancés** (HTTPS, DNS, etc.).
  - Contrôle d'accès basé sur l'application (Application Layer Filtering).
  - Intégration avec des outils de **détection d'intrusion** et d'**analyse comportementale**.

🔹 15.3. Concepts fondamentaux des pare-feux

1. Filtrage de paquets

Le filtrage de paquets est le processus par lequel le pare-feu examine les paquets qui passent par le système et décide de les accepter ou de les bloquer en fonction des règles.

- **Critères de filtrage** :
  - Adresse **IP source**.
  - Adresse **IP destination**.
  - **Port source** et **port destination**.
  - **Protocole** (TCP, UDP, ICMP, etc.).

2. Politiques de filtrage

Les règles de filtrage définissent ce qui peut ou ne peut pas entrer ou sortir d'un réseau. Ces règles peuvent être :

- **Acceptation** (ALLOW) : Permet la communication.
- **Refus** (DENY) : Bloque la communication.
- **Filtrage explicite** : Autorisation ou refus spécifique d'une connexion selon des critères.
- **Filtrage implicite** : Par défaut, les connexions non explicitement autorisées ou refusées sont bloquées.

3. Inspection de l'état des connexions (Stateful Inspection)

Cette méthode de filtrage prend en compte l'état de la connexion pour autoriser ou bloquer un paquet. Par exemple, si une connexion a été établie (handshake TCP), les paquets suivants liés à cette connexion sont automatiquement autorisés.

🔹 15.4. Configuration des pare-feux sous différents systèmes

1. Pare-feu sous Linux (iptables / nftables / ufw)

1.1. iptables (utilisé sur les systèmes Linux classiques) :

- **Configuration de base** :
  - **Afficher les règles** : `sudo iptables -L`
  - **Ajouter une règle** : `sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT` (Autoriser le port 80 pour HTTP)
  - **Bloquer une IP** : `sudo iptables -A INPUT -s 192.168.1.100 -j DROP`
  - **Enregistrer les règles** : `sudo iptables-save > /etc/iptables/rules.v4`

1.2. nftables (successeur d'iptables) :

- Utilisation d'un **fichier de configuration** pour appliquer les règles.
- **Commandes** :
  - Créer une règle pour autoriser le HTTP : `nft add rule inet filter input tcp dport 80 accept`
  - Sauvegarder la configuration : `nft list ruleset > /etc/nftables.conf`

1.3. UFW (Uncomplicated Firewall) :

- Interface simplifiée pour **iptables**.
  - **Activer UFW** : `sudo ufw enable`
  - **Autoriser un port** : `sudo ufw allow 80/tcp`
  - **Vérifier les règles** : `sudo ufw status`

2. Pare-feu sous Windows (Windows Defender Firewall)

- **Activer / Désactiver le pare-feu** :
  - `Control Panel > System and Security > Windows Defender Firewall`
- **Ajouter une règle** :
  - Dans le panneau de configuration, **"Advanced Settings"** permet de créer des règles entrant/sortant selon le port, le protocole ou l'adresse IP.
- **Paramétrage via PowerShell** :
  - **Activer une règle pour le port 80** :

```
New-NetFirewallRule -DisplayName "HTTP" -Direction Inbound -Protocol TCP -LocalPort 80 -Action Allow
```

🔹 15.5. Bonnes pratiques de configuration des pare-feux

1. **Principle of Least Privilege** :
   - Ne permettre que les connexions nécessaires pour le fonctionnement du système.
   - Bloquer par défaut tout le trafic, et autoriser explicitement ce qui est nécessaire.
2. **Utilisation de zones de confiance** :
   - Segmenter le réseau en **zones de sécurité** (ex. DMZ pour les serveurs web, zone interne pour les services critiques).
3. **Filtrage basé sur l'état des connexions (Stateful Inspection)** :
   - Permet de simplifier les règles et d'améliorer la sécurité en validant l'état des connexions.
4. **Logging et surveillance** :
   - Configurer les **logs** des pare-feux pour surveiller les événements de sécurité.

- Intégrer ces logs dans un outil de gestion des incidents (ex. **SIEM**).

5. **Test et validation des règles** :
   - Tester les règles avec des outils comme **nmap** pour vérifier l'exposition du système.
   - Utiliser des outils de **test de pénétration** pour valider la configuration des pare-feux.

🔹 15.6. Cas pratique : Configuration d'un pare-feu avec iptables

Contexte : Vous êtes responsable de la sécurité d'un serveur web. Vous devez configurer un pare-feu pour autoriser uniquement les connexions nécessaires.

Étapes proposées :

1. **Configurer iptables** pour autoriser uniquement le trafic HTTP (port 80) et HTTPS (port 443) :
   - `sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT`
   - `sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT`
2. **Bloquer tout le reste** :
   - `sudo iptables -A INPUT -j DROP`
3. **Enregistrer les règles** pour qu'elles persistent après un redémarrage :
   - `sudo iptables-save > /etc/iptables/rules.v4`
4. **Vérifier la configuration** :
   - `sudo iptables -L`
   -

Module 16 : Pare-feux réseau

🔹 16.1. Introduction aux pare-feux réseau

Un **pare-feu réseau** est un dispositif qui se place généralement entre un réseau interne et un réseau externe (comme Internet), et qui a pour fonction de contrôler les connexions et de filtrer les paquets qui passent entre ces deux réseaux en fonction de règles préétablies. Contrairement au pare-feu du système (exécuté sur chaque machine), un pare-feu réseau protège plusieurs machines à la fois, souvent au niveau du périmètre du réseau.

- **Objectif principal** : Contrôler le trafic réseau pour éviter l'accès non autorisé aux ressources internes tout en permettant les connexions légitimes.
- **Typologie** : Un pare-feu réseau peut être matériel ou logiciel, et peut être intégré dans des appliances dédiées ou dans des systèmes virtualisés.

🔹 16.2. Types de pare-feux réseau

1. Pare-feu matériel (appliance)

- **Description** : Un pare-feu matériel est un appareil dédié qui se place en frontière du réseau, entre le réseau interne et l'Internet ou un autre réseau externe.
- **Exemples de produits** :
  - **Cisco ASA**
  - **FortiGate** de Fortinet
  - **Palo Alto Networks**

- **Caractéristiques** :
  - Fournit une **protection robuste** grâce à des **politiques de filtrage très détaillées**.
  - Peut inclure des fonctionnalités avancées comme la **detection d'intrusions (IDS/IPS)**, des **analyses de protocoles** (Deep Packet Inspection, DPI), et la gestion des **VPN**.
  - Adapté aux grandes entreprises ou aux environnements à fort trafic.

2. Pare-feu logiciel

- **Description** : Un pare-feu logiciel est une solution basée sur un serveur ou un dispositif virtualisé qui exécute un logiciel de filtrage du trafic réseau.
- **Exemples** :
  - **pfSense** : Un pare-feu et routeur open-source très populaire basé sur FreeBSD.
  - **ipFire** : Solution open-source basée sur Linux.
  - **Untangle** : Solution open-source et commerciale de pare-feu et filtrage.
- **Caractéristiques** :
  - Moins coûteux que les pare-feux matériels, mais tout aussi efficaces pour des réseaux de taille moyenne.
  - Peut être déployé sur du matériel standard ou dans des environnements virtualisés.
  - Facile à personnaliser et à intégrer dans des réseaux de petite et moyenne taille.

3. Pare-feu de nouvelle génération (NGFW)

- **Description** : Les NGFW intègrent des fonctionnalités supplémentaires par rapport aux pare-feux traditionnels, comme la détection d'intrusions (IDS), la gestion des applications, et l'inspection des paquets chiffrés (SSL/TLS).
- **Exemples** :
  - **Palo Alto Networks NGFW**
  - **Cisco Firepower NGFW**
  - **FortiGate NGFW**
- **Caractéristiques** :
  - Inspection plus approfondie des paquets.
  - Gestion des **applications** et contrôle d'accès basé sur le **contenu** et le **comportement**.
  - **Inspection des flux SSL/TLS** et protection contre des attaques plus sophistiquées.

🔹 16.3. Fonctionnement des pare-feux réseau

1. Filtrage de paquets

Le filtrage de paquets est l'action de vérifier les informations contenues dans les paquets entrants et sortants du réseau (adresse IP, port, protocole) pour décider de leur autorisation ou de leur blocage.

- **Critères utilisés** :
  - **Adresse IP source** : D'où provient le paquet ?
  - **Adresse IP de destination** : Où le paquet est-il destiné ?
  - **Numéro de port** : Quel service est sollicité (HTTP, FTP, etc.) ?
  - **Protocole** : Quel type de communication est utilisé (TCP, UDP, ICMP) ?

2. Inspection approfondie des paquets (Deep Packet Inspection - DPI)

Cette méthode permet d'analyser le contenu des paquets, au-delà des simples informations d'en-tête, pour détecter des menaces comme les malwares, les tentatives d'intrusion, ou les protocoles non autorisés.

- **Exemples de protocoles analysés** : HTTP, DNS, FTP, SMTP.
- **Avantages** : Permet de bloquer des attaques avancées, telles que les attaques par **exploitation de vulnérabilités**, les attaques de **reconnaissance** (scanning), et les **données malveillantes dissimulées**.

3. Inspection des connexions état

Les pare-feux modernes utilisent l'inspection des états des connexions (Stateful Inspection) pour suivre l'état des connexions réseau et décider si un paquet appartient à une connexion déjà établie ou s'il doit être rejeté.

- **Avantages** :
  - Permet une analyse plus fine du trafic, en considérant non seulement les paquets, mais aussi leur contexte dans le cadre d'une connexion en cours.
  - Par exemple, si une connexion TCP a été établie, tous les paquets associés sont automatiquement autorisés.

🔷 16.4. Architecture et déploiement des pare-feux réseau

1. Pare-feu de périmètre

Le **pare-feu de périmètre** protège l'entrée/sortie d'un réseau entier, et il est généralement placé entre un réseau interne et un réseau externe (Internet). Il agit comme une **barrière principale** entre un réseau privé et l'extérieur.

- **Architecture typique** :
  - **Internet → Pare-feu → DMZ (zone démilitarisée) → Réseau interne**
  - **DMZ** : Zone contenant des serveurs exposés au public (ex. : serveur web, serveur de messagerie).

2. Pare-feu interne

Le **pare-feu interne** est utilisé pour segmenter les différents segments d'un réseau interne. Il permet de contrôler le trafic entre les **différentes zones de confiance** du réseau de l'entreprise.

- **Exemples de segmentation** :
  - **Réseau administratif → Pare-feu interne → Réseau de développement**
  - **Réseau de production → Pare-feu interne → Réseau de tests**

3. Architecture multi-niveau (Zero Trust)

Dans une architecture **Zero Trust**, chaque segment de réseau est protégé de manière isolée, même à l'intérieur de l'organisation, et le principe de **vérification à chaque tentative d'accès** est appliqué. Chaque communication est filtrée, quel que soit le point de départ ou de destination.

🔷 16.5. Configuration des règles de filtrage réseau

1. Contrôle d'accès basé sur les adresses IP

Les règles de filtrage peuvent être configurées pour autoriser ou refuser l'accès en fonction des adresses IP source ou de destination. Cela permet de restreindre l'accès au réseau à des **zones géographiques** spécifiques ou à des **IP de confiance**.

2. Contrôle d'accès par ports et protocoles

Les pare-feux peuvent également bloquer ou autoriser des connexions en fonction des **ports** (ex. : port 80 pour HTTP, port 443 pour HTTPS) et des **protocoles** utilisés.

3. Contrôle d'accès basé sur des critères d'application

Les pare-feux modernes permettent de filtrer le trafic basé sur des critères plus complexes, comme les **applications utilisées** (HTTP, FTP, DNS, etc.), et peuvent appliquer des règles spécifiques aux protocoles applicatifs.

🔹 16.6. Meilleures pratiques pour la gestion des pare-feux réseau

1. **Principle of Least Privilege** :
   - Ne permettre que les connexions nécessaires. Refuser par défaut tous les autres types de connexions.
2. **Segmentation du réseau** :
   - Séparer les zones sensibles du réseau avec des pare-feux internes pour limiter l'impact des intrusions.
   - Utiliser des VLANs et des sous-réseaux pour segmenter les ressources internes.
3. **Gestion des mises à jour** :
   - Assurez-vous que les **firmwares** et les **logiciels de pare-feu** sont régulièrement mis à jour pour se protéger contre les vulnérabilités connues.
4. **Logging et surveillance continue** :
   - Activez les journaux de bord (logs) pour enregistrer tous les événements liés à la sécurité.
   - Intégrez les logs dans des systèmes de **SIEM** (Security Information and Event Management) pour détecter toute activité suspecte.
5. **Test et validation réguliers** :
   - Effectuez des tests de pénétration réguliers pour vérifier l'efficacité des règles de filtrage et détecter des failles de sécurité.

🔹 16.7. Cas pratique : Configuration d'un pare-feu réseau avec pfSense

Contexte : Vous devez configurer un pare-feu pour protéger une infrastructure réseau d'entreprise avec pfSense, en implémentant des règles de filtrage pour sécuriser les accès aux serveurs internes.

Étapes proposées :

1. **Installation de pfSense** sur une machine dédiée ou une VM.
2. **Configuration des interfaces** réseau pour séparer les différentes zones de sécurité (Internet, DMZ, réseau interne).
3. **Création des règles de filtrage** pour autoriser uniquement les connexions HTTP/HTTPS sur la DMZ.
4. **Mise en place de VPN** pour permettre aux employés distants de se connecter en toute sécurité au réseau interne.
5. **Test du filtrage** avec des outils comme **nmap** ou **ping** pour vérifier que seuls les ports nécessaires sont ouverts.

Module 17 : Introduction à OpenSSH

17.1. Introduction à OpenSSH

OpenSSH (Open Secure Shell) est une suite logicielle permettant de sécuriser les communications réseau, notamment les connexions distantes à des serveurs via des protocoles comme **SSH** (Secure Shell), mais aussi des mécanismes comme **SFTP** (SSH File Transfer Protocol) ou **Port forwarding** pour sécuriser des connexions réseaux.

- **SSH** permet une connexion sécurisée en utilisant un canal chiffré, protégeant ainsi l'intégrité et la confidentialité des données échangées entre l'utilisateur et le serveur.
- Il est largement utilisé pour accéder à des machines distantes de manière sécurisée, gérer des serveurs, effectuer des transferts de fichiers, etc.

OpenSSH est l'une des implémentations les plus populaires de SSH et est disponible sur de nombreuses distributions GNU/Linux, ainsi que sur d'autres systèmes comme macOS ou Windows via des clients spécifiques.

17.2. Architecture et fonctionnement de SSH

1. Fonctionnement de SSH

SSH repose sur un modèle **client-serveur** où :

- Le **serveur SSH** est installé sur le système cible, écoutant sur un port (par défaut **port 22**).
- Le **client SSH** est installé sur la machine distante à partir de laquelle l'utilisateur souhaite se connecter au serveur.

Processus de connexion :

- Le client initie une connexion au serveur SSH.
- Le serveur répond en envoyant une **clé publique** pour commencer l'échange sécurisé.
- Le client et le serveur échangent des informations pour établir une session **chiffrée** et **authentifiée**.

2. Types d'authentification SSH

- **Authentification par mot de passe** : L'utilisateur entre son mot de passe pour se connecter. Cette méthode est souvent considérée comme moins sécurisée que l'authentification par clé.
- **Authentification par clé publique/privée** : L'utilisateur génère une paire de clés SSH. La clé publique est copiée sur le serveur, et la clé privée reste sur le client. La clé privée sert à **prouver l'identité** de l'utilisateur sans avoir à envoyer de mot de passe sur le réseau.
- **Authentification par certificat** : Une extension de l'authentification par clé, où un certificat signé par une **autorité de certification (CA)** permet de renforcer la sécurité de l'échange.

17.3. Installation et configuration d'OpenSSH

1. Installation de OpenSSH sur une machine GNU/Linux

Sur **Debian/Ubuntu** :

```
sudo apt update
sudo apt install openssh-server
```

Sur **CentOS/RHEL** :

```
sudo yum install openssh-server
```

Sur **macOS** : OpenSSH est généralement installé par défaut.

2. Vérification du service OpenSSH

Une fois le serveur installé, le service OpenSSH démarre automatiquement. Vous pouvez vérifier que le service SSH fonctionne avec la commande suivante :

```
sudo systemctl status sshd
```

Pour démarrer ou redémarrer le service :

```
sudo systemctl start sshd
sudo systemctl restart sshd
```

3. Configuration du serveur SSH

Le fichier de configuration d'OpenSSH se trouve dans **/etc/ssh/sshd_config**. Il permet de personnaliser des paramètres tels que :

- **Port d'écoute** : Changer le port par défaut (22) pour rendre le serveur moins vulnérable aux attaques automatisées.

```
Port 2222
```

- **Désactivation de l'authentification par mot de passe** : Pour ne permettre que l'authentification par clé publique.

```
PasswordAuthentication no
```

- **Limitation de l'accès à certains utilisateurs ou groupes** : Pour ne permettre l'accès SSH qu'à certains utilisateurs.

```
AllowUsers user1 user2
```

- **Désactivation de la connexion root** : Pour éviter que l'utilisateur root puisse se connecter directement.

```
PermitRootLogin no
```

Après avoir modifié le fichier de configuration, il faut **recharger** la configuration d'OpenSSH :

```
sudo systemctl reload sshd
```

🔷 17.4. Utilisation de SSH pour la connexion distante

1. Connexion à un serveur distant via SSH

Une fois le serveur SSH configuré, vous pouvez vous connecter depuis une machine cliente en utilisant la commande :

```
ssh user@hostname_or_ip
```

- **user** : Nom d'utilisateur sur le serveur distant.
- **hostname_or_ip** : Nom d'hôte ou adresse IP du serveur distant.

Si vous avez configuré un port autre que le port par défaut, vous pouvez spécifier le port avec l'option `-p` :

```
ssh -p 2222 user@hostname_or_ip
```

2. Authentification par clé SSH

L'authentification par clé est plus sécurisée et recommandée. Pour l'utiliser :

- **Générer une paire de clés** :

```
ssh-keygen
```

Cela génère deux fichiers :

- **id_rsa** (clé privée)
- **id_rsa.pub** (clé publique)
- **Copier la clé publique sur le serveur distant** :

```
ssh-copy-id user@hostname_or_ip
```

Cela ajoute votre clé publique au fichier **~/.ssh/authorized_keys** sur le serveur distant.

- Après cela, vous pourrez vous connecter sans mot de passe, en utilisant uniquement la clé privée.

🔻 17.5. Sécurisation d'OpenSSH

1. Utiliser des clés fortes

Il est recommandé d'utiliser des clés de type **RSA** avec une taille d'au moins **2048 bits**, ou de préférer des clés **ED25519**, plus sûres et plus rapides que RSA.

2. Désactiver l'accès root

Il est vivement recommandé de désactiver la connexion directe en tant qu'utilisateur **root** pour minimiser les risques de piratage.

```
PermitRootLogin no
```

3. Changer le port par défaut

Changer le port d'écoute par défaut (port 22) pour rendre votre serveur SSH plus discret face aux attaques automatisées.

```
Port 2222
```

4. Utiliser l'authentification par clé uniquement

Désactiver l'authentification par mot de passe pour éviter les attaques par **brute force**.

```
PasswordAuthentication no
```

5. Limiter l'accès à des adresses IP spécifiques

Il est possible de restreindre l’accès SSH à certaines adresses IP pour améliorer la sécurité du serveur.

```
AllowUsers user1@192.168.1.0/24
```

6. Surveillance et logs

Utilisez les logs d’OpenSSH pour suivre les tentatives de connexion :

```
tail -f /var/log/auth.log
```

🔺 17.6. Avantages et applications d'OpenSSH

- **Sécurisation des communications** : SSH permet de chiffrer les communications entre deux hôtes, assurant ainsi que les informations échangées restent privées.
- **Accès distant sécurisé** : Il est utilisé pour se connecter à des serveurs distants de manière sécurisée.
- **Transfert de fichiers sécurisé** : Avec **SFTP** ou **SCP**, vous pouvez transférer des fichiers entre une machine locale et un serveur distant de manière sécurisée.
- **Tunnelisation (Port Forwarding)** : SSH permet de créer des tunnels sécurisés entre deux machines, ce qui peut être utilisé pour accéder à des services distants via un canal chiffré.

🔺 17.7. Cas pratique : Sécurisation d’un serveur SSH

Contexte : Vous devez sécuriser l'accès à un serveur Ubuntu en utilisant OpenSSH. Vous êtes chargé de configurer le serveur de manière à :

1. Autoriser uniquement les connexions SSH via des clés publiques.
2. Désactiver l’accès SSH direct pour l’utilisateur root.
3. Modifier le port SSH par défaut (22).
4. Limiter les connexions SSH aux utilisateurs autorisés.

Étapes :

1. Installez et configurez OpenSSH.
2. Générez une paire de clés SSH et copiez la clé publique sur le serveur.
3. Modifiez le fichier **/etc/ssh/sshd_config** pour sécuriser le serveur selon les critères ci-dessus.
4. Testez l'accès SSH avec votre clé publique.

📄 Livrables attendus :

- **Rapport de configuration d'OpenSSH**, incluant les modifications appliquées et la justification de ces choix de sécurité.
- **Logs d’accès SSH** montrant la connexion avec des clés SSH.
- **Démarche de sécurisation** du serveur SSH, incluant les bonnes pratiques mises en œuvre.