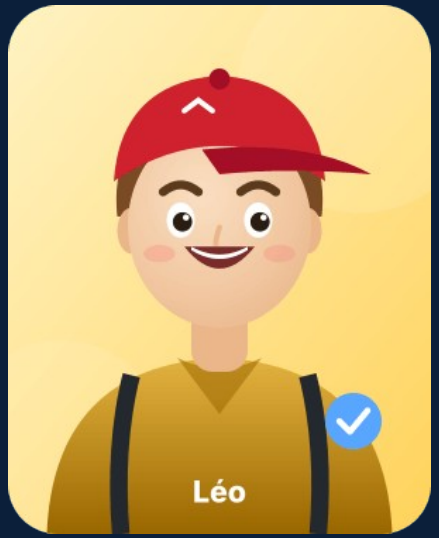
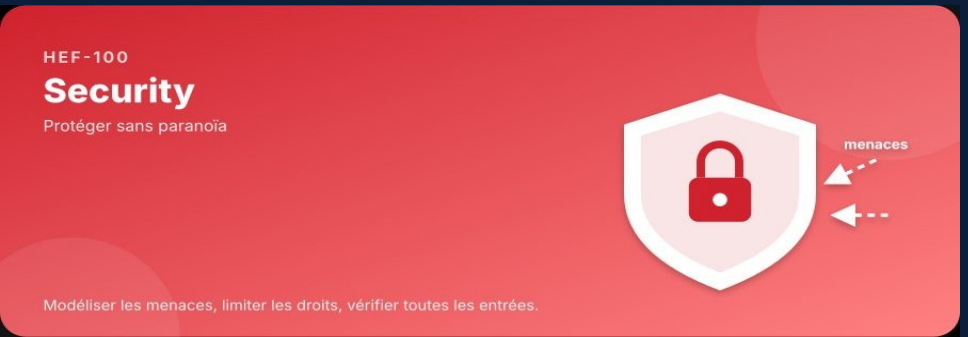
HEF-100 · MISSION 08 / 10

Sécurité réseau et système

Un code irréprochable sur un serveur mal configuré reste vulnérable. Le durcissement ferme les portes inutiles.





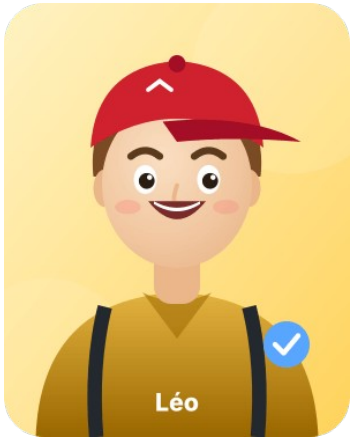
Badge : Durcisseur de systèmes

95 min · intermédiaire

L'histoire

Après l'audit, HashBox découvre un autre problème : le serveur de test faisait tourner SSH avec connexion `root` par mot de passe, une base de données écoutait sur toutes les interfaces, et une bibliothèque vieille de trois ans avait une faille publique connue.

> *« Le code était correct, dit Sam. Mais la machine autour ne l'était pas. La sécurité, c'est aussi la configuration, les mises à jour et les accès. On appelle ça le durcissement (hardening). »*



Tu apprends à auditer et durcir un système : ports, permissions, SSH, TLS, conteneurs, dépendances.

Ta mission

À la fin, je serai capable de…

1. lister les principes du durcissement d'un système
2. auditer ports ouverts, permissions et configuration SSH avec des scripts
3. vérifier une configuration TLS et des paramètres de conteneur
4. contrôler les dépendances vulnérables et la chaîne d'approvisionnement
5. définir une politique de mise à jour

Défi d'échauffement

Défi d'échauffement

Tu emménages dans une maison ancienne. Avant de t'y installer : (1) tu changes les serrures (clés inconnues), (2) tu fermes les pièces inutiles, (3) tu répares ce qui est cassé, (4) tu installes une alarme, (5) tu notes qui a quelle clé. Fais le parallèle avec un serveur : à quoi correspond chaque action ? Quelle action est la plus souvent oubliée ?

Le concept

Le durcissement : principes

Principe	Application
Réduire la surface	désinstaller les services inutiles, fermer les ports non nécessaires
Moindre privilège	pas de `root`, comptes dédiés, droits de fichiers stricts
Configuration sûre par défaut	changer les mots de passe par défaut, désactiver les fonctions risquées
Mises à jour	correctifs de sécurité appliqués rapidement et de façon contrôlée
Segmentation	zones réseau séparées, pare-feu entre elles
Journalisation et supervision	savoir ce qui se passe
Reproductibilité	configuration versionnée (infrastructure as code)

Le concept

Réseau : ports et services

Un **port ouvert** est un point d'entrée potentiel. On compare toujours l'**inventaire attendu** à l'**observé** :

Écart	Signification
ouvert mais non attendu	service oublié ou intrus
attendu mais fermé	panne ou changement non voulu
attendu mais **exposé trop largement**	écoute sur `0.0.0.0` au lieu de `127.0.0.1` ou du réseau privé

⚖ ***Cadre** : n'analyse que **tes propres machines** (ou celles pour lesquelles tu as une autorisation écrite). Une analyse de ports sur un système tiers est illégale dans de nombreux pays.*

Le concept

SSH

Réglage	Recommandation
`PermitRootLogin`	no
`PasswordAuthentication`	no (clés uniquement)
`PermitEmptyPasswords`	no
`X11Forwarding`	no si inutile
`AllowUsers` / `AllowGroups`	liste restreinte
`MaxAuthTries`	faible (3)
Port	changer n'est pas une sécurité, mais réduit le bruit ; ce qui compte : les clés, le pare-feu, la surveillance

Le concept

Permissions du système de fichiers

- Fichiers de configuration avec secrets : `600` (ou `640` avec groupe dédié).
- Aucune écriture pour « tout le monde » (`o+w`) sauf dossiers à bit *sticky* comme `/tmp`.
- Pas de programmes `setuid` inutiles.
- Clés SSH privées en `600`.

TLS : bien configurer

Contrôle	Attendu
Versions	TLS 1.2 minimum, idéalement 1.3 ; SSLv3, TLS 1.0 et 1.1 désactivés
Suites de chiffrement	AEAD (AES-GCM, ChaCha20-Poly1305), échange à confidentialité persistante (ECDHE)

Le concept

TLS : bien configurer (suite)

Contrôle	Attendu
Certificat	valide, non expiré, chaîne complète, nom correct
HSTS	activé
Vérification côté client	**jamais** `verify=False` ou `CERT_NONE` en production

En Python, `ssl.create_default_context()` active par défaut la vérification du certificat et du nom d'hôte.

Conteneurs

Pratique	Raison
Ne pas exécuter en `root` (`USER` non privilégié)	limite l'impact d'une compromission
Images minimales, épinglées par version/digest (pas `latest`)	moins de composants, reproductibilité

Le concept

Conteneurs (suite)

Pratique	Raison
Pas de secrets dans l'image ni dans `ENV`	ils fuient dans les couches
Système de fichiers en lecture seule, capacités retirées	moindre privilège
Analyse des images et des dépendances	détecter les composants vulnérables

Dépendances et chaîne d'approvisionnement

La plupart du code d'une application vient de **tiers**. Risques : bibliothèque vulnérable (A06), paquet malveillant imitant un nom connu (*typosquatting*), compromission d'un mainteneur, dépendance abandonnée.
Mesures :

- **inventaire** (SBOM : liste des composants) ;

Le concept

Dépendances et chaîne d'approvisionnement (suite)

- **verrouiller** les versions (fichier de verrouillage) et vérifier les **empreintes** (`pip install --require-hashes`) ;
- **analyse automatique** des vulnérabilités connues (CVE) en intégration continue ;
- **mises à jour régulières** et revues ;
- limiter les dépendances.

Une politique de correctifs

Gravité	Délai de correction cible
Critique, exploitée activement	24 à 72 h
Haute	7 jours
Moyenne	30 jours
Basse	prochain cycle

Le concept

Une politique de correctifs (suite)

La rapidité vient d'une **chaîne de déploiement fiable** : si déployer fait peur, on corrige trop tard.

🧠 ***Astuce*** *: la sécurité de l'infrastructure se mesure à la facilité de refaire la configuration* ***à l'identique*** *: si tu ne peux pas la reconstruire, tu ne peux pas la vérifier.*

Exemple pas à pas

Problème : écrire des audits automatisés (SSH, permissions, ports, Dockerfile, dépendances) et les tester.

```
import os, re, socket, stat, tempfile
from pathlib import Path

# ---------- 1. Audit de configuration SSH ----------
RECOMMANDATIONS_SSH = {
    "permitrootlogin": ("no", "la connexion root directe est interdite"),
    "passwordauthentication": ("no", "authentification par clés uniquement"),
    "permitemptypasswords": ("no", "pas de mot de passe vide"),
    "x11forwarding": ("no", "désactiver les fonctions inutiles"),
}

def auditer_sshd(config):
    valeurs = {}
    for ligne in config.splitlines():
        ligne = ligne.split("#")[0].strip()
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        if ligne:
            cle, _, valeur = ligne.partition(" ")
            valeurs[cle.lower()] = valeur.strip().lower()
    problemes = []
    for cle, (attendu, raison) in RECOMMANDATIONS_SSH.items():
        reel = valeurs.get(cle, "(défaut)")
        if reel != attendu:
            problemes.append(f"{cle} = {reel} (attendu : {attendu}) — {raison}")
    return problemes

print(auditer_sshd("PermitRootLogin yes\nPasswordAuthentication yes\nX11Forwarding
    no\n"))

# ---------- 2. Audit de permissions ----------
def auditer_permissions(dossier):
    problemes = []
    for chemin in Path(dossier).rglob("*"):
        mode = chemin.stat().st_mode
        if chemin.is_file() and mode & stat.S_IWOTH:
```

Exemple pas à pas (suite)

```
            problemes.append((str(chemin.relative_to(dossier)), "modifiable par to
    us"))
        if chemin.is_file() and re.search(r"(secret|\.env|id_rsa|\.pem)", chemin.n
    ame) and mode & (stat.S_IRGRP | stat.S_IROTH):
            problemes.append((str(chemin.relative_to(dossier)), "fichier sensible
    lisible par d'autres"))
     return problemes

with tempfile.TemporaryDirectory() as d:
    for nom, mode in (("app.py", 0o644), (".env", 0o644), ("id_rsa", 0o600), ("exp
    ort.csv", 0o666)):
        (Path(d) / nom).write_text("x")
        os.chmod(Path(d) / nom, mode)
    for chemin, raison in sorted(auditer_permissions(d)):
        print(f"{chemin:12s} {raison}")

# ---------- 3. Inventaire des ports (UNIQUEMENT sur sa propre machine) ----------
def ports_ouverts(hote, ports, delai=0.3):
    ouverts = set()
```

Exemple pas à pas (suite)

```
    for port in ports:
        with socket.socket() as s:
            s.settimeout(delai)
            if s.connect_ex((hote, port)) == 0:
                ouverts.add(port)
    return ouverts

serveur = socket.socket()
serveur.bind(("127.0.0.1", 0))
serveur.listen()
port_attendu = serveur.getsockname()[1]
port_ferme = port_attendu + 1 if port_attendu < 65535 else port_attendu - 1

observes = ports_ouverts("127.0.0.1", [port_attendu, port_ferme])
attendus = {port_attendu}
print("ouverts non attendus :", observes - attendus, "| attendus fermés :", attend
   us - observes)
serveur.close()
```

Exemple pas à pas (suite)

```
# ---------- 4. Linter de Dockerfile ----------
def linter_dockerfile(texte):
    problemes, a_user = [], False
    for numero, ligne in enumerate(texte.splitlines(), 1):
        l = ligne.strip()
        if re.match(r"(?i)^FROM\s+\S+:latest(\s|$)", l) or (re.match(r"(?i)^FROM\s
   +[^\s:]+(\s|$)", l) and "@" not in l):
            problemes.append((numero, "image de base non épinglée (latest ou sans
   étiquette)"))
        if re.match(r"(?i)^USER\s+(?!root\b)\S+", l):
            a_user = True
        if re.match(r"(?i)^USER\s+root\b", l):
            problemes.append((numero, "exécution en root"))
        if re.match(r"(?i)^ADD\s+https?://", l):
            problemes.append((numero, "ADD depuis une URL : préférer COPY + vérifi
   cation d'empreinte"))
        if re.match(r"(?i)^ENV\s+\S*(PASSWORD|SECRET|TOKEN|KEY)\S*[= ]", l):
            problemes.append((numero, "secret dans ENV (visible dans l'image)"))
    if not a_user:
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        problemes.append((0, "aucune instruction USER : le conteneur s'exécute en
   root"))
    return problemes

DOCKERFILE = '''FROM python:latest
ENV API_TOKEN=abc123
ADD https://exemple.org/install.sh /tmp/
COPY . /app
CMD ["python", "/app/main.py"]'''
for numero, probleme in linter_dockerfile(DOCKERFILE):
    print(f"ligne {numero} : {probleme}")

# ---------- 5. Dépendances vulnérables ----------
AVIS = {"vieillelib": [("< 2.4.1", "CVE-0000-0001 : exécution de code à distance",
    "critique")],
        "parseurxml": [("< 1.9.0", "CVE-0000-0002 : fuite de fichiers", "haute")]}

def version(v):
    return tuple(int(x) for x in v.split("."))
```

Exemple pas à pas (suite)

```
def auditer_dependances(lignes):
    resultats = []
    for ligne in lignes:
        nom, _, v = ligne.strip().partition("==")
        for contrainte, description, gravite in AVIS.get(nom, []):
            borne = contrainte.replace("<", "").strip()
            if version(v) < version(borne):
                resultats.append((nom, v, description, gravite, f"mettre à jour ve
   rs >= {borne}"))
    return resultats

for ligne in auditer_dependances(["vieillelib==2.3.0", "parseurxml==2.0.0", "autre
   ==1.0.0"]):
    print(ligne)
```

Lecture : chaque contrôle est automatisable, rapide, et peut tourner à chaque déploiement :

Exemple pas à pas (suite)

- SSH : `PermitRootLogin yes` et `PasswordAuthentication yes` sont signalés ;
- permissions : `.env` lisible par d'autres et `export.csv` modifiable par tous ;
- ports : aucun écart sur la machine locale (c'est un **inventaire attendu contre observé**) ;
- Dockerfile : image `latest`, secret dans `ENV`, `ADD` depuis une URL, pas d'`USER` ;
- dépendances : `vieillelib 2.3.0` critique, `parseurxml` à jour.

À toi, avec aide

Écris `auditer_tls(configuration)` qui prend un dictionnaire décrivant la configuration TLS d'un serveur et renvoie des problèmes :

```
{"protocoles": ["TLSv1.0", "TLSv1.2"], "suites": ["ECDHE-RSA-AES128-GCM-SHA256", "
   RC4-SHA"],
 "certificat": {"expire": "2024-01-15", "nom": "hashbox.example", "auto_signe": Fa
   lse}, "hsts": False}
```

Règles : refuser SSL/TLS 1.0/1.1, signaler les suites avec `RC4`, `3DES`, `NULL`, `MD5` ou sans `GCM`/`CHACHA20`/`ECDHE`, certificat expiré à une date de référence, auto-signé en production, HSTS absent. Teste sur une configuration très mauvaise, une moyenne et une bonne.

Indices progressifs

Indice 1

Compare les dates au format ISO avec des chaînes (`'2024-01-15' < '2025-01-01'`).

Indice 2

Une suite sans `ECDHE` n'a pas de confidentialité persistante.

Indice 3

Renvoie des tuples `(gravité, message)`.

À toi, avec aide · réponse

Réponse

```
FAIBLES = ("RC4", "3DES", "NULL", "MD5", "DES-CBC")

def auditer_tls(c, aujourdhui="2025-06-01"):
    p = []
    for proto in c["protocoles"]:
        if proto in ("SSLv2", "SSLv3", "TLSv1.0", "TLSv1.1"):
            p.append(("haute", f"protocole obsolète activé : {proto}"))
    for suite in c["suites"]:
        if any(f in suite for f in FAIBLES):
            p.append(("haute", f"suite faible : {suite}"))
        elif "ECDHE" not in suite:
            p.append(("moyenne", f"pas de confidentialité persistante : {suite}"))
        elif "GCM" not in suite and "CHACHA20" not in suite:
            p.append(("moyenne", f"mode non authentifié : {suite}"))
    cert = c["certificat"]
    if cert["expire"] < aujourdhui:
        p.append(("critique", f"certificat expiré le {cert['expire']}"))
    if cert["auto_signe"]:
        p.append(("haute", "certificat auto-signé en production"))
```

À toi, avec aide · réponse

Réponse (suite)

```
    if not c.get("hsts"):
        p.append(("moyenne", "HSTS absent"))
    return p

mauvaise = {"protocoles": ["TLSv1.0", "TLSv1.2"], "suites": ["ECDHE-RSA-AES128-GCM
   -SHA256", "RC4-SHA"],
            "certificat": {"expire": "2024-01-15", "nom": "hashbox.example", "auto
   _signe": False}, "hsts": False}
bonne = {"protocoles": ["TLSv1.2", "TLSv1.3"], "suites": ["ECDHE-RSA-AES256-GCM-SH
   A384", "ECDHE-ECDSA-CHACHA20-POLY1305"],
         "certificat": {"expire": "2026-03-01", "nom": "hashbox.example", "auto_si
   gne": False}, "hsts": True}
for gravite, message in auditer_tls(mauvaise):
    print(gravite, "-", message)
assert auditer_tls(bonne) == []
assert len(auditer_tls(mauvaise)) == 4
```

À toi, avec aide · réponse

Réponse (suite)

Pour vérifier une **vraie** configuration, on utilise des outils dédiés (`openssl s_client`, `testssl.sh`, les analyseurs en ligne) sur **ses propres** serveurs, et on alimente un test comme celui-ci avec leurs résultats.

À toi, en solo

Politique de correctifs. Écris un petit module qui, à partir d'un inventaire de composants et d'une base d'avis (comme dans l'exemple, avec gravité, date de publication, indicateur « exploité activement »), produit un **plan de mise à jour** :

1. échéance de correction selon la gravité et l'exploitation active (24 h, 7 j, 30 j, prochain cycle) ;
2. tri par urgence ;
3. détection des composants **sans mainteneur actif** (dernière version > 2 ans) à remplacer ;
4. un rapport indiquant les retards par rapport à la date du jour ;
5. un test avec 10 composants et 6 avis.

À toi, en solo · réponse

Pistes de correction

```
from datetime import date, timedelta

DELAIS = {"critique": 3, "haute": 7, "moyenne": 30, "basse": 90}

def echeance(avis):
    jours = 1 if avis["exploite"] and avis["gravite"] in ("critique", "haute") els
   e DELAIS[avis["gravite"]]
    return date.fromisoformat(avis["publie"]) + timedelta(days=jours)

def plan(inventaire, avis_par_composant, aujourdhui):
    lignes = []
    for nom, info in inventaire.items():
        v = tuple(int(x) for x in info["version"].split("."))
        for a in avis_par_composant.get(nom, []):
            if v < tuple(int(x) for x in a["corrige_en"].split(".")):
                e = echeance(a)
                lignes.append({"composant": nom, "gravite": a["gravite"], "echeanc
   e": e,
                               "retard_jours": max(0, (aujourdhui - e).days), "act
```

Pistes de correction (suite)

```
   ion": f"passer en >= {a['corrige_en']}"})
        if (aujourdhui - date.fromisoformat(info["derniere_release"])).days > 730:
            lignes.append({"composant": nom, "gravite": "moyenne", "echeance": auj
   ourdhui + timedelta(days=30),
                           "retard_jours": 0, "action": "composant sans maintenanc
   e : prévoir un remplacement"})
    return sorted(lignes, key=lambda l: (l["echeance"], l["composant"]))

INVENTAIRE = {
    "vieillelib": {"version": "2.3.0", "derniere_release": "2023-01-10"},
    "parseurxml": {"version": "1.8.0", "derniere_release": "2025-03-01"},
    "web": {"version": "4.1.0", "derniere_release": "2025-05-01"},
    "orm": {"version": "3.0.0", "derniere_release": "2025-04-01"},
    "abandonnee": {"version": "0.9.0", "derniere_release": "2021-06-01"},
}
AVIS = {
    "vieillelib": [{"gravite": "critique", "exploite": True, "publie": "2025-05-20
   ", "corrige_en": "2.4.1"}],
    "parseurxml": [{"gravite": "haute", "exploite": False, "publie": "2025-05-10",
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    "corrige_en": "1.9.0"}],
    "web": [{"gravite": "basse", "exploite": False, "publie": "2025-04-01", "corri
   ge_en": "4.2.0"}],
}
resultat = plan(INVENTAIRE, AVIS, date(2025, 6, 1))
for l in resultat:
    print(l["echeance"], l["composant"], l["gravite"], f"retard {l['retard_jours']
   } j", "-", l["action"])
assert any(l["composant"] == "vieillelib" and l["retard_jours"] > 0 for l in resul
   tat)
assert any(l["composant"] == "abandonnee" for l in resultat)
```

Un plan de correctifs transforme une liste de CVE en **décisions datées et responsables** : c'est ce qui fait la différence entre « on est au courant » et « c'est traité ».

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Services et ports inutiles	Chaque port est une porte	Inventaire attendu contre observé
Mots de passe et comptes par défaut	Exploités en minutes par les scans automatiques	Changer, désactiver, clés uniquement
Ne pas mettre à jour	Failles publiques exploitées en masse	Politique de correctifs avec délais
Désactiver la vérification TLS « pour tester »	Cela finit en production	Certificats de test valides ou autorité interne

Mini-boss

Dossier de durcissement d'un serveur d'application. À partir d'un « état du serveur » décrit en données (config SSH, liste de services écoutant, permissions de fichiers, Dockerfile, `requirements.txt`, configuration TLS, politique de mises à jour), construis un outil `audit_serveur.py` qui :

1. exécute les contrôles de la mission (SSH, ports, permissions, TLS, Dockerfile, dépendances) ;
2. calcule un **score de conformité** pondéré par gravité ;
3. produit un **rapport** Markdown classé par gravité avec la correction recommandée pour chaque écart ;
4. génère un **script de correction** (liste de commandes) pour les écarts simples, mais sans l'exécuter ;
5. compare deux états (avant/après) et montre les écarts corrigés ;
6. 20 tests, dont des états « bons » (aucun écart) et « terribles » (au moins 12 écarts).

Mini-boss · réponse

Pistes de correction

Un bon travail : une fonction par contrôle qui renvoie une liste d'écarts `(domaine, gravité, message, correction)`; un score `100 - somme(pondération[gravité])` borné à 0; le rapport trie par gravité puis par domaine; les corrections proposées sont des chaînes (`sed -i 's/^PermitRootLogin.*/PermitRootLogin no/' /etc/ssh/sshd_config`, `chmod 600 .env`, `pip install 'vieillelib>=2.4.1'`) **jamais exécutées automatiquement** (revue humaine); la comparaison avant/après calcule l'ensemble des écarts résolus, persistants, nouveaux; les tests vérifient qu'un état conforme donne 100 et aucun écart, et qu'un état dégradé produit au moins 12 écarts dont tous les domaines.

Quiz éclair

Quiz éclair

1. Le durcissement d'un système consiste à :

A. ajouter des fonctions
B. ralentir le serveur
C. réduire la surface d'attaque et sécuriser la configuration

2. Une bonne configuration SSH :

A. interdit la connexion root et les mots de passe, avec des clés
B. autorise tout pour la commodité
C. change seulement le port

3. Dans un conteneur, on évite :

A. d'utiliser une image minimale

Quiz éclair

Quiz éclair (suite)

B. d'exécuter le processus en root
C. d'épingler les versions

4. 🔁 *Révision mission 07* : L'injection consiste à :

A. chiffrer deux fois
B. saturer un serveur
C. mélanger des données non fiables à une instruction interprétée

5. 🔁 *Révision mission 07* : La validation d'entrées doit privilégier :

A. la liste blanche de ce qui est permis
B. la liste noire des caractères dangereux
C. l'absence de contrôle

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

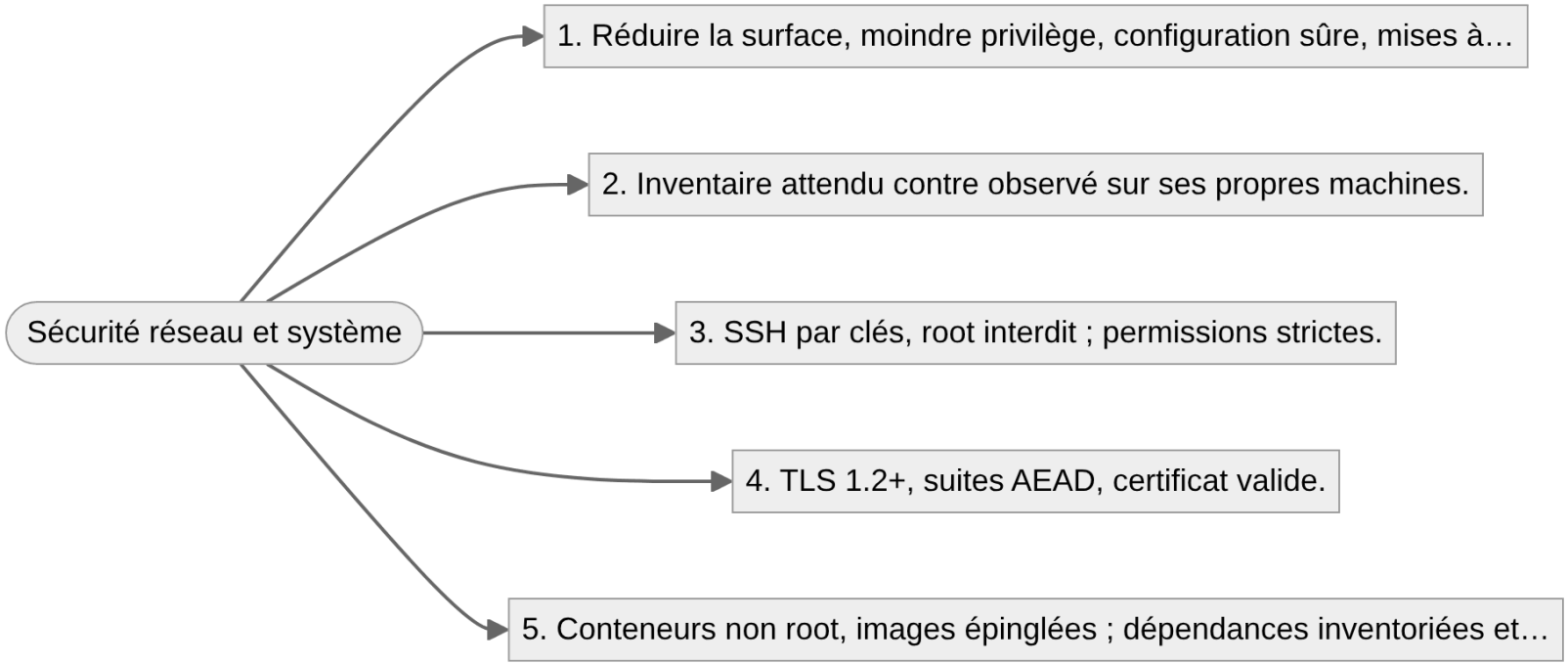
1. Quels services ou comptes « oubliés » existent dans ton environnement (anciens comptes, ports ouverts, applications inutiles) ?
2. Comment saurais-tu, aujourd'hui, si une dépendance de ton projet a une faille connue ?

À retenir

1. Réduire la surface, moindre privilège, configuration sûre, mises à jour.
2. Inventaire attendu contre observé (sur ses propres machines).
3. SSH par clés, root interdit ; permissions strictes.
4. TLS 1.2+, suites AEAD, certificat valide.
5. Conteneurs non root, images épinglées ; dépendances inventoriées et mises à jour.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Durcisseur de systèmes

Prochaine mission : Journalisation, détection et réponse à incident

