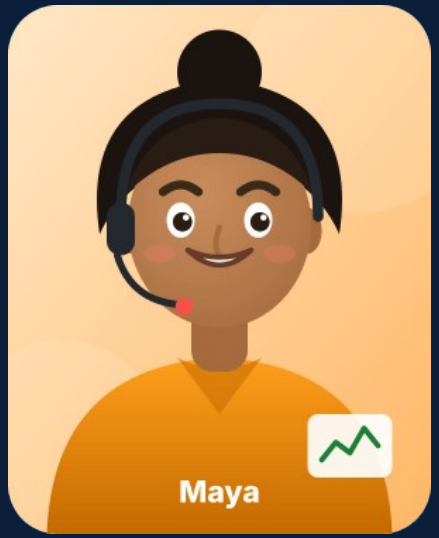
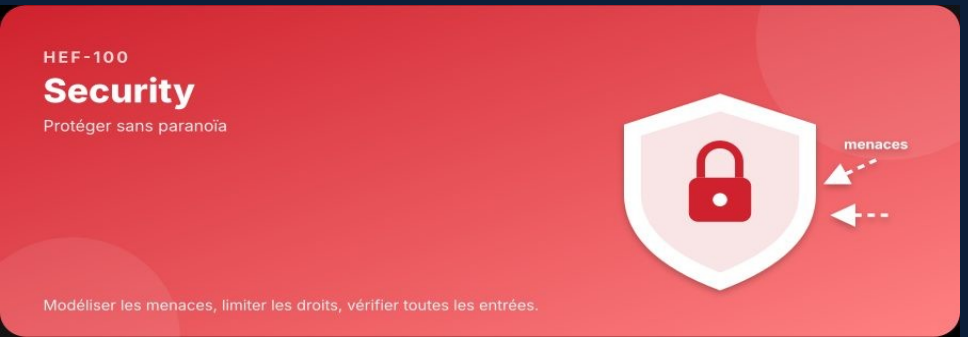
HEF-100 · MISSION 10 / 10

Audit et rapport de sécurité

Trouver des failles ne sert à rien si personne ne les comprend ni ne les corrige. Un bon rapport transforme un constat en décisions.





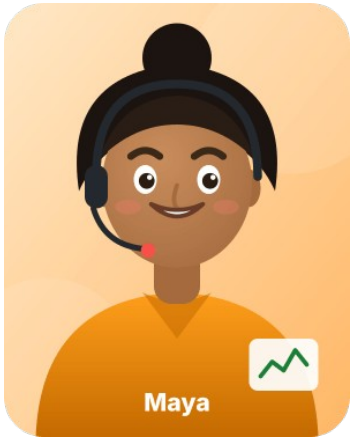
Badge : Auditeur·rice de sécurité

110 min · intermédiaire

L'histoire

La direction de HashBox veut un état des lieux avant l'ouverture à de nouveaux clients. Maya demande à Kodi : « Fais-moi un rapport de sécurité. Je veux savoir ce qui est grave, ce qu'il faut corriger d'abord, et combien ça coûte. »

> *« Un audit, ce n'est pas une liste de failles, dit Sam. C'est une histoire : ce que j'ai regardé, ce que j'ai trouvé, ce que ça signifie pour l'entreprise, et ce qu'on fait. »*



Dans cette dernière mission, tu mènes un audit complet : analyse statique automatisée, évaluation de la gravité (CVSS), rapport professionnel et plan de remédiation.

Ta mission

À la fin, je serai capable de…

1. préparer et cadrer un audit (périmètre, règles d'engagement, autorisation)
2. écrire un analyseur statique simple de code Python avec le module `ast`
3. calculer un score de gravité CVSS v3.1 pour une vulnérabilité
4. rédiger un rapport structuré destiné à des publics techniques et non techniques
5. bâtir un plan de remédiation priorisé et vérifier les corrections (retest)

Défi d'échauffement

Défi d'échauffement

Un inspecteur visite un restaurant. Son compte rendu doit être utile à trois lecteurs : le **patron** (que dois-je faire en premier, quel risque pour mon activité ?), le **cuisinier** (quel geste précis corriger ?) et l'**administration** (est-ce conforme ?). Écris les trois éléments minimaux de chaque partie du rapport. Pourquoi une simple liste de 80 défauts est-elle inutilisable ?

Le concept

Les types d'évaluation

Type	Objectif	Méthode
Revue de code	trouver des failles dans le code	lecture, analyse statique (SAST)
Analyse de dépendances	composants vulnérables	outils de composition logicielle (SCA)
Test d'intrusion	exploiter les failles dans un cadre autorisé	tests manuels et outils
Évaluation de configuration	écarts de durcissement	audits de conformité
Revue d'architecture / modèle de menaces	failles de conception	ateliers, DFD, STRIDE
Exercice d'équipe rouge/bleue	tester détection et réponse	simulation d'adversaire

Le concept

Cadrer un audit

Avant de commencer :

1. **Autorisation écrite** du propriétaire du système (sans elle, c'est une infraction) ;
2. **Périmètre** : quoi est inclus, quoi est exclu ;
3. **Règles d'engagement** : horaires, techniques permises, données sensibles, que faire si on trouve un incident réel ;
4. **Contacts** et canal sûr pour signaler une faille critique **immédiatement** ;
5. **Livrables** et calendrier.

L'analyse statique du code (SAST)

Le concept

L'analyse statique du code (SAST) (suite)

Un analyseur statique lit le code **sans l'exécuter** et repère des motifs dangereux. En Python, le module `ast` donne accès à l'arbre syntaxique (voir HEF-090) ; on y cherche par exemple :

Motif	Risque
`eval`, `exec`	exécution de code arbitraire
`pickle.loads`, `yaml.load` sans `SafeLoader`	désérialisation dangereuse
`subprocess.*(…, shell=True)`	injection de commande
`.execute(f"…")`, `.execute("…" + x)`, `.execute("…" % x)`	injection SQL
`hashlib.md5`, `hashlib.sha1`	hachage faible
`random.*` pour des jetons ou secrets	aléa prévisible
`verify=False`, `ssl._create_unverified_context`	vérification TLS désactivée
affectations `password = "…"`, `secret = "…"`, `api_key = "…"`	secret en dur

Le concept

L'analyse statique du code (SAST) (suite)

Motif	Risque
`debug=True` en production	divulgation d'informations

L'analyse statique produit des **faux positifs** et des **faux négatifs** ; elle complète, mais ne remplace pas, la revue humaine et les tests.

Évaluer la gravité : CVSS

Le **CVSS** (*Common Vulnerability Scoring System*) v3.1 donne un score de 0 à 10 à partir de métriques de base :

Métrique	Valeurs
AV — vecteur d'attaque	Réseau (N), Adjacent (A), Local (L), Physique (P)
AC — complexité	Basse (L), Haute (H)

Le concept

Évaluer la gravité : CVSS (suite)

Métrique	Valeurs
PR — privilèges requis	Aucun (N), Bas (L), Haut (H)
UI — interaction utilisateur	Aucune (N), Requise (R)
C / I / A — impact sur confidentialité / intégrité / disponibilité	Haut (H), Bas (L), Aucun (N)

Score	Gravité
0,0	aucune
0,1 – 3,9	basse
4,0 – 6,9	moyenne
7,0 – 8,9	haute
9,0 – 10,0	critique

Le concept

Évaluer la gravité : CVSS (suite)

Le CVSS mesure la **gravité technique**, pas le **risque pour ton entreprise** : une faille 9,8 sur une machine isolée et sans données peut compter moins qu'une faille 6,5 sur ta base clients. On combine donc avec le **contexte** (exposition, valeur de l'actif, exploitation connue).

Le rapport

Section	Public	Contenu
Résumé exécutif	direction	situation globale, risques majeurs en langage clair, décisions demandées
Périmètre et méthode	tous	ce qui a été testé, comment, limites
Synthèse des constats	direction, responsables	tableau par gravité, tendances
Détail des constats	techniciens	description, preuve, impact, gravité, recommandation

Le concept

Le rapport (suite)

Section	Public	Contenu
Plan de remédiation	responsables	priorités, responsables, échéances, effort
Annexes	techniciens	détails techniques, données brutes

Écrire un bon constat

Chaque constat contient :

1. un **titre** précis (« Injection SQL dans la recherche de cours ») ;
2. la **gravité** et le **vecteur CVSS** ;
3. la **description** et l'**emplacement** (fichier, ligne, URL) ;
4. la **preuve** reproductible (sans publier de données réelles) ;
5. l'**impact** pour l'entreprise ;

Le concept

Écrire un bon constat (suite)

6. la **recommandation** concrète et vérifiable ;
7. des **références** (catégorie OWASP, CWE).

Prioriser la remédiation

Critères : gravité, **exploitabilité**, **exposition**, **valeur** de l'actif, **effort** de correction. Une règle utile : traiter d'abord ce qui est **grave, facile à exploiter et exposé**, ainsi que les **corrections rapides à fort effet** (« quick wins »).

Vérifier : le retest

Un constat n'est clos qu'après un **retest** : on rejoue l'attaque, et on vérifie qu'elle échoue ; on ajoute le test de régression automatisé.

Le concept

Divulgation responsable

Si tu trouves une faille chez un tiers : tu **ne l'exploites pas au-delà du minimum**, tu **contactes le propriétaire** par le canal de sécurité (`security.txt`, programme de bug bounty), tu laisses un **délai raisonnable** pour corriger et tu ne publies pas de détails exploitables avant correction.

🧠 ***Astuce** : mesure la qualité d'un rapport à ce qu'il déclenche : si la direction sait quoi décider et si les développeurs savent quoi corriger, il est réussi.*

Exemple pas à pas

Exemple pas à pas

Problème : analyseur statique, score CVSS, et génération d'un rapport.

```
import ast, math

# ---------- 1. Analyseur statique (SAST) ----------
SOURCE = '''
import hashlib, pickle, random, subprocess, sqlite3, requests

API_KEY = "sk_live_1234567890abcdef"

def jeton():
    return "".join(random.choice("abcdef0123456789") for _ in range(16))

def hacher(mdp):
    return hashlib.md5(mdp.encode()).hexdigest()

def recherche(bd, terme):
    return bd.execute("SELECT * FROM cours WHERE titre LIKE '%" + terme + "%'").fe
```

Exemple pas à pas (suite)

```
   tchall()

def lire(nom):
    return subprocess.run("cat " + nom, shell=True)

def charger(octets):
    return pickle.loads(octets)

def appeler(url):
    return requests.get(url, verify=False)

def calcul(expression):
    return eval(expression)
'''

def nom_appel(n):
    f = n.func
    if isinstance(f, ast.Attribute) and isinstance(f.value, ast.Name):
        return f"{f.value.id}.{f.attr}"
```

Exemple pas à pas (suite)

```
    if isinstance(f, ast.Name):
        return f.id
    return ""

def construit_dynamique(n):                          # concaténation, f-string ou
    formatage %
    return isinstance(n, (ast.JoinedStr, ast.BinOp))

def analyser(source):
    arbre = ast.parse(source)
    constats = []
    sensibles = [f for f in ast.walk(arbre) if isinstance(f, ast.FunctionDef) and
   any(m in f.name for m in ("jeton", "token", "secret"))]
    def ajouter(n, titre, gravite, owasp, reco):
        constats.append({"ligne": n.lineno, "titre": titre, "gravite": gravite, "o
   wasp": owasp, "recommandation": reco})
    for n in ast.walk(arbre):
        if isinstance(n, ast.Assign) and isinstance(n.value, ast.Constant) and isi
   nstance(n.value.value, str):
```

Exemple pas à pas (suite)

```
            for cible in n.targets:
                if isinstance(cible, ast.Name) and any(m in cible.id.lower() for m
     in ("password", "secret", "token", "api_key", "apikey")) and len(n.value.value
    ) >= 8:
                    ajouter(n, f"secret en dur ({cible.id})", "haute", "A02/A05",
    "variable d'environnement ou coffre, révoquer le secret exposé")
        if isinstance(n, ast.Call):
            nom = nom_appel(n)
            if nom in ("eval", "exec"):
                ajouter(n, f"{nom}() sur une entrée", "critique", "A03", "ne jamai
    s évaluer une entrée ; analyseur dédié (ast.literal_eval au plus)")
            if nom in ("pickle.loads", "pickle.load"):
                ajouter(n, "désérialisation pickle", "critique", "A08", "utiliser
    JSON avec validation de schéma")
            if nom.startswith("subprocess.") and any(k.arg == "shell" and isinstan
    ce(k.value, ast.Constant) and k.value.value is True for k in n.keywords):
                ajouter(n, "subprocess avec shell=True", "haute", "A03", "liste d'
    arguments sans shell")
            if nom in ("hashlib.md5", "hashlib.sha1"):
```

Exemple pas à pas

Exemple pas à pas (suite)

```
                ajouter(n, f"{nom} (hachage faible)", "moyenne", "A02", "SHA-256 ;
    pour les mots de passe : scrypt/argon2id")
            if nom.endswith(".execute") and n.args and construit_dynamique(n.args[
    0]):
                ajouter(n, "requête SQL construite dynamiquement", "critique", "A0
    3", "requêtes paramétrées")
            if any(k.arg == "verify" and isinstance(k.value, ast.Constant) and k.v
    alue.value is False for k in n.keywords):
                ajouter(n, "vérification TLS désactivée", "haute", "A02", "laisser
     la vérification activée, installer l'autorité de confiance")
            if nom.startswith("random.") and any(f.lineno <= n.lineno <= f.end_lin
    eno for f in sensibles):
                ajouter(n, "aléa non cryptographique pour un jeton", "haute", "A02
    ", "module secrets")
    return sorted(constats, key=lambda c: c["ligne"])

for c in analyser(SOURCE):
    print(f"ligne {c['ligne']:2d} [{c['gravite']:8s}] {c['titre']}")
```

Exemple pas à pas

Exemple pas à pas (suite)

```
# ---------- 2. Score CVSS v3.1 (métriques de base, portée inchangée) ----------
AV = {"N": 0.85, "A": 0.62, "L": 0.55, "P": 0.2}
AC = {"L": 0.77, "H": 0.44}
PR = {"N": 0.85, "L": 0.62, "H": 0.27}
UI = {"N": 0.85, "R": 0.62}
CIA = {"H": 0.56, "L": 0.22, "N": 0.0}

def arrondi_haut(x):
    entier = round(x * 100000)
    return entier / 100000.0 if entier % 10000 == 0 else (math.floor(entier / 1000
   0) + 1) / 10.0

def cvss(vecteur):
    m = dict(p.split(":") for p in vecteur.split("/")[1:])
    iss = 1 - (1 - CIA[m["C"]]) * (1 - CIA[m["I"]]) * (1 - CIA[m["A"]])
    impact = 6.42 * iss
    exploitabilite = 8.22 * AV[m["AV"]] * AC[m["AC"]] * PR[m["PR"]] * UI[m["UI"]]
    if impact <= 0:
        return 0.0
```

Exemple pas à pas (suite)

```
    return arrondi_haut(min(impact + exploitabilite, 10))

assert cvss("CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H") == 9.8
assert cvss("CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N") == 6.5
assert cvss("CVSS:3.1/AV:L/AC:H/PR:H/UI:R/S:U/C:L/I:N/A:N") == 1.8

def gravite(score):
    return "aucune" if score == 0 else "basse" if score < 4 else "moyenne" if scor
   e < 7 else "haute" if score < 9 else "critique"

print(cvss("CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"), gravite(9.8))
```

Lecture :

- l'analyseur trouve **huit** problèmes dans un code de 25 lignes : secret en dur, jeton avec `random`, MD5, SQL concaténé, `shell=True`, `pickle`, `verify=False`, `eval` ;
- chaque constat porte sa gravité, sa catégorie OWASP et une recommandation ;

Exemple pas à pas (suite)

- le calcul CVSS reproduit les valeurs de référence (9,8 pour une exécution de code distante sans authentification ; 6,5 pour une faille réseau à impact limité) ;
- l'outil est **un auxiliaire** : il prépare la revue humaine, il ne la remplace pas.

À toi, avec aide

Calcule et interprète le score CVSS (avec la fonction `cvss` de l'exemple) pour ces trois vulnérabilités, puis dis laquelle tu corriges en premier **dans le contexte de HashBox** (application publique, base de données clients) :

1. Injection SQL sur la recherche publique, sans authentification : `AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`.
2. XSS stocké dans les commentaires, nécessitant qu'un utilisateur ouvre la page : `AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N`.
3. Mot de passe administrateur par défaut sur une console accessible seulement depuis le réseau interne : `AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise la fonction `cvss` de l'exemple.

Indice 2

Pense à l'exposition : une faille sur une page publique est atteignable par tous.

Indice 3

Une faille interne n'est pas à ignorer : elle devient grave si un attaquant entre dans le réseau.

À toi, avec aide · réponse

Réponse

```
# (fonctions cvss et gravite de l'exemple)
for i, v in enumerate(["CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N",
                       "CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N",
                       "CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"], 1):
    s = cvss(v)
    print(i, s, gravite(s))
```

Résultats : 1 → **9,1 (critique)**, 2 → **4,6 (moyenne)**, 3 → **8,8 (haute)**.

Ordre dans le contexte de HashBox : **1** d'abord (critique, exposée à tout internet, sans authentification, accès à la base clients) ; ensuite **3** (haute, mais limitée au réseau interne : correction rapide et peu coûteuse : changer le mot de passe) — c'est un « quick win » à faire le jour même ; puis **2** (moyenne : XSS stocké, dont l'impact réel dépend du contenu des sessions et des CSP). Le score CVSS donne la gravité **technique** ; l'exposition et la valeur de l'actif décident de l'ordre réel.

À toi, en solo

Générateur de rapport. Écris `generer_rapport(constats, contexte)` qui produit un rapport Markdown à partir d'une liste de constats `{id, titre, vecteur_cvss, description, emplacement, preuve, recommandation, owasp, effort_jours, exposition}` :

1. **résumé exécutif** généré automatiquement : nombre de constats par gravité, les 3 risques majeurs en une phrase simple chacun, une recommandation de décision ;
2. **tableau de synthèse** trié par gravité puis score ;
3. **détail** de chaque constat avec le vecteur et le score calculé (fonction `cvss`) ;
4. **plan de remédiation** priorisé par `score × exposition / effort` avec trois horizons (sous 72 h, ce mois-ci, ce trimestre) ;
5. teste avec 8 constats inventés et vérifie que le rapport contient chaque section et que l'ordre de priorité est cohérent.

À toi, en solo · réponse

Pistes de correction

```
# (fonctions cvss et gravite de l'exemple)
from collections import Counter

EXPOSITION = {"internet": 1.0, "partenaires": 0.6, "interne": 0.3}

def priorite(c):
    return cvss(c["vecteur_cvss"]) * EXPOSITION[c["exposition"]] / max(c["effort_j
   ours"], 0.5)

def horizon(c):
    s = cvss(c["vecteur_cvss"]) * EXPOSITION[c["exposition"]]
    return "sous 72 h" if s >= 8 else "ce mois-ci" if s >= 4 else "ce trimestre"

def generer_rapport(constats, contexte):
    for c in constats:
        c["score"], c["gravite"] = cvss(c["vecteur_cvss"]), gravite(cvss(c["vecteu
   r_cvss"]))
    ordre = ["critique", "haute", "moyenne", "basse"]
    nb = Counter(c["gravite"] for c in constats)
```

Pistes de correction (suite)

```
    tri = sorted(constats, key=lambda c: (ordre.index(c["gravite"]), -c["score"]))
    majeurs = sorted(constats, key=priorite, reverse=True)[:3]
    lignes = [f"# Rapport de sécurité — {contexte['application']}", "", "## 1. Rés
umé exécutif",
              f"L'audit a relevé {len(constats)} constats : " + ", ".join(f"{nb.ge
t(g, 0)} {g}(s)" for g in ordre) + ".", "",
              "Les trois risques à traiter en priorité :"]
    for c in majeurs:
        lignes.append(f"- **{c['titre']}** ({c['gravite']}, exposition : {c['expos
ition']}) — {c['description'].split('.')[0]}.")
    lignes += ["", "**Décision demandée** : valider un gel des nouvelles fonctionn
alités le temps de corriger les risques « sous 72 h ».", "",
               "## 2. Synthèse", "", "| Id | Titre | Score | Gravité | Exposition
|", "|---|---|---|---|---|"]
    lignes += [f"| {c['id']} | {c['titre']} | {c['score']} | {c['gravite']} | {c['
exposition']} |" for c in tri]
    lignes += ["", "## 3. Détail des constats"]
    for c in tri:
        lignes += ["", f"### {c['id']} — {c['titre']}", f"- **Gravité** : {c['grav
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    ite']} ({c['score']}) — `{c['vecteur_cvss']}`",
                    f"- **Emplacement** : {c['emplacement']}", f"- **Catégorie** :
    {c['owasp']}",
                    f"- **Description** : {c['description']}", f"- **Preuve** : {c[
    'preuve']}", f"- **Recommandation** : {c['recommandation']}"]
     lignes += ["", "## 4. Plan de remédiation", "", "| Horizon | Constat | Effort
    (j) |", "|---|---|---|"]
     for c in sorted(constats, key=priorite, reverse=True):
         lignes.append(f"| {horizon(c)} | {c['id']} {c['titre']} | {c['effort_jours
    ']} |")
     return "\n".join(lignes)

CONSTATS = [
     dict(id="F1", titre="Injection SQL (recherche)", vecteur_cvss="CVSS:3.1/AV:N/A
    C:L/PR:N/UI:N/S:U/C:H/I:H/A:N", description="La recherche concatène le terme da
    ns la requête. Un attaquant lit toute la base.", emplacement="api/recherche.py:
    42", preuve="terme `x%' OR 1=1 --` renvoie les cours privés", recommandation="r
    equêtes paramétrées", owasp="A03", effort_jours=0.5, exposition="internet"),
     dict(id="F2", titre="XSS stocké", vecteur_cvss="CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    :U/C:L/I:L/A:N", description="Les commentaires sont affichés sans échappement."
    , emplacement="web/forum.py:88", preuve="commentaire `<b>test</b>` rendu en gra
    s", recommandation="échappement et CSP", owasp="A03", effort_jours=1, expositio
    n="internet"),
     dict(id="F3", titre="Mot de passe admin par défaut", vecteur_cvss="CVSS:3.1/AV
    :A/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H", description="La console d'administration ut
    ilise admin/admin.", emplacement="console interne", preuve="connexion réussie a
    vec les identifiants par défaut", recommandation="changer le mot de passe, MFA"
    , owasp="A07", effort_jours=0.2, exposition="interne"),
     dict(id="F4", titre="MD5 pour les mots de passe", vecteur_cvss="CVSS:3.1/AV:N/
    AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N", description="Les mots de passe sont hachés ave
    c MD5 sans sel.", emplacement="auth/mdp.py:12", preuve="deux comptes au même mo
    t de passe ont le même haché", recommandation="scrypt/argon2id avec sel et migr
    ation progressive", owasp="A02", effort_jours=3, exposition="internet"),
]
rapport = generer_rapport(CONSTATS, {"application": "HashBox"})
for section in ("## 1. Résumé exécutif", "## 2. Synthèse", "## 3. Détail des const
    ats", "## 4. Plan de remédiation"):
     assert section in rapport
```

À toi, en solo · réponse

Pistes de correction (suite)

```
assert rapport.index("F1") < rapport.index("F2")                    # le cr
   itique passe avant le moyen
print(rapport[:900])
```

Le rapport s'adresse à trois publics : la direction (section 1), les responsables (2 et 4), les techniciens (3). La priorité combine gravité, exposition et effort pour désigner les corrections les plus rentables.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Auditer sans autorisation écrite	Infraction pénale, même avec de bonnes intentions	Périmètre, règles d'engagement et contact signés avant de commencer
Livrer une liste brute de failles	Personne ne sait par quoi commencer	Priorisation par gravité, exposition, effort
Se fier uniquement au score CVSS	Il ne tient pas compte du contexte	Combiner avec exposition et valeur de l'actif
Clore un constat sans retest	On croit corrigé ce qui ne l'est pas	Rejouer l'attaque et ajouter un test automatisé

Mini-boss

Mini-boss

Audit de sécurité complet de la mini-application HashBox. Prends l'application jouet du boss de la mission 7 (ou écris-en une de 150 lignes) et mène un audit en conditions réelles :

1. **Cadrage** : rédige le périmètre, les règles d'engagement et le plan de test (1 page) ;
2. **Analyse statique** : étends l'analyseur pour couvrir au moins 12 motifs de la mission et exécute-le ;
3. **Tests dynamiques** : écris un jeu d'attaques automatisées (injections, chemin, IDOR, XSS, CSRF, débit) qui s'exécutent contre l'application et enregistrent les réussites ;
4. **Modèle de menaces** : un DFD avec STRIDE, croisé avec les constats ;
5. **Rapport** : généré par ton code (résumé exécutif, synthèse, détail avec CVSS, plan de remédiation priorisé) ;
6. **Remédiation et retest** : corrige les constats critiques et hauts, rejoue toutes les attaques, montre l'avant/après (nombre de constats par gravité) ;

Mini-boss

Mini-boss (suite)

7. **Retour d'expérience** : une page : ce qui a été le plus efficace pour trouver les failles, ce qui t'a échappé, comment l'équipe peut intégrer ces contrôles en continu (intégration continue, pré-commit, revue).

Mini-boss · réponse

Pistes de correction

Un bon dossier : un document `AUDIT.md` structuré (cadrage, méthode, limites), un outil `audit.py` avec les règles statiques et un banc d'attaques dynamiques, un rapport généré (Markdown) avec résumé exécutif compréhensible par une personne non technique, des vecteurs CVSS cohérents (une injection SQL non authentifiée à 9,1 ; une IDOR authentifiée à 6,5), un plan priorisé (par exemple « sous 72 h : injection SQL, secret en dur ; ce mois-ci : XSS, MD5 ; ce trimestre : journalisation »), un tableau avant/après (par exemple 5 critiques/hauts → 0), les tests de régression conservés dans la suite de tests. Le retour d'expérience souligne souvent que l'analyse statique trouve vite les motifs évidents mais que les failles de logique (IDOR, contrôle d'accès) demandent des tests dynamiques et une revue humaine, et propose d'automatiser les attaques en CI.

Quiz éclair

Quiz éclair

1. Avant un audit ou un test d'intrusion, il faut absolument :

A. une autorisation écrite et un périmètre défini
B. un outil à la mode
C. l'accord de la presse

2. Le score CVSS mesure :

A. le risque exact pour l'entreprise
B. la gravité technique d'une vulnérabilité
C. le coût de correction

3. Un constat d'audit n'est clos qu'après :

A. un e-mail de confirmation

Quiz éclair

Quiz éclair (suite)

B. la fin du trimestre
C. un retest qui prouve que l'attaque échoue

4. 🔁 *Révision mission 09* : Une chaîne de hachés sert à :

A. détecter la modification ou la suppression de lignes d'un journal
B. chiffrer les journaux
C. compresser les journaux

5. 🔁 *Révision mission 09* : La phase de confinement vise à :

A. restaurer les sauvegardes
B. stopper la propagation de l'incident
C. rédiger le rapport

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Qu'est-ce qui te frappe le plus dans l'idée qu'un rapport doit parler à trois publics différents ?
2. Quelle habitude de sécurité de ce module vas-tu appliquer dès demain à tes propres projets ou comptes ?

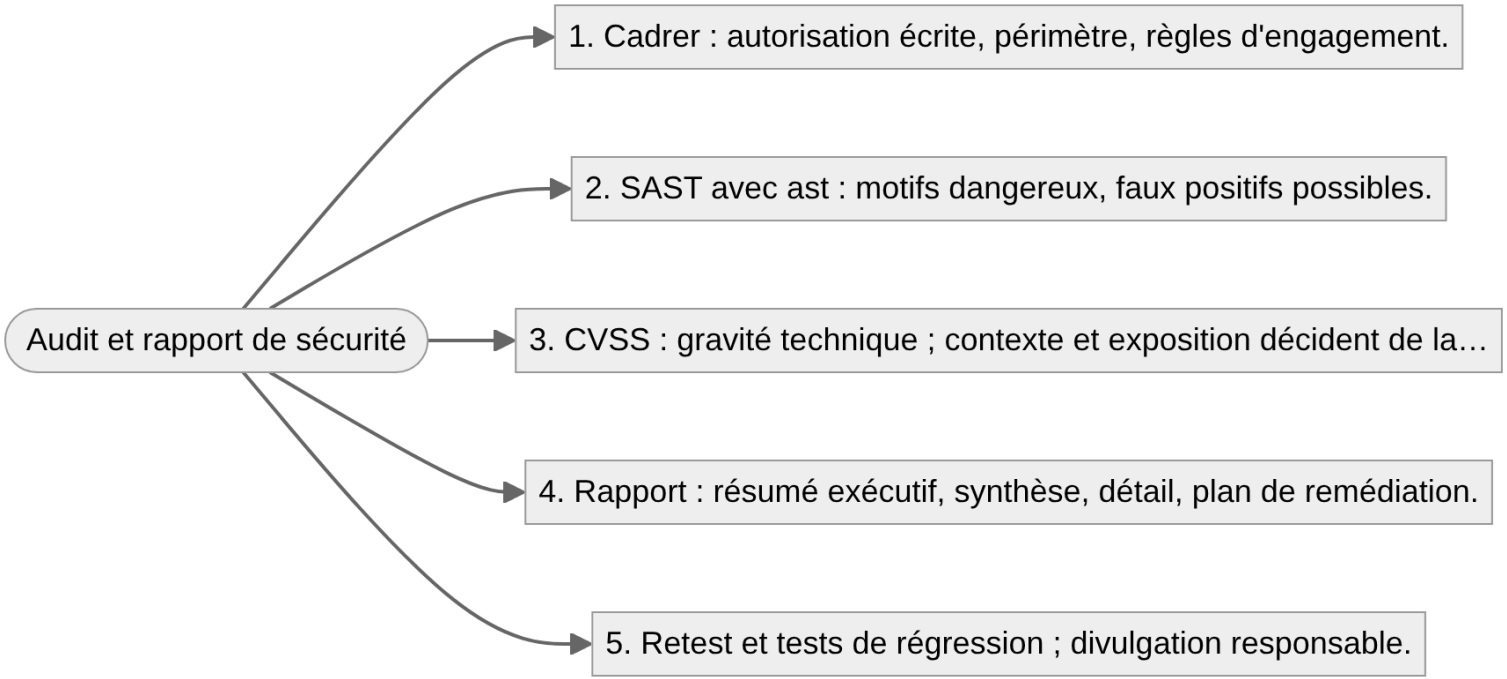
À retenir

À retenir

1. Cadrer : autorisation écrite, périmètre, règles d'engagement.
2. SAST avec `ast` : motifs dangereux, faux positifs possibles.
3. CVSS : gravité technique ; contexte et exposition décident de la priorité.
4. Rapport : résumé exécutif, synthèse, détail, plan de remédiation.
5. Retest et tests de régression ; divulgation responsable.

À retenir

Le schéma de la mission



Félicitations

Félicitations

Tu as terminé **HEF-100 — Sécurité**. Tu sais modéliser les menaces, authentifier et autoriser correctement, utiliser la cryptographie sans l'inventer, protéger les secrets, te défendre contre les attaques web et les injections, durcir un système, détecter et répondre à un incident, puis auditer et rapporter.

→ Prochain module : **HEF-110 — Tests et qualité**

MISSION ACCOMPLIE

+120 XP

Badge débloqué : Auditeur·rice de sécurité

Fin du module HEF-100. Bravo !

