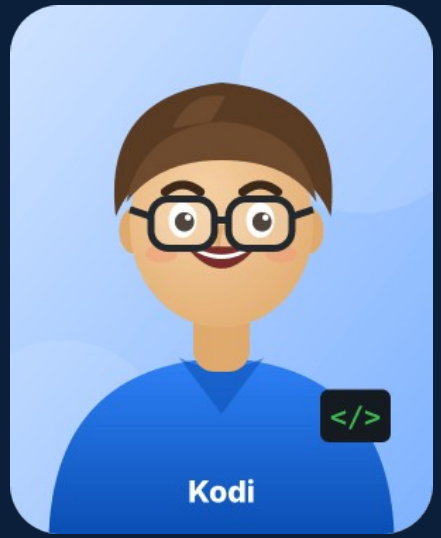
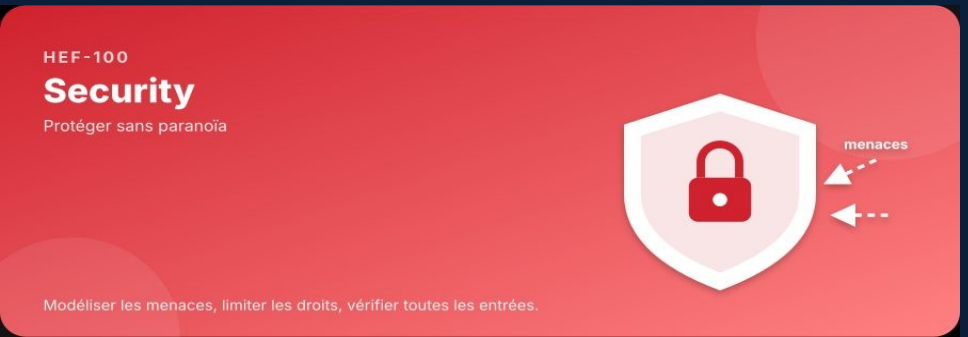
HEF-100 · MISSION 05 / 10

Secrets et données sensibles

Un secret écrit dans le code est un secret publié. Apprends à les garder, à les faire tourner et à ne pas les laisser fuir.



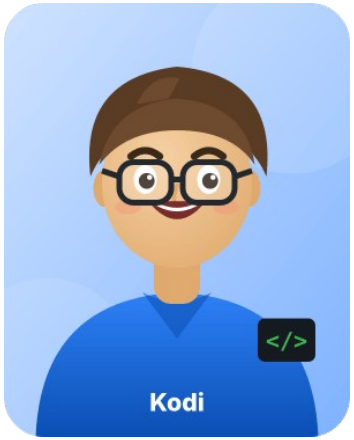


Badge : Gardien des secrets

90 min · intermédiaire

L'histoire

Kodi pousse son code sur un dépôt public pour montrer son travail. Quelques minutes plus tard, un message automatique arrive : « Votre clé d'accès au service cloud a été détectée dans un dépôt public. Elle a été désactivée. » En fouillant, il trouve aussi une facture surprise de 300 € : des inconnus avaient déjà utilisé la clé pour miner de la cryptomonnaie.



> *« Ce n'est pas un cas rare, dit Sam. Des robots parcourent les dépôts publics en continu. Un secret versionné doit être considéré comme **compromis**. »*

Tu apprends à gérer les secrets (clés, mots de passe, jetons), à détecter les fuites et à protéger les données personnelles.

Ta mission

À la fin, je serai capable de…

1. lister les types de secrets et les endroits où ils fuient
2. charger la configuration sans mettre de secret dans le code
3. détecter des secrets dans un texte avec motifs et entropie
4. masquer des données sensibles dans les journaux
5. appliquer rotation, minimisation et chiffrement au repos

Défi d'échauffement

Défi d'échauffement

Tu partages ton appartement avec plusieurs personnes. Tu as une clé de la porte, un code du coffre, le mot de passe du wifi, et un double de clé chez le voisin. Pour chacun : qui doit le connaître ? Que se passe-t-il s'il fuit ? Comment le changes-tu ? Qu'est-ce qui rend le changement facile ou pénible ?

Le concept

Qu'est-ce qu'un secret ?

Tout ce qui donne un accès ou un pouvoir à celui qui le détient : mot de passe, clé d'API, jeton, clé privée, certificat, chaîne de connexion à la base, secret de signature, clé de chiffrement, code de récupération.

Où les secrets fuient-ils ?

Endroit	Exemple
Code source	`PASSWORD = "..."` dans un fichier versionné
Historique Git	secret supprimé du fichier mais toujours dans les anciens commits
Fichiers de configuration	`.env`, `config.json` envoyés par erreur

Le concept

Où les secrets fuient-ils ? (suite)

Endroit	Exemple
Journaux	requête avec jeton, mot de passe affiché dans une erreur
Images et artefacts	secret copié dans une image conteneur
Messageries, tickets, captures d'écran	« voici le mot de passe »
Variables d'environnement	visibles dans `/proc`, les dumps d'erreur, les outils de supervision
Appareils et sauvegardes	sauvegardes non chiffrées

Principes

1. **Aucun secret dans le code ni dans le dépôt.**
2. **Un secret par usage** et par environnement (développement, test, production) : la fuite de l'un ne compromet pas les autres.

Le concept

Principes (suite)

3. **Moindre privilège** : un secret donne les droits minimaux.
4. **Rotation** régulière et **révocation** rapide : savoir changer un secret sans arrêter le service.
5. **Stockage centralisé** dans un **gestionnaire de secrets** (coffre) avec contrôle d'accès et journal : HashiCorp Vault, services cloud, gestionnaires d'équipe.
6. **Pas de secrets longue durée** quand on peut avoir des jetons **courts** et délivrés à la demande.
7. **Si un secret a fui : il est compromis.** On le révoque et on le remplace ; supprimer le commit ne suffit pas.

Charger les secrets proprement

```
import os

def charger_config():
    config = {
```

Le concept

Charger les secrets proprement (suite)

```
        "base_url": os.environ["HASHBOX_DB_URL"],           # lève une KeyError si
    absent : on refuse de démarrer
        "cle_api": os.environ["HASHBOX_API_KEY"],
        "debug": os.environ.get("HASHBOX_DEBUG", "0") == "1",
    }
    return config

class Secret:
    """Valeur qui ne s'affiche jamais par accident (journaux, exceptions, débogage
   )."""
    def __init__(self, valeur): self._valeur = valeur
    def reveler(self): return self._valeur
    def __repr__(self): return "Secret(***)"
    __str__ = __repr__
```

- **Échec bruyant** au démarrage si un secret manque.
- Un fichier `.env` local est acceptable en développement **à condition** d'être dans `.gitignore`.

Le concept

Charger les secrets proprement (suite)

- On n'affiche **jamais** les secrets (ni la configuration complète) dans les journaux.

Détecter les fuites

Deux approches complémentaires :

- **Motifs connus** : formats de clés (`AKIA…` pour des clés AWS, `ghp_…` pour des jetons GitHub, `-----BEGIN PRIVATE KEY-----`).
- **Entropie** : un secret aléatoire a une forte **entropie de Shannon** par caractère, contrairement à un mot ordinaire.

Les outils comme `gitleaks`, `trufflehog` ou les protections de la plateforme (« secret scanning ») s'exécutent avant le commit (*pre-commit*) et en intégration continue.

Le concept

Détecter les fuites (suite)

```
import math
from collections import Counter

def entropie(texte):
    n = len(texte)
    return -sum((c / n) * math.log2(c / n) for c in Counter(texte).values())

print(round(entropie("hashbox"), 2), round(entropie("k8Zp3Qv9Lm2Xw7Rt5Yb1Nc4"), 2)
   )
```

Protéger les données personnelles

Les données sensibles (identité, santé, finances, localisation) sont encadrées (RGPD en Europe). Principes :

Le concept

Protéger les données personnelles (suite)

Principe	Application
Minimisation	ne collecter que ce qui est nécessaire
Limitation de durée	supprimer quand ce n'est plus utile
Chiffrement au repos et en transit	base, sauvegardes, disques, TLS
Pseudonymisation	remplacer l'identité par un identifiant
Contrôle d'accès	seules les personnes autorisées lisent
Journaux propres	pas de données personnelles inutiles
Droits des personnes	accès, rectification, effacement

Masquer dans les journaux

Le concept

Masquer dans les journaux (suite)

Avant d'écrire une ligne dans un journal, on **masque** les données sensibles : numéros de carte, e-mails, jetons, mots de passe.

Quand un secret fuit : la réponse

1. **Révoquer** et **remplacer** immédiatement (c'est la seule vraie réponse).
2. **Évaluer** l'usage qui en a été fait (journaux d'audit du fournisseur).
3. **Nettoyer** l'historique si nécessaire (mais le secret reste compromis).
4. **Corriger la cause** (détection avant commit, formation).
5. **Documenter** (retour d'expérience).

🧠 ***Astuce*** *: traite un secret comme de l'eau chaude : on le fait circuler le moins possible, on le garde au chaud (dans un coffre), et on ne le laisse jamais à l'air libre.*

Exemple pas à pas

Exemple pas à pas

Problème : un petit détecteur de secrets, un masqueur de journaux, et une configuration qui échoue bruyamment.

```
import math, re
from collections import Counter

MOTIFS = {
    "clé AWS": re.compile(r"\bAKIA[0-9A-Z]{16}\b"),
    "jeton GitHub": re.compile(r"\bghp_[A-Za-z0-9]{36}\b"),
    "clé privée": re.compile(r"-----BEGIN (?:RSA |EC |OPENSSH )?PRIVATE KEY-----")
   ,
    "mot de passe en dur": re.compile(r"""(?i)\b(?:password|passwd|mdp|secret|toke
   n)\b\s*[:=]\s*['"][^'"\s]{6,}['"]"""),
}

def entropie(texte):
    n = len(texte)
    return -sum((c / n) * math.log2(c / n) for c in Counter(texte).values())
```

Exemple pas à pas

Exemple pas à pas (suite)

```
def chaines_suspectes(ligne, seuil=3.8, longueur_min=20):
    return [m for m in re.findall(r"[A-Za-z0-9+/=_\-]{%d,}" % longueur_min, ligne)
    if entropie(m) >= seuil]

def scanner(texte):
    trouvailles = []
    for numero, ligne in enumerate(texte.splitlines(), 1):
        for nom, motif in MOTIFS.items():
            if motif.search(ligne):
                trouvailles.append((numero, nom))
        for _ in chaines_suspectes(ligne):
            trouvailles.append((numero, "chaîne à forte entropie"))
    return trouvailles

CODE = '''
import os
DEBUT = "hashbox"
AWS_ACCESS_KEY_ID = "AKIAIOSFODNN7EXAMPLE"
```

Exemple pas à pas

Exemple pas à pas (suite)

```
password = "azerty123"
URL = "https://exemple.org/api/v1/cours"
autre = "k8Zp3Qv9Lm2Xw7Rt5Yb1Nc4Hd6Gs0Jf"
cle = os.environ["HASHBOX_API_KEY"]
'''
for numero, nom in scanner(CODE):
    print(f"ligne {numero} : {nom}")
```

Lecture : le scanner repère la clé AWS (motif), le mot de passe en dur (motif) et la chaîne aléatoire (entropie). Il **ignore** l'URL, `DEBUT` et la lecture de l'environnement. (Les fausses alertes sont inévitables : on les gère avec des listes d'exceptions explicites.)

Masquer dans les journaux :

Exemple pas à pas

Exemple pas à pas (suite)

```
import re

MASQUES = [
    (re.compile(r"\b\d{4}[ -]?\d{4}[ -]?\d{4}[ -]?(\d{4})\b"), r"**** **** **** \1
   "),
    (re.compile(r"[\w.+-]+@([\w-]+\.[\w.-]+)"), r"***@\1"),
    (re.compile(r"(?i)(authorization|token|password|mdp)\s*[:=]\s*\S+"), r"\1=***"
   ),
]

def masquer(ligne):
    for motif, remplacement in MASQUES:
        ligne = motif.sub(remplacement, ligne)
    return ligne

print(masquer("paiement 4111 1111 1111 1111 pour ada@exemple.fr, Authorization: Be
   arer abc123"))
# paiement **** **** **** 1111 pour ***@exemple.fr, Authorization=***
```

Exemple pas à pas (suite)

Configuration qui refuse de démarrer sans secret :

```
import os

class Secret:
    def __init__(self, valeur): self._valeur = valeur
    def reveler(self): return self._valeur
    def __repr__(self): return "Secret(***)"
    __str__ = __repr__

def charger(env):
    manquants = [nom for nom in ("HASHBOX_DB_URL", "HASHBOX_API_KEY") if not env.g
   et(nom)]
    if manquants:
        raise RuntimeError(f"configuration incomplète : {', '.join(manquants)}")
    return {"db": Secret(env["HASHBOX_DB_URL"]), "api": Secret(env["HASHBOX_API_KE
   Y"])}
```

Exemple pas à pas (suite)

```
config = charger({"HASHBOX_DB_URL": "postgres://u:p@h/db", "HASHBOX_API_KEY": "sk_
   live_exemple"})
print(config)                  # {'db': Secret(***), 'api': Secret(***)} : jamais
    de valeur
try:
    charger({})
except RuntimeError as e:
    print(e)
```

À toi, avec aide

Améliore le scanner de secrets :

1. ajoute le motif des **chaînes de connexion** `postgres://utilisateur:motdepasse@hote/base` (secret dans l'URL) ;
2. ajoute une **liste d'exceptions** (`# noscan` en fin de ligne pour ignorer une ligne volontaire) ;
3. teste sur un texte qui contient 3 vrais secrets, 3 fausses alertes possibles (UUID, hachage de commit Git, URL) et une ligne marquée `# noscan`.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Motif : `(?:postgres|mysql|mongodb)://[^:\s]+:[^@\s]+@`.

Indice 2

Ignore la ligne si elle contient `noscan`.

Indice 3

Un hachage Git (40 caractères hexadécimaux) a une entropie assez forte : exclus-le avec un motif `^[0-9a-f]{40}$`.

À toi, avec aide · réponse

Réponse

```
import math, re
from collections import Counter

MOTIFS = {
    "clé AWS": re.compile(r"\bAKIA[0-9A-Z]{16}\b"),
    "mot de passe en dur": re.compile(r"""(?i)\b(?:password|passwd|mdp|secret|toke
   n)\b\s*[:=]\s*['"][^'"\s]{6,}['"]"""),
    "secret dans une URL": re.compile(r"(?:postgres|mysql|mongodb)://[^:\s/]+:[^@\
   s]+@"),
}
EXCLUSIONS = [re.compile(r"^[0-9a-f]{40}$"), re.compile(r"^[0-9a-f]{8}-[0-9a-f]{4}
   -[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$")]

def entropie(texte):
    n = len(texte)
    return -sum((c / n) * math.log2(c / n) for c in Counter(texte).values())

def scanner(texte):
    resultats = []
```

Réponse (suite)

```
    for numero, ligne in enumerate(texte.splitlines(), 1):
        if "noscan" in ligne:
            continue
        for nom, motif in MOTIFS.items():
            if motif.search(ligne):
                resultats.append((numero, nom))
        for candidat in re.findall(r"[A-Za-z0-9+/=_\-]{20,}", ligne):
            if any(e.match(candidat) for e in EXCLUSIONS):
                continue
            if entropie(candidat) >= 3.8:
                resultats.append((numero, "forte entropie"))
    return resultats

TEXTE = '''
AWS = "AKIAIOSFODNN7EXAMPLE"
DB = "postgres://admin:S3cretPw@db.interne/hashbox"
secret = "k8Zp3Qv9Lm2Xw7Rt5Yb1Nc4Hd6Gs0Jf"
commit = "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12"
id = "123e4567-e89b-12d3-a456-426614174000"
```

À toi, avec aide · réponse

Réponse (suite)

```
doc = "https://exemple.org/documentation/securite"
test = "AKIAIOSFODNN7EXAMPLE"  # noscan
'''
assert sorted({n for n, _ in scanner(TEXTE)}) == [2, 3, 4], scanner(TEXTE)
print(scanner(TEXTE))
```

Les vrais secrets (lignes 2, 3, 4) sont trouvés ; l'identifiant Git, l'UUID, l'URL et la ligne `# noscan` sont ignorés.

À toi, en solo

Rotation sans interruption. Imagine que ton service utilise une clé d'API. Écris une classe `ClesRotation` qui :

1. accepte **deux clés valides en même temps** (l'ancienne et la nouvelle) pendant une période de transition ;
2. refuse une clé révoquée ;
3. journalise (sans jamais écrire la clé : seulement son empreinte tronquée) quelle clé est utilisée ;
4. permet de vérifier, avant de retirer l'ancienne clé, qu'**aucun client ne l'utilise plus** (compteurs d'usage par clé).

Teste le scénario : émission de la clé A, émission de B, usage mixte, retrait de A refusé tant que A est utilisée, puis autorisé quand les compteurs sont à zéro depuis X minutes.

À toi, en solo · réponse

Pistes de correction

```
import hashlib, hmac, secrets

class ClesRotation:
    def __init__(self):
        self.cles = {}                  # empreinte complète -> {"etat": "active"/
   "revoquee", "dernier_usage": t, "usages": n}
        self.journal = []

    @staticmethod
    def _empreinte(cle):
        return hashlib.sha256(cle.encode()).hexdigest()

    def emettre(self, maintenant):
        cle = "hb_" + secrets.token_urlsafe(24)
        self.cles[self._empreinte(cle)] = {"etat": "active", "dernier_usage": None
   , "usages": 0, "emise": maintenant}
        return cle

    def verifier(self, cle, maintenant):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        e = self._empreinte(cle)
        info = self.cles.get(e)
        ok = bool(info and info["etat"] == "active")
        if ok:
            info["usages"] += 1
            info["dernier_usage"] = maintenant
        self.journal.append((maintenant, e[:8], "ok" if ok else "refusée"))      #
    jamais la clé, seulement 8 caractères du haché
        return ok

    def retrait_possible(self, cle, maintenant, silence=600):
        info = self.cles[self._empreinte(cle)]
        return info["dernier_usage"] is None or maintenant - info["dernier_usage"]
    >= silence

    def revoquer(self, cle, maintenant, forcer=False):
        if not forcer and not self.retrait_possible(cle, maintenant):
            return False
        self.cles[self._empreinte(cle)]["etat"] = "revoquee"
```

À toi, en solo · réponse

Pistes de correction (suite)

```
        return True

r = ClesRotation()
a = r.emettre(0)
assert r.verifier(a, 10)
b = r.emettre(100)                                  # transition : A et B val
   ides
assert r.verifier(a, 110) and r.verifier(b, 120)
assert not r.revoquer(a, 130)                       # A encore utilisée il y
   a 20 s : retrait refusé
assert r.verifier(b, 400)                           # les clients migrent ver
   s B
assert r.revoquer(a, 800)                           # A silencieuse depuis 69
   0 s : retrait autorisé
assert not r.verifier(a, 810)                       # A est maintenant refusé
   e
assert all(a not in str(ligne) for ligne in r.journal)  # la clé n'apparaît jamais
    dans le journal
print(r.journal[-3:])
```

À toi, en solo · réponse

Pistes de correction (suite)

La rotation sûre repose sur **deux clés valides pendant la transition**, une surveillance de l'usage, puis un retrait. Si un secret est **déjà compromis**, on le révoque immédiatement avec `forcer=True`.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Secret dans le dépôt	Les robots le trouvent en quelques minutes	Variables d'environnement ou coffre ; scanner avant commit
Croire qu'effacer le commit suffit	L'historique et les copies existent	Révoquer et remplacer le secret
Même secret partout	Une fuite compromet tout	Un secret par environnement et par usage
Journaliser des données sensibles	Les journaux sont lus par beaucoup de monde	Masquer avant d'écrire

Mini-boss

Mini-boss

Chaîne anti-fuite de HashBox. Construis un petit outillage :

1. un **scanner de secrets** (motifs + entropie + exceptions + rapport) utilisable comme *hook* de pré-commit : il lit les fichiers modifiés (simulés) et renvoie un code de sortie non nul si un secret est trouvé ;
2. un **chargeur de configuration** qui valide la présence, le format et la force minimale des secrets (longueur, entropie) et refuse de démarrer sinon ;
3. un **filtre de journalisation** (classe de `logging.Filter`) qui masque cartes, e-mails, jetons et mots de passe dans tous les messages et arguments ;
4. un **plan d'incident « clé fuitée »** exécutable : `incident_cle_fuitee(cle)` qui la révoque, liste ses usages récents (journal simulé) et génère un rapport ;
5. 25 tests, dont des fausses alertes volontaires, et la vérification qu'aucun secret ne se retrouve dans les messages d'erreur ni dans la représentation textuelle des objets de configuration.

Mini-boss · réponse

Pistes de correction

Un bon travail : le hook utilise `sys.exit(1)` si le scanner renvoie des trouvailles; le chargeur lève une exception qui cite les **noms** des secrets problématiques (jamais leurs valeurs); le filtre `logging.Filter` surcharge `filter(record)` pour réécrire `record.msg` et `record.args` avant formatage; `Secret.__repr__` renvoie `Secret(***)`; l'incident révoque via le gestionnaire de rotation, calcule les usages après la date de fuite supposée et produit un rapport (clé empreinte tronquée, nombre d'usages, adresses sources, recommandations). Les tests de non-fuite capturent la sortie de `repr(config)`, des exceptions et des journaux et vérifient l'absence de la valeur des secrets.

Quiz éclair

1. Un secret versionné dans un dépôt doit être considéré comme :

A. protégé par l'historique
B. inoffensif s'il est ancien
C. compromis

2. Où stocker les secrets d'une application ?

A. dans un coffre ou des variables d'environnement, hors du dépôt
B. dans le code source
C. dans le README

3. La rotation d'un secret consiste à :

A. le chiffrer

Quiz éclair

Quiz éclair (suite)

B. le remplacer régulièrement (et rapidement en cas de fuite)
C. le supprimer

4. 🔁 *Révision mission 04* : Un sel (salt) sert à :

A. chiffrer le mot de passe
B. accélérer le hachage
C. empêcher les tables précalculées et distinguer les empreintes identiques

5. 🔁 *Révision mission 04* : Pour générer un jeton imprévisible en Python :

A. `secrets`
B. `random`
C. `time`

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quels secrets partages-tu sans y penser (mots de passe par messagerie, fichiers dans un partage) ?
2. Si tu devais changer tous tes mots de passe importants en une heure, serais-tu capable de le faire ? Qu'est-ce qui te manquerait ?

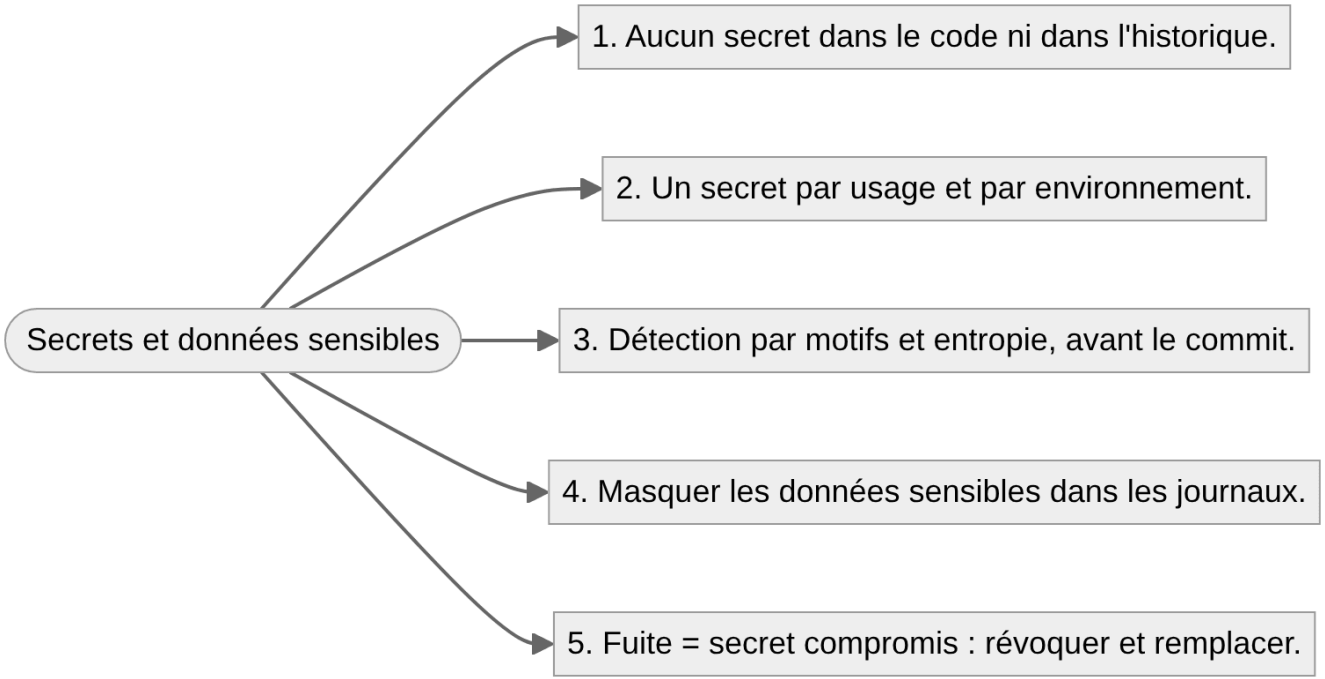
À retenir

À retenir

1. Aucun secret dans le code ni dans l'historique.
2. Un secret par usage et par environnement.
3. Détection par motifs et entropie, avant le commit.
4. Masquer les données sensibles dans les journaux.
5. Fuite = secret compromis : révoquer et remplacer.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien des secrets

Prochaine mission : Sécurité web et API

