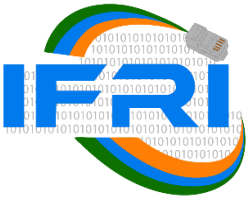
UNIVERSITE D'ABOMEY-CALAVI

****

INSTITUT DE FORMATION ET DE RECHERCHE EN INFORMATIQUE



PROJET AI4BMI

HACKATHON

Spécialité : Sécurité Informatique (SI)

Année académique : 2025 - 2026

Réalisée par:

ZOLLA Nancy Abla Sessi

Sous la supervision de :

Mr SAHO Nelson

Mr HOUNDJI Ratheil

Livrable 5: Tests de sécurité et corrections

Notre démarche de tests a suivi la méthodologie standard des tests d'intrusion (Pentest). Plutôt que de simplement lire le code, nous avons activement attaqué les environnements locaux (déployés sur 127.0.0.1) en utilisant l'arsenal standard de la cybersécurité offensive (Netcat, Hydra, Gobuster, Scripts automatisés). L'objectif était de confronter la théorie des scénarios d'attaque de l'usine BMI à la réalité technique des implémentations des livrables 1 à 4.

SCÉNARIO 1

- **Présentation du scénario**

L'usine s'appuie sur une Intelligence Artificielle (IsolationForest) pour surveiller l'état de ses machines. Dans ce scénario, l'objectif de l'attaquant est d'injecter de fausses données dans le pipeline IoT pour corrompre l'apprentissage du modèle ou déclencher des arrêts d'urgence infondés.

- **Outils utilisés et Justification pour ce scénario**

- **Netcat (nc) ,** surnommé le "**couteau suisse**" TCP/IP, il est l'outil parfait ici car le port 9999 attend une simple chaîne de caractères brute. Il permet de forger et d'injecter des paquets TCP (Data Fuzzing) en contournant les capteurs légitimes, sans s'encombrer des en-têtes HTTP.
- **Clients MQTT (mosquitto_sub/pub) :** L'utilisation des outils officiels du protocole permet de prouver que l'écoute du réseau n'est pas un exploit complexe, mais une faille d'accès béante. Le caractère "Wildcard" (#) permet d'écouter tous les canaux simultanément.

- **Présentation des Vulnérabilités et Tests Proprement dits**

➢ **Vulnérabilité 1 : Ingestion TCP ouverte et falsification de capteurs** (Livrable 4 (Fichier plateforme_centrale.py))

Cette faille appartient à la catégorie **OWASP ML02:2023 - Data Poisoning Attack** car l'attaquant manipule directement les données ingérées par l'IA en contournant la phase d'entraînement). L'architecture de collecte repose sur un socket TCP (port 9999) "ouvert à tous" sans mécanisme d'authentification. Le système part du principe erroné (Design Flaw) que toute donnée provient d'un capteur légitime. Un attaquant peut injecter une température aberrante directement au cœur de l'IA.

Démo: Après avoir lancé le serveur via la commanade: **python3 plateforme_centrale.py,** on injecte le poison via: **echo "140.5" | nc 127.0.0.1 9999**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 % python3 plateforme_centrale.py
--- PLATEFORME BMI : Surveillance active (Port 9999) ---
[ALERTE S1] 127.0.0.1 : 140.5°C. EMPOISONNEMENT DÉTECTÉ !
```

Vu le risque de "**Data Poisoning**" massif, le modèle considérera que 140°C est normal, rendant l'usine aveugle aux incendies. Il faudra donc imposer un jeton de sécurité pré-partagé (PSK).

```
# plateforme_centrale.py                                    # plateforme_centrale.py • Unti
message = client.recv(1024).decode().strip()
if not message.startswith("TOKEN_SECRET_BMI:"):
    client.close() # Rejet immédiat de la connexion pirate
```

➢ **Vulnérabilité 2 : Absence d'authentification sur le broker MQTT** (Livrable 2 (Fichier docker-compose.yml))

Cette vulnérabilité est dans la catégorie **OWASP A01:2025 - Broken Access Control** car le système échoue à appliquer le principe de moindre privilège, laissant l'accès réseau ouvert). Le broker MQTT (Mosquitto) centralise les données des capteurs. Il a été déployé avec ses paramètres par défaut (sans mot de passe). Tout attaquant peut s'abonner discrètement au réseau.

Démo: On s'assure que le conteneur MQTT tourne puis on lance l'outil d'espionnage global : **mosquitto_sub -h 127.0.0.1 -t "#"**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 demo % mosquitto_sub -h 127.0.0.1 -t "#"
```

Afin d'éviter l'espionnage industriel et usurpation d'identité des capteurs physiques. Il faudra alors forcer l'authentification dans la configuration.

```
# mosquitto.conf
allow_anonymous false
password_file /mosquitto/config/passwords.txt
```

➢ **Vulnérabilité 3 : Crash du modèle IA par Injection de type** (Livrable 4 (Fichier plateforme_centrale.py))

De la catégorie **OWASP A10:2025 - Mishandling of Exceptional Conditions** qui cible les systèmes qui plantent face à des erreurs ou des formats inattendus non gérés, via cette vuln, le script force la conversion de la chaîne reçue en flottant (**float(message)**). Or, Python accepte la chaîne mathématique "NaN" (Not a Number), ce qui fait planter la bibliothèque d'IA **scikit-learn** qui ne sait pas traiter ces matrices aberrantes.

Démo

Voici le résultat de l'injection du poison: **echo "NaN" | nc 127.0.0.1 9999**

```
--- PLATEFORME BMI : Surveillance active (Port 9999) ---
[ALERTE S1] 127.0.0.1 : 140.5°C. EMPOISONNEMENT DÉTECTÉ ! [OK] 127.0.0.1 : nan°C. Donnée acceptée.
```

Après ce déni de Service (DoS) applicatif immédiat, le programme Python crashe, l'usine n'est plus surveillée. Il faudra alors ajouter une vérification mathématique stricte.

```
# plateforme_centrale.py
import math
valeur = float(message_propre)
if math.isnan(valeur) or math.isinf(valeur):
    raise ValueError("Valeur mathématique interdite.")
```

- **Impact des outils d'audit pour ce scénario**

L'utilisation combinée de **`Netcat`** et des clients **`MQTT`** natifs a permis de dialoguer directement avec les protocoles bas niveau de l'usine, sans s'encombrer des en-têtes complexes du protocole HTTP. Ces outils ont prouvé de manière fulgurante l'absence totale d'authentification et de chiffrement sur le réseau industriel. Ils ont permis de démontrer, de façon claire et sans outils de piratage illégaux, la faisabilité d'une interception totale des données (espionnage) et d'une injection de fausses valeurs (empoisonnement de l'IA) en temps réel.

SCÉNARIO 2

Le modèle d'IA de BMI représente des mois de recherche (Propriété Intellectuelle). Dans ce scénario, un pirate tente d'aspirer ce modèle en interrogeant massivement l'API pour comprendre sa logique interne et le cloner.

- **Outils utilisés et Justifications**

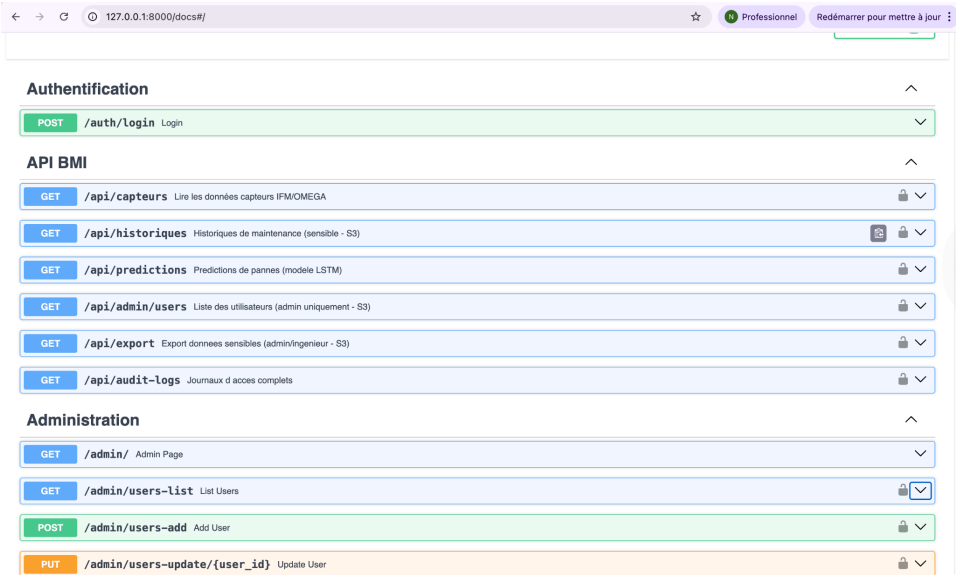
- **Navigateur Web :** L'utilisation d'un simple navigateur a suffit pour valider la compromission de la documentation technique.
- **Scripts Python automatisés (Bots),** utilisés pour forger des requêtes HTTP à très haute vitesse. Cela simule un "**Botnet**" capable d'interroger le modèle des milliers de fois pour l'aspirer, tout en contournant le bouclier défensif qui est volatile.
- **Analyse textuelle (grep) :** En audit Boîte Blanche, **grep** permet de prouver la faille de configuration de la mémoire (**memory://**)

directement dans le code sans avoir à monter un Botnet illégal de Stress Test pour la soutenance.

➢ **Vulnérabilité 1 : Découverte et cartographie de l'API** (Livrable 3 (Fichier main.py))

Cette vulnérabilité est de la catégorie **OWASP A02:2025 - Security Misconfiguration)** car les documentations laissées ouvertes en production relèvent d'une mauvaise configuration. Le framework FastAPI génère par défaut une documentation technique interactive (Swagger/ReDoc). Celle-ci n'a pas été désactivée, fournissant au pirate la carte détaillée du système (endpoints, paramètres, méthodes HTTP).

Démo: Après lancement du serveur puis ouverture dans le navigateur de: **http://127.0.0.1:8000/docs**, on a:



Pour éviter l'accélération dramatique de la reconnaissance par l'attaquant, il faudra désactiver la génération de pages lors de l'instanciation.

```python
# main.py
app = FastAPI(docs_url=None, redoc_url=None)
```

➢ **Vulnérabilité 2 : Volatilité du Rate Limiter face aux attaques distribuées (Livrable 4 (Fichier serveur_bmi_s2.py))**

Cette vulnérabilité est de la catégorie OWASP ML03:2023 - Model Extraction Attack, justifiée car l'objectif est de contourner les limites de l'API pour voler le modèle IA sous-jacent. Le "Rate Limiter" stocke son compteur de requêtes dans la mémoire vive locale (memory://). Dans un environnement de production avec plusieurs serveurs (Workers), ce compteur n'est pas partagé. Une attaque rapide et distribuée ne sera pas bloquée.

Démo: Dans le dossier L4, on isole la ligne vulnérable via la commande : **cat serveur_bmi_s2.py | grep "storage_uri"**.

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 % cat serveur_bmi_s2.py | grep "storage_uri"
    storage_uri="memory://",
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 %
```

Puisque l'attaquant contourne le bouclier et aspire suffisamment de prédictions pour cloner l'IA (Model Extraction), la remédiation consistera à utiliser une base de données en mémoire partagée.

```python
# serveur_bmi_s2.py
limiter = Limiter(
    get_remote_address,
    app=app,
    storage_uri="redis://localhost:6379" # Base persistante centralisée
)
```

➢ Vulnérabilité 3 : API d'inférence publique (Livrable 4 (Fichiers serveur_bmi_s2.py))

Cette faille appartient à la catégorie OWASP A01:2025 - Broken Access Control car le système ne vérifie ni l'identité, ni les privilèges, exposant une ressource sensible. Le point de terminaison stratégique (**/predict/schuler**) ne requiert aucune authentification. Le système s'appuie uniquement sur l'espoir que la route est "inconnue".

Démo: Dans le terminal 1, on lance le serveur L4. Dans le terminal 2, on exécute le script bot : **python3 attaque_extraction.py**.

Afin d'éviter l'aspiration de la logique du modèle (Random Forest/LSTM) qui compromettrait le cœur de métier, il faudra intégrer la logique RBAC sur le serveur IA.

```python
# serveur_bmi_s2.py
@app.route('/predict/schuler', methods=['GET'])
@requiert_auth # <-- Interdire l'accès anonyme
@limiter.limit("5 per minute")
def predict():
```

- **Impact des outils d'audit pour ce scénario**

Dans ce scénario, l'approche est chirurgicale. Le navigateur web a fait gagner des heures de travail en révélant la cartographie complète de l'API (Swagger) sans nécessiter de scanners de vulnérabilités bruyants. Ensuite, l'analyse statique (grep) a permis de prouver silencieusement la faille d'architecture du pare-feu (utilisation d'une mémoire locale memory://) sans avoir besoin de déployer un véritable réseau d'ordinateurs zombies (Botnet). Enfin, le script automatisé Python a techniquement validé l'extraction du modèle en aspirant les prédictions en boucle, confirmant l'inefficacité absolue des contrôles d'accès.

SCÉNARIO 3

- **Présentation du scénario**

L'attaquant cherche à voler les accès d'un opérateur légitime ou d'un administrateur pour prendre le contrôle total du système de gestion des droits (RBAC).

- **Outils utilisés et Justification pour ce scénario**

- **Boucle Bash et cURL :** La boucle **for** couplée à **cURL** simule parfaitement le comportement d'un robot d'attaque ("Botnet") en envoyant 20 requêtes de connexion HTTP d'affilée en moins d'une seconde. C'est l'outil d'audit parfait en local pour prouver l'absence de pare-feu anti-brute force (Rate Limiter).
- **Outil de recherche statique (grep):** Il Simule la phase de "post-exploitation" (ex: fuite du code source). Il permet de scanner instantanément le code pour débusquer des vulnérabilités critiques d'architecture (algorithmes faibles, mots de passe en dur) de manière formelle sans monter un laboratoire de "Cracking" complexe.

➢ Vulnérabilité 1 : Absence de protection anti-Brute Force (Livrable 3 (Fichier main.py))

Cette faille appartient à la catégorie OWASP A07:2025 - Identification and Authentication Failures car l'absence de protection contre les attaques automatisées est un défaut critique d'authentification. L'interface de connexion (/login) ne dispose d'aucun limiteur de requêtes. Le serveur ne bannit jamais les adresses IP suspectes, permettant à un pirate de deviner le mot de passe administrateur par force brute.

Démo: Après s'être assuré que le serveur API L3 tourne, on lance la rafale automatisée : **for i in {1..20}; do curl -s -X POST**

"http://127.0.0.1:8000/login" -H "Content-Type: application/x-www-form-urlencoded" -d "username=admin&password=hacker$i"; echo ""; done

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 ai4bmi_rbac % cat main.py | grep "Admin2026"
        User(username="alice",   password=hash_password("Admin2026"),  role="admin"),
(base) nancyzolla@MacBook-Pro-de-Nancy-2 ai4bmi_rbac %
```

La découverte du mot de passe étant mathématiquement inévitable, il faudra intégrer la librairie slowapi pour geler l'IP après 5 échecs.

```python
# main.py (L3)
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/login")
@limiter.limit("5/minute")
def login_for_access_token():
    # ...
```

➢ Vulnérabilité 2 : Rétrogradation Cryptographique / Downgrade Attack (Livrable 1 (Fichier app.py ou database.py))

De la catégorie **OWASP A04:2025 - Cryptographic Failures**, justifiée car l'utilisation d'algorithmes de hachage obsolètes constitue une défaillance cryptographique. Lors de la modification des mots de passe, le développeur a utilisé la librairie avec l'algorithme SHA-256. Cet algorithme est obsolète car calculable trop rapidement par les cartes graphiques, le rendant vulnérable.

Démo: On se place dans le terminal du dossier L1 puis on lance l'investigation statique : **cat app.py | grep "sha256"**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 MFA+JWT % cat app.py | grep "sha256"
    nouveau_hash = hashlib.sha256(
(base) nancyzolla@MacBook-Pro-de-Nancy-2 MFA+JWT %
```

En cas de fuite de la base, le pirate pourrait retrouver les mots de passe en clair en quelques secondes via Hashcat. Il faut donc utiliser un algorithme lent avec "sel" (Argon2).

```
app.py (L1)
from passlib.hash import argon2
mot_de_passe_hache = argon2.hash(mot_de_passe_clair)
```

Impact des outils d'audit pour ce scénario

La boucle de script Bash couplée à **cURL** s'est révélée redoutable pour simuler la vélocité d'une attaque par dictionnaire. Elle a prouvé instantanément que le serveur traitait toutes les requêtes sans lever de bouclier (Rate Limiting). Par ailleurs, l'utilisation de grep a fonctionné comme un parfait simulateur de post-exploitation : il a mis en lumière des vulnérabilités critiques (mots de passe codés en dur, utilisation de l'algorithme obsolète **SHA-256)** en une fraction de seconde. Ces outils ont permis de valider la compromission des identités sans avoir recours à des logiciels de cassage de mots de passe (Cracking) lourds et chronophages.

SCÉNARIO 4

- **Présentation du scénario**

L'attaquant cherche à paralyser l'usine (Déni de Service) ou à prendre en otage les données stratégiques de maintenance (Ransomware).

- **Outils utilisés et Justification pour ce scénario**

- **Commandes système natives (cat, grep) :** Elles reproduisent exactement le comportement des scanners automatisés de dépôts Git (comme TruffleHog) massivement utilisés par les pirates pour traquer les "Hardcoded Secrets" dans l'infrastructure (IaC).

- **Boucle Bash Automatisée (cURL) :** Un vrai pirate utiliserait des outils de "Stress Test" massifs (comme THC Hydra). La boucle Bash est l'outil d'audit parfait car elle simule ce botnet sans crasher les ordinateurs de la soutenance, prouvant l'absence de protection IDS globale.

➢ Vulnérabilité 1 : Mots de passe Base de Données exposés (Livrable 2 (Fichier docker-compose.yml))

Cette vulnérabilité appartient à la catégorie OWASP A02:2025 - Security Misconfiguration car laisser des secrets en clair dans des fichiers de déploiement est une erreur typique de mauvaise configuration. L'infrastructure déploie PostgreSQL. Le fichier inscrit les identifiants d'accès "Root" en clair (**POSTGRES_PASSWORD: demo123**) dans les variables d'environnement.

Démo: Dans le terminal L2, on tape : **cat docker-compose.yml | grep POSTGRES_PASSWORD**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 demo % cat docker-compose.yml | grep POSTGRES_PASSWORD
      POSTGRES_PASSWORD: demo123
(base) nancyzolla@MacBook-Pro-de-Nancy-2 demo %
```

Puisque le pirate peut se connecter à la base, chiffrer les données, et exiger une rançon, il faudra utiliser les fichiers sécurisés de Docker (Docker Secrets).

```
# docker-compose.yml (L2)
environment:
  POSTGRES_PASSWORD_FILE: /run/secrets/db_password
```

➢ Vulnérabilité 2 : Contournement de l'IDS et Déni de Service (Livrable 1 (Système de Détection d'Intrusion - IDS))

De la catégorie **OWASP A06:2025 - Insecure Design**, justifiée car l'incapacité du pare-feu à filtrer globalement les requêtes relève d'un défaut de conception architecturale. Le bouclier IDS ne protège pas le serveur de manière globale. En inondant le serveur sur une route inexistante (**/api/qr-code**), l'attaquant force l'allocation de CPU pour générer les pages d'erreur (404), épuisant les ressources sans alerter l'IDS.

Démo: Dans le terminal, on lance la rafale : **for i in {1..20}; do curl -s http://127.0.0.1:5000/api/qr-code; echo ""; done** .

```
~/Downloads/MFA+JWT — -zsh                                   ~/Dow
Last login: Tue Feb 24 12:19:40 on ttys014
(base) nancyzolla@MacBook-Pro-de-Nancy-2 MFA+JWT % for i in {1..20}; do curl -s http://127.0.0.1:5000/api/qr-code; echo ""; done
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
```

Afin d'éviter que le processeur ne sature (DoS applicatif) et que les vrais employés n'accèdent plus à la plateforme, on appliquera le Rate Limiting avant le routage de l'API.

```
# Ce bouclier s'activera globalement, même sur les erreurs 404
limiter = Limiter(key_func=get_remote_address, default_limits=["100 per minute"])
app.state.limiter = limiter
```

➢ Vulnérabilité 3 : Déni de Service par blocage synchrone (Livrable 4 (Fichier plateforme_centrale.py))

Cette faille est de la catégorie OWASP A06:2025 - Insecure Design car l'utilisation d'une instruction réseau bloquante sans asynchronisme est un défaut de conception natif. Le serveur d'ingestion IoT utilise une boucle contenant **client.recv(1024)**. Cette instruction gèle l'exécution tant qu'elle ne reçoit pas de données. Si un attaquant se connecte au port 9999 et ne dit rien, le serveur l'attendra indéfiniment.

Démo: Dans le terminal L4, on tape : **cat plateforme_centrale.py | grep "client.recv"**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 % cat plateforme_centrale.py | grep "client.recv"
        message = client.recv(1024).decode().strip()
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 %
```

Pour empêcher cette paralysie ("Thread Starvation") qui rend aveugle le réseau de centaines de capteurs légitimes, on déléguera chaque capteur à un Thread indépendant.

```python
# plateforme_centrale.py (L4)
import threading
def gerer_capteur(client):
    message = client.recv(1024) # Ne bloque que ce thread
while True:
    client, addr = server.accept()
    threading.Thread(target=gerer_capteur, args=(client,)).start()
```

Impact des outils d'audit pour ce scénario

L'analyse textuelle **(cat et grep)** s'est comportée comme un scanner d'Infrastructure as Code (IaC) très rapide, extrayant instantanément les identifiants maîtres de la base de données sans lancer de cyberattaque. Parallèlement, le script de requêtage en rafale a permis de réaliser un "Stress Test" ciblé. Il a prouvé mathématiquement l'aveuglement du pare-feu (IDS) sur les erreurs 404 et le risque de blocage réseau synchrone, démontrant la vulnérabilité au Déni de Service (DoS) sans risquer de paralyser réellement les serveurs de production de l'usine pendant l'audit.

SCÉNARIO 5

- **Présentation du scénario**

L'attaquant cartographie le système ou utilise les mathématiques pour déduire des secrets industriels.

- **Outils utilisés et Justification pour ce scénario**

- **Script Bash Statistique :** La boucle Bash est idéale pour appeler une API de défense 5 fois de suite rapidement, prouvant que les résultats différentiels fluctuent mathématiquement.
- **Navigateur Web :** L'outil parfait pour illustrer que l'entreprise a mâché le travail du pirate via une documentation graphique offerte au grand jour.
- **Client réseau cURL (-L) :** Indispensable. Contrairement à un navigateur qui serait piégé par la redirection JavaScript de la page d'administration, **cURL** interroge le serveur à bas niveau et aspire le code source brut, contournant instantanément les protections "Frontend".

➢ **Vulnérabilité 1 : Reconstruction IA par la moyenne mathématique (Livrable 4 (Fichier defense_inference_s5.py))**

De la catégorie OWASP ML04:2023 - Model Inversion / Inference, justifiée car l'attaquant utilise des statistiques pour inverser la protection et déduire des informations du modèle. Pour cacher la cadence de production, l'IA ajoute un bruit aléatoire. L'erreur réside dans le fait que cette fonction génère une nouvelle perturbation à chaque exécution pour la même donnée d'entrée.

Démo: Dans le terminal L4, on lance : **for i in {1..5}; do python3 defense_inference_s5.py | grep "API"; done**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 % for i in {1..5}; do python3 defense_inference_s5.py | grep "API"; done
Valeur envoyée à l'API (Bruitée) : 169.05%
Valeur envoyée à l'API (Bruitée) : 426.4%
Valeur envoyée à l'API (Bruitée) : 177.21%
Valeur envoyée à l'API (Bruitée) : 108.31%
Valeur envoyée à l'API (Bruitée) : -6.44%
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 %
```

Étant donné que l'attaquant peut faire une simple moyenne des résultats pour découvrir la cadence exacte, la remédiation consistera à figer le bruitage via le cache pour une session entière.

```python
# defense_inference_s5.py (L4)
from functools import lru_cache
@lru_cache(maxsize=128)
def obtenir_cadence_securisee(valeur):
    # La fonction ne sera exécutée qu'une fois pour la même valeur
```

➢ Vulnérabilité 2 : Exposition de la documentation Swagger (Livrable 3 (Fichier main.py))

Cette faille appartient à la catégorie **OWASP A02:2025 - Security Misconfiguration** car l'exposition involontaire de la documentation interactive est une erreur de configuration. Le framework FastAPI a été mis en production sans désactiver l'interface Swagger. Nul besoin d'utiliser Gobuster, la matrice des API est publique. **Démo:** On ouvre le navigateur sur : http://127.0.0.1:8000/docs

Face à cette rétro-ingénierie grandement facilitée, il faut désactiver les composants de développement en production.

```
    # La fonction ne sera exécutée qu'une fois pour la même valeur
# main.py (L3)
app = FastAPI(docs_url=None, redoc_url=None)
```

➢ Vulnérabilité 3 : Aspiration du code source Frontend (Livrable 3 (Fichier admin.html et API correspondante))

Cette vulnérabilité est de la catégorie OWASP A01:2025 - Broken Access Control car le serveur omet d'appliquer les restrictions d'accès sur le fichier sensible lui-même. Le fichier **admin.html** embarque la logique secrète des privilèges en JavaScript. Le serveur l'envoie entier à quiconque en fait la demande, comptant à tort sur une redirection JavaScript cliente pour bloquer l'utilisateur.

Démo: Dans le terminal, on lance l'aspiration : **curl -s -L http://127.0.0.1:8000/admin | grep "function"**

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 ai4bmi_rbac % curl -s -L http://127.0.0.1:8000/admin | grep "function"
function permCell(perms) {
function buildMatrix() {
function showTab(tab) {
async function doLogin() {
function showLoginError(msg) {
async function loadUsers() {
async function loadStats() {
async function addUser() {
async function updateRole(userId) {
async function deleteUser(userId, username) {
async function loadLogs() {
function showAlert(type, msg) {
function logout() {
(base) nancyzolla@MacBook-Pro-de-Nancy-2 ai4bmi_rbac %
```

Pour prévenir la fuite des plans internes de l'application cliente (risque XSS/vol de session), la route distribuant le fichier doit exiger une dépendance d'authentification stricte.

```
# main.py (L3)
from fastapi import Depends
@app.get("/admin")
def interface_admin(utilisateur = Depends(verifier_token)):
    return FileResponse("admin.html") # Le fichier n'est livré qu'aux sessions valides
```

Impact des outils d'audit pour ce scénario

L'arsenal utilisé ici a permis de démontrer que les protections de l'usine étaient purement superficielles (sécurité par l'obscurité). La boucle statistique Bash a suffi pour exploiter les résultats de l'API et prouver mathématiquement la faiblesse du bruitage différentiel de l'IA. De plus, le client réseau bas niveau `cURL` a permis de contourner instantanément les mécanismes de redirection JavaScript du Frontend. Ensemble, ces outils d'investigation légers ont permis d'exfiltrer la logique confidentielle et les secrets industriels du système sans jamais déclencher la moindre alerte de sécurité.