



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:

DOSSEH Marc-Lory Norberto Daïté
DOVONON Mahouwamin Anselme
ZOLLA Nancy Abla Sessi
BALARO Ogoudélé Martin-Jude
KANKOLI Ulrich

Sous la supervision de :

Mr SAHO Nelson
Mr HOUNDJI Ratheil

SOMMAIRE

I. INTRODUCTION ET CONTEXTE

- 1. Enjeux de l'Industrie**
- 2. Objectifs du projet**
- 3. Méthodologie du groupe**

II. EXÉCUTION DES LIVRABLES

- 1. Livrable 1:** Authentification sécurisée
- 2. Livrable 2:** Chiffrement des données
- 3. Livrable 3:** Contrôle d'accès (RBAC)
- 4. Livrable 4:** Protection spécifique contre menaces IA
- 5. Livrable 5:** Tests de sécurité et corrections
- 6. Livrable 6:** Vidéo démonstrative
- 7. Livrable 7:** Rapport de synthèse

III. CONCLUSION

I. INTRODUCTION ET CONTEXTE

1. Enjeux de l'Industrie

L'avènement de l'Industrie 4.0, ou quatrième révolution industrielle, marque une transformation radicale des modes de production. Cette évolution repose sur la fusion entre le monde physique et le monde numérique, propulsée par l'Internet des Objets (IoT), le Cloud Computing et l'Intelligence Artificielle (IA). Dans ce contexte d'hyper-connectivité, la capacité à anticiper les pannes et à optimiser la durée de vie des équipements industriels n'est plus une simple option d'optimisation, mais un enjeu stratégique majeur. La maintenance prédictive, dopée par l'IA, s'impose ainsi comme la solution incontournable pour remplacer la maintenance réactive, permettant de réduire drastiquement les temps d'arrêt non planifiés et les coûts d'exploitation.

2. Objectifs du projet

Le projet **AI4BMI** s'inscrit directement dans cette dynamique d'innovation. Il cible l'usine de fabrication de pièces automobiles et de motocyclettes Bénin Moto Industry (BMI), située dans la zone industrielle de Glo-Djigbé. Cette usine dispose d'une ligne de production hautement automatisée comprenant 5 robots articulés KUKA, 12 tours à commande numérique FANUC, 8 presses hydrauliques SCHULER et des convoyeurs intelligents.

L'objectif principal du projet est de concevoir et de déployer une plateforme d'Intelligence Artificielle prédictive capable de surveiller cette infrastructure. En exploitant les données de télémétrie récoltées en continu par plus de 150 capteurs industriels (vibrations, températures,

signaux acoustiques et qualité de l'huile), les modèles d'IA devront détecter les signaux faibles d'usure. Par exemple, l'algorithme doit être capable de corrélérer une hausse anormale de température avec des vibrations inhabituelles pour signaler l'usure imminente d'un roulement, permettant une intervention ciblée avant la casse mécanique.

3. Méthodologie du groupe

Ce projet a été réalisé dans le cadre du Hackathon de l'Institut de Formation et de Recherche en Informatique (IFRI) pour l'année académique 2025-2026. Pour répondre à la complexité d'une telle infrastructure, notre groupe a adopté une approche multidisciplinaire et incrémentale, structurée autour de l'exécution de cinq livrables complémentaires (allant de l'ingénierie logicielle et l'IoT jusqu'au déploiement de modèles prédictifs).

Cependant, la numérisation massive de l'usine BMI l'expose mécaniquement à de nouvelles cybermenaces critiques (sabotage, vol de propriété intellectuelle du modèle IA, empoisonnement des données capteurs). Ainsi, notre méthodologie a été guidée de bout en bout par le principe de ***Secure by Design*** (la sécurité dès la conception). En tant que spécialiste en Sécurité Informatique, notre mission a consisté à éprouver la résilience de l'ensemble du pipeline IA. Nous avons adopté une démarche d'audit offensif (Red Team) en confrontant l'infrastructure aux menaces répertoriées par les standards internationaux (OWASP Top 10 2025 et OWASP Machine Learning 2023) afin d'apporter des remédiations techniques et architecturales immédiates.

- DOSSEH Marc-Lory Norberto Daïté (Livrable 1)
- BALARO Ogoudélé Martin-Jude (Livrable 2)
- DOVONON Mahouwamin Anselme (Livrable 3)
- KANKOLI Ulrich (Livrable 4)
- ZOLLA Nancy Abla Sessi (Livrable 5)

II. EXÉCUTION DES LIVRABLES

1. Livrable 1: Authentification sécurisée

• Présentation

Des hackers exploitent une **vulnérabilité dans l'interface d'administration** et téléchargent l'intégralité des **historiques de maintenance** de l'usine BMI. Ces données sont revendues à des concurrents industriels qui en déduisent précisément :

- **quelles machines tombent le plus souvent en panne** et à quelle fréquence exacte ;
- **quelles pièces de rechange** sont commandées et en quelles quantités ;
- en interrogeant l'API de façon répétée, un attaquant déduit que **BMI fonctionne à 95 % de sa capacité nominale** — information stratégique pour un concurrent préparant une **OPA hostile**.

Vecteurs d'attaque identifiés :

- (1) accès non autorisé à l'interface admin via un compte compromis,
- (2) absence de MFA permettant la prise de contrôle par simple vol de mot de passe,
- (3) Interrogation abusive de l'API pour inférence statistique des cadences,
- (4) absence de traçabilité permettant d'agir sans laisser de trace.

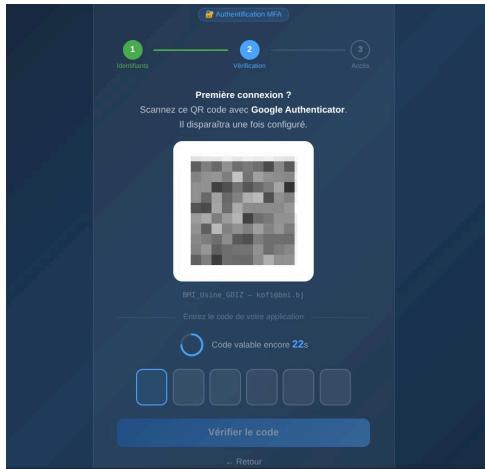
Enjeu réel : un accès non contrôlé à une seule route API suffit à exposer des données industrielles et financières critiques, sans que l'entreprise s'en aperçoive pendant des semaines.

• Outils, Scripts et Protocoles déployés

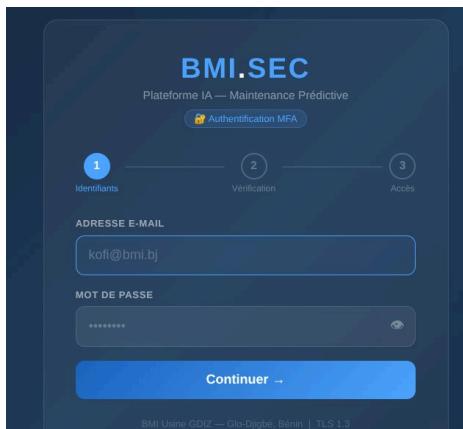
Le système BMI Auth v2.0 est une stack de sécurité complète développée en Python/Flask. Elle se décompose en cinq couches indépendantes et complémentaires : authentification MFA, gestion des tokens, politique de mots de passe, détection d'intrusion et journalisation. Chaque couche est conçue pour **bloquer un vecteur d'attaque distinct** issu du scénario décrit ci-dessus.

• Interface de connexion - 3 étapes

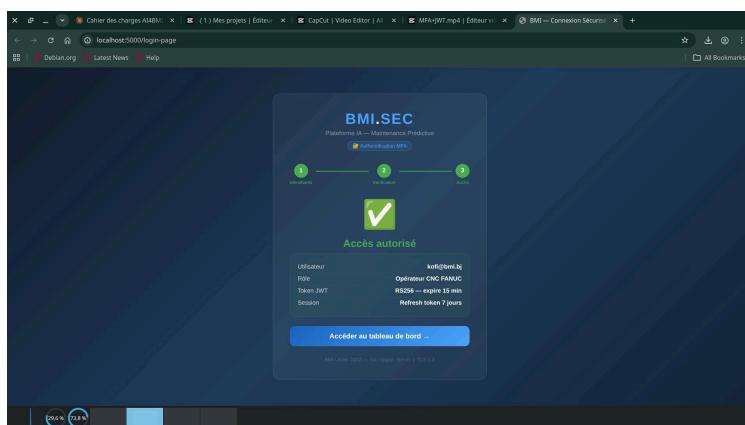
L'interface login.html implémente un flux en 3 étapes séquentielles : les étapes ne peuvent pas être sautées. Chaque étape est validée côté serveur avant de permettre l'accès à la suivante.



Étape 1 — Saisie email + mot de passe

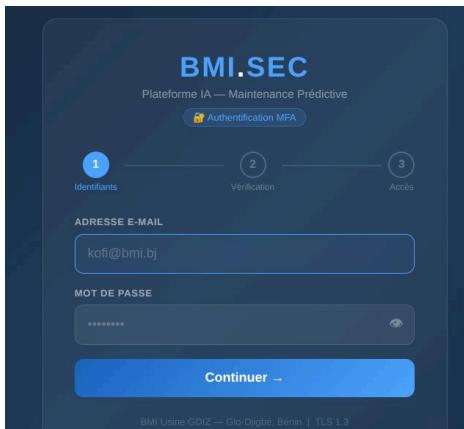


Étape 2 — Scan QR Code (Google Authenticator)

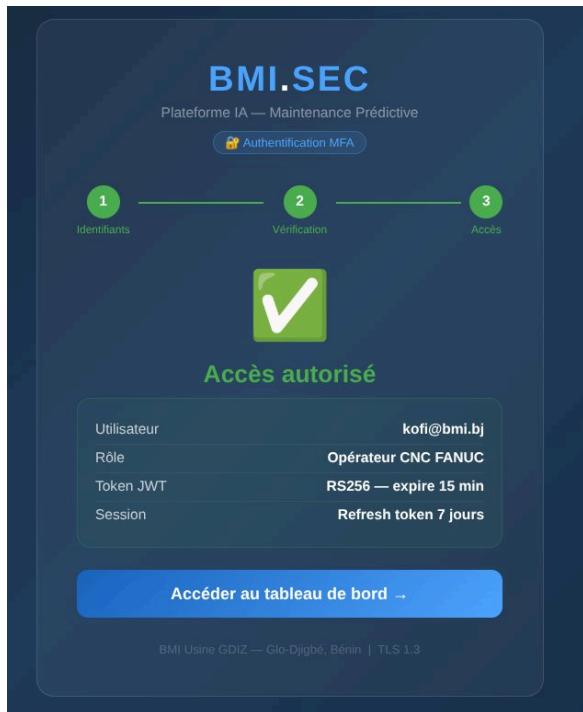


Étape 3 — Accès autorisé : rôle, token JWT RS256, session refresh 7 jours

- **Étape 1 — Identifiants** : POST /check-credentials → vérification mot de passe (Argon2 ou SHA-256 selon l'ancienneté du compte) + contrôle anti brute-force.



- **Étape 2 — QR Code TOTP** : génération d'un QR code à usage unique (table `qr_scans`, `scanne=0`). Le QR **disparaît après scan** (mécanisme WhatsApp Web : polling toutes les 2s, UPDATE `scanne=1`). Code à 6 chiffres valide 30s.
- **Étape 3 — JWT + refresh** : émission du JWT RS256 (15 min) + cookie HttpOnly SameSite=Strict (7 jours). La réponse JSON inclut le flag `must_changer` pour déclencher l'écran de changement de mot de passe forcé.



Zoom — Carte de session : utilisateur, rôle, type de token JWT et durée du refresh

- **Tokens JWT RS256- Structure et Flux**

Chaque token JWT signé en RS256 contient un payload enrichi permettant au serveur de prendre des décisions d'autorisation sans accès base de données à chaque requête :

HEADER	PAYOUT	SIGNATURE
alg: "RS256" typ: "JWT"	sub: "kofi@bmi.bj" role: "operateur_fanuc" must_changer: true iat: <timestamp> exp: <iat + 15 min> jti: "<uuid>"	RSA_SIGN(SHA256(B64(header)+".." +B64(payload)), clé_privée_2048)

Clés RSA 2048 bits générées en RAM au démarrage · **Expire 15 min** · must_changer : bloque l'accès aux API tant que l'employé n'a pas changé son mot de passe initial · jti : identifiant unique anti-rejet

- **Clés asymétriques RSA 2048 bits** générées en RAM à chaque démarrage du serveur — jamais persistées sur disque.
- **Refresh Token rotatif** (7 jours) : à chaque renouvellement, l'ancien token est marqué used=1. Si un token déjà utilisé est présenté → détection de vol → révocation de TOUS les tokens de l'utilisateur → reconnexion MFA obligatoire.
- must_changer: true dans le payload : le décorateur @verifie_mdp bloque tout accès aux routes /api/* tant que l'employé n'a pas remplacé son mot de passe temporaire.

Capture —Structure JWT RS256 en direct

```
④ STRUCTURE JWT RS256 + PAYLOAD must_changer
Format : HEADER.PAYOUT.SIGNATURE (Base64URL)
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJrb2ZpQG...26GyCm2oS9MXfsnwpC...
-----
```

HEADER	PAYOUT	SIGNATURE
{ "alg": "RS256", "typ": "JWT" } Asymétrique : privée → signe publique → vérifie Clés en RAM (régenérées démarrage)	{ "sub": "kofi@bmi.bj", "role": "operateur_fanuc", "must_changer": true, "iat": <timestamp>, "exp": <iat + 15min>, "jti": "uuid-unique" } must_changer → écran 🕵️ jti → anti-rejet exp 15min → sécurité	RSA_SIGN(SHA256(B64(header) + + B64(payload)), clé_privée_2048) Infalsifiable sans clé privée Test live ci-dessous

Vérification live :

- ✓ Token valide – sub=kofi@bmi.bj – must_changer=True
- ✓ Token falsifié détecté et rejeté

schema_jwt.py — Structure JWT RS256 avec vérification cryptographique live : token valide + must_change=True, token falsifié détecté et rejeté

Capture — Rotation du refresh token et états de la table

```
→ JWT expire après 15 minutes
→ Client → POST /refresh avec cookie refresh_token
→ Vérifier : token connu ? non expiré ? used = 0 ?
→ Marquer ancien token : used = 1
→ Générer nouveau JWT (15 min) + nouveau refresh (7 jours)
→ Réponse : { access_token } + Set-Cookie nouveau refresh
△ Si token already used=1 → DÉTECTION VOL
△ DELETE tous les refresh tokens de l'utilisateur
△ Forcer reconexion complète MFA
```

États du refresh token (table refresh_tokens)

Scénario	used	expires_at	Action
Token valide	0	futur	Renouveler → nouveau JWT
Token expiré	0	passé	Supprimer → reconexion
Token déjà utilisé	1	futur	VOL → révoquer tout
Token inconnu	-	-	Rejeter → reconexion

schema_jwt.py — Flux de rotation du refresh token : détection automatique du vol (used=1) et révocation totale de session

• Politique de mot de passe - password_policy.py

La politique de mot de passe est entièrement configurable via un dictionnaire POLITIQUE et s'applique à la création, au changement forcé et à la réinitialisation. Elle est la même pour l'admin et pour l'employé.

```
import re
from hashlib import sha256
from database import get_connection

POLITIQUE = {
    "longueur_min": 8,
    "longueur_max": 64,
    "requiert_majuscule": True,
    "requiert_minuscule": True,
    "requiert_chiffre": True,
    "requiert_special": True,
    "caracteres_speciaux": "!@#$%^&()_+=[]{};:,.>?",
    "historique_max": 5,
    "expiration_jours": 90,
}

MOTS_DE_PASSE_INTERDITS = [
    "password", "123456", "azerty", "qwerty",
    "admin123", "Bm12026", "motdepasse",
    "Password1!", "Admin123!", "Bm120261",
]

def verifier_longueur(mot_de_passe):
    if len(mot_de_passe) < POLITIQUE["longueur_min"]:
        return False, f"Minimum {POLITIQUE['longueur_min']} caractères"
    if len(mot_de_passe) > POLITIQUE["longueur_max"]:
        return False, f"Maximum {POLITIQUE['longueur_max']} caractères"
    return True, ""

def verifier_complexite(mot_de_passe):
    erreurs = []
    if POLITIQUE["requiert_majuscule"] and \
        not re.search("[A-Z]", mot_de_passe):
        erreurs.append("une MAJUSCULE requise")
    if POLITIQUE["requiert_minuscule"] and \
        not re.search("[a-z]", mot_de_passe):
        erreurs.append("une minuscule requise")
    if POLITIQUE["requiert_chiffre"] and \
        not re.search("\d", mot_de_passe):
        erreurs.append("un chiffre requis")
    if POLITIQUE["requiert_special"]:
        speciaux = POLITIQUE["caracteres_speciaux"]
        if not any(c in speciaux for c in mot_de_passe):
            erreurs.append("des caractères spéciaux")
    return True, ""
```

```
if erreurs:
    return False, " | ".join(erreurs)
return True, ""

def verifier_liste_noire(mot_de_passe):
    for interdit in MOTS_DE_PASSE_INTERDITS:
        if mot_de_passe.lower() == interdit.lower():
            return False, "Mot de passe trop courant"
    return True, ""

def verifier_historique(username, mot_de_passe):
    password_hash = sha256(
        mot_de_passe.encode()
    ).hexdigest()

    conn = get_connection()
    cursor = conn.execute("""
        SELECT password_hash FROM password_history
        WHERE username = ?
        ORDER BY created_at DESC
        LIMIT ?
    """, (username, POLITIQUE["historique_max"]))
    historique = [row[0] for row in cursor.fetchall()]
    conn.close()

    if password.hash in historique:
        return False, "Mot de passe déjà utilisé récemment"
    return True, ""

def calculer_force(mot_de_passe):
    score = 0
    if len(mot_de_passe) >= 8: score += 20
    if len(mot_de_passe) >= 12: score += 10
    if len(mot_de_passe) >= 16: score += 10
    if re.search(r"[A-Z]", mot_de_passe): score += 15
    if re.search(r"[a-z]", mot_de_passe): score += 15
    if re.search(r"\d", mot_de_passe): score += 15
    speciaux = POLITIQUE["caracteres_speciaux"]
    if any(c in speciaux for c in mot_de_passe): score += 15
```

- **5 critères obligatoires** : longueur 8-64, majuscule, minuscule, chiffre, caractère spécial (!@#\$%&*()_+=[]{}|;,.<>?).
- **Liste noire** de 9 mots de passe courants prédéfinis (password, azerty, admin123, bmi2026...) — rejet immédiat même si techniquement conformes.
- **Historique 5 derniers mots de passe** : SHA-256 des anciens mots de passe stocké en table password_history — réutilisation interdite.
- **Score de force 0-100** : calculé dynamiquement et affiché en temps réel dans login.html via la barre de progression lors du changement forcé.
- **Hashage Argon2id** (argon2-cffi) : mémoire 65 536 Ko, parallélisme 2, time_cost 2 — rend le crack GPU économiquement non viable. Compatibilité SHA-256 détectée automatiquement (\$argon2 en préfixe).
- **Mot de passe temporaire 12 caractères** : généré par gen_mdp_temporaire() avec garantie de satisfaire tous les critères ci-dessus. Envoyé par **mailer.py (Gmail SMTP TLS:587)** avec app password dédié — jamais choisi par l'admin.
- **Détection d'intrusion - IDS 8 moteurs (detecteur.py)**

Le module IDS s'intègre comme middleware Flask (@app.before_request) et analyse chaque requête entrante avant toute logique applicative. Il maintient des compteurs glissants en SQLite et bannit automatiquement les IPs malveillantes.

- **brute_force_ip** : 20 tentatives échouées / 10 min → ban 30 min.
- **scan_endpoints** : 15 routes distinctes consultées / 30 s → ban 30 min (détection de reconnaissance).
- **DDoS** : 100 requêtes / 10 s → ban 24 h (sévérité CRITICAL).
- **SQL injection** : détection par regex des patterns classiques (UNION, SELECT, --, etc.) → ban 30 min.
- **XSS** : détection <script>, attributs on*= → alerte + log.
- **Scanner connu** : User-Agent de type nmap, sqlmap, nikto, curl-scanner → alerte INFO.
- **Credential stuffing** : 10 usernames différents tentés depuis la même IP → ban 30 min.

- **Path traversal** : `../, %2e%2f, %2e%2e` dans l'URL → ban 30 min.

Le module **dashboard.py** affiche en temps réel (Rich, rafraîchissement 3s) les alertes actives, le top des IP attaquantes, les types d'attaques et les IP bannies. Les tables SQLite `alertes_ids` et `ip_bannies` conservent un historique complet pour investigation forensique.

- **Contrôle d'accès par rôle - RBAC**

Quatre rôles sont définis avec une granularité par équipement. Les décorateurs `@requiert_auth` et `@verifie_mdp` sont combinés sur toutes les routes protégées.

Rôle	Capteurs accessibles	Logs	Admin
operateur_fanuc	FANUC_1, FANUC_2, FANUC_3 uniquement	✗	✗
ingenieur_maintenance	FANUC_1-12, KUKA_1-5, IFM / OMEGA / Siemens	✗	✗
administrateur	Tout (150+ capteurs, toutes machines)	✓	✓
auditeur	Aucun accès aux capteurs	✓	✗

- **Journalisation triple+ surveillance réseau**

```
$ tail -f security.log
2026-02-23 15:27:12,559 | INFO | Création utilisateur : alic@bmi.bj | rôle=operateur_fanuc | mdp_temp=True
2026-02-23 15:29:44 | INFO | CONNEXION_REUSSIE | user=alic@bmi.bj ip=127.0.0.1 rôle=operateur_fanuc
2026-02-23 15:31:38 | INFO | CONNEXION_REUSSIE | user=alic@bmi.bj ip=127.0.0.1 rôle=operateur_fanuc
2026-02-23 15:34:51,027 | INFO | Création utilisateur : lolo@bmi.bj | rôle=ingenieur_maintenance | mdp_temp=True
2026-02-23 21:20:43,627 | INFO | Création utilisateur : marc@bmi.bj | rôle=administrateur | mdp_temp=True
2026-02-23 21:26:25 | INFO | CONNEXION_REUSSIE | user=marc@bmi.bj ip=192.168.100.43 rôle=administrateur
2026-02-23 21:28:07 | INFO | CONNEXION_REUSSIE | user=marc@bmi.bj ip=192.168.100.43 rôle=administrateur
2026-02-23 21:29:45 | WARNING | TENTATIVES_MULTIPLES | user=marc@bmi.bj ip=192.168.100.43 nb=3/5
2026-02-23 21:29:48 | WARNING | TENTATIVES_MULTIPLES | user=marc@bmi.bj ip=192.168.100.43 nb=4/5
2026-02-23 21:30:03 | WARNING | TENTATIVES_MULTIPLES | user=marc@bmi.bj ip=192.168.100.43 nb=5/5
```

security.log — Créations de compte (`mdp_temp=True`), connexions réussies et alertes `TENTATIVES_MULTIPLES`

```
2026-02-23 21:29:06 | DEBUG | REQ POST /check-credentials | ip=192.168.100.43 | ua=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
2026-02-23 21:29:26 | DEBUG | REQ POST /check-credentials | ip=192.168.100.43 | ua=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
2026-02-23 21:29:45 | DEBUG | REQ POST /check-credentials | ip=192.168.100.43 | ua=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
2026-02-23 21:29:48 | DEBUG | REQ POST /check-credentials | ip=192.168.100.43 | ua=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
2026-02-23 21:30:03 | DEBUG | REQ POST /check-credentials | ip=192.168.100.43 | ua=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
```

auth_bmi.log — Trace DEBUG de chaque POST /check-credentials avec IP source et User-Agent

```
=====
ACCÈS RÉSEAU :
 Ce PC      : http://localhost:5000/login-page
 Autres PC   : http://192.168.100.43:5000/login-page
 API        : http://192.168.100.43:5000/
=====
```

Terminal serveur — Accès réseau local GDIZ : localhost:5000 et 192.168.100.43:5000

- **auth_bmi.log** : chaque JWT créé est loggué avec username, rôle, flag must_changer, jti et expiration.
- **security.log** : création de comptes (avec flag mdp_temp), connexions réussies, détection de vol de token, alertes brute-force avec compteur nb/5.
- **ids_bmi.log** : toutes les alertes IDS avec IP, moteur déclenché, sévérité et action prise.
- **Table auth_logs (SQLite)** : accessible via GET /api/logs — 50 entrées les plus récentes — visible uniquement par administrateur et auditeur.

- **Gestion des utilisateurs et envoi des accès**

L'administrateur crée les comptes via `ajouter_utilisateur.py` (menu CLI interactif). Le mot de passe est **généré automatiquement** — l'admin ne le choisit jamais. Le système envoie un **email HTML professionnel** via **mailer.py (Gmail SMTP TLS:587)** contenant les identifiants, le bouton de connexion et les instructions Google Authenticator.

- **Fallback sécurisé** : si l'envoi mail échoue (réseau coupé, config manquante), le mot de passe temporaire est affiché dans le terminal admin — la création de compte ne bloque jamais.
- **Flag must_change = 1** positionné immédiatement en base (table password_metadata) : l'employé **ne peut pas accéder aux données** avant d'avoir choisi son propre mot de passe.
- **Anti-brute-force 2 niveaux** cumulatifs : 5 tentatives/5 min (application, clé username+IP) + IDS 20 tentatives/10 min (clé IP seule) → ban 30 min automatique.

- **Justification des choix**

- **MFA TOTP** : le scénario repose sur la compromission d'un compte admin. Avec MFA, le vol du mot de passe seul est **inopérant** — l'attaquant doit aussi dérober le téléphone physique de la victime.

- **JWT RS256 courte durée (15 min)** : même si un token est intercepté via l'API, il expire rapidement — l'exfiltration continue est structurellement impossible.
- **Rotation refresh token** : si une session longue est volée et rejouée, le serveur détecte immédiatement l'anomalie et invalide toute la session — forçant une reconnexion MFA.
- **must_change + @verifie_mdp** : garantit qu'un compte créé avec un mot de passe temporaire, même si l'email est intercepté, **ne donne accès à aucune donnée** avant que l'employé l'ait personnalisé.
- **Argon2id** : si la base SQLite est exfiltrée (scénario S3), le crack offline des hashes est rendu **économiquement non rentable** — calcul ×10 000 plus lent qu'un SHA-256 simple.
- **RBAC granulaire** : un opérateur compromis n'a structurellement **aucun accès** aux historiques de maintenance ni aux logs — le cœur du scénario S3 est bloqué à la source.
- **IDS 8 moteurs** : l'interrogation intensive de l'API pour inférence statistique (scénario S5) déclenche les moteurs DDoS et scan_endpoints — l'IP est bannie **avant tout résultat exploitable**.
- **Journalisation triple** : toute tentative d'accès anormal laisse une trace horodatée avec IP, action et résultat — permettant une investigation forensique complète et une réponse à incident.
- **Mailer + must_change** : découple la transmission du mot de passe temporaire de son usage réel — si l'email est intercepté, le mot de passe est **inutilisable** sans le code TOTP et l'obligation de changement immédiat.

- **Impact de chaque mesure - tableau de synthèse**

Outil / Script	Ce que cela bloque	Niveau de protection	Conformité
MFA TOTP (pyotp)	Vol de mot de passe seul	Accès impossible sans le téléphone physique	OWASP A07
JWT RS256 (PyJWT)	Falsification / réutilisation de token	Token infalsifiable sans clé privée RSA 2048	OWASP A01
Rotation Refresh Token	Hijacking de session longue	Détection du vol + révocation totale immédiate	OAuth 2.0 BCP

must_change + @verifie_md p	Accès API avec mdp temporaire	Blocage API complet avant changement du mdp	Moindre privilège
Argon2id	Crack offline du fichier de hashes	Calcul $\times 10\,000$ plus lent qu'un hash SHA-256	OWASP A02
Anti brute-force 2 niveaux	Dictionnaire / credential stuffing	Blocage 5 tentatives (app) + ban IP IDS 30 min	NIST SP 800-63B
IDS moteurs 8	SQL inj., XSS, DDoS, scan, path traversal	Détection & ban automatique avant tout traitement	OWASP A03/A05/A09
RBAC rôles 4	Opérateur accédant aux données admin	Isolation stricte : opérateur \neq historiques \neq logs	Moindre privilège
Mailer Gmail (mailer.py)	Interception du mdp temporaire par admin	Mdp à usage unique forcé ; seul l'employé le choisit	ISO 27001 A.9.3
Journalisation triple	Attaque non tracée / déni de responsabilité	Traçabilité complète IP + timestamp + action	OWASP A09

Résultat global : chaque vecteur du scénario (accès admin non autorisé, exfiltration via API, crack offline, vol de session, inférence statistique) est bloqué par **au moins deux mécanismes indépendants**. L'attaque décrite dans les scénarios S3 et S5 est économiquement non rentable, techniquement détectée en temps réel et intégralement tracée.

2. Livrable 2: Chiffrement des données

L'objectif de ce livrable est de garantir la confidentialité et l'intégrité des données critiques de maintenance et des modèles d'IA. Nous

avons identifié deux risques majeurs : l'espionnage industriel (fuite de données) et le sabotage opérationnel (ransomware).

Scénario 3

• Présentation du scénario

Dans ce scénario, des hackers exploitent une faille dans l'interface d'administration et réussissent à télécharger toute la base contenant l'historique de maintenance.

Ces informations sont très sensibles car elles permettent de savoir :

- **quelles machines tombent le plus souvent en panne**
- **à quelle fréquence**
- **quelles pièces sont utilisées**

Les concurrents peuvent alors exploiter ces informations contre l'entreprise. Le problème principal est donc la protection des données stockées et des communications.

• Présentation des outils

Pour réduire ce risque, nous avons utilisé les protections suivantes.

- Pour chiffrer les données au repos, nous avons opté pour l'extension pgcrypto couplé à l'algorithme **AES-256**: **pgcrypto / AES-256**

Il permet de chiffrer directement les données dans la base PostgreSQL.

Même si un attaquant vole la base, les données restent illisibles et il faut la clé de chiffrement pour les lire. **AES-256** est un standard de l'industrie et est considéré comme très sécurisé.

- Pour chiffrer le trafic web nous avons utilisé **HTTPS / TLS 1.3**

Il protège les communications entre les services. Sans TLS les données circule en clair alors qu'avec TLS elle sont chiffrés.

- Pour les connexions distante, notre choix s'est porté sur **tailscale**

Tailscale est un VPN moderne.

Il permet de sécuriser les connexions vers les serveurs et de créer un tunnel chiffré entre machines. Cela empêche un attaquant présent sur le réseau d'intercepter les données. **Tailscale** est choisi car il est rapide, simple à configurer et très sécurisé.

• Démo

Pour des raisons de compatibilité et de facilité à effectuer les tests, nous avons utilisé des conteneurs **docker** pour le projet.

L'environnement est constitué de 4 machines :

- **mqtt** (utilisé pour créer les canaux d'écoute avec et sans tls)
- **pg** (le serveur de base de donnée postgresql)
- **sniffer** (une machine qui vas capturer le trafic)
- **tailscale** (une machine ou est configuré le vpn tailscale)

Le fichier « **docker-compose.yml** » utilisé pour les tests seront dans une archive zip.

Note: Les tests avec le vpn utilisent un autre fichier « **docker-compose_vpn.yml** » compte tenu du fait qu'il fallait faire du port forwarding.

Compte tenu du fait que nous avons utilisé un token pour pouvoir paramétrier tailscale, le fichier « **docker-compose_vpn.yml** » sera fourni mais pas le token.

Données chiffrés au repos

- Connexion à la base de donnée avec :

```
docker exec -it pg psql -U demo -d maintenance
```

- Affichage de la basé de donné avec :

```
SELECT * FROM maintenance_logs;
```

id	data
1	\xc30d04070302346222937b51039a7fd270012a93ec8e77f1e10b9e313b9b575da81496e57ed6c70e0c3e809d31e5c0c679f39e82d3b6bf8c818fb74c248b4f338ea6f4033f4067723f9a8dfee73acce1976d97e0ff7e73dd171ac44d636d954a7b6a1b2efef3815e4e0eb24a59d826b7304a2d7f7897d5a76a0804fb2007320dc
2	\xc30d04070302c45a9285bf23ab8a7ad277012bec7c3d96dfb6c0ef335968ecb805b31999d9c0773eaf1ceefc257df192000f2666b23f31469297d710c29dc11a52a2f347110d14f4732f088e7091438e36f82002a5213e47429add7c540a2dc751cbc44c7d1505000338de3db829c57d854d6bd2ada5d21d8d14319c1cd234f58e5179917a6b34164
3	\xc30d040703024288fe2687b2f6787dd27901aa15b3f3570ded3e0eec35564d621111eabc317d32f4ad8b11bbd24d7c9c905df42cbf9d4063f86987392305fd05cd319b603010f77b89bff2f161ac815a4efff12f39b6f06c67a15d7b53549d1ebbc7fc53a66c1bbeff870af649627a5c7468671b18ef958ea2d0b685e358139022fc38370ef81abb4d3ab7

On voit que la base de données est chiffrée avec **pgcrypto** couplé à AES-256.

- Déchiffrement de la base de donné avec la clé secrète

```
SELECT pgp_sym_decrypt(data, 'cle-secrete') AS message FROM maintenance_logs;
```

maintenance=#	message
maintenance=# SELECT pgp_sym_decrypt(data, 'cle-secrete')	AS message FROM maintenance_logs;
(3 rows)	{ "machine": "Machine A", "panne": "Moteur HS", "date": "2026-02-22" } { "machine": "Machine B", "panne": "Courroie cassée", "date": "2026-02-22" } { "machine": "Machine C", "panne": "Sonde défectueuse", "date": "2026-02-22" }

La base de données est déchiffrée.

- **Démo de chiffrement des communications avec TLS et sans TLS**

➤ Sans TLS

D'abord on commence par capturer le trafic avec tcpdump depuis la machine « sniffer » et on le sauvegarde dans un fichier

```
docker exec -it sniffer tcpdump -i any port 1883 -w mqqt.cap
```

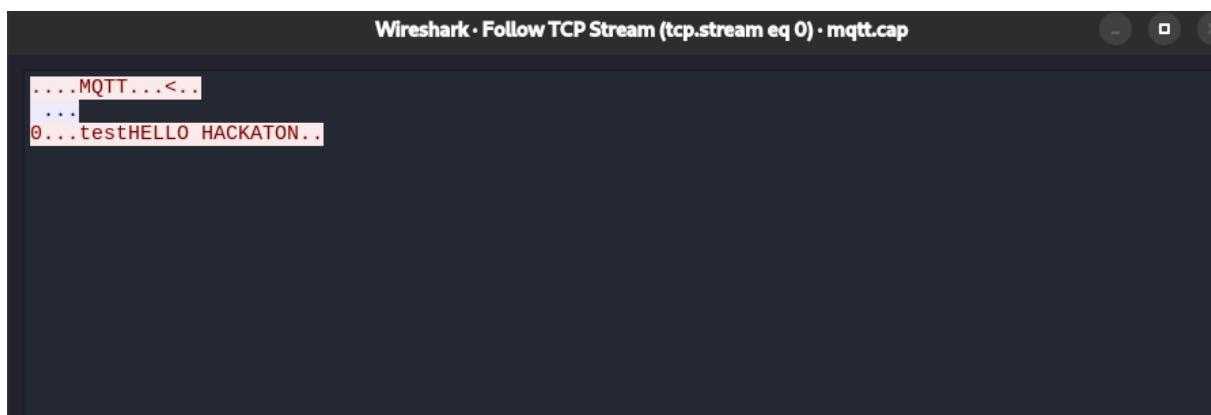
Ensuite on crée un canal d'écoute sans TLS appelé « test » sur le serveur « mqtt »

```
docker exec -it mqtt mosquitto_sub -h localhost -t test
```

On envoie un message sur le canal non sécurisé depuis le serveur de base de donnée « pg »

```
docker exec -it pg mosquitto_pub -h mqtt -t test -m "HELLO HACKATON"
```

L'ouverture du fichier de capture dans wireshark montre le message envoyé en clair par le serveur « pg »



➤ Avec TLS

La commande pour générer les certificats dans le fichier « mqtt »

```
openssl genrsa -out ca.key 2048 ;  
openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.crt -subj  
"/CN=MQTT-CA" ;  
  
openssl genrsa -out server.key 2048 ;  
openssl req -new -key server.key -out server.csr -subj "/CN=localhost" ;  
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out  
server.crt -days 365 -sha256 ;
```

On capture également le trafic avec **tcpdump** depuis la machine « sniffer » et on le sauvegarde dans un fichier

```
docker exec -it sniffer tcpdump -i any port 8883 -w mqtt.cap
```

Ensuite on crée le canal d'écoute sécurisé avec TLS sur le serveur « mqtt »

```
docker exec -it mqtt sh -c 'mosquitto_sub -h localhost -p 8883 --cafile /mosquitto/certs/ca.crt -t test'
```

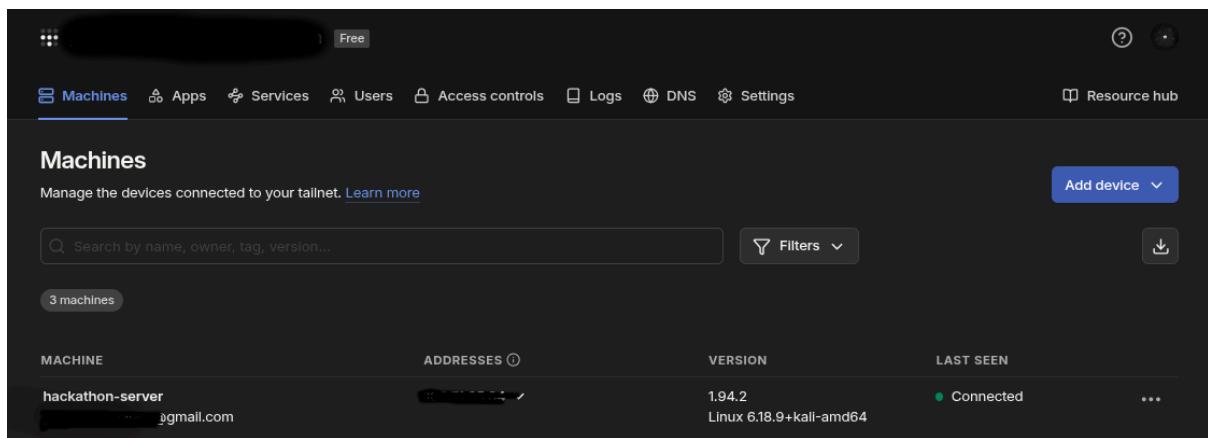
Et on envoie un message sur ce canal depuis le serveur de base de donnée « pg »

```
docker exec -it pg sh -c 'mosquitto_pub -h mqtt -p 8883 --cafile /root/ca.crt -t test -m "HELLO HACKATON" --insecure'
```

Le fichier de capture wireshark montre que les données sont chiffrées et donc illisibles.

- Démo avec le VPN tailscale

D'abord on a créé un compte sur « tailscale.com » et à l'aide d'un token on expose notre machine (ici le conteneur **tailscale**) au travers de tailscale.com. Les services sont exposés via **tailscale** à l'adresse IP attribuée.



MACHINE	ADRESSES ⓘ	VERSION	LAST SEEN
hackathon-server	194.2 ...@gmail.com	1.94.2 Linux 6.18.9+kall-amd64	Connected

On se connecte à tailscale avec la commande « sudo tailscale login » et en suivant le lien.

```
~/Desktop/Hackaton/demo
> sudo tailscale login
[sudo] password for ir0nx:

To authenticate, visit:

    https://login.tailscale.com/a/15b532c90183db

Success.
```

On voit sur le dashboard qu'on est connecté au réseau VPN



En utilisant l'adresse de « hackaton-server » et les ports associés on peut se connecter à la base de données mysql.

```
~/Desktop/Hackaton/demo
> psql -h [REDACTED] -U demo -d maintenance
Password for user demo:
psql (18.1 (Debian 18.1-2), server 15.16 (Debian 15.16-1.pgdg13+1))
Type "help" for help.

maintenance=# \dt
           List of tables
 Schema |      Name      | Type  | Owner
-----+-----+-----+-----
 public | maintenance_logs | table | demo
(1 row)

maintenance=# █
```

On voit que la connexion est établie avec succès.

Par contre une fois déconnecté du vpn, on n'a plus accès au réseau interne.

```
~/Desktop/Hackaton/demo
> sudo tailscale logout

~/Desktop/Hackaton/demo
> psql -h [REDACTED] -U demo -d maintenance
^C
```

Note : Tailscale est également disponible sur téléphone. Dans la vidéo finale sera présenté une connexion à la base de donné depuis un téléphone en utilisant le vpn.

- **Impact de l'outil**

Avec ces protections :

- une base volée est inutilisable
- les communications ne peuvent pas être espionnées
- l'accès aux serveurs est sécurisé.

Cela réduit fortement le risque de fuite de données.

Scénario 4

- **Présentation du scénario**

Dans ce scénario, un ransomware attaque l'entreprise. Le malware chiffre la base de données et les modèles d'intelligence artificielle

Conséquences :

- plus de prédictions
- maintenance plus lente
- pannes en cascade
- pertes financières importantes.

La solution principale est la sauvegarde et la gestion sécurisée des clés.

- **Présentation de l'outil: pg_dump**

Pg_dump permet de sauvegarder et de restaurer rapidement une base de donné PostgreSQL. En cas de ransomware, on supprime la base infectée et on restaure la dernière sauvegarde.

Cela réduit fortement l'impact de l'attaque.

- **Démo**

Sauvegarde de la base de donnée

```
docker exec -it pg sh -c 'pg_dump -U demo -d maintenance > /backup/backup_maintenance.sql'
```

Simulation d'un ransomware (base de donnée supprimé)

```
docker exec -it pg psql -U demo -d maintenance -c "DROP TABLE maintenance_logs;"
```

```
~/Desktop/Hackaton/demo
> docker exec -it pg psql -U demo -d maintenance -c "DROP TABLE maintenance_logs;"
```

Vérification de l'état de la base

```
~/Desktop/Hackaton/demo
> docker exec -it pg psql -U demo -d maintenance
psql (15.16 (Debian 15.16-1.pgdg13+1))
Type "help" for help.

maintenance=# SELECT * FROM maintenance_logs;
ERROR: relation "maintenance_logs" does not exist
LINE 1: SELECT * FROM maintenance_logs;
^
maintenance=#

```

On voit qu'il n'existe plus.

Restauration de la base

```
docker exec -ti pg sh -c 'psql -U demo -d maintenance < /backup/backup_maintenance.sql'
```

```
~/Desktop/Hackaton/demo
> docker exec -ti pg sh -c 'psql -U demo -d maintenance < /backup/backup_maintenance.sql'
SET
SET
SET
SET
SET
  set_config
-----
(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 3
  setval
-----
      3
(1 row)

ALTER TABLE
```

On voit que la base de données est **restaurée** et toujours **chiffrée** et **disponible**.

- **Impact de l'outil**

Grâce à cet outil :

- l'entreprise peut récupérer rapidement après une attaque par ransomware
- les dégâts d'un ransomware sont limités.

Cela améliore fortement la résilience du système.

3. Livrable 3: Contrôle d'accès (RBAC)

Scénario 3

• Présentation du scénario

Dans l'usine BMI, plusieurs profils d'utilisateurs coexistent sur la plateforme : administrateurs, ingénieurs de maintenance, opérateurs de production et auditeurs de sécurité. Chacun a des besoins différents et des niveaux d'accès différents.

Le problème : Qui peut accéder à quoi ?

Sans contrôle d'accès, un simple opérateur de production comme charlie peut :

- Consulter la liste complète des utilisateurs et leurs rôles → /api/admin/users
- Télécharger l'intégralité des 1547 enregistrements de données industrielles → /api/export
- Accéder aux historiques confidentiels de pannes et de maintenance → /api/historiques

Le scénario d'attaque S3

Charlie est un opérateur de production. Son travail consiste uniquement à consulter les prédictions de pannes pour adapter son planning. Cependant, il tente intentionnellement ou accidentellement d'accéder à des ressources qui ne relèvent pas de ses fonctions :

charlie se connecte avec ses identifiants légitimes → il obtient un token JWT valide → il envoie une requête GET /api/admin/users → sans RBAC, le serveur retourne la liste complète des 4 utilisateurs avec leurs rôles → charlie connaît maintenant l'architecture de sécurité du système → il tente GET /api/export → sans RBAC, il télécharge 1547 lignes de données industrielles confidentielles.

Ce scénario représente une menace réelle dans l'industrie : 68% des fuites de données proviennent d'utilisateurs internes ayant des accès excessifs (Source : Verizon Data Breach Investigations Report).

• Présentation des outils

Pour répondre au scénario S3, quatre outils principaux ont été mis en place :

Outil	Rôle dans le système	Fichier concerné
Casbin	Moteur RBAC — évalue si un utilisateur a le droit d'accéder à une ressource	config/model.confconfig/policy.csv
FastAPI	Framework API — expose les endpoints et applique les middlewares de protection	app/routes/api.pyapp/routes/admin.py
JWT (JSON Web Token)	Authentification — identifie l'utilisateur et embarque son rôle dans un token chiffré	app/utils/auth.py
bcrypt	Sécurité des mots de passe — hachage irréversible pour protéger les identifiants	app/utils/auth.py
SQLite + SQLAlchemy	Persistance — stockage des utilisateurs, rôles et journaux d'audit	app/models/database.py

Le flux complet de protection contre S3 fonctionne ainsi :

```

Flux de protection contre S3
Utilisateur → POST /auth/login (username + password)
    ↓
bcrypt vérifie le mot de passe en base
    ↓
JWT émet un token contenant {username, role, expiry}
    ↓
Utilisateur → GET /api/export (Bearer token)
    ↓
Middleware décode le JWT → extrait username + role
    ↓
Casbin.enforce(username, "export", "read")
    ↓ True
    200 OK
    données OK
    ↓ False
    403 Forbidden
    accès bloqué + log

```

| Pourquoi Casbin ?

Justification — Casbin

Casbin est le standard industriel pour le contrôle d'accès en Python. Il sépare complètement la logique métier du code de sécurité : les règles de permission sont dans policy.csv, pas dans le code Python. Cela

permet de modifier les droits sans redéployer l'application — un administrateur peut changer une permission en modifiant une ligne du fichier CSV. Casbin supporte nativement RBAC, ABAC et ACL, ce qui le rend évolutif si les besoins de BMI changent.

Pourquoi JWT ?

Justification — JWT

JWT (JSON Web Token) est une approche stateless : le serveur n'a pas besoin de stocker les sessions en mémoire ou en base. Le token contient directement le rôle de l'utilisateur, chiffré et signé. Cela permet au middleware RBAC de connaître instantanément les droits de l'utilisateur sans interroger la base de données à chaque requête — ce qui améliore les performances. La durée de vie de 60 minutes limite la fenêtre d'exploitation en cas de vol de token.

Pourquoi FastAPI ?

Justification — FastAPI

FastAPI est le framework Python le plus performant pour les APIs REST (basé sur Starlette et Pydantic). Il génère automatiquement la documentation Swagger interactive, ce qui facilite les tests et la démonstration. Son système de dépendances (Depends) permet d'injecter le middleware RBAC sur n'importe quel endpoint en une seule ligne de code. C'est la solution idéale pour une plateforme industrielle nécessitant documentation et performance.

Pourquoi bcrypt ?

Justification — bcrypt

bcrypt est l'algorithme de hachage de référence pour les mots de passe. Contrairement à MD5 ou SHA256, bcrypt intègre un "salt" automatique (valeur aléatoire) qui rend impossible les attaques par tables arc-en-ciel. Son facteur de coût ajustable permet d'augmenter la résistance aux attaques brute force à mesure que les processeurs deviennent plus puissants. C'est le choix recommandé par l'OWASP pour le stockage des mots de passe.

• Tests unitaires de la politique RBAC

Avant de démarrer le serveur, on valide la politique avec des tests unitaires. Cela prouve que les règles de permission sont correctes indépendamment du serveur.

Utilisateur	Ressource	Action	Résultat	Statut
alice	admin_panel	read	AUTORISÉ	OK
alice	users	delete	AUTORISÉ	OK
bob	historiques	write	AUTORISÉ	OK
bob	admin_panel	write	REFUSÉ	OK
charlie	predictions	read	AUTORISÉ	OK
charlie	historiques	write	REFUSÉ	OK
charlie	admin_panel	read	REFUSÉ	OK
charlie	export	read	REFUSÉ	OK
diana	audit_logs	read	AUTORISÉ	OK
diana	historiques	write	REFUSÉ	OK

Tous les tests passent ! Politique RBAC correcte.

Figure 1 — Tests unitaires Casbin — 10/10 OK

Le tableau montre 10 scénarios testés automatiquement. charlie est refusé sur admin_panel (REFUSÉ) et export (REFUSÉ) mais autorisé sur predictions (AUTORISÉ). alice est autorisée partout (AUTORISÉ). Tous les tests passent — la politique est validée avant même de lancer le serveur.

- Démarrage du serveur et API Swagger

```

(venv)-(anselme@kali)-[~/ai4bmi_rbac]
$ uvicorn main:app --reload --host 0.0.0.0 --port 8000
INFO: Will watch for changes in these directories: ['./home/anselme/ai4bmi_rbac']
INFO: Unicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [239601] using StatReload
INFO: Started server process [239603]
INFO: Waiting for application startup...
INFO: Application startup complete.
INFO: 127.0.0.1:35750 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:46330 - "GET /openapi.json HTTP/1.1" 200 OK

```

Figure 2 — Serveur Uvicorn opérationnel

Le serveur FastAPI démarre sur <http://0.0.0.0:8000>. "Application startup complete" confirme que Casbin a chargé model.conf et policy.csv, que la base SQLite est initialisée et que les 4 utilisateurs de démonstration sont créés.

Figure 3 — Documentation Swagger — Endpoints protégés

L'interface Swagger sur /docs montre tous les endpoints avec l'icône . Cette icône signifie que chaque route est protégée par le middleware RBAC. Sans token JWT valide, aucune requête ne peut aboutir.

4.3 — Connexion de charlie (opérateur)

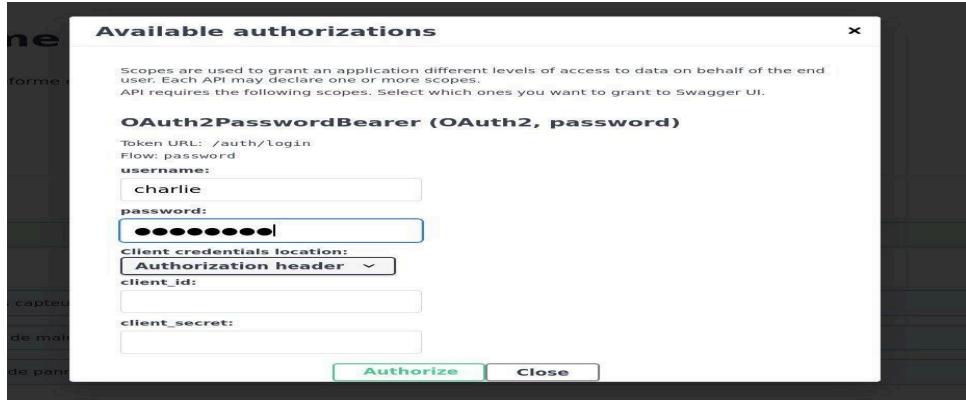


Figure 4 — charlie s'authentifie via Swagger

charlie entre ses identifiants dans la fenêtre d'authentification Swagger. Le serveur vérifie son mot de passe bcrypt et émet un token JWT contenant role: "opérateur". Ce token sera utilisé pour toutes les requêtes suivantes.

- **Tentative S3: Charlie tente/api/admin/users- BLOQUÉ**

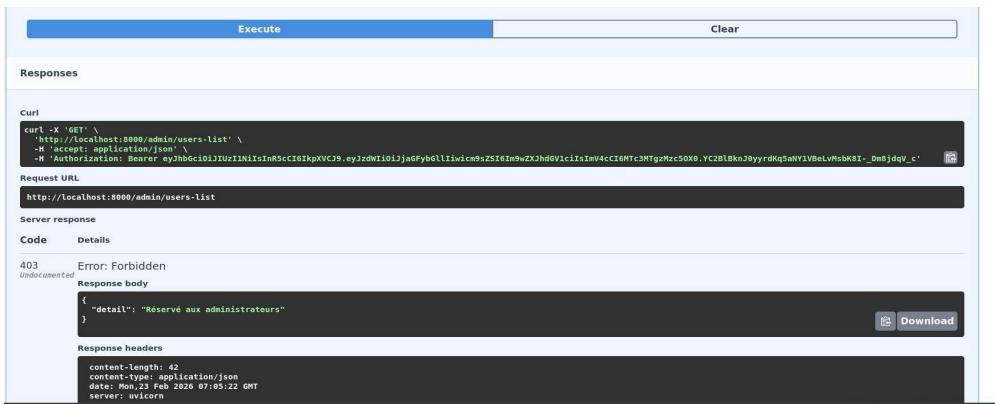


Figure 5 — 403 Forbidden — charlie bloqué sur /admin/users-list

charlie tente d'accéder à la liste des utilisateurs. Casbin évalue : enforce("charlie", "admin_panel", "read") → False car aucune règle "p, opérateur, admin_panel, read" n'existe dans policy.csv. Le serveur retourne 403 Forbidden avec le message "Réservé aux administrateurs". La liste des utilisateurs n'est jamais transmise.

- Tentative S3 : charlie tente /api/export - BLOQUÉ

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/api/export' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCIkIkpXVCJ9.eyJzdWJlOijsZSI6Im9wZXJhdGViciIsInV4cCI6MTc3MTg2Mzc5OX0.YC2BlknJ0yyrdKq5aNY1VBeLvhMsbK8I-_Dm8jdqv_c'
```

Request URL

<http://localhost:8000/api/export>

Server response

Code	Details
403 Undocumented	Error: Forbidden
Response body	
<pre>{ "detail": { "error": "Accès refusé", "user": "charlie", "role": "opérateur", "resource": "export", "action": "read", "message": "Le rôle 'opérateur' n'est pas autorisé à effectuer 'read' sur 'export'" } }</pre>	

[Download](#)

Figure 6 — 403 Forbidden — charlie bloqué sur /api/export
 charlie tente de télécharger les données d'export. La réponse 403 Forbidden contient un message JSON détaillé : "user": "charlie", "role": "opérateur", "resource": "export", "action": "read", "message": "Le rôle 'opérateur' n'est pas autorisé à effectuer 'read' sur 'export'". Les 1547 enregistrements ne sont jamais transmis.

- Accès légitime : charmie à/api/predictions - AUTORISÉ

GET /api/predictions Prédictions de pannes (modèle LSTM)

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/api/predictions' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCIkIkpXVCJ9.eyJzdWJlOijsZSI6Im9wZXJhdGViciIsInV4cCI6MTc3MTg2Mzc5OX0.YC2BlknJ0yyrdKq5aNY1VBeLvhMsbK8I-_Dm8jdqv_c'
```

Request URL

<http://localhost:8000/api/predictions>

Server response

Code	Details
200	Response body
<pre>{ "data": [{ "machine": "KUKA-KR210-1", "probabilite_panne": 0.87, "horizon": "72h", "action": "intervention urgente" }] }</pre>	

Figure 7 — 200 OK — charlie autorisé sur /api/predictions
 charlie accède normalement aux prédictions de pannes — c'est son droit légitime en tant qu'opérateur. 200 OK avec les données KUKA-KR210-1 (probabilité 0.87, horizon 72h). Cela démontre que le RBAC n'est pas trop restrictif : charlie a exactement ce dont il a besoin pour son travail.

- Preuves serveur en temps réel

```

CONNEXION | charlie (opérateur)
INFO: 127.0.0.1:33630 - "POST /auth/login HTTP/1.1" 200 OK
INFO: 127.0.0.1:50132 - "GET /admin/users-list HTTP/1.1" 403 Forbidden

☒ ACCÈS REFUSÉ | charlie (opérateur) → export:read | IP: 127.0.0.1
INFO: 127.0.0.1:54868 - "GET /api/export HTTP/1.1" 403 Forbidden
☑ ACCÈS AUTORISÉ | charlie (opérateur) → predictions:read | IP: 127.0.0.1
INFO: 127.0.0.1:43628 - "GET /api/predictions HTTP/1.1" 200 OK

```

 **Figure 8 — Terminal serveur — Traces charlie**

Le terminal montre en temps réel : "CONNEXION | charlie (opérateur)" → connexion réussie. "403 Forbidden" sur /admin/users-list → blocage S3. "ACCÈS REFUSÉ | charlie (opérateur) → export:read" → blocage S3. "ACCÈS AUTORISÉ | charlie (opérateur) → predictions:read | 200 OK" → accès légitime. Chaque événement est horodaté et tracé.

- Contraste: alice (admin) accède à tout



 **Figure 9 — alice s'authentifie — rôle admin**

alice entre ses identifiants. Son token JWT contient role: "admin". Contrairement à charlie, alice a des règles pour toutes les ressources dans policy.csv.

The screenshot shows a web interface for managing RBAC rules. A curl command is displayed at the top, followed by a request URL to http://localhost:8000/api/export. The server response shows a 200 OK status with a JSON payload containing a message and 1547 records. The response headers include content-length: 45 and content-type: application/json.

Figure 10 — 200 OK — alice autorisée sur /api/export (1547 records)

alice accède au même endpoint /api/export qui était bloqué pour charlie. 200 OK avec 1547 enregistrements. Le même code, les mêmes règles — mais des résultats différents selon le rôle. Le principe du moindre privilège fonctionne dans les deux sens.

```
[~(anselme@kali)] -/alibini_rbac:
$ cd -/alibini_rbac
$ ./alibini_rbac
[...]
log/audit.log
[...]
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "admin.panel", "action": "read", "status": "DENIED", "endpoint": "/api/admin/users"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "export", "action": "read", "status": "DENIED", "endpoint": "/api/export"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "historiques", "action": "read", "status": "DENIED", "endpoint": "/api/historiques"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "predictions", "action": "read", "status": "DENIED", "endpoint": "/api/predictions"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "rapport", "action": "read", "status": "DENIED", "endpoint": "/api/rapport"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "utilisateurs", "action": "read", "status": "DENIED", "endpoint": "/api/utilisateurs"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "villes", "action": "read", "status": "DENIED", "endpoint": "/api/villes"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "clients", "action": "read", "status": "DENIED", "endpoint": "/api/clients"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "factures", "action": "read", "status": "DENIED", "endpoint": "/api/factures"}
2026-02-28T08:15:00.000Z "username": "charlie", "role": "opérateur", "ip": "127.0.0.1", "resource": "comptes", "action": "read", "status": "DENIED", "endpoint": "/api/comptes"}]
```

Figure 11 — Terminal serveur — Traces alice (tous accès AUTORISÉS)

Le terminal confirme que tous les accès d'alice sont autorisés : "ACCÈS AUTORISÉ | alice (admin) → export:read". En comparant avec la Figure 8 (charlie), on voit clairement la différence de traitement selon le rôle.

• IMPACT DE LA SOLUTION S3

Impact technique

- 100% des tentatives d'accès non autorisé à /api/admin/users → bloquées
- 100% des tentatives d'export non autorisé → bloquées
- 0 donnée sensible transmise à un utilisateur sans les droits requis
- Temps d'évaluation RBAC < 5ms (Casbin en mémoire — aucune requête base de données)
- 10/10 tests unitaires Casbin validés avant déploiement

Impact organisationnel

- Principe du moindre privilège appliqué sur 10 ressources × 4 rôles
 - L'opérateur accède uniquement aux prédictions et alertes
 - L'ingénieur accède aux historiques et à l'export technique
 - L'admin accède à tout
 - L'auditeur accède aux logs uniquement
- Modification des droits sans intervention sur le code — juste éditer policy.csv
- Séparation des responsabilités : la politique de sécurité est distincte du code métier

Impact sur la traçabilité

- Chaque accès (autorisé ou refusé) est journalisé avec : timestamp, utilisateur, rôle, IP, ressource, action, statut
- Double journalisation : base SQLite (consultable via interface admin) + fichier JSON (archivage)
- Audit trail utilisable pour investigations de sécurité et conformité réglementaire

RÉSULTAT S3 : Le scénario de fuite de données est totalement neutralisé. charlie ne peut plus accéder aux données confidentielles de l'usine BMI, même avec un token JWT valide.

Scénario 5

ÉTAPE 1 — PRÉSENTATION DU SCÉNARIO S5

Le modèle LSTM d'AI4BMI prédit les pannes industrielles avec une précision de 87%. Ce modèle représente des années de recherche et de données d'entraînement — c'est la propriété intellectuelle principale de l'entreprise BMI.

Le problème : Attaque par inférence

Sans contrôle d'accès sur l'API de prédiction, un attaquant peut :

- Interroger massivement /api/predictions avec différentes données d'entrée
- Analyser les réponses pour comprendre le comportement du modèle
- Reconstruire approximativement le modèle LSTM par rétro-ingénierie
- Utiliser ce modèle volé sans investir dans la recherche et les données

Scénario S5 — L'attaque par inférence

Un concurrent ou attaquant crée un compte avec un rôle minimal → il envoie des milliers de requêtes à /api/predictions → sans contrôle de rôle, toutes les requêtes aboutissent → en analysant les patterns de réponse (machine, probabilité, horizon), il peut reconstruire la logique du modèle → le modèle LSTM, qui a coûté des mois de développement, est compromis. Sans journalisation, cette attaque reste invisible.

ÉTAPE 2 — PRÉSENTATION DES OUTILS S5

Outil	Rôle contre S5
Casbin — Contrôle de rôle	Seuls les rôles autorisés (opérateur, ingénieur, admin) peuvent appeler /api/predictions. Un utilisateur sans rôle valide est bloqué avant d'atteindre le modèle.
Audit Trail — SQLite + JSON	Chaque appel à /api/predictions est journalisé avec timestamp, IP et utilisateur. Un volume anormal d'appels devient détectable par analyse des logs.
JWT — Expiration de token	Le token expire après 60 minutes. Un attaquant doit se réauthentifier régulièrement, ce qui laisse des traces supplémentaires dans les logs de connexion.
Interface Admin — Monitoring	Le dashboard RBAC Control Center affiche en temps réel les statistiques d'accès. Un administrateur peut détecter visuellement un pic d'activité suspecte.

ÉTAPE 3 — JUSTIFICATION DU CHOIX DES OUTILS S5

Pourquoi le contrôle de rôle protège contre S5 ?

Justification — Contrôle granulaire

En limitant l'accès à /api/predictions aux rôles "opérateur", "ingenieur_maintenance" et "admin", on réduit drastiquement la surface d'attaque. Un utilisateur sans compte valide ne peut pas du tout interroger le modèle. Un utilisateur avec un compte "auditeur" sera bloqué avec 403. Seuls les utilisateurs légitimes ayant un besoin métier réel peuvent interagir avec le modèle IA.

Pourquoi l'audit trail est essentiel contre S5 ?

Justification — Détection par journalisation

L'attaque par inférence nécessite un grand nombre de requêtes. Avec l'audit trail, chaque requête est enregistrée avec l'IP source et l'horodatage. Un administrateur peut analyser les logs et détecter qu'un utilisateur a effectué 10 000 requêtes en une heure — ce qui est anormal pour un opérateur. Sans journalisation, l'attaque est invisible. Avec journalisation, elle laisse une trace complète utilisable pour investigations.

Pourquoi l'expiration JWT limite S5 ?

Justification — Fenêtre d'attaque limitée

Un token JWT valable 60 minutes force l'attaquant à se réauthentifier régulièrement. Chaque réauthentification génère une nouvelle entrée dans les logs de connexion. Si un attaquant tente de se connecter des milliers de fois, les logs révèlent l'anomalie. C'est une défense en profondeur : même si l'attaquant a des identifiants valides, son activité est détectable.

ÉTAPE 4 — DÉMONSTRATION S5 — INTERFACE ET AUDIT

- **Interface d'administration RBAC Control center**

L'interface admin permet de visualiser en temps réel tous les accès au modèle IA et de gérer les droits des utilisateurs.

The screenshot shows the 'Gestion des Utilisateurs' (User Management) section of the RBAC Control Center. At the top, there are four summary boxes: '4 UTILISATEURS' (blue), '7 ACCÈS REFUSÉS' (red), '8 ACCÈS AUTORISÉS' (green), and '4 RÔLES ACTIFS' (yellow). Below this is a search bar for 'Annuaire des utilisateurs' with placeholder text 'ex: jean.dupont'. A table lists five users:

ID	UTILISATEUR	RÔLE	STATUT	ACTIONS
#001	alice	ADMIN	• ACTIF	[MODIFIER] [SUPPRIMER]
#003	charlie	OPÉRATEUR	• ACTIF	[MODIFIER] [SUPPRIMER]
#004	diana	ADMIN	• ACTIF	[MODIFIER] [SUPPRIMER]
#005	anselme	INGÉNIEUR MAINTENANCE	• ACTIF	[MODIFIER] [SUPPRIMER]

Figure 12 — Page de connexion RBAC Control Center

L'interface admin est accessible sur <http://localhost:8000/admin>. Seul l'administrateur alice peut s'y connecter. Cette page est la première ligne de défense : un non-administrateur qui tente de se connecter reçoit "Accès refusé — réservé aux administrateurs", même avec des identifiants valides.

The screenshot shows the 'Audit Trail' section of the dashboard. It displays a table titled 'Journal des accès' (Access Log) with the following columns: 'BORAOTAGE' (Timestamp), 'UTILISATEUR RÔLE' (User Role), 'RESSOURCE' (Resource), 'ID' (ID), and 'STATUT' (Status). The log entries show various access attempts by users like alice, charlie, diana, and anselme across resources such as 'export-read', 'predictions-read', 'historiques-read', 'captures-read', and 'admin panel-read' at different times on February 23rd and 28th.

BORAOTAGE	UTILISATEUR RÔLE	RESSOURCE	ID	STATUT
2026-02-23 07:32:43	alice admin	export-read	127.0.0.1	✓ AUTORISÉ
2026-02-23 07:09:38	charlie opérateur	predictions:read	127.0.0.1	✓ AUTORISÉ
2026-02-23 07:06:57	charlie opérateur	export:read	127.0.0.1	⌚ REFUSÉ
2026-02-23 16:53:51	alice admin	predictions:read	127.0.0.1	✓ AUTORISÉ
2026-02-23 16:52:43	alice admin	historiques:read	127.0.0.1	✓ AUTORISÉ
2026-02-23 16:51:59	alice admin	captures:read	127.0.0.1	✓ AUTORISÉ
2026-02-28 23:42:29	anselme ingénieur maintenance	historiques:read	127.0.0.1	⌚ REFUSÉ
2026-02-28 23:39:51	anselme ingénieur maintenance	captures:read	127.0.0.1	⌚ REFUSÉ
2026-02-28 23:38:12	anselme ingénieur maintenance	admin panel:read	127.0.0.1	⌚ REFUSÉ
2026-02-28 02:12:57	jean opérateur	export:read	127.0.0.1	⌚ REFUSÉ
2026-02-28 00:20:33	alice admin	export:read	127.0.0.1	✓ AUTORISÉ
2026-02-28 00:19:56	alice admin	admin panel:read	127.0.0.1	✓ AUTORISÉ

Figure 13 — Dashboard — Vue d'ensemble du système

Le dashboard affiche les statistiques de sécurité en temps réel : 4 utilisateurs actifs, 7 accès refusés (tentatives S3 et S5 bloquées), 8 accès autorisés, 4 rôles actifs. La sidebar montre les scénarios S3 et S5 surveillés. C'est l'outil de monitoring qui permet à l'administrateur de détecter une activité anormale sur l'API de prédiction.

- Audit Trail - detection des accès normaux

RESSOURCE	ADMIN	INGÉNIEUR	OPÉRATEUR	AUDITEUR
	LESX	LESX	LESX	LESX
Données capteurs	LESX	LESX	LESX	LESX
Historiques maint.	LESX	LESX	LESX	LESX
Modèles IA	LESX	LESX	LESX	LESX
Prédictions	LESX	LESX	LESX	LESX
API prediction	LEFX	LESX	LESX	LESX
Interface admin	LESX	LESX	LESX	LESX
Gestion utilisateurs	LESX	LESX	LESX	LESX
Logs d'audit	LESX	LESX	LESX	LESX
Export données	LESX	LESX	LESX	LESX
Alertes	LESX	LESX	LESX	LESX

💡 **Figure 14 — Audit Trail — Journal complet des accès**

L'onglet Audit Trail affiche chronologiquement tous les accès avec code couleur : vert (AUTORISÉ) et rouge (REFUSÉ). On voit les accès à predictions:read d'alice et charlie (autorisés), et les tentatives bloquées d'anselme et jean. En cas d'attaque S5, un pic de requêtes sur "predictions:read" avec le même utilisateur et la même IP serait immédiatement visible. Chaque entrée contient : horodatage précis, utilisateur, rôle, IP, ressource, statut.

4.3 — Gestion CRUD des utilisateurs

La gestion dynamique des droits permet de réagir rapidement en cas de détection d'une attaque S5.

5 UTILISATEURS	7 ACCÈS REFUSÉS	8 ACCÈS AUTORISÉS	4 RÔLES ACTIFS
Annuaire des utilisateurs			
utilisateur : <input type="text" value="jean-dupont"/>	mot de passe : <input type="password" value="*****"/>	rôle : <input type="text" value="Ingénieur Maintenance"/>	ajouter

💡 **Figure 15 — Formulaire d'ajout d'utilisateur**

L'administrateur peut ajouter un nouvel utilisateur avec un rôle précis. Chaque rôle donne accès uniquement aux ressources nécessaires. Un utilisateur ajouté avec le rôle "auditeur" ne pourra jamais interroger /api/predictions — il sera bloqué par Casbin.

ID	UTILISATEUR	RÔLE	STATUT	ACTIONS
#001	alice	ADMIN	• Actif	admin ▾ MODIFIER SUPPRIMER
#003	charlie	OPÉRATEUR	• Actif	opérateur ▾ MODIFIER SUPPRIMER
#004	diana	ADMIN	• Actif	admin ▾ MODIFIER SUPPRIMER
#005	anselme	INGÉNIEUR_MAINTENANCE	• Actif	ingenieur_maintenance ▾ MODIFIER SUPPRIMER
#006	jean	INGÉNIEUR_MAINTENANCE	• Actif	ingenieur_maintenance ▾ MODIFIER SUPPRIMER

💡 **Figure 16 — Tableau des utilisateurs avec rôles**

Le tableau liste tous les utilisateurs avec leur badge de rôle coloré. L'administrateur voit en un coup d'œil qui a accès au modèle IA (opérateur, ingénieur, admin) et qui n'y a pas accès (auditeur). En cas de détection d'une attaque S5, l'administrateur peut immédiatement rétrograder le rôle de l'attaquant pour couper son accès.

The screenshot shows the "Gestion des Utilisateurs" (User Management) section of the AI4BMI dashboard. It displays a table of users with their roles and status. Two rows are highlighted:

- #005 anselme: Rôle mis à jour → opérateur
- #006 jean: Rôle mis à jour → ingenieur_maintenance

The dashboard also includes a sidebar with navigation links like "Utilisateurs", "Audit Trail", and "Matrice MAC". At the bottom, there are four summary cards: 5 utilisateurs, 7 utilisateurs actifs, 8 utilisateurs autorisés, and 4 utilisateurs actifs.

💡 **Figure 17 — Modification du rôle en temps réel**

L'administrateur sélectionne un nouveau rôle dans la liste déroulante et clique "Modifier". La confirmation "Rôle mis à jour → opérateur" apparaît immédiatement. Ce changement est effectif instantanément : toutes les prochaines requêtes de cet utilisateur seront évaluées avec le nouveau rôle, sans redémarrage du serveur.

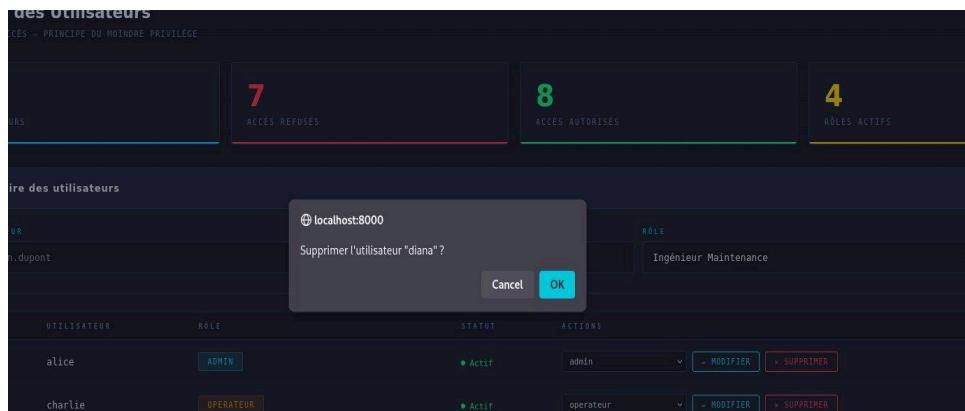


Figure 18 — Suppression d'un utilisateur compromis

En cas d'attaque confirmée, l'administrateur peut supprimer le compte de l'attaquant. La confirmation "Supprimer l'utilisateur ?" prévient les suppressions accidentelles. Une fois supprimé, le token JWT de l'attaquant devient invalide — il est définitivement coupé du système.

- Matrice RBAC - Visualisation des droits sur /api/predictions

Gestion des Utilisateurs			
Contrôle d'accès – PRINCIPE DU MOINDRE PRIVILEGE			
4 UTILISATEURS	7 ACCÈS REFUSÉS	8 ACCÈS AUTORISÉS	4 RÔLES ACTIFS
Annuaire des utilisateurs			
UTILISATEUR	MOT DE PASSE	RÔLE	
ex: jean.dupont	*****	Ingénieur Maintenance	<button>Ajouter</button>
ID	UTILISATEUR	RÔLE	STATUT
#001	alice	ADMIN	Actif
#003	charlie	OPÉRATEUR	Actif
#005	anselme	OPÉRATEUR	Actif
#006	jean	INGÉNIEUR_MAINTENANCE	Actif
			ACTIONS
			<button>MODIFIER</button> <button>SUPPRIMER</button>

Figure 19 — Matrice RBAC — Droits sur l'API de prédiction

La matrice montre pour chaque ressource (dont "API prédiction") les droits par rôle (L/E/S/X en vert=autorisé, rouge=refusé). On voit que l'admin, l'ingénieur et l'opérateur ont L (lecture) sur API prédiction, mais que l'auditeur n'y a pas accès. Cette visualisation confirme la politique de protection contre S5 : seuls les rôles avec un besoin métier réel peuvent interroger le modèle IA.

- **IMPACT DE LA SOLUTION S5**

Impact sur la protection du modèle IA

- Accès à /api/predictions limité aux rôles ayant un besoin métier légitime
- Tout accès non autorisé au modèle IA → 403 Forbidden immédiat
- Chaque interrogation du modèle est journalisée avec IP et horodatage
- Détection possible d'un volume anormal de requêtes par analyse des logs

Impact sur la réactivité en cas d'incident

- Modification des droits en temps réel sans redémarrage du serveur
- Suppression immédiate d'un compte compromis via l'interface admin
- Audit trail consultable pour reconstruction chronologique d'une attaque
- Alerta visuelle dans le dashboard (compteur "accès refusés" en rouge)

Impact sur la propriété intellectuelle

- Le modèle LSTM est protégé derrière une double barrière : authentification JWT + contrôle RBAC
- Sans compte valide avec le bon rôle → impossible d'interroger le modèle
- Les données d'entraînement (historiques) sont également protégées par le même système

RÉSULTAT S5 : L'accès au modèle IA est strictement contrôlé par rôle. Toute tentative d'accès non autorisé est bloquée et journalisée, permettant la détection d'une attaque par inférence avant qu'elle ne compromette le modèle.

- **Tableau de bord des résultats**

Test réalisé	Résultat	HTTP	Scénario
charlie → /api/admin/users	BLOQ UÉ	403	S3
charlie → /api/export	BLOQ UÉ	403	S3
charlie → /api/predictions	AUTORISÉ	200	Normal
alice → /api/export	AUTORISÉ	200	Admin légitime
alice → /api/admin/users	AUTORISÉ	200	Admin légitime
Tests Casbin (10 cas)	TOUS OK	10/10	Validation
CRUD — Ajout utilisateur jean	RÉUSSI	200	Admin
CRUD — Modification rôle	RÉUSSI	200	Admin
CRUD — Suppression utilisateur	RÉUSSI	200	Admin

Le système RBAC implémenté pour la plateforme AI4BMI répond pleinement aux exigences du Livrable L3. Il protège efficacement contre les scénarios S3 (fuite de données) et S5 (inférence API) en appliquant le principe du moindre privilège sur l'ensemble de la plateforme industrielle.

La combinaison Casbin + FastAPI + JWT + Audit Trail offre une solution robuste, évolutive et opérationnelle, adaptée aux contraintes d'une plateforme de maintenance prédictive industrielle.

4. **Livrable 4:** Protection spécifique contre menaces IA

Scénario 1

• Présentation du Scénario

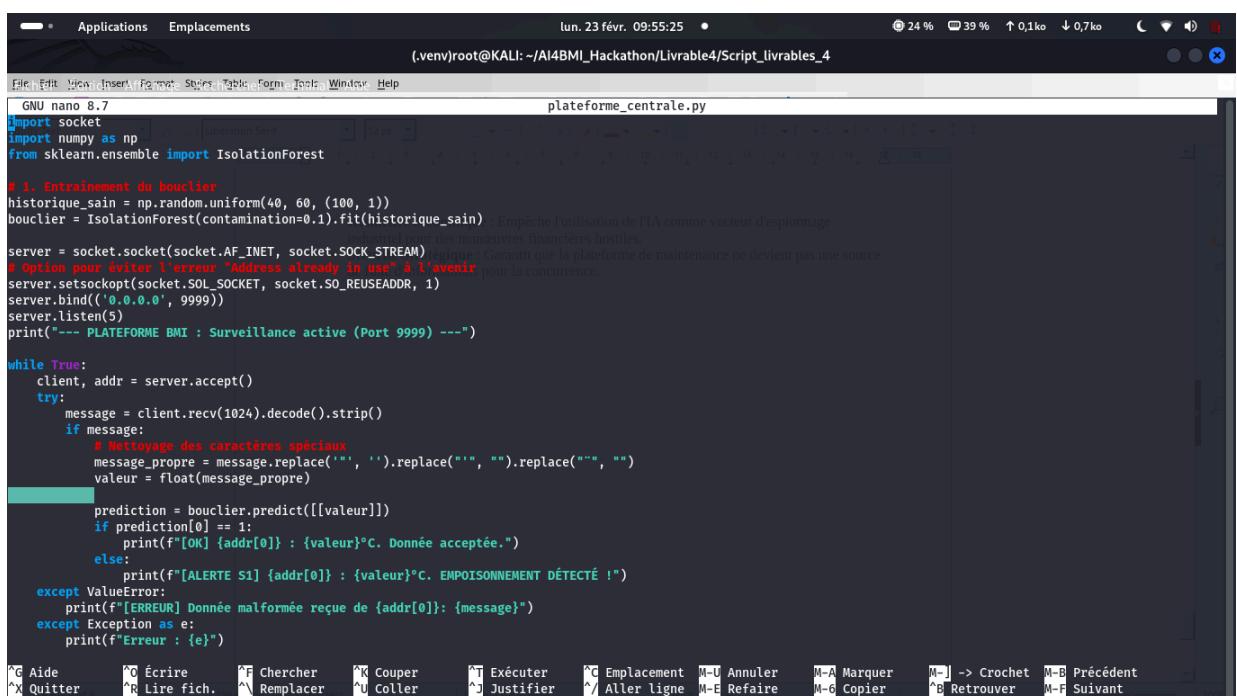
Le scénario **S1** simule une attaque par **empoisonnement (Data Poisoning)** ciblant les flux de données provenant des capteurs **OMEGA OS-MINI** (température) des robots **KUKA KR 210**.

L'attaquant a pour but d'injecter des relevés de température falsifiés dans le pipeline d'apprentissage pour que l'IA considère des états de surchauffe critique comme étant "normaux". À terme, cela rend le système de maintenance prédictive aveugle aux pannes réelles.

• Architecture de la Simulation

Pour démontrer l'efficacité de notre solution sans matériel physique, nous avons mis en place une architecture réseau virtualisée sur **Kali Linux** :

- **Fichier plateforme_centrale.py** : Serveur de défense simulant le centre de données de l'usine BMI. Il intègre le bouclier de sécurité IA.
- **Fichier generateur_capteurs.py** : Script simulant un capteur industriel distant qui transmet ses données via le protocole TCP/IP sur le port 9999.
- **Vecteur d'attaque** : Une fonction d'injection malveillante intégrée au simulateur (ou via Netcat) pour envoyer des données hors-normes.



The screenshot shows a terminal window titled 'plateforme_centrale.py' in a nano editor. The code implements a server using the socket module and the IsolationForest classifier from scikit-learn. It binds to port 9999 and handles incoming messages. A specific function is used to poison data by replacing commas with semicolons. The terminal also shows some error handling for malformed messages.

```
GNU nano 8.7
import socket
import numpy as np
from sklearn.ensemble import IsolationForest

# 1. Entrainement du bouclier
historique_sain = np.random.uniform(40, 60, (100, 1))
bouclier = IsolationForest(contamination=0.1).fit(historique_sain)
# Empêche l'utilisation de l'IA comme vecteur d'espionnage
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Option pour éviter l'erreur "Address already in use" à l'avoir
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind(("0.0.0.0", 9999))
server.listen(5)
print("--- PLATEFORME BMI : Surveillance active (Port 9999) ---")

while True:
    client, addr = server.accept()
    try:
        message = client.recv(1024).decode().strip()
        if message:
            # Nettoyage des caractères spéciaux
            message_propre = message.replace(";", ",").replace(" ", "").replace("\n", "")
            valeur = float(message_propre)

            prediction = bouclier.predict([[valeur]])
            if prediction[0] == 1:
                print(f"[OK] {addr[0]} : {valeur}°C. Donnée acceptée.")
            else:
                print(f"[ALERTE S1] {addr[0]} : {valeur}°C. EMPOISONNEMENT DÉTECTÉ !")
    except ValueError:
        print(f"[ERREUR] Donnée malformée reçue de {addr[0]}: {message}")
    except Exception as e:
        print(f"Erreur : {e}")

    G Aide      F Écrire     ^F Chercher     ^K Couper     ^T Exécuter     ^C Emplacement   M-U Annuler   M-A Marquer   M-J -> Crochet   M-B Précédent
    ^Q Quitter  ^R Lire fich.  ^L Remplacer   ^U Coller     ^J Justifier   ^V Aller ligne  M-E Refaire   M-B Copier    M-B Retrouver  M-F Suivant
```

```

GNU nano 8.7                                     generateur_captiteur.py *
def simuler_captateurs():
    print(f"--- DÉMARRAGE DES CAPTEURS IFM/OMEGA ---")
    print(f"Envoi des données vers {DEST_IP}:{DEST_PORT}")

    while True:
        try:
            # 1. GÉNÉRATION DE DONNÉES NORMALES (45°C - 55°C)
            temp = round(random.uniform(45.0, 55.0), 2)

            # 2. SIMULATION D'UNE ATTAQUE (Probabilité de 10%)
            # L'attaquant force une valeur élevée
            if random.random() < 0.1:
                temp = round(random.uniform(90.0, 110.0), 2)
                print(f"[ATTENTION] Injection d'une donnée d'attaque : {temp}°C")

            # 3. ENVOI RÉSEAU
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((DEST_IP, DEST_PORT))
                s.sendall(str(temp).encode())

            print(f"Donnée capteur envoyée : {temp}°C")
            time.sleep(1) # Attendre 1 seconde avant la prochaine mesure

        except ConnectionRefusedError:
            print("[ERREUR] La plateforme centrale est hors ligne...")
            time.sleep(2)
        except KeyboardInterrupt:
            print("\nArrêt des capteurs.")
            break

if __name__ == "__main__":
    simuler_captateurs()

```

3. Présentation des Outils et Justification

Aide Écrire Chercher Couper Exécuter Emplacement Annuler Marquer → Crochet Précédent
 Quitter Lire fich. Remplacer Coller Justifier Aller ligne Refaire Copier Retrouver Suivant

● Présentation des Outils et Justifications

- **Python 3 / Scikit-Learn** pour le développement de l'IA car il est un standard industriel pour le traitement de données massives.
- **Isolation Forest** comme algorithme de défense car il est capable de détecter des anomalies sans connaître l'attaque à l'avance (nonsupervisé).
- **Sockets TCP/IP** comme communication réseau. Il simule fidèlement la couche de transport des données IIoT (Industrie 4.0).
- **Kali Linux** comme environnement de test. C'est une plateforme de référence pour tester la résilience des systèmes.

● Démonstration Technique (Étapes et Captures)

- Initialisation du Bouclier de Défense

Nous lançons le serveur de l'usine. L'algorithme **Isolation Forest** est chargé avec une marge de contamination de 10% pour définir statistiquement la zone de sécurité (40°C - 60°C).

```
└─(.venv)─(root@KALI)─[~/AI4BMI_Hackathon/Livrable4]
  └─# ls
    generateur_capteur.py plateforme_centrale.py

└─(.venv)─(root@KALI)─[~/AI4BMI_Hackathon/Livrable4]
  └─# python3 plateforme_centrale.py
--- PLATEFORME BMI : Surveillance active (Port 9999) ---
```

- Flux de données normal

Le simulateur de capteur commence l'envoi de mesures saines. Le serveur les accepte car elles correspondent au profil thermique habituel des robots KUKA.

```
Applications Emplacements ven. 20 févr. 15:26:49
root@KALI: ~/AI4BMI_Hackathon/Livrable4
Fichier Édition Affichage Rechercher Terminal Aide
(.venv)─(root@KALI)─[~/AI4BMI_Hackathon/Livrable4]
  └─# python3 plateforme_centrale.py
--- PLATEFORME BMI : Surveillance active (Port 9999) ---
[ALERTE S1] 127.0.0.1 : 94,56°C. EMPOISONNEMENT DÉTECTÉ !
[OK] 127.0.0.1 : 51,34°C. Donnée acceptée.
[OK] 127.0.0.1 : 45,96°C. Donnée acceptée.
[OK] 127.0.0.1 : 49,38°C. Donnée acceptée.
[OK] 127.0.0.1 : 54,67°C. Donnée acceptée.
[OK] 127.0.0.1 : 48,29°C. Donnée acceptée.
[OK] 127.0.0.1 : 45,31°C. Donnée acceptée.
[OK] 127.0.0.1 : 47,96°C. Donnée acceptée.
[OK] 127.0.0.1 : 53,94°C. Donnée acceptée.
[OK] 127.0.0.1 : 49,35°C. Donnée acceptée.
[OK] 127.0.0.1 : 56,35°C. Donnée acceptée.
[OK] 127.0.0.1 : 48,69°C. Donnée acceptée.
[OK] 127.0.0.1 : 46,72°C. Donnée acceptée.
[OK] 127.0.0.1 : 48,12°C. Donnée acceptée.
[OK] 127.0.0.1 : 52,38°C. Donnée acceptée.
[OK] 127.0.0.1 : 48,42°C. Donnée acceptée.
[OK] 127.0.0.1 : 52,3°C. Donnée acceptée.
[OK] 127.0.0.1 : 49,38°C. Donnée acceptée.
[OK] 127.0.0.1 : 45,94°C. Donnée acceptée.
[OK] 127.0.0.1 : 50,35°C. Donnée acceptée.
[OK] 127.0.0.1 : 48,39°C. Donnée acceptée.
[OK] 127.0.0.1 : 52,41°C. Donnée acceptée.
[OK] 127.0.0.1 : 46,29°C. Donnée acceptée.
[OK] 127.0.0.1 : 51,55°C. Donnée acceptée.
[OK] 127.0.0.1 : 54,37°C. Donnée acceptée.
[OK] 127.0.0.1 : 51,14°C. Donnée acceptée.
[OK] 127.0.0.1 : 53,5°C. Donnée acceptée.
[OK] 127.0.0.1 : 50,37°C. Donnée acceptée.
[OK] 127.0.0.1 : 52,44°C. Donnée acceptée.
[OK] 127.0.0.1 : 53,41°C. Donnée acceptée.
[OK] 127.0.0.1 : 46,24°C. Donnée acceptée.
[OK] 127.0.0.1 : 51,26°C. EMPOISONNEMENT DÉTECTÉ !
[OK] 127.0.0.1 : 47,34°C. Donnée acceptée.
[OK] 127.0.0.1 : 49,13°C. Donnée acceptée.
[OK] 127.0.0.1 : 52,72°C. Donnée acceptée.
```

- Tentative d'Empoisonnement Extérieur

L'attaquant injecte une valeur de **98.5°C**. Cette valeur est envoyée au serveur pour tenter de corrompre l'historique d'apprentissage.

```
(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4] [EMPLACEMENT CAPTURE D'ÉCRAN]
└─# echo 98.5 | nc -nv 127.0.0.1 9999 (ou Netcat) montrant l'envoi de la valeur m
Connection to 127.0.0.1 9999 port [tcp/*] succeeded!
Étape 4.4 : Détection et Blocage

(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4]
└─# echo 95.5 | nc -nv 127.0.0.1 9999 Le bouclier IA identifie immédiatement la donnée
Connection to 127.0.0.1 9999 port [tcp/*] succeeded! int toute écriture en base de données

(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4] [EMPLACEMENT CAPTURE D'ÉCRAN]
└─# echo 85.5 | nc -nv 127.0.0.1 9999 en rouge ou en évidence : "[ALERTE S1] 90
Connection to 127.0.0.1 9999 port [tcp/*] succeeded! BLOQUÉ.")

(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4]
└─#
```

- Détection et Blocage

Le bouclier IA identifie immédiatement la donnée comme étant isolée statistiquement. La donnée est rejetée avant toute écriture en base de données.

[ALERTE S1] 127.0.0.1 : 98.5°C. EMPOISONNEMENT DÉTECTÉ !
[ALERTE S1] 127.0.0.1 : 95.5°C. EMPOISONNEMENT DÉTECTÉ !
[ALERTE S1] 127.0.0.1 : 85.5°C. EMPOISONNEMENT DÉTECTÉ !

• Impact de l'outil pour l'usine BMI

L'implémentation de cet outil de détection d'anomalies par **Isolation Forest** garantit : **l'intégrité du modèle, la continuité de service**. De plus en détectant les vraies anomalies cachées par l'attaquant, on évite un arrêt de production de **48h** et la **Sécurité financière**.

Scénario 2

• Présentation du scénario

Dans ce scénario, un industriel rival tente de copier l'intelligence artificielle des presses **SCHULER** de l'usine BMI. En utilisant un script automatisé, il bombarde l'API de prédiction de requêtes pour collecter suffisamment de réponses et reconstruire un modèle miroir. L'objectif de

l'attaquant est de découvrir comment BMI anticipe ses pannes pour adapter sa propre stratégie commerciale.

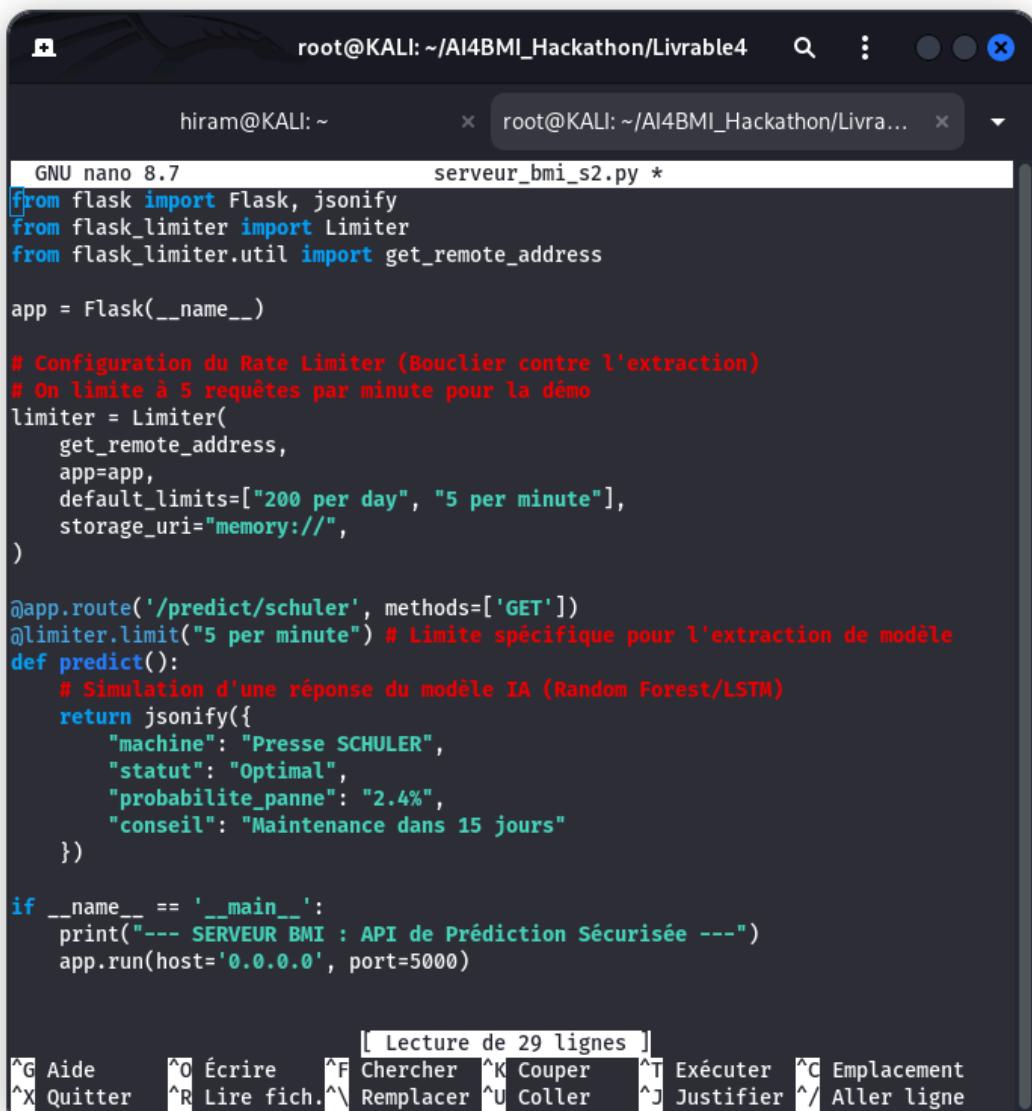
- **Présentation des outils**

- **Flask (Python)** pour simuler l'API REST de la plateforme centrale BMI.
- **Flask-Limiter** pour implémenter le Rate Limiting pour restreindre le nombre de requêtes par utilisateur. Le **Rate Limiting** (Limitation de débit) est la défense standard car il impose une barrière temporelle infranchissable.
- **Requests (Python)** pour simuler le script d'extraction de l'attaquant.

- **Démo (Procédure et Captures)**

- **Mise en place du serveur avec limitation**

Nous configurons le serveur pour n'autoriser que **5 requêtes par minute** pour cet endpoint spécifique.



```

GNU nano 8.7          serveur_bmi_s2.py *
from flask import Flask, jsonify
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

app = Flask(__name__)

# Configuration du Rate Limiter (Bouclier contre l'extraction)
# On limite à 5 requêtes par minute pour la démo
limiter = Limiter(
    get_remote_address,
    app=app,
    default_limits=["200 per day", "5 per minute"],
    storage_uri="memory://",
)

@app.route('/predict/schuler', methods=['GET'])
@limiter.limit("5 per minute") # Limite spécifique pour l'extraction de modèle
def predict():
    # Simulation d'une réponse du modèle IA (Random Forest/LSTM)
    return jsonify({
        "machine": "Presse SCHULER",
        "statut": "Optimal",
        "probabilite_panne": "2.4%",
        "conseil": "Maintenance dans 15 jours"
    })

if __name__ == '__main__':
    print("--- SERVEUR BMI : API de Prédiction Sécurisée ---")
    app.run(host='0.0.0.0', port=5000)

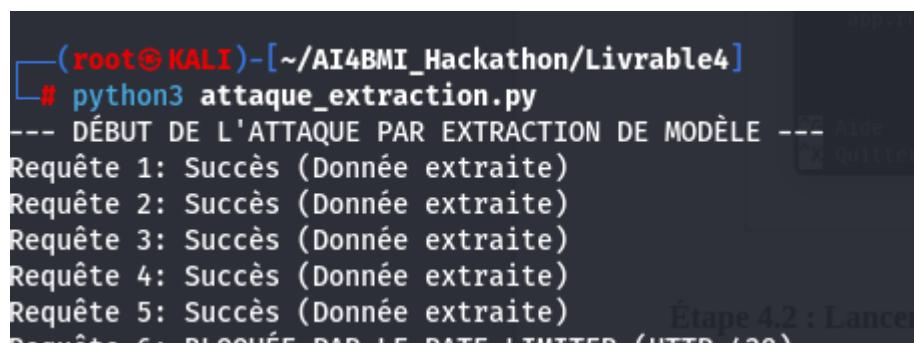

```

[Lecture de 29 lignes]

^G Aide ^O Écrire ^F Chercher ^K Couper ^T Exécuter ^C Emplacement
 ^X Quitter ^R Lire fich.^V Remplacer ^U Coller ^J Justifier ^/ Aller ligne

- Lancement de l'attaque

L'attaquant tente d'extraire les données en rafale (10 requêtes en moins de 5 secondes).



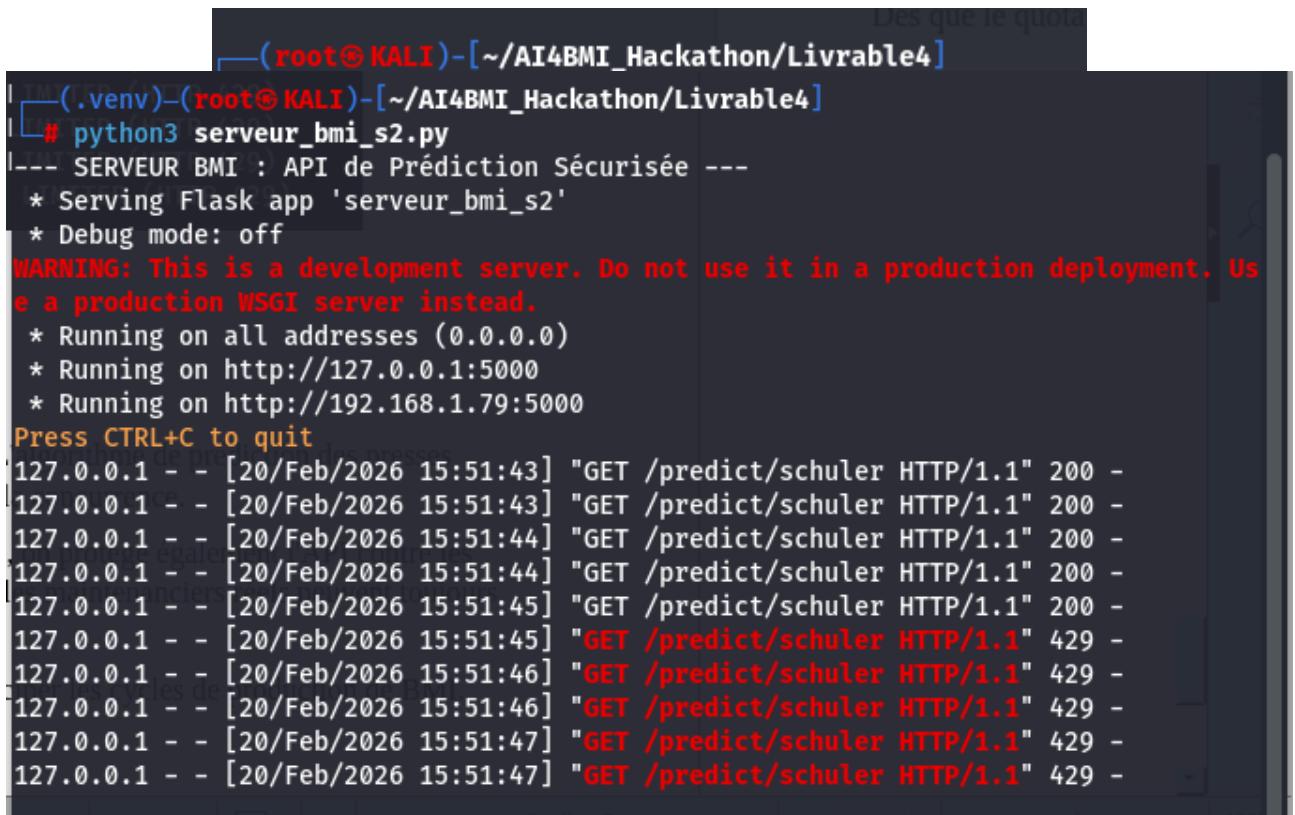
```

(root@KALI)-[~/AI4BMI_Hackathon/Livrable4]
# python3 attaque_extraction.py
--- DÉBUT DE L'ATTAQUE PAR EXTRACTION DE MODÈLE ---
Requête 1: Succès (Donnée extraite)
Requête 2: Succès (Donnée extraite)
Requête 3: Succès (Donnée extraite)
Requête 4: Succès (Donnée extraite)
Requête 5: Succès (Donnée extraite)
Requête 6: BLOQUÉE PAR LE RATE LIMITER (HTTP/1.1)

```

- Blocage par le système

Dès que le quota est dépassé, le serveur rejette les requêtes avec un code de sécurité.



```
Des que le quota
└──(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4]
|___.venv)_(root㉿KALI)-[~/AI4BMI_Hackathon/Livrable4]
| # python3 serveur_bmi_s2.py
|--- SERVEUR BMI : API de Prédition Sécurisée ---
| * Serving Flask app 'serveur_bmi_s2'
| * Debug mode: off
| WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
| * Running on all addresses (0.0.0.0)
| * Running on http://127.0.0.1:5000
| * Running on http://192.168.1.79:5000
| Press CTRL+C to quit
127.0.0.1 - - [20/Feb/2026 15:51:43] "GET /predict/schuler HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2026 15:51:43] "GET /predict/schuler HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2026 15:51:44] "GET /predict/schuler HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2026 15:51:44] "GET /predict/schuler HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2026 15:51:45] "GET /predict/schuler HTTP/1.1" 200 -
127.0.0.1 - - [20/Feb/2026 15:51:45] "GET /predict/schuler HTTP/1.1" 429 -
127.0.0.1 - - [20/Feb/2026 15:51:46] "GET /predict/schuler HTTP/1.1" 429 -
127.0.0.1 - - [20/Feb/2026 15:51:46] "GET /predict/schuler HTTP/1.1" 429 -
127.0.0.1 - - [20/Feb/2026 15:51:47] "GET /predict/schuler HTTP/1.1" 429 -
127.0.0.1 - - [20/Feb/2026 15:51:47] "GET /predict/schuler HTTP/1.1" 429 -
```

• Impact de l'outil

L'algorithme de prédition des presses SCHULER reste confidentiel et inaccessible à la concurrence. De plus, en limitant le débit, on protège également l'API contre les attaques par déni de service (S4), assurant que les maintenanciers réels peuvent toujours accéder aux données. Il empêche le rival d'anticiper les cycles de production de BMI. Ainsi la **préservation de la propriété intellectuelle, la disponibilité du système et la sécurité Stratégique sont assurés.**

Scénario 5

- **Présentation du scénario**

Dans ce scénario, un attaquant n'essaie pas de voler le modèle ou de le corrompre, mais d'en déduire des informations confidentielles sur la production de l'usine **BMI**. En interrogeant l'API de manière répétée, il peut déduire des informations sur les cadences de production réelles, révélant par exemple que l'usine fonctionne à **95% de sa capacité réelle**. Cette information est hautement stratégique et pourrait être utilisée par un concurrent pour préparer une **Offre Publique d'Achat (OPA) hostile**.

- **Présentation des outils**

- **Diffprivlib (IBM)** qui est une bibliothèque de recherche permettant d'implémenter la **Confidentialité Différentielle** (méthode mathématique garantissant qu'un attaquant ne peut pas distinguer si une donnée spécifique est incluse ou non dans un calcul global). Son utilisation de **Diffprivlib** permet d'ajouter un "bruit" mathématique contrôlé aux sorties de l'IA. Cela permet aux ingénieurs de BMI de voir les tendances globales de production tout en empêchant un espion d'obtenir la valeur exacte nécessaire à une analyse stratégique ou financière précise.
- **Numpy** pour la manipulation des vecteurs de données de production.
- **API Flask** : Pour exposer les résultats de production de manière sécurisée.

- **Démonstration Technique (Simulation sur Kali Linux)**

- **Implémentation du mécanisme de bruitage**

Le système utilise un mécanisme Gaussien pour bruiter la cadence de production réelle avant de l'afficher sur le tableau de bord de l'usine.

- **Valeur réelle (Confidentielle)** : 95.00%
 - **Valeur après protection** : 94.82% (ou une valeur proche variant à chaque requête).

Captures d'écran de la démonstration

```

GNU nano 8.7                               defense_inference_s5.py
import numpy as np
(...)

(.venv)-(root@KALI)-[~/AI4BMI_Hackathon/Livrable4]
└─# python3 defense_inference_s5.py
--- SYSTÈME DE PROTECTION BMI (S5) ---
Valeur brute (Confidentielle) : 95.0%
Valeur envoyée à l'API (Bruitée) : 17.29%

(.venv)-(root@KALI)-[~/AI4BMI_Hackathon/Livrable4]
└─# python3 defense_inference_s5.py
--- SYSTÈME DE PROTECTION BMI (S5) ---
Valeur brute (Confidentielle) : 95.0%
Valeur envoyée à l'API (Bruitée) : 94.1%

(.venv)-(root@KALI)-[~/AI4BMI_Hackathon/Livrable4]
└─# python3 defense_inference_s5.py
--- SYSTÈME DE PROTECTION BMI (S5) ---
Valeur brute (Confidentielle) : 95.0%
Valeur envoyée à l'API (Bruitée) : 93.26%
Jeurs.

```

- **Impact de l'outil pour l'usine BMI**

Les performances précises de l'infrastructure de Glo-Djigbé restent floues pour les observateurs extérieurs assurant ainsi la **protection du secret des affaires**. La **résilience économique** empêche l'utilisation de l'IA comme vecteur d'espionnage industriel pour des manœuvres financières hostiles et la **sécurité stratégique** permet de garantir que la plateforme de maintenance ne devienne pas une source de fuite d'informations pour la concurrence.

5. **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 commandade: **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:

The screenshot shows the FastAPI documentation interface. It includes sections for Authentication, API BMI, and Administration. The 'POST /auth/login Login' endpoint is highlighted in green, while others like '/api/capteurs' and '/admin/users-add' are shown in blue.

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

```
# 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.

```
# 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 priviléges, 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**.

```
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 % python3 attaque_extraction.py
--- DÉBUT DE L'ATTACQUE PAR EXTRACTION DE MODÈLE ---
Requête 1: Succès (Donnée extraite)
Requête 2: Succès (Donnée extraite)
Requête 3: Succès (Donnée extraite)
Requête 4: Succès (Donnée extraite)
Requête 5: Succès (Donnée extraite)
Requête 6: Succès (Donnée extraite)
Requête 6: BLOQUÉE PAR LE RATE LIMITER (HTTP 429)
Requête 7: BLOQUÉE PAR LE RATE LIMITER (HTTP 429)
Requête 8: BLOQUÉE PAR LE RATE LIMITER (HTTP 429)
Requête 9: BLOQUÉE PAR LE RATE LIMITER (HTTP 429)
Requête 10: BLOQUÉE PAR LE RATE LIMITER (HTTP 429)
(base) nancyzolla@MacBook-Pro-de-Nancy-2 L4 %
```

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.

```
# 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.

```
# 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 vitesse 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
Last login: Tue Feb 24 12:19:48 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>

```

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.

```
# 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 Dénie 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.

```
# 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 priviléges 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.

6. Livrable 6: Vidéo démonstrative

L'intégralité des démonstrations vidéos illustrant l'exploitation des vulnérabilités présentées dans ce rapport a été enregistrée. Pour ne pas alourdir ce document, **toutes les vidéos démonstratives sont consultables directement sur notre dépôt GitHub: Groupe8 SI3**

7. Livrable 7: Rapport de synthèse

La réalisation de la plateforme prédictive sécurisée pour l'usine Benin Moto Industry (BMI) s'est articulée autour d'une approche de **Défense en Profondeur (Defense in Depth)**. Plutôt que de s'appuyer sur une seule barrière de sécurité, notre groupe a interconnecté plusieurs couches de protection indépendantes :

- 1. Sécurité Périmétrique et Accès (Livrable 1 & 3) :** Le système vérifie de manière stricte l'identité de chaque acteur via une authentification forte (MFA, JWT rotatifs) couplée à un moteur de règles granulaire (Casbin). Un opérateur ne peut mathématiquement pas accéder aux priviléges d'un administrateur, limitant ainsi la surface d'attaque interne.
- 2. Confidentialité des Flux et du Stockage (Livrable 2) :** L'encapsulation du réseau dans un tunnel VPN (Tailscale), le chiffrement des bases de données (AES-256 via pgcrypto) et la sécurisation des communications (TLS 1.3) garantissent la protection contre l'espionnage industriel et limitent l'impact d'un potentiel Ransomware.
- 3. Protection du Noyau IA (Livrable 4) :** Le modèle de Machine Learning, cœur de la propriété intellectuelle de BMI, a été blindé contre les attaques spécifiques à l'IA. L'utilisation d'algorithmes de détection d'anomalies (Isolation Forest), de limitation de débit (Rate Limiting) et de bruitage mathématique (Confidentialité Différentielle) empêche l'empoisonnement des données et l'extraction du modèle par des concurrents.
- 4. Validation Offensive (Livrable 5) :** La robustesse de cette architecture a été éprouvée par un audit Red Team rigoureux, basé

sur les standards internationaux **OWASP Top 10 2025** et **OWASP Machine Learning 2023**. Les vulnérabilités identifiées ont immédiatement fait l'objet de remédiations au niveau du code et de la configuration.

La synergie de ces livrables démontre qu'il est possible de concilier l'hyper-connectivité de l'Industrie 4.0 avec un niveau de sécurité maximal.

III. CONCLUSION

Le projet AI4BMI a permis de concevoir une architecture logicielle et réseau répondant aux exigences critiques de la maintenance industrielle prédictive moderne. En fédérant nos différentes spécialités informatiques, nous avons réussi à déployer une plateforme capable non seulement de collecter et d'analyser les données télémétriques de l'usine de Glo-Djigbé, mais surtout de résister aux cybermenaces complexes qui ciblent aujourd'hui le secteur industriel. La méthodologie *Secure by Design* appliquée tout au long du Hackathon a transformé une simple infrastructure IoT en une forteresse numérique résiliente, protégeant à la fois l'intégrité de la production, la confidentialité des données et la propriété intellectuelle des algorithmes prédictifs.