École Polytechnique de Sousse

Département Informatique

4ème Année - 2025/2026

Vultester

Système Expert de Détection de Vulnérabilités Serveur

Module : Fondements de l'Intelligence Artificielle

Réalisé par :
Mohamed Aziz Mansour

Tuteur :
Mme F. SBIAA

Année Universitaire 2025-2026

Table des matières

1 Introduction

1.1 Objectifs du Projet

Ce projet vise à développer un **système expert** capable de :

1. Identifier automatiquement les vulnérabilités d'un serveur
2. Formaliser l'expertise en sécurité dans une base de connaissances
3. Implémenter un moteur d'inférence (chaînage avant, arrière, mixte)
4. Fournir des diagnostics avec 4 niveaux de sévérité
5. Proposer des actions correctives (patches)
6. Expliquer le raisonnement via une trace d'inférence

1.2 Solution Proposée

Vultester est une application web composée de :

— Un **backend Python/Flask** contenant le moteur d'inférence et 50 règles expertes
— Un **frontend React** offrant une interface moderne et intuitive
— **3 méthodes de chaînage** : avant, arrière et mixte

2 Conception du Système

2.1 Architecture Globale

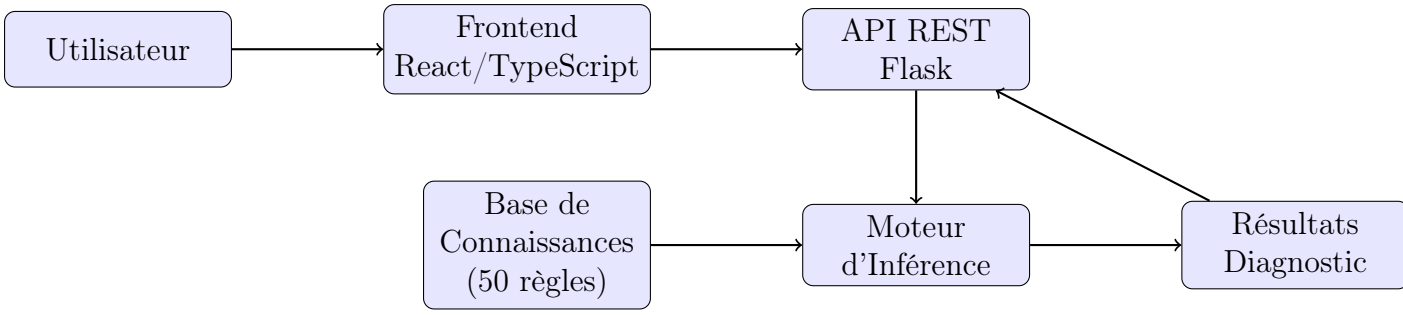

FIGURE 1 – Architecture du système Vultester

2.2 Structure des Données

2.2.1 Représentation des Règles

Chaque règle est représentée par un dictionnaire Python :

```
{
    "id": "R1",
    "conditions": ["port_22_open", "password_auth_enabled"],
    "consequence": "ssh_brute_force_risk",
    "severity": "dangerous",
    "description": "SSH avec authentification par mot de passe..."
}
```

2.2.2 Niveaux de Sévérité

Niveau	Description	Action
CRITICAL	Vulnérabilité critique	Action immédiate requise
DANGEROUS	Configuration dangereuse	Action recommandée
WARNING	Avertissement	Révision recommandée
ACCEPTABLE	Configuration acceptable	Surveillance continue

TABLE 1 – Niveaux de sévérité du diagnostic

3 Moteur d'Inférence

Le système implémente **trois méthodes de chaînage**, codées manuellement sans bibliothèques spécialisées.

3.1 Chaînage Avant (Forward Chaining)

3.1.1 Principe

Le chaînage avant part des **faits connus** (configuration serveur) et applique les règles pour déduire de **nouveaux faits** (vulnérabilités).

3.1.2 Algorithme

```
def forward_chaining(initial_facts):
    queue = copy.deepcopy(initial_facts)
    facts = []
    fired_rules = []

    while queue:
        current_fact = queue.pop(0)
        if current_fact not in facts:
            facts.append(current_fact)

        for rule in rules:
            if rule not in fired_rules:
                if all(cond in facts for cond in rule.conditions):
                    queue.append(rule.consequence)
                    fired_rules.append(rule)

    return analyze_results()
```

3.1.3 Exemple d'Exécution

1. Faits initiaux : `[port_22_open, password_auth_enabled]`
2. Règle R1 vérifiée : conditions satisfaites
3. Nouveau fait déduit : `ssh_brute_force_risk`
4. Diagnostic : Configuration DANGEREUSE

3.2 Chaînage Arrière (Backward Chaining)

3.2.1 Principe

Le chaînage arrière part des **buts** (vulnérabilités possibles) et vérifie si les faits peuvent les **prouver**.

3.2.2 Algorithme

```
def backward_chaining(initial_facts, goals):
    for goal in goals:
        if prove_goal(goal, facts, visited=set()):
            proven_goals.append(goal)
    return analyze_results()

def prove_goal(goal, facts, visited):
    if goal in facts:
        return True
    for rule in rules:
        if rule.consequence == goal:
            if all(prove_goal(cond, facts, visited)
                   for cond in rule.conditions):
                return True
    return False
```

3.3 Chaînage Mixte (Mixed Chaining)

3.3.1 Principe

Le chaînage mixte **combine** les deux approches :

1. **Phase 1** : Chaînage avant pour déduire tous les faits possibles
2. **Phase 2** : Chaînage arrière pour vérifier les vulnérabilités non détectées

3.3.2 Avantages

— Analyse plus **exhaustive**

— Détecte des vulnérabilités que le chaînage avant seul pourrait manquer

— Combine l'efficacité des deux méthodes

3.4 Diagramme de Flux du Moteur d'Inférence

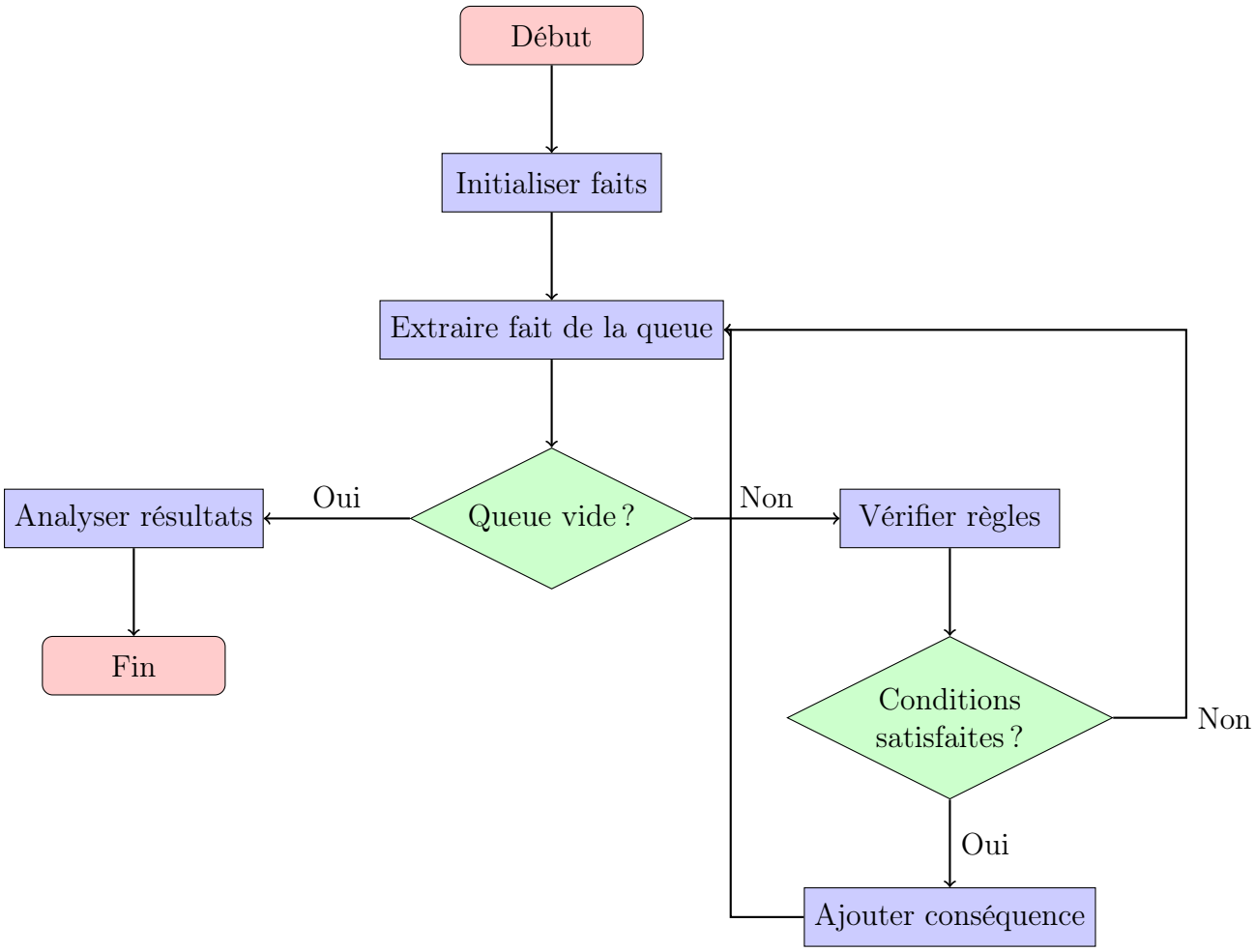

FIGURE 2 – Diagramme de flux du chaînage avant

4 Base de Connaissances

La base contient **50 règles expertes** organisées en 7 catégories.

4.1 Catégories de Règles

Catégorie	**Nb**	**Description**
Ports critiques	10	Ports réseau exposés (SSH, Telnet, FTP...)
SSL/TLS	8	Configuration du chiffrement
Configuration SSH	8	Accès distant sécurisé
Permissions fichiers	8	Droits sur les fichiers système
Versions logiciels	8	Services obsolètes
Réseau/Pare-feu	8	Configuration réseau
Total	**50**	

Table 2 – Répartition des règles par catégorie

4.2 Exemples de Règles

4.2.1 Règles Critiques (CRITICAL)

ID	**Condition**	**Description**
PORT-02	port_23_open	Telnet transmet les données en clair
PORT-03	port_21_open ET ftp_anonymous	FTP anonyme permet accès non autorisé
SSL-06	ssl_3_enabled	SSL 3.0 vulnérable à POODLE
SSH-02	ssh_protocol_1	SSH v1 a des faiblesses crypto
PERM-01	permission_777_found	Permissions 777 dangereuses
SOFT-01	apache_outdated	Apache obsolète avec CVE connus
NET-01	no_firewall_enabled	Pas de pare-feu

Table 3 – Exemples de règles critiques

4.2.2 Règles Dangereuses (DANGEROUS)

ID	Condition	Description
PORT-01	port_22_open ET password_auth	SSH vulnérable au brute force
SSL-01	no_ssl_enabled ET port_80_open	HTTP sans HTTPS
SSL-02	ssl_certificate_expired	Certificat SSL expiré
SSH-01	ssh_root_login_enabled	Login root SSH autorisé
NET-03	ip_forwarding_enabled	IP forwarding actif
SSL-04	ssl_weak_cipher	Chiffrements SSL faibles
SSL-05	tls_1_0_enabled	TLS 1.0 vulnérable (BEAST, POODLE)
SSH-08	ssh_weak_mac	Algorithmes MAC faibles
NET-04	syn_cookies_disabled	Vulnérable aux SYN floods
NET-08	snmp_public_community	SNMP avec community par défaut

Table 4 – Exemples de règles dangereuses

4.2.3 Règles d'Avertissement (WARNING)

ID	Condition	Description
SSL-03	ssl_self_signed	Certificat auto-signé
SSL-07	no_hsts_header	Header HSTS manquant
SSH-04	ssh_x11_forwarding	X11 forwarding peut fuiter des infos
SSH-05	ssh_agent_forwarding	Agent forwarding exploitable
SSH-07	ssh_no_fail2ban	Pas de protection brute force
PERM-08	tmp_not_secured	/tmp non sécurisé
NET-02	icmp_enabled ET no_rate_limiting	Ping flood possible

Table 5 – Exemples de règles d'avertissement

4.2.4 Règles Informatives (INFO)

ID	Condition	Description
SSH-06	ssh_default_port	SSH sur port 22 par défaut

Table 6 – Règles informatives

4.3 Justification des Règles

Les règles sont basées sur :

— **CVE (Common Vulnerabilities and Exposures)** : Base de données des vulnérabilités connues

— **CIS Benchmarks** : Standards de sécurité de l'industrie

— **OWASP** : Recommandations de sécurité web

— **NIST** : Guidelines de cybersécurité

4.3.1 Exemple : Règle PORT-01 - SSH Brute Force

— **Condition** : Port 22 ouvert + Authentification par mot de passe

— **Risque** : Attaques par force brute sur les mots de passe

— **Source** : CIS Benchmark for Linux - Section 5.2.8

— **Recommandation** : Utiliser l'authentification par clé SSH

4.3.2 Exemple : Règle PERM-01 - Permissions 777

— **Condition** : Fichiers avec permissions 777

— **Risque** : Tout utilisateur peut lire, écrire et exécuter

— **Source** : CIS Benchmark - Section 6.1

— **Recommandation** : Restreindre les permissions (chmod 755 ou moins)

4.4 Recommandations (Patches)

Chaque vulnérabilité détectée est accompagnée d'une **action corrective** :

Vulnérabilité	**Recommandation**
ssh_brute_force_risk	PasswordAuthentication no dans sshd_config
telnet_vulnerability	systemctl disable telnet
root_ssh_risk	PermitRootLogin no
world_writable_files	chmod 755 sur les fichiers concernés
no_firewall_protection	ufw enable

Table 7 – Exemples de recommandations

5 Interface Utilisateur

5.1 Technologies Utilisées

— **React 18** : Framework JavaScript moderne
— **TypeScript** : Typage statique pour la fiabilité
— **Tailwind CSS** : Framework CSS utilitaire
— **Framer Motion** : Animations fluides
— **React Router** : Navigation entre pages
— **Vite** : Build tool rapide

5.2 Pages de l'Application

5.2.1 Page d'Accueil (Home)

— Présentation du système
— Statistiques : 50 règles, 4 niveaux, 7 catégories
— Boutons d'action : Lancer l'analyse, Voir les règles

5.2.2 Scanner de Vulnérabilités

Interface de type **survey** en 8 étapes :

1. Ports ouverts (10 options)
2. Configuration SSL/TLS (9 options)
3. Configuration SSH (9 options)
4. Permissions fichiers (6 options)
5. Versions logiciels (6 options)
6. Base de données (8 options)
7. Configuration réseau (8 options)
8. **Choix de la méthode de chaînage**

5.2.3 Base de Connaissances

— Liste des 50 règles avec filtrage
— Recherche par mot-clé
— Filtrage par sévérité
— Affichage format : SI conditions ALORS conséquence

5.2.4 Page À Propos

— Description du système
— Architecture technique
— Crédits (étudiant, tuteur, école)

5.3 Captures d'Écran

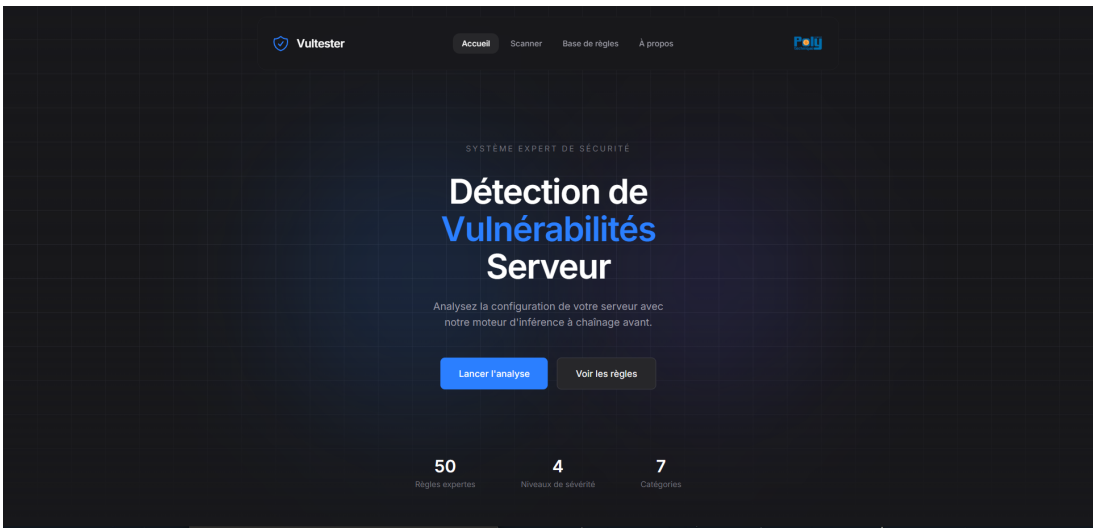

FIGURE 3 – Page d'accueil de Vultester

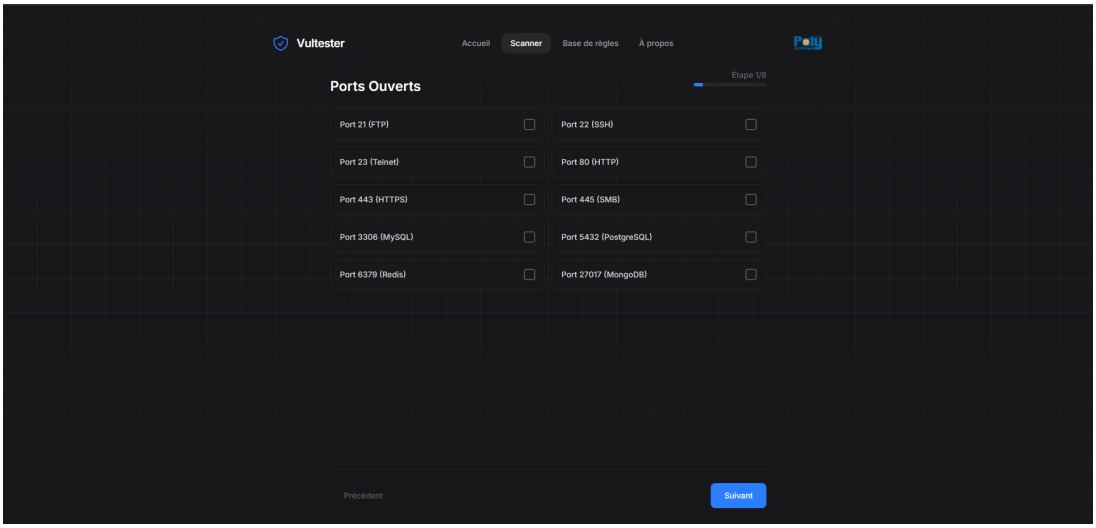

FIGURE 4 – Interface du scanner de vulnérabilités

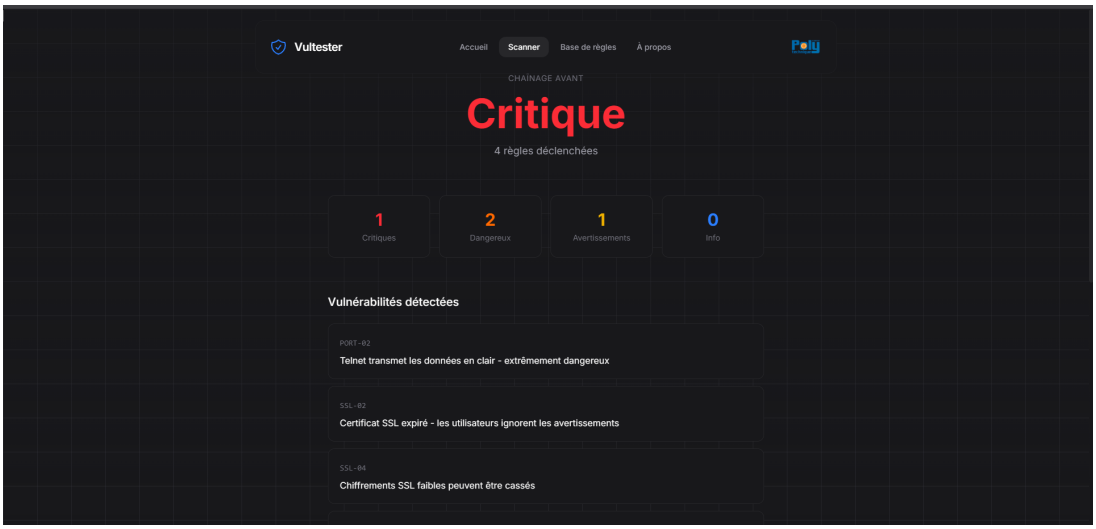

FIGURE 5 – Affichage des résultats avec diagnostic

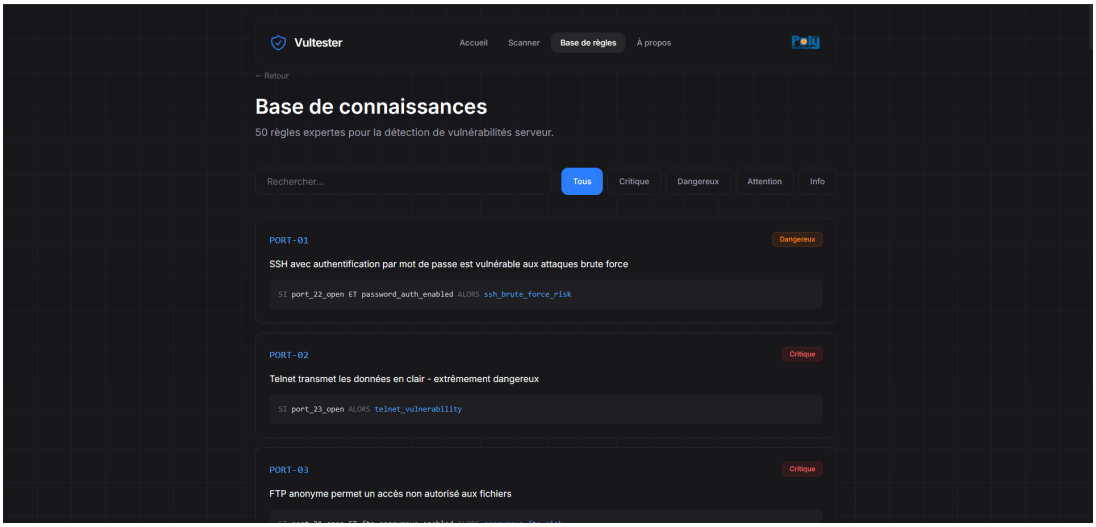

FIGURE 6 – Liste des règles expertes

6 Exemple d'Exécution

6.1 Scénario de Test

Configuration serveur simulée :

- Port 22 (SSH) ouvert
- Port 80 (HTTP) ouvert
- Authentification SSH par mot de passe
- SSL non activé
- Permissions 777 détectées
- Pas de pare-feu

6.2 Faits Initiaux

```
facts = [
    "port_22_open",
    "port_80_open",
    "password_auth_enabled",
    "no_ssl_enabled",
    "permission_777_found",
    "no_firewall_enabled"
]
```

6.3 Trace d'Inférence (Chaînage Avant)

1. **Étape 0** : Initialisation avec 6 faits
2. **Étape 1** : Fait ajouté : port_22_open
3. **Étape 2** : Fait ajouté : password_auth_enabled
4. **Étape 3** : Règle PORT-01 déclenchée → ssh_brute_force_risk
5. **Étape 4** : Fait ajouté : no_ssl_enabled
6. **Étape 5** : Règle SSL-01 déclenchée → unencrypted_traffic
7. **Étape 6** : Règle PERM-01 déclenchée → world_writable_files
8. **Étape 7** : Règle NET-01 déclenchée → no_firewall_protection

6.4 Résultat du Diagnostic

Métrique	Valeur
Statut global	CRITICAL
Règles déclenchées	4
Vulnérabilités critiques	2
Configurations dangereuses	2
Avertissements	0

TABLE 8 – Résumé du diagnostic

7 Limites

7.1 Limites Actuelles

1. **Détection manuelle** : L'utilisateur doit saisir manuellement la configuration
2. **Règles statiques** : Les règles ne s'adaptent pas automatiquement aux nouvelles vulnérabilités
3. **Pas de scan réseau** : Le système ne scanne pas directement les serveurs
4. **Base de connaissances limitée** : 50 règles couvrent les cas courants mais pas exhaustifs
5. **Pas de persistance** : Les analyses ne sont pas sauvegardées

8 Conclusion

Ce projet a permis de développer **Vultester**, un système expert complet pour la détection de vulnérabilités serveur. Les objectifs ont été atteints :

Base de connaissances : 50 règles expertes

Moteur d'inférence : 3 méthodes implémentées (avant, arrière, mixte)

4 niveaux de diagnostic : Critical, Dangerous, Warning, Acceptable

Recommandations : Actions correctives pour chaque vulnérabilité

Trace d'inférence : Explication du raisonnement étape par étape

Interface web : Application React moderne et intuitive

Code manuel : Moteur d'inférence sans bibliothèques spécialisées

---

Références

1. CIS Benchmarks - Center for Internet Security
2. OWASP - Open Web Application Security Project
3. NIST Cybersecurity Framework
4. CVE - Common Vulnerabilities and Exposures Database