Guide technique du site web Insp3ct

Table des matières

1. Description générale :

Insp3ct est une application web conçue pour tester automatiquement la sécurité d'un site web.
Elle simule des attaques courantes, telles que les injections SQL, les failles XSS ou les erreurs de configuration, afin de détecter d'éventuelles vulnérabilités et de fournir un score de sécurité global.

L'outil intègre plusieurs outils open source (par exemple SQLMap pour les injections SQL ou XSStrike pour les failles XSS), une méthodologie d'évaluation, ainsi qu'un système de génération de rapports exploitables au format PDF.

2. Outils de scan intégrés :

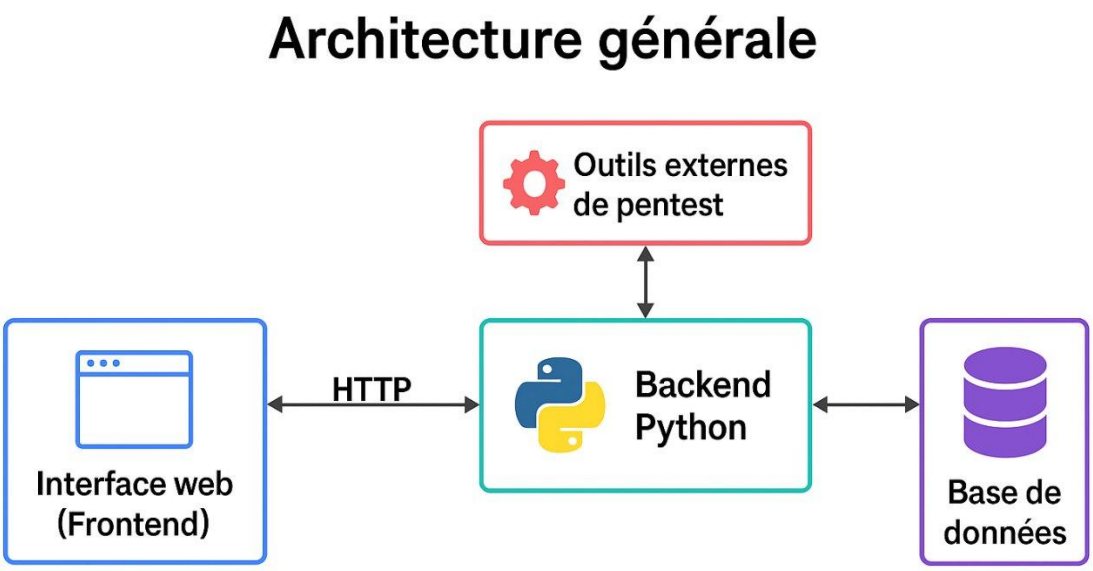
Nous avons choisi de nous concentrer principalement sur les vulnérabilités web. Après une phase d'étude comparative, nous avons retenu les outils open source suivants :

- ✓ XSStrike – Détection des vulnérabilités XSS
- ✓ SQLMap – Détection des injections SQL
- ✓ Nmap – Scan de ports et services réseaux ouverts
- ✓ Nikto – Détection de failles de configuration serveur web
- ✓ Wfuzz – Test de robustesse aux injections via fuzzing

3. Architecture de l'application :

Le projet a été développé en Python, conformément aux contraintes techniques imposées.
Nous avons choisi d'utiliser Flask, un micro-framework web léger, idéal pour les débutants. Il permet de rendre les pages HTML dynamiques et de connecter facilement notre backend à des bases de données grâce à ses nombreuses extensions.

Notre architecture générale ressemble donc à celle-ci :



Le site web et la bdd sont installés sur un container Debian 12 sur un serveur privé virtuel Proxmox . Les clients accèdent à l'outil via l'URL qui leur est fourni.

Tous les outils utilisés ont été téléchargés selon les dernières versions stables disponibles.

4. Pages publiques et privées :

Le site comporte deux types de pages :

- Pages publiques (accessibles à tous via le lien fourni par l'entreprise) :
  - Page d'accueil,
  - Page de démonstration (exemple d'un résultat de scan sans inscription),
  - Page expliquant la méthodologie de calcul du score de sécurité.
- Pages privées (accessibles seulement aux utilisateurs inscrits et connectés) :
  - Les différentes pages de scans.

5. Fonctionnement des différents scans :

Chaque scan a été configuré pour produire un type de résultat précis. Voici un aperçu des commandes et paramètres utilisés :

✓ Nmap – Scan de ports et services

Concernant le scan Nmap, nous avons choisi d'analyser les retours d'un scan permettant de connaitre les ports et services réseaux ouverts sur l'URL du client, ainsi que la version des services avec l'option « nmap_version_detection(target). Cette commande correspondrait à un « nmap -sV *lacible* ». Voici la commande finale :

```
nm = nmap3.Nmap() # Initialisation de Nmap
scan_result = nm.nmap_version_detection(target)
```

✓ Wfuzz – Fuzzing HTTP

Concernant le scan Wfuzz, nous avons choisi d'analyser les retours d'un scan permettant d'envoyer des requêtes http sur le site web client, visant à essayer d'injecter des payloads pour tester les points d'entrée. Ces payloads sont contenus dans une wordlist, et indiqués à l'aide de l'option « -z ». De plus, afin d'indiquer l'URL que nous voulons tester, il est nécessaire d'utiliser l'option « -u ». Nous avons également utilisé l'option « -c » permettant l'affichage en couleur des résultats menant à une compréhension plus efficace par l'API lors du traitement du résultat. Voici la commande finale :

```
command = [
    'wfuzz',
    '-c', # Couleur dans la sortie console
    '-z', f'file,{wordlist}', # Fichier wordlist
    '-u', f'{url}/FUZZ' # url cible avec payload FUZZ
]
```

✓ XSStrike – Détection de failles XSS

Concernant le scan XSStrike, nous avons choisi d'analyser les retours d'un scan permettant de rechercher les failles XSS sur une URL donnée par le client. De plus, nous avons ajoutés l'option « --crawl », afin de lancer automatiquement une recherche sur

toutes les sous-pages de l'URL, pour rechercher tous les points d'entrés susceptibles de contenir des failles XSS. Voici la commande finale :

```
# Commande pour lancer xsstrike
command = [
    'python3', os.getenv("PATH_XSS"),
    '-u', url,
    '--crawl' # Activer crawl pour explorer l'url
]
```

✓ SQLMap – Détection d'injections SQL

Concernant le scan SQLMap, nous avons choisi d'analyser les retours d'un scan permettant de rechercher les possibles injection SQL sur un URL d'un site client. Nous avons choisi les options suivantes :

- L'option « -u », permettant d'indiquer l'URL à scanner.
- L'option « --batch », permettant d'empêcher les demandes d'interaction lors de l'exécution du scan.
- L'option « --random-agent », permettant d'utiliser des user-agent aléatoire afin d'éviter d'être perçu comme un bot.
- L'option « --level=2 », permettant de définir un niveau de test sur une échelle de 1 à 5.
- L'option « --risk=1 », permettant de déterminer le niveau de risk du test.
- L'option « --flush-session », permettant de nettoyer les sessions précédentes afin d'éviter l'usage de cache.
- L'option « --banner », permettant d'afficher les informations de bannière de la base de données.

Voici la commande finale :

```
command = [
    'sqlmap',
    '-u', url,
    '--batch', # Mode batch sans interaction
    '--random-agent', # User-agent aléatoire
    '--level=2', # Niveau de tests
    '--risk=1', # Risque faible
    '--flush-session', # Nettoyer les sessions précédentes
    '--banner' # Afficher la bannière serveur
]
```

✓ Nikto – Scan de configuration serveur

Concernant le scan Nikto, nous avons choisi d'analyser le retour d'un scan permettant de rechercher les fichiers ou répertoires sensibles, les configurations risquées et les failles connues sur les serveurs web, sur ~~le~~ l'URL du site web client. Afin de définir l'url que nous souhaitons scanner, il suffit d'utiliser l'option « -h ». Voici la commande finale :

```
command = [
    'perl', os.getenv("PATH_NIKTO"),
    '-h', url  # URL cible à scanner
]
```

6. Traitement automatisé des résultats :

Les résultats bruts sont collectés du côté du serveur, puis envoyés à l'API OpenAI pour analyse.
Ce traitement externe nous permet de :

- Générer un résumé clair et structuré,
- Limiter l'utilisation des ressources locales (pas de traitement complexe sur le serveur),
- Réduire l'espace de stockage (les résultats bruts sont temporaires),
- Améliorer la confidentialité (aucune conservation longue des données sensibles).

Voici le prompt utilisé :

```
prompt = (
    "Voici les résultats d'un scan de sécurité réalisé sur une machine distante. "
    "Analyse ces résultats, identifie les vulnérabilités détectées (failles, mauvaises configurations, services exposés, etc.), "
    "et propose des recommandations concrètes pour améliorer la sécurité du site.\n\n"
    "À la fin de l'analyse, fournis un score de sécurité sur 100 selon l'échelle suivante. "
    "Le score doit être basé sur une analyse pondérée des points suivants :\n"
    "- Présence de vulnérabilités critiques connues (CVE) : -30 à -50 points selon la sévérité\n"
    "- Mauvaises configurations (ex : indexation, version obsolète de serveur) : -10 à -25 points\n"
    "- Ports ou services exposés non nécessaires : -5 à -20 points\n"
    "- Absence de sécurité réseau minimale (ex : HTTPS absent) : -10 à -20 points\n"
    "- Informations sensibles divulguées (ex : headers, fichiers .git, backup) : -5 à -15 points\n"
    "- Pratiques de sécurité faibles (ex : méthodes HTTP dangereuses autorisées, brute-force possible) : -5 à -15 points\n"
    "- Présence de mécanismes de défense (WAF, redirections, hardening) : +5 à +15 points\n\n"
    "Barème global du score final :\n"
    "- 90-100 : Très sécurisé – Aucune ou très peu de vulnérabilités, bonnes pratiques appliquées.\n"
    "- 70-89 : Satisfaisant – Quelques failles mineures ou améliorations recommandées.\n"
    "- 50-69 : Moyen – Plusieurs failles/modifications nécessaires.\n"
    "- 30-49 : Faible – Nombreuses vulnérabilités importantes, sécurité insuffisante.\n"
    "- 0-29 : Critique – Site très vulnérable, absence de protections essentielles.\n\n"
    "Affiche la ligne suivante à la fin :\n"
    "**Score de sécurité : [XX]**\n\n"
    f"Résultats du scan :\n{scan_result}"
)
```

Les rapports PDF sont ensuite générés à partir de cette analyse simplifiée.

7. Conditions Générales d'Utilisation :

Avant tout scan, l'utilisateur doit accepter les Conditions Générales d'Utilisation (CGU) demandant l'autorisation d'envoyer le résultat du scan à l'API pour traitement.

8. Génération du rapport :

Pour permettre le téléchargement des rapports au format PDF, nous avons choisi de convertir les informations en binaire à l'aide d'un « BytesIO ». Cela nous permet d'éviter de faire un enregistrement du fichier sur le disque du serveur (donc d'avoir un meilleur niveau de performance, de sécurité et de portabilité). Nous manipulons donc le fichier en mémoire « tampon », ce qui permet de transmettre le fichier à Flask plus rapidement avec la méthode « Flask send_file() ». De plus, le passage en binaire est « indispensable » car un fichier PDF n'est pas du texte brut, donc il faut l'encoder avant de pouvoir le transmettre et le télécharger et cette conversion en binaire règle des soucis de compatibilité.