



RAPPORT DU PROJET DE FIN DE module

POO & Algorithme avancées en Python pour la sécurité

PENAUT : SIMULATION D'ATTACK (PENTEST AUTOMATION)

Réalisé par :

BENNANI YAHYA

AMAR SAAD

AMSOU ISMAIL

AOURIK ANAS

LAAMRI SAYF EDDINE

Encadré par :

DR. AZOUGAGHE ALI

Année universitaire 2024-2025

1

REMERCIEMENT :

Nous tenons à exprimer notre sincère gratitude envers toutes les personnes qui ont contribué, de près ou de loin, à la réussite de ce projet.

Tout d'abord, nous remercions chaleureusement Mr. Ali Azougaghe pour son accompagnement tout au long de cette aventure. Son expertise, ses précieux conseils et sa disponibilité ont été des atouts majeurs dans la réalisation de ce projet. Grâce à ses orientations éclairées, nous avons pu surmonter les défis techniques et théoriques, et ainsi mener à bien les différentes étapes de notre travail.

Nous souhaitons également adresser nos remerciements à nos collègues et camarades de promotion pour leurs échanges constructifs et leur soutien. Leurs discussions et leur esprit collaboratif ont enrichi notre réflexion et nous ont permis d'apporter des améliorations continues au projet.

Nos plus sincères remerciements vont à nos proches, qui ont été d'un soutien inestimable pendant toute la durée de ce projet. Leur patience, leur encouragement constant et leur compréhension nous ont permis de surmonter les moments de doute et de persévérer dans les phases les plus difficiles. Ils ont su nous donner la motivation nécessaire pour mener ce projet à terme.

Nous tenons également à remercier toutes les personnes qui nous ont fourni des ressources, des retours ou des idées, que ce soit au niveau technique, théorique ou pratique. Ces contributions ont été essentielles pour améliorer la qualité et la pertinence de notre travail.

Enfin, nous exprimons notre reconnaissance envers les équipes de développement des outils et technologies que nous avons utilisés pour ce projet. Leur travail de qualité a facilité la mise en œuvre et la gestion du projet, et leur support nous a permis de résoudre rapidement les problèmes techniques rencontrés.

À toutes ces personnes, nous adressons nos plus sincères remerciements.

RÉSUMÉ DU PROJET :

Ce projet de cybersécurité a pour ambition de développer une suite d'outils robustes visant à renforcer la détection, l'analyse des vulnérabilités et la sécurisation des systèmes informatiques. Son objectif principal est de fournir aux professionnels de la cybersécurité des solutions performantes pour identifier et atténuer les risques liés aux mots de passe, aux logiciels vulnérables ainsi qu'aux configurations réseau.

Parmi les composants clés, la classe CheckPasswordPwned exploite l'API Have I Been Pwned pour vérifier si un mot de passe a été compromis lors de fuites de données, tout en garantissant la confidentialité de l'utilisateur, puisque le mot de passe n'est jamais transmis en clair. La fonction get_vulnerabilities interroge l'API de la National Vulnerability Database (NVD) afin de récupérer des informations sur les vulnérabilités connues associées à des logiciels spécifiques, fournissant notamment les identifiants CVE et des recommandations de sécurité. Le SubdomainScanner détecte les sous-domaines valides d'un domaine donné à partir d'une liste prédéfinie, tandis que la classe NoSaltedHashCracker permet de casser des mots de passe hachés sans sel en comparant les empreintes générées avec un hachage cible. Un script d'automatisation exécute une reconnaissance active sur un domaine ou une adresse IP en utilisant des outils tels que Nmap, Curl, Eyewitness et Gobuster pour collecter des informations, lesquelles sont ensuite enregistrées dans un fichier JSON pour une analyse ultérieure. Par ailleurs, le NetworkScanner identifie les appareils connectés au sein d'un réseau local, tandis qu'un autre script interroge l'API NVD pour rechercher des vulnérabilités selon divers critères. Le script shodan_scan.py utilise quant à lui l'API de Shodan afin de repérer les dispositifs connectés à Internet et d'en extraire des informations détaillées. Enfin, le DomainScanner effectue un scan exhaustif des sous-domaines et des répertoires d'un domaine spécifié, en optimisant les requêtes HTTP grâce à une exécution parallèle. Ensemble, ces outils constituent une approche intégrée et efficace de la reconnaissance et de la sécurisation des systèmes, permettant aux professionnels de la cybersécurité d'identifier et d'atténuer les risques de manière proactive.

ABSTRACT :

Ce projet présente un ensemble d'outils Python avancés pour l'évaluation de la posture de sécurité et la reconnaissance en cybersécurité, s'appuyant sur des bibliothèques et des interfaces de programmation applicatives (API) spécialisées. L'analyse de la robustesse des mots de passe est réalisée par l'instanciation de fonctions de hachage cryptographique SHA-1 via la librairie hashlib, couplée à une interrogation de l'API "Have I Been Pwned" via des requêtes HTTP pour la détection de potentielles exfiltrations, garantissant une vérification de sécurité sans divulgation des données sensibles en clair. L'identification des vecteurs de vulnérabilité logicielle est effectuée par l'interaction avec l'API REST du National Vulnerability Database (NVD) au moyen de la librairie requests, permettant la récupération structurée (JSON) et l'interprétation des Common Vulnerabilities and Exposures (CVE) et de leurs descriptions associées. La phase de reconnaissance de la surface d'attaque externe est implémentée par un scanner de sous-domaines exploitant des requêtes HTTP GET asynchrones (via requests) et une énumération basée sur des dictionnaires pour cartographier les actifs web d'un domaine cible. Une tentative d'inversion de fonctions de hachage non salées est mise en œuvre à l'aide de hashlib, supportant un éventail d'algorithmes (MD5, SHA-1, SHA-256, etc.) et procédant par une attaque par dictionnaire sur des hachages cibles. La reconnaissance active d'infrastructure est

automatisée par un orchestrateur de tâches intégrant l'utilitaire Nmap pour le balayage de ports TCP/UDP et l'identification de services, Curl pour l'extraction de bannières de service via des requêtes HTTP, Gobuster pour la découverte de ressources web occultées par fuzzing de chemins d'accès, et Eyewitness pour la capture de rendus visuels des interfaces web, agrégeant les résultats dans un format d'échange de données JSON standardisé. L'analyse du segment de réseau local est accomplie par un scanner de réseau s'appuyant sur la manipulation de paquets de la couche liaison de données (ARP) via la librairie scapy et la présentation tabulaire des adresses IP et MAC des hôtes actifs grâce à tabulate.

L'interaction programmatique avec l'API du NVD est étendue pour supporter des requêtes paramétrées, permettant une interrogation ciblée des informations de vulnérabilité, avec la sérialisation des réponses JSON pour une analyse hors ligne. L'exploration de l'écosystème des dispositifs connectés à l'échelle globale est rendue possible par l'intégration de l'API Shodan, utilisant une clé d'authentification pour l'exécution de requêtes de recherche basées

sur des filtres et des opérateurs booléens, la récupération de métadonnées IP détaillées et le décompte de systèmes de contrôle industriel (ICS) via des tags spécifiques, avec une gestion des exceptions et une journalisation des opérations. Enfin, un scanner de domaine avancé combine l'énumération de sous-domaines et la découverte de répertoires par génération dynamique d'Uniform Resource Locators (URLs) à partir de listes de mots, orchestrant des requêtes HTTP concurrentes via un ThreadPoolExecutor pour une exploration rapide et efficace, signalant les ressources accessibles par leur code de statut

HTTP.

GLOSSAIRE ACRONYMES

SHA-1 : Secure Hash Algorithm 1 (Algorithme de hachage sécurisé 1 API : Application Programming Interface (Interface de programmation d'application)

HTTP : Hypertext Transfer Protocol

GET : Méthode de requête HTTP pour récupérer des données

CVE : Common Vulnerabilities and Exposures (Vulnérabilités et expositions communes)

NVD : National Vulnerability Database (Base de données nationale des vulnérabilités)

JSON : JavaScript Object Notation

REST : Representational State Transfer (Transfert d'état représentatif) URL : Uniform Resource Locator (Localisateur uniforme de ressource)

MD5 : Message-Digest Algorithm 5 (Algorithme de hachage par message 5)

SHA : Secure Hash Algorithm (Algorithme de hachage sécurisé) WL : Word List (Liste de mots)

Nmap : Network Mapper (Cartographe de réseau) Curl : Client URL (Client d'URL)

WSL : Windows Subsystem for Linux

ARP : Address Resolution Protocol

MAC : Media Access Control (Contrôle d'accès au support)

IP : Internet Protocol (Protocole Internet)

CPE : Common Platform Enumeration (Énumération de plateformes communes)

CVSS : Common Vulnerability Scoring System

CWE : Common Weakness Enumeration (Énumération des faiblesses communes)

ICS : Industrial Control Systems (Systèmes de contrôle industriel)

TABLE DE FIGURES

• Figure 1 :ENSA.....	9
• Figure 2 : club GCDSTE	10
• Figure 3 :Trchnologies et outils utilisés	12
Figure 4 : outils similaires au projet.....	13
Figure 5 :resultat de port scan	24
Figure 6 :resultat de venurabilité scan	25
Figure 7 :SHODAN interface	27
Figure 8 :tableau de correspondance	36
Figure 9 :resultat de CVE search.....	44
Figure 10 :have i been pawned	52
Figure 11 :github repository	60
Figure 12 : Modélisation par diagramme de classes	68
Figure 13 : Home lab de test.....	70

TABLE DES MATIÈRES

• I. Remerciement	2
• II. Résumé du projet	3
• III. Abstract	4
• IV. Glossaire acronymes.....	5
• V. Table de figures	6
• VI. Table des matières	7
• VII. Introduction générale	8
• 1. Contexte Général	9
○ 1.1. Introduction	10
○ 1.2. Présentation de l'entreprise	11
• 2. Cadre Théorique.....	12
○ 2.1. Introduction aux tests d'intrusion.....	13
○ 2.2. Phases typiques d'un test d'intrusion	13
○ 2.3. Automatisation dans le pentesting	14
○ 2.4 Technologies et outils utilisés dans le projet	15
○ 2.5 Outils similaires existants.....	16
• 3. Les Options de PenAut	17
○ 3.1 Option1 Network Scanner.....	18
○ 3.2 Option2 port scanner	23
○ 3.3 Option3 shodan_scan	31
○ 3.4 Option4 Url Enumeration	40
○ 3.5 - Option5 CVE search.....	42
○ 3.6 - option6 CVE Scan.....	48
○ 3.7 -Option7 hash Cracker	49
○ 3.8 -Option8 Password Pwned.....	56
• 4. Conclusion	62
• 5. Référence.....	63
• 6. Annexes.....	65
○ 6.1 Diagramme de class.....	66
○ 6.2 Home lab	67

1- INTRODUCTION générale :

La cybersécurité est devenue un enjeu majeur dans notre ère numérique. Les entreprises, institutions et particuliers sont de plus en plus exposés à des attaques informatiques, allant du vol de données à l'interruption totale de services. Face à ces menaces croissantes, il est essentiel de mettre en place des méthodes de détection, de prévention et surtout de simulation d'attaques, pour identifier les failles potentielles avant qu'elles ne soient exploitées.

Dans ce contexte, notre projet intitulé "Simulation d'attaques – Pentest Automation" s'inscrit dans une démarche proactive visant à automatiser certaines étapes clés du pentesting (test d'intrusion). L'objectif principal est de développer un outil modulaire en Python, capable d'exécuter plusieurs types de scans et d'analyses de vulnérabilités de manière autonome.

Notre outil regroupe plusieurs fonctionnalités essentielles pour un pentester :

- Scanner de ports pour identifier les services ouverts,
- Scanner de réseau pour détecter les machines actives,
- Hash Cracker pour tester la robustesse de mots de passe hashés,
- Énumération de sous-domaines et de répertoires pour découvrir des points d'entrée cachés,
- Recherche via Shodan pour obtenir des informations sur une cible à partir d'une base de données d'objets connectés,
- Recherche de vulnérabilités (CVE) à partir des versions de technologies utilisées par la cible.

Ce projet est réalisé en programmation orientée objet (POO), ce qui nous a permis de structurer notre code de manière claire et modulaire, facilitant ainsi l'ajout ou la modification de fonctionnalités. Le choix de Python comme langage principal s'est imposé naturellement en raison de la richesse de ses bibliothèques pour la cybersécurité et la simplicité de son utilisation.

À travers ce projet, nous avons souhaité offrir un outil pédagogique, utile pour les étudiants, les passionnés de cybersécurité, ainsi que pour toute personne désirant comprendre le fonctionnement de certaines attaques courantes dans un cadre légal et sécurisé.

Contexte général

I. Introduction

Ce chapitre présente le contexte général du projet envisagé, incluant la présentation de l'organisme d'accueil : l'établissement et la filière, la problématique à laquelle notre projet cherche à répondre, ainsi que les solutions proposées et qui se présentent dans les objectifs du projet.

II. Présentation de l'entreprise

1. ENSA

L'École Nationale des Sciences Appliquées (ENSA) de Marrakech est fondée en 2000, elle a pour mission de former les futurs cadres ingénieurs à travers une pédagogie active, favorisant l'ouverture professionnelle, internationale et culturelle.

Affiliée à l'UCA, l'ENSA de Marrakech joue un rôle de leader dans les filières d'ingénierie. Son impact est significatif, garantissant une formation de qualité et un développement de compétences exceptionnel. L'école favorise un climat de confiance basé sur un engagement mutuel entre les élèves ingénieurs, les enseignants, et le personnel administratif et technique.

Elle comporte 5 filières : Génie Informatique, Génie Industriel et Logistique, Génie Réseau, systèmes et Service programmable, Génie Systèmes Electroniques Embarqués et Commande des Systèmes, et Génie Cyber Défense et Systèmes de Télécommunication Embarqués.



2. GCDSTE

La filière Génie Cyberdéfense et Système de Télécommunications Embarquée (GCDSTE) de l'ENSA-M a pour objectif de former des ingénieurs compétents dotés de solides connaissances scientifiques.

Cette formation intègre des méthodes et des techniques avancées pour assurer la sécurité et gérer les risques liés aux systèmes d'information. De plus, elle vise à habiliter les étudiants à concevoir des systèmes de télécommunications embarqués.

L'accent est mis sur la préparation de chaque élève ingénieur aux méthodes et outils nécessaires pour contrer la cybercriminalité, résoudre les failles des systèmes Figure 1 - Logo ENSA | Cadi Ayyad 4 d'information, et aborder les défis de la sécurité numérique. Parallèlement, les étudiants sont formés à l'implémentation de systèmes embarqués dans divers contextes tels que l'avionique et l'industrie automobile.



CADRE THÉORIQUE :

2.1- INTRODUCTION AUX TESTS D'INTRUSION :

Les tests d'intrusion (ou penetration testing) consistent à simuler des attaques sur un système informatique dans le but d'identifier et corriger ses vulnérabilités. Ces tests sont essentiels pour évaluer la posture de sécurité d'une organisation, détecter les failles exploitables, et vérifier la robustesse des mécanismes de défense.

Un pentest peut être réalisé manuellement, mais l'automatisation joue aujourd'hui un rôle crucial pour accélérer l'analyse, standardiser les résultats, et couvrir un périmètre plus large.

2.2- PHASES TYPIQUES D'UN TEST D'INTRUSION :

Un pentest suit généralement les étapes suivantes :

- Collecte d'informations : reconnaissance passive/active via des outils comme Shodan ou WHOIS...
- Cartographie réseau : scan d'adresses IP et de ports via des outils comme Nmap...
- Énumération : détection de sous-domaines, répertoires cachés, services actifs, etc.
- Recherche de vulnérabilités : comparaison avec des bases comme CVE (Common Vulnerabilities and Exposures).
- Exploitation : tentative d'exploitation de failles détectées.
- Post-exploitation : analyse des accès gagnés, élévation de priviléges, extraction de données sensibles.

Rapport : documentation des vulnérabilités et recommandations.

2.3- AUTOMATISATION DANS LE PENTESTING :

L'automatisation dans le pentesting consiste à utiliser des outils, scripts et processus pour optimiser les tests de sécurité, en réduisant l'intervention manuelle. Elle répond à des besoins critiques dans un contexte où les systèmes informatiques deviennent de plus en plus complexes et les cyberattaques plus fréquentes. Voici une analyse détaillée des points clés :

• Réduire le temps d'exécution des tests

L'automatisation permet d'exécuter des tâches répétitives ou chronophages beaucoup plus rapidement qu'un pentester humain. Par exemple :

- Scan de vulnérabilités : Des outils comme Nmap, Nessus ou OpenVAS peuvent scanner des réseaux entiers en quelques minutes, là où une analyse manuelle prendrait des heures ou des jours.
- Scripts personnalisés : Un script Python peut automatiser des tâches comme la découverte de ports ouverts, l'énumération de services ou l'exploitation de vulnérabilités spécifiques, réduisant le temps nécessaire pour obtenir des résultats exploitables.
- Parallélisation : Les outils automatisés peuvent exécuter plusieurs tests simultanément (par exemple, tester plusieurs hôtes ou applications en parallèle), ce qui accélère considérablement le processus.
- Exemple pratique : Un script utilisant l'API de Shodan peut rapidement identifier des dispositifs exposés sur Internet (comme des caméras IP ou des serveurs mal configurés) en interrogeant la base de données de Shodan, évitant une recherche manuelle longue et laborieuse.

- **Éviter les erreurs humaines :**

Les erreurs humaines, comme l'oubli d'une étape, une mauvaise configuration d'un outil ou une mauvaise interprétation des résultats, peuvent compromettre la qualité d'un pentest. L'automatisation réduit ces risques :

- Standardisation : Les scripts et outils automatisés suivent des processus prédéfinis, garantissant que les tests sont effectués de manière cohérente et complète.
- Validation des résultats : Les outils peuvent inclure des mécanismes de vérification pour s'assurer que les vulnérabilités détectées sont réelles (réduction des faux positifs).
- Exemple pratique : Un script Python qui utilise Metasploit pour tester des exploits peut être configuré pour vérifier automatiquement si une vulnérabilité est exploitable, évitant ainsi des erreurs d'interprétation manuelle.
- Reproductibilité : Les processus automatisés permettent de reproduire les tests dans des conditions identiques, ce qui est essentiel pour valider les correctifs ou comparer les résultats au fil du temps.

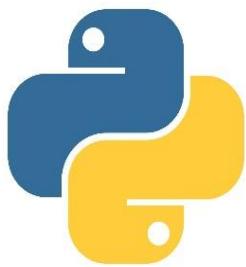
TECHNOLOGIES ET OUTILS UTILISÉS DANS LE PROJET :

L'outil développé repose principalement sur Python, ce qui permet une intégration souple avec de nombreux modules de sécurité. Voici les principales briques techniques intégrées dans le projet (plus de détails dans la prochaine partie) :

- Nmap (nmap.py) : outil puissant pour scanner les ports, identifier les services et les OS.
- Shodan API (shodan_scan.py) : recherche d'actifs exposés publiquement avec vulnérabilités connues.
- Scanner CVE (cve_scan.py) : vérifie si les services identifiés présentent des failles référencées dans des bases publiques.
- Hash Cracking (hashcracker.py) : déchiffrement par force brute ou dictionnaire, pour tester la robustesse de mots de passe ou d'empreintes hash.
- DIRENUM & Subdomain Enumeration : énumération automatique de ressources web non listées, souvent sources d'entrée pour les attaques.
- Checker & Automation : modules qui orchestrent et centralisent l'exécution des différents tests.

Le fichier main.py agit comme un point de contrôle unique pour lancer les différentes phases du pentest de manière automatisée.





2.5 OUTILS SIMILAIRES EXISTANTS :

Plusieurs frameworks open-source existent pour l'automatisation du pentesting, tels que :

- Metasploit Framework : très complet, mais parfois trop lourd ou complexe pour des besoins spécifiques.
- Recon-ng : framework modulaire orienté reconnaissance.
- AutoRecon : script de reconnaissance automatique.
- Nessus / OpenVAS : scanners de vulnérabilités professionnels.

* L'outil développé dans ce projet se distingue par sa simplicité, légèreté, modularité et son exécution locale sans serveur, ce qui est particulièrement adapté aux environnements Linux en ligne de commande.



Metasploit



2- LES OPTIONS DE PENAUT :

Cette partie théorique a permis de présenter le contexte, les étapes typiques d'un test d'intrusion, ainsi que les outils et langages qui sous-tendent l'automatisation de ces processus. Le projet s'inscrit pleinement dans une approche DevSecOps moderne, où la rapidité, l'intégration et la sécurité sont centrales.

```
17      SYSTEMS\PyCharm
18
19      while True:
20          clear()
21          print("\n\n-----> TAP 0 TO QUIT <-----\n\n")
22          print(" 1 - Network Scanner      :")
23          print(" 2 - port scanner         :")
24          print(" 3 - Shodan Scan          :")
25          print(" 4 - Url Enumeration     :")
26          print(" 5 - CVE Search           :")
27          print(" 6 - Hash Cracker         :")
28          print(" 7 - Password Pwned       ?")
```

I. OPTION1 NETWORK SCANNER:

Introduction:

Network Scanner est un module clé du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module effectue une reconnaissance active du réseau local en utilisant le protocole ARP (Address Resolution Protocol) pour identifier les appareils connectés, leurs adresses IP et leurs adresses MAC. Cette étape constitue la première phase d'un test d'intrusion, permettant de cartographier les hôtes actifs dans un réseau cible avant de procéder à des analyses plus approfondies, telles que le scan de ports ou l'identification de vulnérabilités.

Le script est implémenté dans le fichier network_scanner.py et repose sur la bibliothèque Scapy pour manipuler les paquets réseau, ainsi que sur Tabulate pour afficher les résultats de manière lisible. Ce module est conçu pour être simple, efficace et exécutable localement, en ligne avec l'approche DevSecOps du projet PenAut, qui privilégie la modularité, la légèreté et l'automatisation.

2. Rôle dans le Pentesting

Le Network Scanner joue un rôle fondamental dans la phase de reconnaissance (ou "information gathering") d'un test d'intrusion. Ses objectifs principaux sont :

- 1. Découverte des hôtes actifs : Identifier les appareils connectés au réseau local (ex. ordinateurs, routeurs, imprimantes) pour établir une liste de cibles potentielles.*
- 2. Collecte d'informations réseau : Récupérer les adresses IP et MAC des appareils, qui peuvent être utilisées pour des attaques ciblées ou pour corrélérer les informations avec d'autres scans (ex. scan de ports).*
- 3. Cartographie du réseau : Fournir une vue d'ensemble de la topologie réseau, essentielle pour planifier les étapes suivantes du pentest.*

Ce module est particulièrement adapté aux audits internes ou aux tests sur des réseaux locaux, où l'accès direct au réseau cible est disponible. Il est souvent utilisé en combinaison avec d'autres modules de PenAut, comme le Port Scanner ou le CVE Scanner, pour approfondir l'analyse des cibles identifiées.

3. Bibliothèques Utilisées

Le script repose sur deux bibliothèques principales, chacune jouant un rôle spécifique :

1. Scapy :

- *Description : Une bibliothèque Python puissante pour la manipulation de paquets réseau. Elle permet de créer, envoyer, recevoir et analyser des paquets personnalisés à différents niveaux du modèle OSI (couche 2 et 3 dans ce cas).*
- *Rôle : Scapy est utilisé pour construire et envoyer des paquets ARP, puis pour traiter les réponses reçues afin d'extraire les adresses IP et MAC.*
- *Installation : pip install scapy ou sudo apt install python3-scapy (nécessite des priviléges root pour envoyer des paquets réseau).*
- *Avantage : Offre un contrôle granulaire sur les paquets, contrairement à des outils comme Nmap, qui sont moins flexibles pour des scripts personnalisés.*

2. Tabulate :

- *Description : Une bibliothèque Python pour formater des données sous forme de tableaux lisibles dans la console.*
- *Rôle : Tabulate affiche les résultats du scan (adresses IP et MAC) dans un tableau structuré avec des en-têtes et une mise en forme en grille.*
- *Installation : pip install tabulate ou sudo apt install python3-tabulate.*
- *Avantage : Améliore l'expérience utilisateur en rendant les résultats clairs et professionnels, même dans un environnement en ligne de commande.*

4. Fonctionnement Général

Le script utilise le protocole ARP pour découvrir les appareils actifs dans un réseau local. Voici une vue d'ensemble du processus :

1. Construction d'un paquet ARP :

- *Un paquet Ethernet est créé avec une adresse MAC de destination de diffusion (ff:ff:ff:ff:ff:ff), garantissant que tous les appareils du réseau reçoivent la requête.*
- *Une requête ARP est attachée au paquet, demandant l'adresse MAC associée à chaque adresse IP dans une plage spécifiée (ex. 192.168.100.0/24).*

2. Envoi et réception des paquets :

- *Le paquet est envoyé via une interface réseau spécifiée (ex. eth0) en utilisant la fonction srp (send/receive packet) de Scapy.*
- *Les appareils actifs répondent avec leur adresse IP et MAC, qui sont collectées dans une liste.*

3. Traitement des résultats :

- *Les réponses sont analysées pour extraire les adresses IP (psrc) et MAC (hwsr ou src).*
- *Les résultats sont stockés dans une liste pour un affichage ultérieur.*

4. Affichage des résultats :

- *La bibliothèque Tabulate formate les données sous forme de tableau avec deux colonnes : "Adresse IP" et "Adresse MAC".*
- *Le tableau est affiché dans la console avec une mise en forme en grille pour une lisibilité optimale.*

5. Analyse Détailée du Code

```

1   # run with sudo
2   import sys
3   import socket
4   |
5   from tabulate import tabulate # pour affichage self.macList
6   # sudo apt install python3-tabulate / python -m pip install tabulate
7   from scapy.all import Ether, ARP, srp
8   # sudo apt install python3-scapy / python -m pip install scapy
9
10
11  class Network_scanner:
12      def __init__(self, address_resau="192.168.100.0/24", interface="eth0"):
13          self.target = address_resau
14          self.interface = interface
15          self.broadcastMac = "ff:ff:ff:ff:ff:ff"
16          self.macList = []
17
18      def scan_network(self):
19          packet = Ether(dst=self.broadcastMac)/ARP(pdst=self.target)
20          ans, _ = srp(packet, timeout=2, iface=self.interface, inter=0.1, verbose=False)
21          self.macList = [(receive.psrc, receive.src) for _, receive in ans]
22
23      def mac_table(self):
24          print(tabulate(self.macList, headers=["Adresse IP", "Adresse MAC"], tablefmt="grid"))
25
26
27  def main_net() :
28      ip = input("Address reseau (192.168.100.1/24) :")
29      inter = input("Interface :")
30      n = Network_scanner(ip,inter)
31      n.scan_network()
32      n.mac_table()

```

- *sys* : Importé mais non utilisé dans le code actuel. Peut être destiné à gérer les arguments de la ligne de commande ou les erreurs système dans une version étendue.
- *socket* : Importé mais non utilisé. Probablement inclus pour une fonctionnalité future (ex. vérification des connexions réseau).
- *tabulate* : Importe la bibliothèque pour formater les résultats en tableau.
- *scapy.all* : Importe les modules *Ether* (pour les paquets Ethernet), *ARP* (pour les requêtes ARP), et *srp* (pour envoyer/recevoir des paquets).

Constructeur : Initialise la classe avec :

- *address_resau* : Plage d'adresses IP à scanner (par défaut 192.168.100.0/24, correspondant à un sous-réseau local).
 - *interface* : Interface réseau utilisée pour envoyer les paquets (par défaut eth0, typique pour les environnements Linux).
 - *broadcastMac* : Adresse MAC de diffusion (ff:ff:ff:ff:ff:ff), utilisée pour envoyer la requête à tous les appareils du réseau.
 - *macList* : Liste vide pour stocker les paires IP/MAC des appareils détectés.
-
- *Envoi et réception :*
 - *srp(packet, timeout=2, iface=self.interface, inter=0.1, verbose=False)* :
 - *srp* : Fonction Scapy qui envoie des paquets et attend les réponses (send/receive packet).
 - *timeout=2* : Attend les réponses pendant 2 secondes maximum.
 - *iface=self.interface* : Spécifie l'interface réseau (ex. eth0).
 - *inter=0.1* : Intervalle de 0,1 seconde entre l'envoi des paquets pour éviter de surcharger le réseau.
 - *verbose=False* : Désactive les messages de débogage de Scapy pour une sortie propre.
 - *ans, _* : Stocke les réponses reçues dans *ans* (liste des paires envoyé/reçu) et ignore les paquets non répondus (*_*).
 - *Traitement des réponses :*
 - *[(receive.psrc, receive.src) for _, receive in ans]* : Une liste en compréhension extrait :
 - *receive.psrc* : Adresse IP source de la réponse (l'IP de l'appareil).
 - *receive.src* : Adresse MAC source de la réponse (la MAC de l'appareil).
 - Les paires IP/MAC sont stockées dans *self.macList*.

II. - OPTION2 PORT SCANNER:

1. Introduction

Port Scanner est un module essentiel du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module est conçu pour identifier les ports ouverts sur une machine cible, détecter les services associés et, dans certains cas, collecter des informations supplémentaires comme des bannières ou des vulnérabilités potentielles. Ces informations sont cruciales pour évaluer la surface d'attaque d'un système lors d'un audit de sécurité. Le module combine deux approches :

1. Une implémentation initiale utilisant la bibliothèque Python socket pour effectuer un scan TCP des ports, comme décrit dans la documentation fournie.
2. Une implémentation plus avancée (fournie dans le code) utilisant Nmap pour le scan des ports et des services, Curl pour la capture de bannières, et des scripts Nmap pour l'analyse de vulnérabilités.

Ce module s'inscrit dans l'approche DevSecOps de PenAut, mettant l'accent sur la modularité, l'automatisation et l'intégration avec des outils de cybersécurité standards. Les résultats sont sauvegardés dans un fichier JSON pour une analyse ultérieure, facilitant l'intégration dans des workflows de sécurité.

3. Rôle dans le Pentesting

Le Port Scanner joue un rôle central dans la phase de reconnaissance et d'analyse de surface d'attaque d'un test d'intrusion. Ses objectifs principaux sont :

1. Identification des ports ouverts : Déetecter les ports TCP (et potentiellement UDP) actifs sur une machine cible, indiquant les services réseau exposés (ex. HTTP sur le port 80, SSH sur le port 22).
2. Énumération des services : Identifier les logiciels et versions associés aux ports ouverts (ex. Apache 2.4.41 sur le port 80), qui peuvent être liés à des vulnérabilités connues.
3. Collecte de bannières : Récupérer les bannières des services (ex. en-têtes HTTP) pour obtenir des informations supplémentaires sur les logiciels en cours d'exécution.
4. Détection des vulnérabilités : Utiliser des scripts spécialisés (ex. Nmap vuln) pour identifier des failles potentielles dans les services détectés.

Ce module est souvent utilisé après la découverte des hôtes (par exemple, via l'Option 1 : Network Scanner) pour approfondir l'analyse des cibles identifiées. Les informations collectées servent de base pour des étapes ultérieures, comme l'exploitation de vulnérabilités ou l'énumération des ressources web.

4. Bibliothèques et Outils Utilisés

Le module repose sur une combinaison de bibliothèques Python et d'outils externes, selon l'implémentation :

3.1. Implémentation avec socket

- Socket :
 - Description : Bibliothèque standard Python pour les communications réseau de bas niveau.
 - Rôle : Permet de créer des sockets TCP pour tester la connectivité des ports sur une machine cible.
 - Avantage : Léger et personnalisable, idéal pour des scans simples sans dépendances externes.
 - Installation : Inclus dans Python, aucune installation supplémentaire requise.

3.2. Implémentation avec Nmap et Curl

- Subprocess :
 - Description : Bibliothèque standard Python pour exécuter des commandes système externes.
 - Rôle : Lance des commandes Nmap et Curl pour effectuer des scans et capturer des bannières.
 - Avantage : Permet d'intégrer des outils puissants comme Nmap dans un script Python.
- Json :
 - Description : Bibliothèque standard Python pour manipuler des données au format JSON.
 - Rôle : Sauvegarde les résultats des scans dans un fichier JSON structuré.
- Os :
 - Description : Bibliothèque standard Python pour interagir avec le système de fichiers.
 - Rôle : Gère la création de répertoires et la manipulation des chemins de fichiers pour les résultats.
- Nmap :
 - Description : Outil open-source de scan réseau et d'énumération des services.
 - Rôle : Effectue des scans de ports, identifie les services et versions, et exécute des scripts de détection de vulnérabilités.
 - Installation : sudo apt install nmap (Linux) ou équivalent pour d'autres systèmes.
- Curl :
 - Description : Outil en ligne de commande pour effectuer des requêtes HTTP.

- Rôle : Capture les bannières HTTP (en-têtes) des services web détectés.
 - Installation : sudo apt install curl (Linux) ou équivalent.
-

4. Fonctionnement Général

Le module propose deux approches complémentaires pour le scan des ports :

4.1. Approche avec socket

- Principe : Utilise la fonction `socket.connect_ex()` pour tenter une connexion TCP à chaque port spécifié sur la machine cible. Un code de retour 0 indique que le port est ouvert.
- Etapes :
 1. Création d'un socket TCP (AF_INET pour IPv4, SOCK_STREAM pour TCP).
 2. Définition d'un délai d'attente court (ex. 0,5 seconde) pour éviter les blocages.
 3. Tentative de connexion à chaque port via `connect_ex`.
 4. Enregistrement des ports ouverts pour affichage ou traitement ultérieur.
- Avantage : Simple et léger, ne nécessite pas d'outils externes.
- Limitation : Moins performant pour scanner de larges plages when generating images.

4.2. Approche avec Nmap et Curl

- Principe : Utilise Nmap pour un scan complet des ports et services, suivi d'une analyse de vulnérabilités et de la capture de bannières avec Curl.
 - Étapes :
 1. Scan Nmap : Exécute `nmap -sV` pour identifier les ports ouverts et les services/versions associés.
 2. Scan de vulnérabilités : Exécute `nmap --script=vuln` pour détecter les failles connues dans les services identifiés.
 3. Capture de bannières : Utilise Curl pour récupérer les en-têtes HTTP des ports associés aux services web.
 4. Sauvegarde des résultats : Compile les résultats (ports, services, vulnérabilités, bannières) dans un fichier JSON.
-

5. Analyse Détailée du Code

Le code fourni implémente une version avancée du Port Scanner utilisant Nmap et Curl, avec une intégration dans un workflow automatisé. Voici une analyse ligne par ligne, suivie d'un exemple de l'implémentation socket pour compléter la description initiale.

```
1 import subprocess  
2 import json  
3 import os  
4
```

- subprocess : Permet d'exécuter des commandes externes (Nmap, Curl) et de capturer leurs sorties.
- json : Gère la sauvegarde des résultats au format JSON.
- os : Manipule les chemins de fichiers et crée des répertoires.

Fonction run_nmap_scan

```
5 def run_nmap_scan(target):  
6     """Performs Nmap scan for open ports and service enumeration."""  
7     print("[+] Running Nmap scan...")  
8     result = subprocess.run(["nmap", "-sV", "-T4", "--unprivileged", "-oN", "nmap_scan.txt", target], capture_output=True, text=True)  
9     return result.stdout  
10
```

Rôle : Exécute un scan Nmap pour identifier les ports ouverts et les services.

- Commande Nmap :
 - nmap : Appelle l'outil Nmap.
 - -sV : Active l'énumération des versions des services (ex. Apache 2.4.41).
 - -T4 : Définit un niveau d'agressivité (4/5) pour accélérer le scan.
 - --unprivileged : Exécute Nmap sans privilège root, limitant certaines fonctionnalités mais évitant les restrictions de sécurité.
 - -oN nmap_scan.txt : Sauvegarde les résultats dans un fichier texte.
 - target : Adresse IP ou nom de domaine de la cible.
- Sortie : subprocess.run capture la sortie standard (stdout) sous forme de texte.

Fonction run_vuln_scan

```

def run_vuln_scan(target):
    """Runs Nmap using the 'vuln' script category to find known vulnerabilities."""
    print("[+] Running vulnerability scan using Nmap vuln scripts...")
    vuln_output_file = os.path.expanduser("~/result_PenAut/vuln_scan.txt")
    result = subprocess.run(["nmap", "-sV", "--script=vuln", "--unprivileged", "-T4", "-oN", vuln_output_file, target],
                           capture_output=True, text=True)
    return result.stdout

```

- Rôle : Exécute un scan Nmap avec des scripts de détection de vulnérabilités.
- Commande Nmap :
 - --script=vuln : Utilise la catégorie de scripts Nmap vuln pour identifier les failles connues (ex. CVE associées aux services).
 - vuln_output_file : Chemin dynamique pour sauvegarder les résultats (ex. ~/result_PenAut/vuln_scan.txt).
- Sortie : Retourne la sortie texte des résultats du scan.

Fonction grab_banner

```

21  def grab_banner(target, port):
22      """Grabs banners using Curl on Windows."""
23      print(f"[+] Grabbing banner for {target}:{port}...")
24      try:
25          result = subprocess.run(["curl", "-I", f"http://{target}:{port}"], capture_output=True, text=True, timeout=5)
26          return result.stdout.strip()
27      except subprocess.TimeoutExpired:
28          return "Timeout"
29

```

Rôle : Récupère les en-têtes HTTP (bannières) des services web via Curl.

- Commande Curl :
 - curl -I : Effectue une requête HTTP HEAD pour récupérer les en-têtes.
 - http://{target}:{port} : URL construite pour le port spécifié.
 - timeout=5 : Limite la requête à 5 secondes.
- Gestion des erreurs : Retourne "Timeout" si la requête dépasse le délai.
- Sortie : En-têtes HTTP ou message d'erreur.

Fonction main_port

```
def main_port():
    target = input("Entrez l'adresse IP ou le nom de domaine de la cible : ").strip()

    scan_results = run_nmap_scan(target)
    vuln_results = run_vuln_scan(target)

    open_ports = [line.split("/")[0] for line in scan_results.split("\n") if "/tcp" in line and "open" in line]
    banners = {port: grab_banner(target, port) for port in open_ports}

    report = {
        "target": target,
        "nmap_results": scan_results,
        "vulnerability_scan": vuln_results,
        "banners": banners,
    }

    result_dir = os.path.expanduser("~/result_PenAut")
    os.makedirs(result_dir, exist_ok=True)
    report_path = os.path.join(result_dir, "recon_report.json")

    with open(report_path, "w") as f:
        json.dump(report, f, indent=4)

    print(f"[+] Recon terminée. Résultats enregistrés dans {report_path}")
```

Rôle : Point d'entrée interactif pour orchestrer le scan.

- Étapes :

1. Demande à l'utilisateur de saisir la cible (IP ou domaine).
2. Exécute run_nmap_scan et run_vuln_scan.
3. Extrait les ports ouverts à partir de scan_results en filtrant les lignes contenant /tcp et open.
4. Collecte les bannières pour chaque port ouvert avec grab_banner.

- 5. Crée un dictionnaire report avec les résultats (cible, scan Nmap, scan de vulnérabilités, bannières).
 - 6. Crée un répertoire `~/result_PenAut` si nécessaire et sauvegarde le rapport au format JSON.
 - Sortie : Affiche un message de confirmation avec le chemin du fichier JSON.

6. Exemple d'Utilisation

6.1. Avec Nmap et Curl

1. Exécution :
 2. python3 port_scanner.py
 - o Saisir la cible : Ex. 192.168.1.1 ou example.com.
 - o Résultats sauvegardés dans ~/result_PenAut/recon_report.json.
 3. Sortie attendue :

7. Importance dans le Projet PenAut

Le Port Scanner est un module clé de PenAut pour les raisons suivantes :

1. Évaluation de la surface d'attaque : Identifie les points d'entrée potentiels (ports ouverts) et les services associés, qui peuvent être exploités ou mal configurés.
 2. Automatisation : L'approche Nmap/Curl automatise le scan, l'énumération des services, et la détection des vulnérabilités, réduisant l'effort manuel.
 3. Flexibilité : L'implémentation socket offre une alternative légère pour des environnements sans Nmap, tandis que l'approche Nmap/Curl est plus robuste

pour des audits complets.

4. Intégration : Les résultats JSON facilitent l'intégration avec d'autres modules PenAut (ex. CVE Scanner) ou des outils externes.
 5. Accessibilité : L'interface interactive (main_port) rend le module utilisable par des utilisateurs novices.

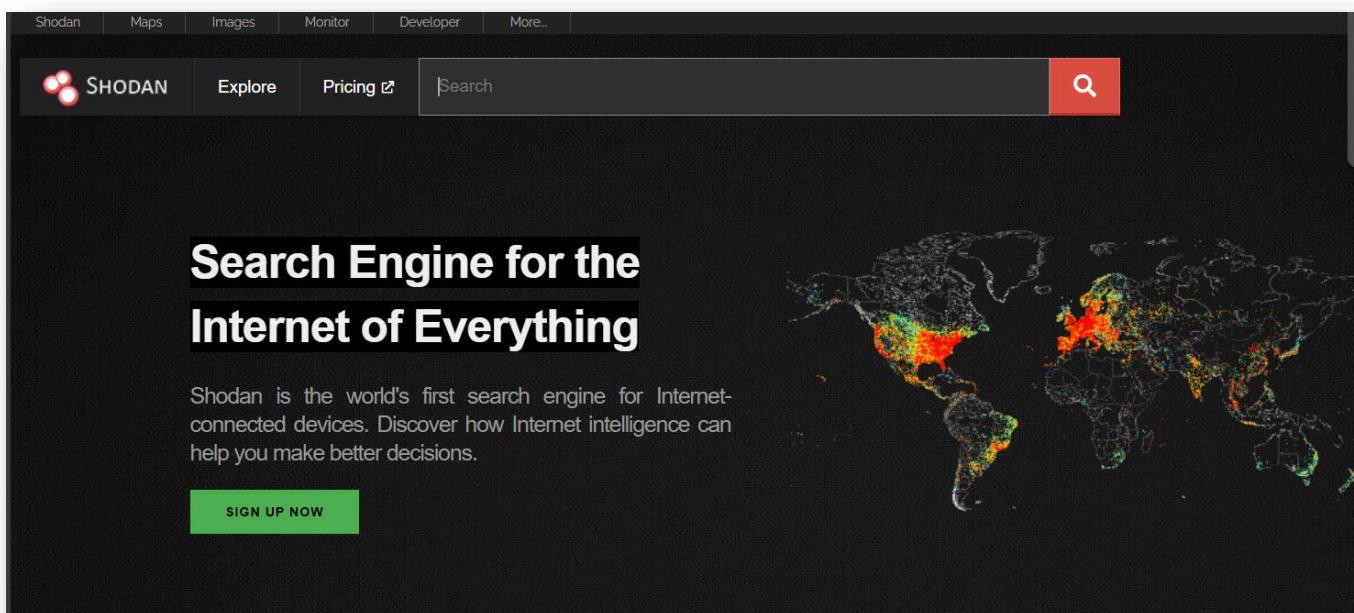
```
vulnerabilities.\nService Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel\n\nService detection performed. Please report any incorrect results at https://nmap.org/submit/. \nNmap done: 1 IP address (1 host up) scanned in 337.61 seconds\n",\n"banners": {\n    "22": "",\n    "80": "HTTP/1.1 200 OK\nDate: Sun, 06 Apr 2025 20:43:42 GMT\nServer: Apache/2.4.18 (Ubuntu)\nLast-Modified: Fri, 29 Nov 2019 09:27:58\nETag: \"2c39-59878d86c765e\"\nAccept-Ranges: bytes\nContent-Length: 11321\nVary: Accept-Encoding\nContent-Type: text/html"\n},\n"dirbuster_results": "=====\\nGobuster v3.6\\nby OJ Reeves (@TheColonial) &\nChristian Mehlmauer (@fireart)\\n=====\\n[+] Url: http://10.10.\n197.73\\n[+] Method:\nGET\\n[+] Threads: 10\\n[+] Wordlist: common.txt\\n[+] Negative Status\nCodes: 404\\n[+] User Agent: gobuster/3.6\\n[+] Timeout:\n10s\\n=====\\nStarting gobuster in directory enumeration\nmode\\n=====\\n\\n\\n/.hta (Status: 403) [Size: 277]\\n\\n\\n/.htpasswd (Status: 403) [Size: 277]\\n\\n\\n/.htaccess (Status: 403) [Size: 277]\\n\\n\\n/.content (Status: 301)\n[Size: 314] [--> http://10.10.197.73/content/]\\n\\n\\n/index.html (Status: 200) [Size: 11321]\\n\\n\\n/server-status (Status: 403) [Size: 277]\n\\n\\n=====\\nFinished\\n=====\n=\\"\\n"
```

III. - OPTION3 SHODAN_SCAN:

1. Introduction

Shodan Scan est un module clé du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module utilise l'API Shodan, un moteur de recherche pour les appareils connectés à Internet, afin de collecter des informations OSINT (Open-Source Intelligence) sur des actifs exposés publiquement. Il permet d'identifier des services, des ports ouverts, des vulnérabilités potentielles, et d'estimer l'exposition des systèmes de contrôle industriels (ICS). Les résultats sont sauvegardés dans des fichiers JSON et texte pour une analyse ultérieure.

Le script est implémenté dans le fichier shodan_scan.py et repose sur la bibliothèque Python Shodan pour interagir avec l'API, ainsi que sur json et logging pour la gestion des résultats et des journaux. Ce module s'inscrit dans l'approche DevSecOps de PenAut, mettant l'accent sur l'automatisation, la modularité, et l'intégration avec des sources de données publiques pour des audits de sécurité efficaces.



2. Rôle dans le Pentesting

Le Shodan Scan joue un rôle crucial dans la phase de reconnaissance passive d'un test d'intrusion. Ses objectifs principaux sont :

1. Identification des services exposés : Recherche des appareils et services correspondant à un mot-clé spécifique (ex. "apache") pour identifier les actifs exposés sur Internet.
2. Collecte d'informations OSINT : Récupération de données détaillées sur une adresse IP spécifique, telles que le fournisseur, les ports ouverts, les services actifs, et les bannières.
3. Évaluation de l'exposition des ICS : Estimation du nombre de systèmes de contrôle industriels accessibles publiquement, souvent vulnérables en raison de mauvaises configurations.
4. Automatisation de la collecte de données : Sauvegarde des résultats dans des formats structurés (JSON et texte) pour une analyse ultérieure ou une intégration dans d'autres outils.

Ce module est particulièrement utile pour les auditeurs de sécurité et les pentesters qui cherchent à cartographier la surface d'attaque d'une organisation sans interagir directement avec les systèmes cibles, réduisant ainsi le risque de détection. Il complète d'autres modules PenAut, comme le Port Scanner ou le CVE Scanner, en fournissant des informations préliminaires sur les actifs exposés.

3. Fonctionnalités

Le module Shodan Scan offre quatre fonctionnalités principales :

1. Recherche Shodan :
 - Recherche des bannières correspondant à un mot-clé donné (ex. "apache") avec une limite de résultats configurable (par défaut, 10 résultats).
 - Exemple : Identifier tous les serveurs Apache exposés publiquement.
2. Récupération d'informations sur une IP :
 - Interroge l'API Shodan via `api.host('8.8.8.8')` pour collecter des détails sur une adresse IP spécifique, tels que le fournisseur, les ports ouverts, les services, et les systèmes d'exploitation détectés.
 - Exemple : Analyse des services actifs sur l'IP 8.8.8.8 (serveur DNS de Google).
3. Comptage des systèmes de contrôle industriels (ICS) :
 - Utilise la requête `tag:ics` pour estimer le nombre de systèmes industriels (ex. SCADA, PLC) exposés à Internet.
 - Exemple : Évaluation des risques liés aux systèmes critiques mal sécurisés.
4. Sauvegarde des résultats :

- Compile les résultats dans un dictionnaire Python et les sauvegarde dans un fichier JSON (shodan_results.json) pour une analyse structurée.
- Génère également un fichier texte (shodan_results.txt) avec une mise en forme lisible pour une consultation rapide.

Ces fonctionnalités permettent aux pentesters de :

- Identifier des cibles potentielles pour des audits approfondis.
 - Collecter des informations exploitables sans interaction directe avec les systèmes.
 - Évaluer l'exposition globale des infrastructures critiques.
-

4. Bibliothèques Utilisées

Le script repose sur les bibliothèques suivantes :

1. Shodan :

- Description : Bibliothèque Python officielle pour interagir avec l'API Shodan.
- Rôle : Fournit des méthodes pour effectuer des recherches, récupérer des informations sur des hôtes, et compter des services spécifiques.
- Installation : pip install shodan.
- Avantage : Simplifie l'accès à l'API Shodan avec une interface Python intuitive.

2. Json :

- Description : Bibliothèque standard Python pour manipuler des données au format JSON.
- Rôle : Sauvegarde les résultats dans un fichier JSON structuré avec une mise en forme lisible.
- Avantage : Permet une intégration facile avec d'autres outils ou scripts.

3. Logging :

- Description : Bibliothèque standard Python pour gérer les journaux d'exécution.
 - Rôle : Enregistre les événements (succès, erreurs, informations) avec des horodatages pour le débogage et le suivi.
 - Avantage : Améliore la traçabilité et facilite l'analyse des problèmes.
-

5. Analyse Détaillée du Code

Le code fourni implémente une classe et des fonctions pour interagir avec l'API Shodan, collecter des données, et sauvegarder les résultats. Voici une analyse ligne par ligne, avec des explications techniques détaillées.

Imports et Configuration

```
1 # File: shodan_scan.py
2 from shodan import Shodan, APIError
3 import json
4 import logging
5
```

- Imports :
 - shodan.Shodan : Classe principale pour interagir avec l'API Shodan.
 - shodan.APIError : Exception spécifique pour gérer les erreurs de l'API.
 - json : Pour sauvegarder les résultats.
 - logging : Pour journaliser les événements.
- Configuration :
 - OUTPUT_JSON_FILE : Nom du fichier JSON pour les résultats (shodan_results.json).
 - OUTPUT_TXT_FILE : Nom du fichier texte pour une version lisible (shodan_results.txt).
 - RESULT_LIMIT : Limite le nombre de résultats de recherche (10 par défaut).
 - API_KEY : Clé API Shodan (doit être remplacée par une clé valide).

Configuration de Logging

```
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
```

- Configure le système de journalisation pour afficher les messages avec un horodatage, un niveau (INFO, ERROR), et un message descriptif.
- Exemple de sortie : 2025-04-23 10:00:00,123 - INFO - Results saved to shodan_results.json.

Fonction save_to_json:

```
1 def save_to_json(data, filename):
2     """Save data to a JSON file with pretty formatting."""
3     try:
4         with open(filename, 'w') as file:
5             json.dump(data, file, indent=4, separators=(',', ': '))
6             logging.info(f"Results saved to {filename}")
```

7. except Exception as e:
8. logging.error(f"Failed to save results to {filename}: {e}")

- Rôle : Sauvegarde les données dans un fichier JSON avec une mise en forme lisible.
- Détails :
 - Ouvre le fichier en mode écriture ('w').
 - Utilise json.dump avec indent=4 pour une présentation structurée.
 - Journalise le succès ou l'échec de l'opération.

Fonction save_to_txt

```

1. def save_to_txt(data, filename):
2.     """Save data to a TXT file with organized formatting."""
3.     try:
4.         with open(filename, 'w') as file:
5.             file.write("Shodan Scan Results\n")
6.             file.write("=====\\n\\n")
7.             # IP Info
8.             file.write("IP Information:\\n")
9.             if "ip_info" in data and data["ip_info"]:
10.                 for key, value in data["ip_info"].items():
11.                     file.write(f" {key}: {value}\\n")
12.                 else:
13.                     file.write(" No IP information available.\\n")
14.                     file.write("\\n")
15.                     # Search Results
16.                     file.write("Search Results:\\n")
17.                     if "search_results" in data and data["search_results"]:
18.                         for idx, result in enumerate(data["search_results"], start=1):
19.                             file.write(f" Result {idx}:\\n")
20.                             for key, value in result.items():
21.                                 file.write(f"   {key}: {value}\\n")
22.                             else:
23.                                 file.write(" No search results available.\\n")
24.                                 file.write("\\n")
25.                                 # ICS Count
26.                                 file.write("Industrial Control Systems (ICS) Count:\\n")
27.                                 if "ics_count" in data:
28.                                     file.write(f" Total: {data['ics_count']}\\n")
29.                                 else:
30.                                     file.write(" ICS count information unavailable.\\n")
31.                                     logging.info(f'Results saved to {filename}')
32.                                     except Exception as e:
33.                                         logging.error(f'Failed to save results to {filename}: {e}')

```

- Rôle : Sauvegarde les résultats dans un fichier texte avec une mise en forme

organisée.

- Détails :

- Structure le fichier avec des sections : "IP Information", "Search Results", "ICS Count".
- Parcourt les données pour afficher les informations de l'IP, les résultats de recherche, et le comptage ICS.
- Gère les cas où les données sont absentes (ex. No IP information available).

Fonction search_shodan

```
61  def search_shodan(api, query, limit):
62      """Search Shodan for the given query and return results."""
63      results = []
64      try:
65          for banner in api.search_cursor(query):
66              results.append(banner)
67              if len(results) >= limit:
68                  break
69          logging.info(f"Found {len(results)} results for query: {query}")
70      return results
71  except APIError as e:
72      logging.error(f"Shodan search error: {e}")
73  return []
```

Rôle : Recherche des appareils correspondant à une requête (ex. "apache").

- Détails :

- Utilise api.search_cursor pour parcourir les résultats de manière efficace (pagination automatique).
- Limite les résultats au nombre spécifié (limit).
- Gère les erreurs API avec APIError (ex. clé invalide, quota dépassé).
- Retourne une liste de bannières (dictionnaires contenant IP, port, produit, etc.).

Fonction get_ics_count:

```
34.  def get_ics_count(api):
35.      """Get the count of industrial control systems services."""
36.      try:
37.          ics_services = api.count('tag:ics')
```

```

38.     total = ics_services.get("total", "Unknown")
39.     logging.info(f"Industrial Control Systems: {total}")
40.     return total
41. except APIError as e:
42.     logging.error(f"Shodan count error: {e}")
43.     return "Unknown"

```

- Rôle : Compte le nombre de systèmes ICS exposés via la requête tag:ics.
- Détails :
 - Utilise api.count pour obtenir une estimation sans récupérer les détails.
 - Extrait le champ total du résultat.
 - Gère les erreurs API et retourne "Unknown" en cas d'échec.

Fonction main_shodan :

```

44. def main_shodan():
45.     # Initialize Shodan API
46.     api = Shodan(API_KEY)
47.     # Get user input for the search query
48.     search_query = input("Enter your Shodan search
        query(EX:apache..): ").strip()
49.     if not search_query:
50.         logging.error("No search query provided. Exiting.")
51.     return
52.     # Lookup an IP address
53.     try:
54.         ipinfo = api.host('8.8.8.8')
55.         logging.info("IP Info retrieved successfully.")
56.     except APIError as e:
57.         logging.error(f"Shodan API error while retrieving IP info: {e}")
58.         ipinfo = {}
59.     # Search for websites that match the query
60.     search_results = search_shodan(api, search_query, RESULT_LIMIT)
61.     # Get the count of industrial control systems services
62.     ics_count = get_ics_count(api)
63.     # Combine all results into a single dictionary
64.     results = {
65.         "ip_info": ipinfo,
66.         "search_results": search_results,
67.         "ics_count": ics_count
68.     }
69.     # Save results to JSON and TXT files
70.     save_to_json(results, OUTPUT_JSON_FILE)
71.     save_to_txt(results, OUTPUT_TXT_FILE)

```

- Rôle : Point d'entrée interactif pour orchestrer le scan Shodan.

- Étapes :
 1. Initialise l'API Shodan avec la clé fournie.
 2. Demande une requête de recherche à l'utilisateur (ex. "apache").
 3. Récupère les informations sur l'IP 8.8.8.8.
 4. Effectue une recherche Shodan avec la requête utilisateur.
 5. Compte les systèmes ICS.
 6. Compile les résultats dans un dictionnaire.
 7. Sauvegarde les résultats en JSON et texte.

6. Exemple d'Utilisation

1. Installation des dépendances :
2. pip install shodan
3. Configuration :
 - o Remplacer API_KEY par une clé API Shodan valide (obtenue sur shodan.io).
4. Exécution :
5. python3 shodan_scan.py
 - o Saisir une requête : Ex. apache.
 - o Résultats sauvegardés dans shodan_results.json et shodan_results.txt.
6. Sortie attendue :
 - o Console :
 - o 2025-04-23 10:00:00,123 - INFO - Found 10 results for query: apache
 - o 2025-04-23 10:00:00,456 - INFO - IP Info retrieved successfully
 - o 2025-04-23 10:00:00,789 - INFO - Industrial Control Systems: 12345
 - o 2025-04-23 10:00:01,012 - INFO - Results saved to shodan_results.json
 - o 2025-04-23 10:00:01,345 - INFO - Results saved to shodan_results.txt

7. Importance dans le Projet PenAut

Le Shodan Scan est un module stratégique de PenAut pour les raisons suivantes :

1. Reconnaissance passive : Permet de collecter des informations sans interagir directement avec les cibles, réduisant le risque de détection.
2. Richesse des données : Fournit des informations détaillées (ports, services, bannières) exploitables pour planifier des audits approfondis.
3. Focus sur les ICS : Identifie les systèmes industriels vulnérables, un domaine

critique pour la cybersécurité.

4. Automatisation : Simplifie la collecte et la sauvegarde des données, intégrable dans des workflows DevSecOps.
5. Modularité : Peut être combiné avec d'autres modules PenAut (ex. CVE Scanner pour vérifier les vulnérabilités des services détectés).

IV. - OPTION4 URL ENUMERATION:

Introduction :

Cette option est basée sur un script Python qui permet d'effectuer une découverte automatique de sous-domaines, de répertoires et de fichiers web à partir de wordlists (listes de mots), en s'appuyant sur la technique de brute-force. Il est particulièrement utile pour les tests d'intrusion web (pentesting) ou les audits de sécurité.

Classe/Composant	Rôle
WordlistLoder	Charge les wordlists à partir de fichiers texte.
DomainScanner	Génère les URLs à tester et lance le scan via HTTP.
ThreadPoolExecutor	Permet le scan multithreadé pour plus de rapidité.
requests.get()	Tente de contacter chaque URL pour vérifier sa validité.

Figure 1: table de correspondance

Architecture du script:

Cas d'utilisation typiques

- Découverte de surface d'attaque : identifier les points d'entrée cachés sur un site.
- Reconnaissance passive/active : complément aux outils comme Gobuster ou Dirb.
- Automatisation des tests de sécurité web.

scan sous domaine:

Bibliothèque request

- Type : Bibliothèque tierce (non incluse dans la bibliothèque standard de Python).
- Installation : pip install requests
- Fonction : Permet d'envoyer des requêtes HTTP. Dans ce script, elle est utilisée pour envoyer des requêtes de type GET vers des sous-domaines construits dynamiquement, afin de vérifier leur accessibilité.

Fichier de données :

• subdomains.txt : Ce fichier texte contient une liste de sous-domaines à tester. Le script lit son contenu ligne par ligne pour générer les URLs complètes à scanner.

Le script implémente un scanner de sous-domaines en :

1. Chargant une liste de sous-domaines depuis un fichier texte ;
2. Générant pour chaque entrée une URL complète du type http://<subdomain>.<domain> ;
3. Testant l'accessibilité de chaque sous-domaine via une requête HTTP
- ; 4. Affichant les sous-domaines valides (accessibles).

scan sous repertoire:

Le script construit automatiquement des URL à tester en concaténant : un nom de répertoire ou de fichier (provenant d'une wordlist),

avec différentes extensions potentielles de fichiers.

["", ".txt", ".php", ".html", ".zip", ".rar", ".htm", ".asp", ".aspx", ".jsp"] Cette approche permet de détecter à la fois :

des répertoires (/admin/)

des fichiers exécutables ou non (/login.php, /backup.zip, etc.)

L'Objectif de sécurité:

Détecter ce type de ressources permet de :

- identifier des interfaces d'administration non protégées,
- découvrir des fichiers sensibles ou archives laissées en ligne par erreur,
- cartographier l'arborescence cachée d'un site web.

V. - OPTION6 CVE SEARCH :

1. Introduction

CVE Search est un module essentiel du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module interroge l'API NVD (National Vulnerability Database) pour rechercher des informations sur les vulnérabilités CVE (Common Vulnerabilities and Exposures) en fonction de critères spécifiques fournis par l'utilisateur, tels que l'ID CVE, le nom CPE, la gravité, les mots-clés, ou l'ID CWE. Les résultats sont récupérés au format JSON et sauvegardés dans un fichier pour une analyse ultérieure.

Le script est implémenté dans le fichier `cve_scan.py` et repose sur les bibliothèques Python `requests` pour les requêtes HTTP et `json` pour la gestion des données. Ce module s'inscrit dans l'approche DevSecOps de PenAut, mettant l'accent sur l'automatisation, la modularité, et l'intégration avec des bases de données publiques pour des audits de sécurité efficaces. Il est particulièrement utile pour les professionnels de la cybersécurité cherchant à identifier les failles potentielles dans les logiciels et matériels utilisés par une organisation.

2. Rôle dans le Pentesting

Le CVE Search joue un rôle clé dans la phase d'analyse des vulnérabilités d'un test d'intrusion. Ses objectifs principaux sont :

1. Identification des vulnérabilités connues : Rechercher des CVE spécifiques associées à des logiciels, systèmes d'exploitation, ou matériels détectés lors des phases de reconnaissance (ex. via le Port Scanner ou le Shodan Scan).
2. Filtrage ciblé : Permettre aux utilisateurs de filtrer les vulnérabilités selon des critères précis (ex. gravité critique, mot-clé "Apache") pour prioriser les failles les plus pertinentes.
3. Automatisation de la collecte de données : Simplifier l'accès aux informations de la NVD via une interface interactive, réduisant le besoin de recherches manuelles.

4. Support à la planification : Fournir des données exploitables pour planifier les correctifs ou les tests d'exploitation dans les étapes ultérieures du pentest.

Ce module est souvent utilisé après la découverte des services et logiciels (par exemple, via l'Option 2 : Port Scanner ou l'Option 3 : Shodan Scan) pour corrélérer les versions détectées avec des vulnérabilités connues. Les résultats peuvent être intégrés dans des rapports de sécurité ou utilisés pour guider les actions correctives.

3. Fonctionnalités

Le module CVE Search offre les fonctionnalités suivantes :

1. Interface utilisateur interactive :

- Propose un menu permettant à l'utilisateur de choisir parmi cinq critères de recherche : ID CVE, nom CPE, gravité CVSSv3, mots-clés, ou ID CWE.
- Exemple : Un utilisateur peut rechercher toutes les vulnérabilités critiques affectant Microsoft Windows 10.

2. Recherche via l'API NVD :

- Construit des requêtes HTTP avec des paramètres spécifiques (ex. cveId=CVE-2023-12345) pour interroger l'API NVD.
- Récupère les résultats au format JSON, incluant des détails comme la description, la gravité, et les références des CVE.

3. Sauvegarde des résultats :

- Enregistre les résultats dans un fichier JSON nommé cve_results.json avec une mise en forme lisible pour une analyse ultérieure.
- Exemple : Les données sauvegardées peuvent inclure l'ID CVE, la gravité, et les systèmes affectés.

4. Flexibilité des critères :

- Permet de rechercher par :
 - CVE ID : Identifiant unique (ex. CVE-2023-12345).
 - CPE Name : Nom de produit (ex. cpe:2.3:o:microsoft:windows_10:1607).
 - Gravité : Niveau CVSSv3 (LOW, MEDIUM, HIGH, CRITICAL).
 - Mots-clés : Termes associés (ex. "Apache").
 - CWE ID : Type de faiblesse (ex. CWE-287 pour une authentification incorrecte).

Ces fonctionnalités permettent aux pentesters de collecter rapidement des informations critiques sur les vulnérabilités, facilitant l'évaluation des risques et la priorisation des correctifs.

4. Bibliothèques Utilisées

Le script repose sur deux bibliothèques standard Python :

1. Requests :

- Description : Bibliothèque pour envoyer des requêtes HTTP (GET, POST, etc.).
- Rôle : Interroge l'API NVD en envoyant des requêtes GET avec des paramètres spécifiques.
- Installation : pip install requests.
- Avantage : Simplifie la gestion des requêtes HTTP avec une syntaxe claire et une gestion robuste des erreurs.

2. Json :

- Description : Bibliothèque standard pour manipuler des données au format JSON.
- Rôle : Parse les réponses de l'API et sauvegarde les résultats dans un fichier JSON structuré.
- Avantage : Permet une intégration facile avec d'autres outils ou scripts.

5. Analyse Détailée du Code

Le code fourni implémente une fonction principale et une fonction utilitaire pour interagir avec l'API NVD, collecter des données, et sauvegarder les résultats. Voici une analyse ligne par ligne, avec des explications techniques détaillées.

Analyse Ligne par Ligne

Imports

```
72. import requests  
73. import json
```

- requests : Importe la bibliothèque pour envoyer des requêtes HTTP à l'API NVD.
- json : Importe la bibliothèque pour parser les réponses JSON et sauvegarder les résultats.

Fonction search_cve

```
74. def search_cve(base_url, **kwargs):  
75.     """  
76.     Searches the NVD CVE API with given parameters.  
77.     :param base_url: Base URL of the API.  
78.     :param kwargs: Query parameters for the API.  
79.     :return: JSON response from the API.
```

```

80.      """
81.      response = requests.get(base_url, params=kwargs)
82.      if response.status_code == 200:
83.          return response.json()
84.      else:
85.          return {"error": f"Request failed with status code {response.status_code}"}

```

- Rôle : Envoie une requête GET à l'API NVD avec les paramètres spécifiés et retourne la réponse JSON.
- Détails :
 - base_url : URL de l'API NVD (<https://services.nvd.nist.gov/rest/json/cves/2.0>).
 - **kwargs : Paramètres de requête (ex. {"cveId": "CVE-2023-12345"}).
 - requests.get(base_url, params=kwargs) : Construit l'URL avec les paramètres (ex. base_url?cveId=CVE-2023-12345).
 - Vérifie le code de statut HTTP :
 - 200 : Retourne la réponse JSON.
 - Autre : Retourne un dictionnaire avec un message d'erreur.
- Exemple : Une requête avec cveId=CVE-2023-12345 retourne les détails de cette CVE.

Fonction main_cve

```

86.  def main_cve():
87.      base_url = "https://services.nvd.nist.gov/rest/json/cves/2.0"
88.      print("Choose search criteria:")
89.      print("1. Search by CVE ID")
90.      print("2. Search by CPE Name")
91.      print("3. Search by Severity (CVSSv3)")
92.      print("4. Search by Keywords")
93.      print("5. Search by CWE ID")
94.      choice = input("Enter your choice (1-5): ")
95.      params = {}
96.      if choice == "1":
97.          cve_id = input("Enter CVE ID (e.g., CVE-2023-12345): ")
98.          params["cveId"] = cve_id
99.      elif choice == "2":

```

```

100.     cpe_name = input("Enter CPE Name (e.g.,  

101.         cpe:2.3:o:microsoft:windows_10:1607): ")  

102.     params["cpeName"] = cpe_name  

103.     elif choice == "3":  

104.         severity = input("Enter severity level (LOW, MEDIUM, HIGH,  

105.             CRITICAL): ")  

106.         params["cvssV3Severity"] = severity.upper()  

107.     elif choice == "4":  

108.         keyword = input("Enter keyword for search (e.g., Windows): ")  

109.         params["keywordSearch"] = keyword  

110.     elif choice == "5":  

111.         cwe_id = input("Enter CWE ID (e.g., CWE-287): ")  

112.         params["cweld"] = cwe_id  

113.     else:  

114.         print("X Invalid choice! X")  

115.     return  

116.     print("Fetching results...")  

117.     result = search_cve(base_url, **params)  

118.     # Save results to a JSON file  

119.     with open("cve_results.json", "w") as file:  

120.         json.dump(result, file, indent=4)  

121.     print("Results saved to cve_results.json")

```

- Rôle : Point d'entrée interactif pour collecter les critères de recherche, interroger l'API, et sauvegarder les résultats.
- Etapes :
 1. Affiche un menu avec cinq options de recherche.
 2. Collecte le choix de l'utilisateur (1 à 5) via input.
 3. Selon le choix, demande une valeur spécifique (ex. ID CVE, mot-clé).
 4. Construit un dictionnaire params avec le paramètre correspondant (ex. {"cveId": "CVE-2023-12345"}).
 5. Appelle search_cve pour interroger l'API.
 6. Sauvegarde la réponse dans cve_results.json avec json.dump.
 7. Affiche un message de confirmation.
- Gestion des erreurs :
 - Vérifie si le choix est valide (1 à 5).

- Les erreurs HTTP sont gérées par search_cve.
-

6. Exemple d'Utilisation

1. Installation des dépendances :

2. pip install requests

3. Exécution :

4. python3 cve_scan.py

- Exemple d'interaction :

- Choose search criteria:

- 1. Search by CVE ID

- 2. Search by CPE Name

- 3. Search by Severity (CVSSv3)

- 4. Search by Keywords

- 5. Search by CWE ID

- Enter your choice (1-5): 1

- Enter CVE ID (e.g., CVE-2023-12345): CVE-2023-12345

- Fetching results...

- Results saved to cve_results.json

5. Sortie attendue :

```
format": "NVD_CVE",
"version": "2.0",
"timestamp": "2025-04-03T19:40:32.162",
"vulnerabilities": [
  {
    "cve": {
      "id": "CVE-2023-49031",
      "sourceIdentifier": "cve@mitre.org",
      "published": "2025-03-03T18:15:28.330",
      "lastModified": "2025-03-05T19:15:37.110",
      "vulnStatus": "Awaiting Analysis",
      "cveTags": [],
      "descriptions": [
        {
          "lang": "en",
          "value": "Directory Traversal (Local File Inclusion) vulnerability in Tikit (now Advanced) eMarketing platform 6.8.3.0 allows remote attackers to upload files via a crafted URL. This issue is tracked by CVE-2023-49031."}
        ],
        "metrics": {
          "cvssMetricV31": [
            "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
          ]
        }
      }
    ]
  }
]
```

7. Importance dans le Projet PenAut

Le CVE Search est un module stratégique de PenAut pour les raisons suivantes :

1. Identification des failles : Permet de corrélérer les services et logiciels détectés (ex. via Shodan ou Port Scanner) avec des vulnérabilités connues, facilitant l'évaluation des risques.
2. Automatisation : Simplifie la recherche de CVE grâce à une interface interactive et une sauvegarde automatisée, réduisant le temps nécessaire pour analyser les failles.
3. Flexibilité : Offre plusieurs critères de recherche, adaptés à différents scénarios (ex. recherche par mot-clé pour une analyse large, ou par ID CVE pour une vérification spécifique).
4. Intégration : Les résultats JSON peuvent être utilisés par d'autres modules PenAut ou intégrés dans des rapports de sécurité.
5. Accessibilité : L'interface utilisateur rend le module utilisable par des auditeurs novices tout en restant puissant pour les experts.

VI. OPTION7 HASH CRACKER:

1. Introduction

Hash Cracker est un module clé du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module est conçu pour casser des hachages non salés (non-salted hashes) en comparant un hachage cible avec des hachages générés à partir d'une liste de mots (wordlist). Il utilise la bibliothèque standard Python hashlib pour appliquer des algorithmes de hachage cryptographique, tels que MD5, SHA-1, SHA-256, et autres, afin d'identifier le mot de passe ou la chaîne d'origine correspondant au hachage cible.

Le script est implémenté dans le fichier hashcracker.py et repose sur hashlib pour générer des empreintes cryptographiques et sur os pour gérer les fichiers de wordlist. Ce module s'inscrit dans l'approche DevSecOps de PenAut, mettant l'accent sur l'automatisation, la modularité, et la simplicité d'utilisation. Il est particulièrement utile pour les auditeurs de sécurité cherchant à tester la robustesse des mots de passe ou à récupérer des informations à partir de hachages exposés lors d'un audit.

2. Rôle dans le Pentesting

Le Hash Cracker joue un rôle important dans la phase d'exploitation ou d'analyse des identifiants d'un test d'intrusion. Ses objectifs principaux sont :

1. Récupération de mots de passe : Identifier le mot de passe ou la chaîne d'origine à partir d'un hachage non salé obtenu, par exemple, à partir d'une base de données compromise ou d'un fichier de configuration.
2. Test de robustesse : Évaluer la force des mots de passe utilisés dans un système en tentant de casser leurs hachages avec une wordlist.
3. Automatisation de l'attaque par dictionnaire : Simplifier l'attaque par dictionnaire en automatisant la génération et la comparaison des hachages.
4. Support à l'audit : Fournir des informations exploitables pour recommander des politiques de mots de passe plus robustes (ex. éviter les mots de passe faibles ou utiliser le salage).

Ce module est souvent utilisé dans des scénarios où des hachages sont extraits lors d'un pentest, par exemple, à partir d'un dump de base de données ou d'un fichier de configuration mal sécurisé. Il complète d'autres modules PenAut, comme l'Option 7 : Password Pwned Check, pour évaluer la sécurité des mots de passe.

3. Fonctionnalités

Le module Hash Cracker offre les fonctionnalités suivantes :

1. Support de multiples algorithmes :

- Prend en charge les algorithmes de hachage fournis par hashlib : MD5, SHA-1, SHA-224, SHA-256, SHA-384, et SHA-512.
- Exemple : Un utilisateur peut tenter de casser un hachage SHA-256 extrait d'une application web.

2. Attaque par dictionnaire :

- Utilise une wordlist (liste de mots potentiels) pour générer des hachages et les comparer au hachage cible.
- Exemple : Utilisation de la wordlist rockyou.txt pour tester des mots de passe courants.

3. Interface utilisateur interactive :

- Permet à l'utilisateur de spécifier l'algorithme, le hachage cible, et le chemin de la wordlist via une interface en ligne de commande.
- Propose une wordlist par défaut (/usr/share/wordlists/rockyou.txt) pour simplifier l'utilisation.

4. Validation et gestion des erreurs :

- Vérifie la validité de l'algorithme et l'existence du fichier de wordlist avant de lancer l'attaque.
- Affiche des messages clairs en cas de succès (mot de passe trouvé) ou d'échec (mot de passe non trouvé).

Ces fonctionnalités permettent aux pentesters de tester rapidement la sécurité des hachages non salés et d'identifier les faiblesses dans les mécanismes de stockage

des mots de passe.

4. Bibliothèque Utilisée

Le script repose principalement sur la bibliothèque standard `hashlib`, avec un usage complémentaire de `os` :

1. Hashlib :

- Description : Bibliothèque standard Python pour générer des empreintes cryptographiques (digests) à partir de données.
- Fonctionnalités :
 - Supporte les algorithmes de hachage : MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.
 - Génère des hachages sous forme de chaînes hexadécimales via la méthode `hexdigest()`.
- Rôle : Applique l'algorithme choisi à chaque mot de la wordlist et compare le résultat avec le hachage cible.
- Avantage : Intégré à Python, rapide, et facile à utiliser sans dépendances externes.
- Limitation : Ne supporte pas les hachages salés ou les algorithmes plus complexes comme `bcrypt`.

2. Os :

- Description : Bibliothèque standard Python pour interagir avec le système de fichiers.
 - Rôle : Vérifie l'existence du fichier de wordlist et gère son accès.
 - Avantage : Simplifie la gestion des chemins de fichiers et la validation.
-

5. Analyse Détailée du Code

Le code fourni implémente une classe `NoSaltedHashCracker` pour effectuer une

attaque par dictionnaire sur des hachages non salés, ainsi qu'une fonction principale pour l'interaction utilisateur. Voici une analyse ligne par ligne, avec des explications techniques détaillées.

Analyse Ligne par Ligne

Imports

```
120. import hashlib  
121. import os
```

- hashlib : Importe la bibliothèque pour générer des hachages cryptographiques.
- os : Importe la bibliothèque pour vérifier l'existence du fichier de wordlist.

Classe NoSaltedHashCracker

```
122. class NoSaltedHashCracker:  
123.     def __init__(self, algo, wordlist_path, hash_value):  
124.         self.algorithm = algo.lower()  
125.         self.wordlist_path = wordlist_path  
126.         self.target_hash = hash_value
```

- Constructeur :
 - algo : Algorithme de hachage (ex. "md5", "sha256").
 - wordlist_path : Chemin vers le fichier de wordlist.
 - hash_value : Hachage cible à casser.
 - algo.lower() : Convertit l'algorithme en minuscules pour une gestion cohérente.

Méthode cracking

```

127.     def cracking(self):
128.         if self.algorithm not in ['md5', 'sha1', 'sha256', 'sha512', 'sha224',
129.             'sha384']:
130.             print(f"❌ Error: '{self.algorithm}' is not a supported hash
131.                 algorithm.")
132.             return
133.         if not os.path.isfile(self.wordlist_path):
134.             print(f"❌ Error: Wordlist file '{self.wordlist_path}' was not
135.                 found.")
136.             return
137.         with open(self.wordlist_path, 'r', encoding='utf-8',
138.             errors='ignore') as file:
139.             for line in file:
140.                 word = line.strip()
141.                 hashed = getattr(hashlib,
142.                     self.algorithm)(word.encode()).hexdigest()
143.                 if hashed == self.target_hash:
144.                     print(f"✅ Password found: {word}")
145.                     return
146.             print("❌ Password not found in the wordlist.")

```

- Rôle : Effectue l'attaque par dictionnaire pour casser le hachage.
- Validation :
 - Vérifie si self.algorithm est supporté (MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512).
 - Vérifie si le fichier de wordlist existe avec os.path.isfile.
- Processus de cracking :

- Ouvre le fichier de wordlist en mode lecture avec encodage utf-8 et errors='ignore' pour gérer les caractères non valides.
- Parcourt chaque ligne, supprime les espaces avec strip().
- Encode le mot en bytes avec word.encode() et génère son hachage avec l'algorithme spécifié via getattr(hashlib, self.algorithm).
- Utilise hexdigest() pour obtenir le hachage sous forme hexadécimale.
- Compare le hachage généré avec self.target_hash.
- Sortie :
 - Si une correspondance est trouvée, affiche le mot et termine.
 - Sinon, affiche un message d'échec.

Fonction main_hashcracker

```

142. def main_hashcracker():

143.     algo = input("Which algorithm? [md5, sha1, sha256, sha512,
   sha224, sha384]: ").strip().lower()

144.     hash_value = input("Enter the hash to crack: ").strip()

145.     wordlist_path = input("Wordlist path? (Press Enter for default:
   /usr/share/wordlists/rockyou.txt): ").strip()

146.     if not wordlist_path:

147.         wordlist_path = "/usr/share/wordlists/rockyou.txt"

148.     cracker = NoSaltedHashCracker(algo, wordlist_path, hash_value)

149.     cracker.cracking()

```

- Rôle : Point d'entrée interactif pour collecter les paramètres et lancer l'attaque.
- Étapes :
 - Demande l'algorithme, le hachage cible, et le chemin de la wordlist.
 - Utilise une wordlist par défaut (/usr/share/wordlists/rockyou.txt) si

aucun chemin n'est fourni.

- Crée une instance de NoSaltedHashCracker et appelle cracking.
-

6. Exemple d'Utilisation

1. Prérequis :

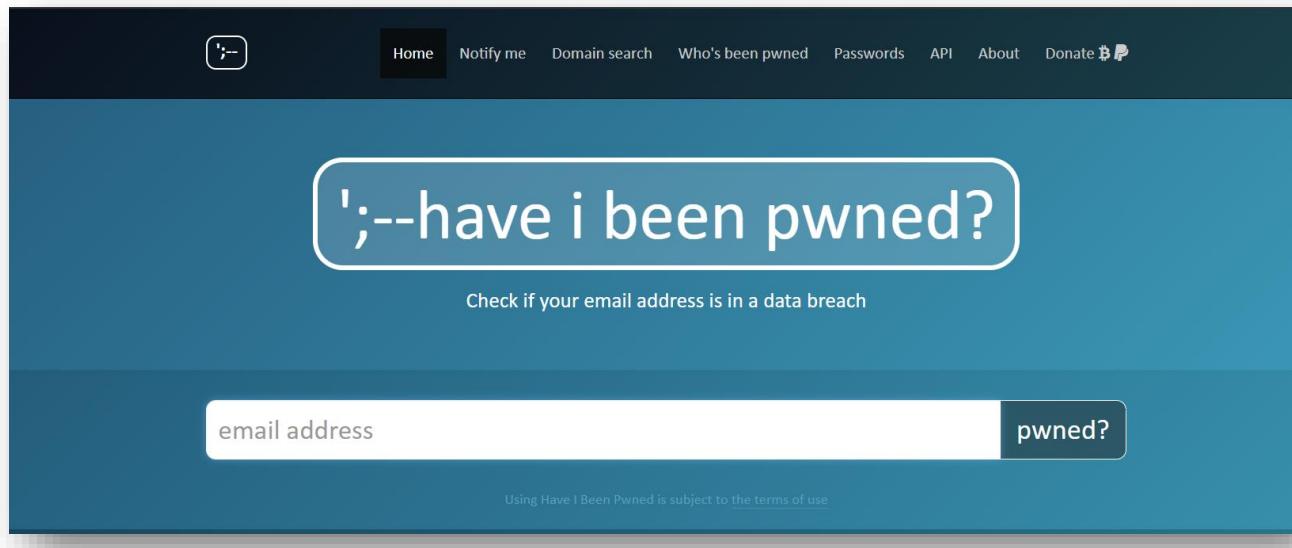
- Une wordlist (ex. rockyou.txt) doit être disponible. Sur Linux, elle est souvent située dans /usr/share/wordlists/.
- Exemple de hachage cible :
 - Mot de passe : password
 - Hachage MD5 : 5f4dcc3b5aa765d61d8327deb882cf99

VII. -OPTION8 PASSWORD PWNED:

1. Introduction

L'Option 7 : Password Pwned Check est un module stratégique du projet PenAut, une suite d'outils d'automatisation de tests d'intrusion (pentesting) développée en Python. Ce module permet de vérifier si un mot de passe donné figure dans des bases de données de mots de passe compromis, en utilisant l'API Pwned Passwords du service Have I Been Pwned. Le script implémente une méthode sécurisée basée sur la technique de K-Anonymity, garantissant que le mot de passe complet n'est jamais transmis à l'API, préservant ainsi la confidentialité des données de l'utilisateur.

Le script est implémenté dans le fichier password_pwned.py et repose sur les bibliothèques Python hashlib pour générer des hachages SHA-1 et requests pour interroger l'API. Ce module s'inscrit dans l'approche DevSecOps de PenAut, mettant l'accent sur la sécurité, l'automatisation, et la simplicité d'utilisation. Il est particulièrement utile pour les auditeurs de sécurité cherchant à évaluer la robustesse des mots de passe utilisés dans une organisation et à identifier ceux qui ont été exposés dans des fuites de données.



2. Rôle dans le Pentesting

Le Password Pwned Check joue un rôle clé dans la phase d'analyse de la sécurité des identifiants d'un test d'intrusion. Ses objectifs principaux sont :

1. Vérification de la compromission des mots de passe : Déterminer si un mot de passe figure dans des bases de données de fuites connues, indiquant un

- risque de réutilisation ou de compromission.
2. Renforcement de la sécurité : Identifier les mots de passe faibles ou exposés pour recommander leur remplacement par des alternatives plus robustes.
 3. Confidentialité des données : Utiliser la K-Anonymity pour vérifier les mots de passe sans révéler leur contenu à l'API ou à des tiers.
 4. Automatisation : Simplifier l'évaluation des mots de passe via une interface interactive, réduisant le besoin de recherches manuelles.

Ce module est souvent utilisé dans des audits de sécurité pour tester les mots de passe des utilisateurs, des systèmes, ou des applications. Il complète d'autres modules PenAut, comme l'Option 6 : Hash Cracker, en fournissant une analyse préventive des mots de passe avant leur utilisation ou après leur extraction.

3. Fonctionnalités

Le module Password Pwned Check offre les fonctionnalités suivantes :

1. Vérification sécurisée via K-Anonymity :
 - Génère un hachage SHA-1 du mot de passe et n'envoie que les 5 premiers caractères (préfixe) à l'API Pwned Passwords.
 - Compare localement le suffixe du hachage avec les suffixes retournés par l'API, garantissant que le mot de passe complet reste confidentiel.
2. Interrogation de l'API Pwned Passwords :
 - Envoie une requête HTTP à <https://api.pwnedpasswords.com/range/{préfixe}> pour récupérer une liste de suffixes de hachages correspondant au préfixe.
 - Analyse la réponse pour déterminer si le mot de passe est compromis.
3. Interface utilisateur interactive :
 - Permet à l'utilisateur de saisir un mot de passe via la ligne de commande.
 - Affiche un message clair indiquant si le mot de passe est compromis ou non.
4. Gestion des erreurs :
 - Vérifie les codes de statut HTTP pour gérer les erreurs de requête (ex. problème de connexion, erreur serveur).
 - Fournit des messages d'erreur explicites pour l'utilisateur.

Ces fonctionnalités permettent aux pentesters de vérifier rapidement la sécurité des mots de passe tout en maintenant un haut niveau de confidentialité, une exigence cruciale dans les audits de cybersécurité.

4. Technique de K-Anonymity

La K-Anonymity est une technique de protection des données visant à empêcher l'identification individuelle dans un ensemble de données. Dans le contexte de ce module, elle est utilisée pour vérifier les mots de passe sans révéler leur contenu à l'API Pwned Passwords. Voici comment elle fonctionne :

1. Principe :

- Un mot de passe est haché avec l'algorithme SHA-1, produisant une empreinte unique (ex. 5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8).
- Le hachage est divisé en deux parties :
 - Préfixe : Les 5 premiers caractères (ex. 5BAA6).
 - Suffixe : Le reste du hachage (ex. 1E4C9B93F3F0682250B6CF8331B7EE68FD8).
- Seule la partie préfixe est envoyée à l'API, qui retourne une liste de suffixes associés à ce préfixe (avec le nombre d'occurrences dans des fuites).

2. Comparaison locale :

- Le script compare localement le suffixe du hachage avec les suffixes renvoyés.
- Si une correspondance est trouvée, le mot de passe est considéré comme compromis.

3. Confidentialité :

- Le mot de passe complet ou son hachage complet ne sont jamais transmis à l'API.
- L'API reçoit uniquement un préfixe, qui correspond à des milliers de hachages possibles (k-anonymat), rendant impossible l'identification du mot de passe spécifique.

4. Avantage :

- Réduit le risque de fuite de données, même si l'API est compromise ou si les communications sont interceptées.
- Permet une vérification sécurisée à distance, alignée avec les bonnes pratiques de cybersécurité.

5. Bibliothèques Utilisées

Le script repose sur deux bibliothèques Python :

1. Hashlib :

- Description : Bibliothèque standard Python pour générer des empreintes cryptographiques.

- Rôle : Génère un hachage SHA-1 du mot de passe pour la vérification via l'API Pwned Passwords.
- Avantage : Intégré à Python, rapide, et fiable pour les hachages SHA-1.
- Limitation : Limité aux algorithmes de hachage standards (ici, uniquement SHA-1 est utilisé).

2. Requests :

- Description : Bibliothèque pour envoyer des requêtes HTTP.
- Rôle : Envoie une requête GET à l'API Pwned Passwords pour récupérer les suffixes des hachages.
- Installation : pip install requests.
- Avantage : Simplifie les interactions HTTP avec une gestion robuste des erreurs.

6. Analyse Détailée du Code

Le code fourni implémente une classe CheckPasswordPwned pour vérifier si un mot de passe est compromis, ainsi qu'une fonction principale pour l'interaction utilisateur. Voici une analyse ligne par ligne, avec des explications techniques détaillées.

Analyse Ligne par Ligne

Imports

```
150. import hashlib
151. import requests
```

- hashlib : Importe la bibliothèque pour générer le hachage SHA-1 du mot de passe.
- requests : Importe la bibliothèque pour envoyer des requêtes HTTP à l'API Pwned Passwords.

Classe CheckPasswordPwned

```
152. class CheckPasswordPwned:
153.     def __init__(self, passwd):
154.         self.password = passwd
```

- Constructeur :
 - passwd : Mot de passe à vérifier.
 - Stocke le mot de passe dans self.password pour une utilisation

ultérieure.

Méthode check

```
155.     def check(self):  
156.         sha1 = hashlib.sha1(self.password.encode('utf-  
157.             8')).hexdigest().upper()  
158.         prefix = sha1[:5]  
159.         suffix = sha1[5:]  
160.         url = f"https://api.pwnedpasswords.com/range/{prefix}"  
161.         response = requests.get(url)  
162.         if response.status_code == 200:  
163.             hashes = response.text.splitlines()  
164.             if found:  
165.                 print("⚠ Ce mot de passe a été compromis !")  
166.             else:  
167.                 print("✓ Ce mot de passe ne figure pas dans les fuites  
168. connues.")  
169.             else:  
170.                 print(f"Erreur {response.status_code} : {response.text}")
```

- Rôle : Vérifie si le mot de passe est compromis en utilisant l'API Pwned Passwords et la K-Anonymity.
- Hachage :
 - hashlib.sha1(self.password.encode('utf-8')) : Encode le mot de passe en UTF-8 et génère son hachage SHA-1.
 - hexdigest().upper() : Convertit le hachage en une chaîne hexadécimale en majuscules (ex. 5BAA61E4...).
 - prefix = sha1[:5] : Extrait les 5 premiers caractères (ex. 5BAA6).
 - suffix = sha1[5:] : Extrait le reste du hachage (ex. 1E4C9B93...).
- Requête API :
 - url = f"https://api.pwnedpasswords.com/range/{prefix}" : Construit l'URL de l'API avec le préfixe.
 - requests.get(url) : Envoie une requête GET pour récupérer les suffixes associés.
- Traitement de la réponse :
 - Vérifie si le code de statut HTTP est 200 (succès).
 - response.text.splitlines() : Divise la réponse en lignes, chaque ligne

- contenant un suffixe et un compteur (ex. 1E4C9B93...:123).
- any(suffix in line for line in hashes) : Vérifie si le suffixe du hachage est présent dans une des lignes.
- Sortie :
 - Si une correspondance est trouvée : Affiche un avertissement de compromission.
 - Sinon : Confirme que le mot de passe est sûr.
 - En cas d'erreur HTTP : Affiche le code d'erreur et le message.

Fonction main_passwnd

```

170. def main_passwend():
171.     pass1 = input("Enter the password to check: ")
172.     password = CheckPasswordPwned(pass1)
173.     password.check()

```

- Rôle : Point d'entrée interactif pour collecter le mot de passe et lancer la vérification.
- Étapes :
 - Demande à l'utilisateur de saisir un mot de passe.
 - Crée une instance de CheckPasswordPwned avec le mot de passe.
 - Appelle la méthode check pour effectuer la vérification.

4 - CONCLUSION

Le projet PenAut concrétise une suite d'outils de cybersécurité robustes et modulaires, intégralement développés en Python et structurés selon une architecture orientée objet où chaque fonctionnalité (analyse de mots de passe, scan de vulnérabilités, reconnaissance de sous-domaines, cracking de hachages, reconnaissance active d'hôtes, scan réseau, interaction avec les API NVD et Shodan, et scan de domaines) est encapsulée dans une classe dédiée et implémentée dans un fichier Python distinct. Cette conception favorise une maintenabilité et une extensibilité accrues, permettant l'importation sélective des modules dans un fichier principal (main) pour orchestrer les différentes analyses. Afin de faciliter le déploiement et l'intégration dans un environnement Linux, un script d'installation en Bash (setup) automatise la préparation de l'environnement de travail, notamment en plaçant le répertoire du projet dans

/bin/lib et en créant un outil système exécutable nommé PenAut. Cette approche permet de transformer cet ensemble de scripts Python en un outil de cybersécurité Linux cohérent et accessible via la ligne de commande, tirant parti des capacités techniques avancées décrites dans l'abstract pour l'évaluation de la sécurité et la reconnaissance de systèmes et de réseaux.

5 - RÉFÉRENCE

- hashlib :
<https://docs.python.org/3/library/hashlib.html> • requests :
<https://requests.readthedocs.io/en/latest/>
- scapy :
<https://scapy.readthedocs.io/en/latest/> • tabulate :
<https://pypi.org/project/tabulate/>
- subprocess :
<https://docs.python.org/3/library/subprocess.html> • argparse :
<https://docs.python.org/3/library/argparse.html>
- json :
<https://docs.python.org/3/library/json.html> • os :
<https://docs.python.org/3/library/os.html>
- concurrent.futures :
<https://docs.python.org/3/library/concurrent.futures.html>
- shodan :
<https://shodan.readthedocs.io/en/latest/>
- logging :
<https://docs.python.org/3/library/logging.html>

Articles et Guides en Ligne :

- Recherchez des termes comme "home lab cybersecurity", "penetration testing lab setup", "virtualization for security testing" sur des blogs de sécurité, des forums (comme Reddit r/homelab, r/netsec), et des sites spécialisés. Vous trouverez de nombreux guides détaillés avec différentes configurations.

BONNE UTILISATION

<https://github.com/amsou-ismail/PenAut.git>

The screenshot shows the GitHub repository page for 'PenAut'. The repository is public and was forked from 'amsou-ismail/PenAut'. The main branch is 'main' (1 branch, 0 tags). The repository has 53 commits. A recent commit by 'AMARS44D' updated 'tool_style.py'. Other files updated include 'PenAut.pack', 'README.md', 'requirement.txt', and 'setup.sh'. The 'About' section describes PenAut as an automation toolkit for penetration testing. The 'Releases' section indicates no releases have been published.

PenAut : Automate penetration testing to detect vulnerabilities in computer systems. This project provides tools to simulate attacks, analyze security flaws, and generate detailed reports, thereby improving cybersecurity and the effectiveness of security assessments.

No releases published
[Create a new release](#)

6 - ANNEXE

6-1 DIAGRAMME DE CLASS

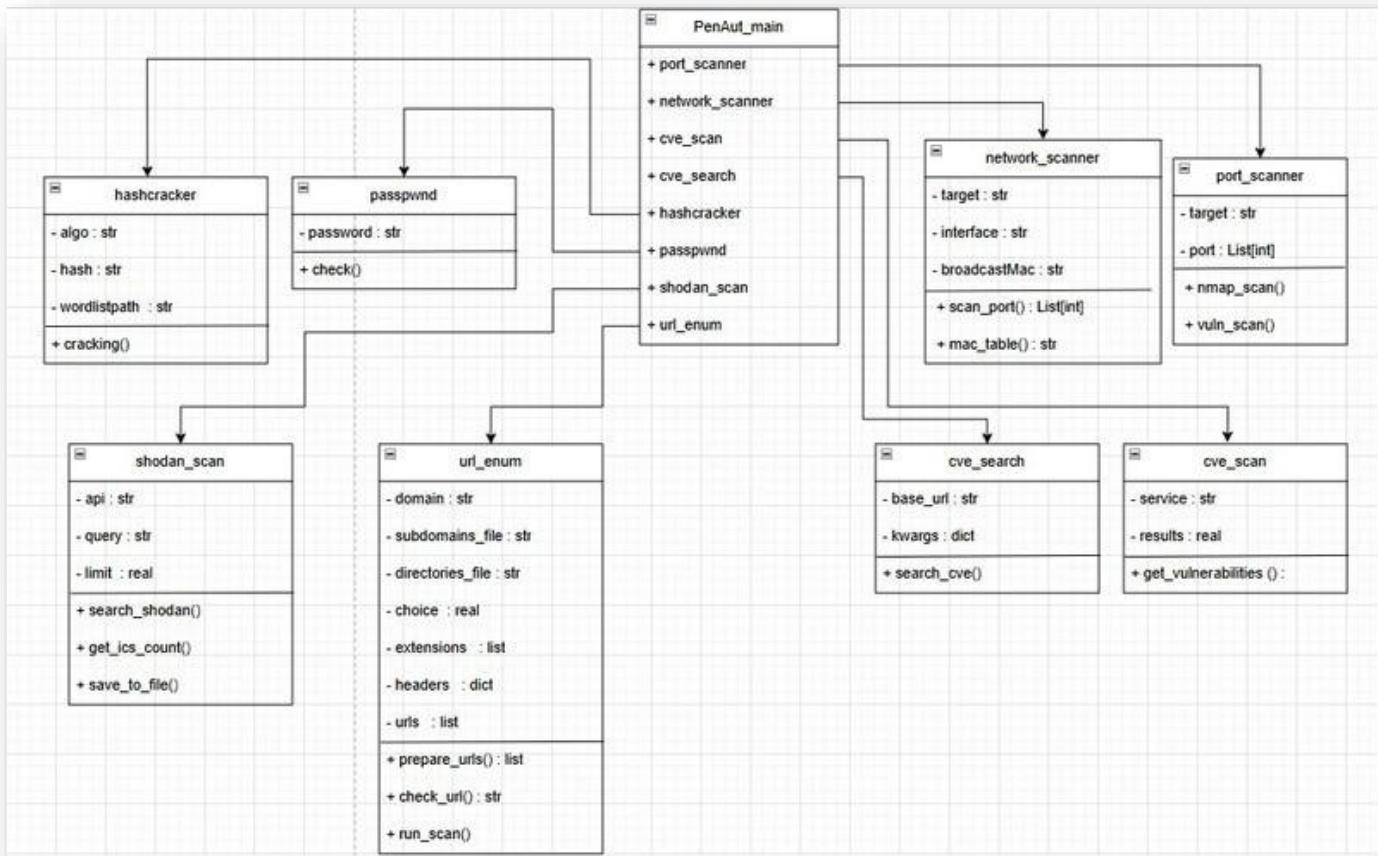


Figure 2 : Modélisation par diagramme de classes

Ce diagramme de classes illustre l'architecture logicielle de l'outil PenAut, mettant en évidence sa conception modulaire axée sur des classes Python dédiées à des fonctionnalités spécifiques de cybersécurité. La classe centrale PenAut_main agit comme un orchestrateur, instanciant et coordonnant les modules spécialisés tels que port_scanner pour l'analyse des ports, network_scanner pour l'exploration du réseau local, cve_scan pour la recherche de vulnérabilités, hashcracker pour la tentative de cassage de hachages, passpwnd pour la vérification de mots de passe compromis, shodan_scan pour l'interrogation de l'API Shodan, et url_enum pour l'énumération d'URLs. Chaque classe encapsule des attributs

représentant les données nécessaires à son fonctionnement et des méthodes définissant ses actions, favorisant ainsi la réutilisabilité du code et une organisation claire des fonctionnalités de l'outil. Les relations de dépendance indiquées soulignent la manière dont PenAut_main utilise les autres modules pour réaliser ses analyses.

6-2 HOME LAB

Pour tester efficacement et en toute sécurité les fonctionnalités de scan réseau et de reconnaissance active implémentées dans les codes fournis, la mise en place d'un home lab dédié est essentielle. Ce banc d'essai contrôlé permet d'observer le comportement des scripts (main_net() pour le scan ARP et main_port() pour le scan de ports, de vulnérabilités et l'énumération web) sans risquer d'effectuer des opérations potentiellement intrusives sur des réseaux ou des systèmes non autorisés, ce qui est illégal et unethical. Les composants clés de ce home lab incluent un ordinateur principal (idéalement sous Linux pour une compatibilité optimale des outils de sécurité) sur lequel exécuter les scripts, ainsi qu'au moins deux machines virtuelles hébergées sur ce système à l'aide d'un logiciel de virtualisation comme VirtualBox ou VMware. Ces machines virtuelles serviront de cibles : l'une configurée comme un serveur avec divers services (HTTP, SSH, etc.) pour tester main_port(), et l'autre pouvant être une machine avec une configuration réseau différente pour observer le scan ARP de main_net(). En configurant ces machines virtuelles sur un réseau interne isolé au sein du logiciel de virtualisation, on s'assure que les tests restent confinés et n'interfèrent pas avec le réseau domestique principal ou d'autres systèmes. Ce setup minimaliste permet de valider la logique des scripts, d'analyser les résultats des scans et de comprendre en profondeur les techniques de reconnaissance réseau dans un environnement d'apprentissage sûr et légal.

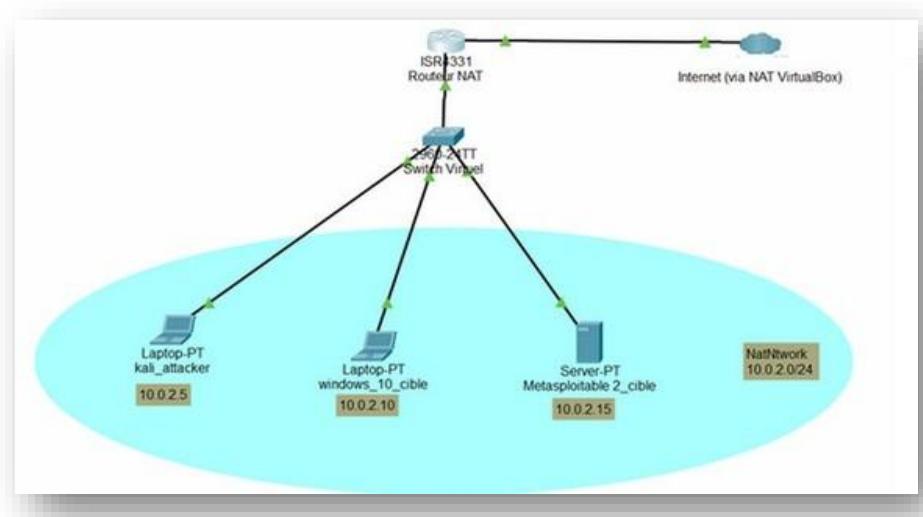


Figure3 : maquette du HomeLab

Merci . . .