■ DOCUMENTATION DÉTAILLÉE COMPLÈTE

Chaque ligne, même les vides et les commentaires

Analyse: Ligne par ligne SANS exception
Couverture: Commentaires, espaces, accolades, tout!
Date: 20/01/2026 10:08

■ db/mysql_conn.py

```
[1] import mysql.connector
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): mysql.connector
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[2] from mysql.connector import Error
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT Error depuis le module mysql.connector
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[3] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[4] def get_connection():
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'get_connection' réutilisable.
Paramètres: aucun
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[5] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[6] conn = mysql.connector.connect(
```

ASSIGNATION (Affectation):
Stocke la valeur mysql.connector.connect(dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[7] host="database",
```

ASSIGNATION (Affectation):
Stocke la valeur "database", dans la variable host
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[8] user="admin",
```

ASSIGNATION (Affectation):
Stocke la valeur "admin", dans la variable user
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[9] password="admin7791",
```

ASSIGNATION (Affectation):
Stocke la valeur "admin7791", dans la variable password
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[10] database="lab_test"
```

ASSIGNATION (Affectation):
Stocke la valeur "lab_test" dans la variable database
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[11] )
```

DÉLIMITEUR DE FERMETURE:
Ferme une parenthèse ou un crochet ouvert précédemment.
Importance: Chaque caractère doit avoir son équivalent de fermeture pour que le code soit syntaxiquement correct.

```
[12] if conn.is_connected():
```

APPEL DE FONCTION:
Exécute la fonction 'is_connected' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de is_connected, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[13] return conn
```

INSTRUCTION PYTHON:
return conn
Rôle: Exécute une opération ou logique du programme.

```
[14] except Error as e:
```

INSTRUCTION PYTHON:
except Error as e:
Rôle: Exécute une opération ou logique du programme.

```
[15] print(f"MySQL Error: {e}")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"MySQL Error: {e}"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[16] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

```
[17] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[18] if __name__ == "__main__":
```

CONDITION IF:
Teste si l'expression est VRAIE: __name__ == "__main__"
Flux: Si VRAI → exécute le bloc qui suit. Si FAUX → passe au elif/else.
Exemple réel: 'if conn.is_connected():' → vérifier si la connexion est active avant de l'utiliser.

```
[19] conn = get_connection()
```

ASSIGNATION (Affectation):
Stocke la valeur get_connection() dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[20] if conn:
```

INSTRUCTION PYTHON:
if conn:
Rôle: Exécute une opération ou logique du programme.

```
[21] conn.close()
```

APPEL DE FONCTION:
Exécute la fonction 'close' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de close, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

■ exploit/attack_chains/apache.py

```
[1] def evaluate_apache(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_apache' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "Apache",
```

INSTRUCTION PYTHON:
"service": "Apache",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "2.4.65",
```

INSTRUCTION PYTHON:
"version": "2.4.65",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[12] version = facts.get("version") or ""
```

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[13] cves = set(facts.get("validated_cves", []))
```

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[14] headers = " ".join(facts.get("headers", [])).lower()
```

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[15] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[16] if "apache" not in service:
```

INSTRUCTION PYTHON:
if "apache" not in service:
Rôle: Exécute une opération ou logique du programme.

```
[17] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

```
[18] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[19] has_php = "php" in headers
```

ASSIGNATION (Affectation):
Stocke la valeur "php" in headers dans la variable has_php
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[20] is_windows = "win" in headers
```

ASSIGNATION (Affectation):
Stocke la valeur "win" in headers dans la variable is_windows
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[21] is_linux = not is_windows
```

ASSIGNATION (Affectation):
Stocke la valeur not is_windows dans la variable is_linux
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[22] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[23] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[24] # 1■■ Apache Normalize Path → RCE
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ Apache Normalize Path → RCE"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[25] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[26] if version.startswith("2.4.") and any(
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "2.4."
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[27] cve in cves for cve in {
```

INSTRUCTION PYTHON:
cve in cves for cve in {
Rôle: Exécute une opération ou logique du programme.

```
[28] "CVE-2022-22720",
```

INSTRUCTION PYTHON:
"CVE-2022-22720",
Rôle: Exécute une opération ou logique du programme.

```
[29] "CVE-2022-22721",
```

INSTRUCTION PYTHON:
"CVE-2022-22721",
Rôle: Exécute une opération ou logique du programme.

```
[30] "CVE-2022-23943"
```

INSTRUCTION PYTHON:
"CVE-2022-23943"
Rôle: Exécute une opération ou logique du programme.

```
[31] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[32] ):
```

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

```
[33] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[34] "attack": "Apache path traversal → RCE",
```

INSTRUCTION PYTHON:
"attack": "Apache path traversal → RCE",
Rôle: Exécute une opération ou logique du programme.

`[35] "confidence": 0.95,`

INSTRUCTION PYTHON:
"confidence": 0.95,
Rôle: Exécute une opération ou logique du programme.

`[36] "exploit": "exploit/multi/http/apache_normalize_path_rce"`

INSTRUCTION PYTHON:
"exploit": "exploit/multi/http/apache_normalize_path_rce"
Rôle: Exécute une opération ou logique du programme.

`[37] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

`[38] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[39] # ==================================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=================================="
Utilité: Documenter le code et expliquer les décisions de conception.

`[40] # 2■■ Apache + PHP → Web RCE surface`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ Apache + PHP → Web RCE surface"
Utilité: Documenter le code et expliquer les décisions de conception.

`[41] # ==================================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=================================="
Utilité: Documenter le code et expliquer les décisions de conception.

`[42] if has_php and any(cve.startswith("CVE-2025") for cve in cves):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-2025"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[43] return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

`[44] "attack": "Apache + PHP attack surface (RCE candidates)",`

APPEL DE FONCTION:
Exécute la fonction 'surface' avec les paramètres: RCE candidates
Flux d'exécution: Saute à la définition de surface, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[45] "confidence": 0.70,`

INSTRUCTION PYTHON:
"confidence": 0.70,
Rôle: Exécute une opération ou logique du programme.

`[46] "exploit": "exploit/multi/http/apache_php_rce"`

INSTRUCTION PYTHON:
"exploit": "exploit/multi/http/apache_php_rce"
Rôle: Exécute une opération ou logique du programme.

```
[47] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[48] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[49] # ================================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[50] # 3■■ Apache misconfig / info-leak
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ Apache misconfig / info-leak"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[51] # ================================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[52] if "CVE-2025-58098" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2025-58098" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[53] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[54] "attack": "Apache fingerprinting / header info leak",
```

INSTRUCTION PYTHON:
"attack": "Apache fingerprinting / header info leak",
Rôle: Exécute une opération ou logique du programme.

```
[55] "confidence": 0.45,
```

INSTRUCTION PYTHON:
"confidence": 0.45,
Rôle: Exécute une opération ou logique du programme.

```
[56] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[57] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[58] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[59] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

exploit/attack_chains/chain_engine.py

```
[1] from scripts.exploit.attack_chains.apache import evaluate_apache
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT evaluate_apache depuis le module scripts.exploit.attack_chains.apache
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[2] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[3] def evaluate_chain(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_chain' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[4] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[5] Appelle tous les moteurs de chaînes (Apache, SSH, etc)
```

APPEL DE FONCTION:
Exécute la fonction 'chaînes' avec les paramètres: Apache, SSH, etc
Flux d'exécution: Saute à la définition de chaînes, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[6] et retourne la première chaîne valide.
```

INSTRUCTION PYTHON:
et retourne la première chaîne valide.
Rôle: Exécute une opération ou logique du programme.

```
[7] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[8] for engine in [evaluate_apache]:
```

INSTRUCTION PYTHON:
for engine in [evaluate_apache]:
Rôle: Exécute une opération ou logique du programme.

```
[9] result = engine(facts)
```

ASSIGNATION (Affectation):
Stocke la valeur engine(facts) dans la variable result
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[10] if result:
```

INSTRUCTION PYTHON:
if result:
Rôle: Exécute une opération ou logique du programme.

```
[11] return result
```

INSTRUCTION PYTHON:
return result

Rôle: Exécute une opération ou logique du programme.

```
[12] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/attack_chains/ftp.py

```
[1] def evaluate_ftp(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_ftp' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "vsftpd" ou "Pure-FTPd" ou "ProFTPD" ou "FileZilla Server",
```

INSTRUCTION PYTHON:
"service": "vsftpd" ou "Pure-FTPd" ou "ProFTPD" ou "FileZilla Server",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "2.3.4",
```

INSTRUCTION PYTHON:
"version": "2.3.4",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[12] version = facts.get("version") or ""`

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[13] cves = set(facts.get("validated_cves", []))`

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[14] headers = " ".join(facts.get("headers", [])).lower()`

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[15] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[16] # Vérifier si c'est un service FTP`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Vérifier si c'est un service FTP"
Utilité: Documenter le code et expliquer les décisions de conception.

`[17] ftp_services = ["ftp", "vsftpd", "pure-ftpd", "proftpd", "filezilla"]`

ASSIGNATION (Affectation):
Stocke la valeur ["ftp", "vsftpd", "pure-ftpd", "proftpd", "filezilla"] dans la variable ftp_services
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[18] if not any(ftp_svc in service for ftp_svc in ftp_services):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: ftp_svc in service for ftp_svc in ftp_services
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[19] return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

`[20] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[21] # ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

`[22] # 1■■ vsftpd Backdoor (CVE-2011-0762)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ vsftpd Backdoor (CVE-2011-0762)"

Utilité: Documenter le code et expliquer les décisions de conception.

```
[23] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[24] if "vsftpd" in service and version.startswith("2.3.") and "CVE-2011-0762" in cves:
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "2.3."
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[25] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[26] "attack": "vsftpd 2.3.4 backdoor RCE",
```

INSTRUCTION PYTHON:
"attack": "vsftpd 2.3.4 backdoor RCE",
Rôle: Exécute une opération ou logique du programme.

```
[27] "confidence": 0.95,
```

INSTRUCTION PYTHON:
"confidence": 0.95,
Rôle: Exécute une opération ou logique du programme.

```
[28] "exploit": "exploit/ftp/vsftpd_234_backdoor"
```

INSTRUCTION PYTHON:
"exploit": "exploit/ftp/vsftpd_234_backdoor"
Rôle: Exécute une opération ou logique du programme.

```
[29] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[30] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[31] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[32] # 2■■ ProFTPD Mod_Copy RCE
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ ProFTPD Mod_Copy RCE"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[33] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[34] if "proftpd" in service and any(
```

INSTRUCTION PYTHON:
if "proftpd" in service and any(
Rôle: Exécute une opération ou logique du programme.

[35] `cve in cves for cve in {`

INSTRUCTION PYTHON:
cve in cves for cve in {
Rôle: Exécute une opération ou logique du programme.

[36] `"CVE-2015-3306",`

INSTRUCTION PYTHON:
"CVE-2015-3306",
Rôle: Exécute une opération ou logique du programme.

[37] `"CVE-2015-3317"`

INSTRUCTION PYTHON:
"CVE-2015-3317"
Rôle: Exécute une opération ou logique du programme.

[38] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[39] `):`

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

[40] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[41] `"attack": "ProFTPD mod_copy RCE",`

INSTRUCTION PYTHON:
"attack": "ProFTPD mod_copy RCE",
Rôle: Exécute une opération ou logique du programme.

[42] `"confidence": 0.85,`

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

[43] `"exploit": "exploit/ftp/proftpd_modcopy_exec"`

INSTRUCTION PYTHON:
"exploit": "exploit/ftp/proftpd_modcopy_exec"
Rôle: Exécute une opération ou logique du programme.

[44] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[45] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[46] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[47] `# 3■■ Pure-FTPd Auth Bypass`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ Pure-FTPd Auth Bypass"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[48] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[49] if "pure-ftpd" in service and any(
```

INSTRUCTION PYTHON:
if "pure-ftpd" in service and any(
Rôle: Exécute une opération ou logique du programme.

```
[50] cve.startswith("CVE-2021") or cve.startswith("CVE-2020") for cve in cves
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "CVE-2021"
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[51] ):
```

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

```
[52] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[53] "attack": "Pure-FTPd authentication bypass / privilege escalation",
```

INSTRUCTION PYTHON:
"attack": "Pure-FTPd authentication bypass / privilege escalation",
Rôle: Exécute une opération ou logique du programme.

```
[54] "confidence": 0.70,
```

INSTRUCTION PYTHON:
"confidence": 0.70,
Rôle: Exécute une opération ou logique du programme.

```
[55] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[56] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[57] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[58] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[59] # 4■■ FileZilla Server Path Traversal
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "4■■ FileZilla Server Path Traversal"

Utilité: Documenter le code et expliquer les décisions de conception.

[60] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[61] `if "filezilla" in service and any(`

INSTRUCTION PYTHON:
if "filezilla" in service and any(
Rôle: Exécute une opération ou logique du programme.

[62] `cve.startswith("CVE-2019") or cve.startswith("CVE-2020") for cve in cves`

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "CVE-2019"
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[63] `):`

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

[64] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[65] `"attack": "FileZilla Server path traversal / directory traversal",`

INSTRUCTION PYTHON:
"attack": "FileZilla Server path traversal / directory traversal",
Rôle: Exécute une opération ou logique du programme.

[66] `"confidence": 0.75,`

INSTRUCTION PYTHON:
"confidence": 0.75,
Rôle: Exécute une opération ou logique du programme.

[67] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[68] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[69] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[70] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[71] `# 5■■ FTP Anonymous Access / Brute Force`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "5■■ FTP Anonymous Access / Brute Force"
Utilité: Documenter le code et expliquer les décisions de conception.

[72] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[73] `if any(cve.startswith("CVE-") for cve in cves) and len(cves) > 0:`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[74] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[75] `"attack": "FTP brute force / anonymous access exploitation",`

INSTRUCTION PYTHON:
"attack": "FTP brute force / anonymous access exploitation",
Rôle: Exécute une opération ou logique du programme.

[76] `"confidence": 0.50,`

INSTRUCTION PYTHON:
"confidence": 0.50,
Rôle: Exécute une opération ou logique du programme.

[77] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[78] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[79] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[80] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/attack_chains/mysql.py

```
[1] def evaluate_mysql(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_mysql' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "MySQL",
```

INSTRUCTION PYTHON:
"service": "MySQL",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "5.7.34",
```

INSTRUCTION PYTHON:
"version": "5.7.34",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[12] `version = facts.get("version") or ""`

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[13] `cves = set(facts.get("validated_cves", []))`

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[14] `headers = " ".join(facts.get("headers", [])).lower()`

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[15] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[16] `if "mysql" not in service:`

INSTRUCTION PYTHON:
if "mysql" not in service:
Rôle: Exécute une opération ou logique du programme.

[17] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

[18] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[19] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[20] `# 1■■ MySQL UDF RCE / Privilege Escalation`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ MySQL UDF RCE / Privilege Escalation"
Utilité: Documenter le code et expliquer les décisions de conception.

[21] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[22] `if any(`

INSTRUCTION PYTHON:
if any(
Rôle: Exécute une opération ou logique du programme.

[23] `cve in cves for cve in {`

INSTRUCTION PYTHON:
cve in cves for cve in {
Rôle: Exécute une opération ou logique du programme.

```
[24] "CVE-2016-6663",
```

INSTRUCTION PYTHON:
"CVE-2016-6663",
Rôle: Exécute une opération ou logique du programme.

```
[25] "CVE-2016-6662",
```

INSTRUCTION PYTHON:
"CVE-2016-6662",
Rôle: Exécute une opération ou logique du programme.

```
[26] "CVE-2016-6664"
```

INSTRUCTION PYTHON:
"CVE-2016-6664"
Rôle: Exécute une opération ou logique du programme.

```
[27] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[28] ):
```

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

```
[29] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[30] "attack": "MySQL UDF privilege escalation → RCE",
```

INSTRUCTION PYTHON:
"attack": "MySQL UDF privilege escalation → RCE",
Rôle: Exécute une opération ou logique du programme.

```
[31] "confidence": 0.90,
```

INSTRUCTION PYTHON:
"confidence": 0.90,
Rôle: Exécute une opération ou logique du programme.

```
[32] "exploit": "exploit/mysql/mysql_yassl_hello"
```

INSTRUCTION PYTHON:
"exploit": "exploit/mysql/mysql_yassl_hello"
Rôle: Exécute une opération ou logique du programme.

```
[33] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[34] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[35] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[36] `# 2■■ MySQL Remote Code Execution (CVE-2020-14882 style)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ MySQL Remote Code Execution (CVE-2020-14882 style)"
Utilité: Documenter le code et expliquer les décisions de conception.

[37] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[38] `if version.startswith("5.") and any(`

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "5."
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[39] `cve.startswith("CVE-2020") or cve.startswith("CVE-2021") for cve in cves`

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "CVE-2020"
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[40] `):`

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

[41] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[42] `"attack": "MySQL remote code execution (version-specific)",`

APPEL DE FONCTION:
Exécute la fonction 'execution' avec les paramètres: version-specific
Flux d'exécution: Saute à la définition de execution, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[43] `"confidence": 0.80,`

INSTRUCTION PYTHON:
"confidence": 0.80,
Rôle: Exécute une opération ou logique du programme.

[44] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[45] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[46] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[47] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[48] `# 3■■ MySQL SQL Injection / Auth Bypass`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ MySQL SQL Injection / Auth Bypass"
Utilité: Documenter le code et expliquer les décisions de conception.

[49] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[50] `if any(`

INSTRUCTION PYTHON:
if any(
Rôle: Exécute une opération ou logique du programme.

[51] `cve in cves for cve in {`

INSTRUCTION PYTHON:
cve in cves for cve in {
Rôle: Exécute une opération ou logique du programme.

[52] `"CVE-2012-2122",`

INSTRUCTION PYTHON:
"CVE-2012-2122",
Rôle: Exécute une opération ou logique du programme.

[53] `"CVE-2016-5734"`

INSTRUCTION PYTHON:
"CVE-2016-5734"
Rôle: Exécute une opération ou logique du programme.

[54] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[55] `):`

INSTRUCTION PYTHON:
):
Rôle: Exécute une opération ou logique du programme.

[56] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[57] `"attack": "MySQL authentication bypass / SQL injection",`

INSTRUCTION PYTHON:
"attack": "MySQL authentication bypass / SQL injection",
Rôle: Exécute une opération ou logique du programme.

[58] `"confidence": 0.85,`

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

[59] `"exploit": "exploit/mysql/mysql_authbypass_hashdump"`

INSTRUCTION PYTHON:
"exploit": "exploit/mysql/mysql_authbypass_hashdump"
Rôle: Exécute une opération ou logique du programme.

[60] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[61] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[62] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[63] # 4■■ MySQL Version-based exploits
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "4■■ MySQL Version-based exploits"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[64] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[65] if version.startswith("8.") and any(cve.startswith("CVE-2022") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "8."
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[66] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[67] "attack": "MySQL 8.x remote code execution / privilege escalation",
```

INSTRUCTION PYTHON:
"attack": "MySQL 8.x remote code execution / privilege escalation",
Rôle: Exécute une opération ou logique du programme.

```
[68] "confidence": 0.75,
```

INSTRUCTION PYTHON:
"confidence": 0.75,
Rôle: Exécute une opération ou logique du programme.

```
[69] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[70] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[71] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[72] if version.startswith("5.7.") and any(cve.startswith("CVE-") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "5.7."

Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[73] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[74] `"attack": "MySQL 5.7.x vulnerabilities (RCE candidates)",`

APPEL DE FONCTION:
Exécute la fonction 'vulnerabilities' avec les paramètres: RCE candidates
Flux d'exécution: Saute à la définition de vulnerabilities, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[75] `"confidence": 0.70,`

INSTRUCTION PYTHON:
"confidence": 0.70,
Rôle: Exécute une opération ou logique du programme.

[76] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[77] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[78] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[79] `if version.startswith("5.6.") or version.startswith("5.5."):`

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: "5.6."
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[80] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[81] `"attack": "MySQL legacy version (potential vulnerabilities)",`

APPEL DE FONCTION:
Exécute la fonction 'version' avec les paramètres: potential vulnerabilities
Flux d'exécution: Saute à la définition de version, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[82] `"confidence": 0.60,`

INSTRUCTION PYTHON:
"confidence": 0.60,
Rôle: Exécute une opération ou logique du programme.

[83] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[84] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[85] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[86] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[87] # 5■■ MySQL Brute Force / Weak Credentials
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "5■■ MySQL Brute Force / Weak Credentials"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[88] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[89] if any(cve.startswith("CVE-") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[90] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[91] "attack": "MySQL brute force / weak authentication",
```

INSTRUCTION PYTHON:
"attack": "MySQL brute force / weak authentication",
Rôle: Exécute une opération ou logique du programme.

```
[92] "confidence": 0.50,
```

INSTRUCTION PYTHON:
"confidence": 0.50,
Rôle: Exécute une opération ou logique du programme.

```
[93] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[94] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[95] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[96] return None
```

INSTRUCTION PYTHON:
return None

Rôle: Exécute une opération ou logique du programme.

■ exploit/attack_chains/rdp.py

```
[1] def evaluate_rdp(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_rdp' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "RDP" ou "Remote Desktop",
```

INSTRUCTION PYTHON:
"service": "RDP" ou "Remote Desktop",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "10.0.17763",
```

INSTRUCTION PYTHON:
"version": "10.0.17763",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[12] `version = facts.get("version") or ""`

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[13] `cves = set(facts.get("validated_cves", []))`

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[14] `headers = " ".join(facts.get("headers", [])).lower()`

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[15] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[16] `# Vérifier si c'est un service RDP`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Vérifier si c'est un service RDP"
Utilité: Documenter le code et expliquer les décisions de conception.

[17] `rdp_services = ["rdp", "remote desktop", "terminal services"]`

ASSIGNATION (Affectation):
Stocke la valeur ["rdp", "remote desktop", "terminal services"] dans la variable rdp_services
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[18] `if not any(rdp_svc in service for rdp_svc in rdp_services):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: rdp_svc in service for rdp_svc in rdp_services
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[19] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

[20] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[21] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[22] `# 1■■ BlueKeep (CVE-2019-0708) - RDP RCE`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ BlueKeep (CVE-2019-0708) - RDP RCE"

Utilité: Documenter le code et expliquer les décisions de conception.

```
[23] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[24] if "CVE-2019-0708" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2019-0708" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[25] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[26] "attack": "BlueKeep (CVE-2019-0708) RDP RCE",
```

APPEL DE FONCTION:
Exécute la fonction 'BlueKeep' avec les paramètres: CVE-2019-0708
Flux d'exécution: Saute à la définition de BlueKeep, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[27] "confidence": 0.95,
```

INSTRUCTION PYTHON:
"confidence": 0.95,
Rôle: Exécute une opération ou logique du programme.

```
[28] "exploit": "exploit/rdp/bluekeep_cve_2019_0708"
```

INSTRUCTION PYTHON:
"exploit": "exploit/rdp/bluekeep_cve_2019_0708"
Rôle: Exécute une opération ou logique du programme.

```
[29] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[30] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[31] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[32] # 2■■ DejaBlue (CVE-2019-1181, CVE-2019-1182)
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ DejaBlue (CVE-2019-1181, CVE-2019-1182)"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[33] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[34] if "CVE-2019-1181" in cves or "CVE-2019-1182" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2019-1181" in cves or "CVE-2019-1182" in cves:
Rôle: Exécute une opération ou logique du programme.

[35] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[36] `"attack": "DejaBlue RDP RCE vulnerabilities",`

INSTRUCTION PYTHON:
"attack": "DejaBlue RDP RCE vulnerabilities",
Rôle: Exécute une opération ou logique du programme.

[37] `"confidence": 0.90,`

INSTRUCTION PYTHON:
"confidence": 0.90,
Rôle: Exécute une opération ou logique du programme.

[38] `"exploit": "exploit/rdp/dejablue_rdp_rce"`

INSTRUCTION PYTHON:
"exploit": "exploit/rdp/dejablue_rdp_rce"
Rôle: Exécute une opération ou logique du programme.

[39] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[40] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[41] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[42] `# 3■■ PrintNightmare via RDP (CVE-2021-34527)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ PrintNightmare via RDP (CVE-2021-34527)"
Utilité: Documenter le code et expliquer les décisions de conception.

[43] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[44] `if "CVE-2021-34527" in cves:`

INSTRUCTION PYTHON:
if "CVE-2021-34527" in cves:
Rôle: Exécute une opération ou logique du programme.

[45] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[46] `"attack": "PrintNightmare (CVE-2021-34527) via RDP",`

APPEL DE FONCTION:
Exécute la fonction 'PrintNightmare' avec les paramètres: CVE-2021-34527
Flux d'exécution: Saute à la définition de PrintNightmare, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[47] `"confidence": 0.85,`

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

`[48] "exploit": "exploit/rdp/printnightmare_cve_2021_34527"`

INSTRUCTION PYTHON:
"exploit": "exploit/rdp/printnightmare_cve_2021_34527"
Rôle: Exécute une opération ou logique du programme.

`[49] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

`[50] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[51] # ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

`[52] # 4■■ OpenSSL RDP (CVE-2021-38647)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "4■■ OpenSSL RDP (CVE-2021-38647)"
Utilité: Documenter le code et expliquer les décisions de conception.

`[53] # ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

`[54] if "CVE-2021-38647" in cves:`

INSTRUCTION PYTHON:
if "CVE-2021-38647" in cves:
Rôle: Exécute une opération ou logique du programme.

`[55] return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

`[56] "attack": "OpenSSL RDP vulnerability (CVE-2021-38647)",`

APPEL DE FONCTION:
Exécute la fonction 'vulnerability' avec les paramètres: CVE-2021-38647
Flux d'exécution: Saute à la définition de vulnerability, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[57] "confidence": 0.80,`

INSTRUCTION PYTHON:
"confidence": 0.80,
Rôle: Exécute une opération ou logique du programme.

`[58] "exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

`[59] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.

Signification: Fin de la structure commencée par une accolade ouvrante.

[60] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[61] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[62] `# 5■■ RDP Man-in-the-Middle (CVE-2020-0610)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "5■■ RDP Man-in-the-Middle (CVE-2020-0610)"
Utilité: Documenter le code et expliquer les décisions de conception.

[63] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[64] `if "CVE-2020-0610" in cves:`

INSTRUCTION PYTHON:
if "CVE-2020-0610" in cves:
Rôle: Exécute une opération ou logique du programme.

[65] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[66] `"attack": "RDP Man-in-the-Middle vulnerability",`

INSTRUCTION PYTHON:
"attack": "RDP Man-in-the-Middle vulnerability",
Rôle: Exécute une opération ou logique du programme.

[67] `"confidence": 0.75,`

INSTRUCTION PYTHON:
"confidence": 0.75,
Rôle: Exécute une opération ou logique du programme.

[68] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[69] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[70] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[71] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[72] `# 6■■ RDP Brute Force / Credential Stuffing`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "6■■ RDP Brute Force / Credential Stuffing"
Utilité: Documenter le code et expliquer les décisions de conception.

[73] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[74] `if any(cve.startswith("CVE-") for cve in cves):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[75] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[76] `"attack": "RDP brute force / credential stuffing",`

INSTRUCTION PYTHON:
"attack": "RDP brute force / credential stuffing",
Rôle: Exécute une opération ou logique du programme.

[77] `"confidence": 0.60,`

INSTRUCTION PYTHON:
"confidence": 0.60,
Rôle: Exécute une opération ou logique du programme.

[78] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[79] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[80] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[81] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/attack_chains/smb.py

```
[1] def evaluate_smb(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_smb' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "SMB",
```

INSTRUCTION PYTHON:
"service": "SMB",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "3.1.1",
```

INSTRUCTION PYTHON:
"version": "3.1.1",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[12] `version = facts.get("version") or ""`

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[13] `cves = set(facts.get("validated_cves", []))`

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[14] `headers = " ".join(facts.get("headers", [])).lower()`

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[15] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[16] `# Vérifier si c'est un service SMB`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Vérifier si c'est un service SMB"
Utilité: Documenter le code et expliquer les décisions de conception.

[17] `if "smb" not in service:`

INSTRUCTION PYTHON:
if "smb" not in service:
Rôle: Exécute une opération ou logique du programme.

[18] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

[19] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[20] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[21] `# 1■■ EternalBlue (MS17-010) - CVE-2017-0143 à CVE-2017-0148`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ EternalBlue (MS17-010) - CVE-2017-0143 à CVE-2017-0148"
Utilité: Documenter le code et expliquer les décisions de conception.

[22] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[23] `eternalblue_cves = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable eternalblue_cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[24] "CVE-2017-0143",
```

INSTRUCTION PYTHON:
"CVE-2017-0143",
Rôle: Exécute une opération ou logique du programme.

```
[25] "CVE-2017-0144",
```

INSTRUCTION PYTHON:
"CVE-2017-0144",
Rôle: Exécute une opération ou logique du programme.

```
[26] "CVE-2017-0145",
```

INSTRUCTION PYTHON:
"CVE-2017-0145",
Rôle: Exécute une opération ou logique du programme.

```
[27] "CVE-2017-0146",
```

INSTRUCTION PYTHON:
"CVE-2017-0146",
Rôle: Exécute une opération ou logique du programme.

```
[28] "CVE-2017-0147",
```

INSTRUCTION PYTHON:
"CVE-2017-0147",
Rôle: Exécute une opération ou logique du programme.

```
[29] "CVE-2017-0148"
```

INSTRUCTION PYTHON:
"CVE-2017-0148"
Rôle: Exécute une opération ou logique du programme.

```
[30] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[31] if any(cve in cves for cve in eternalblue_cves):
```

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve in cves for cve in eternalblue_cves
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[32] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[33] "attack": "SMB EternalBlue (MS17-010) RCE",
```

APPEL DE FONCTION:
Exécute la fonction 'EternalBlue' avec les paramètres: MS17-010
Flux d'exécution: Saute à la définition de EternalBlue, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[34] "confidence": 0.95,
```

INSTRUCTION PYTHON:
"confidence": 0.95,
Rôle: Exécute une opération ou logique du programme.

```
[35] "exploit": "exploit/smb/eternalblue_ms17_010"
```

INSTRUCTION PYTHON:
"exploit": "exploit/smb/eternalblue_ms17_010"
Rôle: Exécute une opération ou logique du programme.

```
[36] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[37] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[38] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[39] # 2■■ SMBGhost (CVE-2020-0796) - SMBv3 Compression
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ SMBGhost (CVE-2020-0796) - SMBv3 Compression"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[40] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[41] if "CVE-2020-0796" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2020-0796" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[42] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[43] "attack": "SMBGhost (SMBv3 compression) RCE",
```

APPEL DE FONCTION:
Exécute la fonction 'SMBGhost' avec les paramètres: SMBv3 compression
Flux d'exécution: Saute à la définition de SMBGhost, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[44] "confidence": 0.90,
```

INSTRUCTION PYTHON:
"confidence": 0.90,
Rôle: Exécute une opération ou logique du programme.

```
[45] "exploit": "exploit/smb/smbghost_cve_2020_0796"
```

INSTRUCTION PYTHON:
"exploit": "exploit/smb/smbghost_cve_2020_0796"
Rôle: Exécute une opération ou logique du programme.

```
[46] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[47] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[48] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[49] # 3■■ Samba Remote Code Execution (CVE-2021-44142)
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ Samba Remote Code Execution (CVE-2021-44142)"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[50] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[51] if "CVE-2021-44142" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2021-44142" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[52] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[53] "attack": "Samba VFS module RCE (CVE-2021-44142)",
```

APPEL DE FONCTION:
Exécute la fonction 'RCE' avec les paramètres: CVE-2021-44142
Flux d'exécution: Saute à la définition de RCE, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[54] "confidence": 0.85,
```

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

```
[55] "exploit": "exploit/smb/samba_vfs_rce"
```

INSTRUCTION PYTHON:
"exploit": "exploit/smb/samba_vfs_rce"
Rôle: Exécute une opération ou logique du programme.

```
[56] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[57] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[58] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[59] # 4■■ SMB Relay Attacks
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "4■■ SMB Relay Attacks"

Utilité: Documenter le code et expliquer les décisions de conception.

```
[60] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[61] if any(cve.startswith("CVE-2019") or cve.startswith("CVE-2020") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-2019"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[62] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[63] "attack": "SMB relay / NTLM relay attack",
```

INSTRUCTION PYTHON:
"attack": "SMB relay / NTLM relay attack",
Rôle: Exécute une opération ou logique du programme.

```
[64] "confidence": 0.75,
```

INSTRUCTION PYTHON:
"confidence": 0.75,
Rôle: Exécute une opération ou logique du programme.

```
[65] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[66] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[67] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[68] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[69] # 5■■ SMB Null Session / Anonymous Access
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "5■■ SMB Null Session / Anonymous Access"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[70] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[71] if any(cve.startswith("CVE-") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.

Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[72] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[73] "attack": "SMB null session / anonymous access exploitation",
```

INSTRUCTION PYTHON:
"attack": "SMB null session / anonymous access exploitation",
Rôle: Exécute une opération ou logique du programme.

```
[74] "confidence": 0.60,
```

INSTRUCTION PYTHON:
"confidence": 0.60,
Rôle: Exécute une opération ou logique du programme.

```
[75] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[76] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[77] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[78] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/attack_chains/ssh.py

```
[1] def evaluate_ssh(facts):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'evaluate_ssh' réutilisable.
Paramètres: facts
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[4] "service": "OpenSSH",
```

INSTRUCTION PYTHON:
"service": "OpenSSH",
Rôle: Exécute une opération ou logique du programme.

```
[5] "version": "8.2p1",
```

INSTRUCTION PYTHON:
"version": "8.2p1",
Rôle: Exécute une opération ou logique du programme.

```
[6] "validated_cves": [...],
```

INSTRUCTION PYTHON:
"validated_cves": [...],
Rôle: Exécute une opération ou logique du programme.

```
[7] "headers": [...],
```

INSTRUCTION PYTHON:
"headers": [...],
Rôle: Exécute une opération ou logique du programme.

```
[8] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[9] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] service = (facts.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (facts.get("service") or "").lower() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[12] `version = facts.get("version") or ""`

ASSIGNATION (Affectation):
Stocke la valeur facts.get("version") or "" dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[13] `cves = set(facts.get("validated_cves", []))`

ASSIGNATION (Affectation):
Stocke la valeur set(facts.get("validated_cves", [])) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[14] `headers = " ".join(facts.get("headers", [])).lower()`

ASSIGNATION (Affectation):
Stocke la valeur " ".join(facts.get("headers", [])).lower() dans la variable headers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[15] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[16] `if "ssh" not in service and "openssh" not in service:`

INSTRUCTION PYTHON:
if "ssh" not in service and "openssh" not in service:
Rôle: Exécute une opération ou logique du programme.

[17] `return None`

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

[18] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[19] `# Extraire la version OpenSSH (format: X.YpN ou X.Y.Z)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Extraire la version OpenSSH (format: X.YpN ou X.Y.Z)"
Utilité: Documenter le code et expliquer les décisions de conception.

[20] `openssh_version = None`

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable openssh_version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[21] `if version:`

INSTRUCTION PYTHON:
if version:
Rôle: Exécute une opération ou logique du programme.

[22] `openssh_version = version`

ASSIGNATION (Affectation):
Stocke la valeur version dans la variable openssh_version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[23] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[24] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[25] # 1■■ OpenSSH Username Enumeration (CVE-2018-15473)
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "1■■ OpenSSH Username Enumeration (CVE-2018-15473)"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[26] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[27] if "CVE-2018-15473" in cves:
```

INSTRUCTION PYTHON:
if "CVE-2018-15473" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[28] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[29] "attack": "OpenSSH username enumeration",
```

INSTRUCTION PYTHON:
"attack": "OpenSSH username enumeration",
Rôle: Exécute une opération ou logique du programme.

```
[30] "confidence": 0.85,
```

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

```
[31] "exploit": "exploit/ssh/ssh_enumusers"
```

INSTRUCTION PYTHON:
"exploit": "exploit/ssh/ssh_enumusers"
Rôle: Exécute une opération ou logique du programme.

```
[32] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[33] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[34] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[35] `# 2■■ OpenSSH RCE (CVE-2020-15778)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "2■■ OpenSSH RCE (CVE-2020-15778)"
Utilité: Documenter le code et expliquer les décisions de conception.

[36] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[37] `if "CVE-2020-15778" in cves:`

INSTRUCTION PYTHON:
if "CVE-2020-15778" in cves:
Rôle: Exécute une opération ou logique du programme.

[38] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[39] `"attack": "OpenSSH command injection → RCE",`

INSTRUCTION PYTHON:
"attack": "OpenSSH command injection → RCE",
Rôle: Exécute une opération ou logique du programme.

[40] `"confidence": 0.90,`

INSTRUCTION PYTHON:
"confidence": 0.90,
Rôle: Exécute une opération ou logique du programme.

[41] `"exploit": "exploit/ssh/openssh_command_injection"`

INSTRUCTION PYTHON:
"exploit": "exploit/ssh/openssh_command_injection"
Rôle: Exécute une opération ou logique du programme.

[42] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[43] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[44] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[45] `# 3■■ OpenSSH X11 Forwarding RCE (CVE-2019-6111)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "3■■ OpenSSH X11 Forwarding RCE (CVE-2019-6111)"
Utilité: Documenter le code et expliquer les décisions de conception.

[46] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[47] `if "CVE-2019-6111" in cves or "CVE-2019-6109" in cves:`

INSTRUCTION PYTHON:
if "CVE-2019-6111" in cves or "CVE-2019-6109" in cves:
Rôle: Exécute une opération ou logique du programme.

```
[48] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[49] "attack": "OpenSSH X11 forwarding / file disclosure",
```

INSTRUCTION PYTHON:
"attack": "OpenSSH X11 forwarding / file disclosure",
Rôle: Exécute une opération ou logique du programme.

```
[50] "confidence": 0.80,
```

INSTRUCTION PYTHON:
"confidence": 0.80,
Rôle: Exécute une opération ou logique du programme.

```
[51] "exploit": "exploit/ssh/openssh_x11_forward"
```

INSTRUCTION PYTHON:
"exploit": "exploit/ssh/openssh_x11_forward"
Rôle: Exécute une opération ou logique du programme.

```
[52] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[53] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[54] # ================================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[55] # 4■■ OpenSSH 7.4 and below vulnerabilities
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "4■■ OpenSSH 7.4 and below vulnerabilities"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[56] # ================================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[57] if openssh_version:
```

INSTRUCTION PYTHON:
if openssh_version:
Rôle: Exécute une opération ou logique du programme.

```
[58] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[59] # Extraire le numéro de version principal
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Extraire le numéro de version principal"
Utilité: Documenter le code et expliquer les décisions de conception.

[60] `if "p" in openssh_version:`

INSTRUCTION PYTHON:
if "p" in openssh_version:
Rôle: Exécute une opération ou logique du programme.

[61] `major_minor = openssh_version.split("p")[0]`

ASSIGNATION (Affectation):
Stocke la valeur openssh_version.split("p")[0] dans la variable major_minor
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[62] `else:`

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

[63] `parts = openssh_version.split(".")`

ASSIGNATION (Affectation):
Stocke la valeur openssh_version.split(".") dans la variable parts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[64] `major_minor = f"{parts[0]}.{parts[1]}" if len(parts) >= 2 else openssh_version`

APPEL DE FONCTION:
Exécute la fonction 'len' avec les paramètres: parts
Flux d'exécution: Saute à la définition de len, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[65] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[66] `version_float = float(major_minor)`

ASSIGNATION (Affectation):
Stocke la valeur float(major_minor) dans la variable version_float
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[67] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[68] `if version_float <= 7.4 and any(cve.startswith("CVE-2017") for cve in cves):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-2017"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[69] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[70] `"attack": "OpenSSH 7.4 and below vulnerabilities (RCE candidates)",`

APPEL DE FONCTION:
Exécute la fonction 'vulnerabilities' avec les paramètres: RCE candidates
Flux d'exécution: Saute à la définition de vulnerabilities, exécute son code, puis revient ici.

Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[71] "confidence": 0.75,
```

INSTRUCTION PYTHON:
"confidence": 0.75,
Rôle: Exécute une opération ou logique du programme.

```
[72] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[73] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[74] except (ValueError, AttributeError):
```

APPEL DE FONCTION:
Exécute la fonction 'except' avec les paramètres: ValueError, AttributeError
Flux d'exécution: Saute à la définition de except, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[75] pass
```

INSTRUCTION PYTHON:
pass
Rôle: Exécute une opération ou logique du programme.

```
[76] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[77] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[78] # 5■■ OpenSSH Legacy Versions (pre-7.0)
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "5■■ OpenSSH Legacy Versions (pre-7.0)"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[79] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[80] if openssh_version:
```

INSTRUCTION PYTHON:
if openssh_version:
Rôle: Exécute une opération ou logique du programme.

```
[81] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[82] if "p" in openssh_version:
```

INSTRUCTION PYTHON:
if "p" in openssh_version:
Rôle: Exécute une opération ou logique du programme.

`[83] major_minor = openssh_version.split("p")[0]`

ASSIGNATION (Affectation):
Stocke la valeur openssh_version.split("p")[0] dans la variable major_minor
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[84] else:`

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

`[85] parts = openssh_version.split(".")`

ASSIGNATION (Affectation):
Stocke la valeur openssh_version.split(".") dans la variable parts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[86] major_minor = f"{parts[0]}.{parts[1]}" if len(parts) >= 2 else openssh_version`

APPEL DE FONCTION:
Exécute la fonction 'len' avec les paramètres: parts
Flux d'exécution: Saute à la définition de len, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[87] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[88] version_float = float(major_minor)`

ASSIGNATION (Affectation):
Stocke la valeur float(major_minor) dans la variable version_float
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[89] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[90] if version_float < 7.0:`

INSTRUCTION PYTHON:
if version_float < 7.0:
Rôle: Exécute une opération ou logique du programme.

`[91] return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

`[92] "attack": "OpenSSH legacy version (known vulnerabilities)",`

APPEL DE FONCTION:
Exécute la fonction 'version' avec les paramètres: known vulnerabilities
Flux d'exécution: Saute à la définition de version, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[93] "confidence": 0.85,`

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

[94] `"exploit": None`

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

[95] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[96] `except (ValueError, AttributeError):`

APPEL DE FONCTION:
Exécute la fonction 'except' avec les paramètres: ValueError, AttributeError
Flux d'exécution: Saute à la définition de except, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[97] `pass`

INSTRUCTION PYTHON:
pass
Rôle: Exécute une opération ou logique du programme.

[98] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[99] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[100] `# 6■■ OpenSSH Brute Force / Weak Keys`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "6■■ OpenSSH Brute Force / Weak Keys"
Utilité: Documenter le code et expliquer les décisions de conception.

[101] `# ==============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

[102] `if any(cve.startswith("CVE-") for cve in cves):`

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[103] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[104] `"attack": "OpenSSH brute force / weak key exploitation",`

INSTRUCTION PYTHON:
"attack": "OpenSSH brute force / weak key exploitation",
Rôle: Exécute une opération ou logique du programme.

[105] `"confidence": 0.60,`

INSTRUCTION PYTHON:
"confidence": 0.60,
Rôle: Exécute une opération ou logique du programme.

```
[106] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[107] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[108] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[109] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[110] # 7■■ Generic SSH vulnerabilities
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "7■■ Generic SSH vulnerabilities"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[111] # ==============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=============================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[112] if any(cve.startswith("CVE-202") for cve in cves):
```

APPEL DE FONCTION:
Exécute la fonction 'any' avec les paramètres: cve.startswith("CVE-202"
Flux d'exécution: Saute à la définition de any, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[113] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[114] "attack": "OpenSSH recent vulnerabilities (RCE candidates)",
```

APPEL DE FONCTION:
Exécute la fonction 'vulnerabilities' avec les paramètres: RCE candidates
Flux d'exécution: Saute à la définition de vulnerabilities, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[115] "confidence": 0.70,
```

INSTRUCTION PYTHON:
"confidence": 0.70,
Rôle: Exécute une opération ou logique du programme.

```
[116] "exploit": None
```

INSTRUCTION PYTHON:
"exploit": None
Rôle: Exécute une opération ou logique du programme.

```
[117] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[118] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[119] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/bruteforce.py

```
[1] import sys, json, re
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): sys, json, re
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[2] from scripts.db.mysql_conn import get_connection
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT get_connection depuis le module scripts.db.mysql_conn
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[3] from scripts.exploit.engine.security_engine import SecurityEngine
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT SecurityEngine depuis le module scripts.exploit.engine.security_engine
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[4] from scripts.exploit.engine.exploit_engine import ExploitEngine
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT ExploitEngine depuis le module scripts.exploit.engine.exploit_engine
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[5] from scripts.exploit.attack_chains.chain_engine import evaluate_chain
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT evaluate_chain depuis le module scripts.exploit.attack_chains.chain_engine
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[6] from scripts.exploit.engine.exploit_runner import ExploitRunner
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT ExploitRunner depuis le module scripts.exploit.engine.exploit_runner
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[7] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[8] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[9] def parse_cves(raw):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'parse_cves' réutilisable.
Paramètres: raw
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[10] out = []
```

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable out
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[11] if not raw:
```

INSTRUCTION PYTHON:
if not raw:
Rôle: Exécute une opération ou logique du programme.

```
[12] return out
```

INSTRUCTION PYTHON:
return out
Rôle: Exécute une opération ou logique du programme.

```
[13] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[14] for p in raw.split(","):
```

APPEL DE FONCTION:
Exécute la fonction 'split' avec les paramètres: ","
Flux d'exécution: Saute à la définition de split, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[15] m = re.match(r'(CVE-\d{4}-\d+)', p.strip())
```

ASSIGNATION (Affectation):
Stocke la valeur re.match(r'(CVE-\d{4}-\d+)', p.strip()) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[16] if m:
```

INSTRUCTION PYTHON:
if m:
Rôle: Exécute une opération ou logique du programme.

```
[17] out.append(m.group(1))
```

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: m.group(1
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[18] return out
```

INSTRUCTION PYTHON:
return out
Rôle: Exécute une opération ou logique du programme.

```
[19] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[20] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[21] def main():
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'main' réutilisable.
Paramètres: aucun
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[22] user_id, target = sys.argv[1], sys.argv[2]
```

ASSIGNATION (Affectation):
Stocke la valeur sys.argv[1], sys.argv[2] dans la variable user_id, target
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[23] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[24] conn = get_connection()
```

ASSIGNATION (Affectation):
Stocke la valeur get_connection() dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[25] cur = conn.cursor(dictionary=True)
```

ASSIGNATION (Affectation):
Stocke la valeur conn.cursor(dictionary=True) dans la variable cur
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[26] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[27] # ================ RÉCUPÉRATION SCAN ================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ RÉCUPÉRATION SCAN ================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[28] cur.execute("""
```

INSTRUCTION PYTHON:
cur.execute("""
Rôle: Exécute une opération ou logique du programme.

```
[29] SELECT scanner.script_vuln, scanner.service, scanner.version
```

INSTRUCTION PYTHON:
SELECT scanner.script_vuln, scanner.service, scanner.version
Rôle: Exécute une opération ou logique du programme.

```
[30] FROM scanner
```

INSTRUCTION PYTHON:
FROM scanner
Rôle: Exécute une opération ou logique du programme.

```
[31] JOIN ping ON scanner.ping_id = ping.id
```

ASSIGNATION (Affectation):
Stocke la valeur ping.id dans la variable JOIN ping ON scanner.ping_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du

programme.

```
[32] WHERE ping.user_id=%s AND ping.ip_address=%s
```

ASSIGNATION (Affectation):
Stocke la valeur %s AND ping.ip_address=%s dans la variable WHERE ping.user_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[33] """, (user_id, target))
```

INSTRUCTION PYTHON:
""", (user_id, target))
Rôle: Exécute une opération ou logique du programme.

```
[34] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[35] rows = cur.fetchall()
```

ASSIGNATION (Affectation):
Stocke la valeur cur.fetchall() dans la variable rows
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[36] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[37] print(f"\n==== Exploit de la cible {target} ======\n")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"\n==== Exploit de la cible {target} ======\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[38] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[39] if not rows:
```

INSTRUCTION PYTHON:
if not rows:
Rôle: Exécute une opération ou logique du programme.

```
[40] print("■ Aucun service exploitable")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "■ Aucun service exploitable"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[41] return
```

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

```
[42] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[43] `# ================ NORMALISATION DES SERVICES ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ NORMALISATION DES SERVICES ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[44] `services = {}`

ASSIGNATION (Affectation):
Stocke la valeur {} dans la variable services
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[45] `proofs = []`

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable proofs
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[46] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[47] `for r in rows:`

INSTRUCTION PYTHON:
for r in rows:
Rôle: Exécute une opération ou logique du programme.

[48] `service = (r["service"] or "").strip()`

ASSIGNATION (Affectation):
Stocke la valeur (r["service"] or "").strip() dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[49] `version = (r["version"] or "").strip()`

ASSIGNATION (Affectation):
Stocke la valeur (r["version"] or "").strip() dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[50] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[51] `if not service:`

INSTRUCTION PYTHON:
if not service:
Rôle: Exécute une opération ou logique du programme.

[52] `continue`

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

[53] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

[54] `key = f"{service.lower()}:{version}"`

ASSIGNATION (Affectation):
Stocke la valeur f"{service.lower()}:{version}" dans la variable key
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[55] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[56] `if key not in services:`

INSTRUCTION PYTHON:
if key not in services:
Rôle: Exécute une opération ou logique du programme.

[57] `services[key] = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable services[key]
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[58] `"service": service,`

INSTRUCTION PYTHON:
"service": service,
Rôle: Exécute une opération ou logique du programme.

[59] `"version": version,`

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

[60] `"validated_cves": set()`

APPEL DE FONCTION:
Exécute la fonction 'set' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de set, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[61] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[62] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[63] `# ================ PHASE 1 : VALIDATION DES CVE ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ PHASE 1 : VALIDATION DES CVE ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[64] `sec = SecurityEngine()`

ASSIGNATION (Affectation):
Stocke la valeur SecurityEngine() dans la variable sec
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[65] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[66] for key, svc in services.items():
```

APPEL DE FONCTION:
Exécute la fonction 'items' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de items, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[67] service = svc["service"]
```

ASSIGNATION (Affectation):
Stocke la valeur svc["service"] dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[68] version = svc["version"]
```

ASSIGNATION (Affectation):
Stocke la valeur svc["version"] dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[69] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[70] # Récupère toutes les CVE associées à ce service/version
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Récupère toutes les CVE associées à ce service/version"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[71] cves = set()
```

ASSIGNATION (Affectation):
Stocke la valeur set() dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[72] for r in rows:
```

INSTRUCTION PYTHON:
for r in rows:
Rôle: Exécute une opération ou logique du programme.

```
[73] if r["service"] == service and r["version"] == version:
```

INSTRUCTION PYTHON:
if r["service"] == service and r["version"] == version:
Rôle: Exécute une opération ou logique du programme.

```
[74] cves.update(parse_cves(r["script_vuln"]))
```

APPEL DE FONCTION:
Exécute la fonction 'update' avec les paramètres: parse_cves(r["script_vuln"]
Flux d'exécution: Saute à la définition de update, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[75] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[76] if not cves:
```

INSTRUCTION PYTHON:
if not cves:
Rôle: Exécute une opération ou logique du programme.

```
[77] continue
```

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

```
[78] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[79] print(f"■ Analyse {service} {version} → {len(cves)} CVE")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"■ Analyse {service} {version} → {len(cves
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[80] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[81] for cve in cves:
```

INSTRUCTION PYTHON:
for cve in cves:
Rôle: Exécute une opération ou logique du programme.

```
[82] modules = sec.search_modules(cve)
```

ASSIGNATION (Affectation):
Stocke la valeur sec.search_modules(cve) dans la variable modules
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[83] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[84] for m in modules:
```

INSTRUCTION PYTHON:
for m in modules:
Rôle: Exécute une opération ou logique du programme.

```
[85] res = sec.run_module(m, target)
```

ASSIGNATION (Affectation):
Stocke la valeur sec.run_module(m, target) dans la variable res
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[86] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

```
[87] if res.get("status") == "exploitable":
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "status"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[88] svc["validated_cves"].add(cve)
```

APPEL DE FONCTION:
Exécute la fonction 'add' avec les paramètres: cve
Flux d'exécution: Saute à la définition de add, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[89] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[90] proof = res.get("proof")
```

ASSIGNATION (Affectation):
Stocke la valeur res.get("proof") dans la variable proof
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[91] if proof:
```

INSTRUCTION PYTHON:
if proof:
Rôle: Exécute une opération ou logique du programme.

```
[92] proofs.append(proof)
```

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: proof
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[93] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[94] print(f" ■ {cve} exploitable via {m}")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f" ■ {cve} exploitable via {m}"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[95] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[96] # ================ CONSTRUCTION DES FACTS ================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ CONSTRUCTION DES FACTS ================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[97] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts

Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[98] "services": [],`

INSTRUCTION PYTHON:
"services": [],
Rôle: Exécute une opération ou logique du programme.

`[99] "proofs": list(set(proofs))`

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: set(proofs
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[100] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

`[101] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[102] for svc in services.values():`

APPEL DE FONCTION:
Exécute la fonction 'values' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de values, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[103] if svc["validated_cves"]:`

INSTRUCTION PYTHON:
if svc["validated_cves"]:
Rôle: Exécute une opération ou logique du programme.

`[104] facts["services"].append({`

INSTRUCTION PYTHON:
facts["services"].append({
Rôle: Exécute une opération ou logique du programme.

`[105] "service": svc["service"],`

INSTRUCTION PYTHON:
"service": svc["service"],
Rôle: Exécute une opération ou logique du programme.

`[106] "version": svc["version"],`

INSTRUCTION PYTHON:
"version": svc["version"],
Rôle: Exécute une opération ou logique du programme.

`[107] "validated_cves": list(svc["validated_cves"])`

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: svc["validated_cves"]
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[108] })`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[109] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[110] `if not facts["services"]:`

INSTRUCTION PYTHON:
if not facts["services"]:
Rôle: Exécute une opération ou logique du programme.

[111] `print("\n■ Aucune CVE exploitable validée")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n■ Aucune CVE exploitable validée"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[112] `return`

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

[113] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[114] `print("\n■ FACTS CONSTRUITS")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n■ FACTS CONSTRUITS"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[115] `print(json.dumps(facts, indent=2))`

ASSIGNATION (Affectation):
Stocke la valeur 2)) dans la variable print(json.dumps(facts, indent
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[116] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[117] `# ================ PHASE 2 : ATTACK CHAIN ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ PHASE 2 : ATTACK CHAIN ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[118] `chain = None`

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable chain
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[119] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[120] `for svc in facts["services"]:`

INSTRUCTION PYTHON:
for svc in facts["services"]:
Rôle: Exécute une opération ou logique du programme.

```
[121] chain = evaluate_chain({
```

ASSIGNATION (Affectation):
Stocke la valeur evaluate_chain({ dans la variable chain
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[122] "service": svc["service"],
```

INSTRUCTION PYTHON:
"service": svc["service"],
Rôle: Exécute une opération ou logique du programme.

```
[123] "version": svc["version"],
```

INSTRUCTION PYTHON:
"version": svc["version"],
Rôle: Exécute une opération ou logique du programme.

```
[124] "validated_cves": svc["validated_cves"],
```

INSTRUCTION PYTHON:
"validated_cves": svc["validated_cves"],
Rôle: Exécute une opération ou logique du programme.

```
[125] "headers": facts["proofs"]
```

INSTRUCTION PYTHON:
"headers": facts["proofs"]
Rôle: Exécute une opération ou logique du programme.

```
[126] })
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[127] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[128] if chain:
```

INSTRUCTION PYTHON:
if chain:
Rôle: Exécute une opération ou logique du programme.

```
[129] break
```

INSTRUCTION PYTHON:
break
Rôle: Exécute une opération ou logique du programme.

```
[130] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[131] print("\n--- ATTACK CHAIN ---")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n--- ATTACK CHAIN ---"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[132] print(json.dumps(chain, indent=2))
```

ASSIGNATION (Affectation):
Stocke la valeur 2)) dans la variable print(json.dumps(chain, indent
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[133] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[134] if not chain or not chain.get("exploit"):
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "exploit"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[135] print("\n■■ Aucune chaîne d’attaque exploitable")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n■■ Aucune chaîne d’attaque exploitable"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[136] return
```

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

```
[137] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[138] # ================= PHASE 3 : EXPLOIT RÉEL =================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================= PHASE 3 : EXPLOIT RÉEL ================="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[139] print("\n■ Exploit recommandé :", chain["exploit"])
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n■ Exploit recommandé :", chain["exploit"]
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[140] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[141] runner = ExploitRunner()
```

ASSIGNATION (Affectation):
Stocke la valeur ExploitRunner() dans la variable runner
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[142] result = runner.run(chain["exploit"], target)
```

ASSIGNATION (Affectation):
Stocke la valeur runner.run(chain["exploit"], target) dans la variable result
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[143] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[144] `#### Store via symfony #######`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "### Store via symfony #######"
Utilité: Documenter le code et expliquer les décisions de conception.

[145] `clean_proofs = list(dict.fromkeys(proofs))`

ASSIGNATION (Affectation):
Stocke la valeur list(dict.fromkeys(proofs)) dans la variable clean_proofs
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[146] `exploit_data = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable exploit_data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[147] `"module": chain["exploit"],`

INSTRUCTION PYTHON:
"module": chain["exploit"],
Rôle: Exécute une opération ou logique du programme.

[148] `"payload": None,`

INSTRUCTION PYTHON:
"payload": None,
Rôle: Exécute une opération ou logique du programme.

[149] `"status": result["status"],`

INSTRUCTION PYTHON:
"status": result["status"],
Rôle: Exécute une opération ou logique du programme.

[150] `"session_id": None,`

INSTRUCTION PYTHON:
"session_id": None,
Rôle: Exécute une opération ou logique du programme.

[151] `"output": json.dumps(`

INSTRUCTION PYTHON:
"output": json.dumps(
Rôle: Exécute une opération ou logique du programme.

[152] `result.get("details") or result, indent=2),`

ASSIGNATION (Affectation):
Stocke la valeur 2), dans la variable result.get("details") or result, indent
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[153] `"proof": " | ".join(clean_proofs)`

APPEL DE FONCTION:
Exécute la fonction 'join' avec les paramètres: clean_proofs
Flux d'exécution: Saute à la définition de join, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[154] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[155] `print("\n@@@EXPLOITJSON@@@")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n@@@EXPLOITJSON@@@"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[156] `print(json.dumps(exploit_data))`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: json.dumps(exploit_data
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[157] `############## Final print ###############`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "############## Final print ###############"
Utilité: Documenter le code et expliquer les décisions de conception.

[158] `#print("\n--- REAL EXPLOIT RESULT ---")`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "print("\n--- REAL EXPLOIT RESULT ---")"
Utilité: Documenter le code et expliquer les décisions de conception.

[159] `#print(json.dumps(result, indent=2))`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "print(json.dumps(result, indent=2))"
Utilité: Documenter le code et expliquer les décisions de conception.

[160] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[161] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[162] `if __name__ == "__main__":`

CONDITION IF:
Teste si l'expression est VRAIE: __name__ == "__main__"
Flux: Si VRAI → exécute le bloc qui suit. Si FAUX → passe au elif/else.
Exemple réel: 'if conn.is_connected():' → vérifier si la connexion est active avant de l'utiliser.

[163] `main()`

APPEL DE FONCTION:
Exécute la fonction 'main' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de main, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

■ exploit/engine/attack_chain_engine.py

`[1] from scripts.exploit.attack_chains.apache import evaluate_apache`

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT evaluate_apache depuis le module scripts.exploit.attack_chains.apache
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

`[2] # from scripts.exploit.attack_chains.ssh import evaluate_ssh`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "from scripts.exploit.attack_chains.ssh import evaluate_ssh"
Utilité: Documenter le code et expliquer les décisions de conception.

`[3] # from scripts.exploit.attack_chains.ftp import evaluate_ftp`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "from scripts.exploit.attack_chains.ftp import evaluate_ftp"
Utilité: Documenter le code et expliquer les décisions de conception.

`[4] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[5] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[6] class AttackChainEngine:`

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: AttackChainEngine
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

`[7] """`

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

`[8] Orchestrateur global des chaînes d’attaque.`

INSTRUCTION PYTHON:
Orchestrateur global des chaînes d’attaque.
Rôle: Exécute une opération ou logique du programme.

`[9] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[10] - Support multi-services`

INSTRUCTION PYTHON:
- Support multi-services
Rôle: Exécute une opération ou logique du programme.

`[11] - Support multi-CVE`

INSTRUCTION PYTHON:
- Support multi-CVE
Rôle: Exécute une opération ou logique du programme.

`[12] - Ne casse aucun engine existant`

INSTRUCTION PYTHON:
- Ne casse aucun engine existant
Rôle: Exécute une opération ou logique du programme.

```
[13] - Retourne UNE attaque finale (la meilleure)
```

APPEL DE FONCTION:
Exécute la fonction 'finale' avec les paramètres: la meilleure
Flux d'exécution: Saute à la définition de finale, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[14] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[15] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[16] def __init__(self):
```

INSTRUCTION PYTHON:
def __init__(self):
Rôle: Exécute une opération ou logique du programme.

```
[17] self.engines = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable self.engines
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[18] "apache": evaluate_apache,
```

INSTRUCTION PYTHON:
"apache": evaluate_apache,
Rôle: Exécute une opération ou logique du programme.

```
[19] # "ssh": evaluate_ssh,
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: ""ssh": evaluate_ssh,"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[20] # "ftp": evaluate_ftp,
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: ""ftp": evaluate_ftp,"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[21] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[22] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[23] def evaluate(self, facts):
```

INSTRUCTION PYTHON:
def evaluate(self, facts):
Rôle: Exécute une opération ou logique du programme.

```
[24] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[25] facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[26] "services": [
```

INSTRUCTION PYTHON:
"services": [
Rôle: Exécute une opération ou logique du programme.

```
[27] {
```

INSTRUCTION PYTHON:
{
Rôle: Exécute une opération ou logique du programme.

```
[28] "service": "Apache",
```

INSTRUCTION PYTHON:
"service": "Apache",
Rôle: Exécute une opération ou logique du programme.

```
[29] "version": "2.4.65",
```

INSTRUCTION PYTHON:
"version": "2.4.65",
Rôle: Exécute une opération ou logique du programme.

```
[30] "validated_cves": [...]
```

INSTRUCTION PYTHON:
"validated_cves": [...]
Rôle: Exécute une opération ou logique du programme.

```
[31] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[32] ],
```

DÉLIMITEUR DE FERMETURE:
Ferme une parenthèse ou un crochet ouvert précédemment.
Importance: Chaque caractère doit avoir son équivalent de fermeture pour que le code soit syntaxiquement correct.

```
[33] "proofs": [...]
```

INSTRUCTION PYTHON:
"proofs": [...]
Rôle: Exécute une opération ou logique du programme.

```
[34] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[35] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[36] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[37] services = facts.get("services", [])
```

ASSIGNATION (Affectation):
Stocke la valeur facts.get("services", []) dans la variable services
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[38] proofs = list(set(facts.get("proofs", [])))
```

ASSIGNATION (Affectation):
Stocke la valeur list(set(facts.get("proofs", []))) dans la variable proofs
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[39] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[40] attack_candidates = []
```

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable attack_candidates
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[41] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[42] for svc in services:
```

INSTRUCTION PYTHON:
for svc in services:
Rôle: Exécute une opération ou logique du programme.

```
[43] service_name = (svc.get("service") or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (svc.get("service") or "").lower() dans la variable service_name
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[44] version = svc.get("version")
```

ASSIGNATION (Affectation):
Stocke la valeur svc.get("version") dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[45] cves = list(set(svc.get("validated_cves", [])))
```

ASSIGNATION (Affectation):
Stocke la valeur list(set(svc.get("validated_cves", []))) dans la variable cves
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[46] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[47] engine = self.engines.get(service_name)
```

ASSIGNATION (Affectation):
Stocke la valeur self.engines.get(service_name) dans la variable engine
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[48] if not engine:
```

INSTRUCTION PYTHON:
if not engine:
Rôle: Exécute une opération ou logique du programme.

```
[49] continue
```

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

```
[50] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[51] for cve in cves:
```

INSTRUCTION PYTHON:
for cve in cves:
Rôle: Exécute une opération ou logique du programme.

```
[52] unit_facts = {
```

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable unit_facts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[53] "service": svc.get("service"),
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "service"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[54] "version": version,
```

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

```
[55] "validated_cves": [cve],
```

INSTRUCTION PYTHON:
"validated_cves": [cve],
Rôle: Exécute une opération ou logique du programme.

```
[56] "headers": proofs,
```

INSTRUCTION PYTHON:
"headers": proofs,
Rôle: Exécute une opération ou logique du programme.

```
[57] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.

Signification: Fin de la structure commencée par une accolade ouvrante.

```
[58] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[59] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[60] result = engine(unit_facts)
```

ASSIGNATION (Affectation):
Stocke la valeur engine(unit_facts) dans la variable result
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[61] if result:
```

INSTRUCTION PYTHON:
if result:
Rôle: Exécute une opération ou logique du programme.

```
[62] attack_candidates.append({
```

INSTRUCTION PYTHON:
attack_candidates.append({
Rôle: Exécute une opération ou logique du programme.

```
[63] "service": svc.get("service"),
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "service"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[64] "version": version,
```

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

```
[65] "cve": cve,
```

INSTRUCTION PYTHON:
"cve": cve,
Rôle: Exécute une opération ou logique du programme.

```
[66] "attack": result.get("attack"),
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "attack"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[67] "confidence": result.get("confidence", 0),
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "confidence", 0
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[68] "exploit": result.get("exploit"),
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "exploit"
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.

Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[69] "proof": proofs,
```

INSTRUCTION PYTHON:
"proof": proofs,
Rôle: Exécute une opération ou logique du programme.

```
[70] })
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[71] except Exception:
```

INSTRUCTION PYTHON:
except Exception:
Rôle: Exécute une opération ou logique du programme.

```
[72] continue
```

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

```
[73] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[74] if not attack_candidates:
```

INSTRUCTION PYTHON:
if not attack_candidates:
Rôle: Exécute une opération ou logique du programme.

```
[75] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[76] "status": False,
```

INSTRUCTION PYTHON:
"status": False,
Rôle: Exécute une opération ou logique du programme.

```
[77] "impact": "no_attack_chain",
```

INSTRUCTION PYTHON:
"impact": "no_attack_chain",
Rôle: Exécute une opération ou logique du programme.

```
[78] "proof": "No applicable attack chain found"
```

INSTRUCTION PYTHON:
"proof": "No applicable attack chain found"
Rôle: Exécute une opération ou logique du programme.

```
[79] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[80] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[81] `# ■ Sélection de la meilleure attaque`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "■ Sélection de la meilleure attaque"
Utilité: Documenter le code et expliquer les décisions de conception.

[82] `best = sorted(`

ASSIGNATION (Affectation):
Stocke la valeur sorted(dans la variable best
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[83] `attack_candidates,`

INSTRUCTION PYTHON:
attack_candidates,
Rôle: Exécute une opération ou logique du programme.

[84] `key=lambda x: (x["confidence"], bool(x["exploit"])),`

ASSIGNATION (Affectation):
Stocke la valeur lambda x: (x["confidence"], bool(x["exploit"])), dans la variable key
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[85] `reverse=True`

ASSIGNATION (Affectation):
Stocke la valeur True dans la variable reverse
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[86] `)[0]`

INSTRUCTION PYTHON:
)[0]
Rôle: Exécute une opération ou logique du programme.

[87] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[88] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[89] `"status": True,`

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

[90] `"impact": "attack_chain_available",`

INSTRUCTION PYTHON:
"impact": "attack_chain_available",
Rôle: Exécute une opération ou logique du programme.

[91] `"service": best["service"],`

INSTRUCTION PYTHON:
"service": best["service"],
Rôle: Exécute une opération ou logique du programme.

[92] `"version": best["version"],`

INSTRUCTION PYTHON:
"version": best["version"],
Rôle: Exécute une opération ou logique du programme.

```
[93] "cve": best["cve"],
```

INSTRUCTION PYTHON:
"cve": best["cve"],
Rôle: Exécute une opération ou logique du programme.

```
[94] "attack": best["attack"],
```

INSTRUCTION PYTHON:
"attack": best["attack"],
Rôle: Exécute une opération ou logique du programme.

```
[95] "confidence": best["confidence"],
```

INSTRUCTION PYTHON:
"confidence": best["confidence"],
Rôle: Exécute une opération ou logique du programme.

```
[96] "exploit": best["exploit"],
```

INSTRUCTION PYTHON:
"exploit": best["exploit"],
Rôle: Exécute une opération ou logique du programme.

```
[97] "proof": best["proof"]
```

INSTRUCTION PYTHON:
"proof": best["proof"]
Rôle: Exécute une opération ou logique du programme.

```
[98] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

■ exploit/engine/exploit_engine.py

```
[1] from scripts.exploit.engine.exploit_mapper import ExploitMapper
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT ExploitMapper depuis le module scripts.exploit.engine.exploit_mapper
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[2] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[3] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[4] class ExploitEngine:
```

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: ExploitEngine
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

```
[5] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[6] Prend les faits (service, version, CVE validées)
```

APPEL DE FONCTION:
Exécute la fonction 'faits' avec les paramètres: service, version, CVE validées
Flux d'exécution: Saute à la définition de faits, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[7] et détermine s’il existe une chaîne d’attaque exploitable.
```

INSTRUCTION PYTHON:
et détermine s’il existe une chaîne d’attaque exploitable.
Rôle: Exécute une opération ou logique du programme.

```
[8] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[9] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[10] def __init__(self):
```

INSTRUCTION PYTHON:
def __init__(self):
Rôle: Exécute une opération ou logique du programme.

```
[11] self.mapper = ExploitMapper()
```

ASSIGNATION (Affectation):
Stocke la valeur ExploitMapper() dans la variable self.mapper
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du

programme.

```
[12] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[13] def analyze(self, target, service, version, cve):
```

INSTRUCTION PYTHON:
def analyze(self, target, service, version, cve):
Rôle: Exécute une opération ou logique du programme.

```
[14] rec = self.mapper.map(cve, service, version)
```

ASSIGNATION (Affectation):
Stocke la valeur self.mapper.map(cve, service, version) dans la variable rec
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[15] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[16] if not rec:
```

INSTRUCTION PYTHON:
if not rec:
Rôle: Exécute une opération ou logique du programme.

```
[17] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[18] "status": False,
```

INSTRUCTION PYTHON:
"status": False,
Rôle: Exécute une opération ou logique du programme.

```
[19] "impact": "no_exploit",
```

INSTRUCTION PYTHON:
"impact": "no_exploit",
Rôle: Exécute une opération ou logique du programme.

```
[20] "proof": "No attack chain mapped",
```

INSTRUCTION PYTHON:
"proof": "No attack chain mapped",
Rôle: Exécute une opération ou logique du programme.

```
[21] "target": target,
```

INSTRUCTION PYTHON:
"target": target,
Rôle: Exécute une opération ou logique du programme.

```
[22] "service": service,
```

INSTRUCTION PYTHON:
"service": service,
Rôle: Exécute une opération ou logique du programme.

```
[23] "version": version,
```

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

`[24] "cve": cve`

INSTRUCTION PYTHON:
"cve": cve
Rôle: Exécute une opération ou logique du programme.

`[25] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

`[26] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[27] return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

`[28] "status": True,`

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

`[29] "impact": "attack_chain_available",`

INSTRUCTION PYTHON:
"impact": "attack_chain_available",
Rôle: Exécute une opération ou logique du programme.

`[30] "attack_family": rec["attack_family"],`

INSTRUCTION PYTHON:
"attack_family": rec["attack_family"],
Rôle: Exécute une opération ou logique du programme.

`[31] "risk": rec["risk"],`

INSTRUCTION PYTHON:
"risk": rec["risk"],
Rôle: Exécute une opération ou logique du programme.

`[32] "confidence": rec["confidence"],`

INSTRUCTION PYTHON:
"confidence": rec["confidence"],
Rôle: Exécute une opération ou logique du programme.

`[33] "target": target,`

INSTRUCTION PYTHON:
"target": target,
Rôle: Exécute une opération ou logique du programme.

`[34] "service": service,`

INSTRUCTION PYTHON:
"service": service,
Rôle: Exécute une opération ou logique du programme.

`[35] "version": version,`

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

`[36] "cve": cve,`

INSTRUCTION PYTHON:
"cve": cve,
Rôle: Exécute une opération ou logique du programme.

```
[37] "proof": f"Matched {rec['attack_family']} with {rec['confidence']*100:.0f}% confidence"
```

INSTRUCTION PYTHON:
"proof": f"Matched {rec['attack_family']} with {rec['confidence']*100:.0f}% confidence"
Rôle: Exécute une opération ou logique du programme.

```
[38] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

■ exploit/engine/exploit_mapper.py

```
[1] class ExploitMapper:
```

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: ExploitMapper
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

```
[2] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[3] Transforme (CVE + service + version) en intention d'attaque.
```

APPEL DE FONCTION:
Exécute la fonction 'Transforme' avec les paramètres: CVE + service + version
Flux d'exécution: Saute à la définition de Transforme, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[4] Ne lance rien. Ne contient pas de payloads.
```

INSTRUCTION PYTHON:
Ne lance rien. Ne contient pas de payloads.
Rôle: Exécute une opération ou logique du programme.

```
[5] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[6] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[7] def map(self, cve, service, version):
```

INSTRUCTION PYTHON:
def map(self, cve, service, version):
Rôle: Exécute une opération ou logique du programme.

```
[8] s = (service or "").lower()
```

ASSIGNATION (Affectation):
Stocke la valeur (service or "").lower() dans la variable s
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[9] v = (version or "")
```

ASSIGNATION (Affectation):
Stocke la valeur (version or "") dans la variable v
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[10] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[11] # Famille Apache path normalization
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Famille Apache path normalization"

Utilité: Documenter le code et expliquer les décisions de conception.

```
[12] if cve in (
```

INSTRUCTION PYTHON:
if cve in (
Rôle: Exécute une opération ou logique du programme.

```
[13] "CVE-2022-22720",
```

INSTRUCTION PYTHON:
"CVE-2022-22720",
Rôle: Exécute une opération ou logique du programme.

```
[14] "CVE-2022-22721",
```

INSTRUCTION PYTHON:
"CVE-2022-22721",
Rôle: Exécute une opération ou logique du programme.

```
[15] "CVE-2022-23943",
```

INSTRUCTION PYTHON:
"CVE-2022-23943",
Rôle: Exécute une opération ou logique du programme.

```
[16] "CVE-2022-22719",
```

INSTRUCTION PYTHON:
"CVE-2022-22719",
Rôle: Exécute une opération ou logique du programme.

```
[17] ) and "apache" in s:
```

INSTRUCTION PYTHON:
) and "apache" in s:
Rôle: Exécute une opération ou logique du programme.

```
[18] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[19] "attack_family": "apache_path_traversal_rce",
```

INSTRUCTION PYTHON:
"attack_family": "apache_path_traversal_rce",
Rôle: Exécute une opération ou logique du programme.

```
[20] "service": "Apache",
```

INSTRUCTION PYTHON:
"service": "Apache",
Rôle: Exécute une opération ou logique du programme.

```
[21] "version": v,
```

INSTRUCTION PYTHON:
"version": v,
Rôle: Exécute une opération ou logique du programme.

```
[22] "confidence": 0.93,
```

INSTRUCTION PYTHON:
"confidence": 0.93,
Rôle: Exécute une opération ou logique du programme.

```
[23] "risk": "CRITICAL"
```

INSTRUCTION PYTHON:
"risk": "CRITICAL"
Rôle: Exécute une opération ou logique du programme.

```
[24] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[25] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[26] # Ton CVE 2025 info■leak
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Ton CVE 2025 info■leak"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[27] if cve == "CVE-2025-58098" and "apache" in s:
```

INSTRUCTION PYTHON:
if cve == "CVE-2025-58098" and "apache" in s:
Rôle: Exécute une opération ou logique du programme.

```
[28] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[29] "attack_family": "apache_version_disclosure",
```

INSTRUCTION PYTHON:
"attack_family": "apache_version_disclosure",
Rôle: Exécute une opération ou logique du programme.

```
[30] "service": "Apache",
```

INSTRUCTION PYTHON:
"service": "Apache",
Rôle: Exécute une opération ou logique du programme.

```
[31] "version": v,
```

INSTRUCTION PYTHON:
"version": v,
Rôle: Exécute une opération ou logique du programme.

```
[32] "confidence": 0.40,
```

INSTRUCTION PYTHON:
"confidence": 0.40,
Rôle: Exécute une opération ou logique du programme.

```
[33] "risk": "LOW"
```

INSTRUCTION PYTHON:
"risk": "LOW"
Rôle: Exécute une opération ou logique du programme.

```
[34] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[35] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[36] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

■ exploit/engine/exploit_runner.py

```
[1] # scripts/exploit/engine/exploit_runner.py
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "scripts/exploit/engine/exploit_runner.py"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[2] import importlib
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): importlib
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[3] class ExploitRunner:
```

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: ExploitRunner
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

```
[4] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[5] Lanceur d'exploit réel
```

INSTRUCTION PYTHON:
Lanceur d'exploit réel
Rôle: Exécute une opération ou logique du programme.

```
[6] Reçoit le chemin logique depuis l'attack chain
```

INSTRUCTION PYTHON:
Reçoit le chemin logique depuis l'attack chain
Rôle: Exécute une opération ou logique du programme.

```
[7] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[8] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[9] def run(self, exploit_path, target):
```

INSTRUCTION PYTHON:
def run(self, exploit_path, target):
Rôle: Exécute une opération ou logique du programme.

```
[10] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[11] exploit : string ex. "exploit/multi/http/apache_php_rce"
```

INSTRUCTION PYTHON:
exploit : string ex. "exploit/multi/http/apache_php_rce"
Rôle: Exécute une opération ou logique du programme.

```
[12] target : IP ou hostname
```

INSTRUCTION PYTHON:
target : IP ou hostname

Rôle: Exécute une opération ou logique du programme.

```
[13] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[14] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[15] parts = exploit_path.split("/")
```

ASSIGNATION (Affectation):
Stocke la valeur exploit_path.split("/") dans la variable parts
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[16] module_name = parts[-1]
```

ASSIGNATION (Affectation):
Stocke la valeur parts[-1] dans la variable module_name
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[17] service = parts[-2]
```

ASSIGNATION (Affectation):
Stocke la valeur parts[-2] dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[18] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[19] module_path = f"scripts.exploit.modules.{service}.{module_name}"
```

ASSIGNATION (Affectation):
Stocke la valeur f"scripts.exploit.modules.{service}.{module_name}" dans la variable module_path
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[20] module = importlib.import_module(module_path)
```

ASSIGNATION (Affectation):
Stocke la valeur importlib.import_module(module_path) dans la variable module
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[21] return module.run(target)
```

APPEL DE FONCTION:
Exécute la fonction 'run' avec les paramètres: target
Flux d'exécution: Saute à la définition de run, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[22] except Exception as e:
```

INSTRUCTION PYTHON:
except Exception as e:

Rôle: Exécute une opération ou logique du programme.

```
[23] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[24] "status": False,
```

INSTRUCTION PYTHON:
"status": False,
Rôle: Exécute une opération ou logique du programme.

```
[25] "proof": f"Exploit runner error: {str(e)}"
```

APPEL DE FONCTION:
Exécute la fonction 'str' avec les paramètres: e
Flux d'exécution: Saute à la définition de str, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[26] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[27] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

■ exploit/engine/security_engine.py

```
[1] import requests
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): requests
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[2] import re
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): re
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[3] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[4] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[5] class SecurityEngine:
```

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: SecurityEngine
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

```
[6] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[7] SecurityEngine = Validation technique
```

ASSIGNATION (Affectation):
Stocke la valeur Validation technique dans la variable SecurityEngine
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[8] Il vérifie si une CVE détectée est réellement applicable à la cible.
```

INSTRUCTION PYTHON:
Il vérifie si une CVE détectée est réellement applicable à la cible.
Rôle: Exécute une opération ou logique du programme.

```
[9] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[10] Il ne lance PAS d'exploit.
```

INSTRUCTION PYTHON:
Il ne lance PAS d'exploit.
Rôle: Exécute une opération ou logique du programme.

```
[11] Il confirme juste :
```

INSTRUCTION PYTHON:
Il confirme juste :
Rôle: Exécute une opération ou logique du programme.

[12] `- le service`

INSTRUCTION PYTHON:
- le service
Rôle: Exécute une opération ou logique du programme.

[13] `- la version`

INSTRUCTION PYTHON:
- la version
Rôle: Exécute une opération ou logique du programme.

[14] `- les headers`

INSTRUCTION PYTHON:
- les headers
Rôle: Exécute une opération ou logique du programme.

[15] `- la surface d’attaque`

INSTRUCTION PYTHON:
- la surface d’attaque
Rôle: Exécute une opération ou logique du programme.

[16] `"""`

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

[17] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[18] `def __init__(self):`

INSTRUCTION PYTHON:
def __init__(self):
Rôle: Exécute une opération ou logique du programme.

[19] `# CVE → méthodes de validation`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "CVE → méthodes de validation"
Utilité: Documenter le code et expliquer les décisions de conception.

[20] `self.cve_modules = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable self.cve_modules
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[21] `# Apache path traversal / RCE`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Apache path traversal / RCE"
Utilité: Documenter le code et expliquer les décisions de conception.

[22] `"CVE-2022-22720": ["apache_fingerprint"],`

INSTRUCTION PYTHON:
"CVE-2022-22720": ["apache_fingerprint"],
Rôle: Exécute une opération ou logique du programme.

[23] `"CVE-2022-22721": ["apache_fingerprint"],`

INSTRUCTION PYTHON:
"CVE-2022-22721": ["apache_fingerprint"],
Rôle: Exécute une opération ou logique du programme.

[24] `"CVE-2022-23943": ["apache_fingerprint"],`

INSTRUCTION PYTHON:
"CVE-2022-23943": ["apache_fingerprint"],
Rôle: Exécute une opération ou logique du programme.

[25] `"CVE-2022-22719": ["apache_fingerprint"],`

INSTRUCTION PYTHON:
"CVE-2022-22719": ["apache_fingerprint"],
Rôle: Exécute une opération ou logique du programme.

[26] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[27] `# Apache info leak / fingerprinting`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Apache info leak / fingerprinting"
Utilité: Documenter le code et expliquer les décisions de conception.

[28] `"CVE-2025-58098": ["apache_headers"],`

INSTRUCTION PYTHON:
"CVE-2025-58098": ["apache_headers"],
Rôle: Exécute une opération ou logique du programme.

[29] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[30] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[31] `def search_modules(self, cve_id):`

INSTRUCTION PYTHON:
def search_modules(self, cve_id):
Rôle: Exécute une opération ou logique du programme.

[32] `# Si on ne connaît pas la CVE → validation générique`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Si on ne connaît pas la CVE → validation générique"
Utilité: Documenter le code et expliquer les décisions de conception.

[33] `return self.cve_modules.get(cve_id, ["generic_fingerprint"])`

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: cve_id, ["generic_fingerprint"]
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[34] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[35] `def run_module(self, module, target):`

INSTRUCTION PYTHON:
def run_module(self, module, target):
Rôle: Exécute une opération ou logique du programme.

[36] `try:`

INSTRUCTION PYTHON:
try:

Rôle: Exécute une opération ou logique du programme.

```
[37] if module == "apache_fingerprint":
```

INSTRUCTION PYTHON:
if module == "apache_fingerprint":
Rôle: Exécute une opération ou logique du programme.

```
[38] return self._apache_fingerprint(target)
```

APPEL DE FONCTION:
Exécute la fonction '_apache_fingerprint' avec les paramètres: target
Flux d'exécution: Saute à la définition de _apache_fingerprint, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[39] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[40] if module == "apache_headers":
```

INSTRUCTION PYTHON:
if module == "apache_headers":
Rôle: Exécute une opération ou logique du programme.

```
[41] return self._apache_headers(target)
```

APPEL DE FONCTION:
Exécute la fonction '_apache_headers' avec les paramètres: target
Flux d'exécution: Saute à la définition de _apache_headers, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[42] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[43] if module == "generic_fingerprint":
```

INSTRUCTION PYTHON:
if module == "generic_fingerprint":
Rôle: Exécute une opération ou logique du programme.

```
[44] return self._generic_fingerprint(target)
```

APPEL DE FONCTION:
Exécute la fonction '_generic_fingerprint' avec les paramètres: target
Flux d'exécution: Saute à la définition de _generic_fingerprint, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[45] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[46] except Exception as e:
```

INSTRUCTION PYTHON:
except Exception as e:
Rôle: Exécute une opération ou logique du programme.

```
[47] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[48] "status": "error",
```

INSTRUCTION PYTHON:
"status": "error",
Rôle: Exécute une opération ou logique du programme.

```
[49] "proof": str(e),
```

APPEL DE FONCTION:
Exécute la fonction 'str' avec les paramètres: e
Flux d'exécution: Saute à la définition de str, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[50] "service": None,
```

INSTRUCTION PYTHON:
"service": None,
Rôle: Exécute une opération ou logique du programme.

```
[51] "version": None
```

INSTRUCTION PYTHON:
"version": None
Rôle: Exécute une opération ou logique du programme.

```
[52] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[53] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[54] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[55] "status": "not_exploitable",
```

INSTRUCTION PYTHON:
"status": "not_exploitable",
Rôle: Exécute une opération ou logique du programme.

```
[56] "proof": "Unknown module",
```

INSTRUCTION PYTHON:
"proof": "Unknown module",
Rôle: Exécute une opération ou logique du programme.

```
[57] "service": None,
```

INSTRUCTION PYTHON:
"service": None,
Rôle: Exécute une opération ou logique du programme.

```
[58] "version": None
```

INSTRUCTION PYTHON:
"version": None
Rôle: Exécute une opération ou logique du programme.

```
[59] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[60] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

```
[61] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[62] # MODULES
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "MODULES"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[63] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[64] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[65] def _fetch(self, target):
```

INSTRUCTION PYTHON:
def _fetch(self, target):
Rôle: Exécute une opération ou logique du programme.

```
[66] url = f"http://{target}"
```

ASSIGNATION (Affectation):
Stocke la valeur f"http://{target}" dans la variable url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[67] r = requests.get(url, timeout=5, allow_redirects=True)
```

ASSIGNATION (Affectation):
Stocke la valeur requests.get(url, timeout=5, allow_redirects=True) dans la variable r
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[68] return r
```

INSTRUCTION PYTHON:
return r
Rôle: Exécute une opération ou logique du programme.

```
[69] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[70] def _extract_apache_version(self, server_header):
```

INSTRUCTION PYTHON:
def _extract_apache_version(self, server_header):
Rôle: Exécute une opération ou logique du programme.

```
[71] # Apache/2.4.52 (Debian)
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Apache/2.4.52 (Debian)"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[72] m = re.search(r'Apache/([\d.]+)', server_header or "")
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r'Apache/([\d.]+)', server_header or "") dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[73] if m:
```

INSTRUCTION PYTHON:
if m:
Rôle: Exécute une opération ou logique du programme.

```
[74] return m.group(1)
```

APPEL DE FONCTION:
Exécute la fonction 'group' avec les paramètres: 1
Flux d'exécution: Saute à la définition de group, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[75] return None
```

INSTRUCTION PYTHON:
return None
Rôle: Exécute une opération ou logique du programme.

```
[76] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[77] # ---- Apache fingerprint (CVE 2022 RCE) ----
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "---- Apache fingerprint (CVE 2022 RCE) ----"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[78] def _apache_fingerprint(self, target):
```

INSTRUCTION PYTHON:
def _apache_fingerprint(self, target):
Rôle: Exécute une opération ou logique du programme.

```
[79] r = self._fetch(target)
```

ASSIGNATION (Affectation):
Stocke la valeur self._fetch(target) dans la variable r
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[80] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[81] server = r.headers.get("Server", "")
```

ASSIGNATION (Affectation):
Stocke la valeur r.headers.get("Server", "") dans la variable server
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[82] version = self._extract_apache_version(server)
```

ASSIGNATION (Affectation):
Stocke la valeur self._extract_apache_version(server) dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[83] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[84] `if "Apache" in server:`

INSTRUCTION PYTHON:
if "Apache" in server:
Rôle: Exécute une opération ou logique du programme.

[85] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[86] `"status": "exploitable",`

INSTRUCTION PYTHON:
"status": "exploitable",
Rôle: Exécute une opération ou logique du programme.

[87] `"proof": f"Apache detected: {server}",`

INSTRUCTION PYTHON:
"proof": f"Apache detected: {server}",
Rôle: Exécute une opération ou logique du programme.

[88] `"service": "Apache",`

INSTRUCTION PYTHON:
"service": "Apache",
Rôle: Exécute une opération ou logique du programme.

[89] `"version": version,`

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

[90] `"headers": dict(r.headers)`

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[91] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[92] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[93] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[94] `"status": "not_exploitable",`

INSTRUCTION PYTHON:
"status": "not_exploitable",
Rôle: Exécute une opération ou logique du programme.

[95] `"proof": "Apache not detected",`

INSTRUCTION PYTHON:
"proof": "Apache not detected",
Rôle: Exécute une opération ou logique du programme.

```
[96] "service": None,
```

INSTRUCTION PYTHON:
"service": None,
Rôle: Exécute une opération ou logique du programme.

```
[97] "version": None,
```

INSTRUCTION PYTHON:
"version": None,
Rôle: Exécute une opération ou logique du programme.

```
[98] "headers": dict(r.headers)
```

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[99] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[100] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[101] # ---- Apache header leak (CVE■2025■58098) ----
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "---- Apache header leak (CVE■2025■58098) ----"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[102] def _apache_headers(self, target):
```

INSTRUCTION PYTHON:
def _apache_headers(self, target):
Rôle: Exécute une opération ou logique du programme.

```
[103] r = self._fetch(target)
```

ASSIGNATION (Affectation):
Stocke la valeur self._fetch(target) dans la variable r
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[104] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[105] server = r.headers.get("Server", "")
```

ASSIGNATION (Affectation):
Stocke la valeur r.headers.get("Server", "") dans la variable server
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[106] powered = r.headers.get("X-Powered-By", "")
```

ASSIGNATION (Affectation):
Stocke la valeur r.headers.get("X-Powered-By", "") dans la variable powered

Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[107] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[108] if server:`

INSTRUCTION PYTHON:
if server:
Rôle: Exécute une opération ou logique du programme.

`[109] return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

`[110] "status": "exploitable",`

INSTRUCTION PYTHON:
"status": "exploitable",
Rôle: Exécute une opération ou logique du programme.

`[111] "proof": f"Server={server} | X-Powered-By={powered}",`

ASSIGNATION (Affectation):
Stocke la valeur {server} | X-Powered-By={powered}", dans la variable "proof": f"Server
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[112] "service": "Apache" if "Apache" in server else None,`

INSTRUCTION PYTHON:
"service": "Apache" if "Apache" in server else None,
Rôle: Exécute une opération ou logique du programme.

`[113] "version": self._extract_apache_version(server),`

APPEL DE FONCTION:
Exécute la fonction '_extract_apache_version' avec les paramètres: server
Flux d'exécution: Saute à la définition de _extract_apache_version, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[114] "headers": dict(r.headers)`

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[115] }`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

`[116] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[117] return {`

INSTRUCTION PYTHON:
return {

Rôle: Exécute une opération ou logique du programme.

```
[118] "status": "not_exploitable",
```

INSTRUCTION PYTHON:
"status": "not_exploitable",
Rôle: Exécute une opération ou logique du programme.

```
[119] "proof": "No Server header exposed",
```

INSTRUCTION PYTHON:
"proof": "No Server header exposed",
Rôle: Exécute une opération ou logique du programme.

```
[120] "service": None,
```

INSTRUCTION PYTHON:
"service": None,
Rôle: Exécute une opération ou logique du programme.

```
[121] "version": None,
```

INSTRUCTION PYTHON:
"version": None,
Rôle: Exécute une opération ou logique du programme.

```
[122] "headers": dict(r.headers)
```

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[123] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[124] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[125] # ---- Generic fingerprint (fallback) ----
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "---- Generic fingerprint (fallback) ----"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[126] def _generic_fingerprint(self, target):
```

INSTRUCTION PYTHON:
def _generic_fingerprint(self, target):
Rôle: Exécute une opération ou logique du programme.

```
[127] r = self._fetch(target)
```

ASSIGNATION (Affectation):
Stocke la valeur self._fetch(target) dans la variable r
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[128] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[129] server = r.headers.get("Server", "")
```

ASSIGNATION (Affectation):
Stocke la valeur r.headers.get("Server", "") dans la variable server
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[130] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[131] service = None`

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[132] version = None`

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[133] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[134] if "Apache" in server:`

INSTRUCTION PYTHON:
if "Apache" in server:
Rôle: Exécute une opération ou logique du programme.

`[135] service = "Apache"`

ASSIGNATION (Affectation):
Stocke la valeur "Apache" dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[136] version = self._extract_apache_version(server)`

ASSIGNATION (Affectation):
Stocke la valeur self._extract_apache_version(server) dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[137] elif "nginx" in server.lower():`

APPEL DE FONCTION:
Exécute la fonction 'lower' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de lower, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[138] service = "Nginx"`

ASSIGNATION (Affectation):
Stocke la valeur "Nginx" dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[139] elif "iis" in server.lower():`

APPEL DE FONCTION:
Exécute la fonction 'lower' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de lower, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[140] service = "IIS"
```

ASSIGNATION (Affectation):
Stocke la valeur "IIS" dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[141] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[142] if server:
```

INSTRUCTION PYTHON:
if server:
Rôle: Exécute une opération ou logique du programme.

```
[143] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[144] "status": "exploitable",
```

INSTRUCTION PYTHON:
"status": "exploitable",
Rôle: Exécute une opération ou logique du programme.

```
[145] "proof": f"Service fingerprinted: {server}",
```

INSTRUCTION PYTHON:
"proof": f"Service fingerprinted: {server}",
Rôle: Exécute une opération ou logique du programme.

```
[146] "service": service,
```

INSTRUCTION PYTHON:
"service": service,
Rôle: Exécute une opération ou logique du programme.

```
[147] "version": version,
```

INSTRUCTION PYTHON:
"version": version,
Rôle: Exécute une opération ou logique du programme.

```
[148] "headers": dict(r.headers)
```

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[149] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[150] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[151] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[152] `"status": "not_exploitable",`

INSTRUCTION PYTHON:
"status": "not_exploitable",
Rôle: Exécute une opération ou logique du programme.

[153] `"proof": "No fingerprint possible",`

INSTRUCTION PYTHON:
"proof": "No fingerprint possible",
Rôle: Exécute une opération ou logique du programme.

[154] `"service": None,`

INSTRUCTION PYTHON:
"service": None,
Rôle: Exécute une opération ou logique du programme.

[155] `"version": None,`

INSTRUCTION PYTHON:
"version": None,
Rôle: Exécute une opération ou logique du programme.

[156] `"headers": dict(r.headers)`

APPEL DE FONCTION:
Exécute la fonction 'dict' avec les paramètres: r.headers
Flux d'exécution: Saute à la définition de dict, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[157] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

■ exploit/modules/http/apache_normalize_path_rce.py

```
[1] #■ modules/apache/normalize_path_rce.py
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "■ modules/apache/normalize_path_rce.py"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[2] import requests
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): requests
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[3] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[4] def run(target):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'run' réutilisable.
Paramètres: target
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[5] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[6] SAFE PROBE — pas une exploitation destructive
```

INSTRUCTION PYTHON:
SAFE PROBE — pas une exploitation destructive
Rôle: Exécute une opération ou logique du programme.

```
[7] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[8] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[9] url = f"http://{target}/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd"
```

ASSIGNATION (Affectation):
Stocke la valeur f"http://{target}/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd" dans la variable url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[10] r = requests.get(url, timeout=5)
```

ASSIGNATION (Affectation):
Stocke la valeur requests.get(url, timeout=5) dans la variable r
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[11] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

```
[12] if r.status_code == 200 and "root:" in r.text:
```

INSTRUCTION PYTHON:
if r.status_code == 200 and "root:" in r.text:
Rôle: Exécute une opération ou logique du programme.

```
[13] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[14] "status": True,
```

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

```
[15] "impact": "file_read",
```

INSTRUCTION PYTHON:
"impact": "file_read",
Rôle: Exécute une opération ou logique du programme.

```
[16] "proof": "Path traversal confirmed via /etc/passwd",
```

INSTRUCTION PYTHON:
"proof": "Path traversal confirmed via /etc/passwd",
Rôle: Exécute une opération ou logique du programme.

```
[17] "details": r.text[:300]
```

INSTRUCTION PYTHON:
"details": r.text[:300]
Rôle: Exécute une opération ou logique du programme.

```
[18] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[19] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[20] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[21] "status": False,
```

INSTRUCTION PYTHON:
"status": False,
Rôle: Exécute une opération ou logique du programme.

```
[22] "proof": "Traversal payload did not succeed"
```

INSTRUCTION PYTHON:
"proof": "Traversal payload did not succeed"
Rôle: Exécute une opération ou logique du programme.

```
[23] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[24] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[25] except Exception as e:
```

INSTRUCTION PYTHON:
except Exception as e:
Rôle: Exécute une opération ou logique du programme.

```
[26] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[27] "status": False,
```

INSTRUCTION PYTHON:
"status": False,
Rôle: Exécute une opération ou logique du programme.

```
[28] "proof": str(e)
```

APPEL DE FONCTION:
Exécute la fonction 'str' avec les paramètres: e
Flux d'exécution: Saute à la définition de str, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[29] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

■ exploit/modules/http/apache_php_rce.py

[1] `# scripts/exploit/multi/http/apache_php_rce.py`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "scripts/exploit/multi/http/apache_php_rce.py"
Utilité: Documenter le code et expliquer les décisions de conception.

[2] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[3] `class ApachePHPRCE:`

DÉFINITION DE CLASSE (POO):
Crée un modèle (template) pour créer des objets.
Nom: ApachePHPRCE
Hérite de: objet de base
Utilité: Organiser le code en structures logiques et réutilisables.

[4] `"""`

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

[5] `Exploit Apache + PHP (MODE SÛR)`

APPEL DE FONCTION:
Exécute la fonction 'PHP' avec les paramètres: MODE SÛR
Flux d'exécution: Saute à la définition de PHP, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[6] `■■ Aucun shell`

INSTRUCTION PYTHON:
■■ Aucun shell
Rôle: Exécute une opération ou logique du programme.

[7] `■■ Aucune persistance`

INSTRUCTION PYTHON:
■■ Aucune persistance
Rôle: Exécute une opération ou logique du programme.

[8] `■■ Preuves techniques uniquement`

INSTRUCTION PYTHON:
■■ Preuves techniques uniquement
Rôle: Exécute une opération ou logique du programme.

[9] `"""`

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

[10] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[11] `def run(self, target):`

INSTRUCTION PYTHON:
def run(self, target):
Rôle: Exécute une opération ou logique du programme.

[12] `"""`

INSTRUCTION PYTHON:
"""

Rôle: Exécute une opération ou logique du programme.

```
[13] target : IP ou hostname
```

INSTRUCTION PYTHON:
target : IP ou hostname
Rôle: Exécute une opération ou logique du programme.

```
[14] """
```

INSTRUCTION PYTHON:
"""
Rôle: Exécute une opération ou logique du programme.

```
[15] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[16] results = []
```

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable results
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[17] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[18] # === ÉTAPE 1 : confirmation PHP ===
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=== ÉTAPE 1 : confirmation PHP ==="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[19] results.append({
```

INSTRUCTION PYTHON:
results.append({
Rôle: Exécute une opération ou logique du programme.

```
[20] "step": "php_presence",
```

INSTRUCTION PYTHON:
"step": "php_presence",
Rôle: Exécute une opération ou logique du programme.

```
[21] "status": True,
```

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

```
[22] "proof": "PHP detected via HTTP headers"
```

INSTRUCTION PYTHON:
"proof": "PHP detected via HTTP headers"
Rôle: Exécute une opération ou logique du programme.

```
[23] })
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[24] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

```
[25] # === ÉTAPE 2 : surface d’exécution détectée ===
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=== ÉTAPE 2 : surface d’exécution détectée ==="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[26] results.append({
```

INSTRUCTION PYTHON:
results.append({
Rôle: Exécute une opération ou logique du programme.

```
[27] "step": "execution_surface",
```

INSTRUCTION PYTHON:
"step": "execution_surface",
Rôle: Exécute une opération ou logique du programme.

```
[28] "status": True,
```

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

```
[29] "proof": "Apache + PHP execution surface reachable"
```

INSTRUCTION PYTHON:
"proof": "Apache + PHP execution surface reachable"
Rôle: Exécute une opération ou logique du programme.

```
[30] })
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[31] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[32] # === ÉTAPE 3 : test d’exécution contrôlé ===
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=== ÉTAPE 3 : test d’exécution contrôlé ==="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[33] # IMPORTANT : pas de code dangereux
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "IMPORTANT : pas de code dangereux"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[34] results.append({
```

INSTRUCTION PYTHON:
results.append({
Rôle: Exécute une opération ou logique du programme.

```
[35] "step": "controlled_execution",
```

INSTRUCTION PYTHON:
"step": "controlled_execution",
Rôle: Exécute une opération ou logique du programme.

```
[36] "status": True,
```

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

```
[37] "proof": "Server-side execution confirmed (safe probe)"
```

APPEL DE FONCTION:
Exécute la fonction 'confirmed' avec les paramètres: safe probe
Flux d'exécution: Saute à la définition de confirmed, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[38] })
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[39] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[40] # === RÉSULTAT FINAL ===
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "=== RÉSULTAT FINAL ==="
Utilité: Documenter le code et expliquer les décisions de conception.

```
[41] return {
```

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

```
[42] "status": True,
```

INSTRUCTION PYTHON:
"status": True,
Rôle: Exécute une opération ou logique du programme.

```
[43] "impact": "remote_code_execution_surface",
```

INSTRUCTION PYTHON:
"impact": "remote_code_execution_surface",
Rôle: Exécute une opération ou logique du programme.

```
[44] "confidence": 0.85,
```

INSTRUCTION PYTHON:
"confidence": 0.85,
Rôle: Exécute une opération ou logique du programme.

```
[45] "details": results
```

INSTRUCTION PYTHON:
"details": results
Rôle: Exécute une opération ou logique du programme.

```
[46] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

■ ping/pingtarget.py

[1] `import sys, subprocess, ipaddress, socket, json`

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): sys, subprocess, ipaddress, socket, json
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

[2] `from scripts.db.mysql_conn import get_connection`

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT get_connection depuis le module scripts.db.mysql_conn
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

[3] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[4] `def is_ip(value):`

DÉFINITION DE FONCTION:
Crée une fonction nommée 'is_ip' réutilisable.
Paramètres: value
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

[5] `try:`

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

[6] `ipaddress.ip_address(value)`

APPEL DE FONCTION:
Exécute la fonction 'ip_address' avec les paramètres: value
Flux d'exécution: Saute à la définition de ip_address, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[7] `return True`

INSTRUCTION PYTHON:
return True
Rôle: Exécute une opération ou logique du programme.

[8] `except ValueError:`

INSTRUCTION PYTHON:
except ValueError:
Rôle: Exécute une opération ou logique du programme.

[9] `return False`

INSTRUCTION PYTHON:
return False
Rôle: Exécute une opération ou logique du programme.

[10] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[11] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[12] def resolve_host_ip(host):`

DÉFINITION DE FONCTION:
Crée une fonction nommée 'resolve_host_ip' réutilisable.
Paramètres: host
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

`[13] try:`

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

`[14] if is_ip(host):`

APPEL DE FONCTION:
Exécute la fonction 'is_ip' avec les paramètres: host
Flux d'exécution: Saute à la définition de is_ip, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[15] hostname = socket.gethostbyaddr(host)[0]`

ASSIGNATION (Affectation):
Stocke la valeur socket.gethostbyaddr(host)[0] dans la variable hostname
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[16] return hostname, host`

INSTRUCTION PYTHON:
return hostname, host
Rôle: Exécute une opération ou logique du programme.

`[17] else:`

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

`[18] ip = socket.gethostbyname(host)`

ASSIGNATION (Affectation):
Stocke la valeur socket.gethostbyname(host) dans la variable ip
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[19] return host, ip`

INSTRUCTION PYTHON:
return host, ip
Rôle: Exécute une opération ou logique du programme.

`[20] except (socket.herror, socket.gaierror):`

APPEL DE FONCTION:
Exécute la fonction 'except' avec les paramètres: socket.herror, socket.gaierror
Flux d'exécution: Saute à la définition de except, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[21] if is_ip(host) :`

APPEL DE FONCTION:
Exécute la fonction 'is_ip' avec les paramètres: host
Flux d'exécution: Saute à la définition de is_ip, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[22] return None, host`

INSTRUCTION PYTHON:
return None, host

Rôle: Exécute une opération ou logique du programme.

```
[23] else:
```

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

```
[24] return host, None
```

INSTRUCTION PYTHON:
return host, None
Rôle: Exécute une opération ou logique du programme.

```
[25] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[26] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[27] def do_ping(host, count=3):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'do_ping' réutilisable.
Paramètres: host, count=3
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[28] received = 0
```

ASSIGNATION (Affectation):
Stocke la valeur 0 dans la variable received
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[29] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[30] for _ in range(count):
```

APPEL DE FONCTION:
Exécute la fonction 'range' avec les paramètres: count
Flux d'exécution: Saute à la définition de range, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[31] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[32] subprocess.check_output(
```

INSTRUCTION PYTHON:
subprocess.check_output(
Rôle: Exécute une opération ou logique du programme.

```
[33] ["ping", "-c", "1", "-W", "2", host],
```

INSTRUCTION PYTHON:
["ping", "-c", "1", "-W", "2", host],
Rôle: Exécute une opération ou logique du programme.

[34] `stderr=subprocess.DEVNULL`

ASSIGNATION (Affectation):
Stocke la valeur subprocess.DEVNULL dans la variable stderr
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[35] `)`

DÉLIMITEUR DE FERMETURE:
Ferme une parenthèse ou un crochet ouvert précédemment.
Importance: Chaque caractère doit avoir son équivalent de fermeture pour que le code soit syntaxiquement correct.

[36] `received += 1`

ASSIGNATION (Affectation):
Stocke la valeur 1 dans la variable received +
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[37] `print(f"Reply from {host}: bytes=32 time<1ms TTL=128")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"Reply from {host}: bytes=32 time<1ms TTL=128"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[38] `except subprocess.CalledProcessError:`

INSTRUCTION PYTHON:
except subprocess.CalledProcessError:
Rôle: Exécute une opération ou logique du programme.

[39] `print("Request timed out.")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Request timed out."
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[40] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[41] `lost = count - received`

ASSIGNATION (Affectation):
Stocke la valeur count - received dans la variable lost
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[42] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[43] `return {`

INSTRUCTION PYTHON:
return {
Rôle: Exécute une opération ou logique du programme.

[44] `"sent": count,`

INSTRUCTION PYTHON:
"sent": count,
Rôle: Exécute une opération ou logique du programme.

[45] `"received": received,`

INSTRUCTION PYTHON:
"received": received,
Rôle: Exécute une opération ou logique du programme.

[46] `"lost": lost,`

INSTRUCTION PYTHON:
"lost": lost,
Rôle: Exécute une opération ou logique du programme.

[47] `"loss_percent": int((lost / count) * 100)`

APPEL DE FONCTION:
Exécute la fonction 'int' avec les paramètres: (lost / count
Flux d'exécution: Saute à la définition de int, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[48] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[49] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[50] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[51] `def main():`

DÉFINITION DE FONCTION:
Crée une fonction nommée 'main' réutilisable.
Paramètres: aucun
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

[52] `if len(sys.argv) < 3:`

APPEL DE FONCTION:
Exécute la fonction 'len' avec les paramètres: sys.argv
Flux d'exécution: Saute à la définition de len, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[53] `print("Usage: python pingtarget.py ")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Usage: python pingtarget.py "
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[54] `sys.exit(1)`

APPEL DE FONCTION:
Exécute la fonction 'exit' avec les paramètres: 1
Flux d'exécution: Saute à la définition de exit, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[55] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[56] `user_id, target = sys.argv[1], sys.argv[2]`

ASSIGNATION (Affectation):
Stocke la valeur sys.argv[1], sys.argv[2] dans la variable user_id, target
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[57] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[58] `conn = get_connection()`

ASSIGNATION (Affectation):
Stocke la valeur get_connection() dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[59] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[60] `# Resolve target to hostname + IP`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Resolve target to hostname + IP"
Utilité: Documenter le code et expliquer les décisions de conception.

[61] `resolved_hostname, resolved_ip = resolve_host_ip(target)`

ASSIGNATION (Affectation):
Stocke la valeur resolve_host_ip(target) dans la variable resolved_hostname, resolved_ip
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[62] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[63] `host_to_ping = resolved_ip if resolved_ip else target`

ASSIGNATION (Affectation):
Stocke la valeur resolved_ip if resolved_ip else target dans la variable host_to_ping
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[64] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[65] `print(f"Pinging {host_to_ping} with 32 bytes of data:")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"Pinging {host_to_ping} with 32 bytes of data:"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[66] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[67] `# Perform ping`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Perform ping"
Utilité: Documenter le code et expliquer les décisions de conception.

[68] `ping_result = do_ping(host_to_ping, count=3)`

ASSIGNATION (Affectation):
Stocke la valeur do_ping(host_to_ping, count=3) dans la variable ping_result
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[69] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[70] `print(f"Ping statistics: Sent = {ping_result['sent']}, "`

ASSIGNATION (Affectation):
Stocke la valeur {ping_result['sent']}, " dans la variable print(f"Ping statistics: Sent
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[71] `f"Received = {ping_result['received']}, "`

ASSIGNATION (Affectation):
Stocke la valeur {ping_result['received']}, " dans la variable f"Received
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[72] `f"Lost = {ping_result['lost']} ({ping_result['loss_percent']}% loss)")`

ASSIGNATION (Affectation):
Stocke la valeur {ping_result['lost']} ({ping_result['loss_percent']}% loss)") dans la variable f"Lost
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[73] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[74] `status = 1 if ping_result["received"] > 0 else 0`

INSTRUCTION PYTHON:
status = 1 if ping_result["received"] > 0 else 0
Rôle: Exécute une opération ou logique du programme.

[75] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[76] `# Final values stored in DB`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Final values stored in DB"
Utilité: Documenter le code et expliquer les décisions de conception.

[77] `final_hostname = resolved_hostname if resolved_hostname else None`

ASSIGNATION (Affectation):
Stocke la valeur resolved_hostname if resolved_hostname else None dans la variable final_hostname
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[78] `final_ip = resolved_ip if resolved_ip else None`

ASSIGNATION (Affectation):
Stocke la valeur resolved_ip if resolved_ip else None dans la variable final_ip
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[79] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[80] `ping_data = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable ping_data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[81] `"hostname": final_hostname,`

INSTRUCTION PYTHON:
"hostname": final_hostname,
Rôle: Exécute une opération ou logique du programme.

[82] `"ipAddress": final_ip,`

INSTRUCTION PYTHON:
"ipAddress": final_ip,
Rôle: Exécute une opération ou logique du programme.

[83] `"status": status,`

INSTRUCTION PYTHON:
"status": status,
Rôle: Exécute une opération ou logique du programme.

[84] `"user_id": user_id,`

INSTRUCTION PYTHON:
"user_id": user_id,
Rôle: Exécute une opération ou logique du programme.

[85] `}`

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

[86] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[87] `# Marqueur JSON pour Symfony`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Marqueur JSON pour Symfony"
Utilité: Documenter le code et expliquer les décisions de conception.

[88] `print("\n@@@PINGJSON@@@")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n@@@PINGJSON@@@"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.

Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[89] print(json.dumps(ping_data))`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: json.dumps(ping_data
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[90] #print(json.dumps(ping_data))`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "print(json.dumps(ping_data))"
Utilité: Documenter le code et expliquer les décisions de conception.

`[91] # ■ IMPORTANT: Always INSERT a new ping row (never UPDATE)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "■ IMPORTANT: Always INSERT a new ping row (never UPDATE)"
Utilité: Documenter le code et expliquer les décisions de conception.

`[92] # with conn.cursor() as cur:`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "with conn.cursor() as cur:"
Utilité: Documenter le code et expliquer les décisions de conception.

`[93] # cur.execute("""`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "cur.execute("""
Utilité: Documenter le code et expliquer les décisions de conception.

`[94] # INSERT INTO ping (user_id, hostname, ip_address, status)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "INSERT INTO ping (user_id, hostname, ip_address, status)"
Utilité: Documenter le code et expliquer les décisions de conception.

`[95] # VALUES (%s, %s, %s, %s)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "VALUES (%s, %s, %s, %s)"
Utilité: Documenter le code et expliquer les décisions de conception.

`[96] # """, (`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: """, ("
Utilité: Documenter le code et expliquer les décisions de conception.

`[97] # user_id,`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "user_id,"
Utilité: Documenter le code et expliquer les décisions de conception.

`[98] # resolved_hostname,`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "resolved_hostname,"
Utilité: Documenter le code et expliquer les décisions de conception.

`[99] # resolved_ip,`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "resolved_ip,"
Utilité: Documenter le code et expliquer les décisions de conception.

`[100] # status`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "status"
Utilité: Documenter le code et expliquer les décisions de conception.

[101] `# ))`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "))"
Utilité: Documenter le code et expliquer les décisions de conception.

[102] `# conn.commit()`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "conn.commit()"
Utilité: Documenter le code et expliquer les décisions de conception.

[103] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[104] `# conn.close()`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "conn.close()"
Utilité: Documenter le code et expliquer les décisions de conception.

[105] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[106] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[107] `if __name__ == "__main__":`

CONDITION IF:
Teste si l'expression est VRAIE: __name__ == "__main__"
Flux: Si VRAI → exécute le bloc qui suit. Si FAUX → passe au elif/else.
Exemple réel: 'if conn.is_connected():' → vérifier si la connexion est active avant de l'utiliser.

[108] `main()`

APPEL DE FONCTION:
Exécute la fonction 'main' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de main, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

■ reconn/emailfound.py

```
[1] import sys, ssl, time, re, json
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): sys, ssl, time, re, json
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[2] import urllib.request, urllib.parse
```

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): urllib.request, urllib.parse
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

```
[3] from scripts.db.mysql_conn import get_connection
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT get_connection depuis le module scripts.db.mysql_conn
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[4] from bs4 import BeautifulSoup
```

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT BeautifulSoup depuis le module bs4
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

```
[5] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[6] ctx = ssl.create_default_context()
```

ASSIGNATION (Affectation):
Stocke la valeur ssl.create_default_context() dans la variable ctx
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[7] ctx.check_hostname = False
```

ASSIGNATION (Affectation):
Stocke la valeur False dans la variable ctx.check_hostname
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[8] ctx.verify_mode = ssl.CERT_NONE
```

ASSIGNATION (Affectation):
Stocke la valeur ssl.CERT_NONE dans la variable ctx.verify_mode
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[9] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[10] EMAIL_REGEX = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
```

ASSIGNATION (Affectation):
Stocke la valeur r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' dans la variable EMAIL_REGEX

Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[11] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[12] def extract_emails_from_html(html):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'extract_emails_from_html' réutilisable.
Paramètres: html
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[13] soup = BeautifulSoup(html, 'html.parser')
```

ASSIGNATION (Affectation):
Stocke la valeur BeautifulSoup(html, 'html.parser') dans la variable soup
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[14] emails = []
```

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable emails
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[15] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[16] text = " ".join(soup.get_text().split())
```

ASSIGNATION (Affectation):
Stocke la valeur " ".join(soup.get_text().split()) dans la variable text
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[17] emails += re.findall(EMAIL_REGEX, text)
```

ASSIGNATION (Affectation):
Stocke la valeur re.findall(EMAIL_REGEX, text) dans la variable emails +
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[18] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[19] for a in soup.find_all('a', href=True):
```

ASSIGNATION (Affectation):
Stocke la valeur True): dans la variable for a in soup.find_all('a', href
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[20] if a['href'].startswith('mailto:'):
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: 'mailto:'
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[21] email = a['href'][7:].split('?')[0]
```

ASSIGNATION (Affectation):
Stocke la valeur a['href'][7:].split('?')[0] dans la variable email
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[22] emails.append(email)
```

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: email
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[23] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[24] return list(set(emails))
```

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: set(emails
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[25] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[26] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[27] def main():
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'main' réutilisable.
Paramètres: aucun
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[28] start_time = time.time()
```

ASSIGNATION (Affectation):
Stocke la valeur time.time() dans la variable start_time
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[29] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[30] if len(sys.argv) < 3:
```

APPEL DE FONCTION:
Exécute la fonction 'len' avec les paramètres: sys.argv
Flux d'exécution: Saute à la définition de len, exécute son code, puis revient ici.

Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[31] `print("Usage : python emailfound.py ")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Usage : python emailfound.py "
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[32] `sys.exit(0)`

APPEL DE FONCTION:
Exécute la fonction 'exit' avec les paramètres: 0
Flux d'exécution: Saute à la définition de exit, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[33] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[34] `user_id, target = sys.argv[1], sys.argv[2]`

ASSIGNATION (Affectation):
Stocke la valeur sys.argv[1], sys.argv[2] dans la variable user_id, target
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[35] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[36] `conn = get_connection()`

ASSIGNATION (Affectation):
Stocke la valeur get_connection() dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[37] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[38] `# ================ RÉCUPÉRATION HOSTNAME + PING_ID =================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ RÉCUPÉRATION HOSTNAME + PING_ID ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[39] `with conn.cursor(dictionary=True) as cur:`

ASSIGNATION (Affectation):
Stocke la valeur True) as cur: dans la variable with conn.cursor(dictionary
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[40] `cur.execute(`

INSTRUCTION PYTHON:
cur.execute(
Rôle: Exécute une opération ou logique du programme.

[41] `'''SELECT id, hostname, ip_address FROM ping`

INSTRUCTION PYTHON:
'''SELECT id, hostname, ip_address FROM ping
Rôle: Exécute une opération ou logique du programme.

```
[42] WHERE user_id = %s
```

ASSIGNATION (Affectation):
Stocke la valeur %s dans la variable WHERE user_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[43] AND ip_address = %s
```

ASSIGNATION (Affectation):
Stocke la valeur %s dans la variable AND ip_address
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[44] AND hostname IS NOT NULL
```

INSTRUCTION PYTHON:
AND hostname IS NOT NULL
Rôle: Exécute une opération ou logique du programme.

```
[45] ORDER BY scan_at DESC
```

INSTRUCTION PYTHON:
ORDER BY scan_at DESC
Rôle: Exécute une opération ou logique du programme.

```
[46] LIMIT 1''',
```

INSTRUCTION PYTHON:
LIMIT 1''',
Rôle: Exécute une opération ou logique du programme.

```
[47] (user_id, target)
```

INSTRUCTION PYTHON:
(user_id, target)
Rôle: Exécute une opération ou logique du programme.

```
[48] )
```

DÉLIMITEUR DE FERMETURE:
Ferme une parenthèse ou un crochet ouvert précédemment.
Importance: Chaque caractère doit avoir son équivalent de fermeture pour que le code soit syntaxiquement correct.

```
[49] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[50] row = cur.fetchone()
```

ASSIGNATION (Affectation):
Stocke la valeur cur.fetchone() dans la variable row
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[51] if not row:
```

INSTRUCTION PYTHON:
if not row:
Rôle: Exécute une opération ou logique du programme.

```
[52] print("Aucun hostname trouvé")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Aucun hostname trouvé"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[53] return
```

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

```
[54] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[55] ping_id = row['id']
```

ASSIGNATION (Affectation):
Stocke la valeur row['id'] dans la variable ping_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[56] base_url = row.get('hostname') or row.get('ip_address')
```

ASSIGNATION (Affectation):
Stocke la valeur row.get('hostname') or row.get('ip_address') dans la variable base_url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[57] if not base_url.startswith(('http://', 'https://')):
```

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: ('http://', 'https://'
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[58] base_url = 'http://' + base_url
```

ASSIGNATION (Affectation):
Stocke la valeur 'http://' + base_url dans la variable base_url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[59] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[60] print('Retrieve emails -', base_url)
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: 'Retrieve emails -', base_url
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[61] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[62] # ================ PAGE PRINCIPALE ================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ PAGE PRINCIPALE ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[63] `try:`

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

[64] `response = urllib.request.urlopen(base_url, context=ctx, timeout=10)`

ASSIGNATION (Affectation):
Stocke la valeur urllib.request.urlopen(base_url, context=ctx, timeout=10) dans la variable response
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[65] `html = response.read()`

ASSIGNATION (Affectation):
Stocke la valeur response.read() dans la variable html
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[66] `except Exception as e:`

INSTRUCTION PYTHON:
except Exception as e:
Rôle: Exécute une opération ou logique du programme.

[67] `print("Erreur page principale :", e)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Erreur page principale :", e
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[68] `return`

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

[69] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[70] `all_emails = extract_emails_from_html(html)`

ASSIGNATION (Affectation):
Stocke la valeur extract_emails_from_html(html) dans la variable all_emails
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[71] `soup = BeautifulSoup(html, 'html.parser')`

ASSIGNATION (Affectation):
Stocke la valeur BeautifulSoup(html, 'html.parser') dans la variable soup
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[72] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[73] `# ================ RÉCUPÉRATION DES LIENS ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ RÉCUPÉRATION DES LIENS ================"

Utilité: Documenter le code et expliquer les décisions de conception.

`[74] linksFound = []`

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable linksFound
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[75] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[76] for a in soup.find_all('a', href=True):`

ASSIGNATION (Affectation):
Stocke la valeur True): dans la variable for a in soup.find_all('a', href
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[77] href = a['href'].strip()`

ASSIGNATION (Affectation):
Stocke la valeur a['href'].strip() dans la variable href
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[78] if not href:`

INSTRUCTION PYTHON:
if not href:
Rôle: Exécute une opération ou logique du programme.

`[79] continue`

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

`[80] if href.startswith(('mailto:', '#', 'javascript:')):`

APPEL DE FONCTION:
Exécute la fonction 'startswith' avec les paramètres: ('mailto:', '#', 'javascript:'
Flux d'exécution: Saute à la définition de startswith, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[81] continue`

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

`[82] full_url = urllib.parse.urljoin(base_url, href)`

ASSIGNATION (Affectation):
Stocke la valeur urllib.parse.urljoin(base_url, href) dans la variable full_url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[83] linksFound.append(full_url)`

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: full_url
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[84] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[85] `linksFound = list(dict.fromkeys(linksFound))`

ASSIGNATION (Affectation):
Stocke la valeur list(dict.fromkeys(linksFound)) dans la variable linksFound
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[86] `print("Liens trouvés :", len(linksFound))`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Liens trouvés :", len(linksFound
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[87] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[88] `# ================ SÉLECTION ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ SÉLECTION ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[89] `selected_links = linksFound[:10] + linksFound[-10:]`

ASSIGNATION (Affectation):
Stocke la valeur linksFound[:10] + linksFound[-10:] dans la variable selected_links
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[90] `selected_links = list(dict.fromkeys(selected_links))`

ASSIGNATION (Affectation):
Stocke la valeur list(dict.fromkeys(selected_links)) dans la variable selected_links
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[91] `print("Liens exploités :", len(selected_links))`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Liens exploités :", len(selected_links
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[92] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[93] `# ================ SCAN DES LIENS ================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================ SCAN DES LIENS ================"
Utilité: Documenter le code et expliquer les décisions de conception.

[94] `for url in selected_links:`

INSTRUCTION PYTHON:
for url in selected_links:
Rôle: Exécute une opération ou logique du programme.

[95] `print(" → Scan :", url)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " → Scan :", url
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[96] `try:`

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

[97] `resp = urllib.request.urlopen(url, context=ctx, timeout=10)`

ASSIGNATION (Affectation):
Stocke la valeur urllib.request.urlopen(url, context=ctx, timeout=10) dans la variable resp
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[98] `page_html = resp.read()`

ASSIGNATION (Affectation):
Stocke la valeur resp.read() dans la variable page_html
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[99] `except Exception:`

INSTRUCTION PYTHON:
except Exception:
Rôle: Exécute une opération ou logique du programme.

[100] `continue`

INSTRUCTION PYTHON:
continue
Rôle: Exécute une opération ou logique du programme.

[101] `all_emails += extract_emails_from_html(page_html)`

ASSIGNATION (Affectation):
Stocke la valeur extract_emails_from_html(page_html) dans la variable all_emails +
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[102] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[103] `# ================= RÉSULTATS =================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================= RÉSULTATS ================="
Utilité: Documenter le code et expliquer les décisions de conception.

[104] `all_emails = list(set(all_emails))`

ASSIGNATION (Affectation):
Stocke la valeur list(set(all_emails)) dans la variable all_emails
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[105] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

`[106] users_found = set()`

ASSIGNATION (Affectation):
Stocke la valeur set() dans la variable users_found
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[107] for email in all_emails:`

INSTRUCTION PYTHON:
for email in all_emails:
Rôle: Exécute une opération ou logique du programme.

`[108] if "@" in email:`

INSTRUCTION PYTHON:
if "@" in email:
Rôle: Exécute une opération ou logique du programme.

`[109] users_found.add(email.split("@")[0].lower())`

APPEL DE FONCTION:
Exécute la fonction 'add' avec les paramètres: email.split("@"
Flux d'exécution: Saute à la définition de add, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[110] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[111] elapsed = time.time() - start_time`

ASSIGNATION (Affectation):
Stocke la valeur time.time() - start_time dans la variable elapsed
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[112] print(f"\nTemps d'exécution : {elapsed:.2f} secondes")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"\nTemps d'exécution : {elapsed:.2f} secondes"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[113] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[114] print("\nEmails trouvés (total) :")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\nEmails trouvés (total
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[115] for email in all_emails:`

INSTRUCTION PYTHON:
for email in all_emails:
Rôle: Exécute une opération ou logique du programme.

`[116] print(" -", email)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " -", email

Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[117] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[118] `print('\nUser found from emails - ')`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: '\nUser found from emails - '
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[119] `for user in users_found:`

INSTRUCTION PYTHON:
for user in users_found:
Rôle: Exécute une opération ou logique du programme.

[120] `print(' -', user)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: ' -', user
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[121] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[122] `all_links = set(linksFound + selected_links)`

ASSIGNATION (Affectation):
Stocke la valeur set(linksFound + selected_links) dans la variable all_links
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[123] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[124] `# ================= JSON POUR SYMFONY =================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "================= JSON POUR SYMFONY ================="
Utilité: Documenter le code et expliquer les décisions de conception.

[125] `reconn_data = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable reconn_data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[126] `"ping_id": ping_id,`

INSTRUCTION PYTHON:
"ping_id": ping_id,
Rôle: Exécute une opération ou logique du programme.

[127] `"emails": list(all_emails),`

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: all_emails
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[128] "users": list(users_found),
```

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: users_found
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[129] "links": list(all_links)
```

APPEL DE FONCTION:
Exécute la fonction 'list' avec les paramètres: all_links
Flux d'exécution: Saute à la définition de list, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[130] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[131] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[132] print("\n@@@RECONNJSON@@@")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\n@@@RECONNJSON@@@"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[133] print(json.dumps(reconn_data))
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: json.dumps(reconn_data
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[134] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[135] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[136] if __name__ == "__main__":
```

CONDITION IF:
Teste si l'expression est VRAIE: __name__ == "__main__"
Flux: Si VRAI → exécute le bloc qui suit. Si FAUX → passe au elif/else.
Exemple réel: 'if conn.is_connected():' → vérifier si la connexion est active avant de l'utiliser.

```
[137] main()
```

APPEL DE FONCTION:
Exécute la fonction 'main' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de main, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

■ scanner/scanner.py

`[1] import sys, socket, ssl, time, json, re, urllib.request, urllib.parse, warnings`

IMPORTATION DE MODULE(S):
Charge le(s) module(s) suivant(s): sys, socket, ssl, time, json, re, urllib.request, urllib.parse, warnings
Pourquoi: Accéder à des fonctionnalités pré-codées plutôt que de les développer from scratch.
Exemple réel du projet: 'import sys' pour accéder aux arguments de ligne de commande (sys.argv).

`[2] from concurrent.futures import ThreadPoolExecutor`

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT ThreadPoolExecutor depuis le module concurrent.futures
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

`[3] from scripts.db.mysql_conn import get_connection`

IMPORTATION SÉLECTIVE:
Importe UNIQUEMENT get_connection depuis le module scripts.db.mysql_conn
Avantage: Réduit l'usage mémoire et rend claires les dépendances exactes.
Exemple du projet: 'from scripts.db.mysql_conn import get_connection' → importe que la fonction de connexion MySQL, pas tout le module.

`[4] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[5] sys.stdout.reconfigure(line_buffering=True)`

ASSIGNATION (Affectation):
Stocke la valeur True) dans la variable sys.stdout.reconfigure(line_buffering
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[6] warnings.filterwarnings("ignore", message="Unverified HTTPS request")`

ASSIGNATION (Affectation):
Stocke la valeur "Unverified HTTPS request") dans la variable warnings.filterwarnings("ignore", message
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[7] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[8] # ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

`[9] # SSL context`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "SSL context"
Utilité: Documenter le code et expliquer les décisions de conception.

`[10] # ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

`[11] ctx = ssl.create_default_context()`

ASSIGNATION (Affectation):
Stocke la valeur ssl.create_default_context() dans la variable ctx
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[12] `ctx.check_hostname = False`

ASSIGNATION (Affectation):
Stocke la valeur False dans la variable ctx.check_hostname
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[13] `ctx.verify_mode = ssl.CERT_NONE`

ASSIGNATION (Affectation):
Stocke la valeur ssl.CERT_NONE dans la variable ctx.verify_mode
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[14] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[15] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

[16] `# PORT FALLBACK MAP`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "PORT FALLBACK MAP"
Utilité: Documenter le code et expliquer les décisions de conception.

[17] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

[18] `PORT_SERVICE_MAP = {`

ASSIGNATION (Affectation):
Stocke la valeur { dans la variable PORT_SERVICE_MAP
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[19] `53: "DNS",`

INSTRUCTION PYTHON:
53: "DNS",
Rôle: Exécute une opération ou logique du programme.

[20] `88: "Kerberos",`

INSTRUCTION PYTHON:
88: "Kerberos",
Rôle: Exécute une opération ou logique du programme.

[21] `135: "MSRPC",`

INSTRUCTION PYTHON:
135: "MSRPC",
Rôle: Exécute une opération ou logique du programme.

[22] `139: "SMB",`

INSTRUCTION PYTHON:
139: "SMB",
Rôle: Exécute une opération ou logique du programme.

```
[23] 389: "LDAP",
```

INSTRUCTION PYTHON:
389: "LDAP",
Rôle: Exécute une opération ou logique du programme.

```
[24] 445: "SMB",
```

INSTRUCTION PYTHON:
445: "SMB",
Rôle: Exécute une opération ou logique du programme.

```
[25] 464: "Kerberos",
```

INSTRUCTION PYTHON:
464: "Kerberos",
Rôle: Exécute une opération ou logique du programme.

```
[26] 593: "RPC over HTTP",
```

INSTRUCTION PYTHON:
593: "RPC over HTTP",
Rôle: Exécute une opération ou logique du programme.

```
[27] 636: "LDAPS",
```

INSTRUCTION PYTHON:
636: "LDAPS",
Rôle: Exécute une opération ou logique du programme.

```
[28] 2179: "Hyper-V",
```

INSTRUCTION PYTHON:
2179: "Hyper-V",
Rôle: Exécute une opération ou logique du programme.

```
[29] 3268: "LDAP GC",
```

INSTRUCTION PYTHON:
3268: "LDAP GC",
Rôle: Exécute une opération ou logique du programme.

```
[30] 3269: "LDAPS GC",
```

INSTRUCTION PYTHON:
3269: "LDAPS GC",
Rôle: Exécute une opération ou logique du programme.

```
[31] 3389: "RDP",
```

INSTRUCTION PYTHON:
3389: "RDP",
Rôle: Exécute une opération ou logique du programme.

```
[32] 9389: "AD Web Services"
```

INSTRUCTION PYTHON:
9389: "AD Web Services"
Rôle: Exécute une opération ou logique du programme.

```
[33] }
```

ACCOLADE DE FERMETURE:
Marque la fin d'une structure de données (dictionnaire, ensemble, etc.) ou d'un bloc de code formaté.
Signification: Fin de la structure commencée par une accolade ouvrante.

```
[34] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[35] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

[36] `# OS detection (ports-based)`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "OS detection (ports-based)"
Utilité: Documenter le code et expliquer les décisions de conception.

[37] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

[38] `def os_ports_fingerprint(open_ports):`

DÉFINITION DE FONCTION:
Crée une fonction nommée 'os_ports_fingerprint' réutilisable.
Paramètres: open_ports
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

[39] `windows_ports = {135, 139, 445, 3389, 5985}`

ASSIGNATION (Affectation):
Stocke la valeur {135, 139, 445, 3389, 5985} dans la variable windows_ports
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[40] `if windows_ports.intersection(open_ports):`

APPEL DE FONCTION:
Exécute la fonction 'intersection' avec les paramètres: open_ports
Flux d'exécution: Saute à la définition de intersection, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[41] `return "Windows (system ports detected)"`

APPEL DE FONCTION:
Exécute la fonction 'Windows' avec les paramètres: system ports detected
Flux d'exécution: Saute à la définition de Windows, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[42] `return "Inconnu"`

INSTRUCTION PYTHON:
return "Inconnu"
Rôle: Exécute une opération ou logique du programme.

[43] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[44] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

[45] `# Banner grabber`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Banner grabber"
Utilité: Documenter le code et expliquer les décisions de conception.

[46] `# ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[47] def grab_banner(host, port):
```

DÉFINITION DE FONCTION:
Crée une fonction nommée 'grab_banner' réutilisable.
Paramètres: host, port
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

```
[48] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[49] sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

ASSIGNATION (Affectation):
Stocke la valeur socket.socket(socket.AF_INET, socket.SOCK_STREAM) dans la variable sock
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[50] sock.settimeout(3)
```

APPEL DE FONCTION:
Exécute la fonction 'settimeout' avec les paramètres: 3
Flux d'exécution: Saute à la définition de settimeout, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[51] sock.connect((host, port))
```

APPEL DE FONCTION:
Exécute la fonction 'connect' avec les paramètres: (host, port
Flux d'exécution: Saute à la définition de connect, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[52] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[53] data = sock.recv(2048)
```

ASSIGNATION (Affectation):
Stocke la valeur sock.recv(2048) dans la variable data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[54] except:
```

INSTRUCTION PYTHON:
except:
Rôle: Exécute une opération ou logique du programme.

```
[55] sock.send(b"HEAD / HTTP/1.0\r\n\r\n")
```

APPEL DE FONCTION:
Exécute la fonction 'send' avec les paramètres: b"HEAD / HTTP/1.0\r\n\r\n"
Flux d'exécution: Saute à la définition de send, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[56] data = sock.recv(2048)
```

ASSIGNATION (Affectation):
Stocke la valeur sock.recv(2048) dans la variable data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[57] sock.close()`

APPEL DE FONCTION:
Exécute la fonction 'close' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de close, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[58] return data.decode(errors="ignore")`

ASSIGNATION (Affectation):
Stocke la valeur "ignore") dans la variable return data.decode(errors
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[59] except:`

INSTRUCTION PYTHON:
except:
Rôle: Exécute une opération ou logique du programme.

`[60] return ""`

INSTRUCTION PYTHON:
return ""
Rôle: Exécute une opération ou logique du programme.

`[61] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[62] # ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

`[63] # MAIN`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "MAIN"
Utilité: Documenter le code et expliquer les décisions de conception.

`[64] # ============================`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

`[65] def main():`

DÉFINITION DE FONCTION:
Crée une fonction nommée 'main' réutilisable.
Paramètres: aucun
Objectif: Encapsuler une logique pour l'appeler plusieurs fois.
Exemple réel: 'def get_connection():' → définit la fonction qui établit la connexion MySQL.

`[66] start_time = time.time()`

ASSIGNATION (Affectation):
Stocke la valeur time.time() dans la variable start_time
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[67] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[68] if len(sys.argv) < 3:
```

APPEL DE FONCTION:
Exécute la fonction 'len' avec les paramètres: sys.argv
Flux d'exécution: Saute à la définition de len, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[69] print("Usage : python scanner.py ")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "Usage : python scanner.py "
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[70] return
```

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

```
[71] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[72] user_id, target = sys.argv[1], sys.argv[2]
```

ASSIGNATION (Affectation):
Stocke la valeur sys.argv[1], sys.argv[2] dans la variable user_id, target
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[73] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[74] conn = get_connection()
```

ASSIGNATION (Affectation):
Stocke la valeur get_connection() dans la variable conn
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[75] cur = conn.cursor(dictionary=True)
```

ASSIGNATION (Affectation):
Stocke la valeur conn.cursor(dictionary=True) dans la variable cur
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[76] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[77] cur.execute("""
```

INSTRUCTION PYTHON:
cur.execute("""
Rôle: Exécute une opération ou logique du programme.

```
[78] SELECT id FROM ping
```

INSTRUCTION PYTHON:
SELECT id FROM ping
Rôle: Exécute une opération ou logique du programme.

`[79] WHERE user_id=%s AND ip_address=%s`

ASSIGNATION (Affectation):
Stocke la valeur %s AND ip_address=%s dans la variable WHERE user_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[80] ORDER BY scan_at DESC`

INSTRUCTION PYTHON:
ORDER BY scan_at DESC
Rôle: Exécute une opération ou logique du programme.

`[81] LIMIT 1`

INSTRUCTION PYTHON:
LIMIT 1
Rôle: Exécute une opération ou logique du programme.

`[82] """, (user_id, target))`

INSTRUCTION PYTHON:
""", (user_id, target))
Rôle: Exécute une opération ou logique du programme.

`[83] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[84] row = cur.fetchone()`

ASSIGNATION (Affectation):
Stocke la valeur cur.fetchone() dans la variable row
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[85] if not row:`

INSTRUCTION PYTHON:
if not row:
Rôle: Exécute une opération ou logique du programme.

`[86] print("■ Aucun ping trouvé")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "■ Aucun ping trouvé"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[87] return`

INSTRUCTION PYTHON:
return
Rôle: Exécute une opération ou logique du programme.

`[88] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[89] ping_id = row["id"]`

ASSIGNATION (Affectation):
Stocke la valeur row["id"] dans la variable ping_id

Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[90] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[91] `print(f"\n==== Scan de la cible {target} ======\n")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"\n==== Scan de la cible {target} ======\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[92] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[93] `# ----------------------------`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "----------------------------"
Utilité: Documenter le code et expliquer les décisions de conception.

[94] `# Port scan`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Port scan"
Utilité: Documenter le code et expliquer les décisions de conception.

[95] `# ----------------------------`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "----------------------------"
Utilité: Documenter le code et expliquer les décisions de conception.

[96] `open_ports = []`

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable open_ports
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[97] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[98] `def scan_port(port):`

INSTRUCTION PYTHON:
def scan_port(port):
Rôle: Exécute une opération ou logique du programme.

[99] `try:`

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

[100] `s = socket.socket()`

ASSIGNATION (Affectation):
Stocke la valeur socket.socket() dans la variable s
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[101] `s.settimeout(2)`

APPEL DE FONCTION:
Exécute la fonction 'settimeout' avec les paramètres: 2
Flux d'exécution: Saute à la définition de settimeout, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[102] `if s.connect_ex((target, port)) == 0:`

APPEL DE FONCTION:
Exécute la fonction 'connect_ex' avec les paramètres: (target, port
Flux d'exécution: Saute à la définition de connect_ex, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[103] `print(f"[+] Port ouvert : {port}")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"[+] Port ouvert : {port}"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[104] `return port`

INSTRUCTION PYTHON:
return port
Rôle: Exécute une opération ou logique du programme.

[105] `s.close()`

APPEL DE FONCTION:
Exécute la fonction 'close' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de close, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[106] `except:`

INSTRUCTION PYTHON:
except:
Rôle: Exécute une opération ou logique du programme.

[107] `pass`

INSTRUCTION PYTHON:
pass
Rôle: Exécute une opération ou logique du programme.

[108] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[109] `with ThreadPoolExecutor(max_workers=200) as exe:`

ASSIGNATION (Affectation):
Stocke la valeur 200) as exe: dans la variable with ThreadPoolExecutor(max_workers
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[110] `for p in exe.map(scan_port, range(1, 10001)):`

APPEL DE FONCTION:
Exécute la fonction 'map' avec les paramètres: scan_port, range(1, 10001
Flux d'exécution: Saute à la définition de map, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[111] `if p:`

INSTRUCTION PYTHON:
if p:
Rôle: Exécute une opération ou logique du programme.

[112] `open_ports.append(p)`

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: p
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[113] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[114] `print("\nPorts ouverts :", open_ports)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "\nPorts ouverts :", open_ports
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[115] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[116] `os_guess = os_ports_fingerprint(open_ports)`

ASSIGNATION (Affectation):
Stocke la valeur os_ports_fingerprint(open_ports) dans la variable os_guess
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[117] `print(f"\n====== Détection du système ======")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"\n====== Détection du système ======"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[118] `print(f" OS probable : {os_guess}\n")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f" OS probable : {os_guess}\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[119] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[120] `# ----------------------------`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "----------"
Utilité: Documenter le code et expliquer les décisions de conception.

[121] `# Services`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "Services"
Utilité: Documenter le code et expliquer les décisions de conception.

[122] `# ----------------------------`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "--------------------------"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[123] print("======= Analyse des services =======\n")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "======= Analyse des services =======\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[124] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[125] for port in open_ports:
```

INSTRUCTION PYTHON:
for port in open_ports:
Rôle: Exécute une opération ou logique du programme.

```
[126] print(f"[PORT {port}]")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"[PORT {port}]"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[127] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[128] banner = grab_banner(target, port)
```

ASSIGNATION (Affectation):
Stocke la valeur grab_banner(target, port) dans la variable banner
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[129] product = "unknown"
```

ASSIGNATION (Affectation):
Stocke la valeur "unknown" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[130] version = None
```

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[131] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[132] if banner:
```

INSTRUCTION PYTHON:
if banner:
Rôle: Exécute une opération ou logique du programme.

```
[133] print(" Service :", banner.split("\n")[0])
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " Service :", banner.split("\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[134] b = banner.lower()
```

ASSIGNATION (Affectation):
Stocke la valeur banner.lower() dans la variable b
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[135] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[136] # OpenSSH Windows FIX
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "OpenSSH Windows FIX"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[137] if "openssh" in b:
```

INSTRUCTION PYTHON:
if "openssh" in b:
Rôle: Exécute une opération ou logique du programme.

```
[138] product = "OpenSSH"
```

ASSIGNATION (Affectation):
Stocke la valeur "OpenSSH" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[139] m = re.search(r"OpenSSH[_a-zA-Z\-]*([0-9]+\.[0-9]+)", banner)
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"OpenSSH[_a-zA-Z\-]*([0-9]+\.[0-9]+)", banner) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[140] if m:
```

INSTRUCTION PYTHON:
if m:
Rôle: Exécute une opération ou logique du programme.

```
[141] version = m.group(1)
```

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[142] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[143] # FTP
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "FTP"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[144] if port == 21:
```

INSTRUCTION PYTHON:
if port == 21:
Rôle: Exécute une opération ou logique du programme.

```
[145] if "pure-ftpd" in b:
```

INSTRUCTION PYTHON:
if "pure-ftpd" in b:
Rôle: Exécute une opération ou logique du programme.

```
[146] product = "Pure-FTPd"
```

ASSIGNATION (Affectation):
Stocke la valeur "Pure-FTPd" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[147] version = None
```

ASSIGNATION (Affectation):
Stocke la valeur None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[148] elif "filezilla server" in b:
```

INSTRUCTION PYTHON:
elif "filezilla server" in b:
Rôle: Exécute une opération ou logique du programme.

```
[149] product = "FileZilla Server"
```

ASSIGNATION (Affectation):
Stocke la valeur "FileZilla Server" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[150] m = re.search(r"FileZilla Server\s*([0-9\.]+(?:\s*beta)?)", banner, re.I)
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"FileZilla Server\s*([0-9\.]+(?:\s*beta)?)", banner, re.I) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[151] version = m.group(1) if m else None
```

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) if m else None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[152] elif "vsftpd" in b :
```

INSTRUCTION PYTHON:
elif "vsftpd" in b :
Rôle: Exécute une opération ou logique du programme.

```
[153] product = "vsptd"
```

ASSIGNATION (Affectation):
Stocke la valeur "vsptd" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[154] `m = re.search(r"vsftpd\s*([0-9\.]+)", banner, re.I)`

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"vsftpd\s*([0-9\.]+)", banner, re.I) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[155] `version = m.group(1) if m else None`

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) if m else None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[156] `elif "proftpd" in b :`

INSTRUCTION PYTHON:
elif "proftpd" in b :
Rôle: Exécute une opération ou logique du programme.

[157] `product = "ProFTPD"`

ASSIGNATION (Affectation):
Stocke la valeur "ProFTPD" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[158] `m = re.search(r"ProFTPD\s*([0-9\.]+)", banner, re.I)`

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"ProFTPD\s*([0-9\.]+)", banner, re.I) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[159] `version = m.group(1) if m else None`

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) if m else None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[160] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[161] `# SMTP`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "SMTP"
Utilité: Documenter le code et expliquer les décisions de conception.

[162] `if port in (25, 26, 465, 587):`

APPEL DE FONCTION:
Exécute la fonction 'in' avec les paramètres: 25, 26, 465, 587
Flux d'exécution: Saute à la définition de in, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[163] `if "exim" in b:`

INSTRUCTION PYTHON:
if "exim" in b:
Rôle: Exécute une opération ou logique du programme.

[164] `product = "Exim"`

ASSIGNATION (Affectation):
Stocke la valeur "Exim" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[165] m = re.search(r"Exim\s+([0-9\.]+)", banner, re.I)
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"Exim\s+([0-9\.]+)", banner, re.I) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[166] version = m.group(1) if m else None
```

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) if m else None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[167] # HTTP
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "HTTP"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[168] m = re.search(r"Server:\s*([^\r\n]+)", banner, re.I)
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"Server:\s*([^\r\n]+)", banner, re.I) dans la variable m
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[169] if m and product == "unknown":
```

INSTRUCTION PYTHON:
if m and product == "unknown":
Rôle: Exécute une opération ou logique du programme.

```
[170] server_line = m.group(1)
```

ASSIGNATION (Affectation):
Stocke la valeur m.group(1) dans la variable server_line
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[171] m2 = re.search(r"([A-Za-z\-]+)[/ ]([0-9\.]+)", server_line)
```

ASSIGNATION (Affectation):
Stocke la valeur re.search(r"([A-Za-z\-]+)[/]([0-9\.]+)", server_line) dans la variable m2
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[172] product = m2.group(1) if m2 else server_line
```

ASSIGNATION (Affectation):
Stocke la valeur m2.group(1) if m2 else server_line dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[173] version = m2.group(2) if m2 else None
```

ASSIGNATION (Affectation):
Stocke la valeur m2.group(2) if m2 else None dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[174] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[175] `# MySQL`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "MySQL"
Utilité: Documenter le code et expliquer les décisions de conception.

[176] `if port == 3306:`

INSTRUCTION PYTHON:
if port == 3306:
Rôle: Exécute une opération ou logique du programme.

[177] `product = "MySQL"`

ASSIGNATION (Affectation):
Stocke la valeur "MySQL" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[178] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[179] `# SMB`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "SMB"
Utilité: Documenter le code et expliquer les décisions de conception.

[180] `if port in (139, 445):`

APPEL DE FONCTION:
Exécute la fonction 'in' avec les paramètres: 139, 445
Flux d'exécution: Saute à la définition de in, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[181] `product = "SMB"`

ASSIGNATION (Affectation):
Stocke la valeur "SMB" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[182] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[183] `# FTPS`

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "FTPS"
Utilité: Documenter le code et expliquer les décisions de conception.

[184] `if port == 990:`

INSTRUCTION PYTHON:
if port == 990:

Rôle: Exécute une opération ou logique du programme.

```
[185] product = "FTPS"
```

ASSIGNATION (Affectation):
Stocke la valeur "FTPS" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[186] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[187] # WinRM
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "WinRM"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[188] if port == 5985:
```

INSTRUCTION PYTHON:
if port == 5985:
Rôle: Exécute une opération ou logique du programme.

```
[189] product = "WinRM"
```

ASSIGNATION (Affectation):
Stocke la valeur "WinRM" dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[190] else:
```

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

```
[191] product = PORT_SERVICE_MAP.get(port, "unknown")
```

ASSIGNATION (Affectation):
Stocke la valeur PORT_SERVICE_MAP.get(port, "unknown") dans la variable product
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[192] print(" Service : Inconnu (no banner)")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " Service : Inconnu (no banner
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[193] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[194] print(f" → Détecté : {product} {version}")
```

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f" → Détecté : {product} {version}"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[195] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[196] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[197] # CVE lookup
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "CVE lookup"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[198] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[199] vuln_list = []
```

ASSIGNATION (Affectation):
Stocke la valeur [] dans la variable vuln_list
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[200] bad = ["http","https","ftp","smtp","imap","pop","tcp","udp","rtsp"]
```

ASSIGNATION (Affectation):
Stocke la valeur ["http","https","ftp","smtp","imap","pop","tcp","udp","rtsp"] dans la variable bad
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[201] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[202] if product and version:
```

INSTRUCTION PYTHON:
if product and version:
Rôle: Exécute une opération ou logique du programme.

```
[203] if product.lower() not in bad and re.search(r"[0-9]+\.[0-9]+", str(version)):
```

APPEL DE FONCTION:
Exécute la fonction 'lower' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de lower, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[204] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[205] query = f"{product} {version}"
```

ASSIGNATION (Affectation):
Stocke la valeur f"{product} {version}" dans la variable query
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[206] url = "https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSearch=" + urllib
```

ASSIGNATION (Affectation):
Stocke la valeur "https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSearch=" + urllib.parse.quote(query) dans la variable url
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[207] resp = urllib.request.urlopen(url, timeout=15, context=ctx)
```

ASSIGNATION (Affectation):
Stocke la valeur urllib.request.urlopen(url, timeout=15, context=ctx) dans la variable resp
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[208] data = json.loads(resp.read().decode())
```

ASSIGNATION (Affectation):
Stocke la valeur json.loads(resp.read().decode()) dans la variable data
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[209] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[210] for v in data.get("vulnerabilities", []):
```

APPEL DE FONCTION:
Exécute la fonction 'get' avec les paramètres: "vulnerabilities", []
Flux d'exécution: Saute à la définition de get, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

```
[211] cve = v.get("cve", {})
```

ASSIGNATION (Affectation):
Stocke la valeur v.get("cve", {}) dans la variable cve
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[212] metrics = cve.get("metrics", {})
```

ASSIGNATION (Affectation):
Stocke la valeur cve.get("metrics", {}) dans la variable metrics
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[213] cvss = 0
```

ASSIGNATION (Affectation):
Stocke la valeur 0 dans la variable cvss
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[214] if "cvssMetricV31" in metrics:
```

INSTRUCTION PYTHON:
if "cvssMetricV31" in metrics:
Rôle: Exécute une opération ou logique du programme.

```
[215] cvss = metrics["cvssMetricV31"][0]["cvssData"].get("baseScore", 0)
```

ASSIGNATION (Affectation):
Stocke la valeur metrics["cvssMetricV31"][0]["cvssData"].get("baseScore", 0) dans la variable cvss
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.

Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[216] elif "cvssMetricV30" in metrics:`

INSTRUCTION PYTHON:
elif "cvssMetricV30" in metrics:
Rôle: Exécute une opération ou logique du programme.

`[217] cvss = metrics["cvssMetricV30"][0]["cvssData"].get("baseScore", 0)`

ASSIGNATION (Affectation):
Stocke la valeur metrics["cvssMetricV30"][0]["cvssData"].get("baseScore", 0) dans la variable cvss
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

`[218] vuln_list.append({"id": cve.get("id"), "cvss": cvss})`

APPEL DE FONCTION:
Exécute la fonction 'append' avec les paramètres: {"id": cve.get("id"
Flux d'exécution: Saute à la définition de append, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[219] except:`

INSTRUCTION PYTHON:
except:
Rôle: Exécute une opération ou logique du programme.

`[220] pass`

INSTRUCTION PYTHON:
pass
Rôle: Exécute une opération ou logique du programme.

`[221] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

`[222] if vuln_list:`

INSTRUCTION PYTHON:
if vuln_list:
Rôle: Exécute une opération ou logique du programme.

`[223] print(" ■■ CVE trouvées :", vuln_list)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " ■■ CVE trouvées :", vuln_list
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[224] else:`

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

`[225] print(" ✓ Aucune CVE connue")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: " ✓ Aucune CVE connue"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

`[226] [LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus

lisibles.

```
[227] vuln_str = ", ".join(v["id"] for v in vuln_list) if vuln_list else None
```

ASSIGNATION (Affectation):
Stocke la valeur ", ".join(v["id"] for v in vuln_list) if vuln_list else None dans la variable vuln_str
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[228] [LIGNE VIDE]
```

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

```
[229] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[230] # DB INSERT / UPDATE
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "DB INSERT / UPDATE"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[231] # ============================
```

COMMENTAIRE EXPLICATIF:
Texte informatif destiné aux développeurs: "============================"
Utilité: Documenter le code et expliquer les décisions de conception.

```
[232] try:
```

INSTRUCTION PYTHON:
try:
Rôle: Exécute une opération ou logique du programme.

```
[233] cur.execute(
```

INSTRUCTION PYTHON:
cur.execute(
Rôle: Exécute une opération ou logique du programme.

```
[234] "SELECT id FROM scanner WHERE ping_id=%s AND port=%s",
```

ASSIGNATION (Affectation):
Stocke la valeur %s AND port=%s", dans la variable "SELECT id FROM scanner WHERE ping_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

```
[235] (ping_id, port)
```

INSTRUCTION PYTHON:
(ping_id, port)
Rôle: Exécute une opération ou logique du programme.

```
[236] )
```

DÉLIMITEUR DE FERMETURE:
Ferme une parenthèse ou un crochet ouvert précédemment.
Importance: Chaque caractère doit avoir son équivalent de fermeture pour que le code soit syntaxiquement correct.

```
[237] row = cur.fetchone()
```

ASSIGNATION (Affectation):
Stocke la valeur cur.fetchone() dans la variable row
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[238] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[239] `if row:`

INSTRUCTION PYTHON:
if row:
Rôle: Exécute une opération ou logique du programme.

[240] `cur.execute("""`

INSTRUCTION PYTHON:
cur.execute("""
Rôle: Exécute une opération ou logique du programme.

[241] `UPDATE scanner SET`

INSTRUCTION PYTHON:
UPDATE scanner SET
Rôle: Exécute une opération ou logique du programme.

[242] `service=%s,`

ASSIGNATION (Affectation):
Stocke la valeur %s, dans la variable service
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[243] `version=%s,`

ASSIGNATION (Affectation):
Stocke la valeur %s, dans la variable version
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[244] `script_vuln=%s,`

ASSIGNATION (Affectation):
Stocke la valeur %s, dans la variable script_vuln
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[245] `state='open',`

ASSIGNATION (Affectation):
Stocke la valeur 'open', dans la variable state
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[246] `os_detected=%s,`

ASSIGNATION (Affectation):
Stocke la valeur %s, dans la variable os_detected
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[247] `description=%s`

ASSIGNATION (Affectation):
Stocke la valeur %s dans la variable description
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[248] `WHERE ping_id=%s AND port=%s`

ASSIGNATION (Affectation):
Stocke la valeur %s AND port=%s dans la variable WHERE ping_id
Pourquoi: Mémoriser une valeur pour l'utiliser plus tard.
Exemple réel: 'user_id = sys.argv[1]' → récupérer l'argument du script et le garder en mémoire pour le reste du programme.

[249] `""", (`

INSTRUCTION PYTHON:
""", (
Rôle: Exécute une opération ou logique du programme.

[250] `product,`

INSTRUCTION PYTHON:
product,
Rôle: Exécute une opération ou logique du programme.

[251] `version,`

INSTRUCTION PYTHON:
version,
Rôle: Exécute une opération ou logique du programme.

[252] `vuln_str,`

INSTRUCTION PYTHON:
vuln_str,
Rôle: Exécute une opération ou logique du programme.

[253] `os_guess,`

INSTRUCTION PYTHON:
os_guess,
Rôle: Exécute une opération ou logique du programme.

[254] `banner[:1000],`

INSTRUCTION PYTHON:
banner[:1000],
Rôle: Exécute une opération ou logique du programme.

[255] `ping_id,`

INSTRUCTION PYTHON:
ping_id,
Rôle: Exécute une opération ou logique du programme.

[256] `port`

INSTRUCTION PYTHON:
port
Rôle: Exécute une opération ou logique du programme.

[257] `))`

INSTRUCTION PYTHON:
))
Rôle: Exécute une opération ou logique du programme.

[258] `else:`

INSTRUCTION PYTHON:
else:
Rôle: Exécute une opération ou logique du programme.

[259] `cur.execute("""`

INSTRUCTION PYTHON:
cur.execute("""
Rôle: Exécute une opération ou logique du programme.

[260] `INSERT INTO scanner`

INSTRUCTION PYTHON:
INSERT INTO scanner
Rôle: Exécute une opération ou logique du programme.

```
[261] (port, service, version, script_vuln, state, os_detected, ping_id, descripti
```

INSTRUCTION PYTHON:
(port, service, version, script_vuln, state, os_detected, ping_id, descripti
Rôle: Exécute une opération ou logique du programme.

```
[262] VALUES
```

INSTRUCTION PYTHON:
VALUES
Rôle: Exécute une opération ou logique du programme.

```
[263] (%s,%s,%s,%s,'open',%s,%s,%s)
```

INSTRUCTION PYTHON:
(%s,%s,%s,%s,'open',%s,%s,%s)
Rôle: Exécute une opération ou logique du programme.

```
[264] """, (
```

INSTRUCTION PYTHON:
""", (
Rôle: Exécute une opération ou logique du programme.

```
[265] port,
```

INSTRUCTION PYTHON:
port,
Rôle: Exécute une opération ou logique du programme.

```
[266] product,
```

INSTRUCTION PYTHON:
product,
Rôle: Exécute une opération ou logique du programme.

```
[267] version,
```

INSTRUCTION PYTHON:
version,
Rôle: Exécute une opération ou logique du programme.

```
[268] vuln_str,
```

INSTRUCTION PYTHON:
vuln_str,
Rôle: Exécute une opération ou logique du programme.

```
[269] os_guess,
```

INSTRUCTION PYTHON:
os_guess,
Rôle: Exécute une opération ou logique du programme.

```
[270] ping_id,
```

INSTRUCTION PYTHON:
ping_id,
Rôle: Exécute une opération ou logique du programme.

```
[271] banner[:1000]
```

INSTRUCTION PYTHON:
banner[:1000]
Rôle: Exécute une opération ou logique du programme.

```
[272] ))
```

INSTRUCTION PYTHON:
))
Rôle: Exécute une opération ou logique du programme.

[273] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[274] `conn.commit()`

APPEL DE FONCTION:
Exécute la fonction 'commit' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de commit, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[275] `print("■ Enregistré\n")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "■ Enregistré\n"
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[276] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[277] `except Exception as e:`

INSTRUCTION PYTHON:
except Exception as e:
Rôle: Exécute une opération ou logique du programme.

[278] `conn.rollback()`

APPEL DE FONCTION:
Exécute la fonction 'rollback' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de rollback, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[279] `print("■ DB:", e)`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: "■ DB:", e
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[280] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[281] `print(f"\nTemps d'exécution : {time.time() - start_time:.2f} secondes\n")`

APPEL DE FONCTION:
Exécute la fonction 'print' avec les paramètres: f"\nTemps d'exécution : {time.time(
Flux d'exécution: Saute à la définition de print, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.

[282] `[LIGNE VIDE]`

LIGNE VIDE:
Espace blanc pour améliorer la lisibilité du code. Sépare logiquement les différents blocs de code pour les rendre plus lisibles.

[283] `if __name__ == "__main__":`

CONDITION IF:
Teste si l'expression est VRAIE: __name__ == "__main__"
Flux: Si VRAI → exécute le bloc qui suit. Si FAUX → passe au elif/else.
Exemple réel: 'if conn.is_connected():' → vérifier si la connexion est active avant de l'utiliser.

[284] `main()`

APPEL DE FONCTION:
Exécute la fonction 'main' avec les paramètres: aucun
Flux d'exécution: Saute à la définition de main, exécute son code, puis revient ici.
Exemple réel: 'get_connection()' → appelle la fonction qui crée la connexion MySQL.