■ DOCUMENTATION DÉTAILLÉE DES SCRIPTS PYTHON

Date: 20/01/2026 09:37:17
Répertoire: app/scripts/
Nombre de fichiers: 28

■ __init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ db/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ db/mysql_conn.py

Nombre de lignes: 21 | **Taille:** 484 bytes

Ligne 1: *Importation de module* - import mysql.connector
Ligne 2: *Importation de module* - from mysql.connector import Error
Ligne 4: *Définition de fonction* - **get_connection**
Ligne 5: *Gestion d'exception* - try:
Ligne 6: *Assignation de variable* - **conn** = mysql.connector.connect(
Ligne 7: *Assignation de variable* - **host** = "database",
Ligne 8: *Assignation de variable* - **user** = "admin",
Ligne 9: *Assignation de variable* - **password** = "admin7791",
Ligne 10: *Assignation de variable* - **database** = "lab_test"
Ligne 11:)
Ligne 12: *Condition* - if conn.is_connected():
Ligne 13: *Retour* - return conn
Ligne 14: *Gestion d'exception* - except Error as e:
Ligne 15: *Appel de fonction/méthode* - print(f"MySQL Error: {e}")
Ligne 16: *Retour* - return None
Ligne 18: *Condition* - if __name__ == "__main__":
Ligne 19: *Assignation de variable* - **conn** = get_connection()
Ligne 20: *Condition* - if conn:
Ligne 21: *Appel de fonction/méthode* - conn.close()

■ exploit/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ exploit/attack_chains/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ exploit/attack_chains/apache.py

Nombre de lignes: 59 | **Taille:** 1730 bytes

Ligne 1: *Définition de fonction* - **evaluate_apache**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "Apache",
Ligne 5: "version": "2.4.65",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 16: *Condition* - if "apache" not in service:
Ligne 17: *Retour* - return None
Ligne 19: *Assignation de variable* - **has_php** = "php" in headers
Ligne 20: *Assignation de variable* - **is_windows** = "win" in headers
Ligne 21: *Assignation de variable* - **is_linux** = not is_windows
Ligne 26: *Condition* - if version.startswith("2.4.") and any(
Ligne 27: cve in cves for cve in {
Ligne 28: "CVE-2022-22720",
Ligne 29: "CVE-2022-22721",
Ligne 30: "CVE-2022-23943"
Ligne 31: }
Ligne 32:):
Ligne 33: *Retour* - return {
Ligne 34: "attack": "Apache path traversal → RCE",
Ligne 35: "confidence": 0.95,
Ligne 36: "exploit": "exploit/multi/http/apache_normalize_path_rce"
Ligne 37: }
Ligne 42: *Condition* - if has_php and any(cve.startswith("CVE-2025") for cve in cves):
Ligne 43: *Retour* - return {
Ligne 44: *Appel de fonction/méthode* - "attack": "Apache + PHP attack surface (RCE candidates)",
Ligne 45: "confidence": 0.70,
Ligne 46: "exploit": "exploit/multi/http/apache_php_rce"
Ligne 47: }
Ligne 52: *Condition* - if "CVE-2025-58098" in cves:
Ligne 53: *Retour* - return {
Ligne 54: "attack": "Apache fingerprinting / header info leak",
Ligne 55: "confidence": 0.45,
Ligne 56: "exploit": None
Ligne 57: }
Ligne 59: *Retour* - return None

■ exploit/attack_chains/chain_engine.py

Nombre de lignes: 12 | **Taille:** 343 bytes

Ligne 1: *Importation de module* - from scripts.exploit.attack_chains.apache import evaluate_apache
Ligne 3: *Définition de fonction* - **evaluate_chain**
Ligne 4: """
Ligne 5: *Appel de fonction/méthode* - Appelle tous les moteurs de chaînes (Apache, SSH, etc)
Ligne 6: et retourne la première chaîne valide.
Ligne 7: """
Ligne 8: *Boucle* - for engine in [evaluate_apache]:
Ligne 9: *Assignation de variable* - **result** = engine(facts)
Ligne 10: *Condition* - if result:
Ligne 11: *Retour* - return result
Ligne 12: *Retour* - return None

■ exploit/attack_chains/ftp.py

Nombre de lignes: 80 | **Taille:** 2615 bytes

Ligne 1: *Définition de fonction* - **evaluate_ftp**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "vsftpd" ou "Pure-FTPd" ou "ProFTPD" ou "FileZilla Server",
Ligne 5: "version": "2.3.4",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 17: *Assignation de variable* - **ftp_services** = ["ftp", "vsftpd", "pure-ftpd", "proftpd", "filezilla"]
Ligne 18: *Condition* - if not any(ftp_svc in service for ftp_svc in ftp_services):
Ligne 19: *Retour* - return None
Ligne 24: *Condition* - if "vsftpd" in service and version.startswith("2.3.") and "CVE-2011-0762" in cve
Ligne 25: *Retour* - return {
Ligne 26: "attack": "vsftpd 2.3.4 backdoor RCE",
Ligne 27: "confidence": 0.95,
Ligne 28: "exploit": "exploit/ftp/vsftpd_234_backdoor"
Ligne 29: }
Ligne 34: *Condition* - if "proftpd" in service and any(
Ligne 35: cve in cves for cve in {
Ligne 36: "CVE-2015-3306",
Ligne 37: "CVE-2015-3317"
Ligne 38: }
Ligne 39:):
Ligne 40: *Retour* - return {
Ligne 41: "attack": "ProFTPD mod_copy RCE",
Ligne 42: "confidence": 0.85,
Ligne 43: "exploit": "exploit/ftp/proftpd_modcopy_exec"
Ligne 44: }
Ligne 49: *Condition* - if "pure-ftpd" in service and any(
Ligne 50: *Appel de fonction/méthode* - cve.startswith("CVE-2021") or cve.startswith("CVE-2020") for cve in cves
Ligne 51:):
Ligne 52: *Retour* - return {
Ligne 53: "attack": "Pure-FTPd authentication bypass / privilege escalation",
Ligne 54: "confidence": 0.70,
Ligne 55: "exploit": None
Ligne 56: }
Ligne 61: *Condition* - if "filezilla" in service and any(
Ligne 62: *Appel de fonction/méthode* - cve.startswith("CVE-2019") or cve.startswith("CVE-2020") for cve in cves

Ligne 63:):
Ligne 64: *Retour* - return {
Ligne 65: "attack": "FileZilla Server path traversal / directory traversal",
Ligne 66: "confidence": 0.75,
Ligne 67: "exploit": None
Ligne 68: }
Ligne 73: *Condition* - if any(cve.startswith("CVE-") for cve in cves) and len(cves) > 0:
Ligne 74: *Retour* - return {
Ligne 75: "attack": "FTP brute force / anonymous access exploitation",
Ligne 76: "confidence": 0.50,
Ligne 77: "exploit": None
Ligne 78: }
Ligne 80: *Retour* - return None

■ exploit/attack_chains/mysql.py

Nombre de lignes: 96 | **Taille:** 2940 bytes

Ligne 1: *Définition de fonction* - **evaluate_mysql**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "MySQL",
Ligne 5: "version": "5.7.34",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 16: *Condition* - if "mysql" not in service:
Ligne 17: *Retour* - return None
Ligne 22: *Condition* - if any(
Ligne 23: cve in cves for cve in {
Ligne 24: "CVE-2016-6663",
Ligne 25: "CVE-2016-6662",
Ligne 26: "CVE-2016-6664"
Ligne 27: }
Ligne 28:):
Ligne 29: *Retour* - return {
Ligne 30: "attack": "MySQL UDF privilege escalation → RCE",
Ligne 31: "confidence": 0.90,
Ligne 32: "exploit": "exploit/mysql/mysql_yassl_hello"
Ligne 33: }
Ligne 38: *Condition* - if version.startswith("5.") and any(
Ligne 39: *Appel de fonction/méthode* - cve.startswith("CVE-2020") or cve.startswith("CVE-2021") for cve in cves
Ligne 40:):
Ligne 41: *Retour* - return {
Ligne 42: *Appel de fonction/méthode* - "attack": "MySQL remote code execution (version-specific)",
Ligne 43: "confidence": 0.80,
Ligne 44: "exploit": None
Ligne 45: }
Ligne 50: *Condition* - if any(
Ligne 51: cve in cves for cve in {
Ligne 52: "CVE-2012-2122",
Ligne 53: "CVE-2016-5734"
Ligne 54: }
Ligne 55:):
Ligne 56: *Retour* - return {
Ligne 57: "attack": "MySQL authentication bypass / SQL injection",
Ligne 58: "confidence": 0.85,
Ligne 59: "exploit": "exploit/mysql/mysql_authbypass_hashdump"

Ligne 60: }
Ligne 65: *Condition* - if version.startswith("8.") and any(cve.startswith("CVE-2022") for cve in cves):
Ligne 66: *Retour* - return {
Ligne 67: "attack": "MySQL 8.x remote code execution / privilege escalation",
Ligne 68: "confidence": 0.75,
Ligne 69: "exploit": None
Ligne 70: }
Ligne 72: *Condition* - if version.startswith("5.7.") and any(cve.startswith("CVE-") for cve in cves):
Ligne 73: *Retour* - return {
Ligne 74: *Appel de fonction/méthode* - "attack": "MySQL 5.7.x vulnerabilities (RCE candidates)",
Ligne 75: "confidence": 0.70,
Ligne 76: "exploit": None
Ligne 77: }
Ligne 79: *Condition* - if version.startswith("5.6.") or version.startswith("5.5."):
Ligne 80: *Retour* - return {
Ligne 81: *Appel de fonction/méthode* - "attack": "MySQL legacy version (potential vulnerabilities)",
Ligne 82: "confidence": 0.60,
Ligne 83: "exploit": None
Ligne 84: }
Ligne 89: *Condition* - if any(cve.startswith("CVE-") for cve in cves):
Ligne 90: *Retour* - return {
Ligne 91: "attack": "MySQL brute force / weak authentication",
Ligne 92: "confidence": 0.50,
Ligne 93: "exploit": None
Ligne 94: }
Ligne 96: *Retour* - return None

■ exploit/attack_chains/rdp.py

Nombre de lignes: 81 | **Taille:** 2603 bytes

Ligne 1: *Définition de fonction* - **evaluate_rdp**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "RDP" ou "Remote Desktop",
Ligne 5: "version": "10.0.17763",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 17: *Assignation de variable* - **rdp_services** = ["rdp", "remote desktop", "terminal services"]
Ligne 18: *Condition* - if not any(rdp_svc in service for rdp_svc in rdp_services):
Ligne 19: *Retour* - return None
Ligne 24: *Condition* - if "CVE-2019-0708" in cves:
Ligne 25: *Retour* - return {
Ligne 26: *Appel de fonction/méthode* - "attack": "BlueKeep (CVE-2019-0708) RDP RCE",
Ligne 27: "confidence": 0.95,
Ligne 28: "exploit": "exploit/rdp/bluekeep_cve_2019_0708"
Ligne 29: }
Ligne 34: *Condition* - if "CVE-2019-1181" in cves or "CVE-2019-1182" in cves:
Ligne 35: *Retour* - return {
Ligne 36: "attack": "DejaBlue RDP RCE vulnerabilities",
Ligne 37: "confidence": 0.90,
Ligne 38: "exploit": "exploit/rdp/dejablue_rdp_rce"
Ligne 39: }
Ligne 44: *Condition* - if "CVE-2021-34527" in cves:
Ligne 45: *Retour* - return {
Ligne 46: *Appel de fonction/méthode* - "attack": "PrintNightmare (CVE-2021-34527) via RDP",
Ligne 47: "confidence": 0.85,
Ligne 48: "exploit": "exploit/rdp/printnightmare_cve_2021_34527"
Ligne 49: }
Ligne 54: *Condition* - if "CVE-2021-38647" in cves:
Ligne 55: *Retour* - return {
Ligne 56: *Appel de fonction/méthode* - "attack": "OpenSSL RDP vulnerability (CVE-2021-38647)",
Ligne 57: "confidence": 0.80,
Ligne 58: "exploit": None
Ligne 59: }
Ligne 64: *Condition* - if "CVE-2020-0610" in cves:
Ligne 65: *Retour* - return {
Ligne 66: "attack": "RDP Man-in-the-Middle vulnerability",
Ligne 67: "confidence": 0.75,
Ligne 68: "exploit": None

Ligne 69: }
Ligne 74: *Condition* - if any(cve.startswith("CVE-") for cve in cves):
Ligne 75: *Retour* - return {
Ligne 76: "attack": "RDP brute force / credential stuffing",
Ligne 77: "confidence": 0.60,
Ligne 78: "exploit": None
Ligne 79: }
Ligne 81: *Retour* - return None

■ exploit/attack_chains/smb.py

Nombre de lignes: 78 | **Taille:** 2395 bytes

Ligne 1: *Définition de fonction* - **evaluate_smb**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "SMB",
Ligne 5: "version": "3.1.1",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 17: *Condition* - if "smb" not in service:
Ligne 18: *Retour* - return None
Ligne 23: *Assignation de variable* - **eternalblue_cves** = {
Ligne 24: "CVE-2017-0143",
Ligne 25: "CVE-2017-0144",
Ligne 26: "CVE-2017-0145",
Ligne 27: "CVE-2017-0146",
Ligne 28: "CVE-2017-0147",
Ligne 29: "CVE-2017-0148"
Ligne 30: }
Ligne 31: *Condition* - if any(cve in cves for cve in eternalblue_cves):
Ligne 32: *Retour* - return {
Ligne 33: *Appel de fonction/méthode* - "attack": "SMB EternalBlue (MS17-010) RCE",
Ligne 34: "confidence": 0.95,
Ligne 35: "exploit": "exploit/smb/eternalblue_ms17_010"
Ligne 36: }
Ligne 41: *Condition* - if "CVE-2020-0796" in cves:
Ligne 42: *Retour* - return {
Ligne 43: *Appel de fonction/méthode* - "attack": "SMBGhost (SMBv3 compression) RCE",
Ligne 44: "confidence": 0.90,
Ligne 45: "exploit": "exploit/smb/smbghost_cve_2020_0796"
Ligne 46: }
Ligne 51: *Condition* - if "CVE-2021-44142" in cves:
Ligne 52: *Retour* - return {
Ligne 53: *Appel de fonction/méthode* - "attack": "Samba VFS module RCE (CVE-2021-44142)",
Ligne 54: "confidence": 0.85,
Ligne 55: "exploit": "exploit/smb/samba_vfs_rce"
Ligne 56: }
Ligne 61: *Condition* - if any(cve.startswith("CVE-2019") or cve.startswith("CVE-2020") for cve in cves)
Ligne 62: *Retour* - return {
Ligne 63: "attack": "SMB relay / NTLM relay attack",
Ligne 64: "confidence": 0.75,
Ligne 65: "exploit": None

Ligne 66: }
Ligne 71: *Condition* - if any(cve.startswith("CVE-") for cve in cves):
Ligne 72: *Retour* - return {
Ligne 73: "attack": "SMB null session / anonymous access exploitation",
Ligne 74: "confidence": 0.60,
Ligne 75: "exploit": None
Ligne 76: }
Ligne 78: *Retour* - return None

■ exploit/attack_chains/ssh.py

Nombre de lignes: 119 | **Taille:** 4002 bytes

Ligne 1: *Définition de fonction* - **evaluate_ssh**
Ligne 2: """
Ligne 3: *Assignation de variable* - **facts** = {
Ligne 4: "service": "OpenSSH",
Ligne 5: "version": "8.2p1",
Ligne 6: "validated_cves": [...],
Ligne 7: "headers": [...],
Ligne 8: }
Ligne 9: """
Ligne 11: *Assignation de variable* - **service** = (facts.get("service") or "").lower()
Ligne 12: *Assignation de variable* - **version** = facts.get("version") or ""
Ligne 13: *Assignation de variable* - **cves** = set(facts.get("validated_cves", []))
Ligne 14: *Assignation de variable* - **headers** = " ".join(facts.get("headers", [])).lower()
Ligne 16: *Condition* - if "ssh" not in service and "openssh" not in service:
Ligne 17: *Retour* - return None
Ligne 20: *Assignation de variable* - **openssh_version** = None
Ligne 21: *Condition* - if version:
Ligne 22: *Assignation de variable* - **openssh_version** = version
Ligne 27: *Condition* - if "CVE-2018-15473" in cves:
Ligne 28: *Retour* - return {
Ligne 29: "attack": "OpenSSH username enumeration",
Ligne 30: "confidence": 0.85,
Ligne 31: "exploit": "exploit/ssh/ssh_enumusers"
Ligne 32: }
Ligne 37: *Condition* - if "CVE-2020-15778" in cves:
Ligne 38: *Retour* - return {
Ligne 39: "attack": "OpenSSH command injection → RCE",
Ligne 40: "confidence": 0.90,
Ligne 41: "exploit": "exploit/ssh/openssh_command_injection"
Ligne 42: }
Ligne 47: *Condition* - if "CVE-2019-6111" in cves or "CVE-2019-6109" in cves:
Ligne 48: *Retour* - return {
Ligne 49: "attack": "OpenSSH X11 forwarding / file disclosure",
Ligne 50: "confidence": 0.80,
Ligne 51: "exploit": "exploit/ssh/openssh_x11_forward"
Ligne 52: }
Ligne 57: *Condition* - if openssh_version:
Ligne 58: *Gestion d'exception* - try:
Ligne 60: *Condition* - if "p" in openssh_version:
Ligne 61: *Assignation de variable* - **major_minor** = openssh_version.split("p")[0]
Ligne 62: *Condition* - else:
Ligne 63: *Assignation de variable* - **parts** = openssh_version.split(".")
Ligne 64: *Appel de fonction/méthode* - major_minor = f"{parts[0]}.{parts[1]}" if len(parts) >= 2 else openssh_version
Ligne 66: *Assignation de variable* - **version_float** = float(major_minor)
Ligne 68: *Condition* - if version_float <= 7.4 and any(cve.startswith("CVE-2017") for cve in cves):

Ligne 69: *Retour* - return {
Ligne 70: *Appel de fonction/méthode* - "attack": "OpenSSH 7.4 and below vulnerabilities (RCE candidates)",
Ligne 71: "confidence": 0.75,
Ligne 72: "exploit": None
Ligne 73: }
Ligne 74: *Gestion d'exception* - except (ValueError, AttributeError):
Ligne 75: pass
Ligne 80: *Condition* - if openssh_version:
Ligne 81: *Gestion d'exception* - try:
Ligne 82: *Condition* - if "p" in openssh_version:
Ligne 83: *Assignation de variable* - **major_minor** = openssh_version.split("p")[0]
Ligne 84: *Condition* - else:
Ligne 85: *Assignation de variable* - **parts** = openssh_version.split(".")
Ligne 86: *Appel de fonction/méthode* - major_minor = f"{parts[0]}.{parts[1]}" if len(parts) >= 2 else openssh_version
Ligne 88: *Assignation de variable* - **version_float** = float(major_minor)
Ligne 90: *Condition* - if version_float < 7.0:
Ligne 91: *Retour* - return {
Ligne 92: *Appel de fonction/méthode* - "attack": "OpenSSH legacy version (known vulnerabilities)",
Ligne 93: "confidence": 0.85,
Ligne 94: "exploit": None
Ligne 95: }
Ligne 96: *Gestion d'exception* - except (ValueError, AttributeError):
Ligne 97: pass
Ligne 102: *Condition* - if any(cve.startswith("CVE-") for cve in cves):
Ligne 103: *Retour* - return {
Ligne 104: "attack": "OpenSSH brute force / weak key exploitation",
Ligne 105: "confidence": 0.60,
Ligne 106: "exploit": None
Ligne 107: }
Ligne 112: *Condition* - if any(cve.startswith("CVE-202") for cve in cves):
Ligne 113: *Retour* - return {
Ligne 114: *Appel de fonction/méthode* - "attack": "OpenSSH recent vulnerabilities (RCE candidates)",
Ligne 115: "confidence": 0.70,
Ligne 116: "exploit": None
Ligne 117: }
Ligne 119: *Retour* - return None

■ exploit/bruteforce.py

Nombre de lignes: 163 | **Taille:** 4625 bytes

Ligne 1: *Importation de module* - import sys, json, re
Ligne 2: *Importation de module* - from scripts.db.mysql_conn import get_connection
Ligne 3: *Importation de module* - from scripts.exploit.engine.security_engine import SecurityEngine
Ligne 4: *Importation de module* - from scripts.exploit.engine.exploit_engine import ExploitEngine
Ligne 5: *Importation de module* - from scripts.exploit.attack_chains.chain_engine import evaluate_chain
Ligne 6: *Importation de module* - from scripts.exploit.engine.exploit_runner import ExploitRunner
Ligne 9: *Définition de fonction* - **parse_cves**
Ligne 10: *Assignation de variable* - **out** = []
Ligne 11: *Condition* - if not raw:
Ligne 12: *Retour* - return out
Ligne 14: *Boucle* - for p in raw.split(","):
Ligne 15: *Assignation de variable* - **m** = re.match(r'(CVE-\d{4}-\d+)', p.strip())
Ligne 16: *Condition* - if m:
Ligne 17: *Appel de fonction/méthode* - out.append(m.group(1))
Ligne 18: *Retour* - return out
Ligne 21: *Définition de fonction* - **main**
Ligne 22: *Assignation de variable* - **user_id, target** = sys.argv[1], sys.argv[2]
Ligne 24: *Assignation de variable* - **conn** = get_connection()
Ligne 25: *Assignation de variable* - **cur** = conn.cursor(dictionary=True)
Ligne 28: cur.execute("""
Ligne 29: SELECT scanner.script_vuln, scanner.service, scanner.version
Ligne 30: FROM scanner
Ligne 31: *Assignation de variable* - **JOIN ping ON scanner.ping_id** = ping.id
Ligne 32: *Assignation de variable* - **WHERE ping.user_id** = %s AND ping.ip_address=%s
Ligne 33: *Appel de fonction/méthode* - """, (user_id, target))
Ligne 35: *Assignation de variable* - **rows** = cur.fetchall()
Ligne 37: *Appel de fonction/méthode* - print(f"\n==== Exploit de la cible {target} ======\n")
Ligne 39: *Condition* - if not rows:
Ligne 40: *Appel de fonction/méthode* - print("■ Aucun service exploitable")
Ligne 41: *Retour* - return
Ligne 44: *Assignation de variable* - **services** = {}
Ligne 45: *Assignation de variable* - **proofs** = []
Ligne 47: *Boucle* - for r in rows:
Ligne 48: *Assignation de variable* - **service** = (r["service"] or "").strip()
Ligne 49: *Assignation de variable* - **version** = (r["version"] or "").strip()
Ligne 51: *Condition* - if not service:
Ligne 52: continue
Ligne 54: *Assignation de variable* - **key** = f"{service.lower()}:{version}"
Ligne 56: *Condition* - if key not in services:
Ligne 57: *Assignation de variable* - **services[key]** = {
Ligne 58: "service": service,
Ligne 59: "version": version,

Ligne 60: *Appel de fonction/méthode* - "validated_cves": set()
Ligne 61: }
Ligne 64: *Assignation de variable* - **sec** = SecurityEngine()
Ligne 66: *Boucle* - for key, svc in services.items():
Ligne 67: *Assignation de variable* - **service** = svc["service"]
Ligne 68: *Assignation de variable* - **version** = svc["version"]
Ligne 71: *Assignation de variable* - **cves** = set()
Ligne 72: *Boucle* - for r in rows:
Ligne 73: *Condition* - if r["service"] == service and r["version"] == version:
Ligne 74: *Appel de fonction/méthode* - cves.update(parse_cves(r["script_vuln"]))
Ligne 76: *Condition* - if not cves:
Ligne 77: continue
Ligne 79: *Appel de fonction/méthode* - print(f"■ Analyse {service} {version} → {len(cves)} CVE")
Ligne 81: *Boucle* - for cve in cves:
Ligne 82: *Assignation de variable* - **modules** = sec.search_modules(cve)
Ligne 84: *Boucle* - for m in modules:
Ligne 85: *Assignation de variable* - **res** = sec.run_module(m, target)
Ligne 87: *Condition* - if res.get("status") == "exploitable":
Ligne 88: *Appel de fonction/méthode* - svc["validated_cves"].add(cve)
Ligne 90: *Assignation de variable* - **proof** = res.get("proof")
Ligne 91: *Condition* - if proof:
Ligne 92: *Appel de fonction/méthode* - proofs.append(proof)
Ligne 94: *Appel de fonction/méthode* - print(f" ■ {cve} exploitable via {m}")
Ligne 97: *Assignation de variable* - **facts** = {
Ligne 98: "services": [],
Ligne 99: *Appel de fonction/méthode* - "proofs": list(set(proofs))
Ligne 100: }
Ligne 102: *Boucle* - for svc in services.values():
Ligne 103: *Condition* - if svc["validated_cves"]:
Ligne 104: facts["services"].append({
Ligne 105: "service": svc["service"],
Ligne 106: "version": svc["version"],
Ligne 107: *Appel de fonction/méthode* - "validated_cves": list(svc["validated_cves"])
Ligne 108: })
Ligne 110: *Condition* - if not facts["services"]:
Ligne 111: *Appel de fonction/méthode* - print("\n■ Aucune CVE exploitable validée")
Ligne 112: *Retour* - return
Ligne 114: *Appel de fonction/méthode* - print("\n■ FACTS CONSTRUITS")
Ligne 115: *Assignation de variable* - **print(json.dumps(facts, indent** = 2))
Ligne 118: *Assignation de variable* - **chain** = None
Ligne 120: *Boucle* - for svc in facts["services"]:
Ligne 121: *Assignation de variable* - **chain** = evaluate_chain({
Ligne 122: "service": svc["service"],
Ligne 123: "version": svc["version"],
Ligne 124: "validated_cves": svc["validated_cves"],
Ligne 125: "headers": facts["proofs"]
Ligne 126: })
Ligne 128: *Condition* - if chain:
Ligne 129: break
Ligne 131: *Appel de fonction/méthode* - print("\n--- ATTACK CHAIN ---")

Ligne 132: *Assignation de variable* - **print(json.dumps(chain, indent** = 2))
Ligne 134: *Condition* - if not chain or not chain.get("exploit"):
Ligne 135: *Appel de fonction/méthode* - print("\n■■ Aucune chaîne d’attaque exploitable")
Ligne 136: *Retour* - return
Ligne 139: *Appel de fonction/méthode* - print("\n■ Exploit recommandé :", chain["exploit"])
Ligne 141: *Assignation de variable* - **runner** = ExploitRunner()
Ligne 142: *Assignation de variable* - **result** = runner.run(chain["exploit"], target)
Ligne 145: *Assignation de variable* - **clean_proofs** = list(dict.fromkeys(proofs))
Ligne 146: *Assignation de variable* - **exploit_data** = {
Ligne 147: "module": chain["exploit"],
Ligne 148: "payload": None,
Ligne 149: "status": result["status"],
Ligne 150: "session_id": None,
Ligne 151: "output": json.dumps(
Ligne 152: *Assignation de variable* - **result.get("details") or result, indent** = 2),
Ligne 153: *Appel de fonction/méthode* - "proof": " | ".join(clean_proofs)
Ligne 154: }
Ligne 155: *Appel de fonction/méthode* - print("\n@@@EXPLOITJSON@@@")
Ligne 156: *Appel de fonction/méthode* - print(json.dumps(exploit_data))
Ligne 162: *Condition* - if __name__ == "__main__":
Ligne 163: *Appel de fonction/méthode* - main()

■ exploit/engine/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ exploit/engine/attack_chain_engine.py

Nombre de lignes: 98 | **Taille:** 3005 bytes

Ligne 1: *Importation de module* - from scripts.exploit.attack_chains.apache import evaluate_apache
Ligne 6: *Définition de classe* - **AttackChainEngine**
Ligne 7: """
Ligne 8: Orchestrateur global des chaînes d’attaque.
Ligne 10: - Support multi-services
Ligne 11: - Support multi-CVE
Ligne 12: - Ne casse aucun engine existant
Ligne 13: *Appel de fonction/méthode* - - Retourne UNE attaque finale (la meilleure)
Ligne 14: """
Ligne 16: *Définition de fonction* - **__init__**
Ligne 17: *Assignation de variable* - **self.engines** = {
Ligne 18: "apache": evaluate_apache,
Ligne 21: }
Ligne 23: *Définition de fonction* - **evaluate**
Ligne 24: """
Ligne 25: *Assignation de variable* - **facts** = {
Ligne 26: "services": [
Ligne 27: {
Ligne 28: "service": "Apache",
Ligne 29: "version": "2.4.65",
Ligne 30: "validated_cves": [...]
Ligne 31: }
Ligne 32:],
Ligne 33: "proofs": [...]
Ligne 34: }
Ligne 35: """
Ligne 37: *Assignation de variable* - **services** = facts.get("services", [])
Ligne 38: *Assignation de variable* - **proofs** = list(set(facts.get("proofs", [])))
Ligne 40: *Assignation de variable* - **attack_candidates** = []
Ligne 42: *Boucle* - for svc in services:
Ligne 43: *Assignation de variable* - **service_name** = (svc.get("service") or "").lower()
Ligne 44: *Assignation de variable* - **version** = svc.get("version")
Ligne 45: *Assignation de variable* - **cves** = list(set(svc.get("validated_cves", [])))
Ligne 47: *Assignation de variable* - **engine** = self.engines.get(service_name)
Ligne 48: *Condition* - if not engine:
Ligne 49: continue
Ligne 51: *Boucle* - for cve in cves:
Ligne 52: *Assignation de variable* - **unit_facts** = {
Ligne 53: *Appel de fonction/méthode* - "service": svc.get("service"),
Ligne 54: "version": version,
Ligne 55: "validated_cves": [cve],
Ligne 56: "headers": proofs,
Ligne 57: }
Ligne 59: *Gestion d'exception* - try:
Ligne 60: *Assignation de variable* - **result** = engine(unit_facts)
Ligne 61: *Condition* - if result:

Ligne 62: attack_candidates.append({
Ligne 63: *Appel de fonction/méthode* - "service": svc.get("service"),
Ligne 64: "version": version,
Ligne 65: "cve": cve,
Ligne 66: *Appel de fonction/méthode* - "attack": result.get("attack"),
Ligne 67: *Appel de fonction/méthode* - "confidence": result.get("confidence", 0),
Ligne 68: *Appel de fonction/méthode* - "exploit": result.get("exploit"),
Ligne 69: "proof": proofs,
Ligne 70: })
Ligne 71: *Gestion d'exception* - except Exception:
Ligne 72: continue
Ligne 74: *Condition* - if not attack_candidates:
Ligne 75: *Retour* - return {
Ligne 76: "status": False,
Ligne 77: "impact": "no_attack_chain",
Ligne 78: "proof": "No applicable attack chain found"
Ligne 79: }
Ligne 82: *Assignation de variable* - **best** = sorted(
Ligne 83: attack_candidates,
Ligne 84: *Assignation de variable* - **key** = lambda x: (x["confidence"], bool(x["exploit"])),
Ligne 85: *Assignation de variable* - **reverse** = True
Ligne 86:)[0]
Ligne 88: *Retour* - return {
Ligne 89: "status": True,
Ligne 90: "impact": "attack_chain_available",
Ligne 91: "service": best["service"],
Ligne 92: "version": best["version"],
Ligne 93: "cve": best["cve"],
Ligne 94: "attack": best["attack"],
Ligne 95: "confidence": best["confidence"],
Ligne 96: "exploit": best["exploit"],
Ligne 97: "proof": best["proof"]
Ligne 98: }

■ exploit/engine/exploit_engine.py

Nombre de lignes: 38 | **Taille:** 1161 bytes

Ligne 1: *Importation de module* - from scripts.exploit.engine.exploit_mapper import ExploitMapper
Ligne 4: *Définition de classe* - **ExploitEngine**
Ligne 5: """
Ligne 6: *Appel de fonction/méthode* - Prend les faits (service, version, CVE validées)
Ligne 7: et détermine s'il existe une chaîne d'attaque exploitable.
Ligne 8: """
Ligne 10: *Définition de fonction* - **__init__**
Ligne 11: *Assignation de variable* - **self.mapper** = ExploitMapper()
Ligne 13: *Définition de fonction* - **analyze**
Ligne 14: *Assignation de variable* - **rec** = self.mapper.map(cve, service, version)
Ligne 16: *Condition* - if not rec:
Ligne 17: *Retour* - return {
Ligne 18: "status": False,
Ligne 19: "impact": "no_exploit",
Ligne 20: "proof": "No attack chain mapped",
Ligne 21: "target": target,
Ligne 22: "service": service,
Ligne 23: "version": version,
Ligne 24: "cve": cve
Ligne 25: }
Ligne 27: *Retour* - return {
Ligne 28: "status": True,
Ligne 29: "impact": "attack_chain_available",
Ligne 30: "attack_family": rec["attack_family"],
Ligne 31: "risk": rec["risk"],
Ligne 32: "confidence": rec["confidence"],
Ligne 33: "target": target,
Ligne 34: "service": service,
Ligne 35: "version": version,
Ligne 36: "cve": cve,
Ligne 37: "proof": f"Matched {rec['attack_family']} with {rec['confidence']*100:.0f}% confidence"
Ligne 38: }

■ exploit/engine/exploit_mapper.py

Nombre de lignes: 36 | **Taille:** 1049 bytes

Ligne 1: *Définition de classe* - **ExploitMapper**
Ligne 2: """
Ligne 3: *Appel de fonction/méthode* - Transforme (CVE + service + version) en intention d’attaque.
Ligne 4: Ne lance rien. Ne contient pas de payloads.
Ligne 5: """
Ligne 7: *Définition de fonction* - **map**
Ligne 8: *Assignation de variable* - **s** = (service or "").lower()
Ligne 9: *Assignation de variable* - **v** = (version or "")
Ligne 12: *Condition* - if cve in (
Ligne 13: "CVE-2022-22720",
Ligne 14: "CVE-2022-22721",
Ligne 15: "CVE-2022-23943",
Ligne 16: "CVE-2022-22719",
Ligne 17:) and "apache" in s:
Ligne 18: *Retour* - return {
Ligne 19: "attack_family": "apache_path_traversal_rce",
Ligne 20: "service": "Apache",
Ligne 21: "version": v,
Ligne 22: "confidence": 0.93,
Ligne 23: "risk": "CRITICAL"
Ligne 24: }
Ligne 27: *Condition* - if cve == "CVE-2025-58098" and "apache" in s:
Ligne 28: *Retour* - return {
Ligne 29: "attack_family": "apache_version_disclosure",
Ligne 30: "service": "Apache",
Ligne 31: "version": v,
Ligne 32: "confidence": 0.40,
Ligne 33: "risk": "LOW"
Ligne 34: }
Ligne 36: *Retour* - return None

■ exploit/engine/exploit_runner.py

Nombre de lignes: 27 | **Taille:** 826 bytes

Ligne 2: *Importation de module* - import importlib
Ligne 3: *Définition de classe* - **ExploitRunner**
Ligne 4: """
Ligne 5: Lanceur d’exploit réel
Ligne 6: Reçoit le chemin logique depuis l’attack chain
Ligne 7: """
Ligne 9: *Définition de fonction* - **run**
Ligne 10: """
Ligne 11: exploit : string ex. "exploit/multi/http/apache_php_rce"
Ligne 12: target : IP ou hostname
Ligne 13: """
Ligne 14: *Gestion d'exception* - try:
Ligne 15: *Assignation de variable* - **parts** = exploit_path.split("/")
Ligne 16: *Assignation de variable* - **module_name** = parts[-1]
Ligne 17: *Assignation de variable* - **service** = parts[-2]
Ligne 19: *Assignation de variable* - **module_path** = f"scripts.exploit.modules.{service}.{module_name}"
Ligne 20: *Assignation de variable* - **module** = importlib.import_module(module_path)
Ligne 21: *Retour* - return module.run(target)
Ligne 22: *Gestion d'exception* - except Exception as e:
Ligne 23: *Retour* - return {
Ligne 24: "status": False,
Ligne 25: *Appel de fonction/méthode* - "proof": f"Exploit runner error: {str(e)}"
Ligne 26: }

■ exploit/engine/security_engine.py

Nombre de lignes: 157 | **Taille:** 4611 bytes

Ligne 1: *Importation de module* - import requests
Ligne 2: *Importation de module* - import re
Ligne 5: *Définition de classe* - **SecurityEngine**
Ligne 6: """
Ligne 7: *Assignation de variable* - **SecurityEngine** = Validation technique
Ligne 8: Il vérifie si une CVE détectée est réellement applicable à la cible.
Ligne 10: Il ne lance PAS d'exploit.
Ligne 11: Il confirme juste :
Ligne 12: - le service
Ligne 13: - la version
Ligne 14: - les headers
Ligne 15: - la surface d’attaque
Ligne 16: """
Ligne 18: *Définition de fonction* - **__init__**
Ligne 20: *Assignation de variable* - **self.cve_modules** = {
Ligne 22: "CVE-2022-22720": ["apache_fingerprint"],
Ligne 23: "CVE-2022-22721": ["apache_fingerprint"],
Ligne 24: "CVE-2022-23943": ["apache_fingerprint"],
Ligne 25: "CVE-2022-22719": ["apache_fingerprint"],
Ligne 28: "CVE-2025-58098": ["apache_headers"],
Ligne 29: }
Ligne 31: *Définition de fonction* - **search_modules**
Ligne 33: *Retour* - return self.cve_modules.get(cve_id, ["generic_fingerprint"])
Ligne 35: *Définition de fonction* - **run_module**
Ligne 36: *Gestion d'exception* - try:
Ligne 37: *Condition* - if module == "apache_fingerprint":
Ligne 38: *Retour* - return self._apache_fingerprint(target)
Ligne 40: *Condition* - if module == "apache_headers":
Ligne 41: *Retour* - return self._apache_headers(target)
Ligne 43: *Condition* - if module == "generic_fingerprint":
Ligne 44: *Retour* - return self._generic_fingerprint(target)
Ligne 46: *Gestion d'exception* - except Exception as e:
Ligne 47: *Retour* - return {
Ligne 48: "status": "error",
Ligne 49: *Appel de fonction/méthode* - "proof": str(e),
Ligne 50: "service": None,
Ligne 51: "version": None
Ligne 52: }
Ligne 54: *Retour* - return {
Ligne 55: "status": "not_exploitable",
Ligne 56: "proof": "Unknown module",
Ligne 57: "service": None,
Ligne 58: "version": None
Ligne 59: }
Ligne 65: *Définition de fonction* - **_fetch**
Ligne 66: *Assignation de variable* - **url** = f"http://{target}"

Ligne 67: *Assignation de variable* - **r** = requests.get(url, timeout=5, allow_redirects=True)
Ligne 68: *Retour* - return r
Ligne 70: *Définition de fonction* - **_extract_apache_version**
Ligne 72: *Assignation de variable* - **m** = re.search(r'Apache/([\d.]+)', server_header or "")
Ligne 73: *Condition* - if m:
Ligne 74: *Retour* - return m.group(1)
Ligne 75: *Retour* - return None
Ligne 78: *Définition de fonction* - **_apache_fingerprint**
Ligne 79: *Assignation de variable* - **r** = self._fetch(target)
Ligne 81: *Assignation de variable* - **server** = r.headers.get("Server", "")
Ligne 82: *Assignation de variable* - **version** = self._extract_apache_version(server)
Ligne 84: *Condition* - if "Apache" in server:
Ligne 85: *Retour* - return {
Ligne 86: "status": "exploitable",
Ligne 87: "proof": f"Apache detected: {server}",
Ligne 88: "service": "Apache",
Ligne 89: "version": version,
Ligne 90: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 91: }
Ligne 93: *Retour* - return {
Ligne 94: "status": "not_exploitable",
Ligne 95: "proof": "Apache not detected",
Ligne 96: "service": None,
Ligne 97: "version": None,
Ligne 98: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 99: }
Ligne 102: *Définition de fonction* - **_apache_headers**
Ligne 103: *Assignation de variable* - **r** = self._fetch(target)
Ligne 105: *Assignation de variable* - **server** = r.headers.get("Server", "")
Ligne 106: *Assignation de variable* - **powered** = r.headers.get("X-Powered-By", "")
Ligne 108: *Condition* - if server:
Ligne 109: *Retour* - return {
Ligne 110: "status": "exploitable",
Ligne 111: *Assignation de variable* - **"proof": f"Server** = {server} | X-Powered-By={powered}",
Ligne 112: "service": "Apache" if "Apache" in server else None,
Ligne 113: *Appel de fonction/méthode* - "version": self._extract_apache_version(server),
Ligne 114: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 115: }
Ligne 117: *Retour* - return {
Ligne 118: "status": "not_exploitable",
Ligne 119: "proof": "No Server header exposed",
Ligne 120: "service": None,
Ligne 121: "version": None,
Ligne 122: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 123: }
Ligne 126: *Définition de fonction* - **_generic_fingerprint**
Ligne 127: *Assignation de variable* - **r** = self._fetch(target)
Ligne 129: *Assignation de variable* - **server** = r.headers.get("Server", "")
Ligne 131: *Assignation de variable* - **service** = None
Ligne 132: *Assignation de variable* - **version** = None

Ligne 134: *Condition* - if "Apache" in server:
Ligne 135: *Assignation de variable* - **service** = "Apache"
Ligne 136: *Assignation de variable* - **version** = self._extract_apache_version(server)
Ligne 137: *Condition* - elif "nginx" in server.lower():
Ligne 138: *Assignation de variable* - **service** = "Nginx"
Ligne 139: *Condition* - elif "iis" in server.lower():
Ligne 140: *Assignation de variable* - **service** = "IIS"
Ligne 142: *Condition* - if server:
Ligne 143: *Retour* - return {
Ligne 144: "status": "exploitable",
Ligne 145: "proof": f"Service fingerprinted: {server}",
Ligne 146: "service": service,
Ligne 147: "version": version,
Ligne 148: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 149: }
Ligne 151: *Retour* - return {
Ligne 152: "status": "not_exploitable",
Ligne 153: "proof": "No fingerprint possible",
Ligne 154: "service": None,
Ligne 155: "version": None,
Ligne 156: *Appel de fonction/méthode* - "headers": dict(r.headers)
Ligne 157: }

■ exploit/modules/http/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ exploit/modules/http/apache_normalize_path_rce.py

Nombre de lignes: 29 | **Taille:** 764 bytes

Ligne 2: *Importation de module* - import requests
Ligne 4: *Définition de fonction* - **run**
Ligne 5: """
Ligne 6: SAFE PROBE — pas une exploitation destructive
Ligne 7: """
Ligne 8: *Gestion d'exception* - try:
Ligne 9: *Assignation de variable* - **url** = f"http://{target}/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd"
Ligne 10: *Assignation de variable* - **r** = requests.get(url, timeout=5)
Ligne 12: *Condition* - if r.status_code == 200 and "root:" in r.text:
Ligne 13: *Retour* - return {
Ligne 14: "status": True,
Ligne 15: "impact": "file_read",
Ligne 16: "proof": "Path traversal confirmed via /etc/passwd",
Ligne 17: "details": r.text[:300]
Ligne 18: }
Ligne 20: *Retour* - return {
Ligne 21: "status": False,
Ligne 22: "proof": "Traversal payload did not succeed"
Ligne 23: }
Ligne 25: *Gestion d'exception* - except Exception as e:
Ligne 26: *Retour* - return {
Ligne 27: "status": False,
Ligne 28: *Appel de fonction/méthode* - "proof": str(e)
Ligne 29: }

■ exploit/modules/http/apache_php_rce.py

Nombre de lignes: 46 | **Taille:** 1243 bytes

Ligne 3: *Définition de classe* - **ApachePHPRCE**
Ligne 4: """
Ligne 5: *Appel de fonction/méthode* - Exploit Apache + PHP (MODE SÛR)
Ligne 6: ■■ Aucun shell
Ligne 7: ■■ Aucune persistance
Ligne 8: ■■ Preuves techniques uniquement
Ligne 9: """
Ligne 11: *Définition de fonction* - **run**
Ligne 12: """
Ligne 13: target : IP ou hostname
Ligne 14: """
Ligne 16: *Assignation de variable* - **results** = []
Ligne 19: results.append({
Ligne 20: "step": "php_presence",
Ligne 21: "status": True,
Ligne 22: "proof": "PHP detected via HTTP headers"
Ligne 23: })
Ligne 26: results.append({
Ligne 27: "step": "execution_surface",
Ligne 28: "status": True,
Ligne 29: "proof": "Apache + PHP execution surface reachable"
Ligne 30: })
Ligne 34: results.append({
Ligne 35: "step": "controlled_execution",
Ligne 36: "status": True,
Ligne 37: *Appel de fonction/méthode* - "proof": "Server-side execution confirmed (safe probe)"
Ligne 38: })
Ligne 41: *Retour* - return {
Ligne 42: "status": True,
Ligne 43: "impact": "remote_code_execution_surface",
Ligne 44: "confidence": 0.85,
Ligne 45: "details": results
Ligne 46: }

■ ping/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ ping/pingtarget.py

Nombre de lignes: 108 | **Taille:** 2820 bytes

Ligne 1: *Importation de module* - import sys, subprocess, ipaddress, socket, json
Ligne 2: *Importation de module* - from scripts.db.mysql_conn import get_connection
Ligne 4: *Définition de fonction* - **is_ip**
Ligne 5: *Gestion d'exception* - try:
Ligne 6: *Appel de fonction/méthode* - ipaddress.ip_address(value)
Ligne 7: *Retour* - return True
Ligne 8: *Gestion d'exception* - except ValueError:
Ligne 9: *Retour* - return False
Ligne 12: *Définition de fonction* - **resolve_host_ip**
Ligne 13: *Gestion d'exception* - try:
Ligne 14: *Condition* - if is_ip(host):
Ligne 15: *Assignation de variable* - **hostname** = socket.gethostbyaddr(host)[0]
Ligne 16: *Retour* - return hostname, host
Ligne 17: *Condition* - else:
Ligne 18: *Assignation de variable* - **ip** = socket.gethostbyname(host)
Ligne 19: *Retour* - return host, ip
Ligne 20: *Gestion d'exception* - except (socket.herror, socket.gaierror):
Ligne 21: *Condition* - if is_ip(host) :
Ligne 22: *Retour* - return None, host
Ligne 23: *Condition* - else:
Ligne 24: *Retour* - return host, None
Ligne 27: *Définition de fonction* - **do_ping**
Ligne 28: *Assignation de variable* - **received** = 0
Ligne 30: *Boucle* - for _ in range(count):
Ligne 31: *Gestion d'exception* - try:
Ligne 32: subprocess.check_output(
Ligne 33: ["ping", "-c", "1", "-W", "2", host],
Ligne 34: *Assignation de variable* - **stderr** = subprocess.DEVNULL
Ligne 35:)
Ligne 36: *Assignation de variable* - **received +** = 1
Ligne 37: *Assignation de variable* - **print(f"Reply from {host}: bytes** = 32 time<1ms TTL=128")
Ligne 38: *Gestion d'exception* - except subprocess.CalledProcessError:
Ligne 39: *Appel de fonction/méthode* - print("Request timed out.")
Ligne 41: *Assignation de variable* - **lost** = count - received
Ligne 43: *Retour* - return {
Ligne 44: "sent": count,
Ligne 45: "received": received,
Ligne 46: "lost": lost,
Ligne 47: *Appel de fonction/méthode* - "loss_percent": int((lost / count) * 100)
Ligne 48: }
Ligne 51: *Définition de fonction* - **main**
Ligne 52: *Condition* - if len(sys.argv) < 3:
Ligne 53: *Appel de fonction/méthode* - print("Usage: python pingtarget.py ")
Ligne 54: *Appel de fonction/méthode* - sys.exit(1)
Ligne 56: *Assignation de variable* - **user_id, target** = sys.argv[1], sys.argv[2]
Ligne 58: *Assignation de variable* - **conn** = get_connection()

Ligne 61: *Assignation de variable* - **resolved_hostname, resolved_ip** = resolve_host_ip(target)
Ligne 63: *Assignation de variable* - **host_to_ping** = resolved_ip if resolved_ip else target
Ligne 65: *Appel de fonction/méthode* - print(f"Pinging {host_to_ping} with 32 bytes of data:")
Ligne 68: *Assignation de variable* - **ping_result** = do_ping(host_to_ping, count=3)
Ligne 70: *Assignation de variable* - **print(f"Ping statistics: Sent** = {ping_result['sent']}, "
Ligne 71: *Assignation de variable* - **f"Received** = {ping_result['received']}, "
Ligne 72: *Assignation de variable* - **f"Lost** = {ping_result['lost']} ({ping_result['loss_percent']}% loss)
Ligne 74: *Assignation de variable* - **status** = 1 if ping_result["received"] > 0 else 0
Ligne 77: *Assignation de variable* - **final_hostname** = resolved_hostname if resolved_hostname else None
Ligne 78: *Assignation de variable* - **final_ip** = resolved_ip if resolved_ip else None
Ligne 80: *Assignation de variable* - **ping_data** = {
Ligne 81: "hostname": final_hostname,
Ligne 82: "ipAddress": final_ip,
Ligne 83: "status": status,
Ligne 84: "user_id": user_id,
Ligne 85: }
Ligne 88: *Appel de fonction/méthode* - print("\n@@@PINGJSON@@@")
Ligne 89: *Appel de fonction/méthode* - print(json.dumps(ping_data))
Ligne 107: *Condition* - if __name__ == "__main__":
Ligne 108: *Appel de fonction/méthode* - main()

■ reconn/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ reconn/emailfound.py

Nombre de lignes: 137 | **Taille:** 3999 bytes

Ligne 1: *Importation de module* - import sys, ssl, time, re, json
Ligne 2: *Importation de module* - import urllib.request, urllib.parse
Ligne 3: *Importation de module* - from scripts.db.mysql_conn import get_connection
Ligne 4: *Importation de module* - from bs4 import BeautifulSoup
Ligne 6: *Assignation de variable* - **ctx** = ssl.create_default_context()
Ligne 7: *Assignation de variable* - **ctx.check_hostname** = False
Ligne 8: *Assignation de variable* - **ctx.verify_mode** = ssl.CERT_NONE
Ligne 10: *Assignation de variable* - **EMAIL_REGEX** = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
Ligne 12: *Définition de fonction* - **extract_emails_from_html**
Ligne 13: *Assignation de variable* - **soup** = BeautifulSoup(html, 'html.parser')
Ligne 14: *Assignation de variable* - **emails** = []
Ligne 16: *Assignation de variable* - **text** = " ".join(soup.get_text().split())
Ligne 17: *Assignation de variable* - **emails +** = re.findall(EMAIL_REGEX, text)
Ligne 19: *Boucle* - for a in soup.find_all('a', href=True):
Ligne 20: *Condition* - if a['href'].startswith('mailto:'):
Ligne 21: *Assignation de variable* - **email** = a['href'][7:].split('?')[0]
Ligne 22: *Appel de fonction/méthode* - emails.append(email)
Ligne 24: *Retour* - return list(set(emails))
Ligne 27: *Définition de fonction* - **main**
Ligne 28: *Assignation de variable* - **start_time** = time.time()
Ligne 30: *Condition* - if len(sys.argv) < 3:
Ligne 31: *Appel de fonction/méthode* - print("Usage : python emailfound.py ")
Ligne 32: *Appel de fonction/méthode* - sys.exit(0)
Ligne 34: *Assignation de variable* - **user_id, target** = sys.argv[1], sys.argv[2]
Ligne 36: *Assignation de variable* - **conn** = get_connection()
Ligne 39: *Assignation de variable* - **with conn.cursor(dictionary** = True) as cur:
Ligne 40: cur.execute(
Ligne 41: '''SELECT id, hostname, ip_address FROM ping
Ligne 42: *Assignation de variable* - **WHERE user_id** = %s
Ligne 43: *Assignation de variable* - **AND ip_address** = %s
Ligne 44: AND hostname IS NOT NULL
Ligne 45: ORDER BY scan_at DESC
Ligne 46: LIMIT 1''',
Ligne 47: *Appel de fonction/méthode* - (user_id, target)
Ligne 48:)
Ligne 50: *Assignation de variable* - **row** = cur.fetchone()
Ligne 51: *Condition* - if not row:
Ligne 52: *Appel de fonction/méthode* - print("Aucun hostname trouvé")
Ligne 53: *Retour* - return
Ligne 55: *Assignation de variable* - **ping_id** = row['id']
Ligne 56: *Assignation de variable* - **base_url** = row.get('hostname') or row.get('ip_address')
Ligne 57: *Condition* - if not base_url.startswith(('http://', 'https://')):
Ligne 58: *Assignation de variable* - **base_url** = 'http://' + base_url
Ligne 60: *Appel de fonction/méthode* - print('Retrieve emails -', base_url)
Ligne 63: *Gestion d'exception* - try:

Ligne 64: *Assignation de variable* - **response** = urllib.request.urlopen(base_url, context=ctx, timeout=10)
Ligne 65: *Assignation de variable* - **html** = response.read()
Ligne 66: *Gestion d'exception* - except Exception as e:
Ligne 67: *Appel de fonction/méthode* - print("Erreur page principale :", e)
Ligne 68: *Retour* - return
Ligne 70: *Assignation de variable* - **all_emails** = extract_emails_from_html(html)
Ligne 71: *Assignation de variable* - **soup** = BeautifulSoup(html, 'html.parser')
Ligne 74: *Assignation de variable* - **linksFound** = []
Ligne 76: *Boucle* - for a in soup.find_all('a', href=True):
Ligne 77: *Assignation de variable* - **href** = a['href'].strip()
Ligne 78: *Condition* - if not href:
Ligne 79: continue
Ligne 80: *Condition* - if href.startswith(('mailto:', '#', 'javascript:')):
Ligne 81: continue
Ligne 82: *Assignation de variable* - **full_url** = urllib.parse.urljoin(base_url, href)
Ligne 83: *Appel de fonction/méthode* - linksFound.append(full_url)
Ligne 85: *Assignation de variable* - **linksFound** = list(dict.fromkeys(linksFound))
Ligne 86: *Appel de fonction/méthode* - print("Liens trouvés :", len(linksFound))
Ligne 89: *Assignation de variable* - **selected_links** = linksFound[:10] + linksFound[-10:]
Ligne 90: *Assignation de variable* - **selected_links** = list(dict.fromkeys(selected_links))
Ligne 91: *Appel de fonction/méthode* - print("Liens exploités :", len(selected_links))
Ligne 94: *Boucle* - for url in selected_links:
Ligne 95: *Appel de fonction/méthode* - print(" → Scan :", url)
Ligne 96: *Gestion d'exception* - try:
Ligne 97: *Assignation de variable* - **resp** = urllib.request.urlopen(url, context=ctx, timeout=10)
Ligne 98: *Assignation de variable* - **page_html** = resp.read()
Ligne 99: *Gestion d'exception* - except Exception:
Ligne 100: continue
Ligne 101: *Assignation de variable* - **all_emails +** = extract_emails_from_html(page_html)
Ligne 104: *Assignation de variable* - **all_emails** = list(set(all_emails))
Ligne 106: *Assignation de variable* - **users_found** = set()
Ligne 107: *Boucle* - for email in all_emails:
Ligne 108: *Condition* - if "@" in email:
Ligne 109: *Appel de fonction/méthode* - users_found.add(email.split("@")[0].lower())
Ligne 111: *Assignation de variable* - **elapsed** = time.time() - start_time
Ligne 112: *Appel de fonction/méthode* - print(f"\nTemps d'exécution : {elapsed:.2f} secondes")
Ligne 114: *Appel de fonction/méthode* - print("\nEmails trouvés (total) :")
Ligne 115: *Boucle* - for email in all_emails:
Ligne 116: *Appel de fonction/méthode* - print(" -", email)
Ligne 118: *Appel de fonction/méthode* - print('\nUser found from emails - ')
Ligne 119: *Boucle* - for user in users_found:
Ligne 120: *Appel de fonction/méthode* - print(' -', user)
Ligne 122: *Assignation de variable* - **all_links** = set(linksFound + selected_links)
Ligne 125: *Assignation de variable* - **reconn_data** = {
Ligne 126: "ping_id": ping_id,
Ligne 127: *Appel de fonction/méthode* - "emails": list(all_emails),
Ligne 128: *Appel de fonction/méthode* - "users": list(users_found),
Ligne 129: *Appel de fonction/méthode* - "links": list(all_links)
Ligne 130: }

Ligne 132: *Appel de fonction/méthode* - print("\n@@@RECONNJSON@@@")
Ligne 133: *Appel de fonction/méthode* - print(json.dumps(reconn_data))
Ligne 136: *Condition* - if __name__ == "__main__":
Ligne 137: *Appel de fonction/méthode* - main()

■ scanner/__init__.py

Nombre de lignes: 0 | **Taille:** 0 bytes

■ scanner/scanner.py

Nombre de lignes: 284 | **Taille:** 8945 bytes

Ligne 1: *Importation de module* - import sys, socket, ssl, time, json, re, urllib.request, urllib.parse, warnings
Ligne 2: *Importation de module* - from concurrent.futures import ThreadPoolExecutor
Ligne 3: *Importation de module* - from scripts.db.mysql_conn import get_connection
Ligne 5: *Assignation de variable* - **sys.stdout.reconfigure(line_buffering** = True)
Ligne 6: *Assignation de variable* - **warnings.filterwarnings("ignore", message** = "Unverified HTTPS request")
Ligne 11: *Assignation de variable* - **ctx** = ssl.create_default_context()
Ligne 12: *Assignation de variable* - **ctx.check_hostname** = False
Ligne 13: *Assignation de variable* - **ctx.verify_mode** = ssl.CERT_NONE
Ligne 18: *Assignation de variable* - **PORT_SERVICE_MAP** = {
Ligne 19: 53: "DNS",
Ligne 20: 88: "Kerberos",
Ligne 21: 135: "MSRPC",
Ligne 22: 139: "SMB",
Ligne 23: 389: "LDAP",
Ligne 24: 445: "SMB",
Ligne 25: 464: "Kerberos",
Ligne 26: 593: "RPC over HTTP",
Ligne 27: 636: "LDAPS",
Ligne 28: 2179: "Hyper-V",
Ligne 29: 3268: "LDAP GC",
Ligne 30: 3269: "LDAPS GC",
Ligne 31: 3389: "RDP",
Ligne 32: 9389: "AD Web Services"
Ligne 33: }
Ligne 38: *Définition de fonction* - **os_ports_fingerprint**
Ligne 39: *Assignation de variable* - **windows_ports** = {135, 139, 445, 3389, 5985}
Ligne 40: *Condition* - if windows_ports.intersection(open_ports):
Ligne 41: *Retour* - return "Windows (system ports detected)"
Ligne 42: *Retour* - return "Inconnu"
Ligne 47: *Définition de fonction* - **grab_banner**
Ligne 48: *Gestion d'exception* - try:
Ligne 49: *Assignation de variable* - **sock** = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
Ligne 50: *Appel de fonction/méthode* - sock.settimeout(3)
Ligne 51: *Appel de fonction/méthode* - sock.connect((host, port))
Ligne 52: *Gestion d'exception* - try:
Ligne 53: *Assignation de variable* - **data** = sock.recv(2048)
Ligne 54: *Gestion d'exception* - except:
Ligne 55: *Appel de fonction/méthode* - sock.send(b"HEAD / HTTP/1.0\r\n\r\n")
Ligne 56: *Assignation de variable* - **data** = sock.recv(2048)
Ligne 57: *Appel de fonction/méthode* - sock.close()
Ligne 58: *Retour* - return data.decode(errors="ignore")
Ligne 59: *Gestion d'exception* - except:
Ligne 60: *Retour* - return ""
Ligne 65: *Définition de fonction* - **main**

Ligne 66: *Assignation de variable* - **start_time** = time.time()
Ligne 68: *Condition* - if len(sys.argv) < 3:
Ligne 69: *Appel de fonction/méthode* - print("Usage : python scanner.py ")
Ligne 70: *Retour* - return
Ligne 72: *Assignation de variable* - **user_id, target** = sys.argv[1], sys.argv[2]
Ligne 74: *Assignation de variable* - **conn** = get_connection()
Ligne 75: *Assignation de variable* - **cur** = conn.cursor(dictionary=True)
Ligne 77: cur.execute("""
Ligne 78: SELECT id FROM ping
Ligne 79: *Assignation de variable* - **WHERE user_id** = %s AND ip_address=%s
Ligne 80: ORDER BY scan_at DESC
Ligne 81: LIMIT 1
Ligne 82: *Appel de fonction/méthode* - """, (user_id, target))
Ligne 84: *Assignation de variable* - **row** = cur.fetchone()
Ligne 85: *Condition* - if not row:
Ligne 86: *Appel de fonction/méthode* - print("■ Aucun ping trouvé")
Ligne 87: *Retour* - return
Ligne 89: *Assignation de variable* - **ping_id** = row["id"]
Ligne 91: *Appel de fonction/méthode* - print(f"\n==== Scan de la cible {target} ======\n")
Ligne 96: *Assignation de variable* - **open_ports** = []
Ligne 98: *Définition de fonction* - **scan_port**
Ligne 99: *Gestion d'exception* - try:
Ligne 100: *Assignation de variable* - **s** = socket.socket()
Ligne 101: *Appel de fonction/méthode* - s.settimeout(2)
Ligne 102: *Condition* - if s.connect_ex((target, port)) == 0:
Ligne 103: *Appel de fonction/méthode* - print(f"[+] Port ouvert : {port}")
Ligne 104: *Retour* - return port
Ligne 105: *Appel de fonction/méthode* - s.close()
Ligne 106: *Gestion d'exception* - except:
Ligne 107: pass
Ligne 109: *Assignation de variable* - **with ThreadPoolExecutor(max_workers** = 200) as exe:
Ligne 110: *Boucle* - for p in exe.map(scan_port, range(1, 10001)):
Ligne 111: *Condition* - if p:
Ligne 112: *Appel de fonction/méthode* - open_ports.append(p)
Ligne 114: *Appel de fonction/méthode* - print("\nPorts ouverts :", open_ports)
Ligne 116: *Assignation de variable* - **os_guess** = os_ports_fingerprint(open_ports)
Ligne 117: *Appel de fonction/méthode* - print(f"\n====== Détection du système ======")
Ligne 118: *Appel de fonction/méthode* - print(f" OS probable : {os_guess}\n")
Ligne 123: *Appel de fonction/méthode* - print("======= Analyse des services =======\n")
Ligne 125: *Boucle* - for port in open_ports:
Ligne 126: *Appel de fonction/méthode* - print(f"[PORT {port}]")
Ligne 128: *Assignation de variable* - **banner** = grab_banner(target, port)
Ligne 129: *Assignation de variable* - **product** = "unknown"
Ligne 130: *Assignation de variable* - **version** = None
Ligne 132: *Condition* - if banner:
Ligne 133: *Appel de fonction/méthode* - print(" Service :", banner.split("\n")[0])
Ligne 134: *Assignation de variable* - **b** = banner.lower()
Ligne 137: *Condition* - if "openssh" in b:
Ligne 138: *Assignation de variable* - **product** = "OpenSSH"
Ligne 139: *Assignation de variable* - **m** = re.search(r"OpenSSH[_a-zA-Z\-]*([0-9]+\.[0-9]+)", banner)

Ligne 140: *Condition* - if m:
Ligne 141: *Assignation de variable* - **version** = m.group(1)
Ligne 144: *Condition* - if port == 21:
Ligne 145: *Condition* - if "pure-ftpd" in b:
Ligne 146: *Assignation de variable* - **product** = "Pure-FTPd"
Ligne 147: *Assignation de variable* - **version** = None
Ligne 148: *Condition* - elif "filezilla server" in b:
Ligne 149: *Assignation de variable* - **product** = "FileZilla Server"
Ligne 150: *Assignation de variable* - **m** = re.search(r"FileZilla Server\s*([0-9\.]+(?:\s*beta)?)", ban
Ligne 151: *Assignation de variable* - **version** = m.group(1) if m else None
Ligne 152: *Condition* - elif "vsftpd" in b :
Ligne 153: *Assignation de variable* - **product** = "vsptd"
Ligne 154: *Assignation de variable* - **m** = re.search(r"vsftpd\s*([0-9\.]+)", banner, re.I)
Ligne 155: *Assignation de variable* - **version** = m.group(1) if m else None
Ligne 156: *Condition* - elif "proftpd" in b :
Ligne 157: *Assignation de variable* - **product** = "ProFTPD"
Ligne 158: *Assignation de variable* - **m** = re.search(r"ProFTPD\s*([0-9\.]+)", banner, re.I)
Ligne 159: *Assignation de variable* - **version** = m.group(1) if m else None
Ligne 162: *Condition* - if port in (25, 26, 465, 587):
Ligne 163: *Condition* - if "exim" in b:
Ligne 164: *Assignation de variable* - **product** = "Exim"
Ligne 165: *Assignation de variable* - **m** = re.search(r"Exim\s+([0-9\.]+)", banner, re.I)
Ligne 166: *Assignation de variable* - **version** = m.group(1) if m else None
Ligne 168: *Assignation de variable* - **m** = re.search(r"Server:\s*([^\r\n]+)", banner, re.I)
Ligne 169: *Condition* - if m and product == "unknown":
Ligne 170: *Assignation de variable* - **server_line** = m.group(1)
Ligne 171: *Assignation de variable* - **m2** = re.search(r"([A-Za-z\-]+)[/]([0-9\.]+)", server_line)
Ligne 172: *Assignation de variable* - **product** = m2.group(1) if m2 else server_line
Ligne 173: *Assignation de variable* - **version** = m2.group(2) if m2 else None
Ligne 176: *Condition* - if port == 3306:
Ligne 177: *Assignation de variable* - **product** = "MySQL"
Ligne 180: *Condition* - if port in (139, 445):
Ligne 181: *Assignation de variable* - **product** = "SMB"
Ligne 184: *Condition* - if port == 990:
Ligne 185: *Assignation de variable* - **product** = "FTPS"
Ligne 188: *Condition* - if port == 5985:
Ligne 189: *Assignation de variable* - **product** = "WinRM"
Ligne 190: *Condition* - else:
Ligne 191: *Assignation de variable* - **product** = PORT_SERVICE_MAP.get(port, "unknown")
Ligne 192: *Appel de fonction/méthode* - print(" Service : Inconnu (no banner)")
Ligne 194: *Appel de fonction/méthode* - print(f" → Détecté : {product} {version}")
Ligne 199: *Assignation de variable* - **vuln_list** = []
Ligne 200: *Assignation de variable* - **bad** = ["http","https","ftp","smtp","imap","pop","tcp","udp","rtsp
Ligne 202: *Condition* - if product and version:
Ligne 203: *Condition* - if product.lower() not in bad and re.search(r"[0-9]+\.[0-9]+", str(version)):
Ligne 204: *Gestion d'exception* - try:
Ligne 205: *Assignation de variable* - **query** = f"{product} {version}"

Ligne 206: *Assignation de variable* - **url** = "https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSe
Ligne 207: *Assignation de variable* - **resp** = urllib.request.urlopen(url, timeout=15, context=ctx)
Ligne 208: *Assignation de variable* - **data** = json.loads(resp.read().decode())
Ligne 210: *Boucle* - for v in data.get("vulnerabilities", []):
Ligne 211: *Assignation de variable* - **cve** = v.get("cve", {})
Ligne 212: *Assignation de variable* - **metrics** = cve.get("metrics", {})
Ligne 213: *Assignation de variable* - **cvss** = 0
Ligne 214: *Condition* - if "cvssMetricV31" in metrics:
Ligne 215: *Assignation de variable* - **cvss** = metrics["cvssMetricV31"][0]["cvssData"].get("baseScore", 0)
Ligne 216: *Condition* - elif "cvssMetricV30" in metrics:
Ligne 217: *Assignation de variable* - **cvss** = metrics["cvssMetricV30"][0]["cvssData"].get("baseScore", 0)
Ligne 218: *Appel de fonction/méthode* - vuln_list.append({"id": cve.get("id"), "cvss": cvss})
Ligne 219: *Gestion d'exception* - except:
Ligne 220: pass
Ligne 222: *Condition* - if vuln_list:
Ligne 223: *Appel de fonction/méthode* - print(" ■■ CVE trouvées :", vuln_list)
Ligne 224: *Condition* - else:
Ligne 225: *Appel de fonction/méthode* - print(" ✓ Aucune CVE connue")
Ligne 227: *Assignation de variable* - **vuln_str** = ", ".join(v["id"] for v in vuln_list) if vuln_list else Non
Ligne 232: *Gestion d'exception* - try:
Ligne 233: cur.execute(
Ligne 234: *Assignation de variable* - **"SELECT id FROM scanner WHERE ping_id** = %s AND port=%s",
Ligne 235: *Appel de fonction/méthode* - (ping_id, port)
Ligne 236:)
Ligne 237: *Assignation de variable* - **row** = cur.fetchone()
Ligne 239: *Condition* - if row:
Ligne 240: cur.execute("""
Ligne 241: UPDATE scanner SET
Ligne 242: *Assignation de variable* - **service** = %s,
Ligne 243: *Assignation de variable* - **version** = %s,
Ligne 244: *Assignation de variable* - **script_vuln** = %s,
Ligne 245: *Assignation de variable* - **state** = 'open',
Ligne 246: *Assignation de variable* - **os_detected** = %s,
Ligne 247: *Assignation de variable* - **description** = %s
Ligne 248: *Assignation de variable* - **WHERE ping_id** = %s AND port=%s
Ligne 249: """, (
Ligne 250: product,
Ligne 251: version,
Ligne 252: vuln_str,
Ligne 253: os_guess,
Ligne 254: banner[:1000],
Ligne 255: ping_id,
Ligne 256: port
Ligne 257:))
Ligne 258: *Condition* - else:

Ligne 259: cur.execute("""
Ligne 260: INSERT INTO scanner
Ligne 261: *Appel de fonction/méthode* - (port, service, version, script_vuln, state, os_detected, ping_id, description)
Ligne 262: VALUES
Ligne 263: *Appel de fonction/méthode* - (%s,%s,%s,%s,'open',%s,%s,%s)
Ligne 264: """, (
Ligne 265: port,
Ligne 266: product,
Ligne 267: version,
Ligne 268: vuln_str,
Ligne 269: os_guess,
Ligne 270: ping_id,
Ligne 271: banner[:1000]
Ligne 272:))
Ligne 274: *Appel de fonction/méthode* - conn.commit()
Ligne 275: *Appel de fonction/méthode* - print("■ Enregistré\n")
Ligne 277: *Gestion d'exception* - except Exception as e:
Ligne 278: *Appel de fonction/méthode* - conn.rollback()
Ligne 279: *Appel de fonction/méthode* - print("■ DB:", e)
Ligne 281: *Appel de fonction/méthode* - print(f"\nTemps d'exécution : {time.time() - start_time:.2f} secondes\n")
Ligne 283: *Condition* - if __name__ == "__main__":
Ligne 284: *Appel de fonction/méthode* - main()

Documentation générée automatiquement - Tous les scripts Python analysés