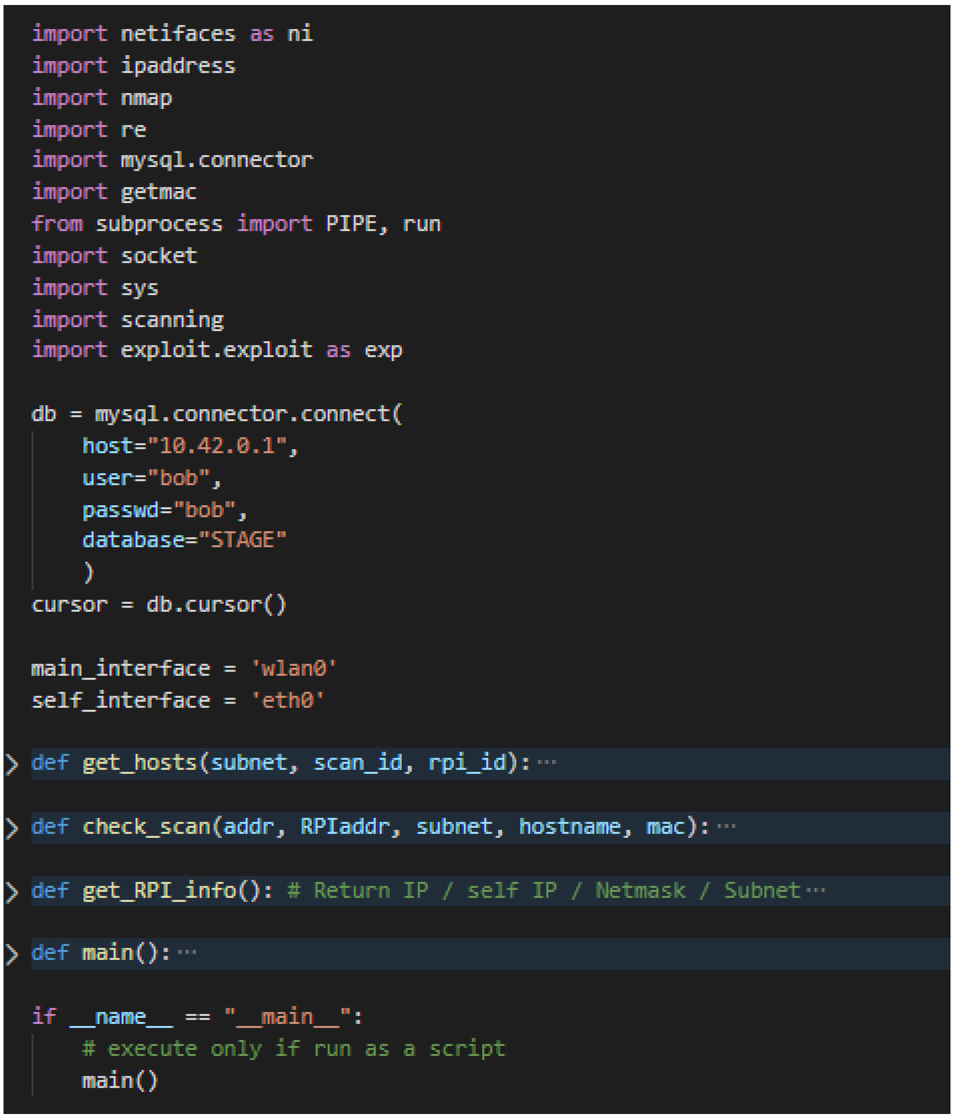
Documentation du code

Serv.py

Structure



```
import netifaces as ni
import ipaddress
import nmap
import re
import mysql.connector
import getmac
from subprocess import PIPE, run
import socket
import sys
import scanning
import exploit.exploit as exp

db = mysql.connector.connect(
    host="10.42.0.1",
    user="bob",
    passwd="bob",
    database="STAGE"
    )
cursor = db.cursor()

main_interface = 'wlan0'
self_interface = 'eth0'

def get_hosts(subnet, scan_id, rpi_id):⋯

def check_scan(addr, RPIaddr, subnet, hostname, mac):⋯

def get_RPI_info(): # Return IP / self IP / Netmask / Subnet⋯

def main():⋯

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

main()

```
def main():

    #1 Récupère les informations du RPI / client
    addr, RPIaddr, subnet, hostname, mac = get_RPI_info()

    #2 Assigne le client à un scan ou le crée s'il n'existe pas
    scan_id, rpi_id, link_id = check_scan(addr, RPIaddr, subnet, hostname,
mac)

    #3 Exécute le script de scanning
    scanning.main(scan_id)
    #4 Exécute le script d'exploitation
    exp.main(scan_id, subnet)

    #5 Définit le scan à l'état fini
    scan_update = "UPDATE SCAN SET STATE = '%s', SCAN_END =
CURRENT_TIMESTAMP WHERE SCAN_id ='%s';"
    rpi_scan_update = "UPDATE RPI_SCAN SET IS_CONNECTED = '%s' WHERE
RPI_id = '%s' AND SCAN_id ='%s';"
    try:
        cursor.execute(scan_update % ('3', scan_id))
        cursor.execute(rpi_scan_update % ('0', rpi_id, scan_id))
        db.commit()
    except:
        print("Database Error On Scan Update")
        sys.exit(0)

    db.close()
```

get_RPI_info()

```
def get_RPI_info(): # Return IP / self IP / Netmask / Subnet

    try :
        # Récupère IP & MASK de l'interface externe
        ni.ifaddresses(main_interface)
        addr = str(ni.ifaddresses(main_interface)[ni.AF_INET][0]['addr'])
        netmask =  str(ni.ifaddresses(main_interface)[ni.AF_INET][0]
['netmask'])

        # Récupère le sous réseau
        addr_mask = addr + '/' + netmask
        subnet = str(ipaddress.ip_network(addr_mask, strict=False))
    except:
        print("no lan connection !!!")
        sys.exit(0)

    # Récupère IP & MASK  de l'interface interne
    ni.ifaddresses(self_interface)
    RPIaddr = str(ni.ifaddresses(self_interface)[ni.AF_INET][0]['addr'])
    RPInetmask =  str(ni.ifaddresses(self_interface)[ni.AF_INET][0]
['netmask'])

    # Récupère HOSTNAME
    hostname = str(socket.gethostname())

    # Récupère MAC adresse
    mac = str(getmac.get_mac_address())

    return addr, RPIaddr, subnet, hostname, mac
```

Check_scan()

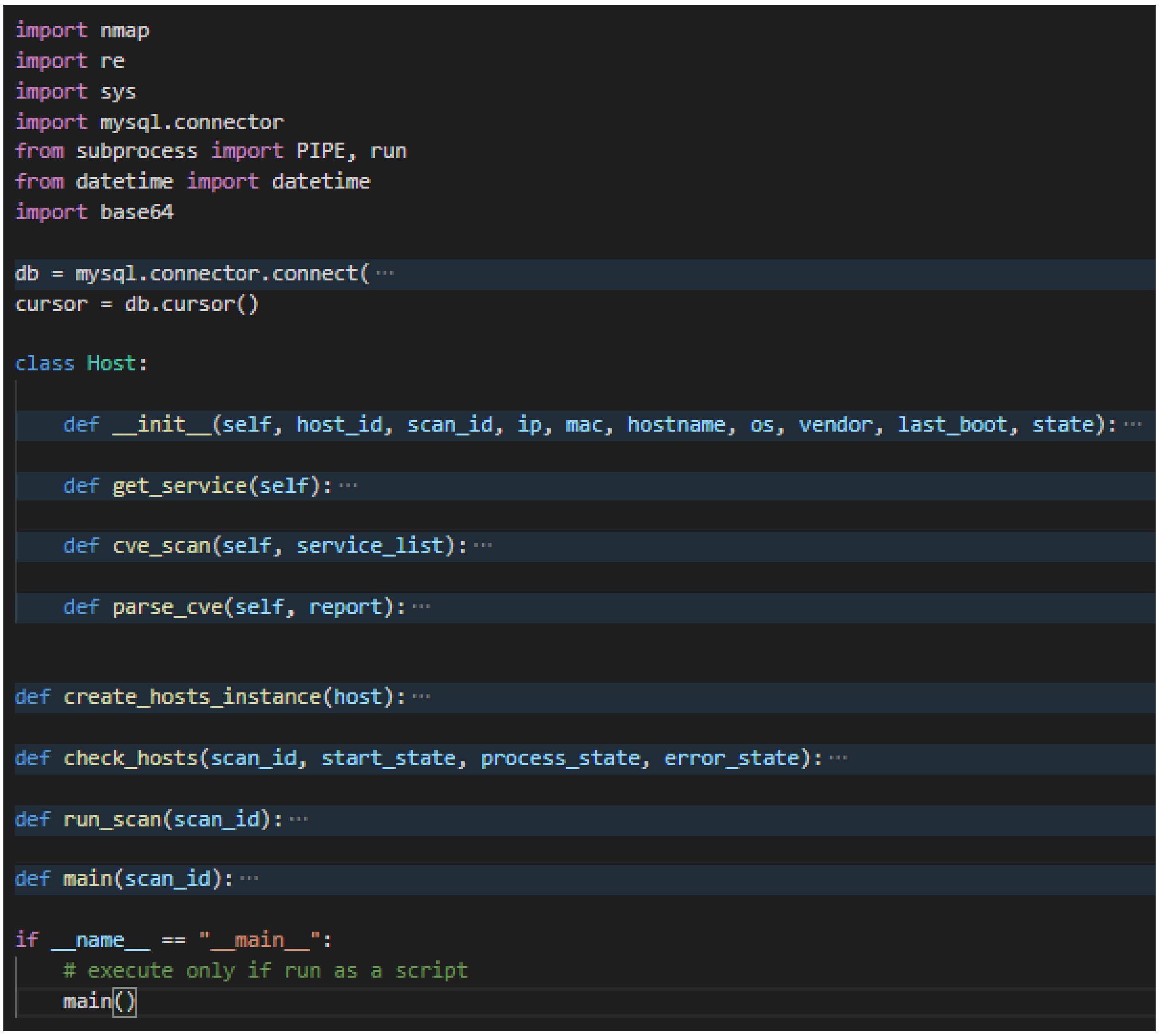
```
def check_scan(addr, RPIaddr, subnet, hostname, mac):

    #1 Vérifie si un scan est déjà en cours dans le même sous-réseau
    scan_select = "SELECT SCAN_id FROM SCAN WHERE SCAN_END IS NULL AND 
STATE = 1 AND SUBNET = '%s';"
    try:
        ...
    except:
        ...
    row = cursor.fetchone()

    #2 Création d'un scan s'il n'en existe pas déjà un
    if not row:
        print("Scan Creation !")
        ...
    else :
        scan_id = row[0]

    #3 Ajout du RPI / client en base de données
    rpi_insert = "INSERT INTO RPI (IP, MAC, HOSTNAME) SELECT '%s', '%s', 
'%s' FROM DUAL WHERE NOT EXISTS( \
        SELECT 1 FROM RPI WHERE IP = '%s' AND MAC = '%s' AND HOSTNAME = 
'%s') LIMIT 1;"
    rpi_select = "SELECT RPI_id FROM RPI WHERE IP = '%s' AND MAC = '%s' 
AND HOSTNAME = '%s';"
    try:
        ...
    except:
        ...

    link_select = "SELECT RPI_id, SCAN_id FROM RPI_SCAN WHERE RPI_id = 
'%s' AND SCAN_id = '%s';"
    try:
        ...
    except:
        ...
    row = cursor.fetchone()

    #4 Création du lien entre le scan et le client
    if not row:
        print("Link Creation")
        link_insert = "INSERT INTO RPI_SCAN (RPI_id, SCAN_id, 
IS_CONNECTED) VALUES ('%s', '%s', '1');"
        try:
            ...
        except:
            ...

        #5 Récupération des hôtes sur le réseau
        get_hosts(subnet, scan_id, rpi_id)
    else :
        link_id = row[0]

    return scan_id, rpi_id, link_id
```

Get_hosts()

```
def get_hosts(subnet, scan_id, rpi_id):

    #1 Utilisation du module NMAP pour scanner le réseau ( simple scan
ping )
    scanner = nmap.PortScanner()
    scanner.scan(hosts=subnet, arguments='-sP')
    hosts = scanner.all_hosts()

    #2 Ajout des hôtes dans la base de données
    for host in hosts:
        host_insert = "INSERT INTO HOST (IP, SCAN_id) SELECT '%s', '%s'
FROM DUAL WHERE NOT EXISTS( \
            SELECT 1 FROM HOST WHERE IP = '%s' AND SCAN_id='%s') LIMIT 1;"
        try:
            cursor.execute(host_insert % (str(host), scan_id, str(host),
scan_id))
            db.commit()
        except:
            db.rollback()
            print("Database Error On Host Creation")
            sys.exit(0)
```

Scanning.py

Structure



```python
import nmap
import re
import sys
import mysql.connector
from subprocess import PIPE, run
from datetime import datetime
import base64

db = mysql.connector.connect(…
cursor = db.cursor()

class Host:

    def __init__(self, host_id, scan_id, ip, mac, hostname, os, vendor, last_boot, state):…

    def get_service(self):…

    def cve_scan(self, service_list):…

    def parse_cve(self, report):…


def create_hosts_instance(host):…

def check_hosts(scan_id, start_state, process_state, error_state):…

def run_scan(scan_id):…

def main(scan_id):…

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

run_scan() & check_hosts()

```
#1 Exécute la fonction check_hosts et définis un flag de contrôle à chacun
#2 Exécute la fonction Create_hosts_instance pour chaque hôte découvert
sur le réseau
```

create_host_instance()

```python
def create_hosts_instance(host):

    #1 Création d'un objet "host"
    host = Host(host[0], host[1], host[2], host[3], host[4], host[5],
host[6], host[7], host[8])

    #2 Fonctions d'analyse de l'hôte
    service_list = host.get_service()
    report = host.cve_scan(service_list)
    host.parse_cve(report)

    return host
```

get_service()

```
def get_service(self):
    service_list = []

    #1 Utilise le module NMAP afin de scanner chacun des hôtes
    scanner = nmap.PortScanner()
    scanner.scan(self.ip, arguments='-sS -O --host-timeout 100')

    #2 Récupération des infos ( uptime, os, os version, vendor, mac
address )
    ...

    host_update = "UPDATE HOST SET MAC = '%s', OS = '%s', VENDOR = '%s',
LAST_BOOT = '%s' WHERE HOST_id = '%s';"

    try:
        ...
    except:
        ...

    #3 Scan et ajoute en BD chacun des ports découverts sur l'hôte
    for proto in scanner[self.ip].all_protocols():
        for port in scanner[self.ip][proto].keys():
            name = scanner[self.ip][proto][port]['name']
            service_list.append([name, port])

    for name, port in service_list:
        service_insert = "INSERT INTO SERVICE (NAME) SELECT '%s' FROM DUAL
WHERE NOT EXISTS( \
            SELECT 1 FROM SERVICE WHERE NAME = '%s') LIMIT 1;"
        host_service_insert = "INSERT INTO HOST_SERVICE (HOST_id,
SERVICE_NAME, SERVICE_PORT) VALUES ('%s', '%s', '%s');"

        try:
            ...
        except:
            ...

    #4 Définit l'état du scan comme terminé
    host_update = "UPDATE HOST SET STATE = 3 WHERE HOST_id = '%s' AND
SCAN_id ='%s';"
    try:
        ...
    except:
        ...

    print("Scan finished for host :", self.ip)

    return service_list
```

cve_scan()

```
def cve_scan(self, service_list):

    #1 Définit le flag de contrôle
    scan_update = "UPDATE SCAN SET VULN_STATE = '%s' WHERE SCAN_id ='%s';"

    ...

    #2 Création d'une liste regroupant les services trouvés
    port_list = []
    for x, y in service_list:
        port_list.append(y)

    port_list = str(port_list).strip('[]')

    #3 Exécution du scan de vulnérabilité sur base de la liste des
services
    try:
        name = "nmap"
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        command = ['nmap', self.ip, '-p', port_list, '--script',
'vulners', '-sV']
        result = run(command, stdout=PIPE, stderr=PIPE,
universal_newlines=True)
        report = result.stdout

        tool_insert = "INSERT INTO TOOL (HOST_id, NAME, COMMAND, RESULT,
TOOL_START) VALUES ('%s', '%s', '%s', '%s', '%s');"

        try:
            #4 Encodage du résultat du scan afin d'éviter les caractères
spéciaux et ajout à la BD
            b64_report = report.encode("utf-8")
            b64_report = base64.b64encode(b64_report)
            b64_report = b64_report.decode("utf-8")
            command = str(command).replace("'", "")
            cursor.execute(tool_insert % (self.host_id, name, command,
b64_report, now))
            db.commit()
        except:
            db.rollback()

    except:
        print("scan crash")
        report = ""

    return report
```

parse_cve()

```
def parse_cve(self, report):

    #1 Utilisation du module regex afin de parser les services et chacune
des cve qui leurs sont associées
    pattern = re.compile(r'(\d*)/(tcp|udp)\s+(open|closed|filtered)\s+
(\w+)(.*)?')
    matches = pattern.finditer(report)

    for match in matches:
        ...

    pattern = re.compile(r'(\w*)/(tcp|udp)\s+(\w+)\s+(\w+-*\w+\/?\w+)\s+
(.*)(\n\|.*)+(CVE-\d{4}-\d{4,5})\s(\d{1,2}\.\d)')
    matches = pattern.finditer(report)

    for match in matches:
        ...

        pattern = re.compile(r'(CVE-\d{4}-\d{4,5})\s(\d{1,2}\.\d)')
        matches = pattern.finditer(match.group(0))

        for match in matches:
            ...
```

---

json_parser.py

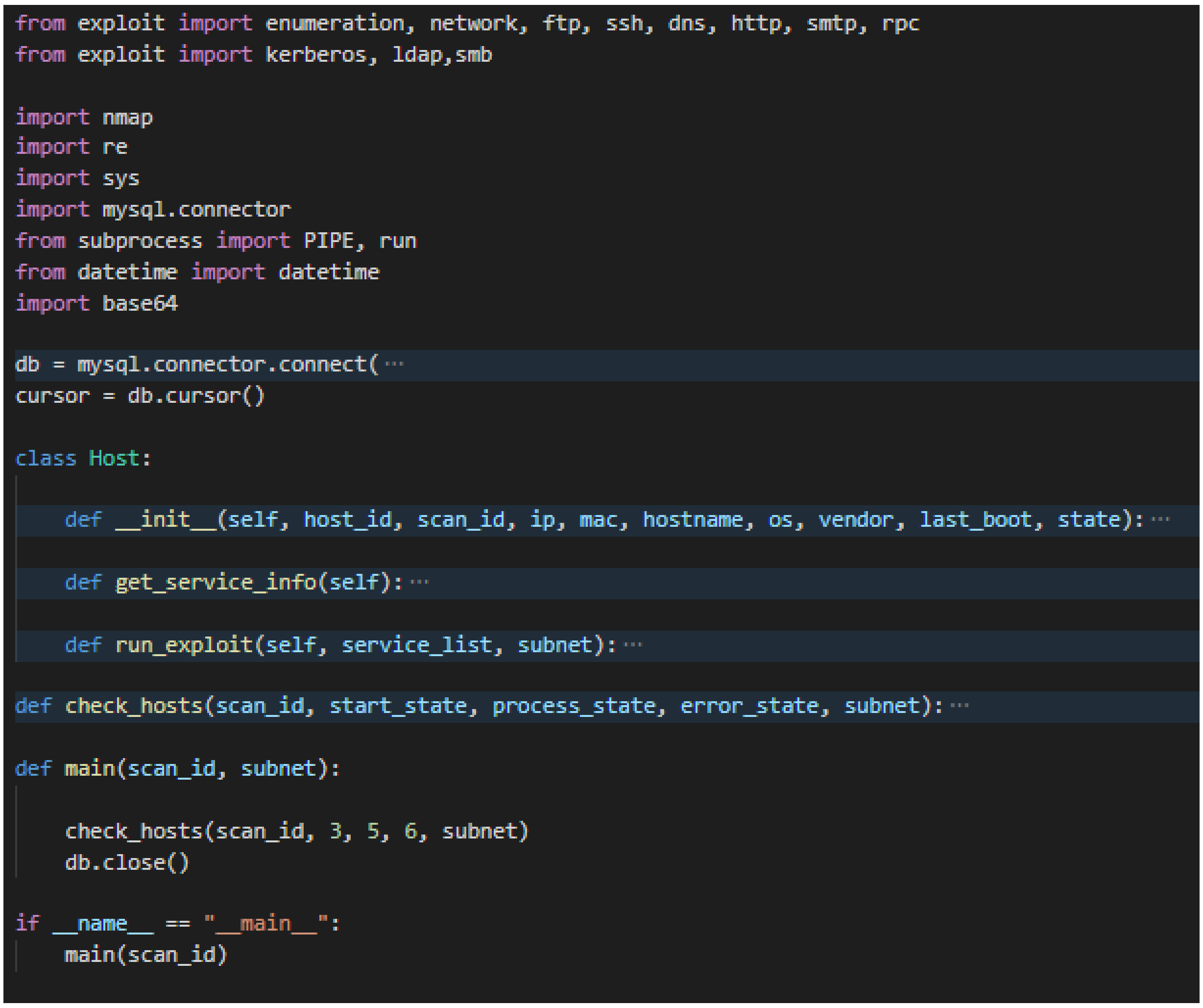
```
import json
import mysql.connector
import glob

...

cursor = db.cursor()
list_full = []

#1 Utilisation du module glob afin de créer une liste contenant des noms
de fichier
file_list = glob.glob("./nvdcve*.json")
print(file_list)

#2 Chacun des fichiers est ouvert en écriture afin d'en extraire les CVE
for file in file_list:
    with open(file) as f:
        data = json.load(f)

    print("\n")

    for cve in data['CVE_Items']:
        cve_ID = cve['cve']['CVE_data_meta']['ID']
        cve_impact = cve['impact']

        for x in cve['cve']['description']['description_data']:
            cve_Description = x['value']

        try:
            cve_BaseScore = cve_impact['baseMetricV3']['cvssV3']
['baseScore']
            cve_complexity = cve_impact['baseMetricV3']['cvssV3']
['attackComplexity']
            cve_severity = cve_impact['baseMetricV3']['cvssV3']
['baseSeverity']
        except:
            try:
                cve_BaseScore = cve_impact['baseMetricV2']['cvssV2']
['baseScore']
                cve_complexity = cve_impact['baseMetricV2']['cvssV2']
['accessComplexity']
                cve_severity = cve_impact['baseMetricV2']['severity']
            except:
                cve_BaseScore = ""
                cve_complexity = "UNKNOWN"
                cve_severity = "UNKNOWN"

        cve_list = [cve_ID, cve_BaseScore, cve_complexity, cve_severity,
cve_Description]
        list_full.append(cve_list)


#3 Ajoute les CVE extraites des fichiers vers la base de données
for cve in list_full:

    query = "INSERT INTO mitre (name, score, severity, complexity,
description) VALUES ('%s','%s', '%s', '%s', '%s');"
    try:
        cursor.execute(query % (cve[0],cve[1],cve[3],cve[2],cve[4]))
        db.commit()
    except:
        db.rollback()

db.close()
```

FEEDS - https://nvd.nist.gov/vuln/data-feeds

---

exploit.py

Structure



```
from exploit import enumeration, network, ftp, ssh, dns, http, smtp, rpc
from exploit import kerberos, ldap,smb

import nmap
import re
import sys
import mysql.connector
from subprocess import PIPE, run
from datetime import datetime
import base64

db = mysql.connector.connect(⋯
cursor = db.cursor()

class Host:

    def __init__(self, host_id, scan_id, ip, mac, hostname, os, vendor, last_boot, state):⋯

    def get_service_info(self):⋯

    def run_exploit(self, service_list, subnet):⋯

def check_hosts(scan_id, start_state, process_state, error_state, subnet):⋯

def main(scan_id, subnet):

    check_hosts(scan_id, 3, 5, 6, subnet)
    db.close()

if __name__ == "__main__":
    main(scan_id)
```

check_hosts

```
def check_hosts(scan_id, start_state, process_state, error_state, subnet):

    #1 Démarrage de la phase d'exploit pour le scan
    scan_update = "UPDATE SCAN SET EXPLOIT_STATE = '%s' WHERE SCAN_id
='%s';"
    try:
        ...
    except:
        ...

    while True:

        #2 Sélectionne les informations du scan et de l'hôte à exploiter
        scan_select = "SELECT * FROM SCAN WHERE SCAN_id = '%s' LIMIT 1;"
        host_select = "SELECT * FROM HOST WHERE SCAN_id = '%s' AND STATE =
'%s' LIMIT 1;"
        host_update = "UPDATE HOST SET STATE = '%s' WHERE HOST_id = '%s';"

        ...

        #3 Crée un objet sur base des informations de l'hôte et lance les
fonctions get_service_info() et run_exploit()
```

```
        if host:

            ...

            host = Host(host[0], host[1], host[2], host[3], host[4],
host[5], host[6], host[7], host[8])
            service_list = host.get_service_info()
            host.run_exploit(service_list, subnet)

        #4 Interrompt l'exploitation en cas d'erreur
        else:

            scan_update = "UPDATE SCAN SET EXPLOIT_STATE = '%s' WHERE
SCAN_id ='%s';"
            try:
                ...
            except:
                ...

            return False
```

get_service_info()

```
    #1 Récupère la liste des services découverts sur l'hôte
    def get_service_info(self):

        service_list = []
        service_select = "SELECT SERVICE_NAME, SERVICE_PORT FROM
HOST_SERVICE WHERE HOST_id = '%s';"

        try:
            cursor.execute(service_select % (self.host_id))
        except:
            print("Database Error On service Select")
            sys.exit(0)

        records = cursor.fetchall()

        for row in records:
            service_list.append([row[0], row[1]])

        return service_list
```

run_exploit()

```
    def run_exploit(self, service_list, subnet):

        ...
        #1 Exécution des scripts ne dépendant pas de service
        result_list = network.main(subnet)
        results.append(result_list)

        result_list = enumeration.main(self.ip)
        results.append(result_list)


        #2 Exécution des scripts propre à chaque service
        for service, port in service_list:

            ...

        #3 Exécution des attaques par brute-force
        if ftp_state == 1:
            result_list = ftp.main(self.ip, ftp_port)
            results.append(result_list)

        if ssh_state == 1:
            result_list = ssh.main(self.ip, ssh_port)
            results.append(result_list)

        #4 Parsing des résultats & push en base de données
        try:
            for x in results:
                for name, command, report in x:
                    ...
        except:
            pass

        #5 Parsing des utilisateurs & push en base de données
        try:
            for x in users:
                for y in x:
                    ...
        except:
            pass

        print("Exploitation finished for host :", self.ip)
```

http.py (Tous les scripts d'exploit ont la même structure)

```
from subprocess import PIPE, run

#2 Fonction d'exploitation
def http_buster(ip, port):

    name = "recursive-gobuster"
    command = ["/home/user/recursive-gobuster/recursive-gobuster.pyz", "-
w", "/usr/share/seclists/Discovery/Web-Content/common.txt", ip]
```

```
    #3 Exécution des tools
    result = run(command, stdout=PIPE, stderr=PIPE,
universal_newlines=True)
    report = result.stdout

    return name, command, report

def http_vuln(ip, port):

    name = "nikto"
    command = ["nikto", "-host", ip, "-port", port, "-Tuning", "x", "6",
"-ask", "no"]
    result = run(command, stdout=PIPE, stderr=PIPE,
universal_newlines=True)
    report = result.stdout

    return name, command, report

def main(ip, port):

    result_list = []

    #1 Exécution des fonctions d'exploitation
    name, command, report = http_buster(ip, port)
    result_list.append([name, command, report])

    name, command, report = http_vuln(ip, port)
    result_list.append([name, command, report])

    return result_list
```