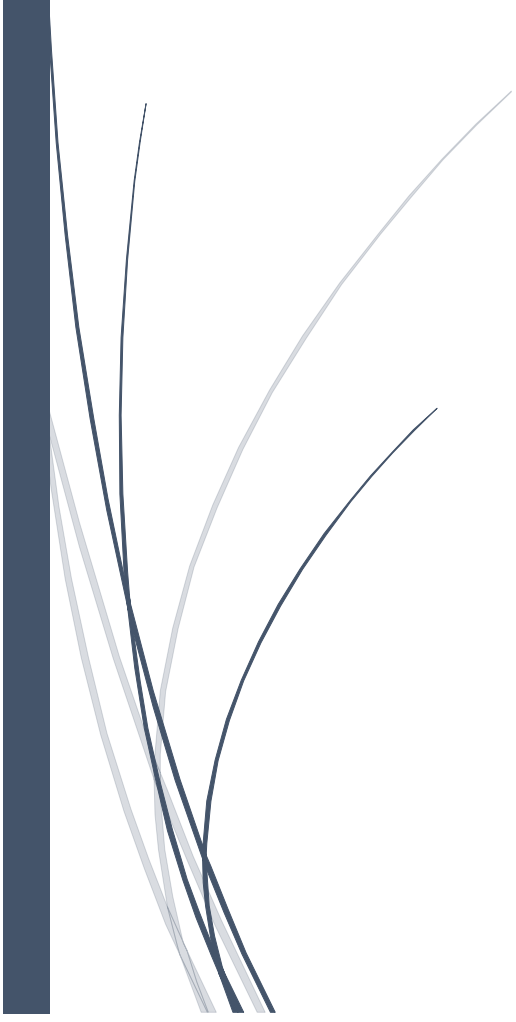
Année académique
2018-2019

Projet Python

DE209 – Développement



BAERT Alexandre et BRAECKMAN Dorian
SECTION 2IR

Introduction

Ce rapport a pour but d'expliquer le fonctionnement d'un script Python permettant d'accéder à un shell sur une machine distante. Nous y présenterons également nos choix concernant le code de ce script. Ce rapport sera divisé en deux parties. La première consistera à présenter le fichier client du programme. Nous y listerons les différentes fonctions que nous avons créées dans celui-ci, en détaillant l'utilité de chacune d'entre elles. La seconde partie portera sur le côté « serveur » du programme, ainsi que sur les différentes fonctions présentes dans celui-ci.

Dans le cadre de ce projet, nous avons choisi d'utiliser le protocole TCP en lieu et place du protocole UDP. En effet, une des fonctionnalités du TCP est la vérification de perte de paquets sur le réseau, ce qui nous semble important dans le cadre d'un programme comme celui présenté dans ce rapport. Nous avons également essayer d'utiliser Cryptodome pour chiffrer les communications sur le réseau.

Fichier client

Nous avons commencé par importer trois librairies : OS, Subprocess et Socket. Là où les deux premières permettent d'interagir avec des systèmes d'exploitations, la troisième permet de communiquer en réseau.

Nous avons ensuite créé différentes fonctions, regroupées sous la fonction « main » qui leur permet de s'exécuter (programmation orientée objet). Chacune de ces fonctions a un rôle particulier et bien défini.

1) create_socket

La fonction « create_socket » rend possible la création d'un compostant aidant à la communication sur le réseau. Concernant ce composant, nous avons déclaré trois variables :

- une variable « host » à laquelle nous avons affecté une adresse IP
- une variable « port » à laquelle nous avons attribué un numéro de port
- une variable « sock » par laquelle nous avons initialisé le socket

De plus, un message (« Error during the creation of the socket ») est envoyé en cas d'erreur durant l'exécution de cette fonction.

2) *socket_connection*

La fonction « socket_connection » utilise les variables de la fonction précédente afin de se connecter au serveur. En cas d'erreur, le message « Error with the connection to the socket » s'affiche.

3) *recv_cmd*

La fonction « recv_cmd » est composée d'une boucle infinie qui permet d'écouter en permanence les instructions reçues de la part du serveur. Chaque commande reçue est vérifiée et renvoie une sortie au format byte et au format string. Le socket permet ensuite d'envoyer le résultat.

Fichier serveur

Dans le cadre du fichier « serveur », nous avons importé cinq librairies : Socket, OS, Subprocess, Threading et Queue. Ces deux dernières permettent au serveur de gérer plusieurs clients à la fois (multi-threading). De plus, concernant Queue, nous n'avons importé que la méthode « queue ».

Nous avons défini trois constantes :

1) THREADS
2) NUMBER
3) COMMANDS

Nous avons ensuite créé trois variables :

1) la variable « queue » à laquelle est affectée la méthode « queue »
2) la variable « connections » qui est une liste qui se rapporte à la connexion elle-même
3) la variable « adresses » qui est une liste d'adresses IP permettant de déterminer quel client se connecte.

Par la suite, nous avons créé huit fonctions.

1) print_help

La fonction « print_help » affiche la constante COMMANDS qui contient une chaîne de caractères expliquant les différents choix proposés : help, list, etc.

2) socket_creation

Cette fonction est similaire à celle présentée dans la partie concernant le fichier client.

3) socket_bind

La fonction « socket_bind » se lance lorsqu'on attend une connexion. Après cinq tentatives infructueuses de connexion, l'accès est refusé et la fonction est relancée.

4) accept_socket

La fonction « accept_socket » accepte la connexion avec les clients et sauvegarde ceux-ci sous forme de liste. Dans le cadre de cette liste, l'adresse IP de chacun des clients est associée à un numéro via une autre liste. En cas d'erreur, le message « Error while accepting the connection » est envoyé.

5) start_prompt

La fonction « start_prompt » représente le shell interactif. C'est dans ce cadre que les différents choix, évoqués dans la fonction « print_help » et dans la constante COMMANDS, sont implémentés.

- « if cmd == 'list' » liste les connexions des clients (leur adresse IP et leur numéro associé).
- « elif 'select' in cmd » permet de sélectionner un client via son numéro associé dans la liste.
- « if cmd == 'quit' » permet de quitter le shell établit avec la machine distante.
- « elif cmd == 'getinfo' » rend possible l'affichage des informations telles que l'adresse IP, etc.
- « elif cmd == 'shutdown' » permet d'éteindre le serveur.
- « elif cmd == 'help' » permet d'afficher l'aide, en faisant appel à la fonction « print_help »

6) create_threads

La fonction « create_threads » rend possible la création de threads.

7) work

La fonction « work » permet de maintenir la connexion et d'envoyer les commandes simultanément.

8) create_jobs

La fonction « create_jobs » associe une place dans la file aux tâches à effectuer en fonction de leur priorité.

Lien Github : https://github.com/etu32270/examen_Dev

Conclusion

Ce projet nous a présenté un certain nombre de difficultés, notamment dans l'implémentation du chiffrement des communications. Nous avons également rencontré un problème avec le module « argparse » que nous avons remplacer par une aide intégrée au shell interactif .