Projet C (Seconde session): Implémentation d'un *Remote Shell*

Adrien Voisin, Didier Valentin, Bastien Bodart
Développement - IR209

Henallux - 2021-2022

1 Introduction

Une fois qu'un attaquant à réussi à exploiter une faille (technique ou humaine) sur un système, la première chose qu'il va souhaiter mettre en place est généralement une invite de commande à distance. Celle-ci lui permettra de lancer des commandes sur la machine de la victime afin d'élever ses privilèges ou d'effectuer des déplacements latéraux. La plupart du temps, l'attaquant va tenter d'utiliser des logiciels préexistants sur la machine attaquée pour mettre en place cette commande à distance (Python, netcat, PHP ou autres). Dans certains cas, il peut s'avérer utile de charger sur la victime un *shell* personnalisé, pour différentes raisons (pour tromper l'antivirus par exemple). L'objectif de ce projet est de réaliser une invite de commande à distance en C qui permettra à l'attaquant de lancer des commandes simples à distance sur une machine pré-infectée.

2 Architecture d'un *remote shell*

Ce projet est constitué d'un serveur (côté victime) et d'un client (côté attaquant). Le serveur est un programme tournant en boucle et écoutant sur un port défini à l'exécution. Il reçoit des messages à travers le réseau depuis le client. Chaque message doit correspondre à une commande système à exécuter. Une fois exécuté, le serveur envoie un message au client avec le résultat de la commande exécutée. Le client est un autre programme que l'attaquant va lancer à sa guise. Celui-ci se connectera au serveur si celui-ci est disponible. L'adresse IP et le numéro de port du serveur seront renseignés à l'exécution du client.

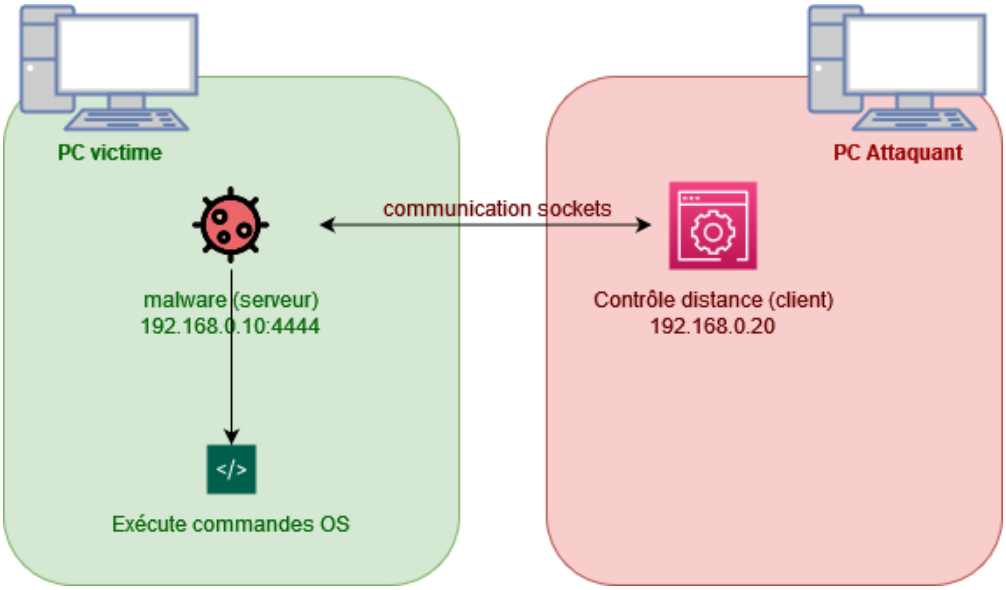

Figure 1: Architecture du système.

Dans l'exemple ci-dessus, on considère que la victime a déjà été infectée par votre *malware* (celui-ci tourne déjà sur la machine). Le client se connecte ensuite au serveur. Une fois la connexion confirmée, l'attaquant pourra entrer via l'invite du client, une commande système de base (whoami, ifconfig, ls pour un système UNIX). Cette commande est envoyée via le réseau (sockets) au serveur. Celle-ci est exécutée sur le système cible. La sortie de cette commande est renvoyée également via les *sockets* au client.

Le code ci-dessous illustre une manière d'exécuter des commandes système. On utilise la fonction *popen* [1] afin de pouvoir récupérer la sortie de la commande exécutée (via le mode 'r'). On lit ensuite le résultat de la commande récupérée dans un *stream*. Ce qui explique l'utilisation du type *FILE*. Dans cet exemple, la commande *whoami* est exécutée. Dans votre programme cet argument passé à la fonction *popen* devra dépendre de ce qui sera reçu via le *socket*. Attention que la fonction *popen* renvoie un flux en bytes et non une chaine de caractères. La nuance est importante car si vous devez manipuler cette information il faut au préalable ajouter le caractère *Null Terminated String*[2] qui permet de signaler au langage que l'on a atteint la fin d'une chaine de caractères. La fermeture du *stream* retourne un code de statut qui permet de vérifier si la commande a rencontré un problème (commande inconnue, permissions incorrectes etc.)

```
#include <stdio.h>
#include <stdlib.h>

#define MAXPATH 2048
```

[1] https://pubs.opengroup.org/onlinepubs/009696799/functions/popen.html
[2] http://www.cs.ecu.edu/karl/2530/spr17/Notes/C/String/nullterm.html

```c
int main( int argc, char *argv[] )
{

  FILE *fp;
  char path[MAXPATH];
  int s, status;
  char c;
  fp = popen("whoami","r");
  if (fp == NULL)
  {
        printf("Failed to run command\n" );
        exit(1);
  }
  s=0;
  // read character by character from the bytes stream until end of file
  while ((c = fgetc(fp)) != EOF)
  {
        printf("%c", c);
        path[s] = c;
        s++;
  }
  path[s]='\0'; // Bytes stream to string
  status = pclose(fp);
  // check status from command execution
  if (status != 0)
  {
        printf("Something went wrong... try again\n");
  }

  return 0;
}
```

3 Utilisation des sockets

Il existe différentes manières d'utiliser les *sockets* pour implémenter ce projet. On vous demande de respecter un fonctionnement particulier afin de simplifier le problème. Il sera nécessaire de mettre en place une communication bidirectionnelle (le client envoie un message au serveur et le serveur répond) et synchrone (le client envoie un message et attend la réponse du serveur avant d'envoyer un nouveau message). Cette implémentation vous permettra d'utiliser un seul *socket* qui sera ouvert à la connexion et fermé lorsque l'utilisateur du client décidera d'arrêter le controle à distance. Le diagramme ci-dessous illustre un échange type.

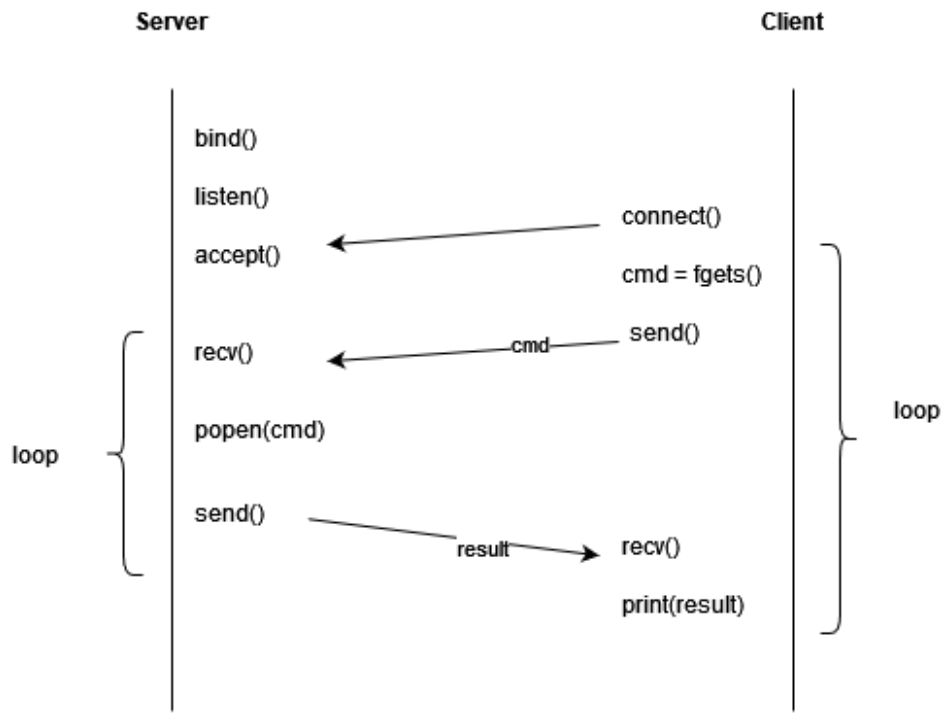

Figure 2: Algorithme d'échange de messages avec les sockets

Attention également que le flux d'information échangé à travers les sockets sont des *bytes* que vous ne pouvez pas directement traiter comme des chaines de caractères. Il faudra au préalable ajouter le caractère de terminaison. Le code ci-dessous illustre une manière de faire:

```
nbr_of_bytes = recv(sockid, bytes_msg, BUFFER_SIZE, 0);
strncpy(string_msg, bytes_msg, nbr_of_bytes);
strings_msg[nbr_of_bytes] = '\0';
printf("%s",strings_msg);
```

Attention également que l'algorithme d'échanges de messages n'est pas complet. En effet il n'est pas précisé comment gérer la fin de la communication de manière propre. Libre à vous de trouver une bonne solution pour la fin d'exécution des deux applications et la fermeture correcte des *sockets*.

4 Evaluation

4.1 Acquis d'apprentissage

L'évaluation a pour objectif de valider les acquis d'apprentissage tels que décrits dans la fiche UE du cours:

- Développer une application C dans le contexte de la sécurité informatique

- Comprendre les enjeux de la gestion de la mémoire en C
- Argumenter les choix d'implémentation

4.2 Critères minimum de réussite

Votre système devra au minimum respecter les contraintes suivantes

- Programmes qui compilent et fonctionnent sans erreurs.
- Respect des consignes de l'énoncé.
- Respect des bonnes pratiques vues durant les cours et laboratoires.
- Le projet se compose de deux programmes C, un jouant le rôle de serveur, un autre jouant le rôle de client. Le programme client est capable d'envoyer des messages au programme serveur à l'aide des *sockets*
- Le programme serveur est capable d'exécuter les commandes de base suivantes: whoami, ifconfig, cat /etc/passwd et de renvoyer le résultat de ces commandes au client qui les affiche à l'écran.
- Le numéro de port et l'adresse IP à utiliser par le serveur pour "binder" le *socket* doivent être renseignés à l'aide des arguments (*argv, argc*). Cette adresse et ce numéro de port devront être renseignés au programme client de la même manière.
- Ces programmes sont compatibles avec un machine Linux type Debian/Kali Linux.
- Respect de la deadline.

4.3 Critères de dépassement

Une fois les critères de base rencontrés, libre aux étudiants de dépasser ces fonctionnalités et d'ajouter une touche personnelle à leur système. Ci-dessous une liste non exhaustive de critères de dépassement :

- Gestion d'un maximum de commandes
- Gestion intelligente des erreurs
- Compatibilité systèmes Windows et Linux
- Chiffrement des communications
- Gestion propre de l'arrêt du *malware*
- Déplacement d'un répertoire à l'autre sur la machine victime
- Gestion des versions [3].

[3] https://github.com/

- Originalité.
- ...

5 Avertissement

Le plus important pour nous est que l'étudiant soit capable de démontrer sa compréhension du code réalisé davantage que l'implémentation d'une multitude de fonctionnalités. Un bon code dont l'étudiant serait incapable de justifier l'implémentation n'est pas suffisant pour réussir. Veillez donc à bien comprendre chaque ligne de code avant d'aller plus loin. Nous préférons de loin un code simple et propre avec le minimum de fonctionnalités de pair avec une bonne défense de l'étudiant plutôt qu'un code complexe difficile à défendre.

6 Modalités pratiques

- Le projet doit être réalisé de manière individuelle.
- L'étudiant doit être capables de défendre chaque partie du système.
- De plus, l'étudiant sera interrogé de manière individuelle sur des questions théoriques liées au cours.
- Toute tentative de triche ou de plagiat sera sanctionnée d'un zéro.
- L'étudiant utilisera deux machines virtuelles pour faire une démonstration de leur système. Attention de travailler avec un réseau virtuel interne pour ne pas dépendre d'une connexion internet le jour de l'examen.
- Le dépôt du projet dans les délais fait office d'inscription à l'examen.

7 Délivrable

Pour le 10/08/2022 à 23h59, l'étudiant devra remettre un délivrable contenant les éléments suivants sur Moodle:

- Le code source de votre programme
- Référence éventuelle à un dépôt *Git*
- Une documentation aussi complète que possible expliquant le fonctionnement et les choix d'implémentation du système. Cette documentation fera office de rapport pour le projet.

8 Le plagiat

Attention de toujours bien indiquer vos références concernant les parties de code dont vous n'êtes pas l'auteur[4].

[4]https://fr.wikipedia.org/wiki/Plagiat