TP de *Systèmes Unix*
Réalisation d'un système *Shell sur Netcat*

Quentin Davin & Firmin Launay

ESIREM — Première année de prépa intégrée, semestre 2

Juin 2022

L'objectif du projet réalisé au cours des quatre séances de TP de *Systèmes Unix* est de créer un système *Shell sur Netcat* (nous abrégerons cela par `ssn` par la suite), à l'aide du shell Unix, et surtout, comme son nom l'indique, de `nc`. Cela consiste en un serveur sur lequel on peut se connecter, s'identifier à l'aide d'un mot de passe et d'une clé de chiffrement monoalphabétique : une fois cette étape passée, on peut exécuter des commandes sur la machine hôte, comme si on y était connectés par l'intermédiaire d'un tunnel `ssh`.

1 Écoute du port par Netcat

La première partie du TP, dont le code est ci-dessous, consistait à configurer Netcat pour écouter en continu le port 12345, afficher la date, l'heure et un message de bienvenue à la connexion.

```
1  #!/bin/bash
2
3  rm -f ./fifo
4  mkfifo ./fifo
5
6  function interpret() {
7    current_date=$(LANG=en_us_88591;date)
8
9    echo $current_date
10   echo "Welcome to Quentin and Firmin's server!"
11 }
12
13 echo "[LOG] Server starting..."
14 while true
15 do
16   nc -l localhost 12345 < ./fifo | interpret &> ./fifo
17 done
18 echo "[LOG] Server ending..."
```

2 Exécution de ligne de commande

Dans cet exercice, on doit exécuter les lignes envoyés par le client et lui renvoyer la sortie.

On utilise alors réellement le fichier FIFO (*first in, first out*), qui permet de gérer les flux de texte en entrée et en sortie du serveur.

Puis, nous avons ajouté, à la fin de la fonction `interpret`, une boucle `while`, dont le code est ci-dessous, qui lit toutes les lignes d'entrée depuis le FIFO, les exécute dans la session `bash` courante, puis réaffiche un prompt pour l'utilisateur (sous la forme d'un `$`, afin de l'informer que le serveur écoute la prochaine commande.

Il faut par ailleurs être bien attentif à rediriger non seulement la sortie standard, mais également la sortie d'erreur de la fonction `interpret` vers le FIFO, avec l'opérateur `&>` et pas seulement `>`. Cela permet à l'utilisateur de voir tous les messages du terminal, les succès et informations comme les erreurs.

```
echo -n " \$ "
while read line
do
  $line
  echo -n " \$ "
done
```

3 Identification par mot de passe

Afin d'implémenter l'identification par mot de passe, il faut le stocker dans un fichier côté serveur.

Pour des raisons de sécurité, nous avons choisi de ne pas enregistrer les mots de passe en clair, mais plutôt un hash de ceux-ci. Un *hash*, ou *somme de contrôle* est une longue chaîne de caractères obtenue à partir d'une autre chaîne de caractère – en l'occurence, le mot de passe à hasher – à l'aide d'opérations mathématiques complexes et théoriquement non réversibles. Chaque hash est presque unique : très peu de chaînes de caractère donneront le même hash (2^{256} hashs possibles dans le cas de l'algorithme SHA-256, que nous utilisons) : il est donc presque impossible de retrouver la chaîne originale à partir du hash.

Afin de faciliter le processus, nous avons créé un script `save_password_hash.sh` afin de générer le hash d'un mot de passe spécifié, dont le code est donné ci-dessous.

Nous avons également créé une fonction `hash_password` dans notre fichier serveur, également donnée ci-dessous, et avons implémenté la réception du mot de passe.

Fichier `save_password_hash.sh`

```
#!/bin/bash

if [ $# -ne 2 ]
then
  >&2 echo "Usage: ./save_password_hash.sh [password] [password_file]"
  exit 1
fi

pwd_hash_full=$(echo "$1" | sha256sum)
pwd_hash=$(echo $pwd_hash_full | rev | cut -c3- | rev)
echo "$pwd_hash" > "$2"
echo "Your password was registered successfully."
```

Extrait du serveur

```
function hash_password () {
    local pwd_hash_full=$(echo $1 | sha256sum)
    local pwd_hash=$(echo $pwd_hash_full | rev | cut -c3- | rev)
    echo "$pwd_hash"
}
```

Extrait de la fonction `interpret` *du serveur*

```
while read line;
do
for text_line in "$(cat "password.conf")"
do
  pwd_hash=$(hash_password "$line")
  if [ "$text_line" == "$pwd_hash" ]
  then
    success="1"
    break
  else
    success="0"
  fi
done
if [ $success == "1" ]
then
  echo "The password is correct. You are now logged in."
  break
else
  echo -n "The password is incorrect. Please enter your password: "
fi
done
```

5 Sécurisation par chiffrement monoalphabétique

Nous avons utilisé le chiffrement monoalphabétique pour sécuriser la connexion, à l'aide d'une fonction `crypt` et d'une fonction `decrypt`, aux versions différentes côté client et serveur, données ci-dessous. Le fonctionnement complet du chiffrement – déchiffrement est visible dans la version finale du projet.

```
function srv_decrypt() {
    echo "$1" | tr "$code" "A-Za-z"
}

function srv_crypt() {
  if [ "$2" == "-n" ]
  then
    encrypted=$(echo "$1" | tr "A-Za-z" "$code")
    echo "$encrypted EOF"
  else
    encrypted=$(echo "$1" | tr "A-Za-z" "$code")
    echo "${encrypted}@"
  fi
}
```

```
15
16 function client_crypt(){
17     exit=$(echo "$1" | tr "A-Za-z" "$code")
18     echo $exit
19 }
20
21 function client_decrypt(){
22   exit=$(echo "$1" | tr "@" "\n" | tr "$code" "A-Za-z")
23   echo -n "$exit"
24 }
```

Version finale

La version finale s'utilise de la façon suivante :

1. Télécharger l'archive depuis GitHub : https://github.com/Firmin-ESIREM/Projet-TP-Netcat/archive/refs/heads/main.zip.
2. La décompresser.
3. Créer une clé de chiffrement monoalphabétique et la mettre dans un fichier texte. Par exemple, on peut créer un fichier `code.key`, avec pour contenu :

   ```
   AZERTYUIOPQSDFGHJKLMWXCVBN
   ```

4. Créer un mot de passe et appeler le script `save_password_hash.sh` de la manière suivante :

   ```
   ./save_password_hash.sh [votre_mot_de_passe] password.conf
   ```

5. Dès lors, le serveur peut être lancé avec la commande :

   ```
   ./ssnsrv code.key password.conf
   ```

   Le port à écouter peut être spécifié en troisième argument, mais il s'agit par défaut du port 12345.
6. Enfin, on peut lancer le client avec la commande :

   ```
   ./ssn code.key localhost
   ```

   Le port auquel se connecter peut là aussi être spécifié en troisième argument, mais il s'agit par défaut toujours du port 12345.
7. Vous êtes connecté ! Le mot de passe va vous être demandé, et la clé de chiffrement sera automatiquement transmise. Vous pourrez ensuite exécuter des commandes simples (et non interactives) à distance !

Vous pouvez nous contacter aux adresses suivantes :
Quentin_Davin@etu.u-bourgogne.fr
Firmin_Launay@etu.u-bourgogne.fr