**Compte rendu Tp Unix**

**Préparé par :**

Eddallel Mohamed malek

Grosjean Guillaume

**Exercice 1 - Serveur Basique**

Etapes pour lancer un serveur

Pour lancer le serveur on cré un fichier texte avec un interpréteur bash

1. On le sauvegarde et le rendre exécutable avec le chmod +x nom\_serveur.sh
2. Et pour le lancer on tape ./ nom\_serveur.sh

**Structure Globale**





* **FIFO (Tube Nommé)** : Permet une communication bidirectionnelle via nc
* **Netcat (nc)** :
  + -l mode écoute
  + -s spécifie l'adresse source
  + -p spécifie le port
* **Redirections** :
  + L'entrée de nc vient du FIFO (< ./fifo)
  + La sortie est redirigée vers la fonction interpret

#!/usr/bin/bash

→ Déclare l’interpréteur utilisé (ici Bash).

rm -f ./fifo

mkfifo ./fifo

→ Supprime (rm -f) le FIFO nommé s’il existe déjà, puis crée un nouveau fichier nommé fifo. Ce fichier est utilisée pour faire communiquer deux processus via un canal nommé.

function interpret () {

→ Définition d’une fonction interpret exécutée lors d'une connexion client.

echo "Bienvenue, utilisateur."

→ Accueil de l’utilisateur connecté.

CONN\_TIME=$(date '+%Y-%m-%d %H:%M:%S')

echo "Date de connexion : $CONN\_TIME"

→ Enregistre et affiche la date de connexion.

while read line; do

echo "Debug, reçu == $line ==" >&2

→ Boucle infinie lisant les lignes envoyées par le client. Affiche chaque ligne reçue sur la sortie d’erreur (>&2 pour débogage).

if [ "$line" = "exit" ]; then

DISCONN\_TIME=$(date '+%Y-%m-%d %H:%M:%S')

echo "Date de déconnexion : $DISCONN\_TIME"

exit

fi

→ Si l’utilisateur envoie exit, on enregistre l’heure de déconnexion et on quitte la fonction (et donc termine la session).

done

}

→ Fin de la fonction interpret.

while true; do

echo "En attente de connexion..."

nc -l -s localhost -p 12345 < ./fifo | interpret > ./fifo

echo "Connexion terminée"

done

→ Boucle serveur infinie

Attend une connexion sur localhost:12345.

nc lit depuis le FIFO et pipe les données dans interpret.

Les réponses de interpret sont envoyées vers ./fifo, donc vers nc

But

Créer un serveur basique, qui affiche un message de connexion et accepte des messages jusqu’à ce que "exit" soit reçu, en restant toujours en écoute si le client déconnecte sur le port 12345

# ****Exercice 2 - Ajout d'Exécution de Commandes****

## ****Modifications Principales****



## ****Explications****

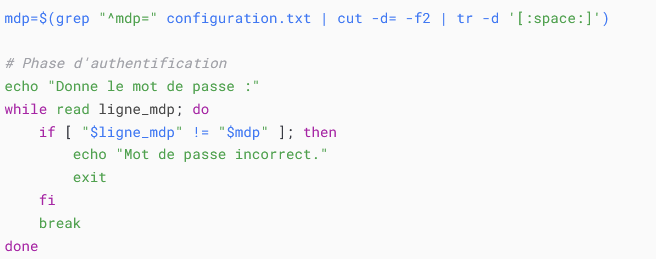
* eval permet d'exécuter n'importe quelle commande Bash

Ce script est **presque identique au premier**, sauf qu’on exécute les lignes reçues comme des **commandes shell** :

→ Chaque commande est interprétée par Bash grâce à eval, et son résultat est envoyé au client. Cela peut être dangereux (permet l'exécution de n'importe quelle commande shell).

# ****Exercice 3 - Authentification par Mot de Passe****

## ****Nouveautés****



## ****Analyse****

* Lecture du mot de passe dans un fichier de configuration
* tr -d '[:space:]' supprime les espaces/tabulations

Ajout d’un fichier configuration.txt contenant un mode passe :exemple (mdp=monsecret)

mdp=$(grep "^mdp=" configuration.txt | cut -d= -f2 | tr -d '[:space:]')

→ Récupère le mot de passe du fichier de config (ligne commençant par mdp=).

echo "Donne le mot de passe :"

while read ligne\_mdp; do

if [ "$ligne\_mdp" != "$mdp" ]; then

echo "Mot de passe incorrect."

exit

fi

break

done

→ On demande le mot de passe au client. Si incorrect : on termine la session.

But

Sécuriser le serveur avec un mot de passe en clair, défini dans un fichier de configuration. Le reste du fonctionnement est identique à l’exercice 2.

# ****Exercice 4 - Chiffrement par Substitution****

## ****Partie Serveur****

## ****Partie Client****



## ****Fonctionnalités Avancées****

1. **Coprocessus (coproc)** :
   * Permet une communication asynchrone
   * ${NET[1]} = descripteur d'écriture
   * ${NET[0]} = descripteur de lecture
2. **Chiffrement Caesar Étendu** :
   * Substitution mono-alphabétique via tr
   * Clé de 26 caractères (ex: "bacdefghijklmnopqrstuvwxyz")
3. **Workflow Sécurisé** :
   * Toutes les communications sont chiffrées
   * Le client déchiffre les réponses du serveur

# ****Analyse Globale des Choix Techniques****

1. **FIFO** :
   * Nécessaire pour la communication bidirectionnelle
   * Alternative aux sockets réseau directes
2. **Netcat** :
   * Outil polyvalent pour le réseau
   * Permet un protocole client-serveur rapide
3. **Architecture Modulaire** :
   * Fonction interpret comme contrôleur principal
   * Séparation claire des couches

Le fichier configuration.txt contient maintenant le clef de cryptage aussi :

mdp=motdepasse

cle =zyxwvutsrqponmlkjihgfedcba

clef : correspondance chiffrée de a-z, pour un **chiffrement par substitution monoalphabétique**.

**Coté serveur**

encrypt() {

echo "$1" | tr 'a-z' "$KEY"

}

decrypt() {

echo "$1" | tr "$KEY" 'a-z'

}

→ Fonctions pour chiffrer/déchiffrer un texte en remplaçant chaque lettre

echo $(encrypt "Donne le mot de passe :")

→ Le serveur demande le mot de passe de manière chiffrée.

if [ "$(decrypt "$ligne\_mdp")" != "$mdp" ]; then

→ Le mot de passe reçu est déchiffré, puis comparé au mot de passe dans le dossier configuration.txt

DECRYPTED\_LINE="$(decrypt "$line")"

result=$(eval "$DECRYPTED\_LINE")

echo $(encrypt "$result")

→ Les commandes envoyées par le client sont :

Déchiffrées → Exécutées→ Résultat chiffré renvoyé

**Coté client**

coproc NET { nc localhost 12345; }

→ Lance nc en coproc (processus coopératif). On utilise :

NET[1] pour écrire

NET[0] pour lire

**Explication du choix coproc**

**Coproc ;** exécute commande dans un sous-processus en arrière-plan.On communique avec le serveur de manière asynchrone mais bien structurée grâce à ces deux flux.

 Il permet de **garder un dialogue continu** avec le serveur (comme un terminal interactif).

 on peut **écrire et lire** indépendamment sans bloquer le script.

 Il est **plus propre** que des solutions manuelles avec des FIFOs pour ce type de client TCP.

Clé et mot de passe :

read -p "Mot de passe : " password

read -p "Clé de chiffrement (26 lettres) : " KEY

→ L'utilisateur entre la clé de substitution.

Dialogue :

echo "$(encrypt $password)" >&"${NET[1]}"

→ Envoie le mot de passe chiffré.

read -u ${NET[0]} encrypted\_response

echo "$(decrypt "$encrypted\_response")"

→ Réception et déchiffrement des messages serveur.

read -p "Entrer une commande: " input

encrypted=$(encrypt "$input")

echo "$encrypted" >&${NET[1]}

….

…

response=$(decrypt "$encrypted\_response")

→ Commandes saisies → chiffrées → envoyées → réponses reçues → déchiffrées