TP/Projet RSA – Programmation Socket
TELECOM Nancy 2A

1. Objectifs du TP

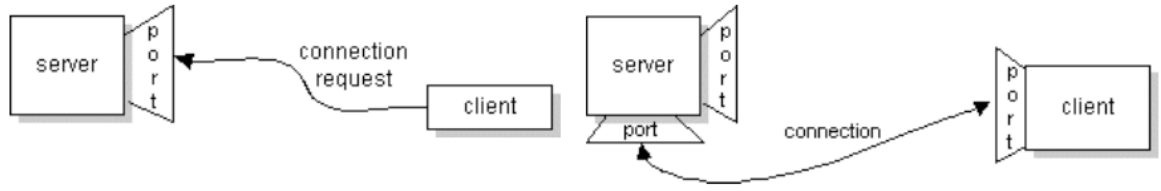
L'objectif de ce TP est de découvrir la programmation socket en langage C, avec le développement de couples client-serveur pour les protocoles de transport TCP et UDP.

Vous déposerez individuellement à la fin de chaque TP les fichiers développés en C dans le dépôt correspondant sous Arche, avec un fichier readme.txt indiquant synthétiquement le travail réalisé et son fonctionnement.

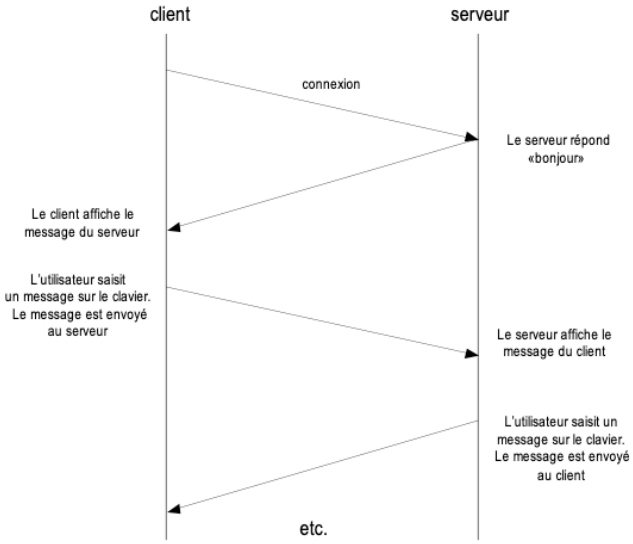
Le projet (section 4) correspond à un projet à réaliser en binôme, et à rendre dans le dépôt correspondant sous Arche, avec un fichier readme.txt indiquant synthétiquement le travail réalisé et son fonctionnement. La date limite de rendu est fixé au **14 mai 2025**.

2. Protocole TCP (communication en mode connecté)

Dans le mode connecté, les données sont transmises via un canal de communication entre deux entités (l'une client, l'autre serveur). Ce canal est appelé, dans le cas de TCP, une connexion.
La connexion nécessite d'être établie et maintenue entre ces deux entités, ce qui génère plus de traitement que dans le cas de la communication par datagramme (UDP).
TCP s'inscrit dans le cadre d'une communication de type client-serveur. Concrètement, cela sous-entend que le client fait appel à des services du serveur qui traite les demandes et retourne le résultat des requêtes. La communication client-serveur repose sur différentes étapes : le serveur **ouvre une socket**, lie la socket à l'adresse locale sur **un de ses ports**, se place **en écoute** de demande de connexion, le client **ouvre une socket** et **se connecte**, le serveur **accepte** et le **dialogue** peut commencer avec le client. La notion de numéro de port permet à TCP de retrouver l'application en question.



Le scénario de la connexion est donc le suivant :



Développer un serveur (serveurTCP.c) et un client (clientTCP.c) qui permettent à deux utilisateurs de discuter à distance (cf. figure). Vous commencerez par développer le serveur. Vous le testerez en utilisant telnet en tant que client.

Une fois le serveur opérationnel, vous pourrez développer le client en C. Pour obtenir l'adresse IP du serveur, vous pourrez utiliser successivement les fonctions inet_addr() et gethostbyname().

3. Protocole UDP (communication en mode datagramme)

Certaines applications ne requièrent pas le canal de communication fourni par TCP. De plus, elles peuvent spécifier un mode de communication qui ne garantisse pas l'ordre d'arrivée des messages. Le protocole UDP fournit ce mode de communication réseau où les applications envoient des paquets de données appelés datagramme. Un datagramme est un message indépendant pour lequel ni son arrivée, ni son délai de transmission, ni la protection de ses données sont garanties.

Le principe de communication par datagramme est plus simple : un client doit récupérer l'adresse d'un serveur (adresse IP + numéro de port pour UDP), puis lui envoyer un paquet (datagramme) de données.

Dans ce cadre, le programme serveur **ouvre une socket** et se met en **attente de lecture** sur un port UDP. Le client **ouvre une socket**. Il **construit ensuite un datagramme** en précisant cette fois outre les données, l'adresse de destination ainsi que le numéro de port visé. Le serveur recevant le message détient alors la possibilité de répondre au client en récupérant l'adresse du client du paquet reçu.

Développer un couple client-serveur UDP en C (nommés serveurUDP.c et clientUDP.c). Le client envoie l'heure au serveur et quitte. Le serveur affiche l'heure reçue et quitte. Utiliser les fonctions ftime() et ctime().

4. Projet : conception d'un orchestrateur de scanners de vulnérabilités

L'objectif de ce projet est de concevoir et implémenter un **orchestrateur** permettant de **contrôler à distance plusieurs scanners de vulnérabilités** (à savoir **Nmap, OWASP ZAP et Nikto**). Cet orchestrateur doit permettre l'exécution, la collecte et la centralisation des résultats des analyses de sécurité sur un ou plusieurs hôtes cibles.

Fonctionnalités attendues

1. **Orchestration des scanners**
    - L'orchestrateur doit être capable d'envoyer des commandes aux scanners distants et de récupérer leurs résultats.
    - Les scanners supportés sont **Nmap**, **OWASP ZAP** et **Nikto**.
    - Il doit permettre de configurer les paramètres de scan (cibles, types de tests, options spécifiques à chaque outil).
2. **Agent distant pour exécuter les scans**
    - Un **agent distant** devra être développé pour exécuter les scanners sur différentes machines.
    - L'agent devra recevoir des instructions depuis l'orchestrateur et lui transmettre les résultats après exécution.
    - La communication entre l'orchestrateur et les agents devra être chiffrée pour garantir la confidentialité et l'intégrité des échanges**.**
3. **Collecte et synthèse des résultats**
    - L'orchestrateur doit agréger les résultats des différents scanners.
    - Il doit fournir une vue synthétique des vulnérabilités détectées.

Autres consignes

- Le projet doit être **développé en C** avec une architecture reposant sur un **orchestrateur central** et des **agents distants**. La communication entre l'orchestrateur et les agents doit être **sécurisée**.
- **Important** : il est impératif que toutes les expérimentations soient effectuées dans un **environnement contrôlé et isolé**. Il est strictement interdit d'utiliser les scanners sur un réseau réel, que ce soit un réseau de l'école, d'un hébergeur, ou d'Internet. Pour éviter tout impact involontaire sur des systèmes externes, vous devez obligatoirement utiliser une **infrastructure virtualisée** comme **docker**, une machine virtuelle locale (VirtualBox, VMware) ou un **cyber-range** dédié.
- L'objectif est de comprendre le fonctionnement de scanners dans un cadre pédagogique et sécurisé, sans causer de perturbations ou de dommages. Toute infraction à cette règle peut avoir des **conséquences légales et disciplinaires**.

Références

- **Scanner Nmap** : https://nmap.org/
- **Scanner OWASP ZAP** : https://www.zaproxy.org/
- **Scanner Nikto** : https://cirt.net/nikto2