جـامعة الحسن الثـاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

UNIVERSITE HASSAN II DE CASABLANCA

ECOLE NATIONALE SUPERIEURE D'ELECTRECITE ET DE LA MECANIQUE ENSEM - CASABLANCA

ENSEM

DEPARTEMENT INFORMATIQUE:

GENIE LOGICIEL ET DIGITALISATION

Rapport
Mini Projet
Partie 2

REALISÉ PAR:

- El Kacimi Younes

PROFESSEUR ENCANDRANT:

PR. Boukhdir Khalid

ANNÉE UNIVERSITAIRE:

2022/2023

Développement d'une application multithreaded pour la détection des attaques DDoS à partir d'un dataset d'un trafic réseau

Table de matière:

I. Introduction :

La programmation multithreaded dans les systèmes informatiques modernes est cruciale. En permettant l'exécution simultanée de tâches parallèles, les applications multithreaded améliorent les performances, mais elles posent également des problèmes en termes de synchronisation et de gestion des ressources partagées.

Dans ce rapport, nous présentons notre application multithreaded conçue pour détecter les attaques de déni de service distribué aussi connu sous le nom de DDoS.

II. Déni de service distribué(DDoS) :

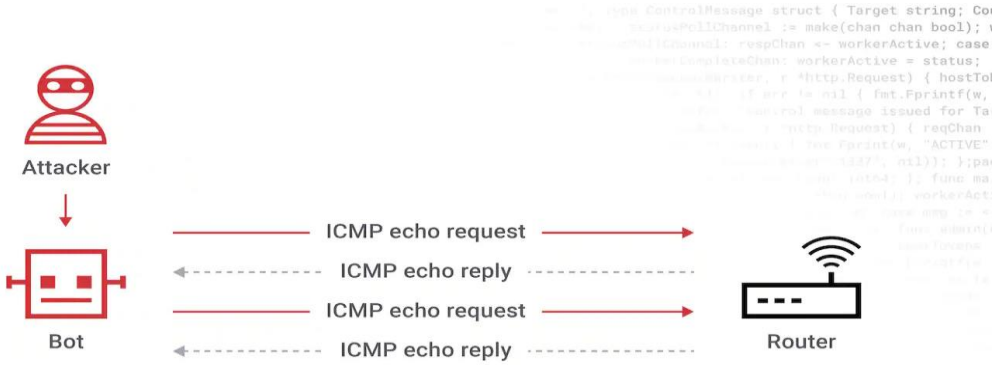
Les attaques par déni de service distribué (DDoS) sont des types d'attaques visant à rendre un service indisponible en submergeant la cible par un trafic provenant de plusieurs sources. En submergeant le système cible de demandes malveillantes, ces attaques exploitent la capacité limitée à traiter les requêtes légitimes.

A. Types d'attaques DDoS

1. Attaques de surcharge de bande passante :

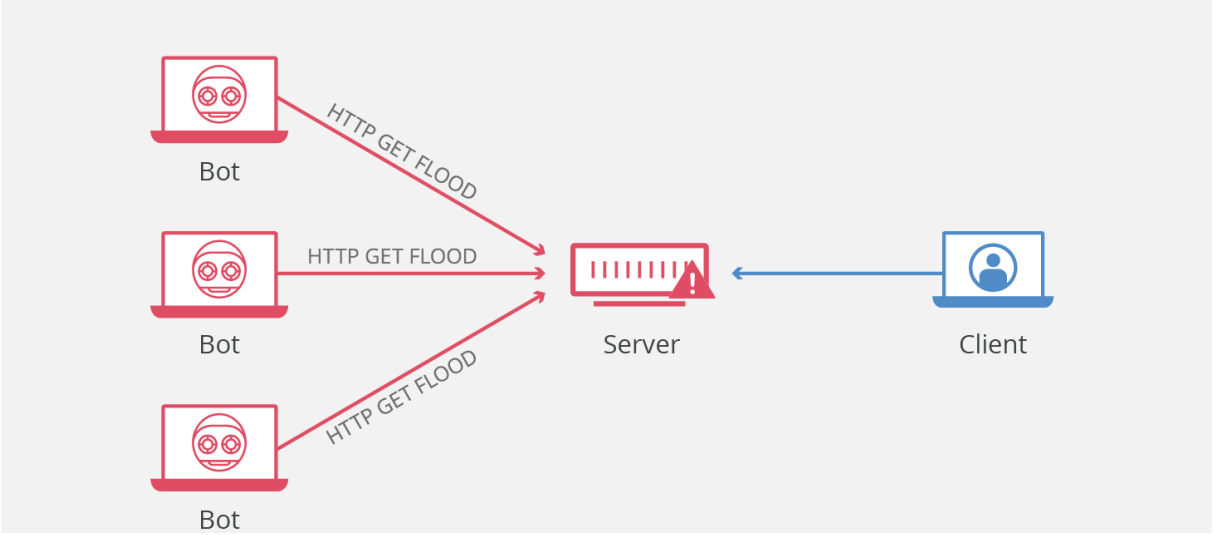
Une attaque DDoS de type surcharge de bande passante, connue sous le nom d'ICMP flood, vise à submerger un système cible en inondant sa bande passante avec de multiples paquets ICMP. Cela entraîne une diminution de la capacité du système à répondre aux requêtes légitimes, provoquant un déni de service pour les utilisateurs légitimes.

Les attaquants utilisent souvent un botnet pour envoyer un grand nombre de paquets ICMP simultanément depuis différentes adresses IP, rendant l'attaque plus difficile à détecter et à neutraliser.



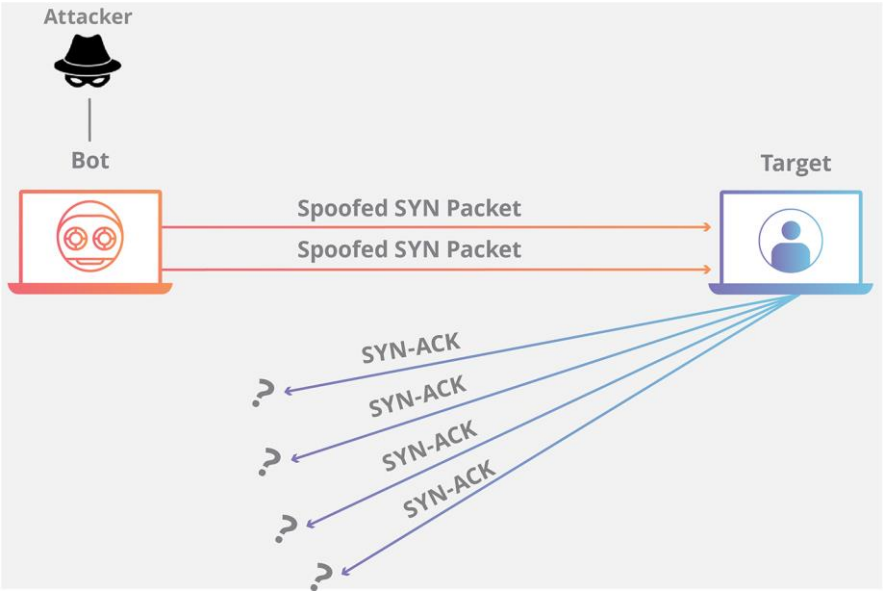
2. Attaques d'épuisement des ressources :

Une attaque DDoS de type épuisement de ressources, connue sous le nom de HTTP flood, consiste à submerger un serveur web cible en envoyant un grand nombre de requêtes HTTP simultanées. L'objectif est d'épuiser les ressources du serveur en le forçant à traiter ces demandes massives, ce qui peut entraîner un déni de service pour les utilisateurs réels.



3. Attaques de faiblesse des protocoles :

Une attaque DDoS de type faiblesse des protocoles, connue sous le nom de SYN flood, exploite une vulnérabilité du protocole TCP/IP en envoyant un grand nombre de demandes de connexion SYN sans les finaliser. Cela submerge le système cible en épuisant ses ressources avec des connexions SYN en attente, entraînant un déni de service en empêchant les utilisateurs légitimes d'établir de nouvelles connexions.



B. Objectifs et motivations des attaques DDoS :

Les attaques DDoS sont généralement motivées par les objectifs suivants :

- **Indisponibilité des services** : Les attaquants cherchent à empêcher les utilisateurs légitimes d'utiliser les services en ligne, ce qui entraîne une perte de revenus, une mauvaise réputation et une perturbation d'activités commerciales.

- **Extorsion et rançon** : Les attaquants peuvent utiliser les attaques DDoS comme moyen de contraindre les organisations ciblées les menaçant de continuer les attaques à moins qu'une rançon ne soit payée.

- **Distraire les équipes de sécurité** : Les attaques DDoS peuvent être utilisées comme une distraction empêcher l'équipe IT de détecter d'autres activités malveillantes, telles que des intrusions ou des vols de données.

III. Concepts théoriques :

A. Programmation multithreaded :

La programmation multithread permet l'exécution simultanée de tâches indépendantes appelées threads dans un programme. Les threads fonctionnent de manière autonome permettant une utilisation efficace et efficiente des ressources du système.

Cependant, des problèmes de synchronisation et de partage de ressources peuvent survenir. C'est pour cela que des mécanismes de synchronisation et d'exclusion mutuelle sont implémentés pour coordonner l'exécution des threads, contrôler l'accès aux ressources partagées et éviter les conflits et/ou incohérence.

Notre application peut analyser et traiter simultanément des données d'un «dataset» de record de trafic réseau grâce à la programmation multithreaded, améliorant ainsi la détection des attaques DDoS.

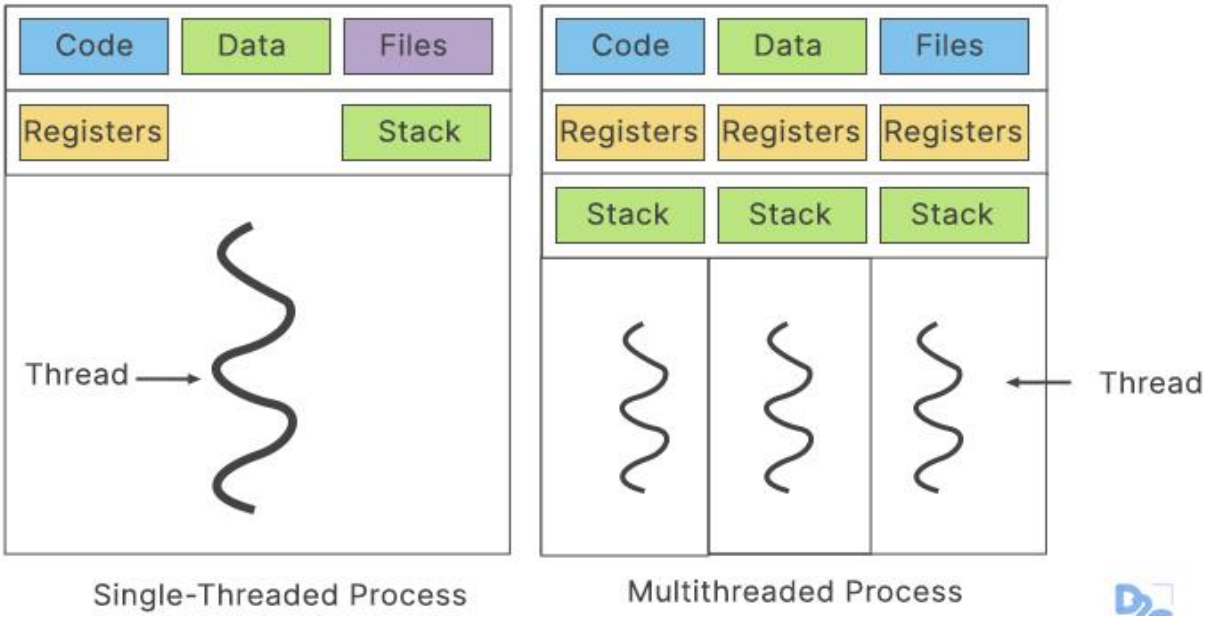


Figure: Multithreading

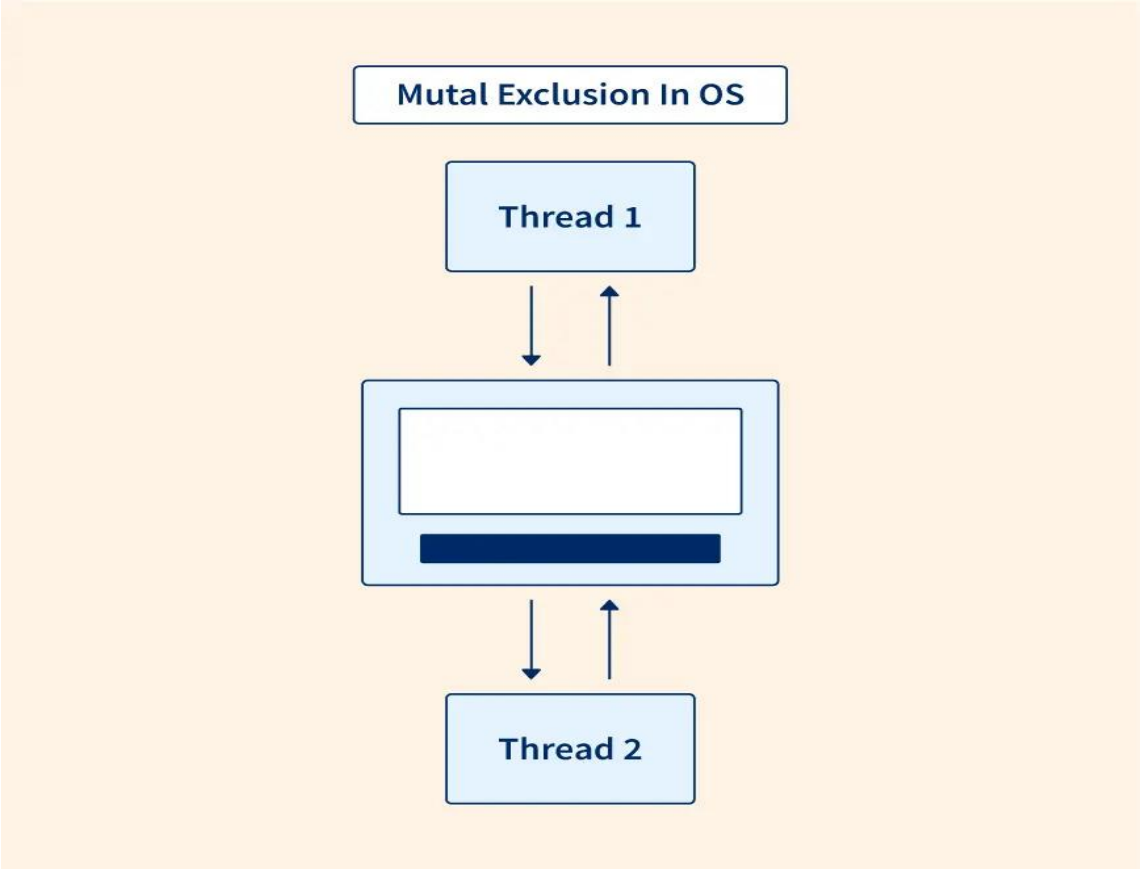


Figure : l'exclusion mutuelle

B. Problèmes de synchronisation des processus :

Plusieurs problèmes de synchronisation peuvent survenir lorsque des threads s'exécutent simultanément. La synchronisation demande la coordination des threads dans un ordre particulier ou d'attendre qu'une condition soit remplie avant de poursuivre l'exécution.

La résolution de ces problèmes de synchronisation est importante pour l'exécution correcte et cohérente des threads.

C. Techniques d'attente active et passive :

Diverses techniques telles que l'attente active et passives sont utilisées pour résoudre les problèmes de synchronisation des processus.

L'attente active recherche constamment une condition jusqu'à ce qu'elle soit remplie. Cependant, cela peut entraîner une utilisation excessive des ressources système.

En revanche, l'attente passive, elle, consiste à suspendre l'exécution du thread jusqu'à ce que certaines conditions soient remplies, économisant ainsi des ressources.

Les applications multithreads utilisent ces techniques de mise en file d'attente active et passive pour assurer une exécution correcte et cohérente des threads afin d'améliorant la performance globale de l'application.

IV. Conception et architecture de l'application :

A. Architecture de l'application :

Il est important de noter que le travail qui suit a été effectuée sous linux

L'architecture de l'application est basée sur le modèle de traitement asynchrone avec plusieurs threads. Chaque ligne d'entrée du flux est traitée de manière indépendante par un thread distinct pour optimiser les performances et permettre le traitement continue des données.

L'application utilise des mutex ainsi que des variables de condition afin de synchroniser l'accès aux structures de données partagées par les threads.

```
    string line;
    while (getline(cin, line)) {
        thread t(processLine, line, ref(trafficMap), ref(detectedIPs),
```

```
                    ref(attackInfoMap), ref(mutex), ref(printedIPs));
        t.detach();
    }
```

Dans cette partie du code, chaque ligne d'entrée est traitée de manière indépendante par un thread distinct (t) en appelant la fonction processLine. Les structures de données partagées (trafficMap, detectedIPs, attackInfoMap, printedIPs) sont passées en référence aux threads. Le détachement du thread (t.detach()) permet au thread de s'exécuter indépendamment.

B. Choix de conception et justifications :

Les données de trafic sont stockées dans une structure de données unordered_map pour permettre un accès rapide et efficace aux données associées à chaque adresse IP.

Les adresses IP détectées comme sources d'attaques sont stockées dans une unordered_set pour faciliter la recherche et l'insertion rapides.

Les informations sur les attaques détectées sont stockées dans une unordered_map, associant chaque adresse IP attaquante à des informations spécifiques sur l'attaque.

Un mutex est utilisé pour garantir l'accès exclusif aux structures de données partagées, évitant les problèmes de concurrence lors de la mise à jour des données.

Une variable de condition est utilisée pour notifier les threads en attente lorsque de nouvelles données sont disponibles pour être traitées.

Les adresses IP déjà imprimées sont stockées dans un unordered_set pour éviter l'impression répétée des informations sur les attaques.

Partie correspondante :

```
struct DataTraffic {
    int packetCount;
    chrono::time_point<chrono::system_clock> timestamp;
    string destIP;
    int destPort;
};

struct AttackInfos {
    int packetCount;
    string destIP;
    int destPort;
};
unordered_map<string, DataTraffic> trafficMap;
unordered_set<string> detectedIPs;
unordered_map<string, AttackInfos> AttackInfosMap;
unordered_set<string> printedIPs;
std::mutex mutex;
```

Dans cette partie du code, les structures de données sont déclarées et initialisées pour stocker les informations de trafic, les adresses IP détectées, les informations sur les attaques et les adresses IP déjà imprimées.

Les mutex et les variables de condition sont également déclarés pour assurer la synchronisation lors de l'accès aux structures de données partagées.

V. Implémentation :

A. Algorithmes et structures de données :

La fonction split est utilisée pour diviser une ligne de texte en un vecteur de champs en utilisant un délimiteur spécifié.

La fonction isIntegerValid est utilisée pour vérifier si une chaîne de caractères représente un entier valide.

La fonction determineTypeAttack détermine le type d'attaque en fonction de l'adresse IP de destination, du port de destination et des indicateurs de drapeau.

La fonction Lineprocessing traite chaque ligne de données d'entrée en extrayant les informations pertinentes, met à jour les structures de données appropriées et détecte les attaques en dépassant le seuil défini.

Code correspondent:

```
vector<string> split(const string& s, char delimiter) {
    vector<string> tokens;
    stringstream ss(s);
    string token;
    while (getline(ss, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}
bool isValidInteger(const std::string& str) {
    return !str.empty() && std::all_of(str.begin(), str.end(), ::isdigit);
}
string determineTypeAttack(const string& destIP, int destPort, int flags) {

    // Logique de détermination du type d'attaque
    // ...
}
void Lineprocessing(const string& line, unordered_map<string, DataTraffic>&
trafficMap,
                    unordered_set<string>& detectedIPs,
```

```
                unordered_map<string, AttackInfos>& AttackInfosMap,
                std::mutex& mutex, unordered_set<string>& printedIPs) {
    // Traitement de chaque ligne
    // ...
}
```

Les fonctions split, isIntegerValid, determineTypeAttack et Lineprocessing sont définies pour effectuer les différentes opérations nécessaires sur les lignes de données d'entrée.

Ces fonctions sont utilisées pour diviser la ligne en champs, vérifier si une chaîne représente un entier valide, déterminer le type d'attaque en fonction des informations fournies, et traiter chaque ligne en mettant à jour les structures de données appropriées.

B. Défis de la programmation multithreaded :

L'utilisation de mutex et de variables de condition assure la synchronisation et prévient les problèmes de concurrence lors de l'accès aux structures de données partagées.

Les threads sont utilisés pour traiter chaque ligne de manière indépendante et asynchrone, optimisant ainsi les performances et permettant un traitement concurrentiel des données.

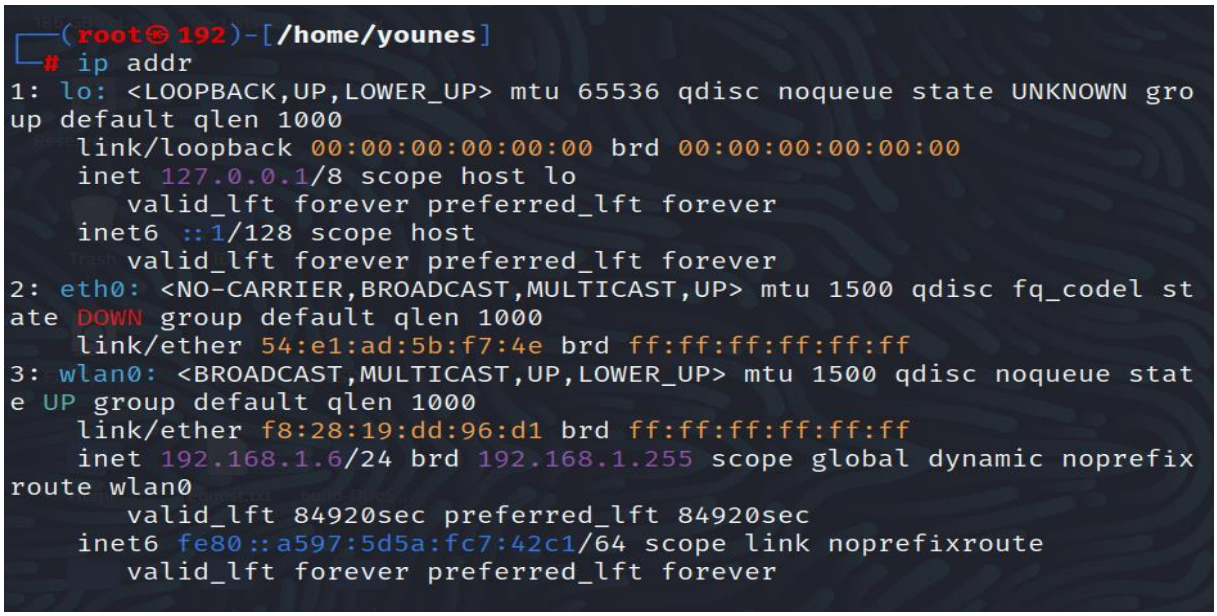
L'élimination efficace et sûre des données obsolètes, selon la fenêtre temporelle spécifiée, est un défi supplémentaire.

L'application doit être testée pour détecter et résoudre les problèmes de concurrence potentiels, tels que les conditions de concurrence, les courses critiques et les verrouillages incorrects.

VI. Évaluation et résultats :

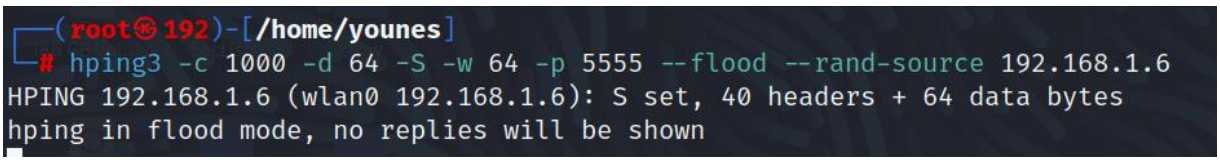
A. Simulation des attaques DDoS :

- Adresse IP de notre machine : 192.168.1.6



Notre application travaille sur des datasets d'attaques réels captés. Cependant il est primordial de savoir comment simuler les attaques déni de service distribué(DDoS). Plusieurs outils pour simuler diffèrent type d'attaques DDoS exist, le plus connu, « hping3 » sous linux.

✓ SYN Flood :



- **-c 1000** : envoie 1 000 paquets.
- **-d 64** : définit la taille des données de chaque paquet sur 64 octets.
- **-S** : définit l'indicateur TCP SYN.
- **-w 64** : définit la taille de la fenêtre sur 64.
- **-p 5555** : définissez le port de destination sur 5555.
- **--flood** : envoie des paquets aussi vite que possible sans délai.
- **--rand-source** : utilise des adresses IP source aléatoires pour chaque paquet.
- **192.168.1.6** : Définissez l'adresse IP cible sur 192.168.1.6.

Voici les paquets captés par "tshark" :

```
12564 0.421966479  192.168.1.6 → 81.57.175.226 TCP 54 5555 → 22793 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12565 0.421973779  192.168.1.6 → 242.17.163.240 TCP 54 5555 → 22794 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12566 0.421981010  192.168.1.6 → 197.95.150.185 TCP 54 5555 → 22795 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12567 0.421989195  192.168.1.6 → 73.255.122.86 TCP 54 5555 → 22796 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12568 0.421996683  192.168.1.6 → 125.149.28.208 TCP 54 5555 → 22797 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12569 0.422004104  192.168.1.6 → 27.222.58.60 TCP 54 5555 → 22798 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12570 0.422011711  192.168.1.6 → 51.228.211.91 TCP 54 5555 → 22799 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12571 0.422019118  192.168.1.6 → 80.186.150.119 TCP 54 5555 → 22800 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12572 0.422026862  192.168.1.6 → 174.174.52.226 TCP 54 5555 → 22801 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12573 0.422034373  192.168.1.6 → 150.179.193.56 TCP 54 5555 → 22802 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12574 0.422042118  192.168.1.6 → 234.121.148.208 TCP 54 5555 → 22803 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12575 0.422049612  192.168.1.6 → 163.54.47.226 TCP 54 5555 → 22804 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12576 0.422056866  192.168.1.6 → 151.117.36.134 TCP 54 5555 → 22805 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
12577 0.422064159  192.168.1.6 → 199.102.115.86 TCP 54 5555 → 22806 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

- Voici une ventilation du modèle des paquets et quelques raisons de sécurité potentielles derrière eux :
    - Provenance : 192.168.1.6
    - Destinataire : 81.57.175.226
    - Protocole : TCP
    - Indicateurs : [RST, ACK]
    - Numéro de séquence : 1
    - Numéro d'accusé de réception : 1
    - Taille de la fenêtre : 0
    - Longueur : 0
- **Explication/Raisons de sécurité** : Ce paquet est un paquet TCP avec les drapeaux [RST, ACK] définis. Le paquet indique une réinitialisation et un accusé de réception. Il peut s'agir d'une réponse à une tentative de connexion précédente ou d'une erreur réseau. L'adresse IP de destination (81.57.175.226) a peut-être reçu une tentative de connexion inattendue ou non valide, et elle répond en réinitialisant la connexion. Cela pourrait être un mécanisme de sécurité pour empêcher l'accès non autorisé ou atténuer les attaques potentielles comme les attaques DDOS ou DoS.

✓ ICMP flood

```
┌──(root㉿192)-[/home/younes]
└─# hping3 --icmp --flood --rand-source 192.168.1.6
HPING 192.168.1.6 (wlan0 192.168.1.6): icmp mode set, 28 headers + 0 data byt
es
hping in flood mode, no replies will be shown
```

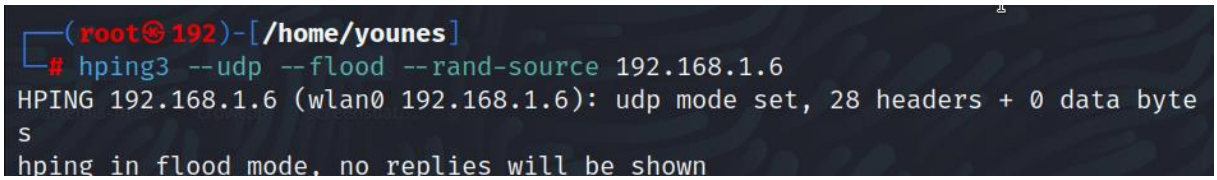
- **--icmp** : utilise les paquets ICMP (ping).
- **--flood** : envoie des paquets aussi vite que possible sans délai.
- **--rand-source** : utilise des adresses IP source aléatoires pour chaque paquet.
- **192.168.1.6** : Définissez l'adresse IP cible sur 192.168.1.6.

- Voici les paquets captés par “tshark” :

```
25452 0.849005108  192.168.1.6 → 59.26.205.37 ICMP 42 Echo (ping) reply    id=0×b23a, seq=10394/39464,
ttl=64
25453 0.849016959  192.168.1.6 → 241.206.44.13 ICMP 42 Echo (ping) reply    id=0×b23a, seq=10650/39465,
 ttl=64
25454 0.849029812  192.168.1.6 → 40.250.37.16 ICMP 42 Echo (ping) reply    id=0×b23a, seq=10906/39466,
ttl=64
25455 0.849043346  192.168.1.6 → 17.44.66.70  ICMP 42 Echo (ping) reply    id=0×b23a, seq=11162/39467,
ttl=64
25456 0.849056942  192.168.1.6 → 61.212.44.8  ICMP 42 Echo (ping) reply    id=0×b23a, seq=11418/39468,
ttl=64
```

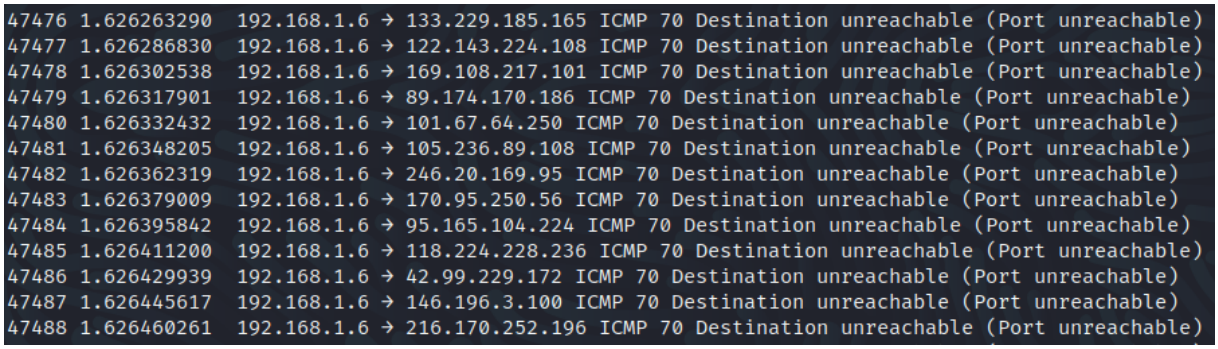
- Voici une ventilation du modèle des paquets et quelques raisons de sécurité potentielles derrière eux :
    - Provenance : 192.168.1.6
    - Destinataire : 59.26.205.37
    - Protocole : ICMP (réponse d'écho)
    - ID ICMP : 0xb23a
    - Séquence ICMP : 10394/39464
    - Durée de vie : 64
- **Explication/Raisons de sécurité :** Ce paquet est une réponse ICMP Echo (réponse ping). Il s'agit d'une réponse à une précédente requête ICMP Echo (ping). L'IP source (192.168.1.6) répond à l'IP de destination (59.26.205.37) avec le même ID ICMP et les mêmes numéros de séquence. ICMP est couramment utilisé pour le dépannage du réseau et les tests de connectivité. En termes de sécurité, ICMP peut être utilisé par des attaquants pour effectuer une reconnaissance ou une analyse du réseau, ainsi que pour d'éventuelles attaques DDoS ou DoS.

✓ UDP flood



- **--udp** : utilise des paquets UDP.
- **--flood** : envoie des paquets aussi vite que possible sans délai.
- **--rand-source** : utilise des adresses IP source aléatoires pour chaque paquet.
- **192.168.1.6** : Définissez l'adresse IP cible sur 192.168.1.6.

- Voici les paquets captés par “tshark” :



- Voici une ventilation du modèle des paquets et quelques raisons de sécurité potentielles derrière eux :
    - Provenance : 192.168.1.6
    - Destination : 133.229.185.165
    - Protocole : ICMP (Destination inaccessible)
    - Type ICMP : Destination inaccessible (Port inaccessible)

- **Explication/Raisons de sécurité** : Ce paquet est un message ICMP Destination inaccessible indiquant que le port de destination est inaccessible. L'IP source (192.168.1.6) essaie d'atteindre l'IP de destination (133.229.185.165) mais rencontre un port inaccessible. Cela peut se produire si un pare-feu ou un dispositif de filtrage de réseau bloque les connexions entrantes vers ce port spécifique sur la destination. Il peut s'agir d'une mesure de sécurité pour protéger les services exécutés sur la destination contre les accès non autorisés ou les attaques potentielles.

B. Tests réalisés :

Pour évaluer les performances et la fonctionnalité de notre application, nous avons effectué les tests suivants :

- **Test de charge** : Nous avons fournis à l'application des datasets qui simulé un trafic réseau intensif en fournissant un flux de données volumineux à l'application. Nous avons mesuré le temps de traitement global et vérifié si l'application était capable de gérer efficacement la charge.

  - Voici le modèle d'information correspondant à chaque ligne du dataset utilisé, chaque paquet avec les différents champs et propriétés des protocoles réseau:

```
frame.encap_type,frame.len,frame.protocols,ip.hdr_len,ip.len,ip.flags.rb
,ip.flags.df,p.flags.mf,ip.frag_offset,ip.ttl,ip.proto,ip.src,ip.dst,tcp
.srcport,tcp.dstport,tcp.len,tcp.ack,tcp.flags.res,tcp.flags.ns,tcp.flag
s.cwr,tcp.flags.ecn,tcp.flags.urg,tcp.flags.ack,tcp.flags.push,tcp.flags
.reset,tcp.flags.syn,tcp.flags.fin,tcp.window_size,tcp.time_delta
```

  - Exemple d'un paquet correspondant:

```
1,74,eth:ethertype:ip:tcp,20,60,0,1,0,0,64,6,154.49.2.126,119.162.193.187,52074,80,0,0,0,0,0,0,0,0,0,1,0,2
8400,0.000000000
```

- **Test de détection d'attaques connues** : Nous avons utilisé un ensemble de « dataset » d'attaques réseau simulées, en utilisant différentes adresses IP sources, ports de destination et indicateurs de drapeau. Nous avons vérifié si l'application détecte correctement ces attaques et génére des alertes appropriées.

C. Résultats et analyses :

Les résultats obtenus lors des tests ont montré que notre application était efficace pour détecter différentes attaques réseau.

- Makefile :

```
CXX = g++
CXXFLAGS = -pthread
install-dependencies:

    sudo apt-get update
    sudo apt-get install -y libpcap-dev
    sudo apt install build-essential libpthread-stubs0-dev


all: Detection_DDoS

Detection_DDoS: DDoSdetection.cpp

    $(CXX) -o $@ $< $(CXXFLAGS)
run: Detection_DDoS

    ./Detection_DDoS < DATASET.txt
clean:

    rm -f Detection_DDoS
.PHONY: all run clean
```

- Voici le résultat de la détection sur une dataset qui a des attaques DDoS :

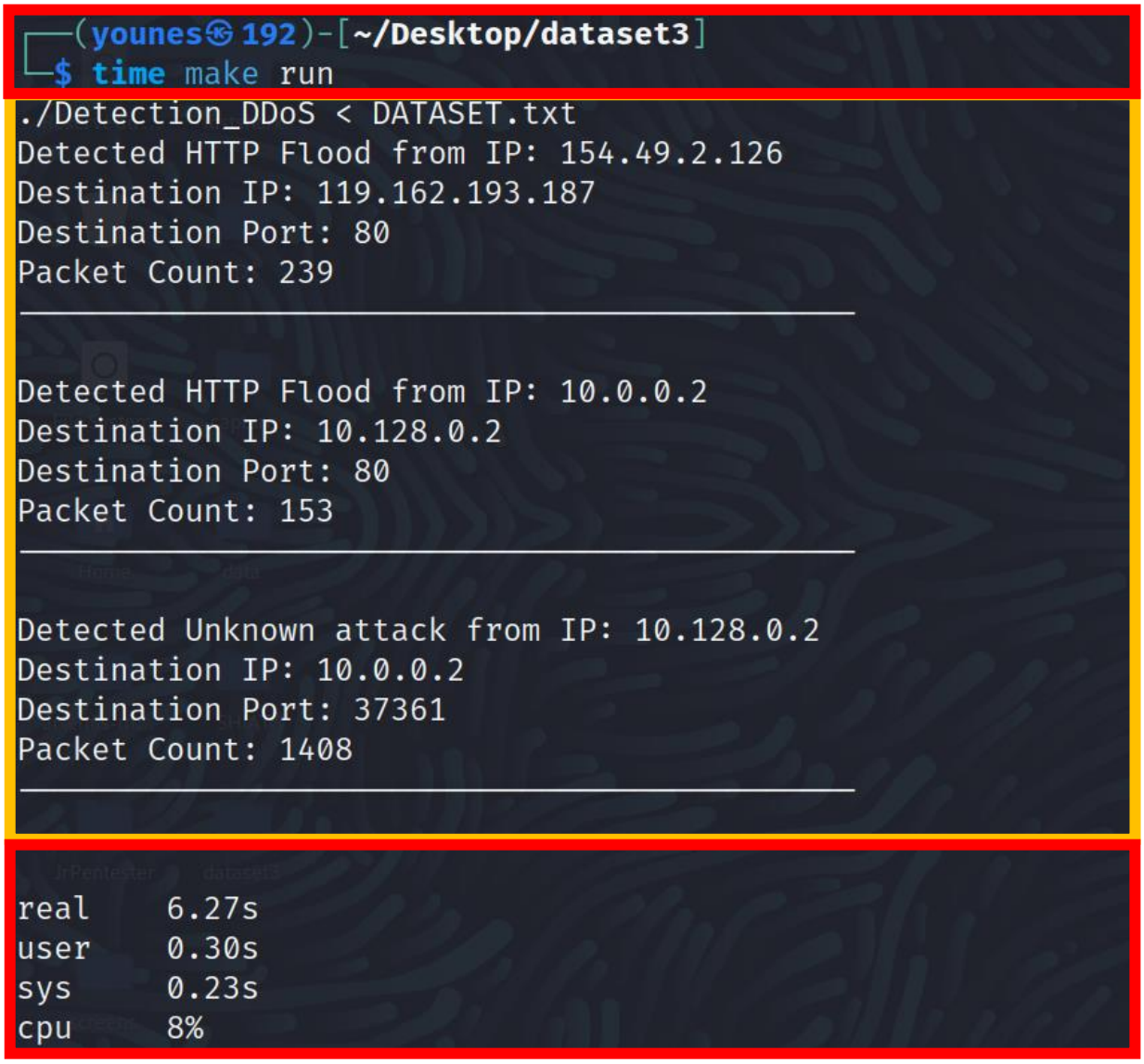
```
┌──(younes㉿192)-[~/Desktop/dataset]
└─$ make run
g++ -o Detection_DDoS DDoSdetection.cpp -pthread -lpcap
./Detection_DDoS < DATASET.txt
Detected HTTP Flood from IP: 154.49.2.126
Destination IP: 119.162.193.187
Destination Port: 80
Packet Count: 239
--------------------------------------------------

Detected HTTP Flood from IP: 10.0.0.2
Destination IP: 10.128.0.2
Destination Port: 80
Packet Count: 153
--------------------------------------------------

Detected Unknown attack from IP: 10.128.0.2
Destination IP: 10.0.0.2
Destination Port: 37361
Packet Count: 1408
--------------------------------------------------
```

Voici quelques observations clés :

- **Performances** : L'application a montré de bonnes performances lors du test de charge, en traitant le flux de données volumineux de manière efficace et en respectant les délais de traitement attendus.
- **Détection d'attaques** : L'application a réussi à détecter avec précision les attaques simulées, en analysant les adresses IP sources, les ports de destination et les indicateurs de drapeau. Les alertes générées étaient cohérentes avec les attaques détectées.



En analysant les résultats, nous avons identifié des opportunités d'amélioration pour optimiser d'avantage les performances et la précision de la détection. Cela pourrait inclure l'utilisation de techniques avancées de détection d'attaques telles que le « Machine Learning », ainsi que l'optimisation des algorithmes de traitement et l'amélioration de l'interface utilisateur pour faciliter l'analyse des alertes.

VII. Conclusion :

Ce rapport a présenté une application de détection d'attaques réseau multithread qui a une applicabilité particulière dans le domaine de la criminalité numérique. Nous avons discuté de la conception, de la mise en œuvre et de l'évaluation de l'application, soulignant son potentiel pour aider les enquêteurs et les analystes à analyser les incidents de sécurité.

Les résultats obtenus lors des tests ont montré que notre application est efficace pour détecter diverses attaques réseau essentielles à la collecte et à l'analyse de preuves numériques. En détectant rapidement les comportements malveillants, notre application peut fournir des informations précieuses pour la reconstitution des événements et la poursuite des responsables.

Cependant, des améliorations peuvent être apportées pour améliorer encore les performances et la précision de la reconnaissance. Le développement et l'amélioration continus de l'application peuvent renforcer la sécurité du réseau et faciliter les enquêtes dans le domaine de la criminalité numérique.

Enfin, nous soulignons l'importance de la programmation multithread dans les domaines de la sécurité des réseaux et de la criminalité numérique, ainsi que les opportunités de développement futur d'applications telles que l'intégration de techniques avancées de détection d'attaques et d'analyse comportementale. Ces potentielles améliorations permettront à l'application de jouer un rôle clé dans la sécurisation des réseaux et l'identification des contrevenants numériques.