Réseaux et Systèmes Avancés : Partie Réseaux

Compte rendu du projet MyAdBlock

Justine BASSELIN
Jordan CESARO

25 avril 2017

1 Présentation du projet

1.1 Introduction

Ce projet consiste en la création d'un proxy permettant le blocage de publicités sur les sites web. Le serveur mandataire doit relayer la requête uniquement si le nom de l'hôte ne correspond pas à un serveur tiers hébergeurs de publicités. Ce filtrage se fait à partir de la liste **EasyList**, proposé par le site web www.easylist.to. Ce proxy doit permettre de gérer plusieurs clients.

1.2 Répartition du travail

Nous nous sommes organisés afin de travailler de concert sur le projet. Nous avons décidé de nous retrouver afin de travailler ensemble.

Afin de gérer au mieux notre code source, nous avons choisi de travailler avec le gestionnaire de version Github, https://github.com. Cette méthode nous a permis de partager notre travail, et le versionnage nous à permis de conserver toutes les modifications effectuées.

Dans la même optique, nous avons choisi de rediger ce document avec ShareLatex, https://fr.sharelatex.com. Cela nous a permis de modifier simultanément le contenu de ce document.

1.3 Références

Liste non exhaustive des sites parcourus :

Capturer et analyser un traffic reseau avec Wireshark : https://blog.nicolargo.com/

EasyList option : https://adblockplus.org/filter-cheatsheetoptions

Getaddrinfo : https://www.inetdoc.net/dev/socket-c-4and6/socket-c-4and6.getaddrinfo.html

Documentation pour le paquet icmp : https://www.cymru.com/Documents/ip_icmp.h

Inspiration pour utilisation de recvfrom et retrouver ip :
http://stackoverflow.com/questions/2251198/received-udp-packet-length

2 Prise en main du projet

2.1 Analyse des échanges TCP et HTTP

Nous avons choisi d'étudier les échanges entre un client d'adresse IP : 192.168.1.27 et un serveur web : www.telecomnancy.univ-lorraine.fr d'adresse IP : 193.50.135.38. (Voir figure 2.1)

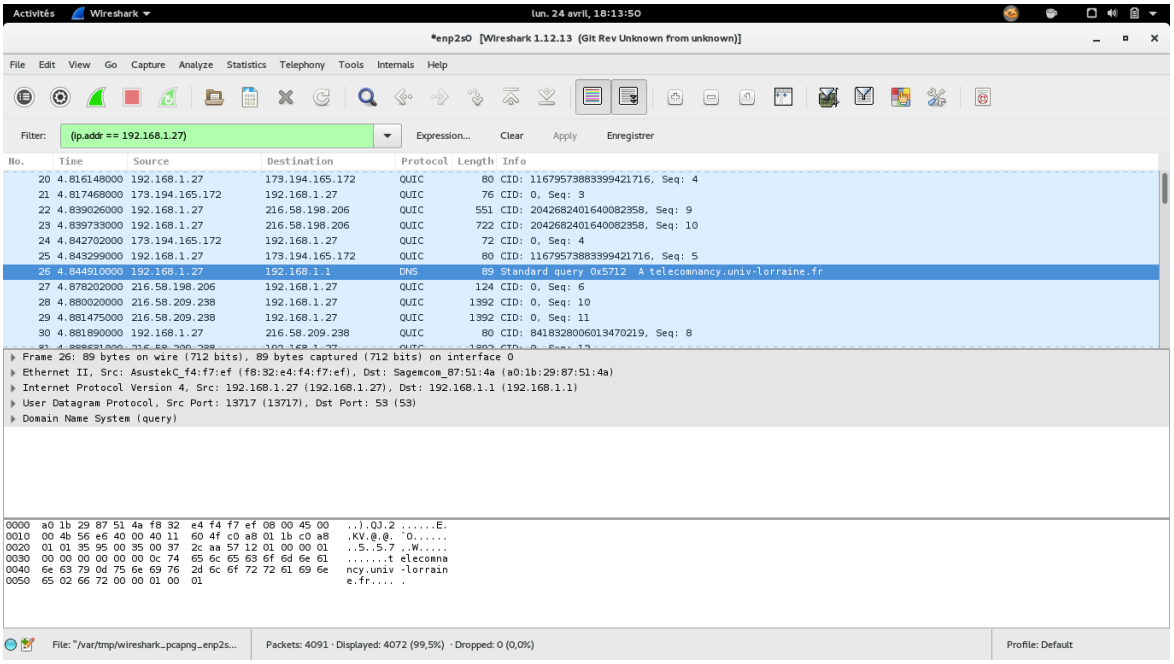

FIGURE 2.1 – Capture Wireshark, sans utilisation de filtre

Dans un premier temps, nous avons commencé en filtrant le trafic qui nous intéresse, c'est-à-dire, les échanges TCP et HTTP entre notre client et la page web. On a donc construit un filtre petit à petit, afin de ne garder que les requêtes nécessaires au chargement de la page, pour obtenir ce filtre : **(ip.src == 192.168.1.27 or ip.dst ==192.168.1.27) and tcp and (ip.src == 193.50.135.38 or ip.dst == 193.50.135.38)**.

En construisant le filtre nous avons remarqué qu'il n'était pas nécessaire de spécifier dans le filtre l'affichage des requêtes HTTP. En effet, HTTP est un protocole de la couche application et TCP un protocole de la couche transport, les paquets s'affichent lorsque l'on saisi le filtre "tcp", puisque que HTTP est une surcouche de TCP.

Les premier paquets que l'on peut alors analyser correspondent à l'établissement de la connexion. (Voir figure 2.2) Les trois paquets envoyés correspondants sont SYN, SYN-ACK et enfin ACK. Le paquet suivant est un paquet HTTP de type GET, il est envoyé dans le but de récupérer une première partie des informations du site web (1/4).

Ensuite, nous pouvons voir différents paquets TCP être échangés entre le client et le serveur, il porte la nomination "TCP segment of a reassembled PDU", ces paquets correspondent donc à l'envoi des informations demandées. Celle-ci sont acquittés par le paquet HTTP portant l'information "HTTP/1.1 200 OK". Ces opérations seront réitérées jusqu'au chargement complet de la page.

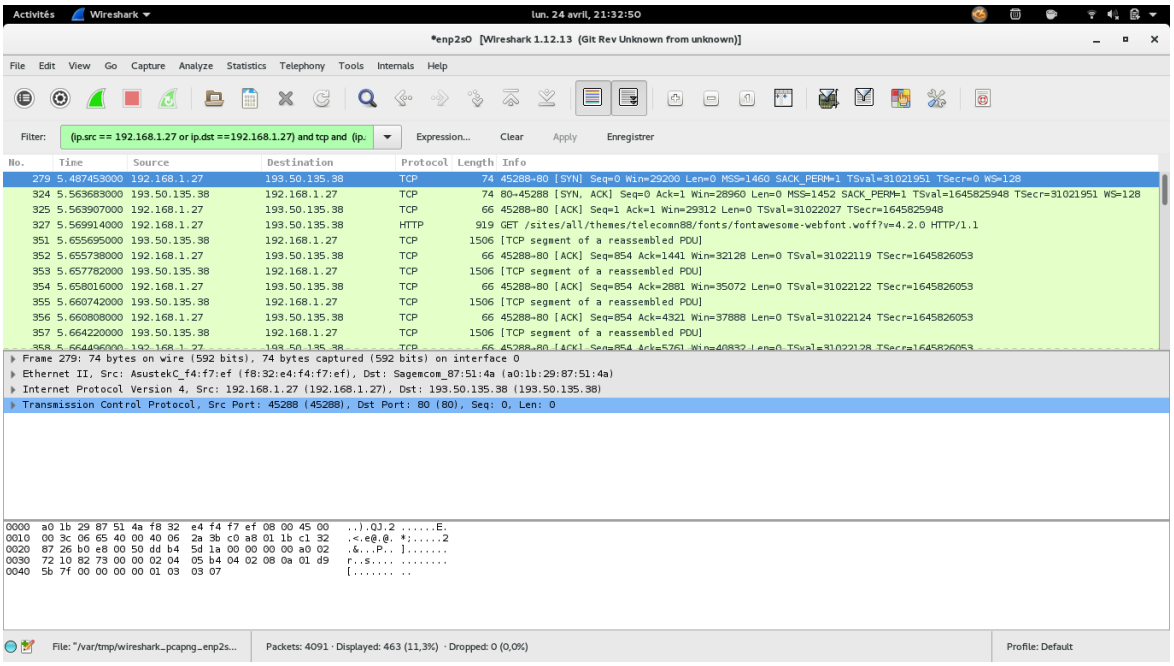

FIGURE 2.2 – Capture Wireshark, avec utilisation de filtre

2.2 Analyse des échanges TCP et HTTP avec proxy

Afin d'utiliser un proxy pour cette analyse, nous avons modifié les paramètres de notre navigateur web, en l'occurrence Mozilla Firefox. Nous avons choisi un proxy gratuit d'adresse IP : 160.202.40.203, disponible sur le site www.freeproxylists.net. Afin d'intégrer cet intermédiaire, nous avons modifié le filtre Wireshark, en remplaçant l'adresse IP du site web par celle du proxy : **(ip.src == 192.168.1.27 or ip.dst ==192.168.1.27) and tcp and (ip.src == 160.202.40.203 or ip.dst ==160.202.40.203)**.

A première vue, beaucoup de paquets sont retransmis dû à la qualité médiocre de la connexion. (Voir figure 2.3) En réalisant l'analyse de la capture, on remarque que le processus est identique au processus précédent. En effet, dans un premier temps des paquets (SYN, SYN-ACK et ACK) sont envoyés pour initialiser la connexion. Ensuite une requête HTML est envoyé afin d'obtenir une partie des informations du site web et un échange de paquets TCP se produit en réponse. Puis ces actions se répètent tout au long de la navigation sur le site web.

Pour conclure, l'échange entre le client et le proxy ressemble en tout point à l'échange analysé auparavant. Le client envoie des requêtes au proxy qui, lui les transmet au serveur. Le serveur va donc répondre à ces requêtes et le serveur mandataire se contentera de les transférer. Le proxy permettra en tant qu'intermédiaire, la journalisation des requêtes, la sécurisation du réseau local, ou encore le filtrages des publicités...

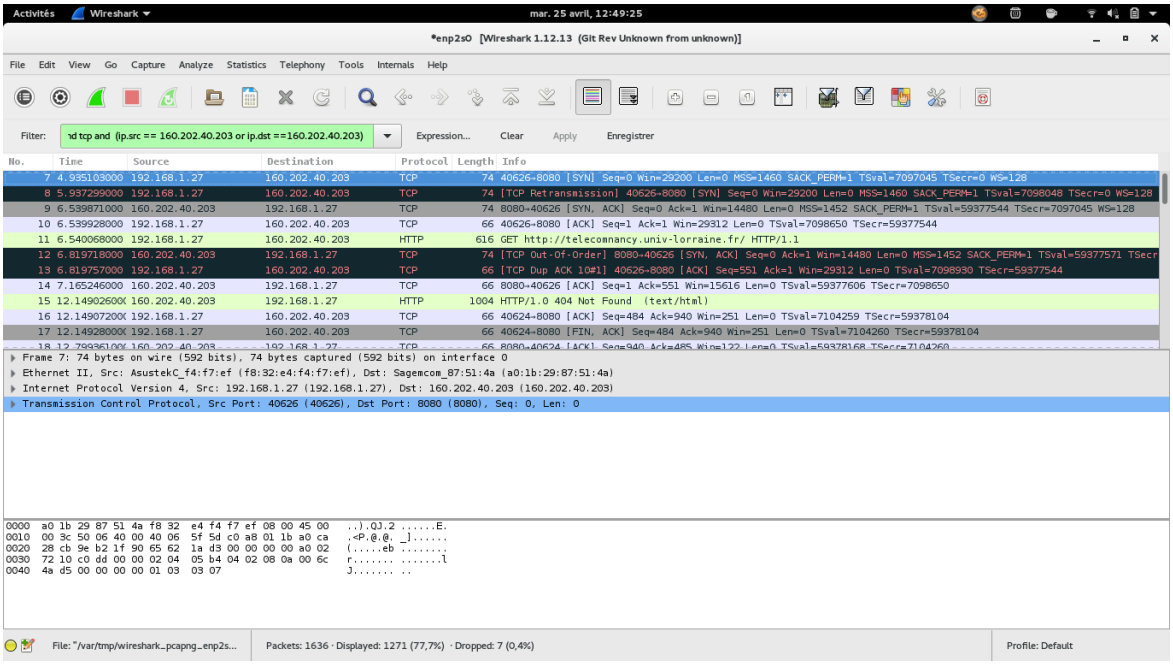

FIGURE 2.3 – Capture Wireshark, retransmissions multiples

2.3 Algorithme général de myAdBlock

Data: socketServeur, socketClient, socketWeb, requete, reponse, serveurAdresse, i
Result: Réception requête client, envoi requête serveur, transfert réponse client
while *Proxy actif* **do**
  Attendre la connexion d'un client;
  requete += recevoir(socketClient);
  while *requete != ' '* **do**
    requete++ ;
  end
  i = 0;
  while *requete[i] != ' '* **do**
    adresseServeur[i++] = requete[i++]
  end
  if *!estPublicite(adresse)* **then**
    socketWeb = créerSocket(adresseServeur);
    envoyer(requete, socketWeb);
    connexion(socketWeb);
    while *taille des données != 0* **do**
      reponse = lire(socketWeb);
      envoyer(reponse, socketClient) ;
    end
    fermer(socketServeur);
    fermer(socketClient) ;
  else
    affiche(adresse)
  end
end

Algorithm 1: Algorithme d'un proxy

3 Conception

3.1 Implémentation de l'algorithme général

La première version du projet est une implémentation naïve de l'algorithme, en effet il permet uniquement la connexion d'un seul client à la fois.

Un affichage en console est réalisé afin de pouvoir rendre compte du déroulement du processus. La détection de publicité, les chargements, envois, déconnexions et connexions sont donc affichés. Des informations plus générales sont affichés également tel que le type de requête (GET, etc.) ou encore le nom de domaine.

3.2 Multi-client

A partir de la première version de proxy que nous avions réalisé nous l'avons adapté afin que celui-ci puisse prendre en compte différents clients. Ce nouveau serveur utilise la primitive Select afin d'observer de multiples descripteurs simultanément.

Nous avons utilisé un tableau de descripteurs afin de gérer plusieurs sockets clients et plusieurs socket web. Select peut gérer jusqu'à 1024 descripteurs, notre tableau est donc limité dans sa taille par cette contrainte.

3.3 HTTPS

Nous avons essayé de comprendre le principe d'utilisation, cependant, nous n'avons pas réussi à le mettre en place. "HTTP/1.1 200 Connection established ° ° "

3.4 Choix de conception

Au cours de ce projet, nous avons du prendre différent choix les plus compliqués portaient sur le traitement de la EasyList. Nous avions réalisé un filtre permettant de traiter une grosse partie de ces spécificité les expressions classiques '*', '@@' et '||'. Les autres options étaient lus puis volontairement ignorés ('^', '$', '|', etc.). Cependant, le parcours du fichier engendrait une complexité temporelle et spatiale importante, nous avons donc choisi d'un commun accord de retirer ces symboles afin de ne garder que les expressions régulières classique du type *exemple*.