Rapport de Projet

Serveur Proxy AdBblock

Jean-Etienne SCHOUVER
Marion TOUSSAINT

Module RSA
2016 - 2017
Télécom Nancy

Table des matières

Introduction :

Un serveur proxy est un serveur qui sert d'intermédiaire entre le client HTTP et le serveur Web. L'avantage d'un serveur proxy est qu'il permet de réaliser un filtrage : en effet, comme toutes les requêtes et les réponses passent par le proxy, il est possible de filtrer les requêtes pour choisir celles autorisées à passer et celles qui ne le sont pas. Le but de ce projet était de réaliser un serveur proxy capable d'agir comme un bloqueur de publicités. Ce proxy empêcherait les requêtes, provenant d'un serveur hébergeur de publicités, d'être envoyé au client HTTP.

Dans ce rapport, nous analyserons, tout d'abord, les échanges TCP et HTTP entre un client et un serveur web. Puis, nous détaillerons les étapes de la conception de notre serveur proxy. Enfin, nous expliquerons la façon dont nous avons développer la partie AdBlock de notre proxy et les difficultés rencontrées ainsi que les solutions trouvées

I. Analyse des trames

Nous avons analysé les échanges TCP et HTTP entre un client et un serveur web à l'aide de l'analyseur de traces *Wireshark*. Nous avons donc observé les trames TCP et HTTP lorsque nous avons essayé de joindre le serveur *http://telecomnancy.net* .

I.1) Sans Proxy

Tout d'abord, nous avons réalisé cette analyse de trames sans serveur proxy. Les premières trames correspondent à la demande de connexion au serveur. Le client va demander la page d'accueil du site hébergé sur le serveur HTTP. Les trames suivantes correspondent aux demandes d'envoi des photos contenues sur le site et aux réponses du serveur qui sont accompagnées par les photos demandées. Nous avons clairement observé que l'échange se fait uniquement entre le client et le serveur HTTP.

I.2) Avec Proxy

Nous avons ensuite réalisé cette analyse avec le serveur proxy configuré. Nous avons, avant de refaire notre requête, vidé le cache du navigateur afin d'éviter que trames correspondants aux demandes d'envoi des photos n'apparaissent plus sur l'analyse. Nous observons que la requête est directement faite au proxy et non au serveur web. Dans sa requête, le client envoie l'URL complet du serveur web convoité. Le proxy cherche alors l'adresse IP du serveur web et il lui transmet ensuite la requête envoyé par le client. Le serveur web va répondre au proxy qui va ensuite transmettre le contenu au client. Les différentes requêtes pour l'envoi des photos contenues sur la page suivent le même schéma que celui expliqué précédemment. On remarque que c'est nettement plus compliqué et plus lourd qu'une requête directe.

II. Serveur Proxy

II.1) Conception

Notre serveur proxy est découpé en plusieurs parties : tout d'abord, on récupère la requête HTTP envoyé par le client. Puis, une fois la connexion effectuée entre le client et le proxy, celui-ci va analyser les données comprises dans l'entête et récupérer l'adresse IP du domaine demandé. On va ensuite les réécrire avant de les transmettre au navigateur web grâce à des sockets. Notre proxy va attendre la réponse du serveur web avant de renvoyer la réponse obtenue et le contenu demandé au client. Par ailleurs, nous créons un nouveau thread à chaque nouvelle connexion pour que notre serveur proxy soit capable de gérer plusieurs clients simultanément.

II.2) Difficultés rencontrées

Nous avons eu de nombreuses difficultés lors de la réalisation du serveur proxy. Au départ, on récupérait la requête et toutes les informations qu'elle contenait, et on la relayait directement au serveur web sans la modifier. Cependant, cela ne fonctionnait, nous n'arrivions pas à accéder aux pages web souhaitées. Nous nous sommes rendus compte que les requêtes HTTP n'étaient pas bonnes quand elles ressortaient du serveur proxy, elle était incompréhensible pour que le client puisse avoir accès aux pages web voulues. Nous avons donc décidé de recréer les requêtes provenant du client, en récupérant notamment les informations comprises dans l'entête de la requête. Cependant, notre serveur proxy ne marchait toujours pas et après avoir cherché d'où venait le problème, nous nous sommes rendus compte que le problème venait de l'adressage. En effet, nous utilisions la primitive *getaddrinfo* pour obtenir l'adresse correspondant à l'URL mais visiblement, nous l'utilisions mal et c'est pour cela que notre serveur proxy ne fonctionnait pas. Nous avons décidé d'utiliser la structure *hostent* à la place. Notre serveur fonctionne avec la réécriture des requêtes donc nous avons décidé de ne pas le modifier et de ne pas tester pour voir si en transmettant directement le buffer, le serveur fonctionnait toujours. Cette méthode a cependant un petit inconvénient, c'est que l'entête de la requête ne contient plus que les informations qu'on a mis dedans, par exemple toutes les informations concernant le navigateur n'apparaissent plus.

III. Serveur Proxy AdBlock

III.1) Conception

Notre première idée était d'utiliser un script Shell pour savoir si le nom du serveur renvoyé par le serveur web était un hébergeur de publicités ou non. Nous avons rencontré quelques difficultés avec cette méthode donc nous avons décidé de changer d'idée et de faire la recherche dans la liste directement en C. Finalement, notre bloqueur de publicités va tester, pour chaque nom de domaine, si celui-ci appartient à la liste des domaines hébergeurs de publicités, celle-ci était téléchargée lors de l'exécution du *Makefile*. Si le domaine héberge des publicités alors notre serveur proxy ne transmet pas cette requête au client et envoie une erreur. Nous

avons réalisé un script Shell permettant de mettre à jour l'*easylist* à chaque nouvelle exécution du *Makefile*.

Finalement, notre *Makefile* génère deux exécutables : *proxyHTTP* et *getlist*. Le premier gère le serveur proxy ainsi que le filtrage de la publicité tandis que le deuxième correspond à la mise à jour de l'*easylist*.

III.2) Difficultés rencontrées

Comme expliqué précédemment, nous voulions au départ utiliser du Shell. Nous ne l'avons finalement pas fait car nous n'arrivions pas à récupérer un retour suite à l'exécution de notre commande. En effet, nous utilisions la commande *grep* pour parcourir le fichier mais nous n'obtenions pas de retour donc nous ne pouvions pas exploiter le résultat. C'est pourquoi, nous avons modifié notre idée de départ et nous remplissions alors un fichier texte avec un 0 ou un 1, si la requête était une publicité ou non. Cependant, à cause de l'allocation processeur des thread, nous avions quelques problèmes : en effet, certaines requêtes qui était des publicités et qui étaient marquées comme telles lors de la première écriture du document, ne l'étaient plus dans les suivantes. Ce problème était sans doute dû au fait que le fichier texte était indépendant des threads mais que par conception de notre proxy, on était obligé de les utiliser pour remplir ce fichier. Par ailleurs, quand nous arrivions enfin à détecter les publicités des autres pages envoyées, nous avons eu un petit problème quand nous empêchions les publicités d'être envoyé au client. Les pages n'étaient pas envoyées mais elles continuaient de tourner en arrière plan.

Conclusion

Le projet était intéressant, il nous a permis de travailler l'analyse de trames et de découvrir l'intérêt d'un serveur proxy. Ce projet nous a aussi permis de créer un serveur proxy fonctionnel et de faire du filtrage pour empêcher les publicités de passer.