Rapport
Écriture d'un proxy
MyAdBlock

2016

AYI Brandon - TRS
LOEMBE Alex-Kevin -TRS

Sommaire

1.Analyse de Trame **3**

2.Configuration du navigateur web pour définir un proxy HTTP **4**

3.Observation via notre proxy **4**

4.Implémentation du proxy **5**

5. Spécification et Implémentation de l'algorithme **6**

6.Difficultées rencontrées **7**

Conclusion **7**

1.Analyse de Trame

La capture d'écran montre l'adresse IP WIFI utilisé pour l'analyse de trames.

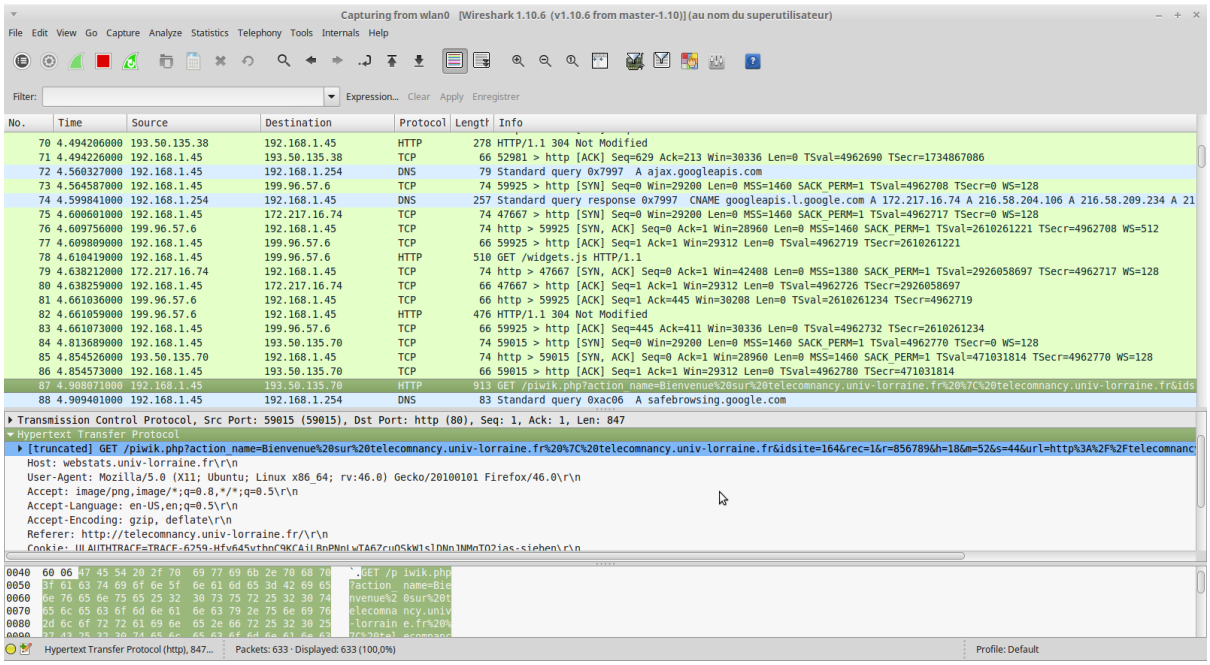

Figure 1

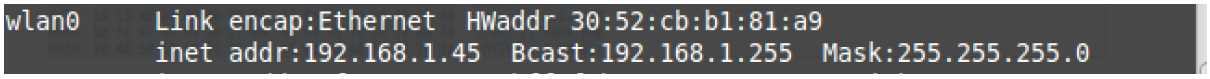

Figure 2

En observant les résultats de la capture, figure 1, nous obtenons différentes requêtes selon les protocoles suivant :

- TCP
- DNS
- HTTP

Celles qui nous intéressent sont les TCP et HTTP. En effet , nous observons une requête TCP vers le site telecomnancy.univ-lorraine.fr. Dans l'encadré listant toutes les trames, on peut voir l'établissement de la connexion TCP avec le serveur (3-way handshake) suivi de la requête HTTP. Ensuite, on retrouve différentes informations comme les ports source et destination, les adresses IP source et destination, la version IP et d'autres informations.
De plus, l'encapsulation des informations du protocole de transfert hypertexte nous montre que le serveur fait bien référence au site telecomnancy.univ-lorraine.fr, que la langue utilisée sur ce site est à première vue française. De plus, il est mentionné qu'une requête a été effectuée sous ce protocole par le biais de la méthode GET.

2.Configuration du navigateur web pour définir un proxy HTTP

La capture d'écran suivante montre comment nous avons configuré le navigateur. Nous utilisons l'adresse 127.0.0.1 (localhost). Nous avons choisi arbitrairement le port 2222.

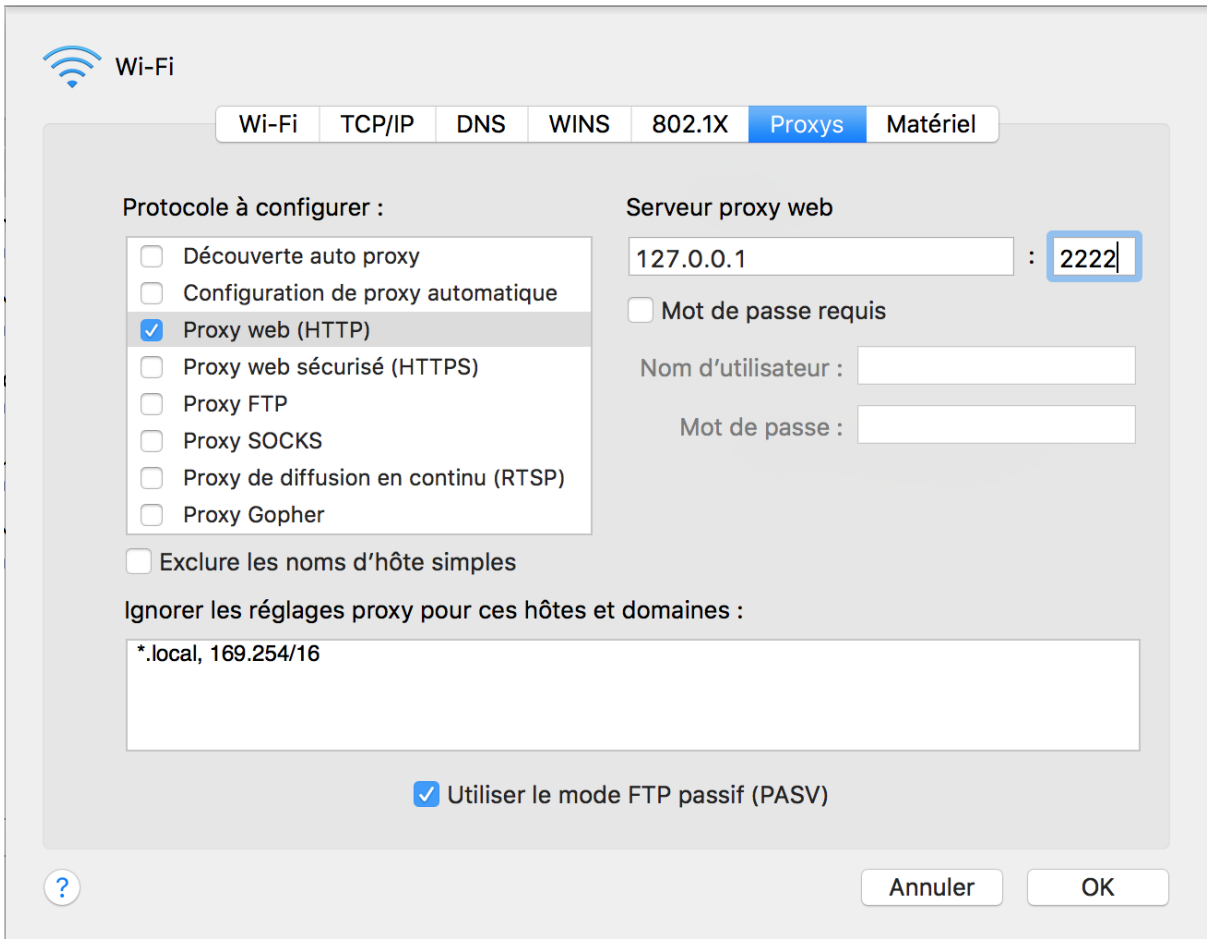

Figure 3

3.Observation via notre proxy

Pour activer notre proxy, on lance alors l'exécutable du fichier codé, en renseignant dans le deuxième argument (dans le terminal) le numéro de port qui est dans notre cas 2222, comme on peut le voir ci-dessous.

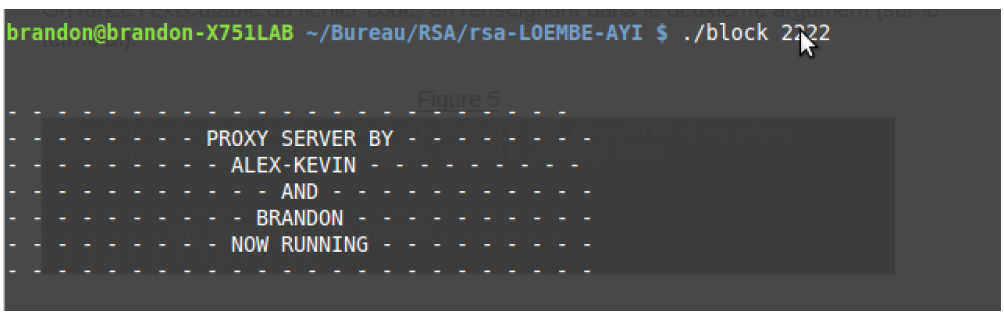

Figure 4

Nous effectuons maintenant une capture Wireshark avec notre proxy en loopback sur l'adresse localhost. Nous avons effectué une requête vers le site de Telecom Nancy. Nous voyons bien que les échanges de requêtes ont lieu sur l'adresse 127.0.0.1.

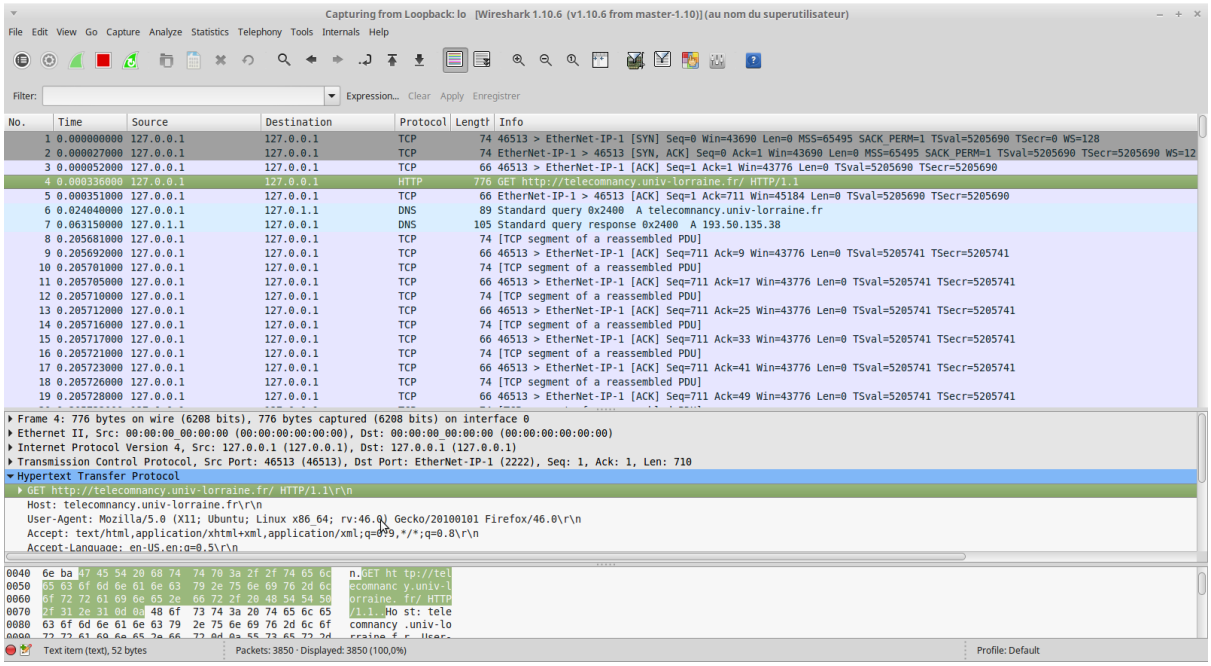

Figure 5

Sur la figure ci-dessus, on observe les informations sur la 1ère requête GET. Cela nous confirme que la requête du navigateur passe bien par notre proxy car le port de destination est celui que nous avons utilisé lors de l'exécution (2222).

4.Implémentation du proxy

Le but de ce projet est de développer un proxy qui sera situé entre le navigateur et le serveur web. Ce proxy devra bloquer un certain nombre de sites qui seront répertoriés dans une "blacklist". Ce proxy sera développé en C.

Pour la réalisation du proxy, nous nous sommes inspirés du schéma modélisant le dialogue client-serveur avec un proxy comme cela est présenté sur le schéma du sujet. Nous avons donc mis en pratique les notions de programmations sockets que nous avons vues en cours.

La première étape a été de configurer nos navigateurs web et de faire des recherches sur des adresses de proxy. Ainsi nous avons pu étudier le comportement d'un proxy et comment sont traités les requêtes avec Wireshark.
Ensuite nous avons commencer à développer notre propre proxy.

5. Spécification et Implémentation de l'algorithme

Tout d'abord nous avons défini les constantes suivante : HTTP_PORT = 80 et PACKAGE_LENGTH = 4096.
Pour implémenter ce proxy, nous avons ensuite défini plusieurs fonctions que nous allons expliciter.

fonction **strReplace()**: Nous avons trouvé cette fonction directement sur internet (http://stackoverflow.com/questions/779875/what-is-the-function-to-replace-string-in-c)
Elle nous permet de remplacer une chaine de caractère par autre chaine de caractère. Nous en avons besoin pour une raison essentielle : remplacer l'état de la connexion "keep-alive" en "close". En effet l'option "keep-alive" d'une connexion n'est pas supportée par les protocoles HTTP/1.0 et les requêtes peuvent être envoyées de manière incorrecte.

fonction **init()**: Cette fonction permet d'initialiser la socket de notre proxy. Cette initialisation se fait en 3 étapes : ouverture de la socket (socket()), liaison de la socket à l'adresse locale (bind()) et paramétrage du nombre maximum de connection (listen())

fonction **sendBrowser()** : Cette fonction permet de faire la transition entre le serveur et le client. On envoie d'abord la requête au serveur dans un buffer, puis celui-ci envoie l'url demandée dans un buffer qui va ensuite être envoyé au navigateur.

fonction **getBlackList()**: Il existe deux manières de bloquer une publicité : par son hostname ou par son path. La fonction getBlackList nous permet de comparer le *hostname* ou le *path* de la requête à toute les pubs répertoriés dans notre fichier blacklist. Nous savons si nous devons regarder le hostname ou le path grâce à la syntaxe dans la Blacklist (Par exemple l'utilisation de "||" au début de la publicité nous indique qu'il faut regarder le nom de domaine).

C'est avec toutes ces fonctions que nous implémentons l'algorithme suivant :
Tout d'abord nous vérifions que le proxy est lancé avec le bon nombre d'argument (un seul qui est le port).
Ensuite nous initialisons la socket du proxy grâce à la fonction *init* puis nous lançons la boucle infinie dans laquelle se passe le plus important du proxy.
Tout d'abord nous ouvrons la connexion avec la fonction accept (bibliothèque sys/socket), on utilise un fork pour avoir l'option de multi-clients, on crée deux socket : une pour le serveur et une pour le client car ceux sont les deux entités avec lesquelles nous allons dialoguer.
Nous recevons le *buffer* du client (via la fonction rcv de la bibliothèque sys/socket) que nous allons *parser* pour récupérer les informations qui nous intéressent (l'url et le *path*). Nous vérifions que la requête n'est pas sur la blacklist. Nous regardons ensuite si c'est une requête ipv4 ou ipv6 (la fonction getaddrinfo nous donne cette information) et nous traitons la requête en conséquence.
Enfin nous créons une socket, nous la préparons pour le buffer et nous l'envoyons grâce à la fonction sendBrowser().

6.Difficultés rencontrées

Une des premières difficultés que nous avons rencontrées fut la conception du proxy. En effet, matérialiser tous les échanges entre le client et le serveur n'a pas été évident et nous a pris beaucoup de temps.

Concernant la partie technique pure, la difficulté majeure était induite du fait que nous avons l'habitude de coder en JAVA, langage où beaucoup de fonctions sont déjà codées et/ou connues. En C, il y a moins de fonctions déjà implémentées ou alors nous ne les connaissons pas. Par exemple nous pouvons citer la fonction *strReplace* que nous avons trouvé sur internet car nous avions du mal à la recoder nous même. Nous pouvons également citer la fonction *strstr* que nous avons d'abord essayé de coder, puis que nous avons trouvé sur internet.

Une autre des difficultés auxquelles nous avons dû faire face était lors de la vérification de l'url pour voir si celui-ci était dans la liste des sites bannis. En effet, il y a des règles de filtrage par adBlock qui impliquent des caractères spéciaux dans certaines lignes du fichier "blacklist" (par exemple "###backgroundadvert" ou "||rubanners.com^$third-party"). Cela nous a conduit à gérer de manières différentes les sites répertoriés car certains devaient être comparés au nom de domaine et d'autres au *path*.

Malheureusement nous n'avons pas réussi à récupérer les requêtes HTTPS pour les traiter.

Conclusion

Ce projet qui est dans la continuité même des cours de RSA partie réseaux, nous a permis de nous exercer sur la prise en main de la bibliothèque sys/socket en C. De plus, ce projet a fait lumière sur nombre de petites subtilités que nous n'avions pas eu l'occasion de comprendre pendant les TP et le cours. Enfin, ce fut très enrichissant de traiter avec un navigateur et des serveurs, choses que nous utilisons au quotidien.