

Projet du module de Réseaux et Systèmes Avancés: *myAdBlock*, un *proxy* bloqueur de publicités

GARCIA Guillaume et DUGUE Clément

April 25, 2017



1 Questions sur le proxy, analyse à l'aide de Wireshark

1.1 Filtre

L'outil wireshark nous permet de faire des captures des traces de connexions. Pour pouvoir les analyser correctement, nous avons d'abord filtré les informations pour ne garder que les échanges entre notre connexion Wifi et le serveur de l'université de Lorraine. Ainsi en tapant "ifconfig" dans un terminal nous avons pu avoir accès à notre adresse ip :

```
surwlp3s0 : inetadr : 192.168.1.14
```

Puis en effectuant un ping de l'adresse de telecom nancy ("ping www.telecomnancy.univ-lorraine.fr"), nous avons pu accéder à l'adresse ip du serveur:

```
PING front.pweb.dc.univ - lorraine.fr (193.50.135.38) 56(84) bytes of data
```

Sur *wireshark* nous avons donc appliqué le filtre :

```
(ip.src == 192.168.1.14 and ip.dst == 193.50.135.38)
```

```
or (ip.dst == 192.168.1.14 and ip.src == 193.50.135.38)
```

1.2 Sans proxy

On voit alors l'établissement de la connexion (*SYN*, *SYN-ACK*, *ACK*) qui initialise la connexion TCP (image 1). Il y a plusieurs renvois de paquets car notre connexion est un peu lente. Ensuite le premier paquet HTTP est envoyé par notre client: requête (*GET*) Puis de multiples échanges TCP se font (image 2):

- le serveur envoie des "TCP segment of a reassembled PDU", c'est à dire des paquets contenant des données d'un protocole étant une surcouche à TCP (HTTP ici).
- notre client renvoie des acquittements ACK pour confirmer la réception des paquets.

Figure 1: Connexion

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|---------------|---------------|----------|--------|--|
| 443 | 8.6819016... | 192.168.1.14 | 193.50.135.38 | TCP | 74 | 54286 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11... |
| 450 | 8.7195737... | 192.168.1.14 | 193.50.135.38 | TCP | 74 | 54288 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11... |
| 452 | 8.7250735... | 193.50.135.38 | 192.168.1.14 | TCP | 74 | 80 → 54286 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1452 SACK_PERM... |
| 453 | 8.7251085... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54286 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1111994 TSecr=498... |
| 454 | 8.7252545... | 192.168.1.14 | 193.50.135.38 | HTTP | 780 | GET / HTTP/1.1 |
| 465 | 8.7688616... | 193.50.135.38 | 192.168.1.14 | TCP | 74 | 80 → 54288 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1452 SACK_PERM... |
| 466 | 8.7688826... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54288 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1112005 TSecr=498... |
| 473 | 8.8198891... | 193.50.135.38 | 192.168.1.14 | TCP | 66 | 80 → 54286 [ACK] Seq=1 Ack=715 Win=30464 Len=0 TSval=498234242 TSecr... |
| 476 | 8.8298611... | 193.50.135.38 | 192.168.1.14 | HTTP | 278 | HTTP/1.1 304 Not Modified |
| 477 | 8.8298820... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54286 → 80 [ACK] Seq=715 Ack=213 Win=30336 Len=0 TSval=1112021 TSecr... |
| 492 | 8.9674699... | 192.168.1.14 | 193.50.135.38 | HTTP | 841 | GET /sites/www.telecomnancy.univ-lorraine.fr/files/styles/image_slid... |
| 493 | 8.9675924... | 192.168.1.14 | 193.50.135.38 | HTTP | 849 | GET /sites/www.telecomnancy.univ-lorraine.fr/files/styles/image_slid... |
| 494 | 8.9678224... | 192.168.1.14 | 193.50.135.38 | TCP | 74 | 54292 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11... |
| 495 | 8.9679262... | 192.168.1.14 | 193.50.135.38 | TCP | 74 | 54294 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11... |
| 496 | 8.9681690... | 192.168.1.14 | 193.50.135.38 | TCP | 74 | 54296 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11... |
| 509 | 9.0365207... | 193.50.135.38 | 192.168.1.14 | HTTP | 1506 | HTTP/1.1 206 Partial Content (image/png) |
| 510 | 9.0365504... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54288 → 80 [ACK] Seq=776 Ack=1441 Win=32128 Len=0 TSval=1112072 TSecr... |
| 511 | 9.0492837... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | 80 → 54288 [ACK] Seq=1441 Ack=776 Win=30592 Len=1440 TSval=498234447... |

▶ Frame 510: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
 ▼ Ethernet II, Src: LiteonTe_7a:9c:b2 (24:fd:52:7a:9c:b2), Dst: ac:84:c9:cf:52:b0 (ac:84:c9:cf:52:b0)
 Destination: ac:84:c9:cf:52:b0 (ac:84:c9:cf:52:b0)
 Address: ac:84:c9:cf:52:b0 (ac:84:c9:cf:52:b0)
 = LG bit: Globally unique address (factory default)
 = IG bit: Individual address (unicast)
 Source: LiteonTe_7a:9c:b2 (24:fd:52:7a:9c:b2)
 Address: LiteonTe_7a:9c:b2 (24:fd:52:7a:9c:b2)
 = LG bit: Globally unique address (factory default)
 = IG bit: Individual address (unicast)

Figure 2: Echanges

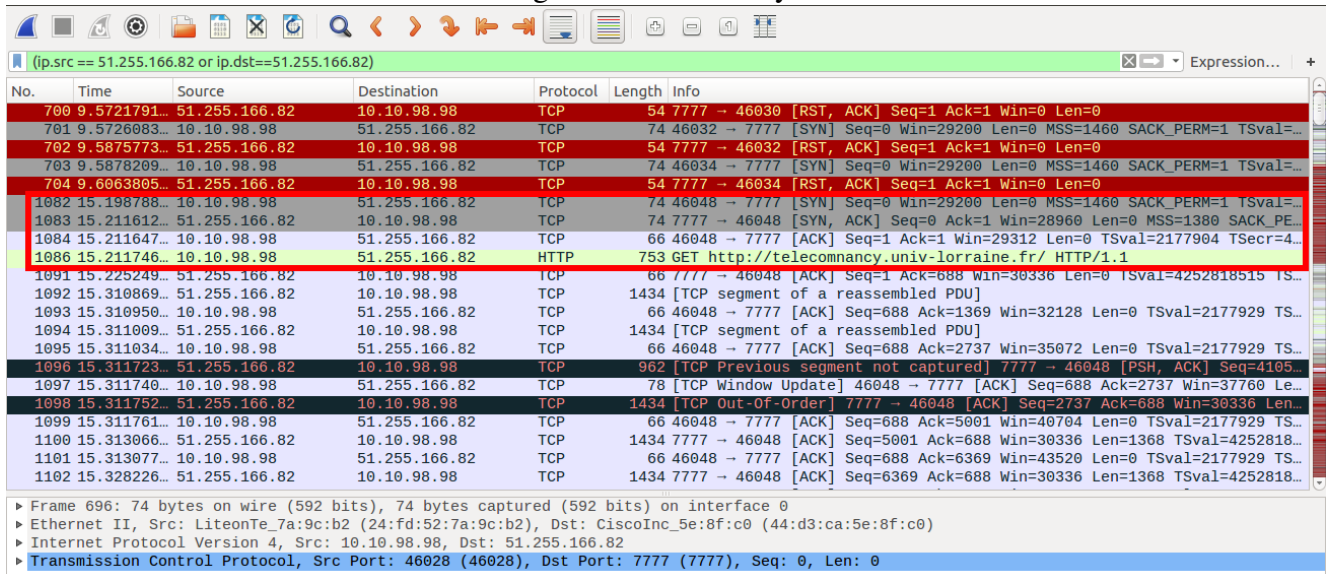
| No. | Time | Source | Destination | Protocol | Length | Info |
|------|--------------|---------------|---------------|----------|--------|---|
| 1364 | 13.462827... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54294 → 80 [ACK] Seq=776 Ack=90721 Win=184832 Len=0 TSval=1113179 TS... |
| 1365 | 13.474551... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1366 | 13.474574... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54294 → 80 [ACK] Seq=776 Ack=92161 Win=184832 Len=0 TSval=1113182 TS... |
| 1367 | 13.487477... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1368 | 13.487495... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54292 → 80 [ACK] Seq=780 Ack=59041 Win=147968 Len=0 TSval=1113185 TS... |
| 1369 | 13.499570... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1370 | 13.499601... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=77761 Win=184832 Len=0 TSval=1113188 TS... |
| 1371 | 13.515698... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1372 | 13.515733... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=79201 Win=184832 Len=0 TSval=1113192 TS... |
| 1373 | 13.524794... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1374 | 13.524843... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=80641 Win=184832 Len=0 TSval=1113194 TS... |
| 1375 | 13.537511... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1376 | 13.537557... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=82081 Win=184832 Len=0 TSval=1113197 TS... |
| 1377 | 13.549656... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1378 | 13.549686... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=83521 Win=184832 Len=0 TSval=1113200 TS... |
| 1379 | 13.562503... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1380 | 13.562557... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54296 → 80 [ACK] Seq=771 Ack=84961 Win=184832 Len=0 TSval=1113204 TS... |
| 1381 | 13.574236... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1382 | 13.574264... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54288 → 80 [ACK] Seq=776 Ack=132481 Win=184832 Len=0 TSval=1113207 T... |
| 1383 | 13.587318... | 193.50.135.38 | 192.168.1.14 | TCP | 1506 | [TCP segment of a reassembled PDU] |
| 1384 | 13.587344... | 192.168.1.14 | 193.50.135.38 | TCP | 66 | 54286 → 80 [ACK] Seq=1498 Ack=118346 Win=184832 Len=0 TSval=1113210 ... |

▶ Frame 509: 1506 bytes on wire (12048 bits), 1506 bytes captured (12048 bits) on interface 0
 ▶ Ethernet II, Src: ac:84:c9:cf:52:b0 (ac:84:c9:cf:52:b0), Dst: LiteonTe_7a:9c:b2 (24:fd:52:7a:9c:b2)
 ▶ Internet Protocol Version 4, Src: 193.50.135.38, Dst: 192.168.1.14
 ▶ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 54288 (54288), Seq: 1, Ack: 776, Len: 1440

1.3 Avec proxy

Avec le proxy, la connexion fonctionne de la même façon qu’au dessus, sauf que les échanges se font entre notre machine et le proxy qui relaye les informations, et non directement avec le site web. Nous nous sommes connectés sur le réseau de l’école pour cette capture d’écran, ainsi notre adresse ip est: 10.10.98.98 et l’adresse du proxy utilisé est: 51.255.166.82 (une machine OVH personnelle).

Figure 3: Avec Proxy



The image shows a Wireshark packet capture interface. The top toolbar and filter bar are visible. The filter bar contains the expression: `(ip.src == 51.255.166.82 or ip.dst == 51.255.166.82)`. The packet list pane shows several packets, with packet 1086 selected. The packet details pane shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane shows the raw data of the packet.

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|--------------|---------------|---------------|----------|--------|---|
| 700 | 9.5721791... | 51.255.166.82 | 10.10.98.98 | TCP | 54 | 7777 → 46030 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 701 | 9.5726083... | 10.10.98.98 | 51.255.166.82 | TCP | 74 | 46032 → 7777 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=... |
| 702 | 9.5875773... | 51.255.166.82 | 10.10.98.98 | TCP | 54 | 7777 → 46032 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 703 | 9.5878209... | 10.10.98.98 | 51.255.166.82 | TCP | 74 | 46034 → 7777 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=... |
| 704 | 9.6063805... | 51.255.166.82 | 10.10.98.98 | TCP | 54 | 7777 → 46034 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 1082 | 15.198788... | 10.10.98.98 | 51.255.166.82 | TCP | 74 | 46048 → 7777 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=... |
| 1083 | 15.211612... | 51.255.166.82 | 10.10.98.98 | TCP | 74 | 7777 → 46048 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PE... |
| 1084 | 15.211647... | 10.10.98.98 | 51.255.166.82 | TCP | 66 | 46048 → 7777 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2177904 TSecr=4... |
| 1086 | 15.211746... | 10.10.98.98 | 51.255.166.82 | HTTP | 753 | GET http://telecomnancy.univ-lorraine.fr/ HTTP/1.1 |
| 1091 | 15.225249... | 51.255.166.82 | 10.10.98.98 | TCP | 66 | 7777 → 46048 [ACK] Seq=1 Ack=688 Win=30336 Len=0 TSval=4252818515 TS... |
| 1092 | 15.310869... | 51.255.166.82 | 10.10.98.98 | TCP | 1434 | [TCP segment of a reassembled PDU] |
| 1093 | 15.310950... | 10.10.98.98 | 51.255.166.82 | TCP | 66 | 46048 → 7777 [ACK] Seq=688 Ack=1369 Win=32128 Len=0 TSval=2177929 TS... |
| 1094 | 15.311009... | 51.255.166.82 | 10.10.98.98 | TCP | 1434 | [TCP segment of a reassembled PDU] |
| 1095 | 15.311034... | 10.10.98.98 | 51.255.166.82 | TCP | 66 | 46048 → 7777 [ACK] Seq=688 Ack=2737 Win=35072 Len=0 TSval=2177929 TS... |
| 1096 | 15.311723... | 51.255.166.82 | 10.10.98.98 | TCP | 962 | [TCP Previous segment not captured] 7777 → 46048 [PSH, ACK] Seq=4105... |
| 1097 | 15.311740... | 10.10.98.98 | 51.255.166.82 | TCP | 78 | [TCP Window Update] 46048 → 7777 [ACK] Seq=688 Ack=2737 Win=37760 Le... |
| 1098 | 15.311752... | 51.255.166.82 | 10.10.98.98 | TCP | 1434 | [TCP Out-Of-Order] 7777 → 46048 [ACK] Seq=2737 Ack=688 Win=30336 Len... |
| 1099 | 15.311761... | 10.10.98.98 | 51.255.166.82 | TCP | 66 | 46048 → 7777 [ACK] Seq=688 Ack=5001 Win=40704 Len=0 TSval=2177929 TS... |
| 1100 | 15.313066... | 51.255.166.82 | 10.10.98.98 | TCP | 1434 | 7777 → 46048 [ACK] Seq=5001 Ack=688 Win=30336 Len=1368 TSval=4252818... |
| 1101 | 15.313077... | 10.10.98.98 | 51.255.166.82 | TCP | 66 | 46048 → 7777 [ACK] Seq=688 Ack=6369 Win=43520 Len=0 TSval=2177929 TS... |
| 1102 | 15.328226... | 51.255.166.82 | 10.10.98.98 | TCP | 1434 | 7777 → 46048 [ACK] Seq=6369 Ack=688 Win=30336 Len=1368 TSval=4252818... |

► Frame 696: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
► Ethernet II, Src: LiteonTe_7a:9c:b2 (24:fd:52:7a:9c:b2), Dst: CiscoInc_5e:8f:c0 (44:d3:ca:5e:8f:c0)
► Internet Protocol Version 4, Src: 10.10.98.98, Dst: 51.255.166.82
► Transmission Control Protocol, Src Port: 46028 (46028), Dst Port: 7777 (7777), Seq: 0, Len: 0

2 Conception

2.1 Algorithme général gérant un client à la fois

Data: Numéro de Port

Creation du Serveur Proxy;

while *Vrai* **do**

Reception Connection Client;

Recuperer Informations du message;

if *Message n'est pas une publicité* **then**

Envoyer Message au Serveur Web;

Reception du Message du serveur du Serveur Web;

Envoi du Message au Client;

else

end

end

Algorithm 1: Algorithme simple MyAdBlock

2.2 Précisions sur l'algorithme

La partie "Recupérer Information du message" comprend une récupération de la requête (*GET*, *CONNECT*...) et une récupération de l'adresse du serveur à contacter. Cette dernière récupération est implémentée telle que:

Algorithme 1: Algorithme de détermination d'adresse serveur à contacter (messages.c)

```
//Extrait hostname de "fromClient" et le met dans "host"
void getHost(char fromClient[], char host[]){
    int i=0;
    int j=0;
    char hostSearch[7] = "";

    //On passe jusqu'a ce qu'on arrive a "Host: "
    while(strcmp(hostSearch,"Host: ")!=0){
        for(j=0;j<6;j++){
            hostSearch[j] = fromClient[i+j];
        }
        hostSearch[6]='\0';
        i++;
    }

    //On saute le 2eme '/'
    i=i+5;
    j=0;
    //On recupere le nom jusqu'au prochain '/'
    while(fromClient[i] != '\n'){
        host[j] = fromClient[i];
        i++;
        j++;
    }
    host[j-1]='\0';
}
```

2.3 Implantation du multi-clients et passage Ipv4/Ipv6

Comme vu au TD2, nous avons utilisé la méthode *SELECT()* pour éviter l'utilisation du multi-processing ("fork" de serveurs) ce qui a permis de gérer les connexions multiples et simultanées. La primitive *getaddrinfo()* nous permet d'avoir une socket d'écoute pour IPv4 et une autre pour IPv6. L'IPv6 coté client du proxy est également géré dans *socket.c*

Algorithme 2: Algorithme de gestion multi-clients et gestion IPv4/IPv6 (myAdBlock.c)

```
for (;;) {
    // Initialisation des fd_set puis SELECT
    pset=rset;
    nbfd=select(maxfdpl,&pset,NULL,NULL,NULL);
    if (nbfd== -1) { erreur ... }

    // si connexion client
    if( FD_ISSET(server[0], &pset) || FD_ISSET(server[1], &pset) ){
        i = placelibre(tab_ecoute_clients);

        // si c'est en ipv4
        if( FD_ISSET(server[0], &pset) ){
            tab_ecoute_clients[i] = newCommunicationSock(server[0]);
        }

        // si c'est en ipv6
        if( FD_ISSET(server[1], &pset) ){
            tab_ecoute_clients[i] = newCommunicationSock(server[1]);
        }

        // mise a jour des variables
        FD_SET(tab_ecoute_clients[i], &rset);
        maxfdpl = MaJ_maxFD(tab_ecoute_clients[i],maxfdpl);
        nbfd--;
    }

    i=0;
    //Parcours des tableau des clients et serveurs connectes
    while((nbfd>0)&&(i<FD_SETSIZE)){

        // on regarde si on a une reponse du serveur
        if(tab_ecoute_servers[i] >= 0
        && FD_ISSET(tab_ecoute_servers[i], &pset)){
            messageDuServeur(...);
            nbfd--;
        }

        // on regarde si on a une requete du client
        if(tab_ecoute_clients[i] >= 0
        && FD_ISSET(tab_ecoute_clients[i], &pset)){
            maxfdpl = messageDuClient(...);
            nbfd--;
        }

        i++;
    }
}
```

2.4 Principe d'exclusion des publicités

Le code source de *myAdBlock* ainsi que la liste *MyAdListLight* des hosts de publicités sont dans le dossier *src/*. Libre à vous d'ajouter des hosts dans la liste.

Cette liste est une version allégée et plus simple de la liste de [EasyList](#), qui constitue notre banque de base de hosts de publicités. Voir [EasyList pour AdBlock Plus 2.0](#) pour consulter la banque complète. Chaque connexion entrante est filtrée, parsée et comparée à cette liste. Si elle appartient à cette liste, cette connexion est simplement bloquée (il s'agit d'une publicité). Compte tenu du fait que notre liste est réduite et ne tient pas compte des expressions régulières de la liste originale, *myAdBlock* est donc moins efficace que *AdBlock* (utilisant la même banque), mais est beaucoup plus simple et léger; et il est plus simple d'appréhender son fonctionnement.

On remarque par ailleurs que notre 'bloqueur', ou plutôt comparateur, se matérialise par l'appel à *contains()* de *util.c*. Il consiste à vérifier que chaque token de notre liste *MyEasyListLight* n'est pas contenu dans l'URL de la connexion entrante (voir Algorithme 3 et 4).

Algorithme 3: Fonction *contains()* (*util.c*)

```
bool contains(char URL[]){
    int i;
    for(i=0;i<sizeList;i++){
        if(strstr(URL, MyAdList[i])!=NULL){
            return true;
        }
    }
    return false;
}
```

Algorithme 4: Fonction *iniListe()* qui charge la liste *MyAdListLight*(*util.c*)

```
void iniListe() {
    FILE * fp; char * line = NULL; size_t len = 0; ssize_t read; int i=0;

    fp = fopen("src/MyAdListLight", "r");
    if (fp == NULL) exit(EXIT_FAILURE);

    // on compte le nb de ligne
    while ((read = getline(&line, &len, fp)) != -1) { i++; }

    MyAdList = malloc(i*sizeof(char*));
    sizeList=i;

    // on se remet le debut du fichier
    fseek(fp, 0, SEEK_SET); i=0;

    while ((read = getline(&line, &len, fp)) != -1) {
        MyAdList[i]=malloc(read*sizeof(char));
        strncpy(MyAdList[i], line, read-1);
        i++;
    }
    fclose(fp);
}
```


2.5 Extension du proxy : HTTPS

Pour gérer les connexions HTTPS entrantes, nous avons utilisé la méthode *CONNECT*, permettant d'établir un tunnel sans déchiffrement. Une fois captée, la connexion HTTPS est filtrée comme une connexion HTTP standard. On peut le voir dans l'algorithme ci-dessous.

Algorithme 5: Partie gérant l'HTTPS issue du CONNECT.

```
else if (strcmp(typeRequete, "CONNECT") == 0){

    // On recupere le hostname du client
    getHost(fromClient, host);
    host[strlen(host)-4] = '\0';

    if (contains(host) == false){
        // on cree le client sur le port d'ecoute 443
        tab_servers[i] = newClient(host, "443");

        // Mise a jour variables
        FD_SET(tab_servers[i], rset);
        maxfdpl = MaJ_maxFD(tab_servers[i], maxfdpl);

        // on repond au client (accept):
        // "HTTP/1.0 200 Connection established";
        send(tab_clients[i], buffConnectionEstablished,
            strlen(buffConnectionEstablished), 0);
    }
    // si c'est une pub
    else {
        printf("_bloque_");
        send(tab_clients[i], buffOK, strlen(buffOK), 0);
        close(tab_clients[i]);
        FD_CLR(tab_clients[i], rset);
        tab_clients[i] = -1;
    }

    printf("_%s_ %s\n", typeRequete, host);
}
```

3 Bibliographie

http://livre.g6.asso.fr/index.php/Exemple_de_client/serveur_TCP
http://livre.g6.asso.fr/index.php/Programmation_d'applications_bis
<https://adblockplus.org/filter-cheatsheet>
<http://manpagesfr.free.fr/man/man3/getaddrinfo.3.html>
https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
<https://en.wikipedia.org/wiki/HTTPS>
https://en.wikipedia.org/wiki/Transport_Layer_Security