

TD0 – Introduction

1 - Analyse de trames TCP et autres protocoles avec Wireshark

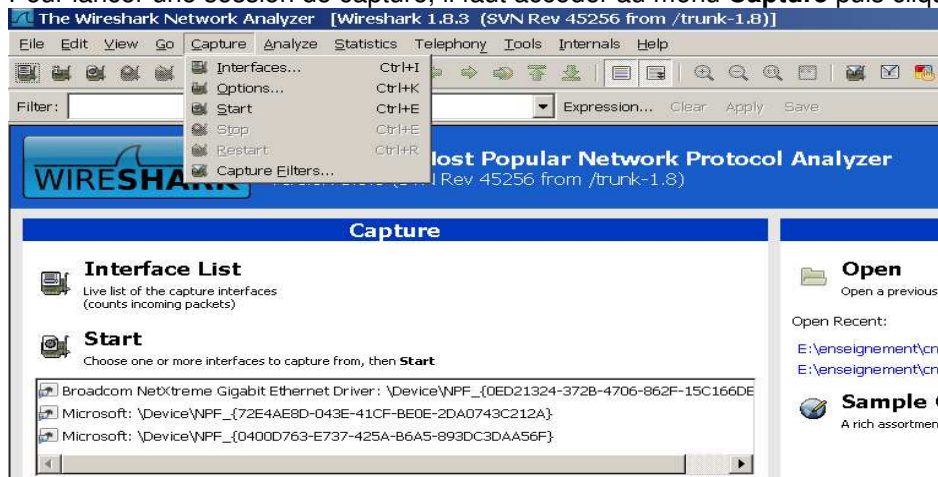
Présentation

Wireshark est un analyseur de trafic réseau, ou "sniffer" qui se télécharge gratuitement sur le Net. Il utilise une interface graphique basée sur GTK+, il est basé sur la bibliothèque winpcap, qui fournit des outils pour capturer les paquets réseau.

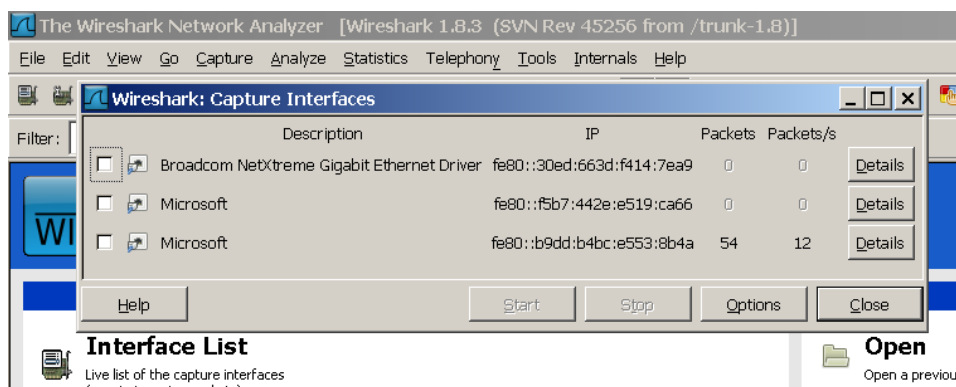
Utilisation de Wireshark

Le principe est simple, vous lancez une session de capture à l'aide du menu **Capture**. Cette session peut être interactive ou pas. En d'autres termes, les paquets capturés peuvent être affichés au fur et à mesure ou à la fin de la capture.

Pour lancer une session de capture, il faut accéder au menu **Capture** puis cliquer sur l'option **Interface....**



Apparaît la boîte de dialogue qui permet de spécifier sur quelle interface vous souhaitez « écouter » les paquets.



Sélectionnez l'interface qui voit passer les packets et cliquez sur « start ».

Durant la capture, une boîte de dialogue récapitule les paquets qui sont conservés. En même temps les paquets apparaissent dans la fenêtre principale. L'appui du bouton Stop de l'interface permet d'arrêter la capture. Les paquets deviennent disponibles dans la fenêtre principale s'ils n'étaient pas déjà disponibles

Wireshark 1.8.3 (SVN Rev 45256 from /trunk-1.8)

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
424	11.469365000	GemtekTe_44:95:47	Broadcast	ARP	60	Who has 10.40.0.17 Tell 10.40.6.76
425	11.878421000	GemtekTe_44:95:47	Broadcast	ARP	60	Who has 10.40.0.17 Tell 10.40.6.76
426	11.879715000	GemtekTe_44:95:47	Broadcast	ARP	60	Who has 10.40.0.17 Tell 10.40.6.76
427	12.082268000	Fe80::ddb:d88e:aa03:7a8	ff02::1:3	LLMNR	84	Standard query 0x5fae A J030
428	12.082529000	10.40.6.132	224.0.0.252	LLMNR	64	Standard query 0x5fae A J030
429	12.287385000	Fe80::ddb:d88e:aa03:7a8	ff02::1:3	LLMNR	84	Standard query 0x5fae A J030
430	12.287670000	10.40.6.132	224.0.0.252	LLMNR	64	Standard query 0x5fae A J030
431	12.287887000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.1777 Tell 10.40.0.1
432	12.394403000	10.40.6.56	63.245.217.39	TCP	54	50383 > http [FIN, ACK] Seq=640 Ack=419 Win=65280 Len=0
433	12.491889000	10.40.6.132	10.40.255.255	NBNS	92	Name query NB J030<20>
434	12.492558000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.83? Tell 10.40.0.1
435	12.580110000	63.245.217.39	10.40.6.56	TCP	60	http > 50383 [FIN, ACK] Seq=419 Ack=641 Win=16384 Len=0
436	12.580198000	10.40.6.56	63.245.217.39	TCP	54	50383 > http [ACK] Seq=641 Ack=420 Win=65280 Len=0
437	13.106240000	10.40.6.132	10.40.255.255	NBNS	92	Name query NB J030<20>
438	13.311044000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.1777 Tell 10.40.0.1
439	13.516078000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.83? Tell 10.40.0.1
440	13.925452000	10.40.6.132	10.40.255.255	NBNS	92	Name query NB J030<20>
441	14.539866000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.1777 Tell 10.40.0.1
442	14.540134000	De11_63:d2:56	Broadcast	ARP	60	Who has 10.40.6.83? Tell 10.40.0.1
443	14.744618000	Fe80::ddb:d88e:aa03:7a8	ff02::1:3	LLMNR	84	Standard query 0xca94 A J030

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
 Ethernet II, Src: De11_63:d2:56 (00:22:19:63:d2:56), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 Address Resolution Protocol (request)

```

0000  ff ff ff ff ff ff ff ff 19 63 d2 56 08 06 00 01  ..... " .C.V....
0010  08 00 06 04 00 01 00 22 19 63 d2 56 0a 28 00 01  ..... " .C.V.(.
0020  00 00 00 00 00 00 0a 28 06 b1 00 00 00 00 00 00  ..... ( .....
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
  
```

Analyse des échanges

1 – Lancer simplement la capture de paquets et attendez quelques instants. Couper la capture et observer tous les paquets capturés :

- Quels protocoles étaient actifs dans le réseau ?
- Lesquels connaissez-vous ?
- Arrivez-vous à déterminer qui a effectué une requête vers qui ?

2 – Lancer la capture de paquets, puis lancer un navigateur et connectez vous à : ffctlr.free.fr

- Combien de requêtes http ont-elles été faites ? Pourquoi ?
- Pour toutes ces requêtes combien de connexion du protocole TCP ont-elles été créées ?
- Identifier les paquets relatifs à cette connexion.
- Identifier les différentes phases d'un échange TCP. Quelles sont les adresses IP et les ports utilisés ?

3 – Recommencez l'analyse avec d'autres sites de votre choix.

Manipulation amusante :

- Lancer la capture de paquets, puis lancer un navigateur et connectez vous à l'adresse : <http://ffctlr.free.fr>
- Remplir le formulaire et le « soumettre »
- Arrêter la capture, repérez les paquets relatifs à la connexion et essayez d'identifier les champs du formulaire.

Remarque : les données sont généralement transportées dans une trame de type POST

2 – Anatomie d'une application client-serveur

Ci-dessous, figurent les codes de deux applications en C, qui échangent des données via le protocole TCP/IP de votre machine, en utilisant l'interface Socket.

Compilez les applications, lancer l'analyseur de trames, procédez à leur exécution et observez les échanges réalisés sur le réseau. Vos commentaires ???

Codes

1 - Programme client

```
#include <stdio.h>
#include <netdb.h>
```

```
#include <fcntl.h>
#include <string.h>           // chaines de caracteres
#include <sys/socket.h>       // interface socket
#include <netinet/in.h>       // gestion adresses ip
#include <sys/types.h>
#include <unistd.h>

#define SERV "127.0.0.1"      // adresse IP = boucle locale
#define PORT 12345           // port d'ecoute serveur
int port,sock;               // n°port et socket
char mess[80];               // chaine de caracteres

struct sockaddr_in serv_addr; // zone adresse
struct hostent *server;       // nom serveur

creer_socket()
{ port = PORT;
  server = gethostbyname(SERV); // verification existence adresse
  if (!server){fprintf(stderr, "Problème serveur \"%s\"\n", SERV);exit(1);}
  // creation socket locale
  sock = socket(AF_INET, SOCK_STREAM, 0); // AF_INET=famille adresse internet
                                           // SOCK_STREAM= mode connecte-TCP
  bzero(&serv_addr, sizeof(serv_addr)); // preparation champs entete
  serv_addr.sin_family = AF_INET;       // Type d'adresses
  bcopy(server->h_addr, &serv_addr.sin_addr.s_addr,server->h_length);
  serv_addr.sin_port = htons(port);     // port de connexion du serveur
}

main()
{ creer_socket();               // creation socket

if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {perror("Connexion impossible:");exit(1);} // connexion au serveur
  printf ("connexion avec serveur ok\n");

  strcpy(mess1,"");
  while (strncmp(mess1,"fin",3)!=0)
  { printf ("Votre message : ");
    gets(mess1);
    write(sock,mess1,80);
    read(sock,mess2,80);
    printf ("Le serveur me dit :%s\n ",mess2);
  }

  close (sock);                // fermeture connexion
}
```

2 - Programme serveur

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define PORT 12345
int sock, socket2, lg;
char mess[80];
struct sockaddr_in local;      // champs pour adresse de la machine
struct sockaddr_in distant;    // champs pour adresses distantes (clients)
```

```

void creer_socket()
{ // preparation des champs d entete
  bzero(&local, sizeof(local)); // Mise à zéro du champs adresse
  local.sin_family = AF_INET; // adresse type IP
  local.sin_port = htons(PORT); // Numéro de port
  local.sin_addr.s_addr = INADDR_ANY; // adresse IP du poste par défaut
  bzero(&(local.sin_zero),8);

  lg = sizeof(struct sockaddr_in);
  // Creation de la socket en mode TCP/IP
  if((sock=socket(AF_INET, SOCK_STREAM,0)) == -1){perror("socket"); exit(1);}
  // Nommage de la socket avec le n° de port 12345
  if(bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr)) == -1) {perror("bind");exit(1);}
}

main()
{ creer_socket(); // Création de la socket en mode TCP/IP

  listen(sock,5); // mise à l écoute
  while(1) // Boucle de traitement d'un client
  {
    printf ("En attente d un client\n");
    socket2=accept(sock, (struct sockaddr *)&distant, &lg);
    printf ("client connecte \n");
    if (fork()==0)
    {
      strcpy(mess,"");
      while (strncmp(mess,"fin",3)!=0)
      { read(socket2,mess,80);
        printf ("le client me dit %s \n",mess);
        write(socket2, "message reçu !",80);
      }
      close(socket2); // fermeture socket d'échange
      exit 1 ;
    }
  }
}

```

3 - Et si on faisait un DoS ???

Compléter le programme serveur.

Un DoS basique !

- 1 – Récupérez les adresses IP et numéro de port du serveur sur un deuxième poste (sous linux)
- 2 – Créez et exécutez le shell ci-dessous

```

while [ 1==1 ]
do
  telnet @ip @port &
done

```

Remarque : Vous pouvez vous y mettre à plusieurs.

- 3 – Vos remarques