Visualiser les échanges de courriels chiffrés avec Wireshark

Présentation

Un **analyseur de trame** est un outil **de base** de l'informaticien car il permet de comprendre ce qu'il se passe à un niveau très bas. Il permet aussi aussi de mettre en évidence de nombreux concepts théoriques du cours.

Wireshark (anciennement Ethereal) est un est un logiciel libre d'**analyse de protocole**, ou **packet sniffer**, utilisé dans le **dépannage** et l'**analyse du fonctionnement** des réseaux informatiques. Il est utilise pour **diagnostiquer** des dysfonctionnement dans un réseau informatique.

Un analyseur de protocoles (ou analyseur de réseaux ou de paquets) est un logiciel permettant **d'intercepter** et de consigner le trafic des données transférées sur un réseau de données. L'analyseur **capture** chaque **PDU** (protocol data unit - unité de données de protocole) des flux de données circulant sur le réseau.

Il permet de décoder et d'analyser leur contenu conformément aux spécifications **RFC** ou autres appropriées.

Wireshark est programmé pour reconnaître la structure de différents protocoles réseau.

Installation de Wireshark dans la VM Ubuntu

Avant d'installer un nouveau logiciel, mettrez à jour votre environnement Ubuntu.

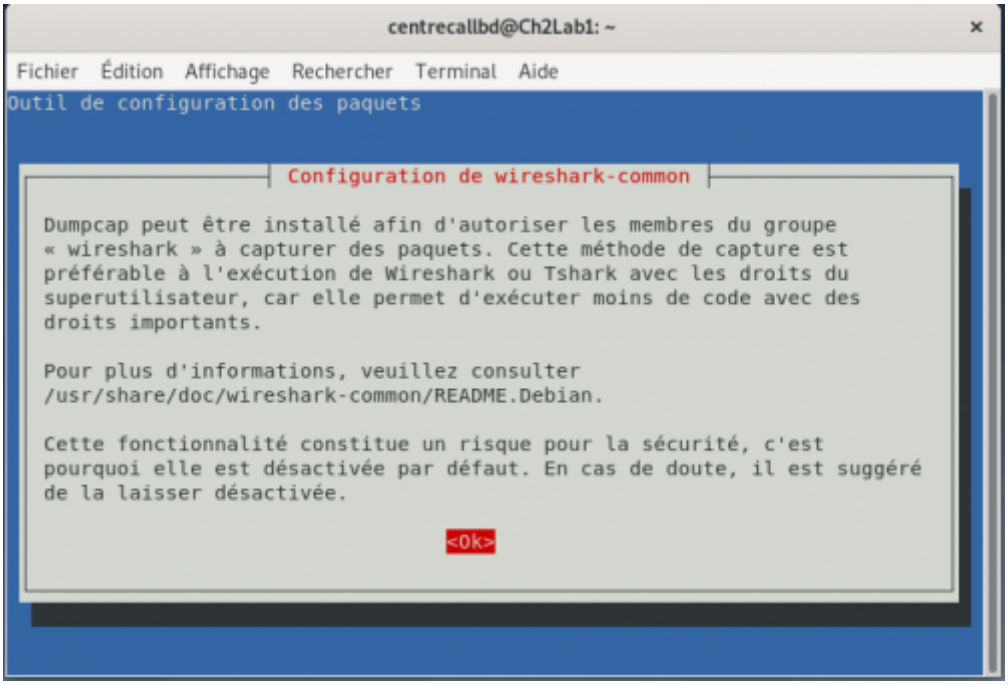
- lancez un Terminal
- Mettez à jour votre environnement Ubuntu

```
$ sudo apt update
$ sudo apt upgrade
```

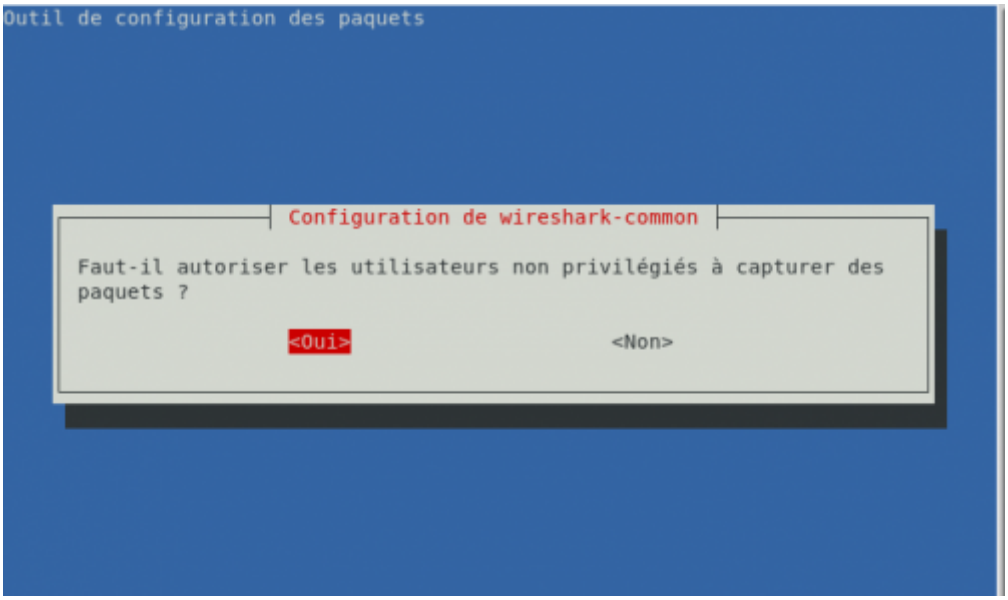
- installez Wireshark

```
$ sudo apt install wireshark
```

Durant l'installation il vous est demandé **d'autoriser** l'installation de **Dumcap**. Utilisez la touche **TAB** pour sélectionner le bouton **OK** et validez avec la touche entrée :



- **Autorisez** les utilisateurs non privilégiés à **utiliser Wireshark** :

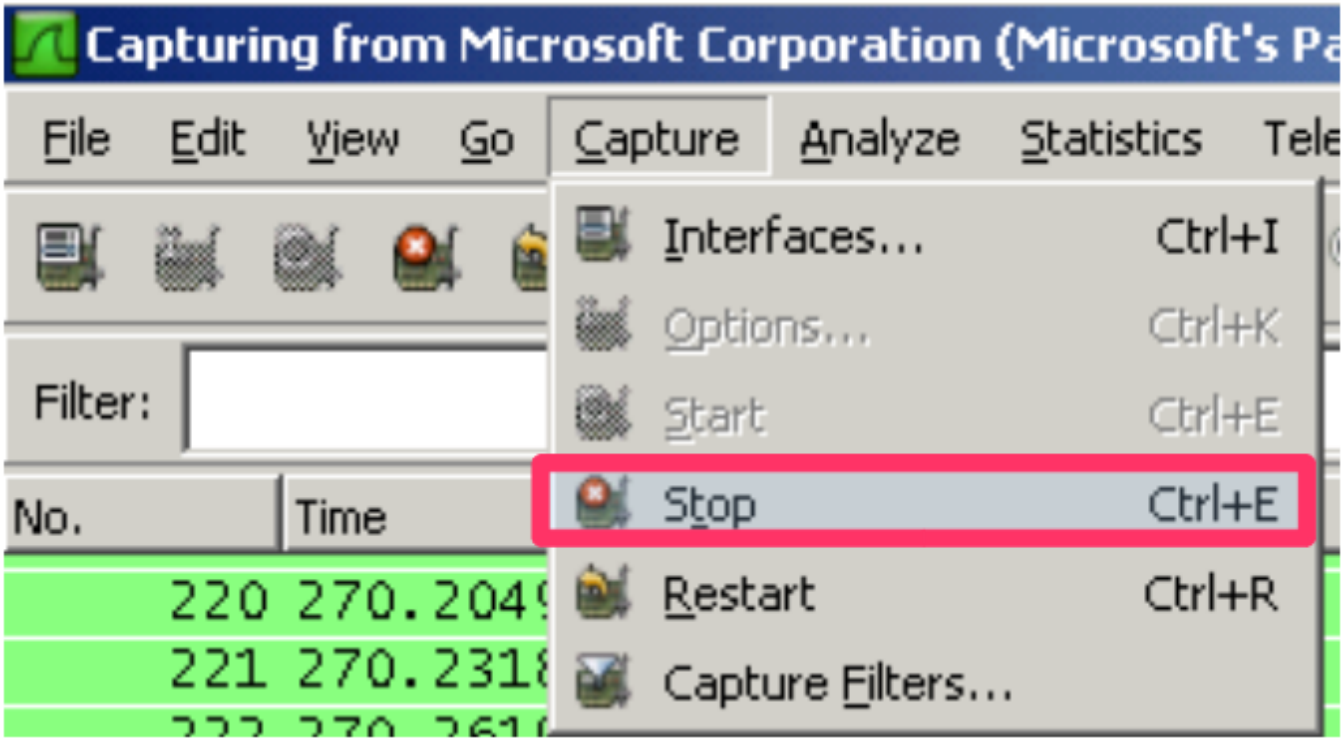


Prise de contact

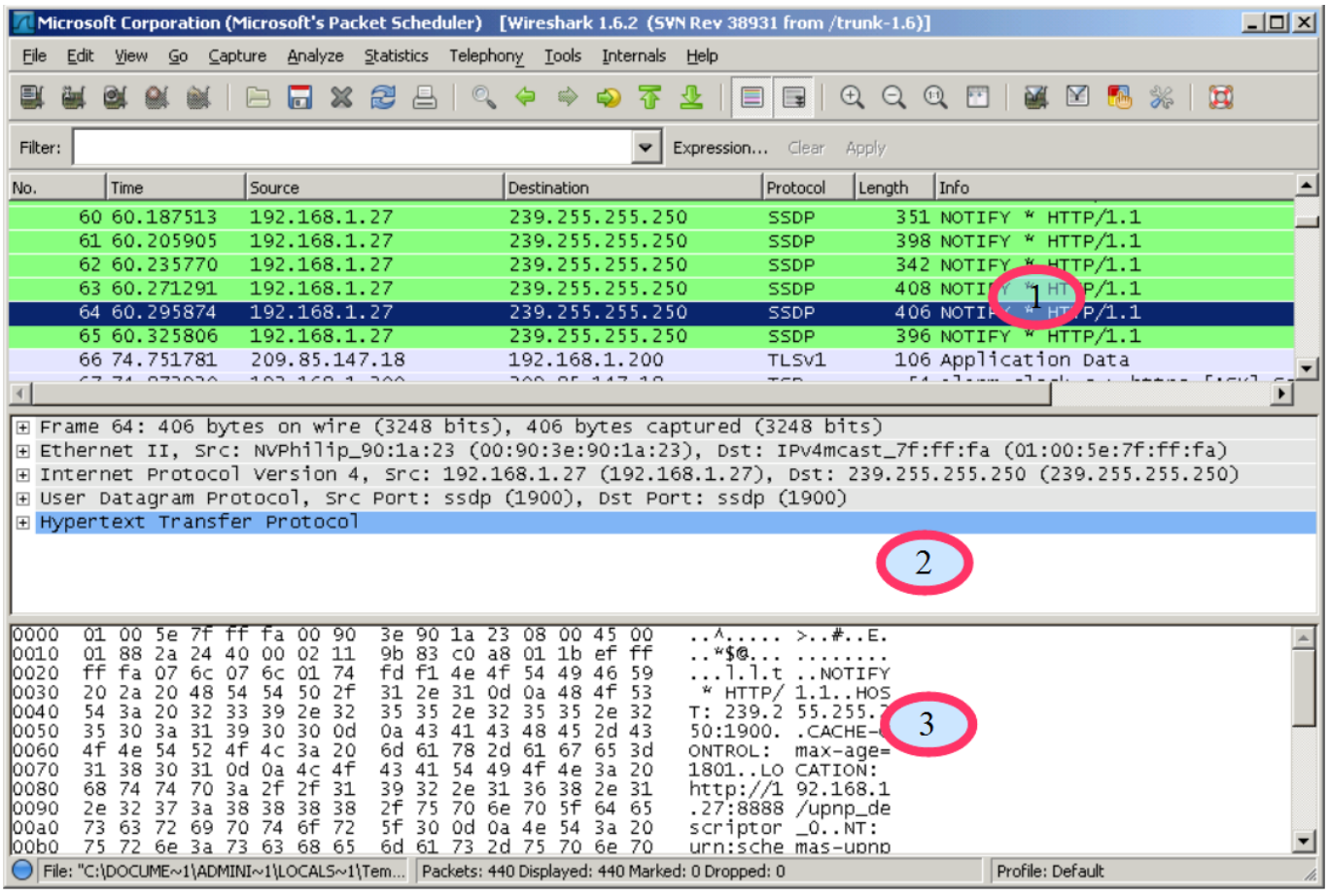
Une fois installé, le logiciel se présente ainsi :

Vous pouvez démarrer une capture en cliquant tout simplement sur l'interface réseau qui vous intéresse. Vous ne verrez donc que le trafic réseau vu par cette carte réseau.

- **Démarrez une capture** ; au bout de quelques instants vous verrez des paquets réseau apparaître dans la fenêtre, ce qui montre que même si vous ne faites rien, il y a des informations qui circulent sur le réseau !
- **Arrêtez** la capture des trames :



Wireshark permet de donner des informations très détaillées. Examinez l'écran principal du logiciel :



On observe en trois parties :

- 1. La **liste des trames Ethernet capturées**. Elles sont chacune numérotées et horodatées

par Wireshark (ces données ne figurent donc pas dans la trame d'origine).

- 2. Pour chaque trame, sa **structure** est présentée sous une forme **hiérarchique** (ainsi ce que vous voyez dans le volet 2 est le détail de la trame numéro 2)
- 3. Le volet 3 est la même chose que le volet 2 mais sous une forme **brute** non structurée avec une présentation ASCII et hexadécimale. Vous pouvez la présenter en binaire (clic droit dans le volet 3).

Examen détaillé

Examinez en détail le volet 2. Vous pouvez cliquer sur les croix pour développer les contenus. Ce volet met en évidence le phénomène **d'encapsulation** :

Le premier élément concerne la **trame** proprement dite (taille, temps, etc.) :

```
⊟ Frame 64: 406 bytes on wire (3248 bits), 406 bytes captured (3248 bits)
    Arrival Time: Sep 29, 2011 18:40:14.997940000 Paris, Madrid (heure d'été)
    Epoch Time: 1317314414.997940000 seconds
    [Time delta from previous captured frame: 0.024583000 seconds]
    [Time delta from previous displayed frame: 0.024583000 seconds]
    [Time since reference or first frame: 60.295874000 seconds]
    Frame Number: 64
    Frame Length: 406 bytes (3248 bits)
    Capture Length: 406 bytes (3248 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:udp:http]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80]
```

Ensuite, en montant d'un cran est présentée la partie liée à **Ethernet**. On retrouve les adresses physiques de **destination** et de **source**, également le type trame de niveau supérieur (ici IP **0x0800**) :

```
⊟ Ethernet II, Src: NVPhilip_90:1a:23 (00:90:3e:90:1a:23), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
  ⊟ Destination: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
      Address: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  ⊟ Source: NVPhilip_90:1a:23 (00:90:3e:90:1a:23)
      Address: NVPhilip_90:1a:23 (00:90:3e:90:1a:23)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Type: IP (0x0800)
```

A la couche supérieure, c'est la partie IP :

```
⊟ Internet Protocol Version 4, Src: 192.168.1.27 (192.168.1.27), Dst: 239.255.255.250 (239.255.255.250)
    Version: 4
    Header length: 20 bytes
  ⊟ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
      0000 00.. = Differentiated Services Codepoint: Default (0x00)
      .... ..00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
    Total Length: 392
    Identification: 0x2a24 (10788)
  ⊞ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 2
    Protocol: UDP (17)
  ⊞ Header checksum: 0x9b83 [correct]
    Source: 192.168.1.27 (192.168.1.27)
    Destination: 239.255.255.250 (239.255.255.250)
```

On retrouve les **adresses IP source** et **destination du paquet**. De plus, certaines données correspondent à des bits d'un octet particulier (**differentiated services field**). Des données techniques comme la longueur du paquet, le numéro de séquence, le temps à vivre (**TTL** ou Time To Live), l'identité du protocole supérieur (**UDP**) sont nécessaires au fonctionnement de cette couche.

En montant encore d'un niveau on observe la partie **transport**. Ici il s'agit de **UDP** qui est un protocole simple sans gestion des erreurs, son contenu est beaucoup plus simple que **TCP** :

```
⊟ User Datagram Protocol, Src Port: ssdp (1900), Dst Port: ssdp (1900)
    Source port: ssdp (1900)
    Destination port: ssdp (1900)
    Length: 372
  ⊟ Checksum: 0xfdf1 [validation disabled]
      [Good Checksum: False]
      [Bad Checksum: False]
```

Comme à chaque fois, une information concernant le protocole de niveau supérieur (ici **ssdp** pour Simple Service Discovery Protocol) est intégré. Nous retrouvons également la notion de **port source** et de **destination** mais aussi de **checksum** qui permet le contrôle d'erreur.

Et enfin, on aborde la partie **application**. vous remarquerez que **Wireshark** sait mettre en relation les **données structurées** et les **données brutes**. Ainsi, sur n'importe quelle couche du paquet, si vous sélectionnez un élément, celui-ci est mis en évidence dans le dernier volet :

```
⊟ Hypertext Transfer Protocol
  ⊟ NOTIFY * HTTP/1.1\r\n
    ⊞ [Expert Info (Chat/Sequence): NOTIFY * HTTP/1.1\r\n]
      Request Method: NOTIFY
      Request URI: *
      Request Version: HTTP/1.1
    HOST: 239.255.255.250:1900\r\n
    CACHE-CONTROL: max-age=1801\r\n
    LOCATION: http://192.168.1.27:8888/upnp_descriptor_0\r\n
    NT: urn:schemas-upnp-org:service:RenderingControl:1\r\n
    NTS: ssdp:alive\r\n
    SERVER: Linux/2.6.10_dev-VT8610 UPnP/1.0 MediabolicUPnP/1.8.225\r\n
    USN: uuid:00000081-0000-5000-0000-00903E901A23::urn:schemas-upnp-org:service:RenderingControl:1\r\n
    \r\n
```

Pour information, le paquet présenté ici correspond au protocole de découverte **UPnP** basé sur le protocole **SSDP** (Simple Service Discovery Protocol). Il s'agit d'un **lecteur Philips MCI730** qui diffuse sur le réseau (adresse **multicast** 239.255.255.250) pour prévenir de ses services (diffuser de la musique).

Retour Accueil Bloc3

- Bloc3