

Jérémy Barrette – 1736976

Alexis Vailles – 1742139

**Rapport TP #3 :**

**Analyse d’applications client-serveur avec WireShark**

Soumis à : Kadi, Mehdi

INF3405 (01 – B1) – Réseaux informatiques

Session Automne 2018

École Polytechnique de Montréal

Mercredi le 5 décembre 2018

# **Analyse de l’application client/serveur du laboratoire 1**

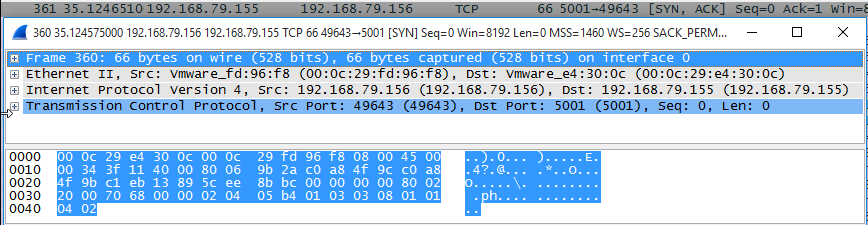
**1)** Le filtre appliqué est :

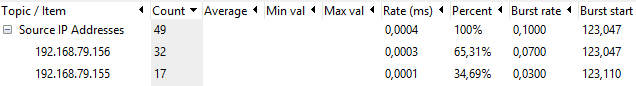
ip.addr == 192.168.79.156 && ip.addr == 192.168.79.155

Il est à noter que le filtre utilisé sur la capture d’écran est :

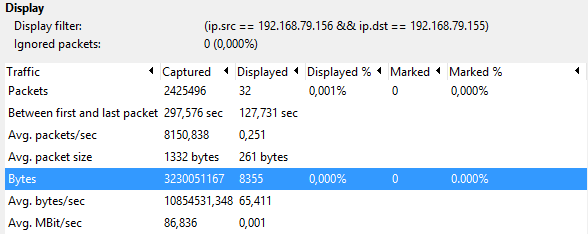
(ip.src == 192.168.79.156 && ip.dst == 192.168.79.155) || (ip.src == 192.168.79.155 && ip.dst == 192.168.79.156)

Ce filtre est un équivalent au filtre énoncé plus haut, mais est passablement plus compliqué à écrire. C’est pourquoi nous l’avons changé après avoir fait la capture d’écran.

**2)** Le protocole de la couche 4 utilisé est le protocole **TCP**.

**3)**

**Client => Serveur**

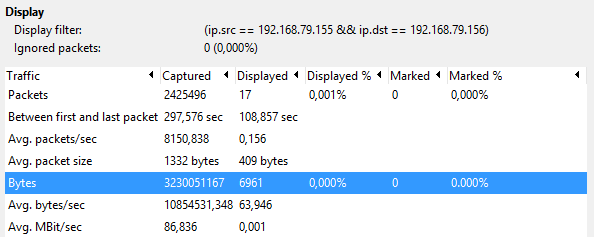
Filtre appliqué : ip.src == 192.168.79.156 && ip.dst == 192.168.79.155

Il y a **32** paquets envoyés du client vers le serveur;

Il y a **8355** octets envoyés du client vers le serveur.

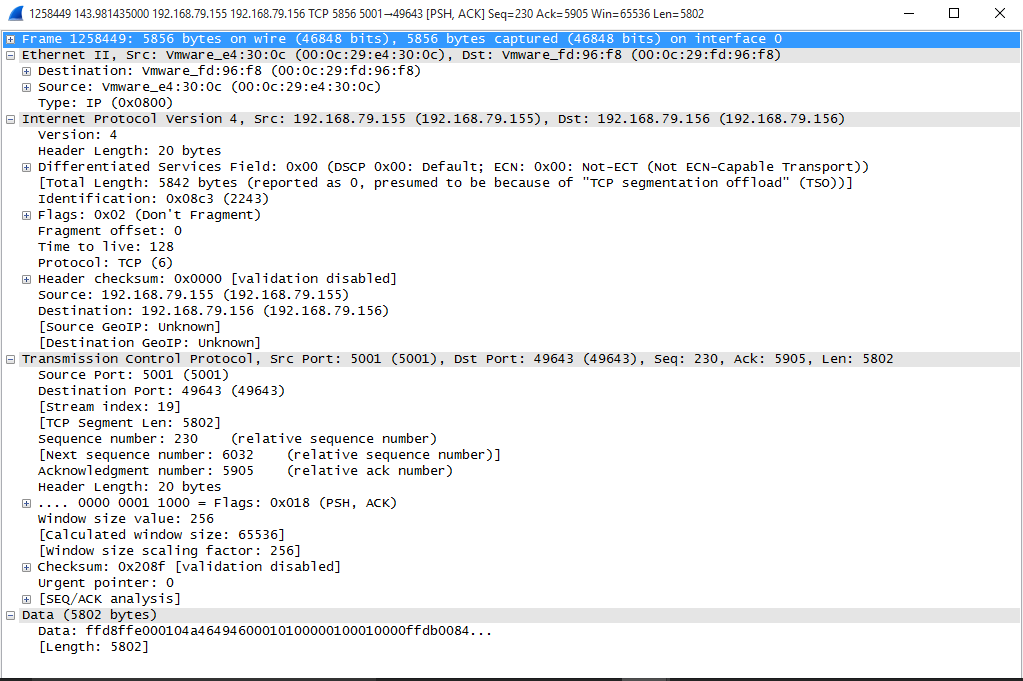
**Serveur => Client**

Filtre appliqué : ip.src == 192.168.79.155 && ip.dst == 192.168.79.156

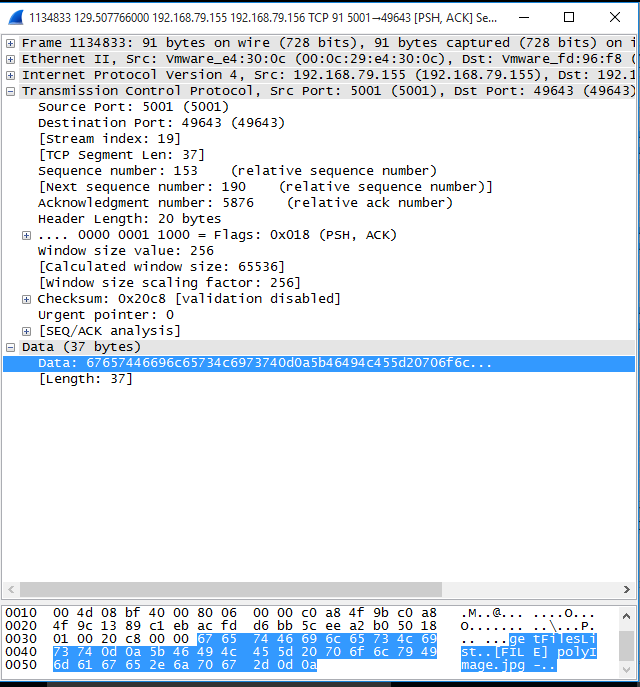


Il y a **17** paquets envoyés du serveur vers le client;

Il y a **6961** octets envoyés du serveur vers le client.

**4)**

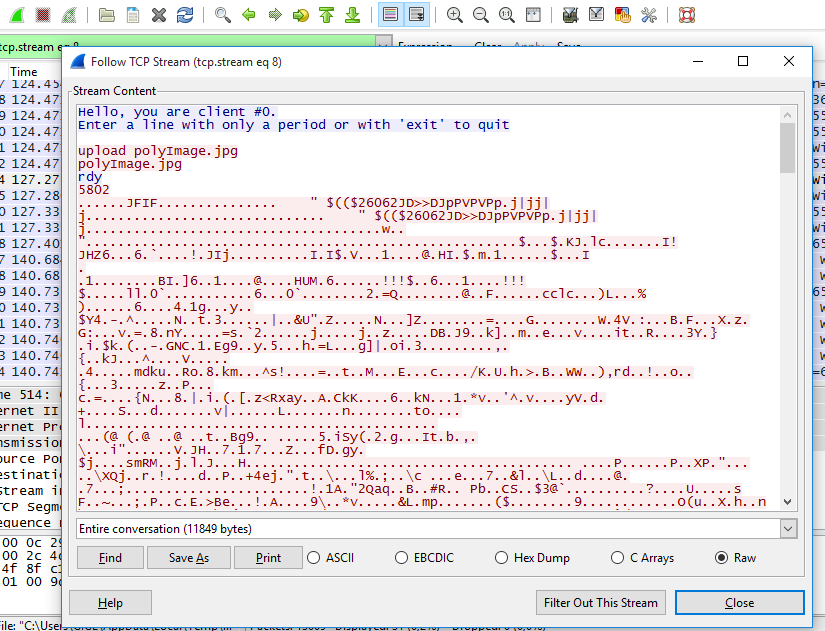
Le serveur envoie un paquet de 5856 octets vers le client, ce qui est plus élevé que 1518. Ce paquet a pu être envoyé parce que ses données n’étaient pas fragmentables. Il a été envoyé selon le format **jumbo frame**.

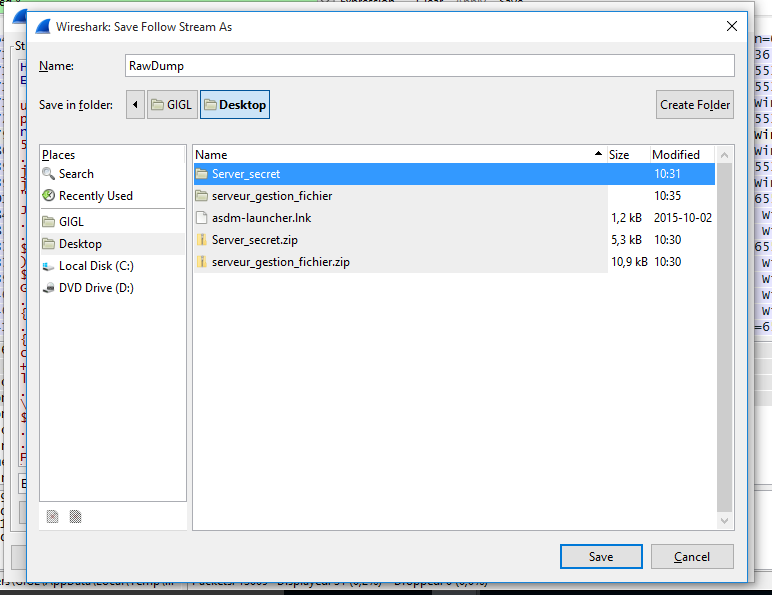
**5)**

La commande « ls » nous permet d’intercepter la liste de fichiers et dossiers contenus dans le serveur.

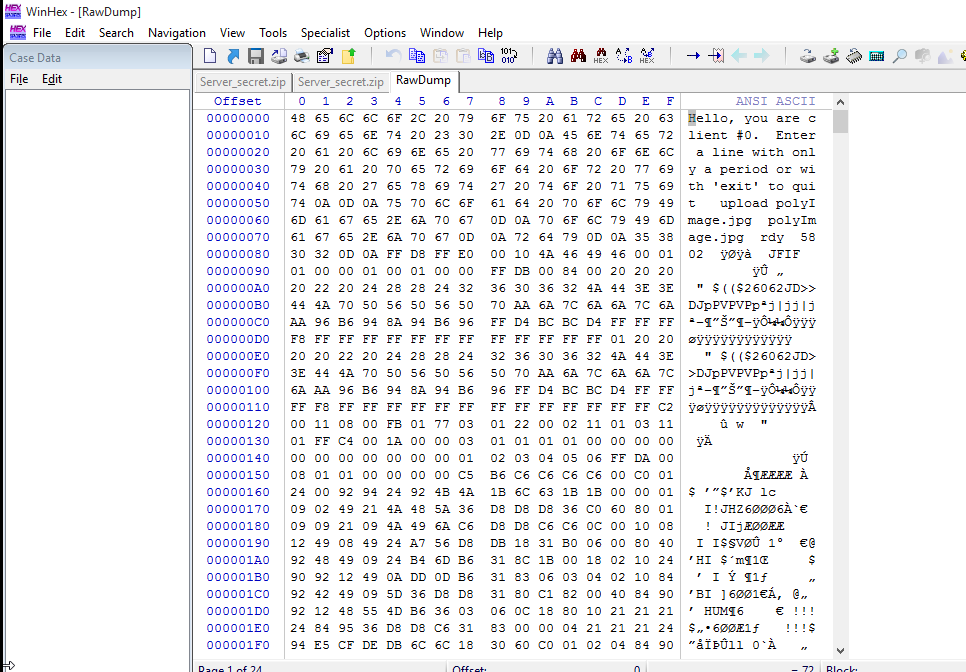
**6)** Comme nous ne sommes pas parvenus à effectuer cet exercice correctement durant les heures de laboratoire, nous avons emprunté les captures d’écran d’une autre équipe. Voici donc comment, avec Wireshark, on peut extraire l’image envoyée par le client ou l’image envoyer par le serveur vers le client :

**Étape 1 :** On sauvegarde le *TCP stream* en format *.raw***:**



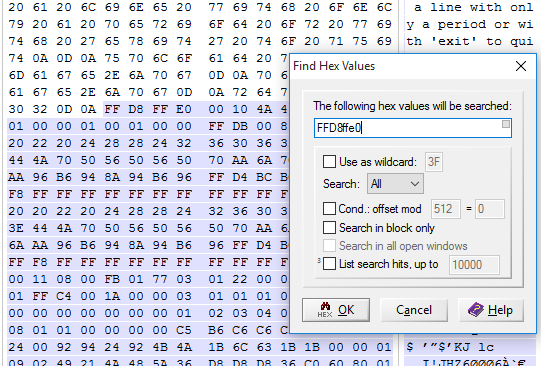


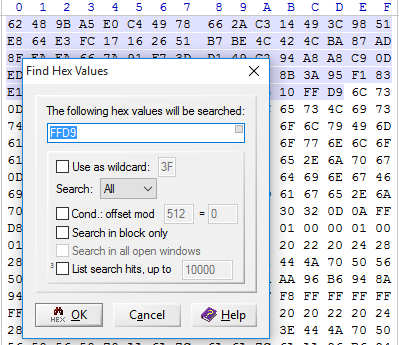
**Étape 2 :** On ouvre le *TCP stream* dans *WinHex* :

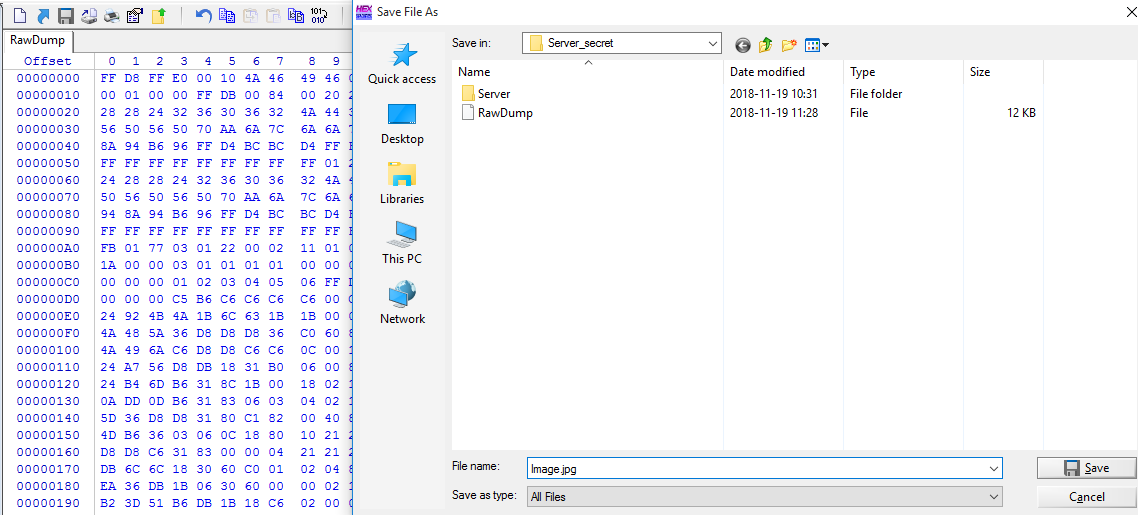


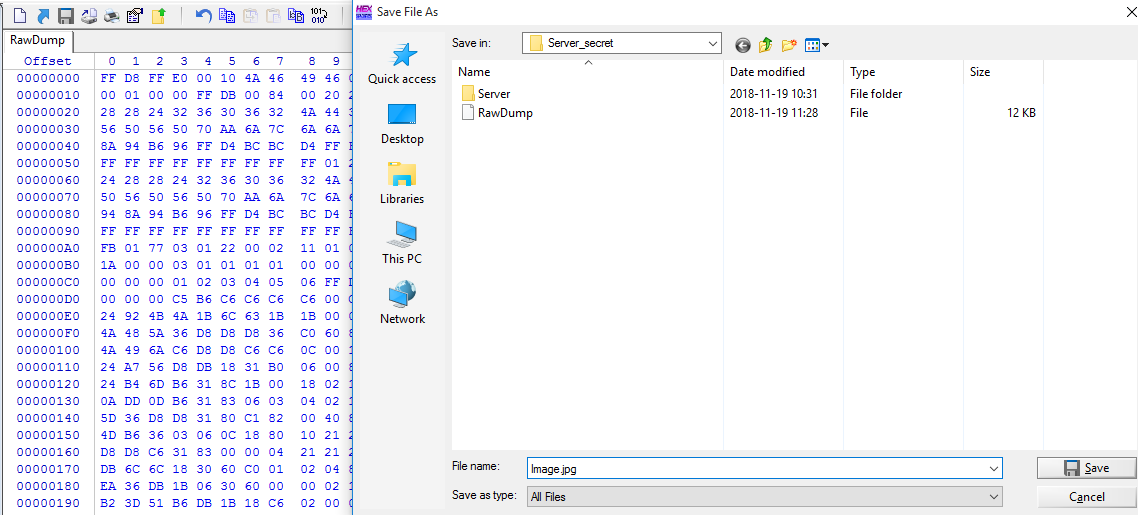
**Étape 3 :** Parmi les données représentées, on repère où commencent les données associées à l’image (après l’en-tête), et où elles se terminent :

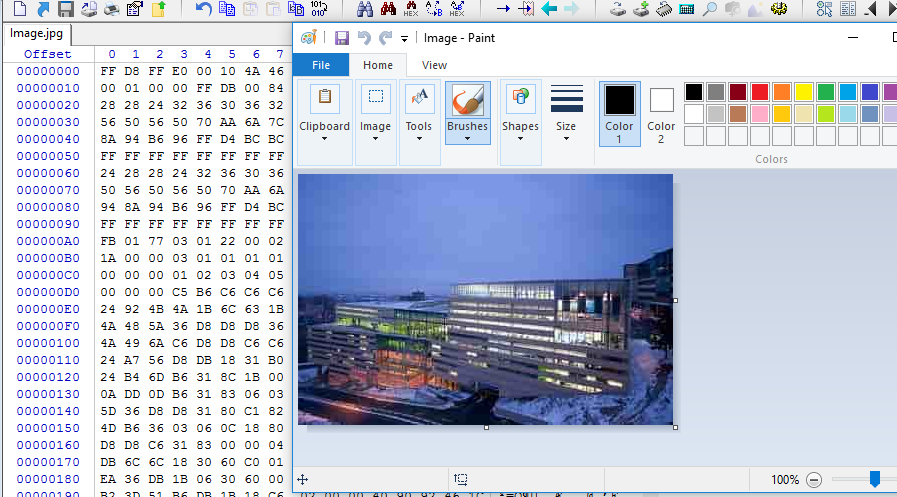
Début :



Fin :

**Étape 4 :** On ne conserve que les données associées à l’image (donc on supprime le reste) :

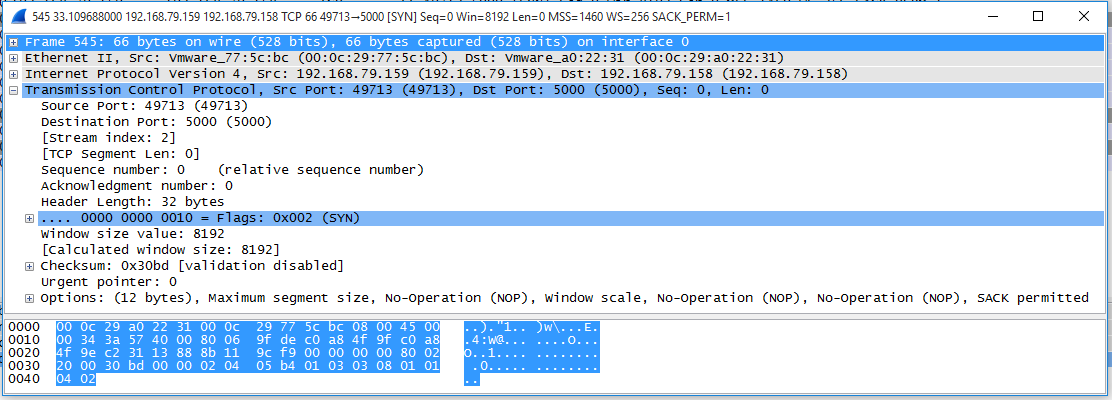
**Étape 5 :** On sauvegarde les données de l’image en format *.jpeg* :

**Étape 6 :** On peut maintenant visualiser l’image en l’ouvrant avec un logiciel qui permet de l’afficher, comme *Paint* :

**7)** Suite à notre analyse, nous constatons qu’il est très facile d’intercepter et de récupérer les données et informations transmises entre le serveur et le client. La sécurité de la connexion est donc plutôt faible.

# **Analyse d’une application client-serveur “secrète”**

## **Mode secret 1**

**1)**

C’est le protocole TCP qui est utilisé pour la couche de transport. Ce protocole est un SYN (pour synchronisation).

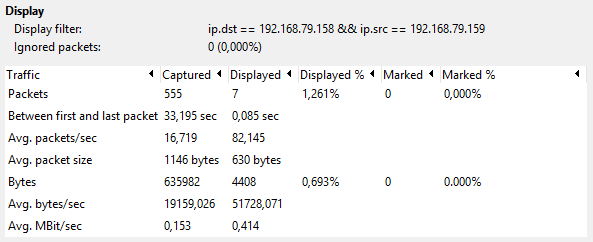
**2)**

Port source : 49713

Port destination : 5000

**3)**

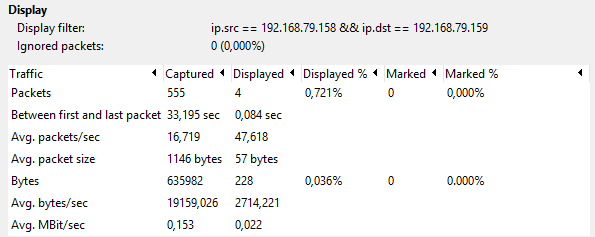
**Client => Serveur**

Filtre appliqué : ip.src == 192.168.79.159 && ip.dst == 192.168.79.158

Il y a **7** paquets envoyés du client vers le serveur;

Il y a **4408** octets envoyés du client vers le serveur;

**Serveur => Client**

Filtre appliqué : ip.src == 192.168.79.158 && ip.dst == 192.168.79.159

Il y a **4** paquets envoyés du serveur vers le client;

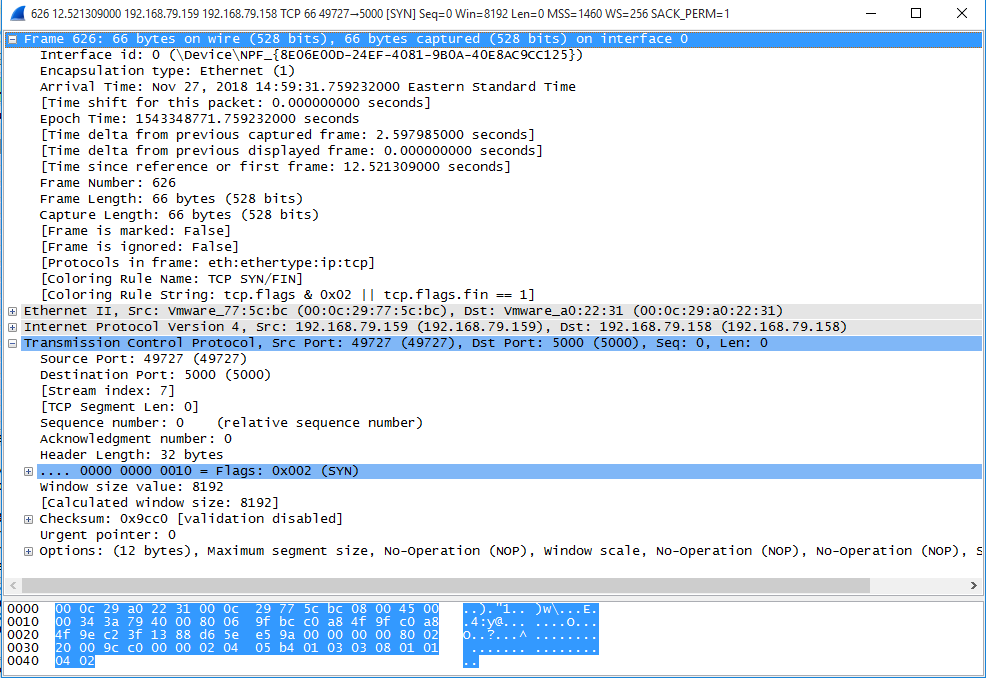
Il y a **228** octets envoyés du serveur vers le client.

**4)**

On observe 1 seule itération, divisée en 3 paquets envoyés simultanément.

Cela signifie donc que le client pour ce mode envoie toute l’information au serveur d’un bloc.

## **Mode secret 2**

**1)**

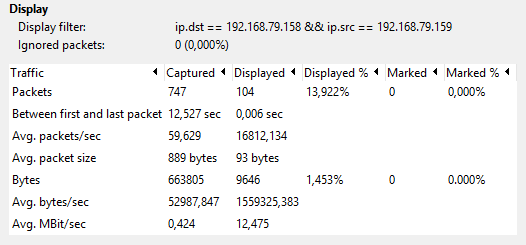
C’est à nouveau le protocole TCP qui est utilisé pour la couche de transport. Ce protocole est aussi un SYN.

**2)**

Port source : 49727

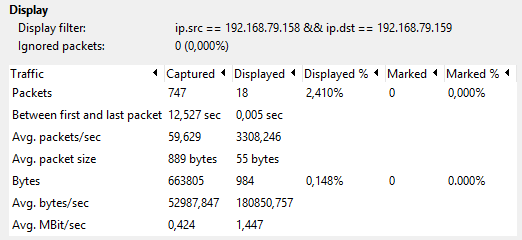
Port destination : 5000

**3)**

**Client => Serveur**

Il y a **104** paquets envoyés du client vers le serveur;

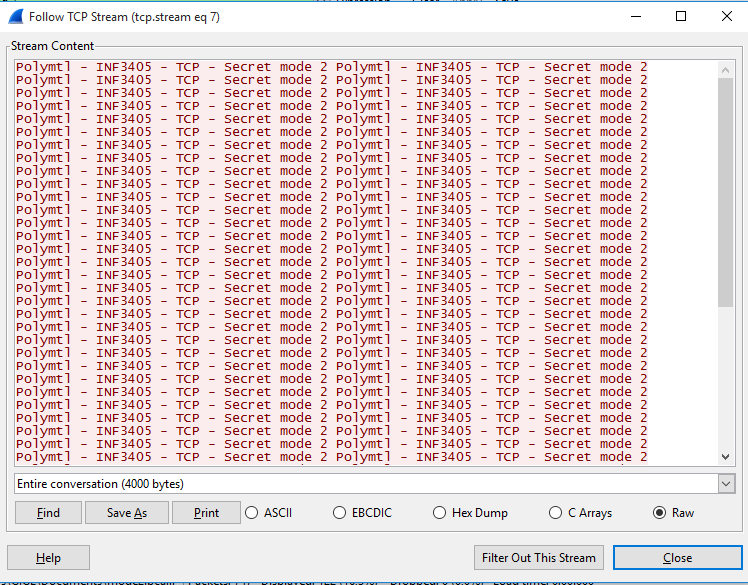
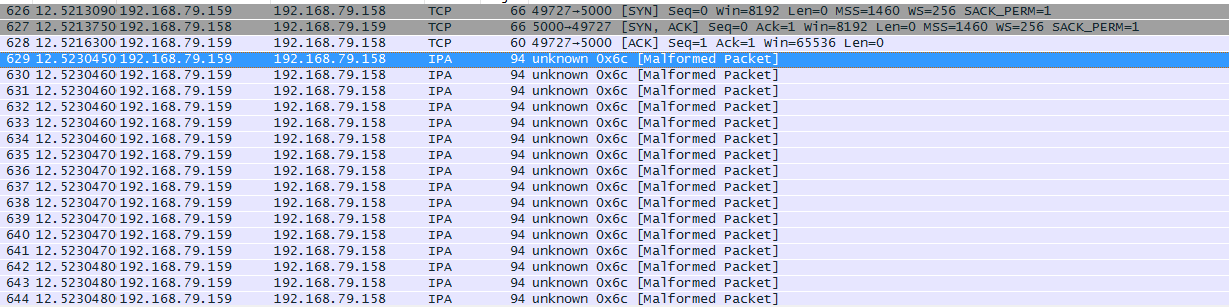
Il y a **9646** octets envoyés du client vers le serveur.

**Serveur => Client**

Il y a **18** paquets envoyés du serveur vers le client;

Il y a **984** octets envoyés du serveur vers le client.

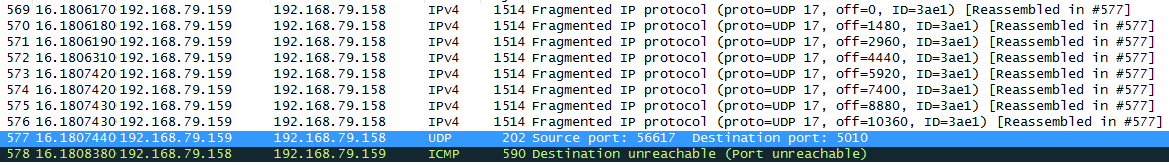
**4)**



Le client envoie au serveur un total de 4000 bytes, dont 40 octets par paquet, donc 100 itérations.

Cela signifie donc que le client envoie les données au serveur par de nombreux petits blocs, au lieu d’un seul grand bloc.

## **Mode secret 3**

**1)**

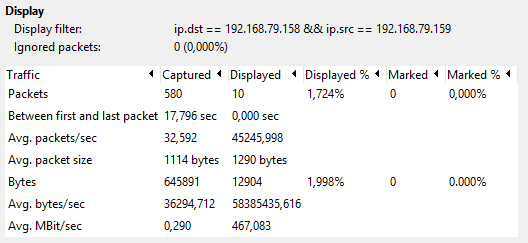
C’est le protocole UDP qui est utilisé pour la couche de transport. Il n’y a pas d’échange de synchronisation, parce qu’il n’y a pas de synchronisation en UDP.

**2)**

Port source : 56617

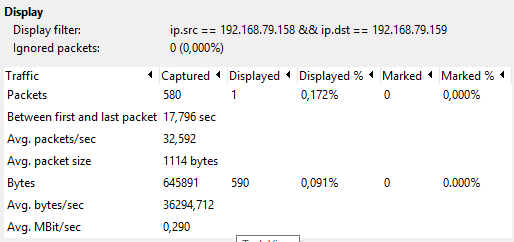
Port destination : 5010

**3)**

**Client => Serveur**

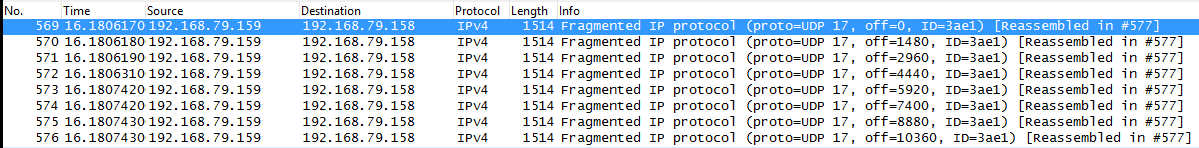
Il y a **10** paquets envoyés du client vers le serveur;

Il y a **12904** octets envoyés du client vers le serveur.

**Serveur => Client**

Il y a **1** paquets envoyés du serveur vers le client;

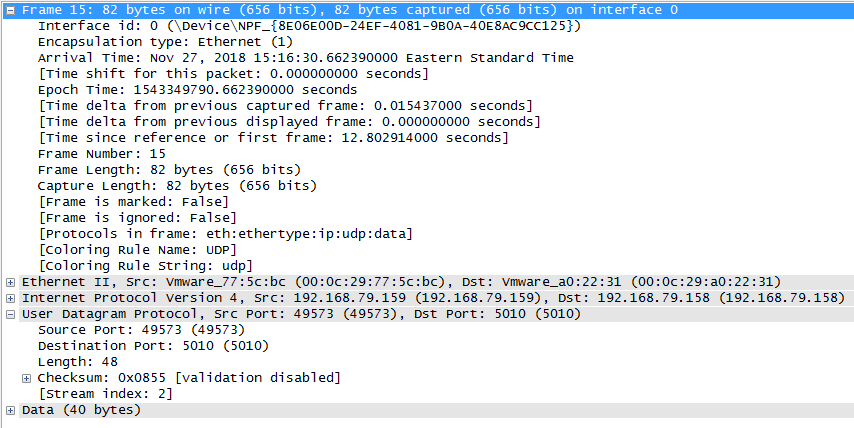
Il y a **590** octets envoyés du serveur vers le client.

**4)**

On observe 1 seule itération, divisée en 8 paquets envoyés simultanément.

Cela signifie donc que le client pour ce mode envoie toute l’information au serveur d’un bloc.

## **Mode secret 4**

**1)**

C’est à nouveau le protocole UDP qui est utilisé pour la couche de transport. Encore une fois, il n’y a pas d’échange de synchronisation, parce qu’il n’y a pas de synchronisation en UDP.

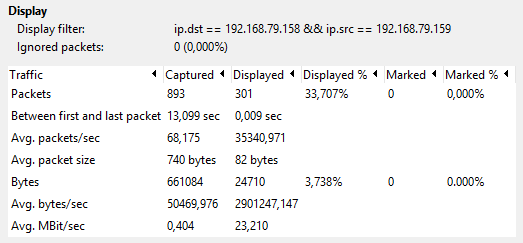
**2)**

Port source : 49573

Port destination : 5010

**3)**

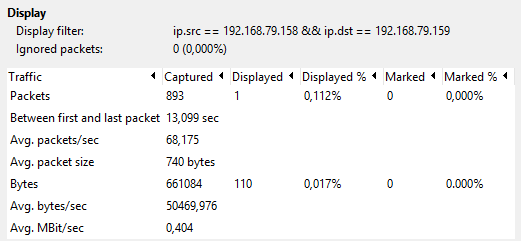
**Client => Serveur**



Il y a **301** paquets envoyés du client vers le serveur;

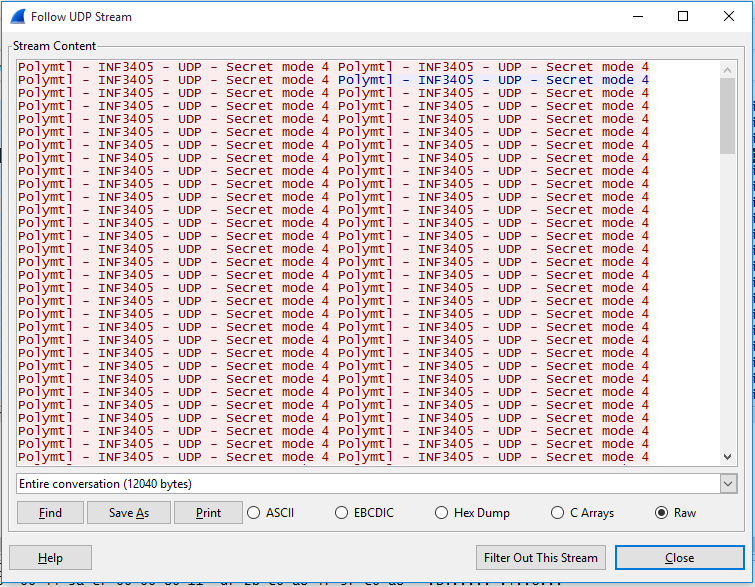
Il y a **24710** octets envoyés du client vers le serveur.

**Serveur => Client**



Il y a **1** paquets envoyés du serveur vers le client;

Il y a **110** octets envoyés du serveur vers le client.

**4)**



Le client envoie au serveur un total de 12040 bytes en 301 itérations, donc 40 octets par paquet.

Cela signifie donc que le client envoie les données au serveur par de nombreux petits blocs, au lieu d’un seul grand bloc.

## **Analyse des performances et protocole TCP**

**1)**

Le premier mode envoie toute l’information d’un bloc, mais comme le message envoyé est plus grand que la limite de taille d’un paquet, le protocole est forcé de fragmenter le bloc en trois divisions.

Le second mode n’a pas ce problème, car il envoie toute l’information en petits blocs séparés. Par contre, chacun de ces petits blocs nécessite son propre en-tête, ce qui en bout de ligne consomme plus de bande passante.

Selon nous, le mode le plus performant est le premier mode, car il consomme beaucoup moins de bande passante en envoyant qu’un seule en-tête pour toutes les données.

**2)**

Tout comme le premier mode, le troisième mode envoie toute l’information d’un bloc, mais comme le message envoyé est plus grand que la limite de taille d’un paquet, le protocole est forcé de fragmenter le bloc en huit divisions.

Le quatrième mode n’a pas ce problème, car il envoie toute l’information en petits blocs séparés. Par contre, chacun de ces petits blocs nécessite son propre en-tête, ce qui en bout de ligne consomme plus de bande passante, comme pour le deuxième mode.

Selon nous, le mode le plus performant est le troisième mode, car il consomme beaucoup moins de bande passante en envoyant qu’un seule en-tête pour toutes les données.

**3)**

Selon nous, les modes les plus performants sont le deuxième et le quatrième mode, car ils envoient les données en de nombreux petits paquets pré-séparés, ce qui ce qui fait que si l’un d’entre eux n’est pas reçu correctement, il est plus facile et moins coûteux de le renvoyer, comparativement à un gros message à renvoyer en entier.

**4)**

L’échange FIN, ACK est la confirmation de terminaison de connexion (ACK) et l’information de fin de connexion de la part du récepteur (FIN) afin de compléter le processus de fin de connexion entre le serveur et le client par un système de « handshaking ».