**Attaques sur TCP**

# Objectifs

L'objectif de ces séances est d'évaluer les mécanismes de session du protocole de transport TCP, de prendre conscience de ces vulnérabilités et les risques de sécurité que cela peut entraîner.

# 1 Introduction

Nous avons vu précédemment que les protocoles ARP et IP comportent des failles pouvant être exploitées et le protocole TCP n'y fait pas exception. Le protocole TCP offre des mécanismes de gestion de session et nous verrons qu'il est possible de les manipuler à des fins malveillantes.

Les attaques sur le protocole TCP, contrairement à celles sur ARP ont l'avantage d'avoir un rayon d'action au-delà des routeurs car elles n'ont pas besoins d'être exécutées sur le même réseau que la victime. La corruption de cache ARP permet de rediriger tous les flux de niveau 2, sans distinction de protocole, qui sont échangés entre deux machines d'un réseau Ethernet. Elles permettent également de cibler un protocole applicatif en particulier sans perturber les autres communications, ce qui a l'avantage de limiter les soupçons.

La suite de ce travail pratique est composée d'un rappel du protocole TCP, de la présentation d'une attaque permettant de terminer prématurément une connexion ainsi que des attaques permettant de faire un vol de session.

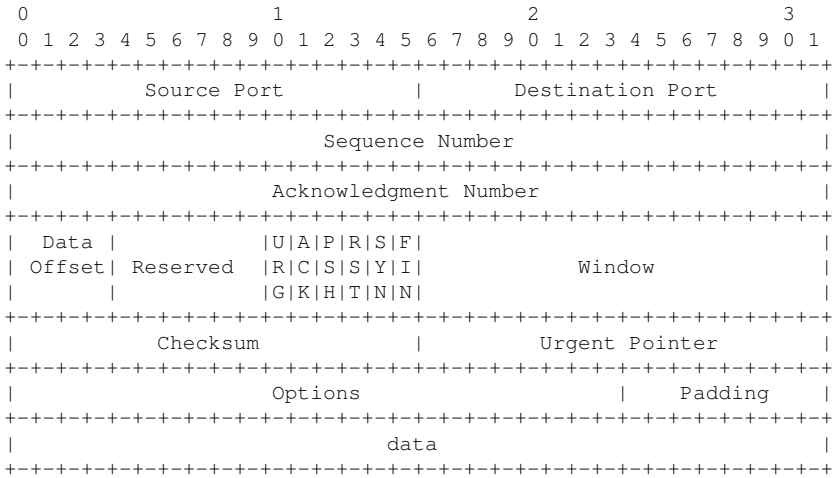
# 2 Rappel TCP

Le protocole TCP est un protocole de transport fiable, en mode connecté, documenté dans la RFC 7931 de l'IETF.

Une session TCP fonctionne en trois phases :

1. l'établissement de la connexion, figure 1 ;
2. les transferts de données, figure 2 ;
3. la fin de la connexion, figure 3.

## 2.1 Entête TCP




**Port source (16 bits)** Numéro du port source.

**Port destination (16 bits)** Numéro du port destination.

**Numéro de séquence (32 bits)** Numéro de séquence du premier octet de ce segment.

**Numéro d'acquittement (32 bits)** Numéro de séquence du prochain octet attendu.

**Taille de l'en-tête (4 bits)** Longueur de l'en-tête en mots de 32 bits (les options font partie de l'en-tête).

**(6 bits)** Champ réservé pour des besoins futurs, il est à 0.

**Drapeaux (12 bits)**

Réservé : réservé pour un usage futur (ECN/NS, CWR, ECE, ...)
URG : signale la présence de données urgentes
ACK : signale que le paquet est un accusé de réception (acknowledgement)
PSH : données à envoyer tout de suite (push)
RST : rupture anormale de la connexion (reset)
SYN : demande de synchronisation (SYN) ou établissement de connexion
FIN : demande la FIN de la connexion

**Fenêtre (16 bits)** Taille de fenêtre demandée, c'est-à-dire le nombre d'octets que le récepteur souhaite recevoir sans accusé de réception.

**Somme de contrôle (16 bits)** Somme de contrôle calculée sur l'ensemble de l'en-tête TCP et des données, mais aussi sur un pseudo en-tête (extrait de l'en-tête IP).

**Pointeur de données urgentes (16 bits)** Position relative des dernières données urgentes, ce champ est activé que lorsque le drapeau URG est à 1.

**Options (8 bits)** Facultatives.

**Remplissage (0 à 7 bits)** Zéros ajoutés pour aligner les champs suivants du paquet sur 32 bits, si nécessaire.

**Données** Données transmises par l'application (login, mot de passe, fichier, page web, etc)

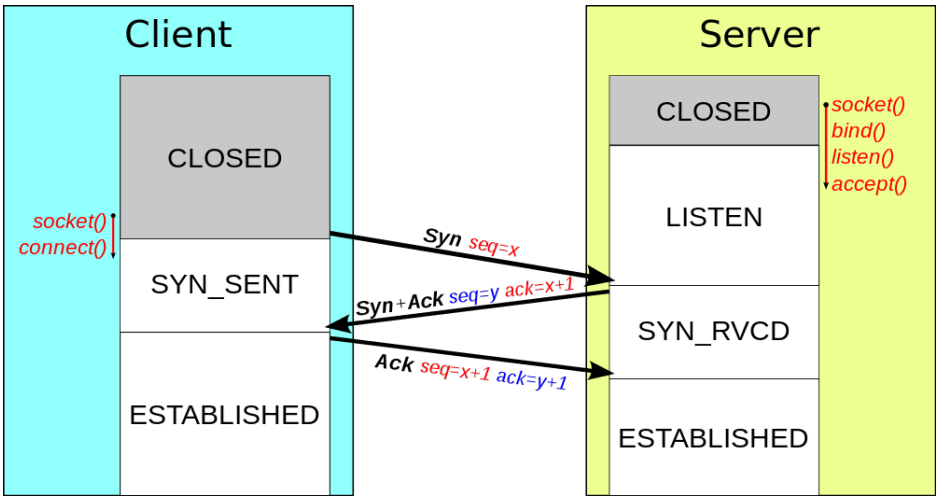



FIGURE 1 – Établissement d'une connexion TCP
**Source :** Wikimedia Commons, Sébastien Koechlin, 6 October 2011

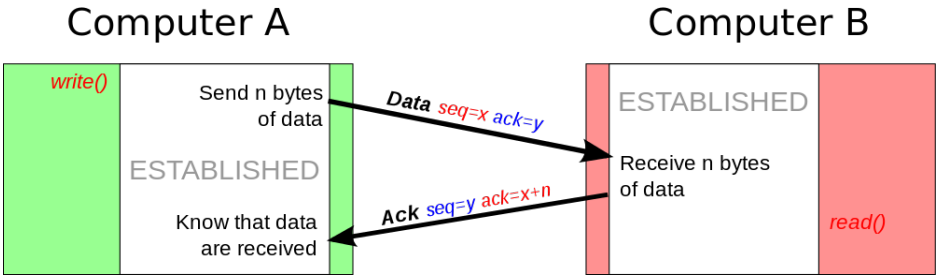



FIGURE 2 – Dialogue TCP entre deux interlocuteurs
**Source :** Wikimedia Commons, Sébastien Koechlin, 6 October 2011

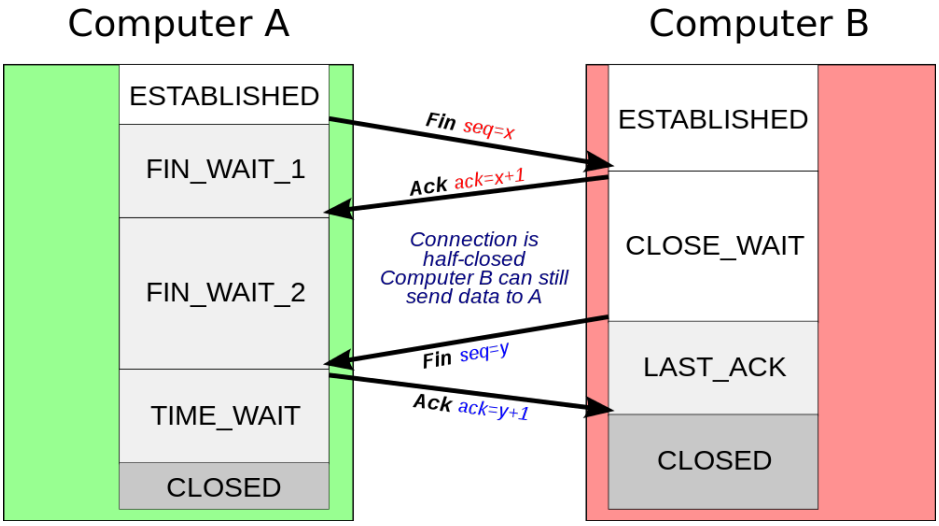



FIGURE 3 – Fermeture de session TCP
**Source :** Wikimedia Commons, Sébastien Koechlin, 6 October 2011

# 3 TCP et Scapy

**Préparation :** Démarrez deux machines et vérifiez qu'elles peuvent communiquer.

**Exercice 1**

1. *Sur 1 des 2 machines, manipulez TCP avec Scapy afin d'établir une connexion, échanger des données et terminer proprement une connexion tel qu'illustré des les figures 1, 2 et 3. Écrivez un fichier exo1.py. Pour utiliser Scapy dans un programme python écrivez au début du programme :* `from scapy.all import *`.

   *Afin de programmer la séquence de* hanshake *du client, démarrez sur la machine qui fait office de serveur une socket en écoute avec la commande* `netcat -l port`. *L'option* `-l` *démarre une socket en mode écoute/serveur tandis que* `port` *indique le port sur laquelle la socket va écouter.*

   *De plus vous aurez besoin d'utiliser la fonction Scapy* `rep = sr1(paquet)`, *cette dernière envoie un paquet comme la fonction* `send` *et attend une réponse (dans l'exemple la réponse est placée dans* `rep`*).*
2. *Vous allez observer un comportement imprévu. Selon vous, quel en est la cause et que peut-on faire pour éviter ça ?*
3. *Une fois le problème résolu ressayez d'établir la connexion.*

# 4 RST Hijacking

Le principe de RST Hijacking est très simple, il consiste en l'injection d'un paquet RST à la victime en se faisant passer pour le serveur. Ainsi la victime croit que le serveur demande de réinitialiser la connexion TCP et la victime va réagir en conséquence. Cependant, la connexion est toujours active du côté serveur puisque la victime n'envoie pas d'acquittement, on peut donc se demander s'il serait possible de profiter de cette situation...

```
A -------- Flag PSH/ACK, seq=1000, ack=1200, DATA=salut --------> B
A <------- Flag ACK, seq=1200, ack=1006 --------- B
P -------- Flag RST, seq=1006, ack=1200 --------> B
```

Pour mettre en oeuvre cette attaque, il est primordial de connaître l'environnement dans lequel l'attaquant se trouve. S'il est connecté au réseau à l'aide d'un concentrateur au même titre que la victime et le serveur, il faut simplement écouter un échange entre les deux à l'aide de Wireshark ou Tcpdump. À l'aide d'un seul paquet, on est capable de connaître les numéros de séquence des deux côtés. La connaissance de ces nombres nous permet d'envoyer un paquet RST à la victime. La victime recevant un paquet RST, elle réinitialise la connexion et incrémente le numéro ACK et tous paquets que la victime enverra au serveur seront ignorés si l'attaquant envoie un paquet au serveur avant la victime.

Dans un réseau commuté, cela se complique un peu mais il est possible d'effectuer une attaque de l'homme du milieu à l'aide d'un ARP Spoofing pour s'aider. Ainsi, on peut procéder à l'écoute passive et capturer les paquets contenant l'information nécessaire. Sinon, lorsque l'attaquant ne se trouve pas sur le même réseau que la victime et le serveur, l'attaquant devra tenter de deviner les numéros de séquences, ce qui jadis était plutôt simple.

**Exercice 2**

*Écrivez un script pour tenter de deviner, à partir de l'attaquant, le prochain numéro de séquence afin d'envoyer un RST à la victime. Que déduisez-vous ?*

**Exercice 3**

1. *Écrivez un programme simple Scapy/Python afin d'automatiser l'attaque du MiTM. Si vous souhaitez passer en argument les adresses IP des machines à empoisonner faites comme dans l'exemple suivant :*

```
import sys

print 'Nombre d'arguments:', len(sys.argv), 'arguments.'
print 'List des arguments:', str(sys.argv)
```

2. *À l'aide de Scapy, tentez de prédire les numéros de séquence de différentes connexions entre 2 machines. Quels sont vos observations ?*

**Infos :** Pour cet exercice vous aurez sûrement besoin option de la fonction `sniff` :

```
sniff(filter="", count=0, prn=None, lfilter=None, timeout=None, iface=All)
```

— `count` : nombre de paquets à capturer. 0 : pas de limite.
— `timeout` : stoppe le sniff après un temps donné.
— `iface` : désigne l'interface sur laquelle sniffer.
— `filter` : filtre les paquets à garder d'après une chaîne de caractère.
Exemple : `filter="host 192.168.1.1 and port 80"` filtre les paquets ayant un lien avec 192.168.1.1 et le port 80.
— `lfilter` : même chose, mais utilise une fonction plutôt qu'une chaîne.
Exemple : `lfilter=lambda x: x.haslayer(TCP)` récupère que les paquets TCP.
— `prn` :fonction à appliquer à chaque paquet. Si la fonction retourne quelque chose, cela s'affiche. Exemple : `prn = lambda x: x.show()` va afficher le détail de chaque paquet.

**Exercice 4**
*Établissez des échanges avec* `netcat` *ou* `ssh` *entre la victime et le serveur. À partir de la machine de l'attaquant, faites-en sorte d'empêcher la victime de communiquer avec le serveur.*

# 5 TCP Session Hijacking

La corruption de cache ARP permet de rediriger tous les flux de niveau 2, sans distinction de protocole, qui sont échangés entre deux machines d'un réseau Ethernet.

Or une multitude de connexions de niveaux supérieurs peuvent être multiplexées dans un flux de niveau 2. Dans le cas de l'empoisonnement d'une machine qui est amenée à établir de nouvelles connexions en permanence, l'attaquant se retrouve avec nombre important de connexions à gérer, dont la plupart n'offrent aucun intérêt à ses yeux.

En effet, dans la plupart des cas, l'attaquant s'intéresse à un type de connexion précis. Par exemple, supposons qu'une machine offre les services FTP et Web. Pour une raison précise, l'attaquant peut être amené à vouloir rediriger les connexions FTP d'une machine vers ce serveur, tandis que les connexions HTTP n'auront aucun intérêt pour lui.

Par conséquent, nous avons besoin d'une nouvelle technique, qui nous permettra de cibler précisément la connexion à détourner. Il n'est pas question ici de rediriger une connexion au moment où elle est initialisée. En effet, nous ne travaillons plus avec des adresses physiques mais avec des adresses logiques (adresses IP), et l'usurpation d'identité logique ne paraît pas envisageable.

En revanche, le détournement d'une connexion déjà établie paraît réalisable. Si nous parvenons à rassembler tous les éléments qui identifient une connexion aux yeux des deux interlocuteurs que nous essayons de berner, nous pourrons alors influer sur cette connexion, et contrôler les données échangées.

NB : Désynchroniser A provoquera ce que l'on appelle un ACK storm, une tempête d'ACK entre A et B.

```
A --------- Flag PSH/ACK, seq=1000, ack=1200, DATA=Hey ---------> B
A <-------          Flag ACK, seq=1200, ack=1004        --------- B
P ------- Flag PSH/ACK, seq=1004, ack=1200, DATA=Mouhaha -------> B
P <-------          Flag ACK, seq=1200, ack=1012        --------- B
```

```
--------------------------- ACK Storm -----------------------------

A ----------------------       ACK       ----------------------> B
A <----------------------       ACK       ---------------------- B
A ----------------------       ACK       ----------------------> B
A <----------------------       ACK       ---------------------- B
```

**Exercice 5**
*Mettez en oeuvre cette attaque pour détourner une session `Telnet` et envoyer des commandes au serveur. Peut-on réaliser ce même scénario avec `ssh` ? Pourquoi ?*

**Infos :** Vous pouvez retrouver les commandes dans le manuel de Telnet.

**Exercice 6**
*Combinez l'attaque ARP Spoofing et TCP Highjacking pour prendre le contrôle complet d'une session entre le client et le serveur tout en passant presque inaperçu sur un réseau commuté.*

### 5.1 "A Simple Active Attack Against TCP" by Laurent Joncheray

Il se peut que la victime et le serveur soient bien protégés contre l'attaque de ARP Spoofing. Pour ce cas-là, nous allons utiliser une autre faille du le protocole TCP. Laurent Joncheray a publié une attaque consistant à désynchroniser la communication entre la victime et le serveur tout en agissant à titre d'intermédiaire ou de traducteur, puisque les deux hôtes ne se comprennent plus.

Pour réaliser cette attaque, il faut être en mesure de faire de l'écoute passive entre la victime et le serveur, donc sur un des deux segments.

**Exercice 7**
*Reconstituez une conversation entre la victime et le serveur seulement avec l'écoute passive.*

**Exercice 8**
*Réalisez l'attaque complète, si vous êtes branchés avec un commutateur, vous devez avoir recours à ARP Spoofing.*

## 6 Pour aller un peu plus loin...

### 6.1 Attaque de Kevin Mitnick

— `https://fr.wikipedia.org/wiki/Attaque_de_Mitnick`

### 6.2 TCP Blind Hijacking

— `http://phrack.org/issues/64/13.html`
— `http://conference.hitb.org/hitbsecconf2009kl/materials/D1T2%20-%20Alex%20Kuza55%20K%20-%20Implementing%20and%20Improving%20Blind%20TCP.pdf`