SAE-204 SCAPY

Étape 2 - Exemple du ping v6

Cette étape va nous permettre de nous familiariser avec Scapy et à l'écriture d'un script Python, à travers l'étude d'un simple ping en IPv6 entre deux machines du même réseau local.

1. Voici la commande Wireshark pour ne filtrer que les trames contenant l'échange initial du protocole "Neighbor Discovery" et les pings echo/request : `icmpv6.type == 135 || icmpv6.type == 136 || icmpv6.type == 128 || icmpv6.type == 129`

2. **Adresse Ipv6 qui initie le ping** : 64:00:6a:6a:c4:01
   Adresse Ipv6 de la machine cible : 00:13:1a:a3:e1:18

3. Voici les différents champs de la première trame : "Neighbor Solicitation" : | **Champ** | **Valeur** | | ---------- | ---------- | | **En-tête Ethernet** | Présent | | **Adresse MAC Source** | `64:00:6a:6a:c4:01` | | **Adresse MAC Destination** | `33:33:ff:00:00:01` | | **En-tête IPv6** | Présent | | **Adresse IP Source** | `f2001:660:6701:30cc:84fc:c335:133c:f204` | | **Adresse IP Destination** | `ff02::1:ff00:1` | | **En-tête du paquet ICMP/ND** | Neighbor Solicitation (type 135) ou Advertisement (136) | | **Adresse IP Cible** | `2001:660:6701:30cc::1` | | **Adresse MAC** (du ND option) | `64:00:6a:6a:c4:01` |

4. Programme permettant de récuperer que la trame qui contient ce message dans le fichier déja filtré :

In [ ]:
```python
from scapy.all import *


trames=rdpcap("filtered.pcapng")
for trame in trames:
    if(trame[IPv6].version)==6:  # type: ignore
        if (trame[IPv6].nh)==58:  # type: ignore
            if (trame[2].type)==135:
                print(f"Ethernet: MAC Source : {trame['Ether'].src}")
                print(f"Ethernet: MAC Destination : {trame['Ether'].dst}")
                print(f"IPv6 : IP Source : {trame[IPv6].src}")
                print(f"IPv6 : IP Destination : {trame[IPv6].dst}")
                print(f"ICMPv6 : IP Target : {trame[ICMPv6ND_NS].tgt}")
                print(f"ICMPv6 : MAC Requested : {trame[ICMPv6ND_NS].lladdr}")
```

6. Programme permettant de récuperer la trame qui contient ce message dans le fichier non filtré (gestion des erreurs) :

In [ ]:
```python
from scapy.all import *


trames=rdpcap("ping6-total.pcapng")
for trame in trames:
    if trame.haslayer(IPv6):
        if(trame[IPv6].version)==6:  # type: ignore
            if (trame[IPv6].nh)==58:  # type: ignore
                if (trame[2].type)==135:
                    print(f"Ethernet: MAC Source : {trame['Ether'].src}")
                    print(f"Ethernet: MAC Destination : {trame['Ether'].dst}")
                    print(f"IPv6 : IP Source : {trame[IPv6].src}")
                    print(f"IPv6 : IP Destination : {trame[IPv6].dst}")
                    print(f"ICMPv6 : IP Target : {trame[ICMPv6ND_NS].tgt}")
                    print(f"ICMPv6 : MAC Requested : {trame[ICMPv6ND_NS].lladdr}")
```

7. Commenter au mieux la sortie de programme ci-dessous, afin de la rapprocher avec les concepts vus en TD (adresses sources et adresses destinations aux différents niveaux, IP Target et MAC Requested).

In [ ]:
```python
from scapy.all import *
ICMPv6_types={ 128 : 'Echo-Request', 129 : 'Echo-Reply', 135 : 'NeighborSolicitation', 136 : 'Neighbor Advertis
              133 : 'Router Solicitaion', 134 : 'Router Advertisement'}

def print_icmpv6 (trame) :
    print(trame.summary())
    type=trame[2].type
    if (type==135 or type==136):
        print(f"TYPE PACKET ICMP : {ICMPv6_types[type]}")
```

```
        print(f"Ethernet: MAC Source : {trame[0].src}")
        print(f"Ethernet: MAC Destination : {trame[0].dst}")
        print(f"IPv6 : IP Source : {trame[1].src}")
        print(f"IPv6 : IP Destination : {trame[1].dst}")
        print(f"ICMPv6 : IP Target : {trame[2].tgt}")
        print(f"ICMPv6 : MAC Requested : {trame[3].lladdr}")
        print ("\n")
    else:
        print(f"TYPE PACKET ICMP : {ICMPv6_types[type]}")
        print(f"Ethernet: MAC Source : {trame[0].src}")
        print(f"Ethernet: MAC Destination : {trame[0].dst}")
        print(f"IPv6 : IP Source : {trame[1].src}")
        print(f"IPv6 : IP Destination : {trame[1].dst}")
        print ("\n")

carte=conf.iface
print(f"On commence le 'sniffing' sur la carte {carte}:")
print("\n")
sniff(filter="ip6 proto 58", prn=print_icmpv6, store=0, iface=carte, count=6)
```

Le programme ci-dessus sert à sniffer les trames qui sortent et entrent de notre carte réseau, puis d'en extraire les paquets ICMPv6 pour ensuite afficher les informations importantes.

Quand on éxecute ce programme, il nous affiche uniquement les paquets IPv6 qui contiennent **ICMPV6**. Lors d'un Ping, quand le cache est vide, notre machine va émettre un message **ICMP NS** avec comme champs : l'adresse IPv6 source de l'émetteur, l'adresse IPv6 cible, et l'adresse MAC source dans l'option "Source Link-Layer Address". Quand ce message est recu par la machine cible, il émet un message **ICMP NA** avec comme champs : l'adresse IPv6 source de la machine cible, l'adresse IPv6 cible (celle de l'émetteur du NS), et son adresse MAC dans l'option "Target Link-Layer Address.

Puis un Ping "lambda" s'execute.

8. Créer un programme équivalent permettant de capturer et aficher les champs essentiels d'un ping en v4, incluant l'échange Req/Rep ARP.

In [ ]:
```
from scapy.all import *

def printlol(trame):

   # Test IPv4
    if trame[0].type == 0x0800:  # Ethernet type IPv4
        if trame[1].proto == 1:  # Protocole ICMP
            icmp = trame[ICMP]
            if icmp.type == 8:  # Echo Request (ping envoyé)
                print(f"Envoi de ping vers {trame[IP].dst}")
                print(f"  IP source : {trame[IP].src}")
                print(f"  IP destination : {trame[IP].dst}")
            elif icmp.type == 0:  # Echo Reply (ping reçu)
                print(f"Réception de ping de {trame[IP].src}")
                print(f"  IP source : {trame[IP].src}")
                print(f"  IP destination : {trame[IP].dst}")



    #Test ARP
    if trame[0].type == 0x0806:
        if trame[0].src == get_if_hwaddr(conf.iface):
            print("  Envoie d'une requète ARP")
            print(f"  IP source : {trame[ARP].psrc}")
            print(f"  MAC source : {trame[ARP].hwsrc}")
            print(f"  IP cible  : {trame[ARP].pdst}")
            print(f"  MAC cible : {trame[ARP].hwdst}")
        else:
            print("  Reçu un paquet ARP :")
            print(f"  IP source : {trame[ARP].psrc}")
            print(f"  MAC source : {trame[ARP].hwsrc}")
            print(f"  IP cible  : {trame[ARP].pdst}")
            print(f"  MAC cible : {trame[ARP].hwdst}")

sniff(prn=printlol, store=0, iface="Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC", count=100)
```

Lors de l'éxecution de ce programme, si la trame contient un paquet ARP, il affiche toutes les informations, sinon, si le paquet IPv4 contient un paquet ICMP de avec Echo Request et Echo Reply, il affiche les informations.

Étape 4 – Challenge FTP – Analyse d'une capture Wireshark d'un transfert FTP

2.0.1) Pour filtrer les trames correspondant à cet échange, il convient d'afficher le handshake TCP ainsi que les communications FTP, qui utilisent les ports 21 (commandes) et 20 (données). Voici donc le filtre qui convient : `((tcp.flags.syn == 1 or tcp.flags.ack == 1) and tcp.port == 21) or ftp-data`

2.0.2) Comme dit précedemment, FTP utilise **deux différents port** pour séparer les commandes et les données. **Le canal de commande**, lui, est **ouvert pendant toute la session** contrairement au **canal de donnée**, qui lui, n'est **ouvert uniquement pour le transfers de données**. En regardant les échanges applicatifs, nous observons plusieurs commandes ainsi que des codes... Voici un tableau résumé de leur role :

Commande	Fonction	Explication
`USER`	Envoi du nom d'utilisateur	Identifie le client auprès du serveur FTP.
`PASS`	Envoi du mot de passe	Authentifie l'utilisateur après la commande `USER`.
`SYST`	Demande d'information sur le système du serveur	Permet au client de savoir si le serveur est sous Unix, Windows, etc.
`PORT`	Indique au serveur où envoyer les données (mode actif)	Le client fournit son IP et un port pour établir la connexion de données.
`LIST`	Demande la liste des fichiers du répertoire courant	Utilise le canal de données pour transférer la liste des fichiers au client.
`TYPE I`	Définit le mode binaire pour le transfert de fichiers.	Utilisé pour transférer des fichiers non-textes comme des images ou des vidéos.
`RETR`	Permet de télécharger un fichier depuis le serveur.	Utilisée pour récupérer un fichier spécifique sur le serveur FTP.

Code	Signification	Utilisation
`220`	Service prêt	Le serveur est prêt à recevoir une connexion.
`331`	Nom d'utilisateur OK, mot de passe requis	Après `USER`, demande `PASS`.
`230`	Connexion réussie	Authentification terminée.
`215`	Type de système	Réponse à `SYST` (ex. : UNIX).
`150`	Ouverture de la connexion de données	Avant un transfert (ex. `LIST`, `RETR`).
`226`	Transfert terminé	Données envoyées, fermeture du canal de données.
`221`	Fin de session	Le serveur ferme la connexion de commande.

Pourquoi existe-t-il des commandes et des codes dans FTP ? Quelle est la différence ?

La réponse est simple : **Les commandes** sont envoyées par le **client** au **serveur** pour lui demander d'effectuer des actions spécifiques, et **les codes**, en revanche, sont les **réponses du serveur** qui indiquent l'état ou le résultat de l'exécution de la commande.

Analyse des commandes et réponses dans la capture :

Nous avons analysé plusieurs commandes FTP dans l'échange, telles que :

- **`USER`** : "touriste"
- **`PASS`** : "3aboqphie=3qbc!"
- **`SYST`** : "UNIX Type: L8"
- **`PORT`** : "10,2,4,3,195,193"
- **`LIST`** : "-rw------- 1 1002 1002 556459 Mar 21 11:16 ftpdoc.odt"
- **`TYPE I`** : "Switching to binary mode"
- **`RETR`** : "le fichier en binaire"

Ce que révèle cette analyse :

Grâce à Wireshark, nous pouvons voir que **FTP n'utilise pas de chiffrement**, ce qui signifie que **toutes les informations sont envoyées en clair**. Cela inclut des données sensibles comme les identifiants (nom d'utilisateur, mot de passe) ainsi que les fichiers transférés. Le danger n'est pas immédiat dans ce cas (car nous capturons uniquement notre propre trafic), mais une **attaque de type Man-in-the-Middle (MITM)** pourrait permettre à un attaquant : **D'intercepter** les trames échangées et **d'analyser** le contenu des informations sensibles, telles que les identifiants ou les fichiers en transfert.

2.0.3) En utilisant le suivi de flux TCP de Wireshark, nous avons pu reconstituer le fichier récuperer par le client. Nous parlerons de ce qu'il contient plus tard...

2.0.4) Dans cette partie du challenge FTP, nous devions concevoir un programme capable de capturer en temps réel le trafic réseau pour extraire les identifiants USER, PASS, ainsi que les fichiers transférés. Pour tester notre outil, nous avons utilisé un script Python se connectant à un serveur FTP public en mode passif, où le client se connecte à un port défini par le serveur pour recevoir les données.

Voici le script "test-ftp" qui permet une connexion FTP:

⚠ *Veuillez d'abord exécuter la commande ci-dessous pour installer Scapy sur Jupyter.*

In [ ]:
```
pip install scapy
```

In [ ]:
```
from ftplib import FTP

def ftp_download():
    try:
        # Connexion au serveur FTP de test
        ftp = FTP('test.rebex.net')
        ftp.login(user='demo', passwd='password')
        print("[+] Connecté au serveur FTP.")

        # Nom du fichier à télécharger
        filename = 'readme.txt'

        # Téléchargement
        with open(filename, 'wb') as f:
            ftp.retrbinary(f'RETR {filename}', f.write)
        print(f"[+] Fichier '{filename}' téléchargé avec succès.")
        with open(filename, 'wb') as f:
            ftp.retrbinary(f'RETR {filename}', f.write)
        print(f"[+] Fichier '{filename}' téléchargé avec succès.")

        # Fermeture de la connexion
        ftp.quit()
        print("[+] Connexion FTP fermée.")

    except Exception as e:
        print("[-] Erreur lors du téléchargement :", e)

if __name__ == "__main__":
    ftp_download()
```

Le script ci-dessous **permet de détecter en temps réel les échanges FTP** en analysant chaque requête. Dans un premier temps, il attend qu'une connexion FTP se présente, puis il identifie le nom d'utilisateur ainsi que le mot de passe. Ensuite, il récupère les données des fichiers transférés, puis les recrée localement sur notre machine.

Étant donné qu'une **connexion FTP peut être en mode actif ou passif**, le programme est conçu pour fonctionner dans les deux cas. De plus, il est capable de détecter et d'enregistrer plusieurs fichiers lors d'une même session.

In [ ]:
```
#_____________________________________          _________      .__  _____  _____
#\_   _____/\__    ___/\_______   \          /   _____/ ____ |__|/ ____\/ ____\_____________
# |    __)    |    |    |     ___/  ______  \_____  \ /    \|  \   __\\   __\/ __ \_  __ \
# |     \     |    |    |    |     /_____/  /        \   |  \  ||  |   |  | \  ___/|  | \/
# \___  /     |____|    |____|             /_______  /___|  /__||__|   |__|  \___  >__|
#     \/                                         \/     \/                      \/
# Mode Automatique | By Emilien

from scapy.all import *

passive_raw_data = b""
active_raw_data = b""
stop_sniffing = False
file = False
name_list = []
t = 0
raw_table = []
passive_port = -2
ftp_connection = False


def sniff_function(trame):
        global passive_port, passive_raw_data, active_raw_data, stop_sniffing, file, name_list, t, raw_table, f

        #Test de connection FTP
```

```
        if trame.haslayer('TCP') and trame.haslayer('Raw'):
                if "220" in trame['Raw'].load.decode(errors="ignore") and trame['TCP'].sport == 21:
                        ftp_connection = True
                        print("\n[-] Connexion FTP en cours...\n")


        if trame.haslayer('TCP') and trame.haslayer('Raw') and ftp_connection == True:
                if "USER" in trame['Raw'].load.decode(errors='ignore'):
                        user = trame['Raw'].load.decode(errors='ignore')
                        user = user.split(' ')
                        print(f"[+] Utilisateur trouvé: {user[1]}")


                if "PASS" in trame['Raw'].load.decode(errors='ignore'):
                        password = trame['Raw'].load.decode(errors='ignore')
                        password = password.split(' ')
                        print(f"[+] Mot de passe trouvé: {password[1]}")


                # --------------    Mode Passif TEST -------------#
                if "227 Entering Passive Mode" in trame['Raw'].load.decode(errors='ignore'):
                        selected_trame = trame['Raw'].load.decode(errors='ignore')

                        match = re.search(r'\((.*?)\)', selected_trame)
                        content = match.group(1)
                        split_match = content.split(',')
                        passive_port = (int(split_match[4]) * 256) + int(split_match[5])


                if "RETR" in trame['Raw'].load.decode(errors="ignore"):
                        name = trame['Raw'].load.decode(errors="ignore").split(' ')[1]

                        if name.strip() in name_list:
                               file = False
                        else:
                                file = True
                                name_list.append(name.strip())


                if trame['TCP'].sport == passive_port and file == True:
                        passive_raw_data += trame['TCP'].load

                        if t < len(raw_table) and raw_table[t]:
                                raw_table[t] = passive_raw_data
                        else:
                                raw_table.append(passive_raw_data)


                if "226 Transfer complete." in trame['Raw']:
                        t += 1


                # --------------  Mode Actif TEST ----------------#
                if trame['TCP'].sport == 20 and file == True:
                        active_raw_data += trame['TCP'].load

                        if t < len(raw_table) and raw_table[t]:
                                raw_table[t] = active_raw_data

                        else:
                                raw_table.append(active_raw_data)


                if "QUIT" in trame['Raw'].load.decode(errors="ignore"):
                        stop_sniffing = True



def stop_filter_function(pkt):
    return stop_sniffing


def create_files():
        nb = len(name_list)
        print(f"[✓] Il y a {nb} fichier(s) qui a(ont) été detecté(s): ", " ".join(name_list))
        for file in raw_table:
                if (file):
                        index = raw_table.index(file)
                        file_name = name_list[index]
                        file_name = file_name
                        with open(file_name, "wb") as f:
                                f.write(file)
```

```
                                print("[✓] Fichier créé sous le nom de", file_name)



# Principal
if __name__ == "__main__":

        print("[Sniffer] Initialisation... Trafic en écoute sur l'interface réseau.")
        sniff(filter="", prn=sniff_function, store=0, iface="Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC", s
        create_files()
```

2.0.5) Pour cette partie du Challenge FTP, nous avons crée un programme qui **permet de récuperer les informations comme le nom d'utilisateur, le mot de passe et le fichier transféré** à partir du fichier pcapng.

Pour cela, dans un premier temps, nous **filtrons les trames réseau** en ne conservant que celles dont les adresses IP correspondent à l'échange entre le client et le serveur FTP. Ces trames sont ensuite ajoutées à une liste appelée "trames_filtered", qui servira de base pour l'extraction des données.

Une fois les trames pertinentes identifiées, nous analysons leur contenu afin de repérer les commandes FTP **USER et PASS**, contenant les identifiants de connexion. Enfin, nous reconstituons les données transférées à partir des trames, et créons un fichier, en l'occurrence un document, car la signature binaire correspond à celle d'un fichier de type .doc. Les informations collectées sont ensuite affichées à l'utilisateur.

⚠ *Pour executer ce code, il faut impérativement être dans le même dossier que la capture WireShark !*

In [ ]:
```
#_____________________________________          _________      .__  _____  _____
#\_   _____/\__    ___/\______   \          /   _____/ ____ |__|/ ____\/ ____\_____________
# |    __)    |    |    |       _/  ______  \_____  \ /    \|  \   __\\   __\/ __ \_  __ \
# |     \     |    |    |    |    /_____/  /        \   |  \  ||  |   |  | \  ___/|  | \/
# \___  /     |____|    |____|            /_______  /___|  /__||__|   |__|  \___  >__|
#     \/                                        \/     \/                       \/
# Mode manuel | By Emilien

from scapy.all import *

trames=rdpcap("ftp-total.pcapng")
trames_filtered = []
user = ""
password = ""
raw_file = b""

def filtrage():
    global user, password

    for trame in trames:
        if trame.haslayer('IP'):
            if trame['IP'].src in ("10.2.4.3", "10.2.12.16") and trame['IP'].dst in ("10.2.4.3", "10.2.12.16"):
                trames_filtered.append(trame)

                #Récuperer les infos (user & password)
                if trame.haslayer('Raw'):
                    if "USER" in trame['Raw'].load.decode(errors="ignore"):
                        user = trame['Raw'].load.decode(errors="ignore")
                    if "PASS" in trame['Raw'].load.decode(errors="ignore"):
                        password = trame['Raw'].load.decode(errors="ignore")

    return trames_filtered


def create_file():
    with open("ftpdoc.odt", "wb") as f:
        f.write(raw_file)
        print("Fichier crée", "sous le nom de", "ftpdoc.odt !\n")



#Main
if __name__ == "__main__":
    print("""
                ##############################
                #                            #
                #  Recupération des infos...  #
                #                            #
                ##############################
                         by Emilien
                ##############################
                """)
    trames = filtrage()
```

```
   #Reconstitution de la "data"
    for trame in trames:
        if trame.haslayer('Raw'):
            if trame['TCP'].sport == 20:
                raw_data = trame['Raw'].load
                raw_file += raw_data


    #Montrer les infos
    create_file()
    print("Voici les informations:")
    print(f"Utilisateur: {user.strip().split(' ')[1]}")
    print(f"Mot-de-passe: {password.strip().split(' ')[1]}")
```

Après avoir éxecuter le programme, un fichier se crée. Dans ce fichier, nous voyons un message à dechiffrer. Pour cela nous avons crée un programme Python permettant de déchiffrer ce message à partir des informations donnés.

In [ ]:

```
#_________                  .__       .__
#\_______ \   ____   ____ |__|_____ |  |__   ____________
# |    |  \_/ __ \_/ ___\|  \____ \|  |  \_/ __ \_  __ \
# |    `   \  ___/\  \___|  |  |_> >   Y  \  ___/|  | \/
#/_______  /\___  >\___  >__|   __/|___|  /\___  >__|
#        \/     \/     \/   |__|        \/     \/
#By Emilien

dictionnaire_inverse = {
    1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7: 'G',
    8: 'H', 9: 'I', 10: 'J', 11: 'K', 12: 'L', 13: 'M',
    14: 'N', 15: 'O', 16: 'P', 17: 'Q', 18: 'R', 19: 'S',
    20: 'T', 21: 'U', 22: 'V', 23: 'W', 24: 'X', 25: 'Y', 26: 'Z'
}

def decoder(coder_mot, debut):
    #Initialisation des variables
    index = debut
    lettre_decode = ""
    value_mot = []
    mot_ori = ""

    #Prendre leur code et les transformer en fonction de l'index
    for lettre in coder_mot:
        if lettre == " ":
            value_mot.append(" ")
        else:
            for i in dictionnaire_inverse:
                if lettre == dictionnaire_inverse.get(i):
                    if index != 27:
                        value_mot.append((i - index) % 26)
                        index += 1
                    elif index == 27:
                        index = 1
                        value_mot.append((i - index) % 26)
                        index+=1


    #A partir des entiers, prendre les lettres correspondantes
    for lettre in value_mot:
        if isinstance(lettre, int):
            mot_ori += dictionnaire_inverse.get(lettre)
        else:
            mot_ori += lettre


    return value_mot, mot_ori

#Il y avait un programme qui permettait de voir avoir quel index de base était le chiffrage (for i ...). Il s'a

#Lancement du programme
if __name__ == "__main__":
    reponse = decoder("KNMT QFH JD DWLMVCB AGIGHUI QK WZNWTQE EJY RJKCQAOY Z LF NLVJ GBZJ VL BBGHYSAECOA A CMRF
    print("Voici le tableau d'entiers: \n")
    print("------------------------------ \n")
    print(reponse[0])
    print("\n")

    print("Voici le message décodé: \n")
    print("--------------------------- \n")
    print(reponse[1])
```

Après avoir déchiffrer ce message, on nous dit d'aller à l'adresse http://cesar.bascou.fr. À cette adresse, nous avons un Google Form qui nous informe que si l'on s'inscrit le premier, nous avons une place certaine en spécialité Cybersécurité.

Étape 5 – Challenge Telnet

Avant de s'attaquer au programme, il faut déjà en apprendre en peu plus sur le protocole Telnet. Pour cela j'ai regardé les trames qui contiennent du Telnet. Puis j'ai vu qu'il y avait des codes et des sous-codes qui permettait à l'application de communiquer. Voici ci-dessous les tableaux récapitulatifs.

Code	Nom	Signification
251	WILL	Le client/serveur **veut activer** une option Telnet.
252	WON'T	Il **refuse** d'activer l'option.
253	DO	Il **demande** à l'autre d'activer une option.
254	DON'T	Il demande à **ne pas** activer une option.
255	IAC	Indique le début d'une commande Telnet.

Code	Option Telnet	Description
0	BINARY	Mode binaire
1	ECHO	Écho de caractères
3	SUPPRESS GO AHEAD	Supprimer le "Go Ahead"
24	TERMINAL-TYPE	Type de terminal
31	NAWS	Négociation de la taille de fenêtre

Voici le programme qui permet de récuperer et d'afficher l'identifiant et le mot-de-passe d'une connexion Telnet stockés dans une capture Wireshark

⚠ *Pour executer ce code, il faut impérativement être dans le même dossier que la capture WireShark !*

In [ ]:
```
from scapy.all import *


trames=rdpcap("Etape 5/telnet-total.pcapng")

handshake = False
src_ip = "0.0.0.0"
dst_ip = "0.0.0.0"
login_table = []
login_user = False
login_password = False
user_table = []
pass_table = []
stop_user = False
stop_pass = False


#Détecter une connexion Telnet (avec le handshake TCP sur le port 23)
def init_connection():
    global handshake, src_ip, dst_ip
    for trame in trames:
        if trame.haslayer('TCP'):
            if trame['TCP'].dport == 23 and trame['TCP'].flags & 0x02:
                syn = True
                src_ip = trame['IP'].src
                dst_ip = trame['IP'].dst
            if trame['TCP'].dport == 23 and trame['TCP'].flags & 0x12:
                syn_ack = True
            if trame['TCP'].dport == 23 and trame['TCP'].flags & 0x10:
                ack = True
    if syn and syn_ack and ack:
            print("[✓] Connexion Telnet detectée !\n")
            handshake = True



def login_info():
     global login_user, login_table, login_password, user_table, pass_table, stop_user, stop_pass
```

```
    if handshake:
         for trame in trames:

           #Pour le user
           if trame.haslayer('TCP') and trame.haslayer('Raw') and  stop_user == False:
               if "login" in trame['Raw'].load.decode(errors="ignore"):

                   login_user = True
                   stop_user = True # Limite à un seul "login" (car il peut avoir une autre string avec login.

           if trame.haslayer('IP') and trame.haslayer('TCP') and trame.haslayer('Raw') and login_user == True:
               if b"\r\n" not in trame['Raw'].load:
                   if trame['IP'].src == src_ip and trame['IP'].dst == dst_ip:
                       a = trame['Raw'].load.decode(errors="ingore")
                       user_table.append(a)

               else:
                   del user_table[len(user_table)-1]
                   login = "".join(user_table)
                   login_table.append(login)
                   login_user = False
                   print(f"[+] Utilisateur trouvé: {login}\n")



           #Pour le mot-de-passe
           if trame.haslayer('TCP') and trame.haslayer('Raw') and stop_pass == False:
               if "Password:" in trame['Raw'].load.decode(errors="ignore"):
                       login_password = True
                       stop_pass = True


           if trame.haslayer('IP') and trame.haslayer('TCP') and trame.haslayer('Raw') and login_password == T

                   if b"\r\n" not in trame['Raw'].load:
                       if trame['IP'].src == src_ip and trame['IP'].dst == dst_ip:
                           a = trame['Raw'].load.decode(errors="ignore")
                           pass_table.append(a)

                   else:
                       del pass_table[len(pass_table)-1]
                       password = "".join(pass_table)
                       login_table.append(password)
                       login_password = False
                       print(f"[+] Mot de passe trouvé: {password}")



if __name__ == "__main__":
    print("[Application] Initialisation... Analyse du fichier\n")
    init_connection()
    login_info()
```

Il nous est maintenant demandé de créer un script python permettant de sniffer le réseau et de trouver une connexion Telnet pour ensuite récupérer le login, mot de passe et les commandes tapées. Pour cela nous avons décidé d'émuler un serveur Telnet sur notre machine pour ensuite s'y connecter via le prompt, puis de créer le programme permettant de sniffer ces connexions.

Voici le programme permettant d'émuler un serveur Telnet

In [ ]:
```
import socket

HOST = "0.0.0.0"
PORT = 2323  # Utilise un port >1024 si tu n'es pas root

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    print(f"Serveur Telnet lancé sur le port {PORT}...")
    conn, addr = s.accept()
    with conn:
        print("Connexion de", addr)
        conn.sendall(b"Bienvenue sur le faux Telnet!\nlogin: ")
        user = conn.recv(1024)
        conn.sendall(b"Password: ")
        passwd = conn.recv(1024)
        conn.sendall(b"\nBienvenue!\n> ")

        while True:
            data = conn.recv(1024)
```

```
            if not data or b"exit" in data:
                break
            response = f"Commande reçue: {data.decode(errors='ignore')}".encode()

            print(response)


        print("Connexion fermée.")
```

Après avoir exécuter ce programme, le serveur est sur écoute sur le port 2323. Après ca nous pouvons éxecuter le deuxième script qui permet de sniffer l'interface réseau

Remarque: Lors de ce test, l'interface que l'on met en écoute est "localhost", dans la vraie vie ce ne sera pas celle-la.

In [ ]:

```
#_____________    .__    __________       .__  _____  _____
#\__    ___/___ |  |   /   _____/ ____ |__|/ ____\/ ____\
#  |    |_/ __ \|  |   \_____  \ /    \|  \   __\\   __\
#  |    ||  ___/|  |__/        \   |  \  ||  |   |  |
#  |____| \___  >____/_______  /___|  /__||__|   |__|
#             \/             \/     \/


from scapy.all import *

port = 2323
src_ip = "0.0.0.0"
dst_ip = "0.0.0.0"
syn = False
syn_ack = False
ack = False
handshake = False
stop_sniffing = False
handshake_detected = False
telnet = False
stop = False
mot = ""
word_list = []
fin_client = False
fin_server = False
pass_on = False
password = ""

def sniff_function(trame):
    global port, src_ip, dst_ip, syn, syn_ack, ack, handshake, stop_sniffing, handshake_detected, telnet, stop,

    if handshake_detected == False:
        if trame.haslayer('TCP') and trame.haslayer('IP'):
            if trame.haslayer('TCP'):
                if trame['TCP'].dport == port and trame['TCP'].flags & 0x02:
                    syn = True
                    src_ip = trame['IP'].src
                    dst_ip = trame['IP'].dst
                if trame['TCP'].sport == port and trame['TCP'].flags & 0x12:
                    syn_ack = True
                if trame['TCP'].dport == port and trame['TCP'].flags & 0x10:
                    ack = True

        if trame.haslayer('TCP') and trame.haslayer(Raw):
            data = trame[Raw].load
            if len(data) >= 2 and data[0] == 0xFF and data[1] in [0xFB, 0xFC, 0xFD, 0xFE]:
                telnet = True



        if syn and syn_ack and ack and telnet and not handshake_detected:
            print("[✓] Connexion Telnet detectée !\n")
            handshake_detected = True

    else:
        if trame.haslayer('TCP') and trame.haslayer('Raw') and pass_on == False:
            if trame['TCP'].dport == port:

                if b"\n" not in trame['Raw'].load:
                    mot += trame['Raw'].load.decode(errors="ignore")

                if b"\n" in trame['Raw'].load:
                    print("[+]", mot)
                    word_list.append(mot)
                    mot = ""

        if trame.haslayer('TCP') and trame.haslayer('Raw'):
            if pass_on:
```

```
                if b"\n" not in trame['Raw'].load:
                    password += trame['Raw'].load.decode(errors="ignore")

                if b"\n" in trame['Raw'].load:
                    print("Le mot de passe est: ", password)
                    password = ""
                    pass_on = False


            if "Password:" in trame[Raw].load.decode(errors="ignore"):
                pass_on = True









        #Détecter fin de connexion sur le port
        if trame.haslayer('TCP'):
            if trame['TCP'].dport == port and 'F' in trame['TCP'].flags:
                fin_client = True
            if trame['TCP'].sport == port and 'F' in trame['TCP'].flags:
                fin_server = True

        if fin_client and fin_server:
            stop = True
            print("\n[-] Connexion fermée")




if __name__ == "__main__":
    print("\n Analyse en cours, veuillez patienter...")
    sniff(filter="", prn=sniff_function, store=0, iface="Loopback Pseudo-Interface 1", stop_filter=lambda x: st
```

Après avoir éxecuter ce programme, nous pouvons simuler une connexion Telnet grâce à l'application PUTTY sur windows. Après quoi le programme Python nous affichera les informations de l'utilisateur ainsi que les commandes qu'il écrit...
Remarque : Nous constatons que PuTTY envoie parfois toute la commande en une seule fois, tandis qu'à d'autres moments, les caractères sont envoyés un par un. Cela est du tout simplement à l'application ici PUTTY

Étape 5 – Challenge HTTP

Dans un premier temps, l'objectif de ce challenge HTTP était de récuperer le login et le mot de passe qui circulaient dans des en-têtes HTTP capturées dans un fichier Wireshark

Voici le programme :

⚠ *Pour executer ce code, il faut impérativement être dans le même dossier que la capture WireShark !*

In [ ]:
```
from scapy.all import *
import base64


syn = False
syn_ack = False
ack = False
connexion = False
once = False
nop = False
decoded_str = ""

trames=rdpcap("www-total.pcapng")

def sniff_connexion():
    global syn, syn_ack, ack, connexion, once
    for trame in trames:
        if trame.haslayer('TCP'):
            if trame['TCP'].dport == 80 and trame['TCP'].flags & 0x02:
                syn = True
            if trame['TCP'].sport == 80 and trame['TCP'].flags & 0x12:
                syn_ack = True
            if trame['TCP'].dport == 80 and trame['TCP'].flags & 0x10:
```

```
                ack = True
        if syn and syn_ack and ack:
            connexion = True
            print("[✓] Connexion HTTP detectée !\n")
            break

def sniff_info():
    global nop, decoded_str
    for trame in trames:
        if trame.haslayer('TCP') and trame.haslayer('Raw'):
            if trame['TCP'].dport == 80 and "Authorization" in trame['Raw'].load.decode(encoding="utf-8", error


                payload = trame['Raw'].load.decode(encoding="utf-8", errors="ignore")
                http_headers = payload.split("\r\n")
                for i in range(len(http_headers)):
                    if "Authorization: Basic" in http_headers[i]:
                        hash = http_headers[i].split(" ")[2]
                        decoded_bytes = base64.b64decode(hash)
                        decoded_str = decoded_bytes.decode("utf-8", errors="ignore")

            if trame['TCP'].sport == 80 and "200" in trame['Raw'].load.decode(encoding="utf-8", errors="ignore"
                payload = trame['Raw'].load.decode(encoding="utf-8", errors="ignore")
                http_headers = payload.split("\r\n")
                for i in range(len(http_headers)):
                    if "Content-Type: text/html" in http_headers[i]:
                        decoded_str = ""

    print("Identifiant:", decoded_str, "\n")


if __name__ == "__main__":
    print("\n[Application] Initialisation... Analyse du fichier\n")
    sniff_connexion()
    sniff_info()
```

Dans la seconde partie de ce challenge, il fallait écrire un programme qui permet de récuperer le login et le mot de passe d'un utilisateur en temps réel (connexion temps réel).

Il m'a donc fallu avoir un script Python qui permet de se connecter directement sur un serveur Web pour avoir acceder à la page.

Voici le programme :

In [ ]:
```
import requests

url = "http://httpbin.org/basic-auth/user/passwd"
username = "user"

# 1ère tentative (échoue)
password = "mauvais"
response = requests.get(url, auth=(username, password))
print("Tentative 1 :")
if response.status_code == 200:
    print("Connexion réussie !")
    print("Réponse JSON :", response.json())
else:
    print("Échec de la connexion. Code HTTP :", response.status_code)
print()

# 2ème tentative (réussit)
password = "passwd"
response = requests.get(url, auth=(username, password))
print("Tentative 2 :")
if response.status_code == 200:
    print("Connexion réussie !")
    print("Réponse JSON :", response.json())
else:
    print("Échec de la connexion. Code HTTP :", response.status_code)
print()

# 3ème tentative (échoue)
password = "motdepasse_incorrect"
response = requests.get(url, auth=(username, password))
print("Tentative 3 :")
if response.status_code == 200:
    print("Connexion réussie !")
    print("Réponse JSON :", response.json())
else:
    print("Échec de la connexion. Code HTTP :", response.status_code)
```

Voici le programme permettant de capture le login et le mot de passe d'un utilisateur en temps réel :

In [ ]:
```
from scapy.all import *


stop_hand = False
syn = False
syn_ack = False
ack = False
yes = ""
login = ""
stop_program = False
fin = False
fin_ack = False
ack2 = False


def sniff_function(trame):
    global stop_hand, syn, syn_ack, ack, yes, login, stop_program, fin, fin_ack, ack2

    #Detecter une connexion TCP
    if stop_hand == False:
        if trame.haslayer(TCP) and trame.haslayer(IP):
            if trame[TCP].dport == 80 and trame[TCP].flags == "S":
                syn = True
            if trame[TCP].sport == 80 and trame[TCP].flags == "SA":
                syn_ack = True
            if trame[TCP].dport == 80 and trame[TCP].flags == "A":
                ack = True
        if syn and syn_ack and ack:
            print("[+] Connexion HTTP détectée !")
            stop_hand = True

    #Detecter USER & MDP
    if trame.haslayer(Raw) and trame.haslayer(TCP) and trame.haslayer(IP):
        if "Authorization: Basic" in trame[Raw].load.decode(errors="ignore"):
            yes = trame[Raw].load.decode(errors="ignore")

        if "HTTP/1.1 200 OK" in trame[Raw].load.decode(errors="ignore"):
            login = yes.split(" ")
            login = login[10].strip()
            decoded_bytes = base64.b64decode(login)
            decoded_str = decoded_bytes.decode('utf-8')
            decoded_str = decoded_str.split(":")
            print(f"Le user est : {decoded_str[0]} et le mot de passe est {decoded_str[1]}")


    #Détecter la fin de la connexion HTTP

    if trame.haslayer(TCP) and trame.haslayer(IP):
        tcp_layer = trame[TCP]

        # FIN + ACK du serveur (sport == 80)
        if tcp_layer.sport == 80 and "F" in tcp_layer.flags and "A" in tcp_layer.flags:
            fin = True

        # FIN + ACK du client (dport == 80)
        if tcp_layer.dport == 80 and "F" in tcp_layer.flags and "A" in tcp_layer.flags:
            fin_ack = True

        # Dernier ACK du serveur sans FIN
        if tcp_layer.sport == 80 and "A" in tcp_layer.flags and "F" not in tcp_layer.flags:
            ack2 = True

    if fin and fin_ack and ack2:
        print("[+] Fin de la connexion HTTP...")
        stop_program = True




def stop_filter_function(pkt):
    return stop_program  # Arrête le sniff quand stop_program est True


sniff(filter="", prn=sniff_function, store=0, iface="Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC", stop_filt
```

Ce programme sait détecter si le mot de passe est le nom d'utilisateur est correcte ou pas. S'il est correcte : il l'affiche, sinon, il passe.

Fin du rapport

Par Auduberteau Emilien