TP n° 03 – Anonymisation de fichiers PCAP

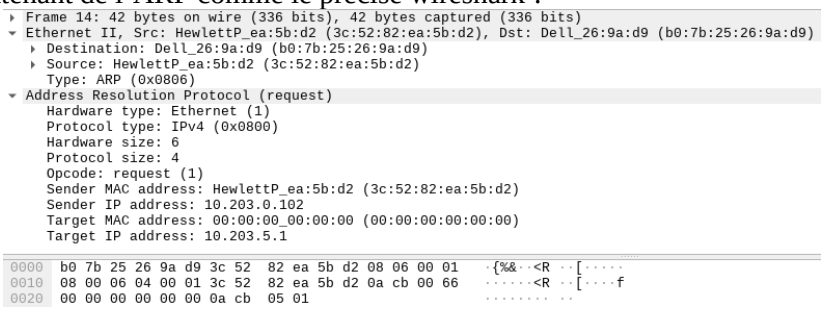
Le but de ce TP est de manipuler des fichiers PCAP (captures de trames réseau) pour les anonymiser en utilisant JSON.

1 Fichier PCAP au format JSON :

1. La commande `tshark` (version texte de wireshark) permet d'afficher le contenu d'un fichier PCAP avec l'option `-r`.

```
...
10   3.798174   10.203.5.1 → 10.203.0.102 ICMP 98 Echo (ping) reply    id=0x7b42, seq=1/256, ttl=64 (request in 9)
11   4.075293 Dell_26:a1:75 → Broadcast    ARP 60 Who has 10.203.0.128? Tell 10.203.14.1
12   5.099252 Dell_26:a1:75 → Broadcast    ARP 60 Who has 10.203.0.128? Tell 10.203.14.1
13   8.650346 HewlettP_ea:5b:d2 → Broadcast    ARP 42 Who has 10.203.6.1? Tell 10.203.0.102
14   8.911059 HewlettP_ea:5b:d2 → Dell_26:9a:d9 ARP 42 Who has 10.203.5.1? Tell 10.203.0.102
15   8.912302 Dell_26:9a:d9 → HewlettP_ea:5b:d2 ARP 60 10.203.5.1 is at b0:7b:25:26:9a:d9
...
```

2. Utiliser en plus l'option `-T` pour avoir un affichage au format JSON « raw ». Créer alors le fichier `tp3.json`.
3. Créer alors un script python qui charge le fichier JSON et affiche le nombre de paquets.
4. Afficher ensuite la trame 14 (ARP) en hexadécimal en utilisant les données JSON. Attention, dans le format JSON, les valeurs peuvent apparaître plusieurs fois. Il faudra donc bien respecter le format d'une trame Ethernet contenant de l'ARP comme le précise wireshark :



Utiliser 2 listes (1 pour `eth` et 1 pour `arp`) contenant les clés à utiliser.

- Couche **`eth`**, 3 champs d'entête : `@dst_mac, @src_mac, type`
- Couche **`arp`**, 9 champs : `hw_type, proto_type, hw_size, proto_size, opcode, @src_hw_mac, @src_ipv4, @dst_hw_mac, @dst_ipv4`

On doit obtenir au final :
`b07b25269ad93c5282ea5bd2080600010800060400013c5282ea5bd20acb00660000000000000acb0501`

2 Création d'un fichier PCAP à partir d'une trame en hexadécimal :

1. Pour générer un fichier PCAP lisible par wireshark/tshark, il faut en premier écrire l'entête PCAP :

```
from struct import pack
pcap=open('arp-cb.pcap','wb')
pcap_hdr=pack('=IHHiIII',
              0xA1B2C3D4,   # magic_number
              2,            # version_major
              4,            # version_minor
              0,            # timezone
              0,            # sigfigs
              0x40000,      # snaplen
              1)            # ethernet
pcap.write(pcap_hdr)
```

2. Ensuite, suivant le temps d'arrivée et la taille du paquet, on doit écrire l'entête spécifique pour le paquet puis le paquet :

```
pkt_hexa='b07b25269ad93c5282ea5bd2080600010800060400013c5282ea5bd20acb00660000000000000acb0501'
ts_sec,ts_usec,pkt_len=0,0,len(pkt_hexa)//2
inc_len=pkt_len if pkt_len <= 65535 else 65535
pkt_hdr=pack('=IIII',
              ts_sec,       # ts_sec
```

```
                ts_usec,      # ts_usec
                inc_len,      # number of octets of packet saved in file
                pkt_len)      # actual length of packet
pcap.write(pkt_hdr)
pcap.write(bytes.fromhex(pkt_hexa))
pcap.close()
```

3. Vérifier alors que le fichier généré est bien valide et qu'il contient bien les données de la trame 14.

3 Anonymisation de trames ARP et ICMP :

1. On désire maintenant anonymiser les adresses MAC de l'entête Ethernet de la trame 14 en remplaçant les 3 derniers octets par un compteur. Utiliser un dictionnaire (par exemple `ethAddr`) pour sauvegarder automatiquement la transformation des adresses MAC. On doit obtenir la trame suivante :

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▾ Ethernet II, Src: HewlettP_00:00:02 (3c:52:82:00:00:02), Dst: Dell_00:00:01 (b0:7b:25:00:00:01)
  ▸ Destination: Dell_00:00:01 (b0:7b:25:00:00:01)
  ▸ Source: HewlettP_00:00:02 (3c:52:82:00:00:02)
    Type: ARP (0x0806)
▸ Address Resolution Protocol (request)
```

NB : On pourra créer les fonctions `getARPHexaFromJSON1(ethAddr,pkts_json,frameNum)` et `savePCAP1(fileName,pkt_hexa)` :

```
js=open('tp3.json').read()
pkts_json=json.loads(js)
ethAddr={}
pkt_hexa=getARPHexaFromJSON1(ethAddr,pkts_json,14)
savePCAP1('arp-anon1.pcap',pkt_hexa)
```

2. Comment faire pour anonymiser également les adresses MAC de la couche ARP (fonction `getARPHexaFromJSON2()`), sauf pour la valeur `000000000000` qui doit rester inchangée ?
On doit obtenir la trame suivante :

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▾ Ethernet II, Src: HewlettP_00:00:02 (3c:52:82:00:00:02), Dst: Dell_00:00:01 (b0:7b:25:00:00:01)
  ▸ Destination: Dell_00:00:01 (b0:7b:25:00:00:01)
  ▸ Source: HewlettP_00:00:02 (3c:52:82:00:00:02)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HewlettP_00:00:02 (3c:52:82:00:00:02)
    Sender IP address: 10.203.0.102
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.203.5.1
```

3. Transformer maintenant tous les paquets ARP avec cette méthode (fonctions `savePCAP3()` et `getARPHexaFromJSON3(ethAddr,p)`). On pourra tester la présence de la couche ARP dans les paquets avant de les traiter. L'adresse MAC de broadcast doit aussi rester inchangée.
4. Comment changer les adresses IPv4 sur le même principe que pour les adresses MAC ?
5. Anonymiser enfin les paquets ICMP en modifiant les adresses MAC de la couche Ethernet et les adresses IP de la couche IPv4. ATTENTION, dans l'entête IPv4, les drapeaux et le décalage de fragment forment 16 bits donc il faut compléter avec des 0 la valeur de `ip.frag_offset_raw`.
On pourra utiliser les listes de champs JSON ci-dessous pour la génération de l'hexadécimal :

```
ipv4=['ip.version_raw','ip.dsfield_raw','ip.len_raw','ip.id_raw','ip.flags
_raw','ip.frag_offset_raw','ip.ttl_raw','ip.proto_raw','ip.checksum_raw','
ip.src_raw','ip.dst_raw']
icmp=['icmp.type_raw','icmp.code_raw','icmp.checksum_raw','icmp.ident_raw'
,'icmp.seq_raw','icmp.data_time_raw','data_raw']
```

6. Activer la vérification du checksum dans les préférences du protocole IPv4 de wireshark, et constater que les paquets ICMP anonymisés ne sont plus valides. Corriger alors les entêtes IPv4 en recalculant le checksum (voir la RFC791 page 14 ou bien par exemple https://en.wikipedia.org/wiki/IPv4_header_checksum).
NB : en Python l'opérateur >> permet de décaler à droite (en binaire) un entier (division par les puissance de 2), l'opérateur & permet de faire un ET bit à bit et l'opérateur ~ donne le complément à 1.