UNIVERSITY OF LIMOGES

Faculty of Science and Technology

Master 2 CRYPTIS

Sécurité de l'Information et Cryptologie

(CRYPTIS)

Parcours Informatique

Administration et sécurité des réseaux et systèmes - Semestre I

IPsec

SALAME Joe

Enseignants

M. CONCHON Emmanuel

27 février 2024

Chapitre 1

Static

1.1 Mode Tunnel

Dans le mode tunnel, le paquet IP d'origine est chiffré et encapsulé dans un nouveau paquet IP. Le mode tunnel est couramment utilisé pour créer des connexions VPN. Ce mode ajoute une couche de sécurité supplémentaire en cachant l'adresse IP source et de destination d'origine, ce qui rend plus difficile la collecte d'informations sur le réseau.

Premièrement j'ai généré les identifiants SPI et RID en utilisant Openssl . Les clés générées (k1, k2, k3, k4) sont des chaînes hexadécimales de 32 octets. Les SPI et les RID sont des chaînes hexadécimales de 4 octets, tous préfixés par "0x" pour indiquer leur format hexadécimal.

Les deux premières commandes ajoutent des états IPsec dans le namespace h1. Ces états spécifient les paramètres de sécurité pour le chiffrement et l'authentification des paquets IP envoyés de h1 à h2 et vice-versa. Les valeurs de SPI et de RID sont utilisées pour identifier ces états. Ensuite j'ai ajouté des politiques IPsec dans h1. Ces politiques définissent quelles sont les communications qui doivent être sécurisées (du point de vue de h1) et quelles sont les règles à appliquer pour sécuriser ces communications. Elles spécifient également les paramètres de tunnel pour la sécurité.

```
ip netns exec h1 ip xfrm state add src 10.10.10.1 dst 10.0.0.1 proto esp spi
    $SPI1 reqid $RID1 mode tunnel auth sha256 $k1 enc aes $k2
ip netns exec h1 ip xfrm state add src 10.0.0.1 dst 10.10.10.1 proto esp spi
    $SPI2 reqid $RID2 mode tunnel auth sha256 $k3 enc aes $k4

ip netns exec h1 ip xfrm policy add src 10.0.0.1 dst 10.10.10.1 dir in tmpl src
    10.0.0.1 dst 10.10.10.1 proto esp reqid $RID2 mode tunnel
ip netns exec h1 ip xfrm policy add src 10.10.10.1 dst 10.0.0.1 dir out tmpl src
     10.10.10.1 dst 10.0.0.1 proto esp reqid $RID1 mode tunnel
```

J'ai fait la même chose pour la configuration du namespace h2.

Dans la capture d'écran ci-dessous, j'ai exécuté une commande socat sur les deux hôtes h1 et h2 sans utiliser IPsec. Comme nous pouvons le voir, la communication n'a pas été chiffrée et le message est en claire dans le segment TCP.

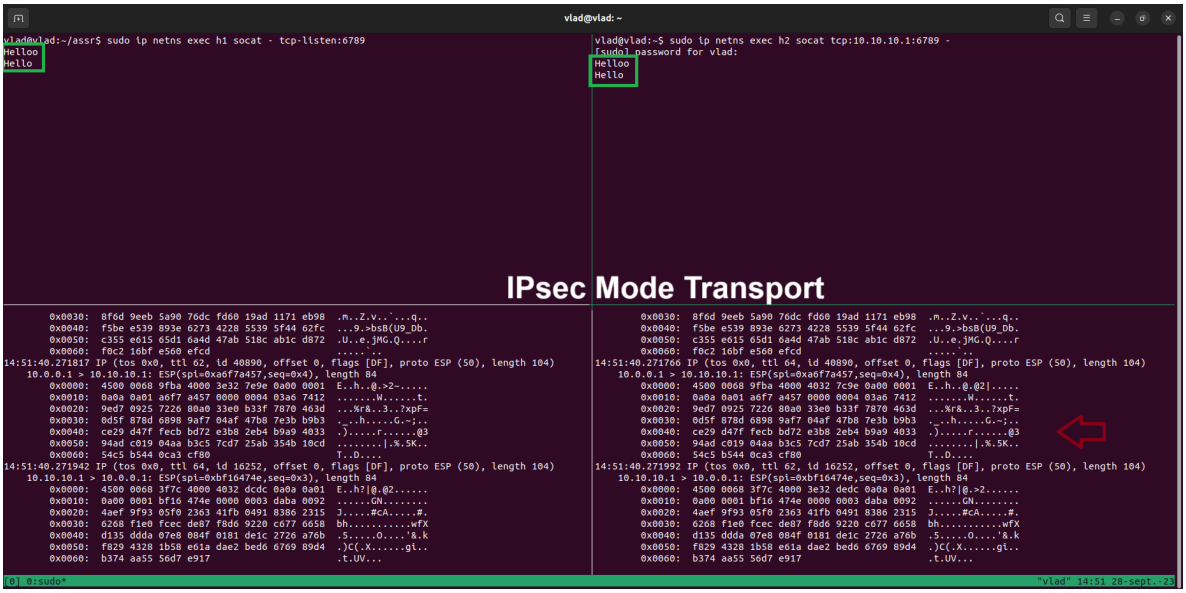
Comme vous remarquez ci-dessous, après l'implémentation d'IPsec nous ne pouvons plus lire le message transmit à l'aide de socat vu qu'il est chiffré.

1.2 Mode Transport

Dans le mode transport, seuls les paquets de données (payload) sont chiffrés ou signés. L'en-tête IP d'origine est conservé, bien que certaines informations puissent être ajoutées. Le mode transport est souvent utilisé pour sécuriser la communication point à point entre deux hôtes. Le mode transport est plus efficace en termes de traitement, car il ne chiffre que les données elles-mêmes, pas les en-têtes IP.

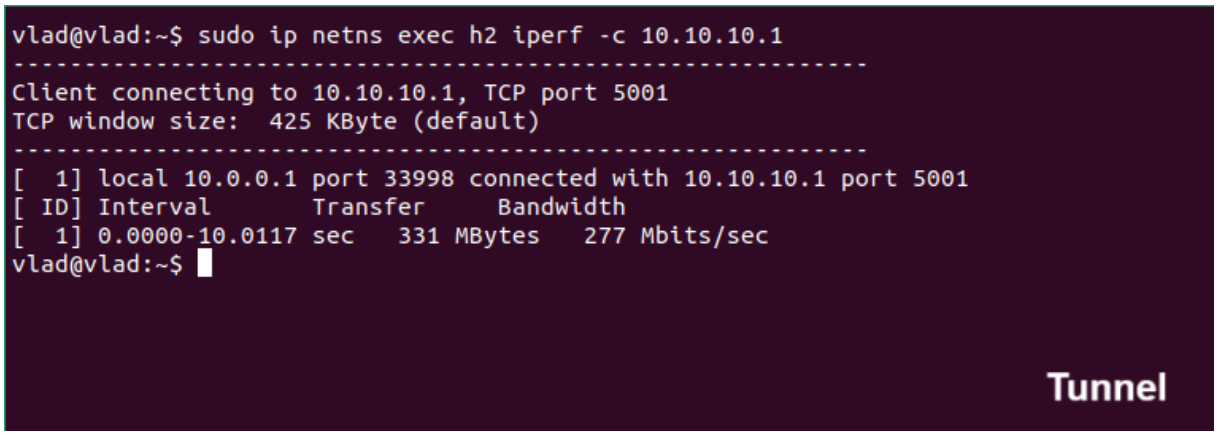
Pour changer en mode transport nous n'avons qu'à changer le mode dans la configuration

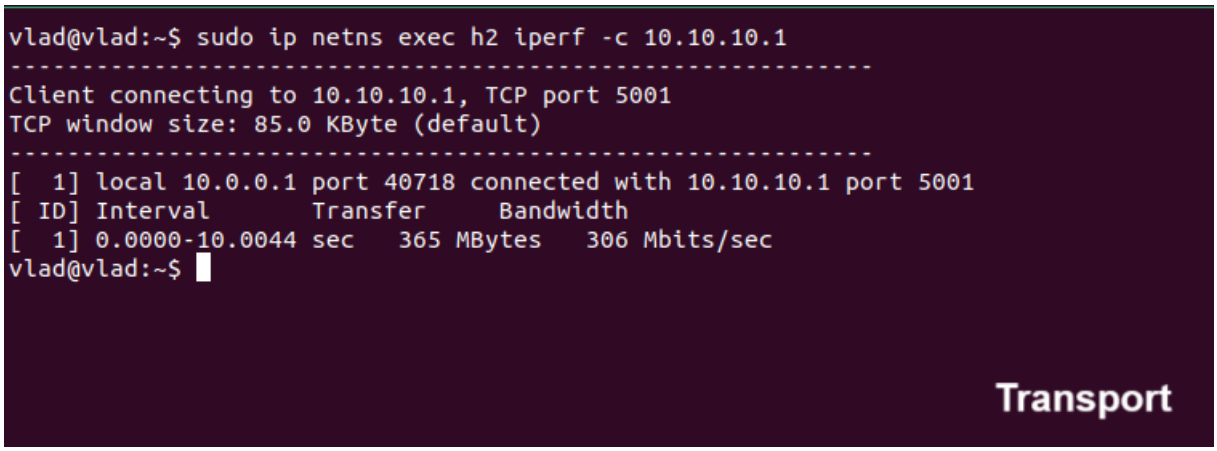
de la policy et du state.



1.3 Comparaison

concernant la vitesse, après avoir utilisé iperf pour mesurer la différence de vitesse entre le mode transport et le mode tunnel nous remarquons que leurs vitesses sont si proches donc nous pouvons constater que le mode choisi n'affecte pas trop la vitesse de transfert.





Pour comparer les tailles des pacquets des deux modes, j'ai sniffé le traffic des deux modes en utilisant tcpdump et à partir des paquets capturés, nous pouvons remarquer que le paquet en mode tunnel fait 120 octets, soit 1,15 fois plus grand que le paquet en mode transport (104 octets). cela est dû à l'encapsulation du paquet IP d'origine avec un nouvel en-tête IP en mode tunnel.

```
        0x0030:  8f6d 9eeb 5a90 76dc fd60 19ad 1171 eb98  .m..Z.v..`...q..
        0x0040:  f5be e539 893e 6273 4228 5539 5f44 62fc  ...9.>bsB(U9_Db.
        0x0050:  c355 e615 65d1 6a4d 47ab 518c ab1c d872  .U..e.jMG.Q....r
        0x0060:  f0c2 16bf e560 efcd                      .....`..
14:51:40.271817 IP (tos 0x0, ttl 62, id 40890, offset 0, flags [DF], proto ESP (50), length 104)
    10.0.0.1 > 10.10.10.1: ESP(spi=0xa6f7a457,seq=0x4), length 84
        0x0000:  4500 0068 9fba 4000 3e32 7e9e 0a00 0001  E..h..@.>2~.....
        0x0010:  0a0a 0a01 a6f7 a457 0000 0004 03a6 7412  .......W......t.
        0x0020:  9ed7 0925 7226 80a0 33e0 b33f 7870 463d  ...%r&..3..?xpF=
        0x0030:  0d5f 878d 6898 9af7 04af 47b8 7e3b b9b3  ._..h.....G.~;..
        0x0040:  ce29 d47f fecb bd72 e3b8 2eb4 b9a9 4033  .).....r......@3
        0x0050:  94ad c019 04aa b3c5 7cd7 25ab 354b 10cd  ........|.%.5K..
        0x0060:  54c5 b544 0ca3 cf80                      T..D....
14:51:40.271942 IP (tos 0x0, ttl 64, id 16252, offset 0, flags [DF], proto ESP (50), length 104)
    10.10.10.1 > 10.0.0.1: ESP(spi=0xbf16474e,seq=0x3), length 84
        0x0000:  4500 0068 3f7c 4000 4032 dcdc 0a0a 0a01  E..h?|@.@2......
        0x0010:  0a00 0001 bf16 474e 0000 0003 daba 0092  ......GN........
        0x0020:  4aef 9f93 05f0 2363 41fb 0491 8386 2315  J.....#cA.....#.
        0x0030:  6268 f1e0 fcec de87 f8d6 9220 c677 6658  bh.........wfX
        0x0040:  d135 ddda 07e8 084f 0181 de1c 2726 a76b  .5.....O....'&.k
        0x0050:  f829 4328 1b58 e61a dae2 bed6 6769 89d4  .)C(.X......gi..
        0x0060:  b374 aa55 56d7 e917                      .t.UV...
```

TRANSPORT

```
vlad@vlad:~$ sudo ip netns exec r1 tcpdump -lnvvX "ip"
tcpdump: listening on r1-eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:18:57.850278 IP (tos 0x0, ttl 62, id 0, offset 0, flags [DF], proto ESP (50), length 120)
    10.0.0.1 > 10.10.10.1: ESP(spi=0x2ff84fae,seq=0x9), length 100
        0x0000:  4500 0078 0000 4000 3e32 1e49 0a00 0001  E..x..@.>2.I....
        0x0010:  0a0a 0a01 2ff8 4fae 0000 0009 6d48 25b1  ..../.O.....mH%.
        0x0020:  c6cf 3763 a27a b614 814d 84fd 9584 e76f  ..7c.z...M.....o
        0x0030:  97c5 1f24 01d6 1049 30d5 d7b2 fcc7 17af  ...$...I0.......
        0x0040:  89a0 8c80 7711 0fc7 2cc9 16ec c4a8 01ef  ....w...,.......
        0x0050:  15b3 866f 01eb 3c44 6e52 2c93 e695 b4dd  ...o..<DnR,.....
        0x0060:  1fd0 e15b 439c 2b92 7c87 cc75 4913 af5a  ...[C.+.|..uI..Z
        0x0070:  6663 5f1b 7296 ff8e                      fc_.r...
14:18:57.850423 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ESP (50), length 120)
    10.10.10.1 > 10.0.0.1: ESP(spi=0xe90fa004,seq=0x7), length 100
        0x0000:  4500 0078 0000 4000 4032 1c49 0a0a 0a01  E..x..@.@2.I....
        0x0010:  0a00 0001 e90f a004 0000 0007 4d25 d5cf  ............M%..
        0x0020:  d07f 3025 59c2 5a89 dfb1 3fac f537 1a08  ..0%Y.Z...?..7..
        0x0030:  c502 67aa 0ac6 e660 bb4f ed3c 0e41 7eab  ..g....`.O.<.A~.
        0x0040:  6b46 ec09 accf cdca cf56 8c29 c4da 0316  kF.......V.)....
        0x0050:  d1c5 c133 c158 16ee aa88 d1ce d50d aaa1  ...3.X..........
        0x0060:  35e2 bc89 1e72 b150 320f c489 c9b6 5197  5....r.P2.....Q.
        0x0070:  28e9 4f64 824b 6e9f                      (.Od.Kn.
```

Tunnel

Chapitre 2

Tunnel avec IKE

2.1 Configuration

Dans la partie configuration j'ai créé pour chaque namespace les paramètres utilisés dans le tunnel (clés, certificats) en appliquant les commandes présentées dans la donné et j'ai créé le fichier de configuration swanctl.conf pour chaque namespace

```
#swanctl.conf h2
    connections {
        host-host {
                remote_addrs = 10.10.10.1
                local {
                        auth=pubkey
                        certs = h2.pem
                }
                remote {
                        auth = pubkey
                        id = "CN=10.10.10.1"
                }
                children {
                        net-net {
                                start_action = trap
                        }
                }
        }
}
```

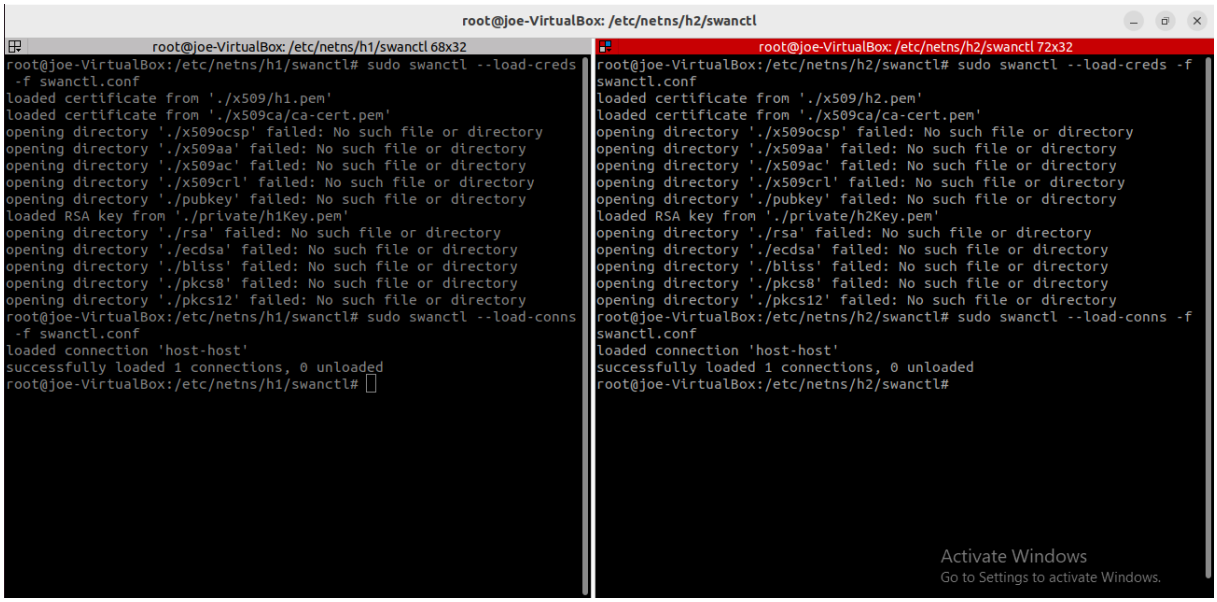
```
#swanctl.conf r2
    connections {
  net-net {
    remote_addrs = 172.16.1.1
  local {
    auth = pubkey
    certs = r2.pem
  }
  remote {
    auth = pubkey
    id = "CN=172.16.1.1"
    }
```

```
  children {
    net-net {
      local_ts = 10.0.0.0/24
      remote_ts = 10.10.10.0/24
      start_action = trap
      }
    }
  }
}
```

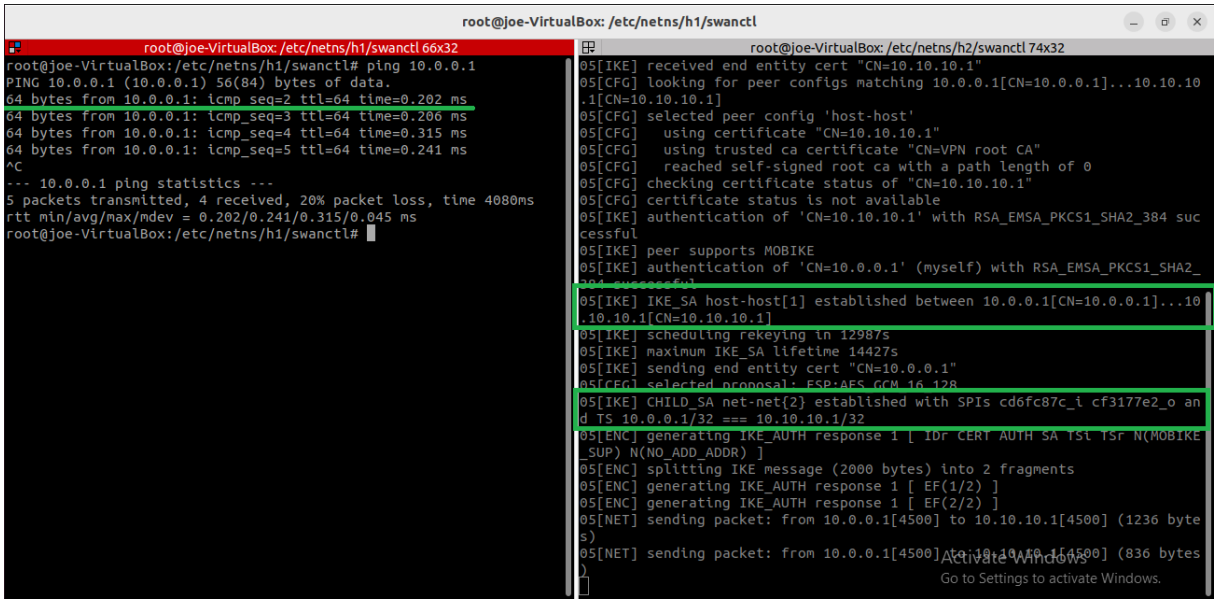
2.2 Établissement du tunnel entre h1 et h2

Après avoir utilisé les commandes suivantes dans les namespaces h1 et h2 j'ai obtenu les résultats ci-dessous.

```
$ sudo ip netns exec h1 bash
$ sudo killall charon
$ sudo killall starter
$ sudo mkdir /tmp/h1
$ sudo mount --bin /tmp/h1 /run
$ sudo ipsec start
$ swanctl --load-creds
$ swanctl --load-conns
```

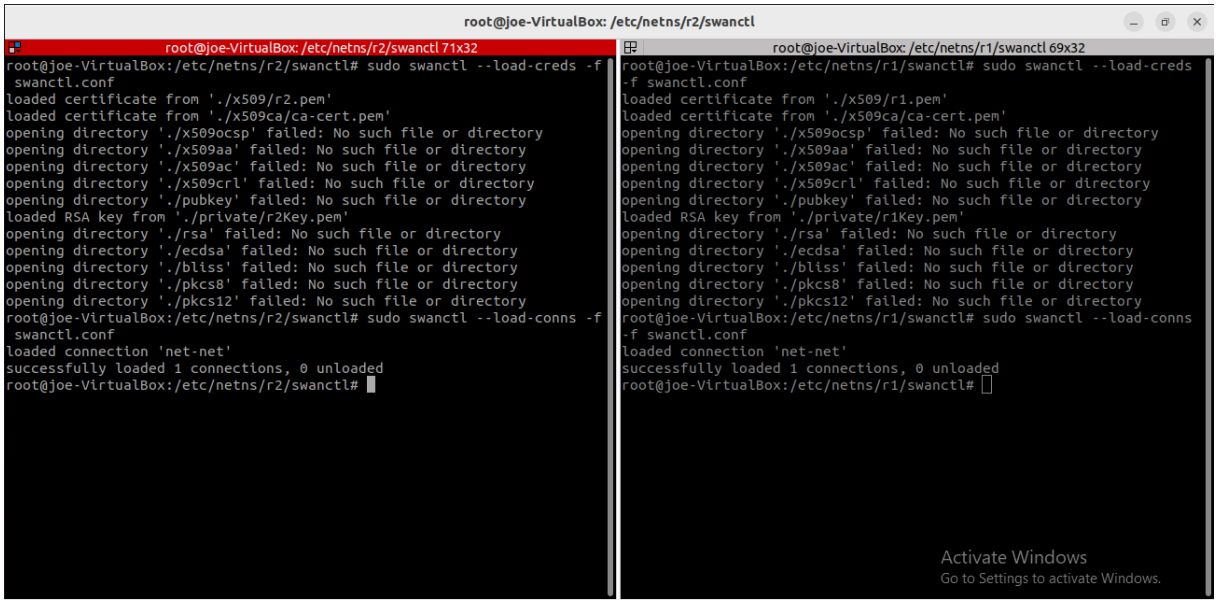


Ensuite, j'ai envoyé un ping de h1 à h2, nous pouvons remarquer que l'icmp-seq a commencé par 2, ce qui indique que le premier paquet icmp a été perdu car le tunnel n'était pas encore établi. Dans l'autre partie de la capture, la commande swanctl –log m'a montré le succès de la négociation entre h1 et h2

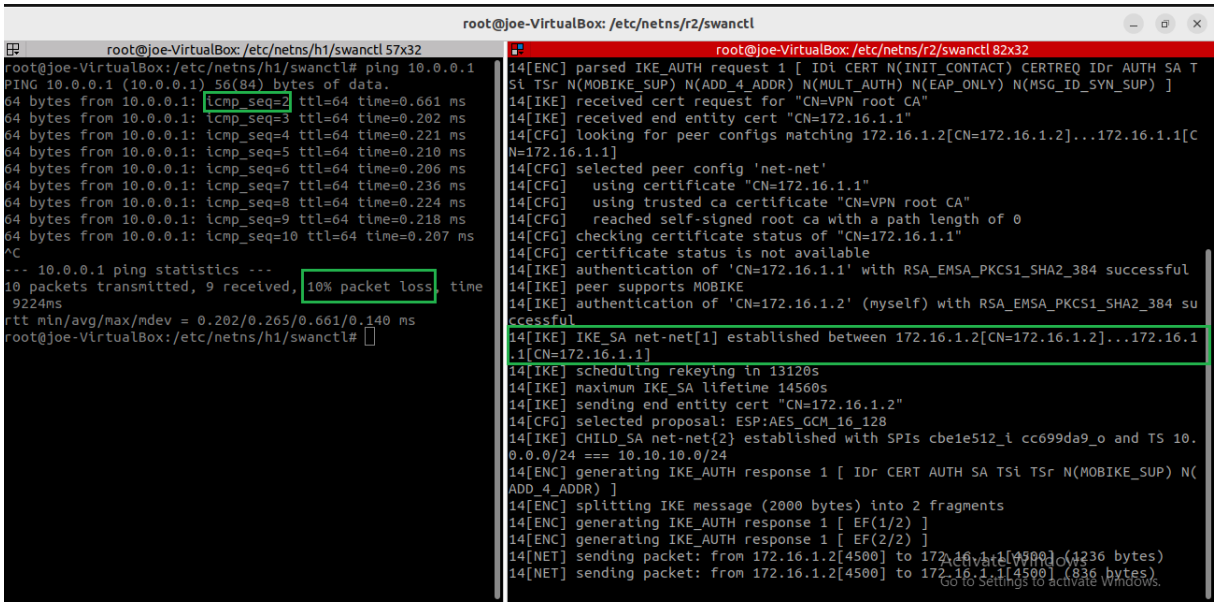


2.3 Établissement du tunnel entre r1 et r2

Pour r1 et r2, j'ai exécuté les mêmes commandes précédentes.



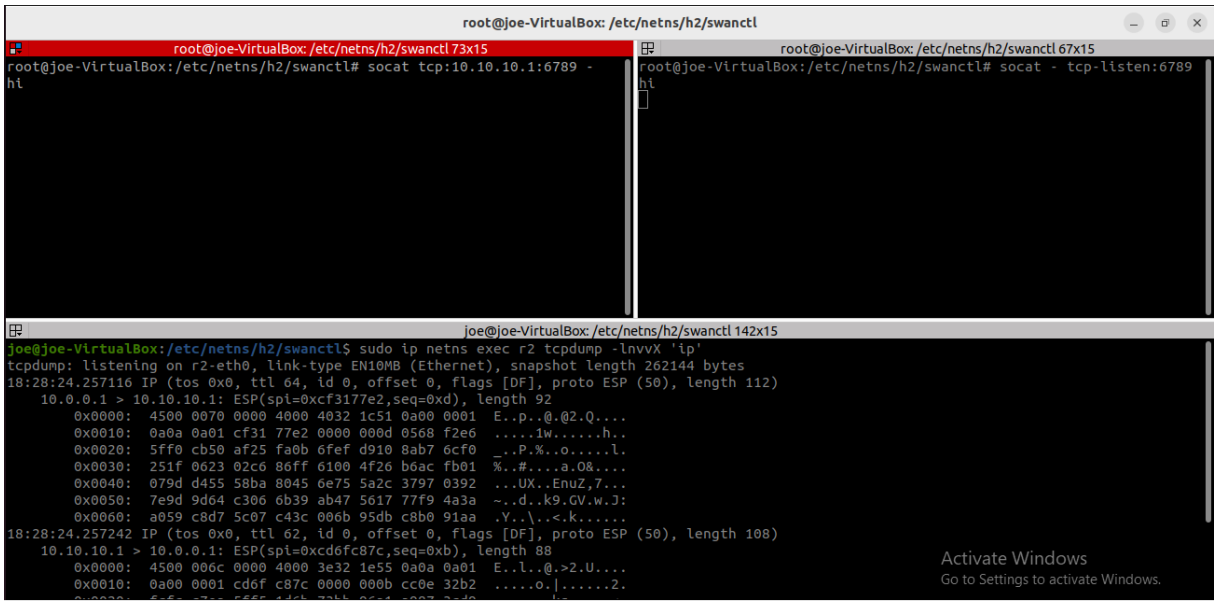
Ensuite, j'ai pingé de h1 à h2 et nous pouvons remarquer sur la capture ci-dessous que le ping arrive à destination après avoir perdu le premier paquet icmp, ce qui indique que la négociation a réussi et que les tunnels ont été combinés.



2.4 Résultats

2.4.1 h1 et h2

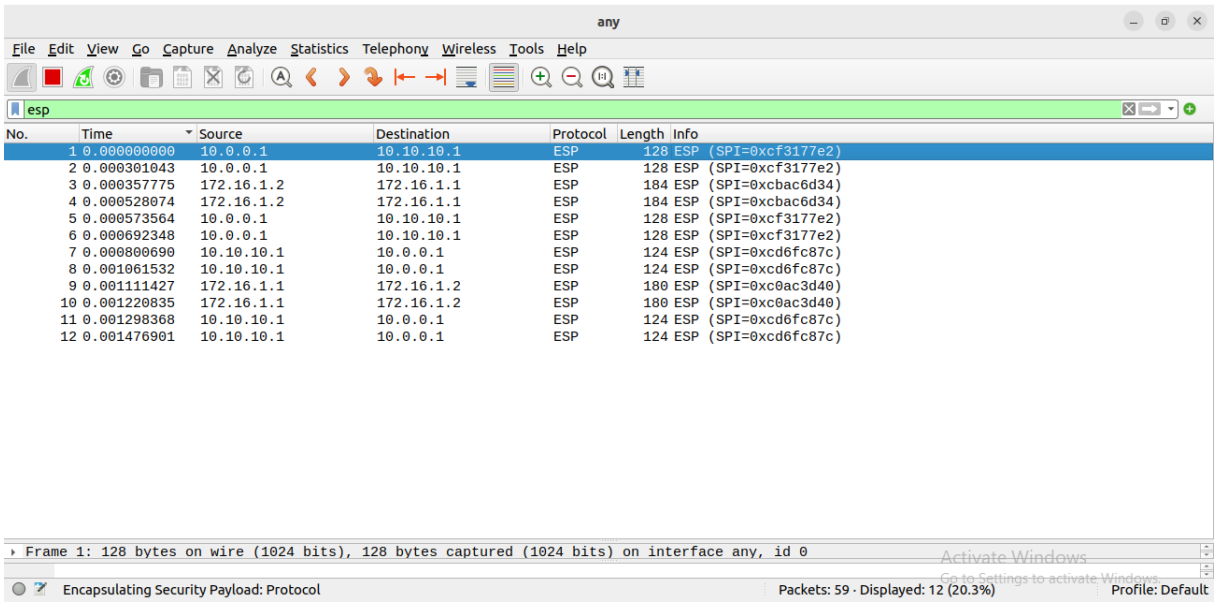
Avant la construction du tunnel entre r1 et r2 j'ai testé le tunnel entre h1 et h2 en établissant une connexion TCP entre les deux en utilisant socat. Ensuite, j'ai envoyé un message de h1 à h2 qui va être sniffé à l'aide de Tcpdump. Nous pouvons remarquer avec la capture ci-dessous que le payload est chiffré et que le protocole utilisé est ESP.

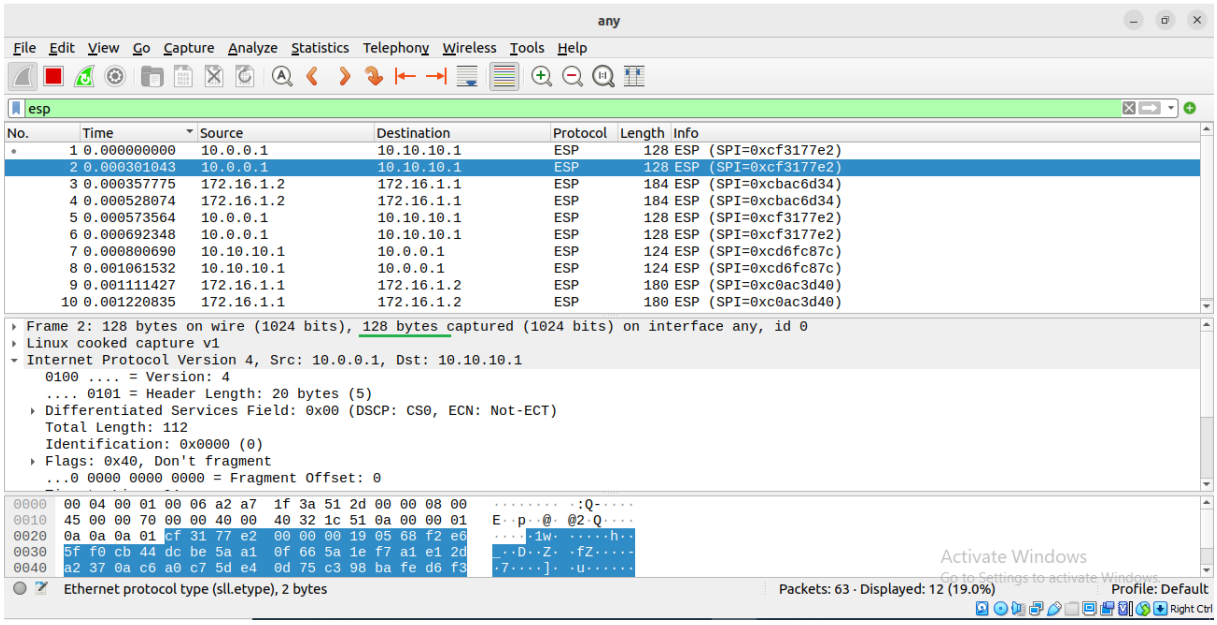


2.4.2 r1 et r2

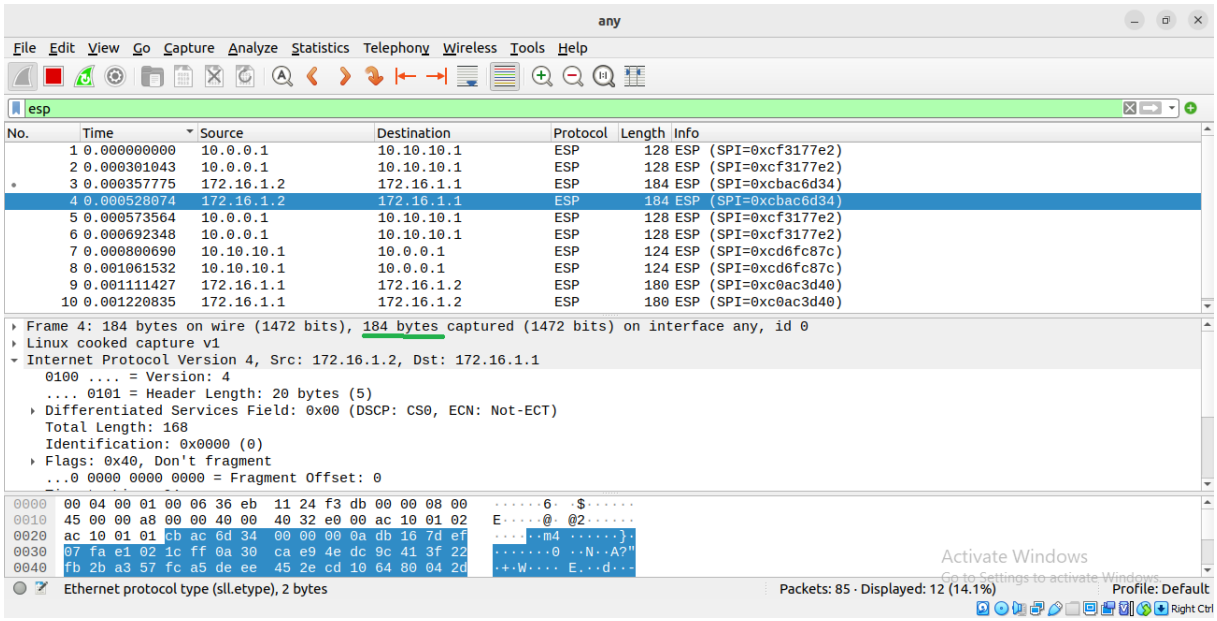
Maintenant, après l'établissement des tunnels entre r1 et r2, j'ai utilisé WireShark pour capturer le traffic entre h1 et h2.

Dans les deux captures suivantes, nous pouvons voir qu'entre h1 et r1, le payload du paquet est chiffré avec le protocole ESP sans aucun changement dans l'en-tête IP.





Dans la capture ci-dessous nous pouvons remarquer que le paquet ip d'origine a été encapsulé avec un nouvel en-tête ip ayant l'ip source 172.16.1.2 (adresse r2) et l'ip destination 172.168.1.1 (adresse r1) et chiffré avec le protocole ESP.
De plus, la longueur du paquet est devenue plus grande après l'encapsulation de 184 octets, soit 1,43 fois plus grande que le paquet avant l'encapsulation (128 octets).



2.5 RoadWarrior

2.5.1 configuration

J'ai créé les mêmes paramètres que les configurations précédentes pour h1 et r2 (clés et certificats). J'ai uniquement modifié le champ CN du certificat des namespaces en mettant le mail à la place de l'adresse ip.

La création des certificats est devenue la suivante :

```
#namespace r2 certificate
pki --pub --in pki/private/r2Key.pem --type rsa | pki --issue --cacert pki/
    cacerts/ca-cert.pem --cakey pki/private/ca-key.pem --lifetime 1825 --dn "CN=
    r2@assr.com" --outform pem > pki/certs/r2.pem
```

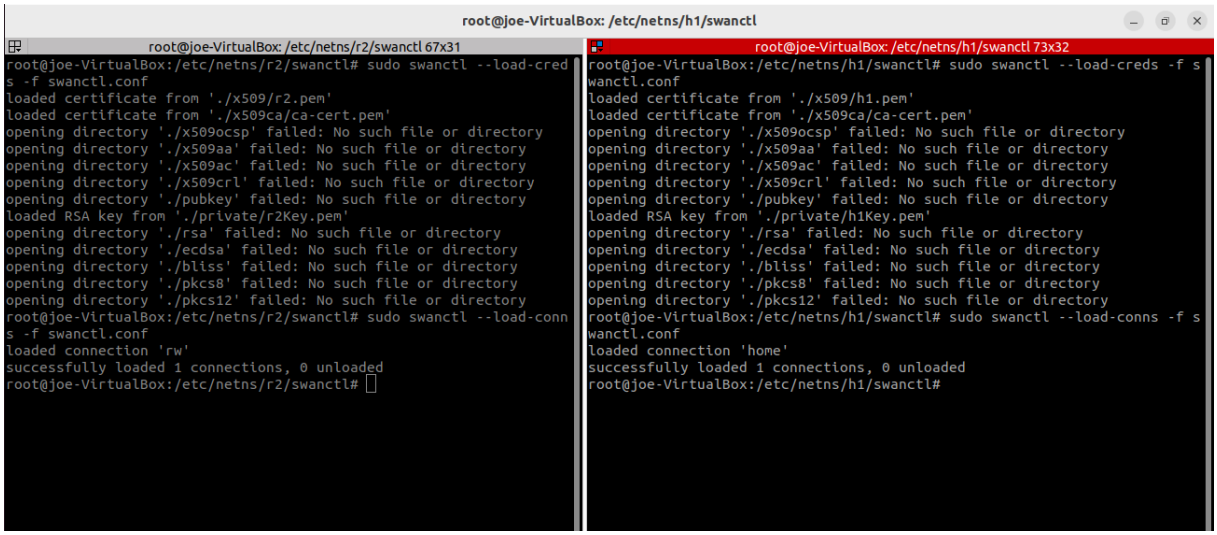
```
#namespace h1 certificate
pki --pub --in pki/private/h1Key.pem --type rsa | pki --issue --cacert pki/
    cacerts/ca-cert.pem --cakey pki/private/ca-key.pem --lifetime 1825 --dn "CN=
    h1@assr.com" --outform pem > pki/certs/h1.pem
```

Concernant les fichiers de swanctl.conf le contenu de chaque namespace est :

```
#r2 swanctl.conf
connections {
    rw {
        local {
            auth=pubkey
            certs=r2.pem
        }
        remote {
            auth=pubkey
        }
        children {
            rw {
                local_ts=10.0.0.0/24
            }
        }
    }
}
```

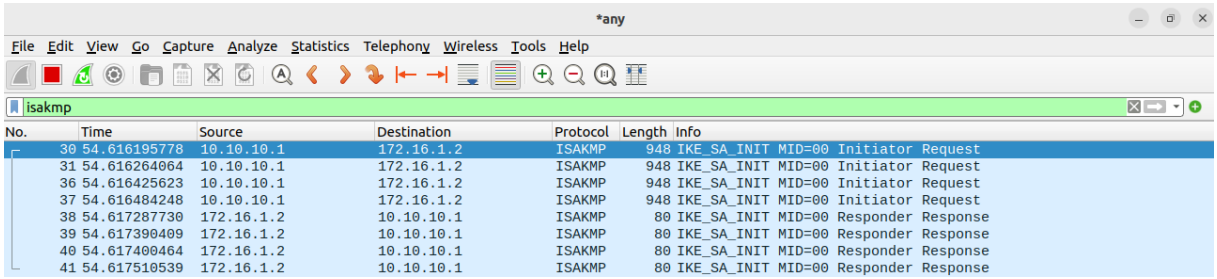
```
#h1 swanctl.conf
connections {
    home {
        remote_addrs = 172.16.1.2
        local {
            auth=pubkey
            certs=h1.pem
        }
        remote {
            auth=pubkey
            id="CN=r2@assr.com"
        }
        children {
            home {
                remote_ts=10.0.0.0/24
                start_action=start
            }
        }
    }
}
```

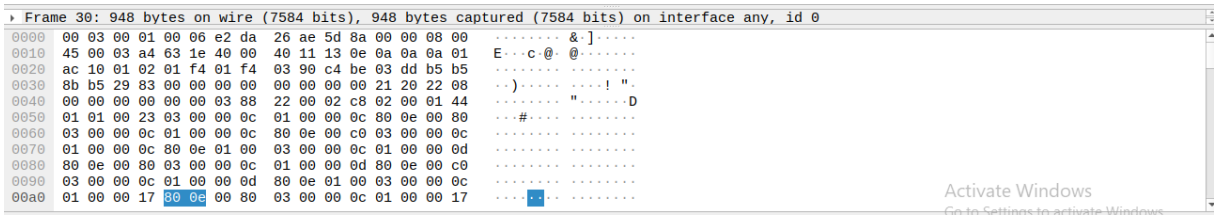
Ensuite, j'ai chargé le certificat et la connexion pour chaque namespace :



2.5.2 Résultat

La capture suivante nous montre des paquets ISAKMP capturés avec Wireshark, ce qui indique que les namespaces ont négocié avec succès les uns avec les autres sur l'établissement du tunnel.





Et dans la capture ci-dessous, j'ai testé le tunnel entre les namespaces h1 et h2 avec socat en utilisant tcpdump dans r1 pour sniffer le trafic. Nous voyons que le traffic a été chiffré avec ESP ce qui indique que le tunnel est bien établi.