README.md

Sécurité Réseau - TP 1

Matthieu Delecluse, Clément RUIZ

Rappels Théoriques

UDP

UDP est un protocole de communication appartenant à la couche Transport (4 OSI). Il permet d'envoyer des données vers un couple IP/port **sans en vérifier la réception** du paquet. Il est notamment utilisé pour transporter de la donnée "volatile", comme par exemple un flux vidéo : si une ou 2 images sont perdues ou incomplètes, la qualité du stream n'est que très faiblement dégradée, et la perte est quasi invisible à l'oeil nu.

TCP

TCP est lui aussi un protocole de communication appartenant à la couche Transport (4 OSI). Contrairement à UDP, il initie un **canal de communication** entre les 2 interlocuteurs, avec une **vérification de l'intégrité de la donnée** par contrôle de hash de chaque paquet et **accusé de réception**. Cette connexion se fait au moyen d'un "Three-way-handshake", une amorce protocolaire en 3 étapes vérifiant la disponibilité des hôtes. Une fois la connexion établie, il est possible de faire transiter de la donnée implémentant d'autres protocoles, comme par exemple HTTP, FTP, avec une possible couche de chiffrement (HTTPS, SSH). *ScreenShot*

Étude approfondie du fonctionnement du réseau avec scapy

Pour ce lab, nous utilisons 2 machines virtuelles :

- *attacker* --> Ubuntu 16.04.3 desktop avec 2 interfaces réseau (Une NAT pour internet, une interne au lab)
- *target* --> Ubuntu 16.04.3 server avec 2 interfaces réseau (Une NAT pour internet, une interne au lab) Héberge un serveur HTTP écoutant sur le port 80

Forger ses paquets avec scapy

Découverte de l'outil

Scapy est une librairie python nous permettant de forger des paquets réseau. Celle-ci nous propose des outils permettant de créer chacune des couches d'encapsulation qui permettent le transfert de données (par exemple: Ether -> MAC -> IP -> TCP). Chaque couche du model OSI (layer) est donc représentée par un objet et un opérateur de composition (`/`) sert à "assembler" notre packet final. Chaque layer peut être personnalisé, tout en respectant certaines règles d'héritages vis-à-vis des layers supérieurs (dans le model OSI).

Pour faciliter la manipulation et la lisibilité, nous avons créé une classe "Machine" décrite dans le snippet ci-dessous.

```
# machines.py
class Machine(object):
   def __init__(self, name, mac=None, ip=None):
      self.name = name
      self.mac = mac
      self.ip = ip

target = Machine('target', mac='08:00:27:b2:89:86', ip='192.168.56.20')
attacker = Machine('attacker', mac='08:00:27:46:7b:6a', ip='192.168.56.10')
```

Les deux instances créées seront réutilisées plus loin dans le code.

Connexion TCP

Le script suivant réalise un Three-Ways-handshake entre Bob (le client) et Alice (le serveur)

```
#!/usr/bin/python
from random import randint
from scapy.all import *
from machines import attacker as bob
from machines import target as alice

_syn_ip = IP(src=bob.ip, dst=alice.ip)
_syn_tcp = TCP(dport=80, sport=1337, flags="S", seq=1337)
syn = _syn_ip/_syn_tcp
print("SYN :\n")
print(syn.show())

synack = sr1(syn)
print("SYN ACK :\n")
print(synack.show())

_ack_ip = IP(src=bob.ip, dst=alice.ip)
_ack_tcp = TCP(dport=80, sport=1337, flags='A', seq=1338, ack=synack.ack)
ack = _ack_ip/_ack_tcp
print("ACK :\n")
print(ack.show())
send(ack)
```

Exécuté, il affiche le détail des paquets envoyés :

```
SYN :

###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 192.168.56.10
  dst       = 192.168.56.20
  \options   \
###[ TCP ]###
     sport     = 1337
     dport     = http
     seq       = 1337
     ack       = 0
     dataofs   = None
     reserved  = 0
     flags     = S
     window    = 8192
     chksum    = None
     urgptr    = 0
     options   = []

None
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
SYN ACK :

###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 44
  id        = 0
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = 0x495d
  src       = 192.168.56.20
```

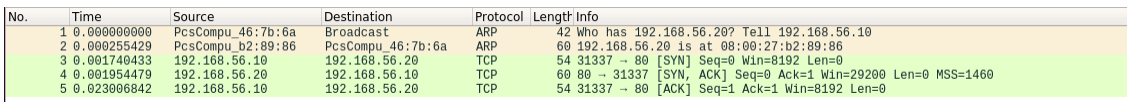
```
  dst       = 192.168.56.10
  \options   \
###[ TCP ]###
     sport     = http
     dport     = 1337
     seq       = 3846541021
     ack       = 1338
     dataofs   = 6
     reserved  = 0
     flags     = SA
     window    = 29200
     chksum    = 0xb5b0
     urgptr    = 0
     options   = [('MSS', 1460)]
###[ Padding ]###
        load      = '\x00\x00'

None
ACK :

###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 192.168.56.10
  dst       = 192.168.56.20
  \options   \
###[ TCP ]###
     sport     = 1337
     dport     = http
     seq       = 1338
     ack       = 1338
     dataofs   = None
     reserved  = 0
     flags     = A
     window    = 8192
     chksum    = None
     urgptr    = 0
     options   = []

None
.
Sent 1 packets.
```

On capture avec wireshark l'échange de paquets.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_46:7b:6a	Broadcast	ARP	42	Who has 192.168.56.20? Tell 192.168.56.10
2	0.000255429	PcsCompu_b2:89:86	PcsCompu_46:7b:6a	ARP	60	192.168.56.20 is at 08:00:27:b2:89:86
3	0.001740433	192.168.56.10	192.168.56.20	TCP	54	31337 → 80 [SYN] Seq=0 Win=8192 Len=0
4	0.001954479	192.168.56.20	192.168.56.10	TCP	60	80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
5	0.023006842	192.168.56.10	192.168.56.20	TCP	54	31337 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0

Notons que le numéro de séquence TCP affiché par wireshark est un numéro *relatif*.

ARP Poisonning.

Un article intéressant expliquant rapidement l'ARP poisonning et surtout fournissant et documentant un script scapy permettant d'effectuer cette attaque. Une bonne inspiration de corrigé : Source