TP2 - Analyse et génération

19 septembre 2014

Dans ce TP, nous allons d'une part utiliser `wireshark` pour analyser une "trace" de connexions réseaux que nous avons enregistrée pour vous sur une machine de test, puis générer "à la main" des paquets à l'aide de `scapy`

1 Analyse : `wireshark`

La machine de test a la configuration suivante :

```
$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:21:70:b4:36:49
          inet adr:193.50.110.76  Bcast:193.50.110.255  Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:249530 errors:0 dropped:12 overruns:0 frame:0
          TX packets:179836 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:177744675 (169.5 MiB)  TX bytes:28469007 (27.1 MiB)
          Interruption:18

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:9555 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9555 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:10158878 (9.6 MiB)  TX bytes:10158878 (9.6 MiB)

$ /sbin/route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
193.50.110.0    *               255.255.255.0   U     0      0        0 eth0
default         193.50.110.254  0.0.0.0         UG    0      0        0 eth0
```

```
$ /usr/sbin/arp
Address                  HWtype  HWaddress           Flags Mask            Iface
193.50.110.254           ether   00:1d:45:2e:54:46   C                     eth0
```

Quelle est l'adresse IP de la machine ?

Quelle est l'adresse de la passerelle par défaut ?

Lancez l'outil `wireshark`, ouvrez le fichier `/net/cremi/sathibau/Reseau/TP2/trace`, on voit apparaître dans la partie du haut une ligne pour chaque trame Ethernet que la carte réseau de la machine a traitée (en émission comme en réception). Lorsqu'une trame y est sélectionnée, le contenu *brut* (*i.e.* octet par octet) apparaît dans la partie du bas, et une version décodée apparaît dans la partie du milieu.

Certaines trames apparaissent en rouge, c'est parce que sur la machine testée, les *checksums* sont calculés par la carte réseau elle-même et sont donc ici faux. Dans *Edit* → *Preferences* → *Protocols* → *UDP*, décochez *Validate the UDP checksum if possible*, faites de même pour TCP.

Notez la présence de la colone *Time* qui permet de regrouper les trames par évènements que l'on va observer. Vous noterez la présence de temps en temps de requêtes ARP qui ne nous concernent pas. Eh oui, le réseau est pollué par les autres machines (et encore, on a appliqué un filtre).

2,28s : Un premier ping

Observez en détails le premier paquet de requête DNS. Utilisez les petits triangles à gauche pour "ouvrir" les différents en-têtes. Lorsque vous cliquez sur une partie d'en-tête, `wireshark` surligne le morceau de trame auquel il correspond, vous pouvez ainsi vérifier comment les champs sont codés octet par octet et comparer aux en-têtes que l'on a vus en cours. Vérifiez ainsi le codage des en-têtes Ethernet, IP, UDP, puis enfin la requête DNS elle-même. Relevez notamment les adresses IP et ports UDP source et destination.

Quelle est l'adresse du serveur DNS ?

Quel est le numéro de port DNS ?

Quel nom de domaine a-t-on demandé à résoudre ?

Observez de manière similaire la réponse DNS. Quel est le résultat de la réponse ?

Vient alors une requête et une réponse ICMP echo (ping). On comprend donc maintenant pourquoi la requête DNS avait été émise. L'adresse MAC de destination utilisée pour cette requête ICMP est la même que pour la requête DNS alors que l'adresse IP de destination est différente, pourquoi ?

Quelle latence obtient-on ?

4,29s : Un deuxième ping

Quelles sont les différences par rapport au premier ping ? Pourquoi ?

7.43s : Une page web

Observons la demande de page web. La requête DNS AAAA est juste une tentative du navigateur d'utiliser IPv6 si possible. Notez qu'avec l'identifiant dans l'en-tête DNS on sait quelle réponse correspond à quelle demande. On peut ensuite observer une connexion TCP, et une fois établie, une demande HTTP passée dedans. Pour lire plus facilement les échanges TCP, vous pouvez cliquer sur un des paquets TCP et utiliser *Analyze -> Follow TCP Stream*. À quoi correspondent les couleurs ?

Quelle est l'URL complète qui avait été demandée ?

Utilisez le bouton *Effacer* pour revenir à la trace complète, et ouvrez cette adresse avec votre propre navigateur Web. On observe une deuxième demande HTTP immédiatement après la première, à quoi est-elle due ? (l'utilisateur ne l'a pas demandée explicitement lui-même).

9.71s : Une deuxième page web

On observe une troisième demande HTTP, qu'a fait l'utilisateur pour cela ?

12.21s : CUPS ?

Comment s'appelle l'imprimante branchée sur le réseau ?

13.7s : Envoyer un mail

Observez la conversation SMTP : dès le mot-clé `STARTTLS`, on n'arrive plus à lire, l'échange s'est effectué de manière cryptée (TLS = *Transport Layer Security*)

15.82s : Lire des mails

Observez la conversation POP3. Quel est le mot de passe de messagerie ? Il existe également des variantes cryptées de POP3.

18.69s : Connection ssh

Observez la conversation SSH. De même, la connexion est crypté et l'on ne peut pas lire le mot de passe utilisé.

2 Génération : `scapy`

Copiez le répertoire `/net/cremi/sathibau/Reseau/TP2`, lancez `make`, trois fenêtres qemu s'ouvrent, au bout d'un certain temps vous obtenez un prompt root dans deux fenêtres (`machine1` et `machine2`), et une sortie de `tcpdump` dans la troisième (`espion`). Ce sont trois machines virtuelles qui sont connectées en réseau, la troisième machine vous permettant

d'observer ce qui passe sur le réseau. Utilisez `netstat -Ainet -a` pour voir quels services (et donc quels ports) sont ouverts.

2.1 Observation

Lancez `scapy` dans `machine1`. Il s'agit en fait d'un simple module Python dont la documentation complète est disponible sur `https://www.secdev.org/projects/scapy/doc/` . Regardez dans `daytime.py` un exemple d'utilisation qui envoie un paquet UDP sur le port `daytime` puis récupère et affiche la réponse, essayez-le en copiant/collant ligne à ligne. Ignorez l'apparition dans `tcpdump` d'un paquet `port 1024 unreachable`. Analysez.

2.2 UDP `echo`

Vous avez pu remarquer que le service `udp echo` est ouvert. En vous inspirant de l'exemple pour `daytime`, essayez de l'utiliser (pour mettre des données dans un datagramme UDP `d`, utilisez `d2 = d/"donnee"`). Ignorez l'éventuel `Padding` dans la réponse, c'est un détail de contrainte d'Ethernet (46 octets au minimum).

2.3 Tester les services TCP

Essayez d'envoyer un datagramme TCP sur le port 21 puis sur le port 7 (dans l'en-tête TCP, ne changez que le port destination, les valeurs par défaut pour le reste iront bien).

Quelle partie de la réponse regarder pour savoir si un port donné est ouvert ?

Écrivez une fonction qui prend en paramètre une adresse IP et qui imprime la liste des ports ouverts. Pour ouvrir un port supplémentaire sur `machine2`, lancez-y `nc -l -p 1234 &` par exemple.

2.4 (Bonus) Une connexion TCP

À vous de jouer. Essayez d'établir (à la main, donc) une connexion vers le service `echo` mais cette fois-ci en TCP. Il faudra donc bien sûr récupérer le numéro de séquence etc.