TP3 - Analyse et génération

1 Analyse : `wireshark`

Dans cet exercice, nous allons utiliser `wireshark` pour analyser une "trace" de connexions réseaux que nous avons enregistrée pour vous sur une machine de test. 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)

$ cat /etc/resolv.conf
nameserver 193.50.111.150
search bordeaux.inria.fr

$ /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 colonne *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).

1.1 Un premier ping (2,28s)

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 ?

1.2 Un deuxième ping (4,29s)

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

1.3 Une page web (7.43s)

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

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

1.4 Une deuxième page web (9.71s)

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

1.5 CUPS (12.21s)

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

1.6 Envoyer un mail (13.7s)

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

1.7 Lire des mails (15.82s)

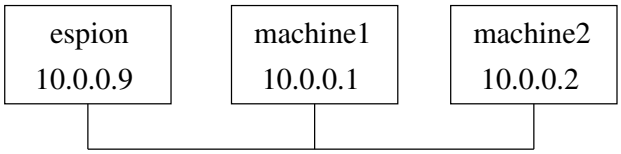
Observez la conversation POP3. Quel est le mot de passe de messagerie ? Il existe également des variantes chiffrées de POP3.

1.8 Connexion SSH (18.69s)

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

2 Génération : `scapy`

Dans cet exercice, nous allons générer "à la main" des paquets à l'aide de `scapy`. Lancez `/net/cremi/sathibau/Reseau/TPscapy/run` , 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` récupéré depuis le site du cours 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 depuis le fichier (sélectionner le texte) vers Scapy (avec le clic milieu ou shift-insert). Ignorez l'apparition dans `tcpdump` d'un paquet `port 12345 unreachable`, il ne se produit que parce que Scapy est fourbe. 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 vers le port 21 puis vers 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 entre 1 et 100. Il s'agit donc de faire la même chose que ce que l'on a fait à la main ci-dessus, dans une boucle for. Pour tester l'absence du bit RST, on peut utiliser `reponse.payload.flags & 4 == 0`

Pour ouvrir un port supplémentaire sur `machine2`, lancez-y `nc -l -p 42 &` par exemple.

2.4 (Bonus) Une connexion TCP

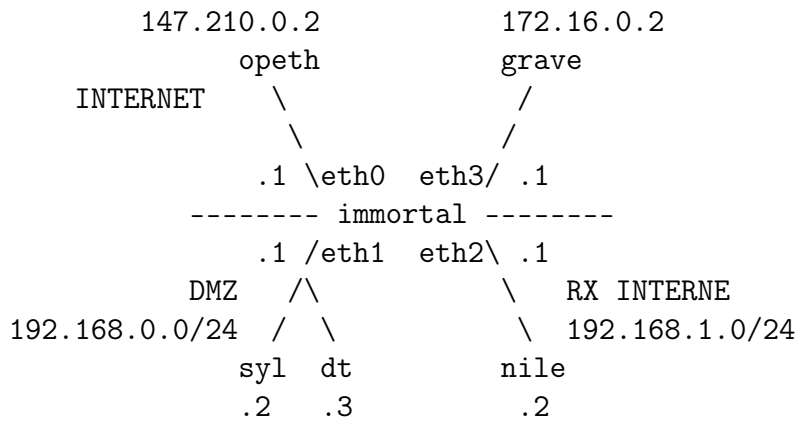
À vous de jouer. Essayez d'établir (à la main, donc) une connexion vers le service `echo` mais cette fois-ci en TCP. Il faudra donc bien sûr récupérer le numéro de séquence etc. Ignorez l'apparition de datagrammes RST dans le tcpdump, ils sont produits par le noyau qui ne comprend pas ce qui se passe, on a ajouté un filtre pour qu'ils ne nous gênent pas.

D'autre exemples :

`http://aurelien.esnard.emi.u-bordeaux.fr/teaching/doku.php?id=admin:tp3`

3 Firewall

Lancez la configuration *QemuNet* suivante :
`/net/ens/qemunet/qemunet.sh -x -s /net/ens/qemunet/demo/dmz.tgz`



Les IP et les tables de routage sont déjà configurées pour vous :-) Par défaut, il n'y a aucun firewall en place, ce qui signifie que tout le monde peut communiquer avec tout le monde sans restriction particulière.

Dans cet exercice, nous jouons le rôle de l'administrateur d'un petit réseau d'entreprise, rélié à Internet par la passerelle *immortal*. Nous allons configurer un firewall sur cette passerelle à l'aide de la commande `iptables`. Un memento dans la section suivante vous donne la syntaxe de cette commande pour vous aider, commencez par y jeter un coup d'œil. D'autre part, rapidement vous aurez besoin d'utiliser `tcpdump` pour vérifier quels paquets parviennent à passer. À noter que `iptables` est en train d'être supplanté par `nftables`, mais dans un futur proche, c'est encore `iptables` que vous verrez en production, et les principes sont les mêmes de toutes façons.

— Au sein d'un réseau d'entreprise, quel différence y-a-t'il entre la DMZ et le réseau interne des employés ?
— Sur *immortal*, positionnez la politique par défaut à DROP :
```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```
— Nous venons donc d'activer le firewall sur *immortal*. Plus aucun traffic réseau n'est autorisé vers ou à travers *immortal*. Vérifiez avec ping.

Nous allons maintenant ajouter des règles pour autoriser explicitement seulement certains traffics réseau. Toutes les questions suivantes nécessitent donc de taper des commandes (sur *immortal* uniquement) de la forme : `iptables -A FORWARD ... -j ACCEPT` , cf le memento plus bas pour les détails de syntaxe. Si vous avez fait une erreur, vous pouvez supprimer une ligne en reprenant la même commande et en remplaçant `-A` par `-D`

— Autoriser le ping (càd le protocole icmp) du réseau Interne vers Internet, sans autoriser l'inverse.
— Faire un test *ping*, constater que cela ne fonctionne pas : il faut effectivemen autoriser la réponse à passer ! Pour simplifier l'autorisation des réponses de manière générale, on peut utiliser
`iptables -A FORWARD -m state - -state RELATED,ESTABLISHED -j ACCEPT`
qui laissera passer tout ce qui est identifié comme réponse à ce que l'on a déjà laissé passer auparavant (c'est donc un "catch-all" pour les réponses).
— Autorisez l'accès au web depuis les machines du réseau interne. Faire un test avec *wget*.
— Autorisez *grave* à accèder au serveur ssh (port 22) de *dt*. Faire un test avec le compte `toto` (mot de passe `toto`).
— Autorisez l'accès depuis n'importe où vers le serveur web de syl (port 80). Faites un test avec wget.
— Depuis opeth et grave, testez votre firewall sur la DMZ avec nmap !

Memento Firewall

La configuration du firewall se base sur la table "filter" de la commande `iptables`. Elle est subdivisée en 3 chaînes, notée <CHAIN> :
— INPUT : tout ce qui est à destination de la machine elle-même ; cela ne concerne donc pas les paquets qui seront relayés
— OUTPUT : tout ce qui est émis par la machine elle-même ; cela ne concerne donc pas les paquets qui seront relayés
— FORWARD : tout ce qui ne fait que traverser la machine (donc les paquets relayés).

Voici la syntaxe de principales commandes `iptables` :
— Pour afficher les règles de la table filter : `iptables -L` , on peut y ajouter l'option `-v` pour voir les statistiques, pour vérifier par exemple qu'une règle attrape bien des paquets.
— Pour effacer toutes les règles ajoutées : `iptables -F`
— Pour chaque règle que l'on ajoute, trois actions sont possibles (notée <ACTION>) :
  — ACCEPT : on accepte ;
  — REJECT : on rejette poliment (réponse d'erreur envoyé à l'émetteur) ;
  — DROP : on jette à la poubelle (pas de réponse d'erreur).
— Pour modifier la politique par défaut du firewall :
`iptables -P <CHAIN> <ACTION>`
— Pour ajouter une nouvelle règle à une chaîne du firewall :
`iptables -A <CHAIN> <SRC> <DST> <...> -j <ACTION>`
  — avec <SRC> des indications éventuelles sur la provenance des paquets IP, comme par exemple `-i eth0` ou `-s 192.168.0.0/24` ;
  — avec <DST> des indications éventuelles sur la destination des paquets IP, comme par exemple : `-o eth1` ou `-d 147.210.0.0/24` ;

— avec <...> des infos complémentaires par exemple sur la nature du protocole `-p icmp` , `-p tcp` , ou `-p udp` avec après éventuellement des précisions spécifiques à ces protocoles (`--dport 80` pour TCP) ou encore sur l'état `-m state --state NEW`, ...

— sur l'état, il est notamment utile d'employer `"-m state --state RELATED,ESTABLISHED"` pour identifier les paquets qui sont une réponse à quelque chose que l'on a déjà laissé passer : une réponse à un ping, une réponse à une demande de connexion TCP, etc.

— Pour supprimer une règle, on peut utiliser `iptables -D <CHAIN>` en mettant derrière soit un numéro de règle (lisible avec `iptables -L --line-numbers`, soit la règle elle-même (i.e. remplacer `-A` par `-D`).

— Pour supprimer toutes les règles, on peut utiliser `iptables -F <CHAIN>`

Pour plus d'info, consulter le manuel : `man iptables` et `man iptables-extensions`

4 Bonus : Et le réseau sous windows ?

C'est (presque) pareil ! Vous pouvez lancer une machine virtuelle windows, dans le menu CREMI, Machine virtuelle Windows 7 (ou bien taper
`bash /addons/virtual_machines/windows-7-amd64-entreprise/starter`
), et cliquez sur démarrer. En utilisant le menu Démarrer, taper `cmd` dans le champ de recherche et valider pour obtenir un prompt. Les commandes s'appellent `ipconfig`, `route print`, `netstat`. Le format est un peu différent mais les informations sont les mêmes (pas besoin de prendre du temps à regarder les détails, donc).