R401
Infrastructures de sécurité
Travaux pratiques

Sami Evangelista

IUT de Villetaneuse

Département Réseaux et Télécommunications

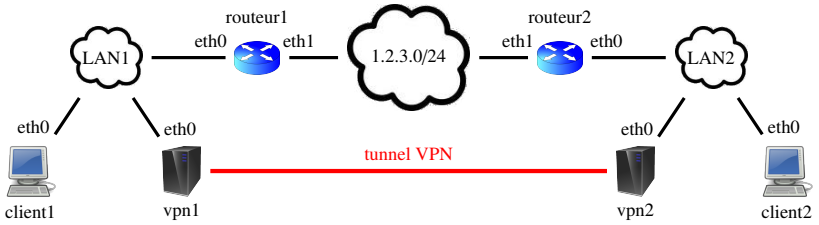
2023–2024

http://www.lipn.univ-paris13.fr/~evangelista/cours/R401

Table des matières

TP 1 — Tunnels VPN

L'objectif de ce TP à réaliser avec marionnet est de réaliser l'architecture de la figure ci-dessous :



On imagine que les réseaux locaux LAN1 et LAN2 sont deux réseaux d'une même université interconnectés par un tunnel VPN entre les deux hôtes vpn1 et vpn2. Ce tunnel passe par le réseau public 1.2.3.0/24. Les hôtes routeur1 et routeur2, en plus d'interconnecter leurs réseaux au réseau public, jouent également le rôle de passerelle NAT. Nous utiliserons le pare-feu iptables pour mettre en place le NAT et l'outil openvpn pour créer notre tunnel. Nous étudierons les possibilités offertes par openvpn pour mettre en place des tunnels de niveau 2 ou de niveau 3, et pour sécuriser le tunnel à l'aide d'un chiffrement symétrique.

Exercice 1 — Travail préliminaire

Aucun compte-rendu n'est demandé pour cet exercice.

I 1.1 Téléchargez le projet à l'URL indiquée suivante :
`https://lipn.univ-paris13.fr/~evangelista/cours/R401/vpn.mar`
Seules les machines vpn1 et vpn2 sont dans ce réseau. Le paquet openvpn a été installé sur celles-ci.

I 1.2 Ouvrez le projet dans marionnet.

I 1.3 Ajoutez tous les équipements pour obtenir le réseau de la figure en introduction. Utilisez des ordinateurs pour les routeurs et des switchs pour interconnecter les machines d'un même LAN. Pour les deux routeurs, respectez bien le câblage de la figure : eth1 mène au réseau public et eth0 au réseau privé.

I 1.4 Démarrez tous les équipements et ouvrez une session root sur chaque ordinateur.

Exercice 2 — Configuration des interfaces

Dans l'exercice suivant nous mettrons en place un tunnel de niveau 2, ce qui signifie que les deux réseaux locaux LAN1 et LAN2 seront sur un même réseau IP. On choisit l'adresse 10.0.0.0/8 et on attribue les IP suivantes aux interfaces :

	hôte	interface	IP
LAN1	client1	eth0	10.1.0.1
	routeur1	eth0	10.1.0.254
		eth1	1.2.3.1
	vpn1	eth0	10.1.0.100

	hôte	interface	IP
LAN2	client2	eth0	10.2.0.1
	routeur2	eth0	10.2.0.254
		eth1	1.2.3.2
	vpn2	eth0	10.2.0.100

On attribuera les IP aux interfaces en modifiant le fichier des interfaces (`/etc/network/interfaces`) plutôt qu'en utilisant `ip` ou `ifconfig`. La syntaxe pour déclarer une interface activée statiquement est la suivante :

```
iface <nom-de-l-interface> inet static
   address <ip-de-l'interface>
   netmask <masque-de-reseau>
   gateway <ip-de-la-passerelle-par-défaut>  # ligne à ajouter seulement pour vpn1 et vpn2
```

I 2.1 Éditez le fichier des interfaces sur les 6 machines pour attribuer aux 8 interfaces les IP indiquées dans le tableau.

I 2.2 Activez toutes les interfaces avec la commande `ifup`.

I 2.3 Sur les deux routeurs, activez le routage en modifiant avec `sysctl` la valeur du paramètre `net.ipv4.ip_forward`.

I 2.4 Testez en échangeant des messages ping entre hôtes d'un même réseau.

Exercice 3 — Mise en place de la translation d'adresse

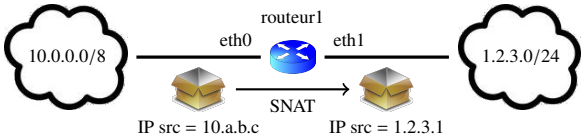
Passons maintenant à la mise en place des règles iptables de translation d'adresse. Nous allons dans un premier temps faire en sorte que les routeurs cachent l'adresse IP source (privée) des paquets sortant du réseau. Il faut pour cela ajouter une règle iptables dans la chaîne `POSTROUTING` de la table `nat`.

I 3.1 Sur routeur1 activez le NAT pour les paquets sortant du réseau :

```
$ iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 1.2.3.1
```

I 3.2 Faites de même sur routeur2.

Cette commande peut se comprendre comme ceci : *aprés* avoir décidé de router un paquet (`-A POSTROUTING`) sur l'interface eth1 (`-o eth1`, `o` pour output) on modifie l'IP source (`-j SNAT`) du paquet (qui est donc une adresse privée dans le réseau 10.0.0.0/8) par l'IP publique du réseau (`--to-source 1.2.3.1`). Ceci est résumé par la figure ci-dessous :



I 3.3 Faites les tests suivants pour vérifier la translation :
- *I 3.3.1* ping de vpn1 vers routeur2
- *I 3.3.2* ping de vpn2 vers routeur1

I 3.4 Pour vérifier que routeur1 effectue bien la translation, refaites le test I 3.3.1 tout en capturant les trames sur routeur2. Vérifiez que le message ICMP reçu par routeur2 a bien pour source 1.2.3.1 et non 10.1.0.100.

I 3.5 Faites de même avec le test I 3.3.2 pour vérifier que la translation d'adresse fonctionne sur routeur2.

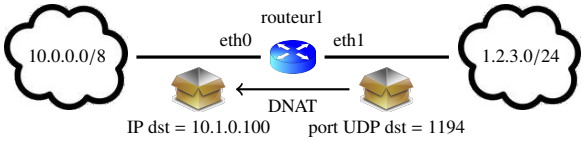
Supposons qu'un hôte veuille contacter le serveur vpn1 depuis le réseau public. Il connaît uniquement l'IP publique de vpn1 qui est 1.2.3.1. S'il envoie un paquet à cette adresse sur le port UDP 1194 (port par défaut de openvpn), c'est routeur1 qui traitera ce paquet. Vu que ce port est fermé sur routeur1, le paquet sera rejeté. Pour que les serveurs VPN puissent être contactés depuis le réseau public, il faut donc que tous les paquets reçus par le routeur depuis le réseau public et destinés au port UDP 1194 soient redirigés vers le serveur VPN. Cela revient à rajouter une ligne statique dans la table NAT des routeurs. Il faut pour cela ajouter une règle iptables dans la chaîne `PREROUTING` de la table `nat`.

I 3.6 Sur routeur1, exécutez la commande ci-dessous :

```
$ iptables -t nat -A PREROUTING -i eth1 -p udp --dport 1194 -j DNAT --to 10.1.0.100
```

I 3.7 Faites de même sur routeur2.

Cette commande peut se comprendre comme ceci : *avant* de router un paquet (`-A PREROUTING`) arrivé sur l'interface eth1 (`-i eth1`, `i` pour input) et contenant un paquet UDP (`-p udp`) destiné au port 1194 (`--dport 1194`) on modifie l'IP destination (`-j DNAT`) du paquet (qui est donc l'adresse publique 1.2.3.1) par l'IP privée du serveur VPN (`--to 10.1.0.100`). Ceci est résumé par la figure ci-dessous :



Nous allons maintenant utiliser nmap sur routeur1 pour vérifier que le port UDP 1194 de vpn2 est bien accessible depuis le réseau public. Si le paquet envoyé par nmap est bien redirigé par routeur2 vers vpn2, alors nmap devrait afficher `open | filtered`. En effet, le paquet envoyé par nmap n'est pas un paquet openvpn correct ce qui fait qu'il est ignoré par vpn2 et que nmap ne reçoit pas de réponse. Par contre, si la translation n'est pas correctement effectuée, routeur2 répondra par un paquet ICMP indiquant que le port est fermé. nmap affichera alors l'état `closed`.

I 3.8 Sur vpn2, lancez le serveur openvpn (nous verrons plus tard l'utilité de `--dev tap1`) :

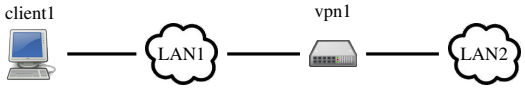
```
$ openvpn --dev tap1
```

I 3.9 Sur routeur1, lancez, avec nmap, un scan UDP (`-sU`) vers le port 1194 de vpn2.

I 3.10 Sur vpn2, arrêtez le serveur openvpn.

Exercice 4 — Création d'un tunnel VPN de niveau 2

On considère dans cet exercice que le tunnel VPN est de niveau 2, c'est-à-dire que vpn1 se comporte comme un switch pour interconnecter le LAN1 au LAN2. Vu du client1, le réseau logique est donc le suivant :



Sous linux, un pont est une interface virtuelle nommée `brX` (avec `X` un numéro). De plus, le tunnel doit aussi être associé à une interface virtuelle. Pour un tunnel de niveau 2, on doit nommer cette interface `tapX` (avec `X` un numéro).

I 4.1 Éditez le fichier `/etc/network/interfaces` de vpn1 pour ajouter une interface `br0`.

```
iface br0 inet static
   bridge_ports eth0 tap1  # crée un pont entre les interfaces eth0 et tap1 (le tunnel)
   address 10.1.0.100
   netmask 255.0.0.0
   gateway 10.1.0.254
```

I 4.2 Faites de même sur vpn2.

Il reste maintenant à ouvrir le tunnel et à activer les ponts.

I 4.3 Lancez le serveur openvpn sur vpn1 et vpn2 (`Y` = numéro de l'autre serveur) :

```
$ openvpn --remote 1.2.3.Y --dev tap1 &
```

L'option `--remote` permet de spécifier l'IP du serveur VPN à l'autre bout du tunnel. L'option `--dev` permet d'associer une nouvelle interface virtuelle `tap1` au tunnel. N'oubliez pas l'esperluette pour que le terminal vous rende la main.

Si le tunnel est bien ouvert vous devriez voir un message similaire à celui-ci dans le terminal de vpn1 :

```
Mon Feb  6 11:21:16 2023 Peer Connection Initiated with [AF_INET]1.2.3.2:1194
Mon Feb  6 11:21:17 2023 Initialization Sequence Completed
```

I 4.4 Activez les interfaces `br0` de vpn1 et vpn2.

I 4.5 Vérifiez que client1 peut bien envoyer des ping vers client2.

Si le test précédent fonctionne, alors la configuration est correcte : le tunnel a bien été créé et les serveurs VPN agissent comme des ponts entre leur LAN (via eth0) et le tunnel (via tap1). Par contre, notre tunnel a un intérêt très limité puisque les paquets qui circulent dessus ne sont pas chiffrés. Nous allons le vérifier avec un simple ping contenant le mot `salut`.

I 4.6 Tout en capturant les paquets sur routeur2, lancez depuis client1 une demande d'écho vers client2 :

```
$ ping -c1 -p 73616C7574 10.2.0.1  # 73616C7574 = codage ASCII de salut en hexadécimal
```

I 4.7 Vérifiez que le mot `salut` apparaît bien dans le panneau inférieur du message ICMP.

Nous allons faire d'autres tests pour bien observer le fonctionnement des tunnels de niveau 2.

I 4.8 Refaites le test I 4.6 mais en capturant cette fois-ci les paquets sur client2.

Q 4.1 À qui appartient l'adresse MAC source de la trame qui contient le message ICMP reçu par client2 ? Pourquoi n'est-ce pas celle de vpn2 (puisque c'est vpn2 qui a relayé le message ICMP vers client2) ?

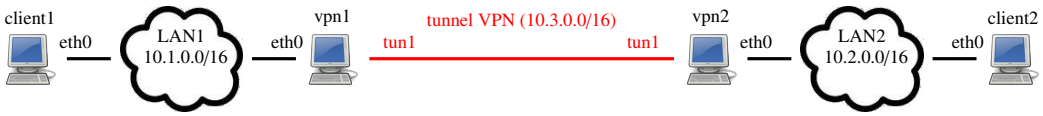
I 4.9 Tout en capturant les trames sur client2, envoyez depuis client1 une demande d'écho ICMP vers 10.1.0.200. Cette IP n'existe pas sur le réseau. L'unique but de ce test est de provoquer l'envoi par client1 de paquets ARP pour connaître l'adresse MAC de 10.1.0.200.

Q 4.2 La machine client2 a-t-elle reçu les paquets ARP envoyés par client1 ? Pourquoi ?

I 4.10 Sur vpn1 et vpn2 : arrêtez le serveur openvpn (`fg` pour remettre le processus `openvpn` au premier plan puis Ctrl+C) puis désactivez l'interface `br0` avec `ifdown`.

Exercice 5 — Création d'un tunnel VPN de niveau 3

Nous allons maintenant transformer notre tunnel en un tunnel de niveau 3. Les serveurs VPN vont maintenant se comporter comme des routeurs pour interconnecter le LAN1 et le LAN2. Ceux-ci correspondront donc à des réseaux IP différents. Nous allons pour cela modifier les masques et les passer en /16. Le LAN1 sera donc associé au réseau 10.1.0.0/16 et le LAN2 sera associé au réseau 10.2.0.0/16. Dans cette configuration le tunnel est lui aussi associé à un réseau IP pour lequel nous choisissons l'adresse 10.3.0.0/16. L'architecture logique de notre réseau est donc la suivante :



Comme dans l'exercice précédent, les serveurs VPN associent une interface virtuelle au tunnel. Dans le cas d'un tunnel de niveau 3, l'interface s'appele `tunX` (avec `X` un numéro). Dans notre cas, nous l'appelerons `tun1` comme sur le schéma. Les interfaces `tun1` des deux serveurs VPN ne doivent pas être déclarées dans le fichier des interfaces. C'est à l'ouverture du tunnel qu'elles seront automatiquement activées, comme dans l'exercice précédent avec l'interface `tap1`.

I 5.1 Désactivez l'interface `eth0` sur les 6 machines.
I 5.2 Modifiez les fichiers des interfaces des 6 machines pour que les interfaces `eth0` aient le masque /16.
I 5.3 Réactivez l'interface `eth0` sur les 6 machines.
I 5.4 Lancez le serveur openvpn sur vpn1 et vpn2 (`X` = numéro du serveur, `Y` = numéro de l'autre serveur) :

```
# openvpn --remote 1.2.3.Y --dev tun1 --ifconfig 10.3.0.X 10.3.0.Y &
```

Nos serveurs VPN sont maintenant des routeurs. Il faut donc mettre en place le routage entre les réseaux privés.

I 5.5 Sur vpn1 et vpn2 : activez le routage avec `sysctl`.
I 5.6 Ajoutez une route sur les clients et les serveurs VPN (une route par machine) pour obtenir le réseau de la figure en introduction de cet exercice. Indication : afficher les tables de routage pour voir les routes manquantes.
I 5.7 Enfin, vérifiez que client1 peut bien envoyer des ping vers client2.

Si le test précédent fonctionne, alors notre tunnel fonctionne bien. Comme dans l'exercice précédent, nous allons faire d'autres tests pour bien observer le fonctionnement des tunnels de niveau 3 et la différence avec les tunnels de niveau 2.

I 5.8 Refaites le test I 5.7 mais en capturant cette fois-ci les paquets sur client2.

Q 5.1 À qui appartient l'adresse MAC source de la trame qui contient le message ICMP reçu par client2 ? Pourquoi ?

I 5.9 Tout en capturant les trames sur client2, envoyez depuis client1 une demande d'écho ICMP vers 10.1.0.200.

Q 5.2 La machine client2 a-t-elle reçu des paquets ARP envoyés par client1 ? Pourquoi ?
Q 5.3 En conclusion, quels sont les avantages et inconvénients des deux types de tunnel (niveau 2 et niveau 3) ?

I 5.10 Sur vpn1 et vpn2 : arrêtez le serveur openvpn (`fg` puis Ctrl+C).

Exercice 6 — Mise en place d'un chiffrement symétrique

Les tunnels mis en place dans les exercices précédents n'étaient pas chiffrés : les données échangées entre les clients circulent en clair sur le réseau public. Dans cet exercice nous allons mettre en place un chiffrement symétrique. Nous allons donc créer une clé de chiffrement qui sera la même sur les deux serveurs.

I 6.1 Sur vpn1, déplacez vous dans le répertoire `/etc/openvpn`, puis générez une clé dans le fichier `static.key` :

```
# openvpn --genkey --secret static.key
```

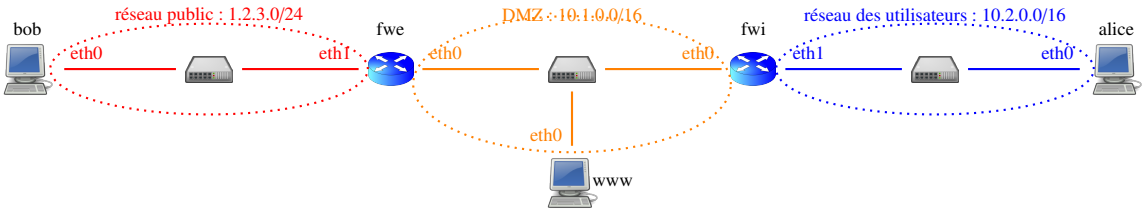
I 6.2 Sur vpn2, copiez-collez le contenu du fichier `static.key` de vpn1 dans le fichier `/etc/openvpn/static.key`.
I 6.3 Relancez les commandes de l'instruction I 5.4 pour réouvrir le tunnel mais en rajoutant l'option `--secret static.key` pour activer le chiffrement.

En fermant le tunnel à la fin de l'exercice précédent, les routes passant par l'interface `tun1` avaient été retirées de la table de routage de vpn1 et de vpn2. Il faut donc les rajouter.

I 6.4 Sur vpn1 et vpn2, ajoutez la route manquante.
I 6.5 Vérifiez que client1 peut bien envoyer un ping à client2.
I 6.6 Refaites le test I 4.6 pour vérifier que le mot `salut` n'apparaît plus dans le message ICMP capturé.

TP 2 — Réalisation d'une DMZ avec iptables

L'objectif du TP est de réaliser avec marionnet un réseau local décomposé en deux sous-réseaux (une DMZ de serveurs (10.1.0.0/16) et un réseau d'utilisateurs (10.2.0.0/16) composé de postes clients) relié à un réseau public (1.2.3.0/24) :



On supposera pour simplifier qu'il n'y a qu'un seul serveur (www) et qu'un seul poste client (alice). De même, le réseau public se réduit à un seul hôte (bob). Deux routeurs/pare-feux protègent la DMZ et le réseau des utilisateurs et les connecte au réseau public : fwe et fwi. Le pare-feu externe (fwe) protège le réseau local du réseau public et effectue de la translation d'adresse. Le pare-feu interne (fwi) protège le réseau des utilisateurs. Voici le cahier des charges.

(C1) Sécurisation des routeurs Il est possible de se connecter en SSH via le port TCP/22 sur fwe et fwi, depuis un hôte du réseau 10.2.0.0/16 uniquement. Sur les routeurs, tout autre trafic entrant ou sortant doit être bloqué.
(C2) Réseau des utilisateurs Les postes client du réseau 10.2.0.0/16 peuvent communiquer sans restriction avec le réseau public et avec les serveurs de la DMZ. Par contre, tout échange initié depuis l'extérieur du réseau des utilisateurs est interdit. Par exemple, www ne doit pas pouvoir envoyer une demande d'écho ICMP à alice ou ouvrir une connexion ssh sur alice. Même chose pour bob.
(C3) Translation d'adresses Le pare-feu fwe fait de la translation d'adresses entre le réseau local (10.0.0.0/8) et le réseau public 1.2.3.0/24. Son adresse publique associée à eth1 est 1.2.3.1.
(C4) Le serveur www C'est un serveur web accessible par tous sur le port TCP/80. Il aussi nécessaire qu'il puisse interroger des serveurs DNS (UDP/53) du réseau public. En dehors de ces deux cas de figure (et aussi du cas d'ICMP, voir C5), tout autre paquet envoyé par www sur le réseau public doit être bloqué par fwe.
(C5) ICMP C'est le serveur www qui répond à tous les messages ICMP envoyés depuis le réseau public. Par exemple, si bob envoie une demande d'écho ICMP vers 1.2.3.1 la demande doit être redirigée vers www.
(C6) Journalisation Toute nouvelle communication interdite initiée depuis le réseau public (p.ex., une tentative de connexion SSH de bob sur 1.2.3.1) doit être journalisée par fwe.
(C7) Blocage des attaques par force brute Le routeur fwe doit laisser entrer au maximum 10 connexions sur le port TCP/80 par seconde.

Les contraintes 5, 6 et 7 sont optionnelles et seront comptabilisées en bonus dans la note de compte-rendu.

Indications

— Dans un premier temps, attribuez les IP suivantes aux interfaces :

Réseau public

Hôte	Int.	IP
fwe	eth1	1.2.3.1
bob	eth0	1.2.3.2

DMZ

Hôte	Int.	IP
www	eth0	10.1.0.100
fwi	eth0	10.1.0.253
fwe	eth0	10.1.0.254

Réseau des utilisateurs

Hôte	Int.	IP
alice	eth0	10.2.0.1
fwi	eth1	10.2.0.254

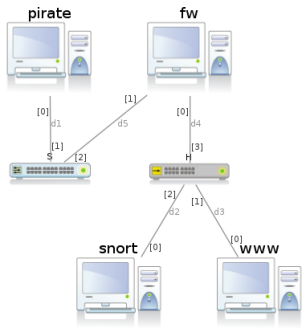
— Ajoutez ensuite des routes :
  — Sur fwe : une route vers 10.2.0.0/16.
  — Sur fwi : une route par défaut.
  — Sur www : une route par défaut et une route vers 10.2.0.0/16.
  — Sur alice : une route par défaut.

  Une fois les routes ajoutées, les communications seront possibles entre hôtes de la DMZ et du réseau des utilisateurs. Le ping vers 1.2.3.1 devrait aussi fonctionner pour tous les hôtes.
— Sur fwe et fwi, écrivez toutes vos commandes iptables dans un script `fw.sh`. Mettez en début de script des commandes iptables permettant de vider la table filter (et nat pour fwe) et de fixer les politiques par défaut.
— Ajoutez progressivement dans vos scripts les règles permettant de répondre aux contraintes du cahier des charges.

Compte-rendu Votre compte-rendu devra contenir :
— les contenu des fichiers `fw.sh` des deux pare-feux (Commentez vos scripts en indiquant les numéros des contraintes du cahier des charge en lien avec les instructions iptables.);
— et des copies d'écran et explications des tests effectués permettant de vérifier que le cahier des charges est respecté : ce qui est autorisé fonctionne bien et, tout aussi important voire plus, ce qui est interdit ne fonctionne pas.

TP 3 — IDS et IPS

L'objectif de ce TP est de se familiariser avec deux logiciels libres pouvant être utilisés comme IDS et IPS : snort et fail2ban ; et de mettre en place une solution de filtrage dynamique utilisant ces deux outils. On travaillera sur le réseau ci-dessous. Nous partirons d'un projet marionnet sur lequel tous les paquets nécessaires ont été installés.



Exercice 1 — Configuration des interfaces et routes

Aucun compte-rendu n'est demandé pour cet exercice.

I 1.1 Téléchargez le projet à l'adresse `https://lipn.univ-paris13.fr/~evangelista/cours/R401/ids.mar`.
I 1.2 Ouvrez le projet dans marionnet et démarrez tous les équipements.
I 1.3 Attribuez des IP à fw(eth0), snort et www sur le réseau 10.0.0.0/24.
I 1.4 Attribuez des IP à pirate et fw(eth1) sur le réseau 1.2.3.0/24.
I 1.5 Sur pirate, www et snort : ajoutez une route par défaut.
I 1.6 Sur fw : modifiez le paramètre système `net.ipv4.ip_forward` pour que fw accepte de router les paquets.
I 1.7 Vérifiez que les pings passent entre les hôtes pirate et www et entre pirate et snort.

Exercice 2 — Prise en main de snort

Nous allons tout d'abord configurer le module de sortie de snort pour qu'il émette des alertes vers le serveur rsyslog local. (Le service rsyslog traite des messages de journal générés par les processus et les stocke dans des fichiers ou les envoie à d'autres serveurs rsyslog. Il sera étudié en détails en troisième année de BUT.) Nous allons pour cela éditer le fichier de configuration `/etc/snort/snort.conf`.

I 2.1 Décommentez la ligne commençant par `# output alert_syslog` dans le fichier `snort.conf`.

Il faut aussi indiquer à snort le réseau qu'il va surveiller.

I 2.2 Dans `/etc/snort/snort.debian.conf`, modifiez la valeur du paramètre `DEBIAN_SNORT_HOME_NET`.

On peut maintenant démarrer les services et faire un premier test de détection d'attaque.

I 2.3 Démarrez les services snort et rsyslog sur snort.
I 2.4 Sur pirate : envoyez, avec nmap, un balayage XMAS vers le port 20 de www. (snort est installé avec de nombreuses règles qui permettent, entre autres, de détecter certains balayages nmap.)
I 2.5 Vérifiez que l'alerte est présente dans le fichier `/var/log/auth.log` : le mot `XMAS` devrait y apparaître.

Le répertoire `/etc/snort/rules` contient les fichiers de signatures données sous la forme de règles snort. Pour tester les règles nous allons utiliser l'outil nc qui permet, entre autres, d'établir une connexion TCP avec un serveur. À titre d'exemple nous allons prendre la règle ci-dessous qui correspond à une attaque sur un serveur Web via le langage PHP. Elle est déjà présente dans le fichier `web-php.rules`.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS ( \
   msg:"WEB-PHP PhpGedView PGV base directory manipulation"; \
   flow:to_server,established; \
   uricontent:"_conf.php"; nocase; \
   content:"PGV_BASE_DIRECTORY"; nocase; \
   reference:bugtraq,9368; classtype:web-application-attack; sid:2926; rev:1; \
)
```

Cette règle, comme beaucoup d'autres, utilise certaines variables (p.ex., `$EXTERNAL_NET`, `$HTTP_SERVERS`) qui, par défaut, valent `any`.

Les lignes 1 et 3 indiquent que seuls les paquets venant de l'extérieur (`$EXTERNAL_NET`), destinés à un de nos serveurs web (`$HTTP_SERVERS`) et faisant partie d'une connexion TCP *établie* sont examinés pour cette règle.

Les lignes 4 et 5 indiquent que les chaînes de caractères `_conf.php`. et `PGV_BASE_DIRECTORY` doivent être présentes dans l'URI de la ressource demandée et dans la charge du paquet respectivement. Dans les deux cas la recherche est insensible à la casse (option `nocase;`).

Pour tester cette règle nous allons d'abord écrire la requête HTTP suivante dans un fichier :

```
GET /chemin/vers/rep/_conf.php HTTP/1.1
file: blablaPGV_bASE_DIRECTORYblabla

```

I 2.6 Éditez le fichier `req.txt` pour avoir le contenu ci-dessus. (N'oubliez pas la ligne vide en fin de fichier pour que la requête soit bien formée.)

I 2.7 Sur www, démarrez un serveur TCP sur le port 80 :

```
$ nc -l -p 80
```

I 2.8 Sur pirate, envoyez la requête sur le port 80 de www :

```
$ cat req.txt | nc @ip-de-www 80
```

I 2.9 Vérifiez que le journal de snort contient bien une nouvelle ligne contenant le message `Web-PHP ....`

I 2.10 En suivant la même procédure, générez une alerte pour l'identifiant de signature 100000691.

Exercice 3 — Prise en main de fail2ban

fail2ban est un outil pouvant servir à la fois d'IDS et d'IPS. C'est un logiciel libre écrit en python. Il examine périodiquement les journaux, recherche dans ceux-ci des *motifs* particuliers (p.ex., la chaîne `authentification failed`) et, si la recherche est fructueuse, exécute une *action* comme, par exemple, envoyer un mail (fonction d'IDS), bloquer une adresse IP ou un port (fonction d'IPS). Mais, plus généralement, l'action peut-être n'importe quelle commande. L'application d'une action particulière quand un motif est trouvé est appelé *bannissement* par fail2ban.

Test de l'outil Pour nous familiariser avec l'outil nous allons voir comment il peut automatiquement bloquer l'IP d'un hôte qui tente successivement plusieurs connexions SSH avec un login ou un mot de passe incorrect.

I 3.1 Arrêtez le service fail2ban sur fw.

I 3.2 Démarrez les service ssh et rsyslog sur fw.

I 3.3 Démarrez le service fail2ban sur fw.

I 3.4 Affichez de nouveau le contenu de la table filter sur fw.

Une nouvelle chaîne (`fail2ban-ssh`) devrait apparaître. On voit que la chaîne INPUT y redirige les paquets ssh (i.e., avec un port destination de 22). Cette nouvelle chaîne contient une unique règle avec l'action RETURN, qui a pour effet de sortir de la chaîne (et donc de revenir dans la chaîne INPUT). Elle n'a donc, pour l'instant, aucun effet.

I 3.5 Sur pirate : lancez successivement 6 tentatives de connexion ssh sur fw avec un login ou un mot de passe incorrect. Lors de la $6^{\text{ème}}$ tentative, ssh devrait se bloquer car fail2ban aura alors bloqué l'adresse IP de pirate.

I 3.6 Vérifiez, sur fw, qu'une règle iptables a bien été ajoutée dans la chaîne créée par fail2ban.

I 3.7 Dans 10 minutes vous vérifierez que la règle a été retirée par fail2ban.

Fichiers de configurations Le répertoire `/etc/fail2ban` contient (entre autres) les fichiers et répertoires suivants :

— `filter.d` — Chaque fichier de ce répertoire contient les motifs à chercher pour une application et un type d'erreur particuliers. Par exemple, `sshd.conf` contient des motifs indiquant une erreur d'authentification SSH.

— `action.d` — Ce répertoire contient les définitions des actions que fail2ban peut effectuer. Par exemple `iptables.conf` contient des règles iptables à exécuter dans différents contextes.

— `jail.conf` — Ce fichier contient la définition des *geôles*. Une geôle décrit les opérations de bannissement effectuées par fail2ban pour un type d'application. Une geôle peut être vue comme un triplet (fichier, motifs, action) qui s'interprète ainsi : *Rechercher dans le fichier des lignes ayant un des motifs précisés. Si un motif est trouvé, l'action est appliquée.*

Fonctionnement des fichiers Nous allons regarder les fichiers de configuration de plus près pour comprendre ce qui s'est passé durant le test.

Tout d'abord le fichier `/etc/fail2ban/jail.conf` contient une section `[DEFAULT]` qui définit un certain nombre de variables qui peuvent être ensuite redéfinies dans les geôles. Cette section contient en particulier la ligne suivante :

```
banaction = iptables-multiport
```

Ce qui signifie que l'action de bannissement par défaut à exécuter quand un motif sera trouvé dans un journal est l'action `iptables-multiport` ("multiport" car fail2ban pourra fermer plusieurs ports simultanément).

On y trouve également, plus bas, la définition de la geôle ssh :

```
1 [ssh]
2 enabled  = true
3 port     = ssh
4 filter   = sshd
5 logpath  = /var/log/auth.log
6 maxretry = 6
```

Nous allons la détailler ligne par ligne :

— l. 1 : le nom de la geôle : `ssh`
— l. 2 : `enabled` étant fixé à `true`, la geôle est activée.
— l. 3 : fixe la valeur du paramètre `port` dans le fichier de l'action (voir plus bas)
— l. 4 : Le fichier `/etc/fail2ban/filter.d/sshd.conf` contient les motifs à rechercher.
— l. 5 : Les motifs doivent être recherchés dans le fichier `/var/log/auth.log`.
— l. 6 : La commande de bannissement est exécutée si fail2ban trouve (dans le fichier `/var/log/auth.log`) 6 occurrences d'un des motifs enumérés dans le fichier `sshd.conf`.

`banaction` n'étant pas redéfini dans cette geôle, c'est l'action par défaut (`iptables-multiport`) qui s'applique.

Le fichier `/etc/fail2ban/action.d/iptables-multiport.conf` décrit cette action. La section `[Definition]` définit certaines variables. Chacune contient la liste des commandes à exécuter dans un contexte particulier et pour chaque geôle utilisant cette action :

— `actionstart` — commandes à exécuter une seule fois au démarrage de fail2ban
— `actionstop` — commandes à exécuter une seule fois à l'arrêt de fail2ban
— `actionban` — commande de bannissement à exécuter quand un motif a été trouvé dans les journaux
— `actionunban` — commandes à exécuter lorsque la période de bannissement d'un hôte a expiré (voir plus bas)

Voici, par exemple, le contenu de la variable `actionstart` :

```
actionstart = iptables -N fail2ban-<name>
              iptables -A fail2ban-<name> -j RETURN
              iptables -I <chain> -p <protocol> -m multiport --dports <port> -j fail2ban-<name>
```

On remarque que les commandes sont paramétrées (avec des noms de paramètres entre chevrons). Les valeurs de ces paramètres sont automatiquement fixées dans les fichiers de geôle ou, à défaut, dans la section `[Init]` du fichier `iptables-multiport.conf`.

Par exemple, si l'on revient sur la déclaration de la geôle `ssh`, on en déduit les paramètres suivants :

— l. 1 : `<name>` est remplacé par `ssh`
— l. 3 : `<port>` est remplacé par `ssh`

Pour les paramètres qui ne sont pas définis dans cette geôle (`protocol`, `chain`), il faudra, comme dit précédemment, regarder la section `[Init]` du fichier `iptables-multiport.conf` pour trouver leurs valeurs.

Les fichiers du répertoire `filter.d`, comme `sshd.conf`, doivent définir une variable `failregex` qui est une liste d'expressions régulières des motifs à rechercher (et dont la reconnaissance entrainera éventuellement un bannissement).

Q 3.1 En consultant les fichiers `jail.conf` et `iptables-multiport.conf` donnez la commande exécutée pour la geôle ssh lors du bannissement d'un hôte d'IP A.B.C.D. Pour cela, remplacez les paramètres par leurs valeurs effectives. Décrire l'effet de cette commande.

Q 3.2 Même question pour la sortie d'un hôte d'IP A.B.C.D de la gêole.

Q 3.3 En analysant les motifs contenus dans le fichier `/etc/fail2ban/filter.d/sshd.conf` et en les comparant au contenu du fichier `/var/log/auth.log`, trouvez dans le journal les lignes qui ont mené au bannissement.

Exercice 4 — Filtrage dynamique

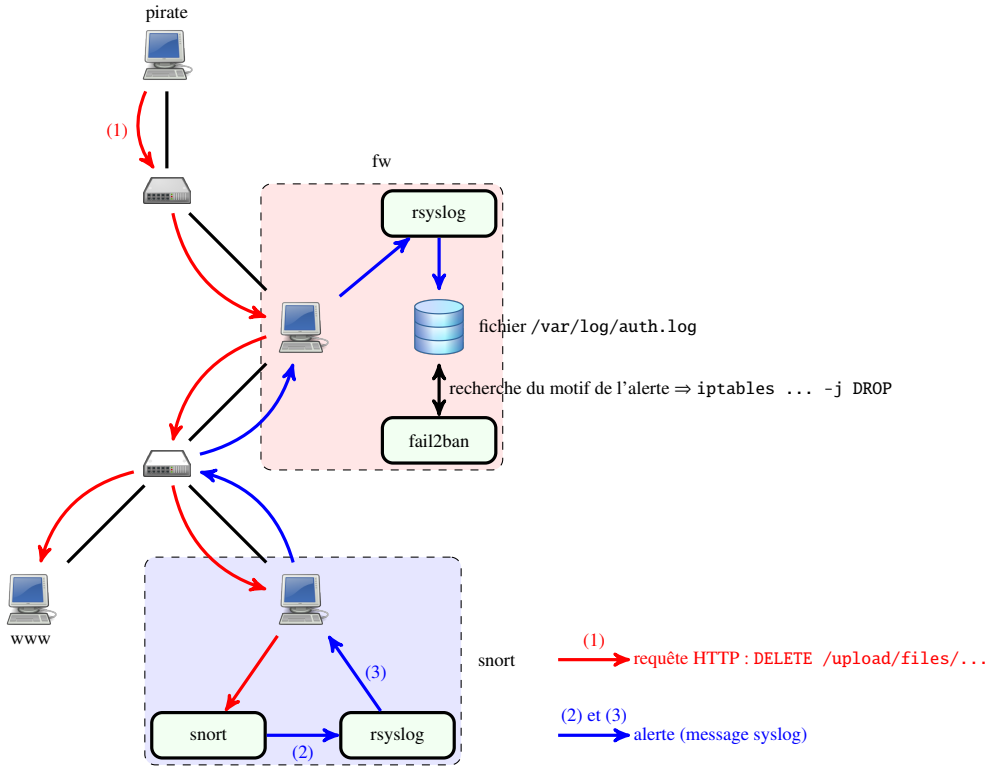
L'objectif de cet exercice est de mettre en œuvre une solution combinant snort, fail2ban et rsyslog pour bloquer dynamiquement des attaques du réseau externe (1.2.3.0/24) sur le serveur web.

Voici une description du problème :

Le protocole HTTP définit la méthode DELETE qui permet de supprimer une ressource. Généralement ces requêtes sont rejetées par le serveur web, ce qui est le cas avec la configuration par défaut d'apache. Notre serveur web contient un répertoire `/upload/files` *dans lequel sont stockés des fichiers déposés par des utilisateurs. Les fichiers de ce répertoire sont parfois ciblés par des requêtes DELETE. Même si apache les rejette on souhaite tout de même que les attaquants ne puissent plus accéder au serveur web par la suite. Par exemple, si pirate envoie une telle requête vers www, fw refusera par la suite de router vers le réseau interne (celui de snort et www) tout paquet en provenance de pirate et destiné au port 80.*

Pour parvenir à ce résultat nous mettrons en œuvre la solution suivante (voir figure ci-dessous) :

— En présence d'une requête DELETE sur un fichier du répertoire `/upload/files` de www (paquet 1), snort va générer une alerte en envoyant un message au serveur rsyslog local (paquet 2).
— Ce serveur va lui même remonter ce message au serveur rsyslog qui s'exécute sur fw (paquet 3).
— Enfin, fail2ban sera configuré sur fw pour rechercher des messages indiquant une telle attaque dans le fichier de journal adéquat et ajoutera dynamiquement une règle de filtrage.



Pour mettre en œuvre cette solution, nous allons d'abord configurer snort pour qu'il génère une alerte. N'oubliez pas de redémarrer snort, rsyslog ou fail2ban après modification des fichiers de configuration de ces services.

I 4.1 Sur snort : ajoutez dans le fichier `/etc/snort/rules/local.rules` une règle snort permettant de lever une alerte en cas de requête DELETE sur un fichier du répertoire `/upload/files`. Vous utiliserez un message (propriété `msg` de votre règle snort) permettant de décrire précisément l'attaque afin qu'il puisse être facilement reconnu par fail2ban. Vous donnerez à votre signature un identifiant (propriété `sid`) compris entre 10^6 et 10^8. Cet intervalle est celui dans lequel les utilisateurs de snort peuvent choisir leurs identifiants.
I 4.2 Sur pirate : envoyez une requête permettant de générer l'alerte.
I 4.3 Sur snort : vérifiez qu'une alerte a effectivement été générée dans le fichier `/var/log/auth.log`.

Nous allons ensuite configurer rsyslog sur les hôtes snort et fw pour qu'ils s'échangent les alertes générées par snort.

I 4.4 Sur snort : configurez le serveur rsyslog afin qu'il redirige les messages d'alerte vers le serveur rsyslog de fw. Il faut pour cela ajouter la ligne suivante à la fin du fichier `/etc/rsyslog.conf` :

```
auth.*   @10.0.0.254   # à remplacer par l'IP de fw(eth0)
```

I 4.5 Sur fw : activez la réception de messages syslog sur le port UDP/514. Il faut pour cela trouver et décommenter deux lignes dans le fichier `/etc/rsyslog.conf`.
I 4.6 Sur pirate : envoyez de nouveau une requête permettant de générer l'alerte.
I 4.7 Sur fw : vérifiez qu'une alerte est bien présente dans le fichier `/var/log/auth.log`.

Enfin nous allons configurer fail2ban pour qu'il reconnaisse notre alerte dans le fichier de journal et bloque l'IP de l'attaquant.

I 4.8 Sur fw : créez un fichier `/etc/fail2ban/filter.d/delete_files.conf` dont le contenu sera le suivant :

```
[Definition]
failregex = ...
```

Vous remplacerez les points de suspension par une expression régulière permettant de reconnaître les messages du journal signalant l'attaque. L'expression régulière pourra contenir les caractères suivants :
— `.*` qui correspond à n'importe quelle suite de caractères ;
— et `<HOST>` qui correspondra à l'IP recherchée par fail2ban (pour pouvoir la bloquer).
Imaginons par exemple que la ligne suivante corresponde à une alerte envoyée par snort :

```
2024-02-26T12:06  attaque SQL depuis 10.1.0.0 (sid=545)
```

et que l'on souhaite que fail2ban analyse cette alerte pour bloquer ensuite 10.1.0.0. Une expression régulière permettant de reconnaître cette ligne et d'en extraire l'IP pourrait être :

```
[Definition]
failregex = .* attaque SQL depuis <HOST> .*
```

I 4.9 Sur fw : créez une geôle `[delete_files]` dans le fichier `/etc/fail2ban/jail.local` :

```
[delete_files]
enabled  = true
bantime  = -1    # -1 = bannissement perpétuel
chain    = XXX    # à remplaçer par la chaîne iptables adéquate
maxretry = 1    # on bannit à la première occurrence du motif trouvée
logpath  = /var/log/auth.log
port     = 80
filter   = delete_files    # les motifs sont définis dans delete_files.conf
```

I 4.10 Sur pirate : envoyez de nouveau une requête permettant de générer l'alerte.

I 4.11 Sur fw : affichez le contenu des chaînes iptables pour vérifier qu'une nouvelle règle permettant de bloquer pirate est bien présente.

I 4.12 Sur pirate : vérifiez qu'il est maintenant impossible d'accéder au serveur web de www.