

Mathieu BIVERT, Sophie VALENTIN

Configuration et sécurisation de services réseaux
8 février 2013

Professeur : Bruno MARTIN



Table des matières

| | | |
|----------|---|-----------|
| 1 | Scénario | 3 |
| 2 | Topologie | 3 |
| 3 | Mise en place d'une passerelle et accès à Internet | 3 |
| 3.1 | Tests | 3 |
| 4 | Mise à disposition d'un accès distant sur la machine et sécurisation | 4 |
| 4.1 | Serveur Telnet et attaques de <i>sniffing</i> | 4 |
| 4.2 | Serveur SSH et optimisations | 5 |
| 4.3 | Accès distant par VPN | 6 |
| 4.3.1 | Clé partagée | 7 |
| 4.3.2 | Utilisation de clés SSL/TLS | 7 |
| 5 | Configuration d'un serveur web et d'OpenSSL | 7 |
| 5.1 | Installation d'OpenSSL | 7 |
| 5.2 | Installation d'Apache2 | 7 |
| 5.3 | Génération d'un certificat X509 | 8 |
| 5.4 | Passage à HTTPS | 8 |
| 5.5 | Tests | 9 |
| 6 | Service de messagerie électronique et sécurisation | 9 |
| 6.1 | Envoi et transfert d'emails avec Postfix (MTA) | 10 |
| 6.2 | Récupération d'emails avec Dovecot (MDA) et Thunderbird (MUA) | 11 |
| 6.2.1 | Passage au format <i>maildir</i> | 11 |
| 6.2.2 | Ajout du serveur IMAP | 11 |
| 6.2.3 | Tests IMAP depuis Thunderbird (MUA) | 12 |
| 6.2.4 | Passage à IMAPS et SMTPS | 12 |
| 6.2.5 | Tests IMAPS et SMTPS depuis Thunderbird (MUA) | 13 |
| 6.3 | "Tunneliser" une connexion IMAP avec SSH | 13 |
| 6.4 | Des messages chiffrés et signés avec GnuPG | 13 |
| 6.4.1 | Principe | 13 |
| 6.4.2 | Mise en pratique | 13 |
| 7 | Outils d'audit : OpenVAS & Metasploit | 13 |
| 8 | Politique de sécurité : Flux autorisés | 13 |
| A | Configuration d'une passerelle réseau sous FreeBSD | 15 |
| A.1 | Configuration de la passerelle | 15 |
| A.1.1 | Interfaces réseaux | 15 |
| A.1.2 | IP forwarding | 16 |
| A.1.3 | NAT | 16 |
| A.1.4 | Firewall | 16 |
| A.1.5 | Syslogd | 17 |
| A.2 | Configuration du client | 17 |
| B | Mise en place de services sur la passerelle | 17 |

1 Scénario

Serveur « public » fournissant accès distant (VPN), compte email (postfix, dovecot, client email), espace web (apache).

Encryption autant que faire ce peut.

Logs.

Outils d'audit.

2 Topologie

3 Mise en place d'une passerelle et accès à Internet

Une passerelle (gateway) est un homme du milieu reliant deux réseaux distincts. Dans le cas présent, la machine *passerelle* doit faire communiquer les deux réseaux SLAN (192.168.1.0/24) et LAN Travaux Pratiques (192.168.2.0/24).

La passerelle doit être capable de transmettre des paquets IP :

```
(passerelle)# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Afin de maintenir l'*IP forwarding* après un reboot de la machine *passerelle*, on décommente dans le fichier */etc/sysctl.conf* la ligne suivante :

```
#net.ipv4.ip_forward=1
```

L'IP Masquerade (Network Address Translation) doit être activée. Cette fonctionnalité modifie les entêtes IPs du trafic passant par *passerelle* afin de rendre invisibles, au niveau IP, les machines de LAN Travaux Pratiques depuis l'extérieur. Ici on utilise du NAT dit de source utilisant la chaîne *POSTROUTING* puisqu'on modifie les adresses sources du paquet.

```
(passerelle)# iptables -t nat -A POSTROUTING -o eth0 -s 192.168.2.0/24 -j MASQUERADE
```

Note : L'interface *eth0* est connectée au réseau SLAN.

Enfin, syslogd doit être activé afin de logger les activités d'iptables

```
(passerelle)# apt-get install inetutils-syslogd
```

```
(passerelle)# edit /etc/syslog.conf # logs dans /var/log/kernel.log
```

```
(passerelle)# services syslog
```

3.1 Tests

On choisit un client, par exemple *client-bsd*. On lui retire l'interface réseau connectée à SLAN *em0*, et on s'assure que la machine est bien connectée sur LAN Travaux Pratiques via *em1*, et qu'elle peut communiquer avec la passerelle.

```
(client-bsd)# ifconfig em0 down
```

```
(client-bsd)# ifconfig em1
```

```
(client-bsd)# ping passerelle.cs.sr
```

Puis, on configure la table de routage du client afin que la route par défaut passe par *passerelle* et on vérifie :

```
(client-bsd)# netstat -r
```

Enfin, on s'assure qu'il est possible de contacter le serveur et d'atteindre Internet.

```
(client-bsd)# ping -c3 google.fr
```

| Internet: | | | | | | |
|------------------|-------------|-------|------|-----|-------|--------|
| Destination | Gateway | Flags | Refs | Use | Netif | Expire |
| default | 192.168.2.1 | UGS | 0 | 4 | em1 | |
| localhost | link#4 | UH | 0 | 160 | lo0 | |
| 192.168.1.3 | link#1 | UHS | 0 | 0 | lo0 | |
| 192.168.2.0 | link#2 | U | 0 | 7 | em1 | |
| obsd.localdomain | link#2 | UHS | 0 | 0 | lo0 | |

FIGURE 1 – Table de routage de *client-bsd*

On vérifie les logs sur la passerelle :

```
(passerelle)# tail -f /var/log/kernel.log
```

nmap? Faire peut-être une partie "outils d'attaque et... d'audit" où on explique l'utilisation de nmap, metasploit, vas (TP de demain) et en quoi ils nous aident à sécuriser notre propre réseau.

4 Mise à disposition d'un accès distant sur la machine et sécurisation

Telnet, SSH, VPN Expliquer comment mettre en place TCP Wrapper avec Telnet mais pas sécurisé. essai ssh avec mitm? (normalement il affiche un message du genre "SOMEONE MAY BE ON THE CABLE!")

4.1 Serveur Telnet et attaques de *sniffing*

On commence par installer un serveur *telnet* afin d'offrir un premier accès distant à la machine *passerelle*.

On n'utilisera pas ici le démon classique de telnet mais un démon générique nommé *inetd* permettant de minimiser le nombre de processus lancés. Quand *inetd* reçoit une demande de connexion, il lance le processus serveur adéquat.

Pour ce faire, on installe le serveur telnet.

```
(passerelle)# apt-get install telnetd
```

Puis, on modifie le fichier */etc/inetd.conf* pour y ajouter le service *telnet* dans la section des services standard, de façon à ce qu'il soit démarré automatiquement en tant qu'utilisateur telnetd :

```
(passerelle)# grep '^telnet' /etc/inetd.conf
telnet      stream    tcp nowait telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

On va maintenant tenter d'intercepter le couple *login/passwd* lors d'une connexion *telnet* entre *client-bsd* et *passerelle*.

Pour opérer, on utilise l'outil *ettercap* depuis la machine BackTrack. Il faut le lancer ainsi :

```
(backtrack)# ettercap -G
```

1. Dans le menu, on clique tout d'abord sur *Sniff* et on sélectionne *Unified sniffing* avant de choisir l'interface *eth1* qui est sur LAN Travaux Pratiques.
2. Dans le menu *Hosts*, on lance un scan du réseau local puis depuis le même menu, on affiche la liste des hôtes.

On reconnaît les adresses IP de *passerelle* et *client-bsd*. Il suffit donc de spécifier que ce sont nos deux cibles avec les boutons *TARGET 1* et *TARGET 2*.

3. Comme il s'agit d'un réseau commuté, on doit empoisonner les tables ARP (*ARP poisoning*) avant de lancer le *sniffing* pour récupérer le trafic entre les deux cibles.

Dans le menu *Mitm*, on sélectionne donc *ARP Poisoning*.

4. Enfin, on sélectionne dans le menu *Start* le choix *Start Sniffing*.

La machine *client-bsd* se connecte avec *telnet*.

Et on constate qu'il est extrêmement facile d'obtenir le couple *login/password* en clair :

On ne veut donc pas proposer un service d'accès à distance non sécurisé, c'est-à-dire basé sur des communications transmises en clair sur le réseau.

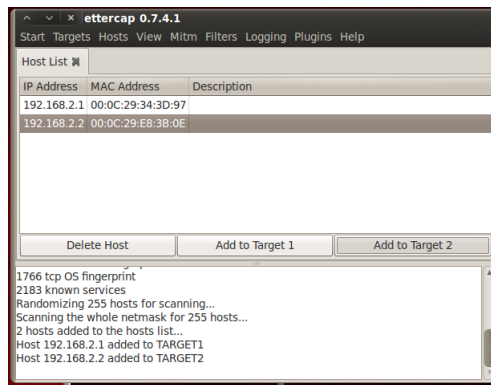


FIGURE 2 – Hôtes détectés par le scan

```
cssr@client-bsd ~1$ telnet 192.168.2.1 23
Trying 192.168.2.1...
Connected to 192.168.2.1.
Escape character is '^I'.
Password:
Last login: Mon Jan 14 08:52:29 CET 2013 from 192.168.2.2 on pts/2
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-32-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

cssr@passerelle:~$
```

FIGURE 3 – Connexion à *passerelle*

Telnet : 192.168.2.1:23 -> USER: cssr PASS: [REDACTED]

FIGURE 4 – Couple en clair

4.2 Serveur SSH et optimisations

SSH (Secure SHell) joue un rôle similaire à telnet, mais contrairement à ce dernier, il intègre des mécanismes de sécurité, comme le chiffrement des communications.

On installe puis démarre un serveur SSH sur la plateforme :

```
(passerelle)# apt-get install openssh-server
(passerelle)# service ssh start
```

On teste une connexion avec mot de passe (l'option *-v* est utilisé pour montrer la différence avec une connexion à base de clefs, voir plus bas)

```
(client-bsd)$ ssh -v cssr@passerelle.cs.sr
<METTRE L'OUTPUT ICI>
```

Afin d'éviter d'avoir à entrer le mot de passe à chaque connexion, on peut utiliser un système à base de clefs. On génère un couple clef privée/publique de type RSA sur le client :

```
(client-bsd)$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/cssr/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/cssr/.ssh/id_rsa
Your public key has been saved in /home/cssr/.ssh/id_rsa
The key fingerprint is:
f4:3e:54:9f:df:eb:72:cc:7a:da:29:a5:79:58:0c:44 cssr@client-bsd
The key's randomart image is:
+--[ RSA 2048 ]-----+
|           .E       |
```

```

|           . |
|         . o |
|       . . . o . |
|         S o = |
|           o =. |
|         o 0 o |
|           . *. *o |
|           oX= |
+-----+

```

Deux fichiers sont alors générés :

id_rsa.pub Clé publique devant être connue du serveur ;

id_rsa Clé privée ne devant pas être dévoilée.

Côté serveur, on doit maintenant ajouter la clé publique sur le serveur. SSH cherche dans le répertoire `$HOME/.ssh/` les fichiers dont le nom commence par *authorized_keys*. Ceux-ci doivent contenir une clé par ligne.

On transmet ainsi la clé publique sur le serveur, par exemple avec *scp* :

```
(client-bsd)$ scp .ssh/id_rsa.pub cssr@passerelle.cs.sr:~/.ssh/authorized_keys2
```

On prend soin d'ajuster les permissions du fichier :

```
(passerelle)# chmod 700 authorized_keys2
```

Enfin, on s'assure que l'authentification par clefs marche bien, et qu'il est difficile de récupérer les informations sur la connexion avec une attaque MITM.

La configuration du serveur SSH se fait dans le fichier `/etc/ssh/sshd_config`. Il peut-être intéressant d'y faire les réglages suivants :

X11Forwarding no pour réduire le trafic X11, et donc réduire les risques de sécurité ;

PermitRootLogin no pour désactiver un accès distant direct au compte root ;

Port 42 , où tout autre port différent du port 22 pour éviter les tentatives de connexions lancées par des bots (dénis de service).

4.3 Accès distant par VPN

Alors que SSH permet l'accès sécurisé à une machine distante (le serveur SSH), un VPN (Virtual Private Network) permet d'accéder au serveur VPN et à son réseau local de façon sécurisée.

On désire établir un réseau virtuel d'adresse 10.0.0.0 entre le client *client-bsd* et le serveur *passerelle*. On installe le paquet *openvpn* sur chaque machine :

```
(passerelle)# apt-get install openvpn
```

```
(client-bsd)# pkg_add -r openvpn
```

On effectue un premier test non chiffré.

```
(passerelle)# openvpn --dev tun0 --ifconfig 10.0.0.1 10.0.0.2
```

```
(client-bsd)# openvpn --remote passerelle.cs.sr --dev tun0 --ifconfig 10.0.0.2 10.0.0.1
```

De chaque côté, une interface réseau *tun0* apparaît : il s'agit d'une interface "Point-à-point". Et on fait quelques tests de connectivité dans le réseau virtuel.

Un VPN a la propriété d'être privé : cela signifie que seules les entités de part et d'autre du VPN peuvent accéder aux données en clair. Il est donc indispensable de chiffrer les communications avec un VPN. Il existe cependant plusieurs techniques cryptographiques.

```

[cssr@client-bsd ~]$ telnet 10.0.0.1
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^J'.
Password:
Last login: Mon Jan 14 09:48:05 CET 2013 from 192.168.2.2 on pts/2
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-32-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

You have mail.

```

FIGURE 5 – Le serveur est accessible via son interface *tun0*

4.3.1 Clé partagée

On engendre une clé partagée sur le serveur.

```
(passerelle)# openvpn --genkey --secret static.key
```

Puis, on recopie cette clé sur *client-bsd* avec la commande *scp* qui est sécurisée.

On peut relancer maintenant le client et le serveur avec les commandes suivantes.

```

(passerelle)# openvpn --dev tun0 --ifconfig 10.0.0.1 10.0.0.2 --secret /etc/openvpn/static.key
(client-bsd)# openvpn --remote passerelle.cs.sr --dev tun0 --ifconfig 10.0.0.2 10.0.0.1
--secret /etc/openvpn/static.key

```

La connexion est maintenant chiffrée.

On remarque que le nombre de paramètres de la commande *openvpn* devient de plus en plus important. C'est d'autant plus le cas si on veut effectuer de la compression. Il est préférable de lancer la commande en faisant référence à un fichier de configuration.

```

(passerelle)# openvpn /etc/openvpn/server.conf
(client-bsd)# openvpn /etc/openvpn/client.conf

```

4.3.2 Utilisation de clés SSL/TLS

Avec cette technique, les pairs s'authentifient entre-eux à l'aide de certificats. Chaque pair (serveur ou un des clients) possède un certificat et une clé privée, gardée secrète. Les différents certificats sont signés par une autorité de certification. Et dans notre cas, l'autorité de certification est en fait le serveur.

5 Configuration d'un serveur web et d'OpenSSL

5.1 Installation d'OpenSSL

```

(passerelle)# ./configure --prefix=/usr/local && make && make install
(passerelle)# cat >> /etc/ld.so.conf
/usr/local/openssl/lib
^D
(passerelle)# ldconfig

```

5.2 Installation d'Apache2

Après avoir récupéré les sources sur le site (et vérifié le hash du fichier), on configure Apache avec le support d'OpenSSL

```

(passerelle)# ./configure --option-du-chemin-ssl=/usr/local/openssl --prefix=/usr/local/apache2
(passerelle)# /usr/local/bin/apache2/bin/apachectl start
(passerelle)# nc passerelle 80
GET /
<html><body><h1>It works!</h1></body></html>

```

5.3 Génération d'un certificat X509

On génère dans l'ordre :

1. une clé privée;
2. un CSR (Certificate Signing Request);
3. un certificat X509.

Enfin, on copie (arbitrairement) ces fichiers dans `/usr/local/apache2/conf/` :

```
(passerelle)# openssl version
OpenSSL 0.9.8x 10 May 2012
(passerelle)# openssl req -new -key ca.key -out ca.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Nice
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CSSR Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:A challenge password
An optional company name []:
(passerelle)# openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
Signature ok
subject=/C=FR/ST=France/L=Nice/O=CSSR Ltd
Getting Private key
(passerelle)# cp ca.* /etc/ssl/
```

Le certificat pourra être utilisé par les serveurs HTTPS, SMTP, IMAPS, etc.

5.4 Passage à HTTPS

On inclut le fichier `extra/httpd-ssl.conf` depuis le `httpd.conf` :

```
Include conf/extra/httpd-ssl.conf
```

Puis, on règle les chemins vers le certificat et la clé dans ce dernier :

```
SSLCertificateFile "/usr/local/apache2/conf/ca.crt"
SSLCertificateKeyFile "/usr/local/apache2/conf/ca.key"
```

Enfin, on règle les paramètres pour le domaine virtuel dans lequel tourne la partie HTTPS :

```
DocumentRoot "/export/wwws/"
ServerName www.example.com:443
ServerAdmin you@example.com
ErrorLog "/usr/local/apache2/logs/error_log"
TransferLog "/usr/local/apache2/logs/access_log"

<Directory /export/wwws/>
```



```
Options FollowSymLinks
AllowOverride None
Order deny,allow
#Deny from all
</Directory>
```

5.5 Tests

Le script d'authentification fourni peut être étendu pour fournir une interface de création de compte sur la passerelle. On peut ainsi créer automatiquement un compte UNIX, un espace web personnel, etc. Pour plus de sécurité, on peut se contenter d'envoyer un email à l'administrateur de la machine avant la création du compte, pour que celui-ci s'assure que l'utilisateur a bien de bonnes raisons de demander un accès.

Dans tout les cas, une attaque sur le serveur sécurisé avec ettercap fonctionne comme les attaques précédentes (ARP spoofing) au détail près qu'ettercap génère un faux certificat de façon à récupérer les données d'authentification.

Un utilisateur averti risque de le remarquer, à moins que cela soit sa première connexion sur le site. En effet, le certificat étant auto-signé, il devra au moins être ajouté manuellement la première fois.

6 Service de messagerie électronique et sécurisation

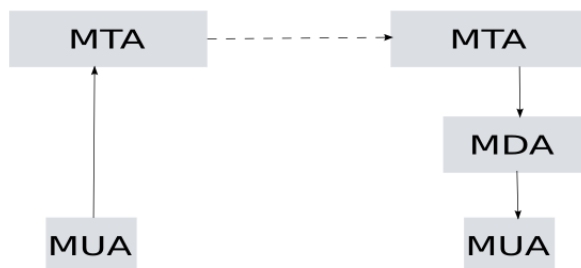


FIGURE 6 – Un MUA envoie un email à un MTA, qui le forward jusqu'à un MTA final, qui le transmet à un MDA. Enfin, le MUA du destinataire récupère l'email depuis ce MDA

La figure 6 donne un exemple simple de routage d'email, faisant intervenir 3 pièces logicielles :

MTA Mail Transfer Agent (eg. serveur SMTP), qui relaye les emails de domaines en domaines jusqu'à arriver à bonne destination ;

MDA Mail Delivery Agent (eg. serveur IMAP(s)), qui reçoit les emails du MTA du domaine et les délivre aux MUAs qui lui demandent ;

MUA Mail User Agent c'est le client email, dont le rôle principal est de récupérer les emails depuis un MDA, et d'en envoyer à un MTA ;

D'autres agents optionnels peuvent venir s'y greffer.

En pratique, on installe et configure postfix sur *passerelle*.

6.1 Envoi et transfert d'emails avec Postfix (MTA)

Postfix est un serveur SMTP modulaire et complexe.

On installe le paquet *postfix*

```
(passerelle)# apt-get install postfix
```

Une fois le paquet installé, il faut choisir la configuration de base : on choisit "SiteInternet" afin que Postfix crée lui-même les fichiers de configuration dans */etc* et qu'il les remplisse de quelques directives de base.

On modifie le fichier */etc/postfix/main.cf*

```
smtpd_banner = # On ne met rien pour ne pas donner inutilement des informations
myhostname = passerelle.cs.sr # De préférence /etc/hostname
mydomain = cs.sr
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases

mynetworks = 127.0.0.0/8 # Réseaux des clients de confiance
mydestination = $myhostname, localhost.$mydomain, localhost, $mydomain # Domaines locaux
inet_interfaces = all
```

On prend soin d'exécuter le démon en mode *debug* en modifiant dans */etc/postfix/master.cf* la ligne du service *smtp* ainsi :

```
smtp      inet  n       -       n       -       -       smtpd -v
```

Il faut ensuite relancer le démon.

```
(passerelle)# /etc/init.d/postfix reload
```

L'utilisateur *alice* est ensuite ajouté.

```
(passerelle)# adduser alice
```

Le mot de passe UNIX doit être précisé.

Un mail depuis l'utilisateur *cssr* est envoyé à Alice.

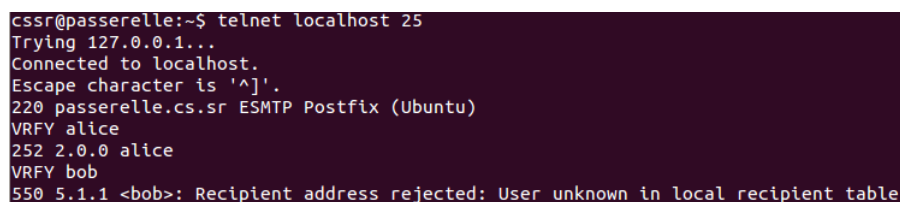
```
(passerelle)# echo "Hello" | mail -s Test alice
```

On peut vérifier que Postfix est utilisé en visualisant les logs dans */var/log/mail.log*

La livraison étant locale, l'agent "local" de *postfix* est utilisé. Ce dernier délègue la livraison des messages à *procmail*. On pourra donc lire le message dans */var/mail/alice*.

```
(passerelle)# less /var/mail/alice
```

Essayons d'envoyer un mail à un utilisateur qui n'existe pas : *bob*. La commande *VRFY* du protocole SMTP nous indique qu'il n'existe pas.



```
cssr@passerelle:~$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 passerelle.cs.sr ESMTP Postfix (Ubuntu)
VRFY alice
252 2.0.0 alice
VRFY bob
550 5.1.1 <bob>: Recipient address rejected: User unknown in local recipient table
```

FIGURE 7 – Résultat *VRFY*

On tente tout de même d'envoyer un mail avec *mail*. Et le *MAILER-DAEMON* nous répond pour nous informer que l'email n'a pas pu être envoyé.

Avec les alias, il est possible d'indiquer à Postfix que certains utilisateurs locaux peuvent être redirigés vers des utilisateurs locaux voire distants. On souhaite ici que le courrier pour Alice.Personne soit délivré à Alice. Pour ce faire, on modifie les alias dans */etc/aliases*.

Il faut ensuite mettre à jour les alias.

```
# See man 5 aliases for format
postmaster:    root
Alice.Personne: alice
```

FIGURE 8 – Fichier des aliases

```
(passerelle)# postalias /etc/aliases
```

Et *alice* reçoit bien les messages destinés à *Alice.Personne*.

On peut noter qu'il est préférable de ne pas mettre le réseau local 192.168.1.0/24 dans *mynetworks*. Effectivement, cela considère qu'il est de confiance et qu'on peut se passer de faire des vérifications sur le nom de domaine des destinataires.

6.2 Récupération d'emails avec Dovecot (MDA) et Thunderbird (MUA)

Nous aimerions que nos utilisateurs UNIX disposent d'un système leur permettant de consulter et gérer à distance leurs messages. Le protocole IMAP offre ses fonctionnalités.

6.2.1 Passage au format *maildir*

On utilise *maildrop* au lieu de *procmail*. L'agent *maildrop* sait notamment délivrer le courrier aux dossiers au format *Maildir*. L'agent *procmail* délivrait, lui, le courrier en utilisant le format *mailbox* c'est-à-dire en concaténant tous les messages dans un même fichier. *Maildir* permet d'avoir un fichier par message. Installons donc *maildrop*

```
(passerelle)# sudo apt-get install maildrop
```

Il faut modifier la configuration de Postfix (*/etc/postfix/main.cf*).

```
mailbox_command = /usr/bin/maildrop
```

On crée une boîte aux lettres par utilisateur UNIX, dans leur \$HOME.

```
(passerelle)# cd ~
(passerelle)# maildirmake
```

Et bien sûr, on recharge la configuration de Postfix.

6.2.2 Ajout du serveur IMAP

On installe les paquets pour Dovecot.

```
(passerelle)# sudo apt-get install dovecot-common
(passerelle)# sudo apt-get install dovecot-imapd
```

A l'installation, un certificat et une clé privée pour Dovecot sont générés automatiquement. Le certificat et la clé se trouvent respectivement dans */etc/ssl/certs/dovecot.pem* et */etc/ssl/private/dovecot.key*. Nous les utiliserons pour activer SSL/TLS sur le serveur IMAP.

On modifie dans */etc/dovecot/dovecot.conf* la ligne 26.

```
listen = *
```

La méthode d'authentification va être modifiée dans */etc/dovecot/conf.d/10-auth.conf*. On autorise le *plain text* parce que SSL/TLS sera utilisé.

```
disable_plaintext_auth = no # ligne 9
auth_mechanisms = plain login # ligne 97
```

Dans */etc/dovecot/conf.d/10-mail.conf*, on définit que les dossier des messages se trouvent dans */Maildir*.

```
mail_location = maildir:~/Maildir
```

Puis, dans `/etc/dovecot/conf.d/10-master.conf` pour la communication Dovecot - Postfix.

```
# Postfix smtp-auth
unix_listener /var/spool/postfix/private/auth { # Ligne 88
    mode = 0666
    user = postfix # à ajouter
    group = postfix # à ajouter
}
```

On remarque dans `/etc/dovecot/conf.d/10-auth.conf` que les utilisateurs sont identifiés grâce à leurs identifiants UNIX.

Finalement on redémarre le service `dovecot`

```
(passerelle)# service dovecot restart
```

6.2.3 Tests IMAP depuis Thunderbird (MUA)

On configure maintenant Thunderbird sans sécuriser la connexion puisque le service utilisé, IMAP, n'implémente pas SSL/TLS.

A la demande de réception des messages, les identifiants d'Alice sont demandés. Quant à l'envoi d'un message, ils ne nécessitent pas d'authentification.

L'email de `cssr` est bien reçu par Alice.

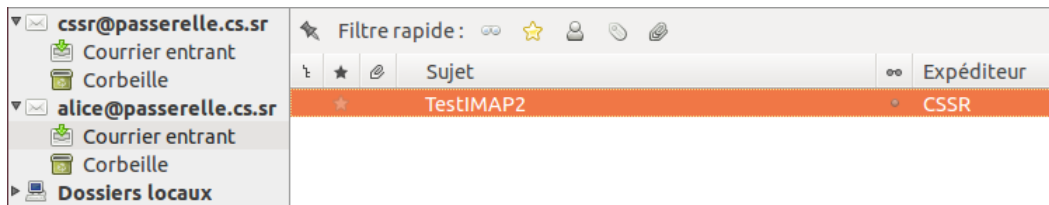


FIGURE 9 – MUA

6.2.4 Passage à IMAPS et SMTPS

Pour utiliser des services sécurisés, on a besoin d'implémenter SSL/TLS. On va donc mettre en place un certificat SSL.

Il a été mentionné qu'un certificat et une clé privée avaient été générés par Dovecot. Nous allons donc les utiliser dans la suite de la configuration. Bien entendu, on pourrait créer "à la main" le certificat `mails.crt` avec les commandes suivantes :

```
(passerelle)# openssl genrsa -out mails.key 2048 # Clé privée
(passerelle)# openssl req -new -days 3650 -key mails.key -out mails.csr # Demande de certificat
(passerelle)# openssl x509 -in mails.csr -out mails.crt -req -signkey mails.key -days 3650
```

Et on pourra supprimer le fichier `mails.csr` qui n'est plus utile. On peut également utiliser le script `mkcert.sh` fourni par Dovecot.

On commence par configurer Dovecot. Dans le fichier `/etc/dovecot/conf.d/10-ssl.conf` :

```
ssl = yes # Ligne 6
ssl_cert = </etc/ssl/certs/dovecot.pem # Ligne 12
ssl_key = </etc/ssl/private/dovecot.pem # Ligne 13
```

On relance le démon Dovecot.

Puis, on configure Postfix qui va utiliser le même certificat pour SMTPS. Dans le fichier `/etc/postfix/main.cf`, il faut ajouter :

```
smtpd_use_tls = yes
smtpd_tls_cert_file = /etc/ssl/certs/dovecot.pem
smtpd_tls_key_file = /etc/ssl/private/dovecot.key
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
```

Il faut également activer le démon *smtps* dans */etc/postfix/master.cf* et décommenter les lignes suivantes :

```
smtps      inet  n       -       -       -       smtpd
-o syslog_name=postfix/smtps
-o smtpd_tls_wrappermode=yes
```

On relance Postfix.

6.2.5 Tests IMAPS et SMTPS depuis Thunderbird (MUA)

Cette fois-ci, on configure Thunderbird pour qu'il utilise une connexion sécurisée pour le serveur entrant (IMAPS, port 993) et pour le serveur sortant (SMTPS, port 465). Dans les deux cas, on fournit un "mot de passe normal". Avec une connexion sécurisée, on ne peut pas lire ce mot de passe même si le mot de passe n'est pas chiffré.

6.3 "Tunneliser" une connexion IMAP avec SSH

Le protocole IMAP peut être sécurisée en utilisant la technique de *SSH tunneling*. En effet, SSH (qui est associé au port 22) est capable de chiffrer et déchiffrer le trafic d'autres applications sur d'autres ports. SSH fournit alors un "tunnel sécurisé" à d'autres connexions TCP/IP.

Le serveur IMAP doit posséder un serveur SSH, ce qui est notre cas. Depuis *client-bsd*, on va mettre en place le tunnel :

```
(client-bsd)# ssh -L2013:localhost:143 passerelle.cs.sr
```

Le client de messagerie doit maintenant envoyer les données sur le port 2013 de sa propre machine.

6.4 Des messages chiffrés et signés avec GnuPG

6.4.1 Principe

Il est bon de chiffrer les connexions, mais si les données circulent par d'autres serveurs de courrier, le chiffrement de la connexion n'est plus garanti. C'est pourquoi il faut envisager un chiffrement des données elles-mêmes.

GnuPG permet de chiffrer et signer des messages électroniques. Chaque individu génère sa propre paire de clés : une publique et une privée.

On envoie sa clé publique à ses correspondants. Ces derniers pourront à terme nous écrire en toute confidentialité car ils chiffreront le message avec notre clé publique et nous pourrons le lire grâce à notre clé privée. Cependant, nos correspondants ne font pas encore confiance à notre clé publique. Ils doivent nous contacter (par un autre moyen de communication) pour comparer l'empreinte de la clé publique qu'ils ont reçu et l'empreinte de notre clé publique. Une fois qu'ils ont confiance en la clé publique envoyée, ils la signent avec leur clé secrète.

Concernant la signature, nous appliquons notre clé privée sur le message que l'on souhaite signer et nos correspondants (qui connaissent notre clé publique) pourront vérifier que le message provient bel et bien de nous.

6.4.2 Mise en pratique

Pour simplifier l'utilisation de GnuPG, on utilise le plugin *Enigmail* pour le MUA *Thunderbird*. Et on crée une paire de clés (et un certificat de révocation) pour l'utilisateur *alice* mais aussi pour l'utilisateur *cssr*.

7 Outils d'audit : OpenVAS & Metasploit

8 Politique de sécurité : Flux autorisés

En partant de la configuration suivante qui bloque le trafic entrant, et autorise tout le trafic sortant/forwardé :

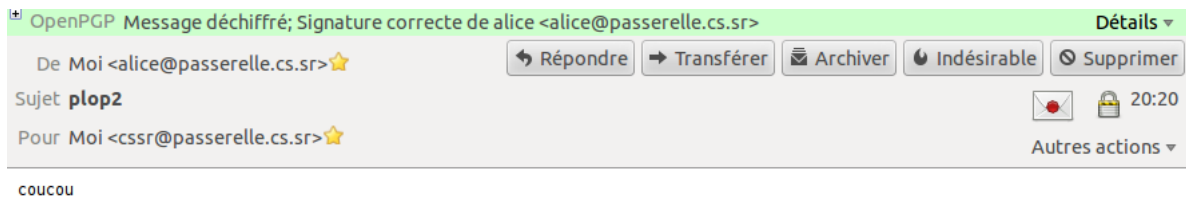


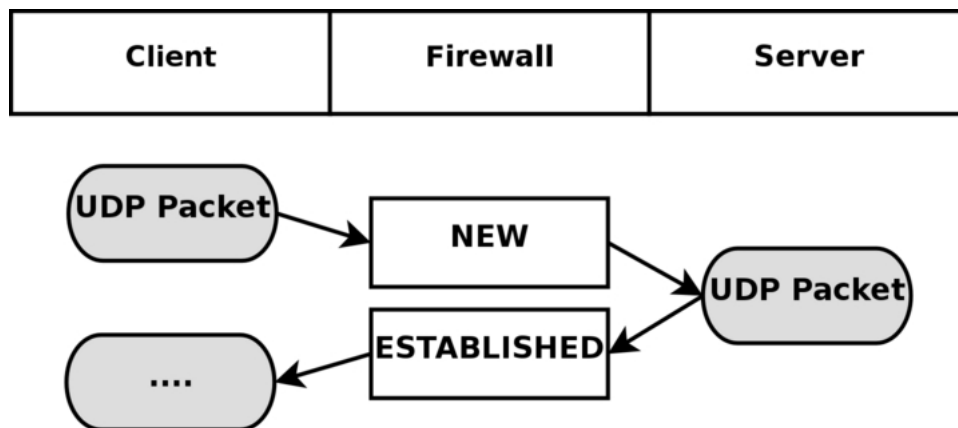
FIGURE 10 – L'utilisateur *cssr* reçoit un message d'Alice qui avait été signé et chiffré

```
(passerelle)# iptables -P INPUT DROP
(passerelle)# iptables -P OUTPUT ACCEPT
(passerelle)# iptables -P FORWARD ACCEPT
```

On autorise le trafic entrant venant de localhost :

```
(passerelle)# iptables -A INPUT -s 127.0.0.1 -j ACCEPT
```

On autorise les accès DNS sur le serveur. Pour ce faire, on autorise le trafic entrant venant du port 53 sur ladite machine, la connexion devant être *ESTABLISHED*. Lors de l'envoi de la requête DNS via UDP par un client, la connexion sera vue comme *NEW* par netfilter, puis lors de la réponse du serveur, comme *ESTABLISHED* :



Cet état est en fait utilisé lors des échanges attendant une réponse. Cela nous donne donc la règle suivante :

```
(passerelle)# iptables -A INPUT -i eth0 -p udp -s 192.168.1.1 --sport 53 \
-m state --state ESTABLISHED -j ACCEPT
```

On ajoute sur la chaîne INPUT le fait qu'en cas de connexion TCP sur le port ssh, il faille accepter le trafic :

```
(passerelle)# iptables -A INPUT -p tcp --dport ssh -j ACCEPT
```

D'une façon similaire, on autorise les accès HTTP et HTTPS :

```
(passerelle)# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
(passerelle)# iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Enfin, pour les services emails (dans l'ordre IMAP, IMAPS, SMTP (client/serveur) SMTPS)

```
(passerelle)# for i in 143 993 587 110 465; do \
iptables -A INPUT -p tcp --dport $i -j ACCEPT \
done
```

A Configuration d'une passerelle réseau sous FreeBSD

Afin de voir les différences de configuration entre un Linux et une FreeBSD, on utilise ici une topologie plus simple, mais similaire à celle utilisée en cours.

On utilise un VMware sous un système hôte Linux (3.7.5-1), faisant tourner deux machines virtuelles :

FreeBSD 9.1 , passerelle, disposant de deux adaptateurs réseaux :

1. un configuré en NAT, switch virtuel `/dev/vmnet8` de l'hôte, permettant d'obtenir une connexion internet ;
2. l'autre en Host-Only, `/dev/vmnet1`, permettant d'obtenir un réseau privé ;

Plan9 (9front-1242.62aa45fc09be) , client, doté d'un seul adaptateur Host-Only, configuré en t

En quelque sorte, la machine hôte joue ici le rôle du serveur dans la topologie du cours, en permettant un accès aux Internets ; le réseau accessible via l'adaptateur en Host-Only permet de créer un réseau local analogue à *LAN Travaux Pratiques*.

Les parties ici présentes portant sur la configuration des services contiennent donc des doublons par rapport à la configuration de la passerelle sous Linux.

A.1 Configuration de la passerelle

A.1.1 Interfaces réseaux

On commence par configurer l'interface NAT, en utilisant par exemple le serveur DHCP du switch virtuel. Ceci est normalement déjà effectué au démarrage (cf. `/etc/rc.conf`) :

```
(passerelle)# dhclient em0
DHCPREQUEST on em0 to 255.255.255.255 port 67
DHCPACK from 192.168.237.254
bound to 192.168.237.132 -- renewal in 900 seconds.
(passerelle)# grep em0 /etc/rc.conf
ifconfig_em0="DHCP"
```

Puis on se connecte au réseau local, en utilisant ici une IP statique, et l'on fait en sorte que la connection soit effective au démarrage de la machine :

```
(passerelle)# ifconfig em1 192.168.98.2
(passerelle)# cat >> /etc/rc.conf
ifconfig_em1="inet 192.168.98.2 netmask 255.255.255.0"
^D
```

On s'assure que l'on peut bien communiquer avec le système hôte (`x.x.x.1`) via les deux interfaces, et que l'on peut accéder aux Internets :

```
(passerelle)# for i in 192.168.98.1 192.168.237.1 google.fr; do ping -c1 -q $i; done
PING 192.168.98.1 (192.168.98.1): 56 data bytes
```

```
--- 192.168.98.1 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.217/0.217/0.217/0.000 ms
PING 192.168.237.1 (192.168.237.1): 56 data bytes
```

```
--- 192.168.237.1 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.117/0.117/0.117/0.000 ms
PING google.fr (173.194.34.24): 56 data bytes
```

```
--- google.fr ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 50.611/50.611/50.611/0.000 ms
```

A.1.2 IP forwarding

On active l'IP forwarding, manuellement, puis automatiquement au démarrage :

```
(passerelle)# sysctl net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
(passerelle)# cat >> /etc/rc.conf
gateway_enable="YES"
^D
```

A.1.3 NAT

Par défaut, le noyau de FreeBSD n'est pas configuré pour faire du NAT ; il recompiler un pépin en lui ajoutant l'option *IPDIVERT* :

```
(passerelle)# cd /sys/i386/conf/
(passerelle)# cp GENERIC LOCAL
(passerelle)# cat >> LOCAL
options          IPDIVERT                # Divert packets
^D
(passerelle)# config LOCAL
Kernel build directory is ../compile/LOCAL
Don't forget to do ''make cleandepend && make depend''
(passerelle)# cd ../compile/LOCAL/ && make cleandepend && make depend && make && make install
...
kldxref /boot/kernel
```

Le NAT en lui-même est géré par le démon *natd*, qu'il faut lancer au démarrage, sur l'interface de sortie (*em0*) :

```
(passerelle)# cat >> /etc/rc.conf
natd_enable="YES"
natd_interface="em0"
^D
```

Enfin, il faut dire au bootloader de charger les modules nécessaire pour l'IP-diverting :

```
(passerelle)# cat >> /boot/loader.conf
ipdivert_load="YES"
^D
```

A.1.4 Firewall

On choisit arbitrairement *ipfw* (deux autres choix possibles : *pf* et *ipf*), que l'on active au démarrage :

```
(passerelle)# cat >> /etc/rc.conf
firewall_enable="YES"
firewall_type="OPEN"
firewall_script="/etc/fw.sh"
```

Le script *fw.sh* configure la table NAT et le diverting pour l'interface *em0*, tout en laissant passer tout le trafic entrant/sortant :

```
(passerelle)# cat /etc/fw.sh
#!/bin/sh
ipfw -q -f flush
ipfw add divert natd all from any to any via em0
ipfw nat 1 config if em0
ipfw add allow ip from any to any
```


Puis on indique au bootloader de charger les bons modules pour que le pare-feu soit chargé, fonctionne avec le NAT, et soit laxiste :

```
(passerelle)# cat >> /boot/loader.conf
ipfw_load="YES"
ipfw_nat_load="YES"
net.inet.ip.fw.default_to_accept="1"
```

On n'oublie pas de redémarrer pour booter sur le nouveau pépin :

```
(passerelle)# reboot
```

A.1.5 Syslogd

Normalement, syslogd est activé par défaut. On s'assure que c'est bien le cas, et qu'il fonctionne effectivement :

```
(passerelle)# ps aux | grep sysl
root 1140  0.0  0.6  9504 1504 ??  Ss      3:54AM  0:00.02 /usr/sbin/syslogd -s
root 1419  0.0  0.7  9636 1688  0  S+     4:20AM  0:00.00 grep sysl
(passerelle)# tail -5 /var/log/auth.log
Feb  6 03:50:41 passerelle su: cssr to root on /dev/pts/0
Feb  6 03:54:06 passerelle sshd[1237]: Server listening on :: port 22.
Feb  6 03:54:06 passerelle sshd[1237]: Server listening on 0.0.0.0 port 22.
Feb  6 03:54:29 passerelle sshd[1307]: Accepted keyboard-interactive/pam for cssr from 192.168.237.1
Feb  6 03:54:31 passerelle su: cssr to root on /dev/pts/0
```

A.2 Configuration du client

On configure l'« interface » réseau avec une IP statique, on ajoute une route par défaut pour faire passer le trafic par la passerelle, et on s'assure que l'on peut communiquer avec l'extérieur :

```
term% ip/ipconfig -g 192.168.98.2 ether /net/ether0 192.168.98.
3 255.255.255.0
term% cat /net/iproute
192.168.98.0    /120 192.168.98.0    4i   ifc   -
192.168.98.0    /128 192.168.98.0    4b   ifc   -
192.168.98.3    /128 192.168.98.3    4u   ifc   0
192.168.98.128  /128 192.168.98.128  4u   ifc   0
192.168.98.255  /128 192.168.98.255  4b   ifc   -
255.255.255.255 /128 255.255.255.255 4b   ifc   -
term% echo add 0.0.0.0 0.0.0.0 192.168.98.2 > /net/iproute
term% ip/ping 8.8.8.8
sending 32 64 byte messages 1000 ms apart to icmp!8.8.8.8!1
0: rtt 27739 µs, avg rtt 27739 µs, ttl = 127
1: rtt 20589 µs, avg rtt 24164 µs, ttl = 127
term% !
```

B Mise en place de services sur la passerelle

Par manque de temps, tous les services vus en cours n'ont pas été testés sous FreeBSD. Néanmoins, pour ceux qui l'ont été, la configuration est à quelques détails près la même dans les deux cas. Seule la gestion du routage/firewall change vraiment, puisque les logiciels utilisés sont dans les deux cas identiques. Cette section est donc volontairement vide, pour éviter les doublons avec le reste de ce document.

On notera par exemple que pour postfix, il est nécessaire de désactiver sendmail pour éviter qu'ils ne se tapent dessus.