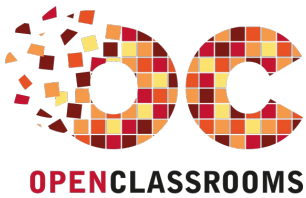
Sécuriser son serveur Linux

Par Binabik



www.openclassrooms.com

Dernière mise à jour le 31/08/2009

Sommaire

Sécuriser son serveur Linux

Par  Binabik

Mise à jour : 31/08/2009
Difficulté : Intermédiaire

Ce guide va vous apprendre à sécuriser un serveur et donc vous initier aux thématiques de la sécurité informatique. En quoi est-ce important ? Par définition, un serveur est ouvert sur le monde, un minimum de sécurité est donc intéressant afin de se prémunir des attaques les plus simplistes.

La marche à suivre sera donc la suivante :

- présentation des failles ;
- présentation des outils pour y pallier.

Bien entendu, je ne fais pas un cours complet de sécurité informatique, ce tutoriel est une initiation. Pour faire simple, c'est un peu comme fermer les volets la nuit.

Point de vue matériel, voilà ce qu'il vous faudra :

- un serveur embarquant une distribution xBuntu ou Debian (pour les autres distribs, le principe sera le même, mais les commandes risquent d'être différentes) ;
- un accès root (en ssh par exemple) et une console.

Tout le tutoriel se passe en ligne de commande afin de pouvoir être accessible à tous. Comme il faudra éditer des fichiers, veuillez vous assurer d'avoir un éditeur de fichier en ligne de commande (j'utiliserai personnellement nano qui est très simple).

C'est parti !
Sommaire du tutoriel :

Filtrer le trafic via le firewall

Présentation

Le *firewall* (pare-feu en français) est l'élément indispensable pour sécuriser son serveur. Il va en effet filtrer tout le trafic en n'autorisant que les échanges permis par l'administrateur. Sans *firewall* correctement réglé, tous les trafics sont plus ou moins permis (c'est-à-dire qu'un attaquant peut faire ce qu'il veut chez vous) et ce genre de faille est détectable par un simple *scan* de ports.

Or, le noyau Linux offre déjà un pare-feu à l'utilisateur, qu'il est possible de configurer via le logiciel *iptables* (normalement contenu dans /sbin/iptables). S'il n'est pas installé :

Code : Console

```
apt-get install iptables
```

Nous allons maintenant détailler le fonctionnement d'un *firewall* - relativement simple. Un *firewall* analyse tout le trafic et vérifie si chaque paquet échangé respecte bien ses règles (critères de filtrage). Donc, il suffit de spécifier de bonnes règles pour interdire tout trafic superflu.

Les critères peuvent être divers (filtrer les ports, les protocoles, les adresses IP, etc). De base, nous allons spécifier nos règles sur les ports. Bien entendu, il faut être le plus strict possible quant au choix des règles ; c'est pourquoi, par défaut, tout *firewall* se règle en premier lieu en bloquant tout, absolument tout. Ensuite, nous allons « ouvrir » (autoriser le trafic) certains ports que nous voulons utiliser (par exemple pour un serveur web, nous allons ouvrir le port 80 afin que le site web soit accessible).

Pour plus de souplesse, nous allons écrire nos règles sous forme de script *bash*. Petite mesure de prudence si vous êtes loggué sur votre machine à distance (ssh), soyez bien sûr de ne pas vous bloquer l'accès ou - le cas échéant - de pouvoir *rebooter* la machine. Sinon, j'ai bien peur que récupérer votre serveur sera compliqué !

Déclaration des règles

Filtrage intégral

Notez que la commande *iptables -L -v* vous permettra de consulter les règles courantes.

Suit la marche à suivre pour créer les règles :

1. Créons le script :

Code : Console

```
nano /etc/init.d/firewall
```

Et on y écrit : #!/bin/sh

2. On efface les règles précédentes pour partir sur de bonnes bases :

Code : Bash

```
iptables -t filter -F
iptables -t filter -X
```

3. On bloque par défaut tout le trafic (si vous êtes en *ssh*, bien entendu, n'exécutez pas encore le script !) :

Code : Bash

```
iptables -t filter -P INPUT DROP
iptables -t filter -P FORWARD DROP
iptables -t filter -P OUTPUT DROP
```

4. On ne ferme pas les connexions déjà établies :

Code : Bash

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Nous indiquons avec les paramètres -m et --state de ne pas fermer les connexions qui sont déjà établies.

5. On autorise le *loopback* (on ne va pas se bloquer nous-mêmes !)

Code : Bash

```
iptables -t filter -A INPUT -i lo -j ACCEPT
iptables -t filter -A OUTPUT -o lo -j ACCEPT
```

Note : lo signifie *localhost* (le serveur lui-même).

Tout est bloqué, il ne nous reste plus qu'à ouvrir les ports utilisés.

Ouvrir les ports utilisés

À partir de maintenant, observons plus en détail les paramètres de *iptables* :

- **-t** : vaudra par défaut « filter » ;
- **-A** : servira à indiquer le sens du trafic : INPUT (entrant) ou OUTPUT (sortant) ;
- **-p** : indique le protocole (TCP ou UDP en principe) ;
- **--dport et --sport** : respectivement port destination et port source (comme nous sommes le serveur, nous utiliserons principalement *dport*) ;
- **-j** : comment traiter le paquet (nous nous servirons d'ACCEPT et de DROP pour respectivement accepter et refuser le paquet).

Plus nous serons précis, plus nous serons sécurisés. Renseigner ces quelques options est donc le minimum.

Ainsi, une règle simple aura la forme suivante :

Code : Bash

```
iptables -t filter -A INPUT/OUTPUT -p protocole --dport
port_a_ouvrir -j ACCEPT
```

Notez enfin que si vous voulez un échange, il faut toujours ouvrir le port dans les deux sens (INPUT et OUTPUT)... logique.

Exemple si l'on a un serveur web (port 80) :

Code : Bash

```
iptables -t filter -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT
```

Vous pourriez me dire qu'il suffirait de n'écrire qu'une seule fois la ligne sans préciser l'argument -A... mais suivant notre politique de précision, ce serait une erreur car il existe d'autres valeurs que INPUT et OUTPUT pour lesquelles nous ne voulons pas permettre le trafic (FORWARD par exemple).

Il ne vous reste qu'à spécifier toutes les règles nécessaires. Voici un petit tableau pour vous aider (il s'agit de données par défaut) :

service	port d'écoute	protocole
ssh	22	tcp
web/HTTP	80	tcp
FTP	20 et 21	tcp
mail/SMTP	25	tcp
mail/POP3	110	tcp
mail/IMAP	143	tcp

DNS	53	tcp et udp

Cas particulier du ping
Le *ping* est basé sur un protocole particulier (ICMP) qui n'a pas de port prédéfini. Mais il faut absolument autoriser le *ping* car c'est la méthode la plus couramment utilisée pour savoir si votre serveur est en vie. Voici donc les règles :

Code : Bash

```
iptables -t filter -A INPUT -p icmp -j ACCEPT
iptables -t filter -A OUTPUT -p icmp -j ACCEPT
```

Démarrage du *firewall*

Enfin, nous allons lancer notre *firewall* :

Code : Console

```
chmod +x /etc/init.d/firewall
/etc/init.d/firewall
```

Tester abondamment que tout se passe bien. Vous pouvez notamment utiliser l'utilitaire *nmap* pour vérifier qu'il n'y a pas plus de ports ouverts que voulu (cf partie 5).

Il est important de charger ce script au démarrage de la machine afin qu'un simple *reboot* ne vous laisse pas sans protection :

Code : Console

```
update-rc.d firewall defaults
```

Exemple de script

Ci-dessous, je vous montre un exemple de script basique autorisant le minimum vital pour un serveur web (HTTP, FTP, mail et résolution de DNS). Je vous encourage à lire des docs et des tutos plus complets si vous voulez aller plus loin dans le paramétrage de votre *firewall*.

Secret (cliquez pour afficher)

Code : Bash

```
#!/bin/sh

# Réinitialise les règles
sudo iptables -t filter -F
sudo iptables -t filter -X

# Bloque tout le trafic
sudo iptables -t filter -P INPUT DROP
sudo iptables -t filter -P FORWARD DROP
sudo iptables -t filter -P OUTPUT DROP

# Autorise les connexions déjà établies et localhost
sudo iptables -A INPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT
sudo iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT
sudo iptables -t filter -A INPUT -i lo -j ACCEPT
```

```
sudo iptables -t filter -A OUTPUT -o lo -j ACCEPT

# ICMP (Ping)
sudo iptables -t filter -A INPUT -p icmp -j ACCEPT
sudo iptables -t filter -A OUTPUT -p icmp -j ACCEPT

# SSH
sudo iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
sudo iptables -t filter -A OUTPUT -p tcp --dport 22 -j ACCEPT

# DNS
sudo iptables -t filter -A OUTPUT -p tcp --dport 53 -j ACCEPT
sudo iptables -t filter -A OUTPUT -p udp --dport 53 -j ACCEPT
sudo iptables -t filter -A INPUT -p tcp --dport 53 -j ACCEPT
sudo iptables -t filter -A INPUT -p udp --dport 53 -j ACCEPT

# HTTP
sudo iptables -t filter -A OUTPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -t filter -A INPUT -p tcp --dport 80 -j ACCEPT

# FTP
sudo iptables -t filter -A OUTPUT -p tcp --dport 20:21 -j ACCEPT
sudo iptables -t filter -A INPUT -p tcp --dport 20:21 -j ACCEPT

# Mail SMTP
iptables -t filter -A INPUT -p tcp --dport 25 -j ACCEPT
iptables -t filter -A OUTPUT -p tcp --dport 25 -j ACCEPT

# Mail POP3
iptables -t filter -A INPUT -p tcp --dport 110 -j ACCEPT
iptables -t filter -A OUTPUT -p tcp --dport 110 -j ACCEPT

# Mail IMAP
iptables -t filter -A INPUT -p tcp --dport 143 -j ACCEPT
iptables -t filter -A OUTPUT -p tcp --dport 143 -j ACCEPT

# NTP (horloge du serveur)
sudo iptables -t filter -A OUTPUT -p udp --dport 123 -j ACCEPT
```

Allez un peu plus loin

Le *firewall*, outre que c'est l'outil de base de tout système de sécurité, permet des manipulations plus poussées que filtrer des ports. Je vais vous montrer quelques exemples.

Flood ou déni de service

Ce genre d'attaque vise à surcharger la machine de requête. Il est possible de s'en prémunir pas mal directement au niveau du *firewall* :

Code : Bash

```
iptables -A FORWARD -p tcp --syn -m limit --limit 1/second -j ACCEPT
```

Le *flags* TCP *syn* engendre des demandes de connexions, et le but de cette règle est donc de les limiter à une par seconde (champs *limit*).

Il est cependant déconseillé de monter au-delà de la seconde (sous peine de gêner le contrôle de flux et la récupération d'erreur de TCP).

On peut faire de même avec les protocoles UDP et ICMP :

Code : Bash

```
iptables -A FORWARD -p udp -m limit --limit 1/second -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --
limit 1/second -j ACCEPT
```

Notez cependant que ce type d'attaque permet de faire tomber le serveur, mais pas d'en prendre l'accès. C'est pourquoi le risque d'en être la cible est assez mince (sauf si l'on s'appelle Google). En général, un logiciel simple anti-intrusion comme *fail2ban* (cf partie 2) est suffisant, suivant le niveau de sécurité recherché.

Scan de ports

On peut aussi limiter un tant soit peu le *scan* de ports (qui consiste à tester tous vos ports afin de détecter ceux qui sont ouverts). Pour cela, une règle de ce genre irait :

Code : Bash

```
iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit
--limit 1/s -j ACCEPT
```

C'est un peu le même principe que ci-dessus. Sachant qu'une connexion TCP en bon et due forme requiert trois paquets avec trois *flags* différents, on voit tout de suite la finesse de cette règle qui peut travailler paquet par paquet. Pour plus d'infos sur les *flags* TCP, cf Wikipédia.

Notez que cette règle basique n'est pas très efficace, c'est une protection de base. La partie 2 ira plus loin dans le blocage des *scans* de ports.

Bannir une IP

Si vous repérez dans les *logs* ou autre une adresse IP suspecte, vous pouvez la bannir aisément au niveau du *firewall* via la commande :

Code : Console

```
iptables -A INPUT -s adresse_ip -j DROP
```

Notez cependant qu'il n'est pas conseillé de bannir les IP à tour de bras.

Se prémunir contre les intrusions

Actuellement, le *firewall* va bloquer toutes tentatives de connexions sur les ports fermés. Mais qu'en est-il des ports ouverts ? Afin de contrôler plus précisément ce qui se passe dessus, le *firewall* n'est pas suffisant et nous allons devoir utiliser d'autres outils, appelés IDS (*Intrusion Detection System*) et IPS (*Intrusion Prevention System*). Ces deux catégories de logiciels vont - comme leur nom l'indique - surveiller toute tentative d'intrusion sur le serveur.

La démarche qui va suivre va vous montrer comment réagir à chaque étape d'une tentative intrusion classique, à savoir :
1. le scan de port (plus généralement, la collecte d'informations) afin de trouver les vulnérabilités ;
2. les attaques « simples », témoins d'une faible sécurité ;
3. l'intrusion (via des techniques qui ne seront pas décrites ici car dépassant notre cadre de travail) ;
4. l'installation d'un moyen de se logguer sur le serveur à volonté (si l'attaquant parvient jusqu'ici avec succès, on peut dire que la machine lui appartient).

Portsentry (*scan* de ports)

Cet utilitaire permet de bloquer en temps réel la plupart des *scans* de port connus (même très discrets et échappant aux règles de filtrage du *firewall* basiques). Je rappelle au passage que scanner les ports signifie tester tous les ports d'une machine afin de déterminer ceux qui sont ouverts (les portes d'entrées en gros). Cependant, il ne faut pas paniquer si votre serveur est la cible d'un simple *scan* de port, cela sera monnaie courante, et si vous êtes bien protégé, le pirate passera sa route.

Portsentry est donc sympa si vous voulez compliquer la tâche de l'attaquant :

Code : Console

```
apt-get install portsentry
```

Pour le configurer :

Code : Console

```
nano /usr/local/psionic/portsentry/portsentry.conf
```

Ou :

Code : Console

```
nano /etc/portsentry/portsentry.conf
```

Commentez les lignes *KILL_HOSTS_DENY*.
Décommentez la ligne *KILL_ROUTE="/sbin/iptables -I INPUT -s $TARGET$ -j DROP"* .

Ainsi, *Portsentry* ajoutera une règle dans le *firewall* (*iptables*) pour rejeter les paquets en cas de *scans*.

On démarre le logiciel (il faut le lancer deux fois, pour TCP et UDP) :

Code : Console

```
portsentry -audp
portsentry -atcp
```

Vous pouvez tester tout ça avec *nmap* (si vous voulez tester en local, il vous faut modifier le fichier portsentry.ignore en enlevant le *localhost*).

Si vous souhaitez que vos réglages restent même après un nouveau lancement de portsentry, vous devrez modifier le fichier portsentry.ignore.static

Fail2ban (brute-force, dictionnaire, déni de service)

Comme je l'ai dit, les ports ouverts sur la machine sont à priori sans grande protection, et sujet à des attaques simples telles que la tentative de connexion par brute-force ou par dictionnaire (par exemple, tester toutes les combinaisons de mots de passe pour se logguer en ssh), le déni de services (surcharger le serveur de requêtes) ou - plus bêtement - la recherche d'utilisateurs sans mots de passe... Si votre machine est infiltrée aussi facilement, l'attaquant sera vraiment content.

Fail2ban est un petit utilitaire qui se base sur les *logs* de la machine pour chercher des actions suspectes répétées (par exemple, des erreurs de mots de passe) dans un laps de temps donné. S'il en trouve, il bannira l'IP de l'attaquant via *iptables*. Ce type de logiciel est indispensable, car, bien que léger, il offre une bonne protection contre les attaques basiques indiquées ci-dessus.

Pour l'installation :

Code : Console

```
apt-get install fail2ban
```

Pour la configuration :

Code : Console

```
nano /etc/fail2ban/jail.conf
```

Je vous encourage à remplir le champ ci-dessous :
- **destmail** : indiquez une adresse mail si vous voulez recevoir des mails d'alerte de la part de *fail2ban*.

Le niveau de protection peut être modulé via les champs suivants (notez que la configuration par défaut suffit normalement) :
- **bantime** : temps de bannissement des IP suspectes ;
- **maxretry** : nombre de tentatives de connexion permise avant bannissement.

Notez que dans la partie JAILS (dans nano : ctrl w => rechercher JAILS) figure tous les services que *fail2ban* surveillera. Si vous avez modifié les ports par défaut, il faut les indiquer là aussi. Par exemple avec *ssh* :

Code : Console

```
nano /etc/fail2ban/jail.conf
ctrl+w => chercher [ssh]
port : indiquer le port
```

Enregistrez et quittez.

Je vous encourage à parcourir rapidement le reste des options afin de personnaliser un peu votre soft.

Enfin, pour recharger la nouvelle configuration :

Code : Console

```
/etc/init.d/fail2ban restart
```

Snort (détection d'intrusions)

Le problème quand on commence à sécuriser est de savoir s'arrêter à un moment donné. *Snort* est un outil très puissant, pouvant en fait détecter la plupart des attaques qui échapperaient à un utilitaire comme *fail2ban*. Bien entendu, il ne servira pas dans 90 % des cas et comme ce n'est qu'un outil de détection, ce sera à vous de rendre les mesures nécessaires s'il détecte une intrusion. Enfin, comme il analyse le trafic en temps réel, cela ralentit forcément un peu les flux. L'installer n'est donc pas indispensable, cela dépend du degré de sécurité recherché !

Comme ce logiciel dépasse un peu le cadre de ce tutoriel, je vous propose de consulter ces guides pour l'installer :
http://www.trustonme.net/didactels/187.html
http://doc.ubuntu-fr.org/snort

J'en ai juste parlé car en matière de sécurité, c'est un outil très poussé.

Notez que ce logiciel fonctionne à partir de règles de sécurité écrites par des utilisateurs, et parfois erronées. En cas d'erreur au démarrage, n'hésitez pas à aller commenter les règles buguées (que vous trouverez dans /var/log/syslog).

Rkhunter (*rootkit* et *backdoors*)

Dernier volet de cette section intrusion, les *backdoors*. Si par malheur un attaquant arrive à prendre possession de votre machine, il y a fort à parier qu'il y laisse une *backdoor* (porte dérobée) qui lui permettrait d'en reprendre le contrôle plus tard, ainsi qu'un *rootkit* pour la dissimuler : l'attaquant maintient ainsi un accès frauduleux à votre machine.

Rkhunter est un utilitaire qui est chargé de détecter d'éventuels *rootkits* sur votre serveur. Il est relativement léger (s'exécute une fois par jour par défaut) donc on aurait tort de se priver.

Code : Console

```
apt-get install rkhunter
```

Il est conseillé de modifier un peu la configuration :

Code : Console

```
nano /etc/default/rkhunter
```

- **REPORT_EMAIL** : indiquez un mail pour recevoir des alertes de *Rkhunter* ;
- **CRON_DAILY_RUN** : mettez « *yes* » pour une vérification quotidienne de votre machine via un *cron*.

Notez que Rkhunter se trompe parfois en déclarant comme infectés des fichiers sains (« faux positifs »), donc il faut être critique à l'égard des rapports. Par contre, s'il s'avère que l'alerte est justifiée, cela signifie que vous avez un *rootkit* ainsi qu'une faille de sécurité qui a été découverte et exploitée. Méfiance donc !

Surveiller les logs

La plupart des logiciels cités plus haut vous enverront des notifications par mail en cas d'alerte. Cependant, surveiller les *logs* est important, car ils reflètent la « vie » de votre serveur. Les *logs* les plus intéressants sont notamment :

- **/var/log/auth.log** qui contient toutes les tentatives d'accès au serveur. Il peut être utile de filtrer le contenu, par exemple : *cat /var/log/auth.log | grep authentication failure* ;
- **/var/log/message** et **/var/log/syslog** contient un peu de tout (erreurs, *bugs*, informations, etc) ;
- **/var/log/fail2ban** est le *log* d'alerte de *fail2ban*. Cherchez notamment : *cat /var/log/fail2ban | grep ban* ;
- **/var/log/snort/alert** vous indiquera les *logs* d'alertes de *Snort* ;
- **/var/log/rkhunter** pour voir les rapports quotidien de *Rkhunter*. Faites attention aux erreurs trouvées, ce n'est pas bon signe (même si le risque de faux positifs existe ici).

Logwatch

Il est aussi possible d'utiliser des utilitaires qui vous simplifient un peu ce travail de lecture des *logs*. *Logwatch* notamment permet de résumer plusieurs logs afin de ne vous retourner que des anomalies si possible. Cela évite un long et fastidieux travail de recherche.

Installation :

Code : Console

```
apt-get install logwatch
nano /usr/share/logwatch/default.conf/logwatch.conf
```

Spécifiez l'option « MailTo » car *logwatch* envoie ses résumés de *logs* par mail.
Il va normalement s'exécuter tous les jours (*ls -l /etc/cron.daily/ | grep logwatch* pour s'en assurer).

Il peut aussi être intéressant de suivre l'état du réseau et du système (monitoring) afin de détecter par exemple une brusque montée en charge, synonyme de problèmes. Mais comme il est délicat de réagir vite, je vous laisse vous renseigner de vous-mêmes.

Du bon usage de son serveur

Le bon sens

90 % des problèmes informatiques relèvent de l'utilisateur. C'est pourquoi, avant même de penser à sécuriser sa machine, il faut garder en mémoire quelques règles de bon sens :

- interdire les utilisateurs sans mot de passe (ce sont d'énormes failles potentielles) ;
- toujours choisir de bons mots de passe : 8 caractères minimum, pas un mot qui se trouve dans le dictionnaire, si possible des chiffres, des majuscules, des symboles... au besoin, un outil comme *pwgen* vous en génèrera automatiquement des bons (*apt-get install pwgen*) ;
- maintenir son système à jour (*apt-get update* et *apt-get upgrade*) ;
- toujours utiliser *ssh* pour l'accès à distance (et non *telnet* ou des services graphiques, sauf s'ils sont en tunnel à travers

ssh).

Configurer les logiciels

Il peut aussi être bon d'améliorer un peu la sécurité des logiciels installés sur la machine, car il seront en première ligne pour traiter les paquets autorisés.

SSH

En premier lieu, il faut regarder du côté de *ssh*, puisque c'est tout de même un accès direct à votre machine.
Pour ce faire...

Code : Console

```
nano /etc/ssh/sshd_config
```

... et il est conseillé de changer les champs suivants :
- **Port** : le port par défaut est 22... et n'importe quel attaquant le sait. Changer le port force à effectuer un *scan* (ou équivalent) avant de réfléchir à attaquer (attention de bien changer le port dans le *firewall*) ;
- **PermitRootLogin** : mettre à « no » afin d'interdire le *login* en *root* ;
- **AllowUsers** : indique une liste d'utilisateur autorisé à se connecter via *ssh*. Cela peut être utile si vous avez des utilisateurs qui ne sont pas censés se connecter sur la machine.

Et on redémarre :

Code : Console

```
/etc/init.d/ssh restart
```

Apache

Apache - le serveur web le plus courant - donne par défaut de nombreuses informations à quiconque s'y connecte. Vu que cela ne sert à rien que votre serveur web donne au monde votre distribution Linux, autant limiter cela :

Code : Console

```
nano /etc/apache2/apache2.conf
```

Passez *ServerSignature* à « off » et *ServerTokens* à « Prod » pour rendre votre serveur web plus discret.

Code : Console

```
/etc/init.d/apache2 restart
```

Autres

Pour la plupart des logiciels de base, il existe quelques recommandations de prudence. Voici une liste non exhaustive :

service	conseil
mysql	interdire les accès sans mot de passe (on peut exécuter l'utilitaire */usr/bin/mysql_secure_installation* fourni avec *mysql-server*)
FTP	ne surtout pas créer de FTP anonymes
Mail	utiliser un anti-spam (*spamassassin* par exemple) et - si possible - utiliser les connexions sécurisées (SSL ou TLS) offertes par tout serveur de mail qui se respecte

Et bien d'autres, à vous de vous renseigner.

Tester la sureté de son serveur

Il est bien connu que pour éprouver la sécurité de son serveur, le plus simple est encore de se mettre dans la peau d'un pirate. Sans aller jusque-là, il existe néanmoins quelques outils intéressants pour rapidement déterminer s'il existe une grosse faille ou non.

Scanner de port

nmap est le meilleur outil de *scan* de ports : il va tenter d'ouvrir des connexions sur un grand nombre de ports de votre machine afin de déterminer s'ils sont ouverts ou non.

Code : Console

```
apt-get install nmap
```

Comme c'est notre serveur, le mieux est d'effectuer le *scan* le plus incisif (et par conséquent, le moins discret) possible :

Code : Console

```
nmap -v ip_ou_nom_de_la_machine
```

Vous aurez alors la liste des ports ouverts.
Vous pouvez aussi tester un port en particulier avec l'argument *-p port*. Il n'est pas recommandé d'utiliser *nmap* sur quiconque autre que vous-mêmes 

Si vous avez pris des mesures pour bloquer (au mieux) le *scan* de port, il est conseillé dans un premier temps de les désactiver le temps du *scan* (car un port ouvert sans que vous le sachiez est une faille), et dans un second temps de jouer avec *nmap* pour voir si vos règles sont efficaces ou pas (par exemple avec les options *-sS*, *-sN* ou *-sI*, cf le manuel).

Scanner de vulnérabilité

Là, le but est de chercher en général les failles de votre machine. Et il convient de les régler toutes si possible, car l'attaquant peut les trouver aussi bien que vous.

Pour cela, *Nessus* est un des utilitaires les plus performants. Comme le logiciel est propriétaire et qu'il s'utilise via une interface graphique, je vous laisse suivre un tuto détaillé dessus, par exemple :
http://doc.ubuntu-fr.org/nessus
http://www.linux-pour-lesnuls.com/nessus.php

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Connectez-vous Inscrivez-vous
Quel est le rôle du firewall ?

- ◯ Fermer les ports inutilisés
- ◯ Rendre le scan de port inutile
- ◯ Filtrer les paquets autorisés ou non du réseau
- ◯ Empêcher toutes les intrusions

Que fait cette règle du firewall :

Code : Bash

```
iptables -t filter -p tcp --dport 2222 -j ACCEPT
```

?

- ◯ elle autorise le trafic TCP sur le port 2222
- ◯ elle autorise tout le trafic sur le port 2222 sauf en TCP
- ◯ elle bloque le trafic TCP sur le port 2222

En cas de présence d'un *rootkit* sur le serveur, quelle proposition ci-dessous est fausse ?

- ◯ Il existe une faille de sécurité dans la configuration du serveur
- ◯ L'attaquant a pu utiliser la machine à mauvais escient
- ◯ L'attaquant bénéficie d'un accès direct à la machine
- ◯ Un audit approfondi des logs et de l'ensemble du serveur est requis

Quel est selon vous le meilleur mot de passe ci-dessous :

- ◯ anticonstitutionnellement
- ◯ dB_45
- ◯ bossdemerde05
- ◯ axt20mp5

Correction !

Statistiques de réponses au QCM

La sécurité informatique est un vaste domaine dont j'espère vous avoir donné un bon aperçu à travers ces exemples d'applications. Cependant, il existe de bien nombreux autres système de défense et si la sécurisation à outrance est inutile, la connaissance à outrance ne l'est sûrement pas !

Quelques liens intéressants donc :
http://www.linux-france.org/prj/inetdo [...] ite/tutoriel/
http://securite.developpez.com/cours/
Et bien d'autres sur Google.