Serveur Web Apache - SSL - PHP

Debian GNU/Linux

Matthieu Vogelweith

24 août 2009

Résumé

L'objectif de ce document est de détailler l'installation d'un serveur Web Apache [1] complet sous Debian GNU/Linux Etch [2] bien sûr.

Ce document a été rédigé en LaTeX en utilisant l'excellent Vim sous Debian GNU/Linux. Il est disponible aux formats XHTML et PDF. Les sources LaTeX sont disponibles ici : LaTeX

Licence

Historique

- **17-08-2009** : Mise à jour pour Debian Lenny

Table des matières

Table des matières **4**

1 Serveur Web Apache **5**
1.1 Installation . . . 5
1.1.1 Installation des paquets . . . 5
1.1.2 Quelques chemins . . . 5
1.1.3 Premier test . . . 6
1.2 Définitions des Virtual Hosts . . . 6
1.2.1 Création des arborescences . . . 6
1.2.2 Création des fichiers de config . . . 6
1.2.3 Choix du vhost par défaut . . . 7
1.2.4 Et maintenant 8
1.3 Support PHP . . . 8
1.3.1 Installation . . . 8
1.3.2 Configuration . . . 8
1.3.3 Premier test . . . 9

2 Éléments de sécurisation d'Apache **10**
2.1 Masquer l'identité d'Apache . . . 10
2.2 Restrictions d'accès sur un dossier . . . 10
2.3 Les modules inutiles . . . 10
2.4 Le module modsecurity . . . 11
2.5 Le module SSL . . . 11
2.5.1 Activation du module SSL . . . 11
2.5.2 Création des certificats . . . 11
2.5.3 Définition du Virtual Host . . . 12
2.5.4 Cryptage d'un répertoire unique . . . 12

3 Authentification LDAP **15**
3.1 Préparation . . . 15
3.2 Authentification par utilisateur . . . 15
3.3 Authentification par groupe . . . 15

4 Références **16**

Chapitre 1

Serveur Web Apache

1.1 Installation

1.1.1 Installation des paquets

La Etch propose maintenant, en plus de la version 1.3, la version 2.2 d'Apache. Cette nouvelle version a été profondément remaniée et offre maintenant un modèle d'exécution multi-processus et multi-thread sous UNIX, le support natif de l'IPv6, une nouvelle API Apache, une meilleur intégration des modules, ...

Le paquet Debian a subit lui aussi de nombreuses modifications puisqu'il fournit maintenant un grand nombre de modules (dont SSL) ainsi que quelques utilitaires permettant de simplifier grandement les tâches d'administration.

```
# aptitude install apache2
```

Parmi les utilitaires qui viennent d'être installés avec apache, on peut remarquer **a2dismod**, **a2dissite**, **a2enmod** et **a2ensite** qui permettent d'activer ou de désactiver les modules et les différents sites (Virtual hosts).

1.1.2 Quelques chemins

Suivants la méthode d'installation choisie et suivant les distributions, les chemins des fichiers de configuration d'Apache sont différents. Voici les principaux répertoires à connaître pour configurer le serveur Web sous Debian :

- /etc/apache2/apache2.conf : fichier principal de configuration d'Apache 2.
- /etc/apache2/ports.conf : fichier contenant les ports sur lesquels Apache doit écouter.
- /etc/apache2/sites-available/ : dossier contenant les définitions des différents vhosts.
- /etc/apache2/sites-enable/ : dossier contenant les vhosts actifs.
- /etc/apache2/mods-available/ : dossier contenant les différents modules installés.
- /etc/apache2/mods-enable/ : dossier contenant les modules actifs.
- /etc/apache2/ssl/ : fichiers relatifs à la configuration du mod_ssl.
- /var/www/ : Répertoire racine du site par défaut.
- /var/log/apache2/ : Répertoire contenant les logs d'apache.

1.1.3 Premier test

Avec la configuration par défaut, il est déjà possible de faire un premier test pour vérifier que tout fonctionne correctement. Pour cela, il suffit d'entrer l'adresse IP du serveur dans un navigateur :

```
http://adresse_ip_du_serveur/
```

En principe, la page a été redirigée vers http://adresse_ip_du_serveur/apache2-default/, répertoire qui contient la page d'accueil d'Apache dans plusieurs langues. Pour pouvoir héberger et visualiser un ou plusieurs site sur ce serveur à la place de cette page d'accueil, il faut maintenant définir un ou plusieurs Virtual Hosts.

1.2 Définitions des Virtual Hosts

La manipulation des VirtualHosts a beaucoup évoluée et est maintenant beaucoup plus simple. Comme indiqué précédemment,la définition des virtualhosts se fait dans le répertoire /etc/apache2/sites-available/. Pour faire les choses dans les règles de l'art, chaque site (qui peut contenir plusieurs Vhosts) est définit dans un fichier qui lui est propre. On peut par exemple héberger deux sites Web sur le serveur : **site1.example.org** et **site2.example.org**. On suppose ici que la configuration DNS associe bien les 2 noms à l'adresse IP du serveur Web.

1.2.1 Création des arborescences

Avant de définir les vhosts dans le configuration d'Apache, il faut créer les arborescences qui accueilleront les sites Web. Pour faciliter le suivi des logs, il peut être intéressant de stocker les logs de chaque site dans des fichiers différents. Sur une machine Debian, on a souvent pour habitude de mettre les sites Web dans /var/www.

```
# mkdir /var/www/site1
# mkdir /var/www/site2
# chown -R www-data:www-data /var/www/site1 /var/www/site2
```

Il faut apporter un soin particulier aux droits de ces répertoires. Il doivent être accessibles en lecture par Apache (peut-être en écriture pour certains sous-répertoires), en lecture-écriture pour les webmasters et c'est tout !

1.2.2 Création des fichiers de config

Il faut donc créer deux nouveaux fichiers que l'on appellera site1 et site2 dans /etc/apache2/sites-available/. Pour simplifier la tâche, on peut commencer par des vhosts très simple.

```
# cd /etc/apache2/sites-available/
# vim site1 site2
```

Le premier fichier site1 peut par exemple contenir les lignes suivantes :

```
# Virtualhost pour site1.example.org
<VirtualHost *:80>
```

```
    ServerName site1.example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /var/www/site1/
    ErrorLog /var/log/apache2/site1_error.log
    CustomLog /var/log/apache2/site1_access.log combined

</VirtualHost>
```

Le fichier site2 est quasiment identique :

```
# Virtualhost pour site2.example.org
<VirtualHost *:80>

    ServerName site2.example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /var/www/site2/
    ErrorLog /var/log/apache2/site2_error.log
    CustomLog /var/log/apache2/site2_access.log combined

</VirtualHost>
```

Il reste maintenant à activer ces 2 sites, tout simplement avec les commandes suivantes :

```
# a2ensite site1
# a2ensite site2
# /etc/init.d/apache2 reload
```

1.2.3 Choix du vhost par défaut

Si tout c'est passé correctement, les deux sites devraient être accessibles via leurs URL respectives.

```
http://site1.example.org
http://site2.example.org
```

Notons que si l'on saisi l'adresse IP du serveur (ou un nom DNS ne correspondant pas à un vhost) dans la barre d'adresse du navigateur, on est redirigé vers le vhost site1. Ceci est simplement dû au fait qu'Apache charge les différents sites dans l'ordre alphabétique et que site1 est devant site2.

Pour choisir le vhost par défaut, c'est à dire le vitualhost utilisé lorsqu'on accède au serveur autrement que par site1.example.org ou site2.example.org, il faut créer un nouveau vhost "_default_". Ce vhost sera alors utilisé à chaque fois que l'url demandée ne correspond à aucun des autres vhosts définis. Supposons que le vhost par défaut est www.example.org, la config sera la suivante :

```
# Virtualhost principal du site
<VirtualHost _default_:80>

    ServerName www.example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /var/www/default/htdocs
    ErrorLog /var/www/default/logs/error_log
    TransferLog /var/www/default/logs/access_log
```

```
</VirtualHost>
```

Pour visualiser l'ordre des vhosts :

```
# apache2ctl -S
```

1.2.4 Et maintenant ...

- Vhosts ok.
- MAJ avec util www-data
- Serveur pas très intéressant : pas de PHP, pas de DB.

1.3 Support PHP

1.3.1 Installation

Pour qu'Apache soit capable d'interpréter le language PHP, il faut installer le module php5. Attention, il faut bien spécifier que l'on veut le module PHP pour Apache 2. Dans le cas contraire, les dépendances font qu'aptitude va installer le module pour Apache 1 et Apache 1 lui-même :

```
# aptitude install libapache2-mod-php5 php5
```

En principe, le module est activé après l'installation. On peut tout de même vérifier en examinant le contenu du répertoire /etc/apache2/mods-enabled. Celui-ci doit contenir deux fichiers (qui sont en fait des liens symboliques) nommés php4.conf et php4.load. Si ces fichiers sont absent, exécuter les commandes suivantes :

```
# a2enmod php5
```

Pour que les modifications prennent effet, il faut recharger la configuration d'apache. Notons qu'il n'est pas utile de re-démarrer Apache, il suffit simplement de recharger les fichiers de configuration :

```
# /etc/init.d/apache2 reload
```

1.3.2 Configuration

Pour inclure du code PHP dans une page écrite (et validée !) en XHTML, il faut faire une petite modification dans le fichier de configuration de PHP. Le problème vient du fait que la page XHTML commence par une balise du type "< ?xml version="1.0" ?>". Pour qu'Apache n'interprète pas cette commande comme du PHP, il désactiver la balise simple < ? pour n'autoriser que la balise longue < ?php. Ceci se fait au debut du fichier /etc/php4/apache/php.ini : il faut mettre le paramètre short_open_tag à Off.

```
short_open_tag = Off
```

Au niveau sécurité, on peut modifier ces quelques variables pour interdire les variables globales, ne pas annoncé la présence de PHP dans les headers d'apache, ... Le détail du rôle de chacune des variables est très bien expliqué dans le fichier de conf.

```
expose_php = Off
display_errors = Off
log_errors= On
register_globals = Off
error_log = syslog
ignore_repeated_errors = On
allow_url_fopen = Off
```

Comme d'habitude, il faut recharger la configuration d'apache pour que les modifications soient prises en compte :

```
# /etc/init.d/apache2 reload
```

1.3.3 Premier test

Pour vérifier le bon fonctionnement du serveur Web et de l'interpréteur PHP, on peut éditer la classique page de test nommée test.php et qui contient les lignes suivantes :

```
<?php
        // Page de test pour le serveur Web
        phpinfo();
?>
```

Il faut ensuite copier cette page dans la racine du serveur (/var/www/) et tester le tout en entrant l'url suivante dans un navigateur Web sur une machine distante.

```
http://nom_ou_ip_du_serveur/test.php
```

La page obtenue doit présenter un grand nombre d'informations sur le serveur Web qui vient d'être installé. Dans le cas contraire, revoir la section précédente qui détaille l'installation du module PHP.

Chapitre 2

Éléments de sécurisation d'Apache

2.1 Masquer l'identité d'Apache

Pour ne pas dévoiler l'identité exacte d'apache, et donc d'éventuelles failles de sécurité, il faut positionner les variables **ServerSignature** et **ServerTokens**. La première défini les informations envoyées avec les pages générées par le serveur (page d'erreur, listings FTP, ...) et la seconde défini les informations retournées dans les entêtes HTTP.

```
ServerSignature Off
ServerTokens Prod
```

La variable ServerTokens peut prendre les valeurs suivantes : Full, OS, Minor, Minimal, Major et Prod (Prod étant le mode le plus discret, il renvoit "Apache"). Merci à Samuel Lefol pour ces informations.

2.2 Restrictions d'accès sur un dossier

Les directives de configuration d'Apache permettent de mettre en place des restrictions d'accès relativement évoluée sur des répertoires. Il est par exemple possible de faire des restrictions au niveau IP ou au niveau utilisateur (authentification).

```
<Directory /var/www/privatedir>
    Order Deny,Allow
    Allow from 192.168.1
    Deny from All
</Directory>
```

La configuration décrite ci-dessus restreint l'accès aux machines provenant du réseau 192.168.1.0.

Pour obtenir des restrictions plus fine, il est possible de restreindre l'accès en fonction d'une authentification utilisateur. Le chapitre suivante décrit la mise en place de cette authentification en utilisant un annuaire OpenLDAP.

2.3 Les modules inutiles

Désactivation de tous les modules inutiles

2.4 Le module modsecurity

2.5 Le module SSL

Les services Web demandent souvent une authentification de la part des utilisateurs. Dans ce cas, ce dernier va fournir un couple login/mot de passe qui devra transiter sur le réseau entre le client et le serveur. Pour que ces identifiants ne circulent pas en clair sur le réseau, Apache offre la possibilité de chiffrer les échanges à l'aide d'OpenSSL.

2.5.1 Activation du module SSL

L'activation du module SSL se fait très simplement avec l'utilitaire a2enmod.

```
# a2enmod ssl
```

Ensuite, il faut savoir qu'un serveur Web qui diffuse des pages en SSL utilise le port 443 (au lieu du port 80 pour les pages non cryptées). Pour que apache "écoute" sur le port 443, il faut donc rajouter la ligne suivante dans /etc/apache2/ports.conf :

```
# listen 443
```

Comme pour tous les autres modules, il est maintenant nécessaire de recharger la configuration d'Apache pour que les modifications soient prises en compte :

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

Avant de définir le ou les sites qui devront être chiffrés avec SSL, Apache doit disposer d'un certificat SSL qui permettra de s'assurer de son authenticité et de chiffrer/déchiffrer les communications.

2.5.2 Création des certificats

Debian fournit maintenant un outil permettant de simplifier grandement la génération des certificats SSL : **make-ssl-cert**. Il est disponible en installant simplement le paquet ssl-cert :

```
# aptitude install ssl-cert
```

make-ssl-cert utilise un système de template pour générer le certificat souhaité. Par exemple, pour générer le certificat du serveur HTTP, il suffit d'exécuter la commande suivante :

```
# make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/ssl/certs/apache.pem
```

Le script utilise alors debconf pour demander les renseignements nécessaires à la génération du certificat. Par exemple, pour le certificat du serveur HTTP :

```
Country Name : FR
State or Province Name : France
Locality Name : Ma_ville
Organisation Name : Ma_societe
Organisational Unit Name : HTTP Server
Host Name : www.example.org
Email Address : webmaster@example.org
```

Attention, lors de la configuration du certificat, le champ "Host Name" est très important. Il doit toujours correspondre au FQDN qui sera utilisé pour accéder au service. Dans le cas contraire, la plupart des clients vont générer une alerte à chaque négociation TLS avec le serveur.

Note : Le certificat ainsi généré est un certificat auto signé qui contient le certificat ET la clé privée dans le même fichier.

2.5.3 Définition du Virtual Host

Le cryptage SSL doit obligatoirement ce faire sur un vhost complet. Dans un premier temps, on peut par exemple rendre le vhost site1 également disponible en HTTPS :

```
# cd /etc/apache2/sites-available/
# vim site1
```

Il suffit de recopier la définition du vhost site1, de modifier le port sur lequel ce virtual host écoute et d'activer le support SSL. La définition du vhost doit donc ressembler à ceci :

```
# Virtualhost securise pour site1.example.org
<VirtualHost *:443>

    ServerName site1.example.com
    ServerAdmin webmaster@example.com

    # Activation du support SSL
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/apache.pem
    SSLCertificateKeyFile /etc/ssl/certs/apache.pem

    DocumentRoot /var/www/site1/htdocs
    ErrorLog /var/www/site1/logs/error_log
    TransferLog /var/www/site1/logs/access_log

</VirtualHost>
```

Après rechargement de la configuration d'Apache, le site devrait être accessible via l'URL "https://site1.example.org/".

2.5.4 Cryptage d'un répertoire unique

Il arrive fréquemment qu'une rubrique d'un site contienne des données un peu confidentielle que l'on souhaite protéger. Une technique consiste à créer un vhost indépendant pour cette rubrique. Une autre méthode, celle expliquée dans ce paragraphe, consiste à ne chiffrer que le répertoire à protéger. Dans l'exemple suivant, on suppose que seul le sous-répertoire securdir du site1 est à protéger.

Dans un premier temps il est bon d'indiquer dans la config générale d'apache (donc en dehors des vhosts) que ce dossier doit absolument être chiffré. Notons qu'il faut bien utiliser la directive

<Directory> et non la directive <Location>. De cette façon, le répertoire ne pourra être afficher par aucun vhost s'il n'est pas en mode SSL :

```
# Protection du repertoire securdir independamment du vhost
<Directory /var/www/site1/securdir/>
    SSLRequireSSL
</Directory>
```

Notons qu'avec cette configuration, si l'on tente d'accéder à ce dossier via l'URL "http://site1.example.org/securdir/" le serveur retourne une erreur 403, mais ne redirige pas vers la page en HTTPS. Cependant si on y accéde via l'URL "https://site1.example.org/securdir/", la page est affichée correctement puisqu'on est en HTTPS.

Il y a donc deux choses à faire :

1. rediriger le repertoire securdir du vhost "normal" vers le vhost sécurisé.
2. rediriger toutes les pages différentes de securdir du vhost sécurisé vers le vhost normal.

Pour cela on peut utiliser le mod_rewrite d'Apache. L'activation de ce module ce fait toujours très simplement avec l'utilitaire a2enmod :

```
# a2enmod rewrite
```

Le mod_rewrite permet de "ré-écrire" l'URL demandée par le client sous certaines conditions. Dans ce cas précis, nous voulons rediriger toutes les URL du vhost normal contenant le mot "securdir" vers le vhost sécurisé. Ce module utilise des expressions régulières standard pour définir les conditions de redirection.

Le premier point énoncé ci-dessus se résout donc en ajoutant les lignes suivantes dans le vhost "normal" du site1 :

```
# Virtualhost pour site1.example.org
<VirtualHost *:80>

    ServerName site1.example.com
    ServerAdmin webmaster@example.com

    # On redirige le repertoire securdir vers le vhost HTTPS
    RewriteEngine on
    RewriteCond %{SERVER_PORT} !^443$
    RewriteCond %{REQUEST_URI} /securdir/
    RewriteRule ^/(.*)$ https://%{SERVER_NAME}%{REQUEST_URI} [R=301,L]

    DocumentRoot /var/www/site1/htdocs
    ErrorLog /var/www/site1/logs/error_log
    TransferLog /var/www/site1/logs/access_log

</VirtualHost>
```

Le second point énoncé ci-dessus se résout donc en ajoutant les lignes suivantes dans le vhost "sécurisé" du site1 :

```
# Virtualhost securise pour site1.example.org
<VirtualHost *:443>

    ServerName site1.example.com
    ServerAdmin webmaster@example.com
```

```
    # Activation du support SSL
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/apache.pem
    SSLCertificateKeyFile /etc/ssl/certs/apache.pem

    # On redirige tout sauf securdir vers le vhost HTTP
    RewriteEngine on
    RewriteCond %{SERVER_PORT} !^80$
    RewriteCond %{REQUEST_URI} !/securdir/
    RewriteRule ^/(.*)$ http://%{SERVER_NAME}%{REQUEST_URI} [R=301,L]

    DocumentRoot /var/www/site1/htdocs
    ErrorLog /var/www/site1/logs/error_log
    TransferLog /var/www/site1/logs/access_log

</VirtualHost>
```

Si tout c'est passé correctement et quelque soir l'URL saisie dans le navigateur, le répertoire /securdir doit toujours être affiché en HTTPS et le reste du site1 doit toujours être affiché en HTTP.

Chapitre 3

Authentification LDAP

3.1 Préparation

- Authentification basée sur OpenLDAP
- Activation du module Apache

```
# a2enmod authnz_ldap
```

3.2 Authentification par utilisateur

Pour authoriser tous les utilisateurs user1 et user2 :

```
<Directory /var/www/site1/authdir>
    AuthType Basic
    AuthBasicProvider ldap
    AuthzLDAPAuthoritative off
    AuthName "Authentification"
    AuthLDAPUrl ldap://ldap.example.org:389/ou=Users,dc=example,dc=org
    AuthLDAPGroupAttribute memberUid
    AuthLDAPGroupAttributeIsDN off
    require ldap-user user1 user2
</Directory>
```

3.3 Authentification par groupe

Pour authoriser tous les membres du groupe **groupe_de_test** :

```
<Directory /var/www/site1/authdir>
    AuthType Basic
    AuthBasicProvider ldap
    AuthzLDAPAuthoritative off
    AuthName "Authentification"
    AuthLDAPUrl ldap://ldap.example.org:389/ou=Users,dc=example,dc=org
    AuthLDAPGroupAttribute memberUid
    AuthLDAPGroupAttributeIsDN off
    require ldap-group cn=groupe_de_test,ou=Groups,dc=example,dc=org
</Directory>
```

Chapitre 4

Références

[1] Site officiel d'apache. www.apache.org.

[2] Site officiel du projet debian. www.debian.org.

[3] Site officiel de php. www.php.net.

[4] Site officiel de openssl. www.openssl.org.