



# Protéger ses données sur un serveur Apache

Le 4 mai 2008, à 18:55 par Ulhume...

Protéger ses données exposées sur le WEB n'est pas toujours évident et comporte quelques pièges qui donnent une illusion de sécurité alors qu'il n'en est rien. Ce tutoriel ne va pas, loin de là, couvrir tous les aspects du sujet, mais juste tenter d'apporter quelques bases permettant de se protéger à travers quelques cas pratiques.

## .htaccess vs configuration

Pour tout ce qui suit, ou presque, existe deux méthodes de paramétrages. La plus simple consiste à placer dans le dossier web à protéger un fichier `.htaccess` qui contiendra une configuration spécifique à ce dossier. Le contenu d'un tel dossier pourrait être :

```
AuthType Basic
AuthName "Accès protégé"
AuthUserFile "/mon_chemin_vers_mots_de_passe/passwd"
require valid-user
```

*Exemple de fichier .htaccess*

Cette méthode a le mérite de la simplicité et protège autant le dossier lui-même que les sous-dossiers qu'il contient, sauf si un autre fichier `.htaccess` y contrevient.

L'autre méthode, sûrement plus pérenne, est de modifier directement un des fichiers de configuration Apache. Ces derniers se trouvent généralement en `/etc/httpd` et vous pouvez soit modifier `/etc/httpd/httpd.conf`, soit créer votre propre fichier `.conf` dans le dossier `/etc/httpd/conf.d`. A noter que les localisations peuvent changer d'une distribution à l'autre. Dans tous les cas la syntaxe est à peu près la même que pour un `.htaccess` :

```
<directory "/mon/dossier/a/protéger">
AuthType Basic
AuthName "Accès protégé"
AuthUserFile "/mon_chemin_vers_mots_de_passe/passwd"
require valid-user
</directory>
```

*Exemple de fichier '/etc/httpd/conf.d/ma\_protection.conf'*

La seule différence ici est donc la balise `Directory` qui encadre le code qui aurait été dans un fichier `.htaccess` et qui indique le chemin absolu à protéger, comme la position d'un `.htaccess` l'aurait fait. Une autre différence cependant tient à la manière dont apache gère les deux types de fichier. En effet, un `.htaccess` est affectif tout de suite, dès que le fichier est créé et contient les données. En revanche un fichier de configuration nécessite le redémarrage du serveur Apache par un redémarrage du serveur apache.

Maintenant le choix `.htaccess` ou fichier de configuration vous appartient. Tout ce qui suit devrait

fonctionner dans les deux cas.

## Fichier de mots de passe

Si nous voulons qu'Apache protège notre dossier, il nous faut lui fournir une liste d'utilisateurs, voire de groupes d'utilisateurs. C'est ce que fait le code suivant en créant d'abord un répertoire et en changeant ses droits, puis en ajoutant avec la commande Apache `htpasswd` le premier utilisateur (le chemin du dossier et le nom du fichier n'ont aucune importance) :

```
mkdir [1] var secured passwords
chown [2] apache:apache var secured passwords
htpasswd -c var secured passwords gaston
```

*Création du fichier et ajout d'un utilisateur*

Pour l'ajout des utilisateurs suivants, le fichier étant déjà créé, le paramètre `-c` n'est plus nécessaire :

```
htpasswd var secured passwords martine
htpasswd var secured passwords robert
```

*Ajout de l'utilisateur 'martine'*

Un utilisateur préalablement créé, pourra être retiré de la manière suivante :

```
htpasswd -D var secured passwords martine
```

Maintenant si nous vidons le contenu du fichier `/var/secured/passwords`, nous aurons ceci :

```
gaston:v/nTde8180.jQ
martine:Cse8VK.qdaFj.
robert:wKahxQnVtqyS6
```



Si vous n'avez pas accès à votre serveur en ligne de commande, rien ne vous empêche d'utiliser `htpasswd` à partir d'une autre machine et de télécharger ensuite le fichier résultant. Il existe aussi pas mal de scripts PHP sur le net permettant de fabriquer sur votre serveur une page de génération de mots de passe (la partie de droite). En revanche les générateurs de mot de passe en ligne sont **simplement à proscrire** pour d'évidentes raisons de sécurité.

Nous avons donc maintenant les utilisateurs, leur mot de passe, il ne nous reste plus qu'à définir des groupes, même si c'est optionnel. Si nous voulons indiquer que l'utilisateur `gaston` appartient au groupe `webmasters` il suffit de créer un second fichier `/var/secured/groups` contenant les données suivantes :

```
webmasters: gaston martine
visiteurs : robert
```

Maintenant que nous avons notre fichier de mots de passe et notre fichier de groupes, il ne nous reste plus qu'à protéger nos dossiers avec.

### Protéger un répertoire physique

Pour ce faire, nous allons, dans le dossier à protéger, créer un fichier `.htaccess`. Mais comme nous l'avons vu plus haut, ceci peut aussi se trouver dans la configuration apache, en dure.

```
<code>
```

```
AuthName "Bienvenue sur mon serveur web"
AuthType Basic
AuthUserFile "/var/secured/passwords"
require valid-user
```

La syntaxe est ici simple à comprendre. `AuthName` est le message qui sera affiché dans la boîte de dialogue de saisie de l'identifiant et du mot de passe. Gardez cela en tête pour ne pas y mettre de chose trop explicites comme "Bienvenue dans le site contenant tous mes numéros de carte bleue".

Le second paramètre, `AuthType` indique à Apache d'utiliser le protocole `Basic` pour authentifier l'utilisateur. Comme son nom le laisse présager, ce protocole est le plus simple, le plus compatibles avec tous les navigateurs, mais aussi le moins sécurisé comme nous le verrons un peu plus loin. D'autres types d'authentification existent sous Apache par l'intermédiaire de modules spécifiques, comme nous allons le voir aussi plus loin.

Ensuite nous avons `AuthUserFile` qui indique le chemin vers le fichier que nous avons créé dans le chapitre précédent, et contenant donc nos utilisateurs et leurs mots de passe.

Enfin, la règle d'authentification à proprement parler, `require valid-user`. Elle indique à Apache que tout utilisateur ayant réussi à rentrer son mot de passe est autorisé à rentrer dans le dossier. Si nous voulions spécifiquement autoriser Robert à rentrer, nous aurions mis `require user robert`, et pour Gaston et Martine, cela aurait été `require user gaston martine`.

Vu que nous avons créé un fichier de groupes d'utilisateur au chapitre précédent, autant en profiter. Et si nous voulions n'autoriser que le groupe `webmaster` à rentrer, nous aurions cette fois un fichier `.htaccess` comme ceci :

```
AuthName "Bienvenue sur mon serveur web"
AuthType Basic
AuthUserFile "/var/secured/passwords"
AuthGroupFile "/var/secured/groups"
require group webmaster
```

Plus complet, les deux chemins vers les deux fichiers mots de passe et groupes sont indiqués (apparition du mot clef `AuthGroupFile`).

Enfin il est possible de panacher tout cela avec des choses du genre : `require user robert, group webmaster`. Ceci indique à apache que seuls les membres du groupe `webmaster` sont autorisés, à l'exception de robert.

## Utiliser un serveur LDAP

Comme je le disais plus haut, il y a de nombreux modules liés à l'authentification disponibles pour Apache sous la forme de modules. Par exemple si vous vouliez utiliser un protocole [Digest](#) [3] au lieu du simple `Basic`, vous utiliseriez le module [mod\\_auth\\_digest](#) [4].

De même différents modules permettent de vérifier les utilisateurs, groupes et mots de passe sur une base de données plutôt qu'un fichier texte, ou encore un serveur `ldap` [5] par le biais du module `mod_authnz_ldap` [6]. Il dépend du module Apache d'accès à LDAP, `mod_ldap` qu'il faut aussi installer. Ce module permet une authentification `Basic` adossée à une base LDAP. Un bloc de configuration pour un tel système se présente comme cela

```
AuthType Basic
AuthName "Bienvenue sur mon serveur web"
AuthBasicProvider ldap
AuthLDAPURL ldap://localhost:389/dc=mon-domaine,dc=com
AuthLDAPGroupAttribute memberUid
AuthLDAPGroupAttributeIsDN off
Require ldap-group cn=webmasters,ou=Group,dc=mon-domaine,dc=com
```

*.htaccess via LDAP*

Comme vous le voyez l'esprit est à peu près le même que précédemment. La ligne `AuthBasicProvider` indique à apache de se baser sur un serveur LDAP plutôt qu'un fichier. Les coordonnées du serveur apparaissent avec la ligne `AuthLDAPURL` qui indique le nom de la machine, le port en écoute, ainsi que le domaine pris en charge.

Les deux attributs suivants permettent au module apache de se repérer dans les enregistrements LDAP et d'y trouver les utilisateurs. Le paramétrage que j'utilise ici se base sur la disposition standard d'une base LDAP sous Unix et est à adapter pour un ActiveDirectory sous Windows, par exemple.

Enfin la dernière ligne indique à apache de n'autoriser à rentrer que le groupe ldap spécifié (ici `webmasters`). L'utilisateur de `ldap-user` (ex. `Require ldap-user Gaston`) permettrait de n'autoriser qu'un utilisateur spécifique. Et ici aussi, le panachage est autorisé en séparant les éléments par des virgules.



Utiliser les mêmes mots de passe pour un service web public et des utilisateurs locaux présente un risque car le protocole `Basic` ne permet pas de se protéger d'une attaque en "brut force". C'est-à-dire qu'un vilain peut faire autant de tentatives qu'il le désire sans qu'apache ne l'en empêche et ainsi trouver un mot de passe critique, même s'il met des jours à le faire. Le risque n'est pas énorme mais doit être pris en compte dans la qualité des mots de passe utilisables sur le WEB.

A ce stade, nous avons déjà pas mal d'outils pour protéger nos données mais si nous nous arrêtons à cela, cela ne servirait à rien... En effet, ce type de sécurité est aussi valable qu'une porte blindée sur des murs en cartons. Car le gros inconvénient du mode `basic` est que les mots de passe circulent **en clair**. Protéger par mot de passe un dossier accessible avec le protocole HTTP est donc très dangereux. Une simple écoute active (mode Promisc) du réseau, si par exemple vous utilisez cela de votre bureau, dévoile aussi sûrement vos secrets que si vous les aviez envoyés à tout le monde par courriel. Pour parfaire notre protection il est donc obligatoire de mettre en œuvre le protocole HTTPS, ce que j'aborderais dans un autre article.

## Empêcher le téléchargement des images à partir d'un autre site

Un problème classique est de se retrouver avec une bande passante "volée" parce qu'un Malotru a eu l'indécatesse de mettre vos images en référence directe sur son site. Ici, pas question de mettre un mot

de passe qui bloquerait les utilisateurs légitimes. Les techniques précédentes sont donc inutiles. A la place nous allons utiliser les variables dont nous disposons au sein d'Apache et particulièrement du `Referer`.

Pour ceux qui ne le savent pas, un navigateur WEB a pour obligation de transmettre au serveur l'adresse WEB de la page qui contient le lien qui a permis d'arriver sur notre site. Cette indiscretion est déjà bien connu et c'est elle qui permet de savoir d'où viennent nos visiteurs. Cette adresse est le fameux `Referer`.

Ce qui est un peu moins connu c'est que chaque élément lié à une page de notre site (ex. une image ou une feuille de style), est demandée au serveur distant avec lui aussi un `Referer`. Sauf que cette fois, c'est l'adresse de notre propre site qui est passé par le navigateur, ce qui est somme toute très logique.

En d'autres termes, toute demande d'image n'ayant pas notre site pour `Referer`, est sûrement l'objet d'un vol de bande passante? C'est cette caractéristique que nous allons exploiter avec ce code qui peut aussi bien être, comme toujours, dans un `.htaccess` qu'un fichier de configuration Apache.

```
SetEnvIfNoCase Referer "^http://www\.monsite\.net/" acces_local=1
<FilesMatch>.(gif|jpg|png|pdf|css|js)">
Deny from all
Allow from env=acces_local
</FilesMatch>
```

Ce qui se passe c'est que si le `Referer` est bien notre site, nous positionnons une variable `acces_local` à 1. Ensuite nous ajoutons une règle qui interdit le téléchargement des images (`Deny from all`) sauf pour les requêtes qui ont la variable `acces_local` définie. Voilà, c'est tout. La directive `FilesMatch` est elle là pour n'opérer cette sécurité que sur les fichiers images, feuilles de style, scripts, etc. Il serait en effet un peu idiot d'appliquer cette règles au fichier `.php` par exemple, cela interdirait systématiquement l'accès du site à tout le monde?.

## Empêcher certains navigateurs

Vous aurez remarqué dans l'exemple précédent la syntaxe un peu étrange du paramètre de la directive `SetEnvIfNoCase`. On appelle cela une [expression régulière](#) [7]. Une expression régulière est un e sorte de motif permettant de décrire une chaîne de caractère attendue. On peut ainsi créer des expressions régulières décrivant un numéro de téléphone ou une adresse courriel. Dans l'exemple précédent, elle signifiait "toute les URL qui commencent par (symbole ^) `http://www.monsite.net/`". Les `\.` étaient là pour indiquer que l'on voulait que le vrai caractère `.` soit utilisé, car sinon, ce symbole a un sens particulier dans une expression régulière.

Maintenant imaginons qu'un vilain spammeur qui me fasse des misères et que son adresse IP varie de `81.95.144.X` à `81.95.147.X`. Je peux le bloquer par la configuration suivante :

```
SetEnvIf Remote_Addr "^81\.95\.14[4-7]\." bad_bot
Deny from env=bad_bot
```

C'est à peu près le même exemple que précédemment sauf que cette fois on utilise, non pas `Referer` mais `Remote_Addr` qui est l'adresse IP du navigateur client. Je ne vais pas faire ici un cours sur les expressions régulières, il y a de [magnifique tutoriaux](#) [8] sur ce sujet.

Juste un dernier exemple, si cette fois c'est la chaîne d'identification ([User Agent](#) <sup>[9]</sup>) d'un navigateur que je veux bloquer, par exemple un bot malveillant :

```
BrowserMatchNoCase ".*HTTrack*" bad_bot
Deny from env=bad_bot
```

Là, j'utilise une directive un peu spéciale qui a le même rôle que les précédentes mais cette fois sur la variable `User_Agent` sans distinction de majuscules/minuscules.

Bien évidemment, vous pouvez mettre une liste complète de définitions **avant** d'insérez votre directive `Deny` :

```
BrowserMatchNoCase ".*HTTrack*" bad_bot
BrowserMatchNoCase ".*ICC-Crawler*" bad_bot
BrowserMatchNoCase ".*Nutch*" bad_bot
SetEnvIf Remote_Addr "^82\.246\.40\.31" bad_bot # un gars de free...
SetEnvIf Remote_Addr "^195\.69\.171\.86" bad_bot # mirotel.net
Deny from env=bad_bot
```

Cette méthode n'est pas aussi fiable que je le voudrais car les adresses change, et les chaînes d'identification de navigateur aussi. Mais au moins est-ce un moyen de contrer efficacement une attaque ponctuelle. Pour une liste complète des directives touchant à la définition de variables, je vous renvoie à la [documentation d'Apache](#) <sup>[10]</sup>.

## Protéger par adresse IP

Même si cela fonctionne, passer par les variables pour interdire une IP avec des expressions régulières est un peu sauvage pour un usage courant. Si vous voulez pour votre dossier, ne soit accessible que pour une seule IP, ou un groupe d'IP, nous pouvons plus simplement utiliser la directive `Deny From`. Par exemple ici, pour n'autoriser que l'IP 192.168.0.10 nous utiliserons la configuration suivante :

```
Deny from all
Allow from 192.168.0.10
```

Nous pouvons assouplir la règle en enlevant un group de chiffre à la fin de l'IP. Par exemple `Allow from 192.168.0` autorise les adresses allant de 192.168.0.0 à 192.168.0.255, à se connecter.

## Mixer les plaisirs

Pour finir, un petit mélange. Imaginons que nous protégeons notre dossier par mot de passe via LDAP mais qu'en même temps nous ayons besoins que les clients se connectant d'un adresse IP spécifique, puisse le faire sans mot de passe. Cela nous donnerais le code suivant :

```
Allow from 192.168.0.10
AuthType Basic
AuthName "Acces au dossier"
AuthBasicProvider ldap
AuthLDAPURL ldap://localhost:389/dc=mon-domaine,dc=com
AuthLDAPGroupAttribute memberUid
```

```
AuthLDAPGroupAttributeIsDN off
Require ldap-group cn=webmaster,ou=Group,dc=mon-domaine,dc=com
Satisfy Any
```

Rien d'inconnu maintenant dans ces syntaxes, à l'exception de la dernière ligne, qui fait toute la différence. En effet, elle indique à Apache que l'une ou l'autre des conditions suffit à autoriser l'accès : soit l'IP, soit le mot de passe.

## Conclusion

L'article est déjà long et nous sommes loin d'avoir totalement couvert le sujet. J'espère seulement que cela vous donnera assez de pistes pour trouver la solution qui convient le mieux à vos besoins. L'agressivité et l'automatisation des attaques comme en témoignent souvent les fichiers de traces, prouvent que ce genre de précaution est simplement obligatoire. Il n'y a à ma connaissance pas de moyens systématique de valider qu'un serveur est correctement fermé, et ce qui est réellement ouvert. Google permet par le biais d'une requête de type `site:mon-site.com` d'avoir la liste de ce qu'il voit mais cela se limite à ce qui a été lié, quelque part, sur le Web, ce qui est loin d'être suffisant.

Car croire que le seul fait de ne pas publier sur le web le lien vers un dossier le rend invisible est pure crédulité comme en témoigne [cette ânerie là](#)<sup>[11]</sup>, que j'avais trouvé par hasard, en bidouillant l'URL d'annulation d'un courriel de pub, sans l'aide google.

### Liens:

[1] <http://pwet.fr/man/linux/commandes/mkdir>

[2] <http://pwet.fr/man/linux/commandes/chown>

[3] <http://artisan.karma-lab.net/> [http://en.wikipedia.org/wiki/Digest\\_access\\_authentication](http://en.wikipedia.org/wiki/Digest_access_authentication)

[4] <http://artisan.karma-lab.net/> [http://httpd.apache.org/docs/2.0/mod/mod\\_auth\\_digest.html](http://httpd.apache.org/docs/2.0/mod/mod_auth_digest.html)

[5] <http://fr.wikipedia.org/wiki/Ldap>

[6] [http://httpd.apache.org/docs/2.2/mod/mod\\_authnz\\_ldap.html](http://httpd.apache.org/docs/2.2/mod/mod_authnz_ldap.html)

[7] [http://fr.wikipedia.org/wiki/expression régulière](http://fr.wikipedia.org/wiki/expression_régulière)

[8] <http://www.regular-expressions.info/>

[9] [http://fr.wikipedia.org/wiki/User Agent](http://fr.wikipedia.org/wiki/User-Agent)

[10] [http://httpd.apache.org/docs/2.0/mod/mod\\_setenvif.html](http://httpd.apache.org/docs/2.0/mod/mod_setenvif.html)

[11] <http://www.zataz.com/news/16768/SOS-Racisme-sauver-des-pirates.html>