Configuration Apache et PHP

Table des matières

1 Configuration Apache

1.1 Version Apache

Il est recommandé de supprimer la diffusion de la version d'**apache** :

```
ServerTokens Prod
ServerSignature Off
```

1.2 Listing des répertoires

Il est recommandé de désactiver le listing des répertoires (**Indexes**) permettant d'obtenir la liste des fichiers contenus dans les répertoires qui ne contiennent pas un des fichiers par défaut :

```
<Directory /opt/apache/htdocs>
Options -Indexes
Order allow,deny
Allow from all</Directory>
```

1.3 ETag fichiers

Il est recommandé de désactiver la fonction **ETag** permettant d'obtenir des informations sur les fichiers

```
FileETag None
```

1.4 Exécution sous une identité dédiée

Il est recommandé d'exécuter **apache** sous une identité dédiée :

```
User apache
Group apache
```

1.5 Désactiver les configurations locales

Il est recommandé de désactiver la prise en compte des fichiers de configuration locale (**.htaccess**) :

```
<Directory />
AllowOverride None
</Directory>
```

1.6 Méthodes autorisées

Il est recommandé de désactiver toutes les méthodes **HTTP** non nécessaires à l'application :

```
<LimitExcept GET POST HEAD>
deny from all
</LimitExcept>
```

Note : les méthodes disponibles sont :

- **OPTIONS**,
- **GET**,
- **HEAD**,
- **POST**,
- **PUT**,
- **DELETE**,
- **TRACE** et
- **CONNECT**

1.7 Fonction Trace

La fonction trace permet de récupérer la requête que le serveur a reçu. Ceci peut être un faiblesse dans plusieurs cas comme l'utilisation d'un relais inverse qui peut être amené à envoyer au serveur contacté des données sensibles.
Il est recommandé de désactiver la fonction **trace** :

```
TraceEnable off
```

Ceci peut-être complété par la désactivation de la méthode **TRACE**.

1.8 Cookies httponly et secure

Certains cookies, dont les cookies de session, doivent être protégés contre l'écoute de réseau et l'utilisation par javascript. Pour cela il est possible d'activer les options **secure** et **httponly** lors de l'envoie du cookie au client via l'entête **Set-Cookie**.

- **secure** empêche le navigateur d'envoyer un cookie dans une connexion **HTTP** non sécurisée, ie qui n'est pas protégée par **SSL**/**TLS**.
- **httponly** empêche le navigateur de rendre accessible le cookie à du code dynamique comme javascript.

Ceci rendra plus complexe le vol de session, notamment via écoute réseau ou exploitation d'une faille **XSS**.
Il est recommandé de forcer les propriétés "**secure**" et "**httponly**" des cookies générés partout où cela est possible :

```
Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure
```

1.9 Attaques clickjacking

Quand un attaquant encapsule un domaine externe dans son domaine via une **frame**, il peut à partir de son domaine simuler des actions utilisateur sur le domaine dans la **frame**. Ceci est attaque **clickjacking**.
Il est recommandé de refuser que son domaine soit chargé dans une **frame** :

```
Header always append X-Frame-Options SAMEORIGIN
```

1.10 Protection anti-XSS

Certains navigateurs ont des protections **anti-XSS**.
Même si elles sont incomplètes et n'existent pas dans tous les navigateurs, il est recommandé d'activer les protections **XSS** dans les navigateurs clients :

```
Header set X-XSS-Protection "1; mode=block"
```

1.11 Timeout

Il est recommandé de limiter le **timeout** du serveur afin de limiter les possibilités de dénis de services :

```
Timeout 60
```

1.12 SSL/TLS

SSL et **TLS** sont des protocoles cryptographiques permettant de sécuriser le traffic entre un client et un serveur.
Il est nécessaire que le client et le serveur soient de confiance pour faire en sorte que le tunnel chiffré ainsi créé soit lui aussi de confiance.
Il faut configurer le serveur **apache** et notamment lui donner un certificat valide. Pour les certificats il est possible de s'en générer un soit même, mais il est préférable de faire appel à une autorité de certification pour en obtenir un qui sera alors reconnu par tous les navigateurs. Des certificats gratuits, mais plus restreints existent, par exemple avec le système letsencrypt.
La couche **SSL**/**TLS** doit être configurée pour éviter les algorithmes d'échange de clé et chiffrement qui ne sont plus suffisament robustes aujourd'hui.
Il est possible de tester la configuration avec un outil comme **SSLScan** :

```
sslscan –no-failed localhost
```

Il est également possible de tester des couches **SSL**/**TLS** installées sur d'autres services que **HTTPS**.
Il existe un service en ligne testant des serveurs **WEB** avec un nombre plus important de tests :

- https://www.ssllabs.com/ssltest/

Il existe également un moyen de tester la configuration de la couche **SSL**/**TLS** de son navigateur **WEB** :

- https://www.ssllabs.com/ssltest/viewMyClient.html

Des exemples de configurations durcies de **SSL**/**TLS** sont disponibles ici :

- ~~https://cipherli.st/~~ Non disponible, voir : https://web.archive.org/web/20171215095545/https://cipherli.st/
- https://wiki.mozilla.org/Security/Server_Side_TLS
- https://wiki.mozilla.org/Security/TLS_Configurations
- https://mozilla.github.io/server-side-tls/ssl-config-generator/

Aujourd'hui il est au minimum nécessaire de désactiver **SSL** pour ne garder que **TLS** et de vérifier que la version 1.2 de **TLS** est bien activée. De plus les algorithmes d'échange de clé à privilégier sont ceux avec la propriété de **PFS** (**EDH** et **EECDH**).

1.13 mod_security

Pour compléter les filtrages de données en entrée dans l'application, il est possible d'utiliser un système de filtrage de données externe à l'application. Un exemple est le module **mod_security** qui peut être installé sur le serveur **apache** final ou sur un relais inverse.
Après installation il est nécessaire de l'activer :

```
SecRuleEngine On
```

Comme dans ce cas les requêtes qui ne sont pas en accord avec la politique implémentée sont refusées, il est possible de remplacer "**On**" par "**DetectionOnly**" pour ne générer que des alertes sans rejet.
Pour compléter la politique par défaut, il est possible d'installer la politique **CRS** gérée par l'**OWASP** qui contient beaucoup plus de règles mais qui du coup peuvent générer plus de faux positifs sur votre application.
Dans tous les cas il est nécessaire de configurer ses propres règles en adéquation avec les descriptions des données acceptées en entrée.
Références :

- https://modsecurity.org/
- https://modsecurity.org/crs/

1.14 Restriction par adresse IP

Il est recommandé de mettre en place une restriction par adresse IP (au niveau **HTTP**, donc précédant l'accès à l'application **WEB**) là où celà est nécessaire :

```
<Directory /yourwebsite>
Options None
AllowOverride None
Require ip 10 172.20 192.168.2
</Directory>
```

Ici les adresses IP incomplètes correspondent à des réseaux.
Il est possible de précéder la restriction par "**not**" pour l'inverser.

```
Require not ip ipid [ipid]...
```

Il est également possible de combiner des restrictions et même d'utiliser des restriction par adresse IP et par utilisateurs simultanément, voir "**RequireAll**" dans le paragraphe sur la restriction par utilisateurs.

1.15 Restriction par utilisateurs

Il est recommandé de mettre en place une restriction par authentification (au niveau **HTTP**, donc précédant l'accès à l'application **WEB**) là où celà est nécessaire :

```
<Directory /yourwebsite>
 AuthType Basic
 AuthName "Restricted Resource"
 AuthBasicProvider file
 AuthUserFile "/web/users"
 AuthGroupFile "/web/groups"
 Require group admin
</Directory>
```

Il est possible d'utiliser une de ces restrictions :

```
Require [not] user userid [userid] ...
Require [not] group group-name [group-name] ...
Require [not] valid-user
```

Il est possible de précéder la restriction par "**not**" pour l'inverser.
Le fichier **AuthUserFile** doit être généré avec la commande **htpasswd** :

```
# htpasswd -c /web/users denis
New password: mypassword
Re-type new password: mypassword
Adding password for user denis
```

Le fichier **AuthGroupFile** doit suivre le format suivant :

```
group-name: userid [userid] ...
```

Ici par exemple cela sera :

```
admin: denis
```

Il est également possible de combiner des restrictions:

```
<Directory "/yourwebsite">
 <RequireAll>
  Require group alpha beta
  Require not group reject
 </RequireAll>
</Directory>
```

Il est même possible d'utiliser des restriction par adresse IP et par utilisateurs simultanément.

2 Configuration PHP

2.1 Gestion des erreurs

Il est recommandé de ne pas envoyer d'erreur explicite à l'utilisateur. Une application doit catcher ses exceptions, mais cela peut être complété par une restriction complémentaire dans **PHP** pour ne pas envoyer les erreurs propres à **PHP**.

```
display_errors=Off
log_errors=On
error_log=/var/log/httpd/php_scripts_error.log
```

2.2 Upload de fichiers

Il est recommandé de désactiver l'**upload** de fichiers si celui-ci est non nécessaire.

```
file_uploads=Off
```

Si l'**upload** est nécessaire alors il est possible de limiter la taille des fichiers acceptables an plus de gérer correctement l'**upload**, puis les autorisations d'accès aux fichiers.

```
file_uploads=On
upload_max_filesize=1M
upload_tmp_dir="/var/lib/php/session"
```

La valeur limite à utiliser est à configurer suivant les besoins de l'application.
Simultanément il est nécessaire que ces fichiers ne se trouvent pas dans une arborescence accessible via une requête **HTTP**.

2.3 Ouverture de fichiers réseaux

Il est recommandé de désactiver le support de l'utilisation de fichiers réseaux dans les fonctions d'ouverture de fichiers ou d'**include**.

```
allow_url_fopen=Off
allow_url_include=Off
```

2.4 Echappement

Il est recommandé de filtrer les données en entrées, aussi bien aux niveaux des méthodes **GET** et **POST**, que des cookies.
L'interpréteur **PHP** offre la fonction **magic_quotes_gpc** qui permet d'échaper les caractères **' " ** reçus dans les méthodes **GET**, **POST** et les **cookies**. Malheureusement cette stratégie ne fonctionne pas et cette fonction a été désactivée dans les versions récentes de **PHP**. Si elle n'est pas désactivée alors il est possible de le faire :

```
magic_quotes_gpc=Off
```

De la même façon, la fonction **mysql_escape_string()** a été déclarée obsolète. Elle permettait à un programme d'échapper ces mêmes caractères, mais ceci n'est pas une méthode idéale. Il est recommandé d'utiliser les requêtes préparées et les procédures stockées.

2.5 Taille des POST

Il est recommandé de limiter la taille des **POST** afin de limiter les possibilités de dénis de service.

```
post_max_size=1K
```

La valeur limite à utiliser est à configurer suivant les besoins de l'application.

2.6 Limitation de ressources

Il est recommandé de limiter les ressources utilisées par l'interpréteur **PHP**, afin de limiter les possibilités de dénis de service :

```
max_execution_time = 30
max_input_time = 30
memory_limit = 40M
```

2.7 Désactivation de fonctions sensibles

Il est recommandé de désactiver les fonctions sensibles de **PHP** :

```
disable_functions=exec,passthru,shell_exec,system,proc_open,popen,
curl_exec,curl_multi_exec,parse_ini_file,show_source
```

2.8 Limitation de l'arborescence de fichiers

Il est recommandé de limiter l'arborescence de fichiers accessibles :

```
open_basedir="/var/www/html/"
```

La valeur limite à utiliser est à configurer suivant les besoins de l'application.
Il est possible de mettre plusieurs arborescences en les séparant par le caractère "**:**".

2.9 Fichiers de sessions

PHP crée un fichier de session, par défaut cela peut être dans un répertoire partagé. Il est recommandé de déplacer ces fichiers de session dans un répertoire dédié :

```
session.save_path="/var/lib/php/session"
```