

Sessions et cookies en PHP

par [julp \(Autres articles\)](#) [Mathieu Lemoine \(Mes articles pour DVP.com\)](#) [Julien Pauli \(Tutoriels, conférences PHP\)](#) ([Blog](#))

Date de publication : 05/07/2006

Dernière mise à jour : 14/01/2008

Les sessions et les cookies sont incontournables dans le développement PHP par leurs multiples applications : authentification, statistiques...

Ce tutoriel vous apprendra à en comprendre le fonctionnement, à les manipuler et enfin à les configurer.

I - Les cookies.....	3
I-A - Introduction.....	3
I-B - Utilisations.....	3
I-C - Sécurité.....	4
II - Les sessions.....	5
II-A - Introduction.....	5
II-B - À quoi servent-elles ?.....	5
II-C - Fonctionnement.....	5
III - Manipuler les sessions.....	6
III-A - Mise en garde.....	6
Les registers_globals.....	6
Les fonctions session_register(), session_unregister(), session_is_registered() et session_unset().....	6
III-B - Débuter une session.....	6
III-C - Sauvegarder une variable.....	7
III-D - Récupération de données dans une session.....	7
III-E - Savoir si une variable appartient à une session.....	7
III-F - Supprimer une variable d'une session.....	8
III-G - Détruire une session.....	8
III-H - Détruire toutes les variables d'une session.....	8
III-I - Régénérer les identifiants de session.....	8
IV - Aller plus loin.....	10
IV-A - Utiliser plusieurs sessions dans la même page.....	10
IV-B - Configuration liée aux sessions.....	10
La transmission de l'identifiant de session.....	10
La sécurité des sessions.....	11
La durée de vie des sessions.....	12
Modifier localement la configuration des sessions.....	12
IV-C - Modifier la façon dont sont stockées les données de sessions.....	13
IV-D - À savoir.....	16
V - Conclusion.....	17
V-A - Épilogue.....	17
V-B - Remerciements.....	17

I - Les cookies

I-A - Introduction

Le **protocole HTTP** est dit "stateless", ou autrement dit sans état. Pour un serveur, chaque requête qu'il reçoit est indépendante de la précédente, ainsi que de la suivante. Un serveur web est "bête" en quelques sortes, il écoute son port, à chaque fois qu'une requête se présente, il la traite, puis passe à la suivante. Ce processus est néanmoins gênant pour une application, car celle-ci a souvent besoin de "se souvenir" d'un (des) paramètre(s) d'un utilisateur. Un cookie est un petit fichier texte (inoffensif donc), crée par le serveur. En rajoutant l'en-tête *Set-cookie*, dans sa réponse, le serveur indique au client qu'il souhaite y stocker un cookie. Le client peut décider, ou non, de s'exécuter et de créer le cookie demandé.

S'il l'accepte, le navigateur client va ensuite joindre le cookie à l'en-tête de toutes ses requêtes vers le même domaine (on verra ça plus précisément). Aux requêtes suivantes, le client rajoute donc l'en-tête HTTP Cookie et le serveur recevant cet en-tête va le transmettre à PHP.

Ainsi sur une seule requête HTTP, on ne peut savoir si le client accepte les cookies, ou non. Il va être obligé de réadresser une requête au même serveur qui va cette fois ci regarder s'il a reçu en retour le cookie, ou non. C'est ainsi que si l'utilisateur décide de changer de navigateur entre 2 requêtes, ce ne seront pas les mêmes cookies qui seront envoyés.

Un cookie est cependant limité en taille, 4Ko maximum; et en nombre de cookies; mais ce chiffre est généralement très grand. On comprendra que pour stocker une quantité importante d'informations, le cookie n'est pas la solution. Il sert dans une grande majorité des cas à identifier un utilisateur, on va voir cela.

I-B - Utilisations

PHP est lié au serveur web, il récupère les cookies à partir que celui-ci lui fournit, il les a a lui-même reçus, par HTTP, du client. Lorsque PHP récupère les cookies en début de traitement, il les place dans un tableau superglobal accessible via `$_COOKIE`.

PHP peut aussi décider d'envoyer un cookie (si on le lui demande), par la fonction `setCookie(name, value, expire)`. Il manque des paramètres, ca sera vu dans la section suivante.

Ainsi lors de l'appel à `setCookie()`, le moteur de PHP va générer les en-tête HTTP appropriés pour nous, en interne. C'est pour cela qu'on dit que `setCookie()` doit être appelé avant toute sortie, car un appel à `echo`, par exemple, demande à PHP d'afficher quelque chose, et l'oblige donc à générer les en-tête HTTP relatifs à cet affichage.

Or après qu'il ait écrit les en-tête et envoyé le contenu du "echo", on ne peut plus faire machine arrière, un appel à `setCookie()` génèrera donc un warning vous informant que les en-tête de la réponse HTTP ont déjà été envoyés : le fameux **"Headers already sent"** très connu des débutants ...

Revenons à `setCookie()` : tout d'abord il faut nommer le cookie, pour pouvoir le reconnaître. Ensuite il faut lui attribuer une valeur, et enfin optionnellement, une date d'expiration.

Si on ne spécifie pas de date d'expiration, le cookie sera stocké dans la mémoire vive du client (en général), et détruit dès la fermeture du navigateur (et non de l'onglet). Ce cas est pratique pour retenir du client un paramètre temporaire, comme son nom, que l'on souhaite afficher sur nos pages.

Si une date est précisée, ca sera un timestamp. Arrivé à la date, le navigateur client efface le cookie :

Un cookie qui dure 24h

```
<?php
setCookie('developpez','tutoriel sur les cookies / sessions',time()+24*3600);
?>
```

Lecture d'un cookie

```
<?php
if (isset($_COOKIE['developpez'])) {
    echo htmlentities($_COOKIE['developpez'], ENT_QUOTES, );
}
?>
```

I-C - Sécurité

Comme un cookie peut être vu et modifié par le client, à la main (c'est un fichier texte), on échappera sa sortie si on l'affiche, et on vérifiera correctement sa valeur. On peut le chiffrer ou lui rajouter une somme de contrôle (md5) pour empêcher sa modification. Le 4ème paramètre de `setCookie()` est "path". Il définit le chemin pour lequel le cookie est valable sur ce domaine, et sera donc envoyé par le client. Par défaut, il vaut "/", ce qui veut dire que le cookie sera fourni à tous les chemins du domaine actuel.

Ainsi le client enverra son cookie si il accède à `http://monsite.com`, mais aussi à (par exemple) `http://monsite.com/sql/`. Pour limiter le cookie à un dossier en particulier, on lui passe le nom de dossier en 4ème paramètre :

Un cookie envoyé uniquement au répertoire /admin du domaine, et à tous ses sous-répertoires

```
<?php
setCookie('developpez_admin','Bienvenue administrateur',time()+24*3600,'/admin/');
```

Le cinquième paramètre de `setCookie()` fait la même chose, mais pour le domaine et les sous-domaines :

```
<?php
setCookie('DBA_developpez','Bienvenue DBA',time()+24*3600,'/admin/','sql.monsite.com');
```

Ce cookie sera envoyé par le navigateur à `http://sql.monsite.com/admin/`, `http://sql.monsite.com/admin/un-repertoire/`, mais pas à `http://sql.monsite.com/` ni à `http://www.monsite.com/admin/` ni à `http://www.monsite.com/`. Si je donne la valeur de ".monsite.com", au 5ème paramètre (remarquez le point avant le domaine), **alors le cookie sera reçu de tous les sous domaines de monsite.com**

Le 6ème paramètre, "secured", est un booléen, qui vaut false par défaut. Mis à true, il spécifiera au navigateur de n'envoyer ce cookie QUE si la connexion est sécurisée comme par exemple via SSL.

Enfin le 7ème et dernier paramètre est apparu en PHP 5.2, il s'appelle `HttpOnly`, il est à false par défaut. Mis à true, il envoie un cookie noté "HttpOnly", si le navigateur implémente cette règle, alors il rendra inaccessible le cookie à javascript ou tout autre interpréteur pouvant y accéder.

Le cookie ne sera envoyé que par HTTP, et stockée dans une zone mémoire sûre sur le navigateur. Cet attribut est d'une importance capitale pour contrer **les attaques de vol de session**

Par contre il n'est pas mis à true par défaut, car si le navigateur ne connaît pas ce type de cookie (vieux navigateurs), il risque tout simplement de l'ignorer.

II - Les sessions

II-A - Introduction

La session est un mécanisme qui permet à PHP de garder "en mémoire" un nombre illimité de valeurs entre plusieurs requêtes d'un même utilisateur.

Car maintenant qu'on a vu qu'on pouvait garder une trace d'un utilisateur, les sessions viennent palier à un problème des cookies : leur faible capacité de stockage. En effet, les données de session sont un fichier stocké (à l'inverse du cookie) non pas sur le disque du visiteur mais sur le serveur. Ainsi, il est impossible d'avoir accès aux variables de session via Javascript par exemple. De plus, elles ne transitent pas sans arrêt entre le navigateur et le serveur (à la différence des cookies). Une session permet comme le cookie de stocker temporairement des valeurs relatives à un même internaute. Les variables de sessions autorisées sont les variables chaînes, entières, flottantes, caractères, tableaux, objets (à la condition que la classe soit chargée avant la réouverture de la session)... Mais en aucun cas les variables ressources comme celles qui sont utilisées pour les bases de données, créer des images... La durée d'une session est définie par l'hébergeur mais on peut parfois la modifier si on a accès à certains paramètres de la configuration de PHP.

II-B - À quoi servent-elles ?

Elles ont plusieurs utilités. De par leur utilisation, on peut pister le visiteur (et établir des statistiques précises) ou transporter aisément des variables de page en page. On les utilise aussi généralement en complément des cookies dans les scripts d'identification, car parfois le client n'accepte pas les cookies, et donc dans ces situations la session remplace parfaitement le cookie, stocke son identifiant et son mot de passe crypté pour que le membre n'ait pas à le répéter sur chaque page.

II-C - Fonctionnement

Les sessions permettent de conserver des variables tant que l'internaute reste connecté au site et environ 30 minutes après (en cas de déconnexion du visiteur par exemple mais ce temps dépend de votre hébergeur : renseignez-vous grâce à la fonction `phpinfo()`). La session est unique, un identifiant est donc généré aléatoirement pour chaque utilisateur. Et c'est cet identifiant qui représente l'état actuel de la session de travail.

C'est sur lui que repose toute la sécurité du compte de l'utilisateur, de manière générale. Il ne s'agit que d'un petit morceau de texte, unique à chaque visiteur, généré aléatoirement et long, en général, de 32 caractères.

Certains paramètres de `php.ini` permettent de personnaliser la génération de cet identifiant : `session.hash_function` - `session.entropy_length` et `session.entropy_file`

Pour reconnaître un internaute, dans un protocole HTTP sans état, son identifiant doit transiter entre chaque requête que celui-ci génère.

L'identifiant est soit stocké dans un cookie qui est passé de requêtes en requêtes, soit il transite dans l'URL, on appelle ça le trans-id, c'est la fameux `?PHPSESSID=une-chaîne-aleatoire`.

Ce mode de transfert laisse en revanche apparaître cette fameuse chaîne, il ne sera utilisé que lorsque le client refuse les cookies, ce qui de nos jours est très rare.

Les moteurs de recherche par exemple, n'acceptent pas les cookies, c'est pour ça qu'ils indexent souvent des pages avec un `PHPSESSID`, si le webmaster ne leur prête pas une attention particulière.

III - Manipuler les sessions

III-A - Mise en garde

Les registers_globals

Attendez-vous éventuellement à des surprises de taille si les autoglobales sont activées sur votre serveur (directive **register_globals** mise à **On** dans votre fichier *php.ini*). Ce comportement est aujourd'hui formellement déconseillé et déprécié ! Que peut-il donc se passer ? La réponse est simple : en plus des problèmes de sécurité évidents que cette configuration peut poser, vous risquez de modifier des variables de session sans même vous en rendre compte. Illustration avec un exemple :

```
<?php
session_start();

if (ini_get('register_globals') == '1') {
    echo 'Vous devriez mettre <b>register_globals</b> à <b>Off</b><br/>';
}


$_SESSION['login'] = 'toto';
$login = 'titi';

// Affiche :
// - 'titi' avec register_globals à On donc modification car dans CE cas
//   $_SESSION['login'] et $login représente la même variable
// - 'toto' avec register_globals à Off
echo $_SESSION['login'];
?>
```

À noter que le chargement des registers_globals est fait avant le début du script. Il est donc inutile de tenter de les désactiver avec *ini_set()*. Cependant, il est possible de le faire sur certains serveurs grâce au fichier *.htaccess*, grâce à la ligne ci-dessous :

```
.htaccess
php_flag register_globals Off
```

Les fonctions session_register(), session_unregister(), session_is_registered() et session_unset()

 Vous pouvez trouver de nombreux tutoriels utilisant les fonctions *session_register()*, *session_unregister()*, *session_is_registered()*. Il **ne faut plus** utiliser ces fonctions. Elles appartiennent à d'anciennes versions de PHP et sont désormais dépréciées. Elles ne doivent plus être utilisées ! De même, la fonction *session_unset()* qui, bien qu'elle ne soit pas officiellement dépréciée, est très fortement déconseillée car elle suit la même optique que les trois précédentes...

III-B - Débuter une session

Il existe deux façons différentes d'ouvrir une session :

- automatique sur toutes vos pages si l'option *session.auto_start* est égale à 1 ou *on*. Cette option a pour valeur par défaut 0 ou *off*.


Extrait du fichier *php.ini* :

```
session.auto_start = 0 ; initialize session on request startup
```

Extrait du phpinfo :

Directive	Local Value	Master Value
session.auto_start	Off	Off

- démarrage explicite et manuel de la session en utilisant la fonction `session_start()`. Elle permet non seulement de créer une session, mais aussi de restaurer une session via l'identifiant de session. Dans la mesure du possible, je vous recommande donc d'utiliser cette fonction pour démarrer les sessions. Elle vous permettra de récupérer la session existante ou d'en créer une si aucune session ne correspond à l'utilisateur. Un appel à `session_start()` va faire générer à PHP des en-tête HTTP pour la réponse : il va spécifier au client qu'il ne veut pas qu'il mette la page en cache (on verra ça plus bas). Si il n'avait pas reçu d'identifiant de session, par cookie ou trans-id, PHP va alors générer aussi un en-tête HTTP set-cookie: pour lui passer l'identifiant qu'il viendra de générer fraîchement pour nous (sauf si on désactive le cookie, voir plus bas). C'est pour cela, que comme on l'a expliqué plus haut avec `setCookie()`, `session_start()` doit être appelé avant toute sortie (echo, print ou autre).

- 
 - ° Cette fonction renvoie toujours vrai, inutile de tester son retour
 - ° Si vous souhaitez nommer la session, vous devez utiliser `session_name()` avant `session_start()`
 - ° Si vous utilisez des sessions basées sur les cookies, alors il faut utiliser la fonction `session_start()` comme une fonction d'en-tête, c'est-à-dire que rien ne doit être affiché avant.

III-C - Sauvegarder une variable

Les variables de sessions sont, tout simplement, stockées dans le tableau super-global `$_SESSION`. Vous pouvez y stocker des variables de type chaîne, entier, flottant, caractère, etc. ainsi que les objets et tableaux dont les attributs ou le contenu correspondent aux types définis ci-avant (sous certaines conditions au niveau des objets notamment).

```
<?php
session_start(); // Création de la session
$_SESSION['prenom'] = 'Jean-Pierre'; // Sauvegarde dans la session créée de la variable "prenom"
?>
```

III-D - Récupération de données dans une session

Quand on démarre une session avec `session_start()`, le tableau super-global `$_SESSION` est automatiquement initialisé avec les variables de la session. S'il contenait quelque chose, ce contenu n'est plus disponible après.

```
<?php
echo $_SESSION['prenom'];
?>
```

III-E - Savoir si une variable appartient à une session

Il est utile de savoir si des variables de session sont bel et bien enregistrées dans la session. Si ce n'est pas le cas, dans le cadre d'un script d'identification, nous allons demander à l'utilisateur de s'identifier. Pour cela, nous utilisons

sur le tableau `$_SESSION` la fonction `isset()` qui renvoie vrai si la variable passée en argument existe réellement. Par exemple, nous allons reprendre la même variable de session que précédemment : `prenom`.

```
<?php
if (isset($_SESSION['prenom'])) {
    echo 'La variable prenom est déjà enregistrée !';
    // On est certain de pouvoir y accéder ici
} else {
    echo 'La variable prenom n\'est pas enregistrée !';
}
?>
```

III-F - Supprimer une variable d'une session

Pour supprimer une variable, uniquement, d'une session, on utilise la fonction `unset()`, qui prend en paramètre la variable à détruire. Nous allons ici continuer la série d'exemples par la destruction de la variable de session `prenom`.

```
<?php
unset($_SESSION['prenom']); // La variable de session "prenom" a été supprimée, on ne peut plus y avoir accès !
?>
```


III-G - Détruire une session

La fonction `session_destroy()`, qui ne prend aucun paramètre et qui renvoie vrai en cas de succès et faux en cas d'erreur, est utilisée lorsque vous souhaitez vous déconnecter. Elle s'emploie donc ainsi :

```
<?php
if (session_destroy()) {
    echo 'Session détruite !';
} else {
    echo 'Erreur : impossible de détruire la session !';
}
?>
```

III-H - Détruire toutes les variables d'une session

Il est aussi possible de détruire toutes les variables de session, ce qui permet de conserver votre session : il suffit tout simplement de réinitialiser le tableau `$_SESSION`.

 *Il ne faut jamais utiliser `unset()` directement sur `$_SESSION`, cela rendrait impossible l'accès aux variables de la session courante jusqu'à sa fermeture.*

```
<?php
$_SESSION = array(); // $_SESSION est désormais un tableau vide, toutes les variables de session ont été supprimées
?>
```

III-I - Régénérer les identifiants de session

Il est possible de demander à PHP un nouvel identifiant de session grâce à la fonction `session_regenerate_id()`. Les données ne sont alors pas effacées de la session en cours. L'identifiant de session est simplement modifié, l'ancien devenant invalide. Cela est utile lorsque, par exemple, l'utilisateur change de niveau d'accès, s'il a accès à

de nouvelles possibilités ou au contraire n'a plus accès à d'anciennes pages. De cette manière, si quelqu'un avait réussi à trouver l'ancien identifiant de session, il n'aura pas accès aux nouveaux privilèges de l'utilisateur. À l'inverse, si les identifiants de sessions sont régénérés trop souvent, cela peut poser des problèmes au niveau du référencement des pages web, surtout dans les cas où ils sont passés via l'URL.


IV - Aller plus loin

IV-A - Utiliser plusieurs sessions dans la même page

Il est impossible d'ouvrir simultanément plusieurs sessions. Cependant, on peut tout à fait ouvrir plusieurs sessions l'une après l'autre. Dans ce cas, il faut fermer la première session sans la détruire, grâce à `session_write_close()`, puis assigner les nouveaux `session_name` et `session_id`, et enfin ouvrir la nouvelle session avec `session_start()`.

```
<?php
session_name('utilisateur');
session_start(); // Création de la première session
[...] // Utilisation de la première session
session_write_close(); // Fermeture de la première session, ses données sont sauvegardées.
session_name('admin'); // Indication du nom de la seconde session
session_start(); // Ouverture de la seconde session
[...] // Utilisation de la seconde session.
?>
```

On peut refaire la manipulation autant de fois que l'on veut.

 Une fois la session fermée, il est toujours possible d'accéder en lecture (les modifications ne seront pas prises en compte) aux variables de l'ancienne session. `$_SESSION` ne sera vidé et rempli qu'au prochain appel à `session_start()`.

IV-B - Configuration liée aux sessions

La transmission de l'identifiant de session

Comme vu au début, un identifiant unique identifie chaque session. PHP se charge lui-même de transmettre cet identifiant d'une page à l'autre, mais on peut indiquer explicitement comment le faire à l'aide de 4 directives de configurations :


- **`session.use_cookies`** : Cette option fait stocker l'identifiant (et seulement l'identifiant) dans un cookie si elle est égale à 1. Sa valeur par défaut est 1. Si l'option est à 0, et qu'un cookie de session est présent dans le navigateur, il sera simplement ignoré.
- **`session.use_only_cookies`** : Si cette option est égale à 1, alors PHP ignorera les identifiants transmis via l'url pour n'utiliser que ceux contenus dans les cookies. Sa valeur par défaut est 0.
- **`session.use_trans_sid`** : Si cette option est égale à 1, alors PHP insèrera automatiquement l'identifiant dans tous les liens non absolus (ne commençant pas par `http://` ou `ftp://` ou `mailto:` ou d'autres liens absolus, les liens commençant par un `/` (relatifs à la racine du site) seront eux pourvus de l'identifiant.) Sa valeur par défaut est 0.
- **`session.url_rewriter_tags`** : Indique où et dans quelles balises HTML insérer l'identifiant de session dans le cas où **`session.use_trans_sid`** est à 1. Sa valeur par défaut est : `a=href,area=href,frame=src,input=src,form=fakeentry,fieldset=` Le format est `balise1=attribut1,balise2=attribut2,...`. Le cas des balises `form` et `fieldset` est particulier, car cela se traduit en fait par l'apparition d'un champ `input` (de type `hidden`) supplémentaire.

Extrait du `phpinfo` :

Directive	Local Value	Master Value
session.use_cookies	On	On
session.use_only_cookies	Off	Off
session.use_trans_id	On	On
session.url_rewriter_tags	a=href, area=href, frame=src, input=src, form=fakeentry, fieldset=	a=href, area=href, frame=src, input=src, form=fakeentry, fieldset=

Voici deux exemples de configuration pour ces options :

- Le premier est celui qui présente le plus de chances de faire transiter l'identifiant :**
 - `session.use_cookies` à 1
 - `session.use_only_cookies` à 0
 - `session.use_trans_sid` à 1
 - `session.url_rewriter_tags` à `a=href,area=href,frame=src,iframe=src,input=src,form=fakeentry,fieldset=`
Si vous souhaitez produire une page respectant les standards XHTML, nous vous conseillons plutôt la valeur : `a=href,area=href,frame=src,iframe=src,input=src,fieldset=` et d'utiliser une balise `fieldset` dans vos balises `form`.
- Le second est celui qui est totalement invisible à l'oeil d'un visiteur classique mais, si les cookies sont désactivés, il ne fonctionnera pas.**
 - `session.use_cookies` à 1
 - `session.use_only_cookies` à 1
 - `session.use_trans_sid` à 0
 - `session.url_rewriter_tags` à vide

 *Il faut savoir que faire passer l'identifiant dans l'url (`session.use_trans_sid` à 1) peut présenter des risques. En effet, quelqu'un copiant un lien pour un ami lui donnerait par là-même son identifiant de session et sa session serait ouverte quand on cliquerait sur le lien.*

La sécurité des sessions

Il existe plusieurs directives de configuration permettant de jouer sur la sécurité des sessions, en voici quelques unes :

- Vérifier la provenance du visiteur**
`session.referer_check` permet de spécifier une chaîne de caractères qui doit être présente dans le "referer" (adresse de la page de provenance) si celui-ci est indiqué par le navigateur. Une bonne idée est d'indiquer le nom de domaine de votre site (par exemple). Par contre, gardez à l'idée que cela ne permet pas d'être sûr que le visiteur provient d'une des pages de votre site. En effet, le referrer provient du protocole HTTP, lequel est très facile à manipuler !
- Augmenter la complexité de l'identifiant de session**
`session.hash_function` permet de choisir entre deux algorithmes de création des identifiants de sessions : MD5 (mettre la valeur 0) ou SHA-1 (mettre la valeur 1), le second permet de générer des identifiants de sessions plus longs.
`session.hash_bits_per_character` permet de définir les plages de caractères utilisées pour l'identifiant de session 4 pour les caractère '0' à '9' et 'a' à 'f', 5 pour '0'-'9' et 'a'-'v' et 6 pour '0'-'9', 'a'-'z', 'A'-'Z', '-' et '_'. Il s'agit uniquement d'une représentation de l'identifiant, en aucun cas ça ne modifie sa complexité intrinsèque. *Ces directives de configuration ne sont disponibles que depuis PHP 5.*
- Modifier la façon dont sont formatées les données de sessions**
`session.serialize_handler` permet de modifier la façon dont sont formatées les données de sessions avant d'être sauvegardées. Par défaut, (valeur `php`) c'est une sérialisation proche de la fonction `serialize` qui est utilisée. Si votre installation de PHP le supporte, vous pouvez utiliser WDDX (valeur `WDDX`), qui permet de récupérer un format XML.

- **Modifier la façon dont sont sauvegardées les données de sessions**
`session.save_handler` permet de modifier la façon dont sont sauvegardées les données de sessions. La configuration actuelle ne permet pas de choisir d'autres "*handler*" que celui par défaut (dans un fichier non crypté, valeur *files*). Cependant, il existe une fonction PHP qui permet de définir un nouveau handler, voir un peu plus loin.
- **Modifier les données utilisées pour créer les identifiants de sessions**
`session.entropy_file` permet de spécifier un fichier dont seront extraites des données permettant de rendre aléatoire l'identifiant de session généré. En général, c'est un "fichier" spécial qui est utilisé, dont le contenu est différent et aléatoire à chaque fois qu'on tente de le lire (`/dev/random` ou `/dev/urandom` sous unix par exemple).
`session.entropy_length` permet de spécifier le nombre de caractères qui seront lus dans ce fichier.
- **Modifier la configuration du cookie stockant les identifiants de sessions**
`session.cookie_lifetime`, `session.cookie_path` et `session.cookie_domain` permettent de modifier la durée de vie, le chemin et le domaine attribués à un cookie.
`session.cookie_secure` restreint l'utilisation du cookie aux connexions HTTPS (connexions cryptées).
`session.cookie_httponly` restreint l'accès au cookie via le protocole HTTP. Autrement dit, les langages de script côté client (Javascript par exemple) n'auront pas accès au cookie. Ce n'est pas supporté par tous les navigateurs, et disponible uniquement depuis PHP5.
- **Modifier la mise en cache des pages par le navigateur et les proxys**
`session.cache_limiter` permet d'autoriser ou non la mise en cache des pages Web par le navigateur et les proxys. *nocache* (valeur par défaut) interdit la mise en page, *private* l'autorise uniquement pour les navigateurs et *public* pour tout le monde. Certains navigateurs peuvent rencontrer quelques problèmes avec l'expiration du cache : dans ce cas, il vaut mieux spécifier *private_no_expire*.
`session.cache_expire` permet de spécifier la durée de vie des pages web mises en cache.



Bien qu'il soit possible de crypter les sessions, elles sont stockées en tant que fichiers sur le serveur Web. À ce titre, elles n'ont pas de système d'identification (login + password spécifiques) comme le shell et la base de données. Nous étudierons plus loin une solution permettant de modifier la manière dont sont enregistrées les sessions (et ainsi d'utiliser la base de données).

La durée de vie des sessions

Pour modifier la durée de vie des sessions, en plus de jouer sur la durée de vie du cookie et du cache, on peut également jouer sur le temps au-delà duquel PHP considèrera les données stockées comme obsolètes.

Pour cela, il existe trois directives de configuration :

- `session.gc_maxlifetime` : Il s'agit effectivement de la durée au-delà de laquelle des données de session seront considérées comme périmées.
- `session.gc_probability` : La vérification du temps de vie des données n'est pas systématique, c'est lancé aléatoirement selon une fréquence prédéfinie. Il s'agit de la probabilité de déclenchement de l'opération.
- `session.gc_divisor` : Pour obtenir la fréquence de lancement de la procédure de vérification des données, il faut diviser la probabilité par ce paramètre. Par exemple, si `gc_probability` vaut 1 et que `gc_divisor` vaut 100, alors ce sera lancé 1 fois toutes les 100 ouvertures de sessions.

Modifier localement la configuration des sessions

Comme pour toutes les directives de configuration de PHP, il est possible de modifier la configuration des sessions sur certains serveurs. On utilise pour cela le fichier `.htaccess`. Pour modifier la valeur d'une directive de configuration, il suffit d'indiquer `php_value` ou `php_flag` (dans le cas d'une directive de type on/off), suivi du nom de la directive de configuration et de sa nouvelle valeur (en séparant le tout par des espaces).


Cependant, gardez à l'esprit que cela dépend entièrement de la configuration du serveur, il est même possible que seule une partie des directives soit modifiable, voire même aucune...

IV-C - Modifier la façon dont sont stockées les données de sessions

Comme indiqué plus haut, PHP ne fournit pas de gestionnaire de données de sessions autre que des fichiers non cryptés. Il est cependant possible de définir vous-même les opérations à faire pour ouvrir, fermer, lire, écrire et vérifier la durée de vie d'une session.

Cette fonctionnalité vous permettra par exemple de crypter et de stocker très facilement vos sessions dans une base de données (par exemple) en écrivant les fonctions de requêtage adéquates. Une fois ceci fait, vous n'aurez alors plus besoin de définir les opérations d'accès à la base de données pour les sessions !

Vous trouverez un peu plus bas un exemple de fonctions vous permettant de stocker les sessions dans une base de données.

 *Dans ce cas, il vous faut alors définir ces opérations au début de chaque script (avant le `session_start()` !*

La fonction permettant de faire ceci se nomme `session_set_save_handler()` et prend en argument les noms des fonctions permettant les opérations citées ci-dessus (dans le même ordre). L'exemple ci-dessous ne fait que redéfinir les opérations par défaut de PHP, pas très utile, certes, mais ça vous permet de voir comment ça fonctionne...

```
<?php
function ouvrir_session($chemin_de_stockage, $nom_de_session)
{
    $_ENV['chemin_de_stockage_des_sessions'] = $chemin_de_stockage;
    return true;
}

function fermer_session()
{
    return true; // Rien à faire
}

function lire_session($identifiant_de_session)
{
    return strval(file_get_contents($_ENV['chemin_de_stockage_des_sessions'].'/' .
    sess_'. $identifiant_de_session));
}

function ecrire_session($identifiant_de_session, $donnees_de_session)
{
    if ($fp = fopen($_ENV['chemin_de_stockage_des_sessions'].'/' . sess_'. $identifiant_de_session, "w")) {
        $return = fwrite($fp, $donnees_de_session);
        fclose($fp);
        return $return;
    } else {
        return false;
    }
}

function detruire_session($identifiant_de_session)
{
    return unlink($_ENV['chemin_de_stockage_des_sessions'].'/' . sess_'. $identifiant_de_session);
}

function verifier_validite_session($temps_de_validite)
{
    $files = glob($_ENV['chemin_de_stockage_des_sessions'].'/' . sess_ '*');

    if (!empty($files))
    {
        foreach($files as $file)
        {
            // Il peut s'agir d'un répertoire ou d'un fichier mais seuls les fichiers nous intéressent
            if (is_file($file) and filemtime($file) + $sessionTime < time ())
            {
                unlink($file);
            }
        }
    }
}
```

```

    }
  }
}

/*foreach ($_ENV['chemin_de_stockage_des_sessions'].'/sess_*) as $fichier_de_session) {
  if (filemtime($fichier_de_session) + $temps_de_validite < time()) {
    unlink($fichier_de_session);
  }
}*/


return true;
}


session_set_save_handler(
  'ouvrir_session',
  'fermer_session',
  'lire_session',
  'ecrire_session',
  'destruire_session',
  'verifier_validite_session'
);

session_start();

// Maintenant on peut se servir des sessions sans modifications par rapport à avant...
// C'est complètement transparent pour l'utilisateur.
?>

```

Nous allons maintenant voir un exemple de fonctions permettant de stocker vos sessions dans une base de données. Nous utiliserons MySQL comme  **SGBD** et MySQLi comme extension PHP pour communiquer avec MySQL. Des commentaires indiquent comment remplacer MySQLi par MySQL.

 **Bien sûr, il ne s'agit ici que d'un exemple ! Ces fonctions ne sont pas parfaitement sécurisées ni optimisées !**
Il ne faut surtout pas, par exemple, les utiliser ainsi dans un environnement de production.

Création de la table pour stocker les données

```

USE my_database;

CREATE TABLE `sessions` (
  `identifiant` VARCHAR( 64 ) NOT NULL ,
  `nom` VARCHAR( 64 ) NOT NULL ,
  `dernier_acces` TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
  `donnees` TINYTEXT NOT NULL ,
  PRIMARY KEY ( `identifiant` , `nom` )
) ENGINE = MYISAM ;

```

Les fonctions d'accès en elles-mêmes

```

<?php
/**
 * SGBD est une constante contenant l'adresse du serveur MySQL. [typiquement 'localhost']
 * LOGIN_SGBD est une constante contenant votre login pour la connection au serveur MySQL.
 * MOT_DE_PASSE_SGBD est une constante contenant votre mot de passe pour la connection au serveur MySQL.
 * SESSIONS_BDD est une constante contenant le nom de la base de données où sont stockées les données. [ici, 'my_
 * SESSION_TABLE est une constante contenant le nom de la table où sont stockées les données. [ici, 'sessions']
 */

function ouvrir_session($chemin_de_stockage, $nom_de_session)
{
  $_ENV['nom_de_session'] = $nom_de_session;
  return true;
}

function fermer_session()

```

Les fonctions d'accès en elles-mêmes

```
{
    return true; // Rien à faire
}

function lire_session($identifiant_de_session)
{
    $mysql = new mysqli(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD, SESSIONS_BDD);
    //$mysql = mysql_connect(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD); mysql_select_db(SESSIONS_BDD, $mysql);
    if(!$mysql)
    {
        return '';
    }
    $resultat = $mysql->query('SELECT donnees FROM '.SESSIONS_TABLE
        .' WHERE identifiant = \''.$identifiant_de_session.'\' AND nom = \''.$_ENV['nom_de_session'].'\'');
    //$resultat = mysql_query('SELECT donnees FROM '.SESSIONS_TABLE
    // .' WHERE identifiant = \''.$identifiant_de_session.'\' AND nom = \''.
    $_ENV['nom_de_session'].'\'', $mysql);
    $donnees = '';
    if($resultat->num_rows == 1) //if(mysql_num_rows($resultat) == 1)
        $donnees = $result->fetch_array(); //$donnees = mysql_fetch_array($resultat);
    $resultat->free(); //mysql_free_result($resultat);
    $mysql->close(); //mysql_close($mysql);
    return strval($donnees['donnees']);
}

function ecrire_session($identifiant_de_session, $donnees_de_session)
{
    $mysql = new mysqli(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD, SESSIONS_BDD);
    //$mysql = mysql_connect(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD); mysql_select_db(SESSIONS_BDD, $mysql);
    if(!$mysql)
    {
        return false;
    }
    $result = true;
    $mysql->query('UPDATE '.SESSIONS_TABLE.' SET donnees = \''.$donnees_de_session.'\'
        .' WHERE identifiant = \''.$identifiant_de_session.'\' AND nom =
        \''.$_ENV['nom_de_session'].'\'') or $result = false;
    //mysql_query('UPDATE '.SESSIONS_TABLE.' SET donnees = \''.$donnees_de_session.'\'
    // .' WHERE identifiant = \''.$identifiant_de_session.'\' AND nom = \''.
    $_ENV['nom_de_session'].'\'', $mysql) or $result = false;
    $mysql->close(); //mysql_close($mysql);
    return $result;
}

function detruire_session($identifiant_de_session)
{
    $mysql = new mysqli(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD, SESSIONS_BDD);
    //$mysql = mysql_connect(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD); mysql_select_db(SESSIONS_BDD, $mysql);
    if(!$mysql)
    {
        return false;
    }
    $result = true;
    $mysql->query('DELETE FROM '.SESSIONS_TABLE.' WHERE identifiant = \''.$identifiant_de_session.'\'
        .' AND nom = \''.$_ENV['nom_de_session'].'\'') or $result = false;
    //mysql_query('DELETE FROM '.SESSIONS_TABLE.' WHERE identifiant = \''.$identifiant_de_session.'\'
    // .' AND nom = \''.$_ENV['nom_de_session'].'\'', $mysql) or $result = false;
    $mysql->close(); //mysql_close($mysql);
    return $result;
}

function verifier_validite_session($temps_de_validite)
{
    $mysql = new mysqli(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD, SESSIONS_BDD);
    //$mysql = mysql_connect(SGBD, LOGIN_SGBD, MOT_DE_PASSE_SGBD); mysql_select_db(SESSIONS_BDD, $mysql);
    if(!$mysql)
    {
        return false;
    }
    $result = true;
}
```

Les fonctions d'accès en elles-mêmes

```
$mysql-
>query('DELETE FROM '.SESSIONS_TABLE.' WHERE ADDDATE(dernier_acces, INTERVAL '.$temps_de_validite.' SECOND) < NOW()
  or $result = false;
  //mysql_query('DELETE FROM '.SESSIONS_TABLE.' WHERE ADDDATE(dernier_acces, INTERVAL '.$
$temps_de_validite.' SECOND) < NOW()', $mysql)
  or $result = false;
$mysql->close(); //mysql_close($mysql);
return $result;
}

session_set_save_handler(
  'ouvrir_session',
  'fermer_session',
  'lire_session',
  'ecrire_session',
  'destruire_session',
  'verifier_validite_session'
);

session_start();

// Maintenant on peut se servir des sessions sans modifications par rapport à avant...
// C'est complètement transparent pour l'utilisateur.
?>
```

IV-D - À savoir

- L'hébergeur Free est une exception puisque, pour pouvoir utiliser les sessions, il vous faudra créer à la racine de votre FTP un répertoire nommé **sessions**.
- La fonction `session_id()` utilisée sans paramètre renvoie l'identificateur actuel de la session et la fonction `session_name()` utilisée sans paramètre renvoie le nom courant de la session. Ainsi, pour passer dans un lien l'identificateur de session, ce qui permet de récupérer la session de l'utilisateur même s'il change de répertoire, il suffit de procéder ainsi :

```
<?php
echo 'http://adresse_de_votre_site?' . session_name() . '=' . session_id();
?>
```


V - Conclusion

V-A - Épilogue

Nous avons vu ce qu'est une session, comment l'utiliser et la configurer. Cependant, nous souhaitons attirer votre attention sur le fait que, malgré leur localisation côté serveur, vous n'êtes pas à l'abri de l'usurpation d'identité par vol de l'identifiant de session. Il existe des méthodes pour s'en prémunir, voir les liens ci-dessous, mais dépassent le cadre de cet article. En effet, l'aspect sécurité a principalement été abordé au travers de la configuration globale des sessions.

Liens Developpez :

-  [Utiliser une base de données pour sécuriser vos sessions](#), par Adrien Pellegrini
-  [Comment débuter avec MySQL et PHP](#), par Eusebius

Liens externes :

-  [Le manuel PHP sur les sessions](#)

V-B - Remerciements

Nous adressons nos remerciements à **Yogui** pour ses recommandations techniques avisées.