plus, une modération du code distant sera un minimum à réaliser : au fond, vous êtes en train de donner accès à un site extérieur sur votre application !

Si le fichier distant ne contient que des données et pas de code PHP, il existe de nombreuses alternatives qui permettent de lire des données à distance et de les recevoir sous forme native en PHP. Json, WDDX, SOAP, REST sont des solutions adaptées aux échanges de données.

Exécution de code à la volée

Outre les inclusions de code distant, il y a d'autres moyens pour exécuter dynamiquement du code PHP dans une application.

eval()

Un autre moyen pour modifier dynamiquement l'exécution d'un script est l'utilisation de `eval()`. Cette fonction prend en argument une chaîne de caractères et l'exécute comme si c'était du code PHP. On s'en sert souvent pour assembler au dernier moment des variables avec du texte littéral, ou bien pour exécuter des morceaux de code et affecter le résultat à une variable.

```
eval('throw $this;');
```

Cet exemple exécute la commande `throw`, qui émet une exception, avec la variable `$this`. Il est issu du code de `PEAR` et permet de lancer une exception à partir de son constructeur, sans avoir à préciser la classe de l'exception : `$this` est alors dynamique.

Comme toujours avec le code dynamique, si la chaîne de caractères qui est exécutée par PHP a pu être manipulée depuis l'extérieur, il devient possible de faire exécuter du code arbitraire à l'application PHP. Par exemple, voici une ligne de code dangereuse :

```
eval('echo $'.$_GET['nom_de_variable']) ;
```

Avec une telle ligne dans l'application PHP, on peut utiliser la valeur suivante pour faire afficher `phpinfo()` :

```
$_GET['nom_de_variable'] = 'x; phpinfo();' ;
```

Heureusement, l'utilisation de `eval()` est assez peu fréquente. Elle résout des problèmes de compatibilité, mais il reste exceptionnel d'avoir à compiler du code PHP à la dernière minute. Et il est important de noter que c'est une solution qui est très lente, par rapport à du code inclus, ou même du code déjà inscrit dans le script PHP. C'est une bonne pratique que de supprimer `eval()` : souvent, on arrive à sécuriser et accélérer un script en même temps.

`eval()` est parfois utilisée pour déporter dans un fichier de traduction des interpolations de variables. Par exemple :

```
// Fichier de traduction
define('AFF_PRIX','Le prix est de $prix Euros') ;
// Fichier d'utilisation
eval('echo '.AFF_PRIX) ;
```

Pour ce type d'application, il est plus efficace d'utiliser la fonction `printf()` :

```
// Fichier de traduction
define('AFF_PRIX','Le prix est de %d Euros') ;
// Fichier d'utilisation
printf(AFF_PRIX, $prix) ;
```

Pour les cas où plusieurs variables sont requises dans un ordre différent suivant les traductions, `printf()` supporte un système de noms dans la syntaxe de formatage : cela permet à cette dernière d'être constante, mais de pouvoir toujours utiliser le même ordre de variables. Comme ceci :

```
// anglais
define('AFF_PRIX','The price is Euros %1$d for item %2$s') ;
// français
define('AFF_PRIX','Le prix de %2$s est de %1$d euros') ;
// l'ordre des arguments est toujours le même,
// quelle que soit la langue
printf(AFF_PRIX, $prix, $article) ;
```

Notez qu'il est très difficile de sécuriser `eval()` et qu'il n'existe pas de fonction pour protéger le contenu contre des utilisations sournoises. La meilleure protection consiste simplement à éviter de l'utiliser. La solution extrême consiste à ajouter `eval()` à la directive `disable_functions` pour être certain de ne pas l'utiliser.

Enfin, notez que la différence entre les guillemets simples et doubles prend tout son sens dans le cas de `eval()` :

```
eval('echo $var ;') ; // affiche la variable $var
eval("echo $var ;") ; // affiche le résultat du code contenu dans la variable var : le résultat va être
                        différent !!
```

assert()

`assert()` est une instruction assez peu connue de PHP, mais très utile : elle permet de placer des points de validation dans le code et d'émettre une alerte PHP lorsque les points de validation ne sont pas vérifiés, par exemple :

```
function carre($i) {
   assert('is_numeric($i)', 'Attention, nous voulons un nombre') ;
   return $i * $i ;
}
```

Avec cette construction, une alerte est émise si la condition `is_numeric($i)` échoue : la fonction `carre()` ci-dessus n'accepte ici que des nombres, entiers ou décimaux. Si le script tente de calculer le carré d'une chaîne de caractères ou d'un tableau, alors `assert()` émet une alerte, comme celle-ci :

```
Warning: assert(): Assertion "is_numeric($i)" failed in /script.php on line 16
```

`assert()` prend du code PHP sous forme de chaîne de caractères et l'exécute à la volée. De ce point de vue, `assert()` se comporte donc exactement comme `eval()` et doit être traité avec les mêmes précautions.

Dans la pratique, les assertions sont utilisées uniquement comme points de validation dans le code du script : elles ont rarement recours à des constructions dynamiques pour construire les tests de validation. Cependant, c'est possible et il importe de garder l'utilisation de la fonction `assert()` à l'œil.

La grande différence entre `assert()` et `eval()` est que les assertions sont pilotées par la directive `assert.active`. Il est possible de désactiver les assertions à l'aide d'une directive de configuration. En production, il est recommandé de mettre `assert.active` à Off dans tous les cas : cela permet de supprimer immédiatement toutes les assertions et d'exécuter le code comme si elles n'existaient pas. Pour `eval()`, il faudrait utiliser la directive `disable_functions`.

preg_replace()

On pense rarement à `preg_replace()`comme source d'injection de code PHP. Elle est pourtant victime de son succès. `preg_replace()` est une fonction de manipulation de texte à l'aide d'expression rationnelle, de type Perl. Elle remplace toutes les occurrences du motif qu'elle reçoit en premier argument par la chaîne de caractères en deuxième argument. `preg_replace()` permet les injections de code PHP, lorsque l'expression rationnelle utilise l'option `e`. Avec cette option, `preg_replace()` remplace les occurrences trouvées avec le résultat du code PHP placé en deuxième argument, au lieu de faire un remplacement par valeur littérale. Voici une illustration :

```
<?php
$html_body = preg_replace("/(<\/?)(\w+)([^>]*>)/e",
                          "'\\1'.strtoupper('\\2').'\\3'",
                          $html_body);
?>
```

Cette ligne traite la chaîne `$html_body` et met en majuscules toutes les balises XML qu'elle trouve. L'expression régulière identifie toutes les balises, à l'aide du caractère ouvrant <, du nom de la balise `\w+` et du reste de la balise jusqu'au signe fermant « > ». Le premier caractère est réutilisé dans le motif de remplacement sous la forme de `\\1`. La fin de la balise est aussi réutilisée sans modification avec `\\3`. Toutefois, le corps de la balise est passé à la fonction `strtoupper()` avant d'être remis dans le code HTML. `strtoupper()` est une fonction PHP qui met en majuscules la chaîne de caractères qui lui est passée.

Pour exploiter la vulnérabilité ci-dessus, il faut produire une injection de code PHP. Les effets sont alors les mêmes que ceux de la fonction `eval()`. Avec une valeur modifiée de `$html_body` telle que :

```
$html_body = "<').phpinfo().printf('>" ;
```

le code PHP utilisé en deuxième argument devient :

```
"'<'.strtoupper('').phpinfo().printf('').'>'",
```

L'injection de code PHP a permis de placer un appel à la fonction `phpinfo()`.

L'option `e` des expressions rationnelles est destinée à réaliser des remplacements complexes, où il faut appliquer des transformations aux occurrences identifiées, avant de les réinjecter dans la chaîne originale. Il est plus sûr d'utiliser la fonction `preg_replace_callback()` : avec cette fonction, le remplacement est délégué à une fonction utilisateur, qui est le deuxième argument, au lieu d'être du code PHP dynamiquement évalué. Le code utilisé est maintenant écrit en dur dans le script PHP, ce qui évite les injections de code PHP.

Téléchargement de fichiers

Le téléchargement de fichiers est une avenue royale pour importer du code PHP pirate sur un site. En effet, pour exécuter un script PHP, il suffit que ce dernier ait les autorisations de lecture et qu'il soit accessible depuis le Web. Il n'y a pas besoin de droits d'exécution ou d'écriture pour pouvoir exécuter un script PHP : simplement les droits de lecture.

PHP est capable de recevoir des fichiers via le Web et de les stocker sur le serveur : c'est le téléchargement (*upload*) de fichiers. L'application la plus populaire consiste à télécharger des images, que ce soit pour un album de famille ou pour un avatar sur un forum. L'image est validée, puis mise immédiatement en production sur le site.

En termes de sécurité, le commun des programmeurs identifie le cas des fichiers exécutables et des virus : une fois chargés sur le serveur, ces derniers sont effectivement une source de problème, à condition qu'ils soient exécutés. En effet, pour les activer, il faut les charger, mais aussi les exécuter. Or, il est rare qu'un site web utilise des applications tierces. En prenant même le cas d'un serveur Apache avec PHP sous forme de module, il n'y a qu'un exécutable, Apache lui-même et quelques autres logiciels système incontournables. Au niveau du site web, aucun exécutable n'est disponible et aucun droit d'exécution n'est donné, sauf aux dossiers.

Ainsi, les fichiers exécutables et les virus représentent une menace réelle, mais, en pratique, très rare. En fait, comme les fichiers sont chargés sur le serveur avec uniquement des droits de lecture, il ne reste qu'une seule menace possible : PHP lui-même.

En effet, les scripts légitimes d'une application web sur un serveur sont simplement dotés du droit de lecture : c'est le serveur Apache qui les lit, puis les exécute. Ainsi, en chargeant un script PHP, il n'y a plus qu'une condition pour que ce dernier soit exécuté sur le serveur : pouvoir y accéder depuis un navigateur web.

Dans ce cas, il devient possible de faire exécuter son propre code PHP. On ne parle plus d'injection, mais carrément de téléchargement de vulnérabilité !

Extensions de fichiers

Une autre option consiste à appliquer aux fichiers téléchargés des extensions de fichier neutres. C'est une pratique moins sûre que la précédente, mais les dossiers hors Web ne sont pas toujours disponibles. Par exemple, le serveur Apache peut avoir configuré les

fichiers `.inc` et `.php` pour être exécutés par PHP. Toutes les autres extensions de fichiers sont simplement servies par Apache sous forme brute, comme un texte `.txt`, une image `.gif`, `.png` ou `.jpg`, ou de tout autre format multimédia, comme `.pdf` ou `.swf`. Dans ce cas, forcez le type du fichier téléchargé à un de ces types neutres, comme `.txt`, `.brut`, `.uploaded` ou `.raw`.

Attention au modérateur

La modération de contenu protège votre installation PHP contre les téléchargements de code sournois, mais attention à ne pas rendre vulnérable le compte de l'administrateur. Durant la phase de modération, ce dernier doit évaluer les fichiers téléchargés. Il est alors important de le protéger contre des attaques détournées comme le chargement d'un script JavaScript sur le serveur à la place d'une image.

Fonctions à surveiller

Certaines fonctions de PHP sont des points de passage obligés. On peut les considérer comme des limites de la plate-forme, dans la mesure où elles transmettent des commandes à des systèmes externes. C'est donc à ces points frontières qu'il faut savoir faire toutes les vérifications nécessaires pour ne pas transmettre de problème provenant du Web.

Code PHP

Les fonctions suivantes ont un impact direct sur le fonctionnement de PHP : elles créent des variables, ajoutent et exécutent du code PHP dynamiquement.

Inclusions de code PHP

Il s'agit des quatre fonctions bien connues : `include()`, `require()`, `include_once()` et `require_once()`.

Nous avons vu que ces quatre fonctions étaient la voie royale pour réaliser des injections de code PHP dans une application web saine. Chacune de leur utilisation doit donc être vérifiée.

Préférez toujours les inclusions statiques, où le nom des fichiers est écrit en dur dans le script, aux inclusions dynamiques. En voici un exemple :

```
include('include/database.inc') ;
```

Si les fichiers inclus sont de nature plus dynamique, comme dans le cas des bibliothèques de modules optionnels, testez la présence du fichier avant de l'ouvrir, à l'aide de `file_exists()`. Si cette fonction représente une charge trop grande pour votre serveur, alors enregistrez la liste des fichiers disponibles dans un tableau et testez votre candidat à l'inclusion avec ce tableau.

Si vous le pouvez, désactivez `allow_url_fopen`, ou bien utilisez PHP 5.2 ou plus récent pour avoir les deux directives séparées, `allow_url_fopen` et `allow_url_include`.

eval()

Comme nous l'avons vu dans la section sur les injections de code PHP, utiliser `eval()` est une mauvaise idée. Repérez toutes les utilisations de `eval()` et demandez-vous si vous en avez vraiment besoin. A priori, vous devriez pouvoir vous en passer.

preg_replace()

`preg_replace()` effectue des remplacements dans une chaîne de caractères, à l'aide d'une expression régulière. L'option `e` permet d'évaluer le code de remplacement comme s'il s'agissait de code PHP.

Remplacez cette fonction par `preg_replace_callback()` afin d'éviter les injections PHP lorsque l'expression rationnelle utilise l'option `e`. Remplacez cette fonction par `str_replace()`, lorsque le remplacement est simple.

extract()

`extract()` est une fonction qui crée des variables en masse à partir d'un tableau, par exemple :

```
<?php
  extract(array('a' => 1));
  echo $a ;
?>
```

Ce script va afficher 1. Les index du tableau deviennent des variables et la valeur associée devient la valeur de la variable. Au passage, les variables qui existent déjà sont écrasées et remplacées par les nouvelles valeurs.

Malheureusement, `extract()` est souvent utilisée pour simuler `register_globals` et assurer facilement une compatibilité ascendante. Ainsi,

```
extract($_POST) ;
```

aura le même effet que `register_globals`, ou encore une boucle telle que :

```
foreach($_POST as $var => $val) {
    $$var = $val ;
}
```

Nous avons déjà insisté sur le fait que `register_globals` est une directive dangereuse. Les techniques de remplacement telles que celles-ci sont non seulement dangereuses, mais en plus néfastes en termes de performances.

import_request_variables()

Cette fonction réalise exactement l'importation que ferait PHP si `register_globals` était active. Contrairement à `extract()`, qui peut servir à autre chose, cette fonction permet uniquement d'assurer la compatibilité avec les anciennes versions de PHP.

Il est recommandé de ne jamais l'utiliser dans les nouvelles applications.

parse_str()

`parse_str()` analyse une chaîne de requête d'URL pour en extraire les variables, comme le fait PHP lui-même. Non seulement les variables identifiées sont alors créées directement dans le code, mais elles deviennent aussi des variables globales. Observez le code suivant :

```
<?php
  parse_str('a=1&b=2');
  print($GLOBALS['a']);
?>
```

Cet exemple va afficher 1. Comme `extract()`, `parse_str()` peut donc servir à écraser des variables avec d'autres variables. Le problème est même plus grave encore car `parse_str()` crée des variables globales, comme le montre l'exemple précédent.

Pour neutraliser ce comportement par défaut, il faut fournir une variable en deuxième argument de `parse_str()` : dans ce cas, les variables décodées de la chaîne de caractères seront rangées dans un tableau et ne seront pas directement injectées dans le code ou dans le tableau `$GLOBALS`.

```
<?php
  parse_str('a=1&b=2', $mes_vars);
  print_r($mes_vars);
?>
```

register_shutdown_function()

`register_shutdown_function()` enregistre la fonction passée en premier argument pour exécution lors de l'extinction du script. Lorsque le script se termine et après l'envoi de tout le contenu au navigateur, PHP entre en mode d'extinction : il nettoie la mémoire, détruit les objets, efface les données et exécute les fonctions enregistrées.

Actuellement, les fonctions enregistrées avec `register_shutdown_function()` ne tombent pas sous la coupe de la directive `max_execution_time`. Ainsi, il est possible d'exécuter indéfiniment du code PHP avec une fonction qui a été enregistrée pour l'extinction.

`register_shutdown_function()` est souvent utilisée par des bibliothèques externes, qui doivent clore proprement leurs ressources avant que PHP ne le fasse pour elle ; par exemple, refermer un système de stockage pour les sessions, un fichier XML ou toute autre opération pour laquelle la déconnexion doit se faire selon un protocole établi.

Le groupe PHP a été prévenu de ce problème, mais à l'heure où nous écrivons ce livre, aucune solution n'a été implémentée pour le régler. En attendant, utilisez `disable_functions` si vous n'avez pas l'utilité de cette fonction.

Affichages d'information

Les fonctions suivantes affichent des informations susceptibles d'intéresser les pirates. Elles en révèlent beaucoup sur votre application web et son fonctionnement. Elles permettront aux pirates d'aller plus loin, sans compromettre directement le système. Il est bon de surveiller leur utilisation.

phpinfo()

La célèbre fonction `phpinfo()` n'est pas un problème de sécurité en soi, mais elle affiche tous les détails de la configuration du site. Inutile de dire que c'est une mine d'or pour un pirate qui met la main dessus. Il peut complètement adapter son attaque en fonction de la configuration.

Il est donc recommandé de surveiller et réduire autant que possible l'utilisation de `phpinfo()` dans votre site. Généralement, un seul `phpinfo()` est suffisant sur un serveur ; il doit être placé à la disposition des administrateurs du site et non pas dans une page publique. Une protection par session ou bien par identification HTTP est raisonnable pour protéger cette page : ne pas la publier est même encore mieux.

Pensez à contrôler si votre site n'a pas de `phpinfo()` public en vérifiant directement sur les moteurs de recherche qui passent le plus souvent sur votre site. Par exemple, en recherchant sur Google :

```
Site :www.monsite.com phpinfo " Zend engine "
```

vous aurez une bonne idée de ce que Googlebot a trouvé sur votre site. Testez rapidement avec les moteurs les plus utilisés sur votre site pour être tranquille.

Sachez qu'en 2006, nexen.net a publié des statistiques de configuration PHP, basées sur un échantillon de 11 000 `phpinfo()` recensés dans les moteurs de recherche : `http://www.nexen.net/articles/dossier/statistiques_phpinfos.php` et `http://www.nexen.net/articles/dossier/statistiques_phpinfos_:_2eme_partie.php`. Le vôtre y est peut-être…

Messages d'erreur

Avec MySQL, cet affichage est le rôle des fonctions `mysqli_error() et mysqli_errno()`. De nombreuses bibliothèques proposent un accès aux messages d'erreur via une fonction spécifique.

`mysql_error()` retourne le message d'erreur indiqué par le serveur. S'il n'y a pas eu d'erreur, une chaîne vide est affichée. En pratique, la fonction `mysql_error()` est utilisée comme ceci :

```
$res = mysql_query($requete) ;
echo mysql_error();
```

Durant le développement, cette technique permet d'avoir un retour rapide sur un message d'erreur potentiel qui apparaîtrait durant l'exécution de la requête. Cependant, en production, ce message d'erreur, s'il survient, sera affiché directement aux utilisateurs.

Contrairement aux messages d'erreur de PHP, `mysql_error()` ne sera pas masqué par la désactivation de la directive `display_errors` ! Cela sera encore plus dangereux si la requête SQL est elle-même affichée, en plus du message d'erreur.

`mysql_error() et mysql_errno()` sont utiles si elles sont enregistrées dans un fichier de log, pour traitement ultérieur, ou bien envoyées par courrier électronique: bref, lorsqu'elles sont envoyées à un responsable de l'application et non pas à l'utilisateur du site web.

D'une manière générale, les conseils consacrés à cette fonction peuvent être adaptés à toutes les fonctions qui retournent les numéros d'erreur et les messages d'erreur d'une bibliothèque externe : `curl_error()`, `imap_errors()`, `pg_last_error()`, etc.

php_logo_guid()

`php_logo_guid()` affiche simplement un logo PHP et, lorsque c'est le premier avril, un œuf de Pâques, ou *easter egg* en anglais, c'est-à-dire une image surprise à la place du logo PHP : c'est une blague inoffensive du groupe de développement. Cette image peut indiquer la version de PHP, même si ce n'est pas très précis comme solution, ou que certaines institutions ont remplacé cette image par d'autres.

Figure 6-1
Surprise lors de l'utilisation de php_logo_guid

Préférez l'utilisation d'un logo statique à celle de cette fonction.

assert()

Comme `eval()`, `assert()` exécute du code PHP passé sous forme de chaîne, mais contrairement à cette première, `assert()` peut être utile dans la programmation quotidienne.

Lorsque vous êtes en production, assurez-vous que `assert.active` est bien désactivée : de cette manière, `assert()` sera rendue automatiquement inopérante sur le serveur.

Idéalement, vous pourriez purement et simplement supprimer ces fonctions entre la phase de développement et la phase de production. Si vous disposez d'un système de mise en production, ajoutez donc un script qui analyse le code PHP publié et supprime toutes les occurrences de la fonction `assert()` : cela se fait par une simple commande de remplacement, c'est encore plus sûr et cela permet de gagner en rapidité d'exécution.

Interfaces externes

Les fonctions suivantes établissent les communications entre PHP et des système externes, tels que le système d'exploitation, le réseau, le serveur web, les bases de données ou encore le serveur de courrier électronique.

Connexion aux bases de données

Avec MySQL, cette connexion est assurée par les fonctions `mysqli_connect()`, `mysql_connect()`, `mysql_pconnect()` et `$mysql->connect()`. Si vous utilisez une autre base de données, il existe un autre jeu de fonctions pour ouvrir l'accès aux données : c'est ces fonctions qu'il faut surveiller.

`mysqli_connect()` assure la connexion au serveur MySQL. Elle permet d'accéder au serveur local, mais dispose aussi des technologies nécessaires pour se connecter à distance. Il est donc important de bien vérifier les valeurs qui sont utilisées avec cette fonction.

Notamment, demandez-vous s'il est possible de détourner le nom d'hôte de la connexion pour établir cette dernière vers un site externe. Une fois la connexion établie avec le serveur, il ne sera plus possible de faire la différence entre un serveur valide et un serveur distant.

Placez les informations de connexion dans une constante, de manière à être certain que les valeurs sont bien les mêmes à travers tout le script et ne risquent pas d'être altérées.

D'une manière générale, les conseils consacrés à cette fonction peuvent être adaptés à toutes les fonctions de connexion, comme `imap_open()`, `pg_connect()`, `sqlite_open()`, `mysql_connect()`, etc.

Exécution de commandes SQL

Avec MySQL, les commandes SQL sont prises en charge par `mysqli_query()`, `mysql_query()`, `mysqli_multi_query()` et `mysql_execute()`. Si vous utilisez une autre base de données, il faudra surveiller les fonctions qui envoient les commandes au serveur.

`mysqli_query()` est la fonction qui exécute une requête. Lorsqu'elle est appelée, la requête SQL est transmise au serveur pour y être traitée. C'est donc un point de passage obligé pour transmettre des commandes au serveur SQL. Avant de l'utiliser, il convient donc que s'assurer que la commande a été bien construite.

Dans le cas de MySQL, il y a plusieurs techniques qui servent à neutraliser les valeurs manipulées, de manière à ce qu'elles ne puissent pas perturber la commande SQL : variables serveur, commandes préparées, procédures stockées (reportez-vous à la section sur les injections SQL).

D'une manière générale, les conseils consacrés à cette fonction peuvent être adaptés à toutes les fonctions d'exécution de commandes qui passent par une chaîne de caractères pour construire la commande, comme `setrawcookie()`, `pg_connect()`, `sqlite_query()`, etc.

mail()

`mail()` est bien sûr la fonction par laquelle votre serveur devient un serveur de spam. Il est donc particulièrement important de la surveiller.

Il faut notamment éviter les injections d'en-têtes et les utilisations abusives du formulaire.

Pour les injections d'en-tête, étudiez comment les variables provenant de l'utilisateur sont utilisées dans la partie en-tête du courrier électronique. Il faut éviter les retours à la ligne avec les caractères `\r` et `\n`, qui introduisent de nouveaux en-têtes, ainsi que les virgules, qui permettent d'envoyer plusieurs informations en même temps. Laissez bien plusieurs lignes vides entre les en-têtes et le corps du message.

Plutôt que d'envoyer des messages au nom de votre visiteur, une bonne pratique consiste à envoyer des messages depuis votre site web et au nom de votre site. L'utilisateur final reçoit alors un message qui dit, en substance : « un ami, M. XXX, a pensé que cette

information vous intéresserait ». De cette manière, l'utilisateur final reconnaît son ami, ainsi que votre site, plutôt que de croire que le message provient d'une autre source.

Pensez à ajouter des en-têtes de courrier complémentaires, qui vous permettront de remonter à la source du spam : IP de l'utilisateur, URL du formulaire appelant, cookies, *user agent*, date de la transaction, utilisation d'un proxy, etc. Toutes ces informations seront précieuses pour savoir comment votre serveur a été détourné de sa mission originale. Pour en savoir plus, reportez-vous à la section sur le détournement de sites web pour en faire des serveurs de spam.

Appels système

Les appels système sont effectués par six fonctions et un opérateur : `system()`, `passthru()`, `exec()`, `shell_exec()`, `popen()`, `proc_open()` et les guillemets obliques `` ` ``.

Ces six fonctions transmettent directement des chaînes de caractères au système d'exploitation pour exécution. Avec PHP, faire appel à un programme externe est rare, mais ce n'est pas exceptionnel. Ces fonctions répondent à un besoin clair, mais, dans tous les cas, leur utilisation doit être surveillée de très près. Il faut notamment protéger les arguments qui leur sont transmis à l'aide de des fonctions de protection `escapeshellcmd()` et `escapeshellarg()`.

L'utilisation de programmes externes à PHP est consommateur de ressources. Il faut noter que ces consommations ne sont pas comptabilisés par PHP, ni en temps processeur, ni en mémoire. Comme pour `eval()`, demandez-vous si vous ne pourriez pas faire la même chose en PHP. Dans la pratique, tâchez de les éviter.

Si vous n'en avez pas besoin, pensez à les désactiver à l'aide de la directive `disable_functions`.

Accès aux fichiers distants

De nombreuses fonctions PHP sont capables d'aller chercher des fichiers à distance, sans être spécifiquement des fonctions de réseau : `fopen()`, `getimagesize()`, `file()`, `file_get_contents()` et `file_exists()`.

Ces cinq fonctions ont la capacité d'accéder à des contenus distants, dès que la directive `allow_url_fopen` est activée. Durant la lecture des contenus distants, PHP attend que les données arrivent, puis il reprend son exécution. Si le serveur distant est lent ou même inaccessible, il faudra attendre que PHP dépasse les délais d'attente maximale pour abandonner et reprendre son exécution. Cela peut occuper le serveur durant longtemps et même si cela ne consomme pas de ressources en termes de mémoire ou de processeur, cela représente un processus complet en attente : durant ce temps, il ne sert pas d'autres clients. Comme le nombre de ces processus est limité, cela peut rapidement conduire à un déni de service.

Il n'est pas possible de s'assurer qu'une ressource est disponible avant de tenter d'y accéder. Cependant, vous pouvez multiplier les tests avant de lancer la lecture : vérifier que l'URL est bien structurée avec `parse_url()`, que le nom de domaine existe bien avec `checkdnsrr()`. Cela devrait vous permettre d'écarter beaucoup de cas problématiques.

Dans cet exercice, la maxime à adopter est : « échouez le plus tôt possible », c'est-à-dire que tout test qui vous permet d'annuler la lecture le plus tôt possible doit être fait avant de lancer le test final : la lecture elle-même.

Ces cinq fonctions sont aussi à surveiller quand elles utilisent des fichiers locaux. Il est important de bien savoir quels fichiers seront ouverts. Pour cela, utilisez la fonction `realpath()` avec le fichier et son chemin d'accès avant l'ouverture : cela vous donne le chemin réel qui sera utilisé et vous permet de détecter des accès aux dossiers interdits.

Par exemple, si vous construisez dynamiquement un chemin d'accès à un fichier et que vous obtenez :

```
/home/./web/.././../etc/passwd.txt
```

alors `realpath()` va vous retourner :

```
/etc/passwd.txt
```

Cela va aussi résoudre les liens symboliques et tous les problèmes d'encodages, à l'aide de votre système.

Accès aux variables d'environnement

`ini_set()`, `ini_get()`, `getenv()`, `putenv()` : ces quatre fonctions servent à lire ou modifier les directives de configuration de PHP et les variables d'environnement du serveur. Leur utilisation n'est pas synonyme de vulnérabilité, mais c'est un point d'entrée pour modifier la configuration du serveur et en perturber le fonctionnement. Par exemple, avec `ini_set()`, on peut lever la limite de consommation de mémoire ou de processeur de PHP. Il convient donc de surveiller les valeurs utilisées par ces fonctions.

Gestion des erreurs

Les messages d'erreur communiquent des informations sur l'état de l'application et les problèmes qu'elle rencontre. Ces messages sont clairement destinés aux développeurs et non pas aux pirates. Il faut savoir les maîtriser, au lieu de les laisser s'afficher par défaut dans la page.

Exceptions ou messages traditionnels ?

Que vous utilisiez les messages d'erreur traditionnels ou les exceptions, les aspects sécurité sont les mêmes. Durant l'exécution d'un script, les messages d'erreur sont affichés directement dans le script et les exceptions qui ne sont pas interceptées remontent jusqu'au script principal, où elles sont finalement affichées dans le résultat : évidemment, les exceptions qui sont interceptées, le sont proprement.

Affichage des erreurs par défaut

La directive la plus importante est `display_errors`, qui affiche ou non les messages d'erreur dans le résultat du script. La meilleure chose à faire est de l'activer sur les serveurs de test et de la désactiver sur les serveurs de production.