Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer des pages Web »

Sécurité en PHP

Apprentissage

Mise en pratique

Evaluation

TABLE DES MATIERES

1. SECURITE AU NIVEAU DES DONNEES TRANSMISES PAR UN FORMULAIRE.

Lors de la validation d'un formulaire en PHP il faut utiliser la fonction : **filter_var**, cette fonction PHP valide et/ou « nettoie » les données. Cette fonction s'utilise à l'aide de **filtres** passés en paramètre.

La fonction PHP **filter_var** permet

1. de valider la forme d'une chaine de caractères attendue suivant son usage (exemple : adresse e−mail, URL, nombre réel, adresse IP, etc.).

2. de nettoyer une chaine de caractères attendue suivant son usage (élimination des caractères inattendus compte tenu du type de données (exemple : élimination d'un caractère inattendu '@' dans un nombre entier).

Le prototype de la fonction **filter_var** est :

mixed filter_var(mixed $variable, int $filter = FILTER_DEFAULT, mixed $options)

• La fonction retourne false en cas de données invalides avec échec du filtre, ou les données elles-mêmes dans le cas de données valides, ou encore les données filtrées en cas de filtres de nettoyage.

• $variable est la valeur à filtrer ;

• $filter est le type de filtre. Bien qu'il soit en option et qu'il est une valeur par défaut définie dans la configuration du serveur (fichier php.ini), il est fortement conseillé de spécifier un, ou plusieurs, filtres)

• $options définit les options et/ou les flags du filtre, plus ou moins strictes (c'est-à-dire que ces filtres n'éliminent pas les mêmes caractères suivant les options choisies).

En toute généralité, les options et les flags sont définis dans un tableau associatif (avec deux clés facultatives 'options' et/ou 'flags') de tableaux associatifs chacun de ces tableaux associatifs définissant les valeurs d'une ou plusieurs options (pour le tableau $options['options']) ou d'un ou plusieurs flags (pour le tableau $options['flags']).

Lien utile : *https://www.php.net/manual/fr/function.filter-var.php*

1.1 Les Filtres de validation.

Les filtres de validation sont les valeurs possibles du deuxième paramètre filter de la fonction **filter_var** qui commencent par **FILTER_VALIDATE_**.

Le but d'un filtre de validation est de dire si une chaîne satisfait certaines condition ; si la forme de la chaîne est conforme à ce qu'on attend d'un certain type de données.

La liste n'est pas très longue.

La plus grosse difficulté vient, comme d'habitude, des conversions automatiques entre formats de nombre et booléens, qui produisent des résultats contrintuitifs qui peuvent conduire à des bugs.

Exemple d'utilisation de cette fonction :

```php
<?php
$email = "sacha@gmail.com";

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo ("$email est une adresse email valide !!!");
} else {
    echo ("$email n'est pas une adresse email valide !!!");
}
?>
```

Les filtres de **validation** les plus utilisés sont :

FILTER_VALIDATE_EMAIL,

FILTER_VALIDATE_INT,

FILTER_VALIDATE_BOOLEAN,

FILTER_VALIDATE_IP,

FILTER_VALIDATE_URL,

…

1.2 Les Filtres de nettoyage.

Les filtres de nettoyage permettent d'appliquer un traitement à une chaîne pour la rendre conforme à la forme attendue des données.

C'est aussi une manière de sécuriser des données incorrectes sans renvoyer les données vers l'utilisateur.

Les filtres de nettoyage sont les valeurs possibles du deuxième paramètre de filter_var qui commencent par **FILTER_SANITIZE**, ce qui signifie en anglais "rendre raisonnable", "rendre hygiénique" ou "ramener à la raison".

Le fait d'appliquer un filtre de nettoyage **améliore la sécurité** mais, en général, cela ne permet pas de rendre correctes des données incorrectes.

Cela permet juste de rendre les données "**raisonnables**".

Par exemple, le filtre **FILTER_SANITIZE_NUMBER_INT** Supprime tous les caractères sauf les chiffres, et les signes plus et moins.

Cela n'empêche pas que si la donnée en entrée n'est pas un nombre, le programme risque de ne pas fonctionner correctement.

Le filtre **FILTER_SANITIZE_STRING** permet de supprimer ou d'encoder différents jeux de caractères spéciaux (suivant la valeur du flag).

Les filtres de **nettoyage** les plus utilisés sont :

FILTER_SANITIZE_STRING,

FILTER_SANITIZE_FULL_SPECIAL_CHARS,

FILTER_SANITIZE_URL,

…

1.3 Le Filtre Personnalise FILTER_CALLBACK.

Ce filtre est un exemple dans lequel on va fournir à **filter_var** une fonction **callback** personnalisée, que l'on codera soi-même, et qui sera appelée automatiquement par **filter_var** pour valider ou nettoyer une chaîne.

Exemple d'utilisation de ce filtre personnalisé :

```php
<?php
    function numeroTelephone($tel){

        if(preg_match("/^(0{1}[0-9]{1}([0-9]{2}){4})$/",$tel))
        {
            return $tel;
        }else{
            return false;
        }
    }

    $telephone="0678541957";
    echo "Numéro de Téléphone : ".$telephone."<br>";

    if(filter_var($telephone, FILTER_CALLBACK, array("options"=>"numeroTelephone"))!==false){
        echo"Numéro de téléphone valide.<br/>";
    }else{
        echo"Numéro de téléphone invalide.<br/>";
    }
?>
```

Tests de `filter_var` avec filtre `FILTER_CALLBACK`

Numéro de Téléphone : 0678541957
Numéro de téléphone valide.

1.4 Les Expressions Regulieres.

Les expressions régulières sont un moyen de vérifier d'une manière très générale qu'une chaîne de caractère a une certaine forme. Nous venons d'en voire un exemple d'utilisation ci-dessus.

L'extension PRCE en PHP permet, via des fonctions comme **preg_match** rencontrée dans l'exemple ci-dessus, de vérifier qu'une chaîne est conforme à une expression régulière.

Un bon tutorial sur la syntaxe et l'utilisation des expressions régulières en PHP se trouve à l'adresse suivante : *https://www.php.net/manual/en/book.pcre.php*

Par exemple, pour valider un nom ou un prénom entre 1 et 50 caractères alphabétiques français avec des espaces ou traits d'unions, on peut utiliser le modèle suivant :

/^(([a-zA-ZàâéèêôùûçÀÂÉÈÔÙÛÇäöëüÄÖËÜ](\-|( )*)){1,50})$/

Notez que pour que le filtrage puisse servir à **éviter les injections**, il ne faut pas omettre le **^** au début et le **$** à la fin de l'expression pour que l'expression

régulière représente l'ensemble de l'input utilisateur et non pas simplement une sous-chaîne.

Afin de vous familiariser avec la validation des formulaires en PHP et avec l'utilisation de la fonction **filter_var** vous pourrez regarder le code source de l'application : **Validation_formulaire** joint dans l'archive.

2. INJECTIONS XSS, FILTRAGE, EXPRESSIONS REGULIERES

2.1 INJECTIONS HTML.

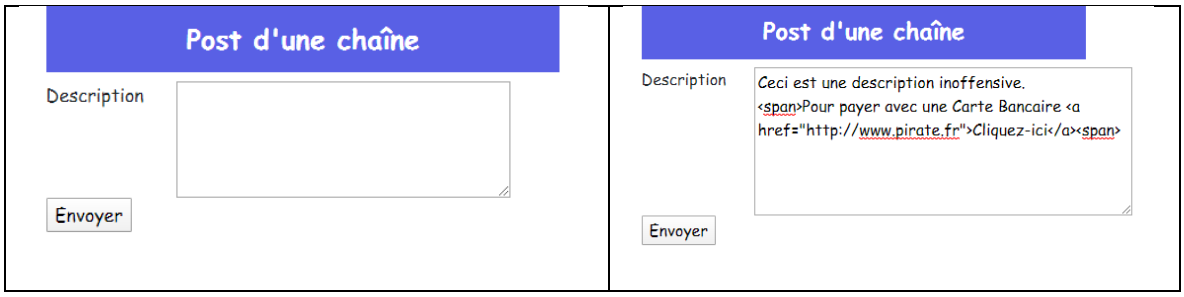
Les **injections XSS** sont un moyen pour un pirate d'exécuter du code non prévu en exploitant les interfaces entre **PHP** et d'autres langages (HTML, JavaScript, SQL, etc.…).

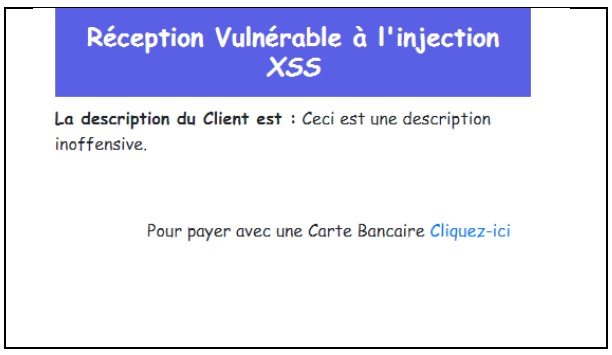
Pour cela, le pirate rentre dans un input du code, qui n'est pas détecté par PHP (on a simplement une chaîne de caractères au niveau de PHP), mais qui est interprété par un autre langage interfacé avec PHP.

Voyons un exemple d'injection HTML.

L'utilisateur malveillant va entrer dans un **textarea**, de nom "description", du code HTML pour introduire dans le site victime un lien vers un site pirate.

Si l'utilisateur non averti ou distrait clique sur ce lien, le pirate peut alors demander à l'utilisateur de rentrer ses identifiants (usurpation d'identité) ou ses données de carte bancaire (escroquerie), etc…





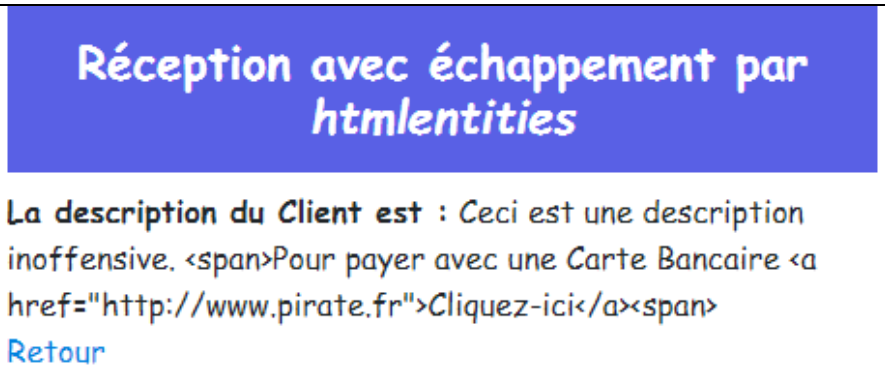
2.2 PREVENTION DES INJECTIONS HTML PAR ECHAPPEMENT.

Différents outils de filtrage sont disponibles en PHP. Le plus simple pour la sécurité consiste à utiliser la méthode **htmlentities** qui transforme dans une chaîne tous les caractères spéciaux en leurs entités HTML (code spécial pour afficher un caractère en HTML).

Il faut cependant prendre garde que si l'utilisateur ne rentre pas les caractères en entrée avec le même encodage que celui utilisé par PHP en sortie, les caractères spéciaux n'ont pas le même code et la fonction **htmlentities** ne fonctionnera pas bien, laissant la porte ouverte à des attaques. On peut spécifier l'encoding de sortie de **htmlentities** dans son troisième paramètre.

Lien : *https://www.php.net/manual/fr/function.htmlentities.php*

```php
<?php
if(isset($_POST["description"])){
    echo htmlentities( $_POST["description"], ENT_COMPAT, "UTF-8");
}
echo"<p><a href='index.php'>Retour</a></p>";
?>
```



Remarque :

ENT_COMPAT Convertit les guillemets doubles, et ignore les guillemets simples.

2.3 INVERSER L'ECHAPPEMENT.

Après un échappement à réception des données, on peut inverser cet échappement pour **restaurer** les données bruts d'origine, par exemple pour renvoyer ces données dans les inputs d'un formulaire.

Pour cela vous devrez utiliser la fonction PHP : **html_entity_decode()**

```php
<?php
if(isset($_POST["description"])){
    echo html_entity_decode(htmlentities( $_POST["description"], ENT_COMPAT, "UTF-8"), ENT_COMPAT, "UTF-8");
}
echo"<p><a href='index.php'>Retour</a></p>";
?>
```

Voici un exemple d'utilisation de cette fonction, si vous testez l'exemple précédent vous verrez bien que cette fonction annule les effets de la fonction **htmlentities().**
Ce sera donc parfait si vous devez par exemple restituer une valeur insérée avec la fonction PHP **htmlentities()** dans le champ d'une table d'une BDD.

2.4 Encodage d'une URL.

Il faudra toujours encoder vos URL, pour cela PHP met à votre disposition la fonction **urlencode($url)** et son pendant la fonction urldecode($url).

Lien : *https://www.php.net/manual/fr/function.urlencode*

2.5 Utilisation d'un generateur de Template comme TWIG.

Une autre façon de lutter efficacement contre les **injections XSS,** c'est d'utiliser un **générateur de Template** comme **Twig.**

Nous verrons à travers un TP comment installer **Twig** dans un projet PHP en utilisant **Composer** qui est un **gestionnaire de dépendances** en PHP. (*https://getcomposer.org/*). **Composer** est à PHP ce que **NPM** est à Node.js.

Twig permet d'améliorer les **performances du site**, de **sécuriser** l'application web, **d'alléger le code** et aussi d'offrir une **séparation forte** entre les couches **Front-End** et **Back-End** d'une application web.

3. La faille CSRF

CSRF (Cross-Site Request Forgery) est une **attaque** qui usurpe l'identité d'un utilisateur de confiance et envoie des commandes non désirées sur un site web.

Ce type d'attaque consiste simplement à faire exécuter à une victime une **requête HTTP** à son insu. Le but est de faire aller la victime sur une page pour qu'il

exécute les actions de la page, avec ses privilèges (généralement plus élevés que ceux du pirate).

Cela peut être réalisé, par exemple, en ajoutant des paramètres malveillants dans une URL associée à un lien qui prétend aller quelque part mais qui va ailleurs.

Pour parer à cette attaque, le développeur doit mettre en place un système **d'authentification par jeton.**

Un jeton (aussi appelé **token** en anglais) est un **nombre** ou **une chaîne de caractère aléatoire** qui va être testée avant toute modification ou édition d'un article. Ceci dans le cadre d'une modification d'un article, par exemple la suppression d'un article.

Dans un fichier PHP global on va écrire le code suivant :

```php
if (!isset($_SESSION['jeton'])) {
    $_SESSION['jeton'] = bin2hex(openssl_random_pseudo_bytes(8));
}
```

Ce code signifie : Si le jeton de session n'est pas encore défini, on le génère aléatoirement et on l'enregistre pour la session courante.

La fonction *openssl_random_pseudo_bytes()* génère une chaîne pseudo-aléatoire d'octets de taille 8 bits * 2 qu'on convertit ensuite en hexadécimal, 8 étant le nombre donné en paramètre de la fonction (on peut le changer).

Ensuite, à chaque connexion d'un utilisateur, on va devoir générer un jeton **qui lui est propre**.

Pour cela, on peut avant chaque connexion régénérer le jeton, en **supprimant** le jeton de la session précédente :

```php
unset($_SESSION['jeton']);
```

Il reste ensuite à modifier dynamiquement les **liens de suppression**, admettons que le lien soit écrit sous cette forme :

```php
<a href="http://www.liste_articles.fr/supprime.php?id="<? echo get_article_id(
);?>>Supprimer l'article</a>
```

On peut le remplacer par :

```php
<a href="http://www.liste_articles.fr/supprime.php?id="<? echo get_article_id(
). "&jeton=". $_SESSION['jeton']; ?>>Supprimer l'article</a>
```

Ainsi l'url de suppression s'affichera comme ceci :

```
http://www.liste_articles.fr/supprime.php?id=1&jeton=b6cf20590a57f4685c9bdc6c53d12ff8
```

Et non plus comme cela :

```
http://www.liste_articles.fr/supprime.php?id=1
```

Et enfin, dans notre fichier de suppression *supprime.php*, on va s'assurer qu'il **existe** un **jeton** et que ce **jeton** soit **valide**.

```php
<?php
    if (
        isset($_GET['id']) && isset($_GET['jeton']) &&
        ($_GET['jeton'] == $_SESSION['jeton'])
    ) {
        supprimer_article($_GET['id']);
    } else {
        die("ID ou jeton de session invalide.");
    }
?>
```

Ce qui signifie : Si l'id de l'article est défini mais aussi le jeton et que ce jeton correspond au jeton de la session actuelle, alors on peut supprimer l'article.

Que ce soit pour éviter les suppressions par erreur ou les tentatives d'exploitation de la faille **CSRF**, il est **indispensable** de demander aussi la **confirmation** de suppression d'un article.

4. INJECTIONS SQL

Une ***injection SQL*** consiste à entrer dans les **inputs utilisateur** du code *SQL*. Ces attaques sont particulièrement dangereuses car elles peuvent être exploitées par un pirate pour :

- Accéder à des données confidentielles, par exemple à toutes les données de la BDD ;
- Détruire ou altérer des données, par exemple supprimer la totalité d'une table.

Afin de vous prémunir face à ce genre d'attaque il faudra utiliser systématiquement des **requêtes préparées** en utilisant l'extension **PDO** (Ou au pire l'extension **MySQLi**).

Lien : *https://www.php.net/manual/fr/pdo.prepared-statements.php*

5. GESTION DES MOTS DE PASSE

Stocker un mot de passe ne se fait pas de n'importe quelle façon et surtout pas en clair.

C'est certes très pratique, car on peut redonner facilement son mot de passe à un utilisateur mais ce n'est pas sécurisé ! Si une personne malveillante accède à vos bases de données, il accède à toutes les informations des utilisateurs. Et comme un utilisateur utilise très souvent le même mot de passe partout (ce qui est une grosse erreur), vous offrez accès à bien plus que votre propre site.

Hacher les mots de passe est donc indispensable !!!

Pour faire ce travail de « *hachage* » correctement PHP met à votre disposition la fonction **password_hash().**

password_hash (string $password, int $algo [, array $options]) : string

Lien : *https://www.php.net/manual/fr/function.password-hash.php*

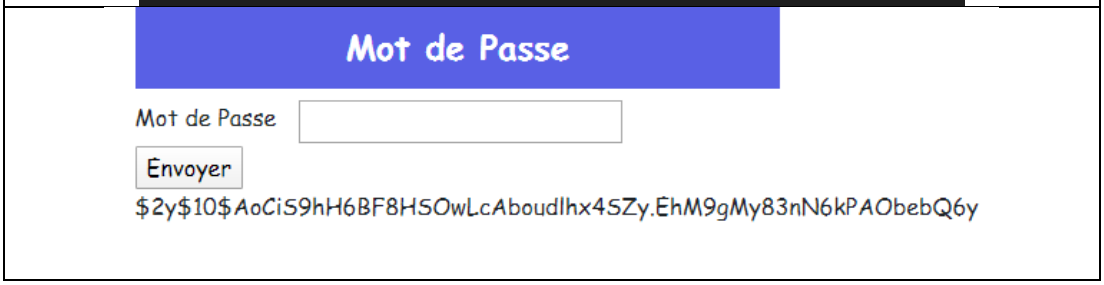
```php
<body>

    <h1>Mot de Passe</h1>
    <form action="#" method="post">

        <label for="pwd">Mot de Passe</label>
        <input type="text" name="pwd" id="pwd">
        <br>
        <input type="submit" value="Envoyer">
    </form>
    <?php
        $pwd = isset($_POST["pwd"]) ? $_POST["pwd"] : "";
        if(!empty($pwd)){
            echo password_hash($pwd, PASSWORD_BCRYPT);
        }

    ?>

</body>
```



Ici nous utilisons la constante PASSWORD_BCRYPT - Utilisation de l'algorithme CRYPT_BLOWFISH pour créer la clé de hachage.

Ceci va créer une clef de hachage standard **crypt()** utilisant l'identifiant "$2y$". Le résultat sera toujours une chaîne de 60 caractères, ou FALSE si une erreur survient.

Vous l'aurez peut-être remarqué, le **hash** d'une même chaîne ne donne jamais le même résultat, dans ce cas comment le comparer ?

PHP met à disposition la fonction **password_verify()**, pour cela :

```php
<?php
    $pwd = isset($_POST["pwd"]) ? $_POST["pwd"] : "";
    if(!empty($pwd)){
        echo password_hash($pwd, PASSWORD_BCRYPT)."<br>";

        if(password_verify($pwd, password_hash($pwd, PASSWORD_BCRYPT))) {
            echo 'Vérification OK';
        } else {
            echo 'ERREUR';
        }
    }
?>
```

6. SESSION ET SECURITE

Dès qu'un utilisateur se connecte à un site internet, le serveur attribue à cet utilisateur un ID unique pour l'identifier : **PHPSESSID**

Si un hacker explore les agissements de l'utilisateur il peut à tout moment récupérer cet ID de Session et se faire passer pour l'utilisateur. Ce qui pose évidemment des problèmes de sécurité.

Nous allons voir comment se prémunir face à ce genre d'attaque, ou *d'usurpation d'identité*.

Il va falloir prévoir un système de communication entre le Client et le Serveur.

Il faudra mettre en place un **système de ticket**, voici le principe :

Le serveur génère un ticket qu'il garde en mémoire. Il le stock ensuite dans les **cookies** de l'utilisateur. A chaque fois que l'utilisateur demande une nouvelle page, le serveur vérifie qu'ils ont bien le **même ticket** avant d'en générer un nouveau pour la page suivante.

1. L'utilisateur se connecte : **début de la session**
2. Il se rend sur la page "*index.php*".
3. Le serveur génère un ticket qui a pour valeur "ticket001"
4. Il l'enregistre simultanément dans une variable de **session** et dans les **cookies** de l'utilisateur
5. L'utilisateur change de page
6. Le serveur vérifie qu'ils ont tous les deux le même ticket
7. Il génère un nouveau ticket qui a pour valeur "ticket002"

Si le hacker vole la **Session**, il se retrouvera avec un **ticket aléatoire** et donc forcément différent de celui de l'utilisateur. Lorsqu'il essaye de se rendre sur une page, le **Serveur** réalise que les tickets sont différents et la **Session est détruite**.

Au départ lorsque l'utilisateur arrive sur la page d'**Accueil** (*index.php*)

```php
<?php
session_start();
$cookie_name = "ticket";
// On génère quelque chose d'aléatoire
$ticket = session_id().microtime().rand(0,9999999999);
// On hash pour avoir un ID  qui aura toujours la même forme
$ticket = hash('sha512', $ticket);
// On vérifie que le Navigateur du Client accepte les cookies
if(!isset($_COOKIE[$cookie_name])) {
    echo "<p>Le navigateur n'accepte pas les cookies</p>";
}

// On enregistre des deux cotés
setcookie($cookie_name, $ticket, time() + (60 * 20)); // Expire au bout de 20
min
$_SESSION['ticket'] = $ticket;
```

```php
echo "<h1>Page Accueil</h1>";
// On affiche la variable de Session générée
echo $_SESSION['ticket'];

// On affiche un lien pour passer à la page suivante
echo "<br><a href='page1.php' >Page1</a>";
?>
```

Lorsque l'utilisateur clique sur le lien pour passer à la **Page1** (*page1.php*)

```php
<?php
session_start();

if ($_COOKIE['ticket'] == $_SESSION['ticket'])
{
    $ticket = session_id().microtime().rand(0,9999999999);
    $ticket = hash('sha512', $ticket);
    $_COOKIE['ticket'] = $ticket;
    $_SESSION['ticket'] = $ticket;

    echo "<h1>Page 1</h1>";
    echo $_SESSION['ticket'];

}else{
    // On détruit la session
    $_SESSION = array(); // On détruit ainsi toutes les variables de session
    session_destroy();
    header('location:index.php');
}
?>
```

Vous pourrez vérifier que ces scripts fonctionnent bien, lorsque vous actualiser la **Page1**, vous êtes automatiquement renvoyé sur la page d'**Accueil** puisque la valeur de la variable de **Cookie** et celle de la variable de **Session** ne correspondent plus.

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)
Formateur : Alexandre RESTOUEIX

Date de mise à jour : 29/01/20