Le filtrage et l'échappement des données en PHP 5 : introduction à la technique du Poka-Yoké

par Guillaume Rossolini (Tutoriels Web)

Date de publication : 5 juin 2007

Dernière mise à jour :

Le filtrage des données est une étape essentielle trop souvent négligée, fût-ce par ignorance ou par négligence.
Nous allons étudier une méthode qui nous permettra de réduire les chances de faire des erreurs lors du filtrage les données, que ce soit en provenance ou à destination de l'extérieur de vos applications PHP 5 : le **filtrage Poka-Yoké**.

Je n'ai pas finalisé ce framework. Les tests n'ont pas tous été faits, je n'ai pas tout validé. Je publie cette documentation et mes sources simplement parce que je ne donnerai plus suite au projet mais je n'aimerais pas pour autant avoir perdu tout ce travail. Si cela peut en inspirer certains, ce sera parfait. Pour ma part, j'arrête de développer un framework à part et je m'aligne sur les frameworks existants.

I - Introduction

I-A - Remerciements

Je remercie chaleureusement tous ceux qui m'ont aidé lors de la conception de ce framework : **pcaboche**, **Yoshio**, **wamania** et **doctorrock**, sans oublier Marco Tabini pour avoir fourni l'idée de départ.

I-B - Problématique

Les développeurs font souvent des erreurs tandis qu'un ordinateur n'oublie rien. Ainsi, il est souvent plus fiable de déléguer à l'ordinateur tout ce qui est sensible, plutôt que d'essayer de nous en occuper nous-mêmes.

La méthode du Poka-Yoké nous oblige à éliminer les mauvaises solutions de manière à ne conserver que la bonne.

Le code suivant, aussi simple soit-il, est une faille de sécurité (XSS) :

```
echo $_GET['test'];
echo $_POST['login'];
```

En effet, il suffit d'insérer du code Javascript dans le contenu de la variable "test" de l'URI pour obtenir une faille XSS.

Cependant, il existe une solution simple pour résoudre ce problème :

```
echo htmlentities($_GET['test'], ENT_QUOTES, 'utf-8');
echo htmlentities($_POST['login'], ENT_QUOTES, 'utf-8');
```

Mais bien sûr, utiliser ce type de code tout au long du script est fastidieux. De plus, nous n'avons aucune garantie contre les oublis du développeur : négliger d'utiliser **htmlentities()** sur une seule variable donne lieu à une faille XSS...

Une autre solution consiste à déclarer une fonction pour réduire la taille du code à écrire :

```
function html($string)
{
    return htmlentities($string, ENT_QUOTES, 'utf-8');
}


echo html($_GET['test']);
echo html($_POST['login']);
```

Chaque ligne est maintenant plus courte à écrire, mais nous n'avons toujours aucune garantie que le développeur utilise systématiquement la fonction **html()** pour protéger l'internaute d'une faille XSS.

II - Le Poka-Yoké

II-A - Principe

Pour reprendre l'exemple précédent, imaginez que l'on ait le code suivant :

```
echo $_GET->getHTML('test');
echo $_POST->getHTML('login');
```

Cela n'est pas plus complexe à écrire que la syntaxe précédente mais cela présente l'avantage énorme d'utiliser une notation objet, ce qui donne bien plus de flexibilité en amont. PHP 5 nous permet d'interdire ce type de code :

```
echo $_GET['test']
```

Le développeur est ainsi contraint à utiliser une méthode de l'objet pour en récupérer la valeur. Il nous suffit donc de donner des noms clairs à nos méthodes pour donner au développeur une idée précise de ce que contient la variable.

```
echo $_GET->getHTML('test'); // Convertir en entités HTML avant d'afficher
echo 'SELECT * FROM user WHERE login="'.$_POST->getSQL('login').'"'; // Échapper le SQL avant
 d'afficher
```

La méthode du Poka-Yoké se résume à contraindre le développeur à utiliser la seule voie correcte (parmi un ensemble de possibles). Ici, il faut le contraindre à utiliser des méthodes pour afficher les variables car cela l'oblige à voir (grâce au nom de la méthode) l'utilisation qu'il fait des variables : les erreurs d'inattention et les oublis sont devenus impossibles.

PHP 5 a introduit un modèle objet plus proche de celui adopté par d'autres langages, plus complet, plus intéressant et plus adapté à des problématiques complexes.

Nous allons utiliser le mécanisme d'exceptions proposé par PHP en version 5 ou ultérieure, afin de mettre en place un mini framework permettant de sécuriser les données transmises depuis (et vers) l'extérieur d'un script PHP.

Comme je l'ai dit précédemment, il nous suffit d'utiliser des Objects au lieu d'Arrays pour être en mesure de contrôler l'utilisation qui est faite des variables. Cependant, les tableaux sont très pratiques car ils permettent d'utiliser des solutions comme **foreach()**. PHP 5 permet de faire en sorte que nos objets se comportent comme des tableaux grâce à l'interface ArrayAccess :

```
class MyObject implements ArrayAccess
{
    public function offsetExists($field){...}
    public function offsetGet($field){...}
    public function offsetSet($field){...}
    public function offsetUnset($field){...}
}
```

La méthode qui nous intéresse le plus parmi les 4 présentées ci-dessus est **offsetGet()**, car elle permet d'obtenir la valeur d'une propriété de l'objet avec la même syntaxe qu'un Array. Il pourrait alors de nouveau utiliser les tableaux $_POST et $_GET directement, or c'est précisément ce que nous voulons éviter.

Ainsi, nous allons désactiver cette méthode :

```
    public function offsetGet($field)
    {
        throw new Exception($field);
    }
```

Il ne reste plus qu'à définir les méthodes d'accès aux données.

II-B - Préparons le terrain

Il faut commencer par préparer la configuration PHP adéquate pour le Poka-Yoké.

Pour cela, utilisons les scripts suivants :

main.php

```
<?php

// It would be best to do this in the php.ini or .htaccess
error_reporting(E_ALL | E_STRICT);


require('functions.php');

set_error_handler('errorHandler');
set_exception_handler('exceptionHandler');


if(PHP_VERSION < 5)
{
    $message = 'Your PHP version is '.PHP_VERSION.'.'
        .' You need at least PHP 5 to run these scripts.';

    trigger_error($message, E_USER_ERROR);
}


function __autoload($class)
{
    if(strpos(strtolower($class), 'exception') === FALSE)
    {
        require('classes/'.$class.'.php');
    }
    else
    {
        require('exceptions/'.$class.'.php');
    }
}

unset($_REQUEST);

if(!ini_get('session.auto_start'))
{
    session_start();
}

?>
```

functions.php

```
<?php

function displayError($message)
{
```

functions.php

```
        // Put here any visual enhancement you want (page footer, etc.)
        ?>
        <div>
            <p><span style="color: red;">Execution stopped!</span></p>
            <?php echo $message; ?>
        </div>
        <?php
        exit;
    }


    function exceptionHandler(Exception $exception)
    {
        displayError($exception->getMessage());
    }


    function errorHandler($code, $string, $file, $line)
    {
        $error = array_search($code, get_defined_constants(), TRUE);

        $message = '<p>'
            .'<span style="color: purple; font-weight: bold;">'.$error.'</span>'
            .' in file <span style="font-weight: bold;">'.$file.'</span>'
            .' on line <span style="font-weight: bold;">'.$line.'</span></p>';

        $message .= '<p><span style="font-style: italic;">'
            .nl2br($string).'</span></p>';

        displayError($message);
    }

    ?>
```

III - Exemples

Le meilleur moyen d'apprendre à utiliser le framework est de voir quelques exemples.

Un lien vers la documentation se trouve en fin d'article.

III-A - Formulaire de login

Prenons comme premier exemple un formulaire de login avec la méthode POST et composé des champs suivants :

- **login** : Le nom d'utilisateur ;
- **password** : Le mot de passe ;
- **persistant** : Si l'utilisateur souhaite rester connecté de manière permanente grâce à ses cookies.

Ce qui nous intéresse ici est le type de chacun des champs. Nous aurons vraisemblablement une variable String contenant un ou plusieurs mots, une autre contenant une valeur String quelconque et enfin une checkbox simple.

Voici ce que cela peut donner en PHP :

Login.php

```php
<?php

class Login extends DVP_DataFilter
{
    public function __construct()
    {
        parent::__construct($_POST);
        unset($_POST);

        $whiteValues = array('yes');

        $this->filterSingleLine('login');
        $this->untaint('password');
        $this->filterUsingArray('persistant', $whiteValues, DVP_DataFilter::ARRAY_STRINGS);

        $this->clean();
    }
}

?>
```

Le traitement (méthode 1) :

```php
<?php

if(!empty($_POST))
{
    //
    // Initialisation
    //
    $_POST = new Login();


    //
    // Utilisation
    //
    $sql = 'SELECT id
```

Le traitement (méthode 1) :

```
                FROM user
                WHERE login = "'.$_POST->getSQL('login').'"
                AND password = "'.md5($_POST->getRaw('password')).'"';

        $result = mysql_query($sql);
        if($user = mysql_fetch_assoc($result))
        {
            echo 'Connecté en tant que '.$_POST->getHTML('login').'<br />';
            if(!$_POST->isEmpty('persistant'))
            {
                // sessions...
            }
        }

        echo '<br /><br />';
    }

    ?>
```

Le traitement (méthode 2) :

```
    <?php

    if(!empty($_POST))
    {
        //
        // Initialisation
        //
        $_POST = new Login();


        //
        // Utilisation
        //
        foreach($_POST as $field)
        {
            echo '<b>'.$field.'</b> : '.$_POST->getHTML($field).'<br />'."\n";
        }

        echo '<br /><br />';
    }

    ?>
```

Le formulaire :

```
    <form method="post" action="">
        <label><input type="text" name="login" /> Nom d'utilisateur</label><br />
        <label><input type="password" name="password" /> Mot de passe</label><br />
        <label><input type="checkbox" name="persistant" value="yes" /> Rester connecté ?</label><br />

        <input type="submit" value="Envoyer" />
        <input type="reset" value="Rétablir" />
    </form>
```

*J'imagine que vous voyez où je veux en venir : chaque formulaire HTML dispose d'un équivalent en PHP : une classe qui hérite de la classe **DVP_DataFilter**.*

III-B - Formulaire d'enregistrement

Je ne vais proposer ici qu'un formulaire minimal. En situation réelle, il faudrait davantage de champs, comme par exemple un captcha et un **timestamp**. Ce n'est cependant pas l'objet de cet article, je laisse donc le soin à d'autres personnes de rentrer dans le détail.

Liste des champs :

- **login** : Le nom d'utilisateur ;
- **password_1** : Le mot de passe ;
- **password_2** : La vérification du mot de passe ;
- **signature** : La signature à afficher dans les forums ;
- **website** : L'adresse du site Web ;
- **e_mail** : L'adresse e-mail ;
- **avatar** : Le fichier image de son avatar ;
- **portrait** : Le fichier image de son portrait.

Ce formulaire contient à la fois des données texte et des fichiers, nous aurons donc besoin de plusieurs classes PHP pour tout couvrir (à l'image de $_POST et $_FILES).

RegisterDataFilter.php

```php
<?php

class RegisterDataFilter extends DVP_DataFilter
{
    public function __construct()
    {
        parent::__construct($_POST);
        unset($_POST);

        $this->filterSingleLine('login');
        $this->untaint('password_1');
        $this->untaint('password_2');
        $this->untaint('signature');
        $this->filterURI('website');
        $this->filterEmail('e_mail');

        $this->clean();
    }
}

?>
```

RegisterFilesFilter.php

```php
<?php

class RegisterFilesFilter extends DVP_HTTPUploadFilter
{
    public function __construct()
    {
        parent::__construct($_FILES);
        unset($_FILES);

        $this->filterSingleLine('login');
        $this->untaint('password_1');
        $this->untaint('password_2');
        $this->untaint('signature');
        $this->filterURI('website');
        $this->filterEmail('e_mail');

        $this->clean();
    }
}

?>
```

Bien entendu, ce n'est ici qu'une base de départ. En situation réelle, il faudrait s'adapter à ce que l'internaute saisit effectivement dans le formulaire, par exemple la présence ou l'omission de "http://" pour l'adresse de son site Web.

IV - Conclusion

Liens :

- Documentation : **Le framework du Poka-Yoké**, par Guillaume Rossolini ;
- Source : **Aedituus, un espace membre sécurisé**, par Guillaume Affringue ;
- Tutoriel : **Souplesse et modularité grâce aux Design Patterns**, par Pierre Caboche.

*Il existe d'autres initiatives du même genre que la mienne. Les plus connues sont **PECL Input Filter** (PHP/ext) et **Zend_Filter_Input** (Zend Framework).*