

# Développement d'applications sécurisées avec Zend Framework

Mickaël Perraud

# Qui suis-je ?

---

- Mickaël Perraud
- Contributeur Zend Framework
- Responsable documentation française
- Site : <http://mikaelkael.fr>
- Twitter : mikaelkael



---

# Partie I

## Introduction

# Introduction

---

Zend Framework devient de plus en plus populaire

La demande de lignes directrices dans le développement sécurisé d'applications basées sur Zend Framework grossit

La plupart des livres, conférences ou articles se concentre sur la programmation d'applications PHP sécurisées sans framework

L'utilisation de frameworks requiert de nouvelles lignes directrices pour la sécurité

Les frameworks sont souvent proposés avec leurs propres fonctionnalités de sécurisation

# Sujets abordés

---

- Authentification centralisée
- Validation et filtrage centralisé des entrées (« input »)
- Sécurité SQL
- Protection envers les Cross Site Request Forgery (CSRF)
- Sécurisation du management des sessions
- Protection envers les Cross Site Scripting (XSS)
- Nouvelles attaques envers les anciennes vulnérabilités

---

# Partie II

Authentification centralisée

et

Validation et filtrage centralisé des entrées

# Applications traditionnelles vs. Zend Framework

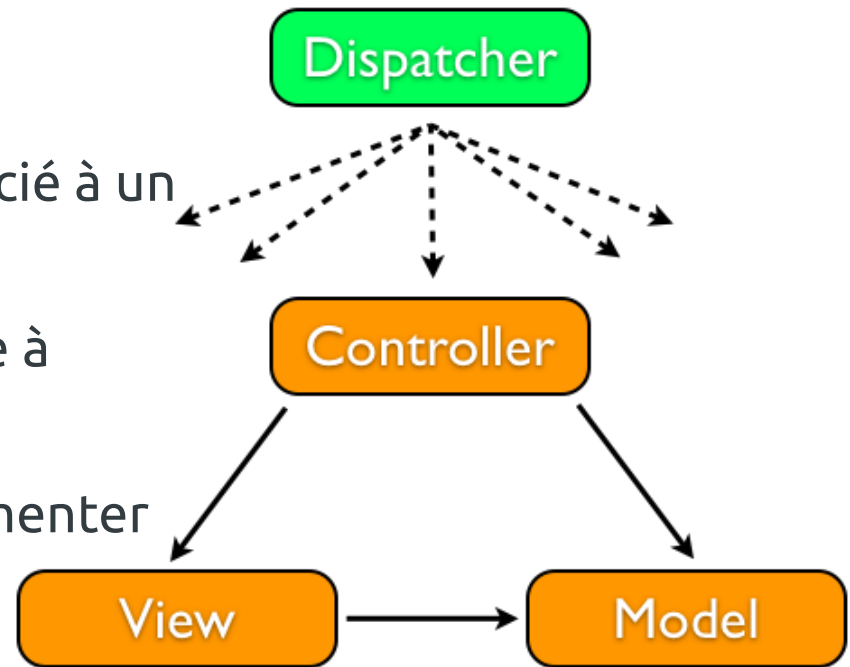
Les applications traditionnelles possèdent de multiples points d'entrées

Les applications ZF utilisent majoritairement le design MVC associé à un distributeur (« dispatcher »)

La manière traditionnelle est encline à produire des erreurs

La manière ZF vous permet d'implémenter des tâches de sécurité centralisées :

- Authentification
- Validation et filtrage des entrées



# Plugin du contrôleur frontal

---

- Ajout de fonctionnalités à `Zend_Controller_Action`
- Pas d'extension de classe requise
- Parfait pour des tâches centralisées comme l'authentification ou la validation/filtrage

```
$front = Zend_Controller_Front::getInstance();  
$front->registerPlugin(new MyPlugin());  
$front->dispatch();
```

# Authentication centralisée

---

```
class ForceAuthPlugin extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        try {
            My_Auth::isLoggedIn();
        } catch (My_Auth_UserNotLoggedInException $e) {
            if (!in_array($request->getControllerName(),
                array('login', 'index', 'error'))) {
                $request->setModuleName('default')
                    ->setControllerName('login')
                    ->setActionName('index')
                    ->setDispatched(false);
            }
            return;
        }
    }
}
```

# Validation et filtrage centralisé des entrées (I)

```
$filters['index']['index'] = array(
    '*'      => 'StringTrim',
    'month' => 'Digits'
);

$filters['login']['index'] = array(
    'login' => 'Alpha',
    'pass'  => 'Alpha'
);

$validators['index']['index'] = array(
    'month' => array(
        new Zend_Validate_Int(),
        new Zend_Validate_Between(1, 12)
    )
);

$validators['login']['index'] = array(
    'login' => array(
        new My_Validate_Username()
    ),
    'pass'  => array(
        new My_Validate_Password()
    ),
);
```

# Validation et filtrage centralisé des entrées (II)

```
class FilterPlugin extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        $params      = $request->getParams();
        $controller  = $request->getControllerName();
        $action      = $request->getActionName();

        @$filter = $GLOBALS['filters'][$controller][$action];
        @$validator = $GLOBALS['validators'][$controller][$action];

        $input = new Zend_Filter_Input($filter, $validator, $params);

        if (!$input->isValid()) {
            $request->setModuleName('default')
                ->setControllerName('error')
                ->setActionName('illegalparam')
                ->setDispatched(false);

            return;
        }
    }
}
```

# Intégration centralisée de PHPIDS

```
class Controller_Plugin_PHPIDS extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        $request = array('GET' => $request->getQuery(),
            'POST' => $request->getPost(),
            'COOKIE' => $request->getCookie(),
            'PARAMS' => $request->getUserParams());

        $init = IDS_Init::init(APPLICATION_PATH.'/config/phpids.ini');
        $ids = new IDS_Monitor($request, $init);

        $result = $ids->run();
        if (!$result->isEmpty()) {
            $compositeLog = new IDS_Log_Composite();
            $compositeLog->addLogger(IDS_Log_Database::getInstance($init));
            $compositeLog->execute($result);
        }
    }
}
```

Lien utile : <http://code.google.com/p/zids/>

---

# Partie III

## Validation et filtrage des formulaires

# Validation et filtrage des formulaires (I)

- Les formulaires ZF utilisent les filtres/validateurs de manière automatique
- Ceux-ci peuvent être ajoutés aux objets `Zend_Form_Element`
- Ils peuvent être chaînés arbitrairement

```
// Création de l'élément Nom
$name = $form->createElement('text', 'name', array('size' => 40, 'maxlength' => 40));
$name->addValidator('Alpha')
    ->addValidator('StringLength', false, array(1, 40))
    ->setLabel('Name')
    ->setRequired(true);

// Création de l'élément Message
$message = $form->createElement('textarea', 'message', array('rows' => 6, 'cols' => 40));
$message->setLabel('Message')
    ->setRequired(true)
    ->addFilter('StripTags');

// Création du bouton Envoi
$submit = $form->createElement('submit', 'send');
$submit->setLabel('Envoyer');

// Ajout de tous les éléments au formulaire
$form->addElement($name)->addElement($message)->addElement($submit);
```

# Validation et filtrage des formulaires (II)

---

- Validation du formulaire dans l'action correspondante

```
$form = My_Custom_Form();

if ($this->getRequest()->isPost()
    && $form->isValid($this->getRequest()->getPost())) {
    // Faire qqch avec les données valides reçues
    $data = $form->getValues();
    //...
}
$this->view->form = $form;
```

---

# Partie IV

## Sécurité SQL

# Injection SQL dans les applications Zend Framework

---

- ZF est fourni avec de multiples classes permettant l'accès aux bases de données
- Les méthodes supportent majoritairement les requêtes préparées (« Prepared Statements »)

```
$sql = "SELECT id FROM _users WHERE lastname=? AND age=?";  
$params = array('Mueller', '18');  
$res = $db->fetchAll($sql, $params);
```

- Une erreur dans la création des requêtes préparées permet une injection SQL
- ZF fournit aussi des fonctions d'échappement permettant la création dynamique de requêtes SQL
- Un oubli d'échappement peut entraîner de l'injection SQL

# Injection SQL + PDO\_MySQL = Danger

---

Traditionnellement MySQL ne permet l'exécution que d'une seule requête à la fois

- ext/mysql – stoppe complètement les requêtes multiples
- ext/mysli – possède une fonction distincte `mysql_multi_query()`

*ATTENTION : PDO\_MySQL n'a pas cette limitation*

Les risques d'injection SQL dans les applications ZF sont plus élevées dans celles utilisant PDO\_MySQL que dans celles qui utilisent les interfaces traditionnelles MySQL

# Zend\_Db – Échappement

---

```
function quote($value, $type = null)
```

Toujours le bon échappement – une seule fonction au lieu d'un grand nombre

ATTENTION : les chaînes sont donc mises entre les guillemets

```
function quoteIdentifiant($ident, $auto=false)
```

Fonction d'échappement pour les identificateurs

Les applications PHP traditionnelles doivent faire cette implémentation elles-même

ATTENTION : les chaînes sont donc mises entre les guillemets

# Zend\_Db\_Select

Permet de créer des requêtes de type SELECT de manière dynamique

Utilise en interne des requêtes préparées autant que possible

L'injection SQL reste possible en cas de mauvaise utilisation

ATTENTION entre autres à **WHERE** et **ORDER BY**

```
// Construire cette requête :  
// SELECT product_id, product_name, price  
// FROM "products"  
// WHERE (price < 100.00 OR price > 500.00)  
// AND (product_name = 'Apple')  
  
$minimumPrice = 100;  
$maximumPrice = 500;  
$prod = 'Apple';  
  
$select = $db->select()  
    ->from('products',  
        array('product_id', 'product_name', 'price'))  
    ->where("price < $minimumPrice OR price > $maximumPrice")  
    ->where('product_name = ?', $prod);
```

---

# Partie V

## Protection envers les attaques Cross Site Request Forgery (CSRF)

# Protection Cross Site Request Forgery (CSRF)

---

Les protections CSRF sont habituellement basées sur des jetons secrets de formulaire dépendants de la session

Zend Framework possède un `Zend_Form_Element_Hash` qui est justement un jeton de ce type intégrant son propre validateur

Les formulaires peuvent donc être immunisés des attaques CSRF en ajoutant cet élément de formulaire

```
$form->addElement('hash',  
                 'csrf_token',  
                 array('salt' => 's3cr3ts41tG%Ek@on9!'));
```

# Protection CSRF automatique

---

Cette protection doit être appliquée manuellement

En étendant la classe `Zend_Form`, il est possible de créer une nouvelle classe de formulaire intégrant automatiquement une protection CSRF

```
class Mkk_Form extends Zend_Form
{
    public function __construct()
    {
        parent::__construct();
        $this->addElement('hash',
                        'csrf_token',
                        array('salt' => get_class($this) . 's3cr3t%Ek@on9!'));
    }
}
```

---

# Partie VI

## Sécurisation du management des sessions

# Configuration des sessions

---

La configuration a un impact important sur la sécurité de la session

Les applications SSL doivent être sécurisées avec le flag *secure*

Avoir son propre nom de session et son propre espace de stockage pour chaque application

Rendre plus difficile les attaques XSS avec le flag *httpOnly*

Spécifier la durée de vie la plus courte

```
Zend_Session::setOptions(array(  
    /* if SSL server */           'cookie_secure'    => true,  
    /* own session name */       'name'           => 'mySSL',  
    /* own session storage */    'save_path'      => '/sessions/mySSL',  
    /* hardening against XSS */  'cookie_httponly' => true,  
    /* short lifetime */         'gc_maxlifetime' => 15 * 60  
));  
Zend_Session::start();
```

# Fixation de session & détournement de session

---

- Fixation de session (« Session fixation »)

Deviens légèrement plus dur avec une gestion stricte de la validation de session

Stoppé par la régénération d'un nouvel identifiant de session à chaque changement d'état

```
session_regenerate_id(true);
```

Le mieux est de l'implémenter dans le module de connexion/déconnexion de l'application

- Détournement de session (« Session Hijacking »)

Possible seulement en utilisant SSL (pour stopper l'écoute du réseau « sniffing »)

Les cookies *httpOnly* protègent envers le vol de session par XSS

Validation de session seulement d'une utilité limitée

# Validation de session (I)

---

Reconnaître les sessions valides en vérifiant aussi des informations additionnelles

Souvent recommandé pour stopper les détournements/fixations de session – mais seulement d'une utilité limitée

Zend Framework supporte la validation de session avec :

## Zend\_Session\_Validator\_HttpUserAgent

```
try {  
    Zend_Session::start();  
} catch (Zend_Session_Exception $e) {  
    // Zend_Session::regenerate_id() support is broken  
    session_regenerate_id(true);  
}  
Zend_Session::registerValidator(new Zend_Session_Validator_HttpUserAgent());
```

# Validation de session (II)

---

Faites cependant attention si vous activez une validation d'information supplémentaire

Une vérification de l'entête HTTP User-Agent sera par exemple problématique avec Microsoft Internet Explorer 8

La vérification de l'entête HTTP Accept sera aussi un problème avec d'anciennes versions d'Internet Explorer

La vérification de l'IP du client n'est pas toujours possible derrière de gros proxy (entreprise, fournisseur d'accès)

- Limiter la vérification à des réseaux de classe C / B / A
- Meilleure compatibilité avec des sites SSL

# Validation de session – Validation de l'IP du client

```
class Zend_Session_Validator_RemoteAddress extends Zend_Session_Validator_Abstract
{
    /**
     * Setup() - this method will get the client's remote address and store
     * it in the session as 'valid data'
     *
     * @return void
     */
    public function setup()
    {
        $this->setValidData( (isset($_SERVER['REMOTE_ADDR'])
            ? $_SERVER['REMOTE_ADDR'] : null) );
    }
    /**
     * Validate() - this method will determine if the client's remote addr
     * matches the remote address we stored when we initialized this variable.
     *
     * @return bool
     */
    public function validate()
    {
        $currentBrowser = (isset($_SERVER['REMOTE_ADDR'])
            ? $_SERVER['REMOTE_ADDR'] : null);

        return $currentBrowser === $this->getValidData();
    }
}
```

---

# Part VI

## Protection envers les Cross Site Scripting (XSS)

# XSS dans les applications Zend Framework

---

Symfony est fourni avec un échappement automatique des sorties

Zend Framework n'est pas fourni avec un tel mécanisme

Stopper les XSS est une tâche du développeur

Les vulnérabilités XSS apparaissent dans les vues

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
<title><?php echo $this->title; ?></title>
</head>
<body>
<h2><?php echo $this->headline; ?></h2>
<ul>
<li><a href="<?php echo $this->link; ?>">Link 1</a></li>
</ul>
</body>
</html>
```

# Protection envers les XSS (I)

---

Traditionnellement 2 alternatives :

- N'afficher que des données propres

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
<h2><?php echo $this->escape($this->headline); ?></h2>
<ul>
<li><a href="<?php echo urlprepare($this->link); ?>">Link 1</a>
</li>
</ul>
</body>
</html>
```

# Protection envers les XSS (II)

---

- N'assigner que des valeurs propres

```
$entityFilter = new Zend_Filter_HtmlEntities();
$urlFilter    = new My_Filter_Url();

$this->view->title      = $this->escape("Seite 1");
$this->view->headline   = $entityFilter->filter($this->getRequest()
                                                ->getPost('link'));
$this->view->link       = $urlFilter->filter($this->getRequest()
                                                ->getPost('link'));
```

# Sauvegarde avec Zend\_View\_Helper

Solutions traditionnelles conduisent à des erreurs – potentiellement chaque erreur == XSS

Le balayage automatique à la recherche d'échec est difficile à réaliser

Prohiber l'affichage direct de variables

N'afficher que via un Zend\_View\_Helper

La protection XSS devient alors une tâche de Zend\_View\_Helper

```
<form action="action.php" method="post">
  <p>
    <label>Your Email:</label>
    <?php echo $this->formText('email', 'you@example.com', array('size' => 32)) ?>
  </p>
  <p>
    <label>Your Country:</label>
    <?php echo $this->formSelect('country', 'us', null, $this->countries) ?>
  </p>
  <p>
    <label>Would you like to opt in?</label>
    <?php echo $this->formCheckbox('opt_in', 'yes', null, array('yes', 'no')) ?>
  </p>
</form>
```

---

## Part VII

unserialize() et les entrées utilisateur

# unserialize() et les entrées utilisateur

---

- Ne jamais utiliser unserialize() sur une entrée utilisateur !
  - Les propriétés peuvent être remplies arbitrairement – même celles privées
  - Les méthodes `__destruct()` et `__wakeup()` seront exécutées
  - L'autoloader permet le chargement arbitraire d'objets
- Zend Framework est fourni avec un grand nombre de classes
  - La combinaison de certaines classes permet de détourner le flux de contrôle

# unserialize() – Exemple d' « exploit »

---

Les classes de Zend-Framework permettent :

- L'upload de fichiers arbitraires
- L'exécution de code PHP arbitraire (ZF >= 1.8.0)
- L'envoi de courriel de SPAM (ZF >= 1.8.0)
- L'inclusion de fichiers arbitraires (ZF >= 1.9.0)

# Zend\_Queue\_Adapter\_Activemq

```
class Zend_Queue_Adapter_Activemq extends Zend_Queue_Adapter_AdapterAbstract
{
    // ...

    /**
     * Close the socket explicitly when destructed
     *
     * @return void
     */
    public function __destruct()
    {
        // Gracefully disconnect
        $frame = $this->_client->createFrame();
        $frame->setCommand('DISCONNECT');
        $this->_client->send($frame);
        unset($this->_client);
    }

    // ...
}
```

*Zend\_Queue\_Adapter\_Activemq*  
*\_client*

# Zend\_Queue\_Stomp\_Client\_Connection

```
class Zend_Queue_Stomp_Client_Connection
    implements Zend_Queue_Stomp_Client_ConnectionInterface
{
    // ...

    public function getFrameClass()
    {
        return isset($this->_options['frameClass'])
            ? $this->_options['frameClass']
            : 'Zend_Queue_Stomp_Frame';
    }

    public function createFrame()
    {
        $class = $this->getFrameClass();

        if (!class_exists($class)) {
            require_once 'Zend/Loader.php';
            Zend_Loader::loadClass($class);
        }

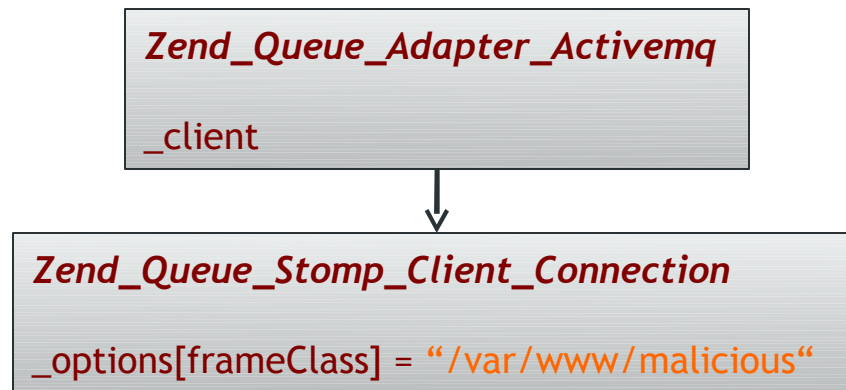
        $frame = new $class();

        // ...
    }
}
```

*Zend\_Queue\_Stomp\_Client\_Connection*  
\_options[frameClass]

# Combined

---



```
O:27:"Zend_Queue_Adapter_Activemq":1:  
{s:36:"\0Zend_Queue_Adapter_Activemq\0_client";O:34:"Zend_Queue_Stomp_Client_Connection":1:{s:11:"\0*\0_options";a:1:  
{s:10:"frameClass";s:18:"/var/www/malicious";}}}
```

# Forum AFUP 2010 !

---

- 9-10 novembre 2010
- <http://www.afup.org/pages/forumphp2010/>
- Quelques conférences sur ZF avec notamment G. Delamarre pour une introduction au ZF et J. Pauli et M. Perraud pour une session sur ZF 2.0

# Merci !