



The PHP Company



ZEND FRAMEWORK

Desarrollo de Aplicaciones Seguras

Karén Nalbandian

knalbandian@alfa9.com

Mira el webinar grabado: <http://bit.ly/vqUYIW>

Versión en español del Webinar realizado por
Stefan Esser de SectionEins GmbH



Acerca de mi

- **Project Manager, Líder tecnológico y Desarrollador Senior en Alfa9 Servicios Web S.L.**
- **12 años de experiencia en diseño y programación de aplicaciones Web en PHP y ASP .NET**
- **Zend Certified PHP5 Engineer**
- **Alfa9 Servicios Web S.L. es Socio de Negocios de Zend Technologies en España.**

INTRODUCCIÓN

Introducción

- **Zend Framework se hace cada vez más y más popular**
- **Crece la demanda de pautas para programar aplicaciones seguras con Zend Framework**
- **Los libros, las charlas y los seminarios se enfocan en seguridad al programar con PHP sin el uso de un Framework**
- **El uso de un Framework requiere pautas diferentes en lo que se refiere al desarrollo seguro**
- **Los Frameworks suelen tener características que ayudan a programar de forma segura**

Tópicos

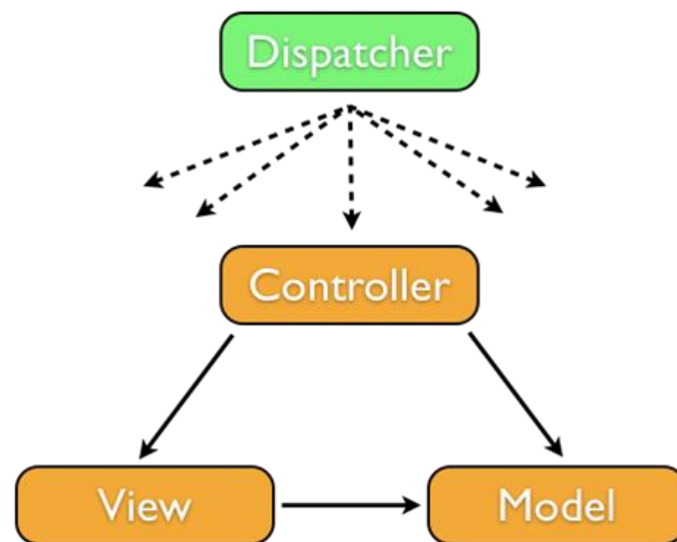
- **Autenticación centralizada**
- **Filtrado y validación centralizada de datos**
- **Seguridad SQL**
- **Protección contra Cross Site Request Forgery (CSRF)**
- **Seguridad en el manejo de sesiones**
- **Protección contra Cross Site Scripting (XSS)**
- **Nuevos ataques usando viejas vulnerabilidades**

Parte I

Autenticación, Filtrado y Validación centralizados

Aplicaciones tradicionales vs. Zend Framework

- Las aplicaciones tradicionales tienen muchos puntos de entrada
- Aplicaciones ZF por lo general usan el patrón MVC
- Aplicaciones ZF permiten centralizar las tareas relativas a la seguridad
 - ▶ Validación y filtrado de datos
 - ▶ Autenticación



Plugins de Front Controller

- **Añaden funcionalidad a `Zend_Controller_Action`**
- **No requieren extender las clases de Controller Action**
- **Son ideales para las tareas de autenticación, validación y filtrado**

```
<?php
$fc = Zend_Controller_Front::getInstance();
$fc->registerPlugin(new App_Controller_Plugin_Auth());
$fc->dispatch();
```

Autenticación centralizada

```
<?php
class App_Controller_Plugin_Auth extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        try {
            App_Auth::isLoggedIn();
        } catch (App_Auth_UserNotLoggedInException $e) {
            if (! in_array($request->getControllerName(),
                array('index', 'error', 'login')
            ) {
                $request->setModuleName('default')
                    ->setControllerName('login')
                    ->setActionName('index')
                    ->setDispatched(false);

                return;
            }
        }
    }
}
```

Filtrado/Validación centralizados (I)

```
<?php
$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(),
    ),
);
```

Filtrado/Validación centralizados (II)

```
<?php
class App_Controller_Plugin_Filter 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('illegal-param')
                ->setDispatched(false);

            return;
        }
    }
}
```

Integración centralizada de PHPIDS

```
<?php
class App_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);
        }
    }
}
```

Parte II

Filtrado y Validación de formularios

Filtrado/Validación en formularios (I)

- Los formularios de Zend Framework usan filtros y validadores de forma automática
- Se pueden añadir validadores a las instancias de `Zend_Form_Element`
- Los Validators se pueden encadenar de forma arbitraria

Filtrado/Validación en formularios (II)

```
<?php
// Se crea el elemento Title
$name = $form->createElement('text', 'title',
    array('size' => 40, 'maxlength' => 40));
$name->addValidator('Alpha')
    ->addValidator('StringLength', false, array(1, 40))
    ->setLabel('Nombre')
    ->setRequired(true);

// Se crea el elemento Message
$message = $form->createElement('textarea', 'message',
    array('rows' => 6, 'cols' => 40));
$message->setLabel('Mensaje')
    ->setRequired(true)
    ->addFilter('StripTags');

// Se crea el botón Submit
$submit = $form->createElement('submit', 'send');
$submit->setLabel('Enviar');

// Se añaden los elementos al formulario
$form->addElement($name)->addElement($message)->addElement($submit);
```

Filtrado/Validación en formularios (III)

- Validación del formulario en un Action

```
<?php
/* . . . más código . . . */
// Se validan los datos
if (! $form->isValid($this->getRequest()->getPost())) {
    // Se establecen los valores originales, no filtrados
    $form->setDefaults($this->getRequest()->getPost());
    // Se asignan las variables al View
    $this->view->form = $form;
    $this->view->title = "Form 1";

    // Se detiene el procesamiento
    return $this->render('form');
}
/* . . . más código . . . */
```

Parte III

Seguridad SQL

Inyección SQL en aplicaciones Zend Framework

- ZF esta equipado con varias clases para acceder a la BBDD
- Los métodos, por lo general, soportan y recomiendan sentencias preparadas (Prepared Statements)
- ZF también provee métodos para escapar los parámetros de las consultas SQL dinámicas
- La falta de estas herramientas facilita inyección SQL
- El mal uso de Prepared Statements puede crear agujeros de seguridad

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

Inyección SQL + PDO_MySQL = Peligro

- Tradicionalmente, MySQL permite sólo una query por vez
 - ▶ ext/mysql - no soporta queries multiples
 - ▶ ext/mysqli - tiene una función específica: `mysqli_multi_query()`
- **ATENCIÓN: PDO_MySQL no tiene esta limitación**
- La inyección SQL que se produce en una aplicación ZF que utiliza PDO_MySQL es más peligrosa que en una aplicación que usa acceso al servidor MySQL de forma tradicional

Escapado en Zend Framework

- **Método quote(\$value, \$type = null)**
 - ▶ Escapado correcto siempre - un método en vez de múltiples
 - ▶ **ATENCIÓN:** las cadenas se acotan mediante comillas simples

- **Método quoteIdentifier(\$ident, \$auto = false)**
 - ▶ Está hecho para escapar identificadores (tablas, campos, etc.)
 - ▶ Las aplicaciones PHP tradicionales deben implementar la funcionalidad

Zend_Db_Select

- Se usa para crear sentencias SQL de forma dinámica
- Internamente utiliza Prepared Statements cuando es posible
- Pueden ocurrir inyecciones SQL si se usa mal
 - ▶ ATENCIÓN: Especialmente en WHERE y ORDER BY

```
<?php
// Genera la siguiente query:
// SELECT `product_id`, `product_name`, `price` FROM `products`
// WHERE (`price` < 100.00 OR `price` > 500.00) AND (`product_name` = 'Apple')
$minimumPrice = $_GET['min_price']; // 100
$maximumPrice = $_GET['max_price']; // 500
$prod = $_GET['product'];           // 'Apple'

$select = $db->select()
    ->from('products', array('product_id', 'product_name', 'price'))
    ->where("`price` < {$minimumPrice} OR `price` > {$maximumPrice}")
    ->where("`product_name` = ?", $prod);
```

Parte IV

Protección contra Cross Site Request Forgery (CSRF)

Protección contra Cross Site Request Forgery (CSRF)

- La protección se suele basar en un token secreto que depende de la sesión
- **Zend Framework dispone de `Zend_Form_Element_Hash` que es un token de este tipo y que encapsula toda la validación necesaria**
- **El formulario se puede proteger contra CSRF añadiéndole este elemento**

```
<?php
$form->addElement('hash', 'csrf_token',
    array('salt' => 's3cr3ts4ltG%Ek@on9!'));
```

Protección contra CSRF automatizada

- La protección se debe realizar de forma manual
- Se puede automatizar la protección extendiendo Zend_Form

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

Parte V

Seguridad en el manejo de Sesiones

Seguridad en el manejo de Sesiones

- La configuración tiene gran impacto en la seguridad
- Las conexiones SSL deben ser aseguradas usando el flag *secure*
- Nombre / Ruta de sesión personalizadas para cada aplicación
- Dificultar ataques XSS mediante el flag *httpOnly*
- Establecer tiempo de vida máximo de sesión

```
<?php
Zend_Session::setOptions(
    'cookie_secure'    => true,           // En el caso de servidor SSL
    'name'             => 'mySSL',       // Nombre personalizado
    'save_path'        => 'sessions/mySSL', // Ruta personalizada
    'cookie_httponly' => true,           // Contra ataques XSS
    'gc_maxlifetime'  => 15 * 60,       // Tiempo de vida
);
```

Fijación y Robo de Sesiones

- **Fijación de Sesión (Session Fixation)**
 - ▶ Se puede lograr una ligera mejora validando la sesión
 - ▶ Se detiene al regenerar el Session ID en cada cambio de estado
 - `session_regenerate_id(true);`
 - ▶ Es mejor implementarlo en el módulo de login/logout
- **Robo de Sesión (Session Hijacking)**
 - ▶ Sólo se puede detener completamente usando SSL en toda la aplicación (previene el code sniffing)
 - ▶ El flag `httpOnly` protege contra robo de Session ID mediante XSS
 - ▶ Es limitado el uso de la validación de sesión

Validación de Sesión (I)

- Reconocimiento de sesión usando información adicional
- Se recomienda a menudo para evitar fijación y robo de sesiones pero tiene utilidad limitada
- Zend Framework soporta validadores de sesión
 - ▶ `Zend_Session_Validator_HttpUserAgent`

```
<?php
try {
    Zend_Session::start();
} catch (Zend_Session_Exception $e) {
    // Zend_Session::regenerate_id() no funciona en todas las versiones
    session_regenerate_id();
}

Zend_Session::registerValidator(new Zend_Session_Validator_HttpUserAgent());
```

Validación de Sesión (II)

- Cuidado al comprobar información adicional
- La comprobación de la cabecera HTTP “User-Agent” no es fiable, al menos a partir de IE8
- La comprobación de la cabecera HTTP “Accept” ya presentaba problemas incluso en versiones anteriores de IE
- La comprobación de IP es imposible para los usuarios de grandes compañías, ISPs y proxies.
 - ▶ Limitar la comprobación a las redes de clase C / B / A
 - ▶ Mejor compatibilidad con sitios SSL

Validando la IP del Usuario

```
<?php
class App_Session_Validator_RemoteAddress extends Zend_Session_Validator_Abstract
{
    /**
     * Obtiene la IP del Usuario y la guarda para la comparación posterior
     */
    public function setup()
    {
        $this->setValidData((isset($_SERVER['REMOTE_ADDR'])
            ? $_SERVER['REMOTE_ADDR'] : null));
    }

    /**
     * Obtiene la IP y la compara con la almacenada anteriormente
     *
     * @return boolean
     */
    public function validate()
    {
        return (isset($_SERVER['REMOTE_ADDR'])
            ? $_SERVER['REMOTE_ADDR'] : null) === $this->getValidData();
    }
}
```

Parte VI

Protección contra Cross Site Scripting (XSS)

XSS en Aplicaciones usando Zend Framework

- Symfony soporta escapado automático de salida
- Zend Framework no lo soporta (previsto en ZF2)
- Prevenir XSS es la tarea del programador
- Las vulnerabilidades XSS ocurren en la vista

```
<!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>
```

Protección contra XSS (I)

- Tradicionalmente existen dos alternativas
 - ▶ Imprimir únicamente valores “limpios”

```
<!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 $this->prepareUrl($this->link); ?>">Link 1</a></li>
  </ul>
</body>
</html>
```

Protección contra XSS (II)

- Tradicionalmente existen dos alternativas
 - ▶ Asignar únicamente valores “limpios”

```
<?php
$filter = new Zend_Filter_HtmlEntities();
$urlFilter = new My_Filter_Url();

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

El uso de Zend_View_Helper

- Las soluciones tradicionales predisponen a errores
- Es complicado realizar el escaneo automático de salida para aplicar filtros
- No se deben imprimir las variables de forma directa
- La salida se deberá realizar únicamente mediante `Zend_View_Helper`
- La prevención de XSS se convierte en tarea de `Zend_View_Helper`

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

Parte VII

unserialize() y entradas de Usuarios

unserialize() y la entrada del Usuario

- **Nunca se debe usar unserialize() en las entradas de Usuario**
 - ▶ Las propiedades, incluso las privadas, pueden contener datos arbitrarios
 - ▶ Se ejecutarán los métodos `__destruct()` y `__wakeup()`
 - ▶ El Autoloader permite cargar objetos arbitrarios
- **Zend Framework está compuesto por muchas clases**
 - ▶ La combinación de clases puede permitir romper el flujo de control

unserialize() - Ejemplo de Exploit

- **Las clases de Zend Framework permiten:**
 - ▶ Subir archivos arbitrarios
 - ▶ Ejecutar código PHP (ZF >= 1.8.0)
 - ▶ Enviar mensajes SPAM (ZF >= 1.8.0)
 - ▶ Incluir archivos arbitrarios (ZF >= 1.9.0)

Zend_Queue_Adapter_Activemq

```
<?php
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

```
<?php
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]***

La combinación fatal

Zend_Queue_Adapter_Activemq_client



Zend_Queue_Stomp_Client_Connection_options[frameClass] = '/var/www/codigo_malo'



```
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/codigo_malo";}}}
```



¡Te agradecemos la participacion! ¿Preguntas?

Karén Nalbandian
knalbandian@alfa9.com

Mira el webinar grabado: <http://bit.ly/vqUYIW>

Si quieres contactar **Zend Technologies** o enviarnos tus comentarios:

España: Ana Maria Valarezo - ana.m@zend.com

América del Sur: Rick Gonwa - rick.gonwa@zend.com