Padrões, PEAR e Frameworks PHP

Professor: FLÁVIO GOMES DA SILVA LISBOA (FGSL)

AULA 3 – Componentes de Segurança

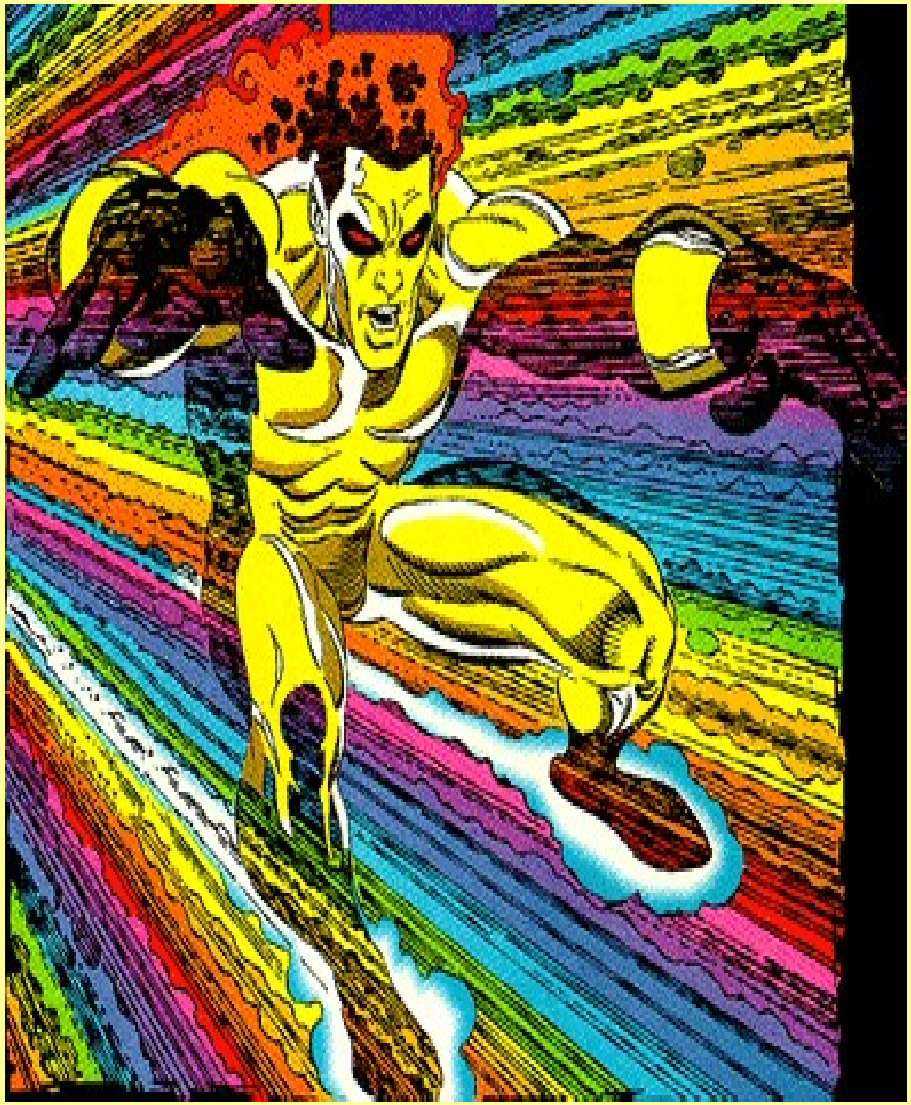
Utilização do framework para implementar técnicas de codificação segura

Plano de Aulas

Dia	Conteúdo
1	Motivação para o uso de frameworks. Instalação e uso do Eclipse com plugin PDT. Padrão de Projeto MVC. Apresentação do Zend Framework. Projeto Mínimo. Padrões de Projeto Singleton, Controller Front e Controller Page. Controle de Erros.
2	Componente de acesso ao banco. Mapeamento Objeto-Relacional. Abstração da camada do banco X Uso de funções específicas. Encapsulamento da sessão como objeto. Padrões de Projeto Factory, Gateway, Iterator e Active Record.
3	Implementação de código seguro com componentes do framework. Filtros e Validadores. Listas de Controle de Acesso. Autenticação. Segurança no acesso ao banco de dados.
4	Separação da aplicação em módulos. Uso de templates e subtemplates de página. Criação de formulários dinâmicos.
5	Encapsulamento de componentes de terceiros (PEAR, Smarty). Criação de novos componentes.

Pausa para Revisão: Padrões de Projeto

Este é o momento de certificar-se de que o conhecimento foi assimilado. Não espere a matéria terminar, porque não temos nenhum DeLorean com capacitor de fluxo para que você volte no tempo e veja a aula novamente.



Pausa para Revisão: Padrões de Projeto

Escreva de forma resumida para que servem (não o que são) os padrões de projeto vistos até agora:

Singleton;

Front Controller;

Page Controller;

MVC;

Pausa para Revisão: Padrões de Projeto

Factory;

Gateway;

Iterator;

Active Record.

Sábado

Em respeito ao sétimo dia da semana, hoje descansaremos dos Padrões de Projeto.

Descanse em paz... com segurança!

Padrões, PEAR e Frameworks PHP

SUA APLICAÇÃO ESTÁ SEGURA?

Segurança: Questões

A segurança é opcional?

Podemos ter segurança total?

Soluções de segurança têm efeito permanente?

Segurança custa pouco?

Segurança em Camadas

Princípio NOT Goonies

"What's good enough for you is good enough for me?"

Se a segurança importa, a resposta é

NO!

Segurança em Camadas: Certezas

A segurança nunca será suficiente.

A segurança custa caro.

A segurança não garante nenhum retorno, mas sua falta pode causar prejuízos.

Sistemas mais seguros = menos amigáveis (e às vezes menos performáticos)

Código Seguro

Código Seguro não é um código de segurança ou um código que implementa os recursos de segurança.

Código Seguro é um código projetado para suportar ataques por invasores mal-intencionados.

Um *Código Seguro* também é um código robusto.

Código Seguro: Resistências Encontradas

A segurança é entediante.

A segurança costuma ser vista como um desativador de funcionalidades, como algo que atrapalha.

A segurança é difícil de medir.

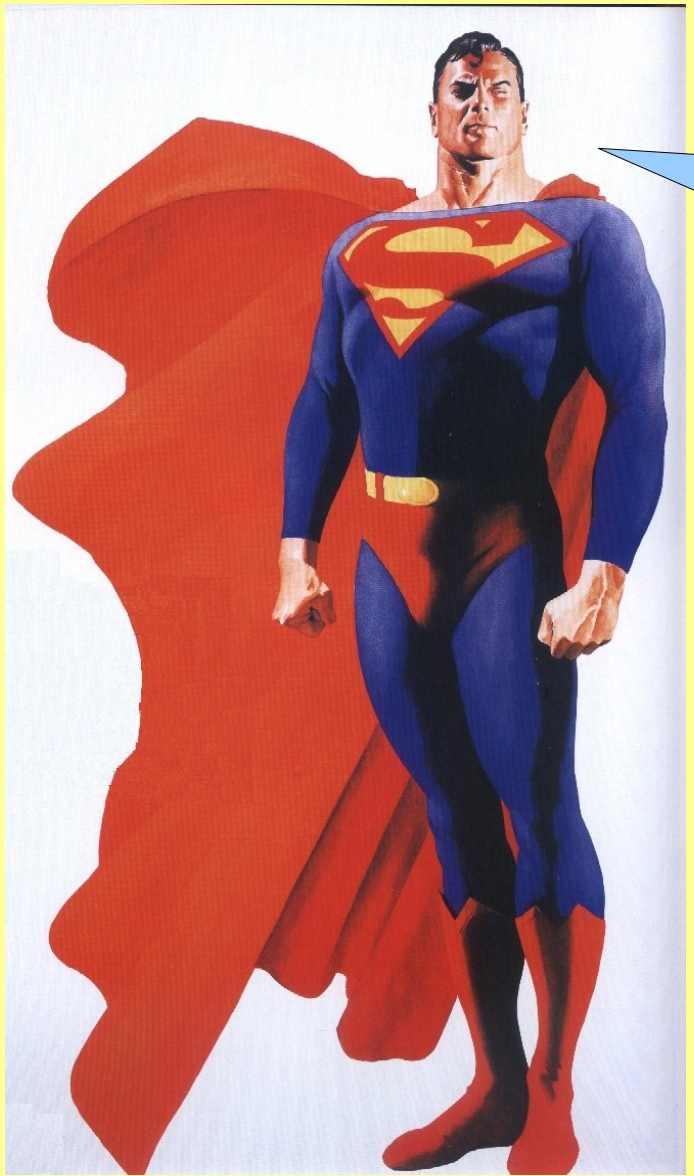
Normalmente, a segurança não é principal habilidade ou interesse dos projetistas e desenvolvedores que criam o produto.

A segurança não significa criar algo novo e animador.

Código Seguro: Necessidade Constante

Como seria maravilhoso se houvesse uma infra-estrutura de software que provesse funcionalidades mínimas que auxiliassem a implementação de código seguro...

Padrões, PEAR e Frameworks PHP



ISSO É UM TRABALHO PARA UM FRAMEWORK!

ESPERE! AINDA NÃO ESTÁ CONVENCIDO DA IMPORTÂNCIA DO CÓDIGO SEGURO?

Mitos da Construção de Programas

- Ninguém fará isso!
- Por que alguém faria isso?
- Nunca fomos atacados.
- Estamos seguros – utilizamos criptografia.
- Revisamos o código e não há bugs de segurança.

Mitos da Construção de Programas

- Sabemos que é o *default*, mas o administrador pode desativá-lo.
- Se não executarmos como administrador, as coisas quebram.
- Mas não cumpriremos os prazos!
- Não é explorável.

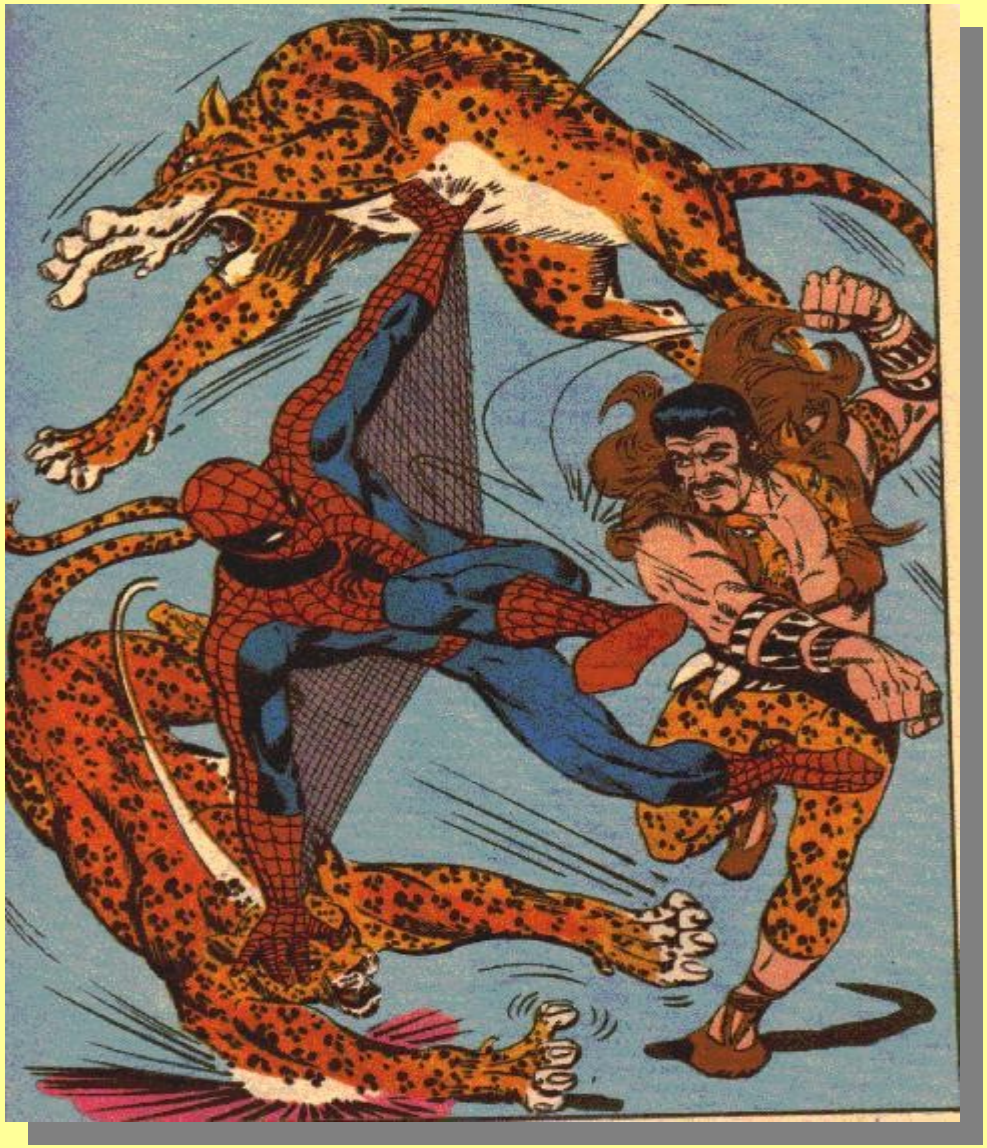
Mitos da Construção de Programas

►Mas essa é a maneira como sempre fizemos isso.

►Se pelo menos tivéssemos ferramentas melhores...

A Vantagem do Invasor e o Dilema do Defensor

1º O defensor deve defender todos os pontos; o invasor pode escolher o ponto mais fraco.



A Vantagem do Invasor e o Dilema do Defensor

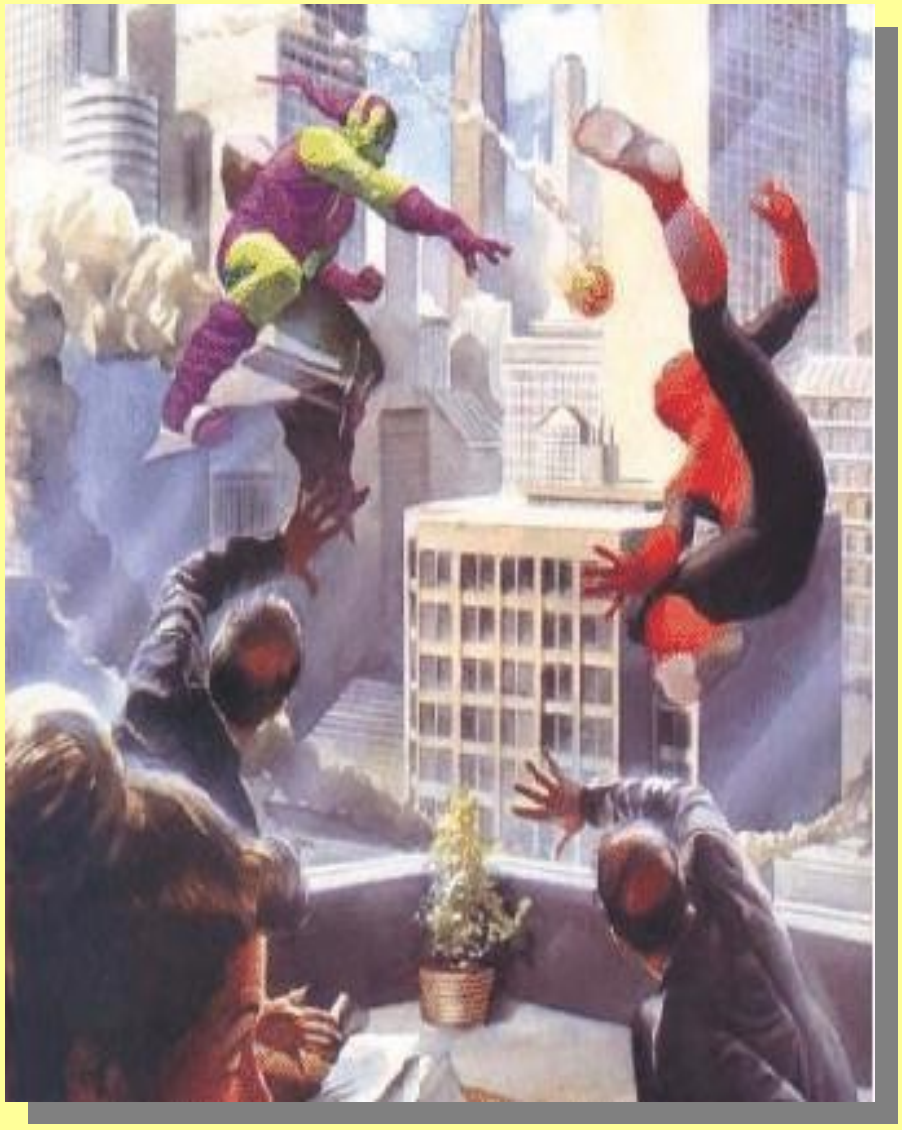
2º O defensor pode se defender somente de ataques conhecidos; o invasor pode investigar vulnerabilidades desconhecidas.

A Vantagem do Invasor e o Dilema do Defensor

3º O defensor deve estar constantemente vigilante; o invasor pode atacar a qualquer momento.

A Vantagem do Invasor e o Dilema do Defensor

4º O defensor deve jogar de acordo com as regras; o invasor pode jogar sujo.



PHP: Poder e Responsabilidade

"Ben Parker avisou uma vez seu jovem sobrinho Peter Peter, cujo alter-ego super-herói é o Homem-Aranha, que 'com um grande poder, vem uma grande responsabilidade'.

PHP: Poder e Responsabilidade

Assim é com a segurança em aplicações PHP. O PHP fornece um rico conjunto de ferramentas com imenso poder – alguns têm argumentado que talvez seja muito poder – e este poder, quando usado com cuidadosa atenção aos detalhes, permite a criação de aplicações complexas e robustas.

PHP: Poder e Responsabilidade

Por outro lado, sem essa atenção para os detalhes, usuários maliciosos podem usar o poder do PHP para seus próprios interesses, atacando aplicações de várias formas.”

Fonte: Zend PHP 5 Certification Study Guide

Falha de Segurança: Mito do PHP

A maior fraqueza na maioria dos programas PHP não é inerente a linguagem em si, mas meramente um problema de código escrito desconsiderando a segurança.

Falha de Segurança: Mito do PHP

Por essa razão, você sempre deve investir um pouco de tempo considerando as implicações de um certo pedaço de código, para ter certeza do dano possível se uma variável não esperada for submetida ao mesmo.

Fonte: Manual do PHP – www.php.net

Toda Entrada está Doente

►Se o dado se origina de uma fonte externa, ele não pode ser confiável.

►Não temos certeza de os dados contém caracteres que podem ser executados no contexto errado.

►Dados de todas as matrizes superglobais, exceto $_SESSION devem ser considerados doentes.

Toda Entrada está Doente

CONCLUSÃO: Todo dado deve ser filtrado.

Fonte: Zend PHP 5 Certification Study Guide

Toda Entrada está Doente

Como seria maravilhoso se houvesse um componente que ajudasse a filtrar os dados de entrada...

Toda Entrada está Doente: Injeção de XML

```
/*
 * Sem filtro
 * A diretiva magic_quotes_gpc insere barras
 * à esquerda de aspas, evitando a interpretação
 * de atributos HTML.
 * Mas não impede o efeito do restante.
 * ini_set() não tem efeito, pois os dados são
 * passados ANTES. A configuração já tem de estar
 * no arquivo php.ini, ou ser modificada pelo
 * .htaccess do Apache Web Server.
*/
foreach ($_GET as $parametro => $valor)
{
   echo "$parametro = $valor";
}
```

Toda Entrada está Doente: Injeção de XML

```
require('Zend/Filter/HtmlEntities.php');

// Com filtro
$entrada = new Zend_Filter_HtmlEntities();

foreach ($_GET as $parametro => $valor)
{
   echo "$parametro = {$entrada-
>filter($valor)}";
}
```

Toda Entrada está Doente: Zend_Filter

Zend_Filter_Alnum
Zend_Filter_Alpha
Zend_Filter_BaseName
Zend_Filter_Digits
Zend_Filter_Dir
Zend_Filter_HtmlEntities
Zend_Filter_Int
Zend_Filter_RealPath
Zend_Filter_StripNewLines
Zend_Filter_StringToLower
Zend_Filter_StringToUpper
Zend_Filter_StringTrim
Zend_Filter_StripTags

Toda Entrada está Doente: Zend_Filter

Mas e se eu precisar de mais de um filtro, ao mesmo tempo?

Toda Entrada está Doente: Zend_Filter

Se você pensou que ia escapar dos Padrões de Projeto hoje, se enganou completamente.

Vamos usar o padrão conhecido como **Decorator**, que consiste em acrescentar funcionalidades à uma classe mãe pela passagens de classes filhas como parâmetros.

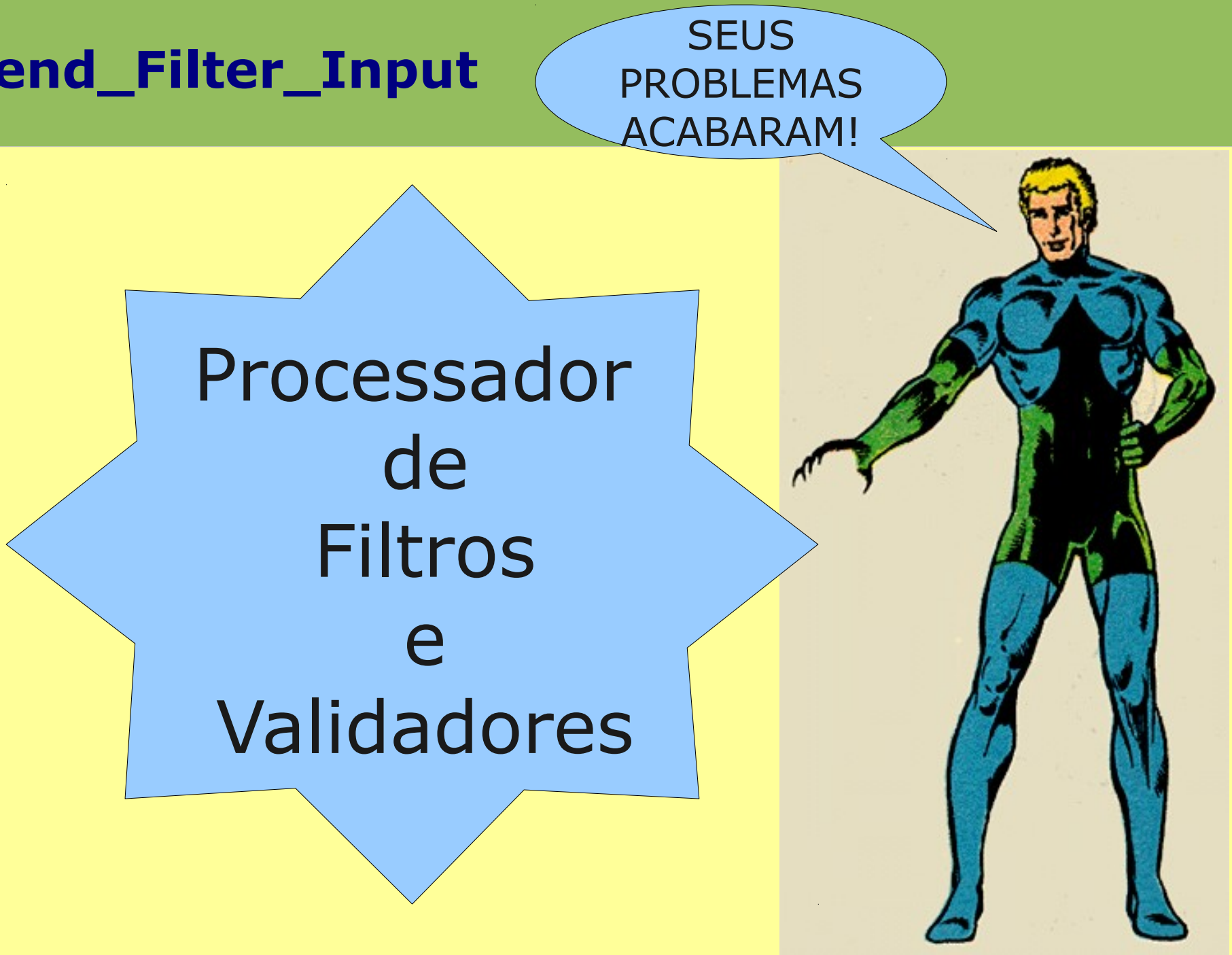
Toda Entrada está Doente: Zend_Filter

```
require('Zend/Filter.php');
require('Zend/Filter/StringToLower.php');
require('Zend/Filter/Word/CamelCaseToDash.php');

// Combinação de Zend_Filter_Word_CamelCaseToDash
// e Zend_Filter_StringToLower
$filtro = new Zend_Filter();
// Interface fluente
$filtro->addFilter(new Zend_Filter_Word_CamelCaseToDash())
       ->addFilter(new Zend_Filter_StringToLower());

foreach ($_GET as $parametro => $valor)
{
    echo "Com Filtro:";
    echo "$parametro = {$filtro->filter($valor)}";
    echo "<br>";
    echo "Sem Filtro:";
    echo "$parametro = $valor";
}
```

Toda Entrada está Doente: Zend_Filter

Como seria maravilhoso se a gente pudesse aplicar vários filtros e regras de validação a um conjunto de dados de uma vez...

Zend_Filter_Input



Filtros X Regras de Validação

Filtros transformam os dados, deixando passar apenas o que interessa.

Regras de validação não transformam os dados; apenas verificam se eles se atendem a um critério.

Se isso não ficar claro, ficará confuso compreender por que um dado com uma regra de validação Digits só será válido se passar pelo filtro Digits.

Filtros X Regras de Validação

```
require('Zend/Filter/Input.php');

$filtros = array(
        'dia'    => Digits,
        'mes'    => Digits,
        'ano'    => Digits,
        'nome'   => HtmlEntities
);

$validadores = array(
        'dia'    => Digits,
        'mes'    => Digits,
        'ano'    => Digits,
        'nome' => Alpha
);
```

Filtros X Regras de Validação

```
$dados = new Zend_Filter_Input($filtros,$validadores,
$_GET);

echo 'Com filtro:<br>';
echo "dia = {$dados->dia}<br>";
echo "mes = {$dados->mes}<br>";
echo "ano = {$dados->ano}<br>";
echo "nome = {$dados->nome}<br>";

echo 'Sem filtro:<br>';
foreach ($_GET as $nome => $dado)
{
   echo "$nome = $dado<br>";
}
```

Filtros e Regras de Validação com Parâmetros

```
$regraAno = array(
      Digits,
      array('Between',1,12)
);

$validadores = array(
      'dia'   => Digits,
      'mes'   => Digits,
      'ano'   => $regraAno,
      'nome' => Alpha
);
```

Filtros e Regras de Validação Genéricas

```
$filtros = array(
        '*'   => Digits,
);


$validadores = array(
        '*'   => Digits,
);
```

Metacomandos para Filtros e Regras de Validação

```
$validadores = array(
        'nome' => array(
                    'Alpha',
                    'presence' => required'),
        'senha' => array(
                     'StringEquals',
                     'default' => 12345,
                     'fields'  => array(
                                     'senha1',
                                     'senha2'))
);
```

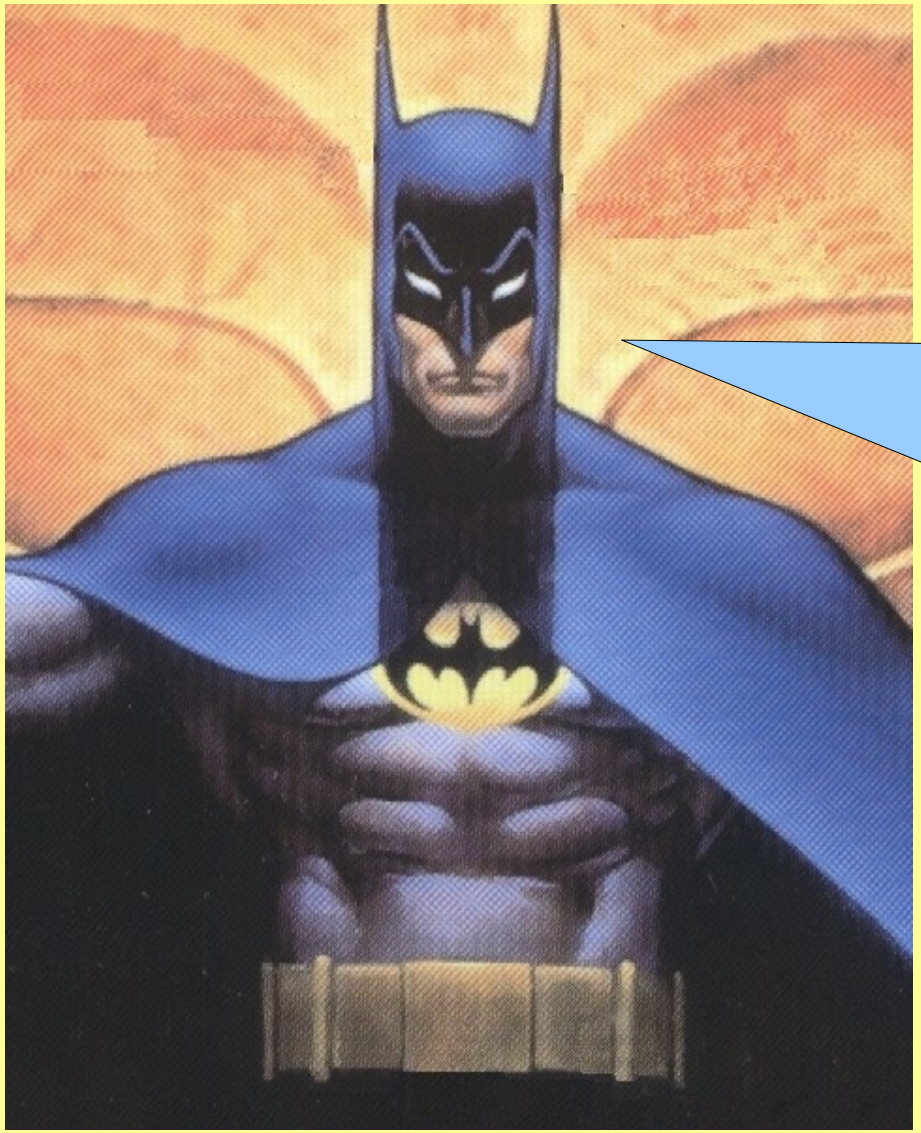
Metacomandos para Filtros e Regras de Validação

```
$validadores = array(
        'endereco' => array(
                    'Alnum',
                    'allowEmpty' => true),
        'senha' => array(
                     'StringEquals',
                     'fields'  => array(
                                    'senha1',
                                    'senha2'),
                     'messages' => 'As senhas
devem ser iguais!')
);
```

Métodos de Verificação e Tratamento de Saída

```
IsValid() ou isValid($campo)

hasValid()
hasMissing()
hasUnknown()

getMessages() = getInvalid() + getMissing()
getUnknown()

getEscaped()
getUnescaped()
```

Listas de Controle de Acesso



Listas de Controle de Acesso

Uma lista de controle de acesso define **papéis**, **recursos** e as **permissões** dos papéis para os recursos.

Em suma, **quem** pode fazer **o quê**.

Papel não é **grupo** nem **usuário**. Esses dois podem exercer um ou mais papéis e um papel pode ser exercido por um ou mais usuários e grupos.

O ZF **não** implementa usuário e grupos.

Zend_Acl

No Zend Framework:

Um **recurso** é um objeto cujo acesso é controlado;

Um **papel** é um objeto que pode requisitar acesso a um recurso.

Colocando de forma simples, papéis requisitam acesso a recursos, ou, de outra forma, papéis requisitam **autorização** a recursos.

Zend_Acl: Criando um Recurso

```
require('Zend/Acl/Resource.php');

$recurso = new
Zend_Acl_Resource('impressao');
```

Zend_Acl: Criando um Recurso

```
require('Zend/Acl/Resource.php');

class Impressao implements
Zend_Acl_Resource_Interface
{
  private $_id;
  public function getResourceId()
  {
    return $this->_id;
  }
}
```

Zend_Acl: Criando um Papel

```
require('Zend/Acl/Role.php');

$papel = new Zend_Acl_Role('administrador');
```

Zend_Acl: Definindo Papéis Básicos

```
require('Zend/Acl.php');
require('Zend/Acl/Role.php');

$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('administrador'))
   ->addRole(new Zend_Acl_Role('membro'))
   ->addRole(new Zend_Acl_Role('convidado'));
```

Pode parecer, mas não é o padrão Decorator!

Zend_Acl: Herdando Papéis

```
require('Zend/Acl.php');
require('Zend/Acl/Role.php');

$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('administrador'))
   ->addRole(new Zend_Acl_Role('membro'))
   ->addRole(new Zend_Acl_Role('convidado'));

$pais =
array('administrador','membro','convidado');

$acl->addRole(new Zend_Acl_Role('superusuario'),
$pais);
```

Zend_Acl: Definindo Recursos

```
require('Zend/Acl.php');
require('Zend/Acl/Resource.php');

$acl = new Zend_Acl();

$recurso = new Zend_Acl_Resource('cadastro');

$acl->add($recurso);
```

Zend_Acl: Dando e Negando Acesso a Recursos

```
require('Zend/Acl.php');
require('Zend/Acl/Resource.php');
require('Zend/Acl/Role.php');

$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('administrador'))
   ->addRole(new Zend_Acl_Role('membro'))
   ->addRole(new Zend_Acl_Role('convidado'));

$acl->add(new Zend_Acl_Resource('cadastro'));

$acl->allow('administrador','cadastro');
$acl->deny('convidado','cadastro');
```

Zend_Acl: Verificando o Acesso a Recursos

```
$acl->add(new Zend_Acl_Resource('cadastro'));

$acl->allow('administrador','cadastro');
$acl->deny('convidado','cadastro');

echo 'Acesso '.($acl-
>isAllowed('administrador','cadastro') ?
'permitido' : 'negado');
echo 'Acesso '.($acl-
>isAllowed('convidado','cadastro') ? 'permitido' :
'negado');
echo 'Acesso '.($acl-
>isAllowed('membro','cadastro') ? 'permitido' :
'negado');
```

Se nada for definido, o acesso é negado, por padrão

Zend_Acl: Permissões e Privilégios

Existem duas possibilidades com relação a permissão. Ela pode ser dada para o recurso como um todo ou para um conjunto de privilégios que o papel tem sobre o recurso.

Quando não definimos privilégios, estamos implicitamente usando um único privilégio, que é o mero acesso ao recurso.

Zend_Acl: Dando e Negando Acesso a Privilégios

```
require('Zend/Acl.php');
require('Zend/Acl/Resource.php');
require('Zend/Acl/Role.php');

$acl = new Zend_Acl();

$acl->addRole(new Zend_Acl_Role('membro'));

$acl->add(new Zend_Acl_Resource('cadastro'));

$acl->allow('membro','cadastro',array('editar','ver'));
$acl->deny('membro','cadastro',array('incluir','excluir'));

echo 'Acesso '.($acl-
>isAllowed('membro','cadastro','ver') ? 'permitido' :
'negado');
echo 'Acesso '.($acl-
>isAllowed('membro','cadastro','excluir') ? 'permitido' :
'negado');
```

Listas Branca X Lista Negra

►**Lista Negra**
→ Menos restritiva.
→ Há um conjunto específico de palavras que são consideradas inapropriadas.
→ Qualquer palavra que não conste da lista é permitida.
→ Listas negras devem ser modificadas continuamente, e expandidas quando novos vetores de ataque tornam-se evidentes.

►**Lista Branca**
→ Mais restritiva.
→ Identifica somente os dados que são aceitáveis.
→ Mantém controle sobre os parâmetros que mudam e não os deixa aos caprichos de pretensos atacantes.

CONCLUSÃO: Listas brancas oferecem mais proteção contra ataque do que listas negras.

Autenticação

Autenticar significa verificar se alguém é quem diz ser baseado em uma série de credenciais.

Autenticação é diferente de **autorização**. Autorização é o processo de decidir se uma entidade pode acessar ou executar operações sobre outras entidades. É implementada com Zend_Acl

Autenticação: Zend_Auth

O componente Zend_Auth usa o padrão de projeto Adapter para autenticar contra um tipo particular de serviço de autenticação, tal como:

- LDAP
- SGBD
- Armazenamento baseado em arquivos

Autenticação: Zend_Auth

O trecho de código a seguir
Um exemplo de uso de Zend_Auth com as classes Zend_Db_Table, Zend_Session e Zend_Registry.

Zend_Auth_Adapter_DbTable

```
$registry = Zend_Registry::getInstance();
try
{
   $post = $registry['post'];
   $matricula = (int)$post->cpf;
   $senha = $post->senha;
   $usuarios = new Usuarios();

   $authAdapter = new
Zend_Auth_Adapter_DbTable($usuarios-
>getAdapter());
```

Zend_Auth_Adapter_DbTable

```
  $authAdapter->setTableName('usuarios')
->setIdentityColumn('cpf')
->setCredentialColumn('senha');
  $authAdapter->setIdentity($cpf);
  $authAdapter-
>setCredential(Usuarios::criptografar($senh
a));
```

Zend_Auth_Adapter_DbTable

```
  $resultado = $authAdapter-
>authenticate();
  if ($resultado->isValid())
  {
    $registry['session']->dataAuth =
$authAdapter-
>getResultRowObject();
  Zend_Registry::set('session',
$registry['session']);
  $this->_redirect('/index/menu');
}
```

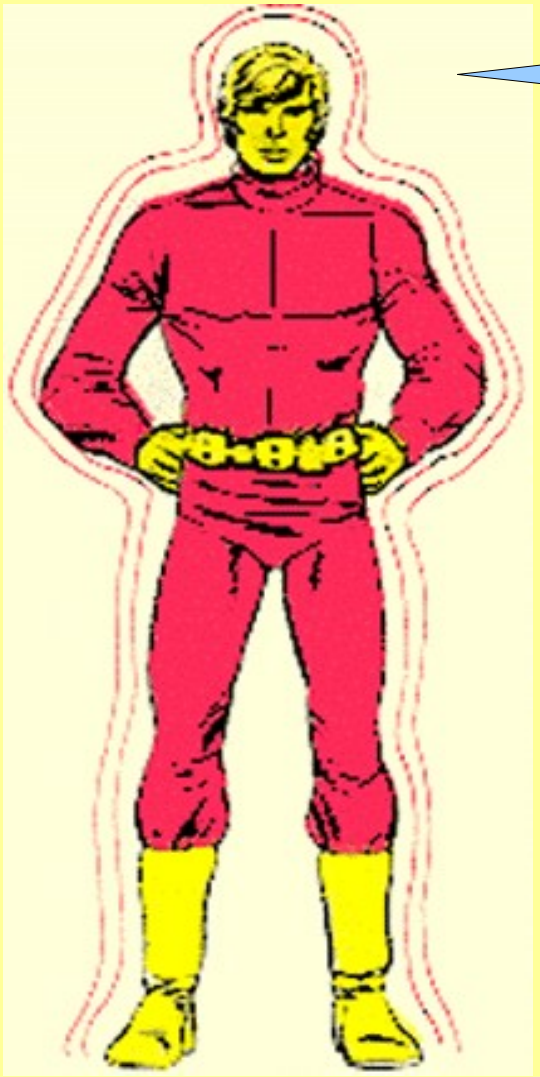
Zend_Auth_Adapter_DbTable

```
  else
  {
    $mensagens = '';
    foreach ($resultado->getMessages() as
$mensagem)
    {
      $mensagens .= $mensagem;
    }
    $registry['session']->mensagem =
$mensagens;
    Zend_Registry::set('session',
$registry['session']);
```

Zend_Auth_Adapter_DbTable

```
    $this->_redirect('/index/index');
  }
}
catch (Exception $e)
{
  $registry['session']->mensagem = $e-
>getMessage();
  Zend_Registry::set('session',
$registry['session']);
  $this->_redirect('/index/index');
}
```

Padrões, PEAR e Frameworks PHP



NOSSA, ENTENDI TUDO!

Zend_Auth_Adapter_DbTable: Passo a Passo

1) Criar a instância de Zend_Auth_Adapter_DbTable;

2) Configurar a tabela onde estão os dados de autenticação com o método *setTableName()*;

3) Configurar o campo da tabela que contém a identidade do usuário, ou o nome de usuário (DE, não DO) com o método *setIdentityColumn()*;

4) Configurar o campo da tabela que contém a senha do usuário com o método *setCredentialColumn()*;

5) Configurar o valor que será confrontado com o campo nome de usuário, pelo método *setIdentity()*;

Zend_Auth_Adapter_DbTable: Passo a Passo

6) Configurar o valor que será confrontado com o campo senha, pelo método *setCredential()*;

7) Efetuar a autenticação pelo método *authenticate();*

8) O sucesso ou não da autenticação pode ser verificado pelo método *isValid();*

9) O objeto Zend_Db_Adapter, em caso de sucesso, pode retornar um objeto Zend_Db_Row com os dados do usuário autenticado, pelo método *getResultRowObject();*

10) Em caso de fracasso, as falhas encontradas são reportadas pelo método *getMessages()*, que retorna uma matriz de texto;

Exercício: Criar uma Aplicação e Juntar Tudo

Como forma de rever os conceitos outrora apresentados, vamos fazer uma aplicação MVC que apresente uma tela de login e faça autenticação dos dados contra uma tabela do banco de dados usando Zend_Auth.

Uma vez autenticado, o usuário será direcionado para uma tela de menu com algumas opções controladas por Zend_Acl.

DÁ PRA ENCARAR?

Segurança no Banco de Dados

Um dos problemas de segurança envolvendo banco de dados é a injeção de SQL através da entrada de dados da aplicação.

O componente Zend_Db e suas extensões oferecem métodos que permitem barrar tentativas de adulterar declarações SQL.

É um ótimo momento para rever Zend_Db e sua turma, não acha? Aqueles métodos com a palavra *quote*...

Faltou alguma coisa?

Faltou MUITA COISA! Mas haverá uma matéria falando somente sobre segurança. Este foi apenas um tópico dentro do tema *frameworks*.

Quando você ver segurança em um contexto mais amplo, certamente perguntará COMO resolver tantos problemas. Aí ficará evidente a necessidade de *frameworks*... **de segurança**!

Mas veremos ainda mais um tópico de segurança na próxima aula, que versará sobre **apresentação**, que envolve **tratamento de saída** de dados.

Referências Bibliográficas

Galvão, E. A. *PHP & Segurança: Uma União Possível*. Disponível em <http://www.galvao.eti.br> em 02/05/2007.

Howard, M. E Leblanc, D. *Escrevendo Código Seguro.* 2. ed. Porto Alegre. Bookman, 2006.

Pessoa, M. *Segurança em PHP.* São Paulo. Novatec, 2007.

http://framework.zend.com