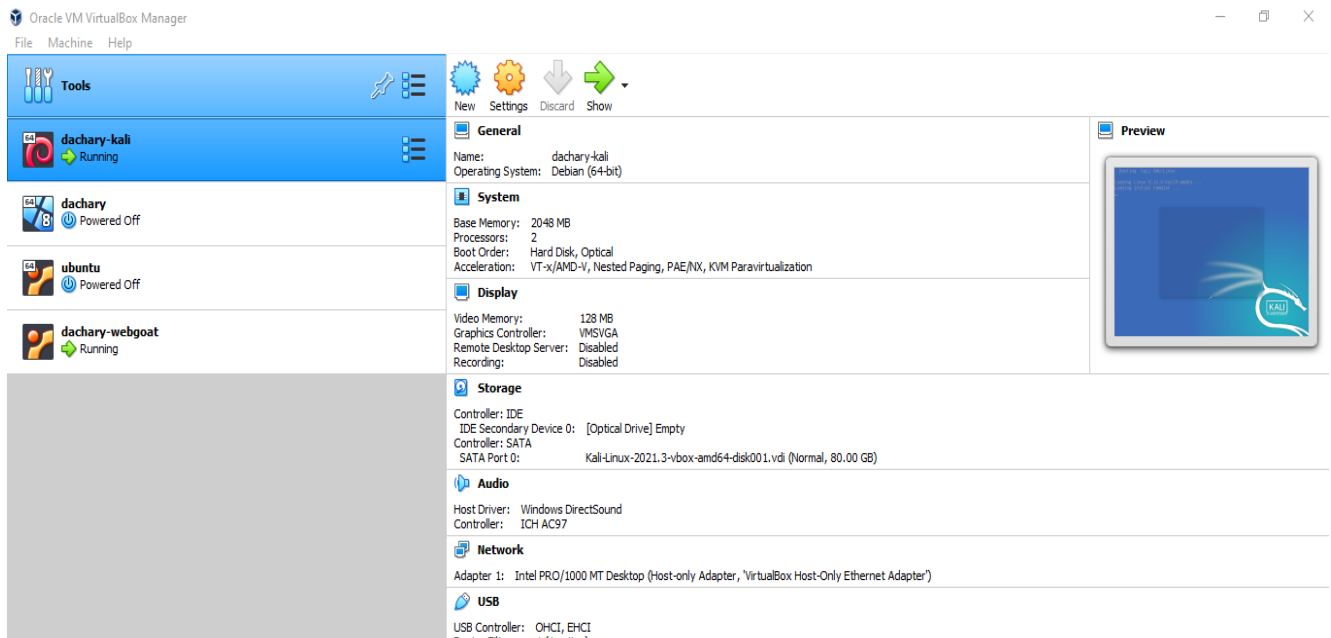


Lab 6: XSS and SQL Injection Attack

Overview:

The main objective of this lab is to install Kali and Webgoat Virtual Machines and perform XSS and SQL injection attack and scan the given WebGoat Virtual Machine (VM) application using OWASP ZAP. By the end of this lab, we will know the idea of these attacks and security measures that are to be implemented to avoid such attacks.

- a. **Machine installation:** The Kali machine was already installed in my VM, but I changed my name to my GMU id, then I installed WebGoat VM. I changed network setting of both machines to Host only to access directly from one another. Screenshot of VirtualBox showing my Kali and WebGoat with prefix is shown below.



The following screenshot shows the IP address and Link that I will use in my kali machine to access the lab. The link used in entire lab will be <http://192.168.56.102>.

```
dachary-webgoat [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

You can access the web apps at http://192.168.56.102/

You can administer / configure this machine through the console here, by SSHing
to 192.168.56.102, via Samba at \\192.168.56.102\\, or via phpmyadmin at
http://192.168.56.102/phpmyadmin.

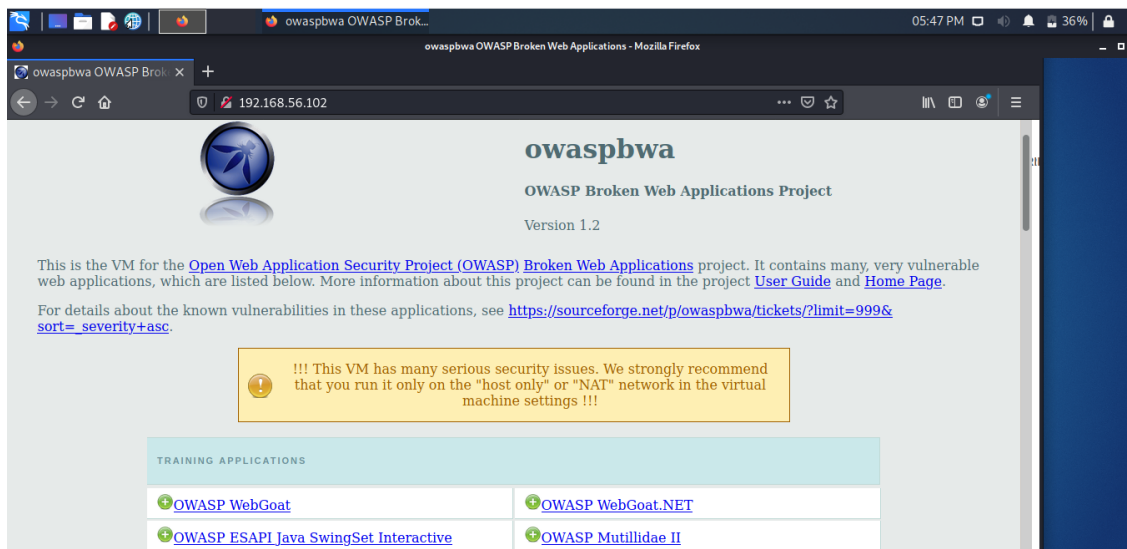
In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:eb:f6:d1
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feeb:f6d1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:52 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5938 (5.9 KB)  TX bytes:7061 (7.0 KB)
          Interrupt:9 Base address:0xd020

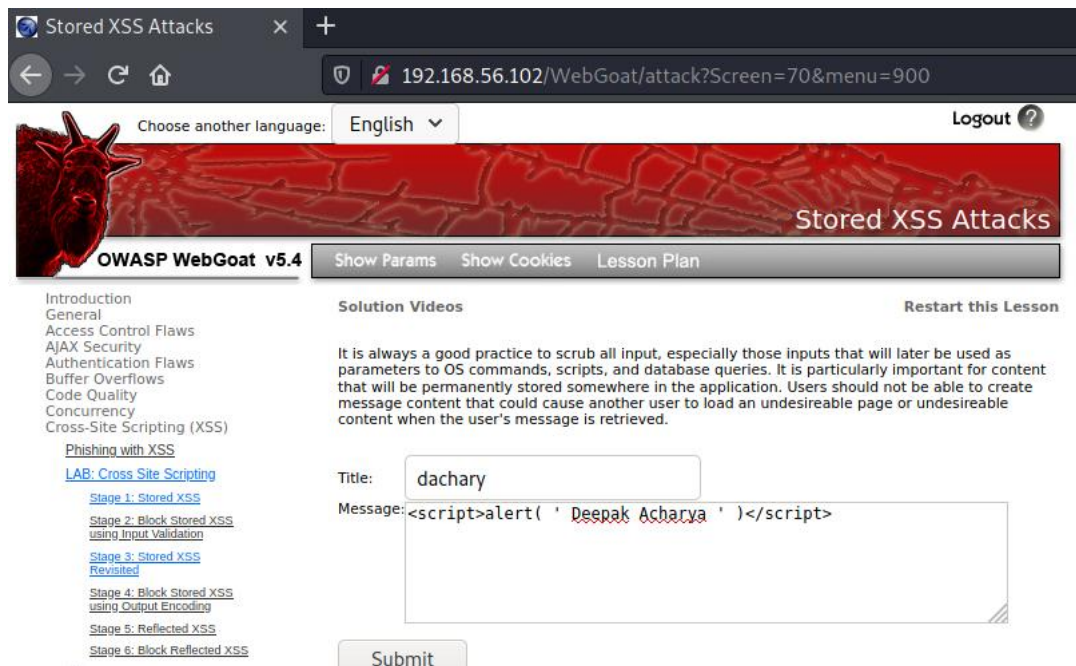
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:58 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:19705 (19.7 KB)  TX bytes:19705 (19.7 KB)

root@owaspbwa:~#
```

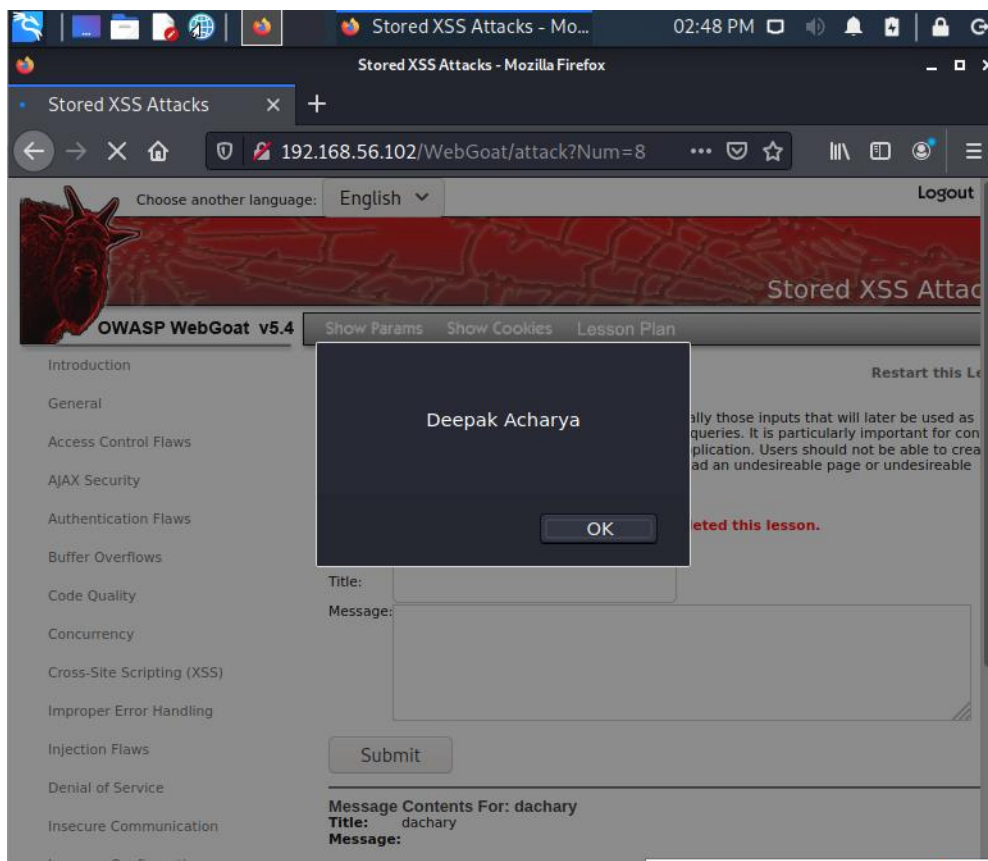
The initial page when I entered above link is shown in the figure below.



b. **Stored XSS attacks:** I opened the XSS attack page as shown in the following screenshot.



In the title field I typed in my GMU ID and in the message field I typed in the following Script. `<script>alert(' Deepak Acharya ')</script>` . When I submitted, I got the output as shown in the figure below. This way I was able to finish the stored XSS attack.



- c. **Reflected XSS attack:** For the reflected attack I typed in the following script in the access code field as shown in the screenshot below. `<script>alert(' Deepak Acharya ')</script>`

Shopping Cart

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

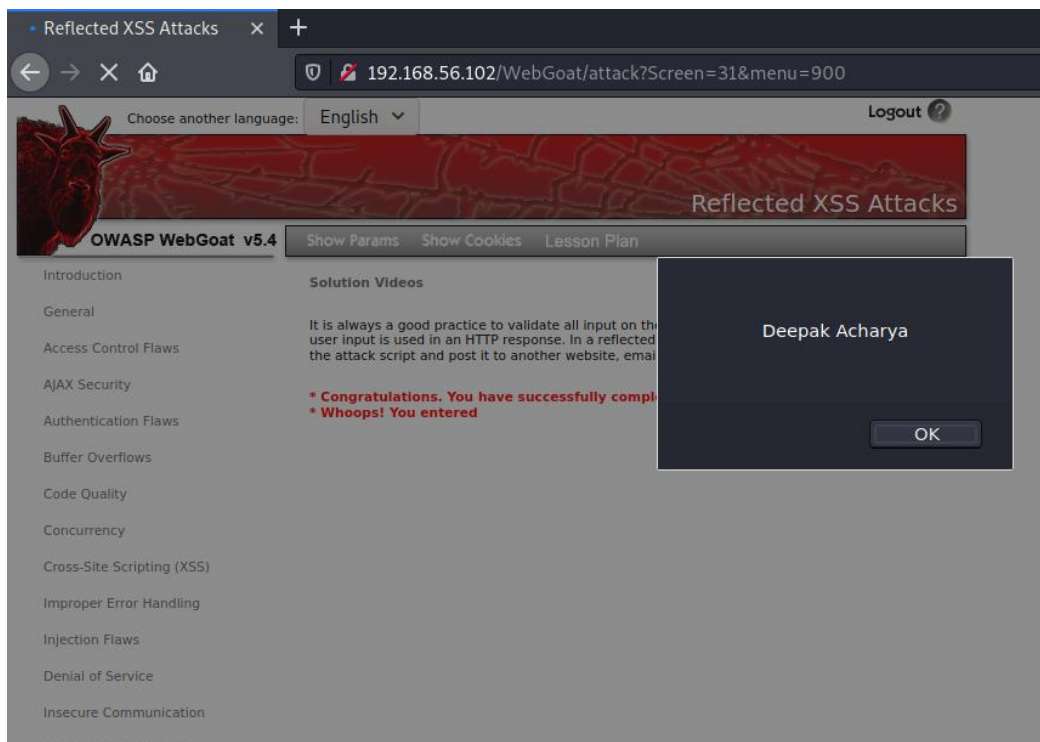
The total charged to your credit card: \$0.00 UpdateCart

Enter your credit card number:

Enter your three digit access code:

Purchase

After I clicked on purchase, I got the following result.



The following screenshot shows that, I completed both stored XSS attack and Reflected XSS attack in the given vulnerable machine.

How to work with WebGoat

←

→

↶

🏠

🔒

192.168.56.102/WebGoat/attack

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

Stage 1: Stored XSS

Stage 2: Block Stored XSS using Input Validation

Stage 3: Stored XSS Revisited

Stage 4: Block Stored XSS using Output Encoding

Stage 5: Reflected XSS

Stage 6: Block Reflected XSS

Stored XSS Attacks

Reflected XSS Attacks

Cross Site Request Forgery (CSRF)

CSRF Prompt By-Pass

CSRF Token By-Pass

HTTPOnly Test

Cross Site Tracing (XST) Attacks

Improper Error Handling

Injection Flaws

Denial of Service

Insecure Communication

Insecure Configuration

Insecure Storage

Malicious Execution

Parameter Tampering

How To Work With WebGoat

Welcome to a short introduction to WebGoat.

Here you will learn how to use WebGoat and additional tools for the lessons.

Environment Information

WebGoat uses the Apache Tomcat server. It is configured to run on localhost although this can be easily changed. This configuration is for single user, additional users can be added in the tomcat-users.xml file. If you want to use WebGoat in a laboratory or in class you might need to change this setup. Please refer to the Tomcat Configuration in the Introduction section.

The WebGoat Interface

Logout ?

2

3

4

5

6

7

8

Http Basics

OWASP WebGoat V5.2

← Hints

Show Params

Show Cookies

Lesson Plan

Show Java

Solution

1

Introduction

General

Http Basics

HTTP Solitaire

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

Denial of Service

Improper Error Handling

Injection Flaws

Insecure Communication

Insecure Configuration

Insecure Storage

Parameter Tampering

Session Management Flaws

Web Services

Admin Functions

Challenge

Restart this Lesson

Enter your name in the input field below and press "go" to submit. The server will accept the request, reverse the input, and display it back to the user, illustrating the basics of handling an HTTP request.

The user should become familiar with the features of WebGoat by manipulating the above buttons to view hints and solution. You have to use WebScarab for the first time.

Enter your name:

Go!

OWASP Foundation | Project WebGoat

Page 5 of 9

- d. **Sql Injection:** For the SQL injection attack, I entered the following string in the given space as shown below. **Smith' 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'.

Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.

Enter your last name:

SELECT * FROM user_data WHERE last_name = ?

No results matched. Try Again.

OWASP Foundation | Project WebGoat | Report Bug

After I Clicked Go I got the following result which showed all the credit card information.

String SQL Injection

←

→

↺

🏠

192.168.56.102/WebGoat/attack?Screen=36&menu=1100

Improper Error Handling

Injection Flaws

Command Injection

Numeric SQL Injection

Log Spoofing

XPATH Injection

String SQL Injection

LAB: SQL Injection

Stage 1: String SQL Injection

Stage 2: Parameterized Query #1

Stage 3: Numeric SQL Injection

Stage 4: Parameterized Query #2

Modify Data with SQL Injection

Add Data with SQL Injection

Database Backdoors

Blind Numeric SQL Injection

Blind String SQL Injection

Denial of Service

Insecure Communication

Insecure Configuration

Insecure Storage

Malicious Execution

Parameter Tampering

Session Management Flaws

Web Services

Admin Functions

Challenge

forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, 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'.

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

*** Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.**

Enter your last name:

SELECT * FROM user_data WHERE last_name = 'Smith' 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	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

Page 6 of 9

- e. **Modify Data with SQL injection:** For the modification of the data we saw above I entered the following string in the userid field. **jsmith';update salaries set salary=500 where userid='jsmith'** - as shown in the screenshot below.

Modify Data with SQL Injection

OWASP WebGoat v5.4

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Improper Error Handling
Injection Flaws

Command Injection
Numeric SQL Injection
Log Spoofing
XPath Injection
String SQL Injection
LAB: SQL Injection
Stage 1: String SQL Injection
Stage 2: Parameterized Query #1
Stage 3: Numeric SQL Injection
Stage 4: Parameterized Query #2
Modify Data with SQL Injection

Solution Videos

Restart this Lesson

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

Enter your userid:

Go!

USERID	SALARY
jsmith	20000

Created by Chuck Willis MANDIANT
INTELLIGENT INFORMATION SECURITY

OWASP Foundation | Project WebGoat | Report Bug

Since, I modified the salary to be 500, the updated result is shown in the following screenshot after I typed jsmith in the given field.

Modify Data with SQL Injection

OWASP WebGoat v5.4

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Improper Error Handling
Injection Flaws

Command Injection
Numeric SQL Injection
Log Spoofing
XPath Injection
String SQL Injection
LAB: SQL Injection
Stage 1: String SQL Injection
Stage 2: Parameterized Query #1
Stage 3: Numeric SQL Injection
Stage 4: Parameterized Query #2
Modify Data with SQL Injection

Solution Videos

Restart this Lesson

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

Enter your userid:

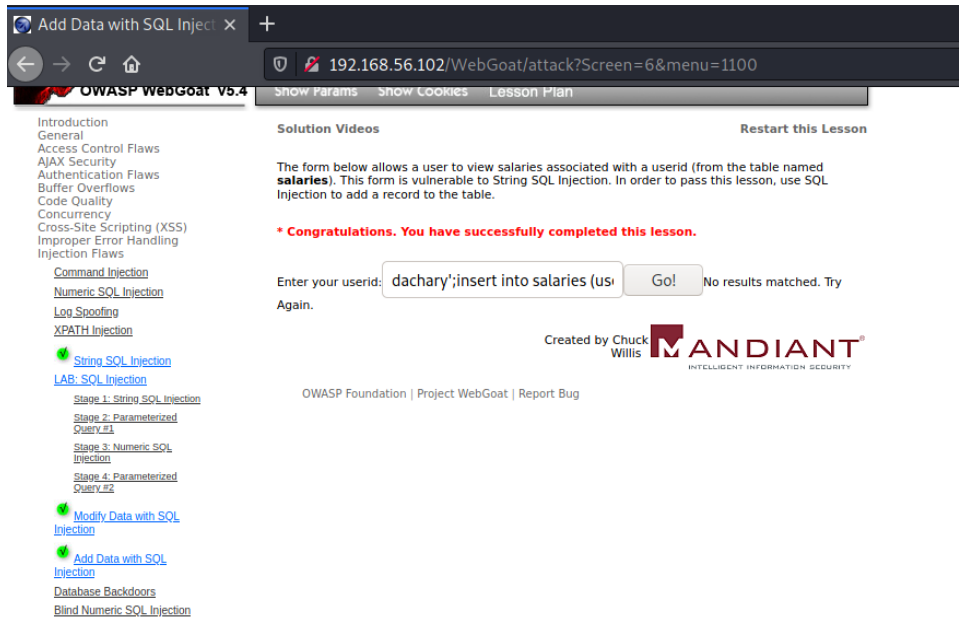
Go!

USERID	SALARY
jsmith	500

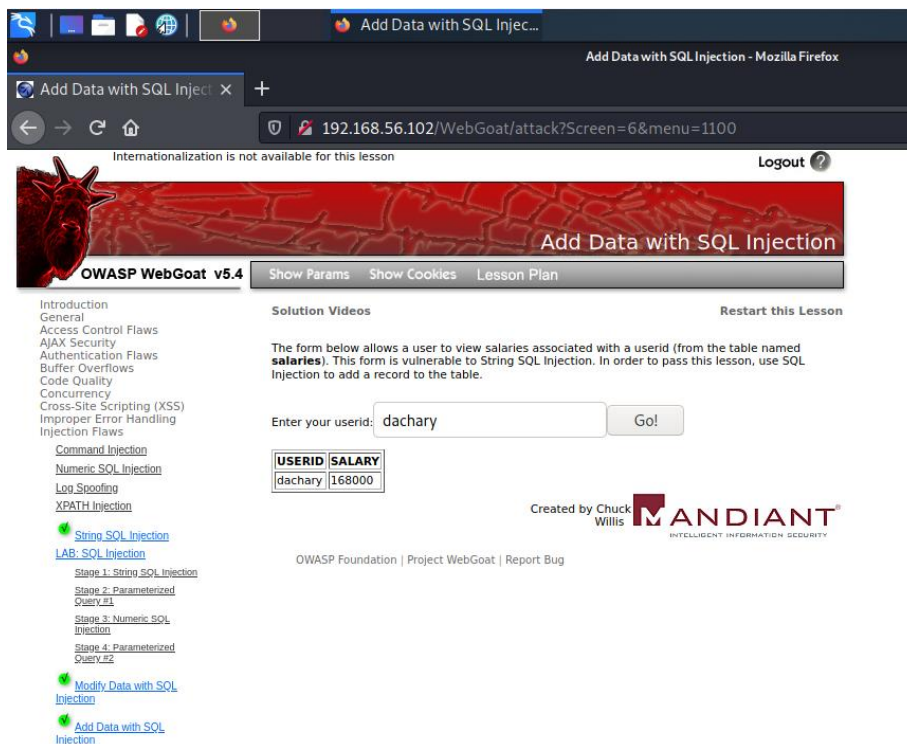
Created by Chuck Willis MANDIANT
INTELLIGENT INFORMATION SECURITY

OWASP Foundation | Project WebGoat | Report Bug

- f. **Adding Data with SQL injection:** For adding data into the database using SQL injection I used the following data with my user id and desired salary. **Script used: add sql: dachary';insert into salaries (userid, salary) values ('dachary', '168000')** - - as shown in the screenshot below.

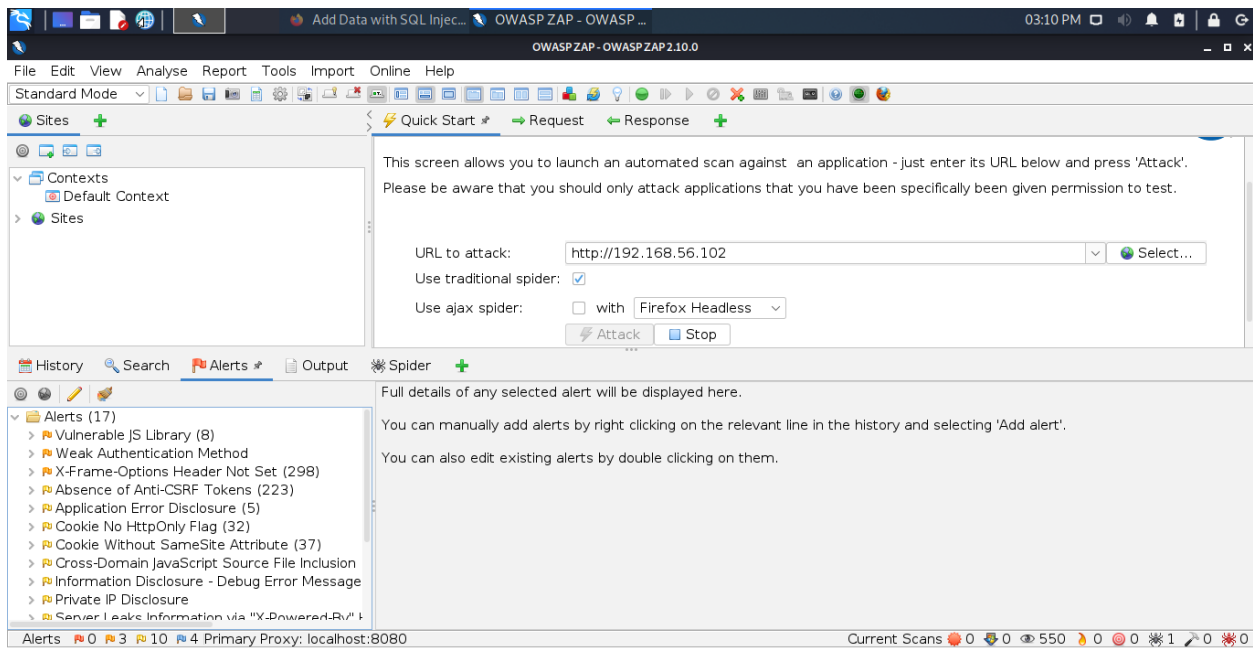


After I clicked, go and searched my user id, I got the following result.



The above screenshot also shows that we successfully completed three different SQL attacks.

- g. **Scan with OWASP ZAP:** For this step, I started Kali machine and started ZAP tool. When I entered IP of WebGoat machine and scanned with proper settings enabled, I got the following alerts.



Conclusion: This way we were successful in performing stored XSS attack, Reflected XSS attack, String SQL injection, Modification of data using SQL injection, and Adding data using SQL injection. Finally, we scanned for all these security alerts using the ZAP tool. Thus, from this ab we understand we should test all the input in all the applications so to ensure that they are free from these attacks. Thus, testing is an integral part in system development. Proper testing can mitigate all these attacks.