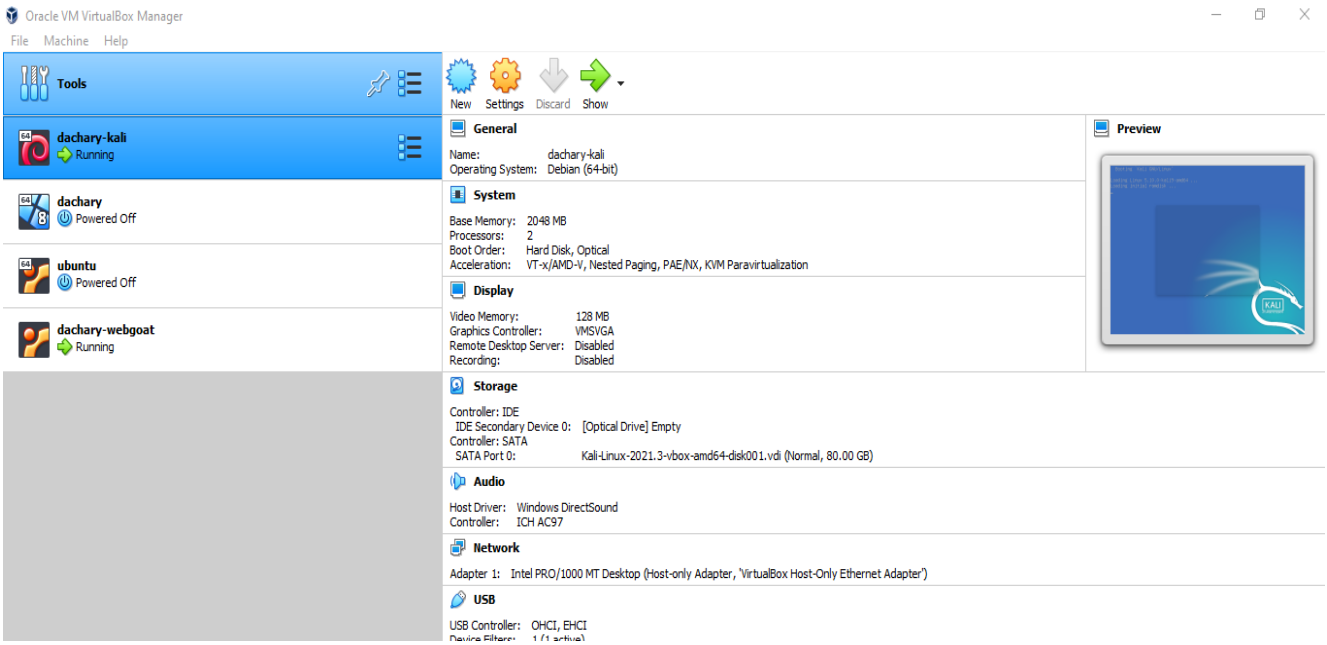
Lab 6: XSS and SQL Injection Attack

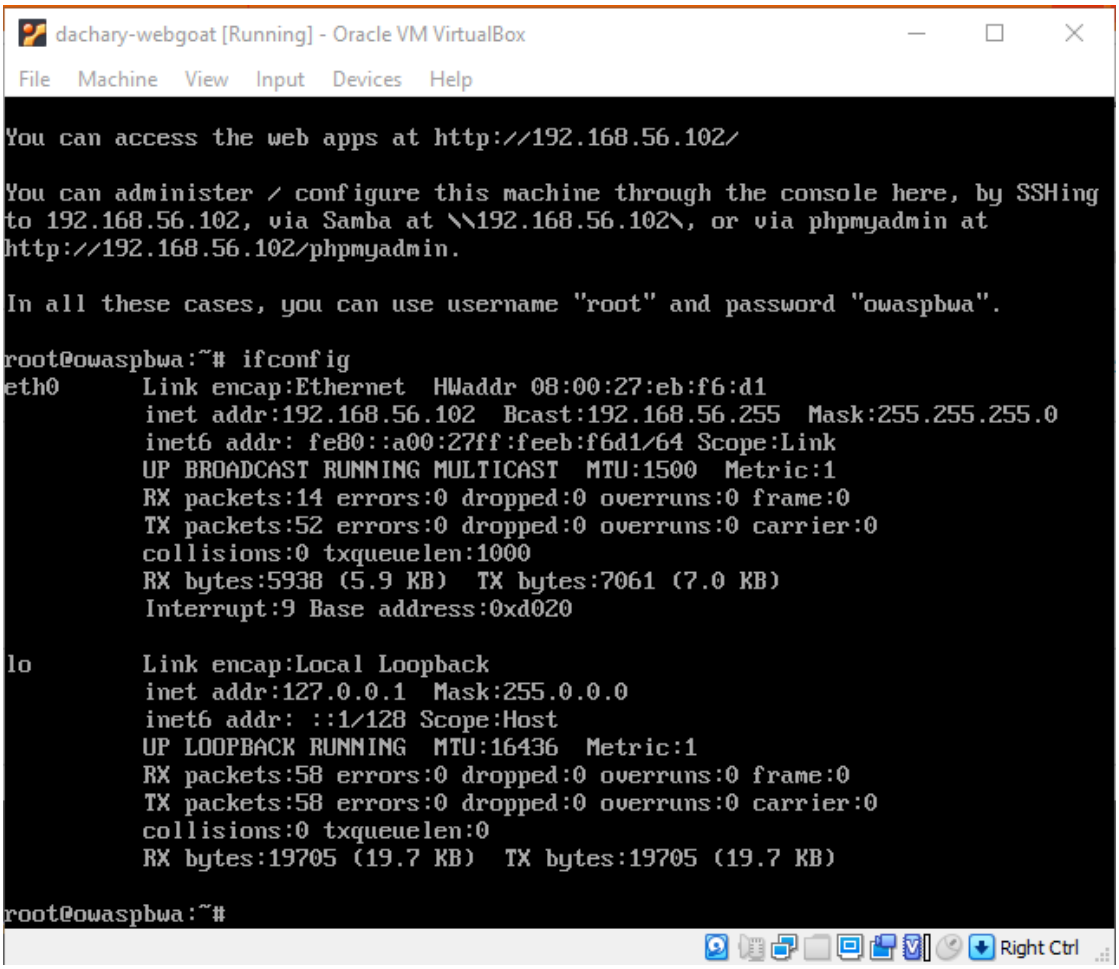
Overview:

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.

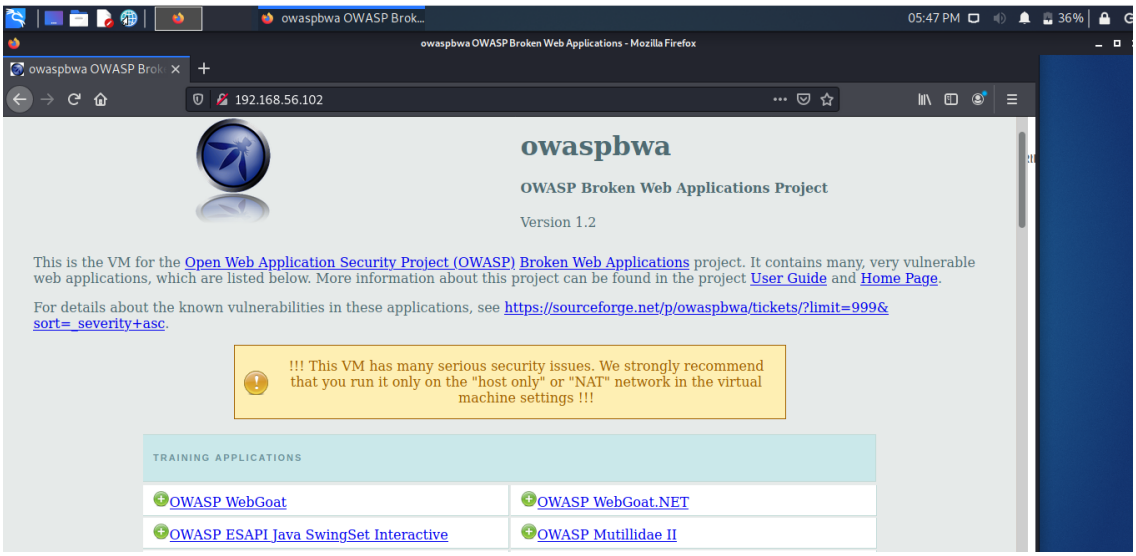
a. **Machine installation:** The Kali machine was already installed in my VM, but I changed my name to my GMU id, then I installed WebGoat VM. I changed network setting of both machines to Host only to access directly from one another. Screenshot of VirtualBox showing my Kali and WebGoat with prefix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



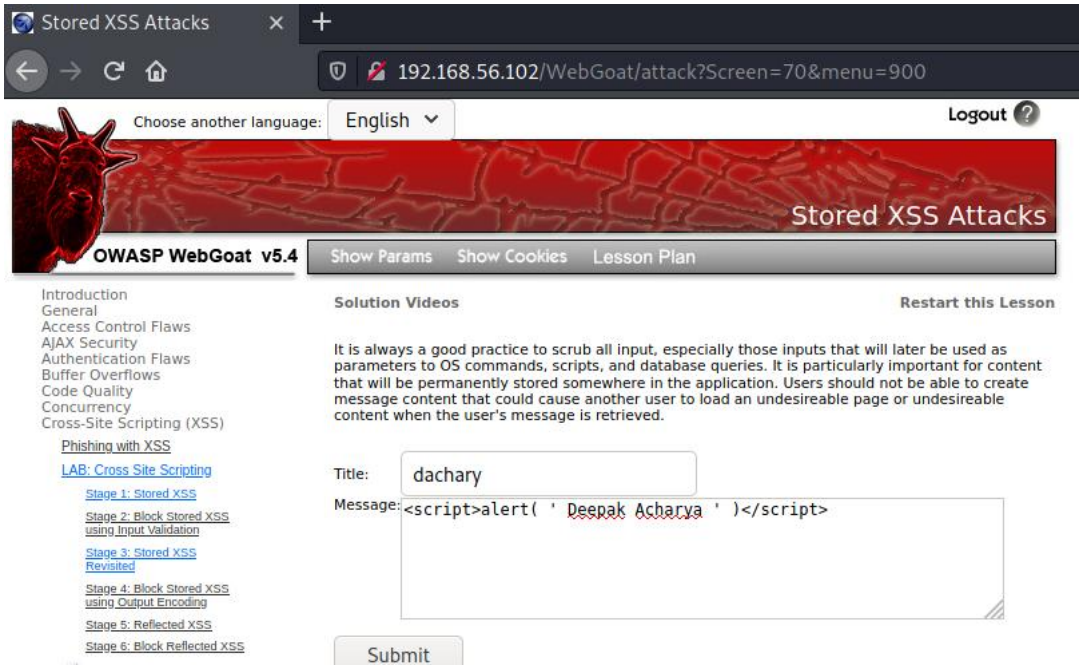
The following screenshot shows the IP address and Link that I will use in my kali machine to access the lab. The link used in entire lab will be http://192.168.56.102.



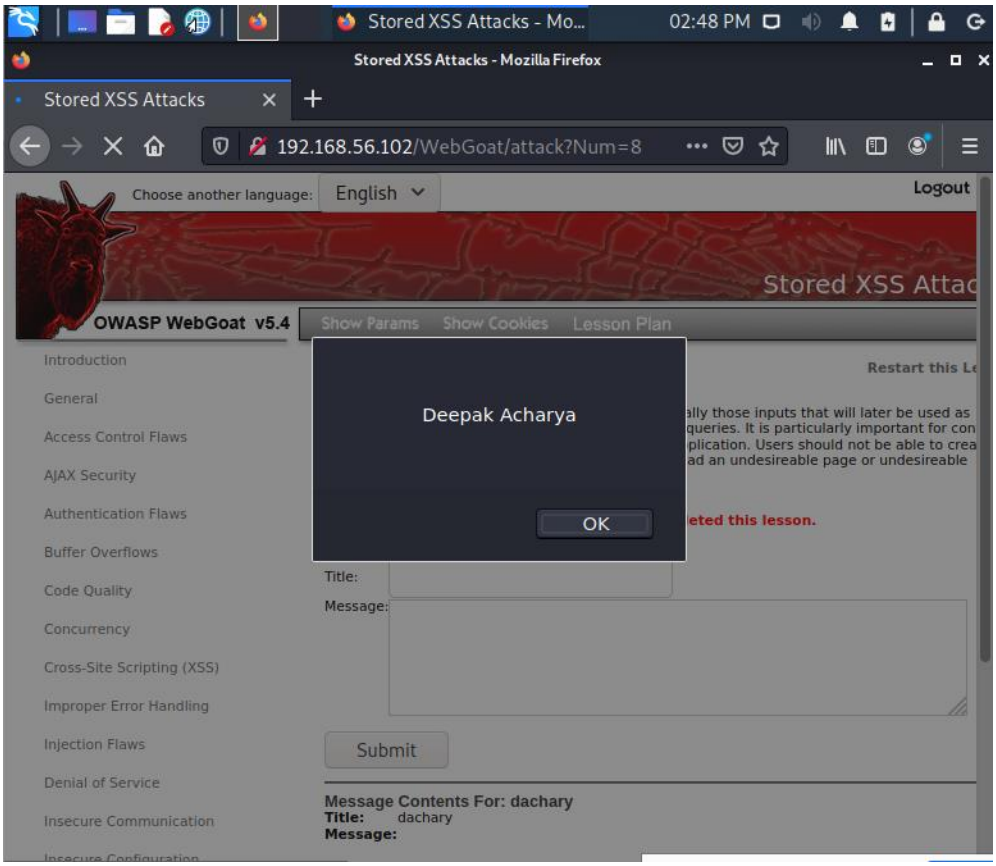
The initial page when I entered above link is shown in the figure below.



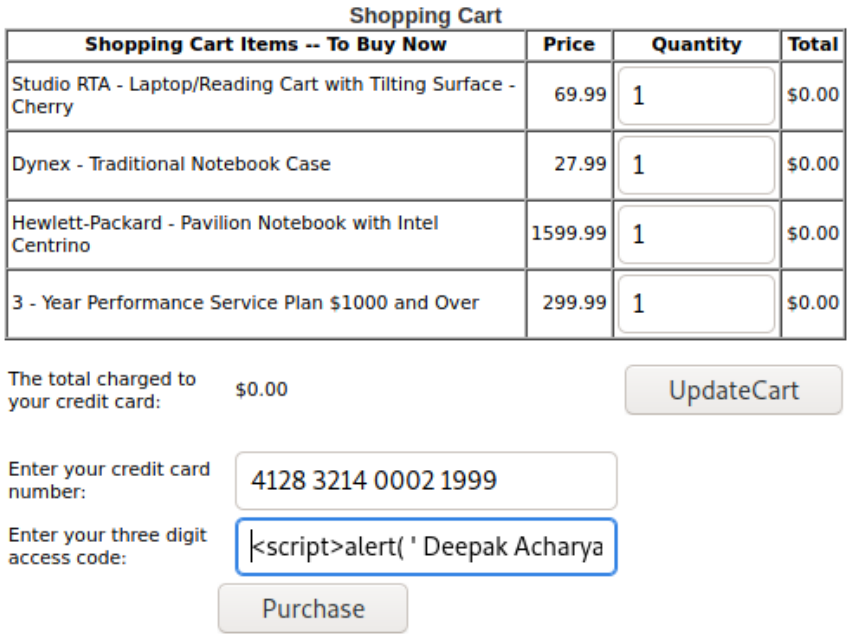
b. **Stored XSS attacks:** I opened the XSS attack page as shown in the following screenshot.



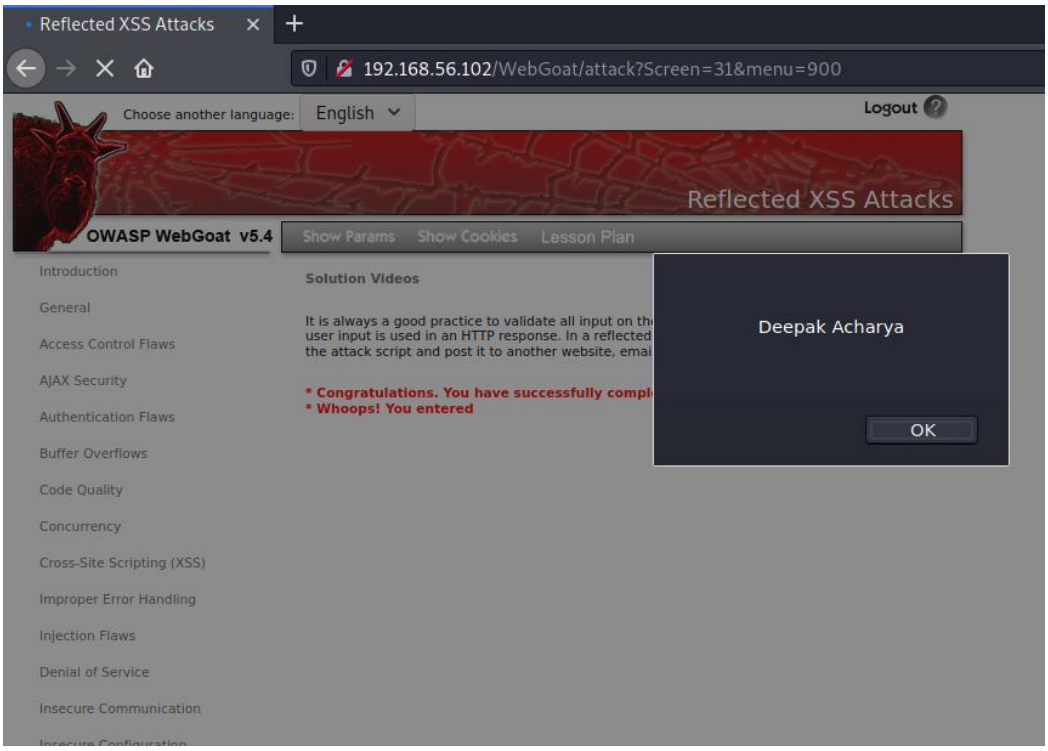
In the title field I typed in my GMU ID and in the message field I typed in the following Script. **<script>alert(' Deepak Acharya ')</script> .** When I submitted, I got the output as shown in the figure below. This way I was able to finish the stored XSS attack.



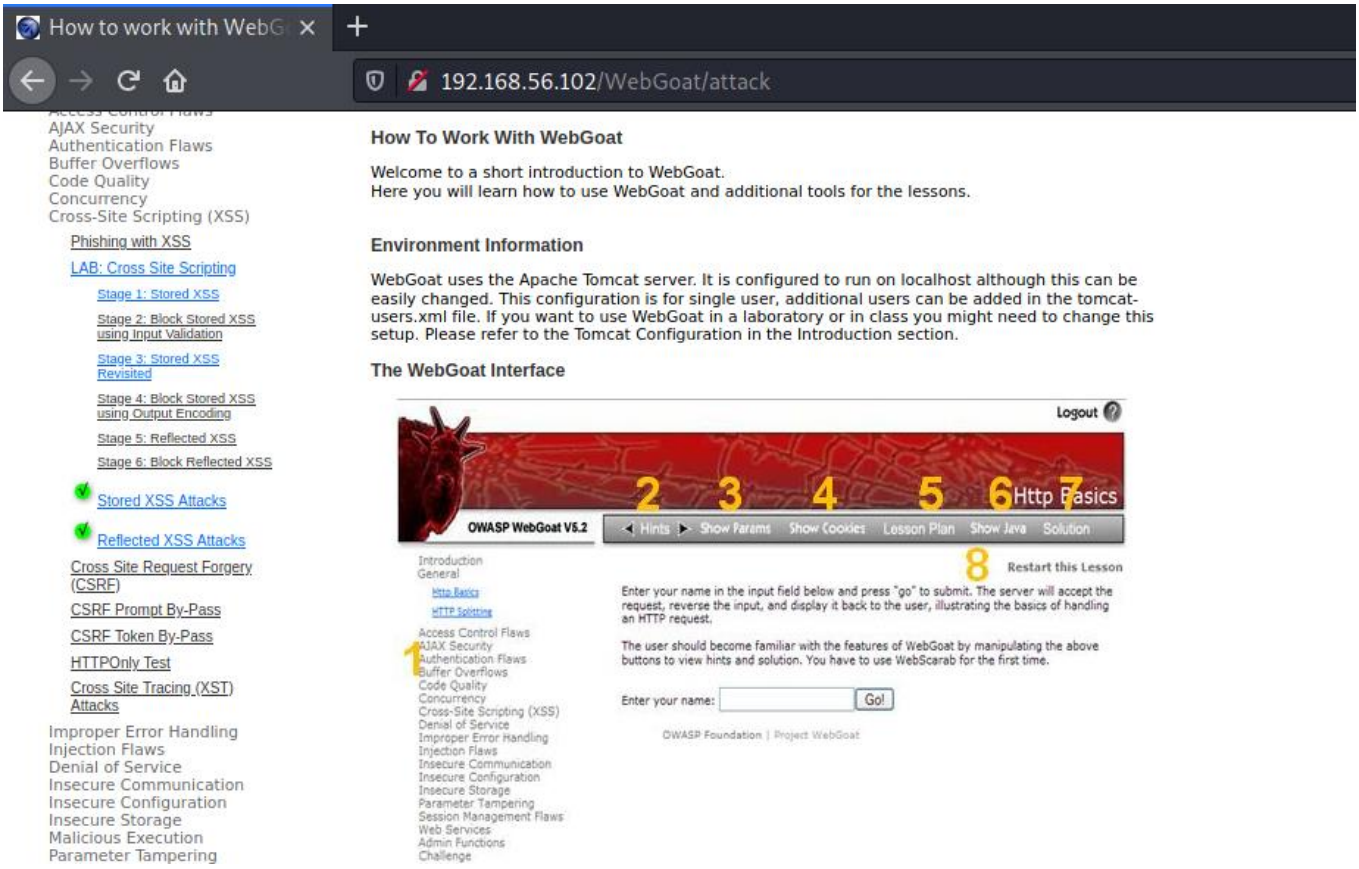
c. **Reflected XSS attack**: For the reflected attack I typed in the following script in the access code field as shown in the screenshot below. **<script>alert(' Deepak Acharya ')</script>**



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 WebG
192.168.56.102/WebGoat/attack
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 Http Basics 7
OWASP WebGoat V5.2
Hints
Show Params
Show Cookies
Lesson Plan
Show Java
Solution
8 Restart this Lesson
Introduction
General
Http Basics
HTTP Splitting
1
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
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

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.

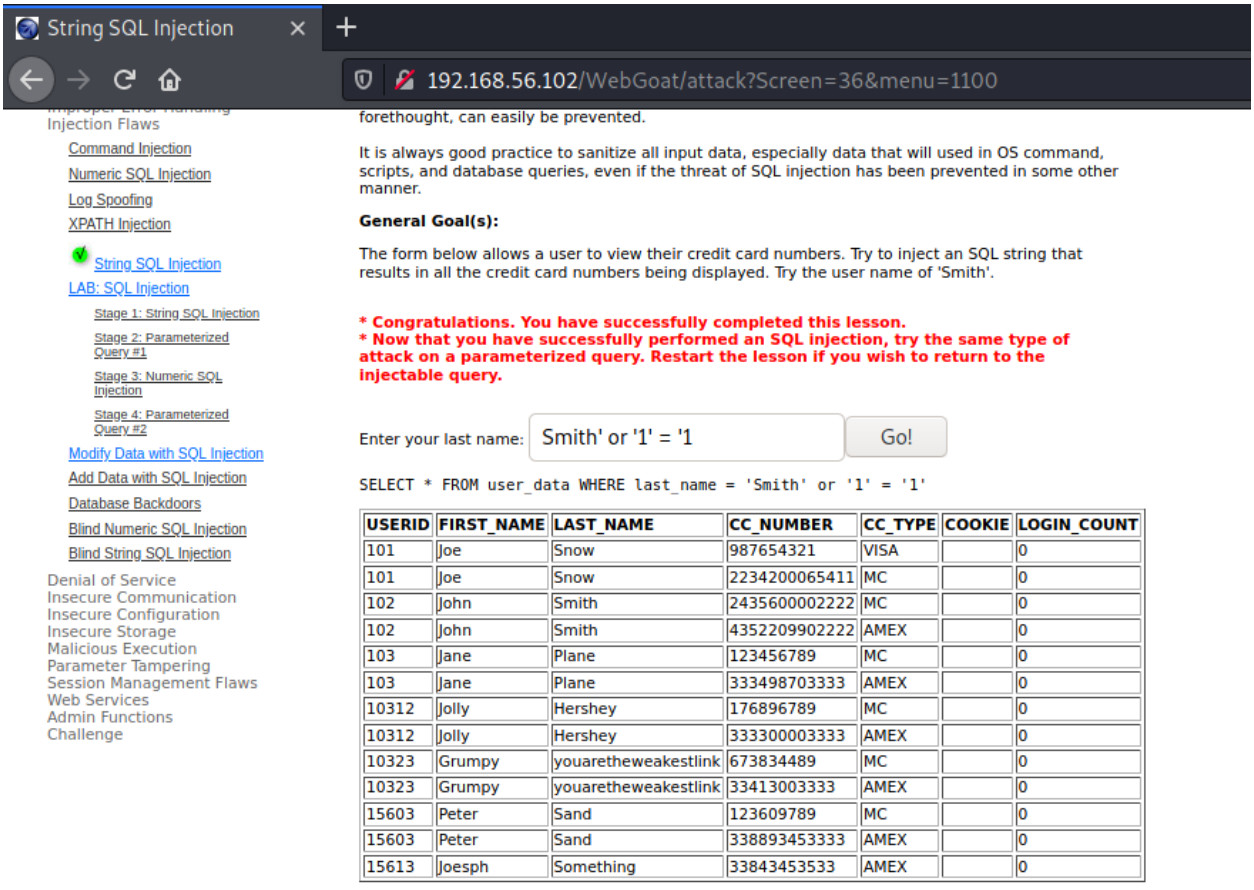
Enter your last name: Smith' or '1' = '1 Go!

```
SELECT * FROM user_data WHERE last_name = ?
```

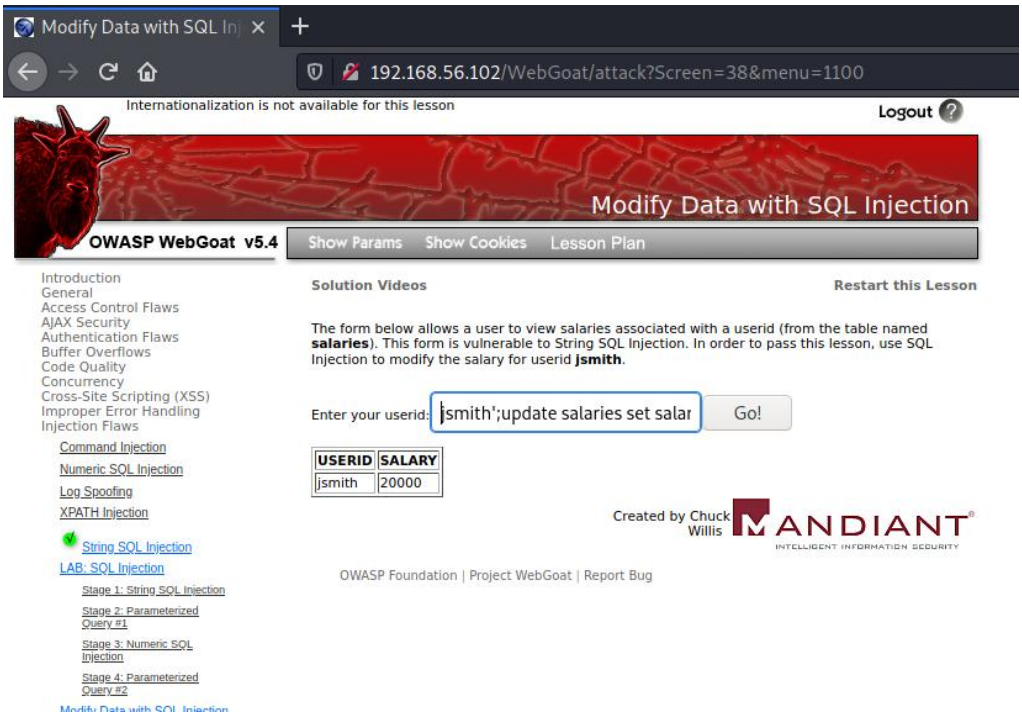
No results matched. Try Again.

OWASP Foundation | Project WebGoat | Report Bug

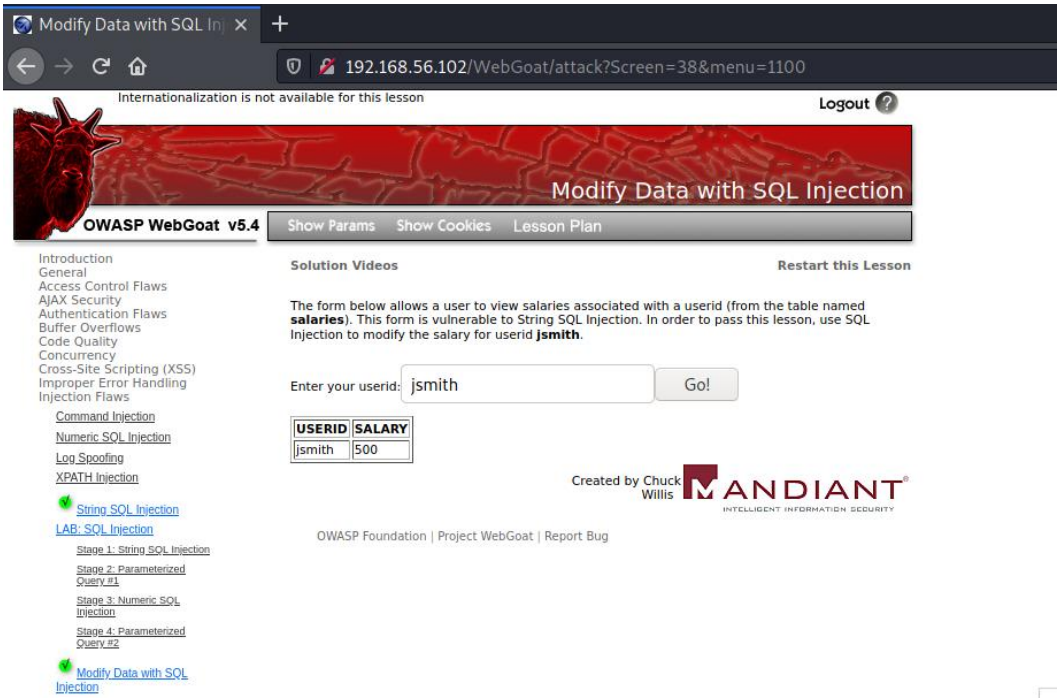
After I Clicked Go I got the following result which showed all the credit card information.



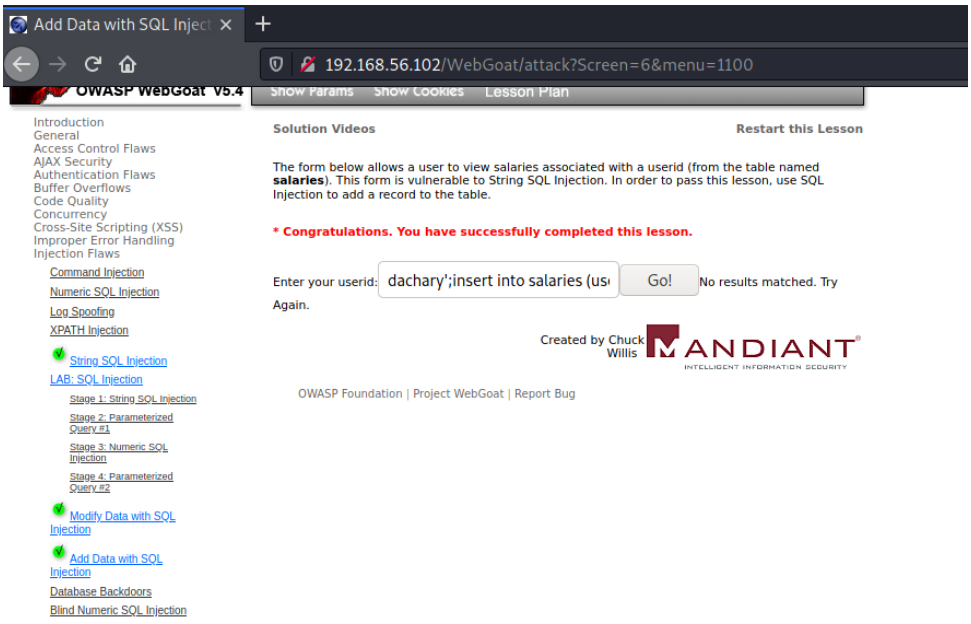
e. **Modify Data with SQL injection**: For the modification of the data we saw above I entered the following string in the userid field. **jsmith';update salaries set salary=500 where userid='jsmith'** - - as shown in the screenshot below.



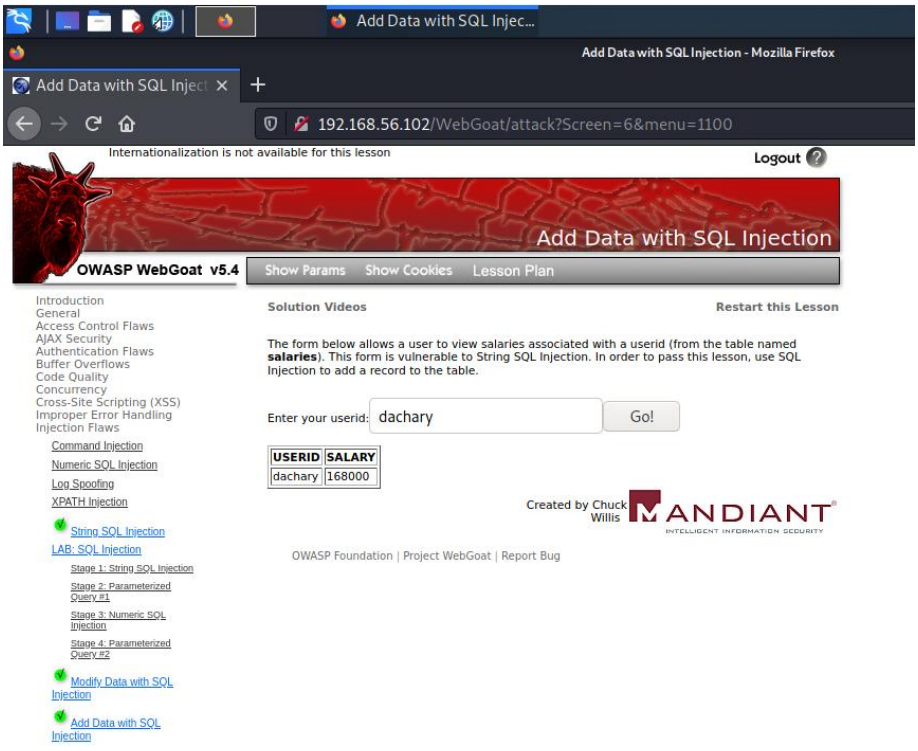
Since, I modified the salary to be 500, the updated result is shown in the following screenshot after I typed jsmith in the given field.



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.

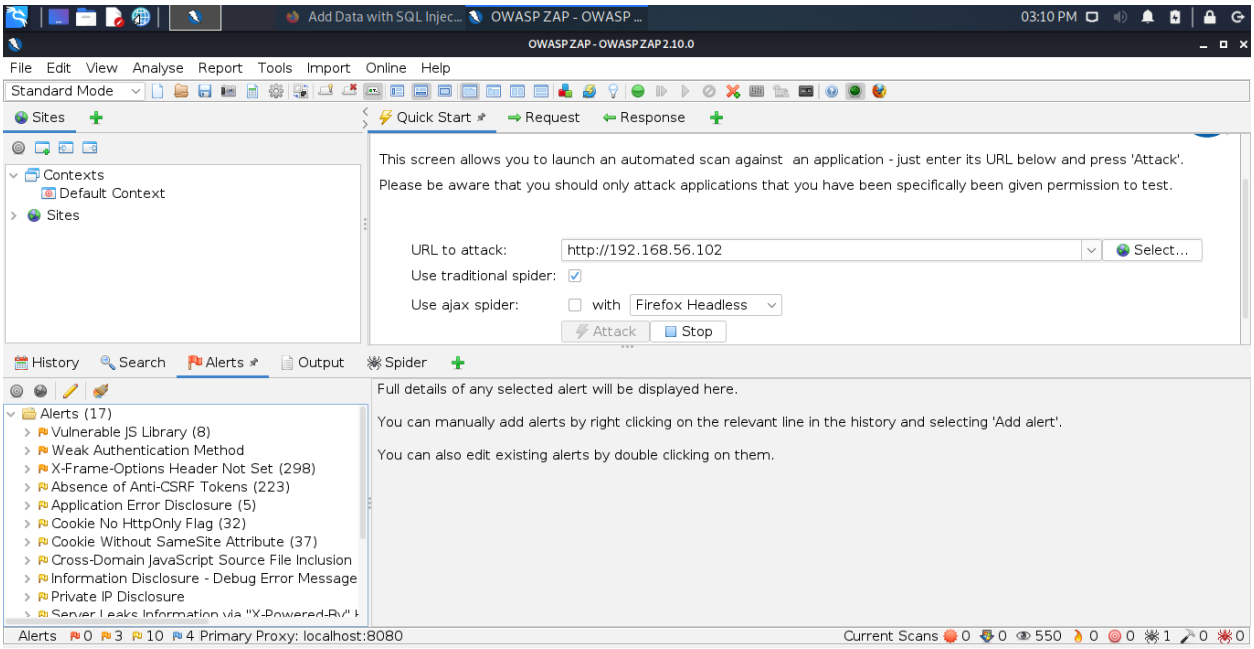


After I clicked, go and searched my user id, I got the following result.



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.