WebGoat – SQL Injection

Name **:** Aaron Mathew

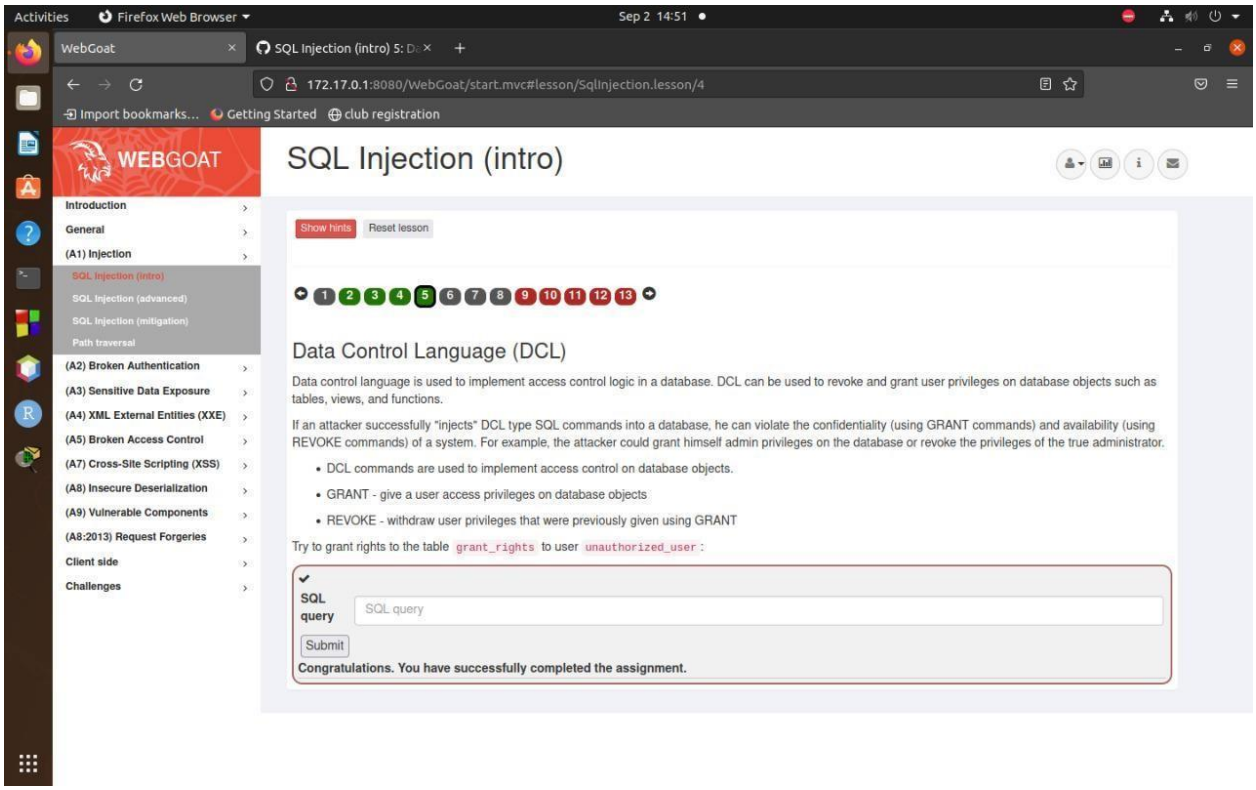
Roll.no **:** 20BIS001.

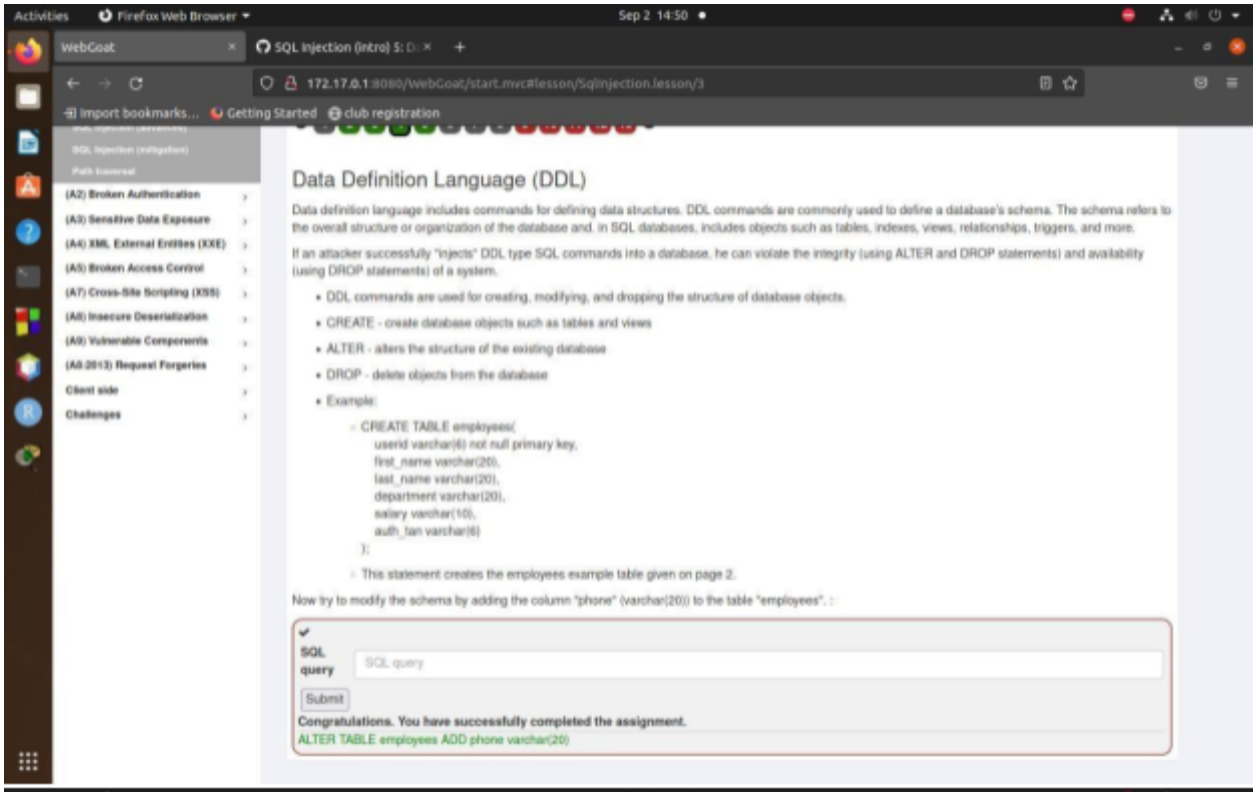
Subject **:** U18INI5600 – Engg Clinics – 5.

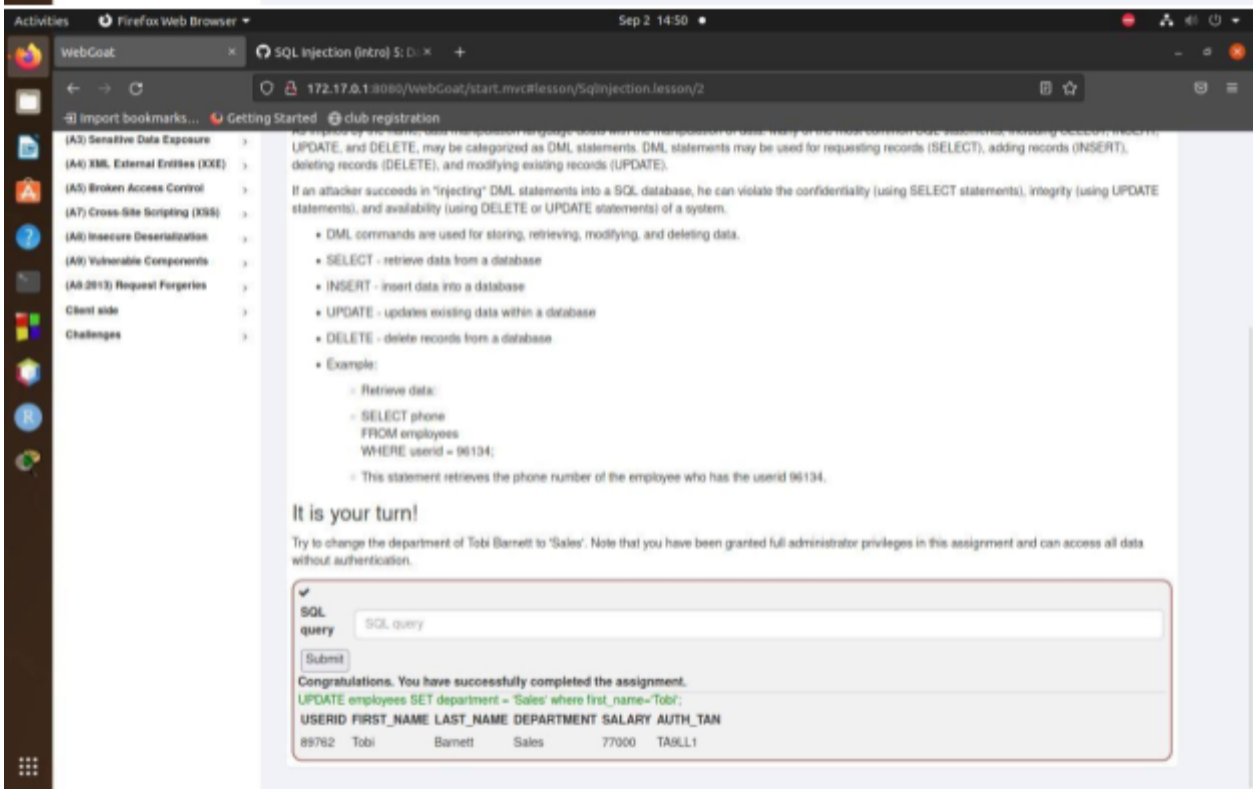
Ex.no **:** A1 – SQL Injection – Intro.

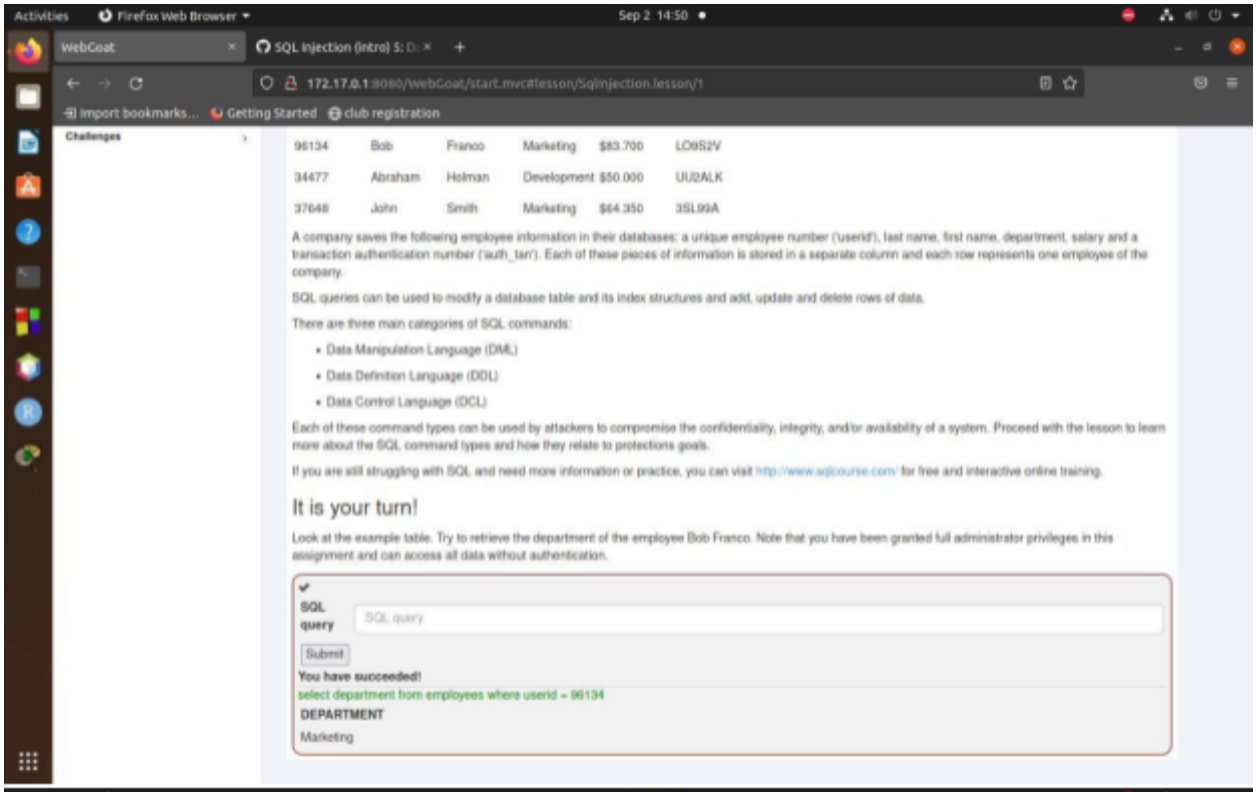
Date **:** 03-09-2022.

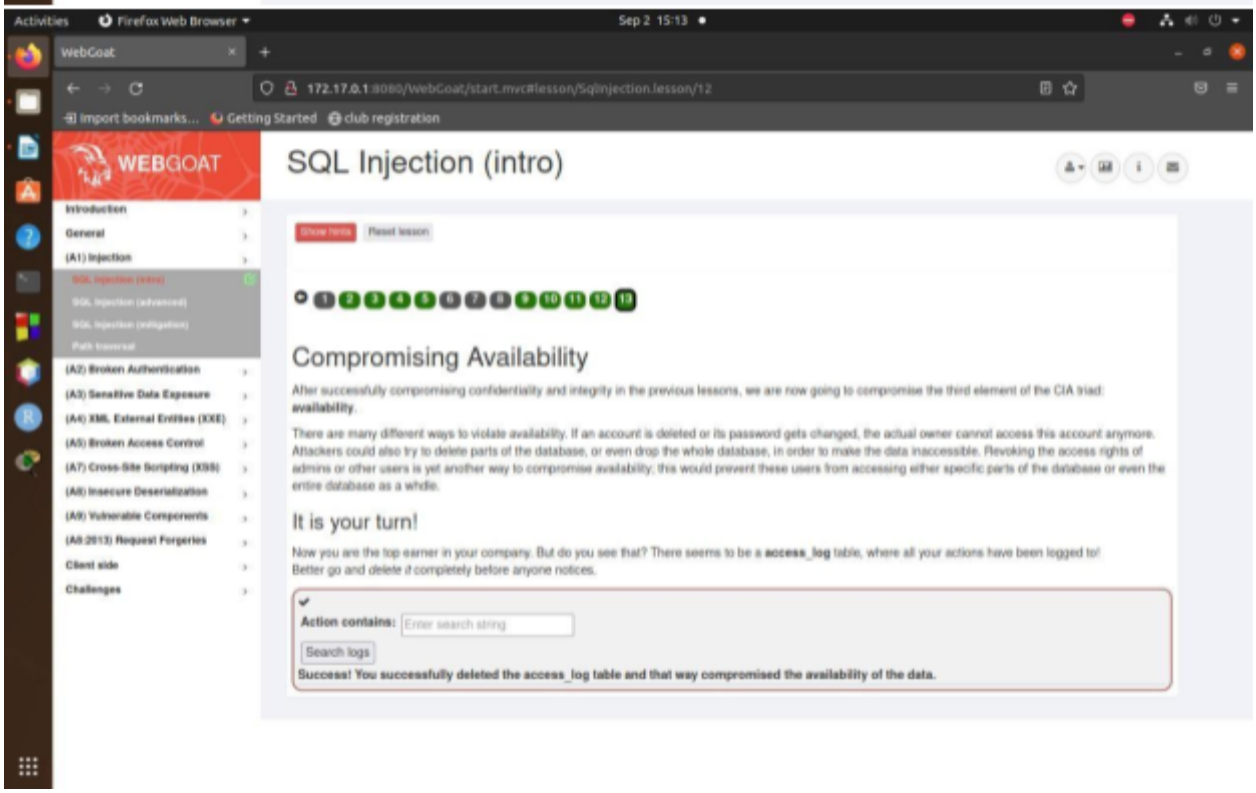
Screenshots of exercises:

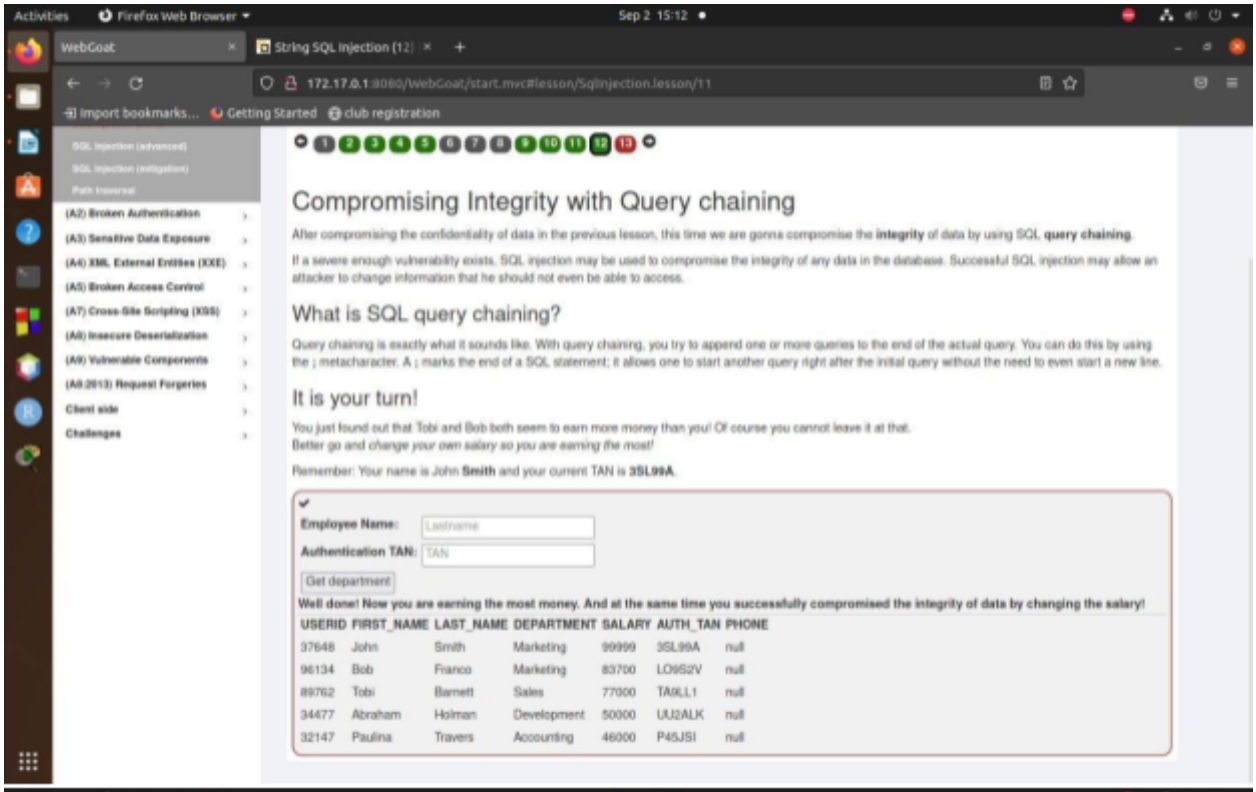


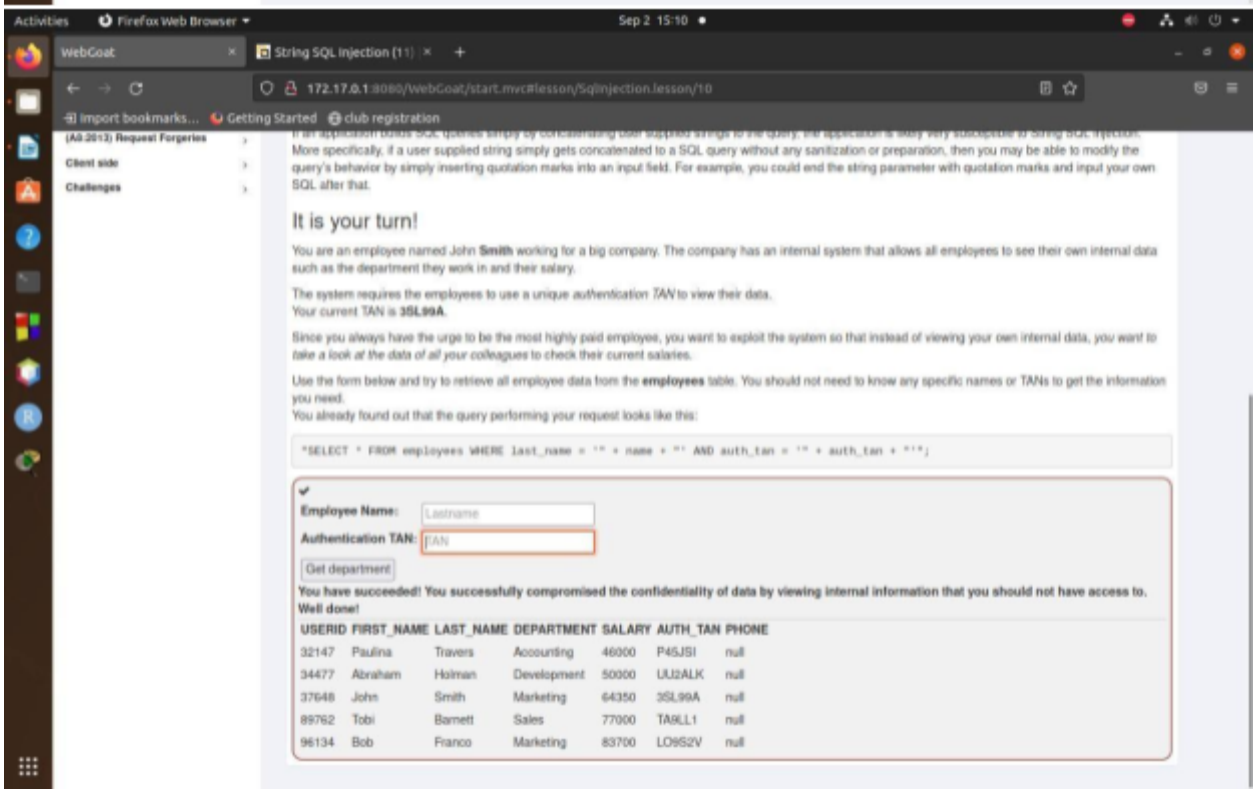

Activities
Firefox Web Browser
Sep 2 14:50
WebGoat
SQL Injection (intro) 5: D...
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/3
Import bookmarks...
Getting Started
club registration
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Data Definition Language (DDL)
Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to the overall structure or organization of the database and, in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.
If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.
DDL commands are used for creating, modifying, and dropping the structure of database objects.
CREATE - create database objects such as tables and views
ALTER - alters the structure of the existing database
DROP - delete objects from the database
Example:
CREATE TABLE employees(
userid varchar(6) not null primary key,
first_name varchar(20),
last_name varchar(20),
department varchar(20),
salary varchar(10),
auth_tan varchar(6)
);
This statement creates the employees example table given on page 2.
Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :
SQL query
SQL query
Submit
Congratulations. You have successfully completed the assignment.
ALTER TABLE employees ADD phone varchar(20)

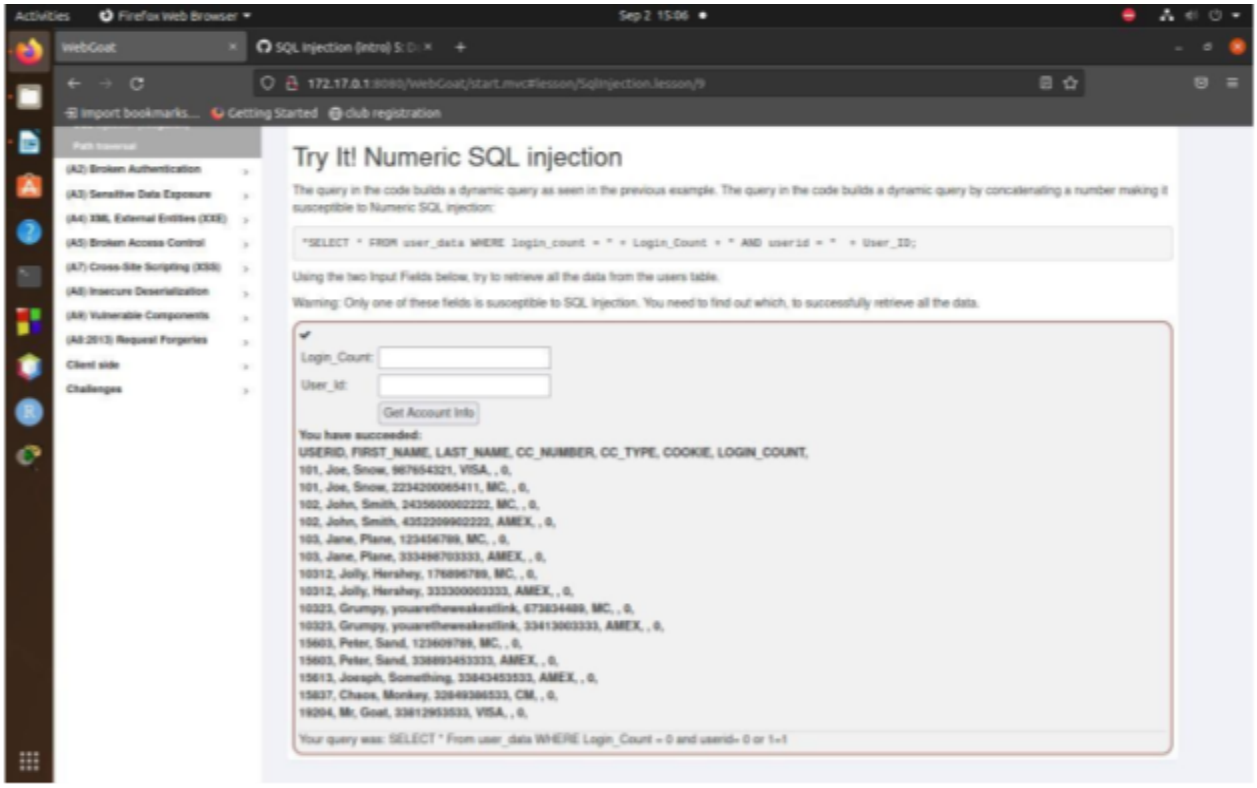

Activities
Firefox Web Browser
Sep 2 14:50
WebGoat
SQL Injection (intro) 5: D...
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/2
Import bookmarks...
Getting Started
club registration
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).
If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.
DML commands are used for storing, retrieving, modifying, and deleting data.
SELECT - retrieve data from a database
INSERT - insert data into a database
UPDATE - updates existing data within a database
DELETE - delete records from a database
Example:
Retrieve data:
SELECT phone
FROM employees
WHERE userid = 96134;
This statement retrieves the phone number of the employee who has the userid 96134.
It is your turn!
Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.
SQL query
SQL query
Submit
Congratulations. You have successfully completed the assignment.
UPDATE employees SET department = 'Sales' where first_name='Tobi';
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
89762 Tobi Barnett Sales 77000 TA9LL1

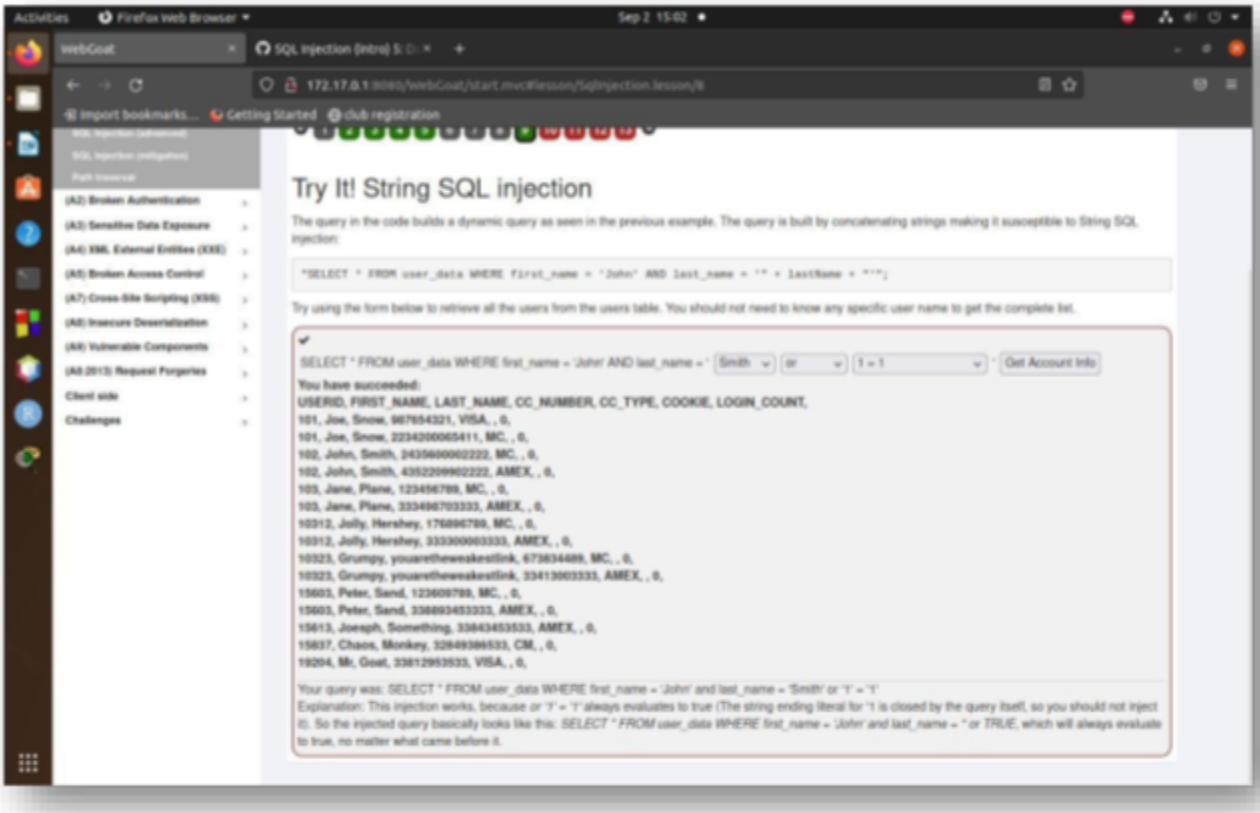

Activities
Firefox Web Browser
Sep 2 14:50
WebGoat
SQL Injection (intro) 5: D...
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/1
Import bookmarks...
Getting Started
club registration
Challenges
96134 Bob Franco Marketing $83.700 LO9S2V
34477 Abraham Holman Development $50.000 UU2ALK
37648 John Smith Marketing $64.350 3SL99A
A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.
SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.
There are three main categories of SQL commands:
Data Manipulation Language (DML)
Data Definition Language (DDL)
Data Control Language (DCL)
Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protections goals.
If you are still struggling with SQL and need more information or practice, you can visit http://www.sqlcourse.com/ for free and interactive online training.
It is your turn!
Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.
SQL query
SQL query
Submit
You have succeeded!
select department from employees where userid = 96134
DEPARTMENT
Marketing


Activities
Firefox Web Browser
Sep 2 15:13
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/12
Import bookmarks...
Getting Started
club registration
WEBGOAT
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
SQL Injection (intro)
Show hints
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12 13
Compromising Availability
After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: availability.
There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.
It is your turn!
Now you are the top earner in your company. But do you see that? There seems to be a access_log table, where all your actions have been logged to! Better go and delete it completely before anyone notices.
Action contains:
Enter search string
Search logs
Success! You successfully deleted the access_log table and that way compromised the availability of the data.


Activities
Firefox Web Browser
Sep 2 15:12
WebGoat
String SQL Injection (12)
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/11
Import bookmarks...
Getting Started
club registration
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
1 2 3 4 5 6 7 8 9 10 11 12 13
Compromising Integrity with Query chaining
After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the integrity of data by using SQL query chaining.
If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.
What is SQL query chaining?
Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.
It is your turn!
You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.
Better go and change your own salary so you are earning the most!
Remember: Your name is John Smith and your current TAN is 3SL99A.
Employee Name:
Lastname
Authentication TAN:
TAN
Get department
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 99999 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


Activities
Firefox Web Browser
Sep 2 15:10
WebGoat
String SQL Injection (11)
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/10
Import bookmarks...
Getting Started
club registration
(A8:2013) Request Forgeries
Client side
Challenges
More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.
It is your turn!
You are an employee named John Smith working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.
The system requires the employees to use a unique authentication TAN to view their data.
Your current TAN is 3SL99A.
Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries.
Use the form below and try to retrieve all employee data from the employees table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
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


Activities
Firefox Web Browser
Sep 2 15:06
WebGoat
SQL injection (intro) 5:
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/9
Import bookmarks...
Getting Started
club registration
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Try It! Numeric SQL injection
The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
Using the two Input Fields below, try to retrieve all the data from the users table.
Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.
Login_Count:
User_Id:
Get Account Info
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, youarethesweakestlink, 673834489, MC, , 0,
10323, Grumpy, youarethesweakestlink, 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 = 0 and userid= 0 or 1=1

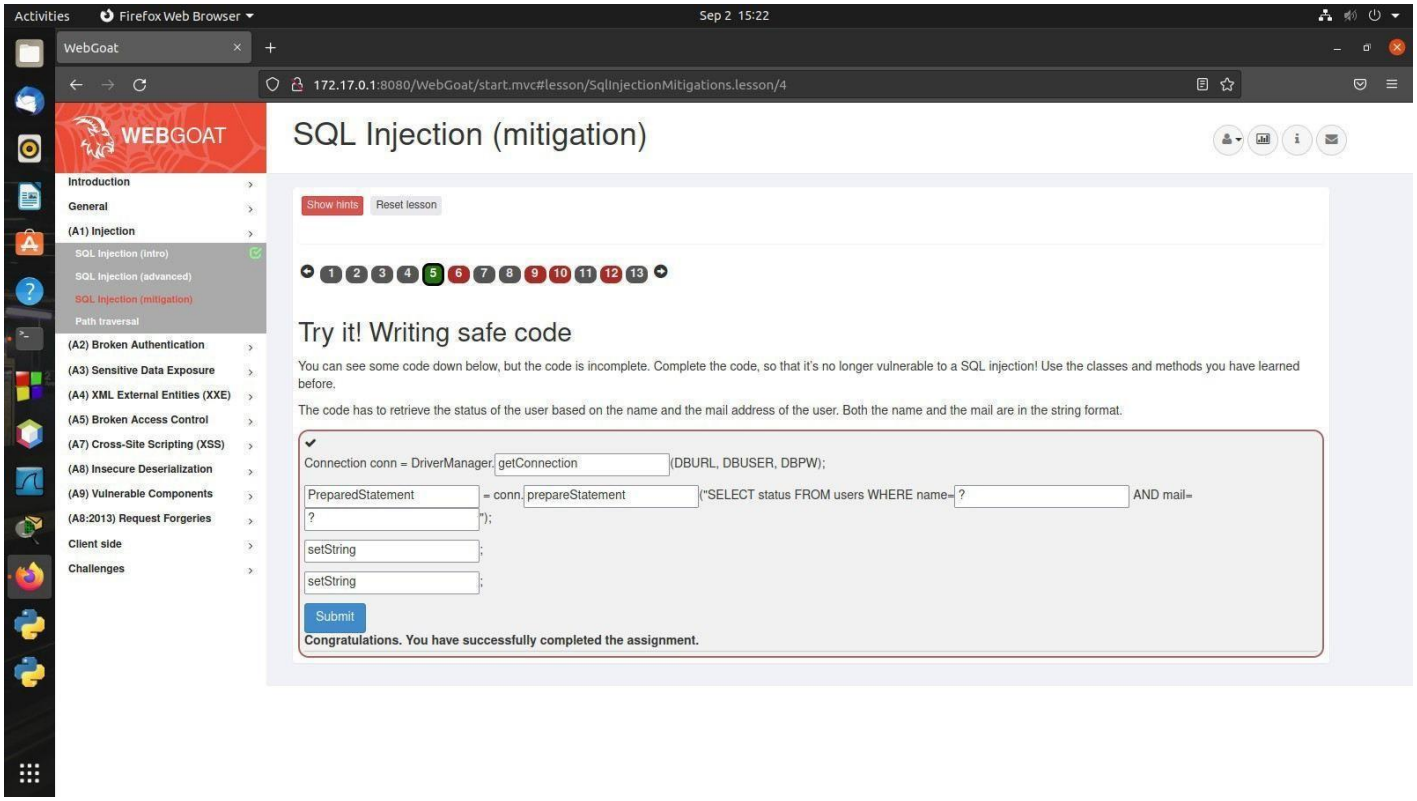

Activities
Firefox Web Browser
Sep 2 15:02
WebGoat
SQL injection (intro) 5:
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/8
Import bookmarks...
Getting Started
club registration
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Try It! String SQL injection
The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '
Smith
or
1 = 1
'
Get Account Info
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, youarethesweakestlink, 673834489, MC, , 0,
10323, Grumpy, youarethesweakestlink, 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 first_name = 'John' and last_name = 'Smith' or '1' = '1'
Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE, which will always evaluate to true, no matter what came before it.

U18INI5600L-ENGINEERING CLINICS V

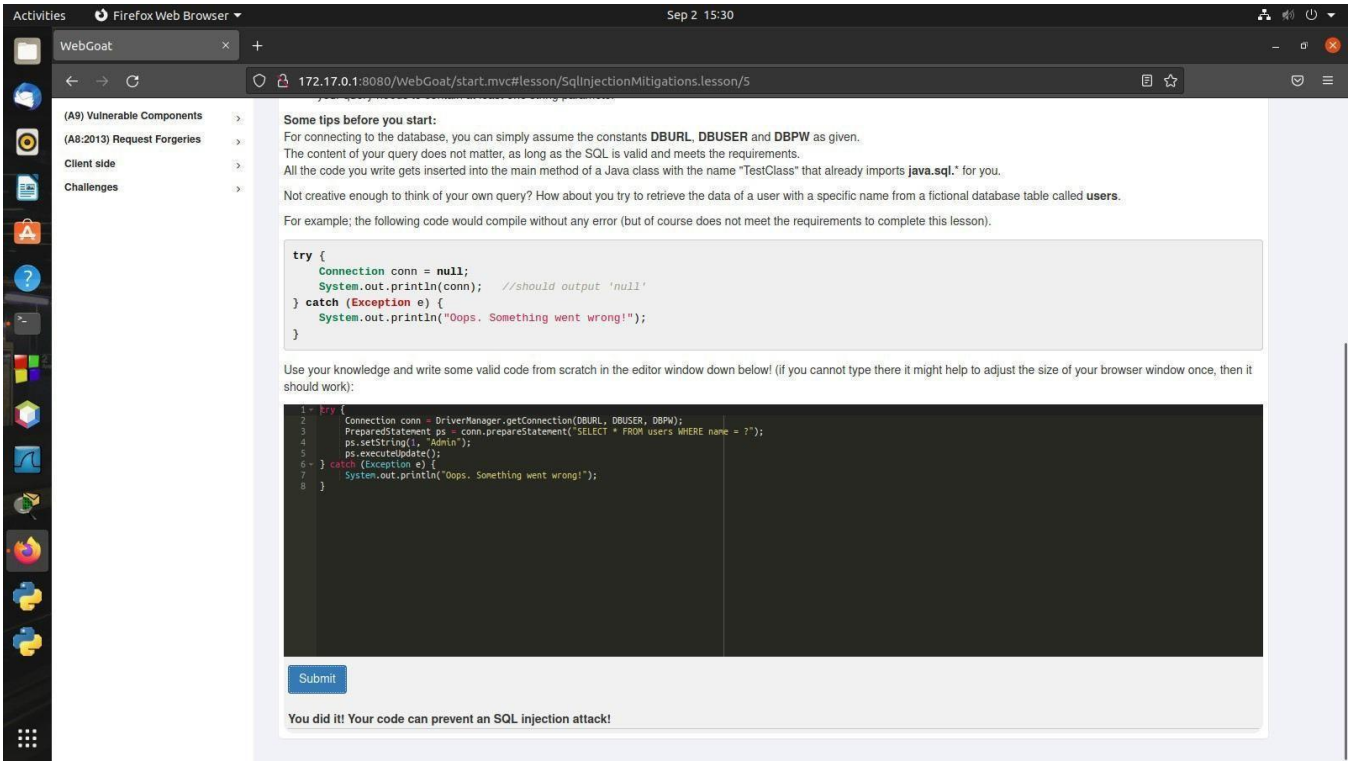
EX.NO 3 : A1 SQL INJECTION-
MITIGATION ASSIGNMENT

Name: Aaron Mathew
Rollno: 20BIS001

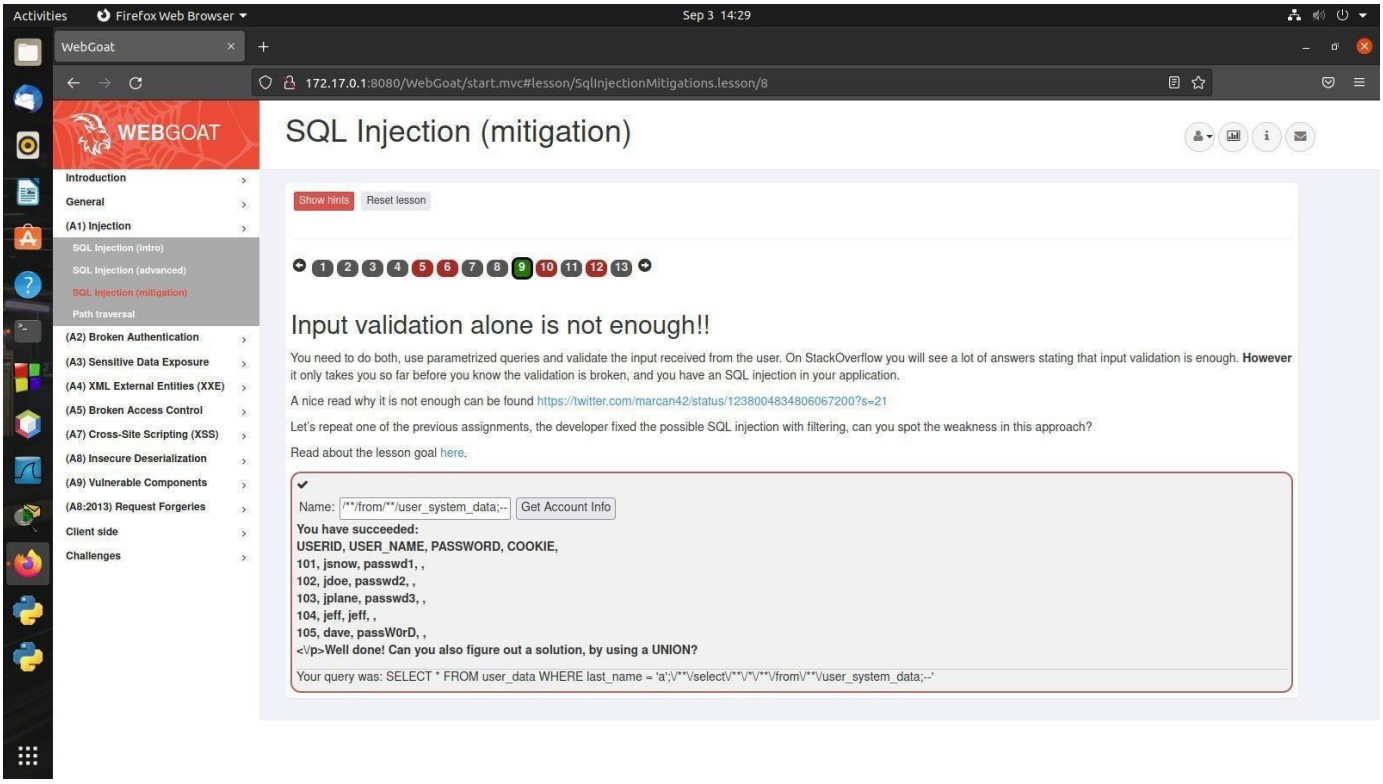
1.



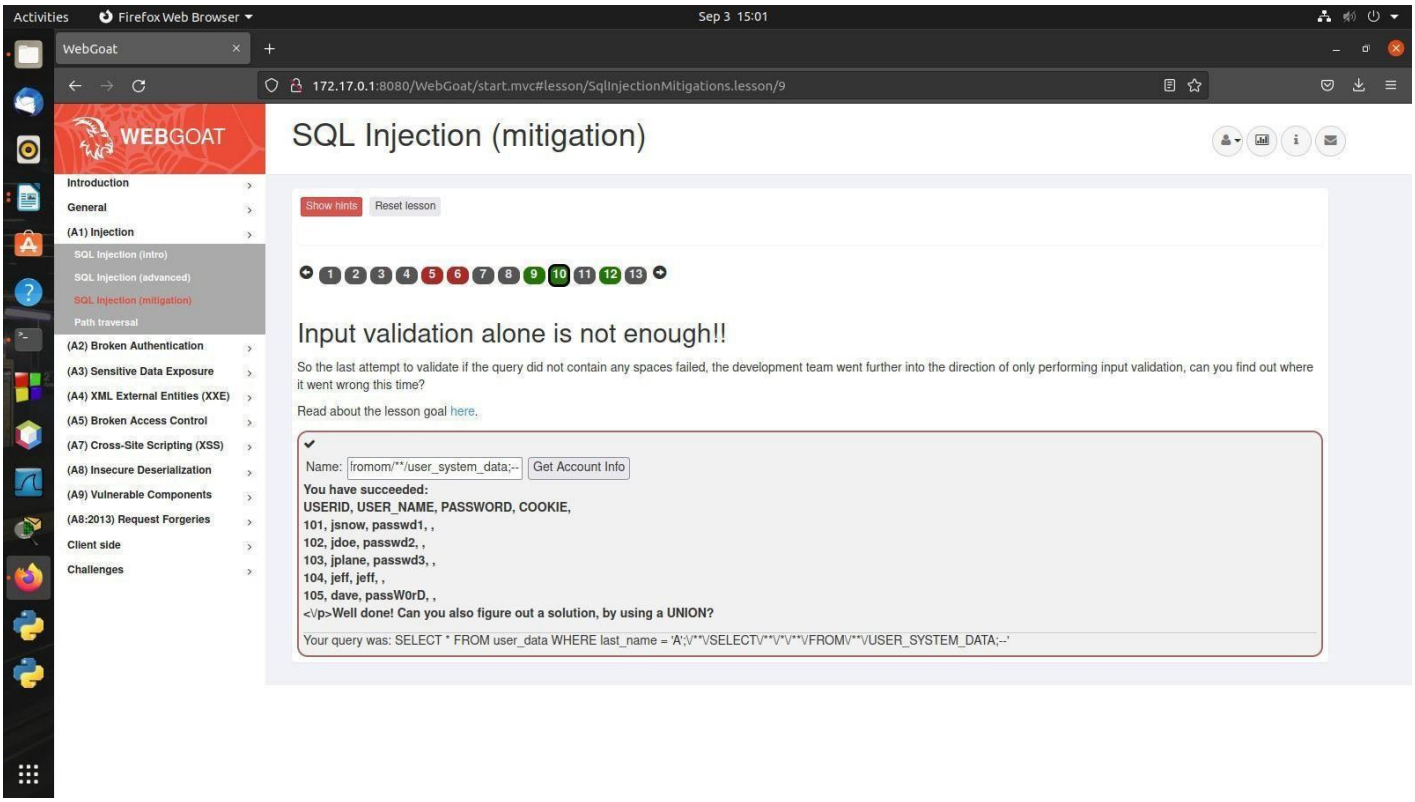
2.


Activities
Firefox Web Browser
Sep 2 15:30
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionMitigations.lesson/5
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Some tips before you start:
For connecting to the database, you can simply assume the constants DBURL, DBUSER and DBPW as given.
The content of your query does not matter, as long as the SQL is valid and meets the requirements.
All the code you write gets inserted into the main method of a Java class with the name "TestClass" that already imports java.sql.* for you.
Not creative enough to think of your own query? How about you try to retrieve the data of a user with a specific name from a fictional database table called users.
For example; the following code would compile without any error (but of course does not meet the requirements to complete this lesson).
try {
Connection conn = null;
System.out.println(conn); //should output 'null'
} catch (Exception e) {
System.out.println("Oops. Something went wrong!");
}
Use your knowledge and write some valid code from scratch in the editor window down below! (if you cannot type there it might help to adjust the size of your browser window once, then it should work):
Submit
You did it! Your code can prevent an SQL injection attack!

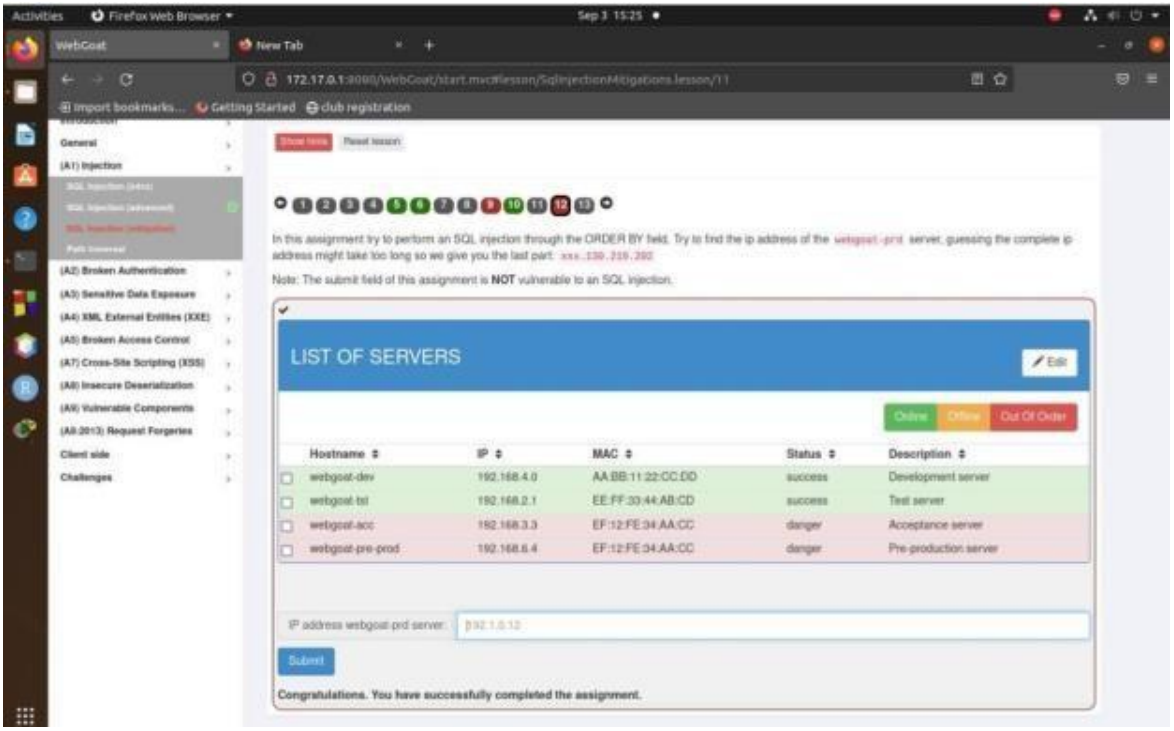
3.

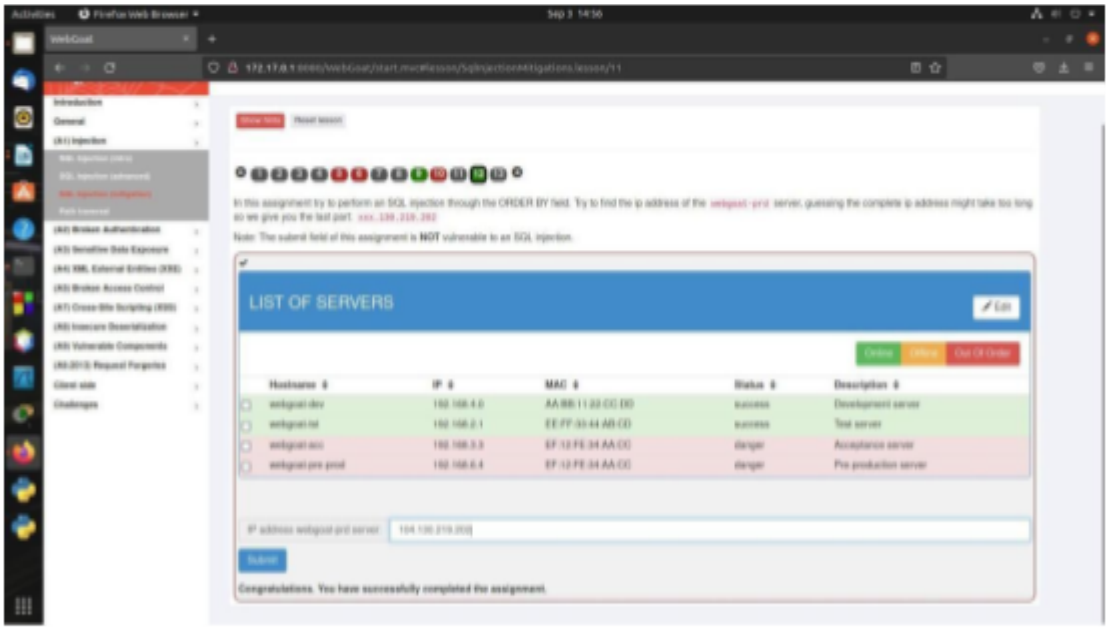

Activities
Firefox Web Browser
Sep 3 14:29
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionMitigations.lesson/8
WEBGOAT
SQL Injection (mitigation)
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12 13
Input validation alone is not enough!!
You need to do both, use parametrized queries and validate the input received from the user. On StackOverflow you will see a lot of answers stating that input validation is enough. However it only takes you so far before you know the validation is broken, and you have an SQL injection in your application.
A nice read why it is not enough can be found https://twitter.com/marcan42/status/1238004834806067200?s=21
Let's repeat one of the previous assignments, the developer fixed the possible SQL injection with filtering, can you spot the weakness in this approach?
Read about the lesson goal here.
Name: /**/from/**/user_system_data;--
Get Account Info
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
<\/p>Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'a';\/**\/select\/**\/*\/**\/from\/**\/user_system_data;--'

4.


Activities
Firefox Web Browser
Sep 3 15:01
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionMitigations.lesson/9
WEBGOAT
SQL Injection (mitigation)
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12 13
Input validation alone is not enough!!
So the last attempt to validate if the query did not contain any spaces failed, the development team went further into the direction of only performing input validation, can you find out where it went wrong this time?
Read about the lesson goal here.
Name: fromom/**/user_system_data;--
Get Account Info
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,
<\/p>Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'A';\/**\/SELECT\/**\/*\/**\/FROM\/**\/USER_SYSTEM_DATA;--'

5.


Activities
Firefox Web Browser
Sep 3 15:25
WebGoat
New Tab
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionMitigations.lesson/11
Import bookmarks...
Getting Started
club registration
Show hints
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12 13
In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the webgoat-prd server, guessing the complete ip address might take too long so we give you the last part: xxx.130.219.202
Note: The submit field of this assignment is NOT vulnerable to an SQL injection.
LIST OF SERVERS
Edit
Online
Offline
Out Of Order
Hostname
IP
MAC
Status
Description
webgoat-dev
192.168.4.0
AA:BB:11:22:CC:DD
success
Development server
webgoat-tst
192.168.2.1
EE:FF:33:44:AB:CD
success
Test server
webgoat-acc
192.168.3.3
EF:12:FE:34:AA:CC
danger
Acceptance server
webgoat-pre-prod
192.168.6.4
EF:12:FE:34:AA:CC
danger
Pre-production server
IP address webgoat-prd server:
Submit
Congratulations. You have successfully completed the assignment.


LIST OF SERVERS
Edit
Hostname
IP
MAC
Risks
Description
Submit
Congratulations. You have successfully completed the assignment.

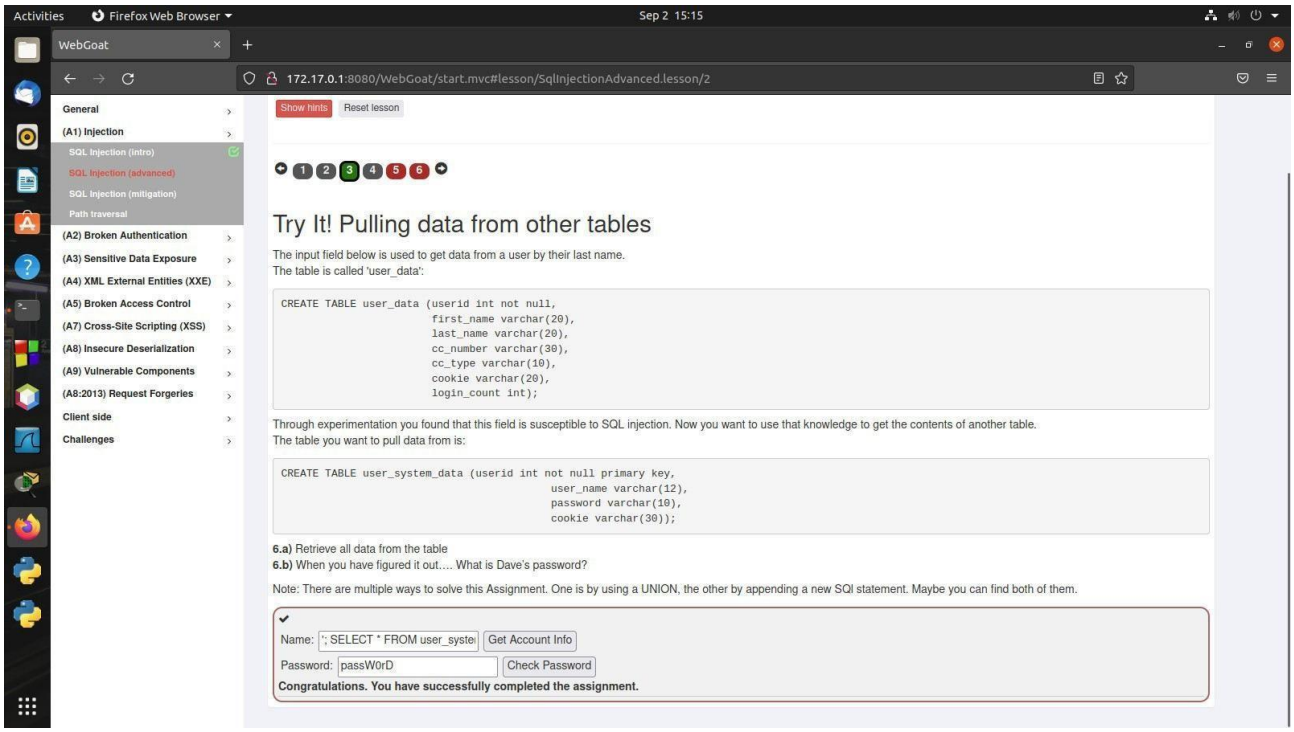
U18INI5600L-ENGINEERING CLINICS V

EX.NO : A1 SQL INJECTION- ADVANCED

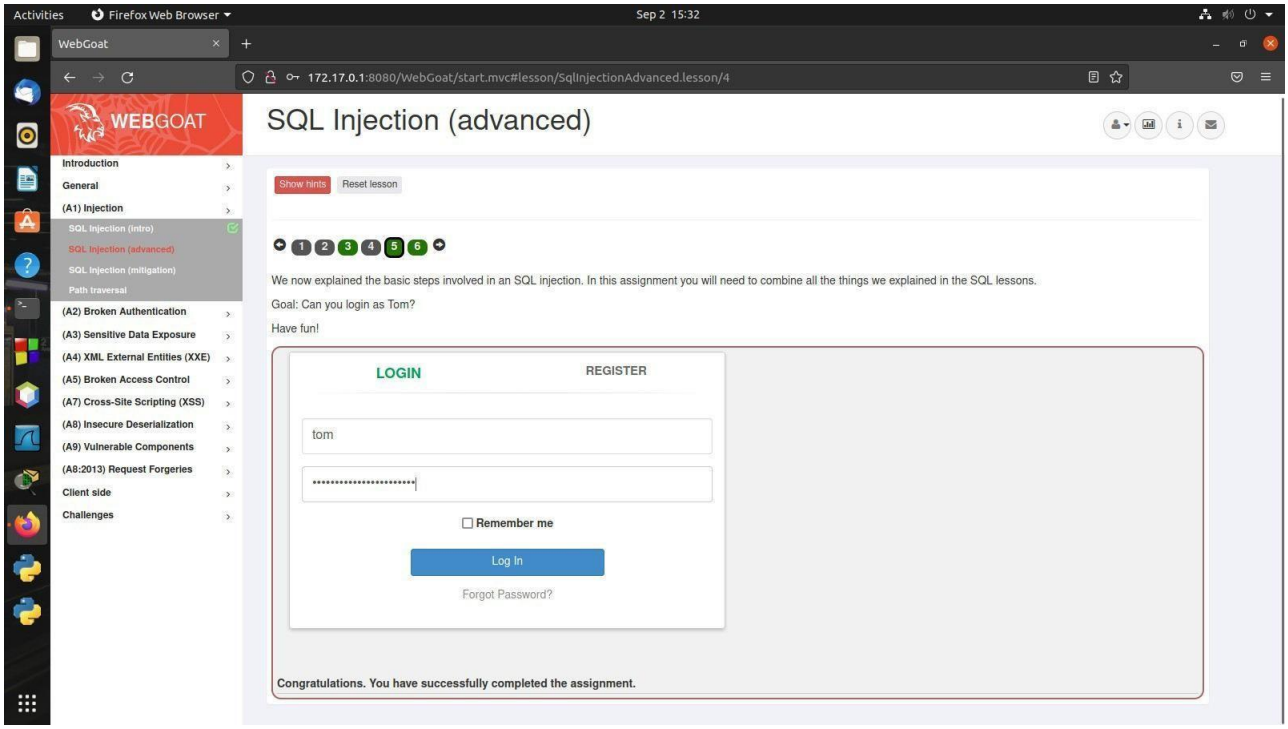
ASSIGNMENT 2

Name: Aaron Mathew
Rollno: 20BIS001

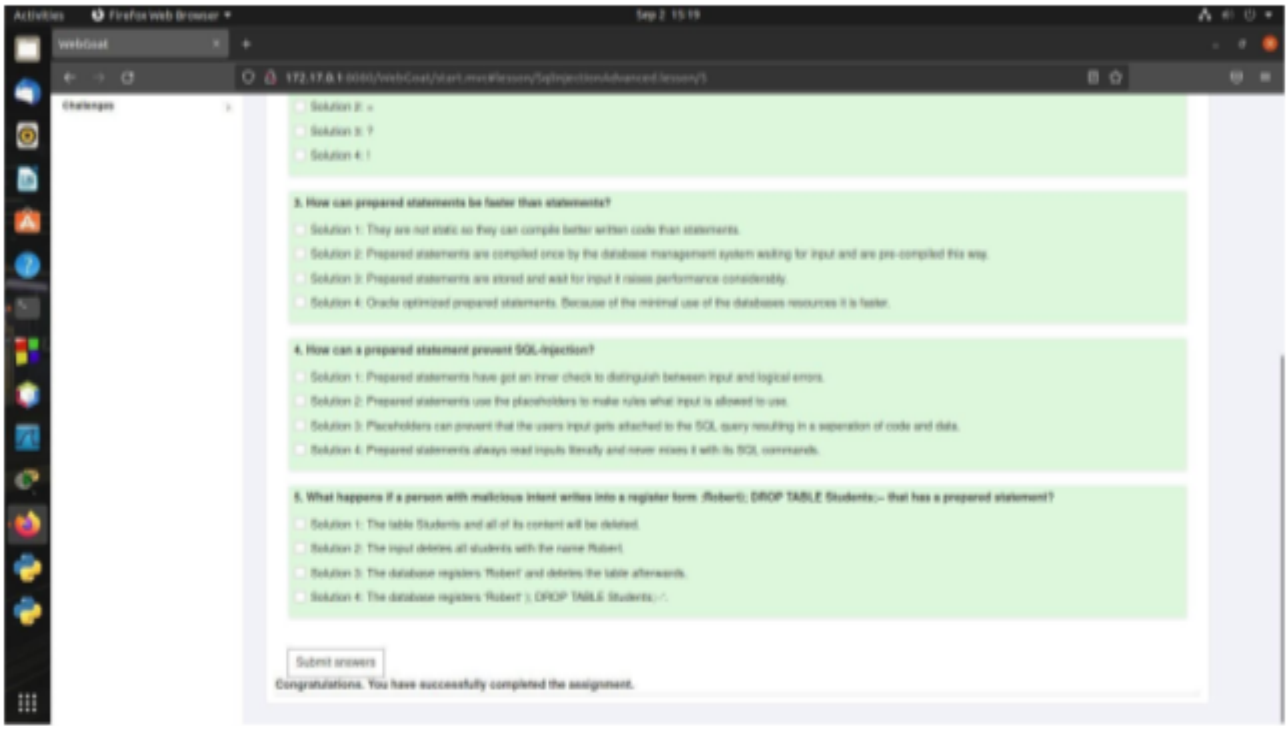
1.



2.


Activities
Firefox Web Browser
Sep 2 15:32
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionAdvanced.lesson/4
WEBGOAT
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
SQL Injection (advanced)
Show hints
Reset lesson
1 2 3 4 5 6
We now explained the basic steps involved in an SQL injection. In this assignment you will need to combine all the things we explained in the SQL lessons.
Goal: Can you login as Tom?
Have fun!
LOGIN
REGISTER
tom
Remember me
Log In
Forgot Password?
Congratulations. You have successfully completed the assignment.

3.

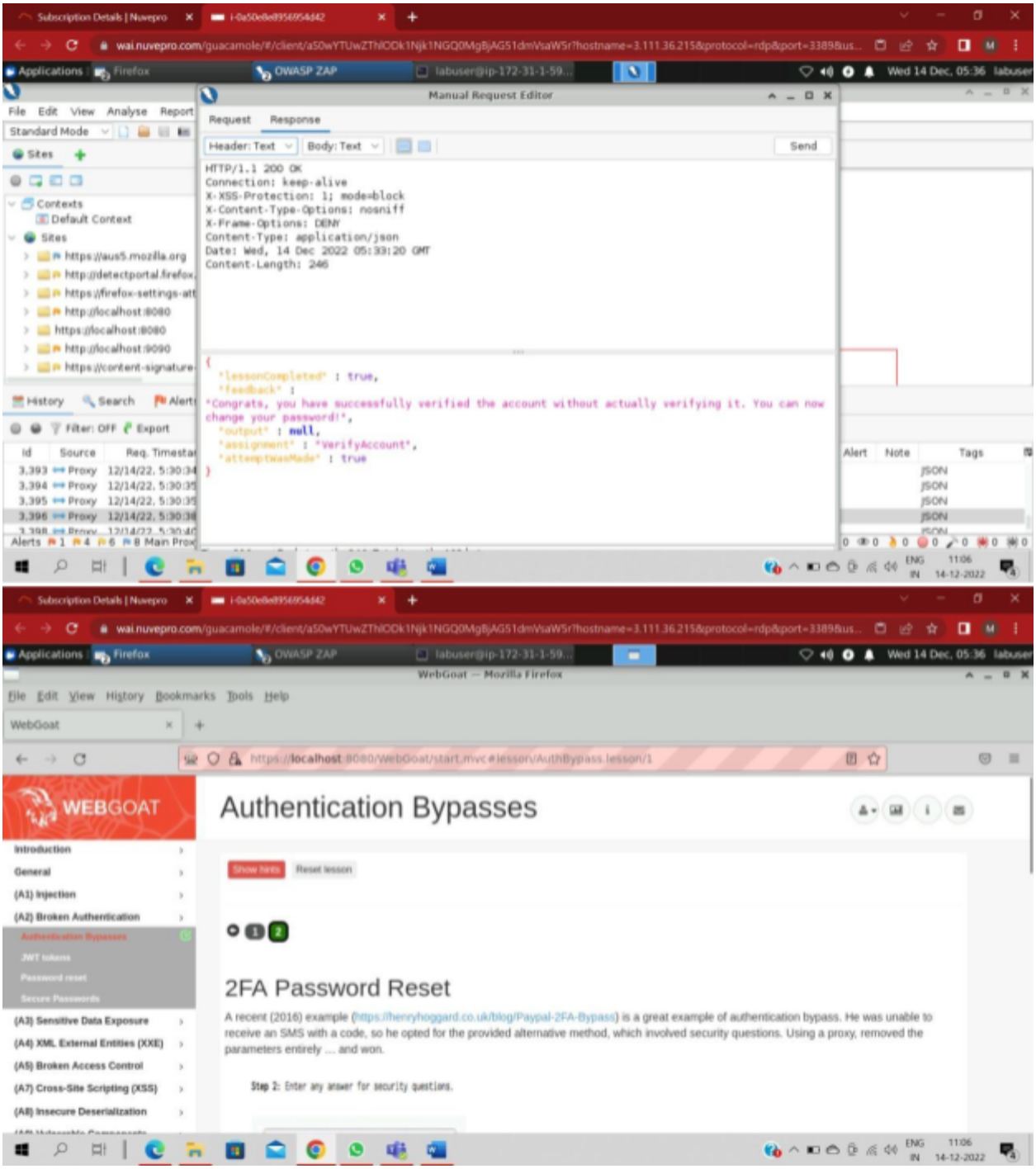

Activities
Firefox Web Browser
Sep 2 15:19
WebGoat
172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionAdvanced.lesson/1
Challenges
Solution 2: x
Solution 3: ?
Solution 4: !
3. How can prepared statements be faster than statements?
Solution 1: They are not static so they can compile better written code than statements.
Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.
Solution 3: Prepared statements are stored and wait for input it raises performance considerably.
Solution 4: Oracle optimized prepared statements. Because of the minimal use of the databases resources it is faster.
4. How can a prepared statement prevent SQL-injection?
Solution 1: Prepared statements have got an inner check to distinguish between input and logical errors.
Solution 2: Prepared statements use the placeholders to make rules what input is allowed to use.
Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a seperation of code and data.
Solution 4: Prepared statements always read inputs literally and never mixes it with its SQL commands.
5. What happens if a person with malicious intent writes into a register form: Robert'); DROP TABLE Students;-- that has a prepared statement?
Solution 1: The table Students and all of its content will be deleted.
Solution 2: The input deletes all students with the name Robert.
Solution 3: The database registers 'Robert' and deletes the table afterwards.
Solution 4: The database registers 'Robert '); DROP TABLE Students;--'.
Submit answers
Congratulations. You have successfully completed the assignment.

OWASP Broken Authentication

Aaron Mathew

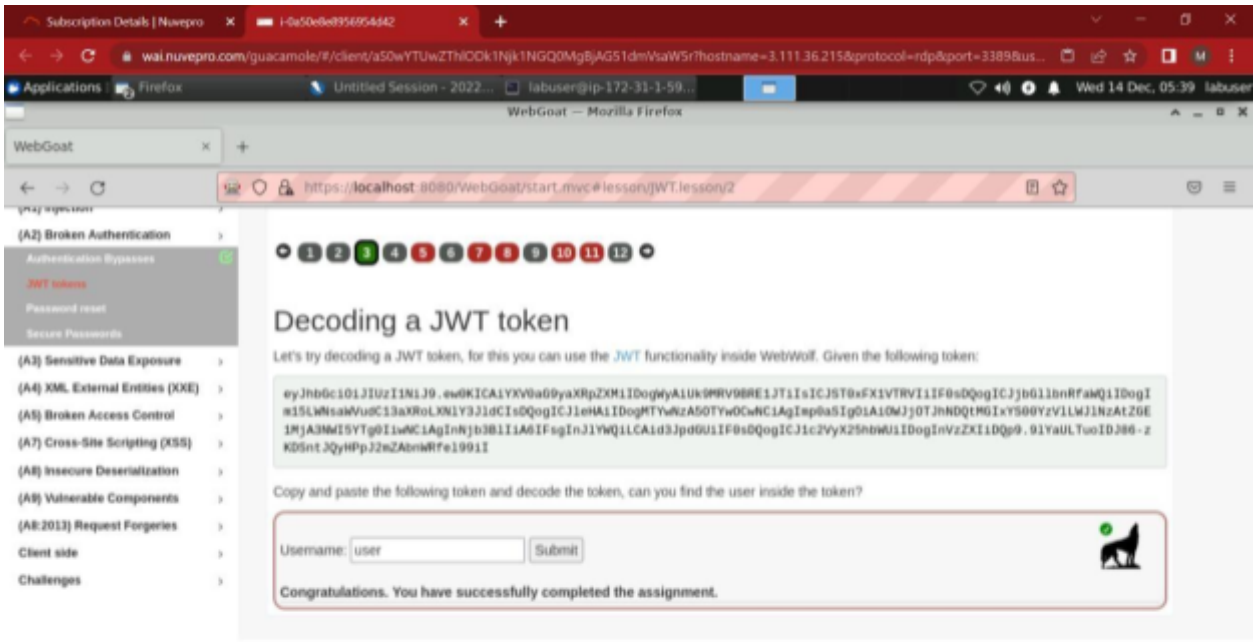
20BIS001

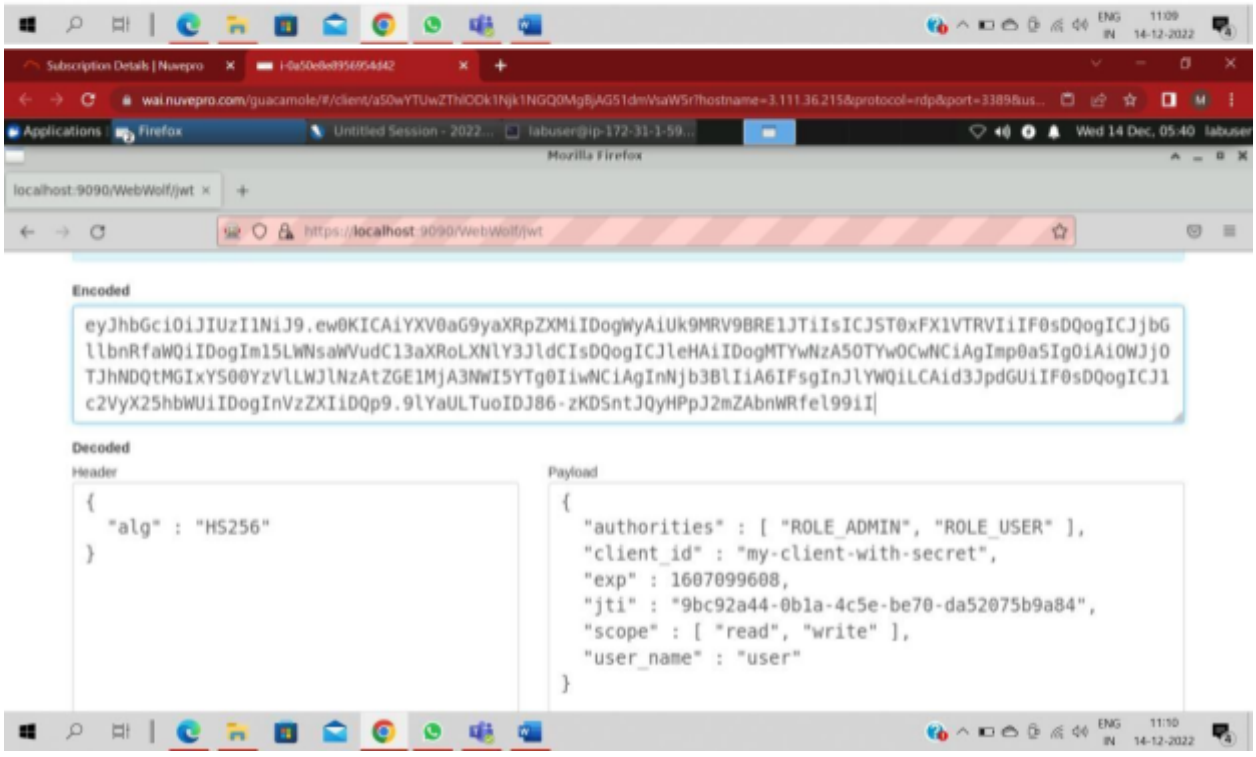
Authentication Bypass:


Manual Request Editor
Request
Response
Send
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 14 Dec 2022 05:33:20 GMT
Content-Length: 246
"lessonCompleted" : true,
"feedback" :
"Congrats, you have successfully verified the account without actually verifying it. You can now change your password!",
"output" : null,
"assignment" : "VerifyAccount",
"attemptWasMade" : true
WebGoat — Mozilla Firefox
WEBGOAT
Authentication Bypasses
2FA Password Reset
A recent (2016) example (https://henryhoggard.co.uk/blog/Paypal-2FA-Bypass) is a great example of authentication bypass. He was unable to receive an SMS with a code, so he opted for the provided alternative method, which involved security questions. Using a proxy, removed the parameters entirely ... and won.
Step 2: Enter any answer for security questions.

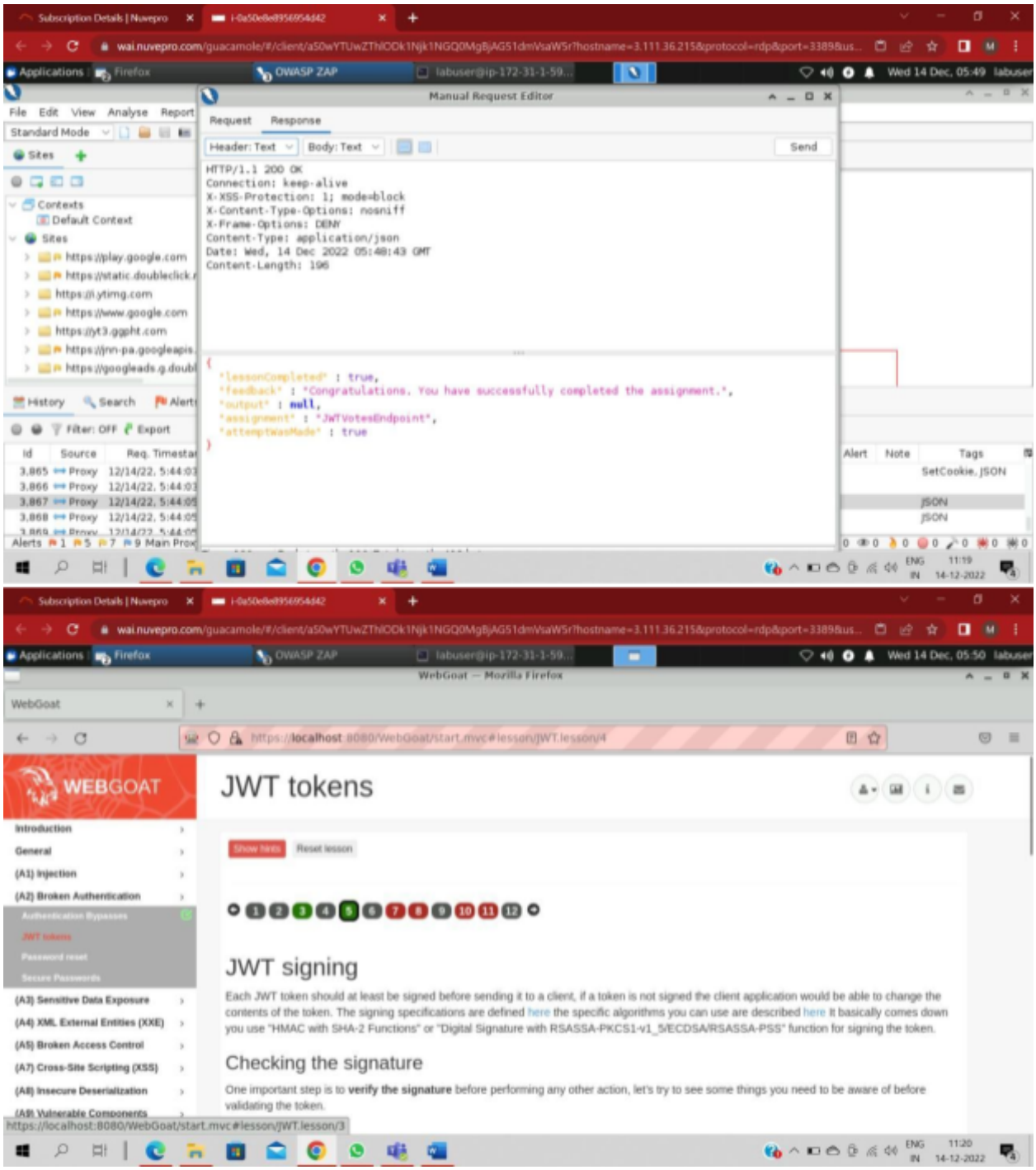
JWT Tokens:

Decoding JWT Tokens:

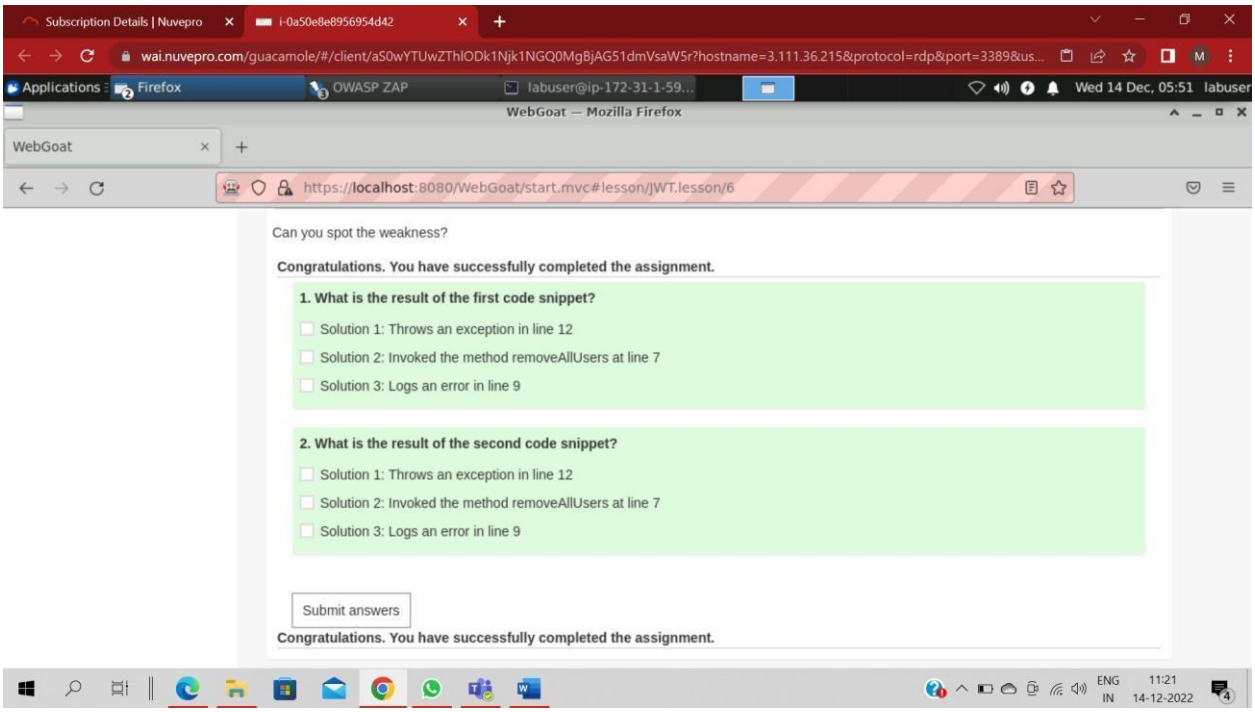




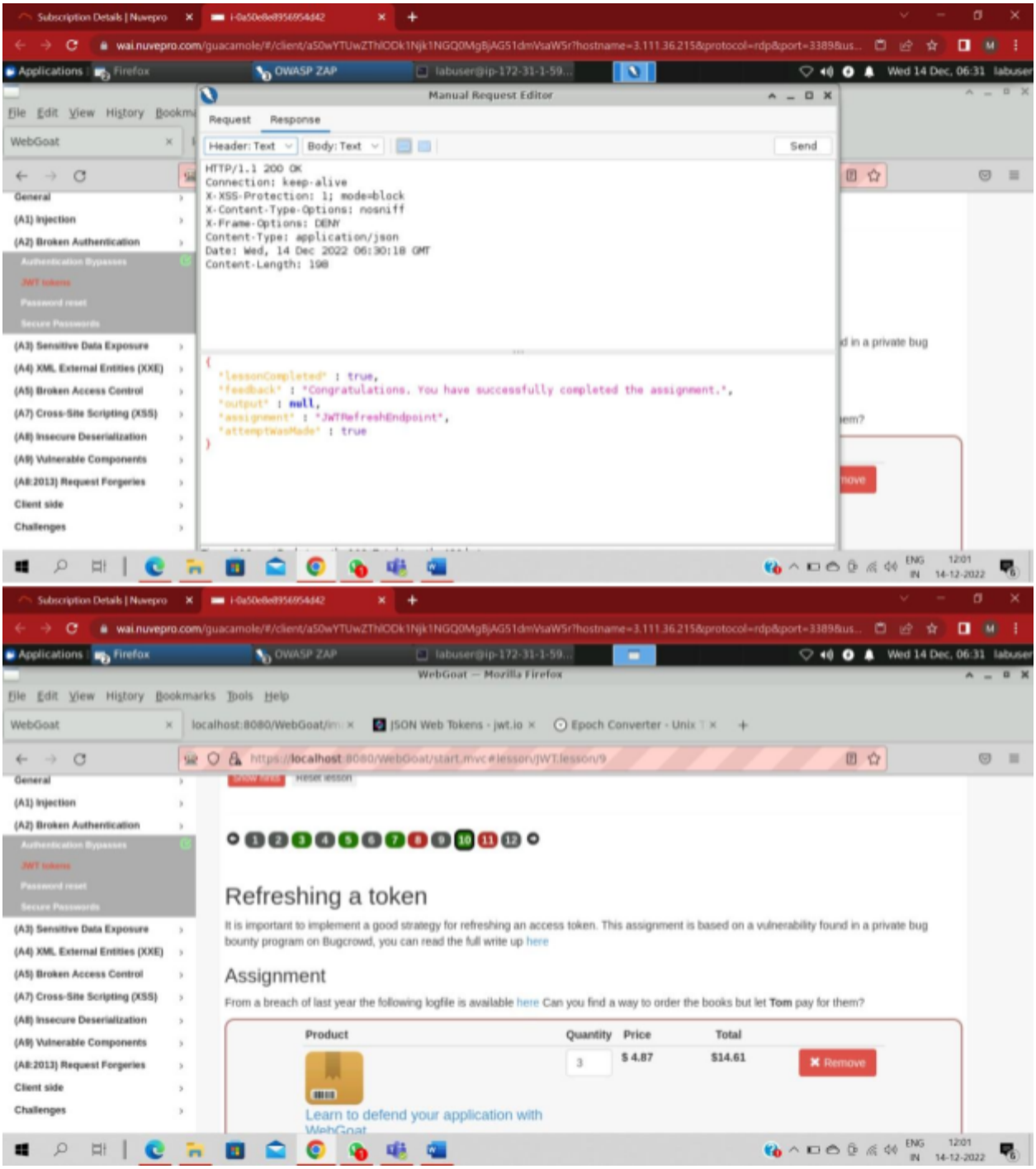
JWT Signing:


Manual Request Editor
Request
Response
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 14 Dec 2022 05:48:43 GMT
Content-Length: 196
{
"lessonCompleted" : true,
"feedback" : "Congratulations. You have successfully completed the assignment.",
"output" : null,
"assignment" : "JWTVotesEndpoint",
"attemptWasMade" : true
}
WEBGOAT
JWT tokens
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
Show hints
Reset lesson
JWT signing
Each JWT token should at least be signed before sending it to a client, if a token is not signed the client application would be able to change the contents of the token. The signing specifications are defined here the specific algorithms you can use are described here It basically comes down you use "HMAC with SHA-2 Functions" or "Digital Signature with RSASSA-PKCS1-v1_5/ECDSA/RSASSA-PSS" function for signing the token.
Checking the signature
One important step is to verify the signature before performing any other action, let's try to see some things you need to be aware of before validating the token.

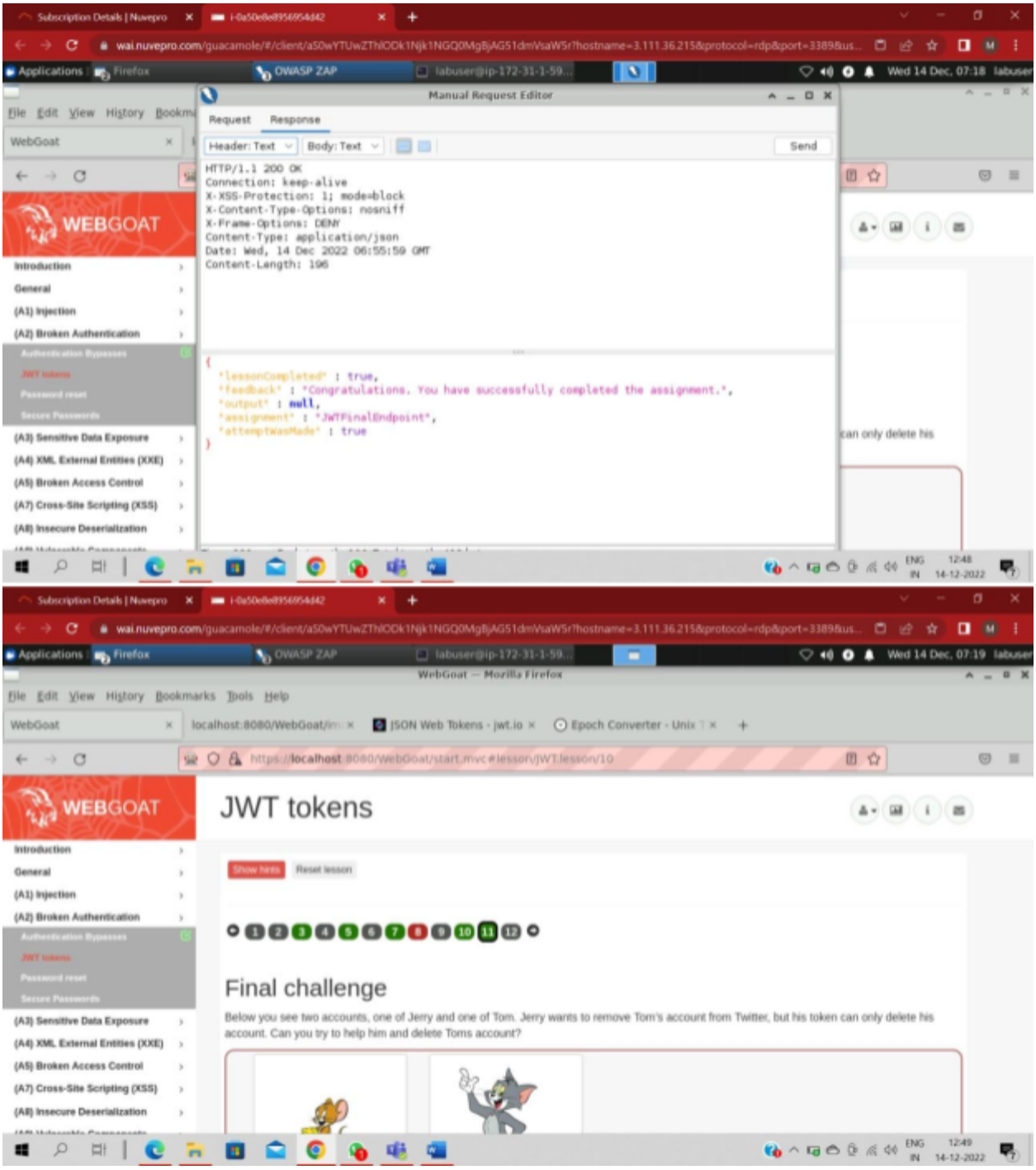
Code Review:



Refreshing a Token:

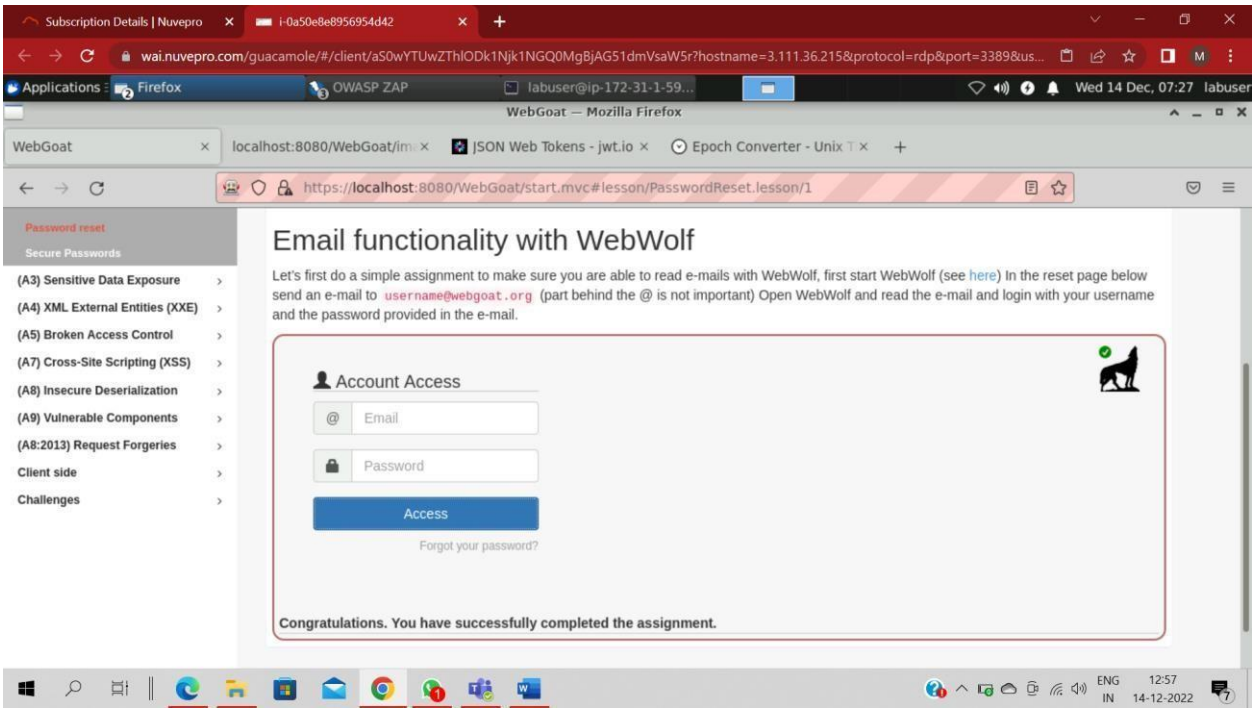


Final Challenge:

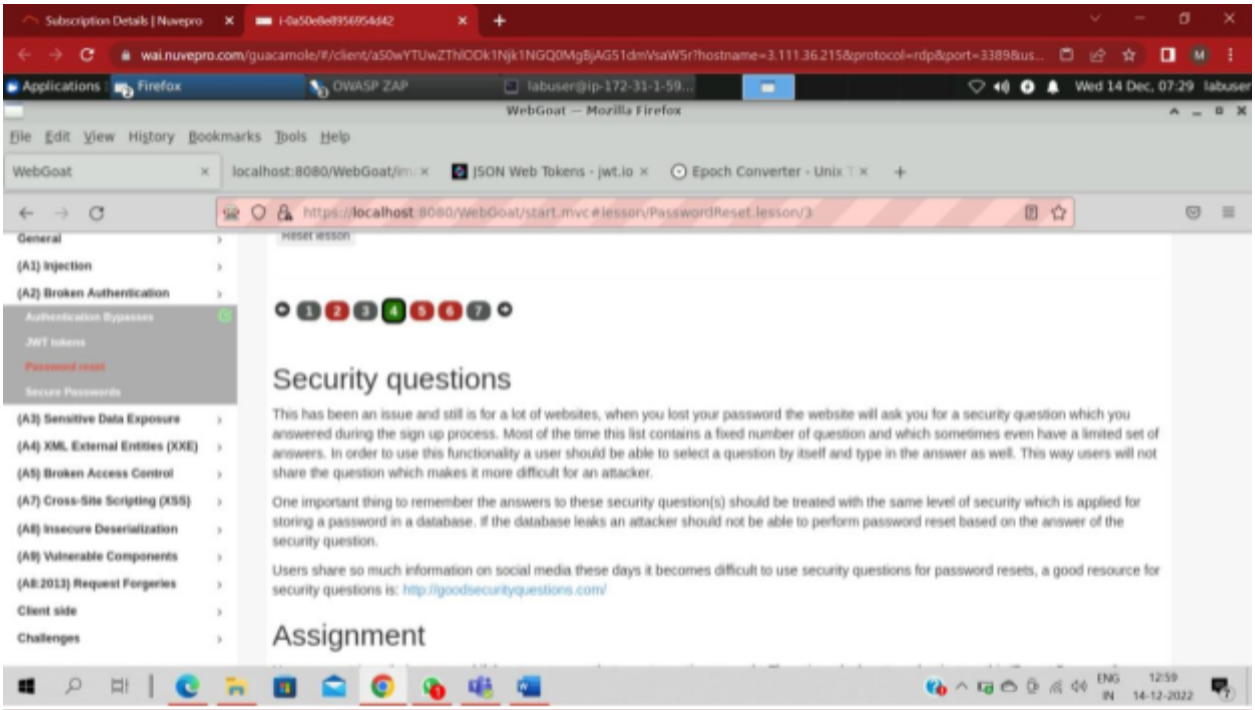


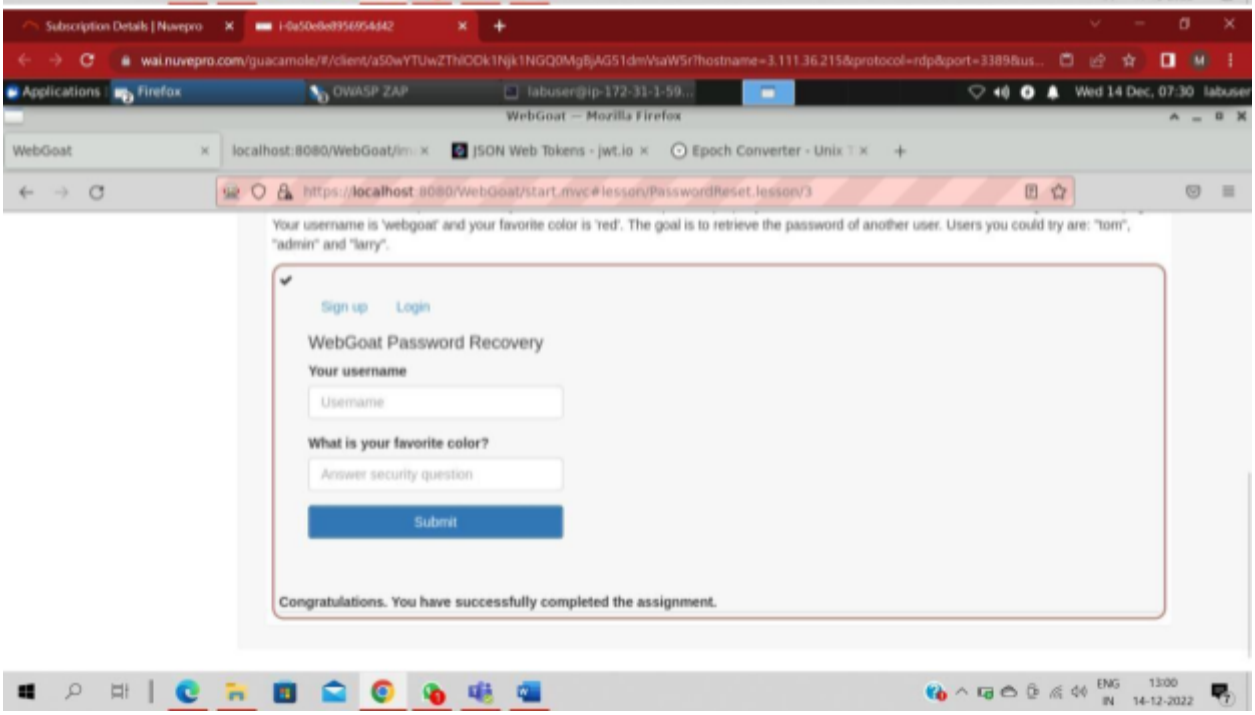
Password Reset:

Email Functionality with Webwolf:

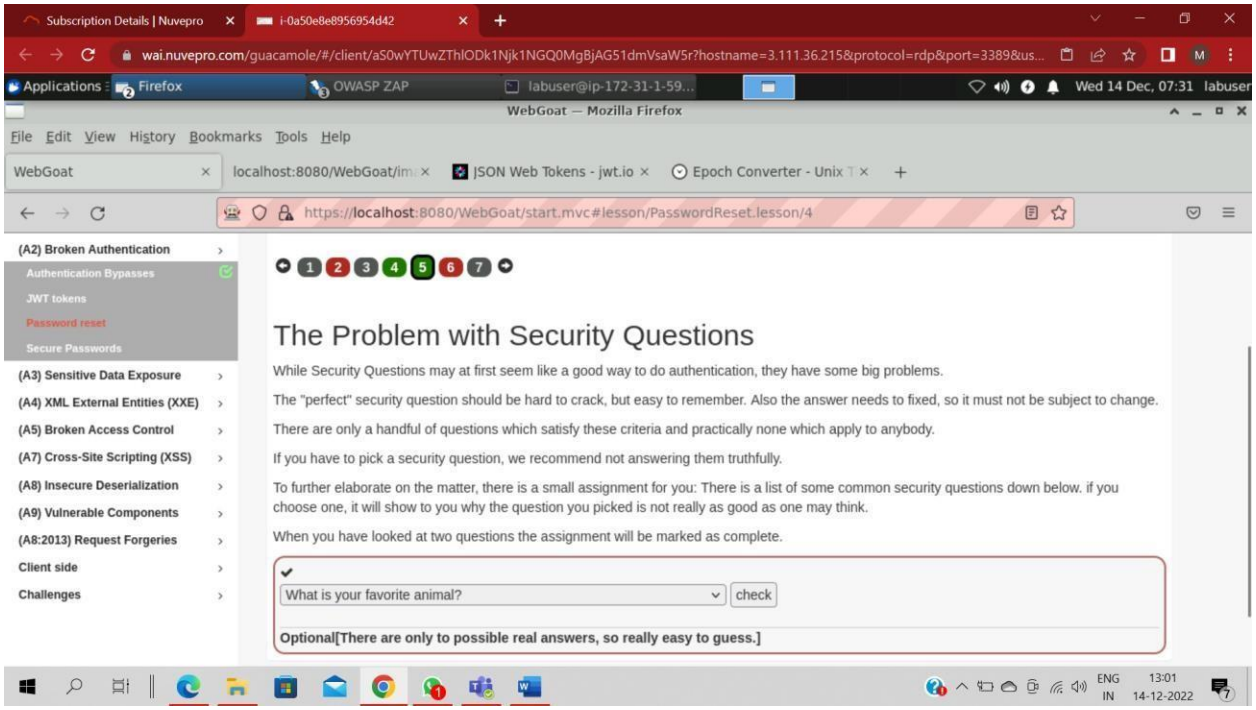


Security Questions:

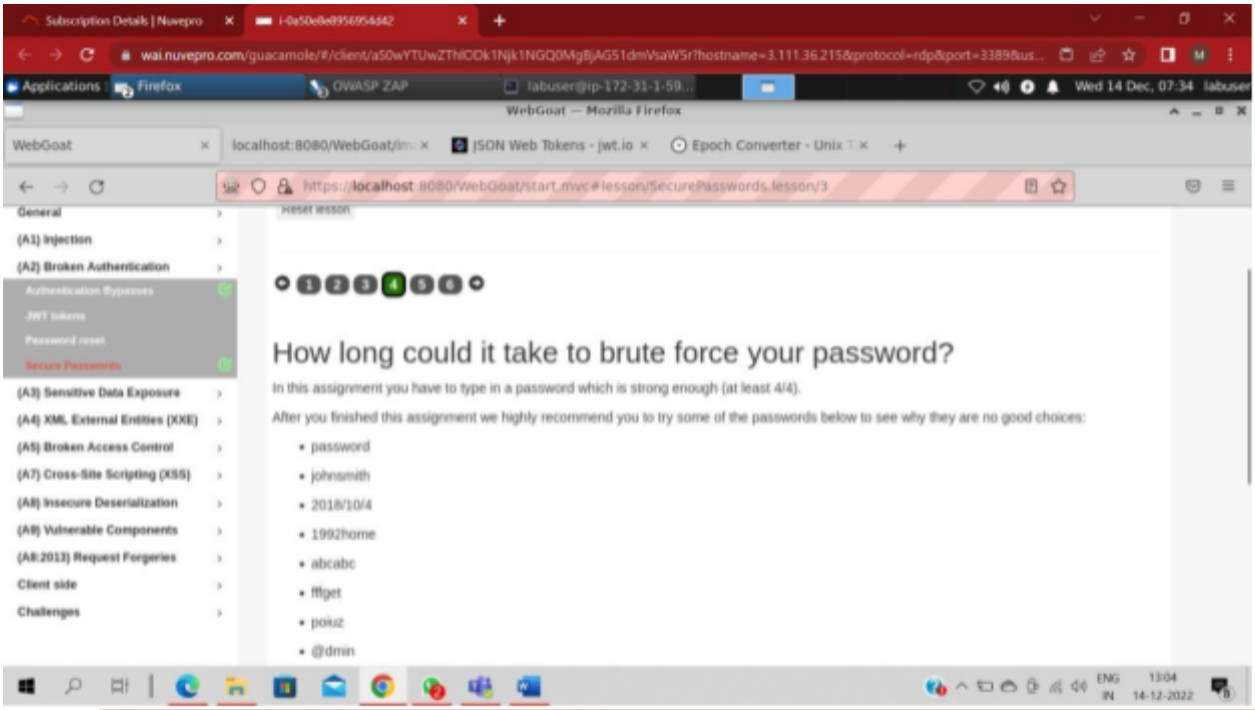


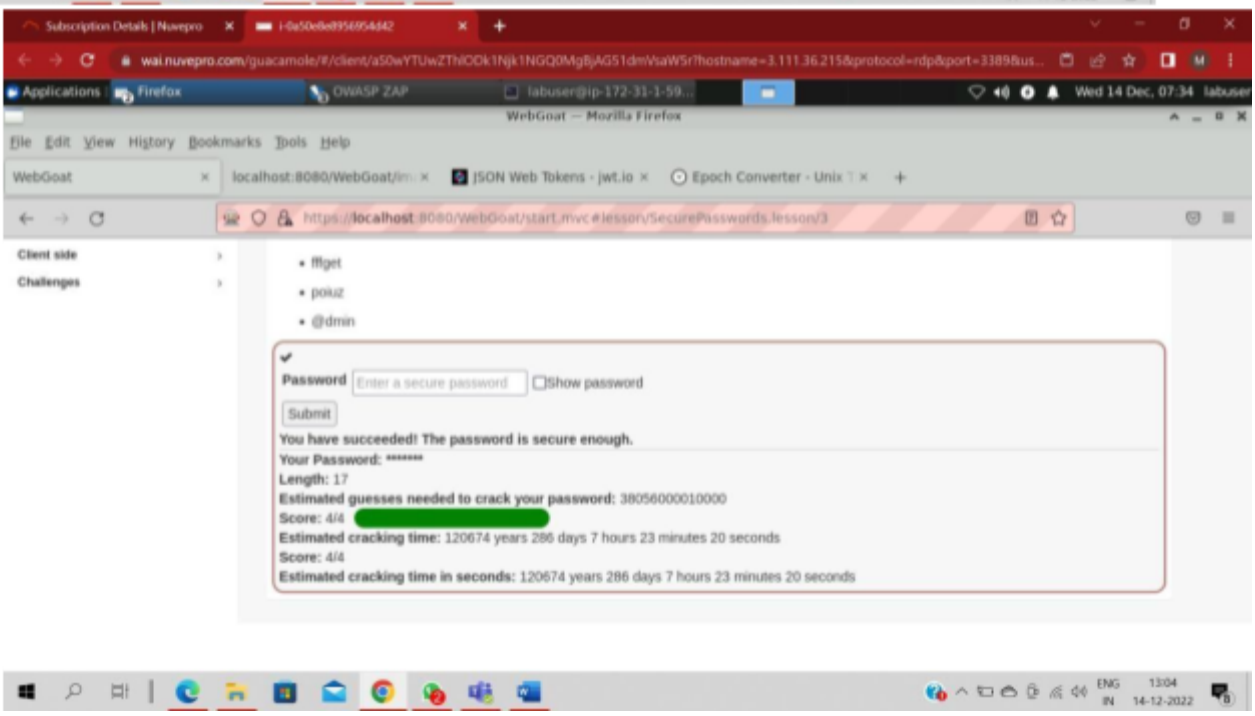


The problem with security questions:



Secure Passwords:


WebGoat — Mozilla Firefox
WebGoat
localhost:8080/WebGoat/lm
JSON Web Tokens - jwt.io
Epoch Converter - Unix T
https://localhost:8080/WebGoat/start.mvc#lesson/SecurePasswords.lesson/3
General
(A1) Injection
(A2) Broken Authentication
Authentication Bypasses
JWT tokens
Password reset
Secure Passwords
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
How long could it take to brute force your password?
In this assignment you have to type in a password which is strong enough (at least 4/4).
After you finished this assignment we highly recommend you to try some of the passwords below to see why they are no good choices:
password
johnsmith
2018/10/4
1992home
abcabc
ffget
poiuz
@dmin

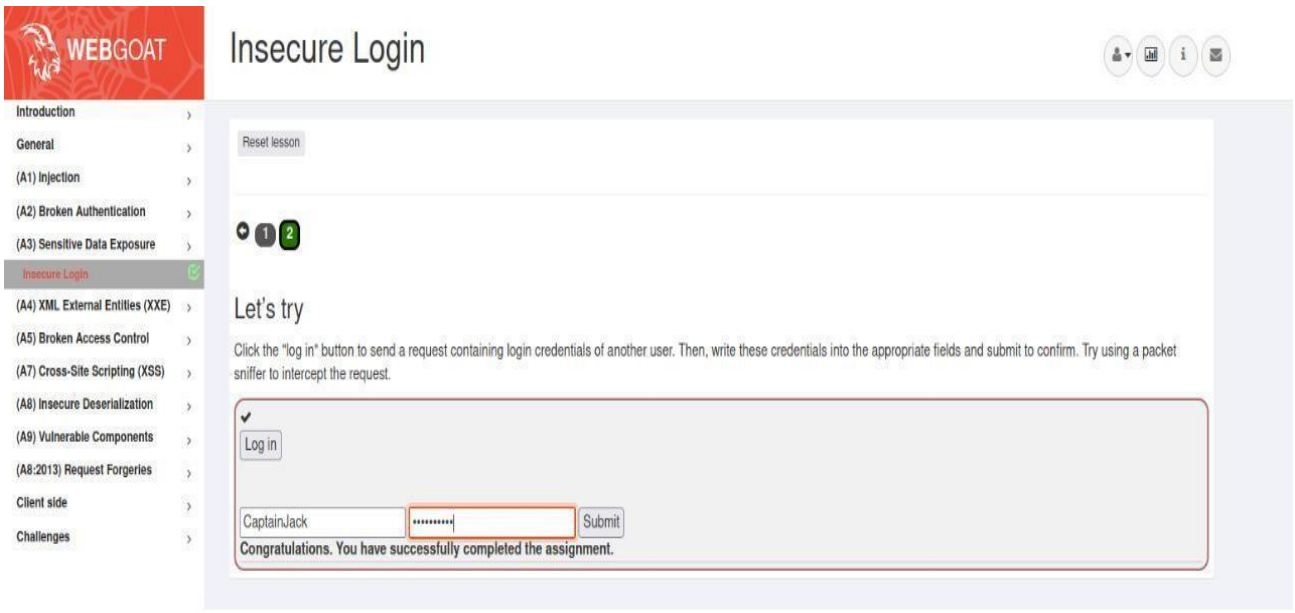

WebGoat — Mozilla Firefox
File Edit View History Bookmarks Tools Help
WebGoat
localhost:8080/WebGoat/lm
JSON Web Tokens - jwt.io
Epoch Converter - Unix T
https://localhost:8080/WebGoat/start.mvc#lesson/SecurePasswords.lesson/3
Client side
Challenges
ffget
poiuz
@dmin
Password
Enter a secure password
Show password
Submit
You have succeeded! The password is secure enough.
Your Password: *******
Length: 17
Estimated guesses needed to crack your password: 38056000010000
Score: 4/4
Estimated cracking time: 120674 years 286 days 7 hours 23 minutes 20 seconds
Score: 4/4
Estimated cracking time in seconds: 120674 years 286 days 7 hours 23 minutes 20 seconds

ENGINEERING CLINICS -U18INI5600

A3-SENSITIVE DATA EXPOSURE

NAME: Aaron Mathew

ROLL.NO:20BIS001

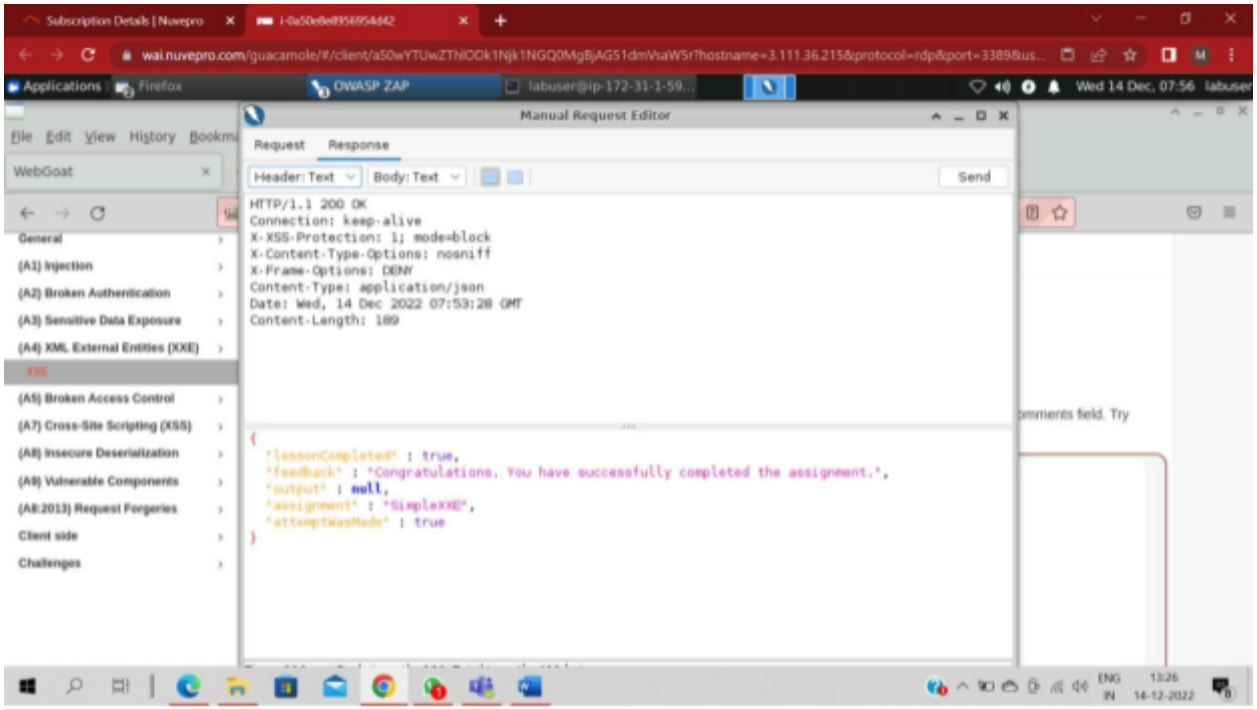


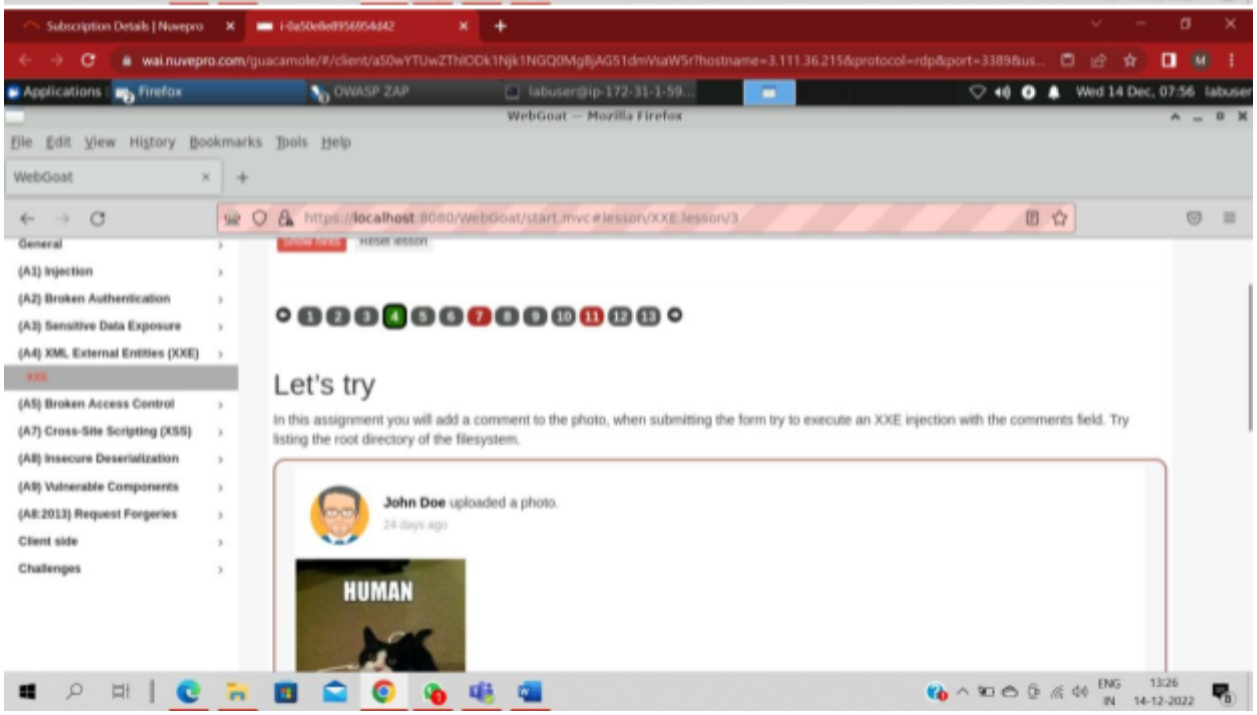
A4 – XML External Entities(XXE)

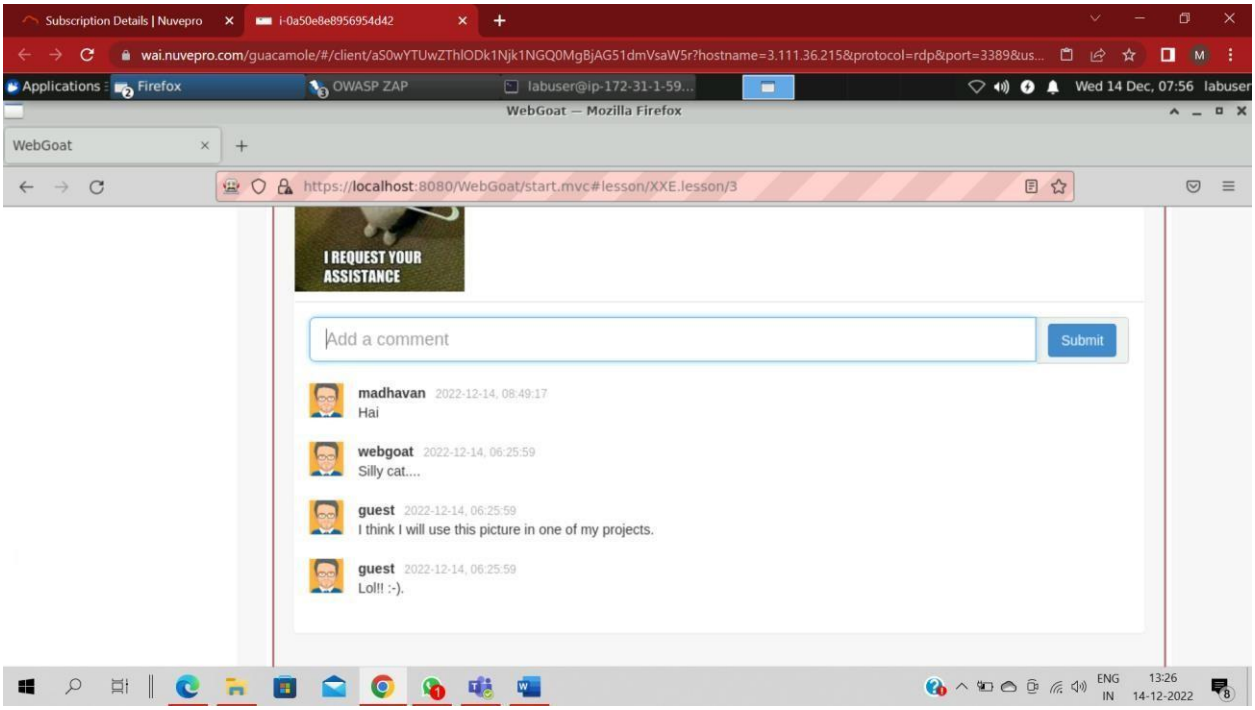
Aaron Mathew

20BIS001

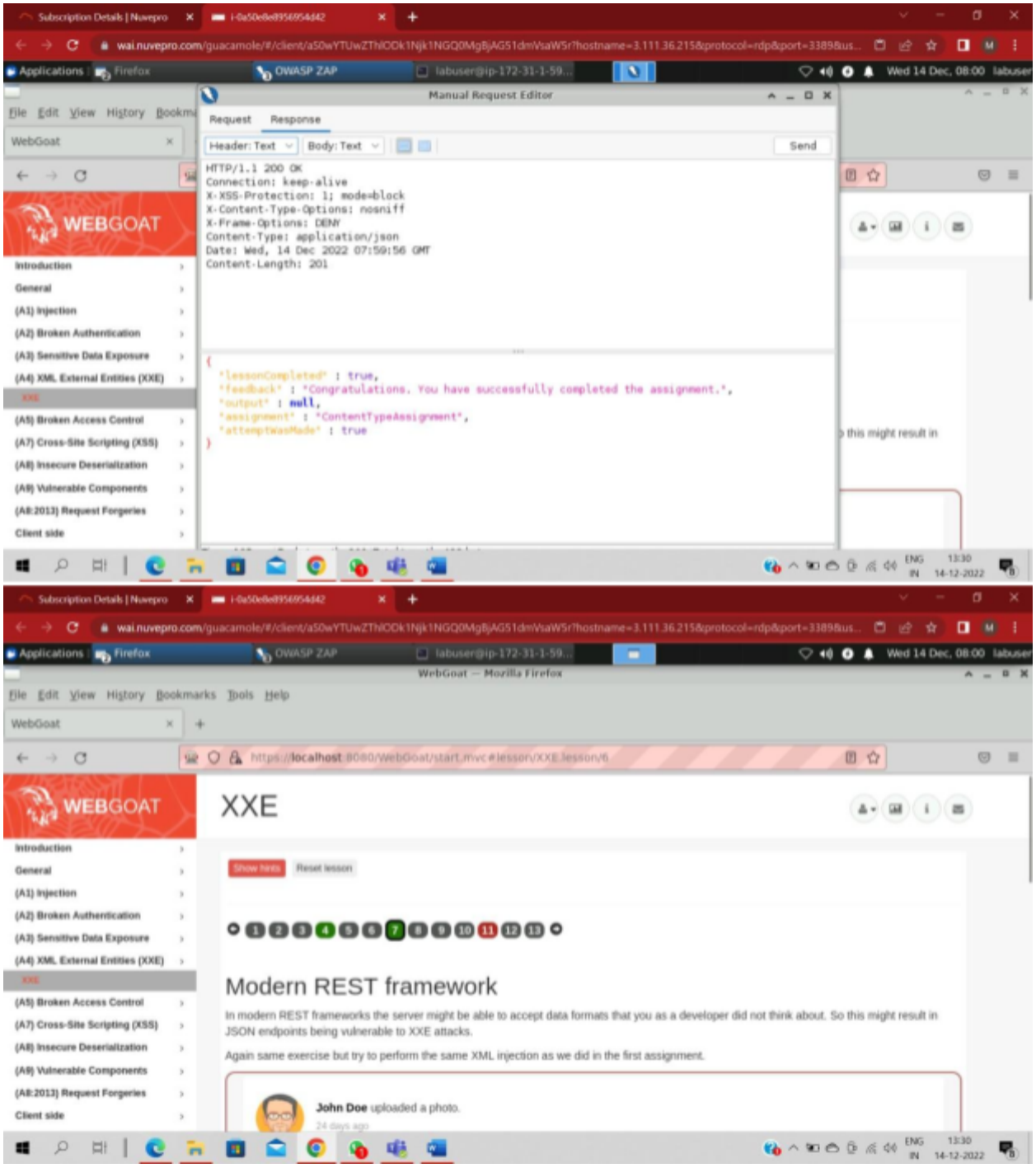
XXE:


Manual Request Editor
Request
Response
Header: Text
Body: Text
Send
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 14 Dec 2022 07:53:28 GMT
Content-Length: 189
{
"lessonCompleted" : true,
"feedback" : "Congratulations. You have successfully completed the assignment.",
"output" : null,
"assignment" : "SimpleXXE",
"attemptWasMade" : true
}

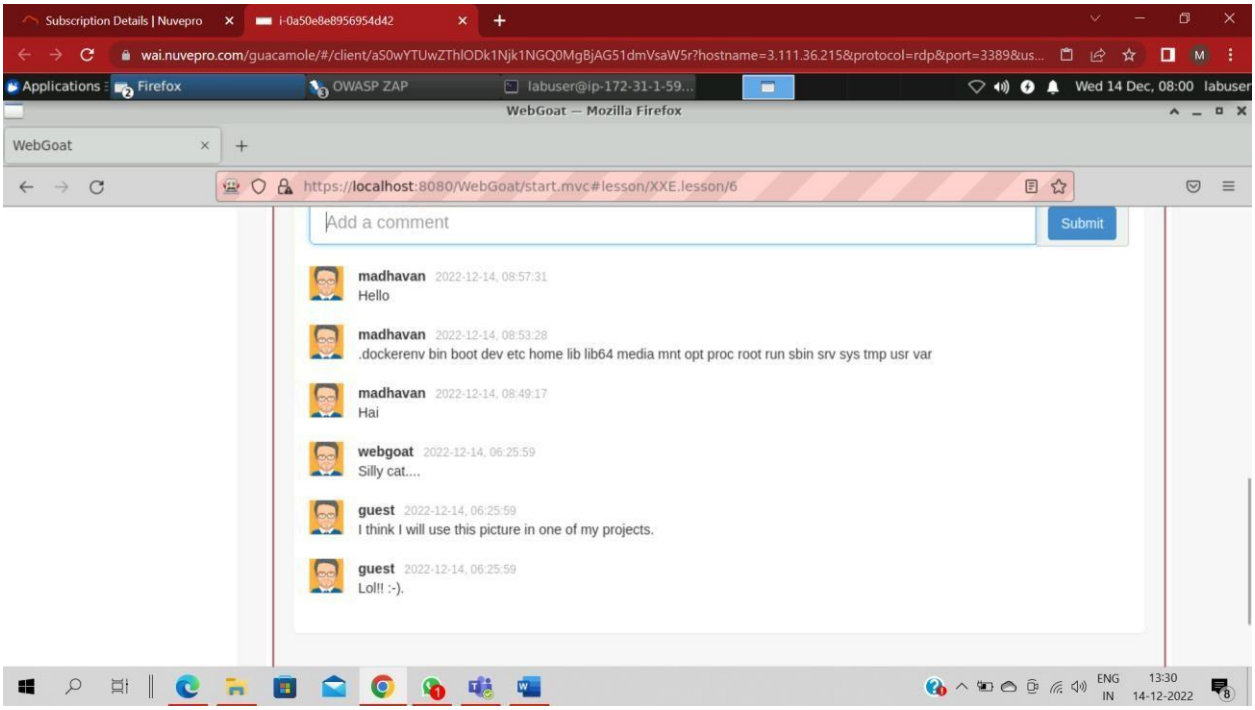

Let's try
In this assignment you will add a comment to the photo, when submitting the form try to execute an XXE injection with the comments field. Try listing the root directory of the filesystem.
John Doe uploaded a photo.
HUMAN



Modern REST Framework:


Manual Request Editor
Request
Response
Header: Text
Body: Text
Send
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 14 Dec 2022 07:59:56 GMT
Content-Length: 201
{
"lessonCompleted" : true,
"feedback" : "Congratulations. You have successfully completed the assignment.",
"output" : null,
"assignment" : "ContentTypeAssignment",
"attemptWasMade" : true
}
WEBGOAT
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
XXE
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side

WebGoat — Mozilla Firefox
WEBGOAT
XXE
Show hints
Reset lesson
Modern REST framework
In modern REST frameworks the server might be able to accept data formats that you as a developer did not think about. So this might result in JSON endpoints being vulnerable to XXE attacks.
Again same exercise but try to perform the same XML injection as we did in the first assignment.
John Doe uploaded a photo.
24 days ago
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
XXE
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side

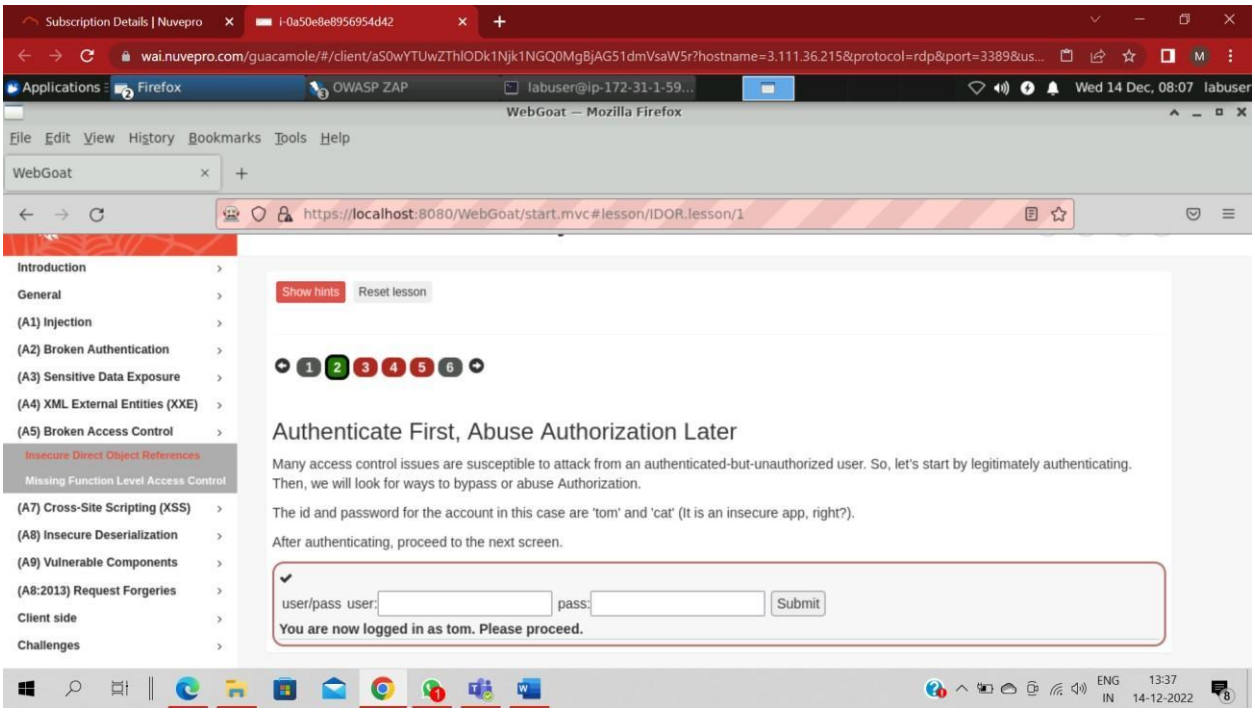

Subscription Details | Nuvepro
i-0a50e8e8956954d42
wai.nuvepro.com/guacamole/#/client/aS0wYTUwZThlODk1Njk1NGQ0MgBjAG51dmVsaW5r?hostname=3.111.36.215&protocol=rdp&port=3389&us...
Applications
Firefox
OWASP ZAP
labuser@ip-172-31-1-59...
Wed 14 Dec, 08:00 labuser
WebGoat — Mozilla Firefox
WebGoat
https://localhost:8080/WebGoat/start.mvc#lesson/XXE.lesson/6
Add a comment
Submit
madhavan 2022-12-14, 08:57:31
Hello
madhavan 2022-12-14, 08:53:28
.dockerenv bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
madhavan 2022-12-14, 08:49:17
Hai
webgoat 2022-12-14, 06:25:59
Silly cat....
guest 2022-12-14, 06:25:59
I think I will use this picture in one of my projects.
guest 2022-12-14, 06:25:59
Lol!! :-).
ENG
IN
13:30
14-12-2022

A5 – Broken Access Control

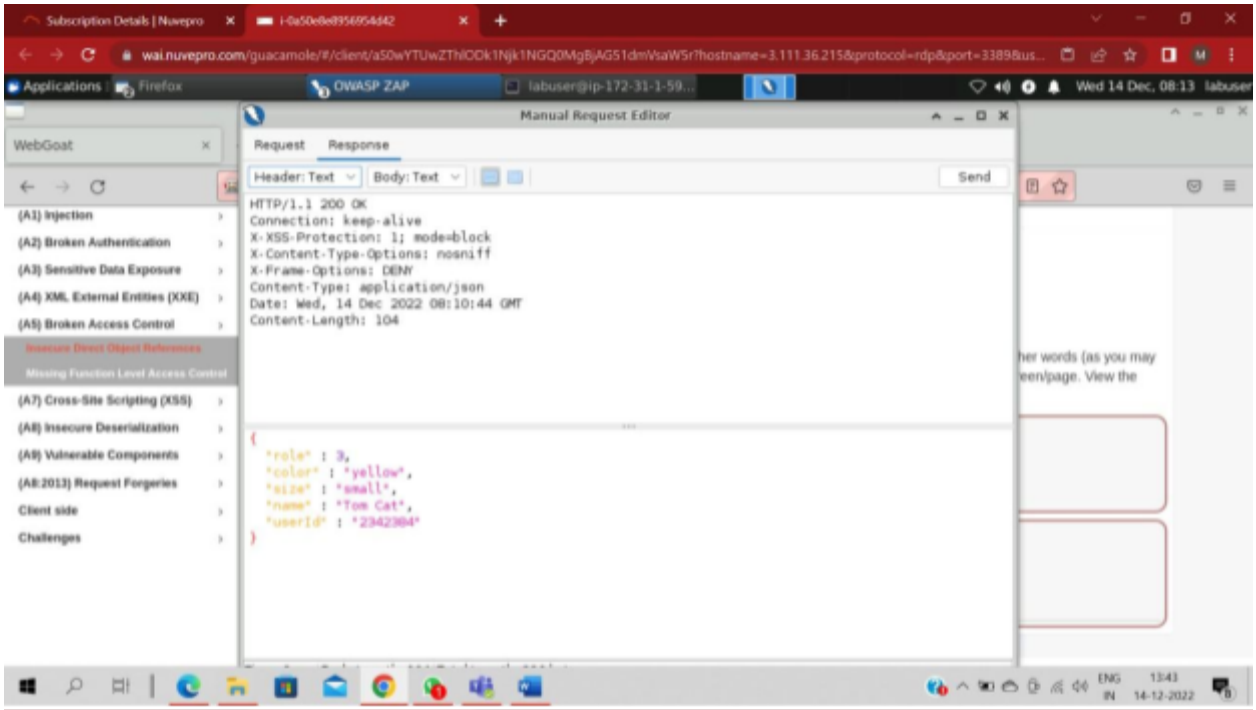
Aaron Mathew

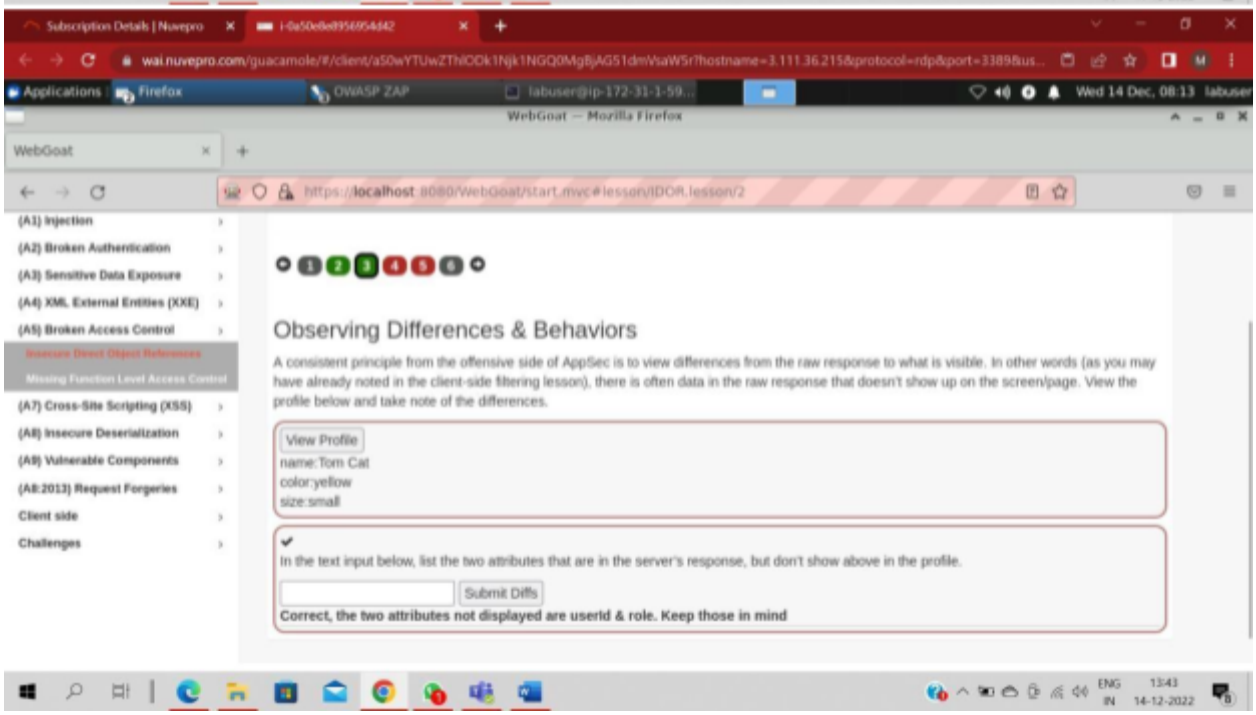
20BIS001 Insecure Direct Object References:

Authenticate first, Abuse Authentication Later:

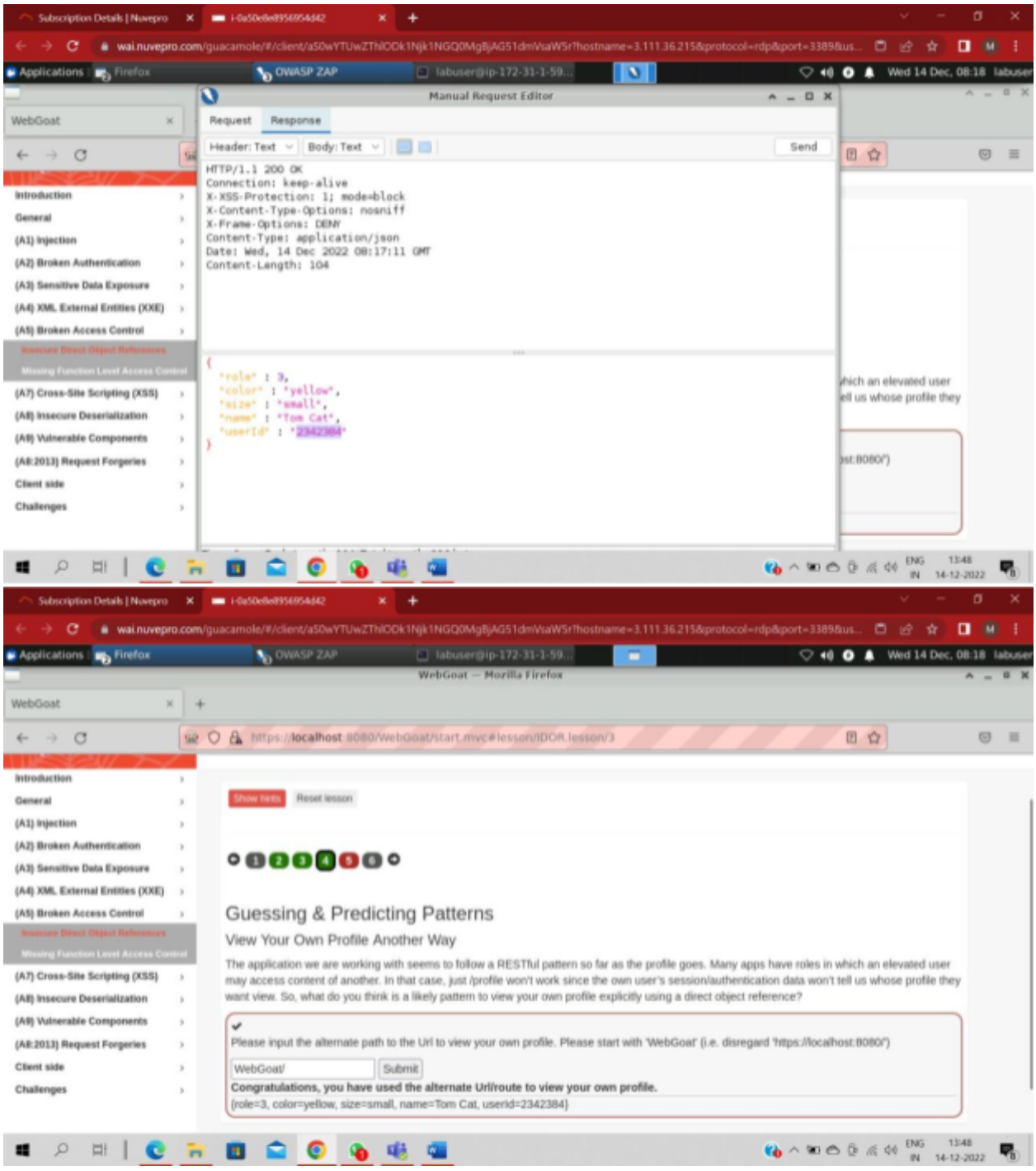


Observing Differences & Behaviours:

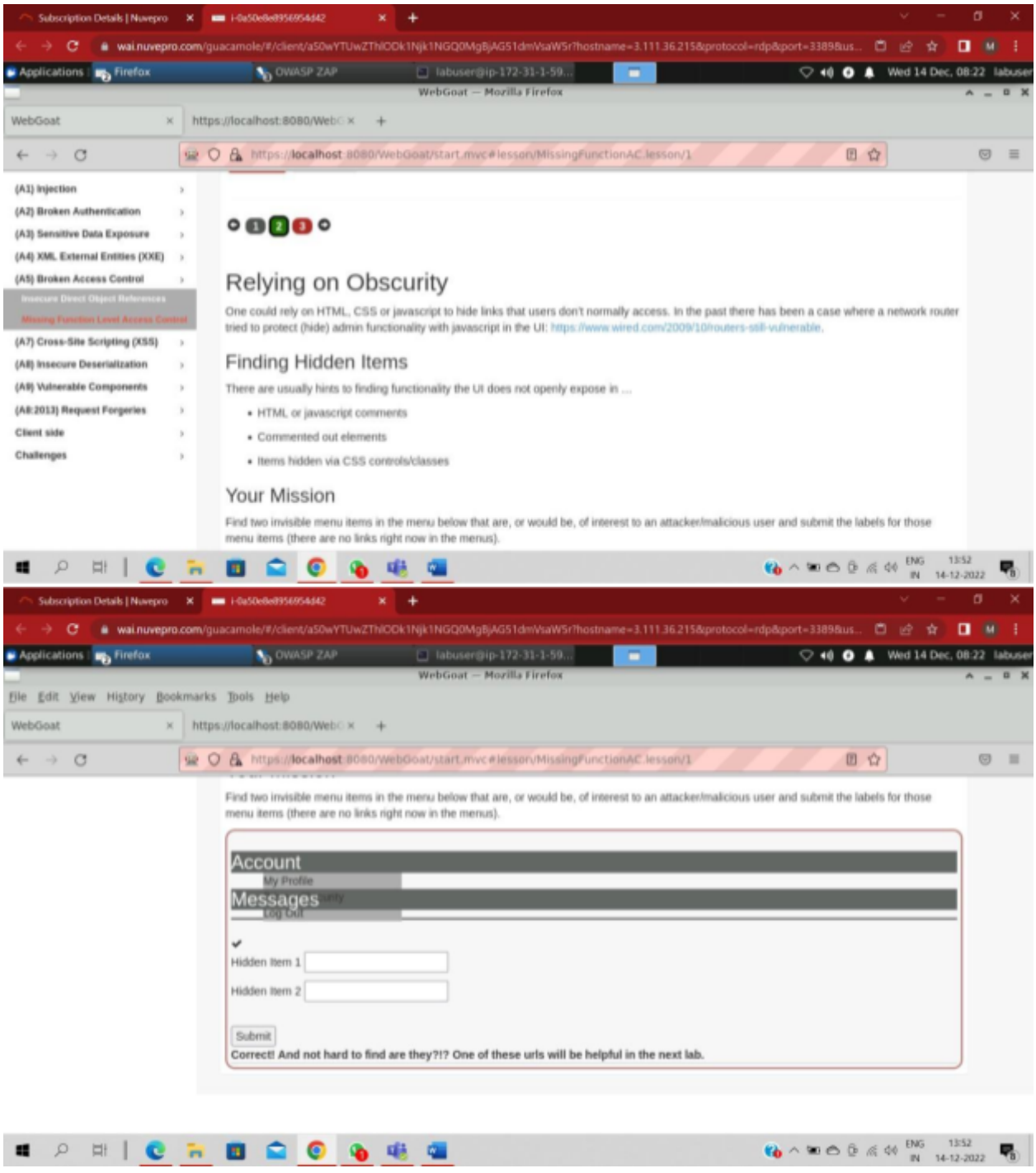

Manual Request Editor
Request
Response
Header: Text
Body: Text
Send
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Wed, 14 Dec 2022 08:10:44 GMT
Content-Length: 104
{
"role" : 3,
"color" : "yellow",
"size" : "small",
"name" : "Tom Cat",
"userId" : "2342384"
}


WebGoat -- Mozilla Firefox
Observing Differences & Behaviors
A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.
View Profile
name:Tom Cat
color:yellow
size:small
In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.
Submit Diffs
Correct, the two attributes not displayed are userId & role. Keep those in mind

Guessing and Predicting Patterns:



Missing Function Level Access Control:



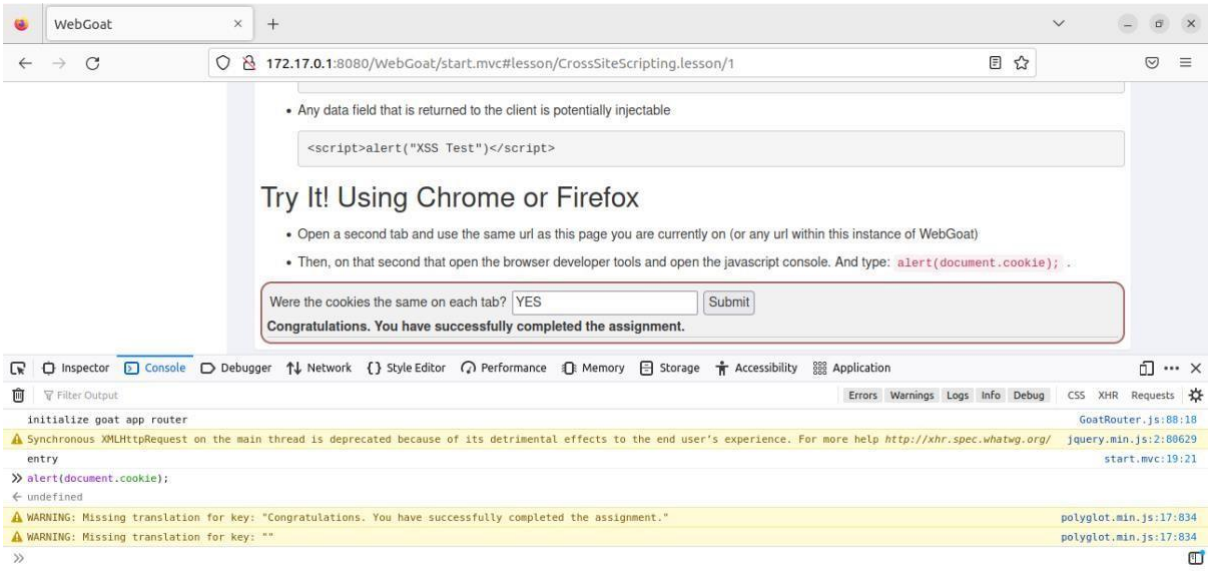
U18INI5600L-ENGINEERING CLINICS V

EX.NO: A7 CROSS SITE SCRIPTING

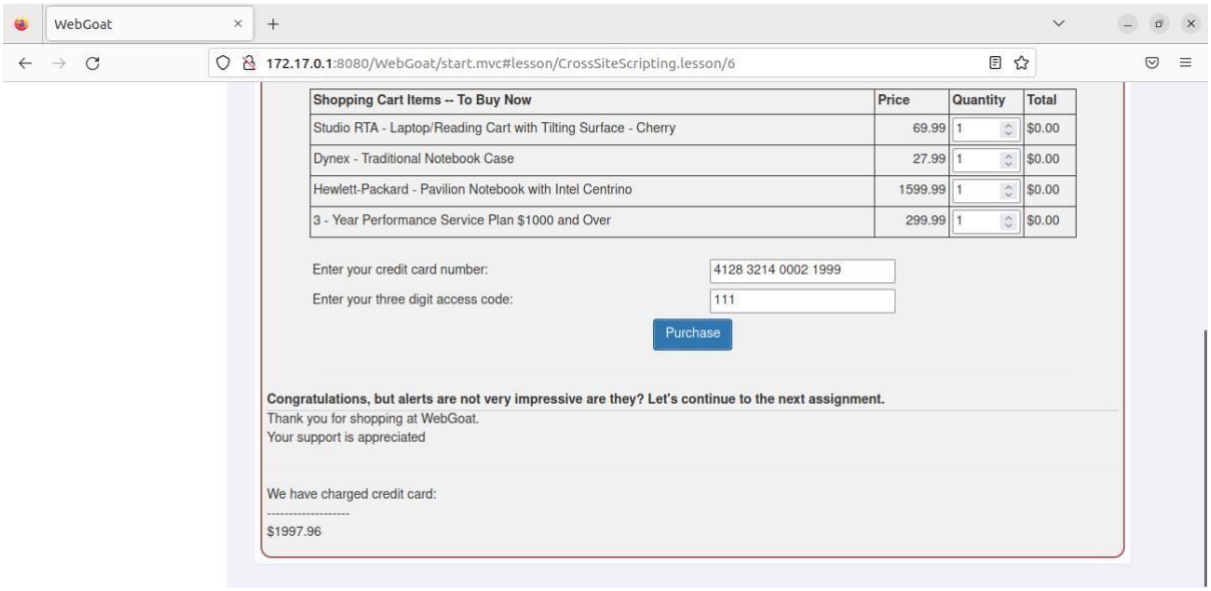
NAME : Aaron Mathew

ROLL NO : 20BIS001

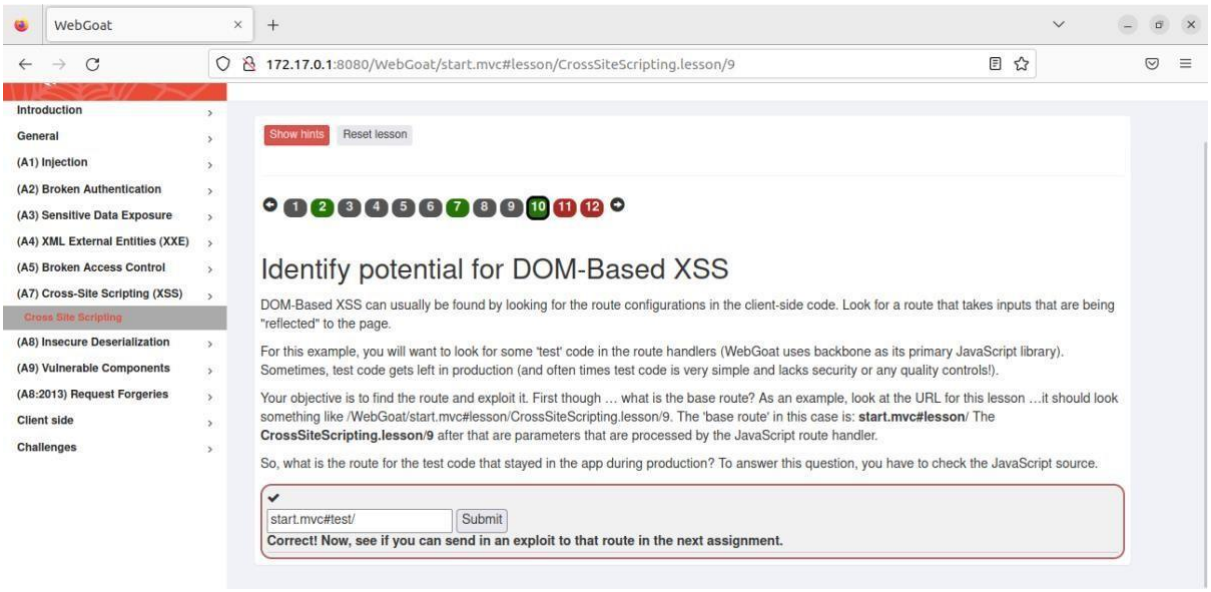
YES

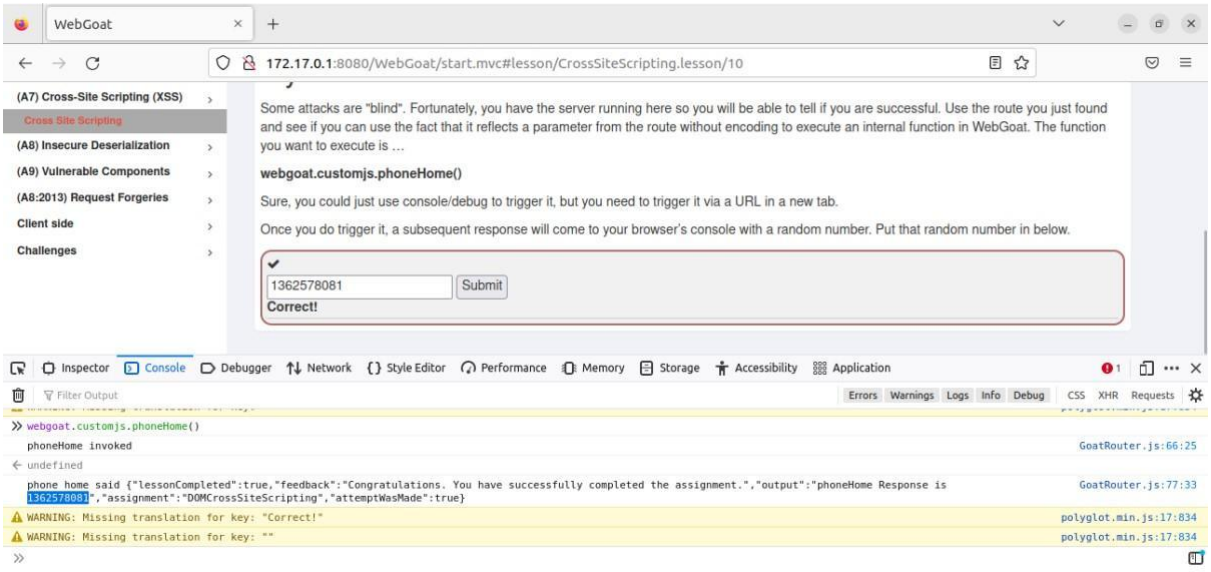


<script>alert(“s?madhav0230”)</script>

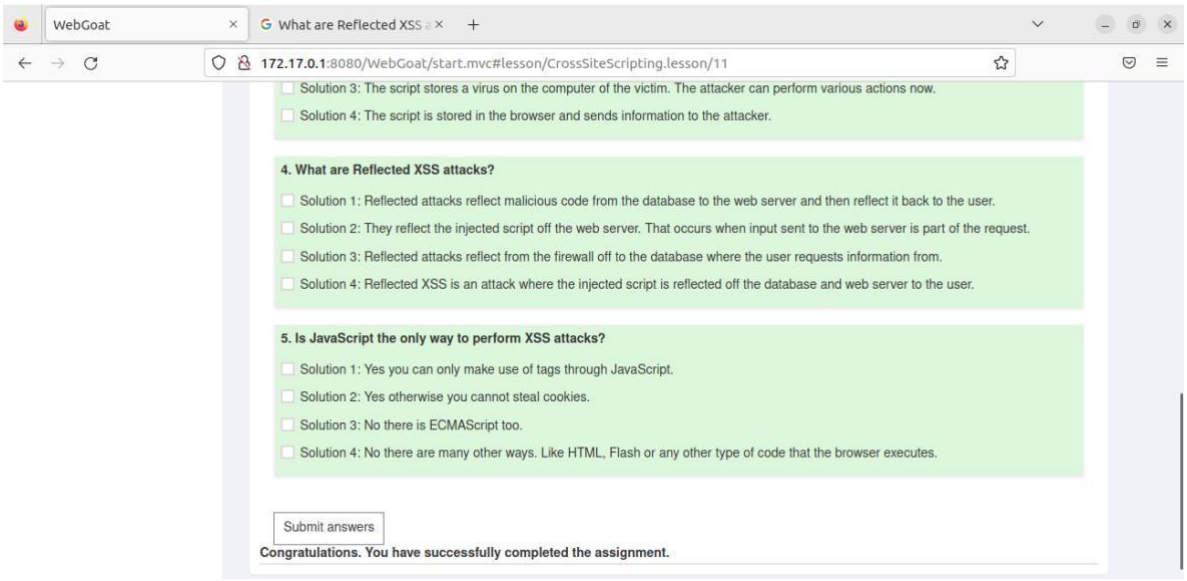


start.mvc#test/





1. Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.
2. Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.
3. Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.
4. Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
5. Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.


WebGoat
What are Reflected XSS
172.17.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/11
Solution 3: The script stores a virus on the computer of the victim. The attacker can perform various actions now.
Solution 4: The script is stored in the browser and sends information to the attacker.
4. What are Reflected XSS attacks?
Solution 1: Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.
Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
Solution 3: Reflected attacks reflect from the firewall off to the database where the user requests information from.
Solution 4: Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.
5. Is JavaScript the only way to perform XSS attacks?
Solution 1: Yes you can only make use of tags through JavaScript.
Solution 2: Yes otherwise you cannot steal cookies.
Solution 3: No there is ECMAScript too.
Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.
Submit answers
Congratulations. You have successfully completed the assignment.