

# 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:

The screenshot shows a Firefox browser window with the title bar "Activities Firefox Web Browser" and the date "Sep 2 14:51". The address bar displays the URL "172.17.0.1:8080/WebGoat/start.mvc#lesson/sqlInjection.lesson/4". The main content area is titled "SQL Injection (intro)". On the left, there's a sidebar with a "WEBGOAT" logo and a navigation menu. The menu items include "Introduction", "General", "(A1) Injection", "SQL Injection (intro)" (which is highlighted in red), "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", and "Challenges". Below the menu, there are numbered steps from 1 to 13, with step 5 highlighted in green. The main content area contains a section titled "Data Control Language (DCL)". It explains that DCL is used to implement access control logic in a database and can be used to revoke and grant user privileges on database objects. It also notes that attackers can violate confidentiality and availability by injecting DCL commands. A list of bullet points includes: "• DCL commands are used to implement access control on database objects.", "• GRANT - give a user access privileges on database objects", and "• REVOKE - withdraw user privileges that were previously given using GRANT". Below this, a note says "Try to grant rights to the table `grant_rights` to user `unauthorized_user`". There is a form with a "SQL query" input field containing "GRANT ALL PRIVILEGES ON grant\_rights TO unauthorized\_user;". A "Submit" button is present below the input field. A success message at the bottom states "Congratulations. You have successfully completed the assignment."

Activities Firefox Web Browser Sep 2 14:50

SQL injection (intro) S: D: +

Import bookmarks... Getting Started club registration

SQL injection (intro)

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

(A10) Request Forgeries

Client side

Challenges

172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/3

## 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

Submit

Congratulations. You have successfully completed the assignment.

```
ALTER TABLE employees ADD phone varchar(20)
```

Activities Firefox Web Browser Sep 2 14:50

SQL injection (intro) S: D: +

Import bookmarks... Getting Started club registration

SQL injection (intro)

Path traversal

(A3) Sensitive Data Exposure

(A4) XML External Entities (XXE)

(A5) Broken Access Control

(A7) Cross-Site Scripting (XSS)

(A8) Insecure Deserialization

(A9) Vulnerable Components

(A10) Request Forgeries

Client side

Challenges

172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson/2

## Data Manipulation Language (DML)

Data manipulation language refers to the manipulation of data. Many of the most common SQL statements involving data are included here.

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

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) 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 LO9B2V  
34477 Abraham Holman Development \$50.000 UU2ALK  
37648 John Smith Marketing \$64.350 3SL9GA

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\_id'). 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 protection 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: select department from employees where userid = 96134

You have succeeded!

select department from employees where userid = 96134  
DEPARTMENT  
Marketing

Activities Firefox Web Browser Sep 2 15:13

WebGoat SQL Injection (intro) 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

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!**

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) +

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

(A10) Request Forgeries

Client side

Challenges

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) +

Import bookmarks... Getting Started club registration

(A10) Request Forgeries

Client side

Challenges

If an application takes SQL queries simply by concatenating user supplied strings to the query, that application is very susceptible to string SQL injection. 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] S: D: +

Import bookmarks... Getting Started club registration

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 user_id = " + 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, 967664321, VISA, , 0,  
101, Joe, Snow, 22342000065411, MC, , 0,  
102, John, Smith, 24356000002222, MC, , 0,  
102, John, Smith, 123456789, MC, , 0,  
103, Jane, Plant, 333486703333, AMEX, , 0,  
103, Jane, Plant, 333486703333, AMEX, , 0,  
10312, Jolly, Hershey, 176896799, MC, , 0,  
10312, Jolly, Hershey, 333300003333, AMEX, , 0,  
10323, Grumpy, youaretheweaklink, 673834488, MC, , 0,  
10323, Grumpy, youaretheweaklink, 33413003333, AMEX, , 0,  
15603, Peter, Sand, 123609789, MC, , 0,  
15603, Peter, Sand, 338893453333, AMEX, , 0,  
15613, Joseph, Something, 33843453333, AMEX, , 0,  
15837, Chaos, Monkey, 32848346533, CM, , 0,  
19294, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT \* FROM user\_data WHERE Login\_Count = 0 and user\_id= 0 or 1=1

Activities Firefox Web Browser Sep 2 15:02

WebGoat SQL Injection [intro] S: D: +

Import bookmarks... Getting Started club registration

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 =  or  1 = 1  Get Account Info

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,  
101, Joe, Snow, 967854321, VISA, , 0,  
101, Joe, Snow, 2234200065411, MC, , 0,  
102, John, Smith, 24356000002222, MC, , 0,  
102, John, Smith, 123456789, MC, , 0,  
103, Jane, Plant, 333486703333, AMEX, , 0,  
103, Jane, Plant, 333486703333, AMEX, , 0,  
10312, Jolly, Hershey, 176896799, MC, , 0,  
10312, Jolly, Hershey, 333300003333, AMEX, , 0,  
10323, Grumpy, youaretheweaklink, 673834488, MC, , 0,  
10323, Grumpy, youaretheweaklink, 33413003333, AMEX, , 0,  
15603, Peter, Sand, 123609789, MC, , 0,  
15603, Peter, Sand, 338893453333, AMEX, , 0,  
15613, Joseph, Something, 33843453333, AMEX, , 0,  
15837, Chaos, Monkey, 32848346533, CM, , 0,  
19294, 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 '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.

The screenshot shows a Firefox browser window titled "WebGoat". The URL is 172.17.0.1:8080/WebGoat/start.mvc#lesson/SqliInjectionMitigations.lesson/4. The main content area is titled "SQL Injection (mitigation)". On the left, there's a sidebar with various challenges like Introduction, General, and specific attacks such as SQL Injection (intro), SQL Injection (advanced), and SQL Injection (mitigation). The SQL Injection (mitigation) challenge is selected. Below the sidebar, there's a navigation bar with numbered steps from 1 to 13, where step 5 is highlighted. The main content area has a heading "Try it! Writing safe code". It contains a snippet of Java code:

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
?;
setString();
setString();
Submit
```

Below the code, a message says "Congratulations. You have successfully completed the assignment."

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):

```
1 + try {
2     Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
3     PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE name = ?");
4     ps.setString(1, "Admin");
5     ps.executeUpdate();
6 } catch (Exception e) {
7     System.out.println("Oops. Something went wrong!");
8 }
```

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

## SQL Injection (mitigation)

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](#).

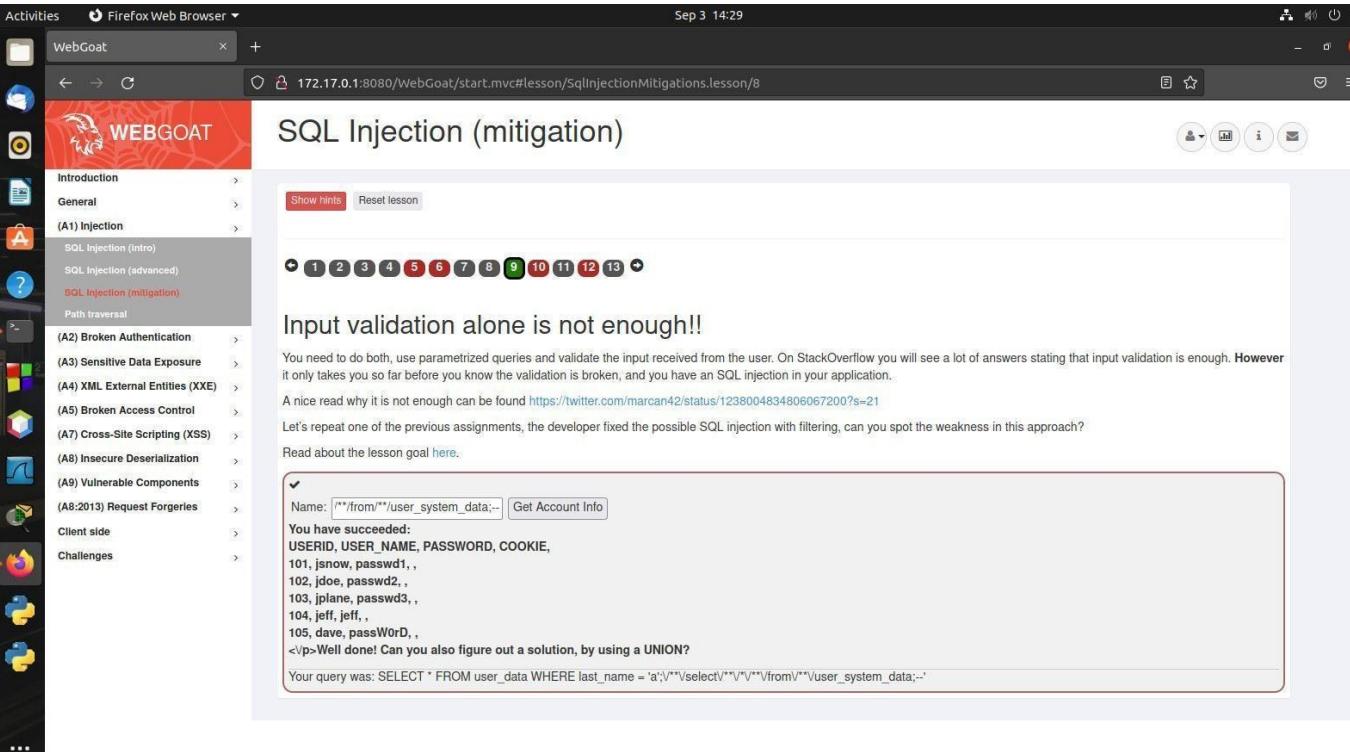
Name: `/**/from/**/user_system_data:-` Get Account Info

You have succeeded:

```
USERID, USER_NAME, PASSWORD, COOKIE,  
101, jsonow, passwd1,,  
102, jdoe, passwd2,,  
103, jplane, passwd3,,  
104, jeff, jeff,,  
105, dave, passWorD,,
```

<\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

## SQL Injection (mitigation)

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, jnow, passwd1, ,  
102, jdoe, passwd2, ,  
103, jplane, passwd3, ,  
104, jeff, jeff, ,  
105, dave, passWorD, ,

<v>Well done! Can you also figure out a solution, by using a UNION?

Your query was: SELECT \* FROM user\_data WHERE last\_name = 'A';V'VSELECTVVVFROMVUSER\_SYSTEM\_DATA;-'

5.

Activities Firefox Web Browser Sep 3 15:25

WebGoat 172.17.0.1:8080/WebGoat/start.mvc#lesson/SqlInjectionMitigations.lesson/11

## SQL Injection (mitigation)

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-prod 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

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server
webgoat-test	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	192.168.8.4	EF:12:FE:34:AA:CC	danger	Pre-production server

IP address webgoat-prod server: 192.168.8.12

Submit

Congratulations. You have successfully completed the assignment.

Activities

Web-GUI

Sep 3 14:56

172.17.0.5:80/webgoat/start.mvc?lessonId=sqlInjectionMitigations.lesson/11

REACH HOME Reset lesson

In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the `webgoat-preprod` server, guessing the complete ip address might take too long so we give you the last part: `xx.129.219.202`

Note: The `server` field of this assignment is **NOT** vulnerable to an SQL injection.

**LIST OF SERVERS**

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:00:00	success	Development server
webgoat-test	192.168.2.1	EE:FF:33:44:AB:CD	success	Test server
webgoat-accel	192.168.3.3	EF:12:FE:34:AA:00	warn	Acceleration server
webgoat-preprod	192.168.6.4	EF:12:FE:34:AA:00	warn	Pro production server

IP address webgoat test server: 104.130.219.202

Submit

Congratulations. You have successfully completed the assignment.

The screenshot shows a Firefox browser window with the URL `172.17.0.5:80/webgoat/start.mvc?lessonId=sqlInjectionMitigations.lesson/11`. The page title is "Web-GUI". On the left, there's a sidebar with various security-related categories like "Broken Authentication", "Broken Date/Time", etc. The main content area has a heading "LIST OF SERVERS" with a table. The table has columns: Hostname, IP, MAC, Status, and Description. It lists four servers: "webgoat-dev" (IP 192.168.4.0, MAC AA:BB:11:22:00:00, status success, desc Development server), "webgoat-test" (IP 192.168.2.1, MAC EE:FF:33:44:AB:CD, status success, desc Test server), "webgoat-accel" (IP 192.168.3.3, MAC EF:12:FE:34:AA:00, status warn, desc Acceleration server), and "webgoat-preprod" (IP 192.168.6.4, MAC EF:12:FE:34:AA:00, status warn, desc Pro production server). Below the table, there's a text input field containing "IP address webgoat test server: 104.130.219.202" and a "Submit" button. At the bottom, a message says "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.

The screenshot shows a Firefox browser window titled "WebGoat" at the URL [172.17.0.1:8080/WebGoat/start.mvc#lesson/SqInjectionAdvanced.lesson/2](http://172.17.0.1:8080/WebGoat/start.mvc#lesson/SqInjectionAdvanced.lesson/2). The page displays a navigation menu on the left with various security challenges. The current challenge is "SQL Injection (advanced)". The main content area is titled "Try It! Pulling data from other tables". It contains a SQL code snippet for creating the "user\_data" table:

```
CREATE TABLE user_data (userid int not null,
    first_name varchar(20),
    last_name varchar(20),
    cc_number varchar(30),
    cc_type varchar(10),
    cookie varchar(20),
    login_count int);
```

Below this, a note states: "Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table. The table you want to pull data from is:" followed by the "user\_system\_data" table definition:

```
CREATE TABLE user_system_data (userid int not null primary key,
    user_name varchar(12),
    password varchar(10),
    cookie varchar(30));
```

Two sub-tasks are listed: "6.a) Retrieve all data from the table" and "6.b) When you have figured it out.... What is Dave's password?". A note at the bottom says: "Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQL statement. Maybe you can find both of them." At the bottom, there is a form with fields for Name and Password, and a "Check Password" button. The Name field contains the value ":SELECT \* FROM user\_syste". The Password field contains "passW0rD". Below the form, a message says: "Congratulations. You have successfully completed the assignment."

2.

Activities Firefox Web Browser Sep 2 15:32

WebGoat

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

Challenges

Solution 2: ✘  
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 optimizes 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 separation 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;--

Congratulations. You have successfully completed the assignment.

The screenshot shows a Linux desktop environment with a window titled 'Firefox Web Browser'. The window displays a challenge page from 'Advanced Lessons'. The page contains three numbered questions:

3. How can prepared statements be faster than statements?
4. How can a prepared statement prevent SQL-injection?
5. What happens if a person with malicious intent writes into a register form (Robert); DROP TABLE Students;-- that has a prepared statement?

Each question has four solution options with radio buttons. The first two questions have their first option selected. The third question has its fourth option selected. A 'Submit answers' button is at the bottom, followed by a success message: "Congratulations. You have successfully completed the assignment."

# **OWASP Broken Authentication**

**Aaron Mathew**

**20BIS001**

**Authentication Bypass:**

The screenshot shows the OWASP ZAP interface. In the top right, a browser window displays a JSON response from [wai.nuvepro.com](https://wai.nuvepro.com/guacamole/#/client/a50wYTUwZThODk1Njk1NGQ0MgBjAG51dmVsW5r?hostname=3.111.36.215&protocol=rdp&port=3389&us). The response header is:

```
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 05:39:20 GMT
Content-Length: 246
```

The JSON body contains:

```
{
  "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
}
```

In the bottom left, the ZAP history pane shows several proxy requests and responses. One entry is highlighted with a red border:

Id	Source	Req. Timestamp
3.393	Proxy	12/14/22, 5:30:34
3.394	Proxy	12/14/22, 5:30:35
3.395	Proxy	12/14/22, 5:30:35
3.396	Proxy	12/14/22, 5:30:36
3.398	Proxy	12/14/22, 5:30:45

The screenshot shows a Mozilla Firefox browser window titled "WebGoat". The URL is <https://localhost:8080/WebGoat/start.mvc#lesson/AuthBypass.lesson/1>. The page title is "Authentication Bypasses".

The left sidebar menu includes:

- Introduction
- General
- (A1) Injection
- (A2) Broken Authentication
- Authentication Bypasses** (highlighted)
- 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

The main content area displays the "2FA Password Reset" section. It includes a note about a 2016 example where a user bypassed 2FA by answering security questions. Below this, there is a step 2 instruction: "Step 2: Enter any answer for security questions."

## JWT Tokens:

## Decoding JWT Tokens:

The screenshot shows a Firefox browser window with the URL <https://localhost:8080/WebGoat/start.mvc#lesson/JWT.lesson/2>. The page title is "Decoding a JWT token". On the left, there's a sidebar with various security challenges. The main content area displays a large JWT token and a success message.

JWT token:

```
eyJhbGciOiJIUzI1NiJ9.ew0KICA1YXV0aG9yaXRpZXMiDogIyAiUK9MRV9BRE1JTiIsICJST0xFX1VTRVIIIF0sD0ogICJjbGllbnRfaWQiIDogIw15LWNsaWVudC13aXR0LXNlY3JldCIsDQogICJleHAIIDogMTYwNzA5OTYwOCwNCiAgImp0aS1g0iAi0WJj0TjhNDQtMGixYS00YzV1LWJlNzAtZGE1MjA3NWI5Tg0i1wNCiAgInNjb3BlIiA6IFsgInJlyWQ1LCAl3JpdGU1IF0sDQogICJ1c2VyX25hbWUiIDogInVzzXi1DQp9.91YaULTuoIDJ86-zKD5ntJQyHPpJ2mZAbnWRfel991I
```

Success message:

Congratulations. You have successfully completed the assignment.

The screenshot shows a Firefox browser window with the URL <https://localhost:9090/WebWolf/jwt>. The page displays the decoded JWT token structure.

Encoded:

```
eyJhbGciOiJIUzI1NiJ9.ew0KICA1YXV0aG9yaXRpZXMiDogIyAiUK9MRV9BRE1JTiIsICJST0xFX1VTRVIIIF0sD0ogICJjbGllbnRfaWQiIDogIw15LWNsaWVudC13aXR0LXNlY3JldCIsDQogICJleHAIIDogMTYwNzA5OTYwOCwNCiAgImp0aS1g0iAi0WJj0TjhNDQtMGixYS00YzV1LWJlNzAtZGE1MjA3NWI5Tg0i1wNCiAgInNjb3BlIiA6IFsgInJlyWQ1LCAl3JpdGU1IF0sDQogICJ1c2VyX25hbWUiIDogInVzzXi1DQp9.91YaULTuoIDJ86-zKD5ntJQyHPpJ2mZAbnWRfel991I
```

Decoded:

Header:

```
{  
  "alg": "HS256"  
}
```

Payload:

```
{  
  "authorities": [ "ROLE_ADMIN", "ROLE_USER" ],  
  "client_id": "my-client-with-secret",  
  "exp": 1607099608,  
  "jti": "9bc92a44-0bla-4c5e-be70-da52075b9a84",  
  "scope": [ "read", "write" ],  
  "user_name": "user"  
}
```

## JWT Signing:

The screenshot shows the OWASP ZAP interface with a captured request and response. The response body contains a JSON object with the following content:

```
{
    "lessonCompleted": true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "output": null,
    "assignment": "JWTVotesEndpoint",
    "attemptWasMade": true
}
```

The ZAP interface includes a sidebar with contexts and sites, a history tab, and an alert table with entries for SetCookie, JSON, and JSON.

The screenshot shows the WebGoat application running in Mozilla Firefox. The URL is <https://localhost:8080/WebGoat/start.mvc#lesson/JWT.lesson/4>. The page title is "JWT tokens". The left sidebar shows a navigation tree with "JWT tokens" selected under "Authentication Bypasses". The main content area contains the following text:

JWT signing

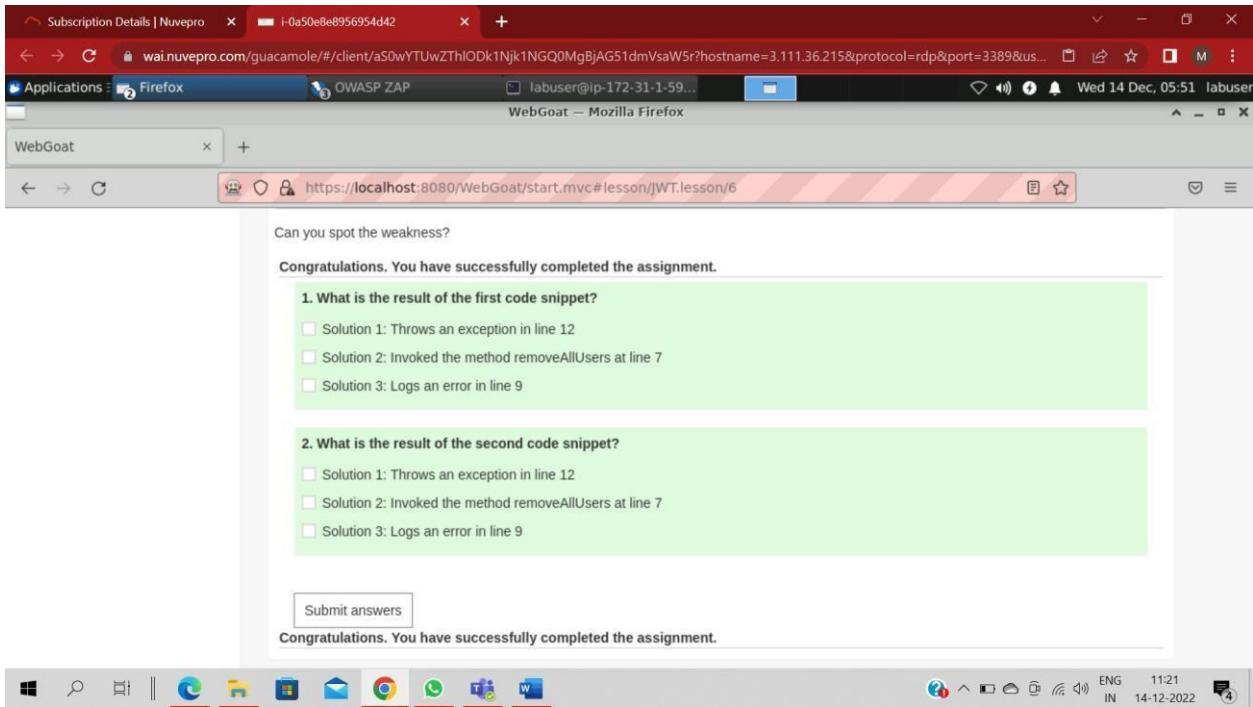
Each JWT token should at least be signed before sending it to a client, if a token is not signed the client application would be able to change the contents of the token. The signing specifications are defined [here](#) the specific algorithms you can use are described [here](#). It basically comes down you use "HMAC with SHA-2 Functions" or "Digital Signature with RSASSA-PKCS1-v1\_5/ECDSSA/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.

At the bottom of the page, the URL is <https://localhost:8080/WebGoat/start.mvc#lesson/JWT.lesson/3>.

## Code Review:



## Refreshing a Token:

Subscription Details | Navepro X i-050e8e8356954d42 X +

Applications Firefox labuser@ip-172-31-1-59... Manual Request Editor

File Edit View History Bookmarks Tools Help

WebGoat X

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 06:30:18 GMT

Content-Length: 198

[{"lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "output": null, "assignment": "JWTRefreshEndpoint", "attemptWasMade": true}]

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 (A10:2013) Request Forgeries Client side Challenges

Subscription Details | Navepro X i-050e8e8356954d42 X +

Applications Firefox labuser@ip-172-31-1-59... WebGoat — Mozilla Firefox

File Edit View History Bookmarks Tools Help

WebGoat X

localhost:8080/WebGoat/mvc#lesson/JWT/lesson/9

JSON Web Tokens - jwt.io Epoch Converter - Unix T +

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 (A10:2013) Request Forgeries Client side Challenges

Refresh a token

It is important to implement a good strategy for refreshing an access token. This assignment is based on a vulnerability found in a private bug bounty program on Bugcrowd, you can read the full write up [here](#)

Assignment

From a breach of last year the following logfile is available [here](#) Can you find a way to order the books but let Tom pay for them?

Product	Quantity	Price	Total
	3	\$ 4.87	\$14.61
<a href="#">Remove</a>			

Learn to defend your application with WebGoat

## Final Challenge:

Subscription Details | Navepro X i-050e8e8356954d42 X +

Applications Firefox labuser@ip-172-31-1-59... Manual Request Editor

File Edit View History Bookmarks Tools Help

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 06:55:59 GMT  
 Content-Length: 198

```
{
  "lessonCompleted": true,
  "feedback": "Congratulations. You have successfully completed the assignment.",
  "output": null,
  "assignment": "JWTFinalEndpoint",
  "attemptWasMade": true
}
```

can only delete his

Introduction 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

Subscription Details | Navepro X i-050e8e8356954d42 X +

Applications Firefox labuser@ip-172-31-1-59... WebGoat — Mozilla Firefox

File Edit View History Bookmarks Tools Help

localhost:8080/WebGoat/mvc#lesson/JWT/lesson/10 https://localhost:8080/WebGoat/start.mvc#lesson/JWT/lesson/10

JSON Web Tokens - jwt.io Epoch Converter - Unix T +

WEBGOAT

JWT tokens

Show help Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12

Final challenge

Below you see two accounts, one of Jerry and one of Tom. Jerry wants to remove Tom's account from Twitter, but his token can only delete his account. Can you try to help him and delete Tom's account?

ENG IN 14-12-2022

## Password Reset:

## Email Functionality with Webwolf:

The screenshot shows a Windows desktop environment with a Firefox browser window open. The browser's address bar displays the URL <https://localhost:8080/WebGoat/start.mvc?lesson=PasswordReset.lesson/1>. The main content of the browser is a page titled "Email functionality with WebWolf". This page contains a form for "Account Access" with fields for "Email" and "Password", and a blue "Access" button. Below the form is a link "Forgot your password?". A message at the bottom of the page says "Congratulations. You have successfully completed the assignment." To the right of the browser window, there is a small icon of a wolf. The desktop taskbar at the bottom shows various pinned icons, including Microsoft Edge, File Explorer, Mail, and Google Chrome. The system tray on the right shows the date and time as "14-12-2022 12:57", along with other system status indicators.

## Security Questions:

Subscription Details | Navepro X I-a50e8e8356954d42 X +

Applications Firefox OWASP ZAP labuser@ip-172-31-1-59... WebGoat — Mozilla Firefox

File Edit View History Bookmarks Tools Help

WebGoat localhost:8080/WebGoat/mvc#lesson/PasswordReset.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 (A10) Request Forgeries Client side Challenges

Reset lesson

1 2 3 4 5 6 7 8

## Security questions

This has been an issue and still is for a lot of websites, when you lost your password the website will ask you for a security question which you answered during the sign up process. Most of the time this list contains a fixed number of question and which sometimes even have a limited set of answers. In order to use this functionality a user should be able to select a question by itself and type in the answer as well. This way users will not share the question which makes it more difficult for an attacker.

One important thing to remember the answers to these security question(s) should be treated with the same level of security which is applied for storing a password in a database. If the database leaks an attacker should not be able to perform password reset based on the answer of the security question.

Users share so much information on social media these days it becomes difficult to use security questions for password resets, a good resource for security questions is: <http://goodsecurityquestions.com/>

## Assignment

Subscription Details | Navepro X I-a50e8e8356954d42 X +

Applications Firefox OWASP ZAP labuser@ip-172-31-1-59... WebGoat — Mozilla Firefox

File Edit View History Bookmarks Tools Help

WebGoat localhost:8080/WebGoat/mvc#lesson/PasswordReset.lesson/3

Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user. Users you could try are: "tom", "admin" and "larry".

Sign up Login

WebGoat Password Recovery

Your username

Username

What is your favorite color?

Answer security question

Submit

Congratulations. You have successfully completed the assignment.

## The problem with security questions:

The Problem with Security Questions

While Security Questions may at first seem like a good way to do authentication, they have some big problems.

The "perfect" security question should be hard to crack, but easy to remember. Also the answer needs to fixed, so it must not be subject to change.

There are only a handful of questions which satisfy these criteria and practically none which apply to anybody.

If you have to pick a security question, we recommend not answering them truthfully.

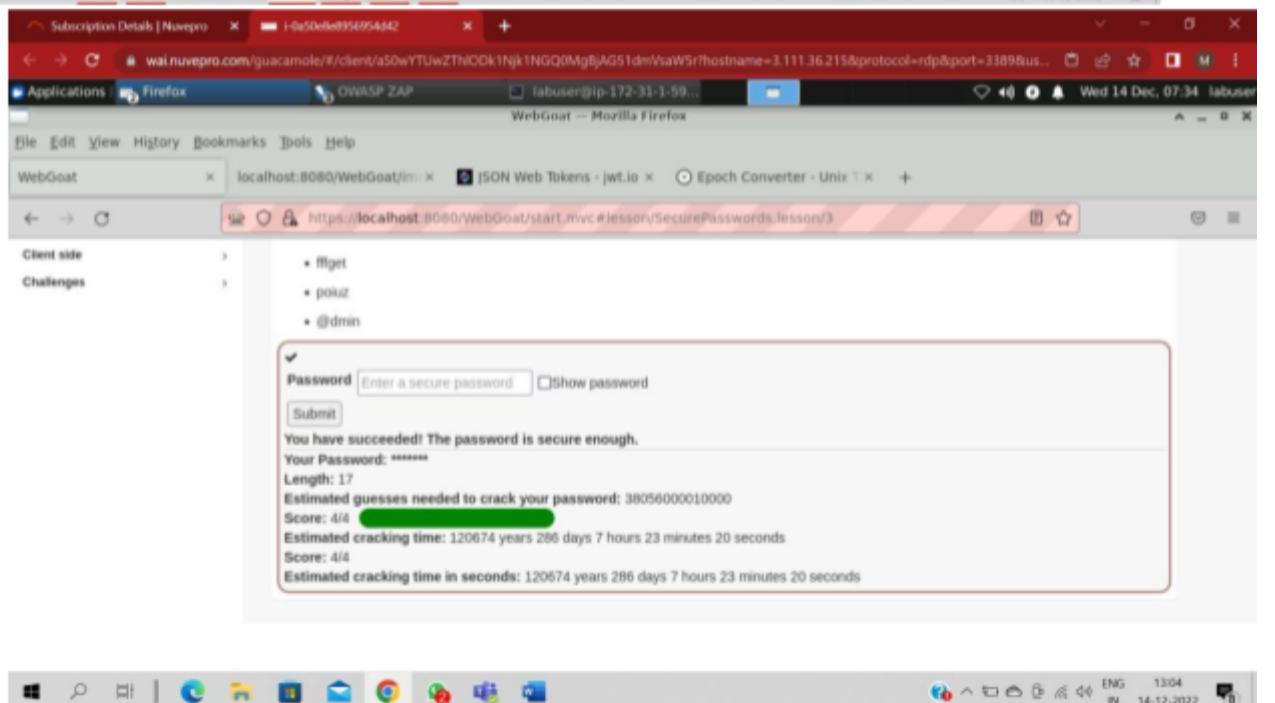
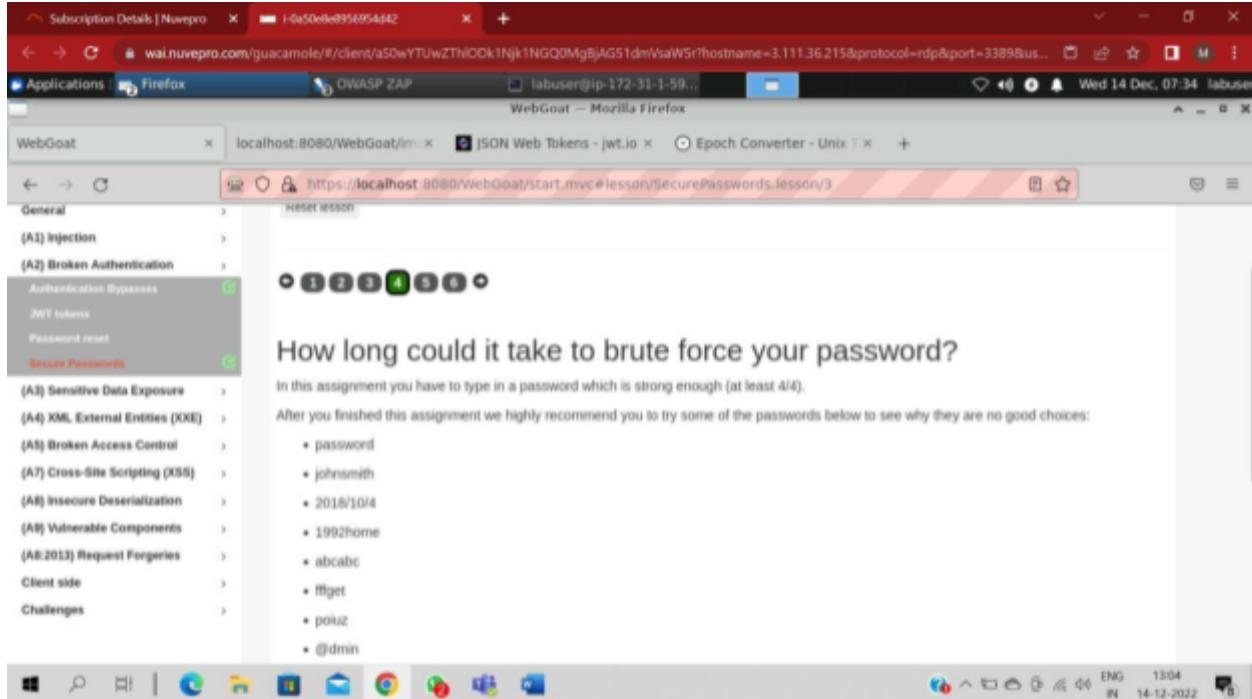
To further elaborate on the matter, there is a small assignment for you: There is a list of some common security questions down below. If you choose one, it will show to you why the question you picked is not really as good as one may think.

When you have looked at two questions the assignment will be marked as complete.

What is your favorite animal?  check

Optional[There are only two possible real answers, so really easy to guess.]

## Secure Passwords:



# ENGINEERING CLINICS -U18INI5600

## A3-SENSITIVE DATA EXPOSURE

NAME: Aaron Mathew

ROLL.NO:20BIS001

The screenshot shows the WebGoat application interface. The left sidebar contains a navigation menu with the following items:

- Introduction
- General
- (A1) Injection
- (A2) Broken Authentication
- (A3) Sensitive Data Exposure
- Insecure Login** (highlighted in red)
- (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

The main content area is titled "Insecure Login". It features a "Reset lesson" button and a progress bar with three circles, where the third circle is green and labeled "2". Below the progress bar, the text "Let's try" is displayed, followed by instructions: "Click the "log in" button to send a request containing login credentials of another user. Then, write these credentials into the appropriate fields and submit to confirm. Try using a packet sniffer to intercept the request." A form is present with a checked checkbox and a "Log in" button. The input fields contain "CaptainJack" and "....." (redacted password). The "Submit" button is visible. At the bottom of the form, a success message reads: "Congratulations. You have successfully completed the assignment."

# **A4 – XML External Entities(XXE)**

**Aaron Mathew**

**20BIS001**

**XXE:**

Subscription Details | Nuvepro X i-0a50e8e8356954d42 X +

← → C wai.nuvepro.com/guacamole/#/client/a50wYTUwZTHlODk1Njk1NGQ0MgBjAG51dmVsW5r?hostname=3.111.36.215&protocol=rdp&port=3389&us...

Applications Firefox OWASP ZAP labuser@ip-172-31-1-59... Send Wed 14 Dec, 07:56 labuser

File Edit View History Bookmarks Tools Help

Request Response

Header: Text Body: Text

WebGoat

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 (A9) Vulnerable Components (A10:2013) Request Forgeries Client side Challenges

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 07:53:29 GMT

Content-Length: 189

{ "lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "output": null, "assignment": "SimpleXXE", "attemptWasMade": true }

Subscription Details | Nuvepro X i-0a50e8e8356954d42 X +

← → C wai.nuvepro.com/guacamole/#/client/a50wYTUwZTHlODk1Njk1NGQ0MgBjAG51dmVsW5r?hostname=3.111.36.215&protocol=rdp&port=3389&us...

Applications Firefox OWASP ZAP labuser@ip-172-31-1-59... WebGoat — Mozilla Firefox Wed 14 Dec, 07:56 labuser

File Edit View History Bookmarks Tools Help

WebGoat

General (A1) Injection (A2) Broken Authentication (A3) Sensitive Data Exposure (A4) XML External Entities (XXE)

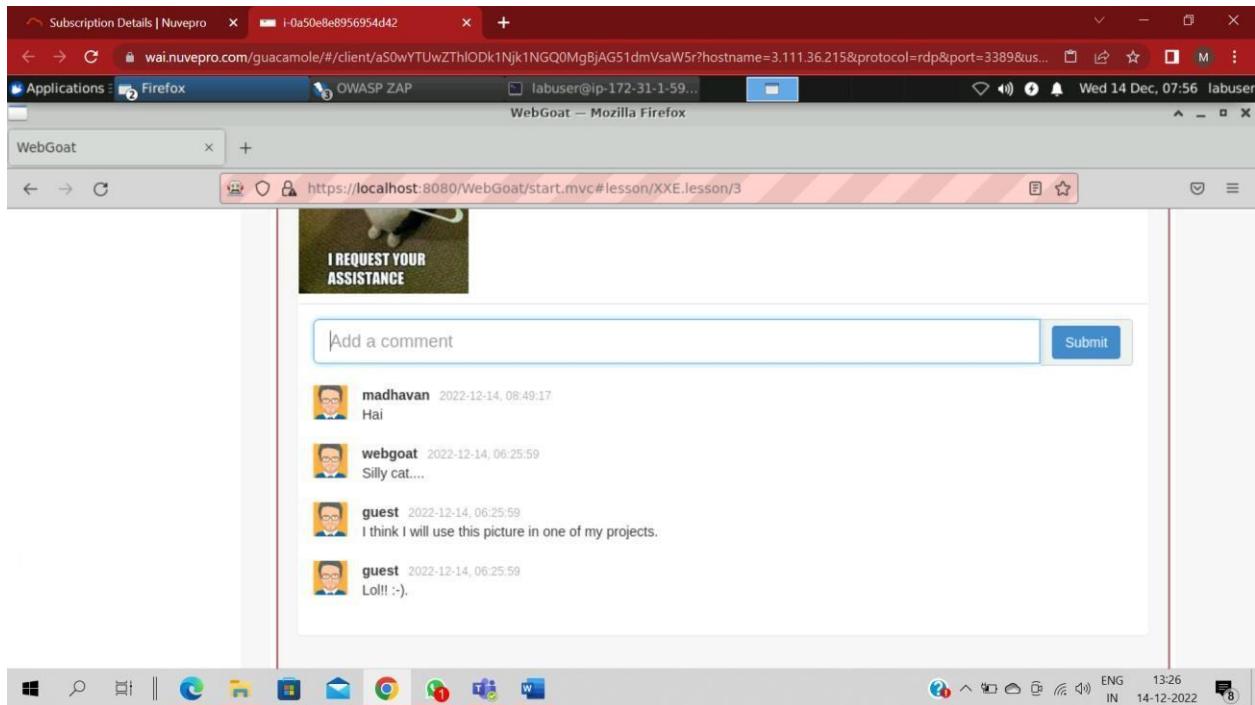
(A5) Broken Access Control (A7) Cross-Site Scripting (XSS) (A8) Insecure Deserialization (A9) Vulnerable Components (A10:2013) Request Forgeries Client side Challenges

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.  
24 days ago

HUMAN



## Modern REST Framework:

The screenshot shows the OWASP ZAP interface with the 'Manual Request Editor' tab selected. The 'Request' tab displays an HTTP response header:

```
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
```

The 'Response' tab shows a JSON payload:

```
{
  "lessonCompleted": true,
  "feedback": "Congratulations. You have successfully completed the assignment.",
  "output": null,
  "assignment": "ContentTypeAssignment",
  "attemptWasMade": true
}
```

A tooltip on the right side of the interface says: "this might result in".

The screenshot shows the WebGoat interface for the XXE exercise. The URL is <https://localhost:8080/WebGoat/start.mvc#lesson/XXE.lesson/6>.

## XXE

Show hints Reset lesson

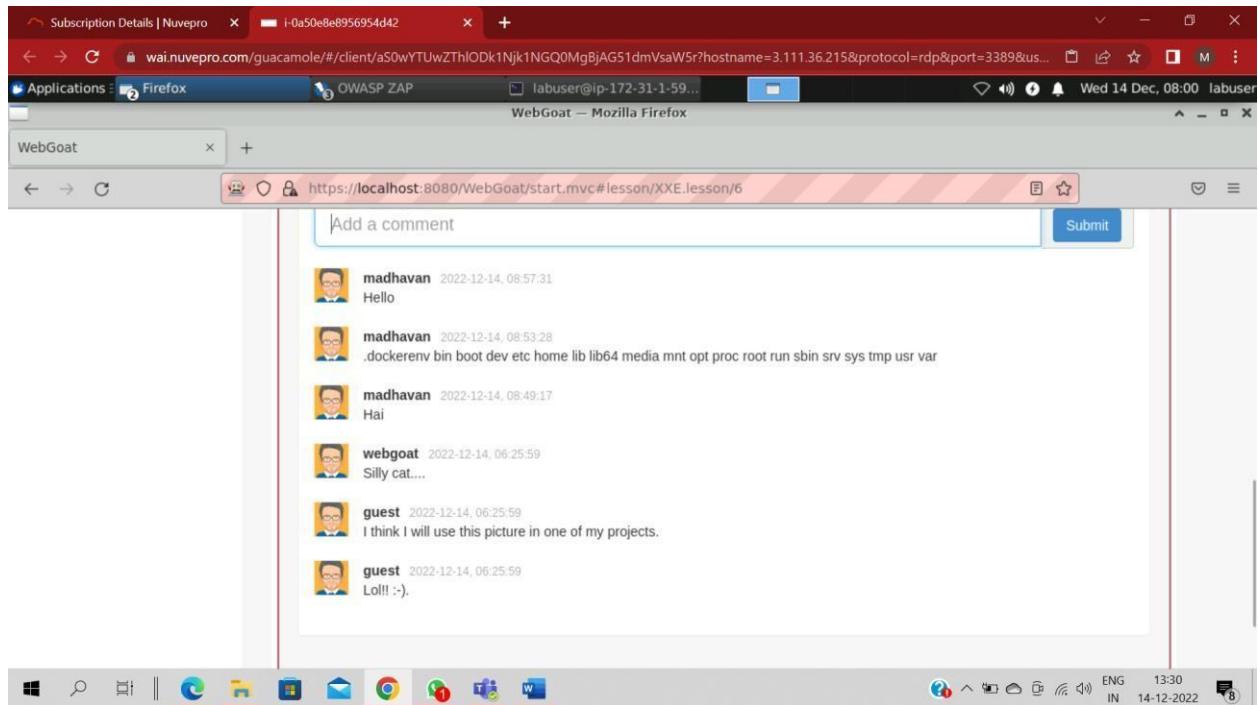
1 2 3 4 5 6 7 8 9 10 11 12 13

### 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



# A5 – Broken Access Control

Aaron Mathew

## 20BIS001 Insecure Direct Object References:

Authenticate first, Abuse Authentication Later:

The screenshot shows a Mozilla Firefox browser window with multiple tabs open. The active tab is 'WebGoat' at the URL <https://localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/1>. The browser's status bar indicates it's running on a Windows 10 system, with the date and time being Wednesday, December 14, 2022, at 08:07. The Firefox toolbar includes icons for Home, Search, and various extensions. The main content area of the browser shows the 'WebGoat' application interface. On the left, there's a navigation sidebar with several lessons listed: Introduction, General, (A1) injection, (A2) Broken Authentication, (A3) Sensitive Data Exposure, (A4) XML External Entities (XXE), (A5) Broken Access Control, Insecure Direct Object References (which is currently selected), Missing Function Level Access Control, (A7) Cross-Site Scripting (XSS), (A8) Insecure Deserialization, (A9) Vulnerable Components, and (A8:2013) Request Forgeries. Below the sidebar, there's a section titled 'Authenticate First, Abuse Authorization Later'. It contains text explaining that many access control issues are susceptible to attack from an authenticated-but-unauthorized user. It instructs the user to log in as 'tom' with password 'cat'. A form is provided for entering the credentials, with a note below it stating 'You are now logged in as tom. Please proceed.' At the bottom of the browser window, there's a taskbar with icons for various Windows applications like File Explorer, Task View, and Control Panel.

Observing Differences & Behaviours:

Subscription Details | Nuvepro

i-050e8e8356954d42

Applications Firefox

OWASP ZAP labuser@ip-172-31-1-59... Wed 14 Dec, 08:13 labuser

WebGoat

Request Response

Header: Text Body: Text

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

Insecure Direct Object References

Missing Function Level Access Control

(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

(A9) Vulnerable Components

(A10) Request Forgeries

Client side

Challenges

{  
  "role": 3,  
  "color": "yellow",  
  "size": "small",  
  "name": "Tom Cat",  
  "userId": "2342384"  
}

her words (as you may  
seen/page. View the

Subscription Details | Nuvepro

i-050e8e8356954d42

Applications Firefox

OWASP ZAP labuser@ip-172-31-1-59... Wed 14 Dec, 08:13 labuser

WebGoat — Mozilla Firefox

WebGoat

https://localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/2

(A1) Injection

(A2) Broken Authentication

(A3) Sensitive Data Exposure

(A4) XML External Entities (XXE)

(A5) Broken Access Control

Insecure Direct Object References

Missing Function Level Access Control

(A7) Cross-Site Scripting (XSS)

(A8) Insecure Deserialization

(A9) Vulnerable Components

(A10) Request Forgeries

Client side

Challenges

1 2 3 4 5 6 7

### 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.

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.

Correct, the two attributes not displayed are userId & role. Keep those in mind

ENG IN 14-12-2022 15:43

# Guessing and Predicting Patterns:

The screenshot shows the OWASP ZAP interface with a manual request editor. The request is an HTTP POST to `/start.mvc#lesson/IDOR.lesson/3`. The JSON payload is:

```
{"role": 3, "color": "yellow", "size": "small", "name": "Tom Cat", "userId": "2342384"}
```

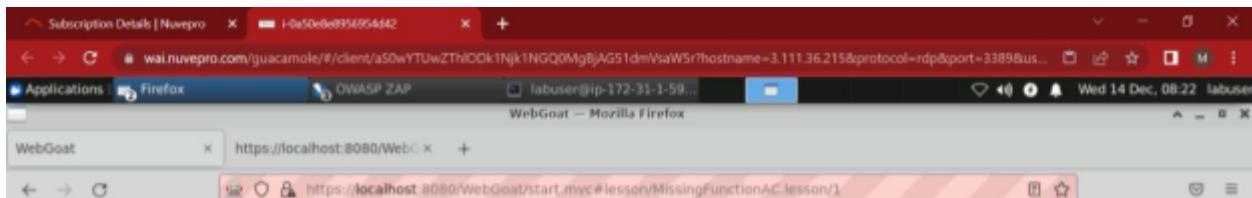
The response body contains the following text:

which an elevated user  
tell us whose profile they  
want to view.

Post:BOBO()

The screenshot shows a Firefox browser window displaying the WebGoat application at `https://localhost:8080/WebGoat/start.mvc#lesson/IDOR.lesson/3`. The page includes a navigation menu and a section titled "Guessing & Predicting Patterns". Below this, there is a form asking for an alternate path to view the own profile, with a note: "Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'https://localhost:8080')". A success message is shown: "Congratulations, you have used the alternate Url/route to view your own profile. [role=3, color=yellow, size=small, name=Tom Cat, userId=2342384]".

# Missing Function Level Access Control:



## Relying on Obscurity

One could rely on HTML, CSS or javascript to hide links that users don't normally access. In the past there has been a case where a network router tried to protect (hide) admin functionality with javascript in the UI: <https://www.wired.com/2009/10/routers-still-vulnerable>.

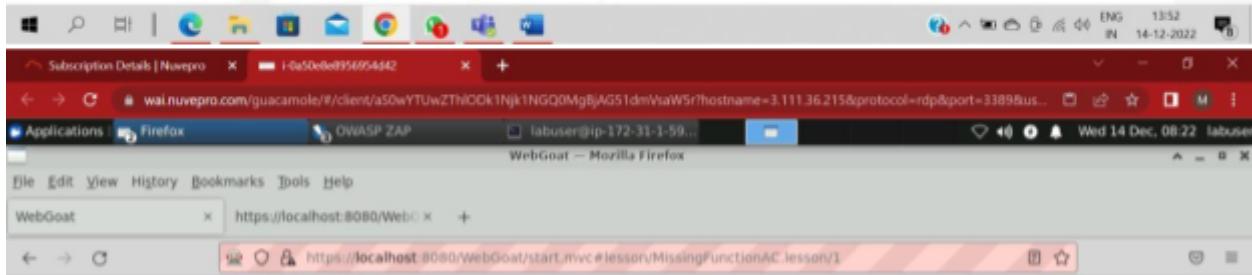
## Finding Hidden Items

There are usually hints to finding functionality the UI does not openly expose in ...

- HTML or javascript comments
- Commented out elements
- Items hidden via CSS controls/classes

## Your Mission

Find two invisible menu items in the menu below that are, or would be, of interest to an attacker/malicious user and submit the labels for those menu items (there are no links right now in the menus).



Find two invisible menu items in the menu below that are, or would be, of interest to an attacker/malicious user and submit the labels for those menu items (there are no links right now in the menus).

### Account

My Profile

### Messages

Log Out

Hidden Item 1

Hidden Item 2

Submit

Correct! And not hard to find are they?!? One of these urls will be helpful in the next lab.





## U18INI5600L-ENGINEERING CLINICS V

### EX.NO: A7 CROSS SITE SCRIPTING

NAME : Aaron Mathew

ROLL NO : 20BIS001

YES

The screenshot shows a browser window titled "WebGoat" with the URL [172.17.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/1](http://172.17.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/1). The page content includes:

- A note: "Any data field that is returned to the client is potentially injectable".
- A text input field containing the XSS payload: <script>alert("XSS Test")</script>.
- A section titled "Try It! Using Chrome or Firefox" with instructions:
  - Open a second tab and use the same URL as this page you are currently on (or any URL within this instance of WebGoat).
  - Then, on that second tab open the browser developer tools and open the JavaScript console. And type: `alert(document.cookie);`.
- A form with the question "Were the cookies the same on each tab? YES" and a "Submit" button.
- A success message: "Congratulations. You have successfully completed the assignment."

Below the page, the browser's developer tools Console tab is open, showing the following logs:

```
initialize goat app router
⚠ Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user's experience. For more help http://xhr.spec.whatwg.org/
entry
» alert(document.cookie);
< undefined
⚠ WARNING: Missing translation for key: "Congratulations. You have successfully completed the assignment."
⚠ WARNING: Missing translation for key: ""
»
```

Timestamps and file names are visible in the log entries:

- GoatRouter.js:88:18
- jquery.min.js:2:80629
- start.mvc:19:21
- polyglot.min.js:17:834
- polyglot.min.js:17:834

<script>alert("s?madhav0230")</script>

Shopping Cart Items -- To Buy Now

		Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00	
Dynex - Traditional Notebook Case	27.99	1	\$0.00	
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00	
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00	

Enter your credit card number:  
4128 3214 0002 1999

Enter your three digit access code:  
111

Purchase

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.  
Thank you for shopping at WebGoat.  
Your support is appreciated

We have charged credit card:  
-----  
\$1997.96

start.mvc#test/

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) >

Cross Site Scripting

(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 +

## Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are being "reflected" to the page.

For example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often times test code is very simple and lacks security or any quality controls).

Your objective is to find the route and exploit it. First though ... what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The CrossSiteScripting.lesson/9 after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

✓ start.mvc#test/ Submit

Correct! Now, see if you can send in an exploit to that route in the next assignment.

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

G 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.