Assignment 1

IDATT2503 - Security in programming and cryptography
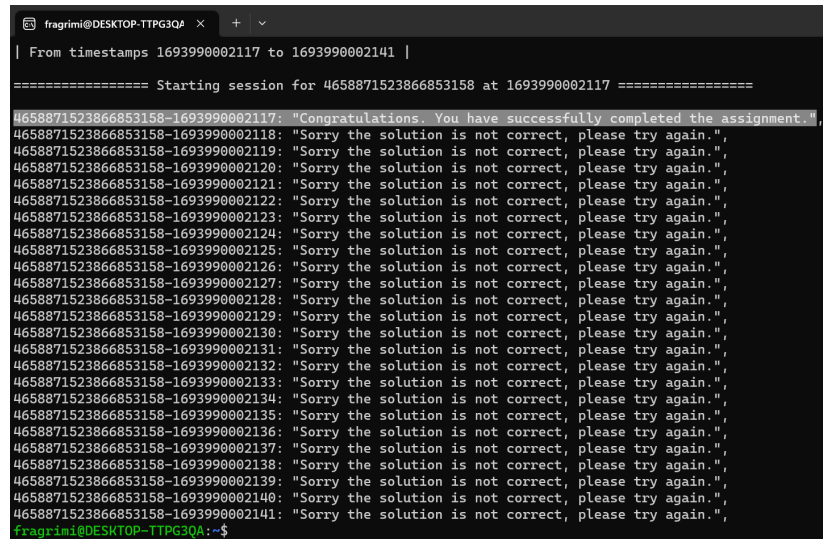### Fall 2023

Francesca Grimaldi

1 WebGoat

1.1 A1 Broken Access Control

1.1.1 Hijack a session

The goal of this challenge was to predict the value of the `hijack_cookie`. First, I tried to access using some random credentials, and I retrieved the value of this cookie and of the session ID. Then I tried to send other POST requests removing that cookie in order to generate other values of it. The first half of it was a sequential number incrementing by two each time, while the second one represented a Unix date timestamp with milliseconds. I solved the challenge bruteforcing the values with a script.

Figure 1: Hijack a session

1.1.2 Insecure Direct Object References

The main focus of this lesson was on intercepting and manipulating requests and URLs to get unauthorized access to data, such as to view and modify another user's profile.

2) Simply consisted in authenticating using the given credentials ('tom' and 'cat').

3) The request was to list two attributes that were present in the server's response but not in the page. Looking at the values displayed after clicking on `View Profile` and comparing them to the ones in the response, the missing ones were `role` and `userId`.
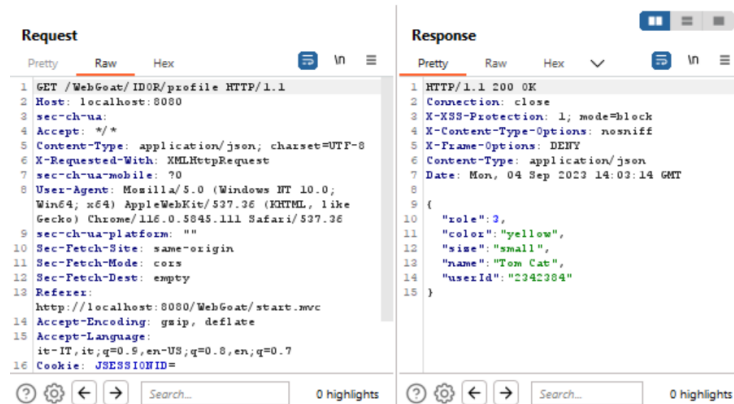
Figure 2: IDOR 3

4) To view my profile, I appended the `userId` found in the previous exercise to the URL of the GET request that retrieves the profile data: `WebGoat/IDOR/profile/2342384`

5) To view someone else's profile, I used the alternate path of the previous exercise but trying increasing the last digit. As I managed to find the profile of 'Buffalo Bill', I edited a PUT request in order to modify it.
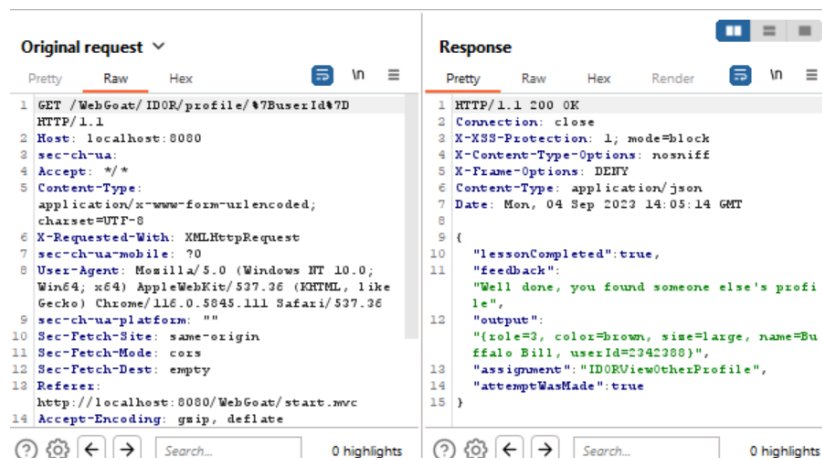
Figure 3: IDOR 5a

Figure 4: IDOR 5b

1.2   A3 Injection

1.2.1   SQL Injection (intro)

This lesson's objective was to use and manipulate Structured Query Language (SQL) to perform tasks that were not the developer's original purpose, in particular through String and Numeric SQL injection. In the following, I included the SQL queries used to complete the various requests.

2) Simply required to retrieve the department of a specific employee from the employees table:
```
SELECT department FROM employees WHERE (first_name='Bob'
AND last_name='Franco');
```

3) Required Data Manipulation Language (DML) to change the department of an employee:
```
UPDATE employees SET department='Sales' WHERE (first_name='Tobi'
AND last_name='Barnett');
```

4) Required Data Definition Language (DDL) to modify the schema and add a column to the table:
```
ALTER TABLE employees ADD phone varchar(20);
```

5) Required Data Control Language (DCL) to grant rights to a table to a specified user:
```
GRANT SELECT ON grant_rights TO unauthorized_user;
```

9) Required to complete a query in order to retrieve the complete list of users:
```
SELECT * FROM user_data WHERE first_name = 'John' and last_name
= 'Smith' or '1' = '1';
```

10) First, one had to find the Input Field susceptible to SQL injection (User_Id), and then take advantage of that to retrieve all the data.
```
Login_Count:  1
User_Id:  1 OR 1=1
```
So the complete query was:
```
SELECT * From user_data WHERE Login_Count = 1 and userid= 1 OR
1=1
```

11) Similarly to the previous one, the Input field susceptible to SQL injection was Authentication TAN, and to retrieve all the data:
```
Employee Name:  Smith
Authentication TAN: 3SL99A' OR 1=1;--
```

12) Here, the task was to change the salary of an employee modifying a table with an injection on the Authentication TAN field:
```
Employee Name:  Smith
Authentication TAN: 3SL99A'; UPDATE employees SET salary='99000'
WHERE auth_tan='3SL99A';--
```

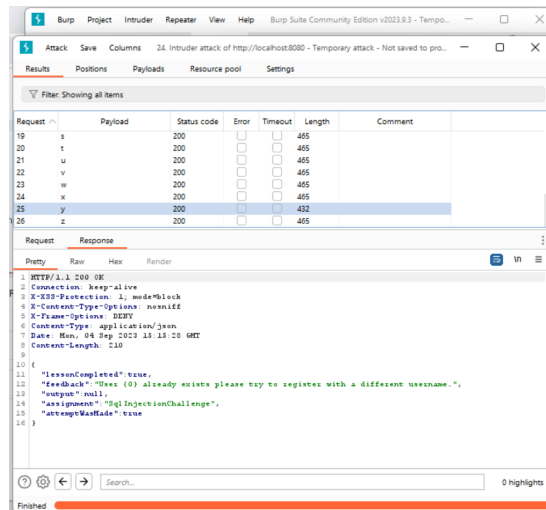13) Required deleting a table:
```
'; DROP TABLE access_log;--
```

1.2.2 SQL Injection (advanced)

The goal of this lesson was to execute Blind SQL injection and combined SQL injection techniques.

3) The task required to retrieve all the contents of the `user_system_data` table through a field susceptible to SQL injection, and then find the password of an user called Dave.
1st step: get all the data
Name: Smith' UNION SELECT userid, user_name, password, cookie, 'a', 'b', 1 FROM user_system_data;--
The values `'a'`, `'b'`, `1` needed to be added as the UNION operator requires the same number of columns on both tables.
2nd step: check the tuples and find the one with 'Dave' as `first_name`
In this case, the tuple was `105, dave, passW0rD, , a, b, 1`, and therefore `passW0rD` was the solution.

5) The objective was to login as 'Tom', having the possibility to play with both a login and a registration form. The field susceptible to SQL injection was the `Username` one in the registration form. In order to login, I tried to retrieve the user's password, bruteforcing inserting `tom' AND substring(password,1,1)='p` in that field. Depending on the response message it was clear whether the first letter of the password was correct or not ('the user already exists', or 'user successfully created'). I proceeded letter by letter with Burp Intruder, checking the response messages for a list of characters, until I found all of them.

Figure 5: SQL 5a

Figure 6: SQL 5b

6) This exercise was actually a quiz on theoretical aspects of SQL injection.
   I completed it with the following answers:
   `a) A statement has got values instead of a prepared statement`
   `b) ?`
   `c) Prepared statements are compiled once by the database management`
   `system waiting for input and are pre-compiled this way.`
   `d) Placeholders can prevent that the users input gets attached`
   `to the SQL query resulting in a seperation of code and data.`
   `e) The database registers 'Robert' ); DROP TABLE Students;--'.`

1.2.3 Path traversal

Here, path (directory) vulnerabilities are exploited to access or store files outside the application's location.

2) The goal was to overwrite a specific file on the file system. I did so by inserting `../test` in the `Full name` field.

3) Similar to the previous one, but with a fix that removed the sequence `../` from the input. It was not possible to use the same exploit, but inserting `.../.test` worked.

4) For this exercise, an additional fix was applied in order to solve the vulnerability of the `Full name` field. I intercepted the POST request when updating the profile and changed the file name to `../mypicture`.

5) The request was to find a file named `path-traversal-secret.jpg`. I managed to do a GET request to `WebGoat/PathTraversal/random-picture?id=%2e%2e%2f%2e%2e%2fpath`

-traversal-secret, and from the response I found out that I had to submit the SHA-512 hash of my username to solve the exercise.
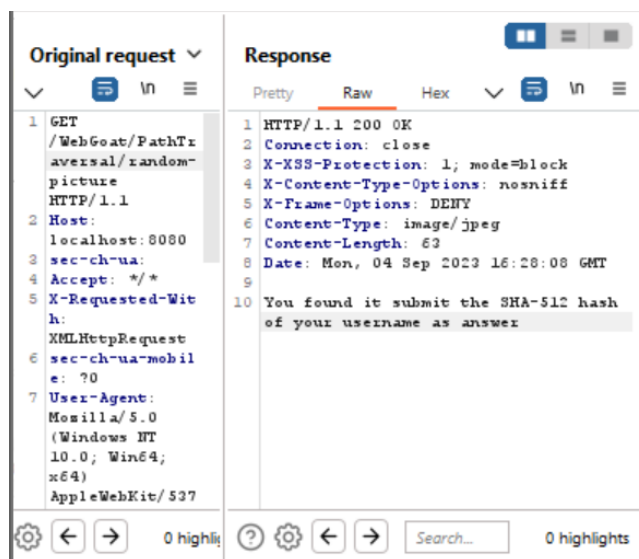

Figure 7: Path traversal 5

7) This exercise required to overwrite my current profile image, circumventing a programming mistake made in the code to extract the zip file. I created a directory structure and a zip file with my image inside of it, with the commands:
`mkdir -p /home/webgoat/.webgoat-2023.4/PathTraversal/fragrimi`
and `zip profile.zip ../../../../../../../../home/webgoat/`
`.webgoat-2023.4/PathTraversal/fragrimi/fragrimi.jpg`.
Uploading this zip file solved the assignment.

2   Hacker101

2.1   Micro-CMS v1

For this web CTF, I found the four flags in the following way:

- For the first one, I entered the `Testing` page and tried editing it. I attempted to manually modify the ID at the end of the URL to see if I could access some other page. For most of them, it gave me a 404 error, but for one ID a `Private Page` appeared, with the flag.
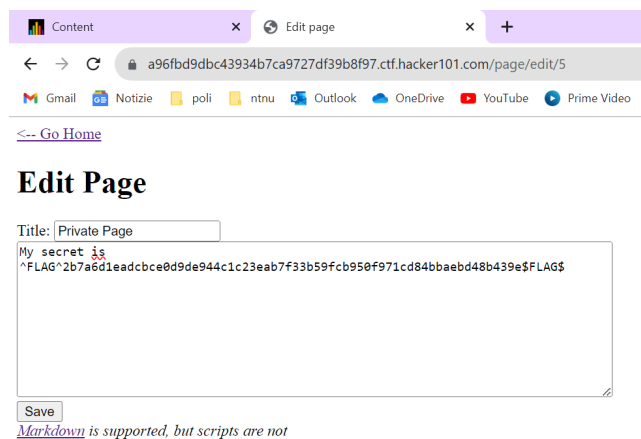
Figure 8: microCMS 1

- I got the second flag after trying to create a page with some script in the title (XSS). I added `<script>alert(a)</script>`, and as I returned to the homepage the script was executed and an alert with the flag was displayed.

- For the third flag, I entered the `Markdown Test` page and tried editing it. I added another script to be executed after clicking on the button: `<button onClick=alert(b)>Some button</button>`. The script was executed but it did not display the flag. Then, inspecting the page and viewing the source code, I found it in the button tag.

- With an SQL injection in the edit page, thus adding `';--` to the URL, the last flag appeared.

3 Screenshots

This section includes the screenshots of all the challenges marked as completed.

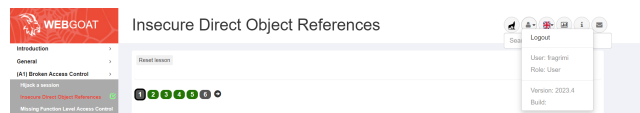Figure 9: Completed Hijack a session

Figure 10: Completed Insecure Direct Object References
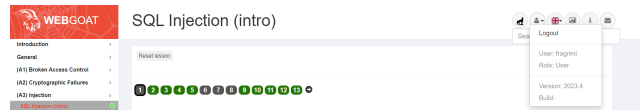

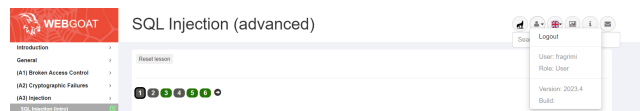Figure 11: Completed SQL (intro)

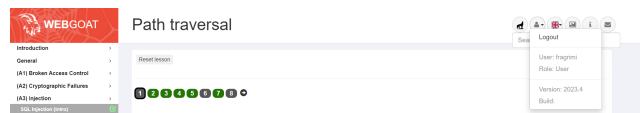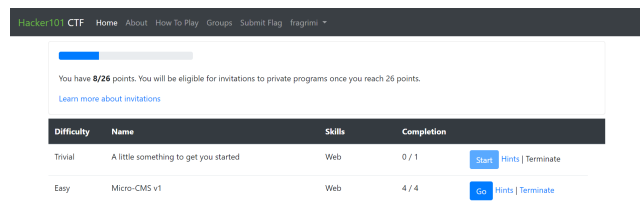
Figure 12: Completed SQL (advanced)

Figure 13: Completed Path traversal

Figure 14: Completed Micro-CMS v1