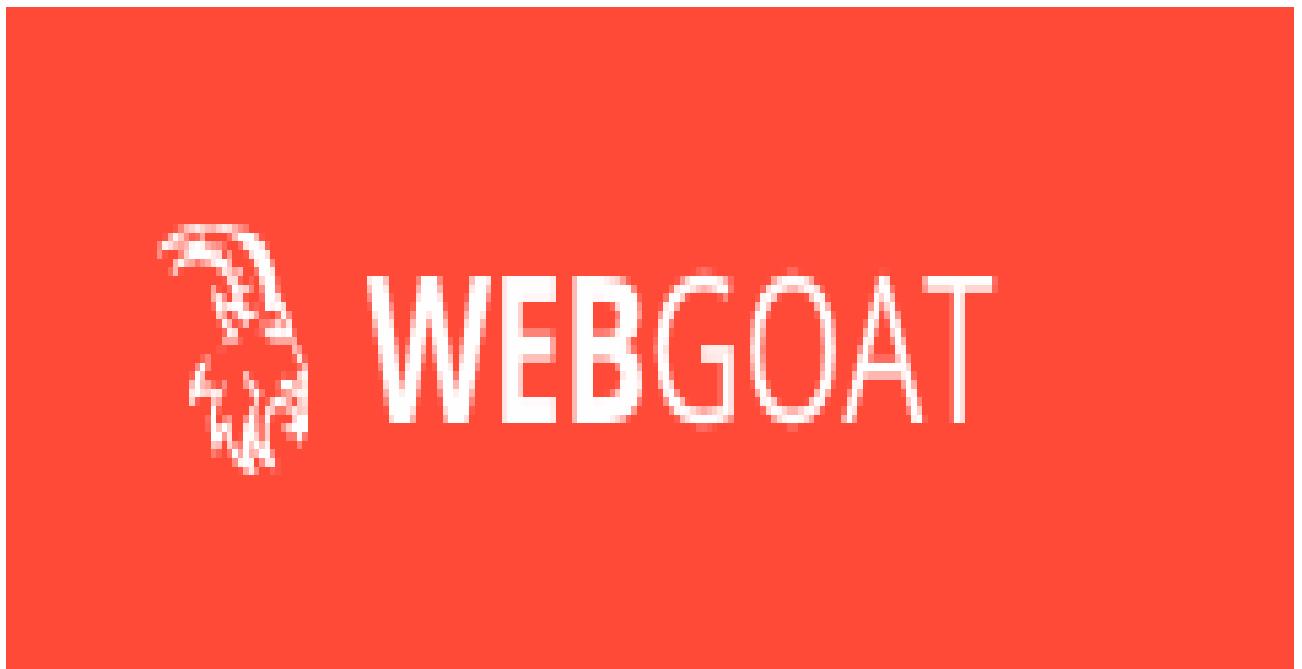


INFORME INTRODUCCION A CIBERSEGURIDAD



Alumno: Brayan Gabriel Gutiérrez
Rebolledo

Módulo: Introducción a la Ciberseguridad

Profesor: Carlos Cilleruelo

Fecha: 18 de enero de 2026

ÍNDICE

Ámbito y alcance de la auditoría.....	3
Informe ejecutivo.....	4
Descripción del proceso de auditoría.....	6
Sistema operativo.....	7
Código(Payload).....	13
Post-explotación.....	19
Conclusión final.....	20

Ámbito y alcance de la auditoría

El objetivo de esta práctica fue realizar una auditoría web básica sobre la aplicación vulnerable WebGoat (v8.1.0), desplegada localmente mediante Docker, para identificar y explotar vulnerabilidades solicitadas en el enunciado, incluyendo evidencias, criticidad e ideas de mitigación.

Objetivo (target): <http://127.0.0.1:8080/WebGoat>

Entorno: Laboratorio local (Docker)

Limitaciones: Algunos ejercicios del laboratorio pueden presentar fallos. El apartado A6 (Vulnerable & Outdated Components) no pudo completarse por un error del ejercicio, se documenta el comportamiento observado.

Informe ejecutivo

2.1 Breve resumen del proceso realizado

Se desplegó WebGoat en un entorno local y se realizó un reconocimiento para identificar puertos abiertos, sistema operativo y tecnologías visibles. Luego se ejecutaron los ejercicios exigidos por la práctica, explotando SQL Injection, XSS, el ejercicio indicado en A5 (solicitud no autorizada/CSRF) y el apartado de autenticación (Secure Passwords).

2.2 Vulnerabilidades destacadas (con criticidad)

Hallazgo	Impacto	Criticidad
SQL Injection (A3 – Apartado 11)	Acceso/extracción de datos desde la base de datos	ALTA
XSS reflejado (A3 – Apartado 7)	Ejecución de scripts en navegador; riesgo de robo de sesión	ALTA
A5 – Solicitud no autorizada (CSRF)	Acciones en nombre de un usuario autenticado	MEDIA
A7 – Secure Passwords	Mayor riesgo de fuerza bruta/diccionario y compromiso de cuentas	MEDIA
A6 – Vulnerable & Outdated Components	No evaluable por fallo del laboratorio	N/A

2.3 Conclusiones

Se observaron vulnerabilidades de alto impacto asociadas a validación insuficiente de entradas (SQLi/XSS). En un escenario real, estos fallos podrían causar fuga de información, secuestro de sesión y manipulación de datos.

2.4 Recomendaciones

- Parametrizar consultas SQL y evitar concatenar entradas de usuario.
- Validar/sanitizar entradas y aplicar codificación de salida (output encoding) en vistas.
- Implementar tokens anti-CSRF y endurecer cookies (SameSite, Secure, HttpOnly).
- Aplicar políticas robustas de contraseñas y controles anti-fuerza bruta (rate limiting / lockout).

2.5 Nivel de riesgo global

Riesgo global estimado: ALTO, por la presencia de hallazgos explotables de tipo SQL Injection y XSS.

Descripción del proceso de auditoría

3.1 Reconocimiento / Information Gathering

En esta fase se desplegó el entorno y se recopiló información técnica del servicio: puertos expuestos, sistema operativo y tecnologías observables desde el lado cliente.

Primera parte

En primer lugar se debe ejecutar este entorno la recomendación es hacerlo con Docker:

- docker run --name webgoat -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 -e TZ=Europe/Amsterdam webgoat/webgoat
- docker start webgoat
- La aplicación a auditar se encuentra en <http://127.0.0.1:8080/WebGoat>

Se aplican los pasos mencionados recientemente

Segunda parte Reconocimiento

Una vez desplegada la aplicación se realizará un reconocimiento sobre la misma

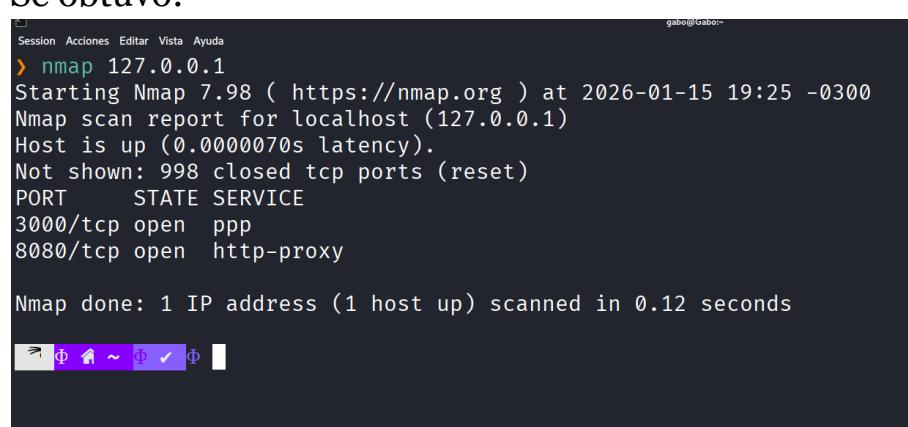
Identificando la máxima información posible, como:

- Puertos abiertos
- Sistema Operativo
- Lenguajes de programación utilizados en la aplicación web

PUERTOS ABIERTOS

Se utilizó nmap para hacer el reconocimiento:
en el terminal se escribió: “nmap 127.0.0.1”

Se obtuvo:



```
Session  Acciones  Editar  Vista  Ayuda
> nmap 127.0.0.1
Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-15 19:25 -0300
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000070s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
3000/tcp   open  ppp
8080/tcp   open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds
```

Aquí nos enfocamos en el puerto 8080 debido a que fue ese el puerto usado para webgoat, entonces 8080/tcp es de webgoat

SISTEMA OPERATIVO

Para saber el sistema operativo utilizaremos el comando: docker exec webgoat uname -a
de esta forma obteniendo:



```
Session Acciones Editar Vista Ayuda
> docker exec webgoat uname -a
Linux 8ffd24309915 6.17.10+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.17.10-1kali1 (2025-12-08) x86_64 x86_64 x86_64 GNU/Linux
root@Gabo:/home/gabo
```

Linux 8ffd24309915 6.17.10 kali-amd64

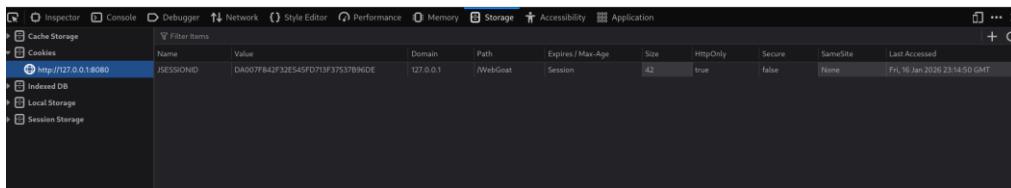
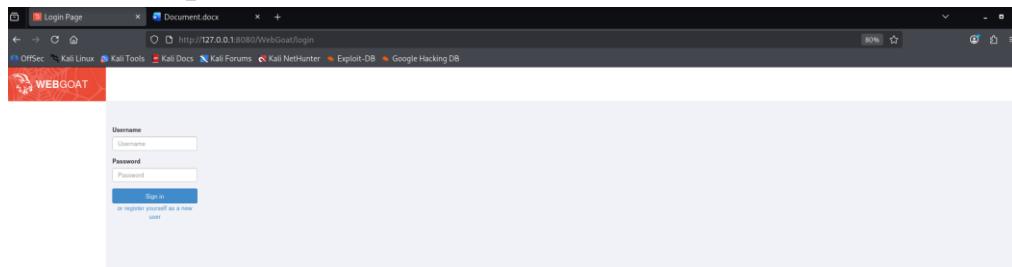
Figura 2: Evidencia de reconocimiento

Esto indica que el sistema operativo es **Linux**

LENGUAJES DE PROGRAMACION UTILIZADOS

Si utilizamos inspeccionar elementos en la pagina de webgoat (Apretando F12) en la sección de Almacenamiento (Storage) o Aplicacion(Application), clickeamos en:

Cookies → <http://127.0.0.1:8080> y buscando “SESSIONID” sabremos que es Java.



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
JSESSIONID	DA007F842F32E545FD713F37537B96DE	127.0.0.1	/WebGoat	Session	42	true	false	None	Fri, 19 Jan 2024 23:14:50 GMT

Filter Items										
	Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
http://127.0.0.1:8080	JSESSIONID	DAo07F842F32E545FD713F37537B96DE	127.0.0.1	/WebGoat	Session	42	true	false	None	Fri, 16 Jan 2026 23:14:50 GMT

Si vemos mas de cerca veremos esto:

JSESSIONID = **DAo07F842F32E545FD713F37537B96DE**

Evidencia de reconocimiento

Para confirmar que JSSESSIDONID es de java se investigo en una pagina web la cual es la siguiente:

domingo, 27 de noviembre de 2011

Más sobre el jsessionid

Curioseando los estats de este blog, me percato que existen muchos accesos en base a la búsqueda del término "jsessionid". Intuyo que casi todos ellos originados por el post ["Agregando el jsessionid a un HTML estático"](#) que, debo decir en mi defensa, fue escrito hace algunos años y en [condiciones](#) de estrés considerables.

Así que me tomé la libertad de crear este otro post aclarando, qué es el jsessionid, y algunas particularidades sobre él.

El jsessionid es, como el nombre indica, un identificador de la sesión en un [contenedor Java](#) (generalmente JEE / Java EE). O un Servidor de Aplicaciones. Y pues generalmente se almacena en una [cookie](#), aunque claro, existen [ambientes](#) donde no tenemos cookies.

3.2 Explotación de vulnerabilidades detectadas

3.2.1 A3 Injection – SQL Injection (Apartado 11)

Criticidad: **ALTA**. Motivo: acceso a datos y posible extracción de información desde la base de datos.

Detección

En este apartado se trabaja una introducción a SQL Injection.

El sistema muestra datos de un empleado usando su apellido y un TAN.

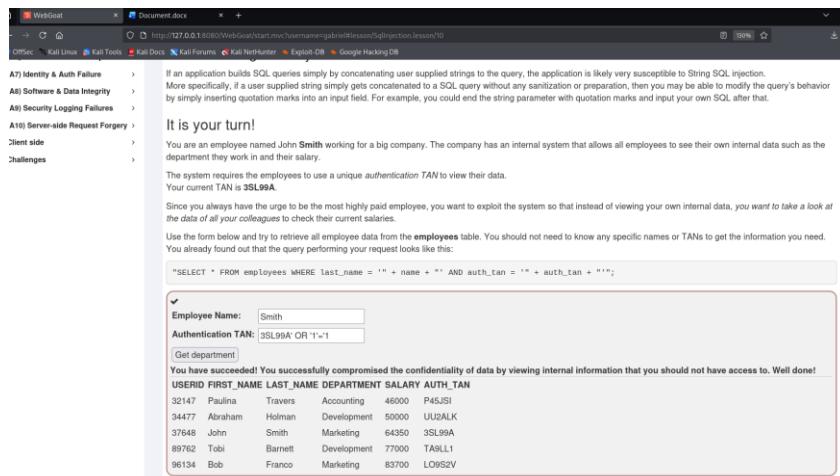
Datos entregados por el ejercicio:

- Nombre: John Smith
- TAN: 3SL99A

Al revisar cómo funciona, se nota que la consulta SQL se arma directamente con lo que ingresa el usuario, sin validaciones en el input y esto deja la posibilidad de una inyección SQL de tipo string.

El principal problema es que se podría acceder a información de otros empleados, afectando la confidencialidad del sistema.

En términos resumidos todo lo que se ingresa en el input pasa a ser parte del query SQL entonces basta con ponerle una comilla al input y permite cerrar el string original y alterar la lógica de la consulta SQL como en este caso que se uso para buguear el filtro de los empleados y poder ver todos los datos en vez de solo el propio.



Evidencia de explotación/resultado de SQL Injection.

Post Explotacion

Tras la explotación fue posible acceder a información de todos los empleados, lo que podría derivar en fuga de datos sensibles.

Mitigacion

No se deben armar consultas SQL concatenando lo que ingresa el usuario, lo correcto sería usar consultas que sean preparadas para que los datos no sean interpretados como parte del SQL y tambien validar formato del TAN ademas de usar una cuenta de base de datos con los permisos justos.

3.2.2 A3 Injection – Cross Site Scripting (Apartado 7)

Criticidad: **ALTA**. Motivo: ejecución de scripts en el navegador; posible robo de sesión/datos.

Detección

En este apartado se trabaja una introducción a la vulnerabilidad **Cross-Site Scripting (XSS) reflejado.**

El ejercicio consiste en identificar qué campo es vulnerable a XSS.

Al probar la aplicación, se observa que algunos datos ingresados por el usuario son devueltos directamente en la página sin ningún tipo de validación.

En la sección de ‘**Enter your credit card number:**’ se ingreso el siguiente script: <**script>alert("XSS Test")</script>**

Lo que dio como resultado una alerta de notificación diciendo “**XSS TEST**” al enviar el formulario, confirmando que el contenido ingresado se refleja en la página y se ejecuta como código, de esta forma se indica que el campo no filtra ni sanitiza correctamente la entrada del usuario, siendo vulnerable a **XSS reflejado.**

The screenshot shows a browser window with the URL <http://127.0.0.1:8080/WebGoat/start.mvc?username=gabriel&lesson=CrossSiteScripting.lesson/6>. The page title is "Try It! Reflected XSS". On the left, there's a navigation sidebar with sections like Introduction, General, SQL Injection (intro), SQL Injection (mitigation), Cross Site Scripting, Cross Site Scripting (stored), and Cross Site Scripting (mitigation). The main content area has a heading "Try It! Reflected XSS" and a sub-section "Shopping Cart". A table titled "Shopping Cart Items – To Buy Now" lists items with columns for Price, Quantity, and Total. Below the table, there are two input fields: one for "Enter your credit card number:" containing "<script>console.log('holo')</script>" and another for "Enter your three digit access code:". A "Purchase" button is visible. At the bottom, a message says "Congratulations, but console logs are not very impressive are they? Let's continue to the next assignment. Thank you for shopping at WebGoat. Your support is appreciated." and "We have charged credit card: \$5909.88".

Evidencia de XSS reflejado

Post explotación

Una vez explotada la vulnerabilidad, el código injectado se ejecuta con los permisos del navegador del usuario afectado, esto puede permitir el acceso a información sensible y comprometer la confidencialidad e integridad de la sesión del usuario ademas de robos como por ejemplo las cookies, que son muy importantes por inicios de sesión no autorizados, ya que el cookies recuerda que se ha

iniciado sesión para que no se tenga que introducir los datos en cada página.

Mitigacion

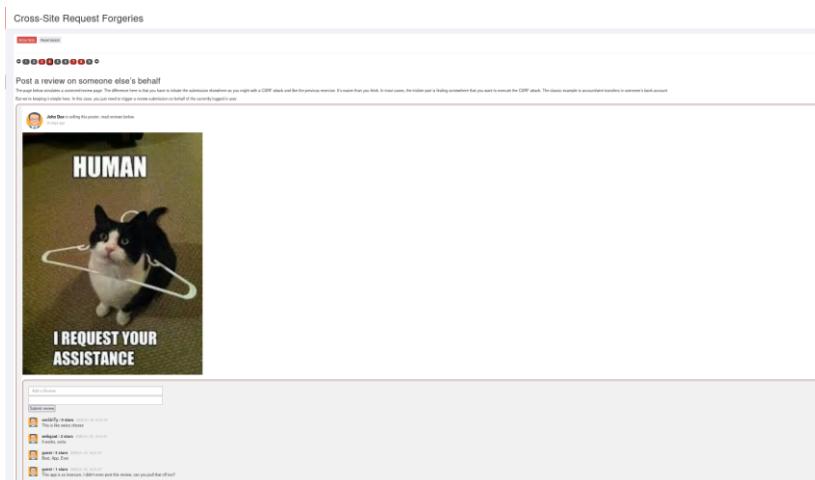
Al igual que en SQL Injection, la vulnerabilidad se produce por una mala validación de la entrada del usuario, aunque en el caso de XSS el impacto se da en el navegador y no en la base de datos.

3.2.3 A5 Security Misconfiguration (Apartado 4) – Solicitud no autorizada (CSRF)

Criticidad: **MEDIA**. Motivo: ejecución de acciones en nombre del usuario autenticado bajo ciertas condiciones.

Detección

En este apartado se trabaja con **Cross-Site Request Forgeries**, el cual su función es realizar peticiones de una víctima desde una página externa sin necesidad de que la víctima realice alguna acción.



En este caso se nos pide activar el envío de una reseña en nombre del usuario que ha iniciado sesión actualmente, entonces abrimos con F12 las herramientas de desarrollador y nos vamos a **Network**, antes se escribió “que gato mas mono!” y abajo un “5” porque ahí van las estrellas que se le pone a la resena.

A screenshot of a review submission interface. At the top, it says "I REQUEST YOUR ASSISTANCE". Below that is a form with a text input containing "que gato mas mono!" and a dropdown set to "5". There is a "Submit review" button. Below the form, there are two existing reviews: one from "secUriTy / 0 stars" dated 2026-01-18, 17:06:01, and another from "webgoat / 2 stars" dated 2026-01-18, 17:06:01. Both reviews say "This is like swiss cheese" and "It works, sorta" respectively. At the very bottom, there is a footer with the number "11".

Luego damos en **Submit Review**, veremos que en **Network** se actualizara la petición y aparecera con el nombre de **review**, nos fijamos despues en la parte derecha en **Request → Form Data**

The screenshot shows a review submission interface with a text input containing "que gato mas mono!" and a dropdown menu set to "5". Below the form, a list of reviews is displayed. In the browser's developer tools, the Network tab is selected, showing a POST request to "review" with the same data. The "Request" section of the Network tab shows the raw data: "reviewText: \"que gato mas mono!\"", "stars: \"5\"", and "validateReq: \"2aa14227b9a13d0bede0388a7fba9aa9\"".

Tenemos los datos necesarios para continuar con la explotacion:

Form data

reviewText: “que gato mas mono”

stars: “5”

Ahora, se da click derecho en review →Copy Value → Copy as Fetch

The screenshot shows a context menu over a review entry with the "Copy Value" option selected. The menu includes "Copy URL", "Copy POST Data", "Copy as cURL", "Copy as PowerShell", "Copy as Fetch", "Copy Request Headers", "Copy Response Headers", "Copy Response", and "Copy All As HAR". The Network tab shows a list of requests, with the "review" POST request highlighted. The "Request" section of the Network tab shows the raw data: "reviewText: \"que gato mas mono!\"", "stars: \"5\"", and "validateReq: \"2aa14227b9a13d0bede0388a7fba9aa9\"".

Evidencia de captura y reproducción de petición

Esto lo que hara es copiarnos el codigo javascript el cual realiza la accion la cual hicimos recientemente de publicar la reseña

Código(Payload):

```
await fetch("http://127.0.0.1:8080/WebGoat/csrf/review",
{
  "credentials": "include",
  "headers": {
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0",
    "Accept": "*/*",
    "Accept-Language": "en-US,en;q=0.5",
    "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
    "X-Requested-With": "XMLHttpRequest",
    "Sec-Fetch-Dest": "empty",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Site": "same-origin",
    "Priority": "u=o"
  },
  "referrer": "http://127.0.0.1:8080/WebGoat/start.mvc?username=gabriel",
  "body": "reviewText=que+gato+mas+mono!&stars=5&validateReq=2aa14227b9a13dobede0388a7fba9aa9",
  "method": "POST",
  "mode": "cors"
});
```

Con esto ya solo basta cambiar únicamente: "**body**":
"reviewText=que+gato+mas+mono!&stars=5&validateReq=2aa14227b9a13dobede0388a7fba9aa9"

Podemos cambiarlo a nuestro antojo, en este caso colocaremos: Keepcoding y 5 estrellas entonces quedaria asi:

body":

**"reviewText=Keepcoding&stars=5&validateReq=2aa14227
b9a13dobedeo388a7fba9aa9"**

Se aprecia que solo modifique en el **body** exclusivamente **reviewText, stars** ya tenia el valor **5** por lo tanto no fue necesario cambiarlo, el **validateReq** en ninguna cirsuntancia hay que modificarlo.

Resultado:

```
await fetch("http://127.0.0.1:8080/WebGoat/csrf/review",
{
  "credentials": "include",
  "headers": {
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64;  
rv:140.0) Gecko/20100101 Firefox/140.0",
    "Accept": "*/*",
    "Accept-Language": "en-US,en;q=0.5",
    "Content-Type": "application/x-www-form-
urlencoded; charset=UTF-8",
    "X-Requested-With": "XMLHttpRequest",
    "Sec-Fetch-Dest": "empty",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Site": "same-origin",
    "Priority": "u=0"
  },
  "referrer":  
  "http://127.0.0.1:8080/WebGoat/start.mvc?username=gabriel",
```

```

"body":  

"reviewText=Keepcoding&stars=5&validateReq=2aa14227  

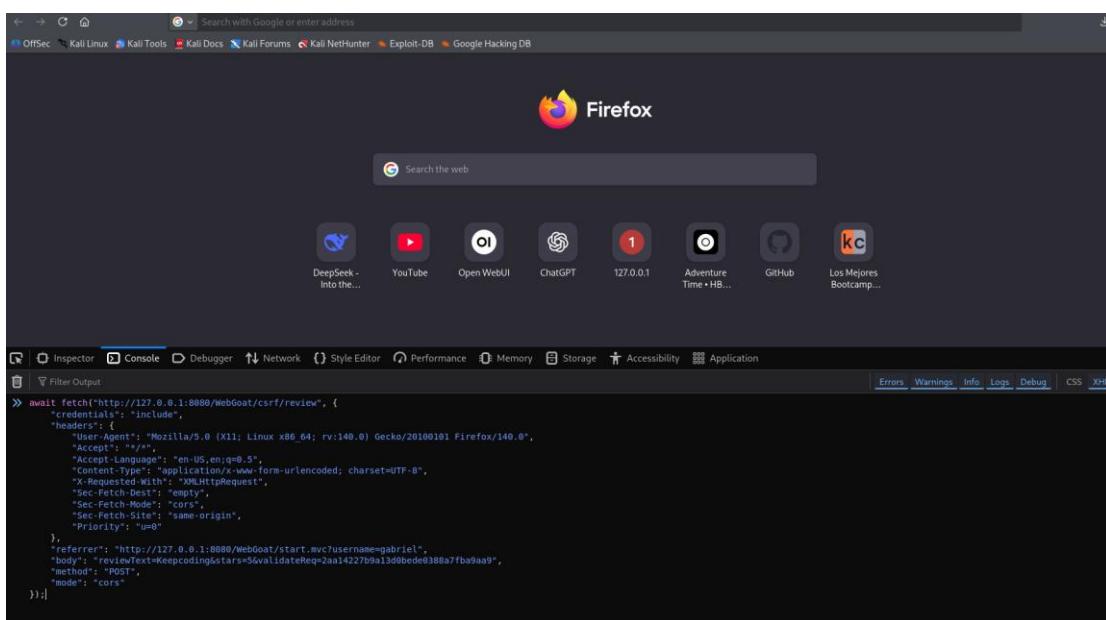
b9a13dobede0388a7fba9aa9",  

"method": "POST",  

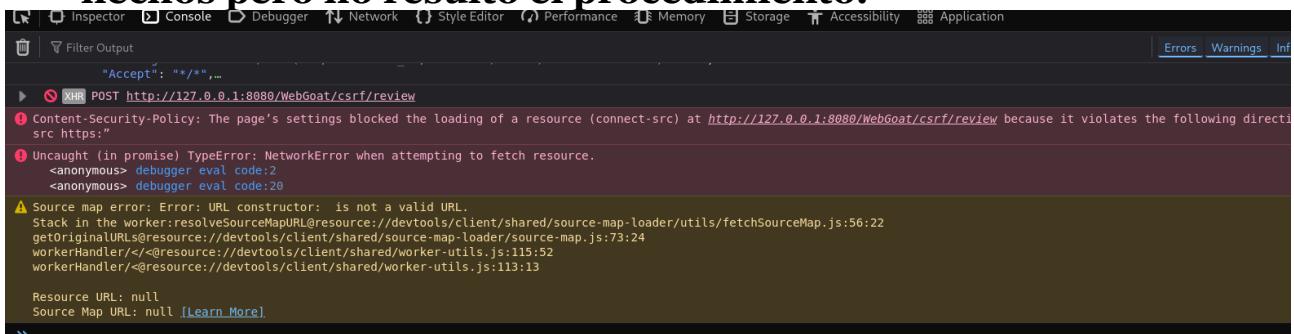
"mode": "cors"  

});
```

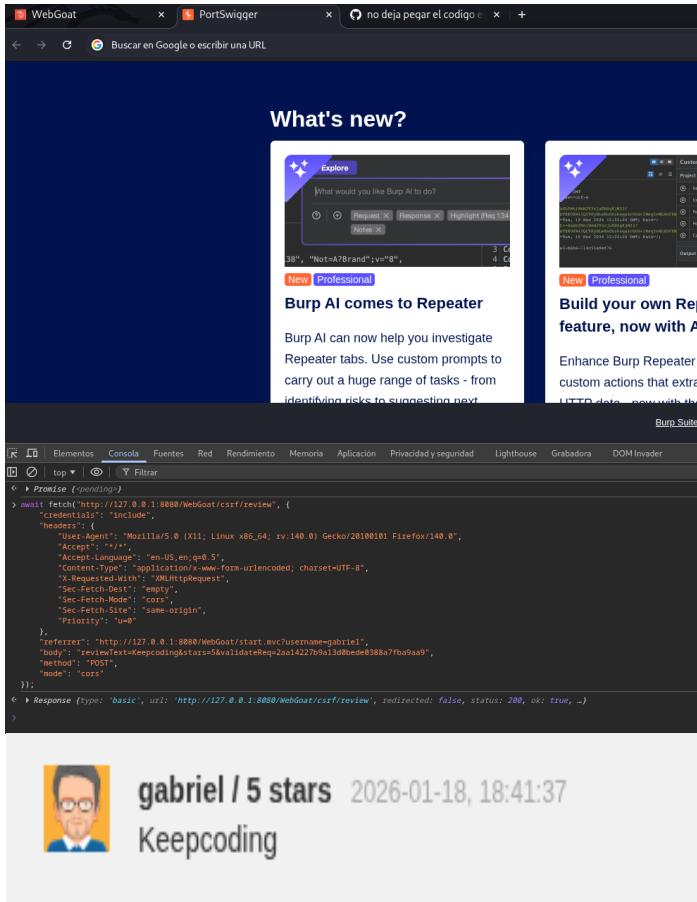
Ahora copiando todo el código, nos vamos a una nueva página y en la consola del desarrollador colocamos el código:



Desafortunadamente nos dio error, los pasos estan bien hechos pero no resulto el procedimiento.



Sin embargo nos cambiamos de navegador, en este caso Chromium y si nos funciono los mismos pasos



En conclusión:

El código que se obtuvo permitió emitir una solicitud sin autorización de la víctima mediante consola en una web externa. Depende de que navegador se utilice para realizar la ejecución del script, en el caso de Firefox no permite su ejecución por políticas de seguridad implementadas (Bloqueó por CSP estricto (connect-src https:)), en cambio chromium si nos dejó deliberadamente realizarlo.

Post explotación

- Publicación de contenido no autorizado
- Posible escalación si la funcionalidad fuera crítica (transferencias, cambios de configuración)

Mitigación

1) Poner un Código Secreto en Cada Formulario

- Como un código de verificación que cambia cada vez
- El servidor revisa que el código sea correcto
- Si no coincide, rechaza la petición

2) Configurar las Cookies Bien

Las cookies deben tener:

- SameSite=Strict → No se envían desde otras páginas
- Secure → Solo por HTTPS
- HttpOnly → No accesible por JavaScript

3.2.4 A7 Identity & Auth Failure – Secure Passwords (Apartado 4)

Criticidad: **MEDIA**. Motivo: contraseñas débiles aumentan el riesgo de fuerza bruta/diccionario.

En este apartado se realizo una demostracion de como las contraseñas pueden ser vulneradas por fuerza bruta como ataque de diccionario, mostrando la importancia de crear contraseñas de longitudes largas y con caracteres variados

En la primera imagen se puede apreciar que se ingresa una contrasena facil y sencilla de longitud 7 que es **zapallo**:

The screenshot shows the WEBGOAT interface with the 'Secure Passwords' assignment selected. The sidebar contains a tree view of security topics. The main content area asks 'How long could it take to brute force your password?' and lists several bad password examples. A red box highlights the input field where 'zapallo' has been typed. Below the input field, an error message says 'You have failed! Try to enter a secure password.' and provides a score of 2/4.

Evidencia del análisis de contraseñas (débil vs. fuerte).

En la segunda imagen se aprecia que se ingresa una contrasena mas larga de longitud teniendo 31, la cual es:
passwordanotheronesupersecurity

The screenshot shows the WEBGOAT interface with the 'Secure Passwords' assignment selected. The main content area shows a successful password entry. The input field contains 'passwordanotheronesupersecurity'. The score is now 4/4, indicating success. The message 'You have succeeded! The password is secure enough.' is displayed.

De esta forma se concluye que es importante la longitud de las contraseñas debido a que son susceptibles a ataque de fuerza bruta como por ejemplo ataques de diccionario.

3.2.5 A6 Vulnerable & Outdated Components (Apartado 5)

Estado: **No completado** por error del ejercicio en WebGoat. Se documenta el resultado observado.

El apartado A6 no pudo ser ejecutado correctamente debido a un error en el funcionamiento del ejercicio dentro de la plataforma WebGoat. A pesar de los intentos realizados, el comportamiento esperado no se presentó, lo que impidió obtener evidencias claras para su documentación.

Debido a la limitación de tiempo y al correcto desarrollo de los demás apartados solicitados, se decidió continuar con el resto de la práctica.

3.3 Post-explotación

Tras la explotación se evidenció: (1) acceso a información de otros usuarios mediante SQLi, (2) ejecución de scripts en navegador (XSS) y (3) ejecución de solicitudes en nombre del usuario autenticado (A5/CSRF). En un entorno real esto podría derivar en fuga de datos, secuestro de sesión o modificación de información.

3.4 Posibles mitigaciones

- SQL Injection: consultas parametrizadas, validación de entradas y mínimo privilegio en la cuenta de BD.
- XSS: sanitización/validación de entradas, output encoding, CSP y evitar reflejar entradas sin escapar.
- CSRF: tokens anti-CSRF, cookies SameSite y validación de origen cuando corresponda.
- Contraseñas: longitud mínima, complejidad, hashing robusto y controles anti-fuerza bruta.
- Componentes: inventario de dependencias y actualización periódica con gestión de vulnerabilidades.

3.5 Herramientas utilizadas

- Docker: despliegue del laboratorio WebGoat.
- Nmap: reconocimiento de puertos.
- Navegador (DevTools): análisis de cookies/peticiones y reproducción de solicitudes (fetch).
- WebGoat: plataforma vulnerable y ejercicios guiados.

Conclusión final

La práctica permitió aplicar un flujo básico de auditoría web: despliegue, reconocimiento, explotación y documentación. Los hallazgos muestran cómo fallos comunes (SQLi/XSS y controles insuficientes en solicitudes) pueden comprometer la confidencialidad e integridad. Se recomienda aplicar las mitigaciones propuestas y reforzar validación, controles de sesión y seguridad en la capa web.