Realizado por Michelli Melo

KeepCoding | Bootcamp de Ciberseguridad XI

17 de enero de 2026

INFORME DE AUDITORÍA DE SEGURIDAD

OWASP WEBGOAT 8.1.0

Índice

1. Ámbito y alcance de la auditoría

1.1 Ámbito de la auditoría

La presente auditoría de seguridad se ha realizado sobre la aplicación **WebGoat**, un entorno de aprendizaje deliberadamente vulnerable desarrollado por OWASP, desplegado en un entorno local de laboratorio. El objetivo de la auditoría es analizar la seguridad de la aplicación web, identificando debilidades alineadas con el **OWASP Top 10**, siguiendo un enfoque práctico basado en reconocimiento, explotación y análisis de impacto.

La auditoría se ha llevado a cabo desde el punto de vista de un usuario con acceso a la aplicación web, sin acceso al código fuente en producción ni a infraestructuras externas al entorno del laboratorio. Las pruebas se han realizado con fines académicos y formativos, en un entorno controlado.

Las vulnerabilidades identificadas y explotadas se documentan de forma detallada en el **Anexo Técnico**, mientras que en el cuerpo principal del informe se describe el proceso general seguido durante la auditoría.

1.2 Alcance técnico

El alcance técnico de la auditoría se centró en el análisis de la **aplicación web WebGoat** desde el punto de vista de seguridad, evaluando su comportamiento frente a ataques comunes en aplicaciones web. Las pruebas se realizaron sobre la aplicación en ejecución, accediendo a sus funcionalidades a través de los servicios web expuestos y utilizando técnicas de auditoría manual apoyadas por herramientas de seguridad.

El análisis incluyó una fase de reconocimiento inicial para comprender el entorno de ejecución y la superficie de ataque, así como la **explotación controlada de vulnerabilidades** alineadas con las categorías del OWASP Top 10 proporcionadas por el propio laboratorio. Las pruebas se limitaron a los escenarios intencionadamente vulnerables de la aplicación y no incluyeron técnicas destructivas ni ataques de denegación de servicio reales.

Explotación controlada de vulnerabilidades correspondientes a las siguientes categorías OWASP, cuyo detalle técnico se recoge en el anexo:

- A2 – Cryptographic Failures

- A3 – Injection (SQL Injection y Cross Site Scripting)
- A5 – Security Misconfiguration (CSRF)
- A6 – Vulnerable & Outdated Components
- A7 – Identity & Authentication Failures
- A8 – Insecure Deserialization

La auditoría se llevó a cabo en un entorno local de laboratorio, sin impacto sobre sistemas de producción ni infraestructuras externas, y con un alcance estrictamente formativo y académico.

1.3 Exclusiones

Quedan fuera del alcance de esta auditoría el análisis de código fuente no expuesto en el laboratorio, la evaluación de infraestructuras externas al entorno WebGoat, la realización de pruebas de denegación de servicio reales y cualquier auditoría de cumplimiento normativo o legal.

---

2. Informe ejecutivo

2.1 Breve resumen del proceso realizado

La auditoría de seguridad se llevó a cabo sobre la aplicación web **WebGoat**, desplegada en un entorno local controlado, con el objetivo de identificar y explotar vulnerabilidades representativas del **OWASP Top 10**. El proceso comenzó con una fase de **reconocimiento**, en la que se analizaron los puertos expuestos, los servicios disponibles, el sistema operativo, el entorno de ejecución, las cabeceras HTTP y la estructura general de la aplicación.

Una vez delimitada la superficie de ataque, se procedió a la **detección y explotación controlada de vulnerabilidades**, siguiendo los ejercicios guiados proporcionados por la plataforma. Las pruebas realizadas permitieron explotar con éxito múltiples categorías de vulnerabilidades, incluyendo fallos criptográficos, inyecciones, configuraciones inseguras, uso de componentes vulnerables, debilidades en autenticación y deserialización insegura.

Todas las vulnerabilidades explotadas fueron documentadas de forma detallada en el **Anexo Técnico**, incluyendo pruebas realizadas, evidencias y referencias a estándares de seguridad. El presente informe resume el proceso seguido y los principales resultados obtenidos, sin entrar en el detalle técnico exhaustivo de cada explotación.

2.2 Vulnerabilidades destacadas

Durante la auditoría se identificaron y explotaron las siguientes vulnerabilidades relevantes, alineadas con el **OWASP Top 10**, que representan riesgos significativos para la seguridad de la aplicación:

- **A2 – Cryptographic Failures**: uso de hashes de contraseñas sin salt, lo que permitió la recuperación directa de credenciales en claro mediante ataques de diccionario.

- **A3 – Injection (SQL Injection y XSS)**: inyecciones SQL que permitieron el bypass completo de controles de acceso y la exposición de información sensible, así como vulnerabilidades de Cross Site Scripting que posibilitan la ejecución de código JavaScript en el navegador del usuario.

- **A5 – Security Misconfiguration (CSRF)**: ausencia de mecanismos efectivos de protección CSRF, permitiendo la ejecución de acciones en nombre de usuarios autenticados sin su consentimiento.

- **A6 – Vulnerable & Outdated Components**: uso de componentes externos vulnerables y desactualizados, concretamente jQuery UI, que permitió la explotación de XSS sin modificar el código propio de la aplicación.

- **A7 – Identity & Authentication Failures**: debilidades en la gestión de contraseñas que incrementan el riesgo de ataques de fuerza bruta y compromiso de cuentas.

- **A8 – Insecure Deserialization**: deserialización insegura de objetos Java, permitiendo alterar el comportamiento del servidor durante el proceso de reconstrucción de objetos en memoria.

El análisis técnico detallado, las pruebas realizadas y las evidencias asociadas a cada vulnerabilidad se recogen en el **Anexo Técnico**.

2.3 Conclusiones

La auditoría realizada sobre la aplicación WebGoat ha permitido constatar la presencia de múltiples vulnerabilidades críticas alineadas con el **OWASP Top 10**, que afectan de forma directa a la confidencialidad, integridad y control de acceso de la aplicación.

Las pruebas realizadas demuestran que la explotación de estas vulnerabilidades **no requiere conocimientos avanzados ni acceso privilegiado**, lo que incrementa significativamente el riesgo en un entorno real. En especial, las vulnerabilidades de inyección, deserialización insegura y fallos criptográficos evidencian deficiencias estructurales en el diseño y la implementación de mecanismos de seguridad fundamentales.

Asimismo, el uso de componentes vulnerables y la ausencia de protecciones adecuadas frente a ataques como CSRF reflejan la importancia de una **correcta gestión de dependencias, configuraciones y controles de seguridad** a lo largo del ciclo de vida de la aplicación.

En conjunto, los resultados de la auditoría ponen de manifiesto la necesidad de **aplicar medidas de seguridad** desde las fases iniciales de desarrollo y de mantener una estrategia continua de revisión y actualización para reducir la superficie de ataque y mitigar riesgos futuros.

---

3. Descripción del proceso de auditoría

3.1 Reconocimiento / Information Gathering

La fase de reconocimiento tuvo como objetivo recopilar la máxima información posible sobre la aplicación WebGoat antes de iniciar la explotación de vulnerabilidades, con el fin de comprender su contexto técnico, delimitar la superficie de ataque y orientar adecuadamente las pruebas posteriores.

3.1.1 Identificación de puertos y servicios

En primer lugar, se identificaron los puertos abiertos expuestos por la aplicación WebGoat en el entorno local. El análisis se realizó sobre los servicios asociados al despliegue del laboratorio, observándose que la aplicación se encuentra accesible a través de puertos utilizados para servicios web.

Los resultados obtenidos indican que los puertos abiertos están asociados a servicios **HTTP**, constituyendo los principales puntos de entrada a la aplicación web y a sus funcionalidades internas. Esta información permitió confirmar que la superficie de ataque inicial se limita a la propia aplicación web.

```
┌──(kali㉿kali)-[~]
└─$ docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}" | grep -i webgoat
webgoat   Up 2 hours (unhealthy)   127.0.0.1:8081→8080/tcp, 127.0.0.1:9091→9090/tcp

┌──(kali㉿kali)-[~]
└─$ nmap -sV -p 8081,9091 127.0.0.1
Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-05 05:53 -0500
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000050s latency).

PORT     STATE SERVICE VERSION
8081/tcp open  http    Apache Tomcat (language: en)
9091/tcp open  http    Apache Tomcat (language: en)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.25 seconds
```

Figura 1 – Identificación de puertos y servicios de la aplicación WebGoat.

3.1.2 Sistema operativo y entorno de ejecución

Durante la fase de reconocimiento se analizó el sistema operativo y el entorno de ejecución sobre el que se despliega la aplicación WebGoat. A partir de la información observada, se determinó que la aplicación se ejecuta sobre un sistema operativo **Linux**, dentro de un entorno de laboratorio basado en **contenedores Docker**.

Este tipo de despliegue permite aislar la aplicación del sistema anfitrión y facilita la reproducción controlada del entorno con fines formativos. La identificación del entorno de ejecución resulta relevante para contextualizar el análisis de seguridad, aunque la auditoría se centra exclusivamente en la aplicación web y no en la infraestructura subyacente.

```
┌──(kali㉿kali)-[~]
└─$ uname -a
Linux kali 6.17.10+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.17.10-1kali1 (2025-12-08) x86_64 GNU/Linux

┌──(kali㉿kali)-[~]
└─$ docker info | head -n 20
Client: Docker Engine - Community
 Version:    29.1.3
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.30.1
    Path:     /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version:  v5.0.1
    Path:     /usr/libexec/docker/cli-plugins/docker-compose

Server:
 Containers: 10
  Running: 2
  Paused: 0
  Stopped: 8
 Images: 7
 Server Version: 29.1.3
 Storage Driver: overlay2
```

Figura 2 – Sistema operativo y entorno de ejecución de la aplicación WebGoat.

3.1.3 Cabeceras HTTP y comportamiento de la aplicación

A continuación, se analizaron las **cabeceras HTTP** devueltas por la aplicación WebGoat con el objetivo de identificar información visible desde el lado cliente y comprender el comportamiento de la aplicación a nivel de comunicación.

Durante este análisis se observaron los códigos de estado HTTP utilizados, los mecanismos de redirección inicial y las cabeceras de respuesta asociadas a la entrega de contenido. Asimismo, se identificó que determinados servicios internos devuelven respuestas estructuradas, lo que permite inferir aspectos relacionados con el funcionamiento interno de la aplicación.

La información obtenida no implica una explotación directa, pero aporta contexto relevante para el análisis posterior de vulnerabilidades, especialmente en relación con configuraciones visibles y comportamiento esperado de la aplicación.

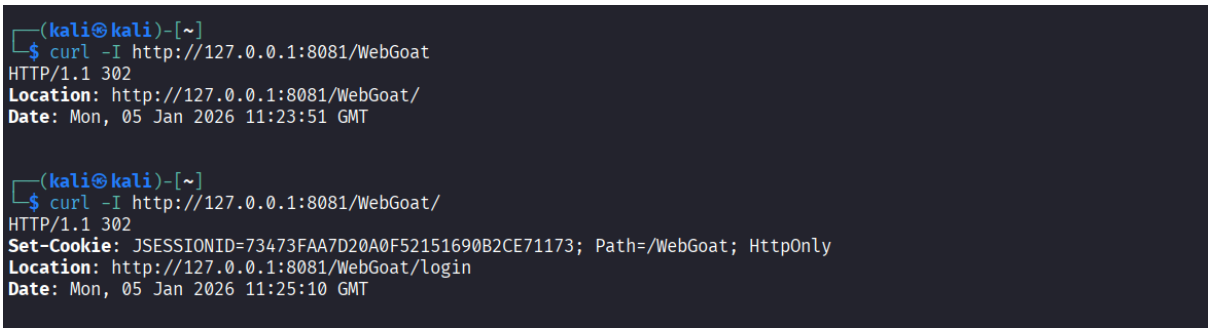

Figura 3 – Cabeceras HTTP y comportamiento de redirección de la aplicación WebGoat.

3.1.4 Estructura de la aplicación y endpoints accesibles

Finalmente, mediante el uso de **Burp Suite** como proxy de interceptación, se analizó la estructura general de la aplicación WebGoat y los **endpoints accesibles** durante la navegación. La aplicación presenta una estructura organizada en diferentes lecciones y funcionalidades, accesibles desde un menú principal que carga dinámicamente los contenidos.

Durante la observación del tráfico HTTP se identificaron múltiples endpoints utilizados para la carga de recursos y la gestión de funcionalidades internas. Algunos de estos endpoints devuelven respuestas en formato **JSON**, lo que evidencia la existencia de servicios internos utilizados por la propia aplicación.

La identificación de estos endpoints permitió conocer los puntos de interacción disponibles desde el lado cliente antes de iniciar la fase de explotación, facilitando la selección de vectores de ataque en las pruebas posteriores.

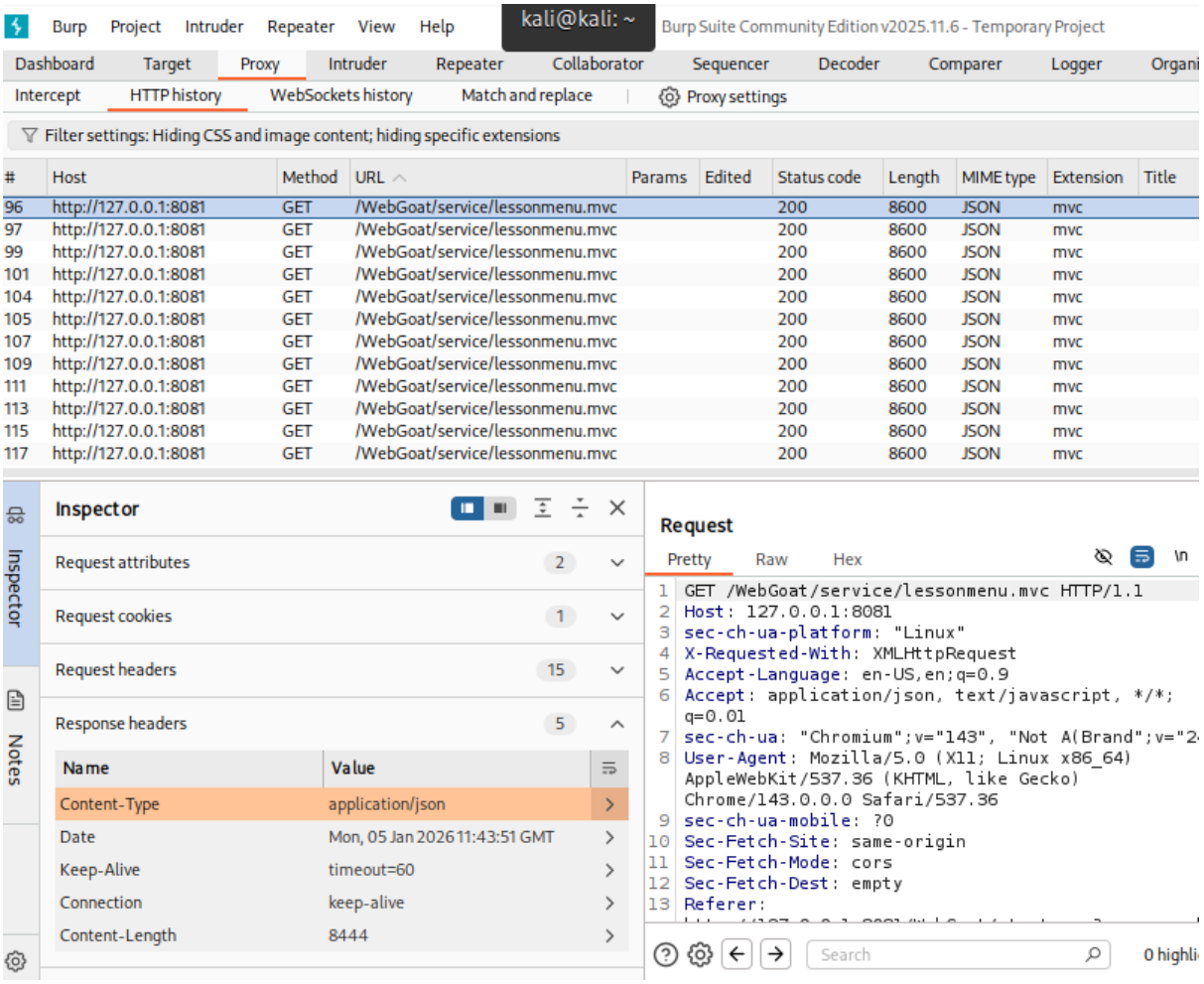

Figura 5 – Estructura de la aplicación y endpoints observados mediante Burp Suite.

3.2 Explotación de vulnerabilidades detectadas

En esta fase de la auditoría se procedió a la explotación controlada de las vulnerabilidades identificadas durante el reconocimiento, siguiendo los escenarios propuestos por la plataforma WebGoat y alineados con las categorías del **OWASP Top 10**.

Las pruebas realizadas permitieron confirmar la existencia de debilidades críticas que afectan a distintos aspectos de la seguridad de la aplicación, incluyendo fallos

criptográficos, inyecciones, configuraciones inseguras, uso de componentes vulnerables, debilidades en autenticación y deserialización insegura.

A continuación, se documenta de forma individual la explotación de cada vulnerabilidad detectada, describiendo la técnica utilizada, la evidencia obtenida y el impacto asociado.

A2 – Cryptographic Failures (Plain Hashing)

1. Descripción de la vulnerabilidad

En este ejercicio se analiza una vulnerabilidad de tipo **Cryptographic Failures**, derivada del uso de **hashes simples sin salt** para el almacenamiento de contraseñas. Aunque algoritmos como MD5 o SHA-256 son criptográficamente válidos, su uso directo sobre contraseñas sin un valor de salt único por usuario constituye un diseño inseguro.

La vulnerabilidad no reside en la rotura del algoritmo criptográfico, sino en una **gestión incorrecta de credenciales**, que permite la recuperación de contraseñas en claro mediante ataques de diccionario o consultas a bases de datos públicas de hashes en caso de filtración.

2. Prueba realizada

El ejercicio proporciona dos hashes sin salt y solicita identificar la contraseña en claro asociada a cada uno de ellos.

Primer hash (MD5)
Hash proporcionado:

```
5F4DCC3B5AA765D61D8327DEB882CF99
```

Para demostrar la vulnerabilidad se emplearon dos métodos complementarios:

- **Comparación directa con CyberChef**, aplicando la función MD5 a una contraseña común y verificando la coincidencia exacta con el hash proporcionado.

- **Consulta en una base de datos pública de hashes**, obteniendo directamente la contraseña mediante un ataque de diccionario.

La contraseña identificada fue:

```
password
```

Segundo hash (SHA-256)
Hash proporcionado:

```
8F0E2F76E22B43E2855189877E7DC1E7D98C226C95DB247CD1D547928334A9
```

En este caso, se utilizó directamente una base de datos pública de hashes, recuperando la contraseña en claro mediante un ataque de diccionario, lo que demuestra que el uso de SHA-256 sin salt sigue siendo vulnerable cuando se emplean contraseñas comunes.

La contraseña identific ada fue:

```
passw0rd
```

Finalmente, ambas contraseñas fueron introducidas correctamente en el ejercicio, completándose la prueba con éxito.

3. Evidencia

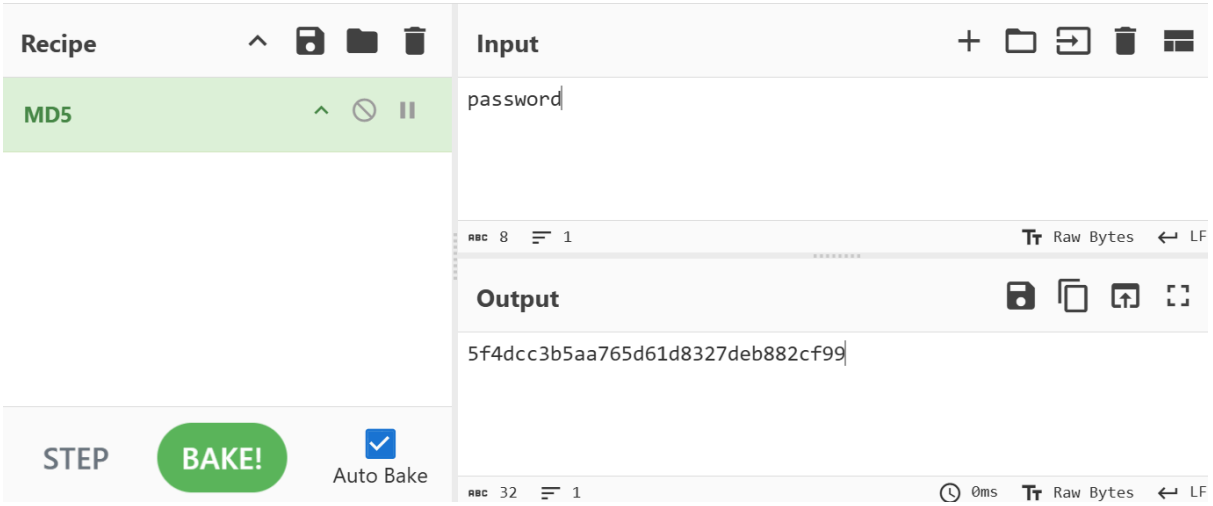

Figura A2.1 – Identificación del primer hash (MD5) mediante comparación en CyberChef, mostrando la coincidencia del hash generado con la contraseña `password`.

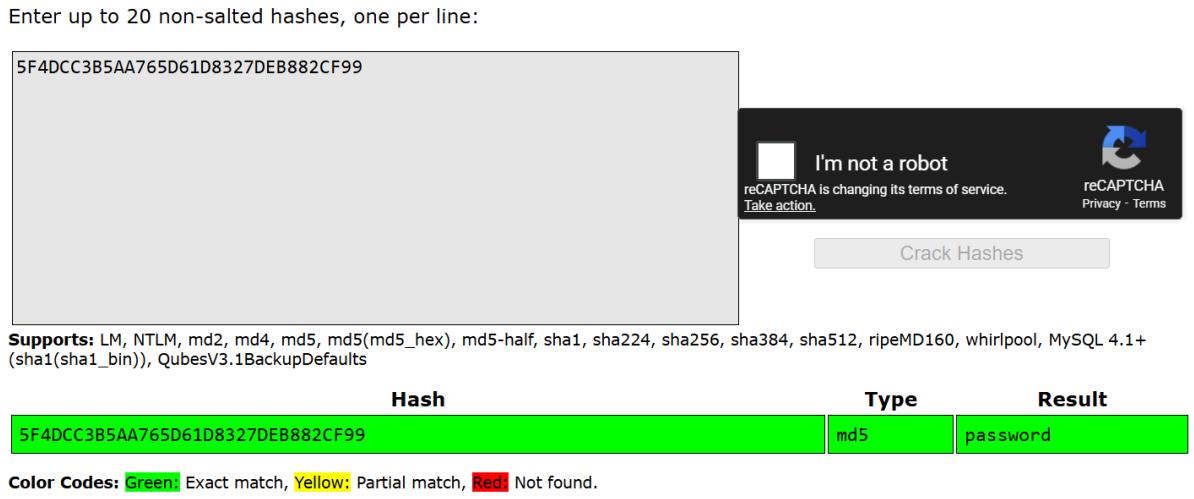

Figura A2.2 – Recuperación del primer hash (MD5) a través de una base de datos pública de hashes, mostrando la contraseña en claro.

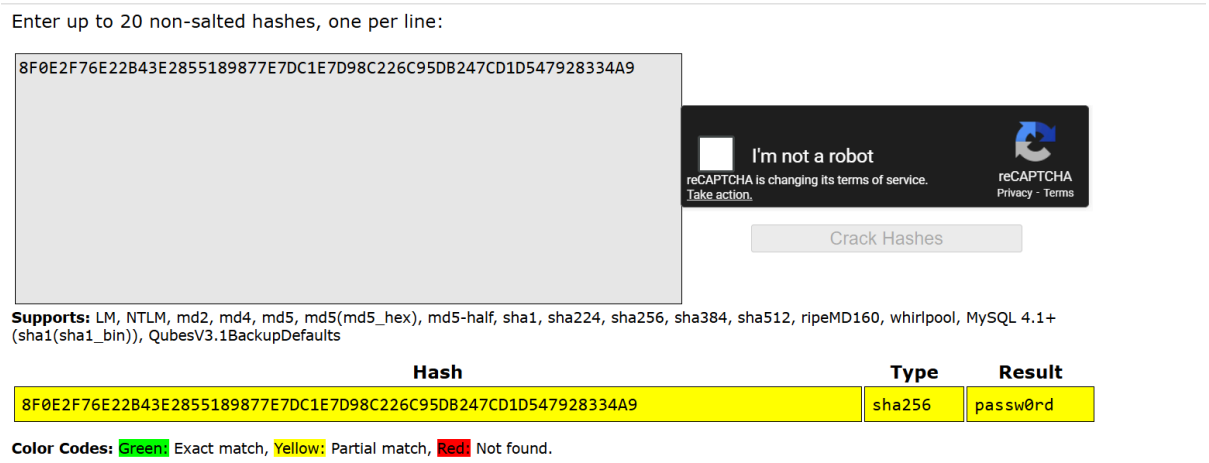

Figura A2.3 – Recuperación del segundo hash (SHA-256) mediante una base de datos pública de hashes, mostrando la contraseña `passw0rd`.

Figura A2.4 – Pantalla final del ejercicio con ambas contraseñas introducidas correctamente y el ejercicio completado.

4. Impacto

El almacenamiento de contraseñas mediante hashes simples sin salt permite que, ante una filtración de la base de datos, un atacante recupere credenciales en claro de forma rápida y masiva, sin necesidad de realizar ataques de fuerza bruta.

El impacto incluye:

- Compromiso de cuentas de usuario.
- Reutilización de credenciales en otros servicios.
- Escalada de privilegios.

- Pérdida de confidencialidad y control de acceso.

Esta vulnerabilidad afecta directamente a los mecanismos de autenticación del sistema y representa un riesgo elevado para la seguridad global de la aplicación.

5. Referencia OWASP

- OWASP Top 10 2021 – A02: Cryptographic Failures
  https://owasp.org/Top10/2021/A02_2021-Cryptographic_Failures/
- OWASP Password Storage Cheat Sheet
  https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- CrackStation – Online Password Hash Cracking
  https://crackstation.net/
- CyberChef – The Cyber Swiss Army Knife
  https://gchq.github.io/CyberChef/

A3 – Injection

String SQL Injection

1. Descripción de la vulnerabilidad

Durante el análisis del formulario de consulta de empleados se identificó una vulnerabilidad de tipo **String SQL Injection**, originada por la construcción insegura de consultas SQL mediante la concatenación directa de datos introducidos por el usuario. La aplicación genera la siguiente consulta:

```
SELECT * FROM employees WHERE last_name = '<name>' AND auth_tan =
'<auth_tan>';
```

La ausencia de mecanismos de sanitización de entradas y el uso de consultas no parametrizadas permiten al atacante manipular la lógica de la cláusula `WHERE`, alterando el comportamiento previsto de la consulta y anulando los controles de acceso implementados.

---

2. Prueba realizada

Con el objetivo de evaluar el alcance real de la vulnerabilidad, se llevaron a cabo diferentes pruebas prácticas que demuestran tanto la obtención masiva de información como el acceso dirigido a registros concretos, sin disponer de credenciales válidas.

Bypass lógico del control de acceso (tautología SQL)
Se introdujeron expresiones lógicas siempre verdaderas en el parámetro `last_name`, manteniendo un valor arbitrario en el campo `Authentication TAN`, el cual queda invalidado por la inyección.

- **PoC A1 – Tautología numérica**
  - Employee Name: `' OR 1=1 --`
  - Authentication TAN: `x`
- **PoC A2 – Tautología de cadena**
  - Employee Name: `' OR 'a'='a' --`
  - Authentication TAN: `x`

En ambos casos, el sistema devolvió múltiples registros de la tabla `employees`, confirmando la pérdida total de confidencialidad.

Bypass selectivo del control de acceso (acceso dirigido)
Se realizó una prueba adicional para demostrar que la vulnerabilidad permite el acceso a información de empleados concretos sin necesidad de autenticación válida.

- **Prueba B1 – Acceso dirigido**
  - Employee Name: `' OR last_name='Smith' --`
  - Authentication TAN: `x`

El sistema devolvió únicamente el registro del empleado indicado, exponiendo su información interna.

Explotación del parámetro `auth_tan`
Finalmente, se comprobó que el parámetro `auth_tan` también es vulnerable a inyección SQL, manteniendo un valor legítimo en el campo `last_name`.

- Employee Name: `Smith`
- Authentication TAN: `' OR '1'='1' --`

Esta prueba permitió obtener el listado completo de empleados, confirmando que la vulnerabilidad afecta a múltiples parámetros de entrada.

3. Evidencia

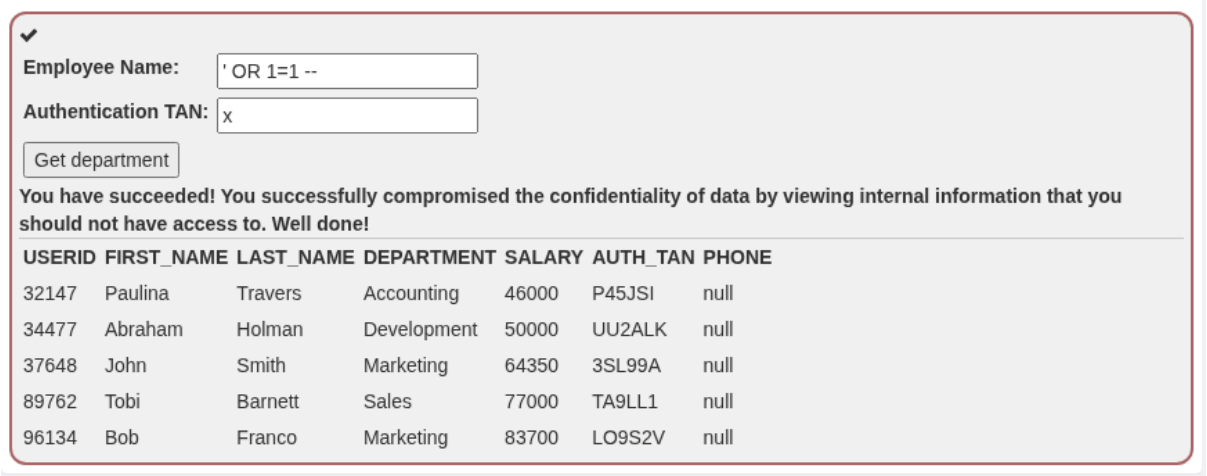

Figura A2.1 – Resultado de la inyección SQL mediante tautología numérica (`' OR 1=1 --`), mostrando el listado completo de empleados.

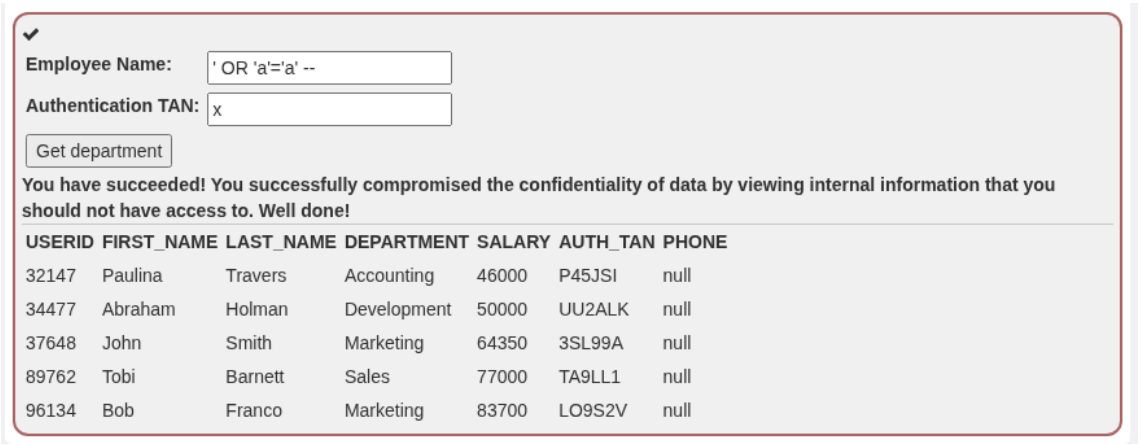

Figura A2.2 – Explotación de la inyección SQL mediante tautología de cadena (`' OR 'a'='a' --`), con exposición de información sensible.

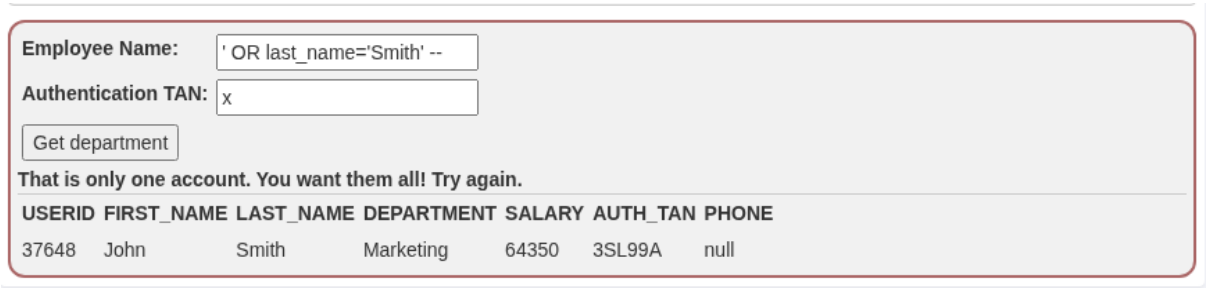

Figura A2.3 – Acceso dirigido a la información de un empleado concreto mediante inyección SQL, sin conocimiento de credenciales válidas.

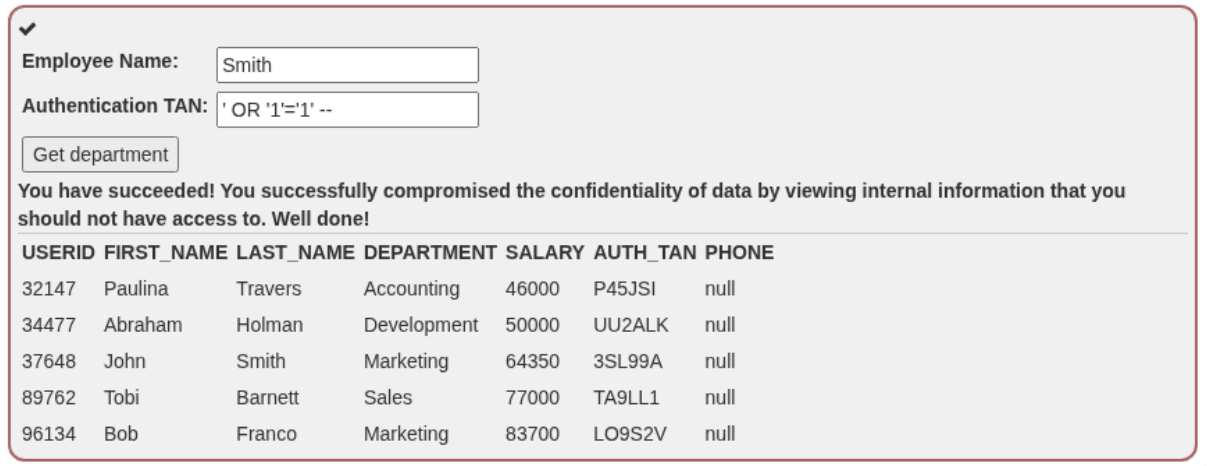

Figura A2.4 – Explotación de inyección SQL a través del parámetro `auth_tan`, con obtención no autorizada de los datos de todos los empleados.

4. Impacto

La explotación de esta vulnerabilidad compromete gravemente el **principio de confidencialidad**, permitiendo el acceso no autorizado a información interna sensible, incluyendo identificadores de empleados, departamentos, salarios y valores de autenticación (TAN).

La posibilidad de realizar tanto accesos masivos como accesos dirigidos incrementa el riesgo, ya que facilita la enumeración de datos sensibles y la focalización de ataques sobre usuarios específicos. Además, la exposición de TANs de autenticación podría habilitar ataques posteriores de suplantación de identidad o accesos indebidos a otros sistemas.

Dado que la explotación no requiere credenciales válidas ni conocimientos técnicos avanzados y afecta a múltiples parámetros, el riesgo asociado se considera **alto**.

5. Referencia OWASP

- OWASP Top 10 2021 – A03: Injection
  https://owasp.org/Top10/A03_2021-Injection/
- OWASP Cheat Sheet – SQL Injection Prevention
  https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Cross Site Scripting (XSS)

1. Descripción de la vulnerabilidad

Durante el análisis de la aplicación se identificó una vulnerabilidad de tipo **Cross Site Scripting reflejado (Reflected XSS)**. Este tipo de vulnerabilidad se produce cuando datos controlados por el usuario son incluidos directamente en la respuesta HTTP generada por el servidor sin aplicarse mecanismos adecuados de validación o codificación de salida.

En este caso, determinados parámetros enviados al servidor mediante una petición HTTP son reflejados en la página de respuesta, permitiendo la inyección y ejecución de código JavaScript arbitrario en el navegador del cliente.

2. Prueba realizada

La explotación de la vulnerabilidad se llevó a cabo interceptando la petición HTTP generada durante el proceso de confirmación de compra. Para ello, se utilizó **Burp Suite** como proxy de interceptación.

Se interceptó la petición `GET /WebGoat/CrossSiteScripting/attack5a` y se modificó el valor del parámetro `field1`, inyectando código JavaScript malicioso. Al reenviar la petición alterada al servidor, el contenido inyectado fue reflejado en la respuesta HTML y ejecutado automáticamente por el navegador del usuario.

La explotación se evidenció mediante la aparición de un cuadro de alerta, confirmando la ejecución de código JavaScript no autorizado.

3. Evidencia

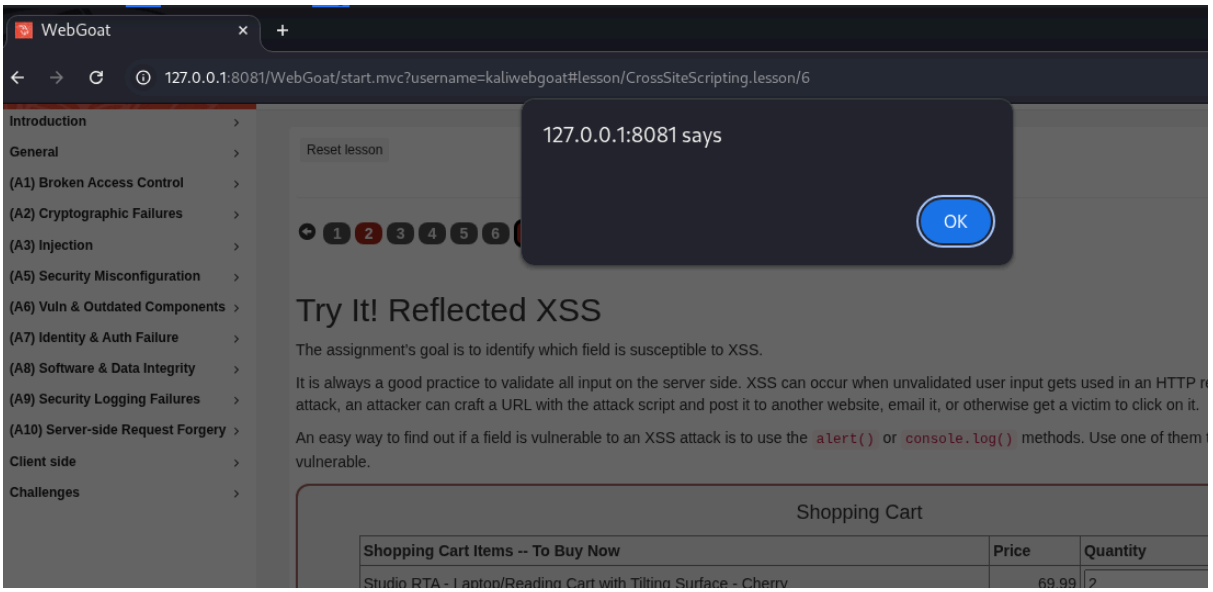

Figura A2.3 – Petición HTTP interceptada y modificada con Burp Suite, mostrando la inyección de código JavaScript en el parámetro `field1`.

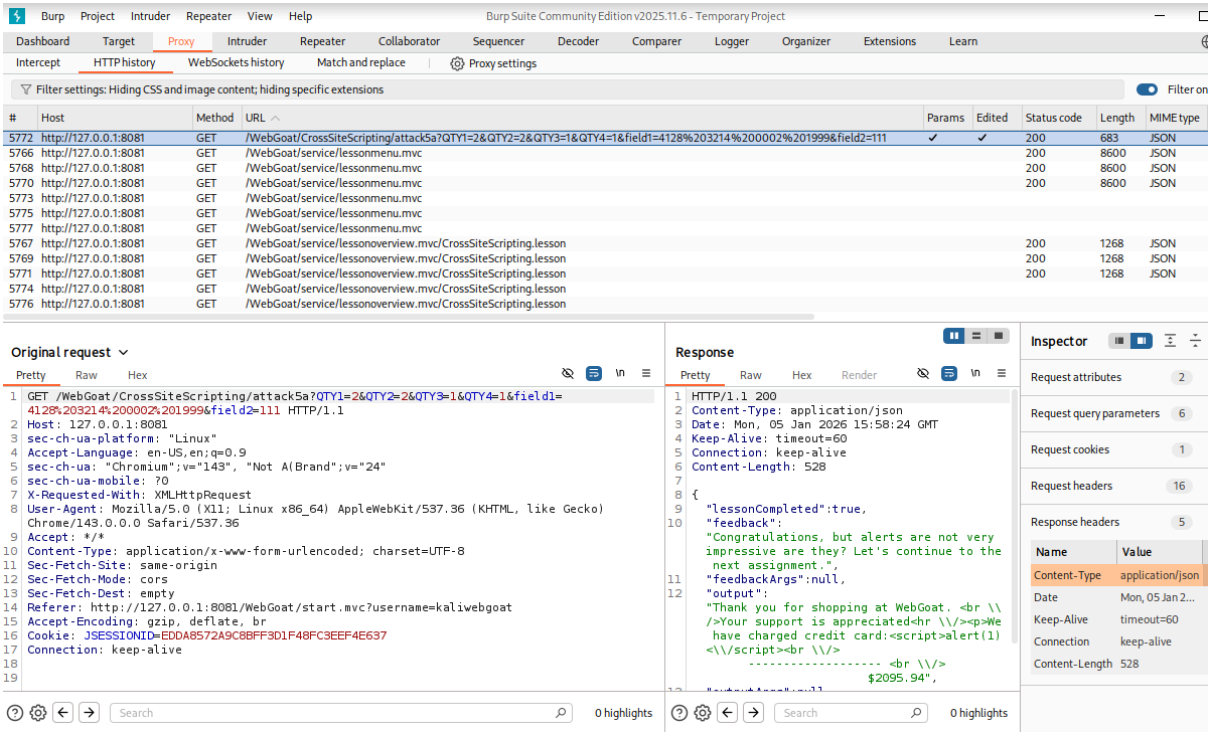

Figura A2.4 – Ejecución de código JavaScript en el navegador tras la explotación de la vulnerabilidad XSS reflejado, evidenciada mediante un cuadro de alerta.

4. Impacto

La explotación de esta vulnerabilidad permite la ejecución de código JavaScript arbitrario en el navegador del usuario afectado. Esto puede derivar en el robo de cookies de sesión, suplantación de identidad, redirección a sitios maliciosos o la ejecución de acciones no autorizadas en nombre del usuario.

Aunque el ataque requiere interacción por parte de la víctima, como el acceso a una URL manipulada, el impacto sobre la **confidencialidad e integridad de la sesión** es elevado, ya que el código malicioso se ejecuta con los privilegios del usuario autenticado.

5. Referencia OWASP

- OWASP Top 10 2021 – A03: Injection
  https://owasp.org/Top10/A03_2021-Injection/

- OWASP Cheat Sheet – Cross Site Scripting Prevention
  https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

---

A5 – Security Misconfiguration: Cross-Site Request Forgery (CSRF)

1. Descripción de la vulnerabilidad

La vulnerabilidad **Cross-Site Request Forgery (CSRF)** se produce cuando una aplicación web no valida correctamente el origen de las peticiones y confía únicamente en la sesión del usuario autenticado. Esto permite que un atacante fuerce a un usuario legítimo a ejecutar acciones no deseadas sin su conocimiento ni consentimiento.

En este ejercicio de WebGoat, la funcionalidad de publicación de reseñas acepta peticiones POST al endpoint `/WebGoat/csrf/review` sin implementar mecanismos de protección CSRF efectivos, como tokens anti-CSRF o validaciones del origen de la petición.

2. Prueba realizada

La explotación de la vulnerabilidad se llevó a cabo en tres fases diferenciadas.

Fase 1 – Observación del comportamiento normal
Se accedió a la funcionalidad de reseñas desde el navegador, comprobando que al utilizar únicamente la interfaz web no se publicaba ninguna reseña adicional y que la aplicación mostraba un mensaje indicando que la petición provenía del mismo host.

Fase 2 – Forjado de la petición CSRF
Mediante **Burp Suite**, se interceptó y forjó manualmente una petición POST al endpoint vulnerable `/WebGoat/csrf/review` utilizando la herramienta Repeater. En la petición se modificó el parámetro `reviewText`, asignándole el valor “Icecream”, manteniendo una cookie de sesión válida (`JSESSIONID`) correspondiente a un usuario autenticado.

El servidor respondió con **HTTP 200 OK**, aceptando la petición forjada a pesar de no haberse originado desde el flujo legítimo de la aplicación.

Fase 3 – Verificación del impacto
Tras reenviar la petición forjada, se recargó la página de reseñas y se comprobó que el

comentario “Icecream” aparecía publicado bajo la identidad del usuario autenticado, incluso con campos adicionales en estado no válido (por ejemplo, valoración nula), confirmando la explotación de la vulnerabilidad.

3. Evidencia

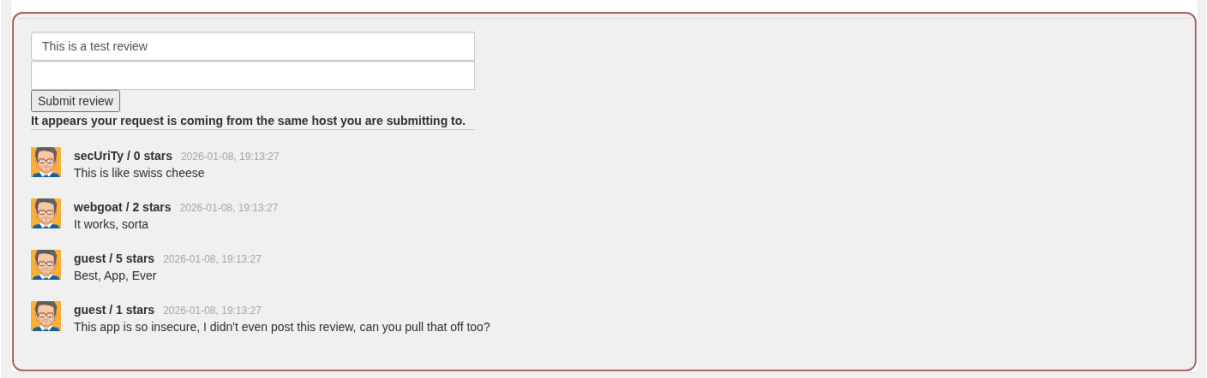

Figura A5.1 – Vista inicial de la página de reseñas antes de la explotación CSRF, sin la reseña “Icecream”.

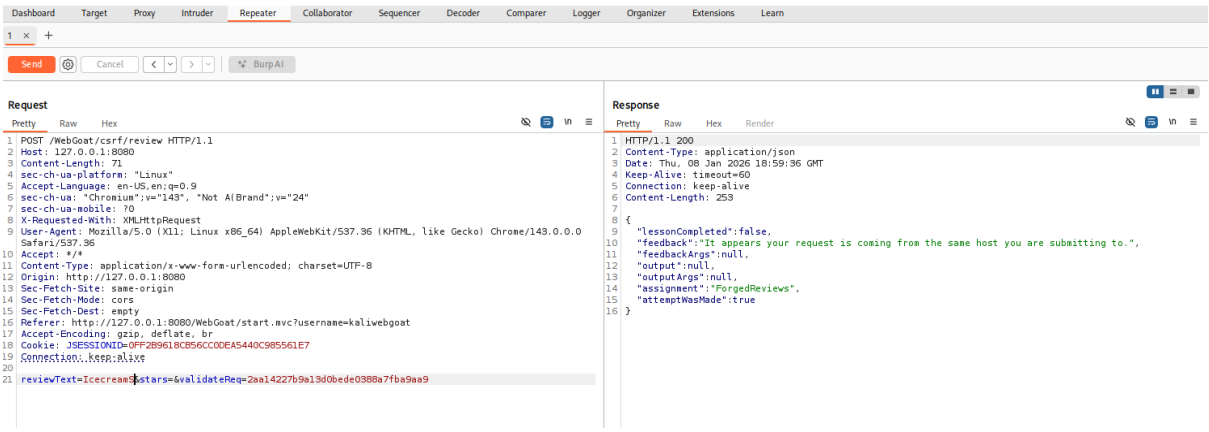

Figura A5.2 – Petición CSRF forjada en Burp Repeater al endpoint `/WebGoat/csrf/review`, mostrando el parámetro `reviewText=Icecream` y la respuesta HTTP 200.

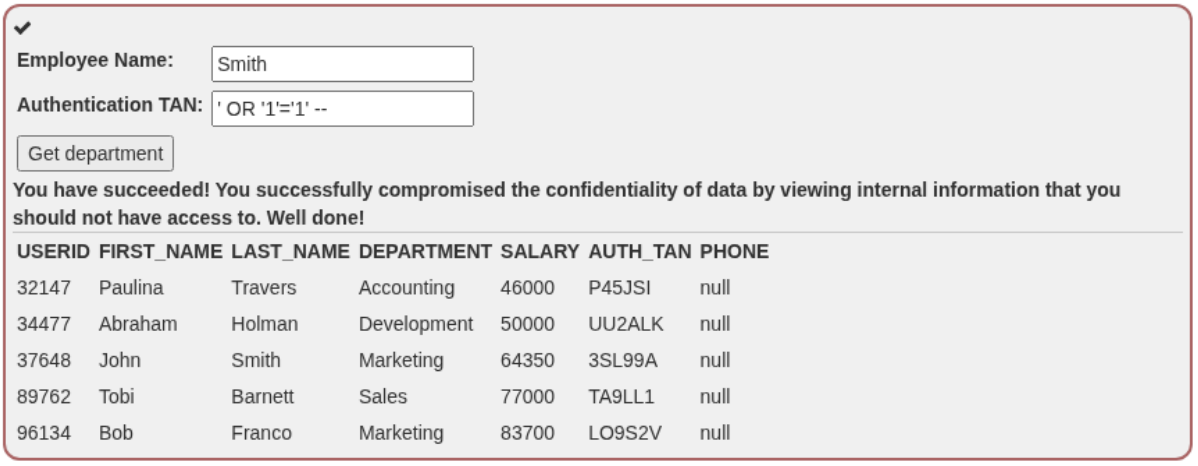

Figura A5.3 – Reseña publicada tras la explotación CSRF, visible en la interfaz web bajo el usuario autenticado.

4. Impacto

La explotación de esta vulnerabilidad permite a un atacante ejecutar acciones en nombre de usuarios legítimos aprovechando su sesión autenticada. Entre los posibles impactos se incluyen la publicación de contenido no autorizado, la manipulación de información y el abuso de la confianza del sistema en la sesión del usuario.

En aplicaciones reales, este tipo de vulnerabilidad puede derivar en consecuencias más graves, como transferencias no autorizadas, cambios de datos personales o ejecución de acciones administrativas. Por ello, el riesgo asociado se considera **alto**.

5. Referencia OWASP

- OWASP Top 10 2021 – A05: Security Misconfiguration
  https://owasp.org/Top10/2021/A05_2021-Security_Misconfiguration/
- OWASP – Cross-Site Request Forgery (CSRF)
  https://owasp.org/www-community/attacks/csrf
- OWASP CSRF Prevention Cheat Sheet
  https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

---

A6 – Vulnerable & Outdated Components

1. Descripción de la vulnerabilidad

Durante el análisis del apartado **A6 – Vulnerable & Outdated Components** se identificó el uso de un componente *open source* vulnerable, concretamente **jQuery UI versión 1.10.4**, que presenta una vulnerabilidad de **Cross-Site Scripting (XSS)** asociada al parámetro `closeText` del componente `dialog`. Este parámetro permite introducir contenido controlado por el usuario sin aplicar mecanismos adecuados de sanitización o codificación de salida.

La vulnerabilidad no reside en el código propio de la aplicación, sino en el comportamiento inseguro del componente externo. Al utilizar una versión corregida (**jQuery UI ≥ 1.12.0**), el mismo escenario deja de ser explotable, lo que confirma que el origen del riesgo es el uso de un componente vulnerable y desactualizado.

2. Prueba realizada

Se intentó inicialmente reproducir la explotación directamente en el laboratorio de WebGoat siguiendo el procedimiento descrito en el ejercicio. En el entorno utilizado, el payload no llegó a ejecutarse y el laboratorio mostró únicamente el mensaje informativo del propio ejercicio, sin producirse la ejecución de código JavaScript.

Este comportamiento se atribuye a limitaciones del laboratorio de WebGoat en navegadores modernos relacionadas con la carga/aislamiento de librerías JavaScript, y no a un error en el payload.

Para validar el riesgo real asociado al componente, se reprodujo el mismo escenario en un **entorno local controlado**, utilizando exactamente **jQuery UI 1.10.4**, definiendo el valor del parámetro `closeText` a partir de una entrada controlada por el usuario, sin sanitización.

Payload utilizado:

```
OK<img src=x onerror=alert('XSS')>
```

Al renderizarse el diálogo, el navegador ejecutó el JavaScript embebido en el payload, confirmando la explotación de XSS.

3. Evidencia

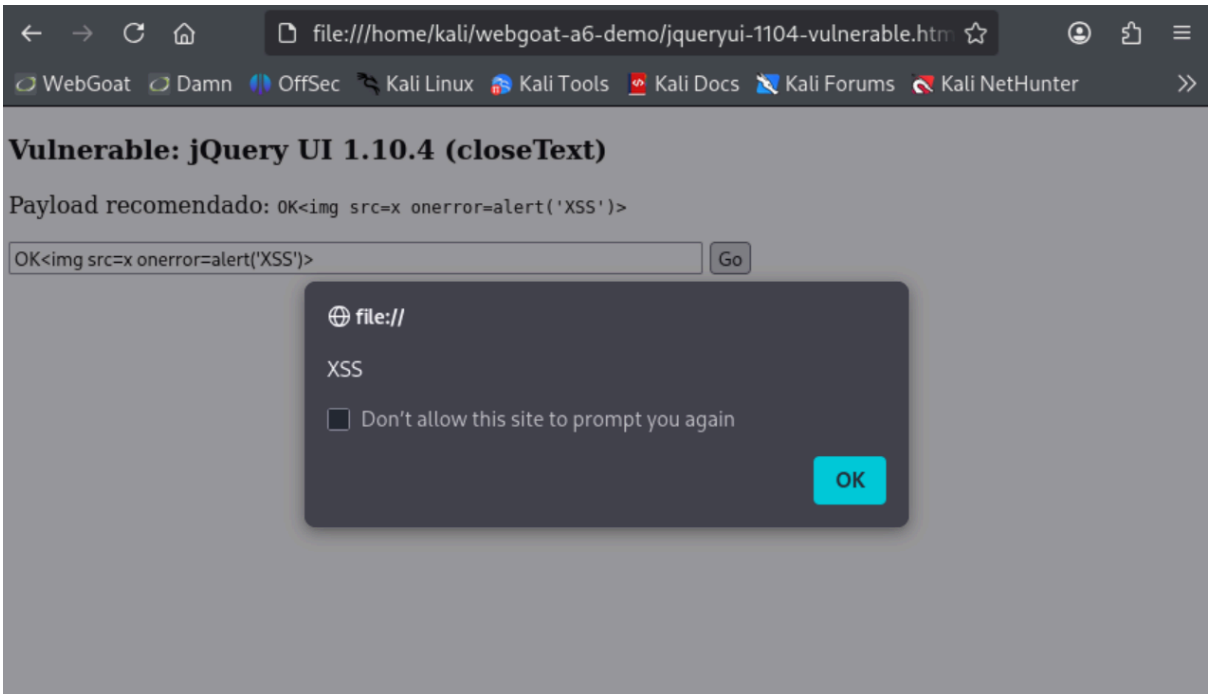

Figura A6.1 – Ejecución de XSS en entorno local utilizando **jQuery UI 1.10.4**, evidenciada mediante la aparición de un `alert('XSS')` tras introducir el payload en `closeText`.

4. Impacto

Esta vulnerabilidad permite la ejecución de código JavaScript arbitrario en el navegador de la víctima, lo que puede derivar en robo de cookies de sesión, suplantación de identidad, redirecciones maliciosas o manipulación del contenido de la página. El impacto es consecuencia directa del uso de un componente vulnerable, lo que refuerza la necesidad de gestionar y actualizar dependencias externas de forma continua.

5. Referencia OWASP

- OWASP Top 10 2021 – A06: Vulnerable and Outdated Components
  https://owasp.org/Top10/2021/A06_2021-Vulnerable_and_Outdated_Components/

A7 – Identity & Authentication Failures

Secure Passwords – Evaluación de fortaleza de contraseña

1. Descripción de la vulnerabilidad

En este ejercicio se evalúa la fortaleza de contraseñas en el contexto del apartado **A7 – Identity & Authentication Failures**, conforme a las recomendaciones actuales del **NIST**. La debilidad abordada no es una vulnerabilidad técnica explotable de forma directa, sino un **riesgo de diseño asociado al uso de contraseñas débiles**, fácilmente comprometibles mediante ataques de fuerza bruta o diccionario.

El ejercicio pone el foco en la **longitud, entropía y aleatoriedad real** de la contraseña, frente a las reglas clásicas de composición forzada (mayúsculas, símbolos obligatorios, cambios periódicos), que ya no se consideran eficaces por sí solas.

2. Prueba realizada

Se introdujo una contraseña diseñada como **passphrase larga**, combinando letras en mayúscula y minúscula, números y caracteres especiales, evitando palabras comunes, nombres propios y patrones previsibles.

Contraseña utilizada:

```
Nube-10-verde@camino_lentosol-40RIO
```

El sistema de evaluación de WebGoat asignó a la contraseña la **puntuación máxima (4/4)**, indicando una alta resistencia frente a ataques de fuerza bruta y un tiempo estimado de ruptura elevado.

3. Evidencia

Figura A7.1 – Evaluación de fortaleza de contraseña en WebGoat, mostrando puntuación **4/4** y un tiempo elevado estimado para un ataque de fuerza bruta.

4. Impacto (en términos de seguridad)

El uso de contraseñas con alta entropía reduce significativamente el riesgo de compromiso de cuentas mediante ataques de fuerza bruta, ataques de diccionario y técnicas de *password spraying*. Una contraseña fuerte mitiga uno de los vectores de ataque más comunes en sistemas de autenticación, aunque debe complementarse con otras medidas como el uso de gestores de contraseñas y autenticación multifactor.

5. Referencias

- OWASP Top 10 2021 – A07: Identification and Authentication Failures
  https://owasp.org/Top10/2021/A07_2021-Identification_and_Authentication_Failures/

- NIST SP 800-63B-4 – Digital Identity Guidelines: Authentication and Authenticator Management
  https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63B-4.pdf

---

A8 – Insecure Deserialization (Deserialización insegura)

1. Descripción de la vulnerabilidad

La deserialización insegura se produce cuando una **aplicación procesa datos serializados procedentes de fuentes no confiables sin aplicar validaciones estrictas** sobre el tipo de objeto, su estado o su comportamiento durante el proceso de deserialización. En Java, este riesgo es especialmente relevante, ya que métodos como `readObject()` pueden ejecutar código automáticamente al reconstruir el objeto en memoria.

En este ejercicio de WebGoat, el servidor recibe un objeto Java serializado en formato Base64 y lo deserializa directamente. Si el objeto pertenece a una clase vulnerable incluida en el *classpath* y contiene lógica peligrosa en el método `readObject()`, es posible provocar efectos no previstos por el desarrollador, como retrasos deliberados en la ejecución, denegación de servicio o ejecución de comandos del sistema.

2. Prueba realizada

Se analizó el código del ejercicio para identificar el tipo de objeto esperado durante la deserialización. El servidor rechaza objetos de tipo `String` y únicamente acepta instancias de la clase `VulnerableTaskHolder`, que implementa `Serializable` y contiene lógica peligrosa en su método `readObject()`.

A partir de este análisis, se creó un objeto `VulnerableTaskHolder` personalizado, configurando el atributo `taskAction` con el comando `sleep 5`, de forma que el retardo se produjera durante el proceso de deserialización. Para evitar los controles temporales implementados en el servidor, el campo `requestedExecutionTime` se estableció de forma válida para el rango aceptado por la aplicación.

El objeto fue serializado utilizando **Java Object Serialization** y posteriormente codificado en **Base64**. El token resultante se envió a WebGoat a través del campo habilitado para el ejercicio. Al procesarlo, el servidor ejecutó el método `readObject()`, provocando un retraso aproximado de cinco segundos y cumpliendo las condiciones del reto.

3. Evidencia

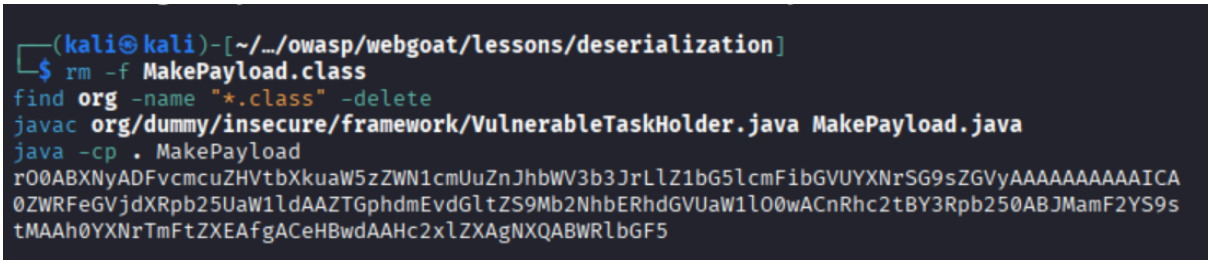

Figura A8.1 – Generación del payload en la terminal mediante la serialización de un objeto `VulnerableTaskHolder` y obtención del token Base64 utilizado en el ataque.

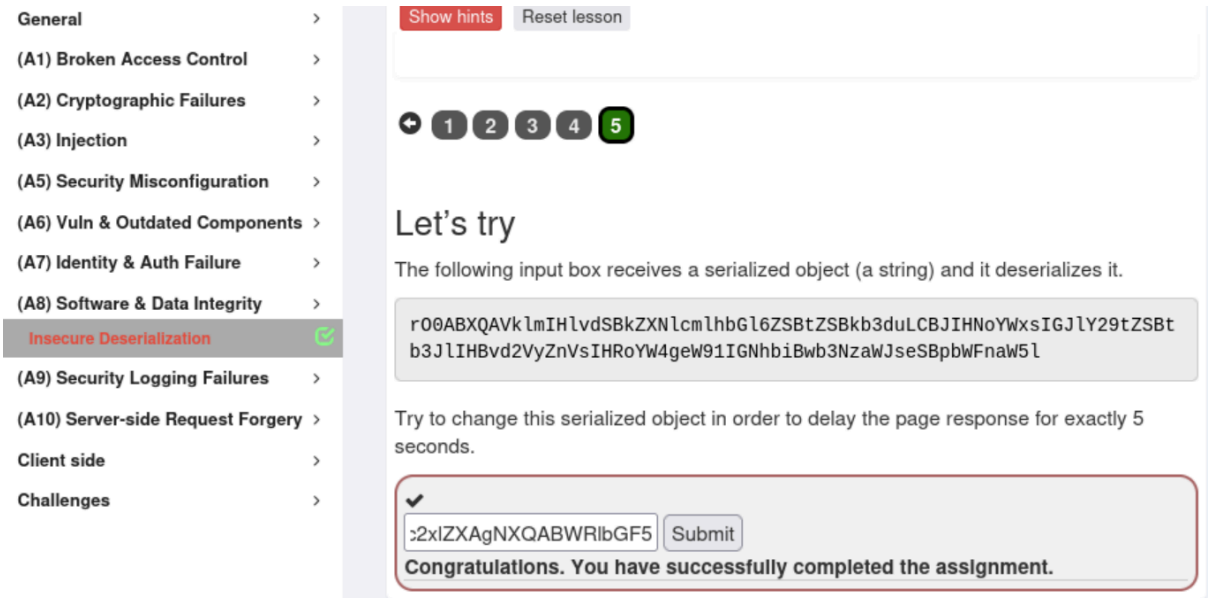

Figura A8.2 – Mensaje de éxito mostrado por WebGoat tras deserializar el objeto malicioso y detectar el retardo temporal esperado:
"Congratulations, you have successfully completed the assignment."

4. Impacto

La deserialización insegura permite a un atacante influir directamente en el flujo de ejecución de la aplicación durante la reconstrucción de objetos en memoria. En entornos reales, este tipo de vulnerabilidad puede derivar en denegaciones de servicio, ejecución remota de comandos o compromiso completo del sistema, especialmente cuando existen cadenas de *gadgets* disponibles en el *classpath*.

El ejercicio demuestra que confiar en datos serializados sin una validación estricta del tipo, estado y comportamiento del objeto supone un riesgo crítico para la seguridad de la aplicación.

5. Referencias

- OWASP Top 10 2021 – A08: Software and Data Integrity Failures
  https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/
- OWASP Deserialization Cheat Sheet
  https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html

3.3 Post-explotación

Una vez explotadas las vulnerabilidades identificadas durante la auditoría, se analizó el **impacto potencial** que un atacante podría alcanzar tras una explotación exitosa, considerando escenarios realistas de abuso y encadenamiento de fallos.

A2 – Cryptographic Failures (Plain Hashing)

Tras recuperar contraseñas en claro a partir de hashes sin salt, un atacante podría **reutilizar dichas credenciales** para acceder a otras cuentas o servicios, especialmente en escenarios de reutilización de contraseñas. Este acceso permitiría la **suplantación de identidad** de usuarios legítimos y **facilitaría ataques posteriores** sobre funcionalidades protegidas por autenticación.

A3 – Injection (SQL Injection)

La explotación de inyecciones SQL permite al atacante **acceder a información sensible** almacenada en la base de datos, como credenciales, datos personales o información interna. En un escenario de post-explotación, esta información podría utilizarse para realizar

movimientos laterales, preparar ataques dirigidos o combinarse con otras vulnerabilidades para escalar el impacto del compromiso.

A3 – Cross Site Scripting (XSS)

Una vez ejecutado código JavaScript en el navegador de la víctima, el atacante podría **robar cookies de sesión**, **tokens de autenticación** o **realizar acciones en nombre del usuario**. En combinación con otras vulnerabilidades, el XSS puede servir como vector inicial para la toma de control de cuentas o para ataques persistentes contra usuarios legítimos.

A5 – Cross-Site Request Forgery (CSRF)

Tras explotar la vulnerabilidad CSRF, un atacante puede **forzar a usuarios autenticados** a realizar acciones no deseadas. En post-explotación, esto permitiría manipular contenido, alterar configuraciones o ejecutar operaciones encadenadas sin interacción consciente del usuario, aprovechando su sesión activa.

A6 – Vulnerable & Outdated Components

El uso de componentes vulnerables como jQuery UI 1.10.4 facilita la ejecución de ataques XSS del lado cliente. En un escenario posterior a la explotación, el atacante podría **mantener persistencia** mediante scripts inyectados, robar información sensible o utilizar el componente vulnerable como punto de entrada para ataques más complejos contra la aplicación o sus usuarios.

A7 – Identity & Authentication Failure (Secure Passwords)

El uso de contraseñas débiles incrementa la probabilidad de éxito de ataques de **fuerza bruta o password spraying**. Tras comprometer una cuenta, un atacante podría acceder a funcionalidades internas, datos sensibles o preparar ataques adicionales aprovechando los privilegios del usuario comprometido.

A8 – Insecure Deserialization

La deserialización insegura permite **alterar el flujo de ejecución del servidor**. En escenarios reales, tras una explotación inicial, un atacante podría provocar denegaciones de servicio repetidas o ejecutar comandos arbitrarios, comprometiendo gravemente la disponibilidad e integridad del sistema.

3.4 Posibles mitigaciones

A partir de las vulnerabilidades explotadas durante la auditoría, se proponen las siguientes medidas de mitigación, orientadas a reducir el riesgo y prevenir escenarios similares en entornos reales.

A2 – Cryptographic Failures (Plain Hashing)

Se recomienda almacenar contraseñas utilizando funciones de derivación de claves seguras, como PBKDF2, bcrypt, scrypt o Argon2, incorporando un **salt único por contraseña** y un factor de coste adecuado. Debe evitarse el uso de hashes criptográficos genéricos (MD5, SHA-1, SHA-256) aplicados directamente a contraseñas.

A3 – Injection (SQL Injection)

Para prevenir inyecciones SQL es necesario utilizar **consultas parametrizadas (prepared statements)** y evitar la concatenación directa de datos controlados por el usuario en las consultas. Adicionalmente, deben aplicarse controles de validación de entrada y limitar los privilegios de las cuentas de acceso a la base de datos.

A3 – Cross Site Scripting (XSS)

Las vulnerabilidades XSS pueden mitigarse mediante la **codificación de salida** adecuada según el contexto (HTML, JavaScript, atributos), evitando reflejar datos del usuario sin tratamiento previo. El uso de políticas de seguridad de contenido (CSP) refuerza la protección frente a la ejecución de scripts no autorizados.

A5 – Cross-Site Request Forgery (CSRF)

Para mitigar ataques CSRF deben implementarse **tokens antifalsificación** únicos por sesión o petición, así como la validación del origen y del método HTTP. Estas medidas garantizan que las acciones sensibles solo puedan ejecutarse desde flujos legítimos de la aplicación.

A6 – Vulnerable & Outdated Components

Es fundamental mantener un **control de dependencias** que permita identificar componentes vulnerables y aplicar actualizaciones de seguridad de forma periódica. El uso de herramientas de análisis de dependencias y la definición de una política de actualización reducen el riesgo asociado a librerías desactualizadas.

A7 – Identity & Authentication Failures (Secure Passwords)

Se recomienda fomentar el uso de **contraseñas largas y con alta entropía**, alineadas con las directrices del NIST, evitando reglas de composición rígidas. El uso de gestores de

contraseñas y la implementación de **autenticación multifactor** refuerzan significativamente la seguridad del proceso de autenticación.

A8 – Insecure Deserialization

Para prevenir deserialización insegura se debe evitar la deserialización de objetos procedentes de fuentes no confiables. En caso de ser necesaria, deben aplicarse **listas blancas de clases permitidas**, validaciones estrictas del contenido y, preferiblemente, el uso de formatos de intercambio de datos más seguros como JSON.

3.5 Herramientas utilizadas

Durante la auditoría se emplearon diversas herramientas de seguridad, tanto locales como basadas en web, para apoyar las fases de reconocimiento, explotación y análisis de resultados:

- **WebGoat**: plataforma vulnerable utilizada como objetivo de la auditoría y base de los ejercicios prácticos.
- **Burp Suite**: proxy de interceptación utilizado para analizar tráfico HTTP, identificar endpoints accesibles y modificar peticiones durante la explotación de vulnerabilidades como XSS y CSRF.
- **Nmap**: herramienta empleada para la identificación de puertos abiertos y servicios expuestos durante la fase de reconocimiento.
- **curl**: utilizada para la inspección de cabeceras HTTP y el análisis del comportamiento de redirección de la aplicación.
- **CyberChef**: herramienta web empleada para el análisis criptográfico y la verificación de hashes durante la evaluación de fallos criptográficos.
- **Plataformas públicas de cracking de hashes** (por ejemplo, CrackStation): utilizadas para demostrar la debilidad de hashes sin salt mediante ataques de diccionario.
- **Java Development Kit (JDK)**: utilizado para la creación y serialización de objetos durante la explotación de la vulnerabilidad de deserialización insegura.

- **Docker**: tecnología empleada para el despliegue controlado del entorno de laboratorio.

Este conjunto de herramientas permitió realizar una auditoría completa y reproducible, alineada con los objetivos formativos de la práctica.