

# Auditoría web WebGoat

Enero 2026

**Arturo Moreno**





## Índice de contenidos

Ámbito y alcance de la auditoría.....	3
Metodología de calificación de riesgos.....	4
Informe ejecutivo.....	5
Descripción del proceso de auditoría.....	7
1. Fase de reconocimiento.....	7
2. SQL Injection.....	9
3. XSS.....	12
4. Cross-Site Request Forgeries (CSRF).....	14
5. Vulnerable and Outdated Components.....	15
6. Identity & Auth Failure.....	16
Herramientas utilizadas.....	18



## Ámbito y alcance de la auditoría

Este informe presenta los hallazgos encontrados durante la auditoría web realizada a la aplicación web WebGoat, desplegada en un entorno local de laboratorio, con el objetivo de identificar y analizar vulnerabilidades de seguridad comunes en sitios web, así como proponer mecanismos de mitigación de dichas vulnerabilidades.

WebGoat es una aplicación deliberadamente vulnerable, con fines formativos, pero la auditoría se ha llevado a cabo con rigurosidad y metodología profesional, simulando procedimientos que se realizarían en una evaluación de una aplicación real. En este caso nos hemos limitado a las 5 vulnerabilidades más destacables, asociadas a los servicios directamente relacionados con el funcionamiento de la aplicación, accesibles a través de los puertos expuestos en el entorno analizado. No se han analizado otros sistemas, infraestructuras, redes externas, etc. que pudieran estar implicadas en la aplicación.

El alcance técnico ha comprendido las siguientes actividades:

- Identificación de la superficie de ataque mediante tareas de reconocimiento y recopilación de la información.
- Detección y explotación de vulnerabilidades de seguridad de la aplicación.
- Evaluación del impacto potencial de las vulnerabilidades detectadas.
- Propuesta de medidas de mitigación y buenas prácticas.

Las pruebas se han realizado de manera manual y asistida por herramientas, habiendo sido el uso de estas adecuadamente documentado en los correspondientes apartados.



## Metodología de calificación de riesgos

Para la calificación de los riesgos se ha utilizado el estándar **Common Vulnerability Scoring System (CVSS)**<sup>1</sup>, un marco de referencia ampliamente adoptado a nivel internacional para evaluar la severidad de vulnerabilidades de seguridad.

CVSS proporciona una puntuación numérica estandarizada que permite estimar el impacto y la explotabilidad de una vulnerabilidad en función de diferentes métricas. En el contexto de esta auditoría se ha utilizado la puntuación base de CVSS v3.0.

La puntuación resultante clasifica las vulnerabilidades en **baja, media, alta y crítica**, lo que facilita la priorización de riesgo y la toma de decisiones en relación con la mitigación de las vulnerabilidades detectadas.

---

<sup>1</sup> <https://nvd.nist.gov/vuln-metrics/cvss>



## Informe ejecutivo

El objetivo de la auditoría consistía en evaluar la seguridad de la aplicación simulando ataques que podrían producirse en el mundo real, puntos débiles que podrían ser utilizados por actores maliciosos para comprometer la confidencialidad, integridad o disponibilidad de activos críticos de la aplicación, así como incluir propuestas de mejora para reducir la vulnerabilidad de la aplicación.

El proceso comenzó por una fase de **reconocimiento y recopilación de la información**, en la que se identificó la superficie de ataque, los servicios expuestos y las tecnologías empleadas, y se extrajeron potenciales vulnerabilidades que debían ser exploradas en siguientes fases. A continuación se procedió a explorar las diferentes funcionalidades de la aplicación con el objetivo de detectar vulnerabilidades relacionadas con la validación de entradas, gestión de autenticación y uso de componente de terceros, así como la protección frente a ataques web habituales.

Durante la auditoría se detectaron **5 vulnerabilidades principales**, siendo la más significativa la de SQL Injection, que podría permitir a los atacantes acceder a la base de datos, descargar toda la información contenida en ella, modificarla e incluso eliminarla. Además de esta vulnerabilidad, se descubrieron también otros problemas de seguridad de gravedad intermedia, relacionados con Cross Site Scripting, Cross-Site Request Forgeries, Vulnerable & Outdated Components, y Identity & Authentication Failure.

Una vez detectadas estas vulnerabilidades, se procedió a la explotación de las mismas de manera controlada, para la posterior evaluación de su potencial impacto. Todos los hallazgos han sido documentados en detalle en este informe, incluyendo pruebas, riesgo potencial de impacto, y recomendaciones de correcciones.

Los resultados de la auditoría ponen de manifiesto que la aplicación analizada presenta **múltiples vulnerabilidades críticas, de severidad alta e intermedia**. La explotación de las vulnerabilidades revelaron cómo un atacante podría comprometer información sensible, suplantar la identidad de usuarios o ejecutar acciones no autorizadas en su nombre.

Como resultado de la auditoría, se recomienda priorizar la subsanación de las vulnerabilidades basándose en las calificaciones de gravedad proporcionadas, con la mayor inmediatez que sea posible, especialmente en las vulnerabilidades con calificación de gravedad más altas. Entre las principales recomendaciones generales destacamos:

- Implementación de mecanismos seguros de acceso a la información, consultas parametrizadas y principio de mínimo privilegio en la gestión de bases de datos.
- Aplicar validación y saneamiento de entradas tanto en el lado del servidor como del cliente.
- Reforzar la protección frente a ataques CSRF mediante tokens correctamente validados y configuraciones adecuadas de Cookies de sesión.
- Mantener actualizados los componentes de terceros y monitorizar vulnerabilidades asociadas a estos.



- Definir políticas de autenticación robustas alineadas con estándares y recomendaciones conocidas, como las del NIST.

La correcta adopción de estas medidas contribuiría significativamente a reducir la superficie de ataque y a mejorar el nivel general de seguridad de la aplicación web.



## Descripción del proceso de auditoría

### 1. Fase de reconocimiento

#### Procedimiento

En primer lugar se llevó a cabo una fase de recopilación de información. El objetivo de esta fase es **identificar la superficie de ataque inicial de la aplicación**, para posteriormente poder proceder a un análisis más detallado de la aplicación web y sus funcionalidades.

Utilizando la herramienta *nmap* se realizó un **escaneo completo de puertos** para intentar identificar todos los servicios potencialmente expuestos y el sistema operativo del servidor, y se exportaron los resultados en formato XML para su posterior análisis. Para ello se utilizó el siguiente comando:

```
nmap -p- -sV -O -oX escaneo_puertos.xml localhost
```

Los resultados muestran los puertos 8080, 9090 y 41031 abiertos corriendo un servicio HTTP. En los dos primeros se identificó un servidor web *Apache* y en el tercero un servidor HTTP implementado en Go. Se trata de puertos no estándar, lo que es habitual en entornos de desarrollo, interfaces de desarrollo, etc. que suelen presentar configuraciones menos restrictivas.

Los servicios usados utilizan el **protocolo HTTP sin cifrado TLS**, lo que implica que la comunicación entre cliente y servidor se realiza sin cifrar, en texto claro. Esto supondría un riesgo potencial para la confidencialidad de la información transmitida.

Además, pudimos estimar que el sistema operativo del servidor corresponde a Linux, con versión 5.0 - 6.2.

Posteriormente, se realizó una enumeración básica de directorios utilizando la herramienta *gobuster*, con el objetivo de identificar posibles endpoints accesibles de forma directa. Para ello se utilizó el siguiente comando:

```
gobuster dir -u http://localhost:8080/WebGoat -w /usr/share/wordlists/dirb/common.txt -o Desktop/practica_bootcamp/Information\gathering/enumeracion.txt -c "JSESSIONID=47CC65601567C6A9DB2301D75CFD4997"
```

Como resultado de este escaneo, se identificaron únicamente los endpoints correspondientes a **registro y login**. Mediante esta enumeración no se detectaron paneles de administración ni otros endpoints potencialmente sensibles accesibles de forma directa. No obstante, cabe destacar que estos resultados se encuentran limitados por la *wordlist* utilizada y el alcance de la enumeración realizada.

Utilizando la herramienta *Wappalyzer* se realizó un **escaneo de las tecnologías** utilizadas por la aplicación web, que nos permitió detectar las siguientes tecnologías:

1. Lenguaje de programación: Java
2. Frameworks de JavaScript: Backbone.js 1.4.0 y RequireJS 2.3.6
3. Librerías de JavaScript: jQuery 2.1.4, jQuery UI 1.10.4, Underscore.js



#### 4. Frameworks de UI: Bootstrap

#### 5. Scripts de fuentes: Font Awesome

Como podemos ver, se trata de algunas librerías desactualizadas y con vulnerabilidades reportadas, como en el caso de jQuery (principalmente relacionadas con Cross-Site Scripting) o RequireJS 2.3.6, que podrían suponer un riesgo potencial en función de cómo sean utilizadas dentro de la aplicación, y que por tanto se intentarán explotar en siguientes fases del proceso de la auditoría.



## 2. SQL Injection

<b>Severidad</b>	<b>Alta</b>
<b>CVSS v3.0 Score</b>	<b>8.3</b>
<b>CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L</b>	
<b>CVSSv3.0 Vector</b>	Attack Vector (AV): Network (N)
	User Interaction (UI): None (N)
	Attack Complexity (AC): Low (L)
	Confidentiality (C): High (H)
	Scope (S): Unchanged (U)
	Integrity (I): High (H)
	Privileged Required (PR): Low (L)
	Availability (A): Low (L)

### Descripción

En la ruta <http://localhost:8080/WebGoat/start.mvc?username=user#lesson/SqlInjection.lesson/10> se encuentra un formulario con dos campos de texto. El sistema está diseñado para introducir el nombre del empleado y su identificador único, y el primero se corresponde con el segundo, mostrar datos sensibles del usuario. Sin embargo, introduciendo el siguiente *payload* en el campo destinado al identificador, la respuesta mostrada se corresponde con los datos de todos los usuarios.

```
' or '1' = '1
```

Esto demuestra que el servidor encadena los parámetros de la *query SQL* como texto plano, y no como parámetros, lo que permite modificar el comportamiento deseado del programa mediante la introducción en el formulario de un texto que modifique la *query* en sí misma. En este caso, entendiendo que la *query* puede tener una forma similar a:

```
"SELECT * FROM employees  
WHERE last_name = '" + $EMPLEADO + "'  
AND auth_tan = '" + $TAN + "';"
```

al encadenar el texto enviado por el formulario quedaría de la siguiente manera:

```
"SELECT * FROM employees  
WHERE last_name = 'prueba'  
AND auth_tan = '' OR '1'='1';"
```

De esta manera, como el predicado "1" = "1" es verdadero, el SELECT devolvería todos los resultados contenidos en la tabla.

Una vez detectada esta vulnerabilidad, se procedió a ejecutar la herramienta **sqlmap**, que automatiza la introducción de parámetros en campos sensibles para conseguir extraer información de la tabla. Con el siguiente comando:



```
sqlmap -u http://localhost:8080/WebGoat/SqlInjection/attack8 -  
data=name=Smith&auth_tan=3SL99A  
--cookie=JSESSIONID=BFB6E97AC932A18BBD6E532F7591CEDF  
-p name,auth_tan  
--level=5  
--risk=3  
--technique=BEUSTQ  
--all
```

realizamos un ataque más agresivo que nos permitió acceder a los nombres de las bases de datos y sus respectivas tablas:

```
[*] CONTAINER  
[*] INFORMATION_SCHEMA  
[*] prueba  
[*] PUBLIC  
[*] SYSTEM_LOBS
```

```
+-----+  
| ASSIGNMENT  
| ASSIGNMENT_PROGRESS  
| EMAIL  
| LESSON_PROGRESS  
| LESSON_PROGRESS_ASSIGNMENTS  
| USER_PROGRESS  
| USER_PROGRESS_LESSON_PROGRESS  
| WEB_GOAT_USER  
+-----+
```

e incluso a usuarios y contraseñas de la base de datos:

```
[*] SA [1]:  
password hash: d41d8cd98f00b204e9800998ecf8427e  
clear-text password:  
[*] UNAUTHORIZED_USER [1]:  
password hash: 033bd94b1168d7e4f0d644c3c95e35bf  
clear-text password: TEST
```

Esto nos indica, además, que el usuario desde el que se ejecutan las consultas es por defecto un usuario con privilegios excesivos para lo que se esperaría de la funcionalidad.



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:

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



Employee Name:

Authentication TAN:

**You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!**

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

## Mitigación

Para mitigar esta vulnerabilidad, la aplicación debería implementar el uso de consultas parametrizadas, evitando la concatenación directa de entradas de usuario en las consultas SQL. De este modo, los datos introducidos por el usuario se tratarían como parámetros y se evitaría su uso como instrucción SQL.

Además, se recomienda la validación y saneamiento de entradas, aplicando restricciones de tipo, longitud y formato cuando sea posible.

Complementariamente, se deberán limitar los permisos del usuario desde el que se ejecutan las consultas en la medida que sea posible, siguiendo el principio de mínimo privilegio.



### 3. XSS

<b>Severidad</b> <b>CVSS v3.0 Score</b>	<b>Media</b> <b>5.3</b>	
<b>CVSSv3.0 Vector</b>	<b>CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N</b>	
	Attack Vector (AV): Network (N)	User Interaction (UI): Required (R)
	Attack Complexity (AC): Low (L)	Confidentiality (C): Low (L)
	Scope (S): Unchanged (U)	Integrity (I): Low (L)
	Privileged Required (PR): None (N)	Availability (A): None (N)

#### Descripción

En la ruta <http://localhost:8080/WebGoat/start.mvc?username=user#lesson/CrossSiteScripting.lesson/6> encontramos un formulario con seis campos de entrada, 4 de número y 2 de texto. Al introducir el payload `<script>alert("Hola")</script>` en los diferentes campos y enviar el formulario, se pudo comprobar que, en el caso del campo destinado a introducir el número de tarjeta, se ejecutó un mensaje de alerta en el navegador con el mensaje introducido, lo que demuestra que este campo es vulnerable a Cross-Site Scripting o XSS.

Esta vulnerabilidad se produce debido a que el valor introducido por el usuario es devuelto por la aplicación en la respuesta HTML sin aplicar mecanismos adecuados de validación, permitiendo que el navegador interprete la entrada como código ejecutable.

La ejecución de código JavaScript en el navegador del usuario implica riesgos relevantes. Sin embargo, tras probar a explotar la vulnerabilidad enviando un formulario externo, simulando la acción de un agente malicioso, detectamos que es consistente con una vulnerabilidad XSS DOM-based: el código vulnerable se encuentra en el JavaScript del lado del cliente, que inserta directamente el valor del campo en el DOM sin sanitización. No se detectó persistencia del payload, por lo que no se trataría de un XSS almacenado.



Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	2	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	3	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	4	\$0.00

Enter your credit card number

Enter your three digit security code

localhost:8080

script>alert("Vulnerable");

Vulnerable

OK

**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

We have charged credit card:  
\$6125.900000000001

## Mitigación

Para prevenir vulnerabilidades de XSS se recomienda la validación de entradas en el lado del servidor y el uso de librerías o frameworks que automaticen el escapado antes de renderizar los datos introducidos por el usuario en el HTML o en el DOM.



## 4. Cross-Site Request Forgeries (CSRF)

<b>Severidad</b> <b>CVSS v3.0 Score</b>	<b>Media</b> <b>6.5</b>	
	<b>CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N</b>	
<b>CVSSv3.0 Vector</b>	Attack Vector (AV): Network (N)	User Interaction (UI): Required (R)
	Attack Complexity (AC): Low (L)	Confidentiality (C): None (N)
	Scope (S): Unchanged (U)	Integrity (I): High (H)
	Privileged Required (PR): None (N)	Availability (A): None (N)

### Descripción

En la ruta <http://localhost:8080/WebGoat/start.mvc?username=user#lesson/CSRF.lesson/3> encontramos un formulario destinado a la publicación de comentarios, con un campo de texto para introducir el comentario y un campo numérico para introducir una puntuación.

Aunque al inspeccionar el código HTML del formulario hallamos un token CSRF de validación, al copiar el formulario, incluso sin el token incluido, y reproducirlo en un fichero diferente y ejecutarlo, observamos que el comentario que registramos desde un origen externo a la aplicación queda guardado en la base de datos desde el perfil del usuario registrado autenticado. Esto se produce incluso a pesar de la respuesta de error que devuelve el servidor, lo que demuestra que el mecanismo que se está utilizando para proteger el formulario de CSRF no está funcionando como se espera.

Esta vulnerabilidad puede permitir a atacantes ejecutar solicitudes al servidor por parte de usuarios sin su consentimiento.

### Mitigación

Para mitigar esta vulnerabilidad se recomienda generar un token CSRF impredecible, único por sesión o petición, y que este sea verificado en cada petición al servidor. Además, se deberá configurar las cookies de sesión con el atributo SameSite=Lax o SameSite=Strict, para evitar que el navegador envíe automáticamente la cookie de sesión en peticiones cross-site iniciadas desde formularios externos. Además deberá revisarse el control de flujo entre lógica de negocio y validación de seguridad, ya que en este caso sí se devuelve un error lógico cuando se hace una petición desde fuera del sitio, pero aun así se procesa la solicitud.



## 5. Vulnerable and Outdated Components

<b>Severidad</b> <b>CVSS v3.0 Score</b>	<b>Media</b> <b>5.3</b>	
<b>CVSSv3.0 Vector</b>	<b>CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N</b>	
	Attack Vector (AV): Network (N)	User Interaction (UI): Required (R)
	Attack Complexity (AC): Low (L)	Confidentiality (C): Low (L)
	Scope (S): Unchanged (U)	Integrity (I): Low (L)
	Privileged Required (PR): None (N)	Availability (A): None (N)

### Descripción

En la ruta <http://localhost:8080/WebGoat/start.mvc?username=user#lesson/VulnerableComponents.lesson/4> encontramos dos formularios con un campo de texto cada uno. Al pulsar sobre el botón “Go!” de cada uno de ellos debería mostrarse un cuadro de diálogo donde el icono de cerrar sería sustituido por el texto introducido por el usuario en el formulario. En el primer formulario se utiliza la versión 1.10.4 de jquery-ui, mientras que en la segunda se utiliza una más actualizada, la 1.12.0.

La versión 1.10.4 de jQuery-UI es un componente obsoleto que presenta vulnerabilidades de seguridad ya conocidas. Esta versión no realiza una correcta sanitización del contenido proporcionado por el usuario del parámetro closeText, lo que puede dar lugar a vulnerabilidades de tipo Cross-Site Scripting (XSS). Un atacante podría aprovecharse de esta vulnerabilidad para ejecutar código JavaScript malicioso en el navegador de la víctima, tal y como se explicó en el apartado dedicado a esta vulnerabilidad. Es importante destacar que la vulnerabilidad no reside en el código de la aplicación sino en el uso de una dependencia desactualizada.

### Mitigación

Para mitigar este riesgo se recomienda la actualización de las dependencias a versiones libres de vulnerabilidades conocidas. Además, mantener un inventario actualizado de componentes de terceros y monitorizar vulnerabilidades conocidas (CVE) asociadas a las librerías utilizadas.



## 6. Identity & Auth Failure

<b>Severidad</b> <b>CVSS v3.0 Score</b>	<b>Media</b> <b>5.3</b>	
<b>CVSSv3.0 Vector</b>	<b>CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N</b>	
	Attack Vector (AV): Network (N)	User Interaction (UI): Required (R)
	Attack Complexity (AC): Low (L)	Confidentiality (C): Low (L)
	Scope (S): Unchanged (U)	Integrity (I): Low (L)
	Privileged Required (PR): None (N)	Availability (A): None (N)

### Descripción

En la ruta <http://localhost:8080/WebGoat/start.mvc?username=user#lesson/SecurePasswords.lesson/3> encontramos un formulario para evaluar la robustez de las contraseñas introducidas. La aplicación permite el uso de contraseñas débiles y fácilmente predecibles, tales como:

- Palabras y patrones comunes: password, contraseña, 123456, abcabc
- Caracteres individuales sin mínimo: a, 1
- Fechas: 2018/10/04, 15061998

Las recomendaciones más importantes del NIST (National Institute of Standards and Technology) de Estados Unidos contemplan lo siguiente:

- Sin reglas de composición: permitir el uso de mayúsculas y minúsculas y caracteres especiales, pero no forzar al usuario a utilizarlas.
- Sin pistas de contraseña.
- Sin preguntas de seguridad.
- Sin cambios de contraseña innecesarios.
- Mínimo de 8 caracteres.
- Soporte todos los caracteres UNICODE, incluyendo emojis y espacios en blanco.
- Comprobación contra contraseñas comunes (contraseñas obtenidas en anteriores brechas de seguridad, palabras de diccionario, patrones repetitivos, palabras relacionadas con el contexto como el nombre del servicio o del usuario, etc.)



Además, recomienda algunas cuestiones de usabilidad como el permitir pegar en el input de contraseña, para favorecer el uso de gestores de contraseñas, permitir mostrar la contraseña en claro y ofrecer un medidor de seguridad.

Como comprobamos, las recomendaciones de usabilidad sí que se cumplen en el formulario, pero las de composición de contraseñas en su mayoría no.

La ausencia de una política estricta de contraseñas supone un fallo en los mecanismos de autenticación, aumentando significativamente el riesgo de acceso no autorizado a cuentas de usuario.



## Herramientas utilizadas

Durante la realización de la auditoría se emplearon diversas herramientas de seguridad, tanto para la fase de reconocimiento como para la detección y explotación de vulnerabilidades, complementadas con análisis manual. Las herramientas utilizadas fueron las siguientes:

- **Wappalyzer:** utilizada para la identificación de las tecnologías empleadas por la aplicación web, incluyendo lenguajes de programación, frameworks y librerías de terceros.
- **Nmap:** empleada para la identificación de puertos abiertos, servicios expuestos y estimación del sistema operativo del servidor, permitiendo determinar la superficie de ataque inicial de la aplicación.
- **Gobuster:** utilizada para la enumeración de directorios y endpoints accesibles en la aplicación web, con el objetivo de detectar recursos expuestos de forma directa.
- **Sqlmap:** herramienta de automatización de ataques de inyección SQL, empleada para confirmar y explotar la vulnerabilidad de SQL Injection detectada, así como para la enumeración de bases de datos, tablas y credenciales.
- **Navegador web y herramientas de desarrollo (DevTools):** empleadas para la inspección del código HTML y JavaScript, análisis del comportamiento del lado cliente, identificación de vulnerabilidades XSS y CSRF, y verificación manual de los resultados obtenidos mediante herramientas automatizadas.

El uso combinado de herramientas automáticas y análisis manual permitió obtener una visión completa del estado de seguridad de la aplicación auditada y validar de forma rigurosa los hallazgos documentados en este informe.