

# Desarrollo de Software Seguro

Diego Saavedra

Aug 15, 2024

# Table of contents

<b>1</b>	<b>Bienvenido</b>	<b>8</b>
1.1	Descripción de la Asignatura: . . . . .	8
1.2	Contribución de la Asignatura: . . . . .	8
1.3	Resultado de Aprendizaje de la Carrera: (Unidad de Competencia) . . . . .	8
1.4	Objetivo de la Asignatura: (Unidad de Competencia) . . . . .	8
1.5	Resultado de Aprendizaje de la Asignatura: (Elemento de Competencia) . . . . .	9
<b>I</b>	<b>Laboratorios de Desarrollo de Software Seguro</b>	<b>10</b>
<b>2</b>	<b>Laboratorio de Owasp Zap</b>	<b>11</b>
2.1	Contenido . . . . .	11
2.2	Objetivo . . . . .	11
2.3	Introducción . . . . .	12
2.4	Requisitos . . . . .	12
2.5	Escenario . . . . .	12
2.6	Ejercicio 1: Escaneo Automático de la aplicación web de DVWA . . . . .	14
2.7	Ejercicio 2: Escaneo Manual de la aplicación web de DVWA . . . . .	18
2.8	Recursos adicionales . . . . .	23
2.9	Conclusiones . . . . .	23
2.10	Recomendaciones . . . . .	23
2.11	Autoevaluación . . . . .	24
2.12	Referencias . . . . .	24
<b>3</b>	<b>Estandar de Seguridad CWE</b>	<b>25</b>
3.1	¿Qué es CWE? . . . . .	25
3.2	¿Cómo se utiliza? . . . . .	25
3.3	¿Por qué es importante? . . . . .	25
3.4	¿Cómo se utiliza en la gestión de vulnerabilidades? . . . . .	25
3.5	¿Qué es un CWE ID? . . . . .	26
3.6	¿Qué es una CWE Weakness? . . . . .	26
3.7	¿Qué es una CWE Category? . . . . .	26
3.8	¿Qué es una CWE View? . . . . .	26
3.9	¿Qué es una CWE Entry? . . . . .	26
3.10	¿Qué es una CWE Relationship? . . . . .	26
3.11	¿Qué es una CWE Mapping? . . . . .	27
3.12	¿Qué es una CWE Group? . . . . .	27
<b>4</b>	<b>Top 25 de las debilidades de software más peligrosas.</b>	<b>28</b>
4.1	¿Cómo se clasifican las debilidades de software en CWE? . . . . .	28

<b>5</b>	<b>¿Cómo se analiza la información de CWE?</b>	<b>32</b>
5.1	1. <b>CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</b> . . . . .	32
5.2	2. <b>CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</b> . . . . .	33
5.3	3. <b>CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</b> . . . . .	33
5.4	4. <b>CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</b> . . . . .	33
5.5	5. <b>CWE-94: Improper Control of Generation of Code ('Code Injection')</b> . . . . .	33
5.6	6. <b>CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</b> . . . . .	33
5.7	7. <b>CWE-190: Integer Overflow or Wraparound</b> . . . . .	34
5.8	8. <b>CWE-200: Information Exposure</b> . . . . .	34
5.9	9. <b>CWE-264: Permissions, Privileges, and Access Controls</b> . . . . .	34
5.10	10. <b>CWE-269: Improper Privilege Management</b> . . . . .	34
5.11	11. <b>CWE-352: Cross-Site Request Forgery (CSRF)</b> . . . . .	34
5.12	12. <b>CWE-434: Unrestricted Upload of File with Dangerous Type</b> . . . . .	35
5.13	13. <b>CWE-601: URL Redirection to Untrusted Site ('Open Redirect')</b> . . . . .	35
5.14	14. <b>CWE-732: Incorrect Permission Assignment for Critical Resource</b> . . . . .	35
5.15	15. <b>CWE-798: Use of Hard-coded Credentials</b> . . . . .	35
5.16	16. <b>CWE-807: Reliance on Untrusted Inputs in a Security Decision</b> . . . . .	35
5.17	17. <b>CWE-862: Missing Authorization</b> . . . . .	36
5.18	18. <b>CWE-863: Incorrect Authorization</b> . . . . .	36
5.19	19. <b>CWE-918: Server-Side Request Forgery (SSRF)</b> . . . . .	36
5.20	20. <b>CWE-943: Improper Neutralization of Special Elements in Data Query Logic ('Injection')</b> . . . . .	36
5.21	21. <b>CWE-959: Use of Password System for Primary Authentication</b> . . . . .	36
5.22	22. <b>CWE-1168: Inadequate Encryption Strength</b> . . . . .	37
5.23	23. <b>CWE-1170: Improper Restriction of XML External Entity Reference ('XXE')</b> . . . . .	37
5.24	24. <b>CWE-1177: Improper Output Neutralization for Logs</b> . . . . .	37
5.25	25. <b>CWE-1189: Improper Isolation of Shared Resources on System</b> . . . . .	37
5.26	Conclusión . . . . .	37
5.27	Referencias . . . . .	38
<b>6</b>	<b>Laboratorio de estandar de Vulnerabilidades con CWE</b>	<b>39</b>
6.1	1. Objetivos . . . . .	39
6.2	2. Desarrollo . . . . .	39
	6.2.1 CWE . . . . .	39
	6.2.2 CVE . . . . .	39
6.3	OWASP Juise Shop. . . . .	40
6.4	Ejercicio . . . . .	42
6.5	Solución . . . . .	42
6.6	Conclusión . . . . .	43

6.7	Motivación . . . . .	44
6.7.1	Herramientas . . . . .	44
6.7.2	Referencias . . . . .	44
<b>7</b>	<b>Laboratorio Webgoat</b>	<b>45</b>
7.1	Introducción . . . . .	45
7.2	Instalación . . . . .	45
7.3	Laboratorio . . . . .	46
7.4	Conclusión . . . . .	49
7.5	Referencias . . . . .	49
<b>8</b>	<b>Laboratorio de Configuración de TLS/SSL</b>	<b>50</b>
8.1	Objetivos: . . . . .	50
<b>9</b>	<b>Herramientas</b>	<b>51</b>
<b>10</b>	<b>Procedimiento</b>	<b>52</b>
10.1	1. Instalación de Nginx . . . . .	52
10.2	2. Configuración de Nginx . . . . .	52
10.3	3. Configuración de TLS/SSL . . . . .	55
10.4	4. Verificación de la Configuración de TLS/SSL . . . . .	56
10.5	Conclusión . . . . .	57
<b>11</b>	<b>Referencias</b>	<b>58</b>
<b>12</b>	<b>Laboratorio de autenticación OAuth 2 y JWT con Spring Security Authorization Server.</b>	<b>59</b>
<b>13</b>	<b>Instrucciones</b>	<b>60</b>
<b>14</b>	<b>Reto</b>	<b>70</b>
<b>15</b>	<b>Conclusiones</b>	<b>71</b>
<b>16</b>	<b>Referencias</b>	<b>72</b>
<b>17</b>	<b>Laboratorio: Aplicación de Buenas Prácticas de Seguridad</b>	<b>73</b>
17.1	Instrucciones del Laboratorio . . . . .	73
17.2	Descripción del laboratorio: . . . . .	73
17.3	Parte 1 - Análisis de Vulnerabilidades . . . . .	73
17.3.1	Configuración del entorno: . . . . .	73
17.4	Parte 2 - Implementación de Medidas de Seguridad . . . . .	74
17.5	Parte 3 - Documentación . . . . .	74
17.6	Entregables: . . . . .	74
17.7	Evaluación: . . . . .	75
17.8	Recursos: . . . . .	75
17.9	Conclusiones . . . . .	75
17.10	Referencias . . . . .	75

<b>18 Laboratorio: Implementación de Doble Factor de Autenticación (2FA) con Google Authenticator</b>	<b>78</b>
18.1 Objetivos . . . . .	78
18.2 Prerrequisitos . . . . .	78
18.3 Paso 1: Configuración del Proyecto . . . . .	79
18.4 Paso 2: Generación del Secreto y Código QR . . . . .	79
18.5 Paso 3: Creación del Archivo HTML . . . . .	80
18.6 Paso 4: Verificación del Token . . . . .	81
<b>19 Conclusión</b>	<b>84</b>
<b>20 Laboratorio de Codificación Segura</b>	<b>85</b>
20.1 Python . . . . .	85
20.2 Objetivo: . . . . .	85
20.3 Requisitos: . . . . .	85
20.4 Java . . . . .	88
20.5 Objetivo: . . . . .	88
20.6 Requisitos: . . . . .	88
<b>21 JavaScript</b>	<b>91</b>
21.1 Objetivo: . . . . .	91
21.2 Requisitos: . . . . .	91
<b>22 C#</b>	<b>95</b>
22.1 Objetivo: . . . . .	95
22.2 Requisitos: . . . . .	95
<b>23 Laboratorio TDD</b>	<b>100</b>
23.1 Objetivo . . . . .	100
23.2 Desarrollo . . . . .	100
<b>24 Reto</b>	<b>103</b>
<b>25 TDD con Python</b>	<b>104</b>
<b>26 Reto</b>	<b>106</b>
<b>27 TDD con JavaScript</b>	<b>107</b>
<b>28 Reto</b>	<b>109</b>
<b>29 Conclusión</b>	<b>110</b>
<b>30 Laboratorio de Pruebas Unitarias</b>	<b>111</b>
30.1 Objetivo . . . . .	111
30.2 Desarrollo . . . . .	111
30.2.1 Java . . . . .	112
30.2.2 Python . . . . .	113
30.2.3 JavaScript . . . . .	114

<b>II Laboratorios de Pentesting</b>	<b>117</b>
31 Introducción	118
32 ¿Qué me ofrece la academia PortSwigger?	119
33 ¿Cómo se divide la academia PortSwigger?	120
34 Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data	128
35 Objetivo	129
36 Herramientas	130
37 Desarrollo	131
38 Conclusión	135
39 Lab 2: SQL injection vulnearbility allowing login bypass	136
39.1 Objetivo . . . . .	136
39.2 Desarrollo . . . . .	137
40 Conclusión	140
<b>III Laboratorios de Fuzzing</b>	<b>141</b>
41 Laboratorio: Integración de Proyectos de Código Abierto con OSS-Fuzz	142
41.1 Objetivo: . . . . .	142
41.2 Requisitos Previos . . . . .	142
41.3 Temas del Laboratorio . . . . .	142
41.4 1. Introducción a OSS-Fuzz . . . . .	142
41.4.1 Arquitectura de OSS-Fuzz . . . . .	143
41.5 2. Configuración del Entorno . . . . .	143
41.5.1 2.1. Clonar el Proyecto de Ejemplo . . . . .	143
41.5.2 2.2. Instalación de Dependencias . . . . .	144
41.6 3. Integración de un Proyecto en OSS-Fuzz . . . . .	144
41.6.1 3.1. Archivos de Configuración . . . . .	144
41.6.2 3.2. Dockerfile . . . . .	144
41.6.3 3.3. Fuzzers . . . . .	144
41.6.4 3.4. build.sh . . . . .	145
41.7 4. Ejecución de Fuzzing y Análisis de Resultados . . . . .	145
41.7.1 4.1. Compilar y Ejecutar el Fuzzer . . . . .	145
41.7.2 4.2. Análisis de Resultados . . . . .	145
42 Conclusión	146

<b>IV OWASP Top 10</b>	<b>147</b>
<b>43 Introducción a OWASP top 10</b>	<b>148</b>
43.1 ¿Por qué es importante conocer los riesgos de seguridad en las aplicaciones web? . . . . .	148
43.2 ¿Qué es el OWASP Top 10? . . . . .	149
43.3 ¿Cuáles son los riesgos más críticos en la seguridad de las aplicaciones web? . . . . .	149
43.4 1. Inyección SQL . . . . .	149
43.5 2. Autenticación y gestión de sesiones defectuosas . . . . .	150
43.6 3. Exposición de datos sensibles . . . . .	151
43.7 4. XML External Entities (XXE) . . . . .	151
43.8 5. Control de acceso inadecuado . . . . .	152
43.9 6. Configuración de seguridad defectuosa . . . . .	152
43.107. Cross-Site Scripting (XSS) . . . . .	153
43.118. Deserialización insegura . . . . .	153
43.129. Utilización de componentes con vulnerabilidades conocidas . . . . .	154
43.1310. Insuficiente monitorización y registro de eventos . . . . .	155
43.14 Conclusión . . . . .	155
<b>44 Referencias</b>	<b>156</b>

# 1 Bienvenido

¡Bienvenido a la asignatura de Desarrollo de Software Seguro!

## 1.1 Descripción de la Asignatura:

Esta asignatura analiza los aspectos de desarrollo de software cuando se trata de desarrollar un sistema software que debe tener la propiedad de ser seguro.

## 1.2 Contribución de la Asignatura:

Esta asignatura permitirá conocer y aplicar los procesos, modelos, normas y estándares de seguridad para garantizar la integridad, seguridad y fiabilidad de los datos en un sistema software

## 1.3 Resultado de Aprendizaje de la Carrera: (Unidad de Competencia)

Reconoce los principios fundamentales de la Ingeniería de Software a través de la elaboración de algoritmos, pseudocódigo, pruebas de escritorio y codificación de programas básicos de computadores con pensamiento lógico y creativo a fin de solucionar problemas en diferentes dominios.

Desarrollar sistemas software seguros aplicando en el proceso de desarrollo de software para garantizar la integridad, seguridad y fiabilidad de los datos al usuario final de la aplicación software.

## 1.4 Objetivo de la Asignatura: (Unidad de Competencia)

Formar profesionales en Ingeniería de Software capaces de aplicar modelos, normas y estándares necesarios para obtener productos y procesos software de alta calidad que garanticen el cumplimiento de requerimientos, demostrando creatividad, eficiencia, eficacia y responsabilidad profesional; con el propósito de contribuir a la innovación tecnológica en las organizaciones y a la mejora de los sectores productivos del País.

Esta asignatura corresponde a la primera etapa del eje de formación profesional, proporciona al futuro profesional las bases conceptuales para garantizar la calidad del proceso y producto del desarrollo del software.

## **1.5 Resultado de Aprendizaje de la Asignatura: (Elemento de Competencia)**

Comprender la ciencia detrás de los sistemas de información basados en la Web y los principios de ingeniería de los sistemas de Información Web.

Efectúa análisis sobre los componentes software necesarios en proyectos de desarrollo de sistemas Web para con ello lograr la implementación recuperación de la información web y extracción de la información aplicado en la web para análisis de información.

Conoce los componentes, arquitectura, tecnología y la seguridad que se puede dar en las aplicaciones Web, para tomar medidas respectivas de los posibles ataques que se pueden producirse en las mismas. Aplica, discute y reconoce de forma efectiva las diferentes medidas para evitar los posibles ataques en las aplicaciones web.

Participa pro-activamente en un equipo de trabajo, resolviendo problemas que empleen conceptos de seguridad en las aplicaciones web.

Integrar los conceptos utilizados en el diseño, desarrollo y el despliegue de aplicaciones basadas en la Web mostrando participación activa como parte de trabajo colaborativo en diferentes casos de estudios.

## **Part I**

# **Laboratorios de Desarrollo de Software Seguro**

## 2 Laboratorio de Owasp Zap



### 2.1 Contenido

- [Objetivo](#)
- [Introducción](#)
- [Requisitos](#)
- [Escenario](#)
- [Ejercicio 1](#)
- [Ejercicio 2](#)
- [Recursos adicionales](#)
- [Conclusiones](#)
- [Recomendaciones](#)
- [Autoevaluación](#)
- [Referencias](#)

### 2.2 Objetivo

Este laboratorio tiene como objetivo enseñar a los estudiantes a utilizar OWASP ZAP, una herramienta de seguridad que nos permite realizar pruebas de seguridad en aplicaciones web.

Owasp Zap es una herramienta de seguridad que nos permite realizar pruebas de seguridad en aplicaciones web. En este laboratorio vamos a aprender a utilizar Owasp Zap para realizar pruebas de seguridad en una aplicación web.

## 2.3 Introducción

En este laboratorio vamos a aprender a utilizar OWASP ZAP, una herramienta de seguridad que nos permite realizar pruebas de seguridad en aplicaciones web.

## 2.4 Requisitos

Para este laboratorio necesitamos tener instalado OWASP ZAP en nuestra máquina. Puedes descargarlo desde la página oficial de OWASP: <https://www.zaproxy.org/>

Además, necesitamos tener una aplicación web vulnerable para realizar pruebas de seguridad con OWASP ZAP. En este laboratorio vamos a utilizar una aplicación web vulnerable llamada DVWA (Damn Vulnerable Web Application). Puedes descargar DVWA desde el siguiente enlace: <https://github.com/digininja/DVWA>

## 2.5 Escenario

Vamos a utilizar una aplicación web vulnerable llamada DVWA (Damn Vulnerable Web Application) para realizar pruebas de seguridad con OWASP ZAP. Puedes descargar DVWA desde el siguiente enlace: <https://github.com/digininja/DVWA/>

Puede utilizar git para clonar el repositorio de DVWA en su máquina:

```
git clone https://github.com/digininja/DVWA.git
```

A continuación necesitamos ejecutar en local el servidor web de DVWA. Para ello, podemos utilizar un servidor web como Apache o Nginx. En este laboratorio vamos a utilizar Docker, por lo que necesitamos tener instalado Docker en nuestra máquina.

### Tip

Tenemos algunas alternativas para ejecutar DVWA:

1. Instalar DVWA en un servidor web local como Apache o Nginx [link](#).
2. Utilizar un contenedor Docker para ejecutar DVWA [link](#).
3. Utilizar VirtualBox o VMware para ejecutar DVWA en una máquina virtual [link](#).

En este laboratorio vamos a utilizar DVWA con Docker, para inciciar el laboratorio podemos utilizar el siguiente comando:

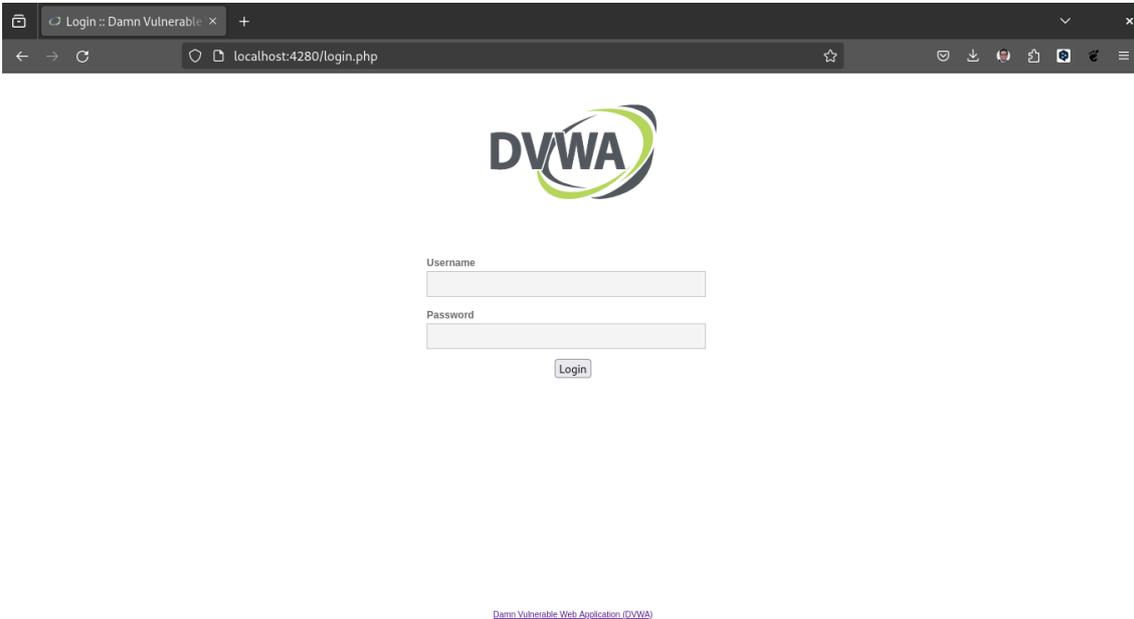
```
docker compose up --build -d
```

```

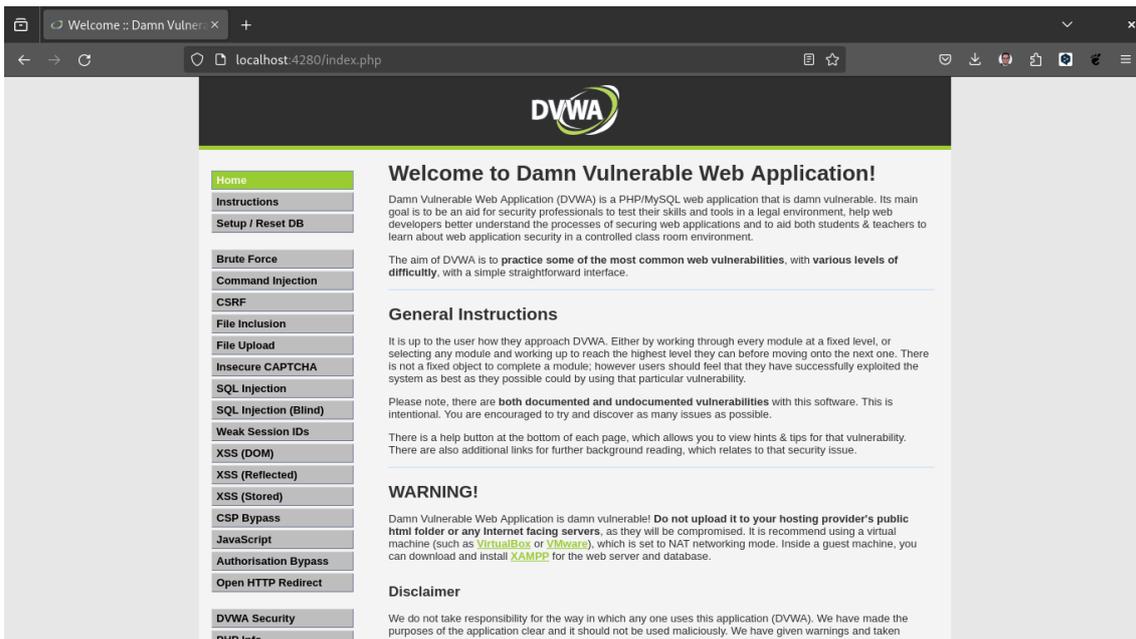
=> => sha256:2501a14769274afe54d0be386b8459c80897c1ceeb1cf327be67af08897dbb34 492B / 492B 2.8s
=> => sha256:df3805506bf7147e7fb98efaa6451681e21b261bc066f7572a9c16aaa52974fd 11.64MB / 11.64MB 3.9s
=> => sha256:dbc13994001c458d528afe09a8edc6fa3f5e72f9e357c0da7297c804f280d385 247B / 247B 4.2s
=> => sha256:6019b47fb1809aa59c6239a94bdb523e7b8d28510e012e9899575f7f9cd5e3f8 2.46kB / 2.46kB 4.2s
=> => sha256:87c35eeeba3a99c5d7b4294d22c12858d9b0a5d3c5fd92079e907cbee879e27d 894B / 894B 5.2s
=> => extracting sha256:e74cc574d0d2183a964ea5880bba6308b7653c4cc7ff2010beb2c1b157168a3a 10.6s
=> => extracting sha256:e4782e138a9004836453694f515681a93d6635ab26f05b7f2dfea2baa4b761ad 0.0s
=> => extracting sha256:cfeec87621aeb5b3bf8907398acc95a5e9b85a5f3faa6a68e0640e7d6653f850 3.8s
=> => extracting sha256:c1badcd002c0a773002eaf87724b5a303e17873ae752e7e69dfd6b22c05a7dea 0.0s
=> => extracting sha256:e0d463a60cb62b1b64dd43c0defef018c6ab557e474c029d17824cc9d1b5d103 0.0s
=> => extracting sha256:e517da2e974fb9838a1bb2edf8418fdda3bd56bebb24aba4e6345511b182a8e2 0.8s
=> => extracting sha256:2501a14769274afe54d0be386b8459c80897c1ceeb1cf327be67af08897dbb34 0.0s
=> => extracting sha256:df3805506bf7147e7fb98efaa6451681e21b261bc066f7572a9c16aaa52974fd 3.2s
=> => extracting sha256:6019b47fb1809aa59c6239a94bdb523e7b8d28510e012e9899575f7f9cd5e3f8 0.0s
=> => extracting sha256:dbc13994001c458d528afe09a8edc6fa3f5e72f9e357c0da7297c804f280d385 0.0s
=> => extracting sha256:87c35eeeba3a99c5d7b4294d22c12858d9b0a5d3c5fd92079e907cbee879e27d 0.0s
=> [dvwa internal] load build context 0.4s
=> transferring context: 1.34MB 0.3s
=> [dvwa 2/5] WORKDIR /var/www/html 0.5s
=> [dvwa 3/5] RUN apt-get update && export DEBIAN_FRONTEND=noninteractive && apt-get install 60.6s
=> [dvwa 4/5] COPY --chown=www-data:www-data . . 0.2s
=> [dvwa 5/5] COPY --chown=www-data:www-data config/config.inc.php.dist config/config.inc.php 0.1s
=> [dvwa] exporting to image 0.3s
=> => exporting layers 0.2s
=> => writing image sha256:484a9a1b37a08d0af5b2071d407c96b04db658a9175fb316aaa30aee2abfae2e 0.0s
=> => naming to ghcr.io/digininja/dvwa:latest 0.0s
[+] Running 4/4
✔ Network dvwa_dvwa Created 0.1s
✔ Volume "dvwa_dvwa" Created 0.0s
✔ Container dvwa-db-1 Started 0.6s
✔ Container dvwa-dvwa-1 Started 1.2s
static@lenovo ~/.../lab001/DVWA master

```

Una vez que el contenedor de DVWA esté en ejecución, podemos acceder a la aplicación web en la dirección <http://localhost:4280>. La aplicación web de DVWA nos pedirá que iniciemos sesión con un usuario y contraseña. Por defecto, el usuario es **admin** y la contraseña es **password**.



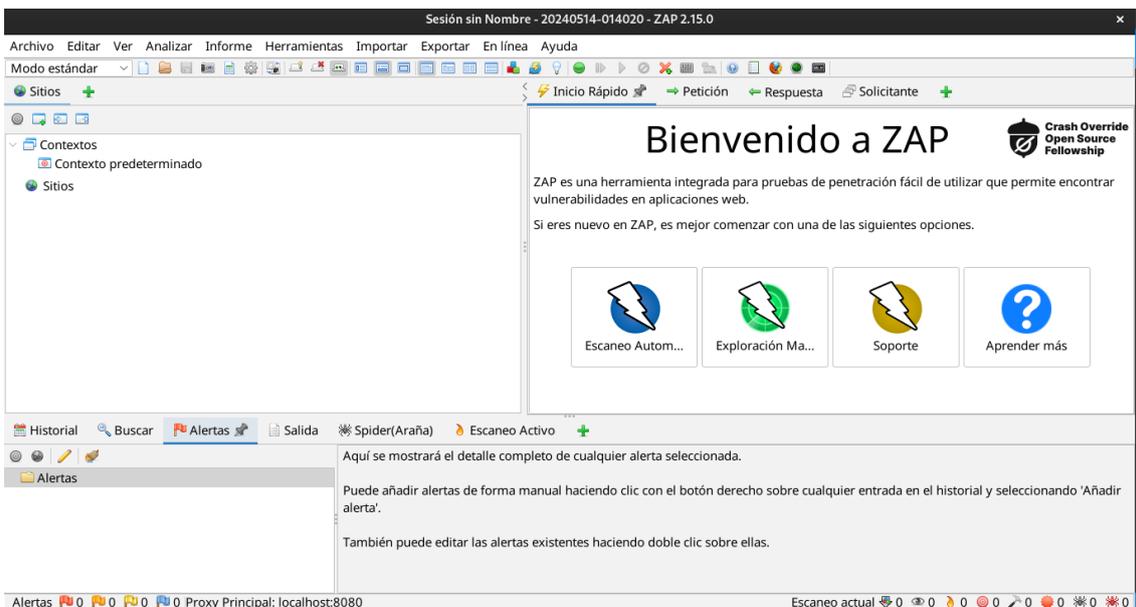
Una vez que hayamos iniciado sesión en la aplicación web de DVWA, podemos comenzar a realizar pruebas de seguridad con OWASP ZAP.



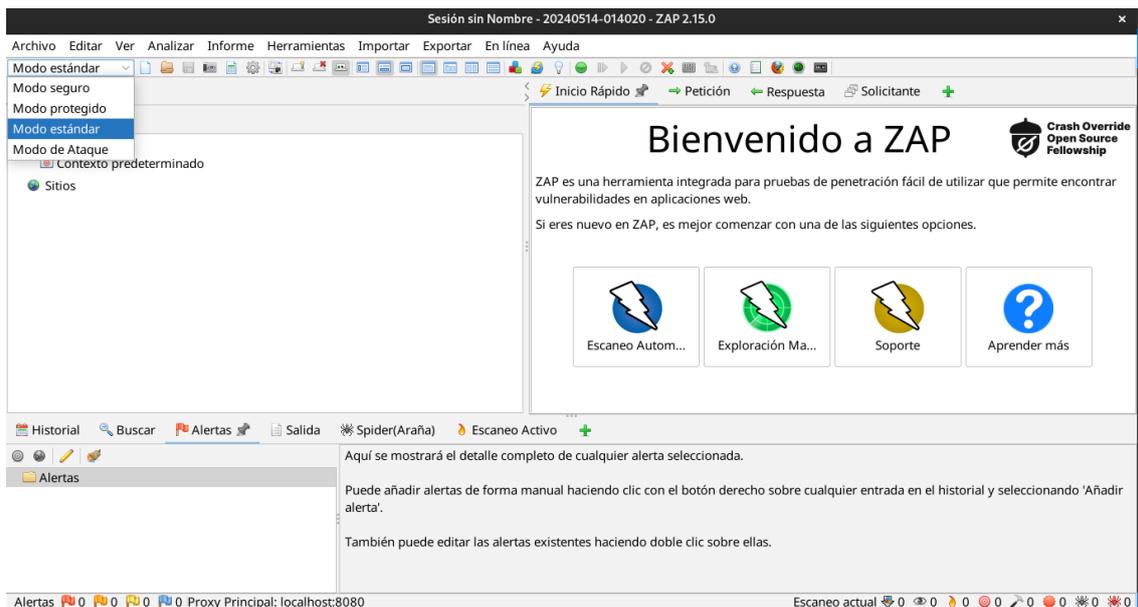
## 2.6 Ejercicio 1: Escaneo Automático de la aplicación web de DVWA

En este ejercicio vamos a configurar OWASP ZAP para realizar pruebas de seguridad en la aplicación web de DVWA.

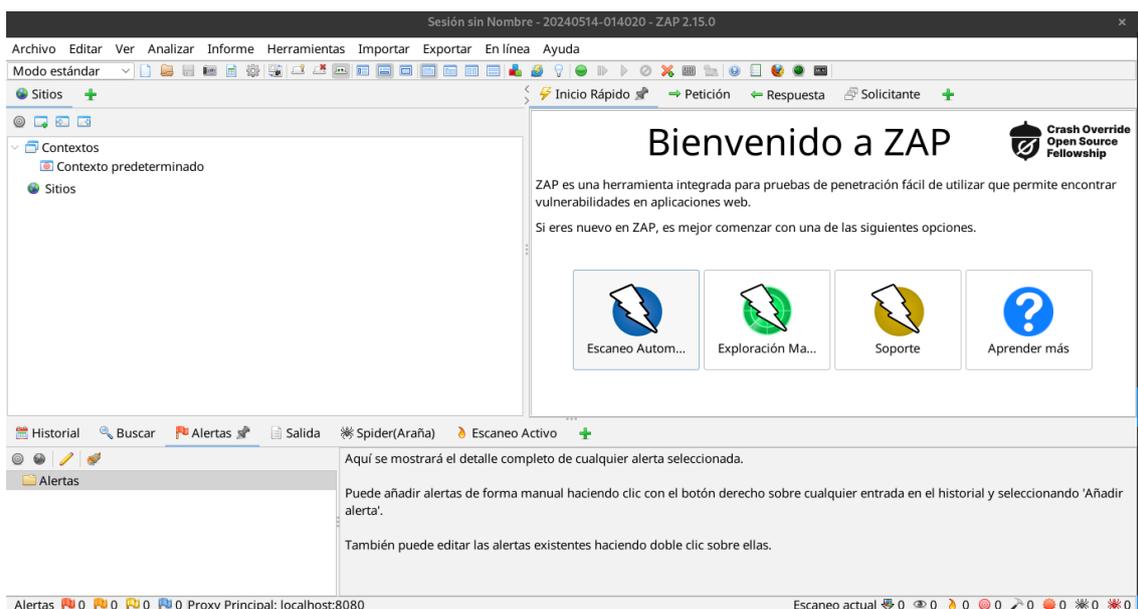
1. Abre OWASP ZAP en tu máquina. Puedes abrir OWASP ZAP



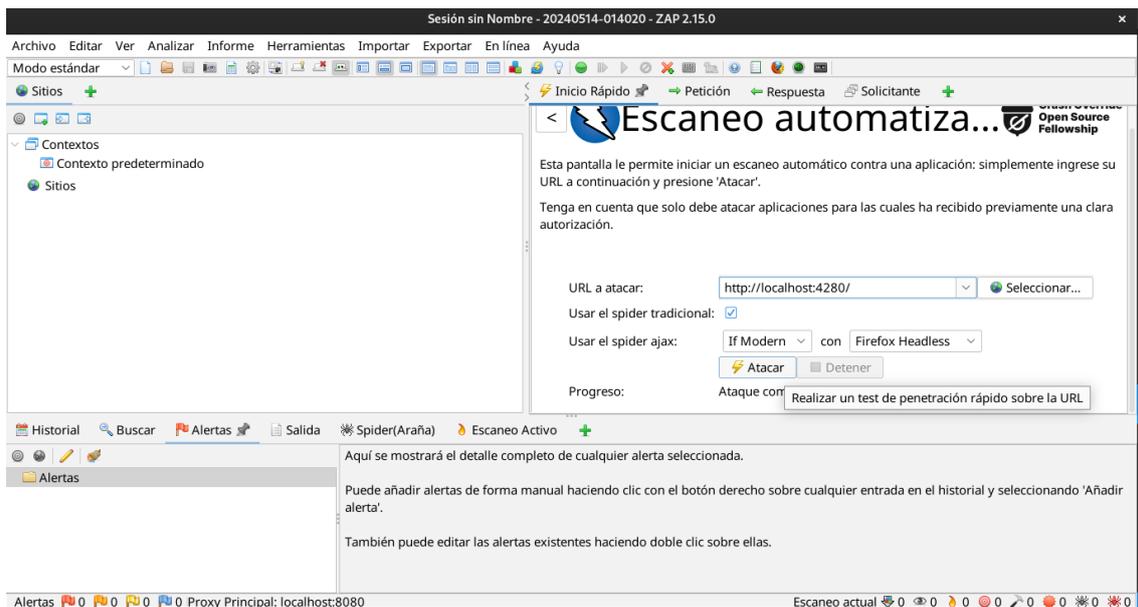
2. Una vez que OWASP ZAP esté abierto, haz clic en el botón Modo Estandar para cambiar al modo estándar.



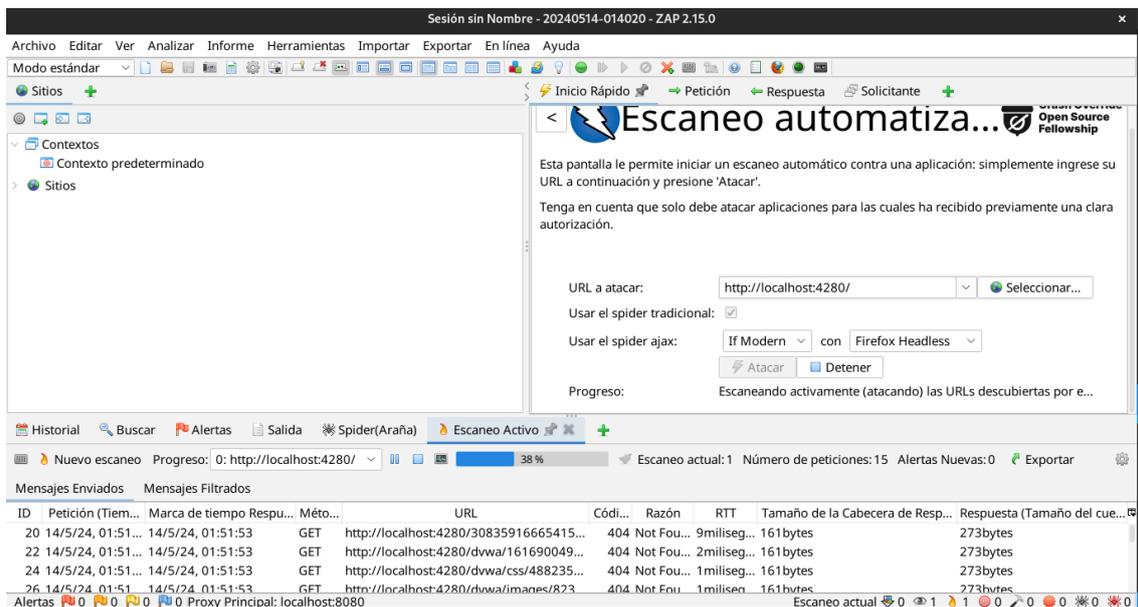
3. Vamos a realizar un Escaneo Automático de la aplicación web de DVWA. Para ello, haz clic en la pestaña Escaneo y selecciona la opción Escaneo Automático.



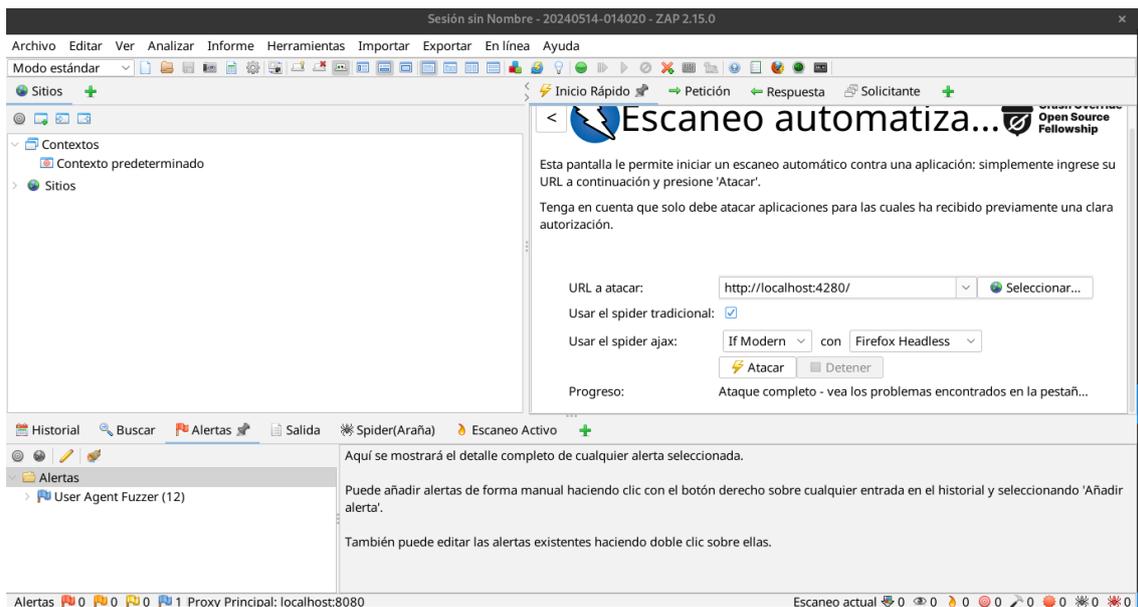
4. En la ventana de configuración del escaneo automático, selecciona el objetivo de escaneo. En este caso, vamos a escanear la URL de la aplicación web de DVWA. En la sección **URL a atacar** introduce la URL de la aplicación web de DVWA: <http://localhost:4280>. Haz clic en el botón **Atacar** para iniciar el escaneo automático.



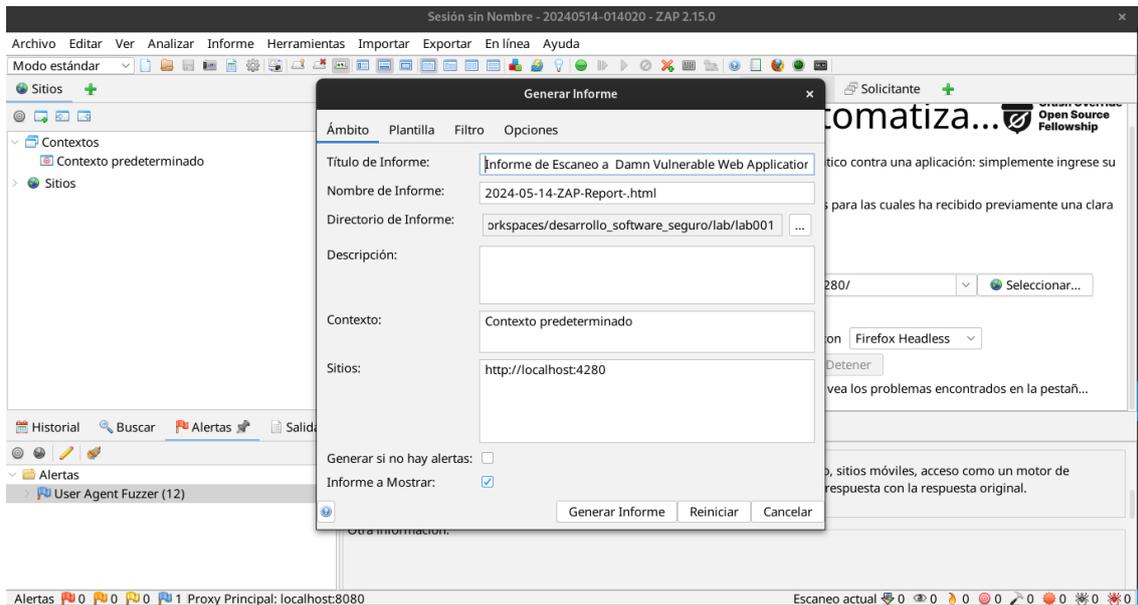
6. Espera un momento mientras se procesa el escaneo automático. Una vez que el escaneo haya finalizado, puedes ver los resultados del escaneo en la pestaña **Alertas**. Aquí puedes ver una lista de las vulnerabilidades encontradas en la aplicación web de DVWA.



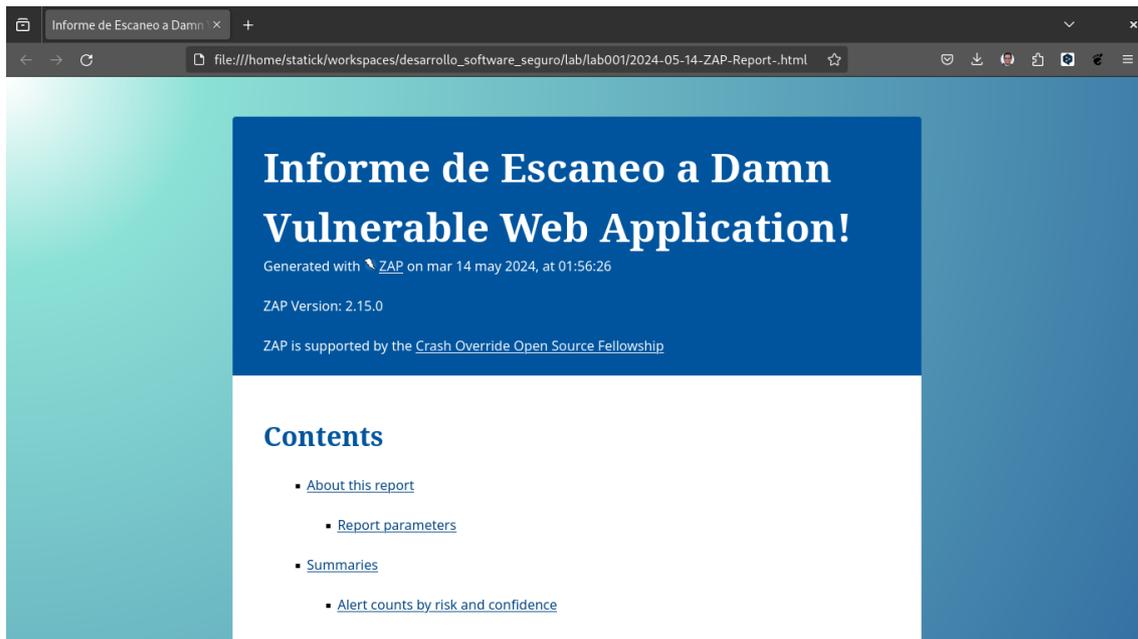
7. Puedes hacer clic en cada alerta para ver más detalles sobre la vulnerabilidad encontrada. Aquí puedes ver información detallada sobre la vulnerabilidad, incluyendo una descripción de la vulnerabilidad, el impacto de la vulnerabilidad y una recomendación sobre cómo solucionar la vulnerabilidad.



8. Una vez terminado el escaneo, puedes exportar los resultados del escaneo en un archivo HTML. Para ello, haz clic en la pestaña **Informe** y selecciona la opción **Generar Informe**. Puedes guardar el archivo HTML en tu máquina para revisarlo más tarde.



9. Finalmente puedes observar los resultados del escaneo en el archivo HTML generado.



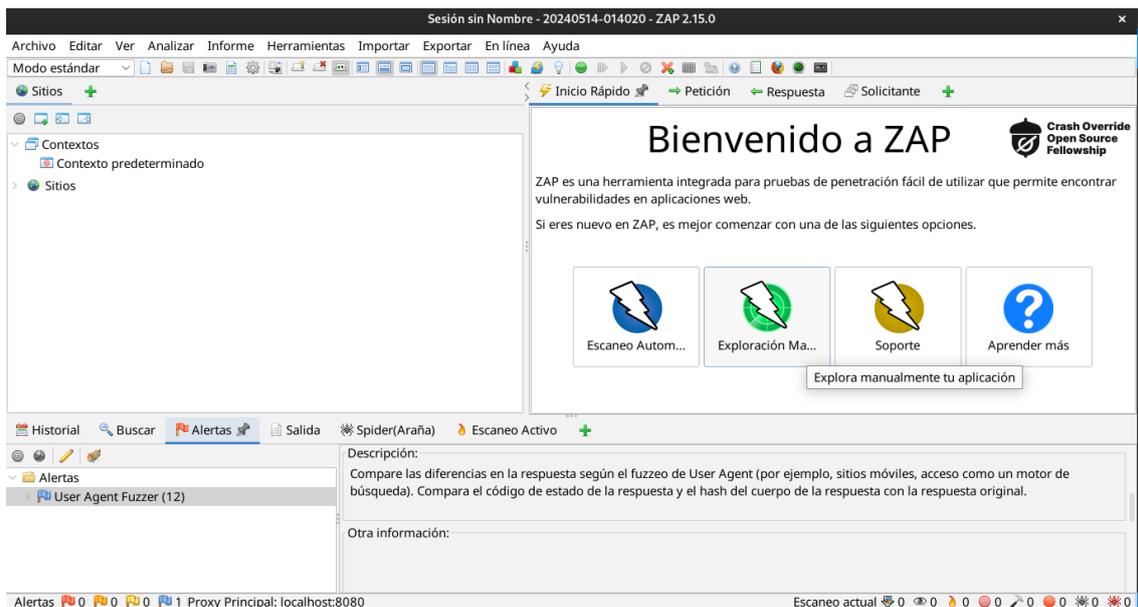
Felicitaciones! Has completado el ejercicio 1.

---

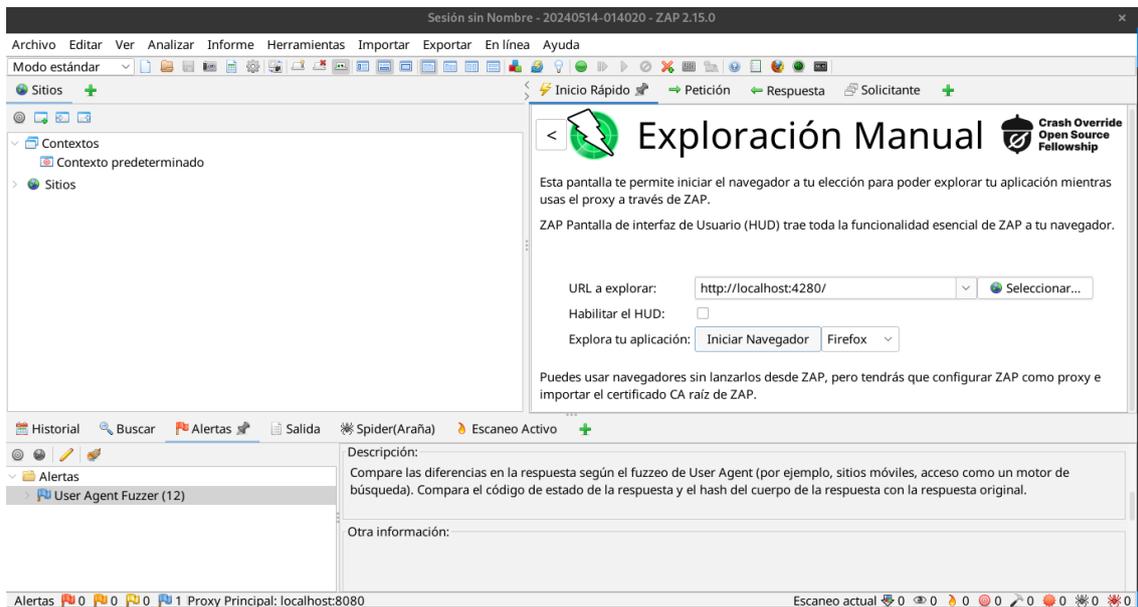
## 2.7 Ejercicio 2: Escaneo Manual de la aplicación web de DVWA

En este ejercicio vamos a realizar un escaneo manual de la aplicación web de DVWA utilizando OWASP ZAP.

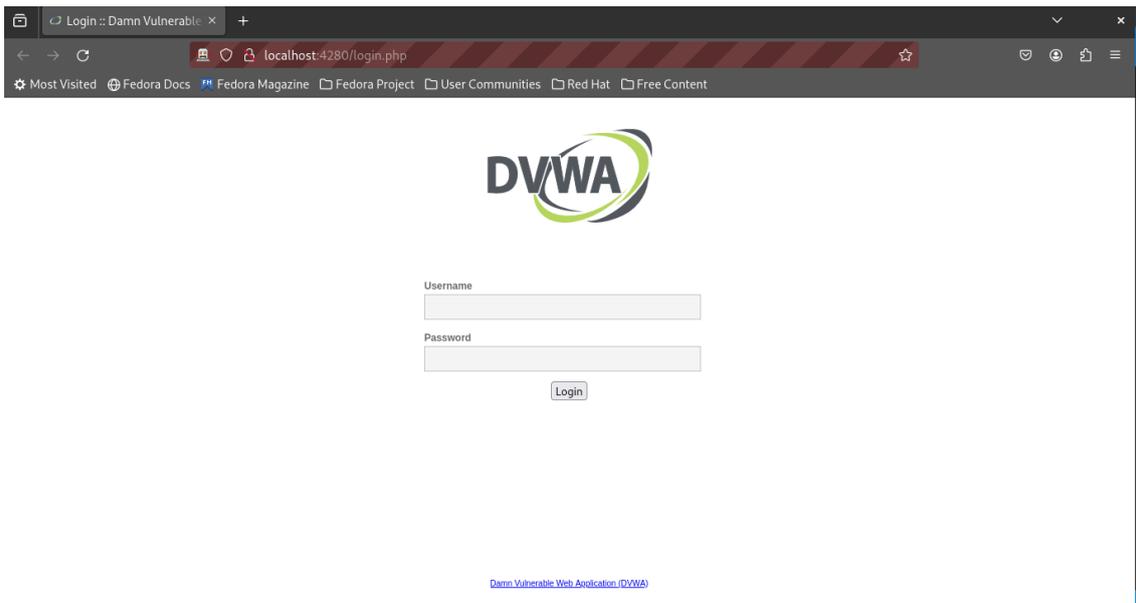
1. Abre OWASP ZAP en tu máquina. Puedes abrir OWASP ZAP
2. Una vez que OWASP ZAP esté abierto, haz clic en el botón Modo Estandar para cambiar al modo estándar.
3. Vamos a realizar un Escaneo Manual de la aplicación web de DVWA. Para ello, haz clic en la pestaña **Atacar** y selecciona la opción **Iniciar Ataque Activo**.



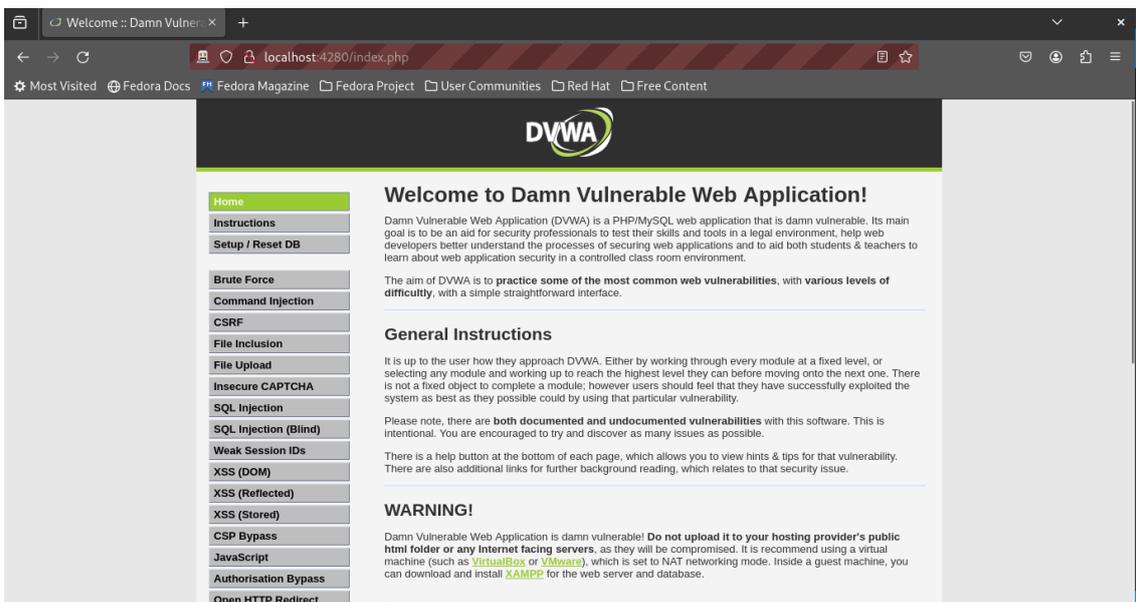
4. En la ventana de configuración del ataque activo, selecciona el objetivo de ataque. En este caso, vamos a atacar la URL de la aplicación web de DVWA. En la sección **URL a atacar** introduce la URL de la aplicación web de DVWA: <http://localhost:4280>. Haz clic en el botón **Iniciar Navegador** para iniciar el ataque activo.



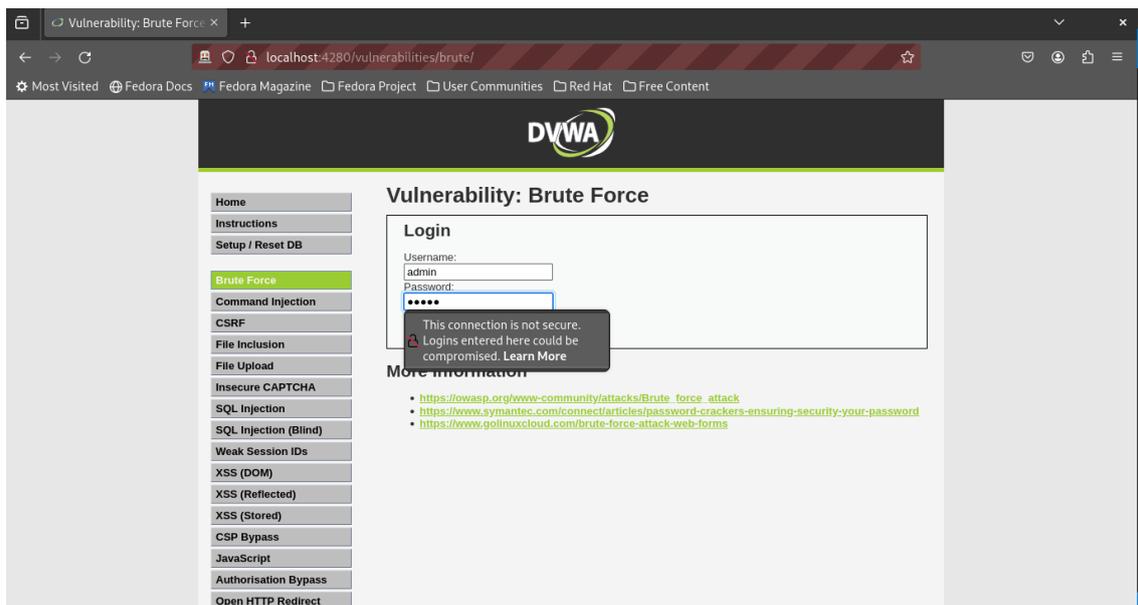
En esta sección podemos observar como se abre el navegador bajo el control de OWASP ZAP.



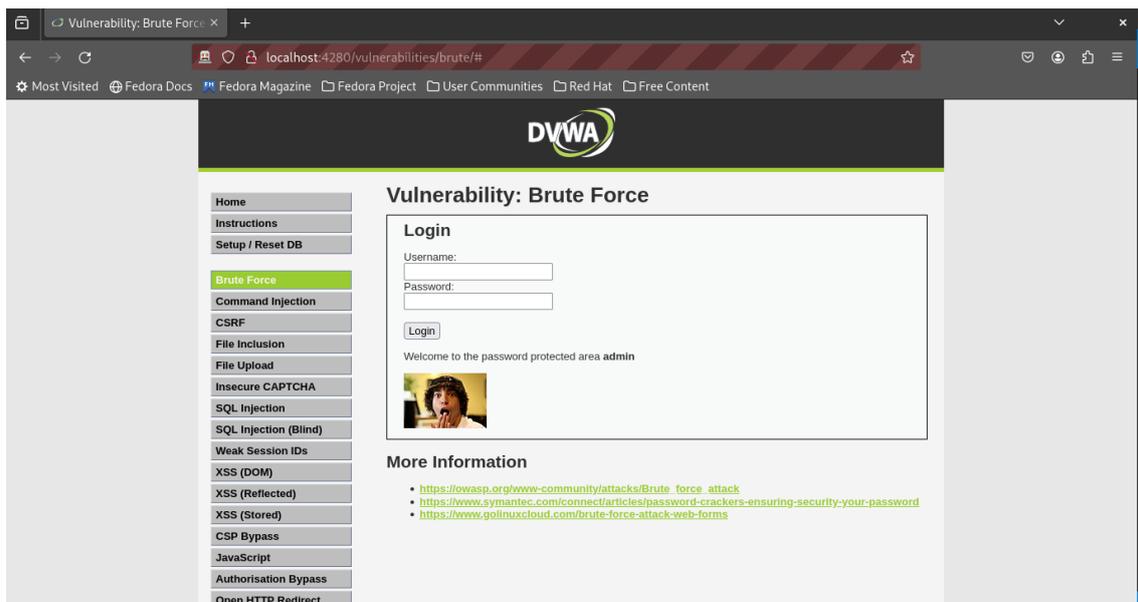
5. Una vez que el navegador esté abierto, puedes navegar por la aplicación web de DVWA y realizar pruebas de seguridad manualmente. Puedes hacer clic en los enlaces de la aplicación web para explorar las diferentes páginas de la aplicación web.



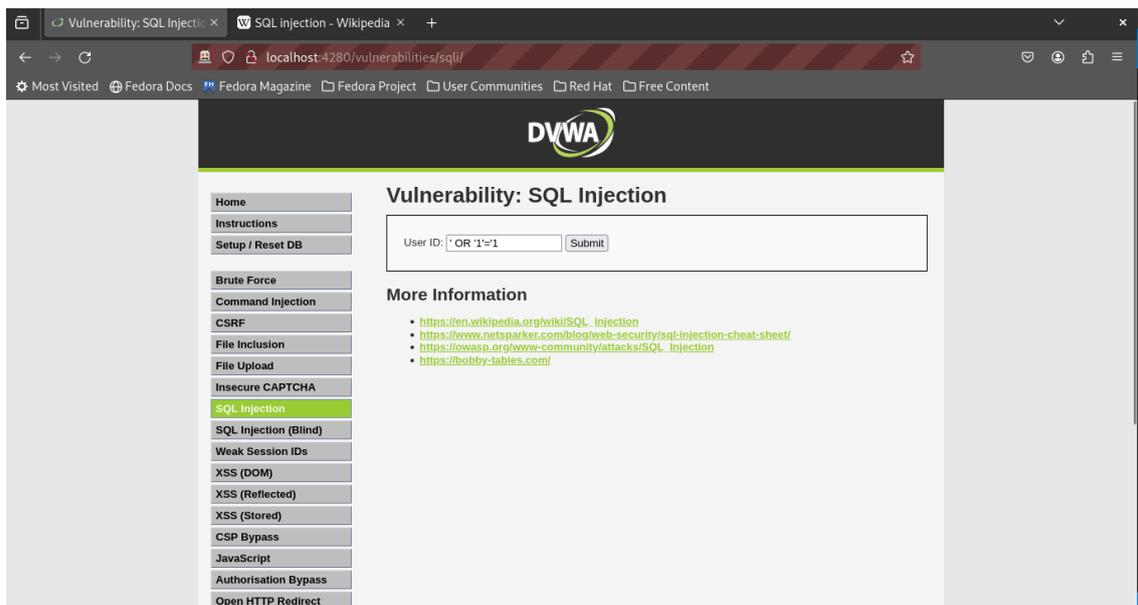
6. Podemos observar los distintos tipos de ataques que podemos realizar en la aplicación web. Para hacer una prueba vamos a empezar por Vulnerability: Brute Force.



Esta sería la vista que podemos observar al utilizar el ataque por Fuerza Bruta



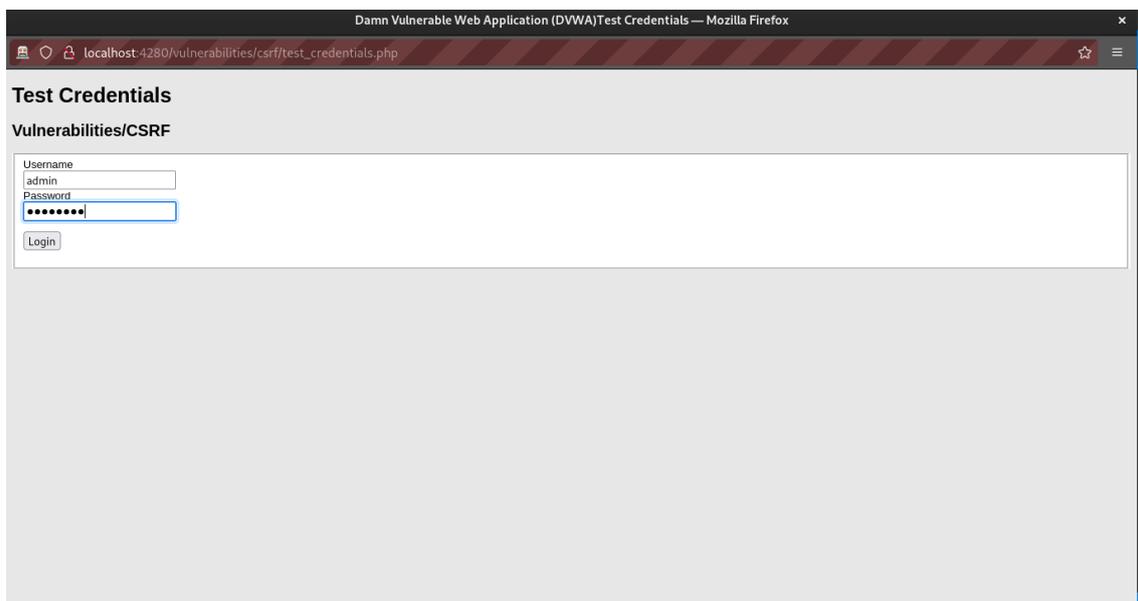
7. Agreguemos algunos ataques más, por ejemplo, el ataque de SQL Injection.



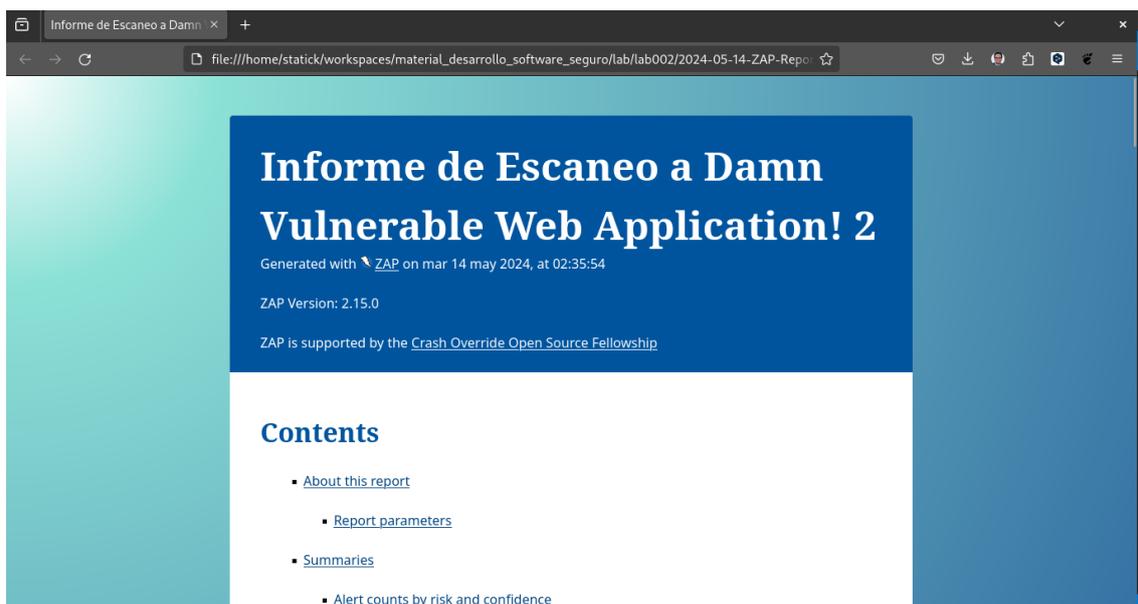
En la imagen anterior podemos observar como se realiza un ataque de SQL Injection. Utilizando la información disponible en los enlaces que proporciona la misma página.

- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com/>

8. Un ataque muy utilizado es CSRF, en la siguiente imagen podemos observar como se realiza un ataque de CSRF.



9. Una vez que hayas realizado las pruebas de seguridad manualmente, puedes ver los resultados de la misma forma que en el ejercicio anterior.



## 2.8 Recursos adicionales

- [Installing DVWA in Kali Linux](#)
- [Installing DVWA in Windows with XAMPP](#)
- [Finding and exploiting reflected XSS in DVWA](#)

## 2.9 Conclusiones

En este laboratorio hemos aprendido a utilizar OWASP ZAP para realizar pruebas de seguridad en aplicaciones web.

Aprendimos a realizar escaneos automáticos y manuales de aplicaciones web, y a identificar vulnerabilidades en aplicaciones web.

OWASP ZAP es una herramienta muy potente que nos permite identificar vulnerabilidades en aplicaciones web y mejorar la seguridad de las mismas.

## 2.10 Recomendaciones

Se recomienda revisar la documentación oficial de **OWASP ZAP** para obtener más información sobre cómo utilizar la herramienta y realizar pruebas de seguridad en aplicaciones web.

## 2.11 Autoevaluación

1. ¿Qué es OWASP ZAP?
2. ¿Para qué se utiliza OWASP ZAP?
3. ¿Cómo se realiza un escaneo automático de una aplicación web con OWASP ZAP?
4. ¿Cómo se realiza un escaneo manual de una aplicación web con OWASP ZAP?
5. ¿Qué tipos de vulnerabilidades se pueden identificar con OWASP ZAP?
6. ¿Qué recomendaciones puedes dar para mejorar la seguridad de una aplicación web?
7. ¿Qué es DVWA y para qué se utiliza?
8. ¿Cómo se puede descargar DVWA?
9. ¿Cómo se puede ejecutar DVWA con Docker?
10. ¿Qué es un ataque de SQL Injection y cómo se puede prevenir?

## 2.12 Referencias

- OWASP ZAP: <https://www.zaproxy.org/>
- Damn Vulnerable Web Application (DVWA): <https://github.com/digininja/DVWA>
- Damn Vulnerable Web Application Docker container: <https://hub.docker.com/r/vulnerables/web-dvwa>

## 3 Estandar de Seguridad CWE

Primero vamos a analizar una serie de conceptos necesarios para conocer lo que es CWE.

### 3.1 ¿Qué es CWE?

**CWE (Common Weakness Enumeration)** es un estándar que proporciona una lista de debilidades de software comunes que pueden ocurrir durante el desarrollo de software. CWE es un proyecto de la comunidad que se enfoca en la identificación y mitigación de estas debilidades, las cuales pueden ser explotadas por atacantes.

### 3.2 ¿Cómo se utiliza?

CWE se utiliza para identificar y mitigar debilidades de software comunes durante el desarrollo de software. Proporciona una lista exhaustiva de debilidades que pueden ocurrir, lo que ayuda a los desarrolladores a tomar medidas preventivas para evitar que estas debilidades sean explotadas por atacantes.

### 3.3 ¿Por qué es importante?

Es importante utilizar CWE porque ayuda a identificar y mitigar debilidades de software comunes que pueden ser explotadas por atacantes. Al tener una lista de debilidades conocidas, los desarrolladores pueden tomar medidas proactivas para fortalecer la seguridad de su software y protegerlo contra posibles ataques.

### 3.4 ¿Cómo se utiliza en la gestión de vulnerabilidades?

CWE se utiliza en la gestión de vulnerabilidades para identificar y mitigar debilidades de software comunes. Al conocer las debilidades específicas que pueden ocurrir, los equipos de seguridad pueden priorizar y abordar estas vulnerabilidades de manera efectiva, reduciendo así el riesgo de ataques exitosos.

### **3.5 ¿Qué es un CWE ID?**

Un CWE ID es un identificador único utilizado para identificar una debilidad de software común en la base de datos de CWE. Cada debilidad de software común tiene un CWE ID asociado, lo que facilita la referencia y el seguimiento de estas debilidades en la comunidad de seguridad.

### **3.6 ¿Qué es una CWE Weakness?**

Una CWE Weakness es una debilidad de software común que puede ocurrir durante el desarrollo de software. Estas debilidades son conocidas y documentadas en la base de datos de CWE, lo que permite a los desarrolladores y equipos de seguridad comprender y abordar estas vulnerabilidades de manera efectiva.

### **3.7 ¿Qué es una CWE Category?**

Una CWE Category es una categoría de debilidades de software comunes que pueden ocurrir durante el desarrollo de software. Estas categorías agrupan debilidades similares y ayudan a los desarrolladores y equipos de seguridad a comprender y abordar las debilidades de manera más eficiente.

### **3.8 ¿Qué es una CWE View?**

Una CWE View es una vista de la base de datos de CWE que proporciona una lista de debilidades de software comunes que pueden ocurrir durante el desarrollo de software. Estas vistas ayudan a los desarrolladores y equipos de seguridad a explorar y comprender las debilidades específicas que pueden afectar a su software.

### **3.9 ¿Qué es una CWE Entry?**

Una CWE Entry es una entrada en la base de datos de CWE que describe una debilidad de software común. Cada entrada de CWE proporciona información detallada sobre la debilidad, incluyendo una descripción, impacto potencial, mitigaciones recomendadas y referencias adicionales.

### **3.10 ¿Qué es una CWE Relationship?**

Una CWE Relationship es una relación entre dos debilidades de software comunes en la base de datos de CWE. Estas relaciones pueden ser de varios tipos, como “ChildOf” o “ParentOf”, y ayudan a los desarrolladores y equipos de seguridad a comprender cómo las debilidades están relacionadas entre sí.

### **3.11 ¿Qué es una CWE Mapping?**

Una CWE Mapping es una asignación de una debilidad de software común a un estándar o marco de seguridad específico. Estas asignaciones ayudan a los desarrolladores y equipos de seguridad a comprender cómo las debilidades de software comunes se relacionan con los estándares y marcos de seguridad existentes.

### **3.12 ¿Qué es una CWE Group?**

Una CWE Group es un grupo de debilidades de software comunes que comparten características similares. Estos grupos ayudan a los desarrolladores y equipos de seguridad a comprender y abordar las debilidades de software comunes de manera más eficiente.

Lo más importante de CWE es que proporciona una lista de debilidades de software comunes que pueden ocurrir durante el desarrollo de software. Al identificar y mitigar estas debilidades, los desarrolladores y equipos de seguridad pueden fortalecer la seguridad de su software y protegerlo contra posibles ataques.

## 4 Top 25 de las debilidades de software más peligrosas.

Antes de conocer el Top 25 de las debilidades de software más peligrosas, es importante tener en cuenta que CWE clasifica las debilidades de software en diferentes categorías y subcategorías.

### 4.1 ¿Cómo se clasifican las debilidades de software en CWE?

CWE clasifica las debilidades de software en diferentes categorías y subcategorías para facilitar su identificación y mitigación. Algunas de las categorías y subcategorías más comunes son:

- **CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer** significa que una aplicación no restringe adecuadamente las operaciones dentro de los límites de un búfer de memoria, lo que puede permitir a un atacante sobrescribir la memoria y ejecutar código malicioso.
- **CWE-200: Information Exposure** significa que una aplicación expone información sensible a un atacante, lo que puede comprometer la seguridad de la aplicación.
- **CWE-400: Uncontrolled Resource Consumption** significa que una aplicación consume recursos de manera incontrolada, lo que puede provocar una denegación de servicio.
- **CWE-600: Failure to Constrain Operations within the Bounds of a Memory Buffer** significa que una aplicación no restringe adecuadamente las operaciones dentro de los límites de un búfer de memoria, lo que puede permitir a un atacante sobrescribir la memoria y ejecutar código malicioso.
- **CWE-800: Control of Execution Context** significa que una aplicación no controla adecuadamente el contexto de ejecución, lo que puede permitir a un atacante modificar el comportamiento de la aplicación.
- **CWE-1000: Behavior** significa que una aplicación tiene un comportamiento inesperado, lo que puede comprometer la seguridad de la aplicación.
- **CWE-1200: Security Features** significa que una aplicación no implementa adecuadamente las características de seguridad, lo que puede permitir a un atacante comprometer la seguridad de la aplicación.
- **CWE-1400: Quality** significa que una aplicación tiene problemas de calidad, lo que puede afectar su seguridad y confiabilidad.

- **CWE-1600: Environment** significa que una aplicación no controla adecuadamente el entorno en el que se ejecuta, lo que puede comprometer su seguridad.
- **CWE-1800: Time and State** significa que una aplicación no controla adecuadamente el tiempo y el estado, lo que puede permitir a un atacante realizar ataques temporales y de estado.
- **CWE-2000: Language and Locale** significa que una aplicación no controla adecuadamente el idioma y la configuración regional, lo que puede comprometer su seguridad.
- **CWE-2200: Configuration** significa que una aplicación no configura adecuadamente los parámetros de configuración, lo que puede permitir a un atacante modificar la configuración y comprometer la seguridad de la aplicación.
- **CWE-2400: Fault Tolerance** significa que una aplicación no implementa adecuadamente la tolerancia a fallos, lo que puede afectar su seguridad y confiabilidad.
- **CWE-2600: Encapsulation** significa que una aplicación no encapsula adecuadamente los componentes, lo que puede comprometer su seguridad.
- **CWE-2800: Cryptography** significa que una aplicación no implementa adecuadamente la criptografía, lo que puede comprometer la seguridad de la aplicación.
- **CWE-3000: Authorization** significa que una aplicación no implementa adecuadamente la autorización, lo que puede permitir a un atacante obtener acceso no autorizado a recursos sensibles.
- **CWE-3200: Authentication** significa que una aplicación no implementa adecuadamente la autenticación, lo que puede permitir a un atacante obtener acceso no autorizado a la aplicación.
- **CWE-3400: Session Management** significa que una aplicación no gestiona adecuadamente las sesiones, lo que puede permitir a un atacante obtener acceso no autorizado a la aplicación.
- **CWE-3600: Access Control** significa que una aplicación no controla adecuadamente el acceso a los recursos, lo que puede permitir a un atacante obtener acceso no autorizado a recursos sensibles.
- **CWE-3800: Input Validation** significa que una aplicación no valida adecuadamente la entrada, lo que puede permitir a un atacante inyectar código malicioso en la aplicación.
- **CWE-4000: Error Handling** significa que una aplicación no maneja adecuadamente los errores, lo que puede comprometer su seguridad y confiabilidad.
- **CWE-4200: Data Handling** significa que una aplicación no maneja adecuadamente los datos, lo que puede comprometer la seguridad de la aplicación.
- **CWE-4400: File Management** significa que una aplicación no gestiona adecuadamente los archivos, lo que puede permitir a un atacante acceder a archivos sensibles en el sistema de archivos.

- **CWE-4600: Memory Management** significa que una aplicación no gestiona adecuadamente la memoria, lo que puede permitir a un atacante sobrescribir la memoria y ejecutar código malicioso.
- **CWE-4800: Network Communication** significa que una aplicación no gestiona adecuadamente la comunicación en red, lo que puede permitir a un atacante interceptar o modificar la comunicación.
- **CWE-5000: Privilege Management** significa que una aplicación no gestiona adecuadamente los privilegios de usuario, lo que puede permitir a un atacante obtener privilegios elevados y realizar acciones maliciosas en el sistema.
- **CWE-5200: Information Leak** significa que una aplicación expone información sensible, lo que puede comprometer la seguridad de la aplicación.
- **CWE-5400: Code Quality** significa que una aplicación tiene problemas de calidad de código, lo que puede afectar su seguridad y confiabilidad.
- **CWE-5600: API Abuse** significa que una aplicación abusa de una API, lo que puede comprometer la seguridad de la aplicación.
- **CWE-5800: Link Following** significa que una aplicación sigue enlaces de manera insegura, lo que puede permitir a un atacante acceder a recursos sensibles en la aplicación.
- **CWE-6000: Trust Management** significa que una aplicación no gestiona adecuadamente la confianza, lo que puede permitir a un atacante comprometer la seguridad de la aplicación.
- **CWE-6200: Cross-Site Scripting** significa que una aplicación no neutraliza adecuadamente la entrada durante la generación de una página web, lo que puede permitir a un atacante ejecutar scripts maliciosos en el navegador del usuario.
- **CWE-6400: SQL Injection** significa que una aplicación no neutraliza adecuadamente elementos especiales utilizados en un comando SQL, lo que puede permitir a un atacante ejecutar comandos SQL maliciosos en la base de datos.
- **CWE-6600: Path Traversal** significa que una aplicación no limita adecuadamente una ruta de acceso a un directorio restringido, lo que puede permitir a un atacante acceder a archivos o directorios sensibles en el sistema de archivos.
- **CWE-6800: Command Injection** significa que una aplicación no neutraliza adecuadamente elementos especiales utilizados en un comando del sistema operativo, lo que puede permitir a un atacante ejecutar comandos maliciosos en el sistema.
- **CWE-7000: Cross-Site Request Forgery** significa que una aplicación no protege adecuadamente contra ataques de falsificación de solicitudes entre sitios, lo que puede permitir a un atacante realizar acciones maliciosas en nombre del usuario.
- **CWE-7200: Unrestricted Upload of File with Dangerous Type** significa que una aplicación permite la carga de archivos sin restricciones, lo que puede permitir a un atacante cargar archivos maliciosos en el sistema y comprometer la seguridad de la aplicación.

- **CWE-7400: Open Redirect** significa que una aplicación redirige a un usuario a un sitio no confiable, lo que puede permitir a un atacante redirigir a un usuario a un sitio malicioso y realizar ataques de phishing.
- **CWE-7600: Use of Hard-coded Credentials** significa que una aplicación utiliza credenciales codificadas en el código fuente, lo que puede permitir a un atacante obtener acceso no autorizado a la aplicación y comprometer la seguridad de la misma.
- **CWE-7800: Reliance on Untrusted Inputs in a Security Decision** significa que una aplicación confía en entradas no confiables para tomar decisiones de seguridad, lo que puede permitir a un atacante manipular las entradas y comprometer la seguridad de la aplicación.
- **CWE-8000: Missing Authorization** significa que una aplicación no implementa adecuadamente la autorización, lo que puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Estas categorías y subcategorías ayudan a los desarrolladores y equipos de seguridad a identificar y mitigar debilidades de software comunes de manera efectiva, reduciendo así el riesgo de ataques exitosos.

## 5 ¿Cómo se analiza la información de CWE?

Cada una de las vulnerabilidades registradas en CWE tiene una serie de atributos que permiten analizar y comprender la debilidad de software. Algunos de los atributos más importantes son:

- **Descripción:** Describe la debilidad de software y cómo puede ser explotada por un atacante.
- **Impacto potencial:** Describe el impacto que la debilidad puede tener en la seguridad de la aplicación.
- **Mitigaciones recomendadas:** Proporciona recomendaciones para mitigar la debilidad y fortalecer la seguridad de la aplicación.
- **Referencias adicionales:** Proporciona enlaces a recursos adicionales que pueden ser útiles para comprender y abordar la debilidad.

Al analizar la información de CWE, los desarrolladores y equipos de seguridad pueden identificar y mitigar debilidades de software comunes de manera efectiva, reduciendo así el riesgo de ataques exitosos.

Es importante tener en cuenta que CWE es un estándar dinámico que se actualiza periódicamente para reflejar las nuevas debilidades de software y las mejores prácticas de seguridad. Por lo tanto, es fundamental mantenerse actualizado con las últimas versiones de CWE y aplicar las recomendaciones de seguridad correspondientes.

A continuación se presenta el Top 25 de las debilidades de software más peligrosas según CWE.

### 5.1 1. CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

CWE-22 es una debilidad de software común que ocurre cuando una aplicación no limita adecuadamente una ruta de acceso a un directorio restringido. Esto puede permitir a un atacante acceder a archivos o directorios sensibles en el sistema de archivos.

Puede consultar más información en el siguiente enlace: [CWE-22](#)

## **5.2 2. CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**

CWE-78 es una debilidad de software común que ocurre cuando una aplicación no neutraliza adecuadamente elementos especiales utilizados en un comando del sistema operativo. Esto puede permitir a un atacante ejecutar comandos maliciosos en el sistema.

Puede consultar más información en el siguiente enlace: [CWE-78](#)

## **5.3 3. CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**

CWE-79 es una debilidad de software común que ocurre cuando una aplicación no neutraliza adecuadamente la entrada durante la generación de una página web. Esto puede permitir a un atacante ejecutar scripts maliciosos en el navegador del usuario.

Puede consultar más información en el siguiente enlace: [CWE-79](#)

## **5.4 4. CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**

CWE-89 es una debilidad de software común que ocurre cuando una aplicación no neutraliza adecuadamente elementos especiales utilizados en un comando SQL. Esto puede permitir a un atacante ejecutar comandos SQL maliciosos en la base de datos.

Puede consultar más información en el siguiente enlace: [CWE-89](#)

## **5.5 5. CWE-94: Improper Control of Generation of Code ('Code Injection')**

CWE-94 es una debilidad de software común que ocurre cuando una aplicación no controla adecuadamente la generación de código. Esto puede permitir a un atacante ejecutar código malicioso en el sistema.

Puede consultar más información en el siguiente enlace: [CWE-94](#)

## **5.6 6. CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')**

CWE-120 es una debilidad de software común que ocurre cuando una aplicación copia un búfer sin verificar el tamaño de la entrada. Esto puede permitir a un atacante sobrescribir la memoria y ejecutar código malicioso en el sistema.

Puede consultar más información en el siguiente enlace: [CWE-120](#)

## **5.7 7. CWE-190: Integer Overflow or Wraparound**

CWE-190 es una debilidad de software común que ocurre cuando una aplicación realiza una operación aritmética que resulta en un desbordamiento de entero. Esto puede permitir a un atacante modificar el comportamiento de la aplicación y causar un comportamiento inesperado.

Puede consultar más información en el siguiente enlace: [CWE-190](#)

## **5.8 8. CWE-200: Information Exposure**

CWE-200 es una debilidad de software común que ocurre cuando una aplicación expone información sensible a un atacante. Esto puede permitir a un atacante obtener acceso a información confidencial y comprometer la seguridad de la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-200](#)

## **5.9 9. CWE-264: Permissions, Privileges, and Access Controls**

CWE-264 es una debilidad de software común que ocurre cuando una aplicación no implementa adecuadamente permisos, privilegios y controles de acceso. Esto puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-264](#)

## **5.10 10. CWE-269: Improper Privilege Management**

CWE-269 es una debilidad de software común que ocurre cuando una aplicación no gestiona adecuadamente los privilegios de usuario. Esto puede permitir a un atacante obtener privilegios elevados y realizar acciones maliciosas en el sistema.

Puede consultar más información en el siguiente enlace: [CWE-269](#)

## **5.11 11. CWE-352: Cross-Site Request Forgery (CSRF)**

CWE-352 es una debilidad de software común que ocurre cuando una aplicación no protege adecuadamente contra ataques de falsificación de solicitudes entre sitios. Esto puede permitir a un atacante realizar acciones maliciosas en nombre del usuario.

Puede consultar más información en el siguiente enlace: [CWE-352](#)

## **5.12 12. CWE-434: Unrestricted Upload of File with Dangerous Type**

CWE-434 es una debilidad de software común que ocurre cuando una aplicación permite la carga de archivos sin restricciones. Esto puede permitir a un atacante cargar archivos maliciosos en el sistema y comprometer la seguridad de la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-434](#)

## **5.13 13. CWE-601: URL Redirection to Untrusted Site ('Open Redirect')**

CWE-601 es una debilidad de software común que ocurre cuando una aplicación redirige a un usuario a un sitio no confiable. Esto puede permitir a un atacante redirigir a un usuario a un sitio malicioso y realizar ataques de phishing.

Puede consultar más información en el siguiente enlace: [CWE-601](#)

## **5.14 14. CWE-732: Incorrect Permission Assignment for Critical Resource**

CWE-732 es una debilidad de software común que ocurre cuando una aplicación asigna permisos incorrectos a un recurso crítico. Esto puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-732](#)

## **5.15 15. CWE-798: Use of Hard-coded Credentials**

CWE-798 es una debilidad de software común que ocurre cuando una aplicación utiliza credenciales codificadas en el código fuente. Esto puede permitir a un atacante obtener acceso no autorizado a la aplicación y comprometer la seguridad de la misma.

Puede consultar más información en el siguiente enlace: [CWE-798](#)

## **5.16 16. CWE-807: Reliance on Untrusted Inputs in a Security Decision**

CWE-807 es una debilidad de software común que ocurre cuando una aplicación confía en entradas no confiables para tomar decisiones de seguridad. Esto puede permitir a un atacante manipular las entradas y comprometer la seguridad de la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-807](#)

## 5.17 17. CWE-862: Missing Authorization

CWE-862 es una debilidad de software común que ocurre cuando una aplicación no implementa adecuadamente la autorización. Esto puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-862](#)

## 5.18 18. CWE-863: Incorrect Authorization

CWE-863 es una debilidad de software común que ocurre cuando una aplicación implementa incorrectamente la autorización. Esto puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-863](#)

## 5.19 19. CWE-918: Server-Side Request Forgery (SSRF)

CWE-918 es una debilidad de software común que ocurre cuando una aplicación permite a un atacante forzar al servidor a realizar solicitudes no autorizadas. Esto puede permitir a un atacante obtener acceso a recursos sensibles en la red interna.

Puede consultar más información en el siguiente enlace: [CWE-918](#)

## 5.20 20. CWE-943: Improper Neutralization of Special Elements in Data Query Logic ('Injection')

CWE-943 es una debilidad de software común que ocurre cuando una aplicación no neutraliza adecuadamente elementos especiales en la lógica de consulta de datos. Esto puede permitir a un atacante ejecutar comandos maliciosos en la base de datos.

Puede consultar más información en el siguiente enlace: [CWE-943](#)

## 5.21 21. CWE-959: Use of Password System for Primary Authentication

CWE-959 es una debilidad de software común que ocurre cuando una aplicación utiliza un sistema de contraseñas para la autenticación principal. Esto puede permitir a un atacante obtener acceso no autorizado a la aplicación y comprometer la seguridad de la misma.

Puede consultar más información en el siguiente enlace: [CWE-959](#)

## 5.22 22. CWE-1168: Inadequate Encryption Strength

CWE-1168 es una debilidad de software común que ocurre cuando una aplicación utiliza cifrado débil para proteger datos sensibles. Esto puede permitir a un atacante descifrar los datos y comprometer la seguridad de la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-1168](#)

## 5.23 23. CWE-1170: Improper Restriction of XML External Entity Reference ('XXE')

CWE-1170 es una debilidad de software común que ocurre cuando una aplicación no restringe adecuadamente las referencias a entidades XML externas. Esto puede permitir a un atacante leer archivos sensibles en el sistema.

Puede consultar más información en el siguiente enlace: [CWE-1170](#)

## 5.24 24. CWE-1177: Improper Output Neutralization for Logs

CWE-1177 es una debilidad de software común que ocurre cuando una aplicación no neutraliza adecuadamente la salida en los registros. Esto puede permitir a un atacante inyectar código malicioso en los registros y comprometer la seguridad de la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-1177](#)

## 5.25 25. CWE-1189: Improper Isolation of Shared Resources on System

CWE-1189 es una debilidad de software común que ocurre cuando una aplicación no aísla adecuadamente los recursos compartidos en el sistema. Esto puede permitir a un atacante obtener acceso no autorizado a recursos sensibles en la aplicación.

Puede consultar más información en el siguiente enlace: [CWE-1189](#)

## 5.26 Conclusión

CWE es un estándar importante que proporciona una lista de debilidades de software comunes que pueden ocurrir durante el desarrollo de software. Al utilizar CWE, los desarrolladores y equipos de seguridad pueden identificar y mitigar estas debilidades de manera efectiva, reduciendo así el riesgo de ataques exitosos. Es fundamental comprender y abordar las debilidades de software comunes para fortalecer la seguridad de las aplicaciones y protegerlas contra posibles amenazas.

## 5.27 Referencias

- [CWE - Common Weakness Enumeration](#)
- [CWE - Wikipedia](#)
- [CWE - Top 25 Most Dangerous Software Weaknesses](#)

# 6 Laboratorio de estandar de Vulnerabilidades con CWE

## 6.1 1. Objetivos

- Conocer el estándar de vulnerabilidades CWE.
- Identificar vulnerabilidades en un código fuente.
- Conocer la relación entre CWE y CVE.

## 6.2 2. Desarrollo

### 6.2.1 CWE

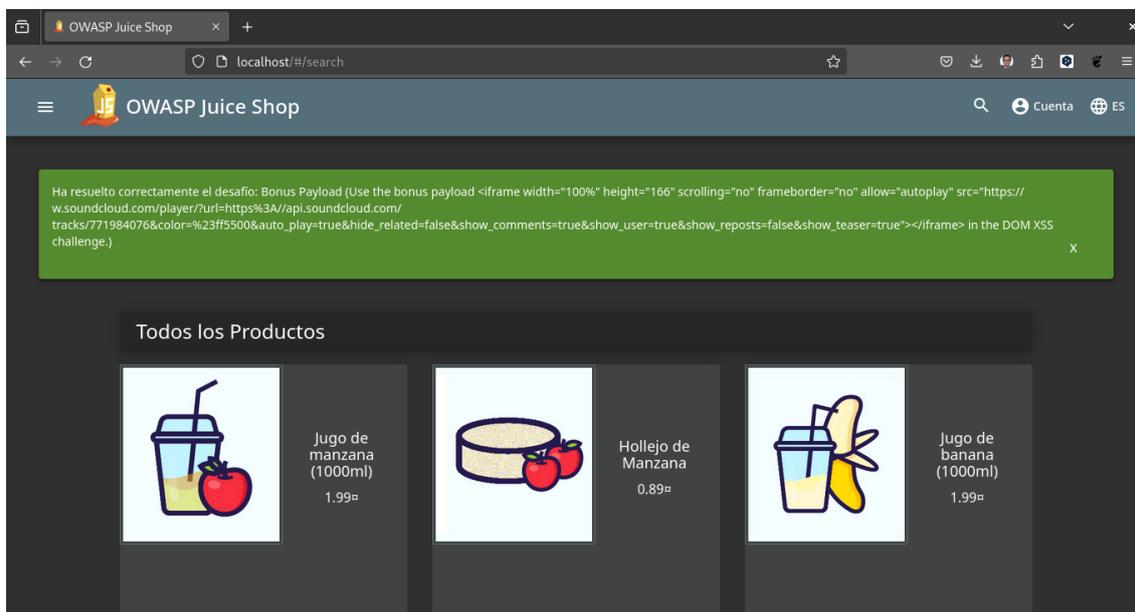
Significa *Common Weakness Enumeration* y es un estándar que se utiliza para identificar y clasificar vulnerabilidades en software. Es mantenido por el MITRE Corporation y se utiliza para identificar debilidades en el software.

### 6.2.2 CVE

Significa *Common Vulnerabilities and Exposures* y es un estándar que se utiliza para identificar y clasificar vulnerabilidades en software. Es mantenido por el MITRE Corporation y se utiliza para identificar debilidades en el software.

Para entender este laboratorio vamos a utilizar la aplicación web vulnerable OWASP Juice Shop. Esta aplicación tiene varias vulnerabilidades que podemos explotar para obtener información confidencial.

## 6.3 OWASP Juice Shop.



OWASP Juice Shop es una aplicación web vulnerable que se utiliza para enseñar a los desarrolladores cómo identificar y explotar vulnerabilidades en el software. La aplicación tiene varias vulnerabilidades que podemos explotar para obtener información confidencial.

Para poder utilizarla es necesario tener instalado Docker en nuestra máquina, una vez instalado procedemos a clonar el repositorio de la aplicación.

```
git clone https://github.com/juice-shop/juice-shop.git
```

Una vez clonado el repositorio procedemos a ejecutar el contenedor de Docker.

```
docker run --rm -p 3000:3000 bkimminich/juice-shop
```

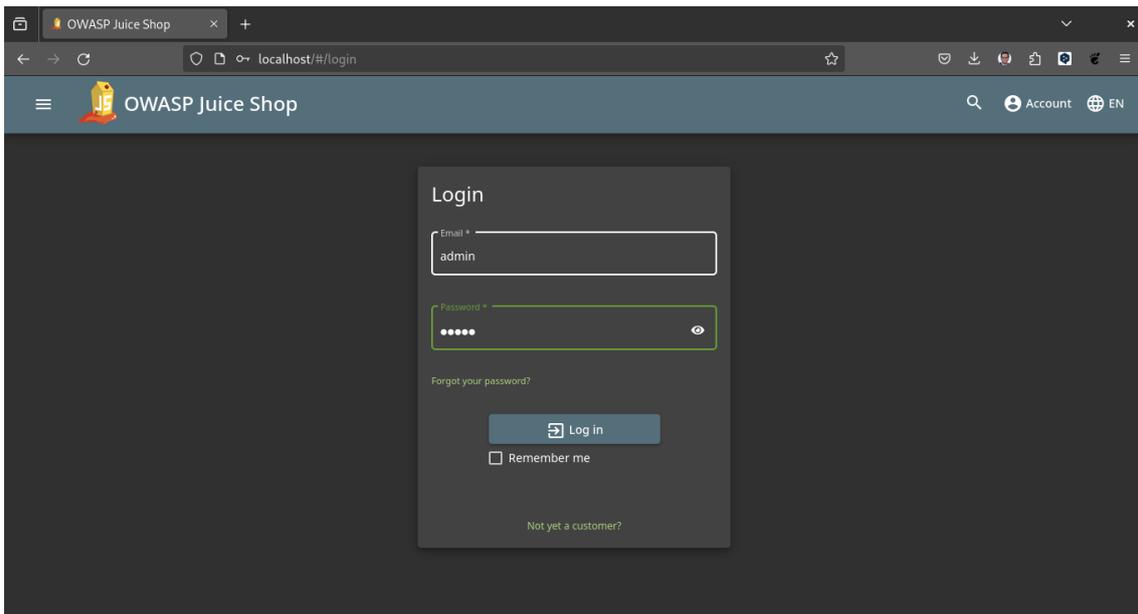
Una vez ejecutado el contenedor podemos acceder a la aplicación en la siguiente dirección: <http://localhost:3000>.

Para analizar que realmente es una aplicación vulnerable podemos utilizar la herramienta OWASP ZAP. Esta herramienta nos permite identificar vulnerabilidades en aplicaciones web, sin embargo vamos a hacer una prueba más práctica.

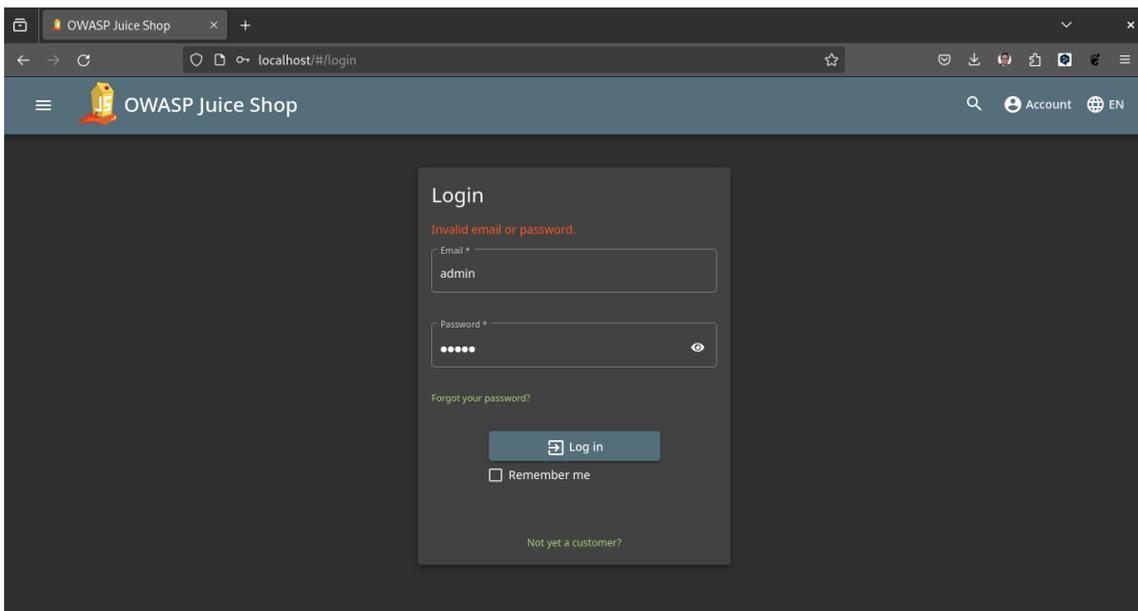
### ¿Cual es la vulnerabilidad más sencilla de todas las que ha escuchado o leído?

La vulnerabilidad más sencilla es la CWE-79, que es una vulnerabilidad que se produce cuando un atacante puede inyectar código en una aplicación web. Esto puede permitir al atacante ejecutar código en el servidor y obtener información confidencial.

Para explotar esta vulnerabilidad lo primero será tratar de loguearnos en la aplicación con un usuario y contraseña incorrectos. Una vez que lo hagamos veremos que la aplicación nos muestra un mensaje de error.

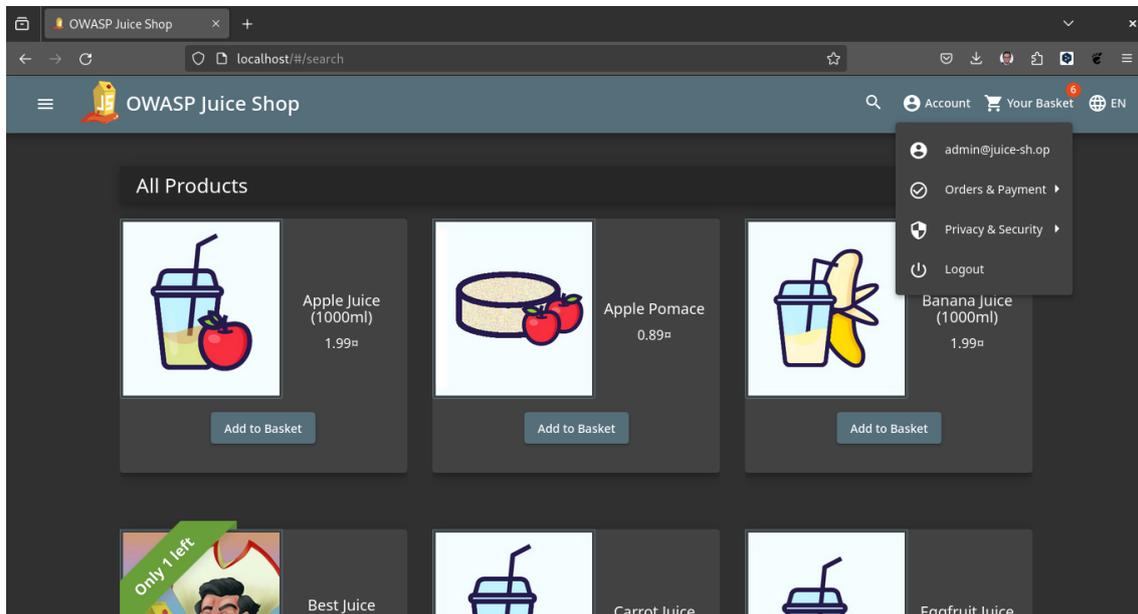


Nuestro primer intento fue infructuoso.



Sin embargo no es motivo para desistir, vamos a intentar inyectar código en la aplicación. Para hacer esto vamos a utilizar el siguiente código:

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



Hemos conseguido entrar en la aplicación utilizando una vulnerabilidad de inyección de código.

## 6.4 Ejercicio

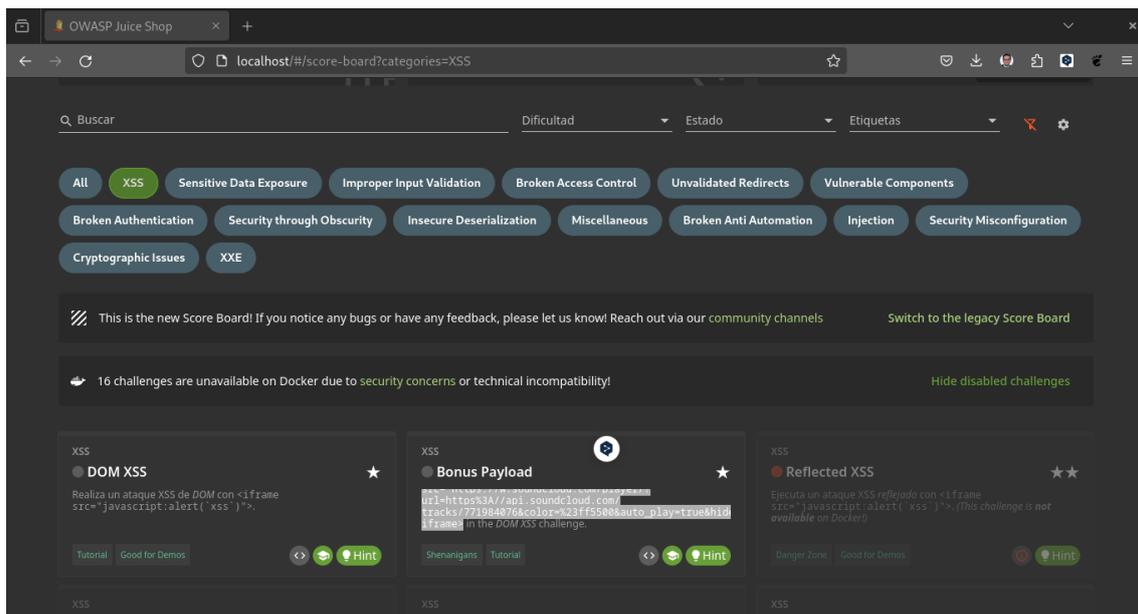
1. Identificar una vulnerabilidad en la aplicación OWASP Juice Shop.
2. Explotar la vulnerabilidad y obtener información confidencial.

## 6.5 Solución

Ver solución

1. Identificar una vulnerabilidad en la aplicación OWASP Juice Shop.

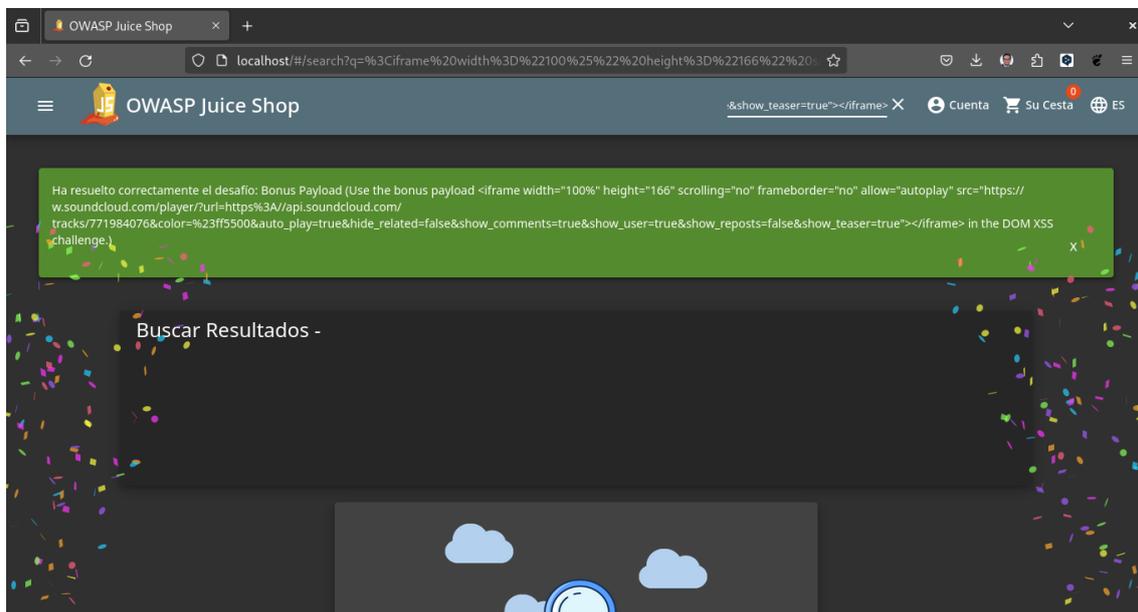
La vulnerabilidad que vamos a explotar es XSS para ello vamos a utilizar la opción en la url agregar score-board y seleccionamos la opción XSS.



Luego vamos copiar el siguiente código:

```
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A%2F%2Fapi.soundcloud.com%2Ftracks%2F771984076&color=%23ff5000&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>
```

Lo copiamos y lo llevamos a la barra de búsqueda y lo pegamos.



## 6.6 Conclusión

El estándar CWE es una herramienta muy útil para identificar y clasificar vulnerabilidades en el software. Nos permite identificar debilidades en el software y nos ayuda a proteger nuestra información confidencial. En este laboratorio hemos aprendido a identificar y explotar dos vulnerabilidades en la aplicación OWASP Juice Shop.

## 6.7 Motivación

El estándar CWE es una herramienta muy útil para identificar y clasificar vulnerabilidades en el software. Nos permite identificar debilidades en el software y nos ayuda a proteger nuestra información confidencial. En este laboratorio hemos aprendido a identificar y explotar dos vulnerabilidades en la aplicación OWASP Juice Shop.

Analiza la aplicación OWASP Juice Shop y trata de identificar y explotar una vulnerabilidad en ella.

### 6.7.1 Herramientas

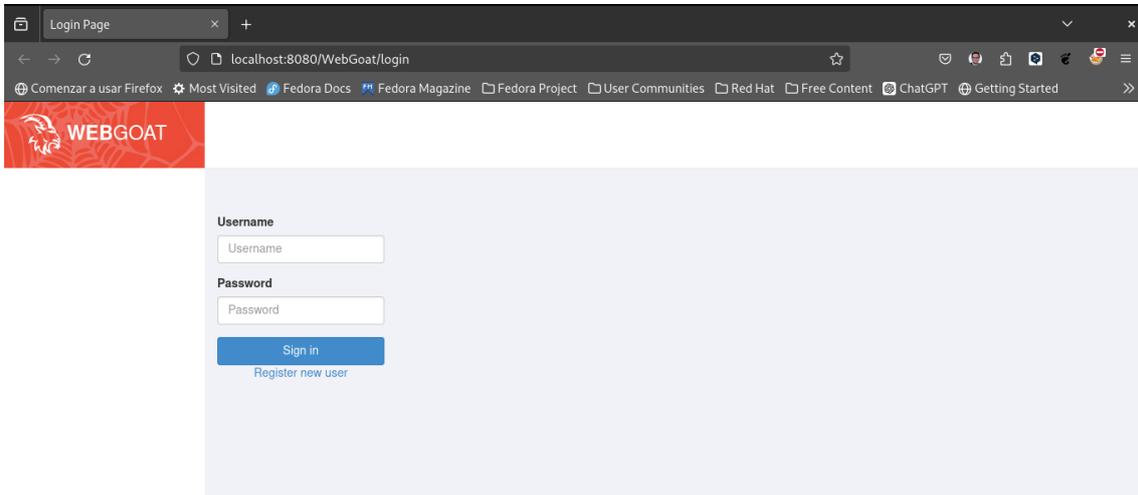
- Docker
- OWASP Juice Shop

### 6.7.2 Referencias

- [OWASP Juice Shop](#)
- [CWE](#)

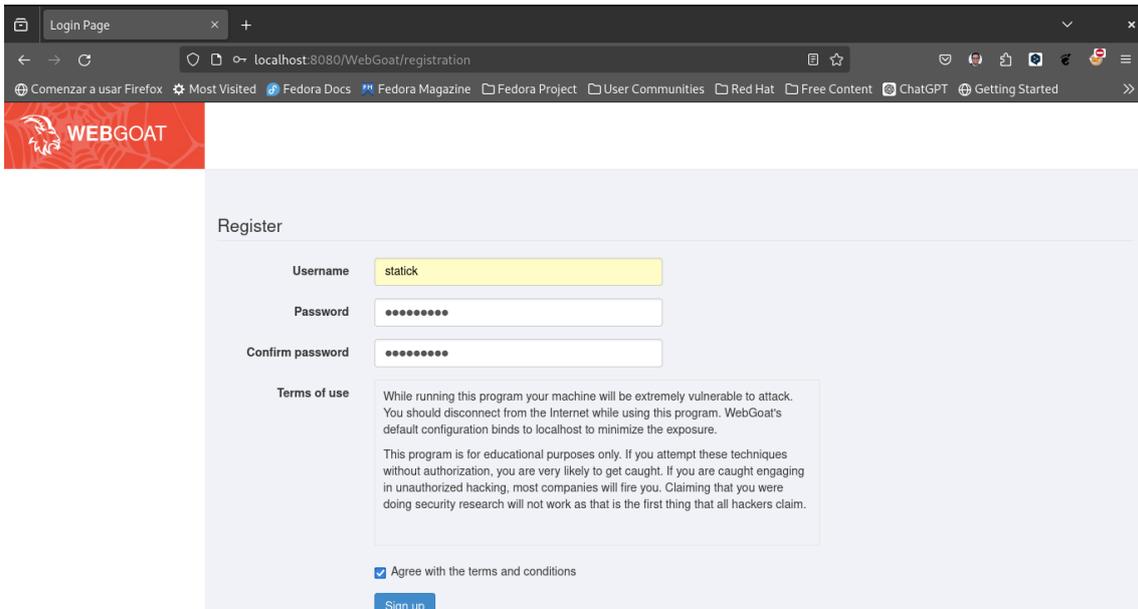


Una vez ejecutado el comando, abrimos un navegador y accedemos a la dirección <http://localhost:8080/WebGoat>.

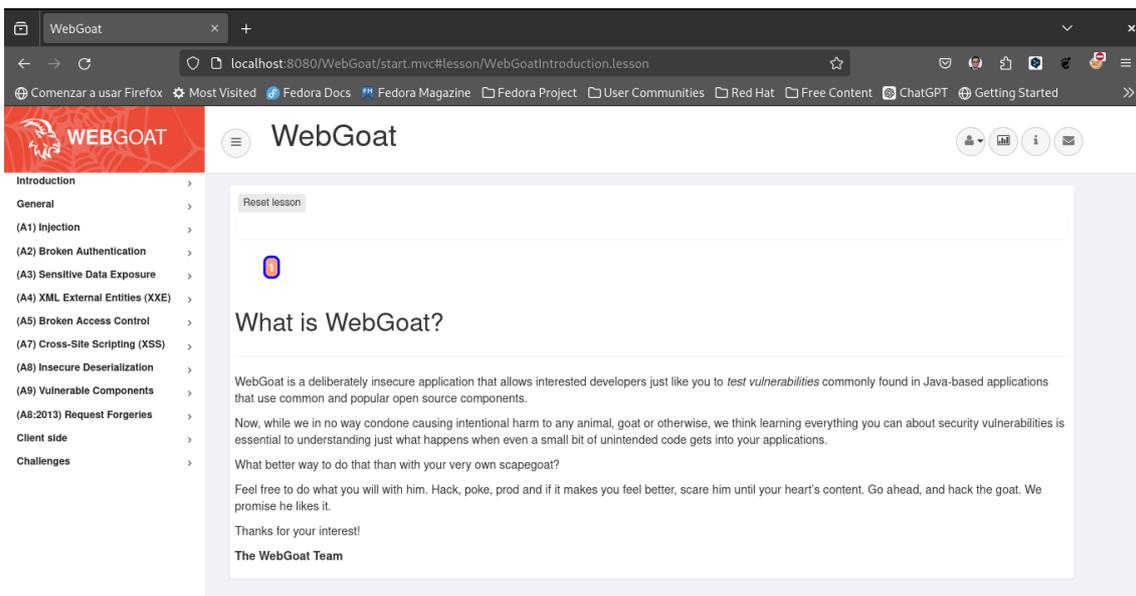


## 7.3 Laboratorio

Una vez que hemos accedido a la aplicación, veremos una pantalla de inicio en la que se nos pide que introduzcamos un usuario y una contraseña. En este caso, es necesario registrarse para poder acceder a la aplicación.

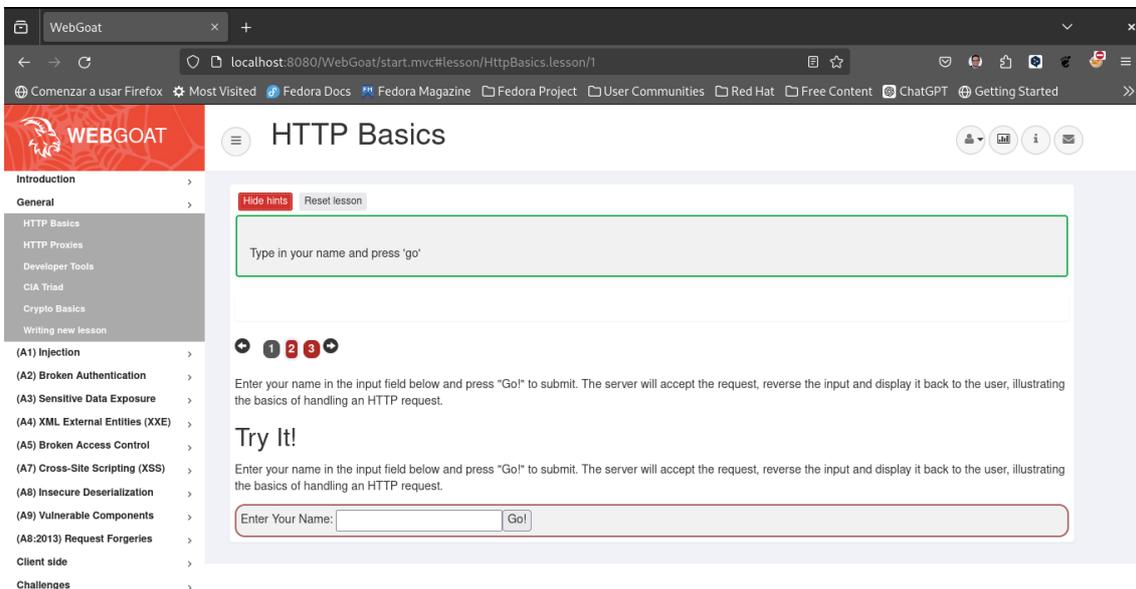


Ahora es posible acceder a la aplicación con el usuario y la contraseña que hemos introducido.

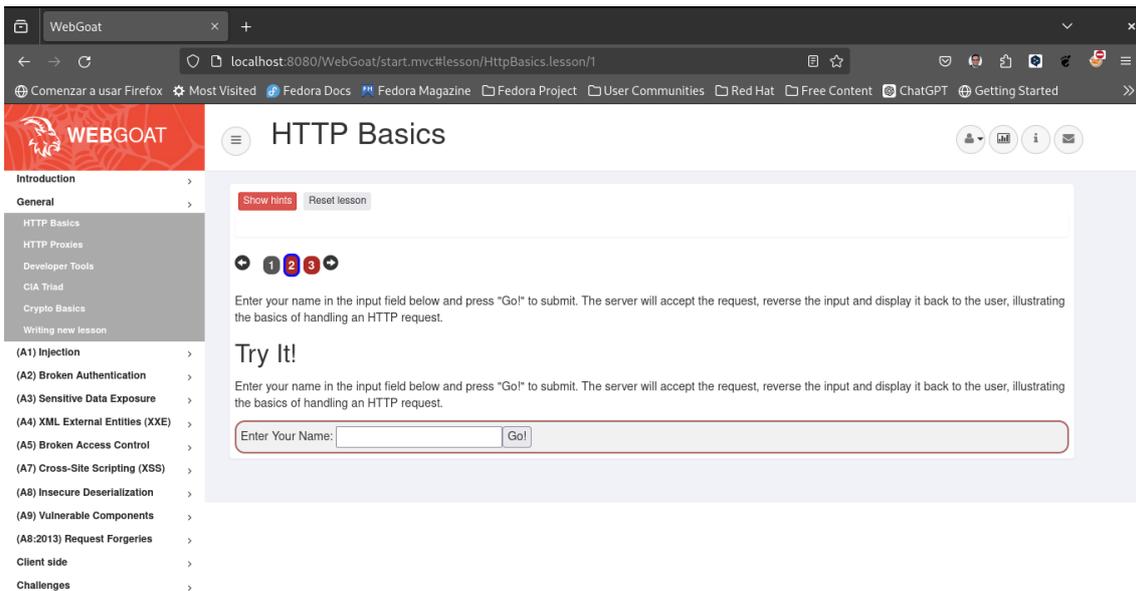


Una vez dentro de la aplicación, veremos una serie de lecciones que podemos realizar para aprender sobre seguridad en aplicaciones web.

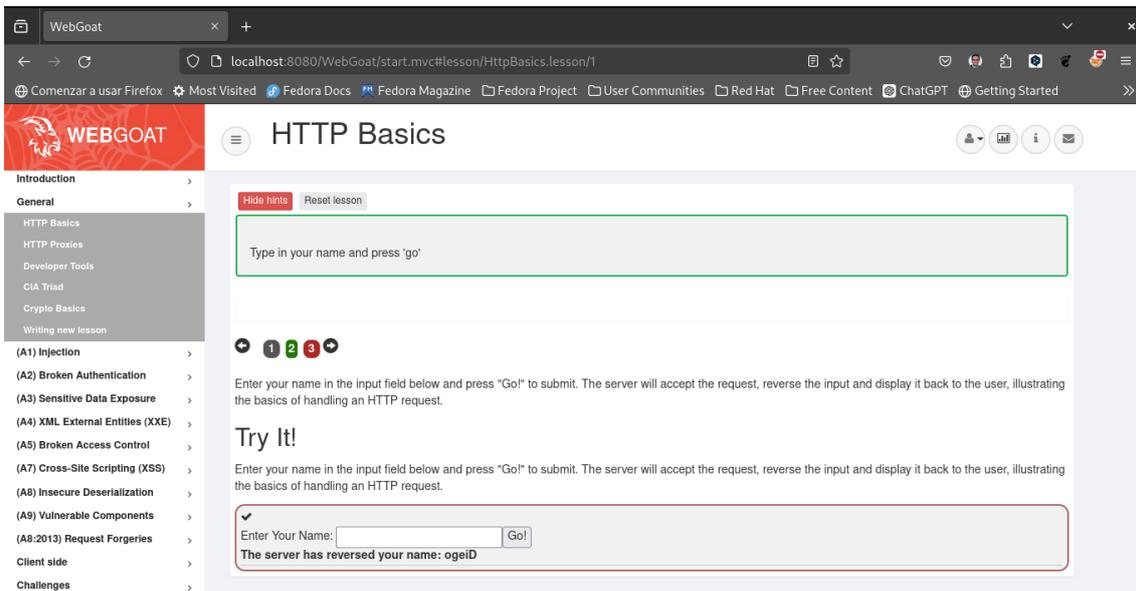
Por ejemplo, en la sección general podemos ver una lección llamada “HTTP Basics” donde se nos explica cómo funciona HTTP y cómo podemos interceptar las peticiones y respuestas con un proxy HTTP. en la misma vamos a encontrar la siguiente estructura:



En la parte superior de la pantalla, podemos ver una serie de botones que nos permiten ver pistas, mostrar los parámetros de la solicitud HTTP, las cookies de la solicitud HTTP y el código fuente Java.



En este ejemplo ingresé mi nombre “**Diego**” y el servidor le dio la vuelta a la cadena y me devolvió “**ogeID**”. También podemos observar que el número 2 se cambia a color verde, lo que significa que hemos completado la lección.



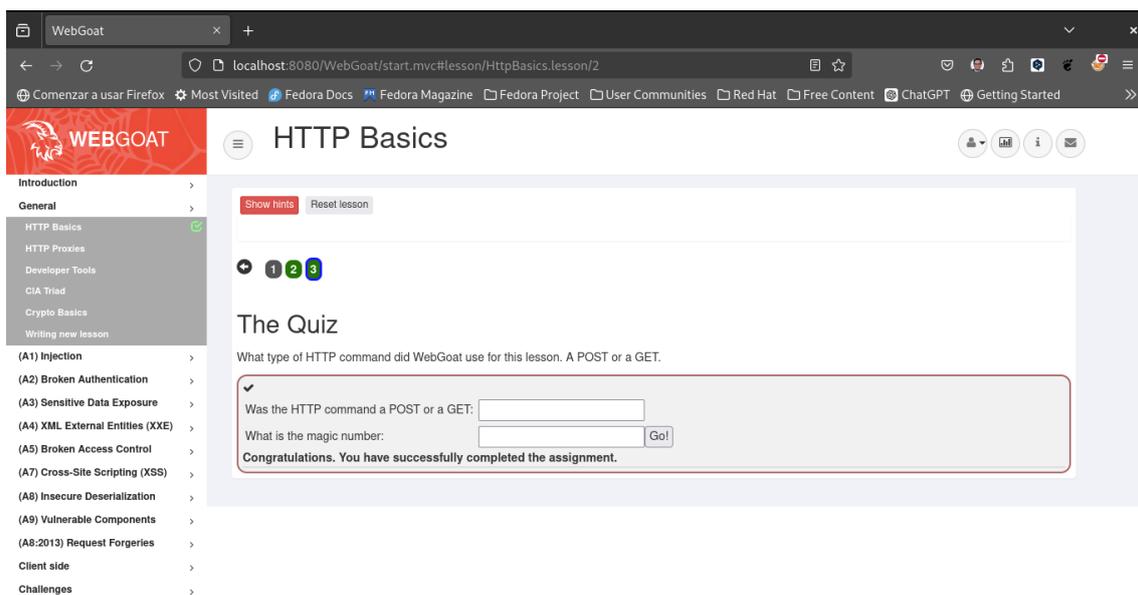
Completada la lección 2, podemos pasar a la lección 3 donde tenemos un pequeño quiz sobre HTTP.

### The Quiz

**What type of HTTP command did WebGoat use for this lesson. A POST or a GET.**

**What the HTTP command a POST or a GET: POST**

**What is the magic number: 42**



De esta forma vamos analizando este laboratorio y aprendiendo sobre seguridad en aplicaciones web.

## 7.4 Conclusión

WebGoat es una herramienta muy útil para aprender sobre seguridad en aplicaciones web. Nos permite realizar diferentes lecciones y aprender sobre vulnerabilidades comunes en aplicaciones web. Además, es muy fácil de instalar y utilizar, por lo que es una buena opción para principiantes que quieran aprender sobre seguridad en aplicaciones web.

## 7.5 Referencias

- [WebGoat](#)
- [WebGoat Docker](#)
- [WebGoat Online](#)
- [WebGoat GitHub](#)
- [OWASP](#)
- [HTTP Basics](#)
- [HTTP Basics Quiz](#)

## 8 Laboratorio de Configuración de TLS/SSL



### 8.1 Objetivos:

- Configurar un servidor web Apache o Nginx con TLS/SSL.
- Verificar la configuración utilizando herramientas como SSL Labs.

## 9 Herramientas

- Máquina Virtual con Ubuntu 24.04 LTS.
- Servidor Web Nginx.
- Certificado SSL autofirmado.

# 10 Procedimiento

## 10.1 1. Instalación de Nginx

Para instalar Nginx en Ubuntu 24.04, primero actualizamos el índice de paquetes y luego instalamos el paquete de Nginx.

```
sudo apt update
sudo apt install nginx
```

### Tip

Podemos conectarnos por SSH a la máquina virtual para ejecutar los comandos.

```
ssh usuario@direccion_ip
```

Para verificar que Nginx se ha instalado correctamente, podemos ejecutar el siguiente comando:

```
sudo systemctl status nginx
```

## 10.2 2. Configuración de Nginx

Verificamos los sitios disponibles con nginx:

```
ls /etc/nginx/sites-available/
```

Analizamos el archivo de configuración por defecto:

```
cat /etc/nginx/sites-available/default
```

Creamos un directorio llamado example para un nuevo sitio en nuestro servidor nginx:

```
sudo mkdir /var/www/example
```

Le brindamos permisos de nuestro usuario al directorio:

```
sudo chown -R $USER:$USER /var/www/example
```

Tambien le asignamos los siguientes permisos:

```
sudo chmod -R 755 /var/www/example
```

Con ello podemos crear un archivo index.html en el directorio example:

```
nano /var/www/example/index.html
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome to Example.com!</title>
</head>
<body>
  <h1>Success! The example.com server block is working!</h1>
</body>
</html>
```

Nos movemos a los sitios disponibles de nginx:

```
cd /etc/nginx/sites-available/
```

Copiamos el archivo de configuración por defecto y le asignamos el nombre example:

```
sudo cp default example
```

Verificamos con un ls que se haya creado el archivo:

```
ls
```

Vamos a modificar el archivo example que acabamos de crear:

```
sudo nano example
```

```
server {
    listen 80;
    listen [::]:80;

    root /var/www/example;
    index index.html index.htm index.nginx-debian.html;

    server_name example.com www.example.com;
```

```
location / {
    try_files $uri $uri/ =404;
}
}
```

Guardamos y cerramos el archivo.

Creamos un enlace simbólico en la carpeta sites-enabled para habilitar el sitio:

```
sudo ln -s /etc/nginx/sites-available/example /etc/nginx/sites-enabled/
```

Verificamos la configuración de Nginx:

```
sudo nginx -t
```

#### Tip

Si todo salio bien deberíamos ver un mensaje como el siguiente:

```
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Reiniciamos el servicio de Nginx:

```
sudo systemctl restart nginx
```

Antes de proceder con la instalación del certificado SSL, verificamos que el sitio web esté funcionando correctamente. Para ello, abrimos un navegador web y accedemos a la dirección IP de la máquina virtual.

#### Tip

Podemos configurar el dominio en nuestro archivo hosts para acceder al sitio web con un nombre de dominio en lugar de la dirección IP. Para ello, agregamos una entrada en el archivo `/etc/hosts` de la máquina cliente.

```
sudo nano /etc/hosts
```

Agregamos la siguiente línea:

```
127.0.1.1 ubuntu example.com
```

Guardamos y cerramos el archivo.

Volvemos a reiniciar el servidor Nginx:

```
sudo systemctl restart nginx
```

Mediante el navegador web, accedemos al sitio web utilizando el nombre de dominio que configuramos en el archivo hosts.

Si todo salio bien deberíamos ver el mensaje que configuramos en el archivo index.html.

### 10.3 3. Configuración de TLS/SSL

Para empezar esta sección vamos a crear una carpeta para almacenar los certificados SSL:

```
sudo mkdir /etc/nginx/ssl
```

Le damos permisos al directorio que acabamos de crear:

```
sudo chmod 700 /etc/nginx/ssl
```

Ahora vamos a generar con openssl el fichero de certificado y la clave privada:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.key
```

El comando anterior generará un certificado autofirmado válido por 365 días. Se le pedirá que proporcione información sobre su organización y ubicación. Puede completar esta información o dejarla en blanco.

Comprobamos que los archivos se hayan generado correctamente:

```
ls /etc/nginx/ssl
```

Ahora volvemos a nuestros sitios disponibles de nginx:

```
cd /etc/nginx/sites-available/
```

Vamos a modificar el archivo de configuración del sitio para habilitar TLS/SSL:

```
sudo nano example
```

```
server {
    listen 80;
    listen [::]:80;

    root /var/www/example;
    index index.html index.htm index.nginx-debian.html;
```

```
server_name example.com www.example.com;

location / {
    try_files $uri $uri/ =404;
}

listen 443 ssl; # managed by Certbot
ssl_certificate /etc/nginx/ssl/nginx.crt; # managed by Certbot
ssl_certificate_key /etc/nginx/ssl/nginx.key; # managed by Certbot
}
```

Guardamos y cerramos el archivo.

Recargamos el servidor Nginx:

```
sudo systemctl reload nginx
```

Se recomienda en este punto reiniciar la máquina virtual para asegurarnos de que los cambios se apliquen correctamente.

Si todo salió bien deberíamos poder acceder al sitio web mediante HTTPS.



**Success! The example.com server block is working!**

## 10.4 4. Verificación de la Configuración de TLS/SSL

Para verificar la configuración de TLS/SSL, podemos utilizar la herramienta SSL Labs. Para ello, abrimos un navegador web y accedemos a la siguiente URL <https://www.ssllabs.com/ssltest/>

En el campo de texto, ingresamos la URL de nuestro sitio web y hacemos clic en el botón “Submit”.

La herramienta SSL Labs analizará la configuración de TLS/SSL de nuestro sitio web y nos proporcionará un informe detallado con una calificación.

 Tip

Es posible que la calificación no sea la máxima si estamos utilizando un certificado autofirmado. Sin embargo, el informe proporcionará información útil sobre la configuración de TLS/SSL de nuestro sitio web.

## 10.5 Conclusión

En este laboratorio, hemos configurado un servidor web Nginx con TLS/SSL en una máquina virtual Ubuntu 24.04. Hemos creado un certificado SSL autofirmado y hemos verificado la configuración utilizando la herramienta SSL Labs.

## 11 Referencias

- [How To Install Nginx on Ubuntu 20.04](#)
- [How To Configure Nginx with SSL as a Reverse Proxy for Jenkins](#)
- [How To Secure Nginx with Let's Encrypt on Ubuntu 20.04](#)
- [How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 20.04](#)

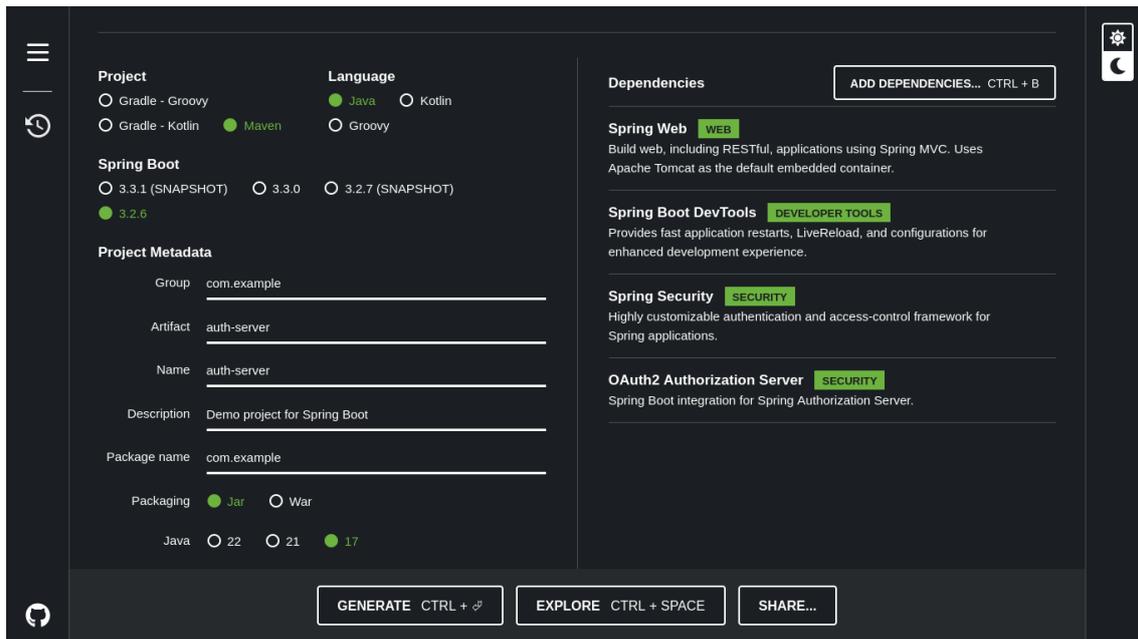
## **12 Laboratorio de autenticación OAuth 2 y JWT con Spring Security Authorization Server.**

# 13 Instrucciones

1. Crear el proyecto con Spring Initializr <https://start.spring.io/>.

Nos aseguramos de agregar los siguientes campos:

- Project: Maven Project
- Language: Java
- Spring Boot: 3.2.1
- Project Metadata:
  - Group: com.example
  - Artifact: auth-server
  - Name: auth-server
  - Description: Demo project for Spring Boot
  - Package name: com.example
- Packaging: Jar
- Java: 17
- Dependencies:
  - Spring Boot DevTools
  - Spring Web
  - Spring Security
  - Spring Security OAuth2 Authorization Server



Generamos el proyecto y lo descomprimos para empezar.

#### 💡 Tip

Puedes utilizar este [link](#)

2. Abrimos el proyecto en nuestro IDE (Eclipse, IntelliJ, NetBeans, etc). En este tutorial se utilizará IntelliJ IDEA.

#### 💡 Tip

Canviamos la version de Spring Boot por 3.2.1 en el archivo pom.xml

3. Creamos un nuevo paquete llamado **auth** y dentro de este paquete creamos una clase llamada **SecurityConfig**.

Podemos utilizar la documentación oficial en el siguiente link <https://docs.spring.io/spring-authorization-server/reference/getting-started.html>

```
package com.example.auth;

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.UUID;

import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
```

```

import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.http.MediaType;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
import org.springframework.security.oauth2.core.oidc.OidcScopes;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.server.authorization.client.InMemoryRegisteredClientRepository;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepository;
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configuration.OAuth2AuthorizationServerConfiguration;
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configuration.OAuth2AuthorizationServerConfiguration;
import org.springframework.security.oauth2.server.authorization.settings.AuthorizationServerSettings;
import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
import org.springframework.security.provisioning.InMemoryUserDetailsService;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint;
import org.springframework.security.web.util.matcher.MediaTypeRequestMatcher;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    @Order(1)
    public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http)
        throws Exception {
        OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
        http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
            .oidc(Customizer.withDefaults()); // Enable OpenID Connect 1.0

        http
            // Redirect to the login page when not authenticated from the
            // authorization endpoint
            .exceptionHandling((exceptions) -> exceptions
                .defaultAuthenticationEntryPointFor(
                    new LoginUrlAuthenticationEntryPoint("/login"),
                    new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
                )
            )
    }
}

```

```

    )
    // Accept access tokens for User Info and/or Client Registration
    .oauth2ResourceServer((resourceServer) -> resourceServer
        .jwt(Customizer.withDefaults()));

    return http.build();
}

@Bean
@Order(2)
public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
    throws Exception {
    http
        .authorizeHttpRequests((authorize) -> authorize
            .anyRequest().authenticated()
        )
        // Form login handles the redirect to the login page from the
        // authorization server filter chain
        .csrf(csrf -> csrf.disable())
        .formLogin(Customizer.withDefaults());

    return http.build();
}

@Bean
public UserDetailsService userDetailsService() {
    UserDetails userDetails = User.builder()
        .username("juan")
        .password("{noop}12345")
        .roles("USER")
        .build();

    return new InMemoryUserDetailsManager(userDetails);
}

@Bean
public RegisteredClientRepository registeredClientRepository() {
    RegisteredClient oidcClient = RegisteredClient.withId(UUID.randomUUID().toString())
        .clientId("client-app")
        .clientSecret("{noop}12345")
        .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
        .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
        .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
        .redirectUri("http://127.0.0.1:8080/login/oauth2/code/client-app")
        .redirectUri("http://127.0.0.1:8080/authorized")
        .postLogoutRedirectUri("http://127.0.0.1:8080/logout")
        .scope("read")
        .scope("write")
}

```

```

        .scope(OidcScopes.OPENID)
        .scope(OidcScopes.PROFILE)
        .clientSettings(ClientSettings.builder().requireAuthorizationConsent(false)
            .build());

    return new InMemoryRegisteredClientRepository(oidcClient);
}

@Bean
public JWKSSource<SecurityContext> jwkSource() {
    KeyPair keyPair = generateRsaKey();
    RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
    RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
    RSAKey rsaKey = new RSAKey.Builder(publicKey)
        .privateKey(privateKey)
        .keyID(UUID.randomUUID().toString())
        .build();
    JWKS jwkSet = new JWKS(rsaKey);
    return new ImmutableJWKS(jwkSet);
}

private static KeyPair generateRsaKey() {
    KeyPair keyPair;
    try {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2048);
        keyPair = keyPairGenerator.generateKeyPair();
    }
    catch (Exception ex) {
        throw new IllegalStateException(ex);
    }
    return keyPair;
}

@Bean
public JwtDecoder jwtDecoder(JWKSSource<SecurityContext> jwkSource) {
    return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
}

@Bean
public AuthorizationServerSettings authorizationServerSettings() {
    return AuthorizationServerSettings.builder().build();
}
}

```

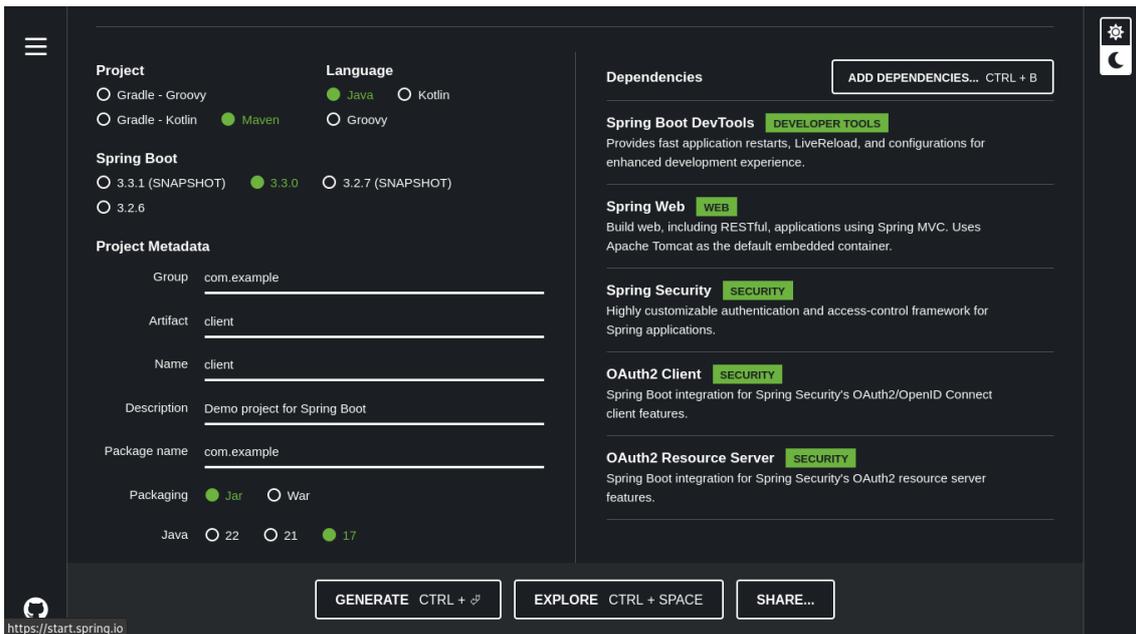
4. Creamos una configuración para el puerto de autorización en el archivo **application.properties**.

```
server.port=9000
```

5. Creamos un segundo proyecto con Spring Initializr <https://start.spring.io/>. Que servirá como cliente de nuestro servidor de autorización.

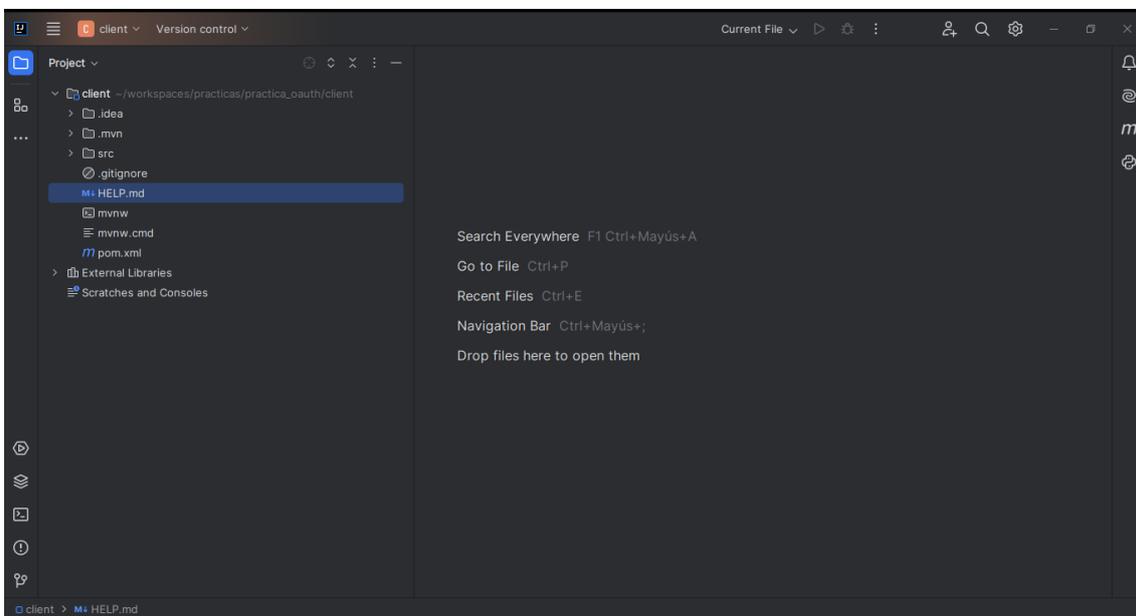
Nos aseguramos de agregar los siguientes campos:

- Project: Maven Project
- Language: Java
- Spring Boot: 3.2.1
- Project Metadata:
  - Group: com.example
  - Artifact: client
  - Name: client
  - Description: Demo project for Spring Boot
  - Package name: com.example
- Packaging: Jar
- Java: 17
- Dependencies:
  - Spring Boot DevTools
  - Spring Web
  - Spring Security
  - Spring Security OAuth2 Client
  - Spring Security OAuth2 Resource Server



Podemos utilizar la siguiente url <https://start.spring.io/#!type=maven-project&language=java&platformVersion=3.3.0&packaging=jar&jvmVersion=17&groupId=com.example&artifactId=client&name=client&description=Demo%20project%20for%20Spring%20Boot&packageName=com.example&dependencies=devtools,web,security,oauth2-client,oauth2-resource-server>

Abrimos el proyecto en una nueva ventana de nuestro IDE.



6. Creamos un archivo de configuración en la sección resources llamado **application.yml**.

```

spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: "http://127.0.0.1:9000"
      client:
        registration:
          client-app:
            provider: spring
            client-id: client-id
            client-secret: 12345
            authorization-grant-type: authorization_code
            redirect-uri: "http://127.0.0.1:8080/authorized"
            scope:
              - openid
              - profile
              - read
            client-name: client-app
        provider:
          spring:
            issuer-uri: "http://127.0.0.1:9000"

```

7. Creamos un paquete llamado **controllers** y dentro de este paquete creamos una clase llamada **AppController**.

```

package com.example.controller;

import com.example.models.Message;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Map;

@RestController
public class AppController {

    @GetMapping("/list")
    public List<Message> list(){
        return Collections.singletonList(new Message( "Test List"));
    }

    @PostMapping("/create")
    public Message create(@RequestBody Message message){
        System.out.println("mensaje guardado: " + message);
    }
}

```

```

        return message;
    }

    @GetMapping("/authorized")
    public Map<String, String> authorized(@RequestParam String code){
        return Collections.singletonMap("code", code);
    }
}

```

8. Creamos un nuevo paquete llamado **models** y dentro de este paquete creamos una clase llamada **Message**.

```

package com.example.models;

public class Message {

    private String text;

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public Message() {
    }

    public Message(String text) {
        this.text = text;
    }
}

```

9. Creamos un nuevo paquete llamado **auth** y dentro de este paquete creamos una clase llamada **SecurityConfig**.

```

package com.example.auth;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import static org.springframework.security.config.Customizer.withDefaults;

```

```

@Configuration
public class SecurityConfig {

    @Bean
    SecurityFilterChain SecurityFilterChain(HttpSecurity http) throws Exception {

        http.authorizeHttpRequests((authHttp) -> authHttp
            .requestMatchers(HttpMethod.GET, "/authorized").permitAll()
            .requestMatchers(HttpMethod.GET, "/list").hasAnyAuthority("SCOPE_read", "SCOPE_write")
            .requestMatchers(HttpMethod.POST, "/create").hasAnyAuthority("SCOPE_write")
            .anyRequest().authenticated()
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .oauth2Login(login -> login.loginPage("/oauth2/authorization/client-app"))
            .oauth2Client(withDefaults())
            .oauth2ResourceServer(resourceServer -> resourceServer.jwt(withDefaults()))

        return http.build();
    }
}

```

10. Levantamos el servidor de autorización y el cliente.
11. Ingresamos a la siguiente url <http://127.0.0.1:9000/oauth2/authorization/client-app> y nos autenticamos con el usuario **juan** y la contraseña **12345**.
12. Una vez autenticados, realiza una redirección
13. Podemos comprobar con Postman que el servidor de autorización está funcionando correctamente.
14. Podemos comprobar con Postman que el cliente está funcionando correctamente.
15. Podemos comprobar con Postman que el cliente está funcionando correctamente.

Con esto hemos terminado el laboratorio de autenticación OAuth 2 y JWT con Spring Security Authorization Server.

 Warning

Hay un error en el código del cliente, por favor resuelve el error y vuelve a ejecutar el laboratorio.

## 14 Reto

Implementar un servidor de autorización con Spring Security Authorization Server y un cliente con Spring Security OAuth2 Client y Spring Security OAuth2 Resource Server.

## 15 Conclusiones

- Spring Security Authorization Server es una herramienta muy útil para la autenticación de aplicaciones.
- Spring Security OAuth2 Client y Spring Security OAuth2 Resource Server son herramientas muy útiles para la autenticación de aplicaciones.
- OAuth2 es un protocolo de autorización que permite a una aplicación obtener acceso a los recursos de un usuario sin tener que almacenar las credenciales del usuario.

## 16 Referencias

- <https://docs.spring.io/spring-authorization-server/reference/getting-started.html>
- <https://start.spring.io/>
- <https://jwt.io/>
- [https://github.com/statick88/practica\\_jwt\\_auth](https://github.com/statick88/practica_jwt_auth)
- <https://www.youtube.com/watch?v=zDzvuTBrP1I>

# 17 Laboratorio: Aplicación de Buenas Prácticas de Seguridad

## 17.1 Instrucciones del Laboratorio

**Objetivo:** Aplicar los conocimientos adquiridos sobre la evolución de los ataques, taxonomías de amenazas y vulnerabilidades, validación y manejo de entradas, y buenas prácticas según OWASP en un entorno práctico.

## 17.2 Descripción del laboratorio:

- **Parte 1:** Análisis de vulnerabilidades en una aplicación web existente.
- **Parte 2:** Implementación de medidas de seguridad para mitigar las vulnerabilidades identificadas.
- **Parte 3:** Documentación de los pasos realizados y resultados obtenidos.

## 17.3 Parte 1 - Análisis de Vulnerabilidades

### 17.3.1 Configuración del entorno:

Descargar y configurar una aplicación web vulnerable (e.g., OWASP Juice Shop).

**Herramientas recomendadas:** OWASP ZAP, Burp Suite.

**Realización de pruebas de seguridad:**

**Escaneo de vulnerabilidades:** Utilizar herramientas de escaneo para identificar vulnerabilidades comunes.

**Pruebas manuales:** Realizar pruebas manuales para detectar problemas de seguridad no identificados automáticamente.

**Documentación de vulnerabilidades:** Crear un informe detallado de las vulnerabilidades encontradas, incluyendo descripciones, posibles impactos y recomendaciones para su mitigación.

## 17.4 Parte 2 - Implementación de Medidas de Seguridad

### Medidas a implementar:

**Validación de entradas:** Añadir validación y sanitización de todas las entradas de usuario.

**Configuración segura:** Asegurar que la configuración de la aplicación y los servidores sea segura (e.g., deshabilitar opciones no necesarias, configurar permisos adecuados).

**Control de acceso y autenticación:** Implementar un control de acceso riguroso y autenticación multifactor.

**Cifrado de datos:** Utilizar cifrado para proteger datos sensibles tanto en tránsito como en reposo.

**Manejo seguro de sesiones:** Configurar cookies seguras y gestionar sesiones de forma segura.

**Pruebas de seguridad post-implementación:** Realizar un nuevo análisis de vulnerabilidades para verificar que las medidas de seguridad implementadas han sido efectivas.

## 17.5 Parte 3 - Documentación

**Informe de Seguridad:** Crear un informe detallado que incluya:

- Descripción de la aplicación web analizada.
- Resumen de las vulnerabilidades identificadas.
- Descripción de las medidas de seguridad implementadas.
- Resultados de las pruebas de seguridad post-implementación.
- Recomendaciones para futuras mejoras de seguridad.

**Presentación:** Preparar una presentación para compartir los resultados del análisis con el equipo de desarrollo y la dirección de la empresa.

## 17.6 Entregables:

- Informe de Seguridad.
- Presentación.

## 17.7 Evaluación:

- Calidad del análisis de vulnerabilidades.
- Eficacia de las medidas de seguridad implementadas.
- Claridad y completitud del informe y la presentación.

## 17.8 Recursos:

- OWASP Top Ten Project.
- OWASP Juice Shop.
- OWASP ZAP.
- Burp Suite.

## 17.9 Conclusiones

Este laboratorio proporciona una oportunidad para aplicar los conocimientos adquiridos sobre la seguridad de las aplicaciones web en un entorno práctico. Al analizar una aplicación web existente, identificar vulnerabilidades y aplicar medidas de seguridad, los estudiantes pueden mejorar sus habilidades en la identificación y mitigación de riesgos de seguridad. Además, la documentación y presentación de los resultados ayudan a comunicar eficazmente los hallazgos y recomendaciones a los interesados relevantes. En general, este laboratorio es una forma efectiva de fortalecer la comprensión y la práctica de las buenas prácticas de seguridad en el desarrollo de aplicaciones web.

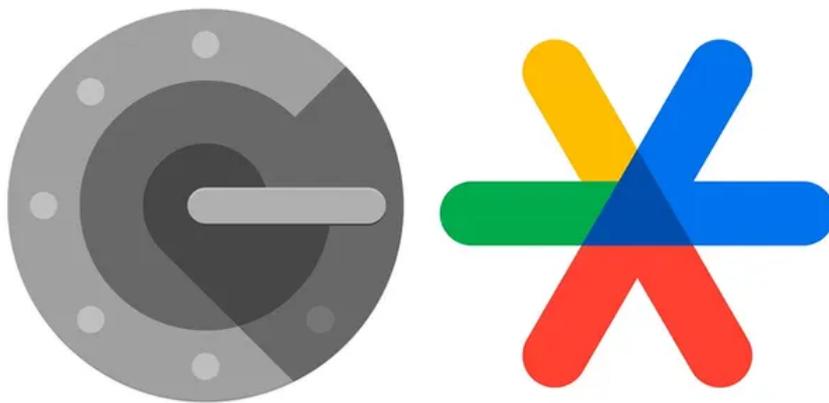
## 17.10 Referencias

- OWASP Top Ten Project: <https://owasp.org/www-project-top-ten/>
- OWASP Juice Shop: <https://owasp.org/www-project-juice-shop/>
- OWASP ZAP: <https://www.zaproxy.org/>
- Burp Suite: <https://portswigger.net/burp>
- “The Web Application Hacker’s Handbook” by Dafydd Stuttard and Marcus Pinto: <https://www.wiley.com/en-us/The+Web+Application+Hacker%27s+Handbook%3A+Finding+and+Exploiting+Security+Flaws%2C+2nd+Edition-p-9781118026472>
- “Web Security Testing Cookbook” by Paco Hope and Ben Walther: <https://www.oreilly.com/library/view/web-security-testing/9780596514839/>

- “The Tangled Web: A Guide to Securing Modern Web Applications” by Michal Zalewski: <https://www.amazon.com/Tangled-Web-Securing-Modern-Applications/dp/1593273886>
- “Secure Programming Cookbook for C and C++” by John Viega and Matt Messier: <https://www.oreilly.com/library/view/secure-programming-cookbook/0596003943/>
- “Cryptography Engineering: Design Principles and Practical Applications” by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno: <https://www.amazon.com/Cryptography-Engineering-Principles-Practical-Applications/dp/0470474246>
- “The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities” by Mark Dowd, John McDonald, and Justin Schuh: <https://www.amazon.com/Art-Software-Security-Assessment-Vulnerabilities/dp/0321444426>
- “Threat Modeling: Designing for Security” by Adam Shostack: <https://www.amazon.com/Threat-Modeling-Designing-Adam-Shostack/dp/1118809998>
- “The Basics of Web Hacking: Tools and Techniques to Attack the Web” by Josh Pauli: <https://www.amazon.com/Basics-Web-Hacking-Techniques-Attack/dp/0124166008>
- “Hacking: The Art of Exploitation” by Jon Erickson: <https://www.amazon.com/Hacking-Art-Exploitation-Jon-Erickson/dp/1593271441>
- “The Shellcoder’s Handbook: Discovering and Exploiting Security Holes” by Jack Koziol, David Litchfield, Dave Aitel, Chris Anley, Sinan Eren, Neel Mehta, and Riley Hassell: <https://www.amazon.com/Shellcoders-Handbook-Discovering-Exploiting-Security/dp/047008023X>
- “Gray Hat Python: Python Programming for Hackers and Reverse Engineers” by Justin Seitz: <https://www.amazon.com/Gray-Hat-Python-Programming-Engineers/dp/1593271921>
- “Black Hat Python: Python Programming for Hackers and Pentesters” by Justin Seitz: <https://www.amazon.com/Black-Hat-Python-Programming-Pentesters/dp/1593275900>
- “Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers” by TJ O’Connor: <https://www.amazon.com/Violent-Python-Cookbook-Penetration-Engineers/dp/1597499579>
- “Python Web Penetration Testing Cookbook” by Cameron Buchanan, Terry Ip, Andrew Mabbitt, and Benjamin May: <https://www.amazon.com/Python-Web-Penetration-Testing-Cookbook/dp/178439293X>
- “The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws” by Dafydd Stuttard and Marcus Pinto: <https://www.amazon.com/Web-Application-Hackers-Handbook-Exploiting/dp/1118026470>
- “Web Hacking 101: How to Make Money Hacking Ethically” by Peter Yaworski: <https://www.amazon.com/Web-Hacking-101-Peter-Yaworski/dp/1593277334>

- “Mastering Modern Web Penetration Testing” by Prakhar Prasad: <https://www.amazon.com/Mastering-Modern-Web-Penetration-Testing/dp/1785284584>

# 18 Laboratorio: Implementación de Doble Factor de Autenticación (2FA) con Google Authenticator



En este laboratorio, aprenderemos a implementar la doble autenticación (2FA) utilizando Google Authenticator en una aplicación Node.js. Usaremos las bibliotecas `speakeasy` y `qrcode` para generar secretos y códigos QR.

## 18.1 Objetivos

- Entender el concepto de doble factor de autenticación (2FA).
- Implementar 2FA en una aplicación Node.js.
- Generar y verificar tokens de Google Authenticator.

## 18.2 Prerrequisitos

- Conocimientos básicos de JavaScript y Node.js.
- Node.js y npm instalados en tu sistema.

## 18.3 Paso 1: Configuración del Proyecto

Crea un nuevo directorio para tu proyecto y navega dentro de él:

```
mkdir 2fa-demo
cd 2fa-demo
```

Inicializa un nuevo proyecto **Node.js**:

```
npm init -y
```

Instala las dependencias necesarias:

```
npm install qrcode speakeasy
```

## 18.4 Paso 2: Generación del Secreto y Código QR

- Crea un archivo **index.js** con el siguiente contenido:

```
const speakeasy = require('speakeasy');
const qrcode = require('qrcode');

// Generar un secreto
let secret = speakeasy.generateSecret({ name: "Demo 2FA" });

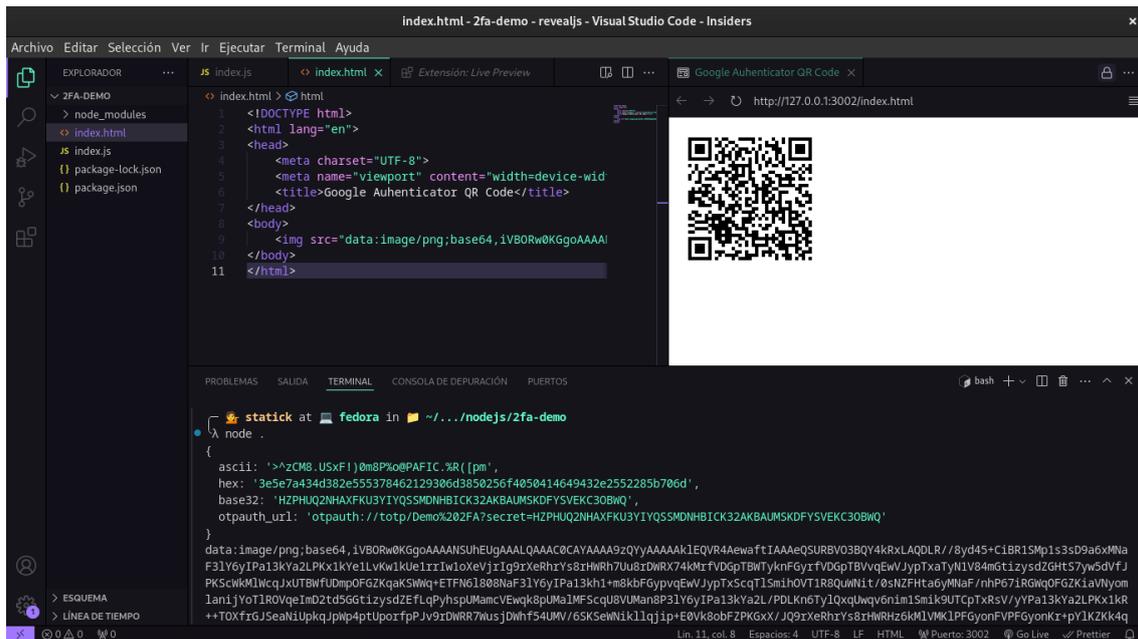
// Imprimir el secreto en consola
console.log(secret);

// Generar un código QR con el URL de otpauth
qrcode.toDataURL(secret.otpauth_url, function(err, data) {
  if (err) {
    console.error("Error generando el código QR", err);
    return;
  }
  console.log(data);
});
```

Ejecuta el script para **generar el secreto y el código QR**:

```
node .
```





## 18.6 Paso 4: Verificación del Token

Crema un archivo `verify.js` con el siguiente contenido:

```
const speakeasy = require('speakeasy');

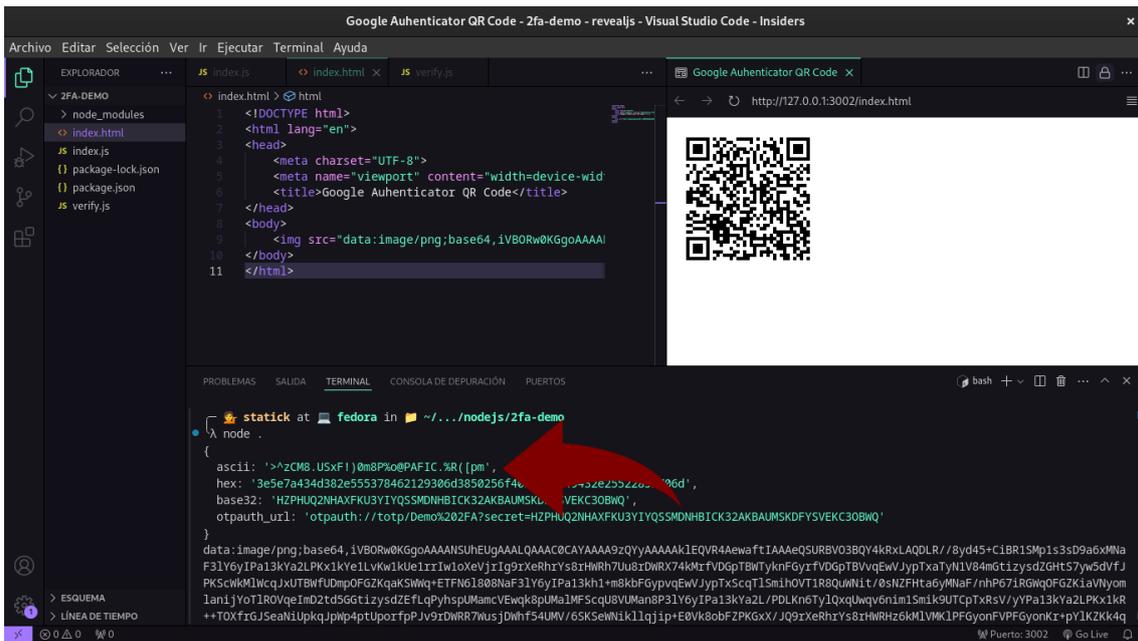
// Reemplaza 'YOUR_SECRET' con el secreto generado en el paso 2
const secret = 'YOUR_SECRET';

// Obtén el token desde Google Authenticator y reemplázalo aquí
const token = 'YOUR_TOKEN';

let verified = speakeasy.totp.verify({
  secret: secret,
  encoding: 'ascii',
  token: token
});

console.log(verified);
```

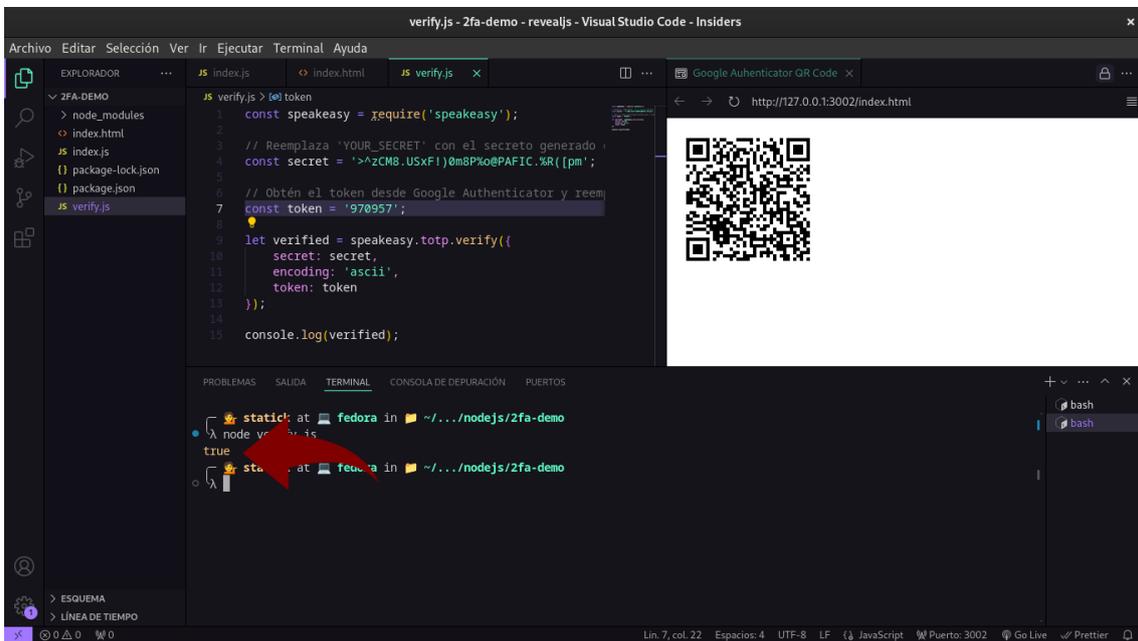
Reemplaza **YOUR\_SECRET** con el secreto generado en el paso 2 y **YOUR\_TOKEN** con el token generado en Google Authenticator.



Ejecuta el script para verificar el token:

```
node verify.js
```

Si el token es válido, verás true en la consola. Si no, verás false.



verify.js - 2fa-demo - reveals.js - Visual Studio Code - Insiders

Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda

EXPLORADOR ... JS index.js index.html JS verify.js X

2FA-DEMO  
node\_modules  
index.html  
index.js  
package-lock.json  
package.json  
verify.js

```
1 JS verify.js > token
2 const speakeasy = require('speakeasy');
3
4 // Reemplaza 'YOUR_SECRET' con el secreto generado
5 const secret = '^zCM8.USxF10m8Pxo@PAFIC.XR([pm';
6
7 // Obtén el token desde Google Authenticator y reem
8 const token = '970957';
9
10 let verified = speakeasy.totp.verify({
11   secret: secret,
12   encoding: 'ascii',
13   token: token
14 });
15 console.log(verified);
```

Google Authenticator QR Code X

http://127.0.0.1:3002/index.html



PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN PUERTOS

```
statick at fedora in ~/.../nodejs/2fa-demo
λ node verify.js
true
statick at fedora in ~/.../nodejs/2fa-demo
λ node v
false
statick at fedora in ~/.../nodejs/2fa-demo
λ
```

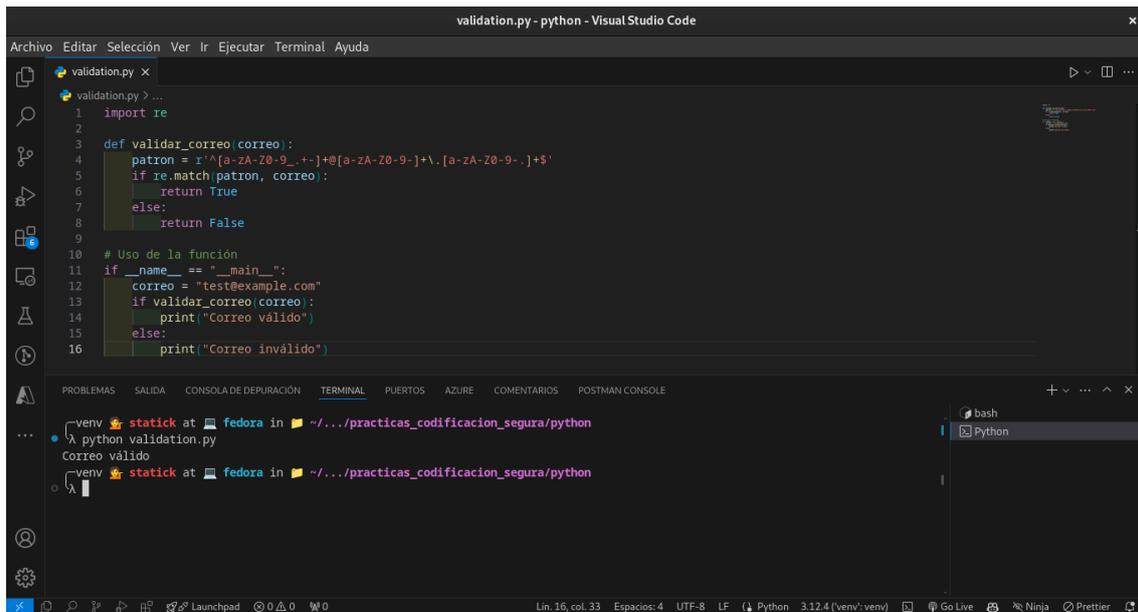
ESQUEMA  
LINEA DE TIEMPO

Lin. 7, col. 22 Espacios: 4 UTF-8 LF JavaScript Puerto: 3002 Go Live Prettier

## 19 Conclusión

En este laboratorio, aprendimos cómo implementar la doble autenticación (2FA) utilizando **Google Authenticator** en una aplicación **Node.js**. Generamos un **secreto** y un **código QR**, y luego verificamos el token generado por **Google Authenticator**. Esta implementación proporciona una capa adicional de seguridad a las aplicaciones, ayudando a proteger contra accesos no autorizados.

## 20 Laboratorio de Codificación Segura



```
validation.py - python - Visual Studio Code
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
validation.py x
validation.py > ...
1 import re
2
3 def validar_correo(correo):
4     patron = r'^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$'
5     if re.match(patron, correo):
6         return True
7     else:
8         return False
9
10 # Uso de la función
11 if __name__ == "__main__":
12     correo = "test@example.com"
13     if validar_correo(correo):
14         print("Correo válido")
15     else:
16         print("Correo inválido")
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS AZURE COMENTARIOS POSTMAN CONSOLE
-venv statick at fedora in ~/.../practicass_codificacion_segura/python
λ python validation.py
Correo válido
-venv statick at fedora in ~/.../practicass_codificacion_segura/python
λ
Lin. 16, col. 33 Espacios: 4 UTF-8 LF Python 3.12.4 ('venv':venv) Go Live Ninja Prettier
```

### 20.1 Python

En este primer laboratorio vamos a aprender a implementar prácticas de codificación segura en Python mediante la validación de entradas y el manejo seguro de datos sensibles.

### 20.2 Objetivo:

Aprender a implementar prácticas de codificación segura en Python mediante la validación de entradas y el manejo seguro de datos sensibles.

### 20.3 Requisitos:

- Python 3.12.4 o superior
- Editor de código (por ejemplo, VS Code, PyCharm)

#### Paso 1: Configuración del entorno

- Instalación de Python:
- Asegúrate de tener Python instalado. Puedes descargarlo desde [www.python.org](http://www.python.org).

### Creación de un entorno virtual:

```
python -m venv venv
source venv/bin/activate # En Windows usa `venv\Scripts\activate`
```

### Instalación de las dependencias necesarias:

En este laboratorio no se requieren dependencias externas adicionales.

### Paso 2: Validación de entradas

#### Creación del archivo de validación:

- Crea un archivo llamado validation.py.
- Implementación de la función de validación de correo electrónico:
- Copia y pega el siguiente código en validation.py:

```
import re

def validar_correo(correo):
    patron = r'^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$'
    if re.match(patron, correo):
        return True
    else:
        return False

# Uso de la función
if __name__ == "__main__":
    correo = "test@example.com"
    if validar_correo(correo):
        print("Correo válido")
    else:
        print("Correo inválido")
```

### Ejecución del código:

En la terminal, navega hasta el directorio que contiene validation.py y ejecuta:

```
python validation.py
```

Deberías ver el mensaje “Correo válido” en la salida.

### Paso 3: Validación adicional

#### Validación de números de teléfono:

- Agrega la siguiente función a validation.py para validar números de teléfono:

```

def validar_telefono(telefono):
    patron = r'^\+?1?\d{9,15}$'
    if re.match(patron, telefono):
        return True
    else:
        return False

# Uso de la función
if __name__ == "__main__":
    telefono = "+12345678901"
    if validar_telefono(telefono):
        print("Teléfono válido")
    else:
        print("Teléfono inválido")

```

### Ejecución del código:

The screenshot shows the Visual Studio Code interface with a Python file named 'validation.py' open. The code in the editor matches the code block above. The terminal window at the bottom shows the following execution:

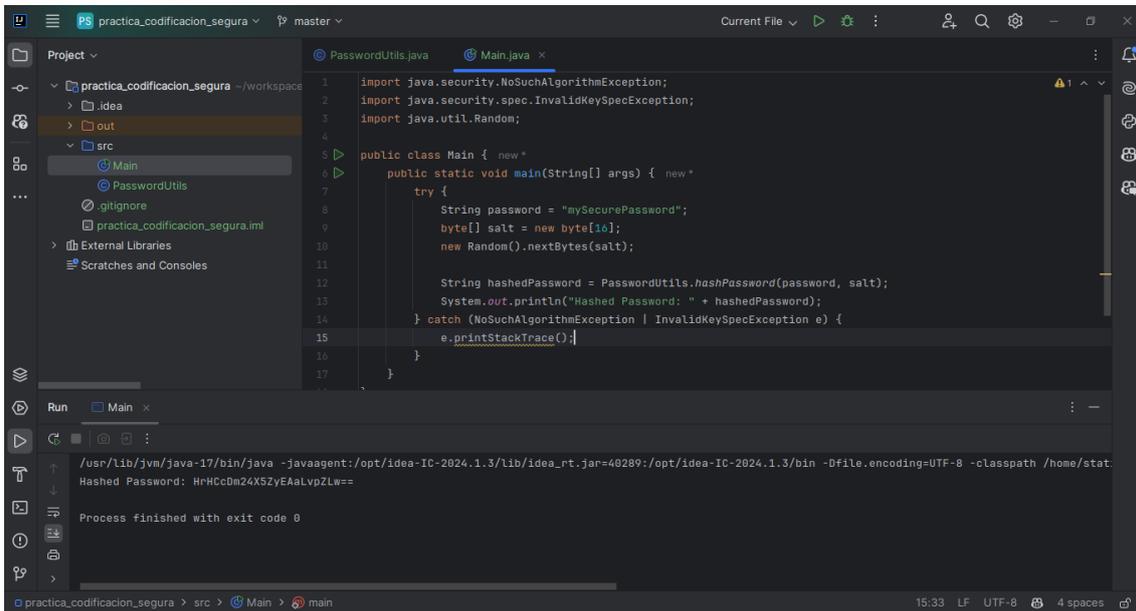
```

~venv static at fedora in ~/.../practicas_codificacion_segura/python
λ python validation.py
Correo válido
Teléfono válido
~venv static at fedora in ~/.../practicas_codificacion_segura/python
λ

```

Ejecuta el archivo de nuevo para ver la validación de teléfono.

## 20.4 Java



```
1 import java.security.NoSuchAlgorithmException;
2 import java.security.spec.InvalidKeySpecException;
3 import java.util.Random;
4
5 public class Main {
6     public static void main(String[] args) {
7         try {
8             String password = "mySecurePassword";
9             byte[] salt = new byte[16];
10            new Random().nextBytes(salt);
11
12            String hashedPassword = PasswordUtils.hashPassword(password, salt);
13            System.out.println("Hashed Password: " + hashedPassword);
14        } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

Run Main x

```
/usr/lib/jvm/java-17/bin/java -javaagent:/opt/idea-IC-2024.1.3/lib/idea_rt.jar=40289:/opt/idea-IC-2024.1.3/bin -Dfile.encoding=UTF-8 -classpath /home/stat
Hashed Password: HrHCCDm24X52yEAALvpZLw==
Process finished with exit code 0
```

En este laboratorio vamos a aprender a implementar prácticas de codificación segura en Java mediante la gestión segura de contraseñas.

## 20.5 Objetivo:

Aprender a implementar prácticas de codificación segura en Java mediante la gestión segura de contraseñas.

## 20.6 Requisitos:

- Java 8 o superior
- IDE de Java (por ejemplo, IntelliJ IDEA, Eclipse)

### Paso 1: Configuración del entorno

#### Instalación de Java:

Asegúrate de tener Java instalado. Puedes descargarlo desde [www.oracle.com](http://www.oracle.com).

#### Creación de un proyecto en tu IDE:

Abre tu IDE y crea un nuevo proyecto Java.

### Paso 2: Gestión de contraseñas

#### Creación de la clase de utilidades de contraseña:

Creación de la clase de utilidades de contraseña: Crea una clase llamada **PasswordUtils** en tu proyecto.

#### Implementación de la función de hash de contraseña:

Copia y pega el siguiente código en PasswordUtils.java:

```
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.util.Base64;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class PasswordUtils {
    public static String hashPassword(String password, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException {
        PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 65536, 128);
        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] hash = factory.generateSecret(spec).getEncoded();
        return Base64.getEncoder().encodeToString(hash);
    }
}
```

Uso de la función de hash de contraseña:

Crea una clase Main para probar la función:

```
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        try {
            String password = "mySecurePassword";
            byte[] salt = new byte[16];
            new Random().nextBytes(salt);

            String hashedPassword = PasswordUtils.hashPassword(password, salt);
            System.out.println("Hashed Password: " + hashedPassword);
        } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
            e.printStackTrace();
        }
    }
}
```

Ejecución del código:

The screenshot shows an IDE window for a project named 'practica\_codificacion\_segura'. The main editor displays the following Java code in 'Main.java':

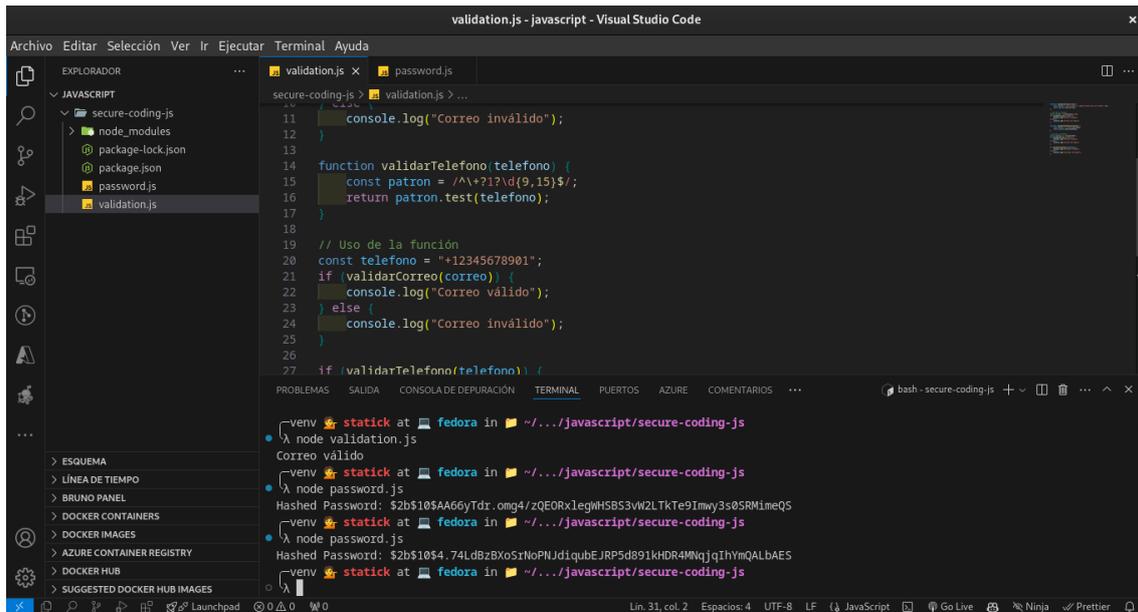
```
1 import java.security.NoSuchAlgorithmException;
2 import java.security.spec.InvalidKeySpecException;
3 import java.util.Random;
4
5 public class Main {
6     public static void main(String[] args) {
7         try {
8             String password = "mySecurePassword";
9             byte[] salt = new byte[16];
10            new Random().nextBytes(salt);
11
12            String hashedPassword = PasswordUtils.hashPassword(password, salt);
13            System.out.println("Hashed Password: " + hashedPassword);
14        } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

The 'Run' console at the bottom shows the execution command and output:

```
/usr/lib/jvm/java-17/bin/java -javaagent:/opt/idea-IC-2024.1.3/lib/idea_rt.jar=40289:/opt/idea-IC-2024.1.3/bin -Dfile.encoding=UTF-8 -classpath /home/stat
Hashed Password: HrhCCDm24X52yEAALvpZLw==
Process finished with exit code 0
```

Ejecuta la clase Main en tu IDE. Deberías ver la contraseña hasheada impresa en la consola.

# 21 JavaScript



```
validation.js - javascript - Visual Studio Code
Archivo  Editar  Selección  Ver  Ir  Ejecutar  Terminal  Ayuda
EXPLORADOR
  secure-coding-js
  node_modules
  package-lock.json
  package.json
  password.js
  validation.js
  validation.js x
secure-coding-js > validation.js > ...
11 console.log("Correo inválido");
12 }
13
14 function validarTelefono(telefono) {
15   const patron = /^+?1?\d{9,15}$/;
16   return patron.test(telefono);
17 }
18
19 // Uso de la función
20 const telefono = "+12345678901";
21 if (validarCorreo(correo)) {
22   console.log("Correo válido");
23 } else {
24   console.log("Correo inválido");
25 }
26
27 if (validarTelefono(telefono)) {
  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  AZURE  COMENTARIOS  ...
  bash - secure-coding-js
  venv static at fedora in ~/.../javascript/secure-coding-js
  node validation.js
  Correo válido
  venv static at fedora in ~/.../javascript/secure-coding-js
  node password.js
  Hashed Password: $2b$10$AA66yTdr.omg4/zQEORx1legWHSB53vW2L.TkTe9Imwy3s8SRMimeQS
  venv static at fedora in ~/.../javascript/secure-coding-js
  node password.js
  Hashed Password: $2b$10$4.74LdBzBXo5rNoPNJdiqubEJRP5d891KHDR4MnqjqIhYmQALbAeS
  venv static at fedora in ~/.../javascript/secure-coding-js
  Lin. 31, col. 2  Espacios: 4  UTF-8  LF  JavaScript  Go Live  Ninja  Prettier
```

## 21.1 Objetivo:

Aprender a implementar prácticas de codificación segura en JavaScript mediante la validación de entradas y el manejo seguro de datos sensibles.

## 21.2 Requisitos:

- Node.js y npm instalados
- Editor de código (por ejemplo, VS Code)

### Paso 1: Configuración del entorno

#### Instalación de Node.js:

Asegúrate de tener Node.js instalado. Puedes descargarlo desde [nodejs.org](https://nodejs.org).

#### Creación de un proyecto Node.js:

En la terminal, crea una nueva carpeta para tu proyecto y navega a ella:

```
mkdir secure-coding-js
cd secure-coding-js
```

### Inicialización del proyecto:

#### Inicializa un nuevo proyecto Node.js:

```
npm init -y
```

### Instalación de dependencias:

Para este laboratorio, instalaremos las siguientes dependencias:

- bcrypt para el manejo seguro de contraseñas

```
npm install bcrypt
```

## Paso 2: Validación de entradas

### Creación del archivo de validación:

Crea un archivo llamado validation.js.

- Implementación de la función de validación de correo electrónico:
- Copia y pega el siguiente código en validation.js:

```
function validarCorreo(correo) {
  const patron = /^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$/;
  return patron.test(correo);
}

// Uso de la función
const correo = "test@example.com";
if (validarCorreo(correo)) {
  console.log("Correo válido");
} else {
  console.log("Correo inválido");
}
```

Ejecución del código:

En la terminal, ejecuta el archivo validation.js:

```
node validation.js
```

Deberías ver el mensaje “Correo válido” en la salida.

## Paso 3: Gestión de contraseñas

- Creación del archivo de gestión de contraseñas:

- Crea un archivo llamado password.js.

### Implementación de la función de hash de contraseña:

- Copia y pega el siguiente código en password.js:

```
const bcrypt = require('bcrypt');

async function hashPassword(password) {
  const saltRounds = 10;
  const hashedPassword = await bcrypt.hash(password, saltRounds);
  return hashedPassword;
}

async function main() {
  const password = "mySecurePassword";
  const hashedPassword = await hashPassword(password);
  console.log("Hashed Password:", hashedPassword);
}

main().catch(err => console.error(err));
```

### Ejecución del código:

- En la terminal, ejecuta el archivo password.js:

```
node password.js
```

Deberías ver la contraseña hashada impresa en la consola.

### Paso 4: Validación adicional

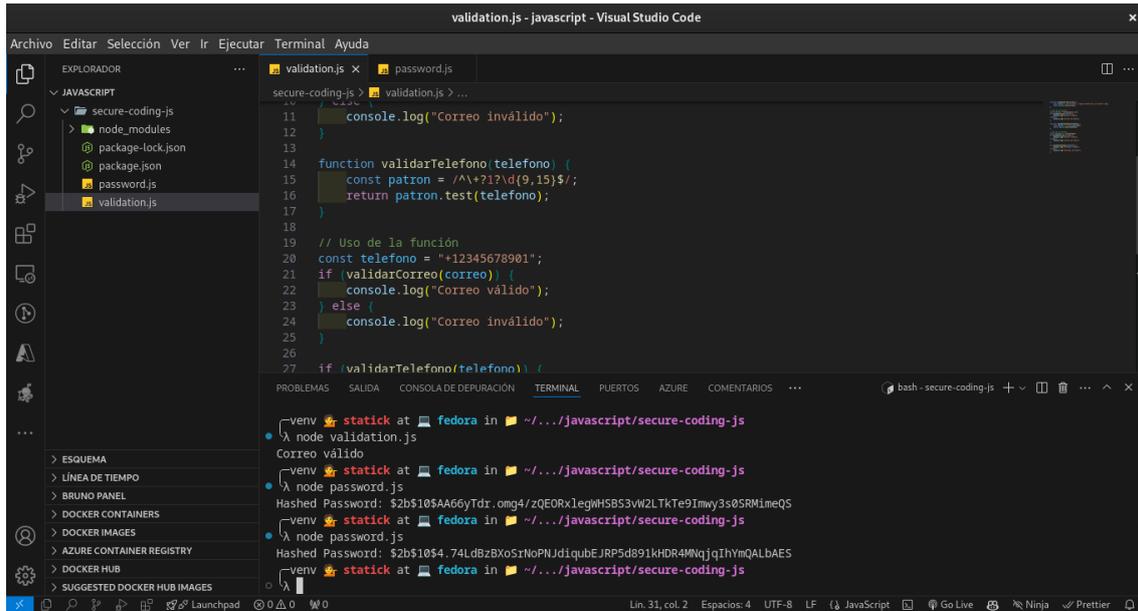
- Validación de números de teléfono:
- Agrega la siguiente función a validation.js para validar números de teléfono:

```
function validarTelefono(telefono) {
  const patron = /^+?1?\d{9,15}$/;
  return patron.test(telefono);
}

// Uso de la función
const telefono = "+12345678901";
if (validarTelefono(telefono)) {
  console.log("Correo válido");
} else {
  console.log("Correo inválido");
}
```

```
if (validarTelefono(telefono)) {
  console.log("Teléfono válido");
} else {
  console.log("Teléfono inválido");
}
```

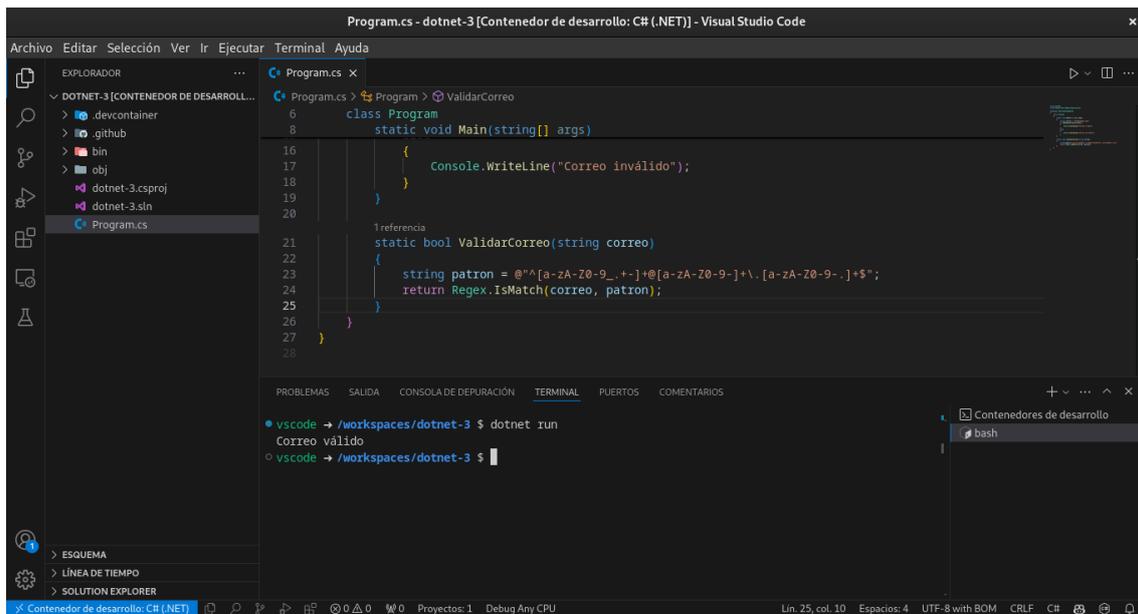
## Ejecución del código:



```
validation.js - javascript - Visual Studio Code
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
EXPLORADOR
JAVASCRIPT
secure-coding-js
node_modules
package-lock.json
package.json
password.js
validation.js
ESQUEMA
LÍNEA DE TIEMPO
BRUNO PANEL
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMAGES
validation.js x password.js
secure-coding-js > validation.js > ...
11 console.log("Correo inválido");
12 }
13
14 function validarTelefono(telefono) {
15   const patron = /^\+?1?\d{9,15}$/;
16   return patron.test(telefono);
17 }
18
19 // Uso de la función
20 const telefono = "+12345678901";
21 if (validarTelefono(telefono)) {
22   console.log("Teléfono válido");
23 } else {
24   console.log("Teléfono inválido");
25 }
26
27 if (validarTelefono(telefono)) {
28   console.log("Teléfono válido");
29 } else {
30   console.log("Teléfono inválido");
31 }
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS AZURE COMENTARIOS ...
bash - secure-coding-js
venv static at fedora in ~/.../javascript/secure-coding-js
λ node validation.js
Correo válido
venv static at fedora in ~/.../javascript/secure-coding-js
λ node password.js
Hashed Password: $2b$10$AA66yTdr.omg4/zQEORxLegWHSB53vW2LTkTe9Imwy3s05RMimeQS
venv static at fedora in ~/.../javascript/secure-coding-js
λ node password.js
Hashed Password: $2b$10$4.74LdBzBXoSiNoPNJdiqubEJRP5d891KHDR4MNqjgIhYmQALbAES
venv static at fedora in ~/.../javascript/secure-coding-js
λ
Lin. 31, col. 2 Espacios: 4 UTF-8 LF JavaScript Go Live Ninja Prettier
```

Ejecuta el archivo validation.js de nuevo para ver la validación de teléfono.

# 22 C#



## 22.1 Objetivo:

Aprender a implementar prácticas de codificación segura en C# mediante la validación de entradas y el manejo seguro de contraseñas.

## 22.2 Requisitos:

- .NET SDK instalado
- Visual Studio o Visual Studio Code

### Paso 1: Configuración del entorno

- Instalación de .NET SDK:
- Asegúrate de tener el .NET SDK instalado. Puedes descargarlo desde [dotnet.microsoft.com](https://dotnet.microsoft.com).

### Creación de un nuevo proyecto:

- En la terminal, crea una nueva carpeta para tu proyecto y navega a ella:

```
mkdir SecureCodingCSharp
cd SecureCodingCSharp
```

Inicialización de un nuevo proyecto de consola:

```
dotnet new console
```

## Paso 2: Validación de entradas

- Creación de la clase de validación:
- En el archivo Program.cs, implementa la siguiente función de validación de correo electrónico:

```
using System;
using System.Text.RegularExpressions;

namespace SecureCodingCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            string correo = "test@example.com";
            if (ValidarCorreo(correo))
            {
                Console.WriteLine("Correo válido");
            }
            else
            {
                Console.WriteLine("Correo inválido");
            }
        }

        static bool ValidarCorreo(string correo)
        {
            string patron = @"^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$";
            return Regex.IsMatch(correo, patron);
        }
    }
}
```

## Ejecución del código:

En la terminal, ejecuta el proyecto:

```
dotnet run
```

Deberías ver el mensaje “Correo válido” en la salida.

### Paso 3: Gestión de contraseñas

- Creación de la clase de gestión de contraseñas:
- Crea un nuevo archivo llamado PasswordUtils.cs y agrega el siguiente código:

```
using System;
using System.Security.Cryptography;

public class PasswordUtils
{
    public static string HashPassword(string password, byte[] salt)
    {
        using (var rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt, 10000))
        {
            return Convert.ToBase64String(rfc2898DeriveBytes.GetBytes(256));
        }
    }

    public static byte[] GenerateSalt()
    {
        byte[] salt = new byte[16];
        using (var rngCryptoServiceProvider = new RNGCryptoServiceProvider())
        {
            rngCryptoServiceProvider.GetBytes(salt);
        }
        return salt;
    }
}
```

### Uso de la función de hash de contraseña:

Modifica el archivo Program.cs para incluir el uso de la clase PasswordUtils:

```
using System;

namespace SecureCodingCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            string correo = "test@example.com";
            if (ValidarCorreo(correo))
            {
                Console.WriteLine("Correo válido");
            }
            else
        }
    }
}
```

```

        {
            Console.WriteLine("Correo inválido");
        }

        string password = "mySecurePassword";
        byte[] salt = PasswordUtils.GenerateSalt();
        string hashedPassword = PasswordUtils.HashPassword(password, salt);
        Console.WriteLine("Hashed Password: " + hashedPassword);
    }

    static bool ValidarCorreo(string correo)
    {
        string patron = @"^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+\.$";
        return Regex.IsMatch(correo, patron);
    }
}

```

### Ejecución del código:

En la terminal, ejecuta el proyecto de nuevo:

```
dotnet run
```

Deberías ver la contraseña hashada impresa en la consola.

### Paso 4: Validación adicional

#### Validación de números de teléfono:

Agrega la siguiente función a Program.cs para validar números de teléfono:

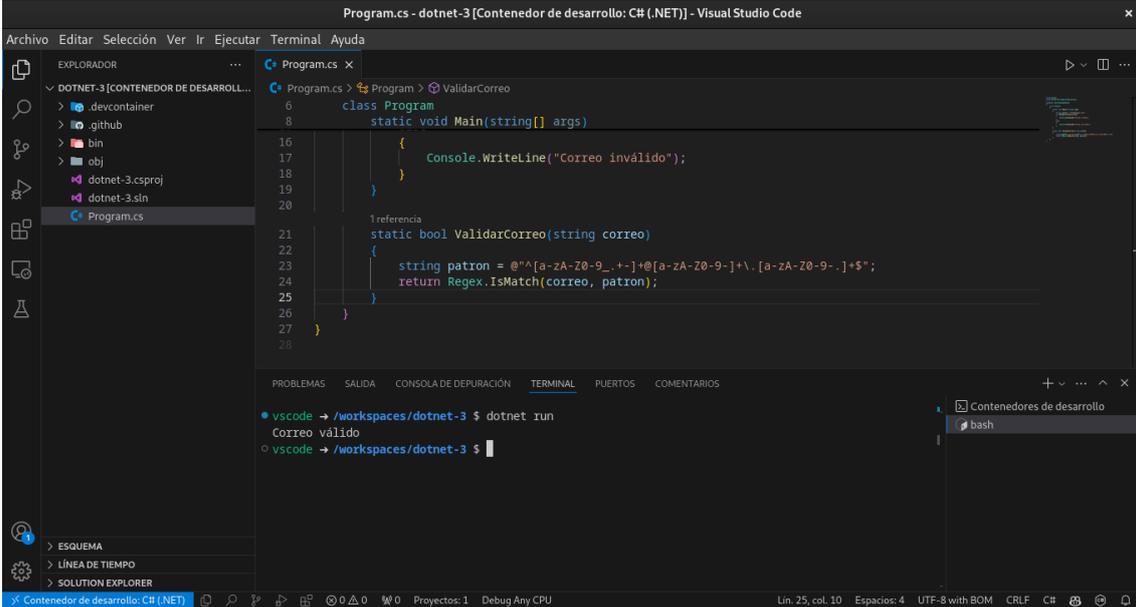
```

static bool ValidarTelefono(string telefono)
{
    string patron = @"^\+?1?\d{9,15}$";
    return Regex.IsMatch(telefono, patron);
}

// Uso de la función
string telefono = "+12345678901";
if (ValidarTelefono(telefono))
{
    Console.WriteLine("Teléfono válido");
}
else
{
    Console.WriteLine("Teléfono inválido");
}

```

Ejecución del código:

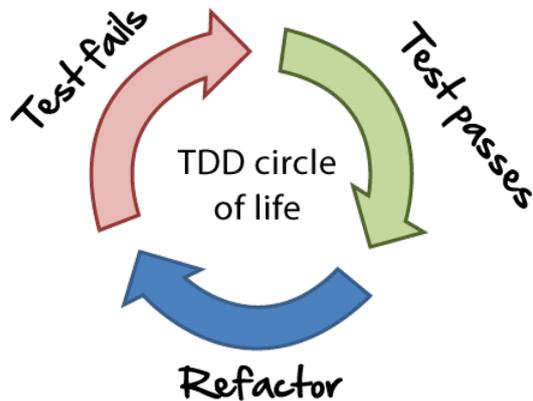


```
Program.cs - dotnet-3 [Contenedor de desarrollo: C# (.NET)] - Visual Studio Code
Archivo  Editar  Selección  Ver  Ir  Ejecutar  Terminal  Ayuda
EXPLORADOR
DOTNET-3 [CONTENEDOR DE DESARROLL...
  devcontainer
  github
  bin
  obj
  dotnet-3.csproj
  dotnet-3.sln
  Program.cs
Program.cs
6      class Program
7      {
8      static void Main(string[] args)
9      {
10         Console.WriteLine("Correo inválido");
11     }
12 }
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
1 referencia
static bool ValidarCorreo(string correo)
{
    string patron = @"^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$";
    return Regex.IsMatch(correo, patron);
}
}

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS  COMENTARIOS
vscode → /workspaces/dotnet-3 $ dotnet run
Correo válido
vscode → /workspaces/dotnet-3 $
```

Ejecuta el proyecto de nuevo para ver la validación de teléfono.

## 23 Laboratorio TDD



### 23.1 Objetivo

El objetivo de este laboratorio es que el estudiante pueda aplicar los conceptos de TDD (Test Driven Development) en la creación de una clase que permita realizar operaciones aritméticas básicas.

### 23.2 Desarrollo

1. Crear un proyecto de Java en Intelligent Idea llamado **Calculadora** y una clase llamada **Calculadora** con los siguientes métodos:

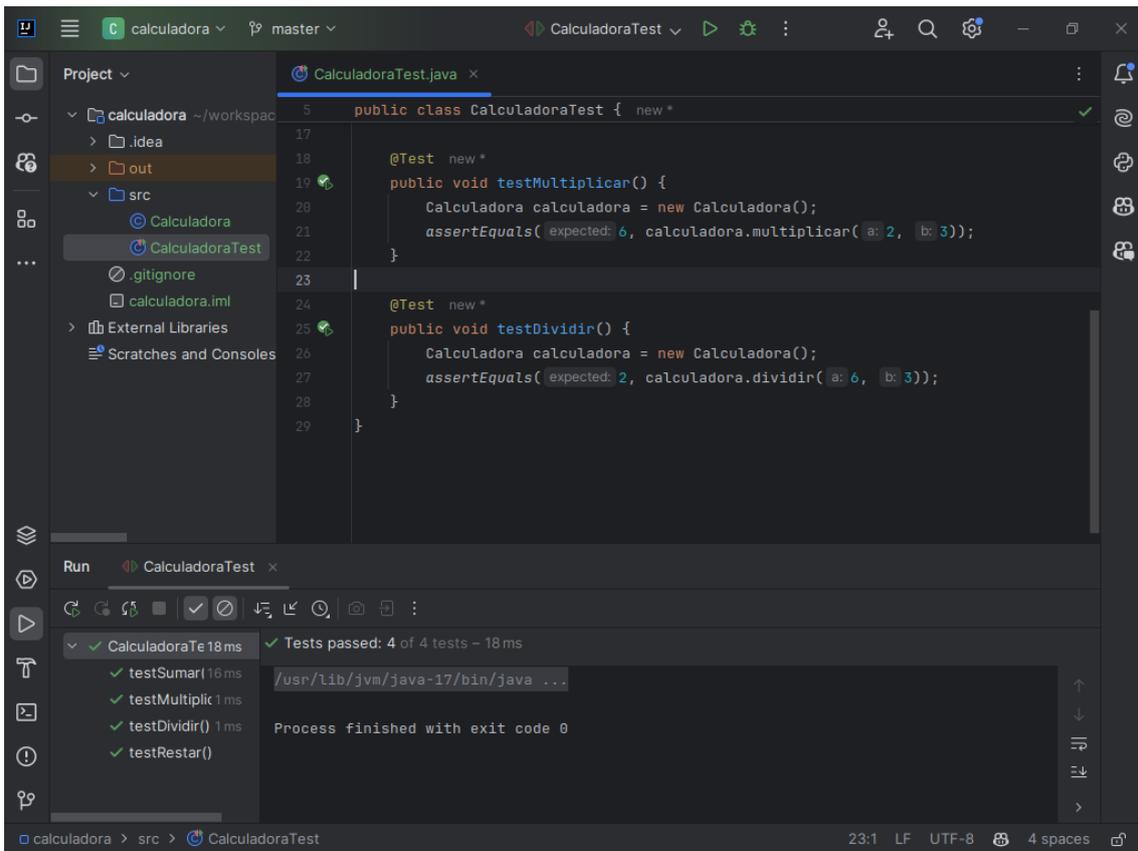
```
public class Calculadora {  
    public int sumar(int a, int b) {  
        return a + b;  
    }  
  
    public int restar(int a, int b) {  
        return a - b;  
    }  
  
    public int multiplicar(int a, int b) {  
        return a * b;  
    }  
}
```

```
public int dividir(int a, int b) {  
    return a / b;  
}  
}
```

2. Crear una clase llamada **CalculadoraTest** con los siguientes métodos:

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.assertEquals;  
  
public class CalculadoraTest {  
    @Test  
    public void testSumar() {  
        Calculadora calculadora = new Calculadora();  
        assertEquals(5, calculadora.sumar(2, 3));  
    }  
  
    @Test  
    public void testRestar() {  
        Calculadora calculadora = new Calculadora();  
        assertEquals(2, calculadora.restar(5, 3));  
    }  
  
    @Test  
    public void testMultiplicar() {  
        Calculadora calculadora = new Calculadora();  
        assertEquals(6, calculadora.multiplicar(2, 3));  
    }  
  
    @Test  
    public void testDividir() {  
        Calculadora calculadora = new Calculadora();  
        assertEquals(2, calculadora.dividir(6, 3));  
    }  
}
```

3. Ejecutar los tests y verificar que todos los tests pasen.



## 24 Reto

1. Crear un método llamado **potencia** en la clase **Calculadora** que reciba dos parámetros enteros y devuelva el resultado de elevar el primer parámetro a la potencia del segundo parámetro.
2. Crear un test llamado **testPotencia** en la clase **CalculadoraTest** que verifique que el método **potencia** funciona correctamente.
3. Ejecutar los tests y verificar que todos los tests pasen.

## 25 TDD con Python

1. Crear un proyecto de Python llamado **Calculadora** y un archivo llamado **calculadora.py** con los siguientes métodos:

```
class Calculadora:
    def sumar(self, a, b):
        return a + b

    def restar(self, a, b):
        return a - b

    def multiplicar(self, a, b):
        return a * b

    def dividir(self, a, b):
        return a / b
```

2. Crear un archivo llamado **test\_calculadora.py** con los siguientes métodos:

```
import unittest
from calculadora import Calculadora

class TestCalculadora(unittest.TestCase):
    def test_sumar(self):
        calculadora = Calculadora()
        self.assertEqual(5, calculadora.sumar(2, 3))

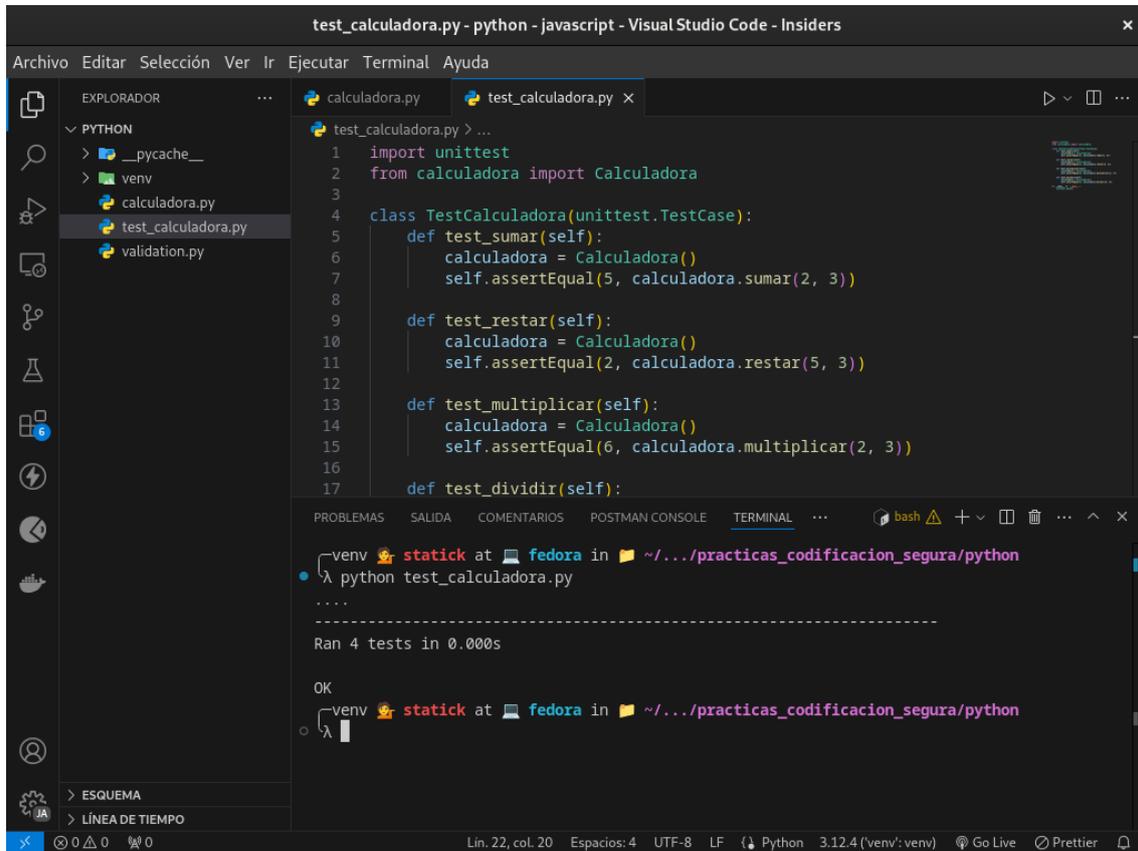
    def test_restar(self):
        calculadora = Calculadora()
        self.assertEqual(2, calculadora.restar(5, 3))

    def test_multiplicar(self):
        calculadora = Calculadora()
        self.assertEqual(6, calculadora.multiplicar(2, 3))

    def test_dividir(self):
        calculadora = Calculadora()
        self.assertEqual(2, calculadora.dividir(6, 3))

if __name__ == '__main__':
    unittest.main()
```

3. Ejecutar los tests y verificar que todos los tests pasen.



## 26 Reto

1. Crear un método llamado **potencia** en la clase **Calculadora** que reciba dos parámetros enteros y devuelva el resultado de elevar el primer parámetro a la potencia del segundo parámetro.
2. Crear un test llamado **test\_potencia** en la clase **TestCalculadora** que verifique que el método **potencia** funciona correctamente.
3. Ejecutar los tests y verificar que todos los tests pasen.

## 27 TDD con JavaScript

1. Creamos un proyecto con npm y ejecutamos el siguiente comando:

```
npm init -y
```

2. Instalamos Jest con el siguiente comando:

```
npm install --save-dev jest
```

3. Creamos un archivo llamado **calculadora.js** con los siguientes métodos:

```
class Calculadora {
  sumar(a, b) {
    return a + b;
  }

  restar(a, b) {
    return a - b;
  }

  multiplicar(a, b) {
    return a * b;
  }

  dividir(a, b) {
    return a / b;
  }
}

module.exports = Calculadora;
```

4. Creamos un archivo llamado **calculadora.test.js** con los siguientes métodos:

```
const Calculadora = require('./calculadora');

test('sumar 2 + 3 es igual a 5', () => {
  const calculadora = new Calculadora();
  expect(calculadora.sumar(2, 3)).toBe(5);
});

test('restar 5 - 3 es igual a 2', () => {
```

```

const calculadora = new Calculadora();
expect(calculadora.restar(5, 3)).toBe(2);
});

test('multiplicar 2 * 3 es igual a 6', () => {
  const calculadora = new Calculadora();
  expect(calculadora.multiplicar(2, 3)).toBe(6);
});

test('dividir 6 / 3 es igual a 2', () => {
  const calculadora = new Calculadora();
  expect(calculadora.dividir(6, 3)).toBe(2);
});

```

5. Agregamos el siguiente script en el archivo **package.json**:

```

"scripts": {
  "test": "jest"
}

```

6. Ejecutamos los tests con el siguiente comando:

```
npm test
```

The screenshot shows the Visual Studio Code editor with the `package.json` file open. The file content is as follows:

```

1  {
2    "name": "tdd",
3    "version": "1.0.0",
4    "main": "calculadora.js",
5    "scripts": {
6      "test": "jest"
7    },
8    "author": "",
9    "license": "ISC",
10   "description": "",
11   "devDependencies": {
12     "jest": "^29.7.0"
13   }

```

The terminal window shows the execution of `npm test` in the `~/../javascript/tdd` directory. The output is:

```

static at fedora in ~/../javascript/tdd
λ npm test

PASS ./calculadora.test.js
  ✓ sumar 2 + 3 es igual a 5 (2 ms)
  ✓ restar 5 - 3 es igual a 2 (1 ms)
  ✓ multiplicar 2 * 3 es igual a 6
  ✓ dividir 6 / 3 es igual a 2 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:  0 total
Time:        0.313 s
Ran all test suites.
static at fedora in ~/../javascript/tdd

```

## 28 Reto

1. Crear un método llamado **potencia** en la clase **Calculadora** que reciba dos parámetros enteros y devuelva el resultado de elevar el primer parámetro a la potencia del segundo parámetro.
2. Crear un test llamado **testPotencia** en el archivo **calculadora.test.js** que verifique que el método **potencia** funciona correctamente.
3. Ejecutar los tests y verificar que todos los tests pasen.

## 29 Conclusión

El TDD (Test Driven Development) es una técnica de desarrollo de software que consiste en escribir primero los tests y luego el código que hace que los tests pasen. Esta técnica nos permite escribir código de calidad y con menos errores.

En este laboratorio hemos aplicado los conceptos de TDD en la creación de una clase que permite realizar operaciones aritméticas básicas. Hemos creado los tests antes de escribir el código y hemos verificado que todos los tests pasen.

# 30 Laboratorio de Pruebas Unitarias



## 30.1 Objetivo

El objetivo de este laboratorio es que el estudiante pueda aplicar los conceptos de pruebas unitarias en la creación de una clase que permita realizar operaciones aritméticas básicas.

## 30.2 Desarrollo

```
1 import org.junit.jupiter.api.Assertions;
2 import org.junit.jupiter.api.Test;
3
4 import static org.junit.Assert.assertEquals;
5
6 public class PresupuestoTest { new *
7     @Test new *
8     public void testCalcularTotal() {
9         Presupuesto presupuesto = new Presupuesto();
10        Assertions.assertEquals(expected: 110.0, presupuesto.calcularTotal(subtotal: 100.0, impuesto: 10.0));
11    }
12
13
14    @Test new *
15    public void testCalcularImpuesto() {
16        Presupuesto presupuesto = new Presupuesto();
17        Assertions.assertEquals(expected: 10.0, presupuesto.calcularImpuesto(subtotal: 100.0, tasaImpuesto: 0.10));
18    }
19 }
```

Run PresupuestoTest x

PresupuestoTest	22 ms	Tests passed: 2 of 2 tests - 22 ms
testCalcularTotal()	21 ms	
testCalcularImpuesto()	1 ms	

Process finished with exit code 0

### 30.2.1 Java

1. Crear un proyecto de Java en Intelligent Idea llamado **Presupuesto** y una clase llamada **Presupuesto** con los siguientes métodos:

```
public class Presupuesto {
    public double calcularTotal(double subtotal, double impuesto) {
        return subtotal + impuesto;
    }

    public double calcularImpuesto(double subtotal, double tasaImpuesto) {
        return subtotal * tasaImpuesto;
    }
}
```

2. Crear una clase llamada **PresupuestoTest** con los siguientes métodos:

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

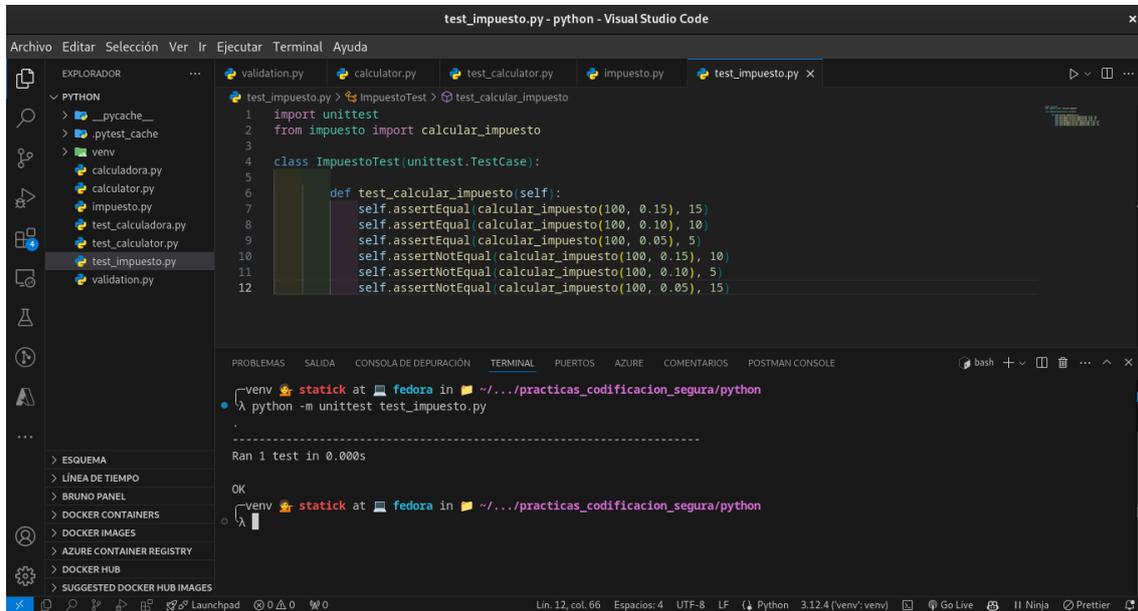
public class PresupuestoTest {
    @Test
    public void testCalcularTotal() {
        Presupuesto presupuesto = new Presupuesto();
        assertEquals(110.0, presupuesto.calcularTotal(100.0, 10.0));
    }

    @Test
    public void testCalcularImpuesto() {
        Presupuesto presupuesto = new Presupuesto();
        assertEquals(10.0, presupuesto.calcularImpuesto(100.0, 0.10));
    }
}
```

3. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.
4. Implementar el método **calcularDescuento** en la clase **Presupuesto** y agregar una prueba unitaria en la clase **PresupuestoTest** para verificar su correcto funcionamiento.
5. Implementar el método **calcularSubtotal** en la clase **Presupuesto** y agregar una prueba unitaria en la clase **PresupuestoTest** para verificar su correcto funcionamiento.
6. Implementar el método **calcularTotalConDescuento** en la clase **Presupuesto** y agregar una prueba unitaria en la clase **PresupuestoTest** para verificar su correcto funcionamiento.
7. Implementar el método **calcularTotalConImpuestoYDescuento** en la clase **Presupuesto** y agregar una prueba unitaria en la clase **PresupuestoTest** para verificar su correcto funcionamiento.

8. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.

## 30.2.2 Python



1. Crear un proyecto de Python llamado **Impuesto** que calcule el incremento del iva de 12% a 14% y un módulo llamado `calcular_impuesto` con las siguientes funciones:

```
def calcular_impuesto(subtotal, tasa_impuesto):
    return subtotal * tasa_impuesto
```

2. Crear un módulo llamado `test_impuesto` con las siguientes pruebas unitarias:

```
import unittest
from impuesto import calcular_impuesto

class ImpuestoTest(unittest.TestCase):

    def test_calcular_impuesto(self):
        self.assertEqual(calcular_impuesto(100, 0.15), 15)
        self.assertEqual(calcular_impuesto(100, 0.10), 10)
        self.assertEqual(calcular_impuesto(100, 0.05), 5)
        self.assertNotEqual(calcular_impuesto(100, 0.15), 10)
        self.assertNotEqual(calcular_impuesto(100, 0.10), 5)
        self.assertNotEqual(calcular_impuesto(100, 0.05), 15)
```

3. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.

```
python -m unittest test_impuesto.py
```

4. Implementar el método **calcular\_descuento** en el módulo **calcular\_impuesto** y agregar una prueba unitaria en el módulo **test\_impuesto** para verificar su correcto funcionamiento.
5. Implementar el método **calcular\_subtotal** en el módulo **calcular\_impuesto** y agregar una prueba unitaria en el módulo **test\_impuesto** para verificar su correcto funcionamiento.
6. Implementar el método **calcular\_total\_con\_descuento** en el módulo **calcular\_impuesto** y agregar una prueba unitaria en el módulo **test\_impuesto** para verificar su correcto funcionamiento.
7. Implementar el método **calcular\_total\_con\_impuesto\_y\_descuento** en el módulo **calcular\_impuesto** y agregar una prueba unitaria en el módulo **test\_impuesto** para verificar su correcto funcionamiento.
8. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.

### 30.2.3 JavaScript

The screenshot shows a Visual Studio Code editor window titled 'inventario.test.js - pruebas\_unitarias - Visual Studio Code'. The editor displays the following code in 'inventario.test.js':

```

1 const Inventario = require('./inventario');
2
3 test('agregar producto', () => {
4   const inventario = new Inventario();
5   inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
6   expect(inventario.listarProductos()).toEqual([
7     { codigo: 1, nombre: 'Producto 1', precio: 100 }
8   ]);
9
10  test('buscar producto', () => {
11    const inventario = new Inventario();
12    inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
13    expect(inventario.buscarProducto(1)).toEqual({
14      codigo: 1, nombre: 'Producto 1', precio: 100
15    });
16
17    test('actualizar producto', () => {
18      const inventario = new Inventario();
19      inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
20      inventario.actualizarProducto(1, { codigo: 1, nombre: 'Producto 2', precio: 200 });
21      expect(inventario.buscarProducto(1)).toEqual({
22        codigo: 1, nombre: 'Producto 2', precio: 200
23      });
24    });
25  });

```

Below the code, the 'TERMINAL' output shows the test results:

```

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:  0 total
Time:        0.527 s

```

The terminal also shows the command 'Run all test suites' and 'waiting for the debugger to disconnect...'. The status bar at the bottom indicates 'Lin. 24, col. 57' and 'Espacios: 4 UTF-8 LF JavaScript'.

En esta sección vamos a crear un proyecto llamado inventario que permita:

- Ingresar productos
  - Listar productos
  - Buscar productos
  - Actualizar productos
1. Crear un proyecto de Node.js llamado **Inventario** y un archivo llamado **inventario.js** con las siguientes funciones:

```

class Inventario {
  constructor() {
    this.productos = [];
  }

  agregarProducto(producto) {
    this.productos.push(producto);
  }

  listarProductos() {
    return this.productos;
  }

  buscarProducto(codigo) {
    return this.productos.find(producto => producto.codigo === codigo);
  }

  actualizarProducto(codigo, producto) {
    const index = this.productos.findIndex(producto => producto.codigo === codigo);
    this.productos[index] = producto;
  }
}

module.exports = Inventario;

```

2. Crear un archivo llamado **inventario.test.js** con las siguientes pruebas unitarias:

```

const Inventario = require('./inventario');

test('agregar producto', () => {
  const inventario = new Inventario();
  inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
  expect(inventario.listarProductos()).toEqual([{ codigo: 1, nombre: 'Producto 1', precio: 100 }]);
});

test('buscar producto', () => {
  const inventario = new Inventario();
  inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
  expect(inventario.buscarProducto(1)).toEqual({ codigo: 1, nombre: 'Producto 1', precio: 100 });
});

test('actualizar producto', () => {
  const inventario = new Inventario();
  inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
  inventario.actualizarProducto(1, { codigo: 1, nombre: 'Producto 2', precio: 200 });
  expect(inventario.buscarProducto(1)).toEqual({ codigo: 1, nombre: 'Producto 2', precio: 200 });
});

```

```

test('listar productos', () => {
  const inventario = new Inventario();
  inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
  inventario.agregarProducto({ codigo: 2, nombre: 'Producto 2', precio: 200 });
  expect(inventario.listarProductos()).toEqual([{ codigo: 1, nombre: 'Producto 1', precio: 100 }, { codigo: 2, nombre: 'Producto 2', precio: 200 }]);
});

test('buscar producto no existente', () => {
  const inventario = new Inventario();
  inventario.agregarProducto({ codigo: 1, nombre: 'Producto 1', precio: 100 });
  expect(inventario.buscarProducto(2)).toBeUndefined();
});

```

3. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.

```
npm test
```

4. Implementar el método **eliminarProducto** en la clase **Inventario** y agregar una prueba unitaria en la clase **InventarioTest** para verificar su correcto funcionamiento.
5. Implementar el método **categoriaProducto** en la clase **Inventario** y agregar una prueba unitaria en la clase **InventarioTest** para verificar su correcto funcionamiento.
6. Implementar el método **actualizarPrecioProducto** en la clase **Inventario** y agregar una prueba unitaria en la clase **InventarioTest** para verificar su correcto funcionamiento.
7. Implementar el método **actualizarNombreProducto** en la clase **Inventario** y agregar una prueba unitaria en la clase **InventarioTest** para verificar su correcto funcionamiento.
8. Ejecutar las pruebas unitarias y verificar que todas las pruebas pasen.

## **Part II**

# **Laboratorios de Pentesting**

## 31 Introducción

En esta serie de oratorios utilizaremos la academia de Burpsuit llamada PortSwigger para aprender a explotar vulnerabilidades comunes en aplicaciones web. PortSwigger es una plataforma de aprendizaje en línea que ofrece una amplia variedad de oratorios prácticos para aprender sobre seguridad web y pentesting.

## **32 ¿Qué me ofrece la academia PortSwigger?**

PortSwigger ofrece una amplia variedad de oratorios prácticos para aprender sobre seguridad web y pentesting. Los oratorios están organizados en diferentes categorías, como inyección SQL, cross-site scripting (XSS), cross-site request forgery (CSRF), vulnerabilidades de seguridad en aplicaciones móviles, entre otros.

## 33 ¿Cómo se divide la academia PortSwigger?

La academia PortSwigger se divide en diferentes secciones, como oratorios, retos, guías, y cursos. Los oratorios son ejercicios prácticos que permiten aprender sobre vulnerabilidades comunes en aplicaciones web. Los retos son ejercicios más avanzados que requieren aplicar conocimientos previos para resolverlos. Las guías son tutoriales que explican cómo explotar diferentes vulnerabilidades en aplicaciones web. Los cursos son lecciones en línea que cubren diferentes temas de seguridad web y pentesting.

A continuación se muestra cada una de las categorías:

- SQL Injection
  - SQL injection vulnerability in WHERE clause allowing retrieval of hidden data
  - SQL injection vulnerability allowing login bypass
  - SQL injection attack, querying the database type and version on Oracle
  - SQL injection attack, querying the database type and version on MySQL and Microsoft
  - SQL injection attack, listing the database contents on non-Oracle databases
  - SQL injection attack, listing the database contents on Oracle
  - SQL injection UNION attack, determining the number of columns returned by the query
  - SQL injection UNION attack, finding a column containing text
  - SQL injection UNION attack, retrieving data from other tables
  - SQL injection UNION attack, retrieving multiple values in a single column
  - Blind SQL injection with conditional responses
  - Blind SQL injection with conditional errors
  - Visible error-based SQL injection
  - Blind SQL injection with time delays
  - Blind SQL injection with time delays and information retrieval
  - Blind SQL injection with out-of-band interaction
  - Blind SQL injection with out-of-band data exfiltration
  - SQL injection with filter bypass via XML encoding
- Cross-site scripting
  - Reflected XSS into HTML context with nothing encoded
  - Stored XSS into HTML context with nothing encoded
  - DOM XSS in document.write sink using source location.search
  - DOM XSS in innerHTML sink using source location.search
  - DOM XSS in jQuery anchor href attribute sink using location.search source
  - DOM XSS in jQuery selector sink using a hashchange event

- Reflected XSS into attribute with angle brackets HTML-encoded
  - Stored XSS into anchor href attribute with double quotes HTML-encoded
  - Reflected XSS into a JavaScript string with angle brackets HTML encoded
  - DOM XSS in document.write sink using source location.search inside a select element
  - DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded
  - Reflected DOM XSS
  - Stored DOM XSS
  - Reflected XSS into HTML context with most tags and attributes blocked
  - Reflected XSS into HTML context with all tags blocked except custom ones
  - Reflected XSS with some SVG markup allowed
  - Reflected XSS in canonical link tag
  - Reflected XSS into a JavaScript string with single quote and backslash escaped
  - Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped
  - Stored XSS into onclick event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped
  - Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped
  - Exploiting cross-site scripting to steal cookies
  - Exploiting cross-site scripting to capture passwords
  - Exploiting XSS to perform CSRF
  - Reflected XSS with AngularJS sandbox escape without strings
  - Reflected XSS with AngularJS sandbox escape and CSP
  - Reflected XSS with event handlers and href attributes blocked
  - Reflected XSS in a JavaScript URL with some characters blocked
  - Reflected XSS protected by very strict CSP, with dangling markup attack
  - Reflected XSS protected by CSP, with CSP bypass
- Cross-site request forgery (CSRF)
    - CSRF vulnerability with no defenses
    - CSRF where token validation depends on request method
    - CSRF where token validation depends on token being present
    - CSRF where token is not tied to user session
    - CSRF where token is tied to non-session cookie
    - CSRF where token is duplicated in cookie
    - SameSite Lax bypass via method override
    - SameSite Strict bypass via client-side redirect
    - SameSite Strict bypass via sibling domain
    - SameSite Lax bypass via cookie refresh
    - CSRF where Referer validation depends on header being present
    - CSRF with broken Referer validation
- Clickjacking
    - Basic clickjacking with CSRF token protection
    - Clickjacking with form input data prefilled from a URL parameter

- Clickjacking with a frame buster script
- Exploiting clickjacking vulnerability to trigger DOM-based XSS
- Multistep clickjacking
- DOM-based vulnerabilities
  - DOM XSS using web messages
  - DOM XSS using web messages and a JavaScript URL
  - DOM XSS using web messages and JSON.parse
  - DOM-based open redirection
  - DOM-based cookie manipulation
  - Exploiting DOM clobbering to enable XSS
  - Clobbering DOM attributes to bypass HTML filters
- Cross-origin resource sharing (CORS)
  - CORS vulnerability with basic origin reflection
  - CORS vulnerability with trusted null origin
  - CORS vulnerability with trusted insecure protocols
- XML external entity (XXE) injection
  - Exploiting XXE using external entities to retrieve files
  - Exploiting XXE to perform SSRF attacks
  - Blind XXE with out-of-band interaction
  - Blind XXE with out-of-band interaction via XML parameter entities
  - Exploiting blind XXE to exfiltrate data using a malicious external DTD
  - Exploiting blind XXE to retrieve data via error messages
  - Exploiting XInclude to retrieve files
  - Exploiting XXE via image file upload
  - Exploiting XXE to retrieve data by repurposing a local DTD
- Server-side request forgery (SSRF)
  - Basic SSRF against the local server
  - Basic SSRF against another back-end system
  - Blind SSRF with out-of-band detection
  - SSRF with blacklist-based input filter
  - SSRF with filter bypass via open redirection vulnerability
  - Blind SSRF with Shellshock exploitation
  - SSRF with whitelist-based input filter
- HTTP request smuggling
  - HTTP request smuggling, confirming a CL.TE vulnerability via differential responses
  - HTTP request smuggling, confirming a TE.CL vulnerability via differential responses
  - Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability

- Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability
  - Exploiting HTTP request smuggling to reveal front-end request rewriting
  - Exploiting HTTP request smuggling to capture other users' requests
  - Exploiting HTTP request smuggling to deliver reflected XSS
  - Response queue poisoning via H2.TE request smuggling
  - H2.CL request smuggling
  - HTTP/2 request smuggling via CRLF injection
  - HTTP/2 request splitting via CRLF injection
  - CL.0 request smuggling
  - HTTP request smuggling, basic CL.TE vulnerability
  - HTTP request smuggling, basic TE.CL vulnerability
  - HTTP request smuggling, obfuscating the TE header
  - Exploiting HTTP request smuggling to perform web cache poisoning
  - Exploiting HTTP request smuggling to perform web cache deception
  - Bypassing access controls via HTTP/2 request tunnelling
  - Web cache poisoning via HTTP/2 request tunnelling
  - Client-side desync
  - Server-side pause-based request smuggling
- OS command injection
    - OS command injection, simple case
    - Blind OS command injection with time delays
    - Blind OS command injection with output redirection
    - Blind OS command injection with out-of-band interaction
    - Blind OS command injection with out-of-band data exfiltration
- Server-side template injection
    - Basic server-side template injection
    - Basic server-side template injection (code context)
    - Server-side template injection using documentation
    - Server-side template injection in an unknown language with a documented exploit
    - Server-side template injection with information disclosure via user-supplied objects
    - Server-side template injection in a sandboxed environment
    - Server-side template injection with a custom exploit
- Path traversal
    - File path traversal, simple case
    - File path traversal, traversal sequences blocked with absolute path bypass
    - File path traversal, traversal sequences stripped non-recursively
    - File path traversal, traversal sequences stripped with superfluous URL-decode
    - File path traversal, validation of start of path
    - File path traversal, validation of file extension with null byte bypass
- Access control vulnerabilities

- Unprotected admin functionality
  - Unprotected admin functionality with unpredictable URL
  - User role controlled by request parameter
  - User role can be modified in user profile
  - User ID controlled by request parameter
  - User ID controlled by request parameter, with unpredictable user IDs
  - User ID controlled by request parameter with data leakage in redirect
  - User ID controlled by request parameter with password disclosure
  - Insecure direct object references
  - URL-based access control can be circumvented
  - Method-based access control can be circumvented
  - Multi-step process with no access control on one step
  - Referer-based access control
- Authentication
    - Username enumeration via different responses
    - 2FA simple bypass
    - Password reset broken logic
    - Username enumeration via subtly different responses
    - Username enumeration via response timing
    - Broken brute-force protection, IP block
    - Username enumeration via account lock
    - 2FA broken logic
    - Brute-forcing a stay-logged-in cookie
    - Offline password cracking
    - Password reset poisoning via middleware
    - Password brute-force via password change
    - Broken brute-force protection, multiple credentials per request
    - 2FA bypass using a brute-force attack
- WebSockets
    - Manipulating WebSocket messages to exploit vulnerabilities
    - Cross-site WebSocket hijacking
    - Manipulating the WebSocket handshake to exploit vulnerabilities
- Web cache poisoning
    - Web cache poisoning with an unkeyed header
    - Web cache poisoning with an unkeyed cookie
    - Web cache poisoning with multiple headers
    - Targeted web cache poisoning using an unknown header
    - Web cache poisoning via an unkeyed query string
    - Web cache poisoning via an unkeyed query parameter
    - Parameter cloaking
    - Web cache poisoning via a fat GET request
    - URL normalization
    - Web cache poisoning to exploit a DOM vulnerability via a cache with strict cacheability criteria

- Combining web cache poisoning vulnerabilities
  - Cache key injection
  - Internal cache poisoning
- Insecure deserialization
  - Modifying serialized objects
  - Modifying serialized data types
  - Using application functionality to exploit insecure deserialization
  - Arbitrary object injection in PHP
  - Exploiting Java deserialization with Apache Commons
  - Exploiting PHP deserialization with a pre-built gadget chain
  - Exploiting Ruby deserialization using a documented gadget chain
  - Developing a custom gadget chain for Java deserialization
  - Developing a custom gadget chain for PHP deserialization
  - Using PHAR deserialization to deploy a custom gadget chain
- Information disclosure
  - Information disclosure in error messages
  - Information disclosure on debug page
  - Source code disclosure via backup files
  - Authentication bypass via information disclosure
  - Information disclosure in version control history
- Business logic vulnerabilities
  - Excessive trust in client-side controls
  - High-level logic vulnerability
  - Inconsistent security controls
  - Flawed enforcement of business rules
  - Low-level logic flaw
  - Inconsistent handling of exceptional input
  - Weak isolation on dual-use endpoint
  - Insufficient workflow validation
  - Authentication bypass via flawed state machine
  - Infinite money logic flaw
  - Authentication bypass via encryption oracle
  - Bypassing access controls using email address parsing discrepancies
- HTTP Host header attacks
  - Basic password reset poisoning
  - Host header authentication bypass
  - Web cache poisoning via ambiguous requests
  - Routing-based SSRF
  - SSRF via flawed request parsing
  - Host validation bypass via connection state attack
  - Password reset poisoning via dangling markup
- OAuth authentication

- Authentication bypass via OAuth implicit flow
- SSRF via OpenID dynamic client registration
- Forced OAuth profile linking
- OAuth account hijacking via redirect\_uri
- Stealing OAuth access tokens via an open redirect
- Stealing OAuth access tokens via a proxy page
- File upload vulnerabilities
  - Remote code execution via web shell upload
  - Web shell upload via Content-Type restriction bypass
  - Web shell upload via path traversal
  - Web shell upload via extension blacklist bypass
  - Web shell upload via obfuscated file extension
  - Remote code execution via polyglot web shell upload
  - Web shell upload via race condition
- JWT
  - JWT authentication bypass via unverified signature
  - JWT authentication bypass via flawed signature verification
  - JWT authentication bypass via weak signing key
  - JWT authentication bypass via jwk header injection
  - JWT authentication bypass via jku header injection
  - JWT authentication bypass via kid header path traversal
  - JWT authentication bypass via algorithm confusion
  - JWT authentication bypass via algorithm confusion with no exposed key
- Essential skills
  - Discovering vulnerabilities quickly with targeted scanning
  - Scanning non-standard data structures
- Prototype pollution
  - Client-side prototype pollution via browser APIs
  - DOM XSS via client-side prototype pollution
  - DOM XSS via an alternative prototype pollution vector
  - Client-side prototype pollution via flawed sanitization
  - Client-side prototype pollution in third-party libraries
  - Privilege escalation via server-side prototype pollution
  - Detecting server-side prototype pollution without polluted property reflection
  - Bypassing flawed input filters for server-side prototype pollution
  - Remote code execution via server-side prototype pollution
  - Exfiltrating sensitive data via server-side prototype pollution
- GraphQL API vulnerabilities
  - Accessing private GraphQL posts
  - Accidental exposure of private GraphQL fields
  - Finding a hidden GraphQL endpoint

- Bypassing GraphQL brute force protections
- Performing CSRF exploits over GraphQL
- Race conditions
  - Limit overrun race conditions
  - Bypassing rate limits via race conditions
  - Multi-endpoint race conditions
  - Single-endpoint race conditions
  - Exploiting time-sensitive vulnerabilities
  - Partial construction race conditions
- NoSQL injection
  - Detecting NoSQL injection
  - Exploiting NoSQL operator injection to bypass authentication
  - Exploiting NoSQL injection to extract data
  - Exploiting NoSQL operator injection to extract unknown fields
- API testing
  - Exploiting an API endpoint using documentation
  - Exploiting server-side parameter pollution in a query string
  - Finding and exploiting an unused API endpoint
  - Exploiting a mass assignment vulnerability
  - Exploiting server-side parameter pollution in a REST URL
- Web LLM attacks
  - Exploiting LLM APIs with excessive agency
  - Exploiting vulnerabilities in LLM APIs
  - Indirect prompt injection
  - Exploiting insecure output handling in LLMs
- Web cache deception
  - Exploiting path mapping for web cache deception
  - Exploiting path delimiters for web cache deception
  - Exploiting origin server normalization for web cache deception
  - Exploiting cache server normalization for web cache deception
  - Exploiting exact-match cache rules for web cache deception

Como podemos analizar son muchas las categorías que nos ofrece la academia PortSwigger, por lo que en esta serie de laboratorios nos enfocaremos en las vulnerabilidades más comunes en aplicaciones web, como inyección SQL, cross-site scripting (XSS), cross-site request forgery (CSRF), entre otros.

## **34 Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data**

## 35 Objetivo

El objetivo de este laboratorio es que el estudiante pueda identificar y explotar una vulnerabilidad de inyección SQL en una cláusula WHERE que permite recuperar datos ocultos.

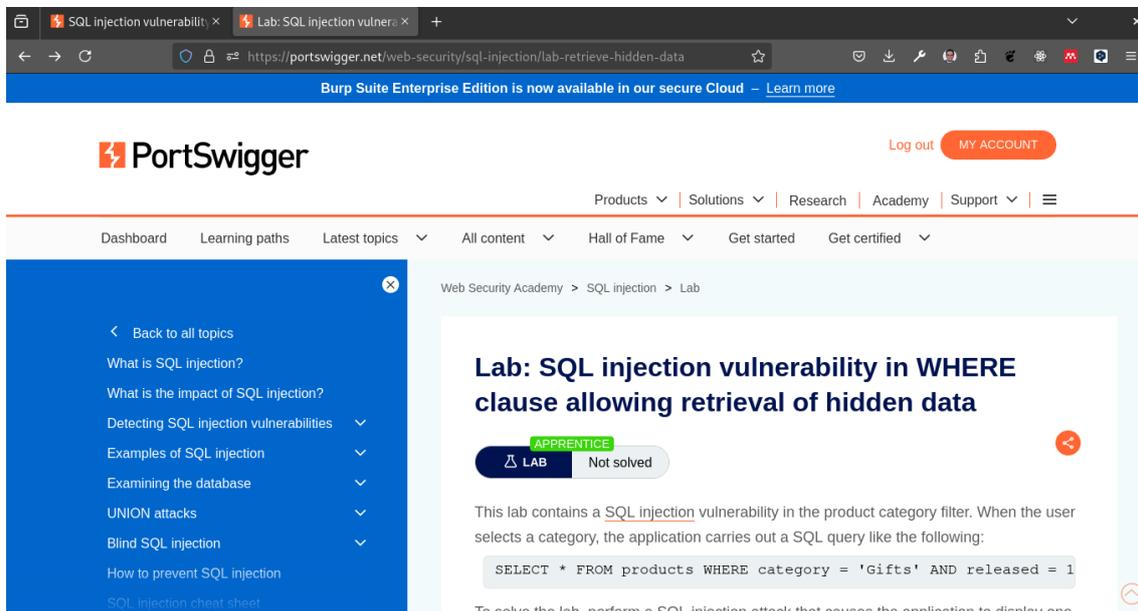
## 36 Herramientas

Para este laboratorio se utilizará el siguiente software:

- Kali Linux
- Burp Suite

## 37 Desarrollo

1. Arrancar la máquina virtual con Kali Linux y abrir Burp Suite. Todo el tráfico de red se redirigirá a través de Burp Suite.
2. Abierto Burp Suite, ir a la pestaña **Proxy** y asegurarse de que el **Intercept is on**.
3. Abrimos el navegador que intercepta el tráfico y navegamos a la URL del laboratorio.

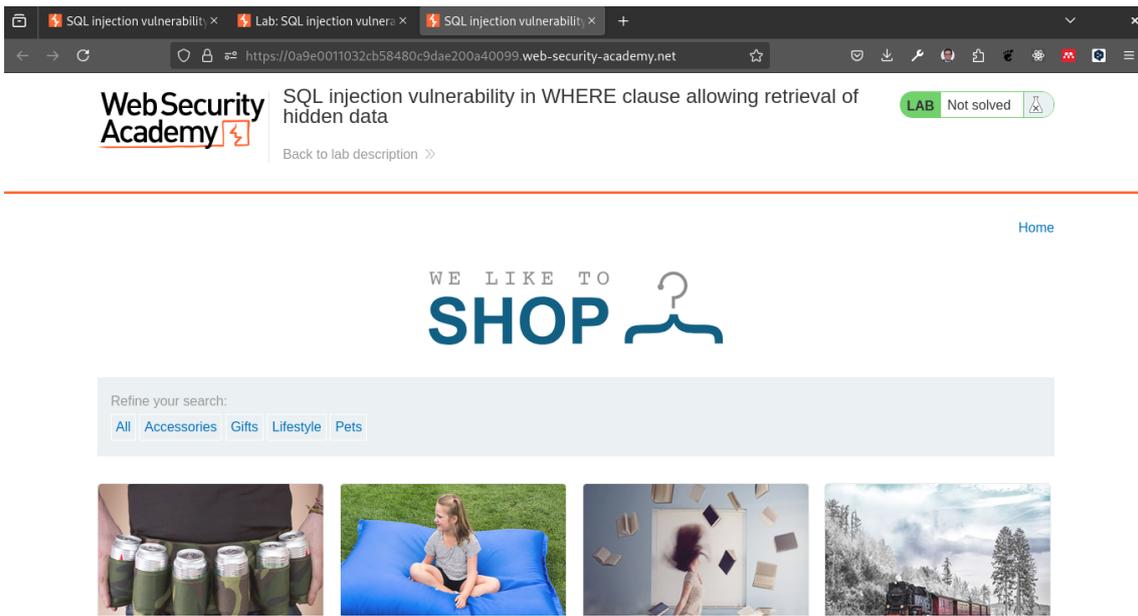


The screenshot shows a web browser window with two tabs: 'SQL injection vulnerabili...' and 'Lab: SQL injection vulnera...'. The address bar shows the URL 'https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data'. Below the browser, the PortSwigger website interface is visible. The top navigation bar includes 'Log out' and 'MY ACCOUNT'. The main navigation menu has 'Products', 'Solutions', 'Research', 'Academy', and 'Support'. The 'Academy' section is expanded, showing 'Dashboard', 'Learning paths', 'Latest topics', 'All content', 'Hall of Fame', 'Get started', and 'Get certified'. The 'Latest topics' menu is open, listing various SQL injection topics. The main content area displays the lab title 'Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data', which is categorized as 'APPRENTICE' and 'LAB'. The lab status is 'Not solved'. The description states: 'This lab contains a SQL injection vulnerability in the product category filter. When the user selects a category, the application carries out a SQL query like the following:'. A code block shows the SQL query: 

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

. Below the code, it says: 'To solve the lab, perform a SQL injection attack that causes the application to display one'.

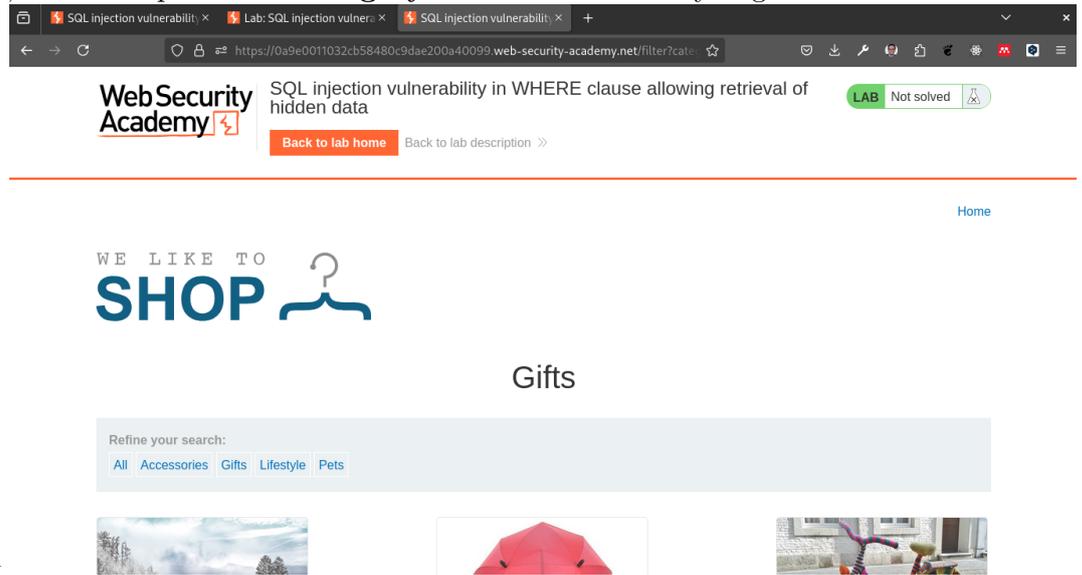
4. Iniciar sesión con las credenciales proporcionadas.



En la página principal, haga clic en **Listado de productos**.

5. En la página de **Listado de productos**, haga clic en **Gifts**.

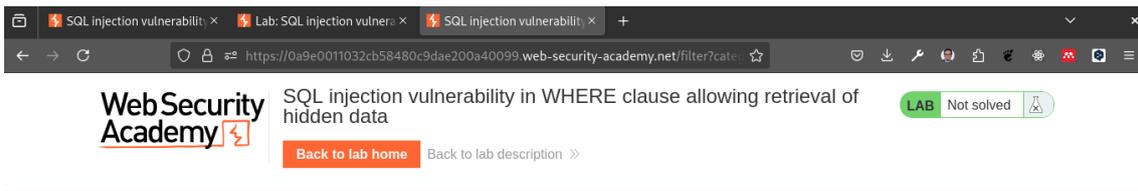
6. En la URL, cambie el parámetro **category** de **Gifts** a **Gifts'**– y haga clic en **Go**.



::: {.center}

7. La aplicación muestra ahora 3 productos, probamos la inyección SQL en el campo **category** y la aplicación nos muestra todos los productos.

`https://0a9e0011032cb58480c9dae200a40099.web-security-academy.net/filter?category=Gifts'`



Gifts'--

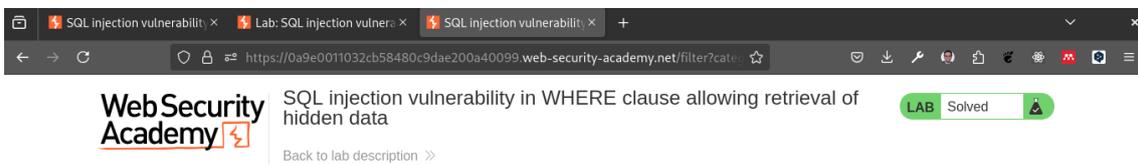
Refine your search:

All Accessories Gifts Lifestyle Pets



8. En la URL, cambie el parámetro **category** de **Gifts'--** a **Gifts' OR '1'='1** y haga clic en **Go**.

`https://0a9e0011032cb58480c9dae200a40099.web-security-academy.net/filter?category=Gifts'`



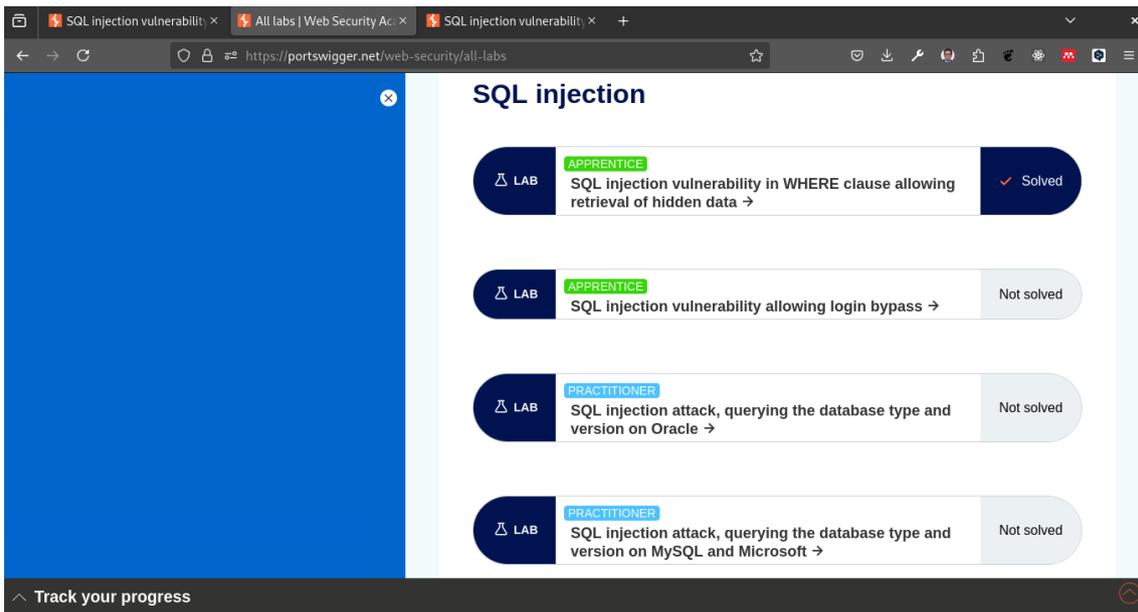
Gifts' OR '1'='1

Refine your search:

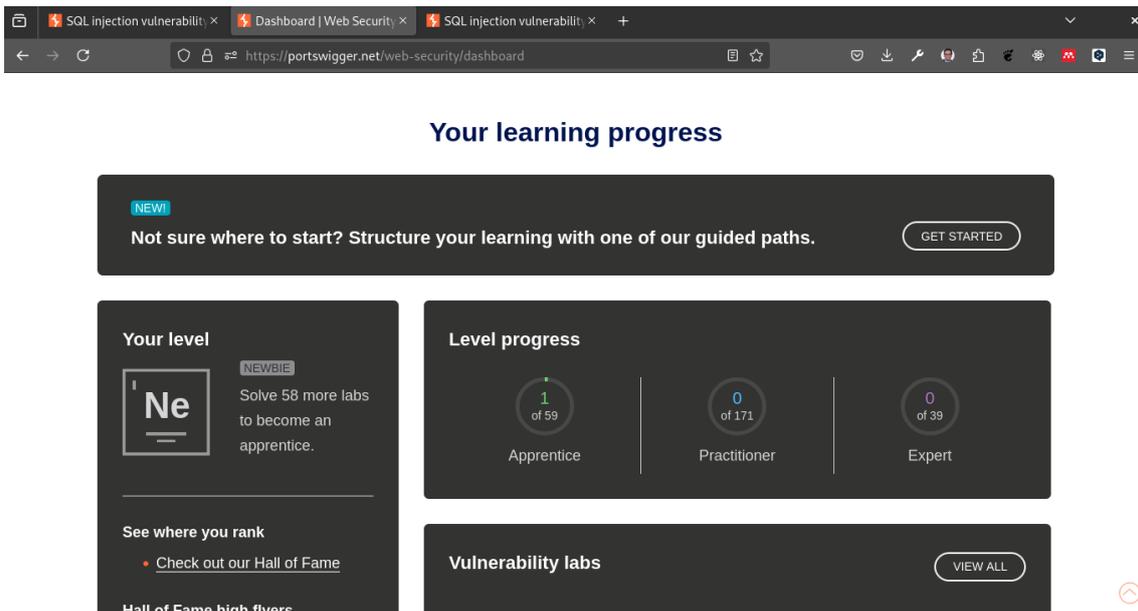
All Accessories Gifts Lifestyle Pets



9. La aplicación muestra ahora todos los productos, lo que indica que la inyección SQL en la cláusula **WHERE** es exitosa.
10. En todos los laboratorios ahora se puede observar que el laboratorio **SQL injection vulnerability in WHERE clause allowing retrieval of hidden data** aparece como **Solved**.



- En la sección Academy, haga clic en **Progress** y verifique que el laboratorio **SQL injection vulnerability in WHERE clause allowing retrieval of hidden data** esté marcado como **Solved**.



Con esto se ha completado el laboratorio **SQL injection vulnerability in WHERE clause allowing retrieval of hidden data**.

## 38 Conclusión

En este laboratorio, el estudiante ha aprendido a identificar y explotar una vulnerabilidad de inyección SQL en una cláusula WHERE que permite recuperar datos ocultos.

El estudiante ha utilizado Burp Suite para interceptar el tráfico de red y modificar los parámetros de la URL para realizar la inyección SQL.

El estudiante ha verificado que la inyección SQL es exitosa al observar que la aplicación muestra todos los productos en lugar de los productos de la categoría seleccionada.

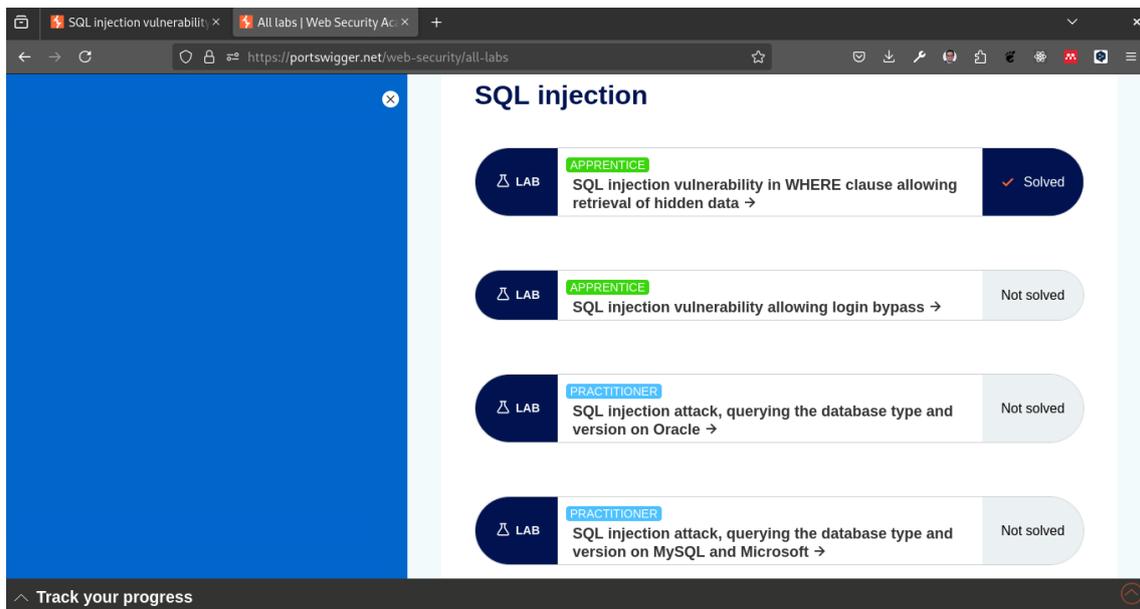
El estudiante ha completado el laboratorio con éxito y ha verificado su estado en la sección de Progreso de la Academia.

# 39 Lab 2: SQL injection vulnerability allowing login bypass

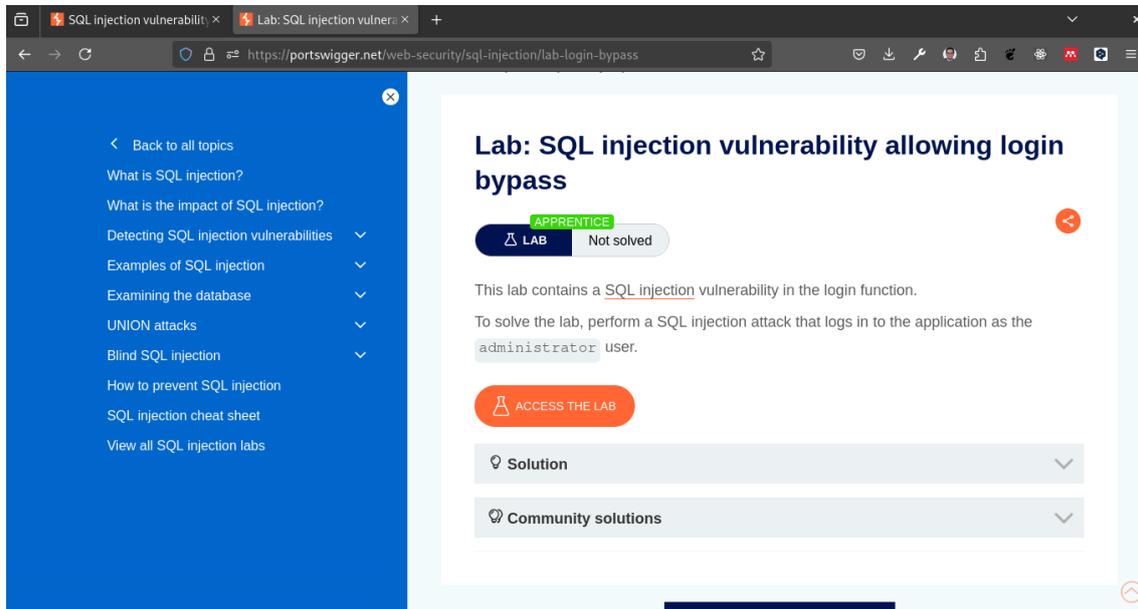
## 39.1 Objetivo

Este laboratorio demuestra cómo un atacante puede explotar una vulnerabilidad de inyección SQL en una consulta de autenticación para iniciar sesión como el primer usuario en la base de datos, incluso si la contraseña es desconocida.

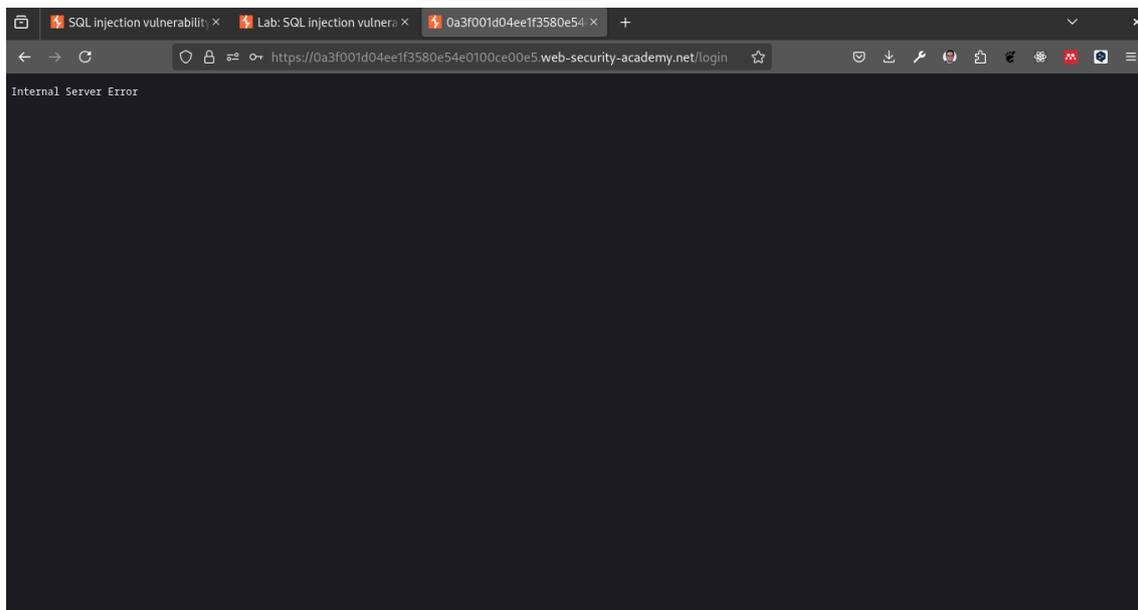
Para resolver el laboratorio, inicie sesión como el usuario **administrator**.



## 39.2 Desarrollo

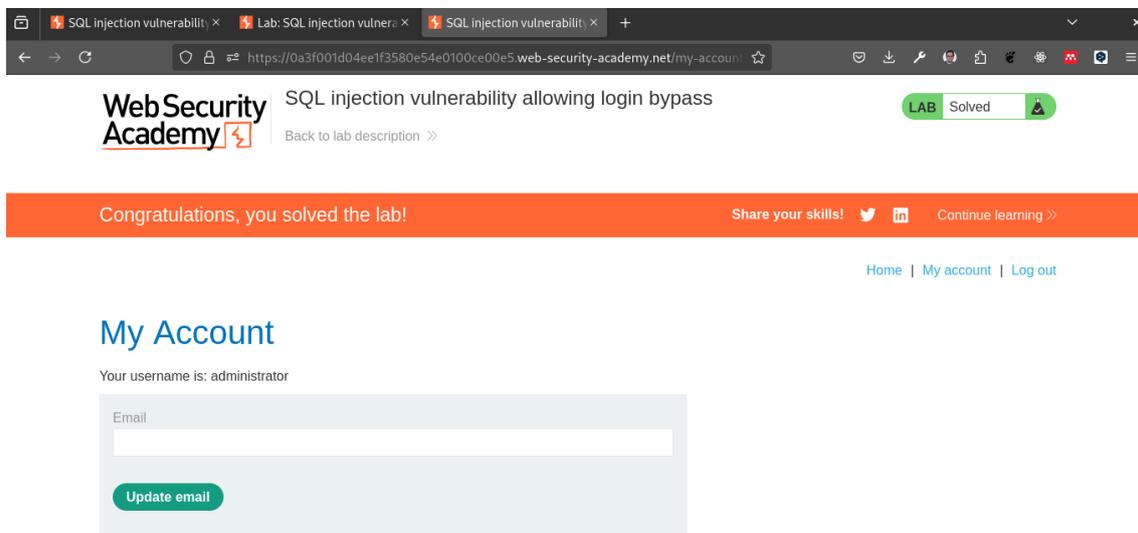


1. Iniciar sesión en la aplicación web, navegar a la página de inicio y hacer clic en **Login**.
2. Debemos iniciar sesión como **administrator**. Sin embargo no conocemos la contraseña. Intente iniciar sesión con probando como usuario **administrator** y contraseña `'`. Si esto nos muestra un error de SQL, entonces podemos asumir que la aplicación es vulnerable a inyección SQL.

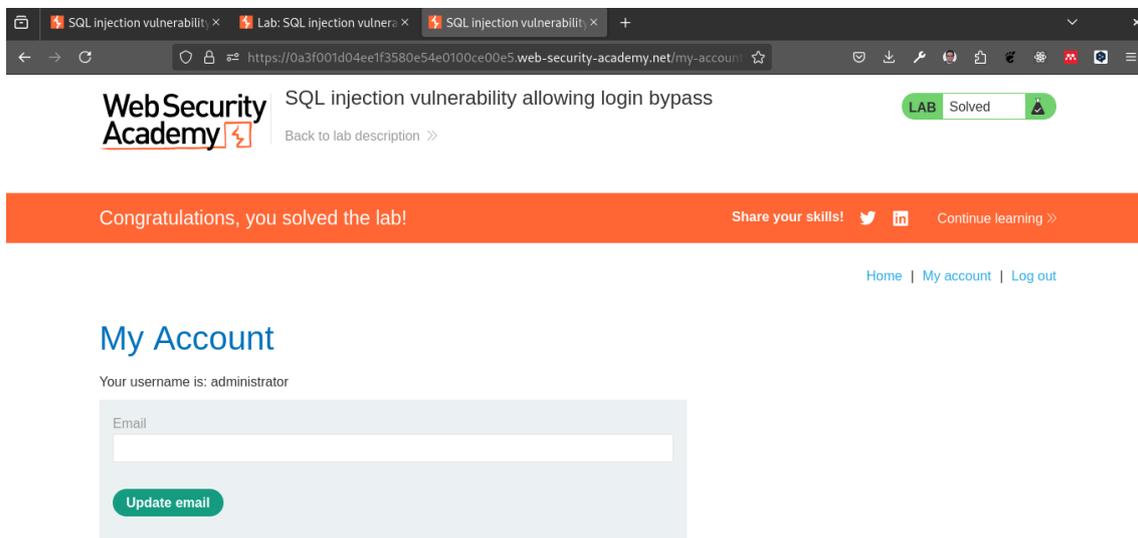


3. Ingrese el siguiente texto en el campo **Password** y haga clic en **Login**:

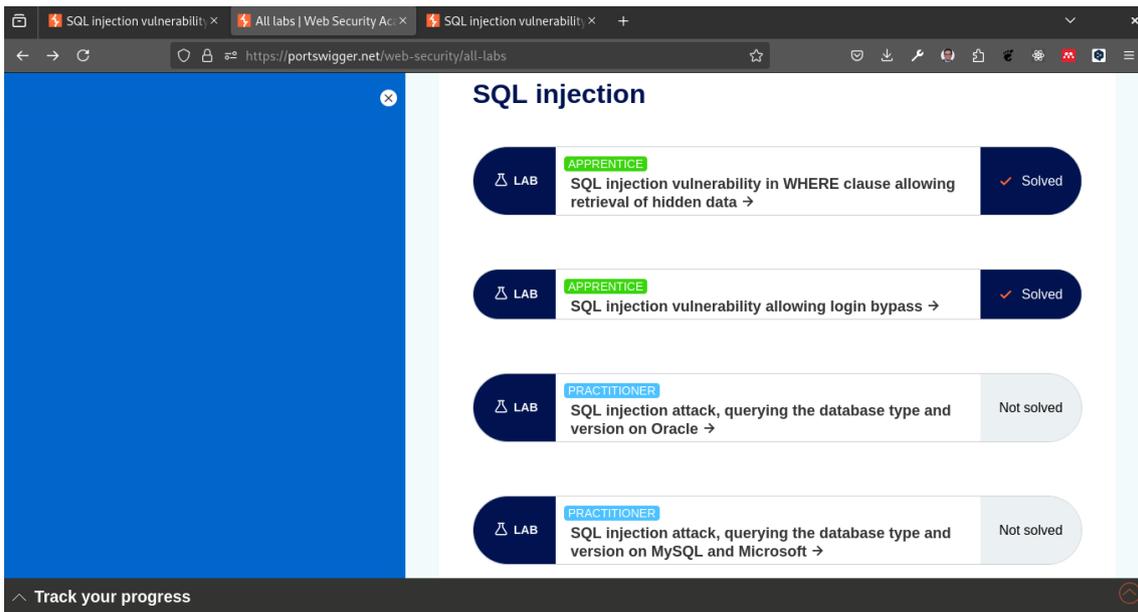
```
' OR '1'='1
```



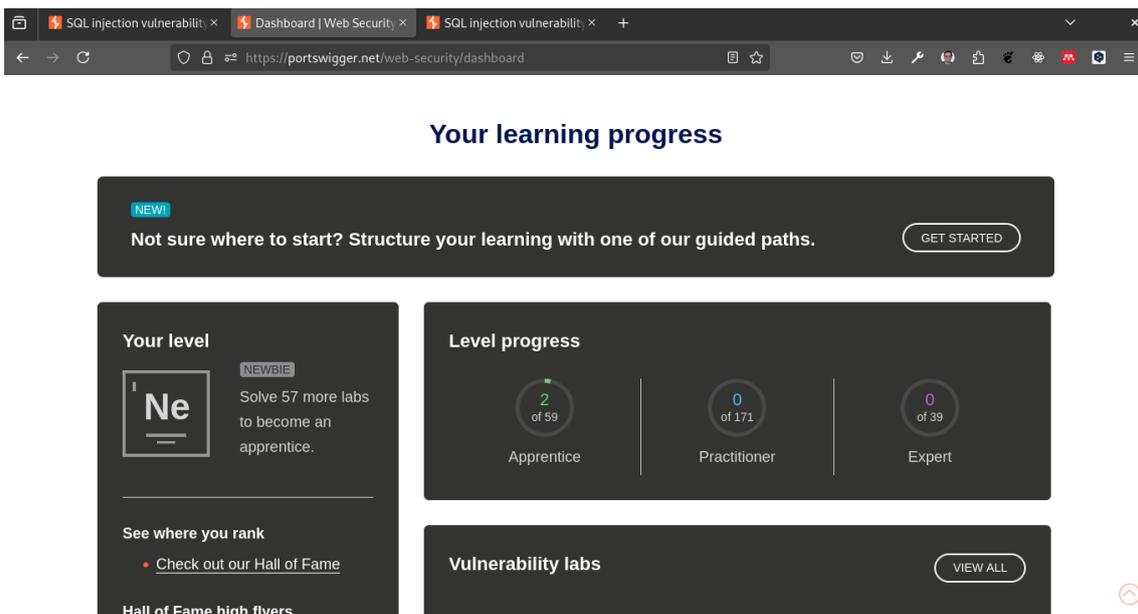
4. La aplicación ahora nos muestra la página de inicio, lo que indica que la inyección SQL en la consulta de autenticación ha tenido éxito.
5. Haga clic en **Account** para ver la página de la cuenta del usuario **administrator**.



6. Si verificamos en los laboratorios, ahora podemos observar que el laboratorio **SQL injection vulnerability allowing login bypass** aparece como **Solved**.



7. En la sección Academy, haga clic en **Progress** y verifique que el laboratorio **SQL injection vulnerability allowing login bypass** esté marcado como **Solved**.



Con esto se ha completado el laboratorio **SQL injection vulnerability allowing login bypass**.

## 40 Conclusión

En este laboratorio, el estudiante ha aprendido a identificar y explotar una vulnerabilidad de inyección SQL en una consulta de autenticación para iniciar sesión como el primer usuario en la base de datos, incluso si la contraseña es desconocida.

El estudiante ha utilizado Burp Suite para interceptar el tráfico de red y modificar los parámetros de la URL para realizar la inyección SQL.

## **Part III**

# **Laboratorios de Fuzzing**

# 41 Laboratorio: Integración de Proyectos de Código Abierto con OSS-Fuzz

## 41.1 Objetivo:

En este laboratorio, aprenderás a integrar un proyecto de código abierto con OSS-Fuzz para mejorar su seguridad y estabilidad mediante técnicas modernas de fuzzing y ejecución distribuida. Al final del laboratorio, habrás configurado un proyecto de JavaScript para Node.js y lo habrás integrado con OSS-Fuzz utilizando Jazzer.js, siguiendo un ejemplo práctico.

## 41.2 Requisitos Previos

- Conocimientos básicos de programación en JavaScript.
- Familiaridad con Docker y manejo de imágenes Docker.
- Acceso a una cuenta de GitHub para subir configuraciones y código.
- Un proyecto de código abierto en JavaScript o TypeScript (puede ser un proyecto de ejemplo).

## 41.3 Temas del Laboratorio

- Introducción a OSS-Fuzz
- Configuración del Entorno
- Integración de un Proyecto en OSS-Fuzz
  - Archivos de Configuración
  - Dockerfile
  - Fuzzers
- Ejecución de Fuzzing y Análisis de Resultados

## 41.4 1. Introducción a OSS-Fuzz

OSS-Fuzz es un servicio gratuito ofrecido por Google para mejorar la seguridad y estabilidad de los proyectos de código abierto mediante técnicas de fuzzing. Desde su lanzamiento en 2016, ha ayudado a descubrir y corregir más de 10,000 vulnerabilidades y 36,000 bugs en más de 1,000 proyectos.

El fuzzing es una técnica que alimenta a un programa con entradas aleatorias para encontrar errores como desbordamientos de buffer y otros problemas de seguridad.

### 41.4.1 Arquitectura de OSS-Fuzz

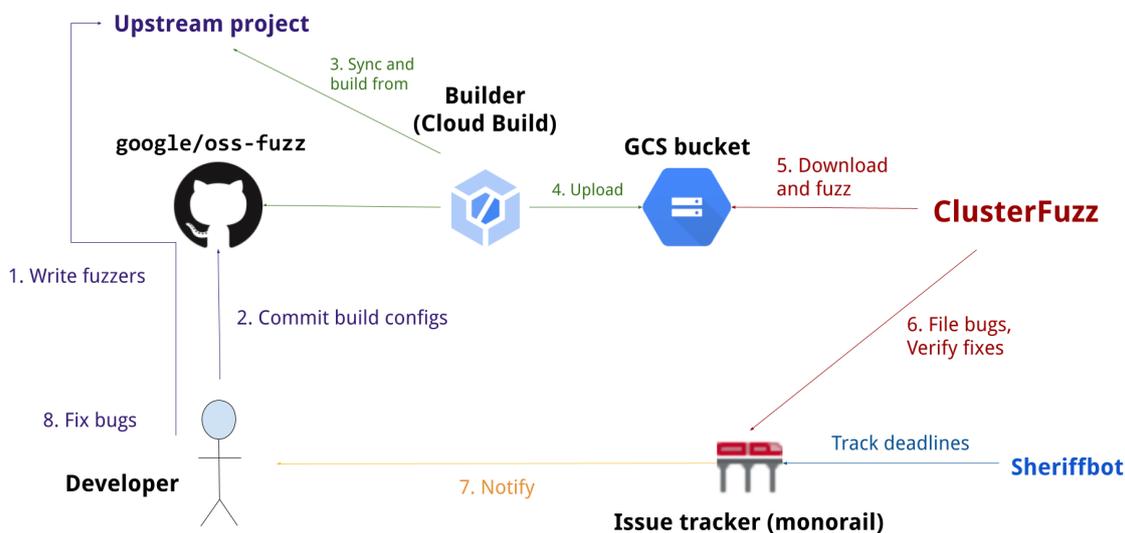


Figure 41.1: Arquitectura de OSS-Fuzz

El proceso de integración en OSS-Fuzz sigue estos pasos:

1. Un mantenedor crea objetivos de fuzzing y los integra con el sistema de compilación del proyecto.
2. El proyecto es aceptado en OSS-Fuzz y las configuraciones de compilación se suben.
3. OSS-Fuzz construye el proyecto y sube los objetivos de fuzzing a un bucket de GCS.
4. ClusterFuzz descarga los objetivos y comienza el fuzzing.
5. Si se encuentra un bug, ClusterFuzz lo reporta en el issue tracker de OSS-Fuzz.
6. Los desarrolladores corrigen el bug y lo acreditan a OSS-Fuzz.
7. ClusterFuzz verifica la corrección y cierra el issue.

## 41.5 2. Configuración del Entorno

### 41.5.1 2.1. Clonar el Proyecto de Ejemplo

Clona el proyecto de ejemplo de JavaScript proporcionado por OSS-Fuzz para utilizarlo como referencia.

```
git clone https://github.com/google/oss-fuzz.git
cd oss-fuzz/projects/javascript-example
```

## 41.5.2 2.2. Instalación de Dependencias

Asegúrate de tener Docker instalado en tu máquina:

```
sudo dnf install docker
```

Luego, instala Node.js si aún no lo tienes:

```
sudo dnf install nodejs
```

## 41.6 3. Integración de un Proyecto en OSS-Fuzz

### 41.6.1 3.1. Archivos de Configuración

Dentro del proyecto, crea un archivo `project.yaml` con el siguiente contenido:

```
language: javascript

fuzzing_engines:
  - libfuzzer

sanitizers:
  - none
```

### 41.6.2 3.2. Dockerfile

Crea un archivo `Dockerfile` que utilice la imagen base de OSS-Fuzz para JavaScript:

```
FROM gcr.io/oss-fuzz-base/base-builder-javascript

COPY . $SRC/
WORKDIR $SRC

RUN npm install
```

### 41.6.3 3.3. Fuzzers

Crea un fuzzer simple en `fuzz_string_compare.js`:

```
module.exports.fuzz = function (data) {
  const s = data.toString();
  if (s.length !== 16) {
    return;
  }
}
```

```
    if (s.slice(0, 8) === "Awesome " && s.slice(8, 15) === "Fuzzing" && s[15] === "!") {  
        throw Error("Welcome to Awesome Fuzzing!");  
    }  
};
```

#### 41.6.4 3.4. build.sh

Crea un archivo build.sh que utilice el script de compilación de OSS-Fuzz:

```
#!/bin/bash -eu  
  
compile_javascript_fuzzer $SRC example/fuzz_string_compare.js --sync
```

## 41.7 4. Ejecución de Fuzzing y Análisis de Resultados

### 41.7.1 4.1. Compilar y Ejecutar el Fuzzer

Compila y ejecuta el fuzzer usando Docker:

```
docker build -t oss-fuzz-javascript .  
docker run --rm -it oss-fuzz-javascript
```

### 41.7.2 4.2. Análisis de Resultados

Si ClusterFuzz encuentra algún problema, recibirás una notificación en el issue tracker de OSS-Fuzz, donde podrás ver los detalles del bug encontrado y tomar acción para corregirlo.

## 42 Conclusión

Al completar este laboratorio, habrás configurado con éxito un proyecto de JavaScript para Node.js utilizando OSS-Fuzz. Esto contribuirá a la seguridad y estabilidad del proyecto, siguiendo las mejores prácticas en fuzzing.

**Part IV**

**OWASP Top 10**

## 43 Introducción a OWASP top 10



Figure 43.1: OWASP

OWASP es una organización sin fines de lucro que se dedica a mejorar la seguridad de las aplicaciones. Para ello, publica una lista de los 10 riesgos más críticos en la seguridad de las aplicaciones web. Esta lista se actualiza cada 3 años y es una guía para los desarrolladores y profesionales de la seguridad informática.

En esta sección, vamos a ver los 10 riesgos más críticos en la seguridad de las aplicaciones web según OWASP.

### 43.1 ¿Por qué es importante conocer los riesgos de seguridad en las aplicaciones web?

Las aplicaciones web son un objetivo común para los atacantes, ya que pueden contener información sensible, como contraseñas, datos personales o información financiera. Por lo tanto, es importante conocer los riesgos de seguridad en las aplicaciones web para proteger la información y la privacidad de los usuarios.

Además, las vulnerabilidades en las aplicaciones web pueden tener consecuencias graves, como la pérdida de datos, el robo de información confidencial o la interrupción del servicio. Por lo tanto, es importante identificar y mitigar los riesgos de seguridad en las aplicaciones web para evitar posibles ataques y proteger la integridad y la confidencialidad de la información.

## 43.2 ¿Qué es el OWASP Top 10?

El OWASP Top 10 es una lista de los 10 riesgos más críticos en la seguridad de las aplicaciones web. Esta lista se actualiza cada 3 años y es una guía para los desarrolladores y profesionales de la seguridad informática. El OWASP Top 10 proporciona información sobre los riesgos de seguridad más comunes en las aplicaciones web y cómo mitigarlos.

El OWASP Top 10 se basa en datos reales de vulnerabilidades en aplicaciones web y en la experiencia de profesionales de la seguridad informática. La lista se actualiza regularmente para reflejar las últimas tendencias y amenazas en la seguridad de las aplicaciones web.

## 43.3 ¿Cuáles son los riesgos más críticos en la seguridad de las aplicaciones web?

Los 10 riesgos más críticos en la seguridad de las aplicaciones web según OWASP son:

1. Inyección SQL
2. Autenticación y gestión de sesiones defectuosas
3. Exposición de datos sensibles
4. XML External Entities (XXE)
5. Control de acceso inadecuado
6. Configuración de seguridad defectuosa
7. Cross-Site Scripting (XSS)
8. Deserialización insegura
9. Utilización de componentes con vulnerabilidades conocidas
10. Insuficiente monitorización y registro de eventos

En las siguientes secciones, vamos a ver en detalle cada uno de estos riesgos y cómo mitigarlos.

### 43.4 1. Inyección SQL

Las inyecciones SQL son una vulnerabilidad que permite a un atacante ejecutar código SQL en la base de datos de una aplicación. Esto puede permitir al atacante leer, modificar o eliminar datos de la base de datos.

Para prevenir las inyecciones SQL, es importante utilizar consultas parametrizadas en lugar de concatenar cadenas de texto para formar las consultas SQL.

Ejemplo:

```
# Ejemplo de consulta SQL vulnerable
username = request.form['username']
password = request.form['password']
query = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password
cursor.execute(query)
```

```
# Ejemplo de consulta SQL segura
username = request.form['username']
password = request.form['password']
query = "SELECT * FROM users WHERE username=%s AND password=%s"
cursor.execute(query, (username, password))
```

En el ejemplo anterior, la primera consulta es vulnerable a inyección SQL porque concatena cadenas de texto para formar la consulta SQL. La segunda consulta es segura porque utiliza consultas parametrizadas y evita la inyección SQL.

Para más información, puedes consultar la [documentación de OWASP sobre inyecciones SQL](#).

## 43.5 2. Autenticación y gestión de sesiones defectuosas

La autenticación y la gestión de sesiones son fundamentales para la seguridad de una aplicación web. Una autenticación defectuosa puede permitir a un atacante acceder a cuentas de usuario sin autorización, mientras que una gestión de sesiones defectuosa puede permitir a un atacante secuestrar la sesión de un usuario legítimo.

Para prevenir problemas de autenticación y gestión de sesiones, es importante utilizar prácticas seguras de autenticación, como el uso de contraseñas seguras, la autenticación de dos factores y el bloqueo de cuentas después de varios intentos fallidos de inicio de sesión. Además, es importante proteger las sesiones de los usuarios con medidas como el uso de cookies seguras, la expiración de sesiones y la renovación de tokens de sesión.

Ejemplo:

```
# Ejemplo de autenticación defectuosa
username = request.form['username']
password = request.form['password']
query = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password
user = cursor.execute(query)

# Ejemplo de autenticación segura
username = request.form['username']
password = request.form['password']
query = "SELECT * FROM users WHERE username=%s AND password=%s"
user = cursor.execute(query, (username, password))
```

En el ejemplo anterior, la primera consulta es vulnerable a inyección SQL y no verifica si el usuario y la contraseña son correctos. La segunda consulta es segura porque utiliza consultas parametrizadas y verifica si el usuario y la contraseña son correctos.

## 43.6 3. Exposición de datos sensibles

La exposición de datos sensibles es una vulnerabilidad que permite a un atacante acceder a información confidencial, como contraseñas, números de tarjetas de crédito o datos personales. Esto puede ocurrir cuando los datos sensibles se almacenan o se transmiten de forma insegura, o cuando se muestran en la interfaz de usuario de la aplicación.

Para prevenir la exposición de datos sensibles, es importante cifrar los datos sensibles en reposo y en tránsito, utilizar conexiones seguras (HTTPS), y limitar el acceso a los datos sensibles solo a los usuarios autorizados.

Ejemplo:

```
# Ejemplo de exposición de datos sensibles
query = "SELECT * FROM users WHERE id=" + user_id
user = cursor.execute(query)

# Ejemplo de protección de datos sensibles
query = "SELECT * FROM users WHERE id=%s"
user = cursor.execute(query, (user_id,))
```

En el ejemplo anterior, la primera consulta es vulnerable a inyección SQL y expone datos sensibles al mostrarlos directamente en la interfaz de usuario. La segunda consulta es segura porque utiliza consultas parametrizadas y no expone datos sensibles en la interfaz de usuario.

## 43.7 4. XML External Entities (XXE)

Las entidades XML externas (XXE) son una vulnerabilidad que permite a un atacante leer archivos locales, realizar escaneos de puertos y realizar ataques de denegación de servicio a través de documentos XML maliciosos.

Para prevenir los ataques de XXE, es importante deshabilitar el procesamiento de entidades XML externas, validar y filtrar las entradas de los usuarios, y utilizar bibliotecas seguras para el procesamiento de XML.

Ejemplo:

```
# Ejemplo de procesamiento de entidades XML externas
xml_data = request.data
dom = minidom.parseString(xml_data)
```

En el ejemplo anterior, el código es vulnerable a ataques de XXE porque no deshabilita el procesamiento de entidades XML externas. Para prevenir los ataques de XXE, es importante deshabilitar el procesamiento de entidades XML externas utilizando la configuración adecuada en la biblioteca de procesamiento de XML.

## 43.8 5. Control de acceso inadecuado

El control de acceso inadecuado es una vulnerabilidad que permite a un atacante acceder a recursos protegidos sin autorización. Esto puede ocurrir cuando no se implementan controles de acceso adecuados, como la autenticación, la autorización y la validación de permisos.

Para prevenir problemas de control de acceso inadecuado, es importante implementar controles de acceso basados en roles, validar los permisos de los usuarios en cada solicitud y proteger los recursos sensibles con medidas de seguridad adicionales, como el cifrado y la autenticación de dos factores.

Ejemplo:

```
# Ejemplo de control de acceso inadecuado
if user.role == 'admin':
    # Acceso a recursos sensibles
else:
    # Acceso denegado

# Ejemplo de control de acceso adecuado
if user.has_permission('admin'):
    # Acceso a recursos sensibles
else:
    # Acceso denegado
```

En el ejemplo anterior, el primer código es vulnerable a control de acceso inadecuado porque solo verifica el rol del usuario, pero no valida los permisos específicos. El segundo código es seguro porque valida los permisos específicos del usuario antes de permitir el acceso a los recursos sensibles.

## 43.9 6. Configuración de seguridad defectuosa

La configuración de seguridad defectuosa es una vulnerabilidad que permite a un atacante acceder a recursos protegidos debido a una configuración incorrecta o insegura. Esto puede ocurrir cuando se utilizan configuraciones predeterminadas inseguras, se exponen servicios innecesarios o se utilizan credenciales débiles.

Para prevenir problemas de configuración de seguridad defectuosa, es importante seguir las mejores prácticas de seguridad, como cambiar las credenciales predeterminadas, deshabilitar servicios innecesarios, y utilizar herramientas de escaneo de seguridad para identificar y corregir configuraciones inseguras.

Ejemplo:

```
# Ejemplo de configuración de seguridad defectuosa
DEBUG = True
SECRET_KEY

# Ejemplo de configuración de seguridad segura
DEBUG = False
SECRET_KEY
```

En el ejemplo anterior, la primera configuración es vulnerable a configuración de seguridad defectuosa porque expone información sensible (DEBUG y SECRET\_KEY). La segunda configuración es segura porque oculta la información sensible y utiliza valores seguros para las configuraciones.

## 43.10 7. Cross-Site Scripting (XSS)

El Cross-Site Scripting (XSS) es una vulnerabilidad que permite a un atacante ejecutar scripts maliciosos en el navegador de un usuario a través de páginas web vulnerables. Esto puede permitir al atacante robar cookies de sesión, redirigir a los usuarios a sitios maliciosos o realizar ataques de phishing.

Para prevenir los ataques de XSS, es importante validar y escapar las entradas de los usuarios, utilizar encabezados de seguridad como Content-Security-Policy, y utilizar bibliotecas seguras para la generación de HTML.

Ejemplo:

```
# Ejemplo de vulnerabilidad de XSS
name = request.form['name']
return "<h1>Welcome, " + name + "</h1>"
```

En el ejemplo anterior, el código es vulnerable a ataques de XSS porque no escapa la entrada del usuario antes de mostrarla en la página web. Para prevenir los ataques de XSS, es importante escapar las entradas del usuario utilizando funciones de escape HTML.

## 43.11 8. Deserialización insegura

La deserialización insegura es una vulnerabilidad que permite a un atacante ejecutar código malicioso a través de la deserialización de objetos. Esto puede permitir al atacante leer, modificar o eliminar datos de la aplicación, o incluso ejecutar código remoto en el servidor.

Para prevenir problemas de deserialización insegura, es importante validar y filtrar los datos de entrada, utilizar mecanismos de serialización seguros como JSON en lugar de pickle, y limitar los privilegios de los objetos deserializados.

Ejemplo:

```
# Ejemplo de deserialización insegura
data = request.data
object = pickle.loads(data)

# Ejemplo de deserialización segura
data = request.data
object = json.loads(data)
```

En el ejemplo anterior, el código es vulnerable a deserialización insegura porque utiliza la biblioteca pickle para deserializar objetos. Para prevenir problemas de deserialización insegura, es importante utilizar mecanismos de serialización seguros como JSON en lugar de pickle.

## 43.12 9. Utilización de componentes con vulnerabilidades conocidas

La utilización de componentes con vulnerabilidades conocidas es una vulnerabilidad que permite a un atacante explotar vulnerabilidades conocidas en bibliotecas, frameworks o plugins utilizados por la aplicación. Esto puede permitir al atacante ejecutar código malicioso, robar información sensible o realizar ataques de denegación de servicio.

Para prevenir problemas de utilización de componentes con vulnerabilidades conocidas, es importante mantener actualizadas todas las bibliotecas y componentes utilizados por la aplicación, utilizar herramientas de escaneo de seguridad para identificar vulnerabilidades conocidas, y seguir las mejores prácticas de seguridad al seleccionar y utilizar componentes de terceros.

Ejemplo:

```
# Ejemplo de utilización de componentes con vulnerabilidades conocidas

# Biblioteca vulnerable
import vulnerable_library

# Ejemplo de utilización de componentes seguros
# Biblioteca segura
```

En el ejemplo anterior, el código es vulnerable a utilización de componentes con vulnerabilidades conocidas porque importa una biblioteca vulnerable. Para prevenir problemas de utilización de componentes con vulnerabilidades conocidas, es importante utilizar componentes seguros y mantener actualizadas todas las bibliotecas y componentes utilizados por la aplicación.

## 43.13 10. Insuficiente monitorización y registro de eventos

La insuficiente monitorización y registro de eventos es una vulnerabilidad que dificulta la detección y respuesta a incidentes de seguridad. Esto puede permitir a un atacante realizar actividades maliciosas sin ser detectado, o dificultar la identificación de la causa de un incidente de seguridad.

Para prevenir problemas de insuficiente monitorización y registro de eventos, es importante implementar un sistema de monitorización y registro de eventos que registre todas las actividades de la aplicación, establecer alertas para eventos sospechosos, y realizar análisis de registros de forma regular para identificar posibles incidentes de seguridad.

Ejemplo:

```
# Ejemplo de insuficiente monitorización y registro de eventos
# No se registran eventos de inicio de sesión
# No se establecen alertas para eventos sospechosos
# No se realizan análisis de registros de forma regular
```

En el ejemplo anterior, el código es vulnerable a insuficiente monitorización y registro de eventos porque no registra eventos de inicio de sesión, no establece alertas para eventos sospechosos y no realiza análisis de registros de forma regular. Para prevenir problemas de insuficiente monitorización y registro de eventos, es importante implementar un sistema de monitorización y registro de eventos que registre todas las actividades de la aplicación y establezca alertas para eventos sospechosos.

## 43.14 Conclusión

En esta sección, hemos visto los 10 riesgos más críticos en la seguridad de las aplicaciones web según OWASP. Es importante tener en cuenta estos riesgos al desarrollar aplicaciones web y seguir las mejores prácticas de seguridad para prevenir posibles vulnerabilidades. La seguridad de las aplicaciones web es un aspecto fundamental en la protección de la información y la privacidad de los usuarios, por lo que es importante dedicar tiempo y recursos a garantizar la seguridad de las aplicaciones web.

## 44 Referencias

- [OWASP Top 10](#)
- [OWASP Top 10 2021](#)
- [OWASP Top 10 2017](#)