Documentado por
Sergio Garcia

DOCUMENTACION WEBGOAT

Índice de contenido

1. OBJETIVO DEL PROYECTO

Aplicación: WebGoat 8.1.0
Entorno: Local (Docker)

El objetivo de esta auditoría es evaluar la seguridad de la aplicación web **WebGoat 8.1.0**, desplegada en un entorno local mediante contenedores Docker, identificando vulnerabilidades incluidas en el **OWASP Top 10** y evaluando su impacto sobre la confidencialidad, integridad y disponibilidad del sistema.

La auditoría se ha centrado exclusivamente en pruebas de tipo **white-box guiadas**, sobre vulnerabilidades intencionadamente incluidas en la aplicación con fines formativos.

2. INFORME DEL PROYECTO

2.1 Breve resumen del proceso realizado

Durante la auditoría se realizaron tareas de reconocimiento del entorno, identificación de tecnologías utilizadas y explotación controlada de vulnerabilidades conocidas presentes en la aplicación WebGoat.
Se utilizaron herramientas como navegador web, Burp Suite y Docker para interceptar y modificar peticiones, así como para validar el impacto de las vulnerabilidades detectadas.

2.2 Vulnerabilidades destacadas

Las principales vulnerabilidades identificadas fueron:

- Inyección SQL (SQL Injection)
- Cross-Site Scripting (XSS)
- XML External Entity Injection (XXE)
- Uso de componentes vulnerables y desactualizados (jquery-ui)
- Fallos en autenticación y gestión de contraseñas

EVALUACION DE RIESGOS

Tabla con calculadora de riesgos OWASP

ID	Vulnerabilidad	Severidad	Probabilidad	Impacto	Estado
4.1	Inyección SQL	CRÍTICA	Alta	Muy Alto	Explotado
4.2	XSS Reflejado	ALTA	Media	Medio-Alto	Explotado
4.3	XXE (XML External Entity)	CRÍTICA	Media-Alta	Muy Alto	Explotado
4.4	Componentes Obsoletos (jQuery UI)	ALTA	Alta	Medio-Alto	Confirmado
4.4	Fallos de Autenticación (jQuery UI)	ALTA	Alta	Medio-Alto	Confirmado
4.5	Fallos de Autenticación (Passwords)	MEDIA	Alta	Alto	Confirmado

CRÍTICA ALTA
MEDIA Confirmado

https://microhackers.ai/es/herramientas-ciberseguridad/calculadora-evaluacion-riesgos/

2.3 Conclusiones

La aplicación presenta múltiples vulnerabilidades críticas que permiten a un atacante obtener información sensible, ejecutar código malicioso en el navegador de los usuarios y comprometer la seguridad general del sistema.
Estas vulnerabilidades evidencian la importancia de validar correctamente las entradas de usuario, utilizar mecanismos seguros de autenticación y mantener los componentes actualizados.

2.4 Recomendaciones

Se recomienda:

- Utilizar consultas preparadas para evitar inyecciones SQL
- Validar y codificar correctamente la salida de datos
- Deshabilitar entidades externas en parsers XML
- Mantener las librerías y componentes de terceros actualizados
- Aplicar políticas de contraseñas robustas
- Seguir las buenas prácticas definidas por OWASP

3. DESCRIPCIÓN DEL PROCESO DE AUDITORÍA

3.1 Reconocimiento / Information Gathering

Durante la fase de reconocimiento se identificó:

- **Puerto expuesto:** 8080 (se utilizó nmap -p 8080 para confirmar el servicio)
- **Sistema operativo:** Linux
- **Lenguaje y framework:** Java (Spring / Servlet)
- **Entorno:** Aplicación desplegada en contenedor Docker

No se realizaron escaneos agresivos al tratarse de un entorno local controlado.

3.2 Explotación de vulnerabilidades detectadas

3.2.1 Inyección SQL (A3 – Injection)

Vulnerabilidad explotada dentro del módulo A3 Injection – SQL Injection (intro), apartado 11 de WebGoat.

La aplicación construye consultas SQL concatenando directamente los datos introducidos por el usuario.

Introduje el siguiente valor en los campos de entrada:

Employee Name: ' OR '1'='1

Authentication TAN: ' OR '1'='1

El valor introducido (' OR '1'='1) fuerza a que la condición de la consulta SQL sea siempre verdadera. De este modo, la aplicación deja de filtrar los datos por usuario y devuelve todos los registros de la base de datos.

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

Employee Name: ' OR '1'='1

Authentication TAN: ' OR '1'='1

Get department

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	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

También se probaron variantes como:

Employee Name: Smith'

Authentication TAN: ' OR '1'='1

Employee Name: Smith'

Authentication TAN: OR '1' ='1

Get department

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

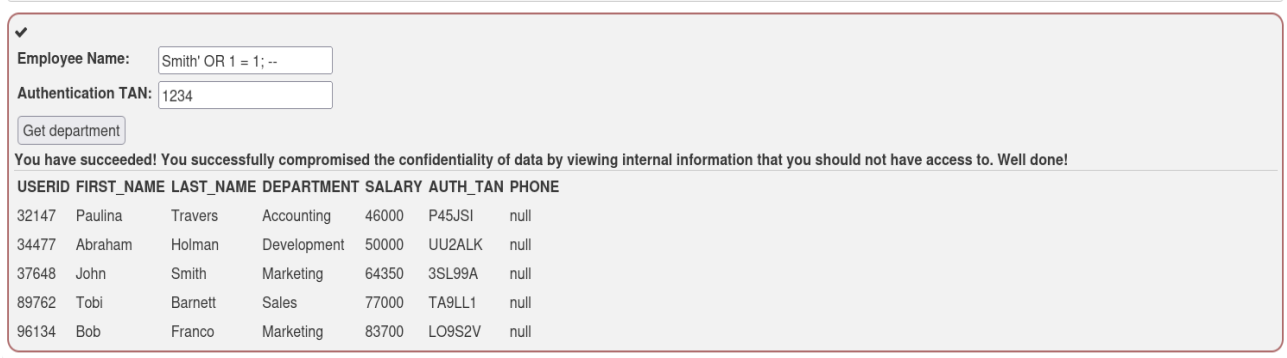
USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

- Otra variante:

Introduje el siguiente valor en los campos de entrada:

```
Smith' OR 1 = 1; --
```

Esta inyección SQL hace lo mismo que la anterior la diferencia es que al poner – comenta todo lo que viene después, es una opción mas robusta.



- Otra Variante:

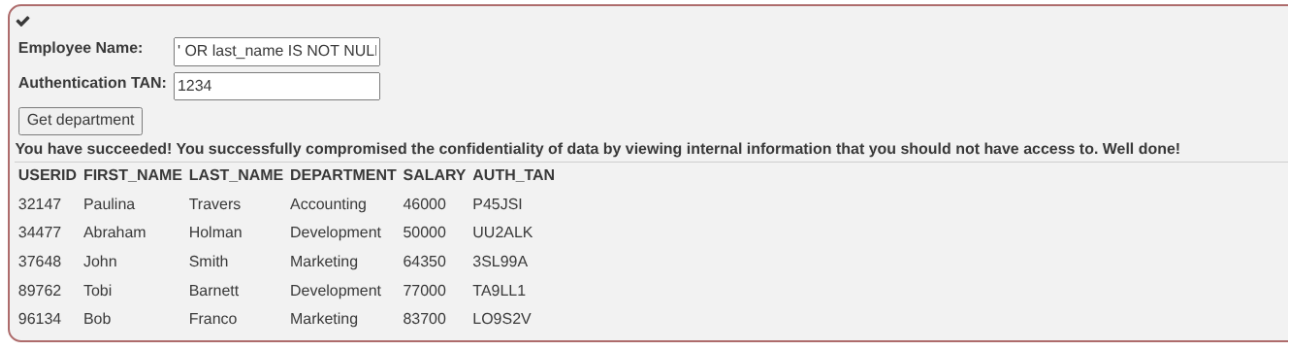
Introduje el siguiente valor en los campos de entrada:

Employee Name: ' OR last_name IS NOT NULL --

Authentication TAN: 1234

Este valor permite alterar la consulta SQL para que se cumpla siempre.
La condición IS NOT NULL hace que se muestren todos los empleados, y el comentario SQL (--) provoca que la comprobación del TAN sea ignorada.

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



Gravedad de vulnerabilidad: CRITICA

Esta vulnerabilidad compromete la confidencialidad del sistema, ya que permite acceder a información sensible

Mitigación / Solución

La vulnerabilidad se soluciona evitando la construcción de consultas SQL mediante la concatenación directa de los datos introducidos por el usuario.
Para ello, se deben utilizar consultas preparadas, que separan la lógica de la consulta de los datos introducidos, impidiendo que estos puedan modificar su comportamiento. Ejemplo seguro de consulta preparada:

```
String query = "SELECT * FROM employees WHERE last_name = ? AND auth_tan = ?";
PreparedStatement stmt = connection.prepareStatement(query);
stmt.setString(1, name);
stmt.setString(2, tan);
```

El SQL y los datos van separados, el usuario no puede cambiar consultas, así la inyección SQL no funciona

Referencias

- https://owasp.org/www-community/attacks/SQL_Injection

https://owasp.org/Top10/A03_2021-Injection/

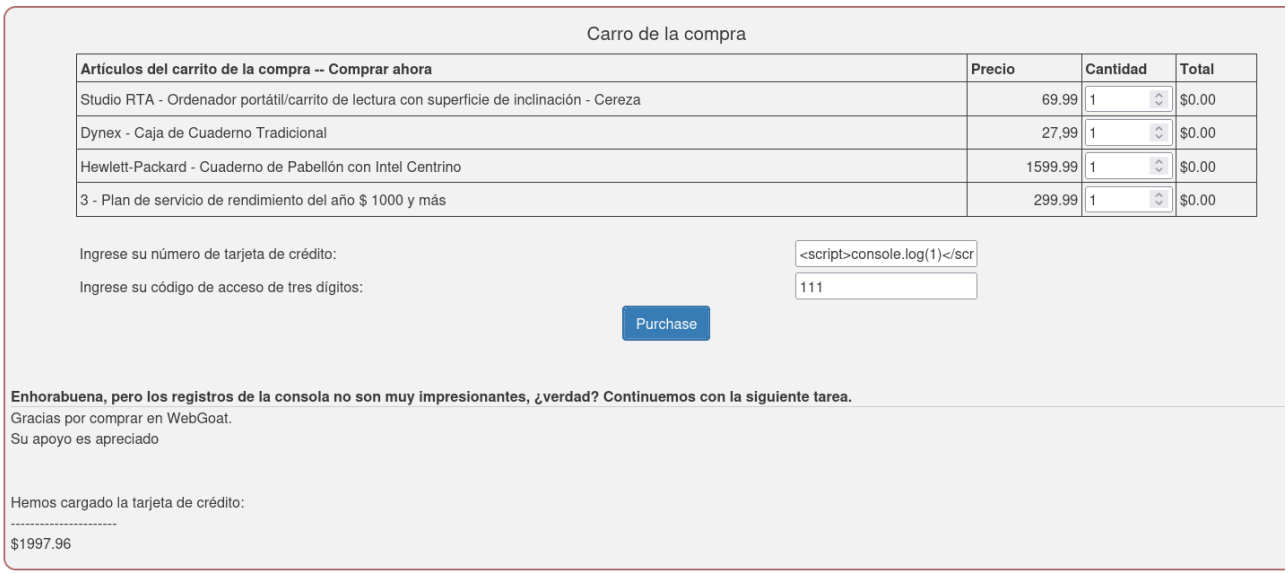
3.2.2 Cross-Site Scripting (XSS)

Vulnerabilidad explotada dentro del módulo A3 Injection – Cross-Site Scripting (XSS), apartado 7 de WebGoat.

Se detectó XSS reflejado en el campo de introducción de tarjeta de crédito.

El campo ingrese tarjeta de crédito es vulnerable a XSS reflejado, ya que permite la ejecución de código JavaScript introducido por el usuario.
Introduje el siguiente valor en el campo:

```
<script>alert(1)</script>
<script>console.log(1)</script>
```



Gravedad de la vulnerabilidad: ALTA

Esta vulnerabilidad compromete la confidencialidad del sistema**,** un atacante podría ejecutar código JavaScript en el navegador del usuario,
lo que podría derivar en robo de información o ejecución de acciones no autorizadas.

Mitigación / Solución

Validar y codificar correctamente la salida de los datos introducidos por
el usuario y evitar que se renderice código HTML o JavaScript sin control.

Referencias

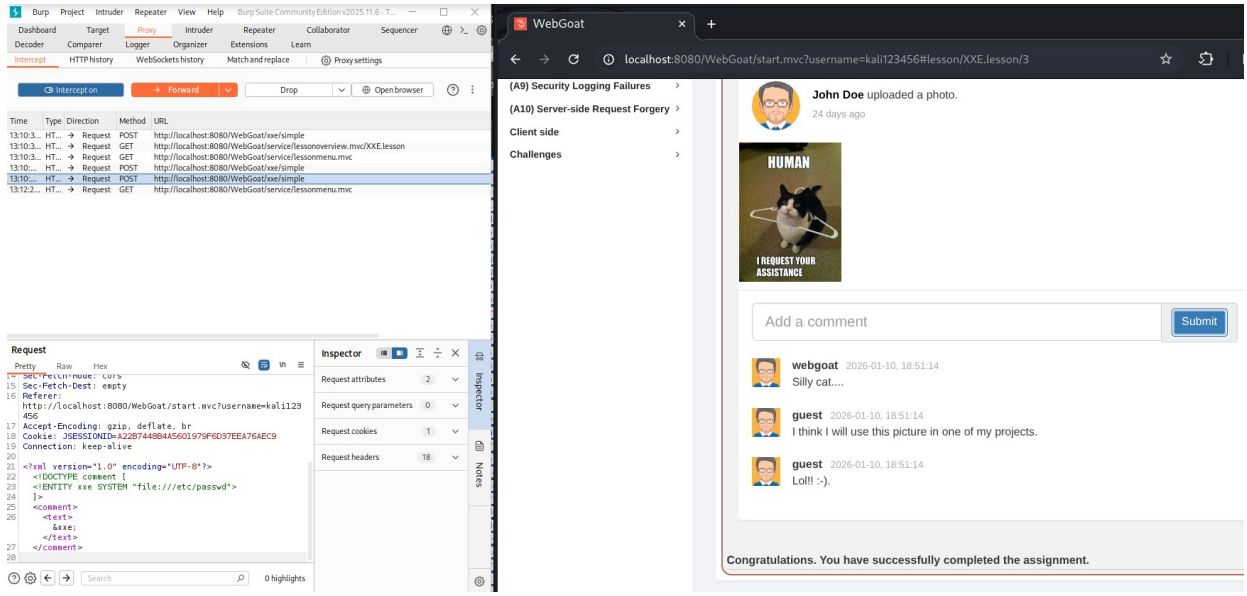
- https://owasp.org/www-community/attacks/xss/
- https://cheatsheetseries.owasp.org/cheatsheets/XSS_Prevention_Cheat_Sheet.html

3.2.3 Security Misconfiguration – XXE

Vulnerabilidad explotada dentro del módulo A5 Security Misconfiguration – XML External Entity (XXE), apartado 4 de WebGoat.

La aplicación procesa XML permitiendo entidades externas.

La aplicación procesa datos en formato XML utilizando un parser que permite la definición de entidades externas. Mediante la inyección de una entidad que referencia a un archivo local del sistema, se demuestra la existencia de una vulnerabilidad XXE que puede comprometer la confidencialidad del servidor.



Introduje en comentario: hola, lo intercepté con Burp Suite y cambié el xml por este:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE comment [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<comment>
  <text>&xxe;</text>
</comment>
```

Gravedad de la vulnerabilidad: CRITICA

Esta vulnerabilidad compromete la confidencialidad del sistema el atacante puede:

- Leer archivos del sistema (`/etc/passwd`, `/etc/hosts`)
- Leer variables de entorno
- A veces hacer peticiones internas (SSRF)
- Comprometer la confidencialidad del servidor

Mitigación / Solución

Deshabilitar el uso de entidades externas y DTD en el parser XML y validar estrictamente las entradas proporcionadas por el usuario.

Referencias

- https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

3.2.4 Uso de componentes vulnerables y desactualizados

Vulnerabilidad explotada dentro del módulo A6 Vulnerable & Outdated Components, apartado 5 de WebGoat.

La aplicación utiliza jquery-ui 1.10.4, versión vulnerable a XSS.

La aplicación es vulnerable debido al uso de una versión obsoleta y vulnerable de la librería jquery-ui (v1.10.4).
Esta versión no valida ni escapa correctamente el contenido proporcionado por el usuario en el parámetro `closeText` del componente `dialog`, permitiendo la ejecución de código JavaScript arbitrario en el navegador de la víctima.

El mismo código fuente, al utilizar una versión actualizada de jquery-ui (v1.12.0), no presenta la vulnerabilidad, lo que confirma que el fallo reside en el componente externo, no en el código propio de la aplicación.

Uso de una librería externa vulnerable y desactualizada que no aplica medidas de sanitización ni escape de datos controlados por el usuario.

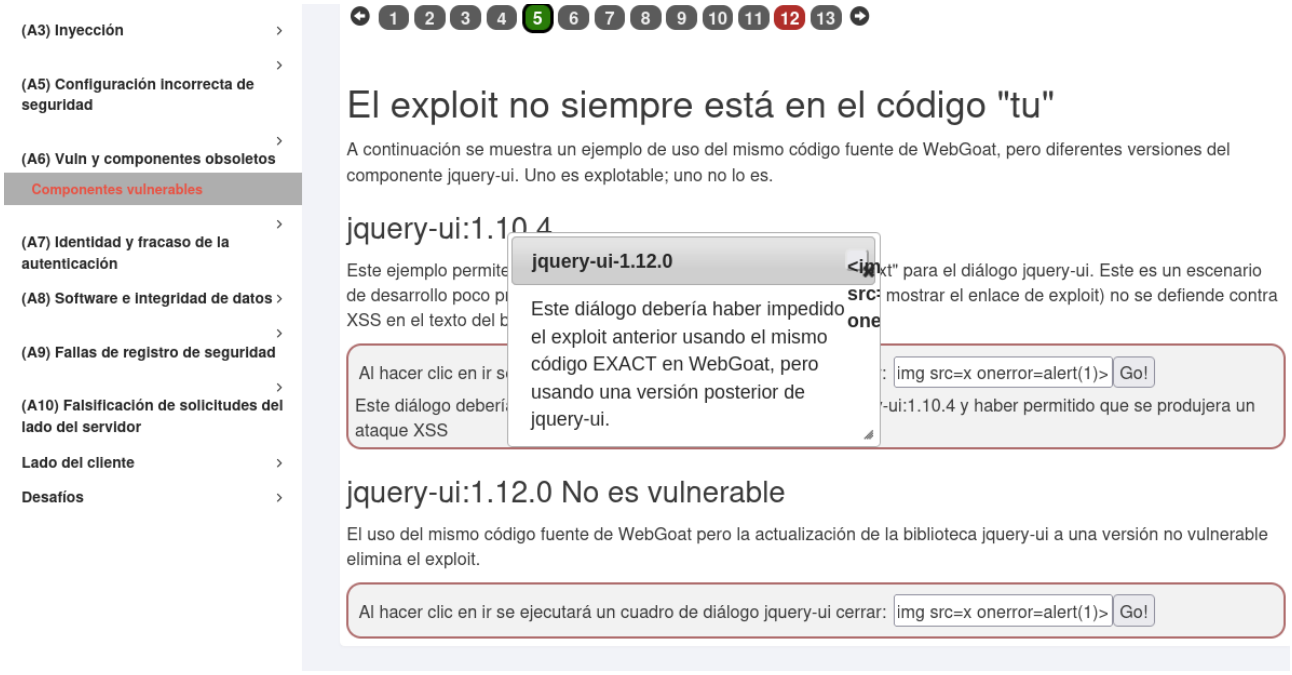
Al introducir código HTML/javascript como valor del parámetro closeText , por ejemplo:

<img src=x onerror=alert(1)>
o este

<script>alert('XSS')</script>

y activar el diálogo, el código JavaScript se ejecuta automáticamente en el navegador del usuario cuando se utiliza jquery-ui versión 1.10.4.

Con la versión jquery-ui 1.12.0, el mismo payload no se ejecuta.



Gravedad de la vulnerabilidad: ALTA

Esta vulnerabilidad compromete la confidencialidad del sistema:

- Ejecutar JavaScript arbitrario en el navegador de la víctima
- Robar cookies de sesión
- Secuestrar sesiones de usuario
- Redirigir al usuario a sitios maliciosos
- Realizar acciones en nombre del usuario autenticado

Mitigación / Solución

- Actualizar jquery-ui a una versión no vulnerable (≥ 1.12.0)
- Mantener actualizadas todas las dependencias de terceros
- Realizar análisis periódicos de dependencias (SCA)
- No confiar en librerías antiguas para la validación de entradas
- Implementar Content Security Policy (CSP) como defensa adicional

Referencias

- https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

Conclusión

Esta vulnerabilidad demuestra que una aplicación puede ser explotable incluso cuando su código propio no contiene fallos directos, si utiliza componentes externos vulnerables. Mantener las dependencias actualizadas es una medida crítica de seguridad.

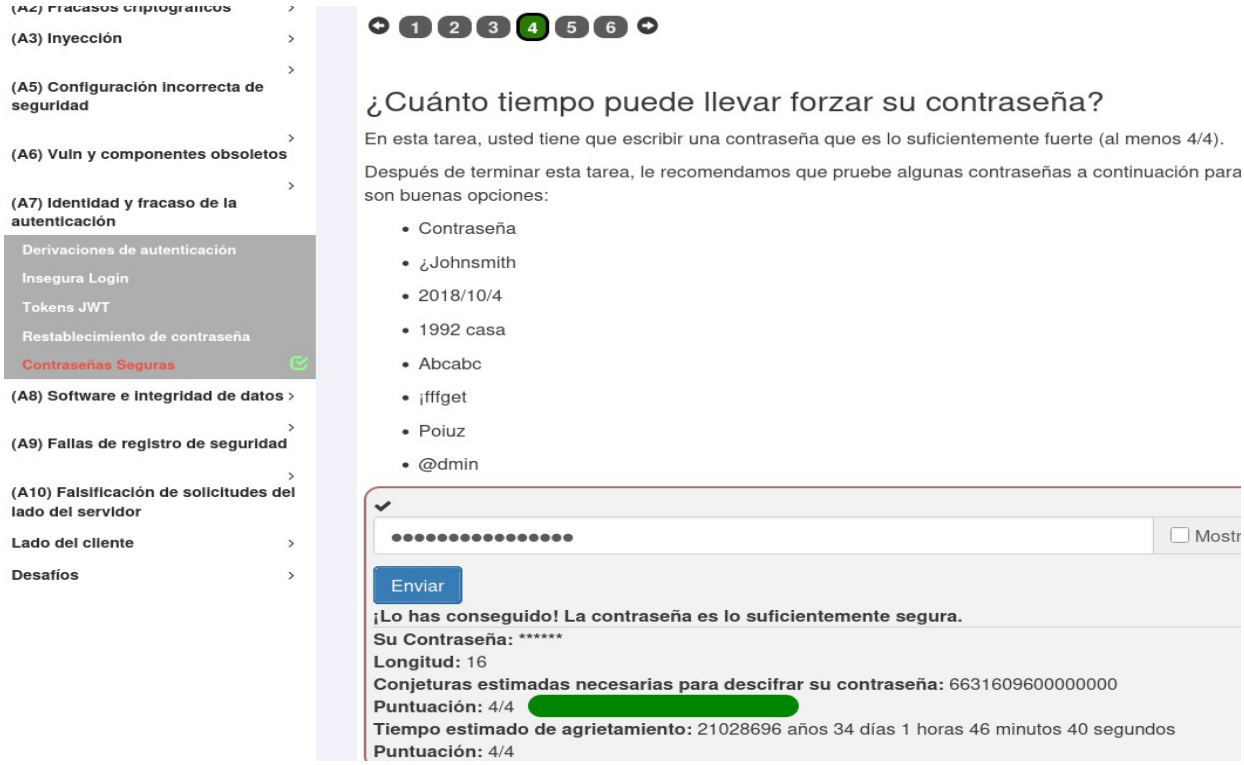
3.2.5 Fallos en autenticación y gestión de contraseñas

Vulnerabilidad explotada dentro del módulo A7 Identity & Authentication Failures – Secure Passwords, apartado 4 de WebGoat.

Análisis de fortaleza de contraseñas:

En este ejercicio se comprobó la importancia del uso de contraseñas robustas frente a ataques de fuerza bruta. La contraseña generada, con una longitud de 16 caracteres y alta complejidad, obtuvo la puntuación máxima (4/4), estimándose un tiempo de descifrado superior a 21 millones de años.

En contraste, contraseñas cortas, predecibles o basadas en datos personales pueden ser descifradas en segundos o minutos, suponiendo un riesgo crítico para la seguridad de la aplicación.



Gravedad de la vulnerabilidad: Media

Esta vulnerabilidad compromete la confidencialidad del sistema

- Acceso no autorizado
- Robo de cuentas
- Escalada de privilegios
- Compromiso total del sistema

Mitigación / Solución

Una contraseña fuerte:

- Es larga min 8 caracteres max 64 caracteres
- No usa palabras comunes
- Mezcla letras, números y símbolos
- No tiene relación con datos personales

Cuanto más larga y variada es la contraseña:
- más combinaciones posibles
- más tiempo necesita un atacante
- menos viable es el ataque

Referencias

- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

Conclusión:

El uso de contraseñas largas y complejas incrementa exponencialmente el número de combinaciones posibles, haciendo que los ataques de fuerza bruta sean computacionalmente inviables.

4. CONCLUSIÓN FINAL

La auditoría demuestra que una aplicación web puede presentar vulnerabilidades críticas tanto por errores propios como por el uso de componentes externos inseguros.
La adopción de buenas prácticas de desarrollo seguro y el mantenimiento continuo de las dependencias son medidas fundamentales para reducir el riesgo y mejorar la postura de seguridad de las aplicaciones web.

En conjunto, las vulnerabilidades identificadas demuestran que una explotación encadenada podría comprometer de forma significativa la confidencialidad de los datos, la integridad de la aplicación y la confianza de los usuarios.

4.1 Post-explotación

Tras la explotación se demostró la posibilidad de:

- Acceder a información sensible
- Ejecutar código en el navegador del usuario
- Leer archivos del sistema
- Comprometer cuentas de usuario

No se realizaron acciones destructivas.

4.2 Posibles soluciones

- Uso de consultas preparadas
- Validación y escape de entradas y salidas
- Deshabilitar XXE en parsers XML
- Actualizar dependencias vulnerables
- Aplicar políticas de contraseñas fuertes
- Implementar medidas defensivas como CSP

4.3 Herramientas utilizadas

- Burp Suite
- Navegador web
- Docker
- Nmap
- SQLmap