**INFORME DE AUDITORIA**

Pagina Analizada: WebGoat

**TABLA DE CONTENIDO**

1. Introducción
   1. Objetivos de la revisión
   2. Resumen
2. Metodología de análisis (Information gathering)
   1. Hallazgos pasivos
   2. Hallazgos activos
   3. Otros hallazgos
3. Vulnerabilidades Identificadas
   1. A3 injection - SQL Injection
   2. A3 injection - Cross Site Scripting
   3. A5 Security Misconfiguration
   4. A6 Vuln & outdated Components
   5. A7 Identity & Auth Failure – Secure Passwords
4. Análisis de riesgos
   1. Resumen de vulnerabilidades
5. Recomendaciones
   1. A3 injection - SQL Injection
   2. A3 injection - Cross Site Scripting
   3. A5 Security Misconfiguration
   4. A6 Vuln & outdated Components
   5. A7 Identity & Auth Failure – Secure Passwords
6. Conclusión
7. Referencias y Bibliografia
8. **INTRODUCCIÓN**

En el módulo de introducción a la ciberseguridad, aprendimos los fundamentos sobre los conceptos, principios y practicas esenciales para identificar y mitigar vulnerabilidades en sistemas y entornos digitales cada vez más expuestos a riesgos, entendiendo su impacto en la seguridad y confianza del entorno tecnológico actual.

Este informe se enfoca en la auditoria de la aplicación WebGoat, una plataforma educativa diseñada para demostrar vulnerabilidades comunes de seguridad en aplicaciones web. Este análisis permite aplicar los conocimientos adquiridos de manera práctica, desarrollando una comprensión más profunda de las amenazas que enfrentan a diario las organizaciones y algunas estrategias para abórdalas.

* 1. Objetivos de la revisión
* Identificar las vulnerabilidades de WebGoat simulando escenarios reales de ataque.
* Aplicar los conceptos teóricos de los módulos de introducción a la ciberseguridad y ciberseguridad101, comprendiendo como las fallas pueden comprometer la seguridad de las aplicaciones web.
* Evaluar el impacto de estas vulnerabilidades y buscar estrategias de mitigación que funcionen y sean efectivas en sistemas reales.
* Mejorar la capacidad de análisis y lógica para detectar patrones comunes o errores de seguridad en aplicaciones modernas.
  1. Resumen

Durante la auditoria de WebGoat, se examinaron diferentes módulos y ejercicios enfocados en vulnerabilidades como SQL injection, control de acceso insuficiente y exposición de datos sensibles. A través de metodologías investigadas y aprendidas, se logró identificar las principales áreas de riesgo y se documentaron los pasos realizados para asi mitigar dichas fallas. Este informe representa un paso significativo en la consolidación del aprendizaje practico, destacando la importancia de los conocimientos teóricos para la identificación y remediación de vulnerabilidades en el mundo actual.

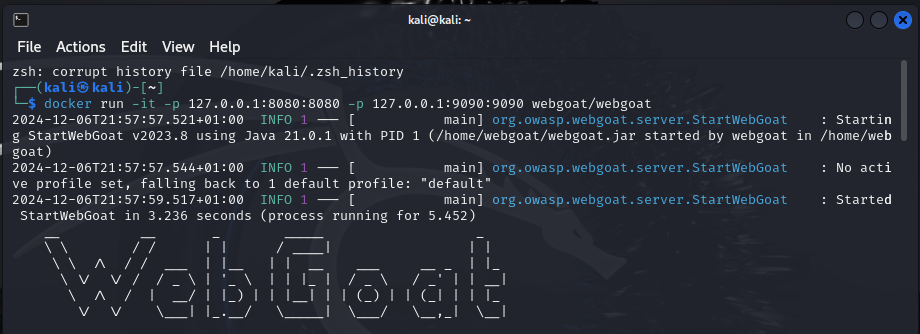
1. **METODOLOGIA DE ANALISIS (INFORMATION GATHERING)**

El análisis se llevó acabo utilizando un enfoque mixto basándonos en la recolección de información para determinar posibles puntos débiles, combinando técnicas manuales y herramientas automatizadas. Las herramientas principales utilizadas fueron OWAS ZAP, Burp Suite, Nmap.

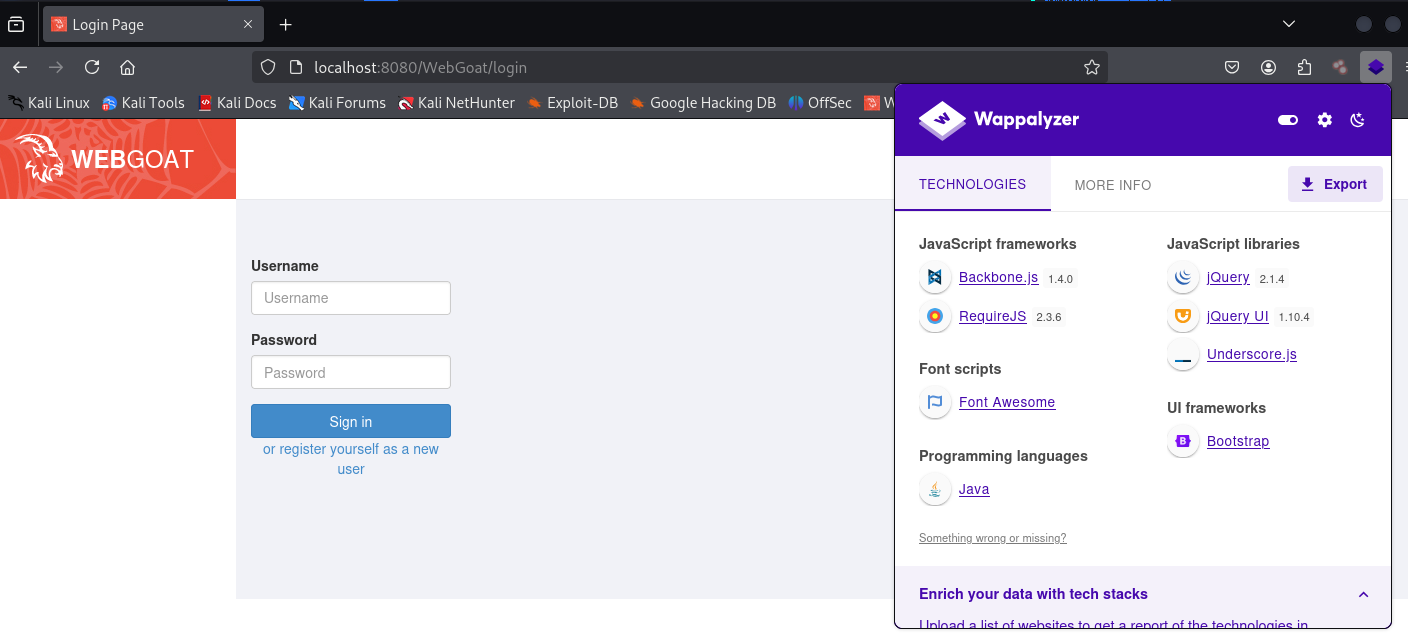
* 1. Hallazgos pasivos

En esta sección se documentan los hallazgos obtenidos mediante técnicas de recolección de información pasiva enfocada en el entorno de WebGoat. El objetivo fue identificar información pública y metadatos relacionados sin interactuar con ella directamente. Esto nos permite explotar posibles focos de ataque o configuraciones expuestas de una manera más discreta, utilizando únicamente fuentes abiertas y herramientas no intrusivas. A continuación, adjunto los hallazgos y la evidencia recopilada mediante técnicas de OSINT (Open Soucurse Intelligence) y consultas a recursos públicos.

* Se realiza el comando Docker run -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 webgoat/webgoat



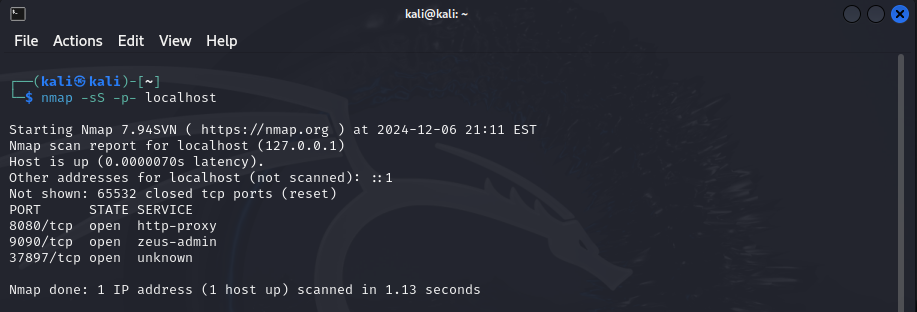
* Se busca información en la web con la extensión Wappalyzer, nos indica Lenguaje de programación, librerías JavaScript, bibliotecas JavaScript, Servidores Web entre otros.



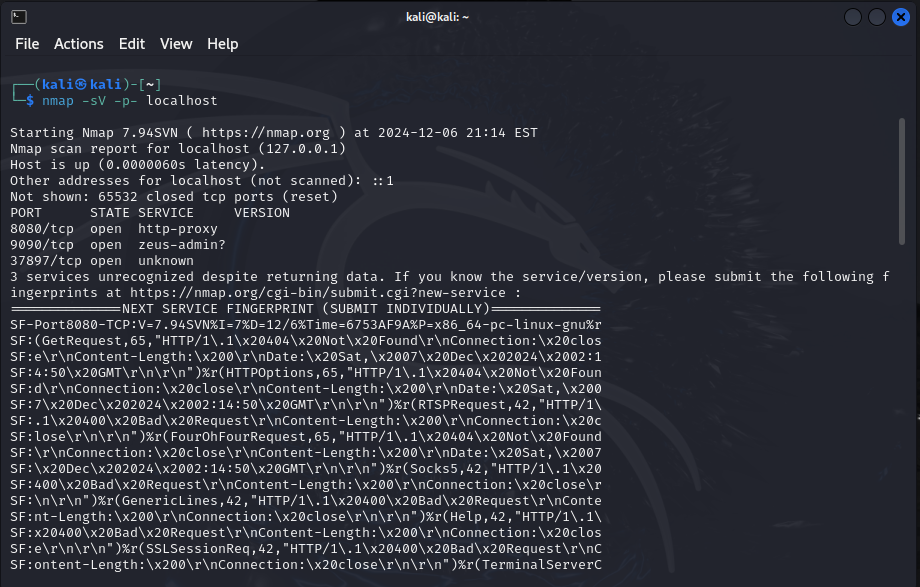
* 1. Hallazgos activos

En esta sección se presentan los resultados obtenidos mediante la recolección de información activa sobre WebGoat. Esta fase implico la interacción directa con el sistema a través de herramientas específicas para identificar servicios, puertos abiertos y configuraciones expuestas; La información recopilada complementa los hallazgos pasivos y nos permite tener una visión más detallada del entorno, lo que nos sirve para identificar posibles vulnerabilidades explotables. A continuación, se detalla los pasos de las técnicas utilizadas junto con la evidencia de los resultados.

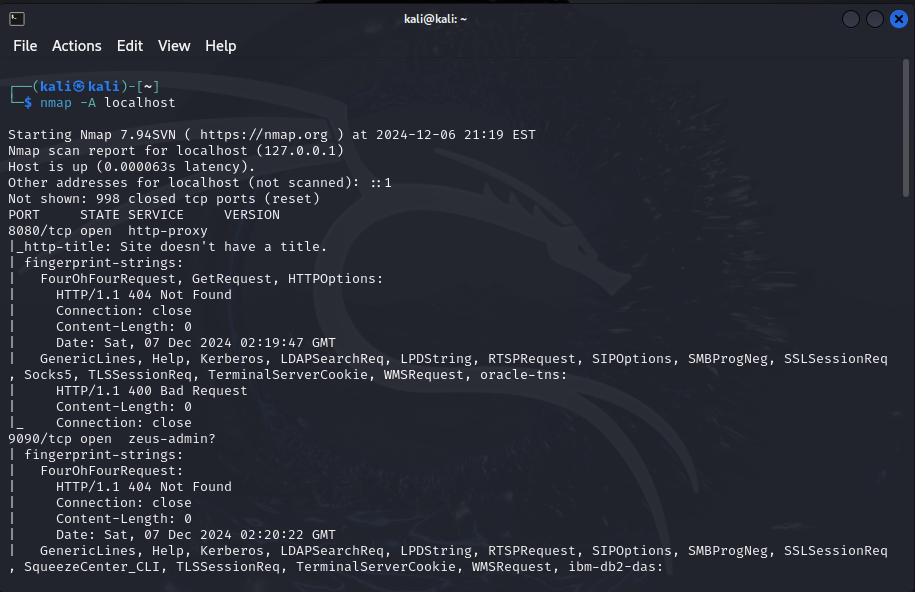
* Se realiza el comando nmap -sS -p- localhost nos permite identificar los puertos abiertos y servicios en ejecución.

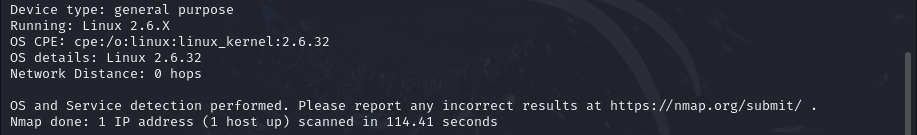


* Se realiza el comando nmap -sV -p- localhost detalla la detección de servicios.



* Se realiza el comando nmap -A localhost nos permite identificar el sistema operativo, servicios activos.

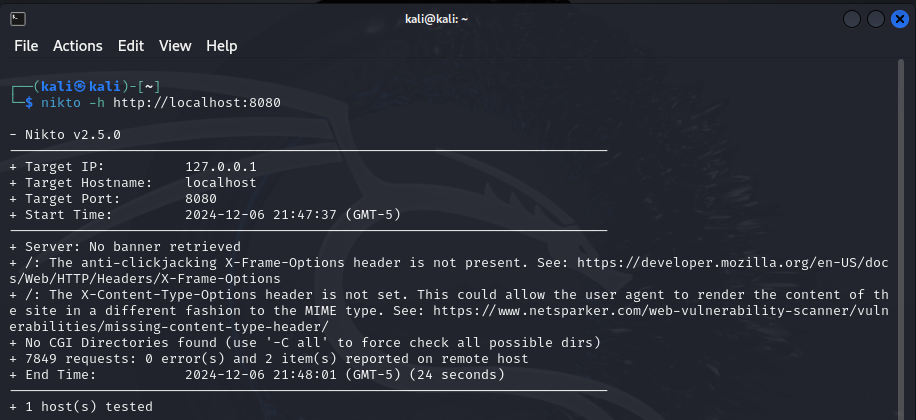




* 1. Otros hallazgos
* Se realiza el comando (nikto -h http://localhost:8080)

Nos resalta algunos problemas de configuración de seguridad relacionados con los encabezados HTTP en el servidor que se está corriendo en localhost.

* La ausencia del encabezado X-Frame-Options permite que el sitio sea vulnerable a ataques de **Clickjacking.**
* La ausencia del encabezado X-Content-Type-Options puede permitir que navegadores intenten interpretar el contenido en un formato distinto al especificado por el servidor, abriendo la puerta a ataques como **MIME sniffing.**



1. **VULNERABILIDADES IDENTIFICADAS**

Durante la auditoría realizada a WebGoat, se identificaron varias vulnerabilidades que reflejan configuraciones faltantes de seguridad comunes en aplicaciones web. Estas fallas no solo exponen el sistema a posibles ataques, sino que también sirven como ejemplo para comprender como prevenir y mitigar riesgos en entornos reales.

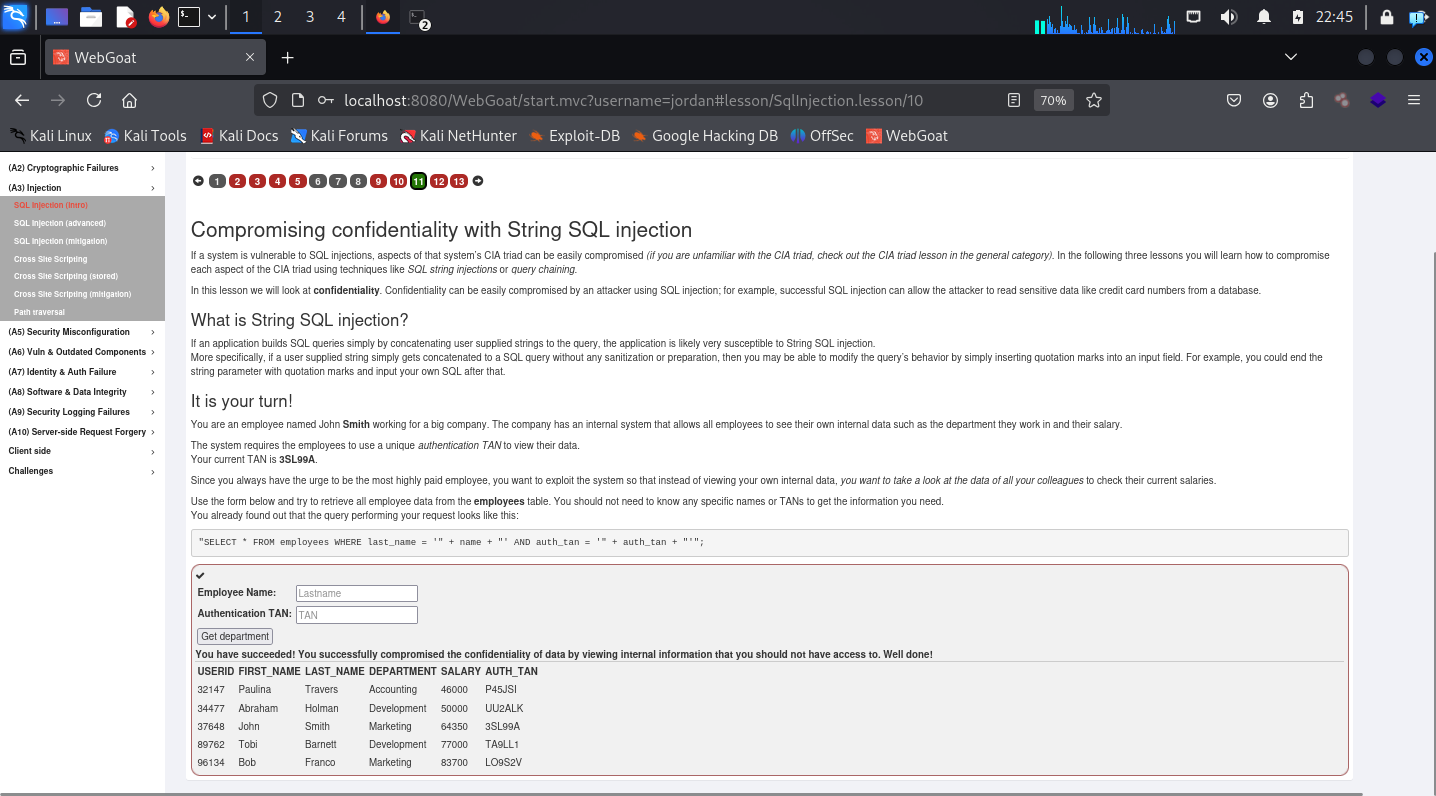
A continuación, presentare las vulnerabilidades detectadas acompañadas de evidencias especificas obtenidas durante el análisis y recomendaciones para abordar cada caso.

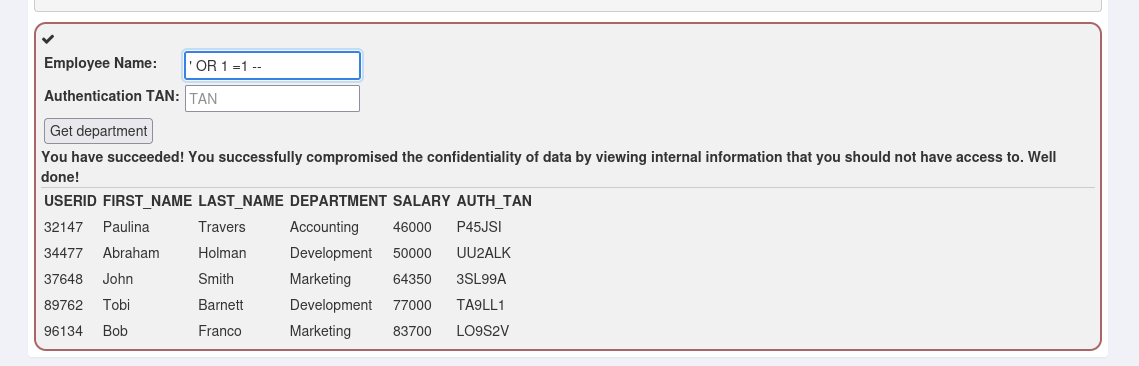
* 1. A3 Inyección SQL (SQL Injection)

La inyección SQL ocurre cuando una aplicación permite a un atacante manipular consultas SQL ejecutadas en la base de datos mediante entradas no validadas, este fallo puede comprometer datos confidenciales, modificar información o incluso en algunos casos otorgar acceso total al sistema.

Prueba #1 realizada:

* Ingresé la cadena maliciosa (‘OR 1 = 1 --) en el campo de inicio de sesión sin ingresar una contraseña valida.
* El servidor respondió con acceso permitido, demostrando que la consulta SQL fue manipulada.





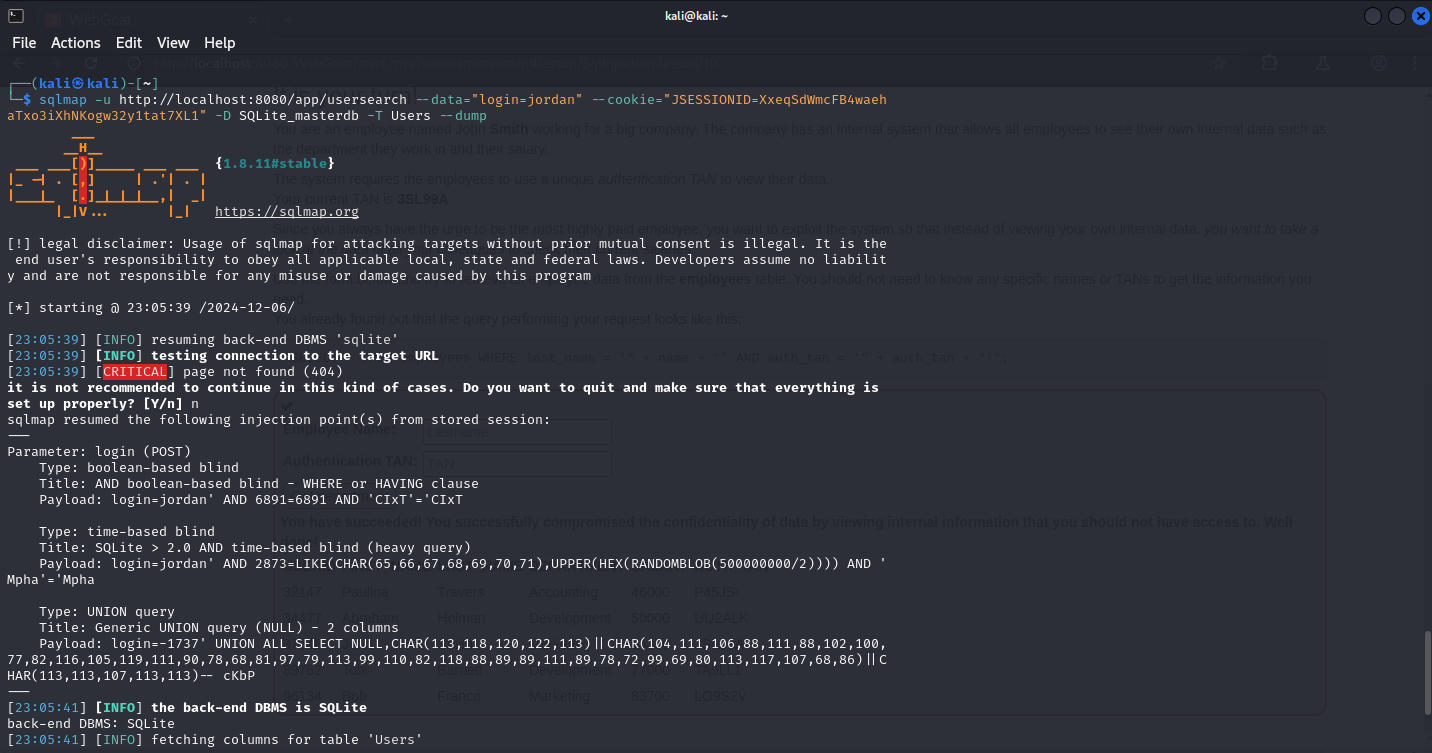
Prueba #2 realizada:

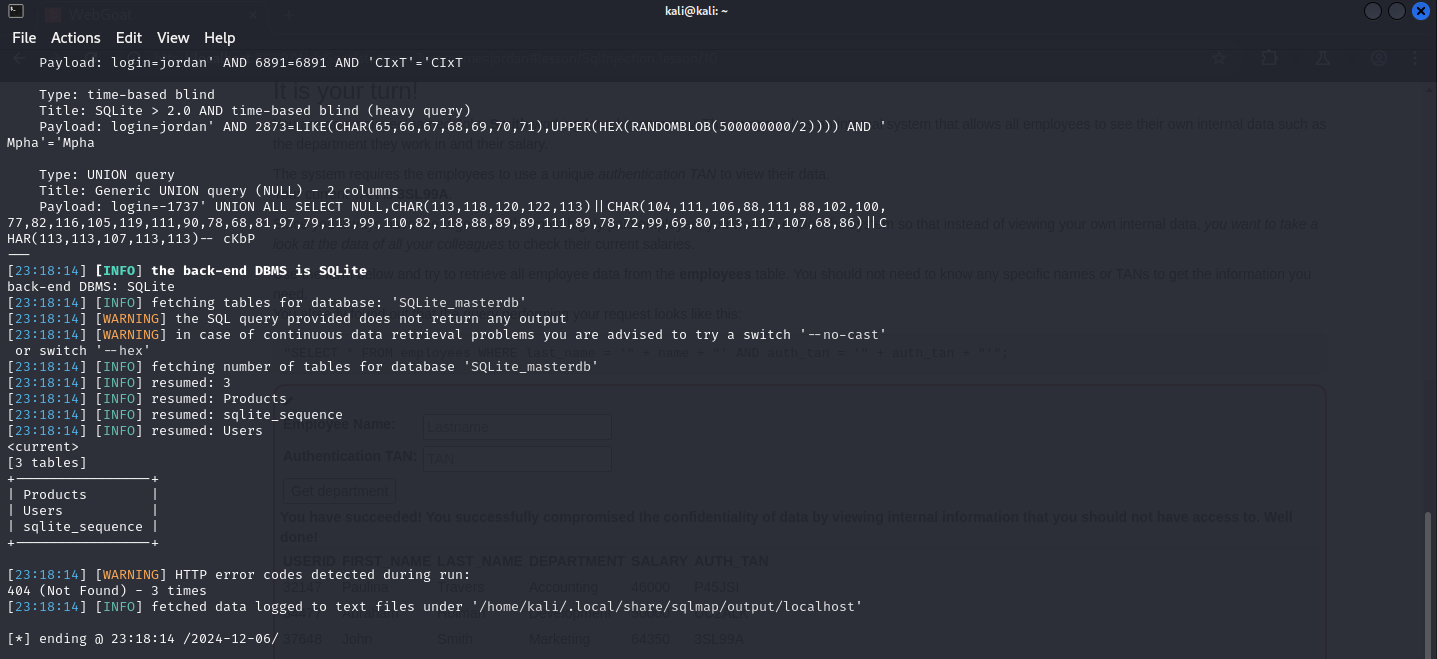
Ingrese los siguientes comandos en la terminal de kali Linux, y utilice la herramienta Burp Suite con el fin de adquirir más información y fallos.

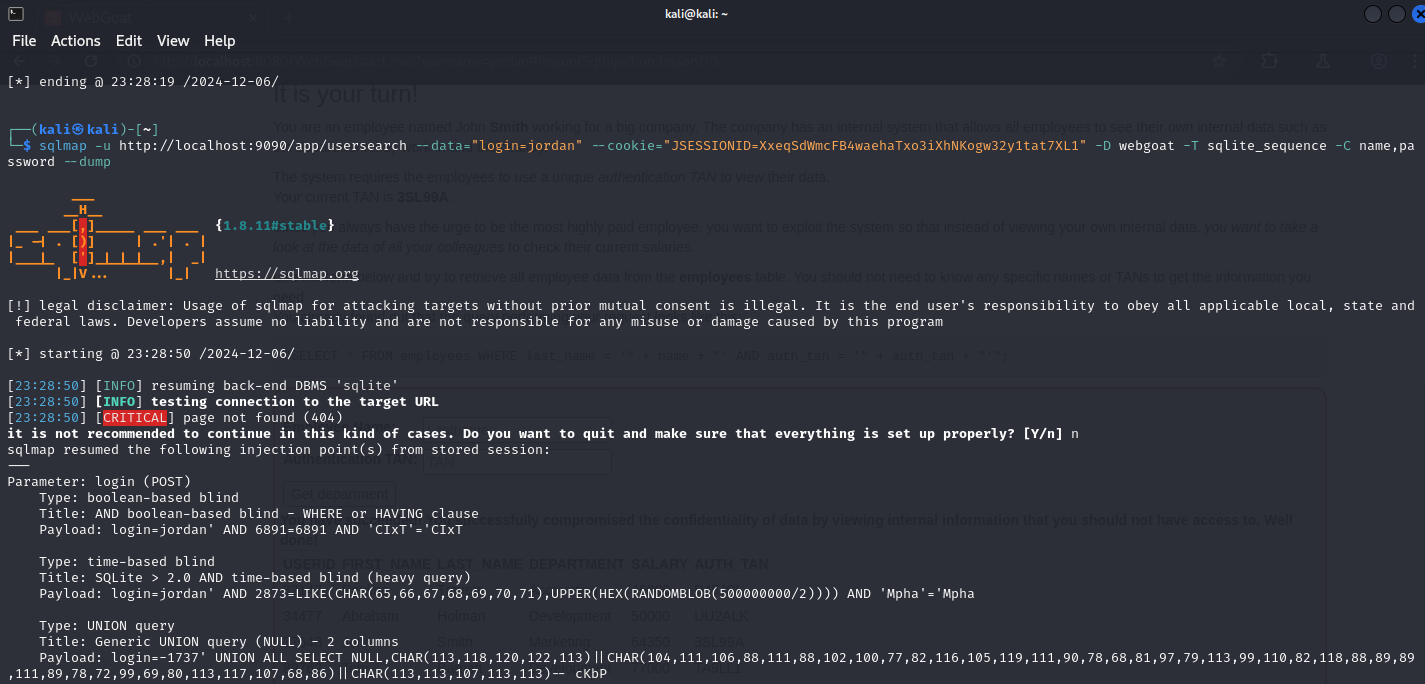
* sqlmap -u http://localhost:9090/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D webgoat -T users –columns
* sqlmap -u http://localhost:8080/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D SQLite\_masterdb -T Users –dump
* sqlmap -u http://localhost:9090/app/usersearch --data="login=jordan" --cookie="JSESSIONID=XxeqSdWmcFB4waehaTxo3iXhNKogw32y1tat7XL1" -D webgoat -T sqlite\_sequence -C name,password –dump

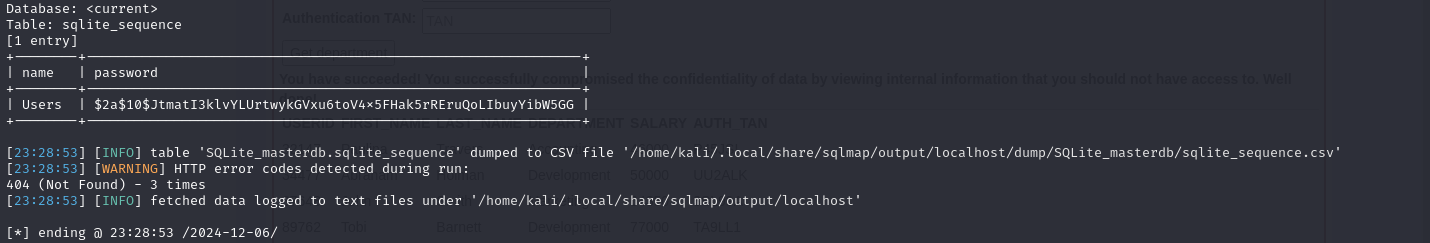
De lo cual se obtiene la siguiente información:

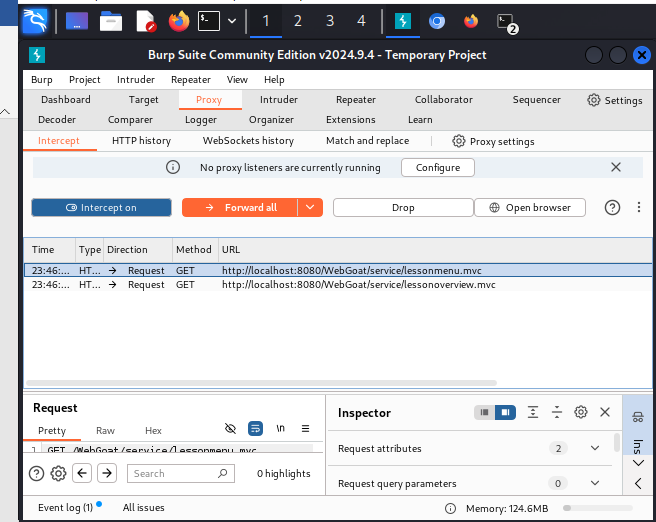
* Tres tablas denominadas (Products, Users, sqlite\_sequence) demostrando la vulnerabilidad ya mencionada.
* Un usuario y una contraseña (en formato Hash) ($2a$10$JtmatI3klvYLUrtwykGVxu6toV4x5FHak5rREruQoLIbuyYibW5GG)











* 1. A3 Cross-Site Scripting (XSS)

El XSS ocurre cuando una aplicación web permite que un atacante inyecte código JavaScript malicioso en una página visitada por otros usuarios. Esto sucede porque la entrada proporcionada por el usuario no es correctamente valida antes de mostrarse.

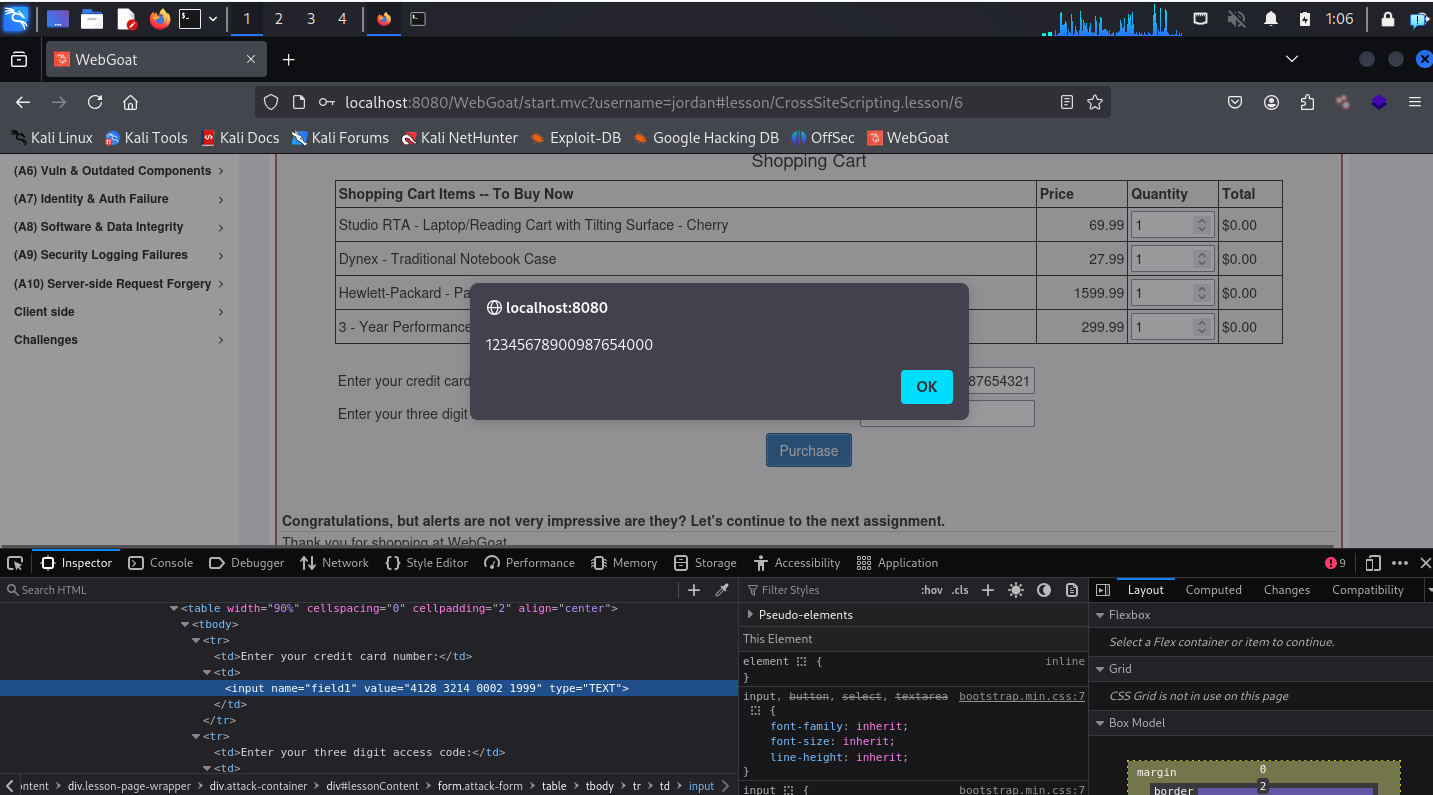
En WebGoat, se simula este ataque permitiendo que el usuario inyecte scripts en campos de entrada o URL. El impacto de esta vulnerabilidad puede incluir el robo de cookies de sesión, redirección a sitios maliciosos o la manipulación de contenido mostrado a otros usuarios.

Prueba realizada:

* Ingresé el siguiente script malicioso en un campo de texto vulnerable inspeccionando y tocando el código fuente.

((“><script>alert(12345678900987654000)</script>))

* Este script fue procesado y ejecutado por el navegador al cargar la página, generando una alerta que confirma la existencia de la vulnerabilidad al inyectar código.



* 1. A5 Security Misconfiguration (Configuración de Seguridad Incorrecta)

La configuración incorrecta de seguridad ocurre cuando el sistema, servidor o aplicación no esta configurada adecuadamente para protegerse contra acceso no autorizado o explotación de recursos.

El impacto de esta vulnerabilidad incluye la exposición de datos sensibles, escalar en los privilegios y la apertura de puertas para otros posibles ataques, como la inyección de comandos o el robo de información.

Prueba realizada:

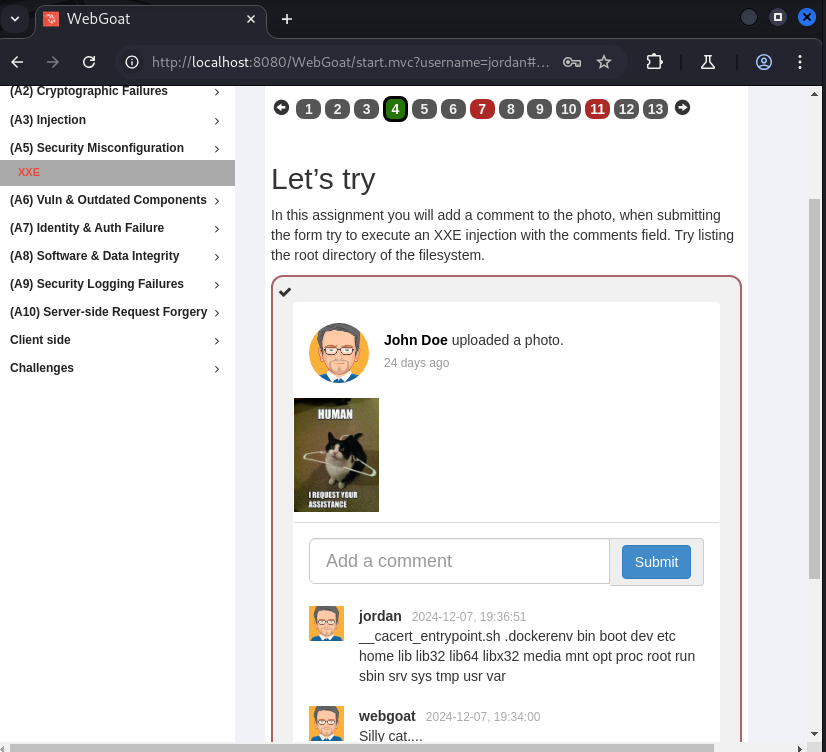
* Accedí a áreas de configuración que no requieren autenticación adecuada o donde se utilizan credenciales predeterminadas.
* Verifique los recursos expuestos como directorios no protegidos o archivos de configuración accesibles públicamente.
* Utilice este código:

<?xml version="1.0"?>

<!DOCTYPE comment [<!ENTITY root SYSTEM "file:///">]>

<comment> <text>&root;</text>

</comment>



* 1. A6 Vuln & outdated Components

Esta vulnerabilidad se produce cuando una aplicación utiliza bibliotecas, frameworks, o componentes que contienen fallos de seguridad conocidos debido a versiones desactualizadas.

En WebGoat, se identificó que el servidor utiliza componentes obsoletos con vulnerabilidades conocidas. Esto permite a los atacantes explotar estos fallos para comprometer la aplicación o el servidor.

Por ejemplo, una biblioteca Java obsoleta puede tener vulnerabilidades críticas que permitan ataques de ejecución remota de código o escalamiento de privilegios.

Impacto:

* Compromiso del sistema por exploits conocidos.
* Pérdida de datos confidenciales debido a brechas de seguridad.
* Escalamiento de ataques mediante la explotación de dependencias.

Prueba realizada:

* La deserialización del XML por parte de XStream convirtió el XML en un objeto Java.
* Durante este proceso, se creó un objeto ProcessBuilder que intentó ejecutar el comando especificado.
* El sistema, vulnerable a deserialización insegura, ejecutó el programa calc.exe.

<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>

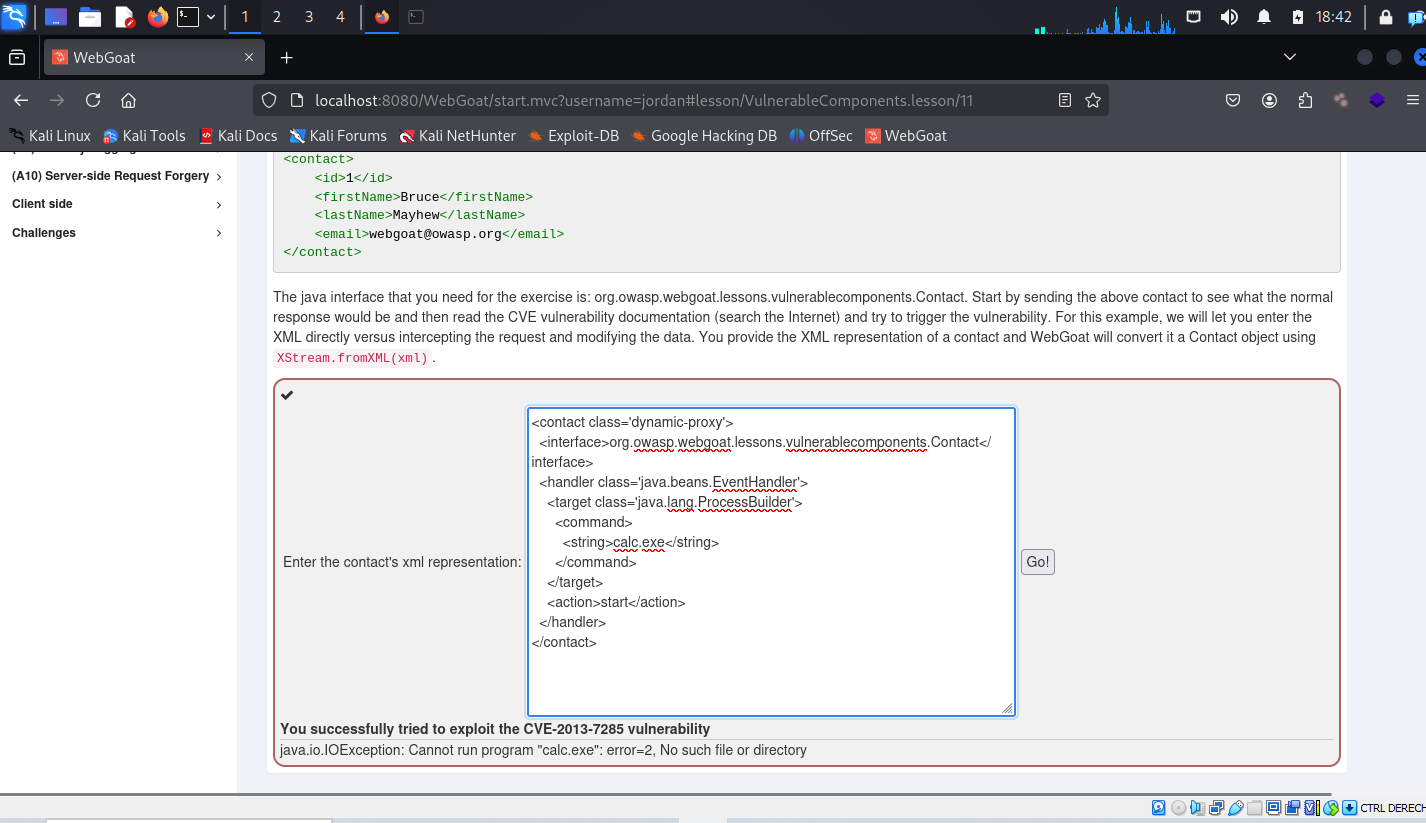
</command>

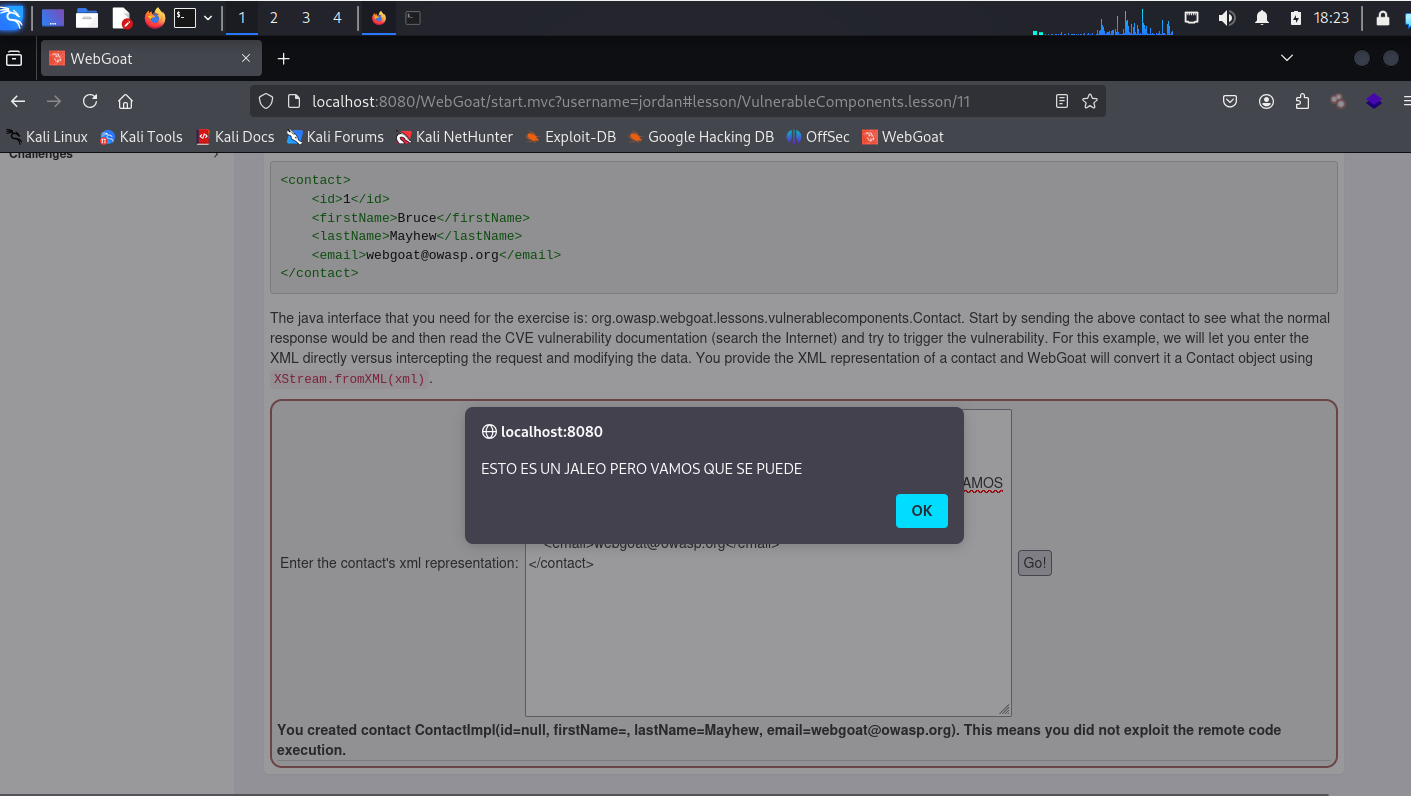
</target>

<action>start</action>

</handler>

</contact>





* 1. A7 Identity & Auth Failure – Secure Passwords

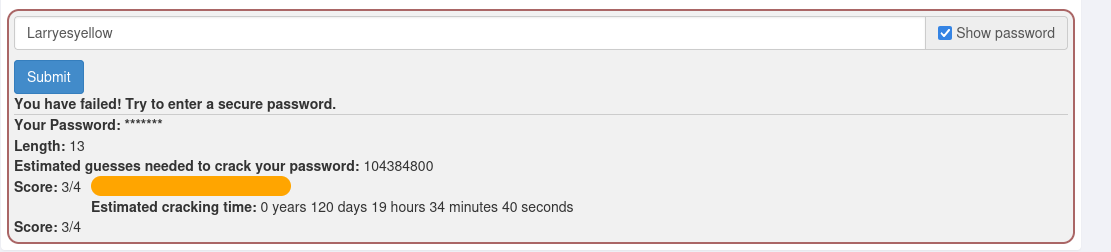
Este fallo de seguridad ocurre cuando una aplicación permite a los usuarios establecer contraseñas que no cumplen con las políticas de seguridad (como longitud mínima, uso de caracteres especiales, y evitación de contraseñas comunes). Las contraseñas débiles facilitan ataques de fuerza bruta y otros métodos de cracking.

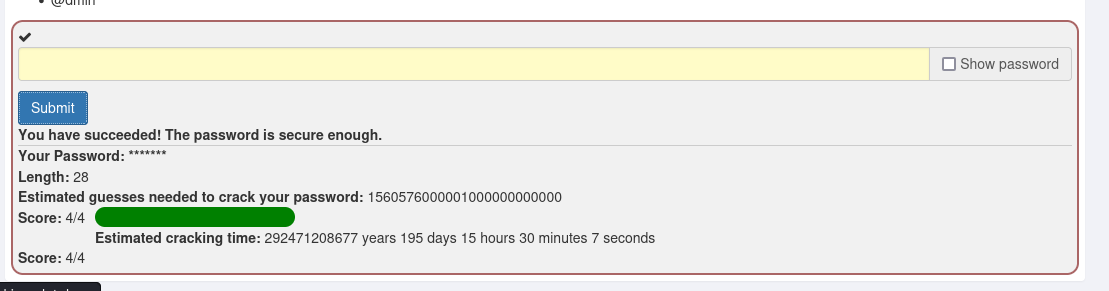
Impacto en la seguridad:

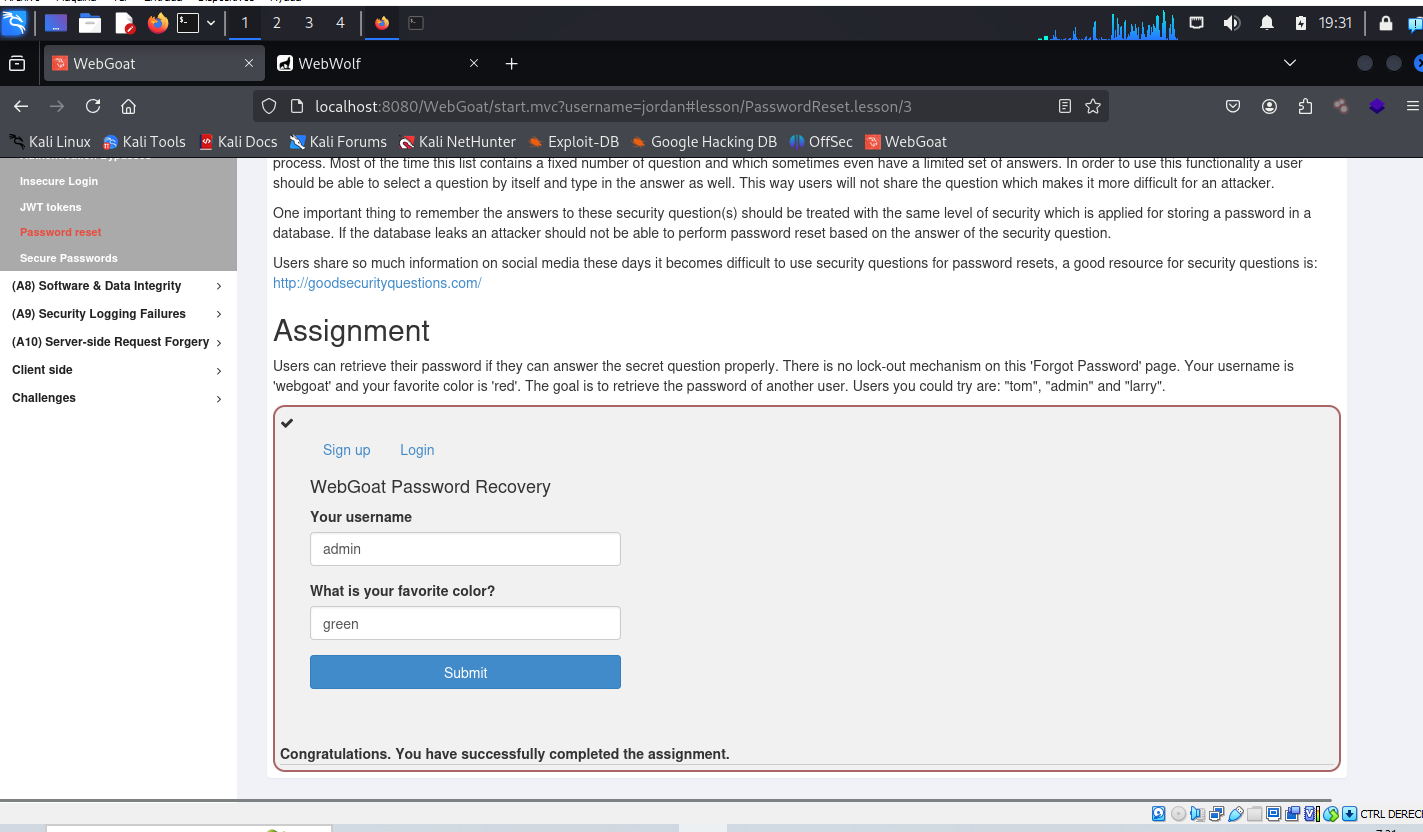
* Compromiso de cuentas: Los atacantes pueden adivinar contraseñas débiles para acceder a cuentas de usuario.
* Escalación de privilegios: Una vez dentro, podrían explotar permisos de administrador.
* Pérdida de datos confidenciales: Acceso no autorizado a información sensible.

Prueba realizada:

* Se intenta crear un usuario con contraseña débil, como ‘123456’ la aplicación rechaza esta contraseña con advertencia, también nos demuestra como una contraseña débil es vulnerable frente a ataques de diccionarios o fuerza bruta.
* Se ejecuta un ataque simulado y nos muestra cuantas contraseñas podrían crackearse en pocos segundos.







1. **ANALISIS DE RIESGOS**
   1. Resumen de vulnerabilidades

Este análisis evalúa las vulnerabilidades encontradas en el entorno de practica WebGoat, relacionadas con riesgos comunes en aplicaciones reales y como mitigar dichos riesgos.

A3 Injection - SQL Injection

* Descripción: Permite a un atacante manipular consultas SQL para acceder, modificar o eliminar datos no autorizados.
* Impacto en organizaciones reales: Puede comprometer datos confidenciales, como información personal o financiera, afectando la reputación y cumplimiento legal.
* Severidad: Crítica

A3 Injection - Cross-Site Scripting (XSS)

* Descripción: Inserta scripts maliciosos en sitios web legítimos, afectando a los usuarios.
* Impacto en organizaciones reales: Robo de credenciales, suplantación de identidad y explotación de sesiones.
* Severidad: Alta

A5 Security Misconfiguration

* Descripción: Configuraciones inseguras, como permisos excesivos o información sensible expuesta.
* Impacto en organizaciones reales: Facilita ataques posteriores y compromete la seguridad general del sistema.
* Severidad: Alta

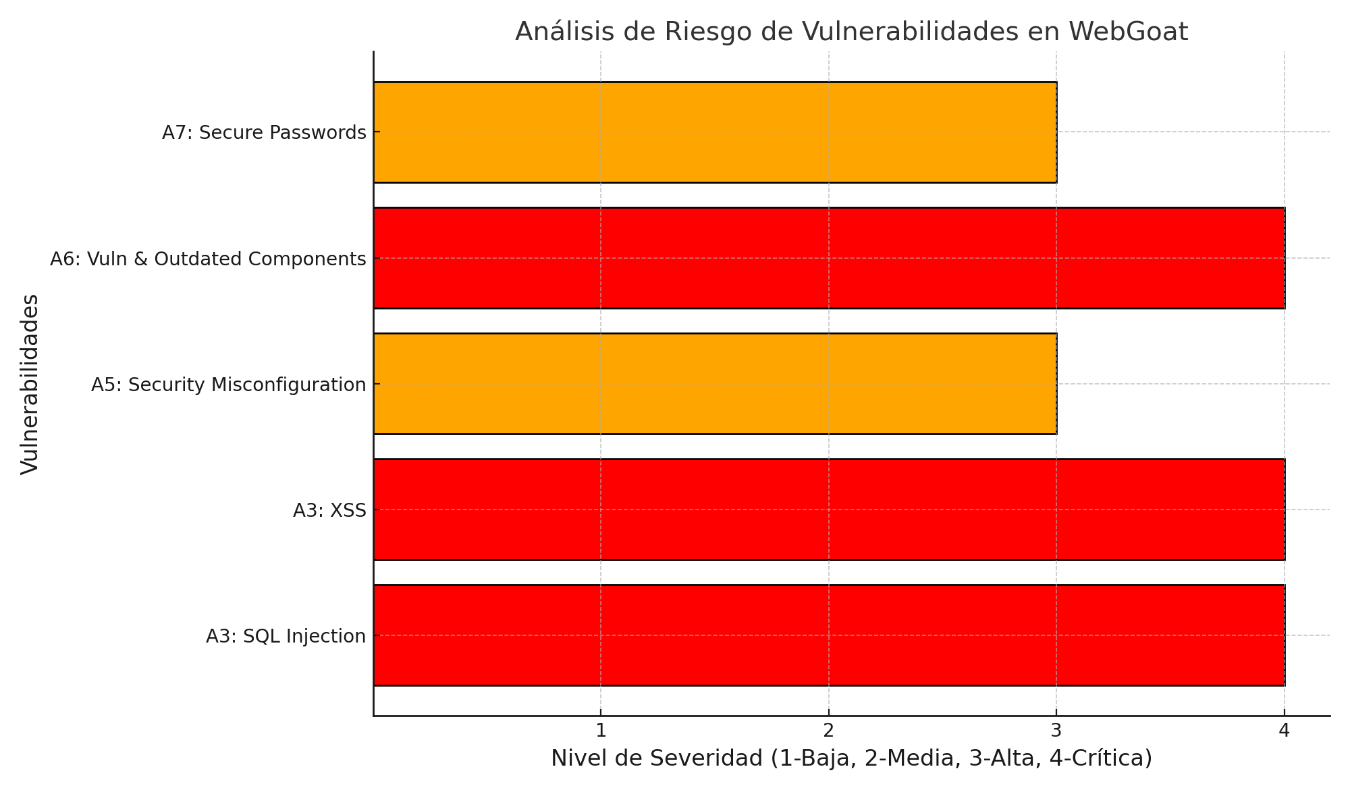
A6 Vulnerable & Outdated Components

* Descripción: Uso de componentes desactualizados o con fallos conocidos.
* Impacto en organizaciones reales: Los atacantes pueden aprovechar vulnerabilidades conocidas para infiltrarse.
* Severidad: Crítica

A7 Identity & Auth Failure - Secure Passwords

* Descripción: Falta de gestión adecuada de contraseñas, como almacenamiento inseguro o requisitos débiles.
* Impacto en organizaciones reales: Permite accesos no autorizados y aumenta el riesgo de ataques de fuerza bruta.
* Severidad: Alta

En la siguiente grafica se puede ver reflejado el impacto de las vulnerabilidades y otros datos claves y no menos importantes:



1. **RECOMENDACIONES**
   1. A3 Inyección SQL (SQL Injection)

* Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos <https://owasp.org/www-community/attacks/SQL_Injection>
* Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:

1. Validar y corregir entradas de usuario:

Utilizar funciones que limpien datos antes de incluirlos en consultas SQL (por ejemplo, usar declaraciones preparadas o consultas parametrizadas).

1. Restringir privilegios en la base de datos:

El usuario que ejecuta las consultas desde la aplicación debe tener privilegios mínimos necesarios.

1. Implementar un WAF (Web Application Firewall):

Ayuda a bloquear patrones comunes de inyección SQL.

1. Realizar pruebas periódicas de seguridad:

Utilizar herramientas como SQLmap para identificar posibles vulnerabilidades en el sistema.

* 1. A3 Cross-Site Scripting (XSS)
* Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>
* Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:

1. Validar y corregir entradas:

Verificar todos los datos ingresados por el usuario en el lado del servidor y del cliente. Utilizar funciones que eliminen caracteres peligrosos y de código como **< > “ ‘**

1. Evitar el uso de funciones inseguras:

No utilizar funciones eval() o innerHTML para procesar datos ingresados por el usuario.

1. Pruebas regulares de seguridad:

Realizar escaneos automáticos y manuales para detectar XSS con herramientas existentes como OWASP ZAP o Burp Suite.

* 1. A5 Security Misconfiguration (Configuración de Seguridad Incorrecta)
* Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>
* Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:

1. Eliminar configuraciones por defecto:

Revisar que no existan configuraciones predeterminadas en el entorno de producción.

1. Restringir el acceso a archivos y directorios sensibles:

Proteger los archivos como config, logs o backup con permisos adecuados y evitar su exposición pública.

1. Pruebas regulares de seguridad:

Realizar escaneos automáticos y manuales para detectar configuraciones inseguras utilizando herramientas existentes como OWASP ZAP o Nessus.

* 1. A6 Vuln & outdated Components
* Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos

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

* Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:

1. Actualizaciones regulares:

Asegurarse de mantener actualizadas las bibliotecas y dependencias utilizadas por la aplicación además de configurar alertas para la actualización de seguridad y parches en tiempo real.

1. Prueba de seguridad:

Integrar análisis estático y dinámico en el ciclo de desarrollo para identificar componentes vulnerables.

1. Eliminar componentes no utilizados:

Realizar limpiezas periódicas para eliminar bibliotecas y componentes que ya no son necesarios.

* 1. A7 Identity & Auth Failure – Secure Passwords
* Se sugiere inicialmente validar la siguiente página para mitigar un poco más los riesgos

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

* Por otra parte, para prevenir esta vulnerabilidad en aplicaciones reales, se recomienda:

Para mitigar estas vulnerabilidades, es crucial implementar políticas de seguridad para contraseñas.

1. Requisito de contraseñas fuertes:

Mínimo 12 caracteres, combinación de letras mayúsculas y minúsculas, números y símbolos.

1. Monitoreo y auditoria:

Detectar intentos de inicio de sesión sospechosos además enviar alertas al usuario cuando se detecten accesos desde ubicaciones desconocidas.

1. Educación al usuario:

Asegurarse de que los usuarios comprendan los riesgos de contraseñas débiles, proveer generadores de contraseñas seguras dentro de la aplicación.

1. **CONCLUSION**

El análisis de las vulnerabilidades identificadas en WebGoat me han permitido adquirir una comprensión practica de los riesgos más comunes en las aplicaciones web. A través de la exploración de fallos como **Inyección SQL, Cross-Site Scripting (XSS), Security Misconfiguration, Componentes Vulnerables y Desactualizados, Fallos en Identidad y Autenticación (Contraseñas Seguras)**, se han destacado no solo las debilidades técnicas sino también los impactos críticos que estas vulnerabilidades pueden tener en entornos empresariales reales.

Este aprendizaje ha reforzado la importancia de adoptar una mentalidad proactiva en Ciberseguridad, integrando buenas practicas en cada etapa del ciclo de la vida y del desarrollo de software. Además, subraya la necesidad de mantener los sistemas actualizados, implementar controles adecuados y realizar pruebas periódicas de seguridad.

La experiencia obtenida en este proyecto aporta una base solida para avanzar en mi carrera en TI, fomentando un enfoque ético y responsable hacia la seguridad informática. Este conocimiento es esencial para garantizar la integridad, confidencialidad y disponibilidad de los sistemas en el ámbito organizacional.

1. **REFERENCIAS Y BIBLIOGRAFIA**

* WebGoat - Herramienta de aprendizaje utilizada para simular y explotar vulnerabilidades comunes. Disponible en: <https://owasp.org/www-project-webgoat/>
* OWASP Top 10 (2021). "Guía de las vulnerabilidades más críticas en aplicaciones web". Disponible en: https://owasp.org/www-project-top-ten/
* Documentación y foros de Docker: "Ejecución de aplicaciones en contenedores para entornos controlados". Disponible en: https://www.docker.com
* "OWASP Testing Guide v4". Fundación OWASP. Manual completo sobre pruebas de penetración en aplicaciones web.
* Consultas previas realizadas con ChatGPT en relación con WebGoat y la explotación de vulnerabilidades.
* Apuntes y lecturas complementarias sobre ciberseguridad obtenidas en cursos y recursos abiertos como Coursera y edX.