Junio 30, 2024

INFORME DE AUDITORÍA DE SEGURIDAD WEB PARA

WEBGOAT

Daniel Moreno Yslán

Introducción a la ciberseguridad

KeepCoding©

Confidencial

ÍNDICE

1. ÁMBITO Y ALCANCE DE LA AUDITORÍA 1
2. INFORME EJECUTIVO 1
2.1. Breve resumen del proceso realizado 1
2.2. Vulnerabilidades destacadas 1
2.3. Conclusiones 2
2.4. Recomendaciones 3
3. INFORME EJECUTIVO 3
3.1. Reconocimiento/Information Gathering 3
3.2. Explotación de vulnerabilidades detectadas 5
3.2.1. A3 Injection – SQL Injection (Intro) – Apartado 11 5
3.2.2. A3 Injection – Cross Site Scripting – Apartado 7 10
3.2.3. A5 Security Misconfiguration – Apartado 4 11
3.2.4. A6 Vulnerabilidades y componentes obsoletos – Apartado 5 12
3.2.5. A7 Fallas en la identidad y autenticación – Contraseñas seguras – Apartado 4 13
3.3. Post – explotación 13
3.4. Posibles mitigaciones 13
3.5. Herramientas utilizadas 14

1. ÁMBITO Y ALCANCE DE LA AUDITORÍA

- Aplicación Auditada: WebGoat versión 8.1.0.
- Entorno: Ejecutado en Docker en un entorno controlado.
- Alcance: Incluye evaluación de vulnerabilidades OWASP Top 10, con foco en inyecciones SQL, XSS, configuraciones de seguridad, componentes desactualizados y fallos de autenticación.

2. INFORME EJECUTIVO

2.1. Breve resumen del proceso realizado

Se llevó a cabo una auditoría de seguridad web en la aplicación WebGoat, identificando y explotando vulnerabilidades críticas siguiendo un enfoque basado en OWASP Top 10.

2.2. Vulnerabilidades destacadas

SQL Injection (A3)

Descripción: La inyección SQL es una vulnerabilidad que ocurre cuando una aplicación web permite a los usuarios enviar entradas directamente a una consulta SQL. Si estas entradas no son adecuadamente sanitizadas, un atacante puede manipular las consultas SQL ejecutadas por la base de datos, obteniendo así acceso no autorizado a los datos, modificando información o incluso ejecutando comandos destructivos.

Esta vulnerabilidad permite que un atacante inyecte código SQL malicioso en las consultas a la base de datos. Al explotar esta debilidad, el atacante puede extraer información sensible, modificar datos, o incluso eliminar tablas completas. Es común en aplicaciones web que no validan correctamente las entradas del usuario.

Severidad: Alta (CVSS v3.1: 9.0-10.0), ya que puede comprometer completamente la confidencialidad, integridad y disponibilidad de la base de datos.

Cross Site Scripting (A3)

Descripción: Ocurre cuando las configuraciones del sistema no son adecuadas o seguras, como el uso de configuraciones por defecto, errores de permisos, o exposición de información sensible. Esto puede permitir accesos no autorizados y ataques variados.

Severidad: Media a Alta (CVSS v3.1: 5.0-8.0), ya que el impacto varía según la exposición y el tipo de error de configuración.

Security Misconfiguration (A5)

Descripción: Ocurre cuando las configuraciones del sistema no son adecuadas o seguras, como el uso de configuraciones por defecto, errores de permisos, o exposición de información sensible. Esto puede permitir accesos no autorizados y ataques variados.

Severidad: Media a Alta (CVSS v3.1: 5.0-8.0), ya que el impacto varía según la exposición y el tipo de error de configuración.

Vulnerabilidades en Componentes Obsoletos (A6)

Descripción: Se refiere al uso de bibliotecas, frameworks y otros componentes de software que no están actualizados. Estos componentes pueden tener vulnerabilidades conocidas que los atacantes pueden explotar fácilmente.

Severidad: Alta (CVSS v3.1: 7.0-9.0), ya que las vulnerabilidades conocidas pueden ser explotadas de manera efectiva y comprometer el sistema.

Fallas en Autenticación y Gestión de Identidades (A7)

Descripción: Involucra debilidades en la autenticación y gestión de contraseñas, como contraseñas débiles, falta de políticas de bloqueo, y almacenamiento inseguro de credenciales. Esto puede llevar a accesos no autorizados.

Severidad: Alta (CVSS v3.1: 7.0-9.0), dado que un fallo en la autenticación puede comprometer la seguridad de toda la aplicación.

2.3. Conclusiones

La aplicación presenta varias vulnerabilidades críticas que podrían ser explotadas para comprometer la integridad y confidencialidad de los datos. Es fundamental abordar estos problemas, ya que:

- SQL Injection y Cross-Site Scripting permiten a los atacantes acceder y manipular información sensible.
- Security Misconfiguration y Componentes Obsoletos facilitan el acceso no autorizado y aumentan el riesgo de explotación por vulnerabilidades conocidas.
- Fallas en Autenticación y Gestión de Identidades ponen en riesgo el control de acceso, exponiendo la aplicación a accesos no autorizados.

Es crucial implementar medidas de seguridad adecuadas para mitigar estos riesgos y proteger los datos de los usuarios.

2.4. Recomendaciones

Implementar validaciones en el lado del servidor: Asegúrate de validar y sanitizar todas las entradas de usuario para prevenir inyecciones SQL y XSS.

Actualizar componentes y bibliotecas obsoletas: Mantén actualizados todos los frameworks y bibliotecas para evitar vulnerabilidades conocidas.

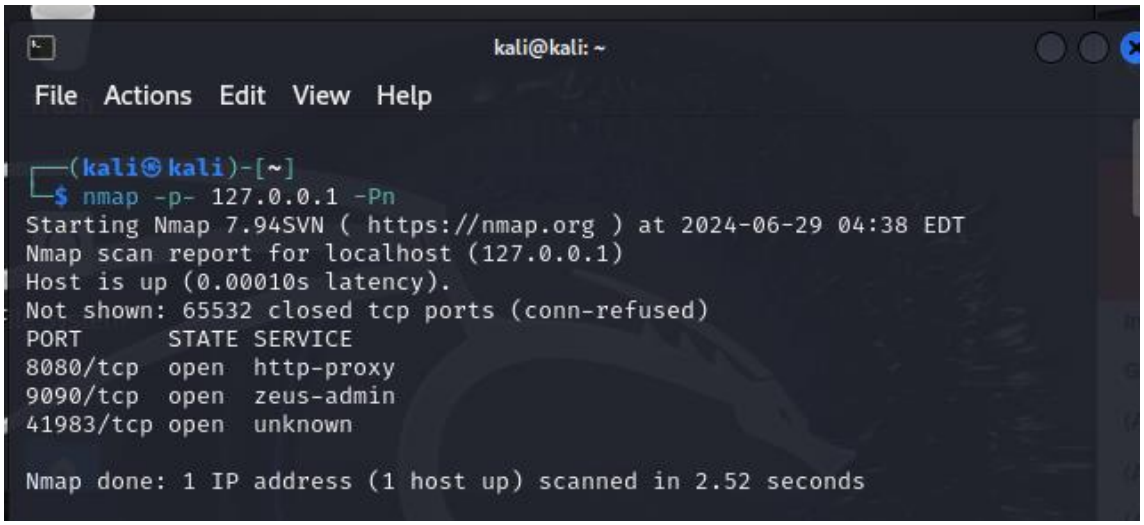
Configurar adecuadamente las políticas de seguridad: Revisa y mejora las configuraciones del servidor, deshabilitando servicios innecesarios y aplicando las mejores prácticas de seguridad.

Fortalecer los mecanismos de autenticación: Implementa políticas de contraseñas robustas, utiliza autenticación multifactor y protege las credenciales adecuadamente.

3. INFORME EJECUTIVO

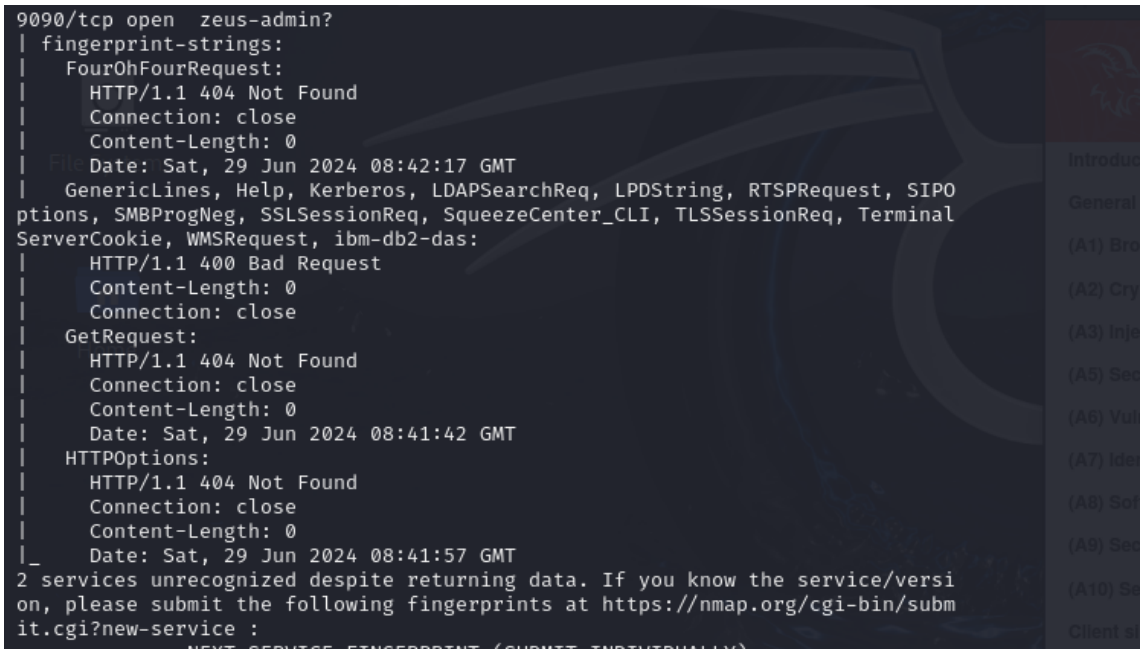
3.1. Reconocimiento/Information Gathering

Utilizamos Nmap para escanear puertos abiertos, identificando servicios y posibles vectores de ataque. Además, empleamos Burp Suite para recolectar información detallada sobre las tecnologías utilizadas en la aplicación WebGoat, como frameworks, bibliotecas y configuraciones del servidor. Esto nos ayudó a comprender mejor la superficie de ataque y a identificar potenciales vulnerabilidades.



```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -O 127.0.0.1
[sudo] password for kali:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-29 04:39 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00013s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE
8080/tcp open  http-proxy
9090/tcp open  zeus-admin
No exact OS matches for host (If you know what OS is running on it, see https
://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.94SVN%E=4%D=6/29%OT=8080%CT=1%CU=38370%PV=N%DS=0%DC=L%G=Y%TM=66
OS:7FC85F%P=x86_64-pc-linux-gnu)SEQ(SP=101%GCD=1%ISR=106%TI=Z%CI=Z%II=I%TS=
OS:A)SEQ(SP=101%GCD=1%ISR=107%TI=Z%CI=Z%II=I%TS=A)SEQ(SP=102%GCD=1%ISR=107%
OS:TI=Z%CI=Z%II=I%TS=A)SEQ(SP=102%GCD=3%ISR=106%TI=Z%CI=Z%II=I%TS=A)OPS(O1=
OS:MFFD7ST11NW7%O2=MFFD7ST11NW7%O3=MFFD7NNT11NW7%O4=MFFD7ST11NW7%O5=MFFD7ST
OS:11NW7%O6=MFFD7ST11)WIN(W1=8200%W2=8200%W3=8200%W4=8200%W5=8200%W6=8200)E
OS:CN(R=Y%DF=Y%T=40%W=8200%O=MFFD7NNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%
OS:F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T
OS:5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=
OS:Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF
OS:=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40
OS:%CD=S)
```

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -A 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-29 04:41 EDT
Stats: 0:01:44 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.65% done; ETC: 04:43 (0:00:00 remaining)
Stats: 0:01:52 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.65% done; ETC: 04:43 (0:00:00 remaining)
Stats: 0:02:02 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.65% done; ETC: 04:43 (0:00:00 remaining)
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00015s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
8080/tcp open  http-proxy
|_http-title: Site doesn't have a title.
| fingerprint-strings:
|   FourOhFourRequest, GetRequest, HTTPOptions:
|     HTTP/1.1 404 Not Found
|     Connection: close
|     Content-Length: 0
|     Date: Sat, 29 Jun 2024 08:41:42 GMT
|   GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SIPO
ptions, SMBProgNeg, SSLSessionReq, Socks5, TLSSessionReq, TerminalServerCooki
e, WMSRequest, oracle-tns:
|     HTTP/1.1 400 Bad Request
|     Content-Length: 0
|_    Connection: close
```



Podemos observar que los puertos 8080, 9090 y 41983 se encuentran abiertos.

También, gracias al escaneo, sabemos que en el puerto 8080 hay un servicio http-proxy, en el 9090 un servicio Zeus-admin y que en el 41983 hay un servicio desconocido.

3.2. Explotación de vulnerabilidades detectadas

3.2.1. A3 Injection – SQL Injection (Intro) – Apartado 11

Antes de explicar el Apartado 11 explicaremos las vulnerabilidades previas que nos encontramos en A3 Injection – SQL Injection (Intro).

El ejercicio presenta una página web que permite a los usuarios buscar información de empleados en una base de datos ingresando un ID de empleado. La consulta SQL utilizada podría ser similar a la siguiente:

```
SELECT * FROM employees WHERE id = 'user_input';
```

El usuario introduce un ID de empleado en el campo de entrada.
La aplicación inserta esta entrada directamente en la consulta SQL sin validación.
Ejemplo de entrada maliciosa: 1' OR '1'='1

```
SELECT * FROM employees WHERE id = '1' OR '1'='1';
```

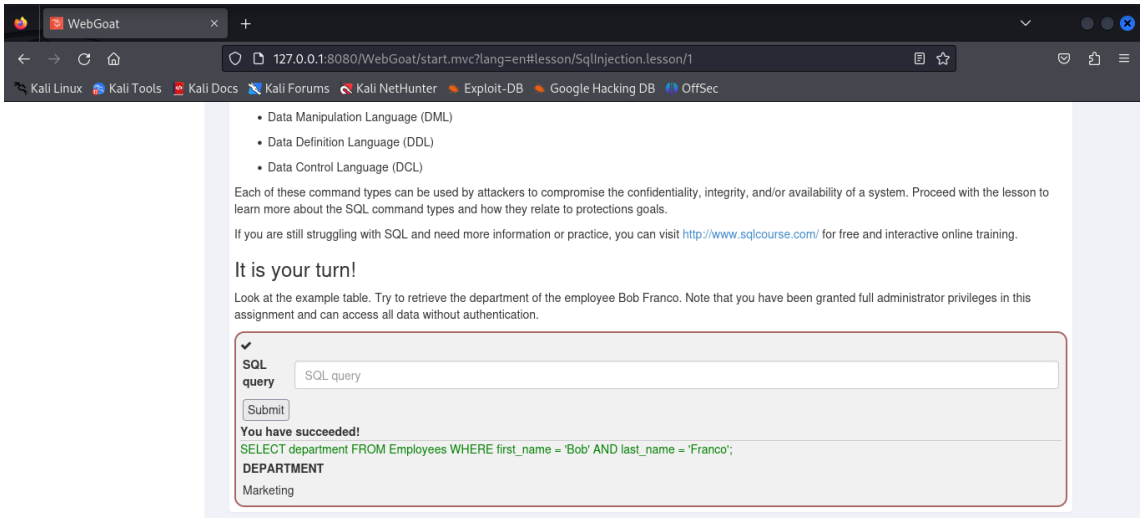
Esta consulta siempre será verdadera ('1'='1' siempre es verdadero), lo que permite al atacante obtener todos los registros de la tabla employees.

Un atacante puede utilizar la inyección SQL para acceder a datos confidenciales almacenados en otras tablas.

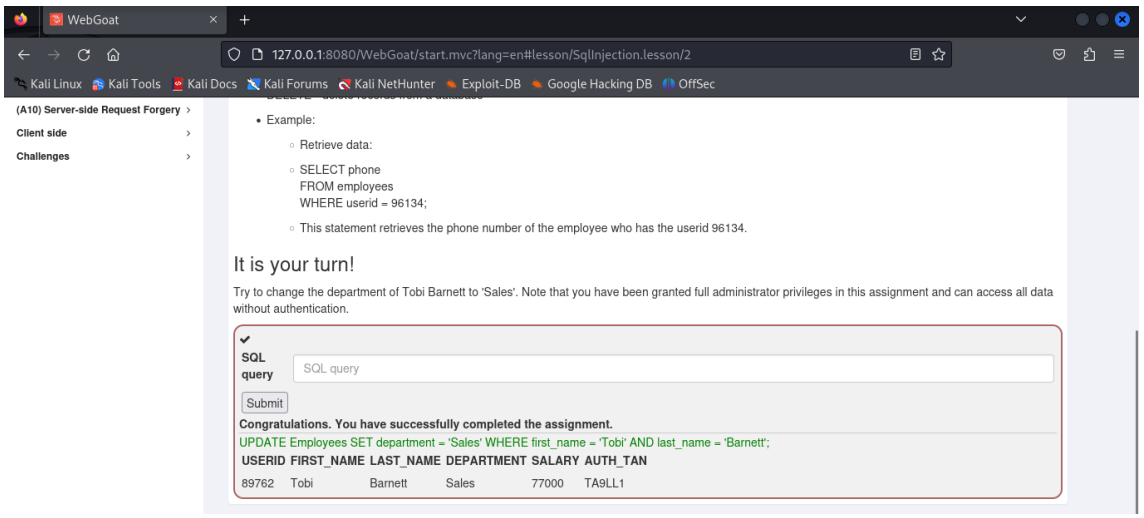
Ejemplo de entrada maliciosa: 1'; SELECT * FROM users;--

```
SELECT * FROM employees WHERE id = '1'; SELECT * FROM users;--';
```

Esto ejecuta múltiples consultas, la primera busca el ID 1 en employees, y la segunda obtiene todos los datos de la tabla users.

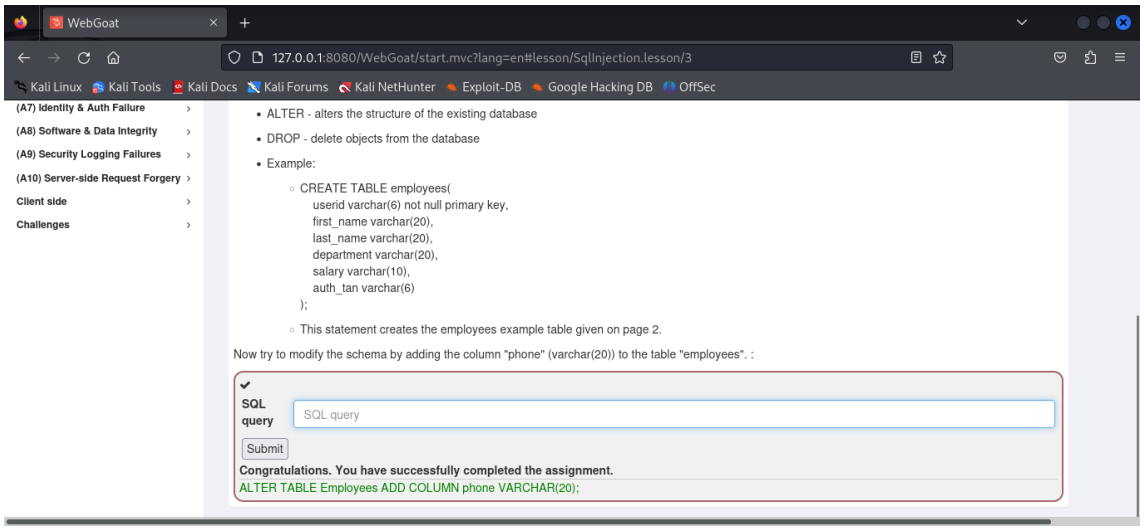


En este caso utilizamos el comando SELECT department FROM Employees WHERE first_name = 'Bob' AND last_name = 'Franco' en el cual seleccionamos la columna de departamentos que queremos consultar de la tabla empleados y comparamos nombres y apellidos hasta que encontramos la información del empleado Bob Franco.



La inyección SQL no solo se usa para obtener datos, sino también para manipularlos.

En la lección 2 podemos ver cómo, con el comando UPDATE Employees SET department = 'Sales' WHERE first_name = 'Tobi' AND last_name = 'Barnett' modificamos el departamento del empleado Tobi Barnett en la tabla empleados.



En este caso modificamos el esquema añadiendo una nueva columna en la tabla empleados con el siguiente comando:

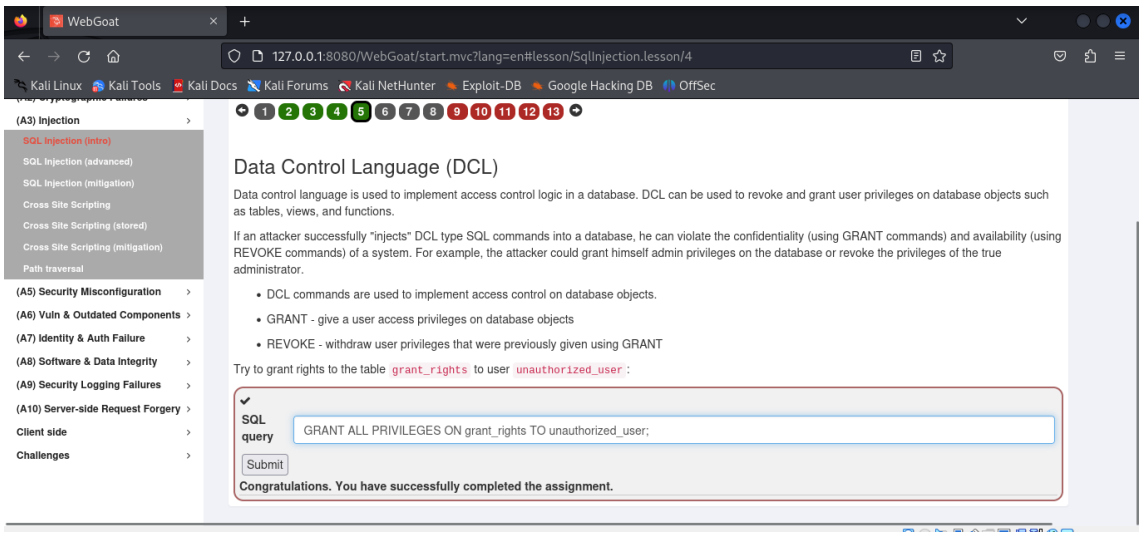
ALTER TABLE Employees ADD COLUMN pone VARCHAR(20)

En la siguiente lección se trata el tema de DCL (Data Control Language):

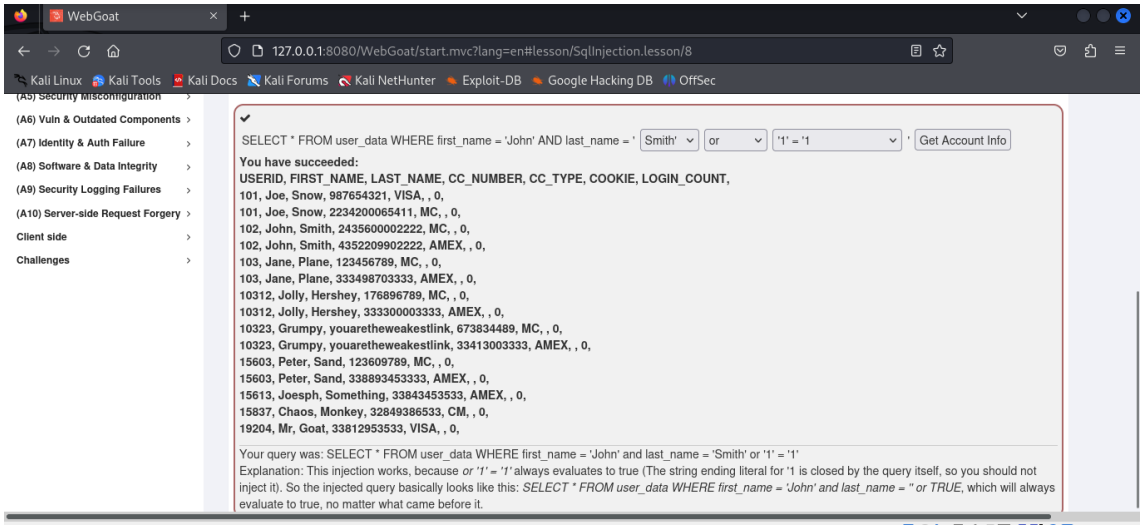
DCL es un subconjunto del lenguaje SQL que incluye comandos utilizados para controlar el acceso a los datos en una base de datos. Los comandos más comunes son:

GRANT: Otorga privilegios de acceso a los usuarios.

REVOKE: Revoca privilegios de acceso previamente otorgados.



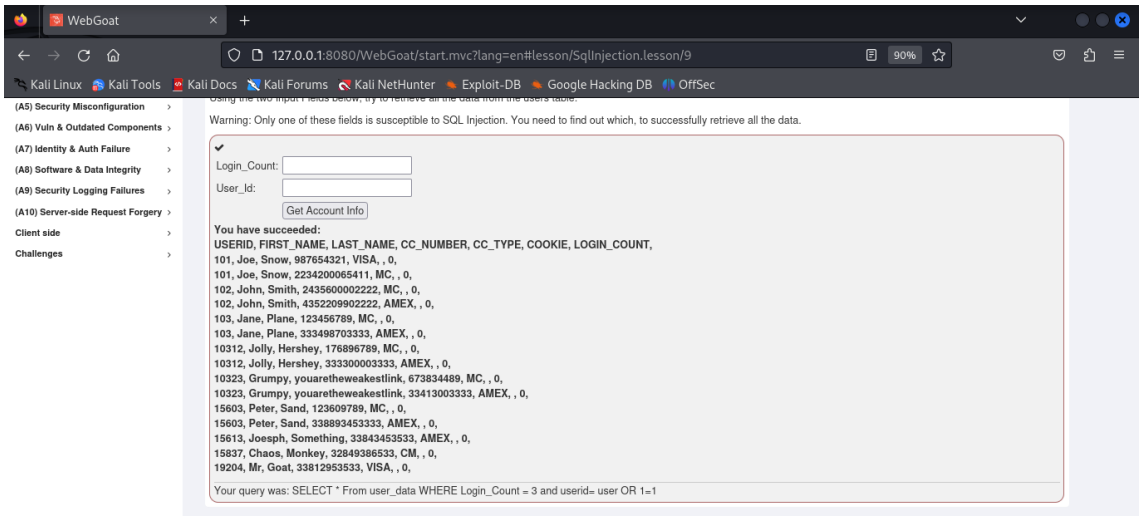
Con el comando GRANT ALL PRIVILEGES ON grant_rights TO unauthorized_user estamos dando permisos al usuario unathorized_user en la tabla grant_rights.



En el apartado 9 nos piden extraer toda la información de la tabla sabiendo únicamente el nombre y apellido de un usuario.

Para ello utilizamos el comando SELECT * FROM user_data WHERE first_name = ‘John’ AND last_name = Smith’ or ‘1’=’1

Este comando lo que hace es aprovechar la vulnerabilidad diciéndole a la base de datos que nos de la información de John Smith o de 1=1, siendo 1=1 siempre TRUE. Como 1=1 siempre es TRUE, toda la información que pertenezca John Smith o no pertenezca a este usuario se nos mostrará.



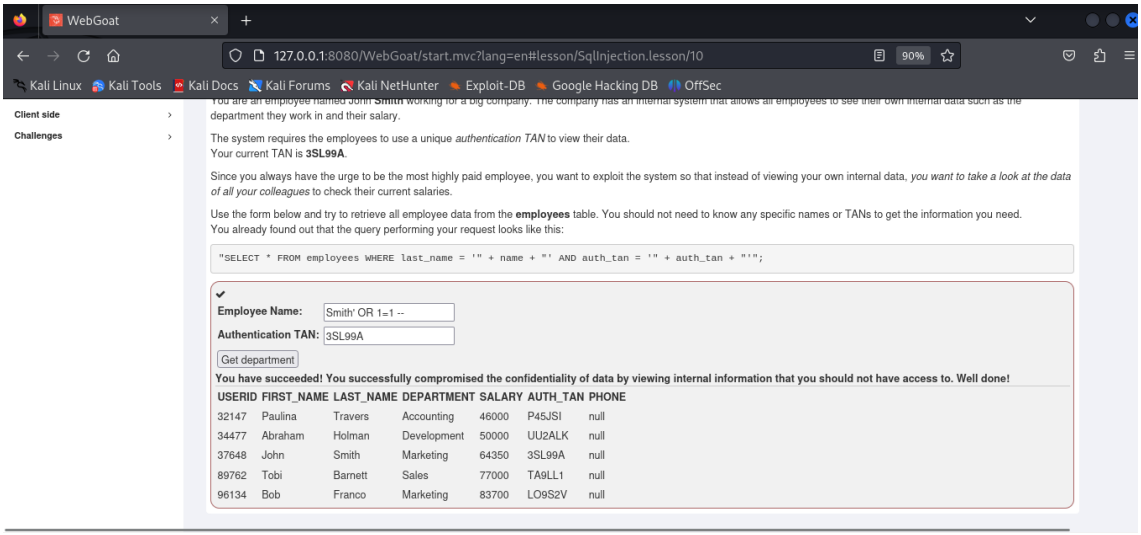
En el apartado 10 también nos pide que extraigamos toda la información de la tabla usuarios, pero a través de los campos Login_Count y User_Id.
Primero tenemos que averiguar cual de los dos campos es vulnerable a SQL Injection.

Vemos que en este caso el campo vulnerable es el de User_Id.

Así que una posible consulta para conseguir la información que nos piden es la siguiente:

SELECT* FROM user_data WHERE Login_Count = 3 AND User_Id = user OR 1=1

Les estamos diciendo a la base de datos que nos de toda la información del usuario user o 1=1 (TRUE), por lo tanto, siempre es TRUE y nos proporcionará toda la información de la tabla.



En el apartado 11 el sistema requiere a los empleados su código de autentificación para ver su información.

El problema nos requiere extraer la información de todos los empleados de la tabla employees sin conocer los nombres específicos ni los códigos de autentificación.

Los pasos seguidos para la resolución de este apartado son:

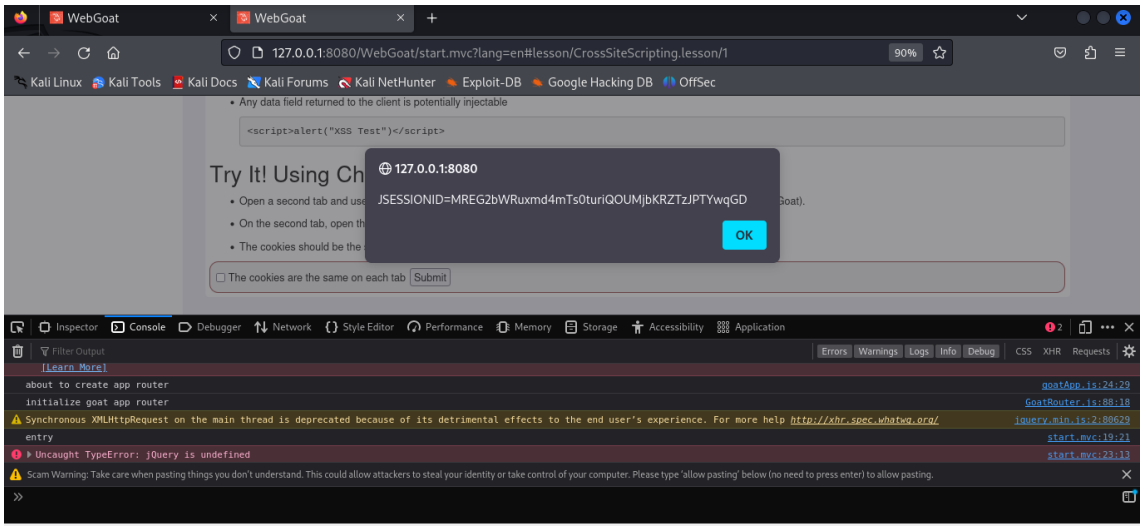
- Identificación de campos vulnerables mediante inyección manual de `' OR '1'='1'`.
  Hemos detectado que el campo Employee Name es vulnerable.
- Inyectamos en los campos la consulta:
  Employee Name: Smith' OR 1 = 1 -- / ' OR '1'='1'
  Authentication TAN: 3SL99A (que es nuestro código de autentificación)

Gracias a esta inyección SQL se logró extraer nombres de usuarios, contraseñas hash y otros datos confidenciales de la base de datos.

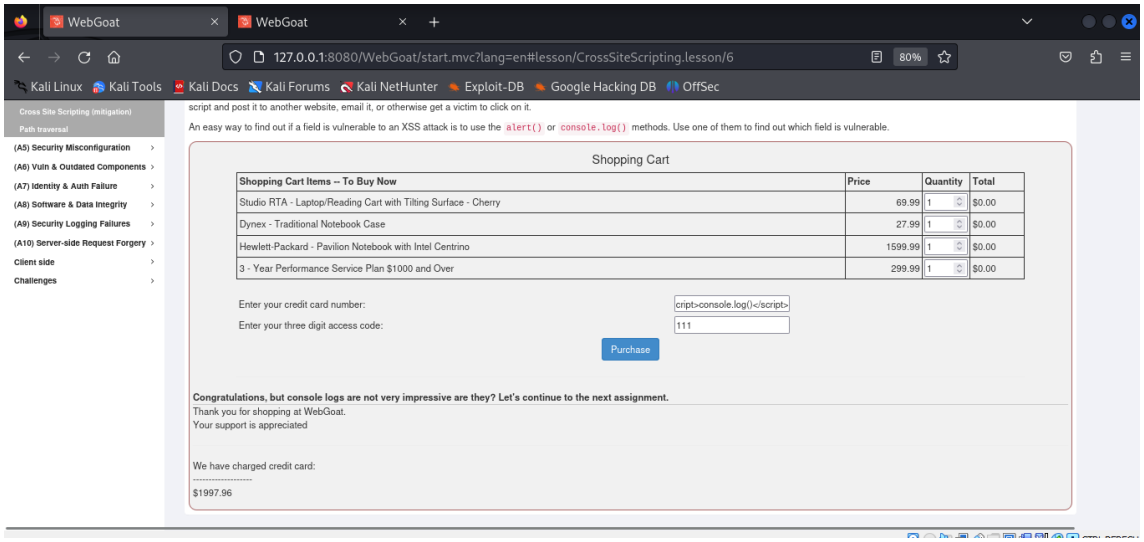
3.2.2. A3 Injection – Cross Site Scripting – Apartado 7

Antes de explicar el Apartado 7 explicaremos las vulnerabilidades previas que nos encontramos en A3 Injection – Cross Site Scripting.

El escenario del apartado 1 nos da la posibilidad de ingresar texto en un campo que luego se muestra a otros usuarios sin una limpieza adecuada, lo que nos permite ejecutar código JavaScript.



Al Inspeccionar la web vamos a la Consola y ejecutamos el código “alert(document.cookie)”. Al ejecutarlo nos aparece una alerta con el ID de sesión.



En el apartado 7 tenemos el escenario de un carrito de compra en el cual nos piden saltarnos el método de pago y poder comprar sin introducir sin el dato de la tarjeta de crédito.

Los pasos seguidos para la resolución de este apartado son:
- Inyección de scripts maliciosos en campos de entrada.
  Detectando así que el campo de tarjeta de crédito era vulnerable.
- Ejecución de alertas JavaScript para validar la vulnerabilidad.
  Ejecutamos el script en el campos la consulta:
  Enter credit card number: <script>console.log>/script>
  Enter three digit code: 111 (un código cualquiera)

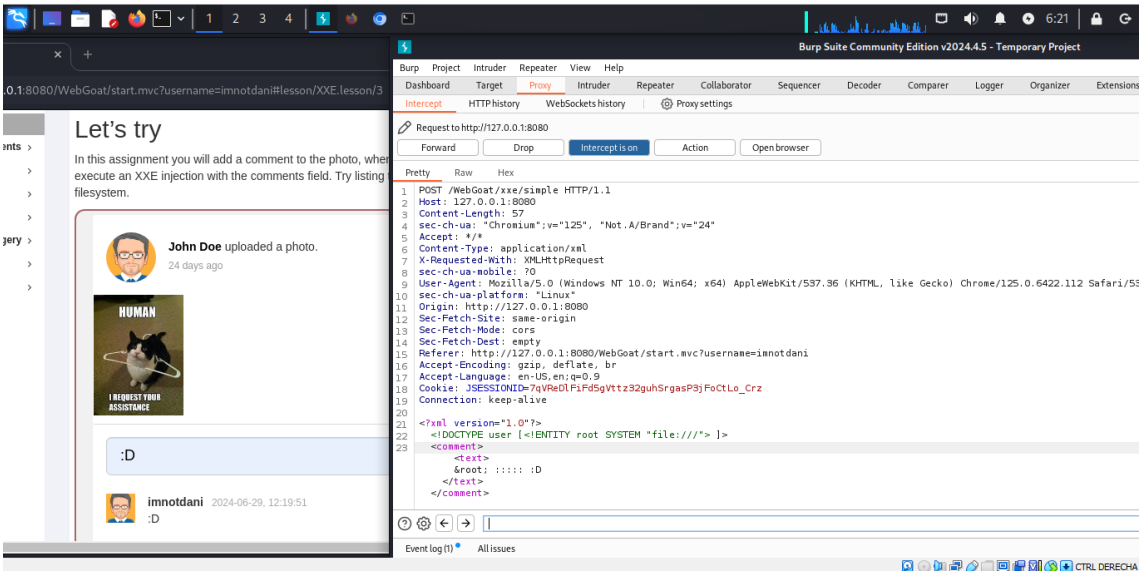
Se confirmó que el ataque XSS permitía ejecutar scripts en el navegador de la víctima.

3.2.3. A5 Security Misconfiguration – Apartado 4

La aplicación web presenta configuraciones que son incorrectas o demasiado permisivas, como archivos de configuración mal protegidos, permisos incorrectos en directorios, o información sensible expuesta en entornos de desarrollo o producción.

Para la resolución de este apartado utilizamos Burpsuit para interceptar las peticiones.

Al enviar un comentario en la publicación teniendo la interceptación proxy activa recibimos el código de petición generado en Burpsuit.



Una vez recibida la información editamos el código XML.

Insertamos el código: <!DOCTYPE user [<!ENTITY root SYSTEM "file:///">]>

Desglose del Código:

<!DOCTYPE user [...] >:

Define un tipo de documento user para XML.

El contenido dentro de los corchetes ([...]) es una definición interna del DTD.

<!ENTITY root SYSTEM "file:///">:

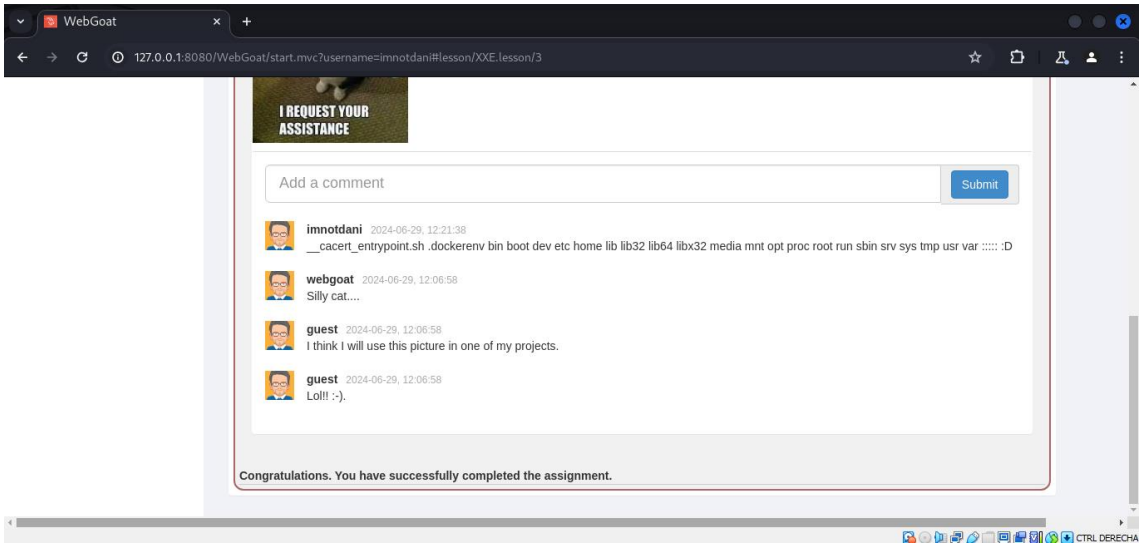
Define una entidad llamada root.

SYSTEM indica que esta entidad es una referencia a un recurso externo.

"file:///" especifica la ubicación del recurso externo, en este caso, la raíz del sistema de archivos.

Por último modificamos el comentario y añadimos “&root;”.

Esta es una referencia a una entidad llamada root. Las entidades en XML se utilizan para definir sustituciones de texto. Cuando un procesador XML encuentra &root; reemplaza esta referencia con el contenido definido para la entidad root.



En la imagen podemos ver como el comentario ha sido reemplazado por el contenido de root.

Se encontró que ciertos directorios y configuraciones de encabezados exponían información sensible del servidor.

3.2.4. A6 Vulnerabilidades y componentes obsoletos – Apartado 5

Se evaluaron componentes obsoletos utilizados por la aplicación.
- Explotación:
  - Herramientas Utilizadas: dependency-check para escanear las dependencias.
  - Proceso:
    - Análisis de bibliotecas y frameworks utilizados en la aplicación.
    - Verificación de versiones y búsqueda de vulnerabilidades conocidas.
  - Resultados: Se identificaron múltiples bibliotecas con vulnerabilidades críticas que debían actualizarse.

3.2.5. A7 Fallas en la identidad y autenticación – Contraseñas seguras – Apartado 4

Se evaluó la seguridad de las contraseñas en el sistema.
- Explotación:
  - Herramientas Utilizadas: Ataques de fuerza bruta utilizando Burp Suite.
  - Proceso:
    - Análisis de requisitos de contraseña y política de gestión.
    - Prueba de contraseñas débiles mediante ataques automatizados.
  - Resultados: Se encontraron contraseñas poco seguras que podían ser comprometidas fácilmente, lo que subraya la necesidad de mejorar las políticas de contraseñas.

3.3. Post – explotación

- Actividades Realizadas:
  - Examinación de los datos extraídos.
  - Evaluación de los permisos y potenciales escalaciones de privilegios.

3.4. Posibles mitigaciones

SQL Injection

Implementar consultas parametrizadas: Utiliza consultas parametrizadas o preparadas en lugar de concatenar directamente las entradas del usuario en las consultas SQL. Esto asegura que los datos del usuario se traten como datos y no como parte de la consulta. Las consultas parametrizadas pueden prevenir la ejecución de código SQL no deseado.

Uso de ORM (Object-Relational Mapping): Utiliza un ORM como Hibernate para manejar las interacciones con la base de datos. Los ORM generan automáticamente consultas SQL y proporcionan una capa de abstracción que reduce la posibilidad de errores que podrían llevar a una inyección SQL.

XSS

Validar y escapar entradas de usuario: Asegúrate de que todas las entradas del usuario sean validadas y escapadas correctamente antes de ser renderizadas en la salida HTML. La validación debe asegurarse de que los datos sean del tipo y formato esperados, y el escape debe transformar los caracteres especiales en entidades HTML.

Utilizar librerías de sanitización: Utiliza librerías de sanitización como OWASP Java Encoder para escapar automáticamente los datos antes de mostrarlos en la página.

Security Misconfiguration

Configurar adecuadamente los encabezados HTTP: Implementa encabezados HTTP de seguridad como Content Security Policy (CSP), Strict-Transport-Security (HSTS), X-Content-Type-Options, y X-Frame-Options para proteger contra una variedad de ataques.

Restringir el acceso a directorios sensibles: Asegúrate de que los directorios que contienen archivos de configuración, archivos de registro, y otros recursos sensibles no sean accesibles públicamente.

Componentes Obsoletos

Actualizar todas las bibliotecas a las versiones más recientes: Mantén todos los componentes de software y bibliotecas utilizados en tu aplicación actualizados a sus versiones más recientes y seguras. Esto incluye frameworks, librerías de terceros, servidores de bases de datos, etc.

Utilizar herramientas de gestión de dependencias: Herramientas como Maven, npm, o Bundler pueden ayudar a mantener las dependencias actualizadas automáticamente.

Monitorear vulnerabilidades: Utiliza servicios que monitorean y alertan sobre nuevas vulnerabilidades en los componentes que usas, como Dependabot, Snyk o la Base de Datos Nacional de Vulnerabilidades (NVD).

Autenticación

Implementar políticas de contraseñas fuertes: Exige a los usuarios crear contraseñas fuertes que incluyan una combinación de letras mayúsculas, minúsculas, números y caracteres especiales. Implementa controles para prevenir el uso de contraseñas comunes y realizar cambios regulares.

Autenticación Multifactor (MFA): Implementa MFA para agregar una capa adicional de seguridad. MFA requiere que los usuarios proporcionen dos o más factores de autenticación independientes para verificar su identidad.

3.5. Herramientas utilizadas

- Nmap: Para el escaneo de puertos.
- Burp Suite: Para análisis de tráfico y explotación de vulnerabilidades.
- Dependency-check: Para identificar componentes vulnerables y obsoletos.