Informe sobre WebGoat v8.1.0

INDICE:

Ámbito y alcance de la auditoría

Objetivos específicos involucrados en la auditoría:

-Entre los objetivos de esta auditoría sobre webgoat v8.1.0 podemos destacar:

- Identificar la máxima información posible sobre la aplicación:
    - Que puertos se encuentran abiertos
    - Sistema Operativo que se usa
    - Lenguajes de programación utilizados en la aplicación Web
- Detectar y explotar las diversas vulnerabilidades mediante diferentes tipos de ataques
- Posibles medidas de atenuación potenciales

Los sistemas y entornos involucrados en la auditoría:

Sistema operativo (Mediante un entorno virtualizado bajo VirtualBox 7.0.12):

Kernel: 6.5.0-kali3-amd64 arch: x86_64 bits: 64 Desktop: Xfce v: 4.18.1

Distro: Kali GNU/Linux 2023.3 kali-rolling

Herramientas utilizadas:

Burp suite community edition 2023.10.3.7

→(Trabajando en un entorno en local: 127.0.0.1:8080)

Informe Ejecutivo

Breve resumen del proceso realizado

-La auditoría se realizará como ya he comentado en kali linux a través del entorno de virtualización VirtualBox con el objetivo de explotar las diferentes vulnerabilidades que nos brinda WebGoat.

-La instalación de WebGoat se ha realizado a través de Docker dada su comodidad.

1. (Sql inyection) Numeric SQL Injection
    a. Inyectamos en una consulta un número y en el ID un estado de verdadero o falso
2. (Sql inyection) Compromising confidentiality with sql
    a. Inyectamos a través de nuestro login una consulta para acceder a todos los salarios de la tabla
3. (Sql inyection) Cross Site Scripting
    a. Comprobamos si los campos en los que introducimos nuestra tarjeta están bien definidos en el código de la página para evitar inyecciones de otros caracteres que no sean números.
4. Security Misconfiguration XXE
    a. A través de burp probamos a introducir una entidad externa(XML) para sacar todo el árbol de directorios de root
5. Modern REST framework XXE
    a.
6. Vuln & outdated Components (Vulnerable Components)
    a.
7. Identity & Auth Failure - Secure Passwords
    a. Para asegurarnos de que las contraseñas que utilizamos son o no seguras lo mejor es obviar contraseñas comunes con el nombre del usuario o combinaciones débiles y predecibles.

Vulnerabilidades destacadas

Inyecciones SQL → Las consultas que hace la página a través del código al conectarse con la base de datos deberían de estar protegidas y no solo contestar a un 1 o un 0 o a un verdadero o falso (booleanos) ya que a través de una inyección sql ya sea hecha a mano o mediante una herramienta podemos intentar “colarnos” en la base de datos y hacer un select o cualquier otro tipo de consulta para que nos devuelva un dato en concreto o bien manipulemos la base de datos.

Ataques XXE→A través de las aplicaciones Web podemos procesar datos XML externos los cuales provoquen errores para darnos información como por ejemplo la ruta o archivos dentro del directorio.

Contraseñas inseguras→ Una contraseña predecible ,corta o con un solo tipo de carácter aumenta el riesgo que mediante un ataque por fuerza bruta pueda uno acceder al perfil o la cuenta protegida por contraseña.

Conclusiones

-A pesar de que hablamos de un entorno de desarrollo en el que las vulnerabilidades se encuentran para aprender podemos sacar la conclusión de que a la hora de crear un código para una aplicación web siempre hay que revisar todos los inputs he intentar imaginar cómo se podría explotar dicha entrada de diferentes formas, aunque no solo el que ofrece el servicio debe de tener cuidado ya que los usuarios de una plataforma también debe saber que contraseñas introducen y a donde se redirigen realmente cada vez que se hace “click” en un botón que ejecute código como por ejemplo javascript.

Recomendaciones

-En cuanto a inyecciones sql mis recomendaciones es proteger toda aquella conexión que tenga la página con la base de datos limitando y minimizando las conexiones para reducir así las inyecciones y por consiguiente los ataques.

-Haciendo referencia a los ataques XXE la mejor manera para mitigarlos es deshabilitar la expansión de entidades externas o validar las entradas XML y así mostrar errores antes de que puedan afectar a nuestro código.

-Finalmente hablando de la creación de contraseñas y los ataques por fuerza bruta la idea es que las contraseñas fuertes deben ser únicas, no contener información fácilmente accesible o predecible, y ser lo suficientemente largas y complejas para resistir intentos de fuerza bruta o adivinación.

Descripción del proceso de Auditoria

Reconocimiento/Information Gathering

1.(Sql inyection) Numeric SQL Injection

Analizamos la consulta que realiza al introducir el login_count y el User_id para comprobar si inyectando una combrobacion 1=1 o 0 provoca error o nos devuelve información adicional.

2.(Sql inyection) Compromising confidentiality with sql

Analizamos de nuevo la consulta para mostrar datos de nuestro usuario y el salario que en función de cómo se consulten dichos datos en la base de datos nos pueda devolver más datos o rompamos la conexión.

3.(Sql inyection) Cross Site Scripting

Comprobando desde el editor web donde podemos ver información de la estructura de la página podemos observar que tipo de dato han determinado al input y explotar dicha vulnerabilidad introduciendo otro tipo de dato que pueda romper o provocar errores en la página.

4.Security Misconfiguration XXE

Podemos comprobar y analizar la manera en que se comprueba el código ya que introduciendo de algún modo a través de alguna herramienta un fichero o código XML podamos generar una salida distinta de la información.

5.Modern REST framework XXE

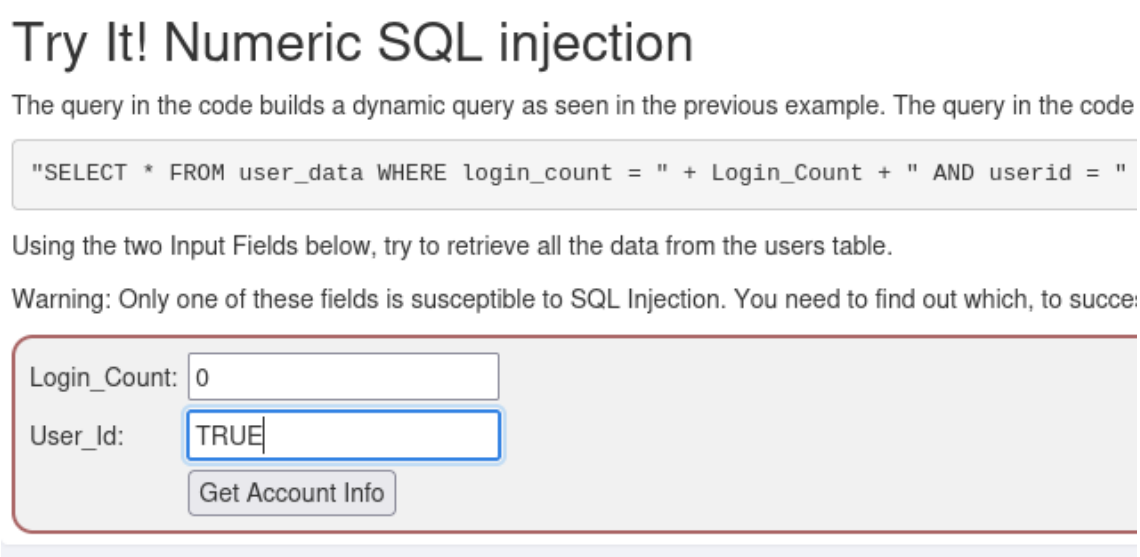
6.Vuln & outdated Components (Vulnerable Components)

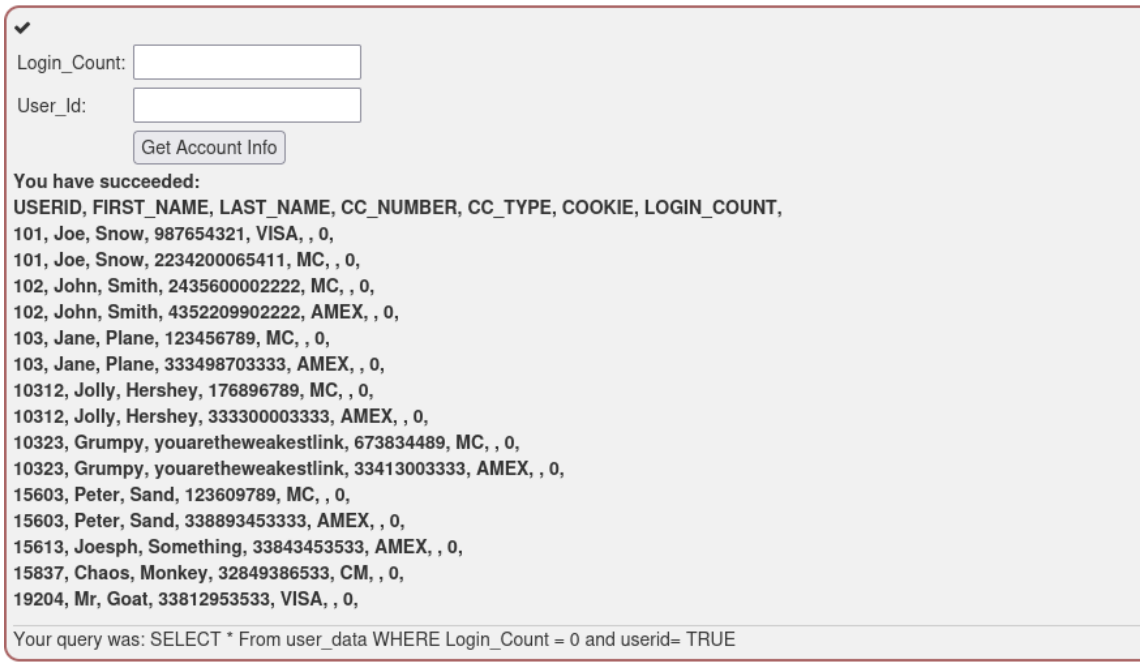
7.Identity & Auth Failure - Secure Passwords

Observamos que la contraseña que nos ha dejado crear a nuestro usuario (123root) es una contraseña que con ataque con fuerza bruta sería fácil de descubrir o romper.

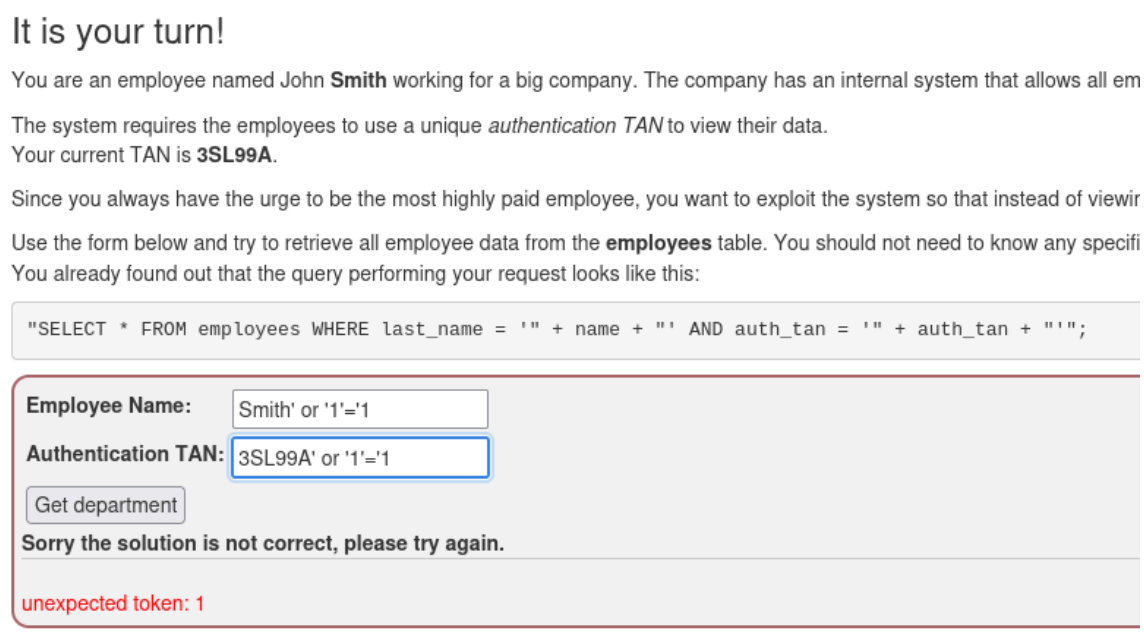
Explotación de Vulnerabilidades Detectadas

1.(Sql inyection) Numeric SQL Injection

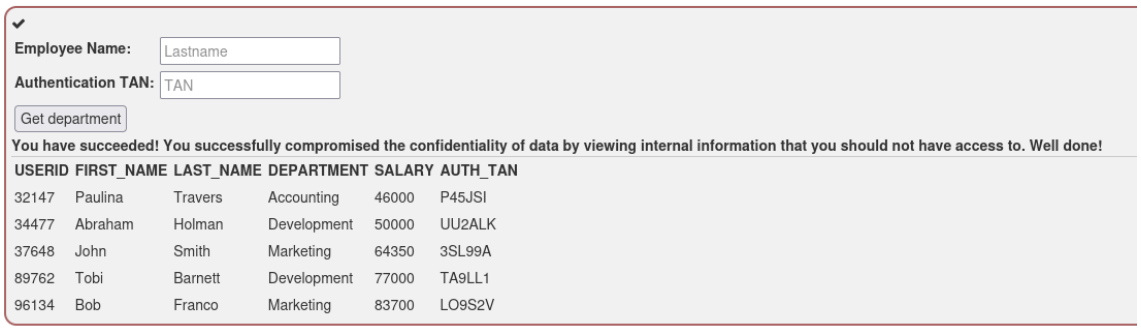




2.(Sql inyection) Compromising confidentiality with sql



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



✔

Employee Name: Lastname

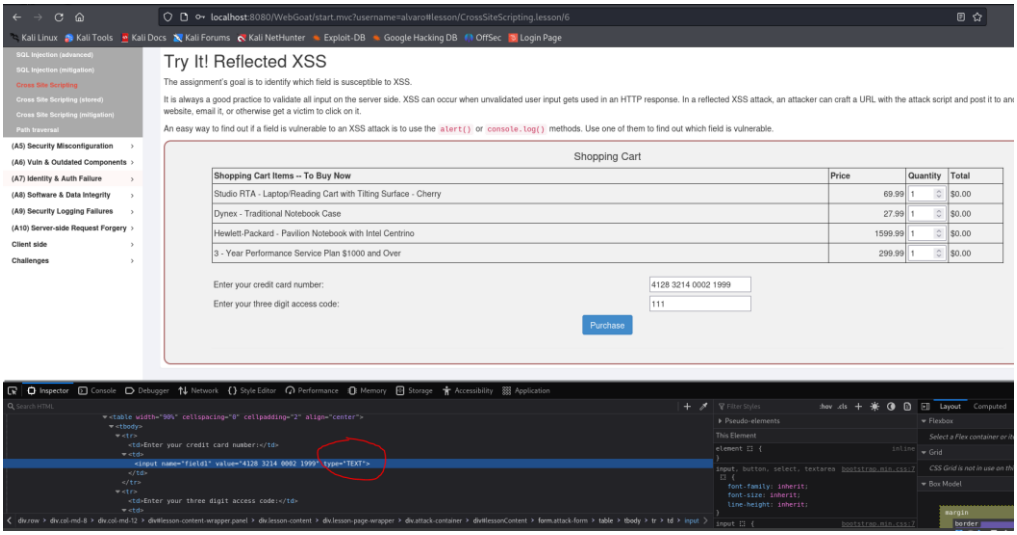
Authentication TAN: TAN

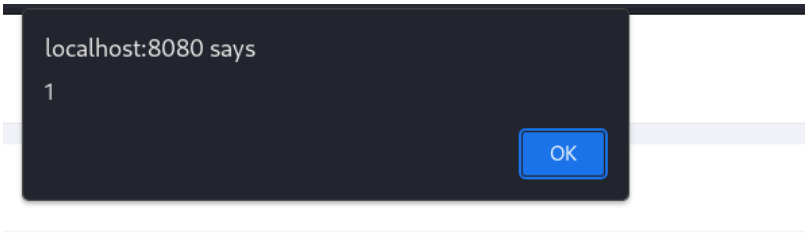
Get department

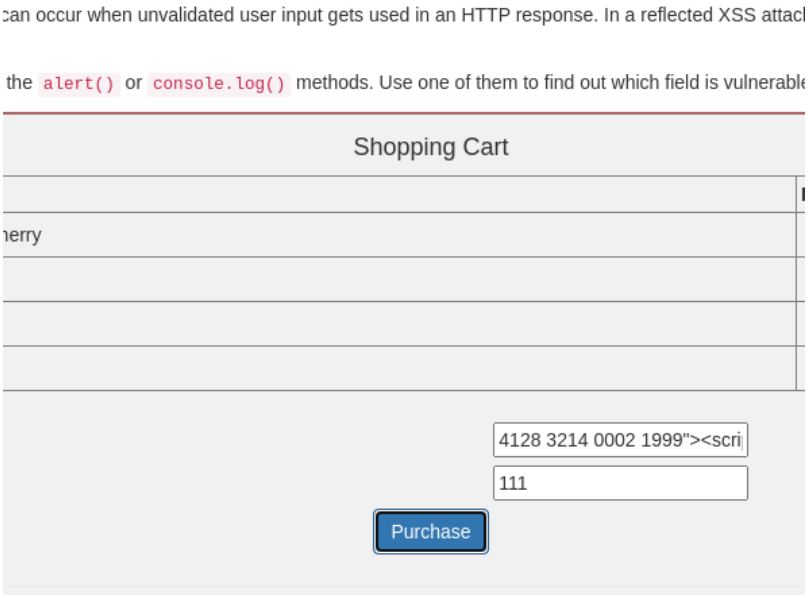
You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

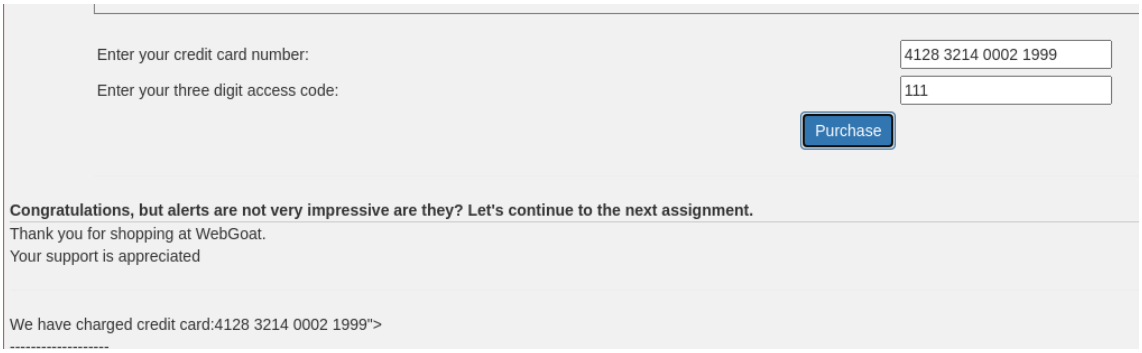
USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
32147	Paulina	Travers	Accounting	46000	P45JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

3.(Sql inyection) Cross Site Scripting





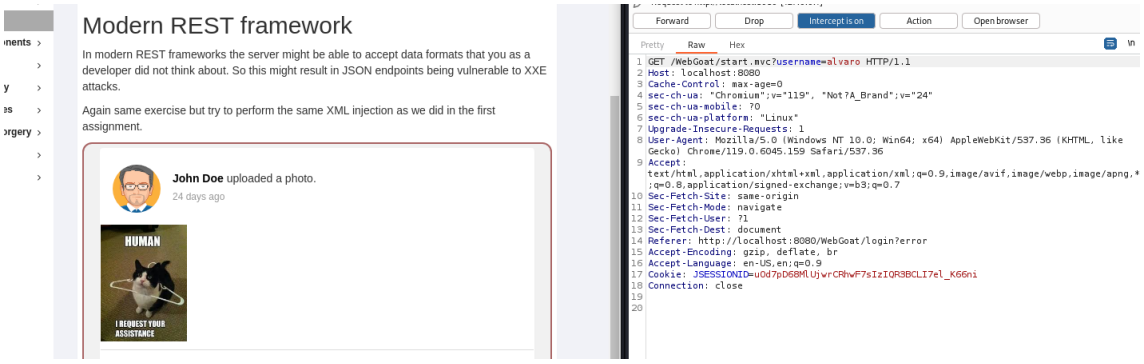


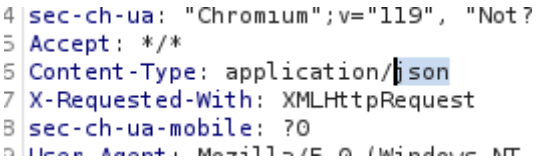


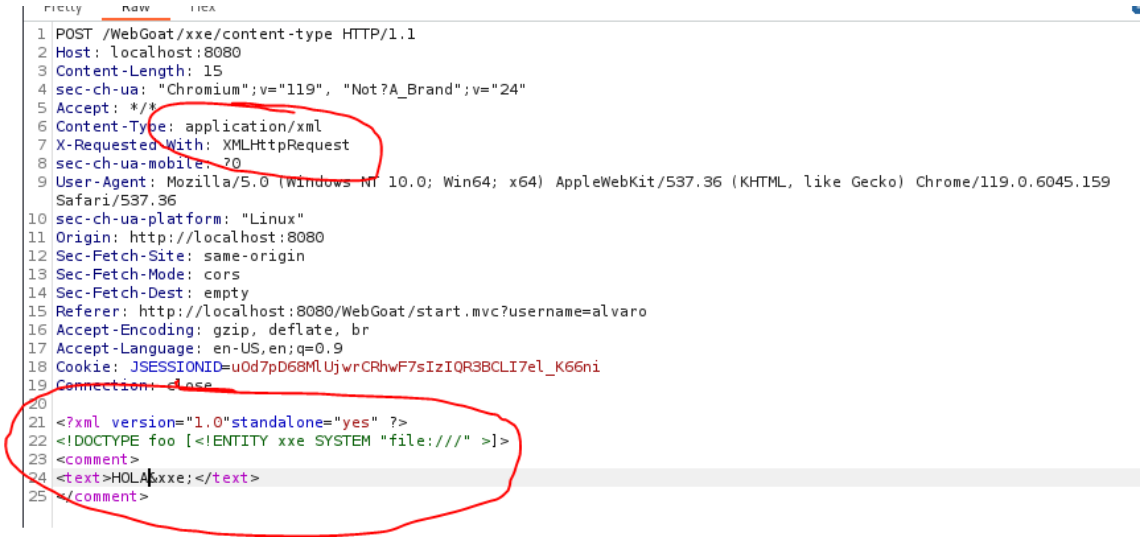
4128 3214 0002 1999"><script>alert(EJEMPLO DE INYECCION)</script>

4128 3214 0002 1999"><script>alert(1)</script>

4.Security Misconfiguration XXE



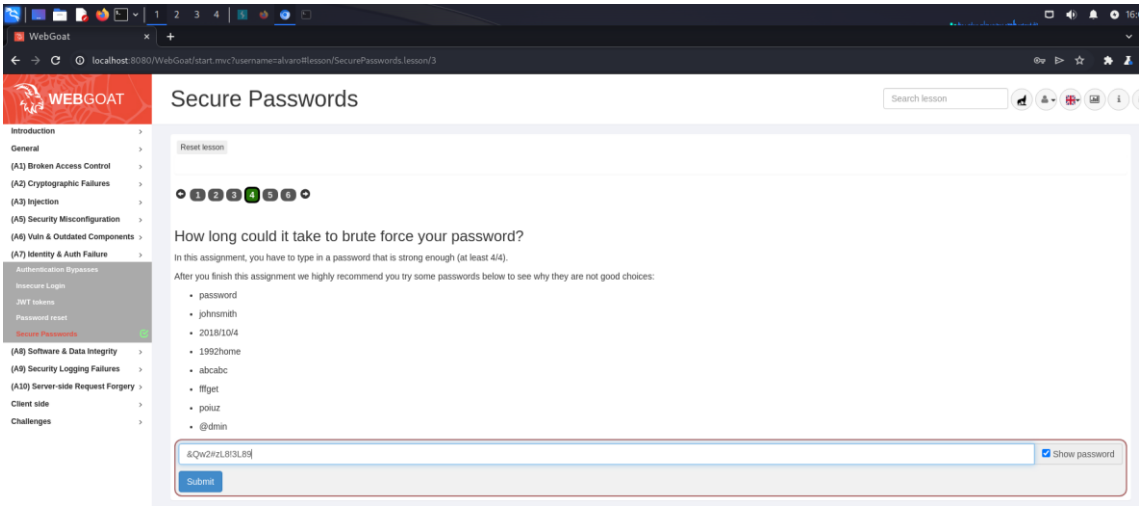


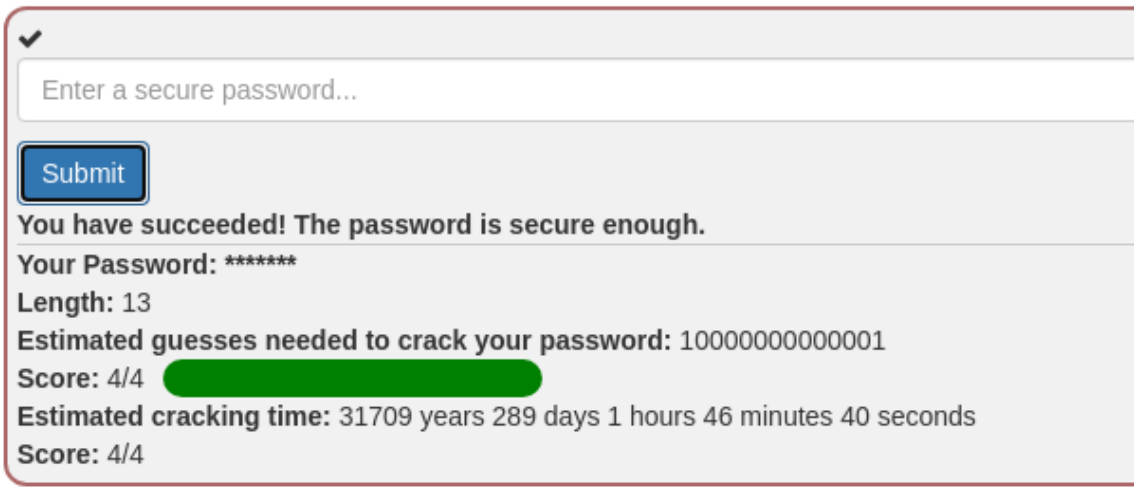


5.Modern REST framework XXE

6.Vuln & outdated Components (Vulnerable Components)

7.Identity & Auth Failure - Secure Passwords





Post-explotación

1.(Sql inyection) Numeric SQL Injection

Inyectando 0 y TRUE en los inputs logramos tener acceso no solo a nuestro usuario sino a todos los usuarios de la tabla junto a sus datos.

2.(Sql inyection) Compromising confidentiality with sql

Inyectando nuestros datos (usuario y número TAN) o la comprobación que hace la sonulta de si se encuentra o no el dato “1=1” conseguimos mostrar en este caso el salario de todos nuestros compañeros almacenado en la base de datos.

3.(Sql inyection) Cross Site Scripting

Inyectando código javascript aprovechando un campo de tipo texto que no debería de ser tipo texto para hacer aparecer una alerta.

4.Security Misconfiguration XXE

Mediante la herramienta BURP en kali Linux aprovechamos para espiar el puerto 8080 he introducir un fragmento de texto xml y así poder introducir una entidad externa mostrándonos datos relevantes de la página que no deberíamos de poder ver.

5.Modern REST framework XXE

6.Vuln & outdated Components (Vulnerable Components)

7.Identity & Auth Failure - Secure Passwords

Mediante una de las herramientas que proporciona kali como medusa o hydra podríamos atacar con diccionarios conseguidos en la Deep web o proporcionados en cualquier sitio de la web con nombres y combinaciones comunes así demostrando lo débiles que son las contraseñas con pocos caracteres, comunes,débiles por sus caracteres básicos o combinaciones de nombre como root o 1234.

Posibles Mitigaciones

1.(Sql inyection) Numeric SQL Injection

Proteger las comprobaciones de las consultas haciéndolas más complejas y no comprobando con valores booleanos sería una posible solución.

2.(Sql inyection) Compromising confidentiality with sql

Inyectando valores como 1 o 0 similares al de TRUE o FALSE se podrían solucionar de la misma manera como en el caso anterior. Una mejora de código respecto a la conexión de la base de datos con la página web solucionaría en gran parte esta brecha de seguridad.

3.(Sql inyection) Cross Site Scripting

Una mejora de código haciendo una revisión del mismo para que en un campo que es numérico no se pueda ingresar otro tipo de caracteres (aunque este erro es un muy grande existen corporaciones que al tener un código muy extenso en ocasiones el error humano en la programación se tiene que tener en cuenta).

4.Security Misconfiguration XXE

Una protección en el código frente a inyecciones de entidades externas permitiría no introducir código o alternándolo permitiendo introducir imágenes o alertas falsas.

5.Modern REST framework XXE

6.Vuln & outdated Components (Vulnerable Components)

7.Identity & Auth Failure - Secure Passwords

Una concienciación sobre la utilización de contraseñas más complejas a los usuarios y una obligación por parte de las plataformas a la hora de crear un usuario y exigir unos requerimientos mínimos de seguridad para contratacar a los ataques de fuerza bruta.

Herramientas Utilizadas

- BURP(KALI)
- WEBGOAT
- KALI LINUX
- VIRTUALBOX
- HYDRA(KALI)
- FIREFOX
- CHRONIUM
- DOCKER
- Bash: