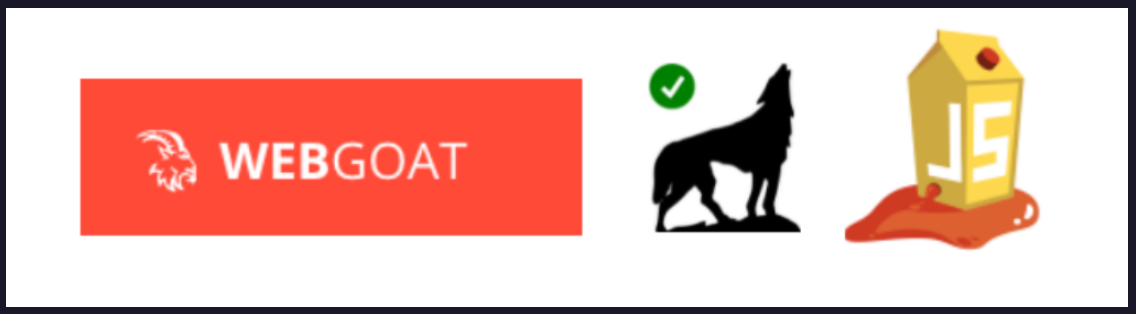
INTRODUCCIÓN A LA CIBERSEGURIDAD

Proyecto auditoría Web básica de WEBGOAT



Alexandru Comanescu

7 Diciembre 2023

Versión 1.0

Tabla de contenidos

Declaración de confidencialidad

Este informe de auditoría web sobre la aplicación de práctica WebGoat ha sido creado exclusivamente para el uso interno y educativo del destinatario designado, así como para los profesores encargados de realizar la corrección correspondiente. La información detallada en este informe está protegida por derechos de autor y es estrictamente confidencial. Cualquier reproducción, distribución o divulgación no autorizada de este informe, total o parcialmente, a cualquier individuo o entidad está prohibida sin el consentimiento expreso y por escrito del autor.

El contenido de este informe puede incluir información delicada relacionada con vulnerabilidades identificadas, recomendaciones de seguridad y detalles técnicos. El destinatario asume la responsabilidad de implementar todas las medidas necesarias para garantizar la seguridad y confidencialidad de esta información, y se compromete a no divulgar a terceros sin el previo consentimiento por escrito del autor.

Este informe se proporciona con el único propósito de la evaluación educativa y la mejora continua de la seguridad en WebGoat. No se ofrece ninguna garantía sobre la exactitud o integridad de la información presentada en este informe. El autor no asume ninguna responsabilidad por el uso indebido o interpretación incorrecta de la información contenida en este documento.

Firma: Alexandru Comanescu
Fecha: 7 de Diciembre de 2023

Ámbito y alcance de la auditoría

Este informe tiene como propósito poner en práctica las metodologías y herramientas vistas en clase de Introducción a la Ciberseguridad impartida por mi profesor Carlos Cilleruelo Rodríguez realizando una auditoría que se llevará a cabo sobre la aplicación educativa WebGoat. WebGoat es una aplicación diseñada con fines pedagógicos para enseñar conceptos de seguridad en aplicaciones web, será sometida a una evaluación exhaustiva con el objetivo de identificar posibles vulnerabilidades, evaluar su rendimiento y verificar su cumplimiento con buenas prácticas de desarrollo.

La auditoría se centrará en las secciones A3, A5, A6 y A7 de WebGoat, dando prioridad a áreas que podrían ser propensas a vulnerabilidades de seguridad. Esto incluye un análisis detallado de la descripción de WebGoat y la comprensión de sus componentes clave.

Los objetivos específicos de la auditoría se enfocarán en la evaluación de seguridad, la evaluación del rendimiento y la escalabilidad, y la verificación del cumplimiento de buenas prácticas de desarrollo. El público objetivo de esta auditoría son los desarrolladores y educadores que utilizan WebGoat con fines pedagógicos, buscando mejorar la aplicación para un aprendizaje más seguro y efectivo.

Para llevar a cabo la auditoría, se emplea una metodología estándar que abarque pruebas de seguridad, evaluación de rendimiento y análisis de código. Los criterios de evaluación se basarán en estándares reconocidos de seguridad web del ranking top 10 de OWASP, buenas prácticas de desarrollo y métricas de rendimiento establecidas.

Informe ejecutivo

El departamento de Ciberseguridad de Keepcoding nos ha encomendado la tarea de llevar a cabo la práctica de auditoría de la aplicación web WebGoat. Nuestro objetivo consiste en identificar y abordar las vulnerabilidades más recurrentes que suelen manifestarse en entornos prácticos de programación y fallos de seguridad.

Resumen del proceso realizado

Durante el proceso de auditoría en WebGoat, se llevó a cabo una evaluación exhaustiva de las secciones A3 (Injection), A5 (Security Misconfiguration), A6 (Vuln & outdated Components) y A7 (Identity & Auth Failure). Se aplicaron técnicas de reconocimiento, explotación de vulnerabilidades y análisis post-explotación para identificar posibles debilidades en la seguridad y rendimiento de la aplicación. Se utilizó una metodología sólida, incluyendo pruebas de seguridad y análisis de código, para garantizar la exhaustividad de la evaluación.

Vulnerabilidades destacadas

La auditoría reveló varias vulnerabilidades destacadas en WebGoat. Entre ellas, se identificaron posibles puntos de acceso no seguros, vulnerabilidades de inyección de código, y situaciones de manejo inadecuado de sesiones. Estos hallazgos subrayan la importancia de abordar aspectos críticos de seguridad para garantizar la integridad y confidencialidad de los datos.

En la sección A3 (Injection) se han detectado las siguientes vulnerabilidades que en este apartado serán listadas:

Sección	Gravedad	Vulnerabilidad	Técnicas
A3 Injection (Intro)	Alta	SQL Injection	• String SQL Injection • Number SQl Injection • CIA Triad violation
A3 Injection (Advanced)	Crítica	Combined SQL Injection	• Union SQL Injection • Blind SQL Injection
A3 Injection (XSS)	Media	Cross Site Scripting (XSS)	• Reflected XSS • DOM-Based XSS • Stored XSS

En la sección A5 (Security Misconfiguration) se han detectado las siguientes vulnerabilidades que en este apartado serán listadas:

Sección	Gravedad	Vulnerabilidad	Técnicas
A5 Security Misconfiguration	Alta	XXE Injection	• XXE Injection • XML Injection • Blind XXE Injection

En la sección A6 (Vuln & outdated Components) se han analizado posibles vulnerabilidades que en este apartado serán listadas:

Sección	Gravedad	Vulnerabilidad	Técnicas
A6 Vulnerable Components	Media	OSS Risks	• Analysis of OSS • Analysis of versions

En la sección A7 (Identity & Auth Failure) se ha realizado la correcta auditoría de las contraseñas de seguridad listadas y se han recomendado contraseñas más seguras y métodos para crearlas.

Recomendaciones clave

1. SQL Injection:

- ➢ *Utilizar consultas parametrizadas en lugar de concatenar cadenas o procedimientos almacenados para prevenir la inyección de SQL.*
- ➢ *Usar funciones de escapado específicas del lenguaje.*
- ➢ *Validar y filtrar las entradas de usuario para evitar caracteres maliciosos.*
- ➢ *Implementar controles de acceso y cifrado de datos.*

2. Combined SQL Injection:

- ➢ *Implementar consultas parametrizadas para prevenir inyecciones.*
- ➢ *Validar y filtrar entradas para evitar uniones maliciosas.*
- ➢ *Realizar validación y filtrado estricto de entradas del usuario.*
- ➢ *Monitorear y auditar consultas SQL en tiempo real.*

3. Cross Site Scripting (XSS):

- ➢ *Utilizar codificación adecuada al mostrar datos en la interfaz.*
- ➢ *Validar y filtrar entradas para prevenir XSS reflejado.*
- ➢ *Aplicar controles de salida seguros para evitar ataques basados en DOM.*
- ➢ *Validar y filtrar entradas de usuario.*
- ➢ *Implementar validación y codificación de salida para prevenir almacenamiento de scripts maliciosos.*

4. XXE Injection:

- ➢ *Configurar el procesamiento de XML de forma segura.*
- ➢ *Deshabilitar resolución de entidades externas innecesarias.*

5. OSS Risks:

- ➢ *Implementar un sistema de gestión de dependencias.*
- ➢ *Monitorear fuentes de información sobre vulnerabilidades de OSS*

6. Secure passwords:

- ➢ *Establecer políticas de contraseñas robustas.*
- ➢ *Fomentar contraseñas únicas y complejas.*
- ➢ *Bloqueo por intentos fallidos.*
- ➢ *Educar sobre contraseñas seguras.*
- ➢ *Fomentar el uso de frases de contraseña y gestores de contraseña*

Conclusiones

En conclusión, la auditoría en WebGoat ha proporcionado valiosa información sobre posibles vulnerabilidades y señalando áreas de mejora que actúan como escalones hacia un entorno educativo más blindado. Las recomendaciones presentadas buscan fortalecer la seguridad del entorno así como recalcar los principios de la tríada CIA (Confidentiality, Integrity and Availability)

La tabla de descubrimientos en seguridad revela un paisaje crítico y elevado, donde las inyecciones SQL y XSS emergen como focos de preocupación. Las recomendaciones aquí presentadas sirven como guía estratégica para robustecer las prácticas de seguridad, con un enfoque especial en el fortalecimiento de contraseñas y la gestión eficiente de componentes vulnerables.

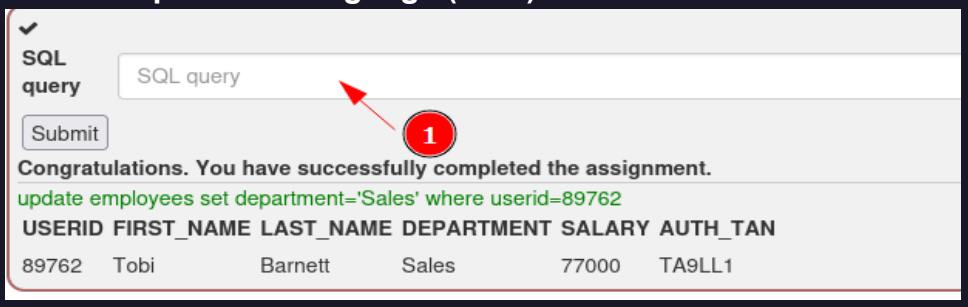
Además, se resalta la importancia de una configuración segura y la implementación de monitoreo continuo. Se propone un enfoque educativo para los usuarios, respaldado por una estrecha colaboración entre desarrolladores y equipos de seguridad, tejiendo así un escudo impenetrable frente a las vulnerabilidades identificadas.

Metodología de la auditoría web

Reconocimiento

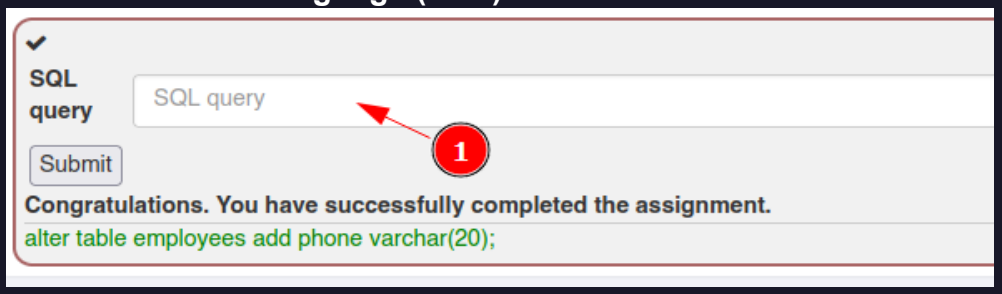
La primera sección en la que la auditoría empezó fue la (A3) injection, en la que explica claramente y en detalle qué es el lenguaje SQL y que vulnerabilidades podría acarrear dando lugar a filtrados de información sensible a la base de datos de WebGoat. En esta sección nos detalla cómo se pueden manipular datos a través de vulnerabilidades de programación, a continuación se muestran algunas imágenes con detalles sobre los pasos realizados durante el proceso.

Data Manipulation Language (DML)



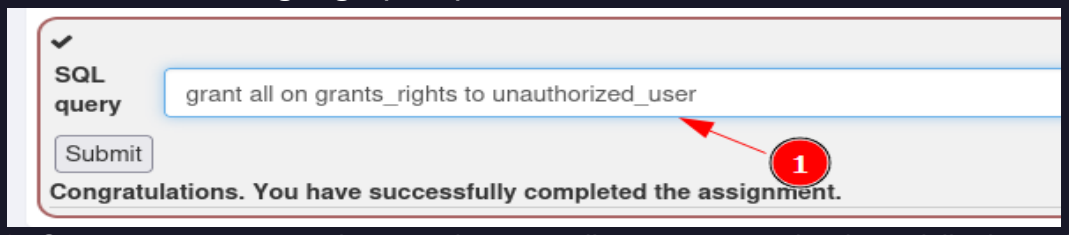
1. Se utiliza esta query para cambiar el valor de la columna 'department' por 'Sales' que tiene el userid '89762' que en este caso corresponde a Tobi Barnett.

Data Definition Language (DDL)



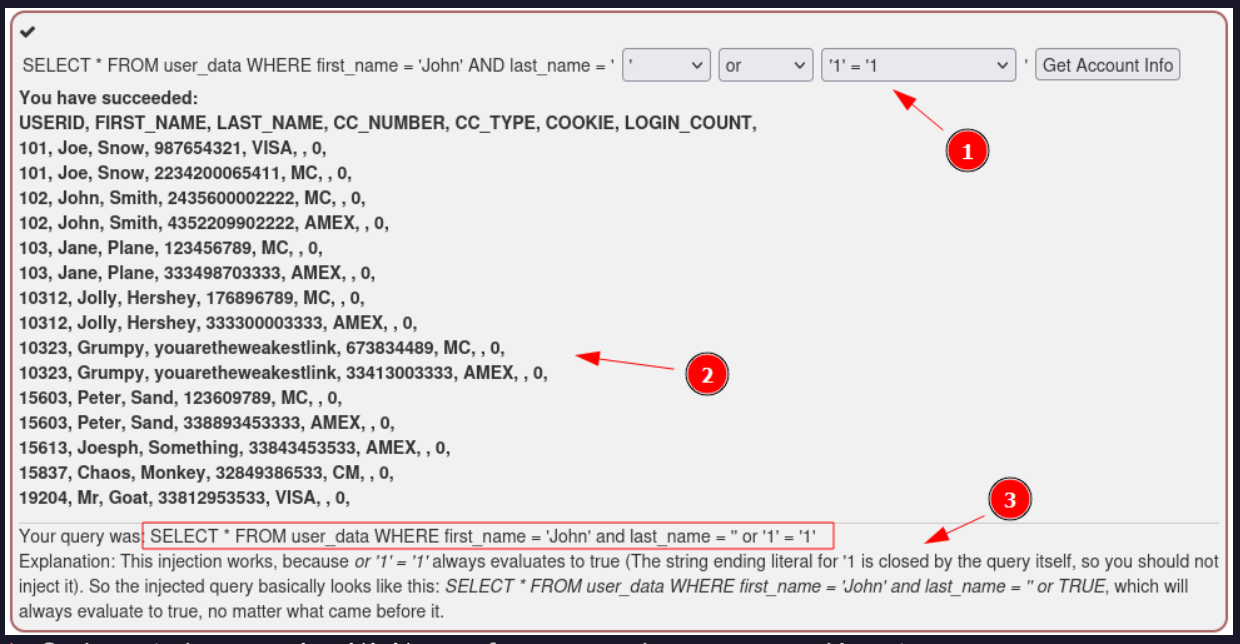
1. Se utiliza la query alter table para acceder a la tabla 'employees' y añadir con el comando add una columna con caracteres variables de longitud 20 con el nombre 'phone'.

Data Control Language (DCL)



1. Con esta query, con el comando grant all otorgamos todos los privilegios a la tabla 'grant_rights' a un usuario no autorizado.

Después de esta fase introductoria, seguimos encontrando unas vulnerabilidades más críticas, en las que destaca la SQL Injection, que consiste en la inserción de código malicioso a través de la entrada de una consulta SQL desde el cliente hacia la aplicación, estas inyecciones afectan directamente a la integridad y seguridad de los datos. Las vulnerabilidades más destacadas de esta sección son la ***inyección numérica de SQL***, ***la inyección SQL mediante cadenas***, compromisos de la Integridad y de la Disponibilidad de los datos de la base de datos. Por ejemplo:



1. Se inyecta la query *' or '1'='1* para forzar que siempre se evalúe a true
2. Obtenemos toda la información que hay en la tabla, lo que es una violación a la Triada CIA
3. La query completa que se ha utilizado es:
   - *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or '1' = '1'*

También se encontraron vulnerabilidades de tipo XSS (Cross-Site Scripting) que consisten en inyectar scripts maliciosos a través de etiquetas HTML. Los scripts inyectados pueden ser ejecutados en el navegador de la víctima, lo que podría llevar a diferentes tipos de ataques, como robo de sesiones, redirección a sitios maliciosos o modificación del contenido de la página. Se detectaron tanto ***Reflected XSS***, que se entrega y ejecuta de inmediato a través de la interacción directa con el usuario, como ***Stored XSS***, que se almacena en el servidor y se entrega a los usuarios cuando acceden a la página que muestra el contenido almacenado.

La segunda sección en la que la auditoría tuvo lugar, fue (A5) Security Misconfiguration, en la que explica claramente cómo se pueden abusar de los archivos XML dónde se encontró ***XXE Injection***, que es el procesamiento de datos XML sin validarlos adecuadamente y puede conducir a la divulgación de información sensible o ataque DoS. También se detectó ***XML Injection***, que es cuando la aplicación web no valida ni filtra correctamente los datos XML proporcionados por un usuario y puede llevar a la ejecución de scripts no autorizados, la revelación de información sensible o manipulación de la lógica de la aplicación.

Progresando en el escaneo de vulnerabilidades de la aplicación WebGoat, dentro de la sección (A6) Vuln & Outdated Components se han listado una serie de problemas en las que puede dar lugar a vulnerabilidades críticas que ponen en riesgo la información sensible de la base de datos, la integridad, la seguridad y la disponibilidad de esta. Los problemas principales son el mantenimiento y actualización de los frameworks que usa la aplicación y la verificación y gestión del código importado de repositorios OSS (Open Source Software).

Finalizando la auditoría de la web se decide monitorizar la vulnerabilidad de las contraseñas que están listadas en la sección (A7) Identity & Auth Failure, realizando un escaneo de la seguridad y la integridad de dichas contraseñas, aportando consejos y mejoras para concienciar de la importancia de tener unas contraseñas fuertes y seguras para complicar los ataques por fuerza bruta y la rotura de la encriptación de estas.

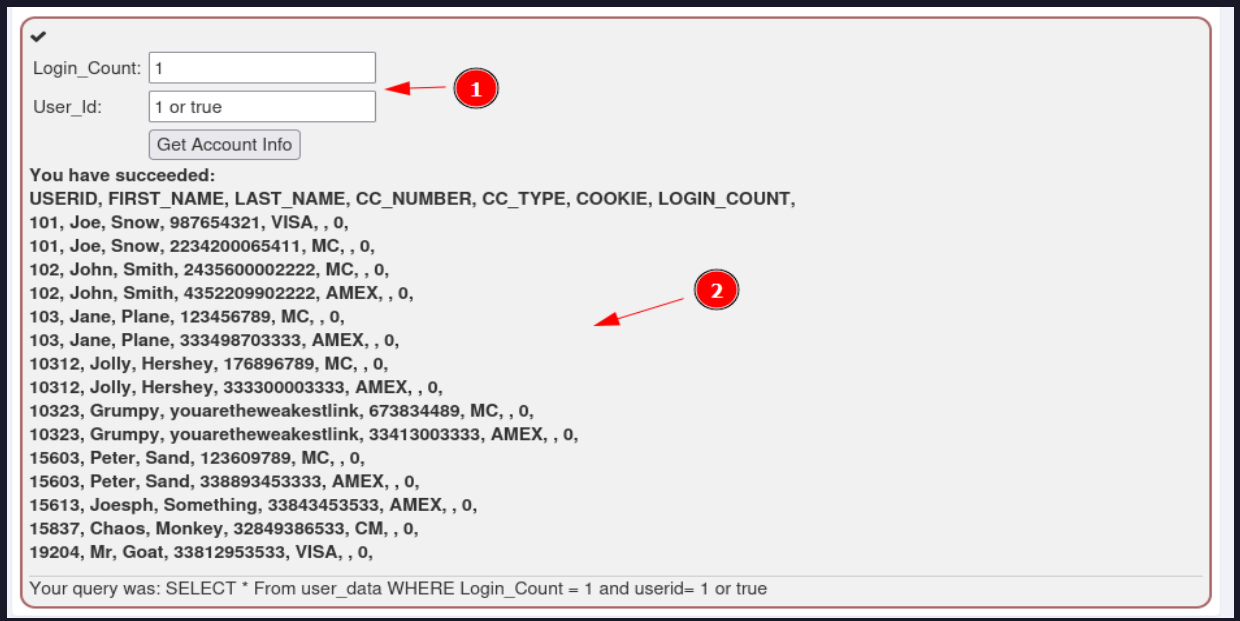
Explotación de Vulnerabilidades Destacadas

Number SQL Injection:

Descripción:
Se identificó una vulnerabilidad de inyección SQL en el campo *Login_Count* de la tabla *user_data*. Al realizar pruebas de inyección SQL con los valores *user: 1* y *user_id: 1 or True*, se observó un comportamiento inseguro que permite recuperar información de la base de datos sin restricciones.

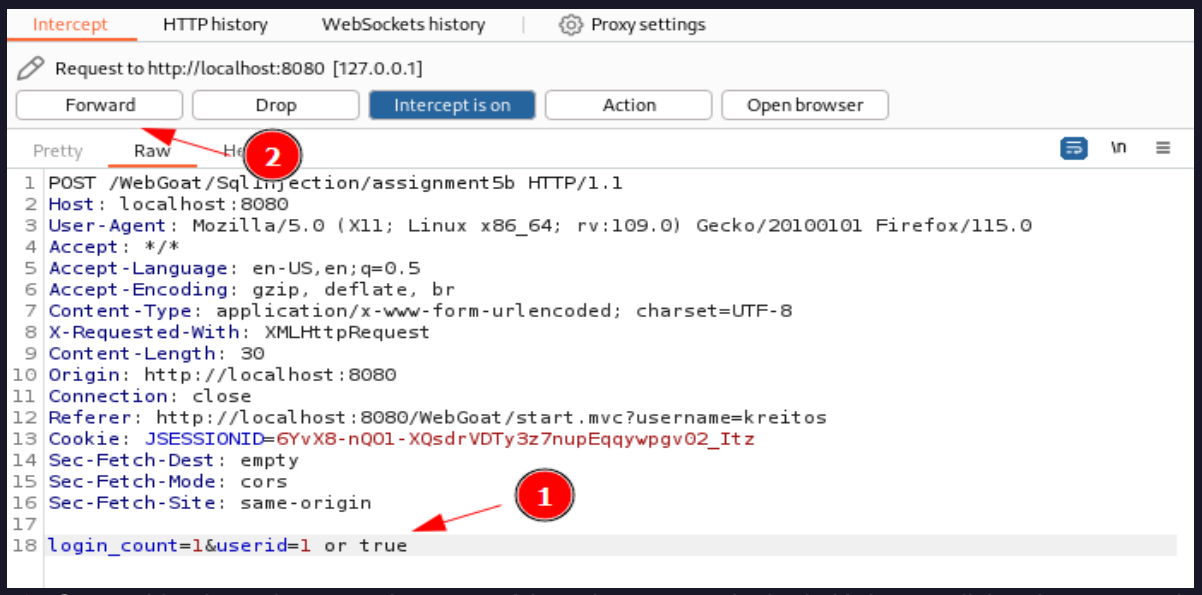
Impacto:
Un atacante podría explotar esta vulnerabilidad para recuperar datos confidenciales de la tabla *user_data*, comprometiendo así la integridad y confidencialidad de la información.

Detalles de la auditoría:



1. La consulta *SQL SELECT * FROM user_data WHERE Login_Count = 1 AND userid = 1 OR TRUE* busca recuperar todos los registros de la tabla *user_data* donde la condición sea verdadera. La condición incluye dos partes:
   - *Login_Count = 1*: Esto busca registros donde el valor de la columna *Login_Count* sea igual a *1*.
   - *userid = 1 OR TRUE*: Esta parte está diseñada para hacer que la condición sea siempre verdadera. La expresión *userid = 1* verifica si el valor de la columna *userid* es igual a *1*. Sin embargo, la adición de *OR TRUE* significa que la condición completa será verdadera, ya que *TRUE* siempre es verdadero. Esto hace que la consulta devuelva todos los registros de la tabla *user_data*, independientemente de su valor real en la columna *userid*.
2. Se consigue la información sensible de la tabla.

También es posible interceptar la petición con BurpSuite:



1. Se cambian los valores a valores numéricos, intenta manipular la lógica condicional para que siempre sea verdadera, lo que podría permitir el acceso no autorizado a datos sensibles o realizar acciones no deseadas en la base de datos. Esto ocurre porque los datos de entrada no se validan correctamente, lo que permite insertar o manipular comandos SQL en las consultas de la base de datos.
2. Se envía la petición modificada.

String SQL Injection CIA

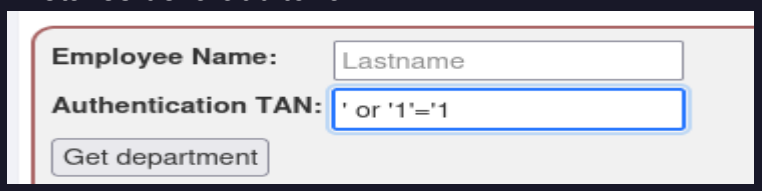
Descripción:

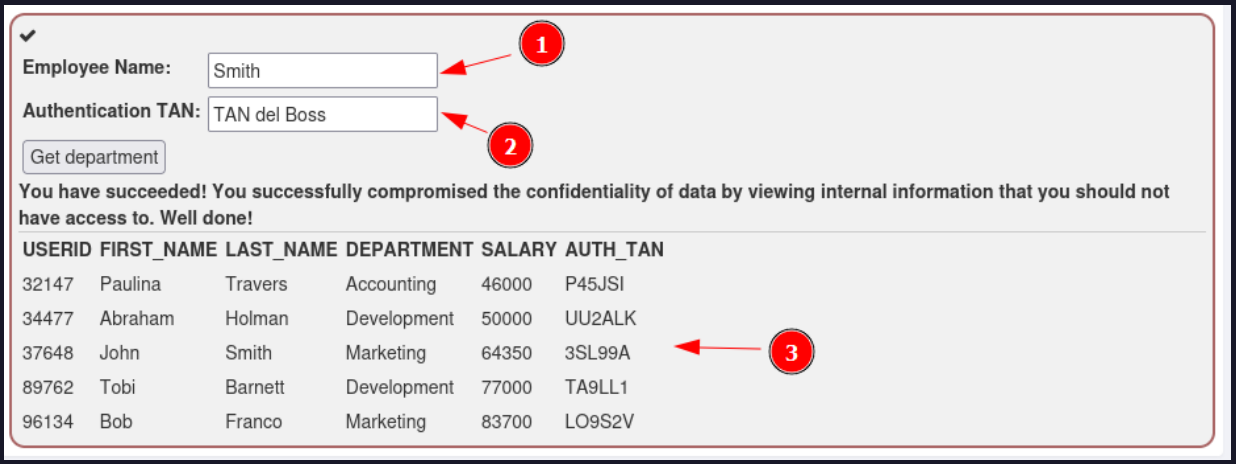
Se ha identificado una vulnerabilidad significativa en la auditoría de la aplicación web. La vulnerabilidad está relacionada con la inyección de SQL de cadena, específicamente utilizando la cadena maliciosa *' or '1'='1*. Esto podría permitir a un atacante comprometer la integridad y confidencialidad de los datos almacenados en la base de datos.

Impacto:

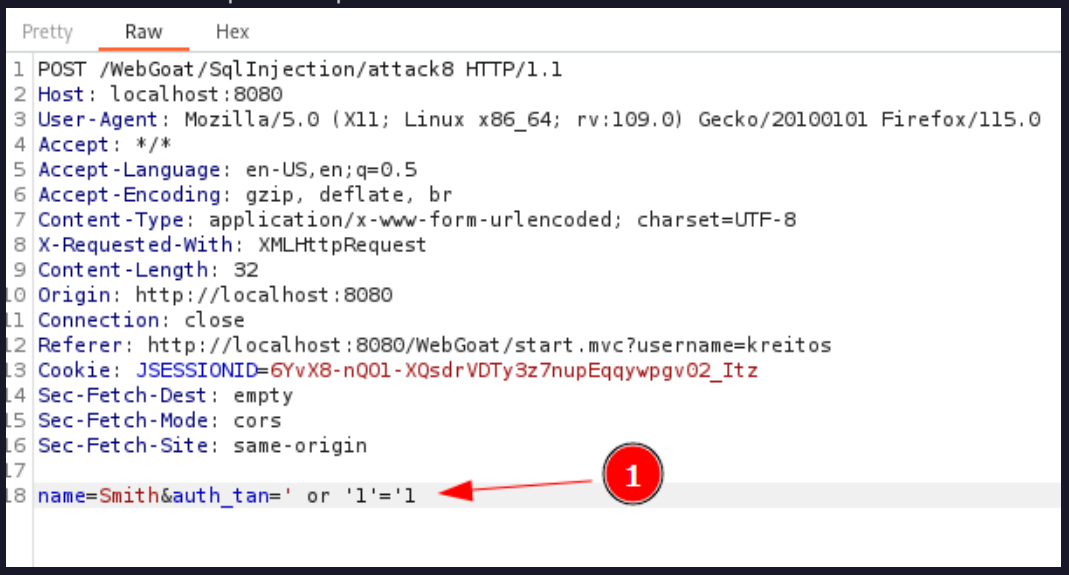
Un atacante podría explotar esta vulnerabilidad y obtener acceso no autorizado a datos confidenciales. Modificar indebidamente de registros en la base de datos y la posibilidad de realizar acciones administrativas no autorizadas.

Detalles de la auditoría:





1. Acceder a la interfaz de auditoría de la aplicación. Ingresar cualquier valor, ya que será interceptado posteriormente con BurpSuite (imágen inferior [1])
2. En el campo correspondiente, ingresar la cadena *' or '1'='1*.
3. Observar cualquier comportamiento inusual o acceso no autorizado a datos.



Triad violation Union SQL

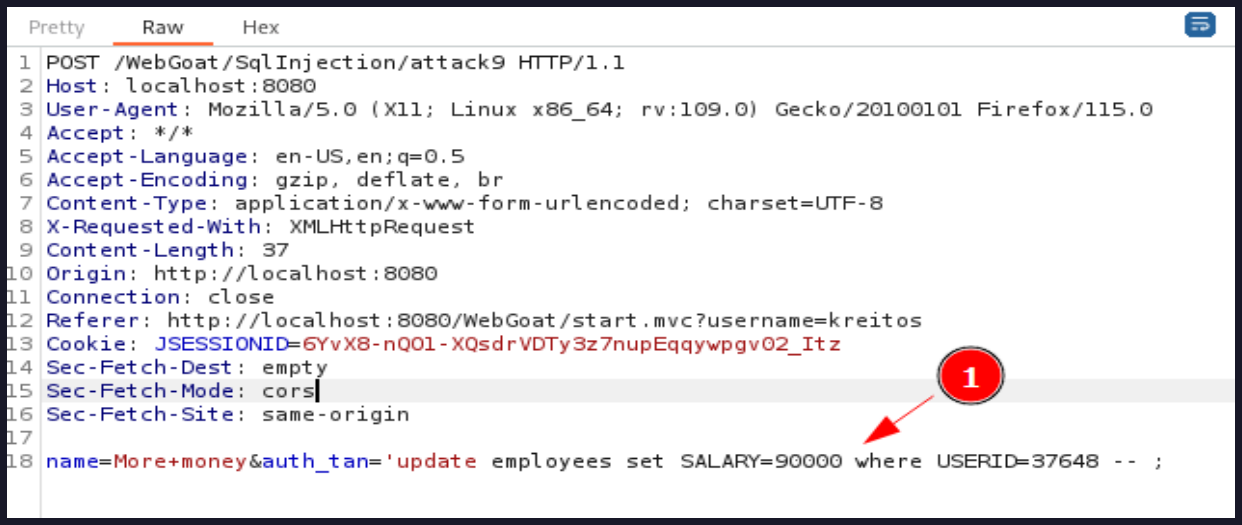
Descripción:

Se identificó una grave vulnerabilidad relacionada con el compromiso de la *integridad* y *disponibilidad* de los datos mediante el uso del encadenamiento de consultas SQL. Esta vulnerabilidad permite a un atacante modificar información crítica en la base de datos, *violando la integridad del sistema.*

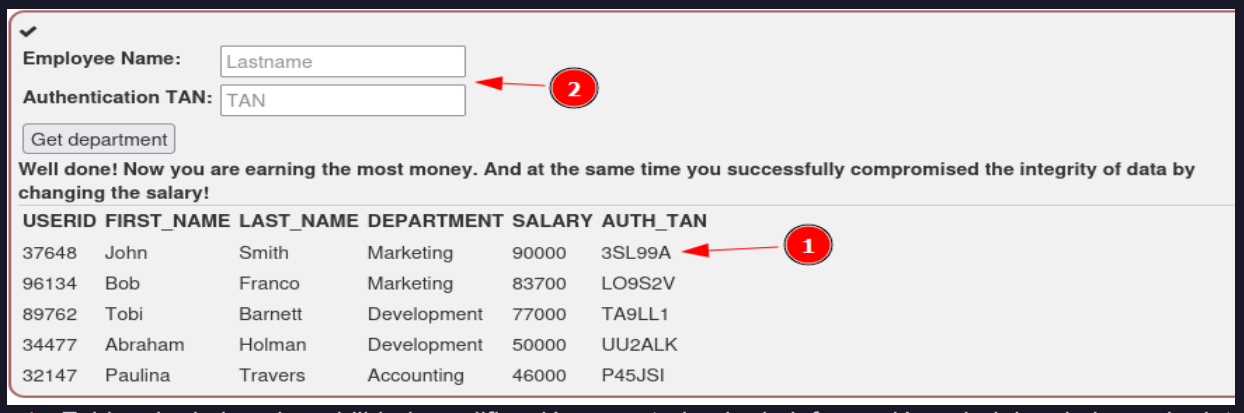
Impacto:

Un atacante podría explotar esta vulnerabilidad para poder modificar de forma no autorizada información salarial en la base de datos. También puede eliminar la tabla *access_log*, comprometiendo así la disponibilidad de los registros de acceso.

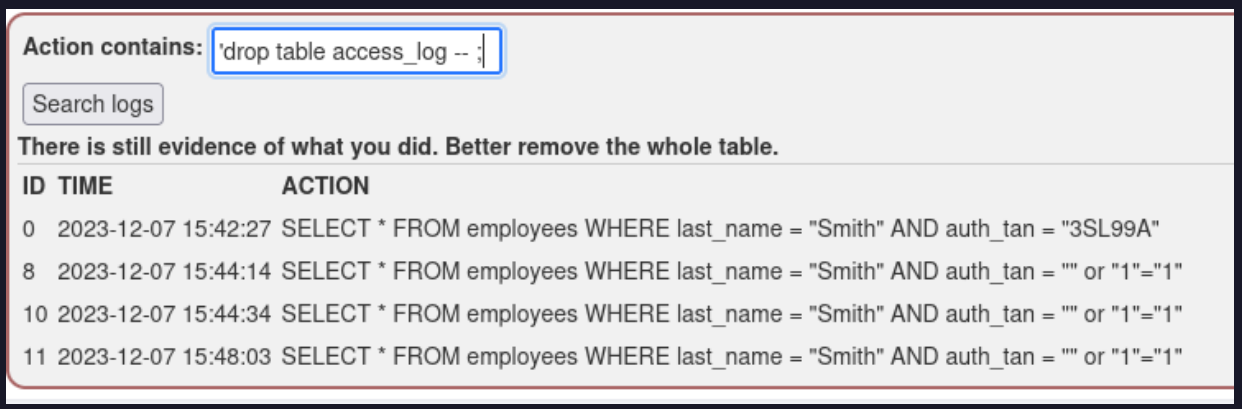
Detalles de la auditoría:



1. Se descubrió que al introducir la siguiente consulta en el campo de búsqueda, se logra modificar el salario del empleado con *USERID 37648*, comprometiendo así la integridad de los datos.
   - *'update employees set SALARY=90000 where USERID= 37648 -- ;*



1. Evidencia de la vulnerabilidad, modificación no autorizada de información salarial en la base de datos.
2. Se puede introducir la query maliciosa directamente en el campo *Authentication TAN*



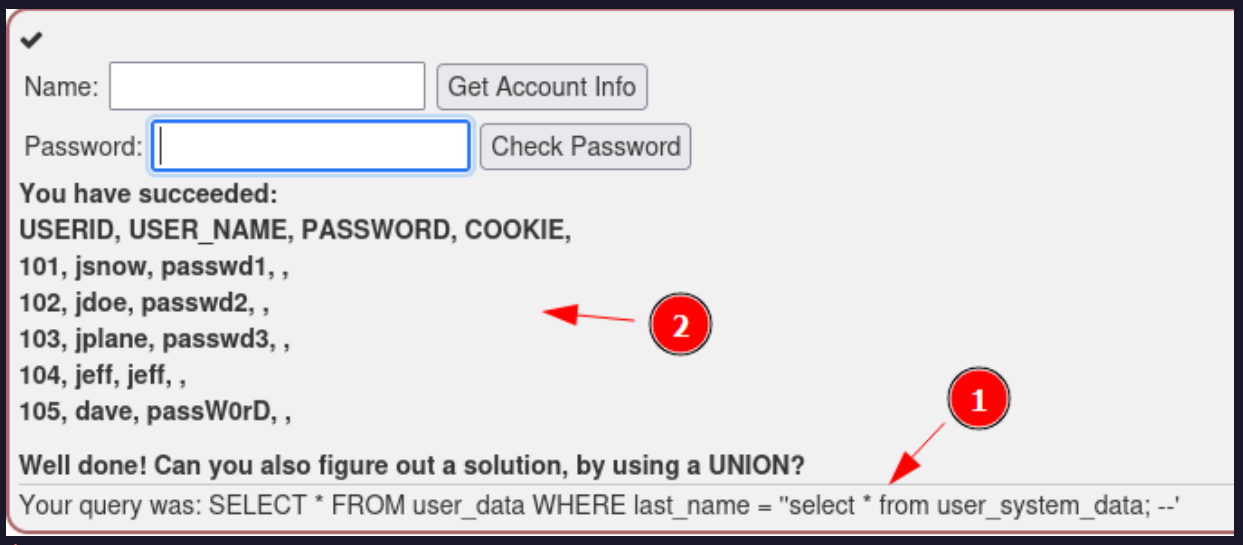
1. Se identificó que la plataforma dispone de un *registro de actividades* lo cual es bueno para hacer seguimiento de las actividades de los usuarios
2. Se quedaron registradas todas las *querys maliciosas* que el atacante registro hasta el momento, pero a la vez se detecto una vulnerabilidad que permite violar la disponibilidad de la tabla *access_log* borrando las evidencias:
   - 'drop table access_log - - ;

Union SQL Injection

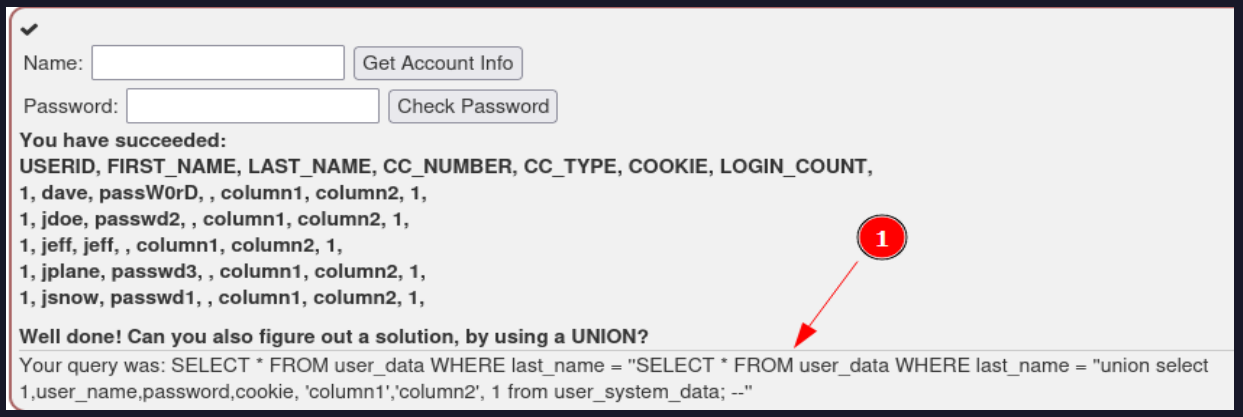
Descripción:
Se identificó la posibilidad de concatenar queries utilizando la cláusula *UNION* para inyectar peticiones. Esta consulta intenta recuperar datos de dos tablas diferentes *user_data* y *user_system_data* y unirlos en una sola salida.

Impacto:
Un atacante podría explotar esta vulnerabilidad para revelar información sensible de la base de datos, comprometiendo así la integridad y confidencialidad de la información.

Detalles de la auditoría:



1. La consulta original busca datos en la tabla *user_data* donde *last_name* está vacío *''*. Luego, utiliza la cláusula *UNION* para agregar los resultados de una segunda consulta que selecciona datos específicos de la tabla *user_system_data*.



1. *union select 1, user_name, password, cookie, 'column1', 'column2', 1 from user_system_data* busca seleccionar ciertos valores de la tabla *user_system_data*. Estos valores incluyen *1* como un marcador, seguido por *user_name*, *password*, *cookie* de la tabla *user_system_data*, y algunas cadenas *'column1', 'column2', 1* para coincidir con la estructura de la consulta original.
2. Si no se introduce el número de columnas que tiene la segunda tabla para que se realice la petición query correctamente da un error, que te indica por qué no funcionó la query, por lo tanto es información que los atacantes pueden usar a su favor.
    - *column number mismatch detected in rows of UNION, INTERSECT, EXCEPT, or VALUES operation*

Injection Blind SQL

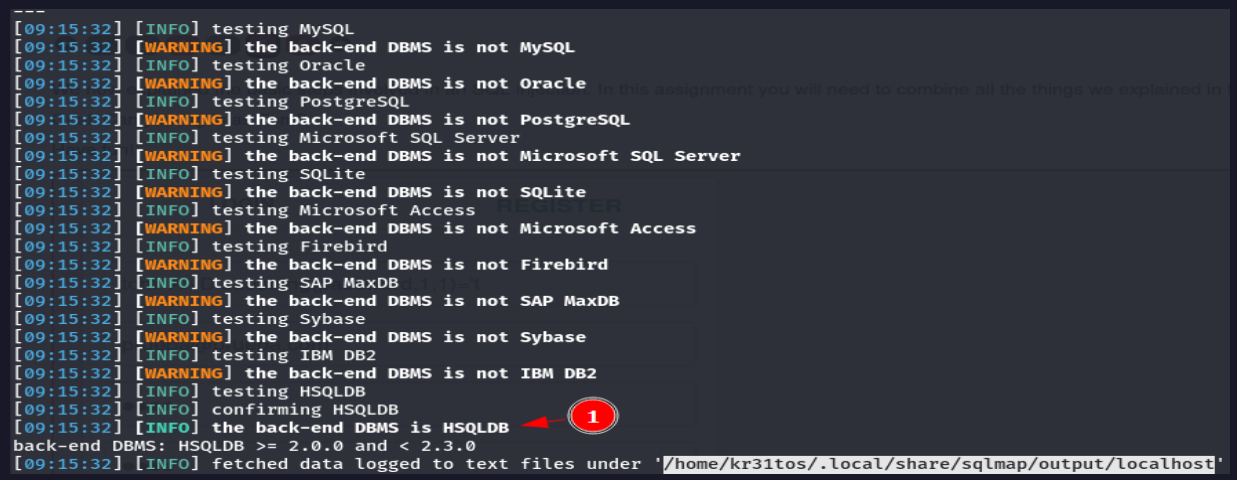
Descripción:

Una Blind SQL Injection es una amenaza seria que permite a un atacante extraer información confidencial de una base de datos sin ver los resultados directos. Se procede a la intrusión a través del apartado *Register*.

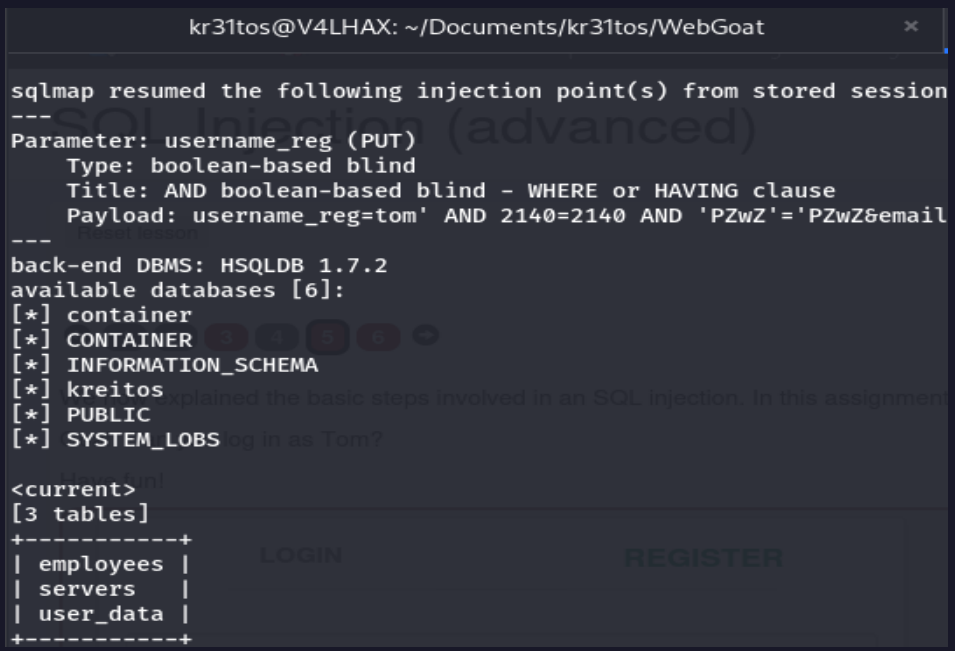
Impacto:

Un atacante puede conseguir datos sensibles, como contraseñas y detalles financieros, además de exponer la estructura interna de la base de datos. Puede conducir a ataques de denegación de servicio (DoS) y, en casos extremos, comprometer por completo el sistema, dando al atacante control sobre el servidor.

Detalles de la auditoría:



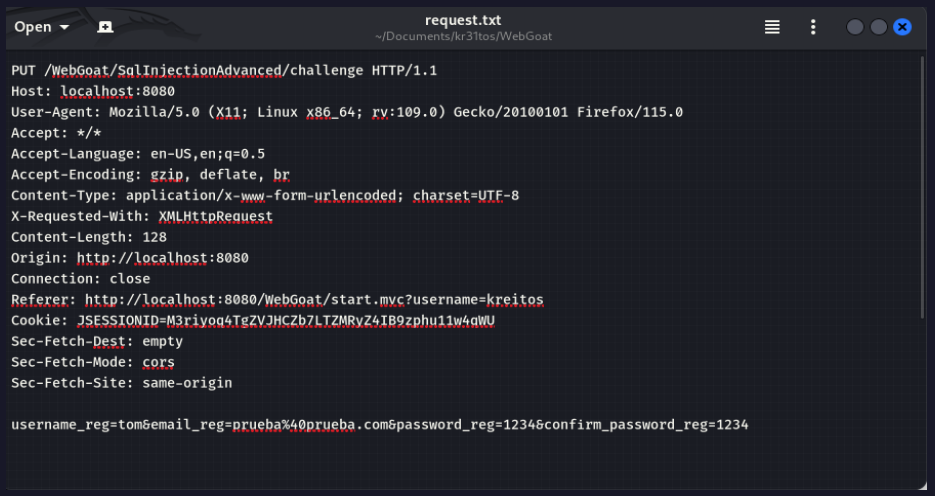
1. Se procede a escanear con la herramienta SQLMap en busca de vulnerabilidades. Se identifica la base de datos subyacente, revelando que se trata de HSQLDB versión 1.7.2.



1. Durante el análisis, se descubrieron tres tablas en la base de datos, *employees, servers y user_data.*

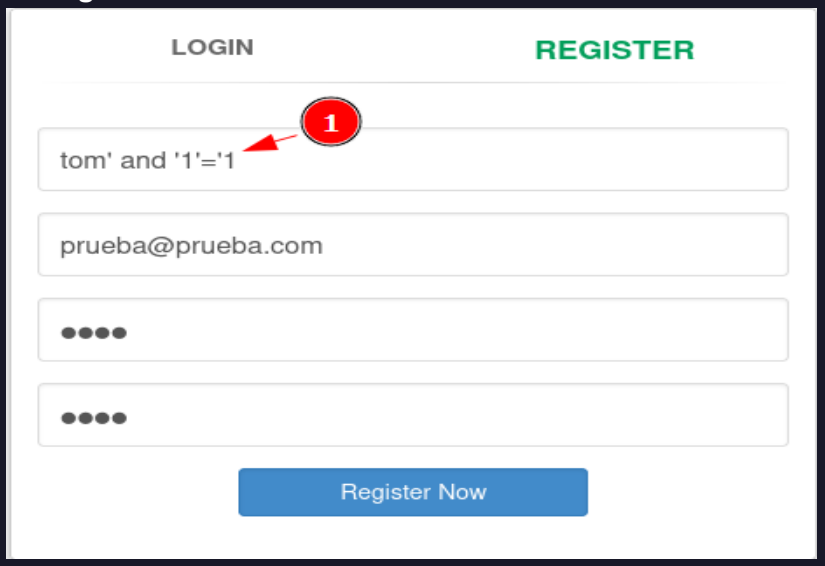
```
┌──(kr31tos㉿V4LHAX)-[~/…/share/sqlmap/output/localhost]
└─$ ls
log  session.sqlite  target.txt

┌──(kr31tos㉿V4LHAX)-[~/…/share/sqlmap/output/localhost]
└─$ cat target.txt
http://localhost:8080/WebGoat/SqlInjectionAdvanced/challenge (POST)  # /usr/bin/sqlmap -r request.txt

username_reg=tom&email_reg=prueba@prueba.com&password_reg=1234&confirm_password_reg=1234

┌──(kr31tos㉿V4LHAX)-[~/…/share/sqlmap/output/localhost]
└─$ cat log
sqlmap identified the following injection point(s) with a total of 429 HTTP(s) requests:
---
Parameter: username_reg (PUT)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: username_reg=tom' AND 6626=6626 AND 'cFIS'='cFIS&email_reg=prueba@prueba.com&password_reg=1234&confirm_password_reg=1234
---
back-end DBMS: HSQLDB >= 2.0.0 and < 2.3.0

┌──(kr31tos㉿V4LHAX)-[~/…/share/sqlmap/output/localhost]
└─$
```

1. Se introduce en un archivo creado *request.txt* los datos de la petición que se obtuvo con BurpSuite y se escanean.

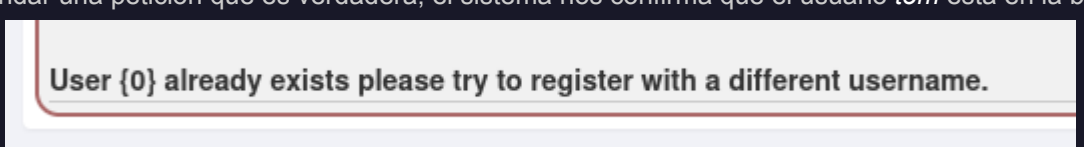


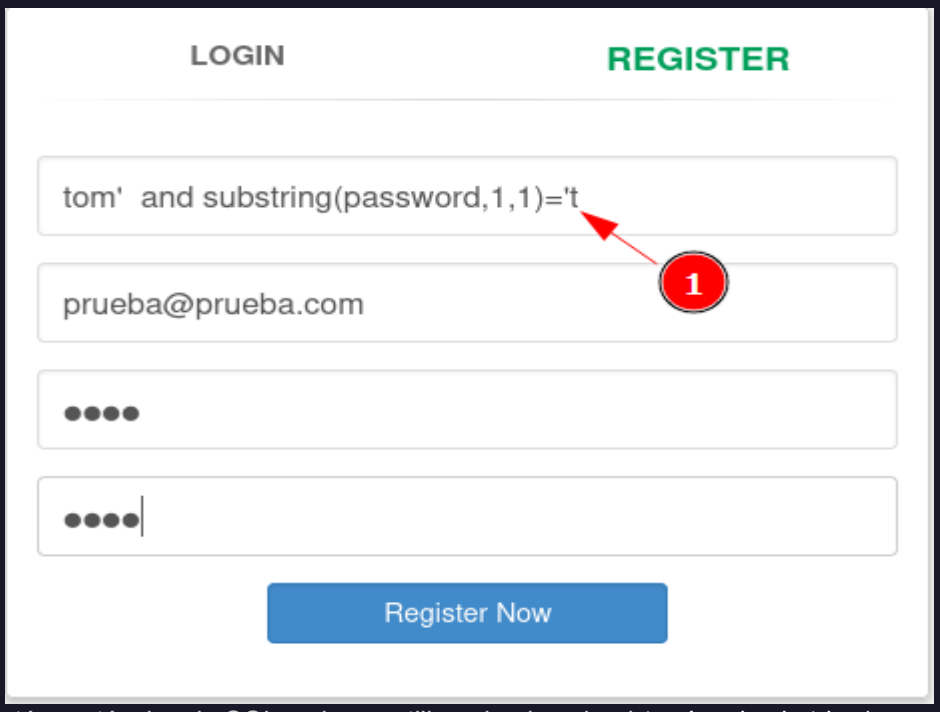
2. La herramienta SQL indicó la existencia de una posible vulnerabilidad de *SQL Blind Injection*. Se procedió con pruebas adicionales.
3. También determina el rango de la posible versión de la base de datos.

Pruebas en el Registro de Nuevo Usuario:

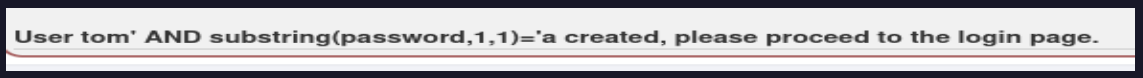


1. Se realizó una inyección SQL exitosa al intentar registrar un nuevo usuario con el payload *' or '1'='1*, confirmando la existencia de la vulnerabilidad.
2. Al mandar una petición que es verdadera, el sistema nos confirma que el usuario *tom* está en la base de datos.

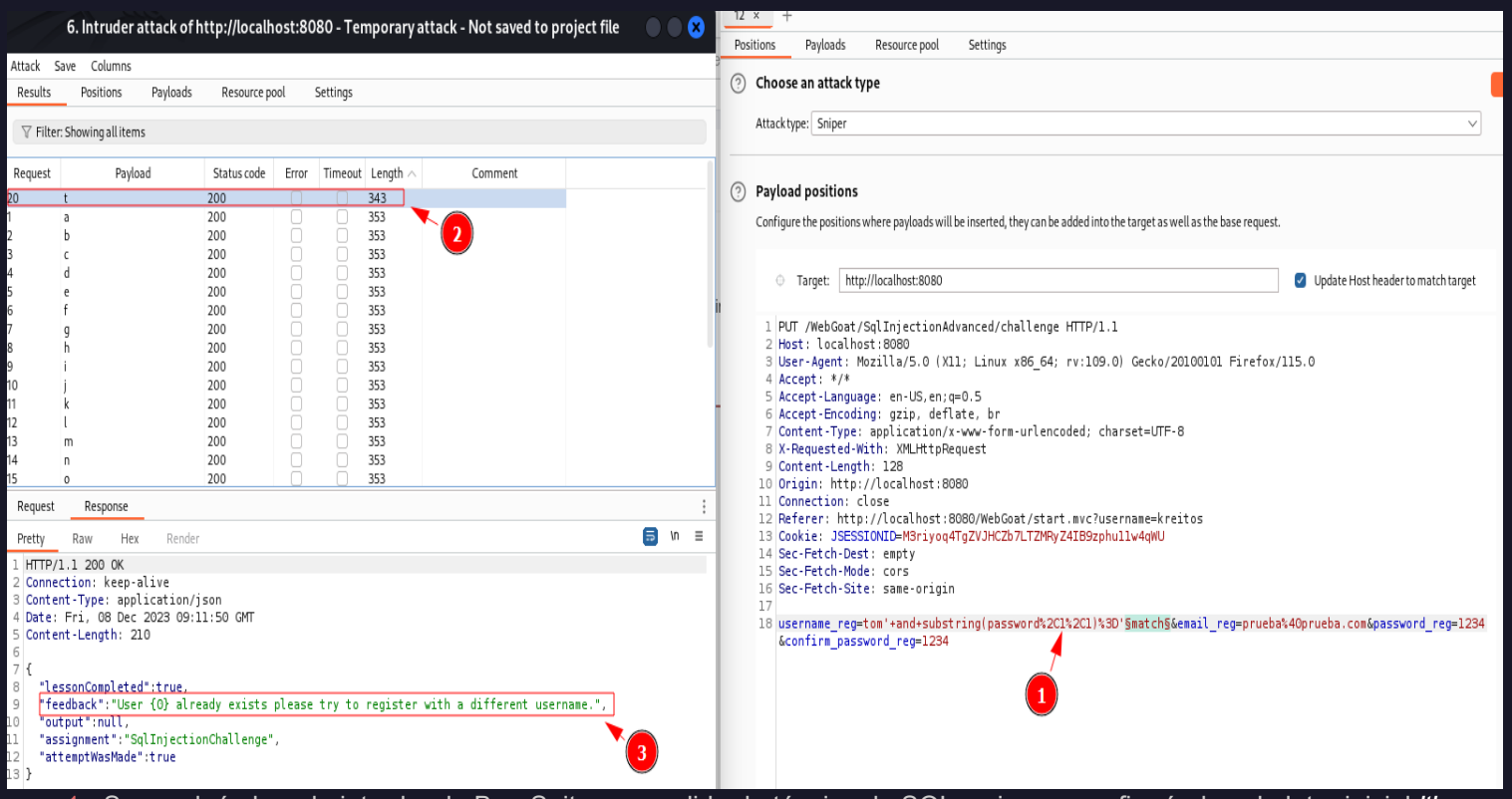




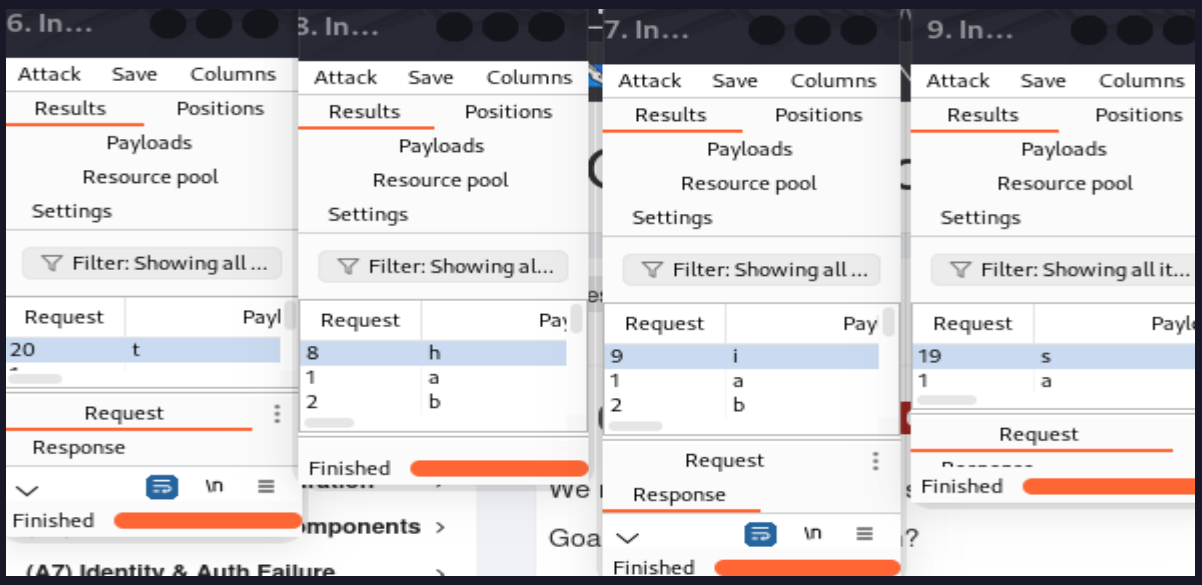
1. Se implementó una técnica de SQL a ciegas utilizando el payload *tom' and substring(password,1,1)='a* para identificar cual es la primera letra de la contraseña. Si la contraseña no contenía la letra, sacaba lo siguiente por consola *(imagen inferior)*, creando un nuevo usuario e indicando que no hay nada parecido en la base de datos.



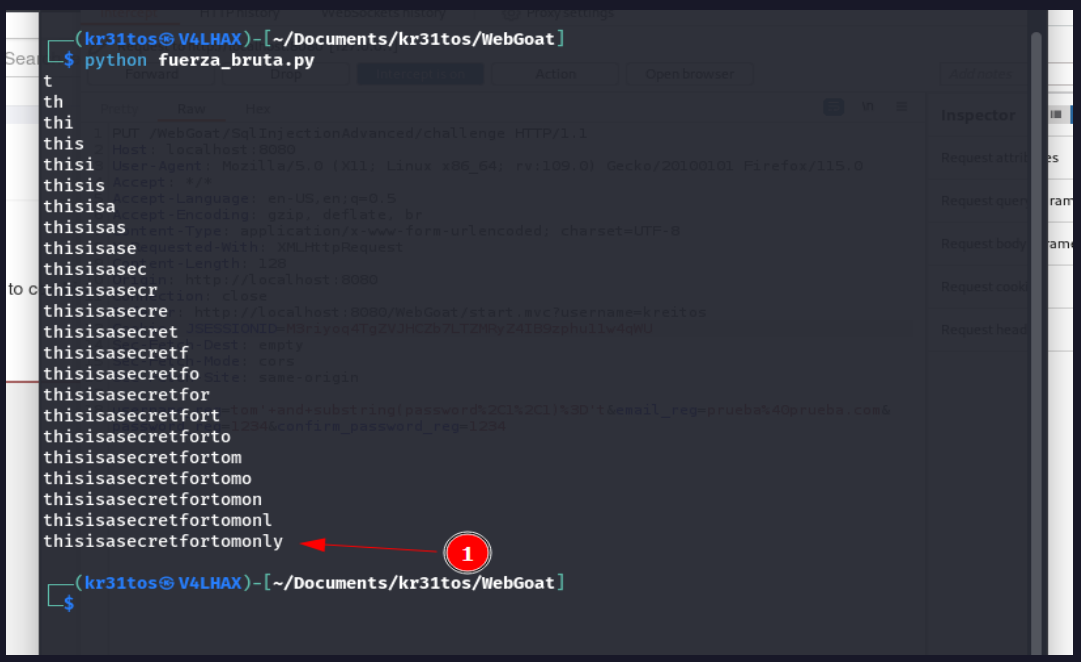
2. Se prueba todo el abecedario hasta hacer match, el primer match es con la letra *t,* así, el atacante ya tiene un dato muy importante, la primera letra de la contraseña.



1. Se empleó el modo intruder de BurpSuite para validar la técnica de SQL a ciegas, confirmándose la letra inicial *'t'* de la contraseña. Posteriormente, se introduce un *payload [1]* para hacer match y aumentando de 1 en 1 los valores del índice de la contraseña descubriendo que la primera palabra de la contraseña es *this (imágen inferior)*



Al tratarse de un proceso muy lento y tedioso, que consume bastante recurso y tiempo, se desarrolló un script en Python que automatiza el proceso de fuerza bruta mediante la URL del login y la cookie obtenida con Burpsuite.



1. El script logró descifrar la contraseña de Tom como *thisisasecretfortomonly*, proporcionando acceso a la web.
2. Se consigue acceso a la cuenta del usuario

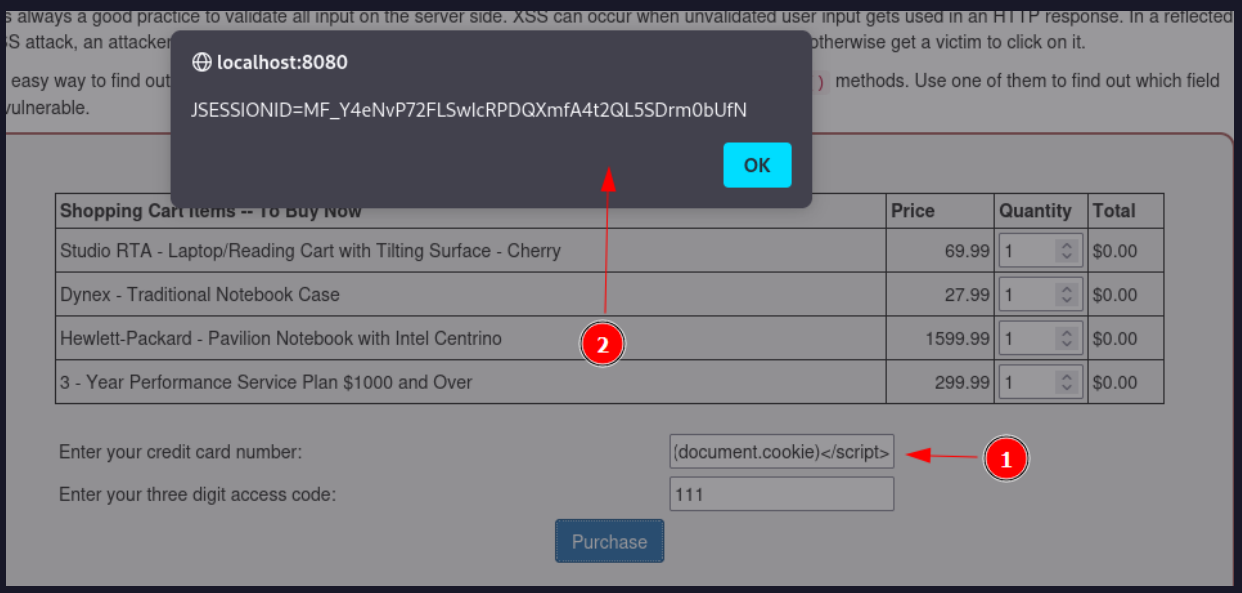
Injection Reflected XSS

Descripción:
Esta vulnerabilidad permite al atacante insertar scripts maliciosos en los datos de entrada, que luego son reflejados y ejecutados por el navegador del usuario. El atacante prueba los métodos *alert()* o *console.log()* para comprobar si los campos son vulnerables.

Impacto:
Puede ser significativo, ya que los scripts maliciosos pueden robar información confidencial, como *cookies de sesión*, realizar acciones en nombre del usuario autenticado y redirigir a sitios web maliciosos. También puede ser utilizado para propagar malware o realizar *ataques de phishing*.

Detalles de la auditoría:

1. A través de la inserción de etiquetas de scripts de HTML en el campo validado incorrectamente se pueden obtener

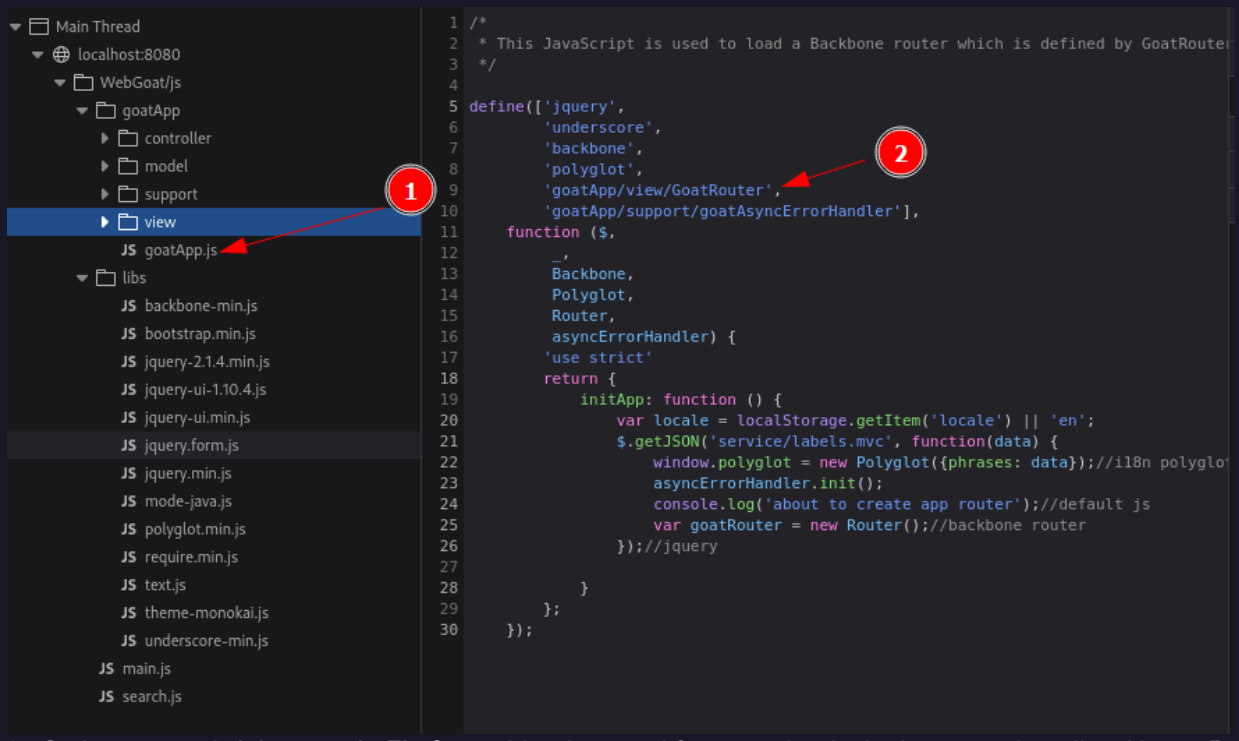


   cookies de sesión, cambiar precios de productos, introducir productos nuevos…
2. Cookie de la sesión actual

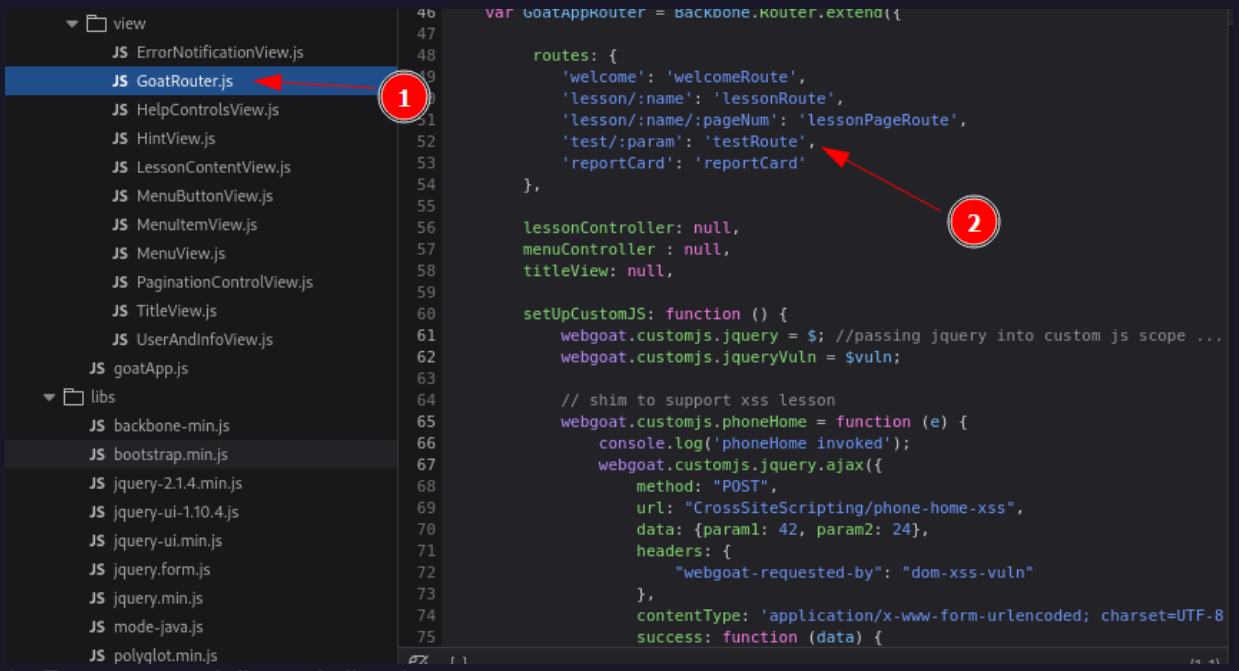
DOM-Based XSS

Descripción:
Es una vulnerabilidad en la que los ataques ocurren en el DOM, generalmente se puede encontrar buscando las *configuraciones de ruta* en el código del lado del cliente. A diferencia de otras formas de XSS, en este caso, el navegador es el responsable de ejecutar el script malicioso.

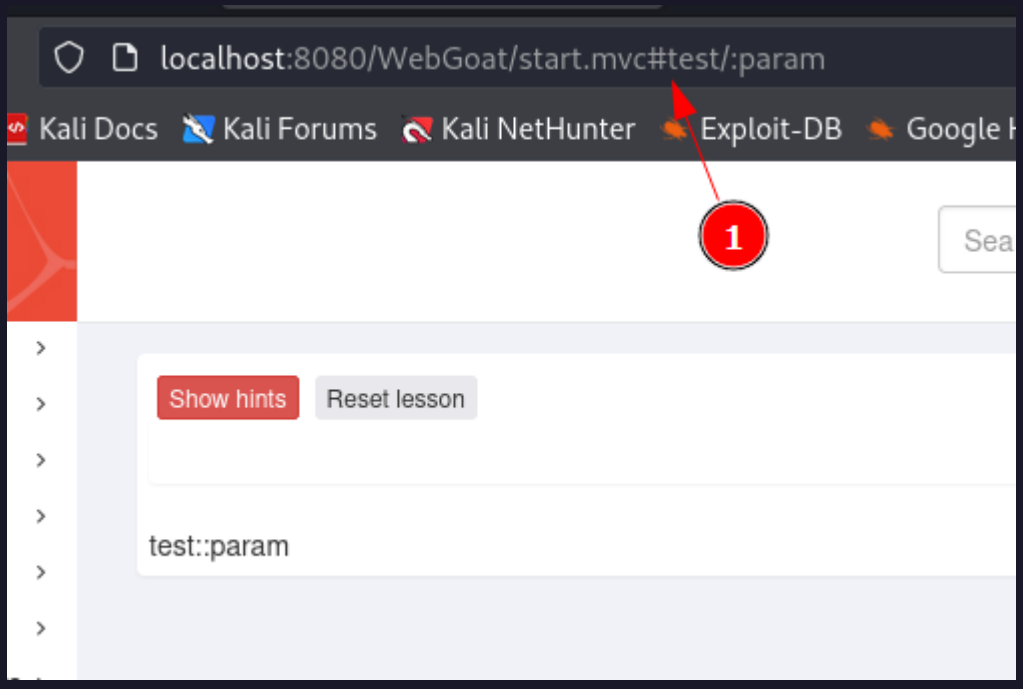
Impacto:
El impacto puede ser grave, ya que los scripts maliciosos pueden alterar dinámicamente el contenido de la página web, afectando a otros usuarios que visualizan la misma página. Puede ser utilizado para robar información sensible, realizar acciones no autorizadas en nombre del usuario y comprometer la integridad de la página.
Detalles de la auditoría:



1. Se busca en el *debugger de Firefox*, sabiendo que el framework principal que usa la aplicación es *Backbone*, donde está la ruta que contiene el posible archivo manipulable.
2. Investigando y buscando en los directorios, se detecta que en la ruta *goatApp/view/goatRouter* hay un directorio que se llame *test.*

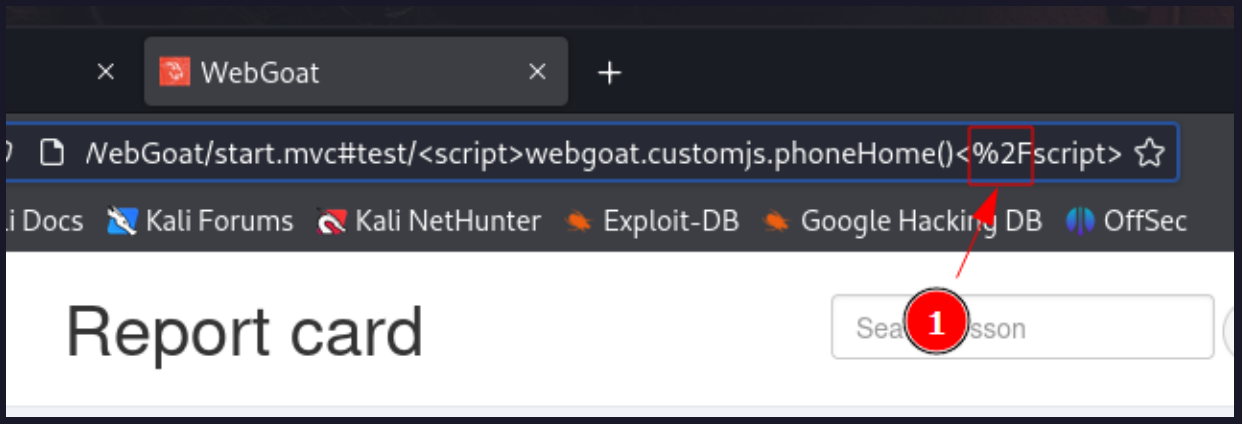


1. Entramos en el directorio listado
2. Nos encontramos con la ruta del directorio test donde puede haber un archivo manipulable e intentamos probarlo en la URL del navegador introduciendo la siguiente ruta
   - *localhost:8080/WebGoat/start.mvc#test/:param (imagen inferior)*



También se puede ejecutar etiquetas scripts maliciosas directamente en la URL del navegador, lo único que hay que tener en cuenta es cambiar la barra diagonal / para que no la lea como un directorio, por su código equivalente de las URLs, *%2F (imágen inferior y demostración)*

1. Una forma sencilla de determinar el código de un carácter es interceptando la petición, y el navegador lo transforma solo, esto se ha hecho mediante BurpSuite



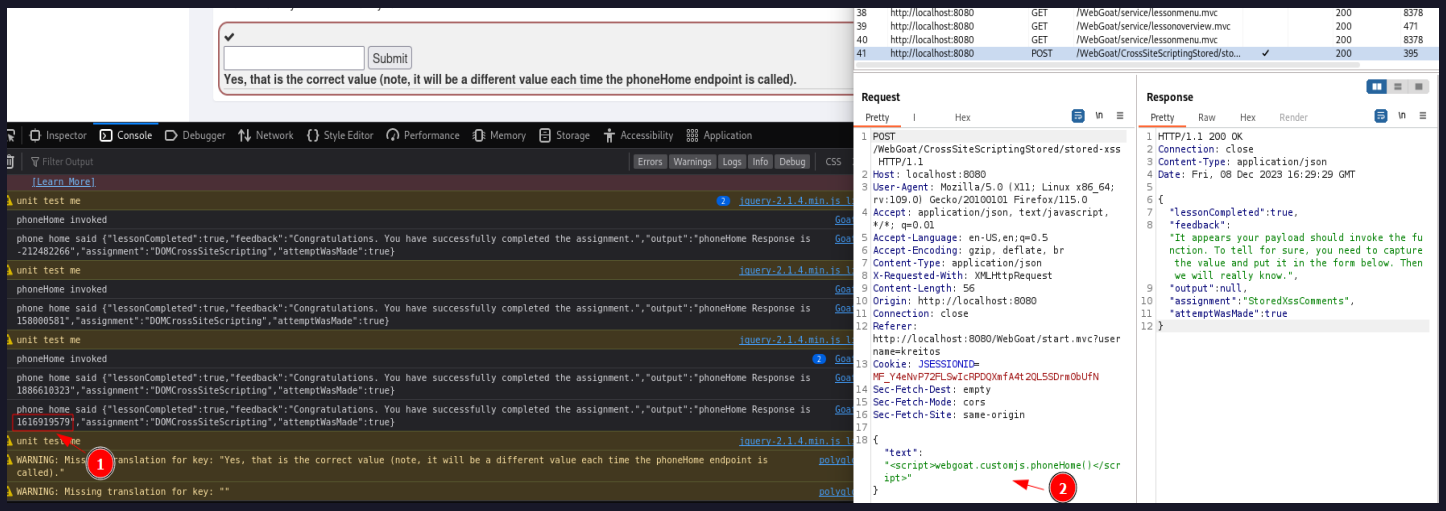
1. Se introduce el script malicioso directamente en la URL, en esta ocasión se ha llamado a la función *webgoat.customjs.phoneHome()* para poder completar el ejercicio, pero hay riesgo de que un atacante consiga inyectar XSS, redirecciones maliciosas, robo de cookies, keyloggins entre los más destacados.

Stored XSS

Descripción:
Es una vulnerabilidad de seguridad en aplicaciones web donde un atacante inserta scripts maliciosos que se almacenan y luego se ejecutan en el navegador de otros usuarios. Estos scripts pueden ser almacenados en una base de datos u otro lugar persistente, y cuando un usuario legítimo accede a la página afectada, el script se ejecuta sin su conocimiento.

Impacto:
Puede ser grave, ya que permite a los atacantes inyectar scripts maliciosos directamente en la aplicación, lo que puede conducir al robo de sesiones, robo de información confidencial, manipulación de contenido y otros ataques. Los usuarios afectados pueden ser redirigidos a sitios maliciosos, tener sus cookies comprometidas o ser víctimas de phishing.

Detalles de la auditoría:



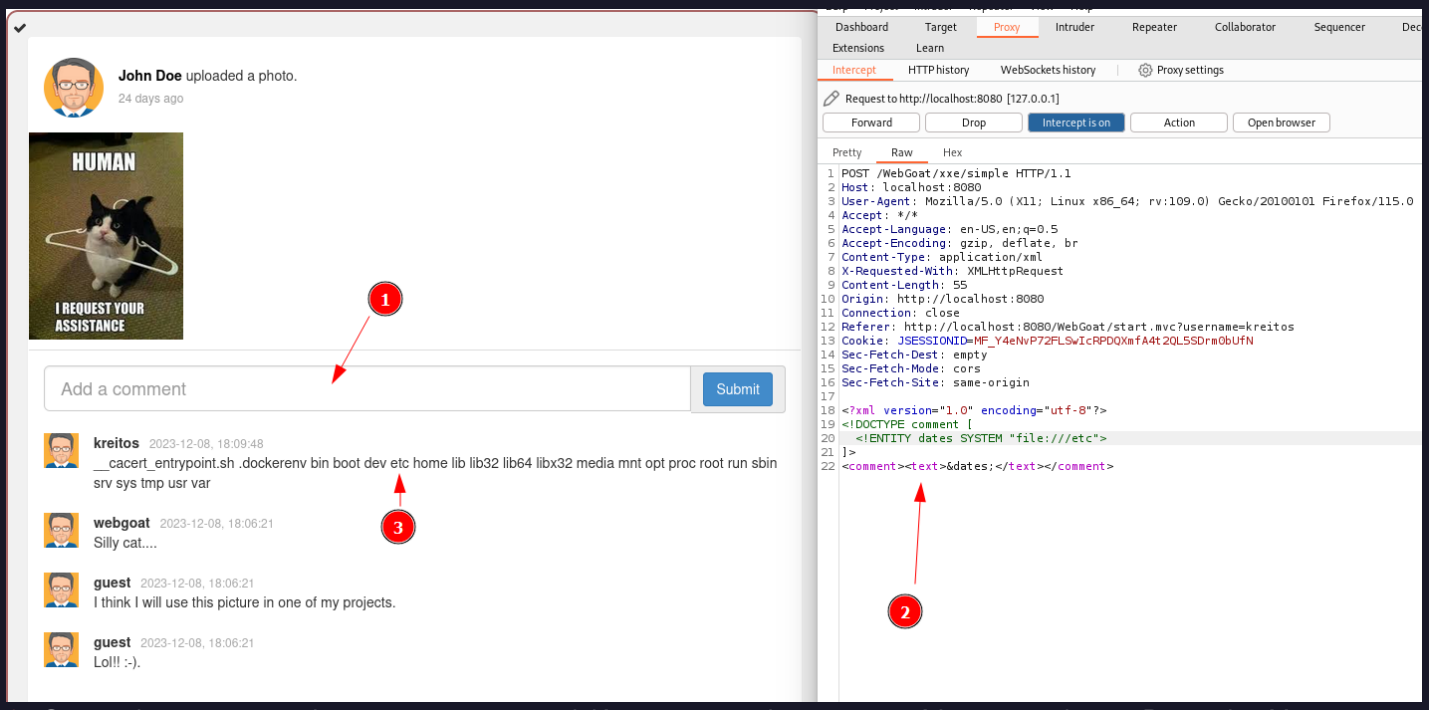
1. Al escribir en el campo submit de la aplicación, se intercepta la petición y se cambia el valor del comentario para que llame a una función interna que está dentro de DOM
2. Se intercambia el comentario por la siguiente script malicioso:
   - *<script>webgoat.customjs.phoneHome()</script>*

XXE Injection

Descripción:
Es una vulnerabilidad que ocurre cuando una aplicación procesa *datos XML* no confiables que contienen referencias a entidades externas. Las entidades externas permiten la inclusión de recursos externos, como archivos locales o URLs, en el documento XML, lo que puede ser explotado de manera maliciosa.

Impacto:
Puede ser grave, ya que los atacantes pueden aprovecharlo para leer archivos sensibles del sistema, realizar escaneos internos de la red, llevar a cabo ataques de denegación de servicio y, en algunos casos, ejecutar código remoto. La explotación exitosa de XXE puede conducir a la revelación de información confidencial y comprometer la seguridad de la aplicación y del sistema en general.

Detalles de la auditoría:

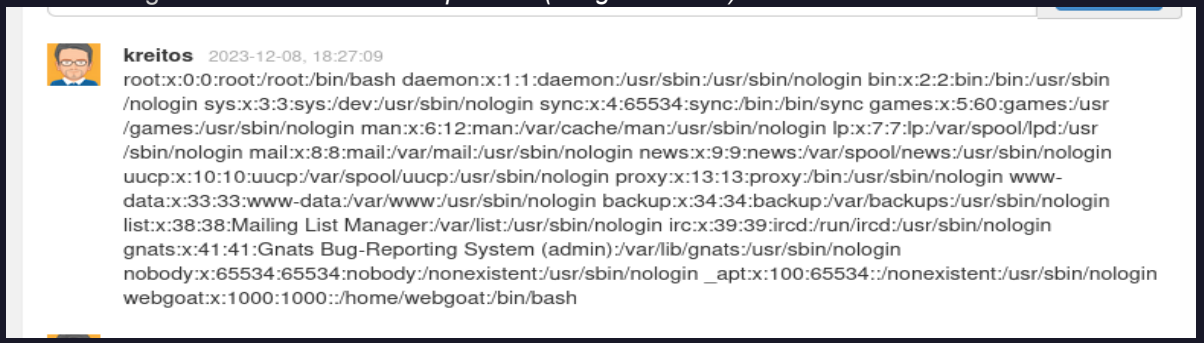


1. Se añade un comentario que genera una petición, que posteriormente será interceptada con Burpsuite. Y se cambia por el siguiente bloque XML:
2. Se sustituye el comentario original por la referencia a una entidad exterior que llame al archivo que puede ser comprometido

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

3. Devuelve en el campo del comentario las rutas y los directorios de la aplicación.

1. Se pueden comprometer carpetas sensibles como *etc/passwd*, *etc/shadow*, ejecutar consolas remotas
2. Se consigue acceder a la ruta *etc/passwd (imágen inferior)*

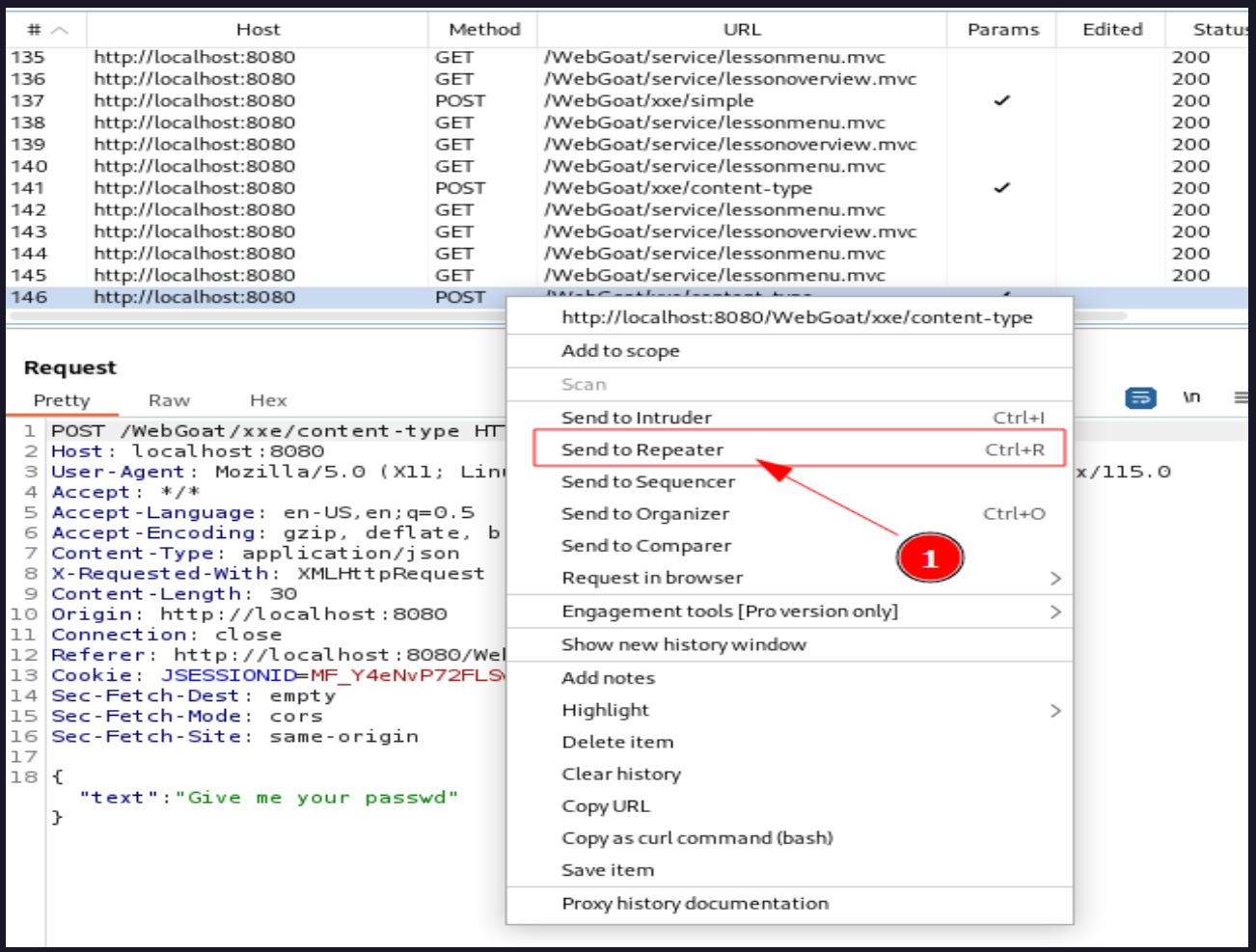


XML Injection

Descripción:
Es una vulnerabilidad que ocurre cuando una aplicación procesa datos XML no confiables sin validar o escapar adecuadamente los caracteres especiales. Los atacantes pueden insertar contenido malicioso en documentos XML, alterando su estructura original y llevando a cabo ataques.

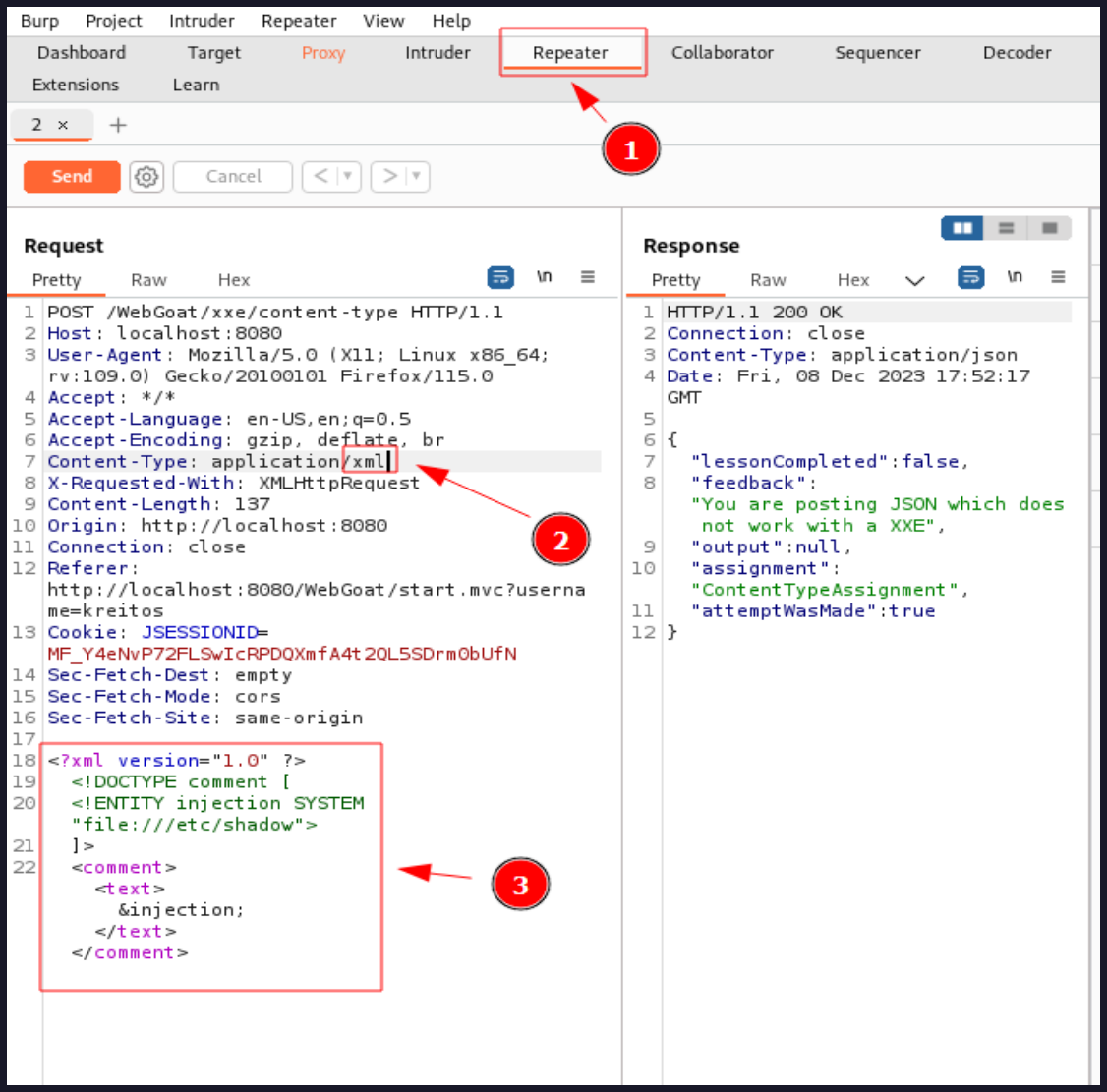
Impacto:
Puede variar, pero generalmente incluye la manipulación no autorizada de datos XML, la alteración de la lógica de la aplicación y la exposición de información sensible. Los atacantes pueden realizar cambios en la estructura del XML para afectar la funcionalidad de la aplicación, llevar a cabo ataques de denegación de servicio (DoS), o incluso ejecutar código arbitrario en algunos casos, comprometiendo la integridad y seguridad del sistema.

Detalles de la auditoría:

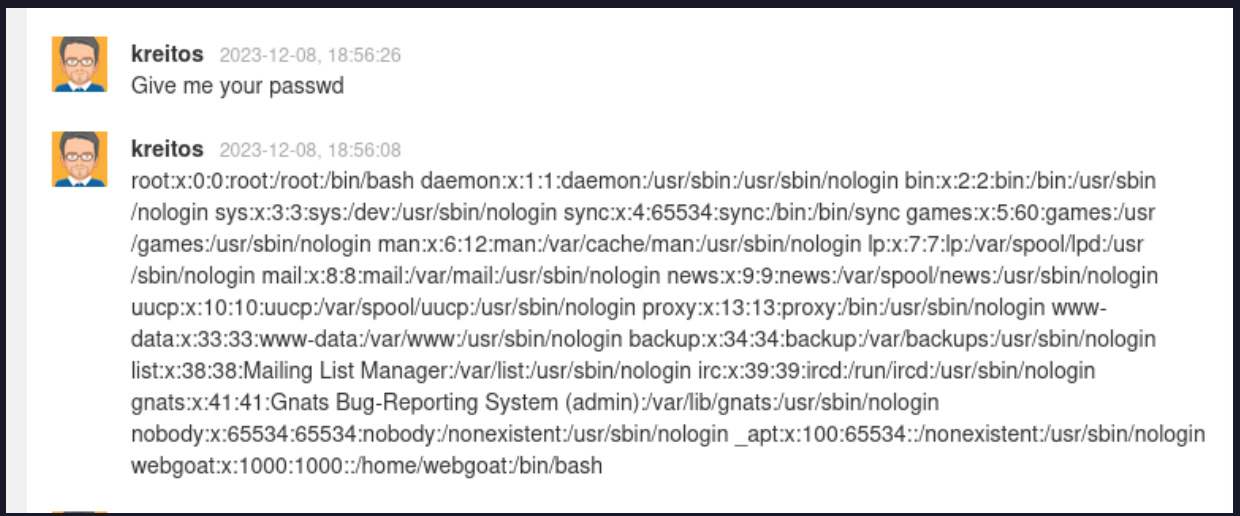


#	Host	Method	URL	Params	Edited	Status
135	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
136	http://localhost:8080	GET	/WebGoat/service/lessonoverview.mvc			200
137	http://localhost:8080	POST	/WebGoat/xxe/simple	✓		200
138	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
139	http://localhost:8080	GET	/WebGoat/service/lessonoverview.mvc			200
140	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
141	http://localhost:8080	POST	/WebGoat/xxe/content-type	✓		200
142	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
143	http://localhost:8080	GET	/WebGoat/service/lessonoverview.mvc			200
144	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
145	http://localhost:8080	GET	/WebGoat/service/lessonmenu.mvc			200
146	http://localhost:8080	POST				

1. Se manda la petición interceptada con Burpsuite al repetidor de este para poder modificar manualmente la petición y realizar los cambios pertinentes que se muestran a continuación *(imágen inferior)*



1. Una vez mandada la petición al repetidor, se hace click en este apartado
2. Acto seguido se procede a cambiar la extensión de *json* a *xml* para que se ejecute el código XML
3. Cambiamos el comentario por el código malicioso obteniendo los siguientes resultados *(imágen inferior)*

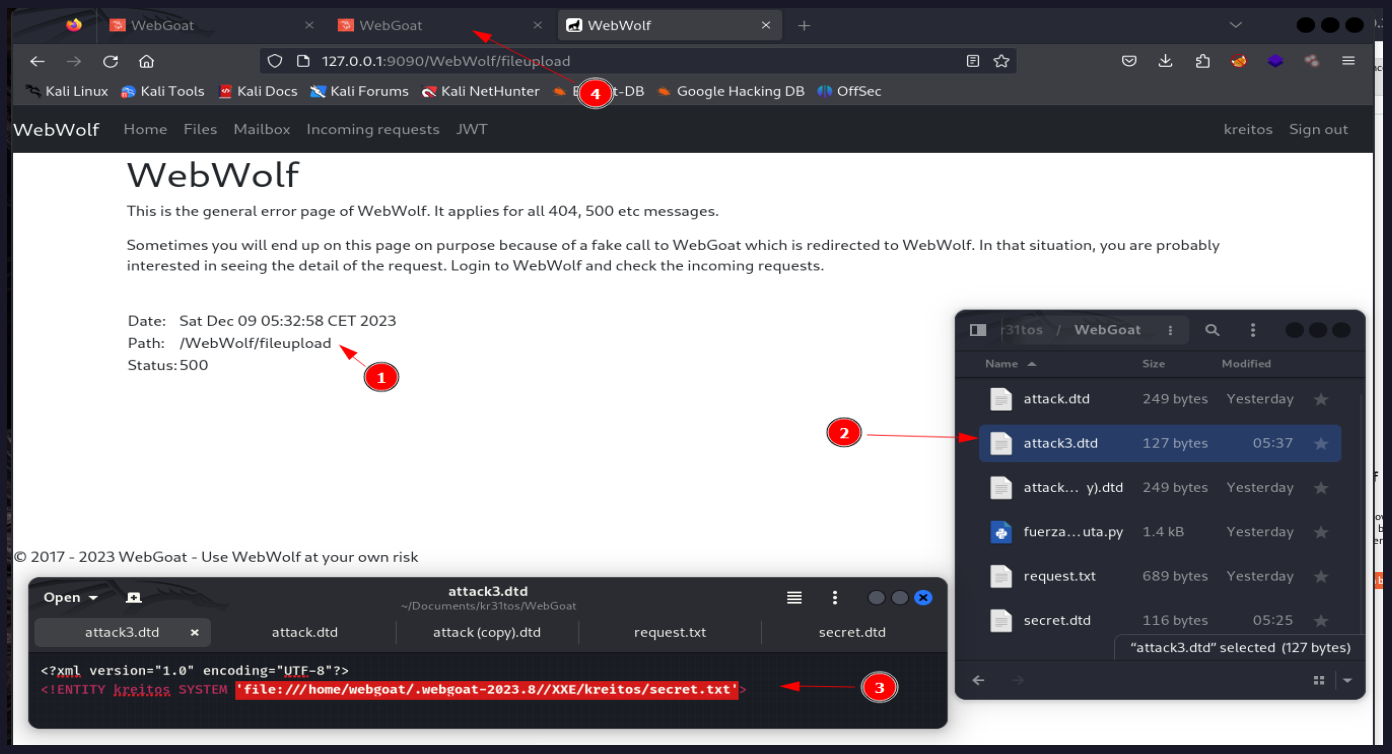


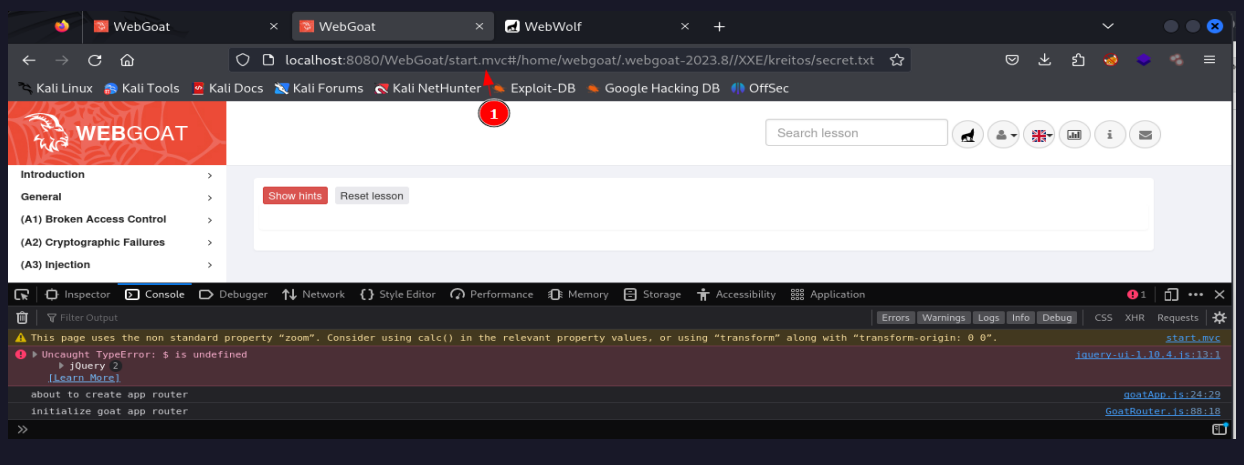
Blind XXE Injection

Descripción:
Es una vulnerabilidad que explota la capacidad de una aplicación para analizar XML con referencias externas como por ejemplo un archivo que contiene código malicioso. En este caso, el atacante inyecta entidades XML maliciosas que pueden resultar en la revelación no autorizada de información sensible.

Impacto:
El impacto lleva a la posibilidad de que un atacante pueda extraer información confidencial del sistema sin recibir directamente la respuesta. Esto puede conducir a la exposición de datos críticos, comprometiendo la confidencialidad y la integridad de la información manejada por la aplicación. Además, puede abrir la puerta a ataques más sofisticados, comprometiendo la seguridad general del sistema.

Detalles de la auditoría:
En esta parte de la auditoría he tenido un problema con la subida de archivos a WebWolf, se han realizado los pasos correctos que pedía el ejercicio, pero sin tener éxito. Dejo las evidencias de los procedimientos realizados.





Analysis of OSS & Analysis of version

Descripción:
El análisis de versiones y el análisis de Software de Código Abierto (OSS) son prácticas cruciales para evaluar la seguridad de una aplicación web. Consisten en revisar las versiones de los componentes y bibliotecas utilizados, así como examinar la seguridad de los componentes de código abierto incorporados en el desarrollo.

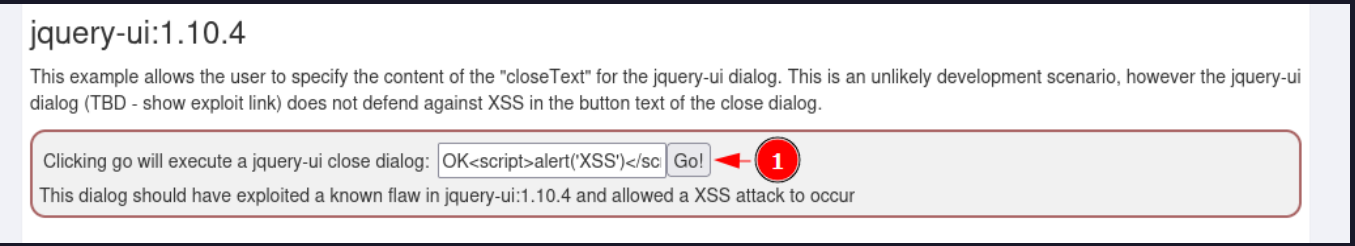
Impacto:
1. El uso de versiones obsoletas de componentes puede exponer la aplicación a vulnerabilidades conocidas. En el ejemplo mencionado, la actualización de jquery-ui a una versión no vulnerable evitó la explotación de una falla de seguridad.

2. El análisis de OSS ayuda a identificar y abordar vulnerabilidades de seguridad en bibliotecas de código abierto. Esto es crucial ya que las amenazas pueden provenir no solo del código personalizado, sino también de componentes externos.

3. Mantener actualizadas las versiones de los componentes y bibliotecas utilizadas en la aplicación es esencial para incorporar parches de seguridad. Esto contribuye a una mejora continua de la seguridad y a la protección contra amenazas conocidas.

4. Al identificar y mitigar las vulnerabilidades a través del análisis de versiones y OSS, se reduce significativamente el riesgo de explotación. Las actualizaciones oportunas y la selección de componentes seguros son fundamentales para mantener la integridad y la seguridad de la aplicación web.

Detalles de la auditoría:
En esta parte de la auditoría no funcionaba correctamente el botón *Go!*, por lo tanto realice un pequeño seguimiento para poder encontrar el problema que estará reflejado en la sección *Anexo* de este informe.

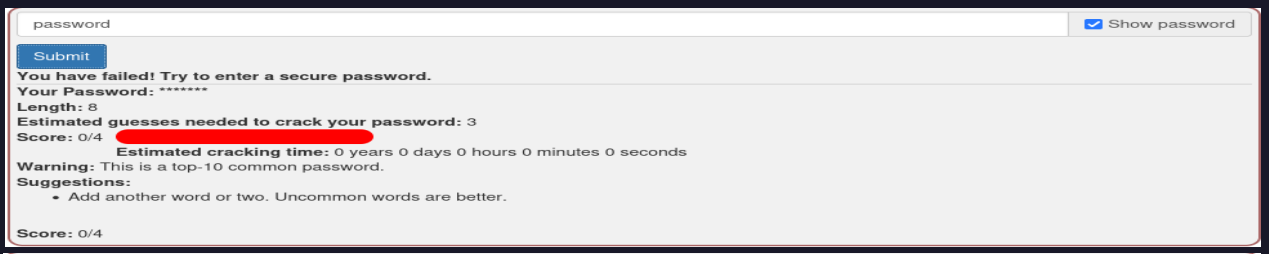


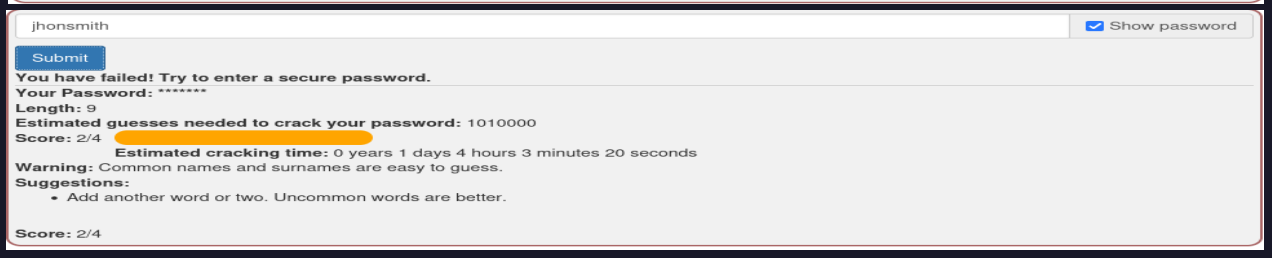
Secure Passwords

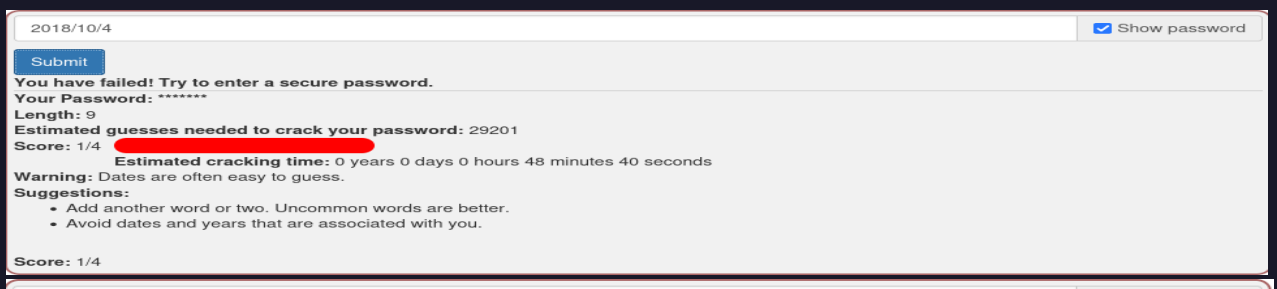
Descripción:
Se ha realizado la auditoría y análisis a las contraseñas listadas en la sección A7 - Secure Passwords y se ha realizado un informe que estará más detalladamente representado en el área de mitigaciones de las vulnerabilidades

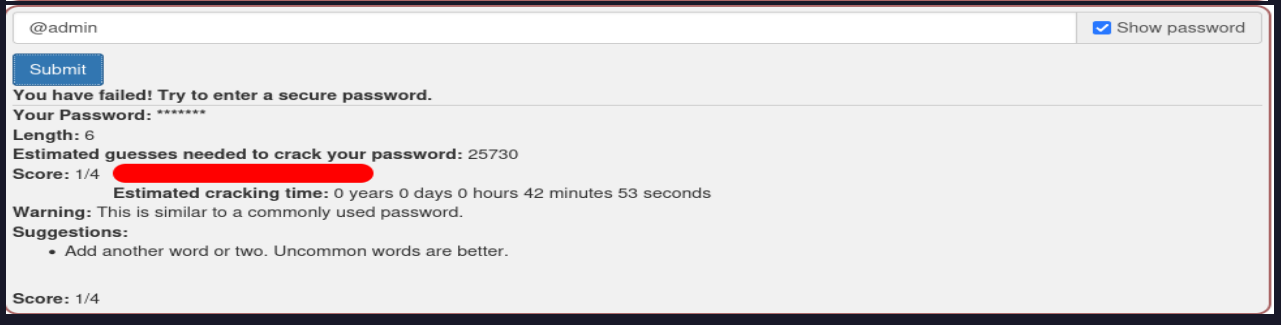
Impacto:
El uso de contraseñas débiles y fáciles de adivinar presenta riesgos significativos para la seguridad en línea. Estas contraseñas facilitan el acceso no autorizado a cuentas, ya que los atacantes pueden intentar adivinarlas o probar combinaciones comunes. Además, son vulnerables a ataques de fuerza bruta y de diccionario, que pueden ser automatizados con scripts. El riesgo se agrava si los usuarios reutilizan contraseñas en varias plataformas, ya que un atacante podría comprometer otras cuentas. Esto podría llevar a la pérdida de datos sensibles y a acciones maliciosas, incluida la suplantación de identidad y la exposición a ataques de phishing. En entornos inseguros, las contraseñas débiles son más propensas a ataques de inyección y otras técnicas de explotación. La educación sobre buenas prácticas de seguridad y la implementación de medidas como la autenticación de dos factores son esenciales para mitigar estos riesgos.

Detalles de la auditoría:









Posibles mitigaciones

Mitigación de Inyección SQL:

La Inyección SQL puede ser mitigada implementando prácticas sólidas de manejo de datos en el lado del servidor. Se recomienda:

- Validación exhaustiva de todas las entradas del usuario, incluida la sanitización y la limitación de caracteres permitidos.
- Utilización de consultas parametrizadas o procedimientos almacenados en lugar de construcciones de consultas dinámicas.
- Implementación de listas blancas para caracteres permitidos en las entradas del usuario.
- Restricción de privilegios de la cuenta de base de datos para minimizar el impacto de una posible inyección.

Mitigación de Blind SQL Injection:

La mitigación de Blind SQL Injection implica:

- Implementación de medidas para detectar y prevenir ataques de tiempo, como límites de tiempo de ejecución.
- Limitación de acceso a la base de datos mediante cuentas con privilegios mínimos, reduciendo el impacto de una posible inyección.
- Validación y sanitización exhaustiva de todas las entradas del usuario.

Mitigación de XSS:

La prevención de XSS implica estrategias tanto en el lado del servidor como en el lado del cliente. Las medidas incluyen:

- Implementación de encabezados de seguridad HTTP, como Content Security Policy (CSP), para controlar las fuentes de ejecución de scripts.
- Validación y escape riguroso de datos antes de renderizarlos en el navegador.
- Uso de frameworks y bibliotecas seguras para manipulación del DOM, como React o Angular.
- Educación y concientización del equipo de desarrollo sobre las prácticas seguras de codificación.

Mitigación de XXE:

La explotación de XXE se puede prevenir con las siguientes medidas:

- Deshabilitación de funciones XML potencialmente peligrosas, como la expansión de entidades externas.
- Validación y filtrado de entrada XML para bloquear entidades externas maliciosas.
- Uso de bibliotecas que implementen protecciones contra ataques de expansión de entidades XML.
- Configuración adecuada de procesadores XML para prevenir la inclusión de entidades externas.

Análisis de Componentes de Software:

El análisis de componentes de software se centra en mantener actualizadas las bibliotecas y componentes utilizados en la aplicación. Se recomienda:

- Actualización regular de todas las dependencias a las últimas versiones seguras.
- Monitoreo continuo de bases de datos de vulnerabilidades.
- Evaluación de la autenticidad de las versiones descargadas y adopción de firmas digitales para garantizar la integridad.

Violación de la Triada CIA:

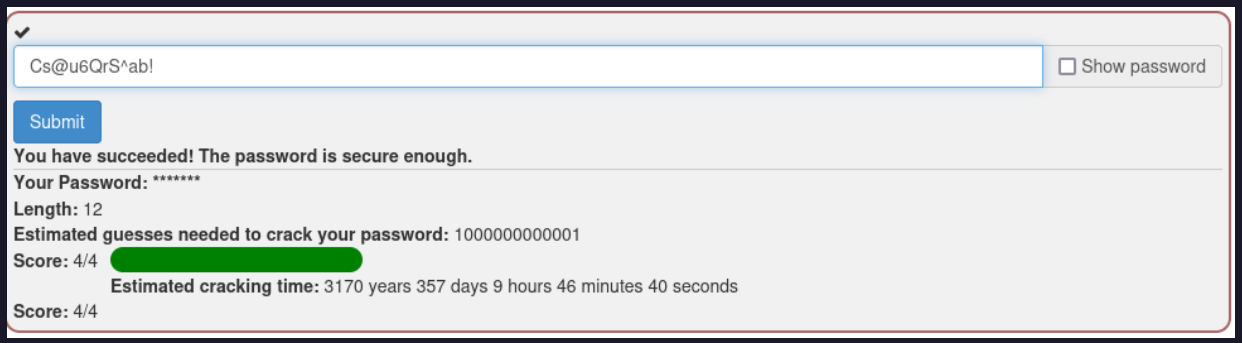
Para abordar la violación de la Triada CIA, se deben implementar las siguientes medidas:

- Establecimiento de controles de acceso rigurosos para garantizar que solo usuarios autorizados accedan a recursos sensibles.
- Auditorías regulares de seguridad para identificar y abordar posibles violaciones.
- Encriptación de datos sensibles en reposo y en tránsito mediante el uso de protocolos seguros como HTTPS.
- Implementación de mecanismos de registro y monitoreo para detectar actividades sospechosas.

Mitigación de contraseñas débiles:

Seguridad de Contraseñas:
La verificación de si has sido comprometido es esencial, se recomienda que el usuario utilice sitios web como *Have I Been Pwned* o *DEHASHED* para asegurarse de que las cuentas no hayan sido comprometidas previamente. Si encuentras alguna vulneración, cambia tus contraseñas de inmediato. Para mejorar la seguridad, emplea contraseñas diferentes para cada cuenta, considera el uso de frases de contraseña generadas por *Diceware* para mayor seguridad y memorabilidad, y gestiona tus contraseñas de manera segura mediante un administrador dedicado como por ejemplo Bitewarden. Además, implementa la autenticación de dos factores siempre que sea posible.

Almacenamiento Seguro de Contraseñas:
Cuando se trata del almacenamiento seguro de contraseñas, es crucial utilizar cifrado aprobado y un canal autenticado protegido al solicitar contraseñas. Además, asegúrate de que las contraseñas estén almacenadas de manera que sean resistentes a ataques offline. Incorporar el uso de sal antes de almacenar las contraseñas es fundamental para prevenir colisiones de valores de *sal*. Aplica funciones de derivación de clave unidireccionales aprobadas, como *PBKDF2* o *BALLOON*, antes de almacenar las contraseñas. Asegúrate de utilizar funciones de hash aprobadas, como *HMAC*, *SHA-3*, *CMAC*, *Keccak*, *cSHAKE* o *ParallelHash*. Asimismo, considera el uso de funciones de derivación de clave resistentes a la memoria para aumentar la seguridad. Ajusta el factor de costo de la función de derivación de clave según lo permita el rendimiento del servidor, pero se recomienda al menos *10,000 iteraciones*.



Este es un ejemplo de contraseña segura que consta de 12 caracteres en la que hay combinaciones de letras, números y caracteres especiales. Esta contraseña se generó con el gestor de contraseñas *Bitwarden*. Para conseguir adivinarla o romperla mediante fuerza bruta, a día de hoy le llevaría más de 3000 años a cualquier ordenador.

Herramientas utilizadas

Burp Suite:
Burp Suite es una herramienta integral de prueba de seguridad web que proporciona funcionalidades para el escaneo, la detección y la explotación de vulnerabilidades. Su interfaz intuitiva facilita la interceptación y modificación del tráfico web, permitiendo identificar y corregir fallos de seguridad, como inyecciones SQL y XSS. Durante la investigación se ha utilizado junto a la extensión de Firefox FoxyProxy para interceptar las peticiones y realizar alguna pequeña tarea de fuerza bruta mediante payloads.

Nmap:
Nmap (Network Mapper) es una herramienta de escaneo de red que se utiliza para descubrir hosts y servicios en una red, creando un mapa detallado de la topología. Proporciona capacidades avanzadas de detección de sistemas y puertos abiertos, siendo una opción valiosa para evaluar la superficie de ataque de una infraestructura.

```
┌──(kr31tos㉿V4LHAX)-[~]
└─$ sudo nmap -O 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-10 03:50 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00018s latency).
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE
8080/tcp open  http-proxy
8090/tcp open  opsmessaging
9090/tcp open  zeus-admin
Device type: general purpose
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

OS detection performed. Please report any incorrect results at https://nmap.org/
submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.63 seconds
```

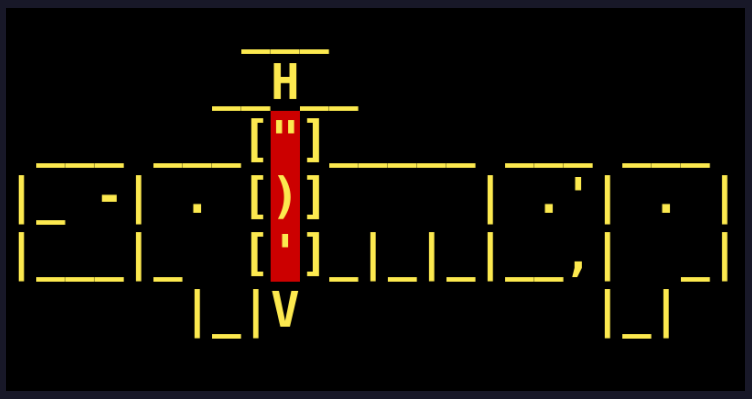
Durante la auditoría se ha utilizado nmap para comprobar qué puertos están abiertos

- *8080* que lo utilizaba la aplicación WebGoat
- *9090* que lo utilizaba la extensión de WebGoat, WebWolf
- *8090* que lo usaba BurpSuite a través de FoxyProxy

También se listo el sistema operativo utilizado, que es *Linux versión 2.6.32*

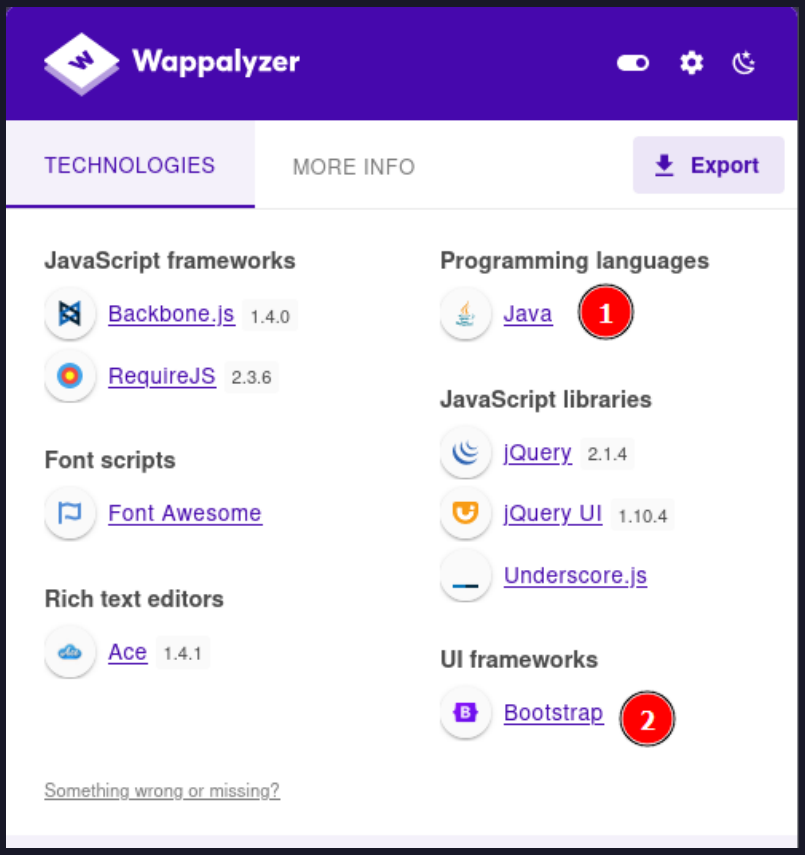
SQLmap:

SQLmap es una herramienta de prueba de penetración especializada en la detección y explotación de vulnerabilidades de inyección de SQL en aplicaciones web. Su capacidad para automatizar la identificación de fallos en bases de datos y extraer información confidencial lo convierte en una herramienta esencial para evaluar la seguridad de sistemas que interactúan con bases de datos SQL. Se ha utilizado durante la detección de vulnerabilidades Blind SQL Injection.



Wappalyzer:

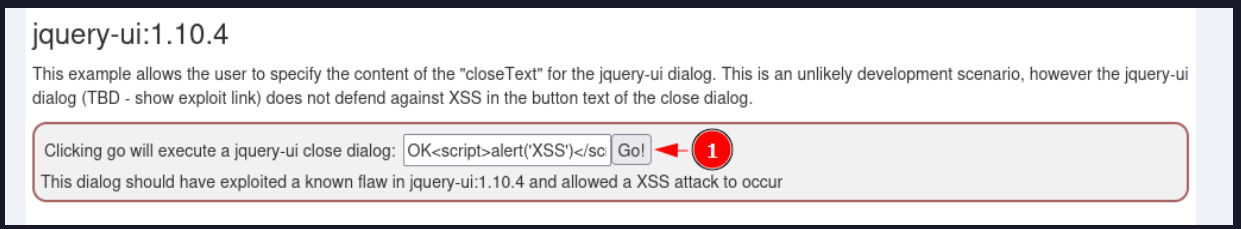
Wappalyzer es una herramienta de código abierto que permite a los usuarios descubrir las tecnologías utilizadas en sitios web específicos. Proporciona información detallada sobre las tecnologías empleadas, como el sistema de gestión de contenidos (CMS), frameworks, bibliotecas JavaScript, servicios de análisis y más. Wappalyzer se integra como una extensión en navegadores web, como Chrome y Firefox, y muestra íconos discretos en la barra de direcciones para identificar las tecnologías presentes en el sitio web visitado. Esta herramienta es valiosa para analizar la infraestructura tecnológica de un sitio web durante evaluaciones de seguridad o investigaciones.



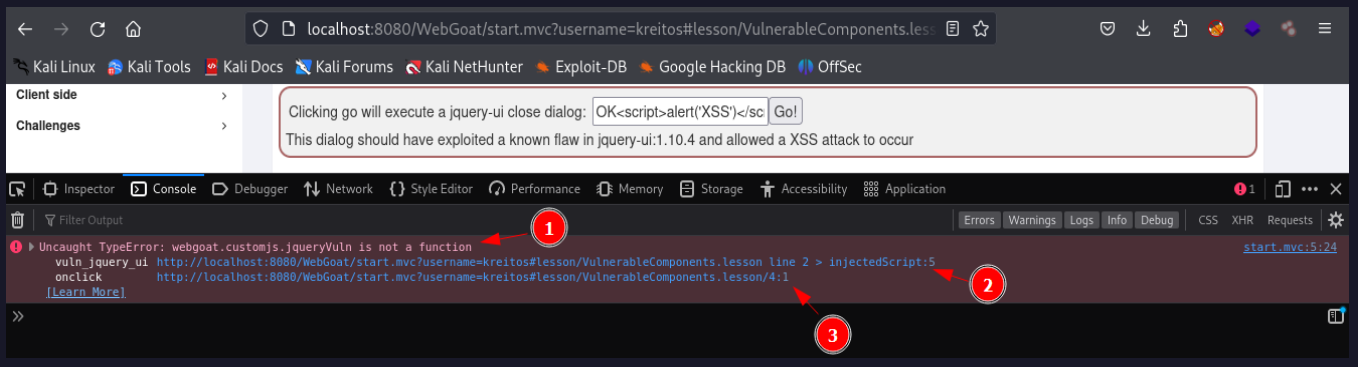
Se ha utilizado durante la auditoría para determinar qué tipo de lenguaje de programación usa la Web, que frameworks usa JavaScript, cuales son las librerías y los frameworks de diseño. Toda esta información es útil para poder encontrar vulnerabilidades en versiones desactualizadas o realizar ataques más efectivos.

Anexo

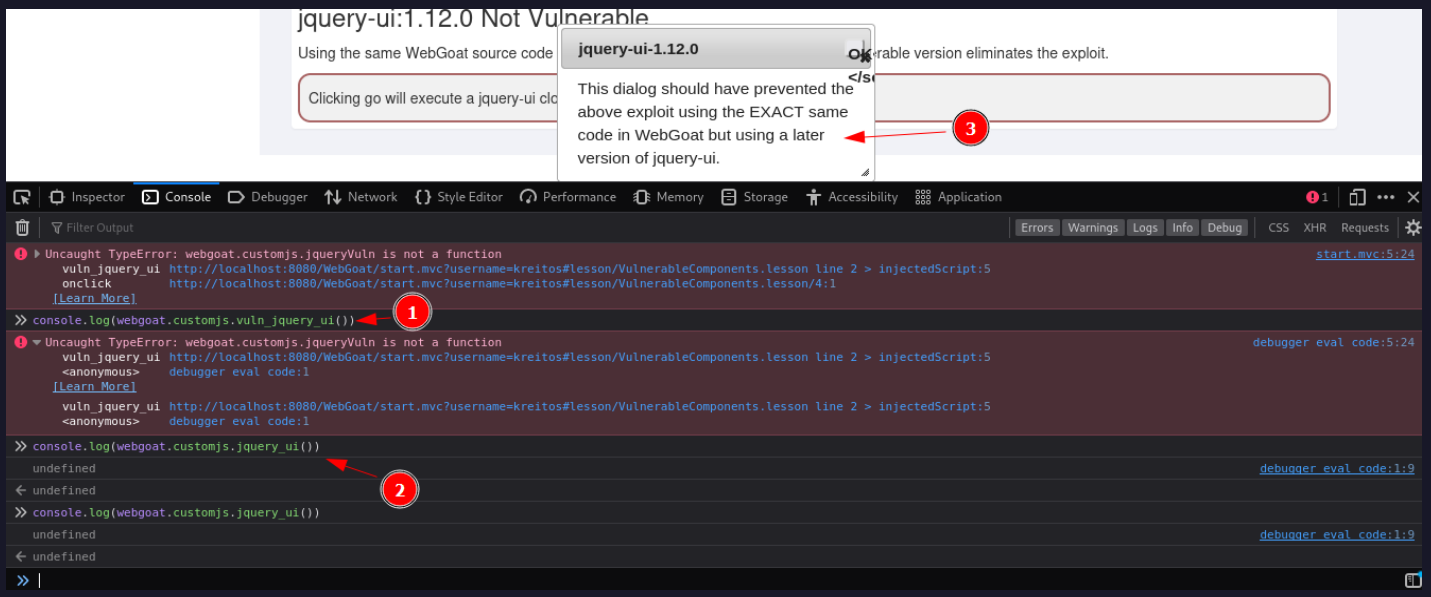
En este apartado me gustaría dejar reflejada mi pequeña investigación del apartado de control de versiones.



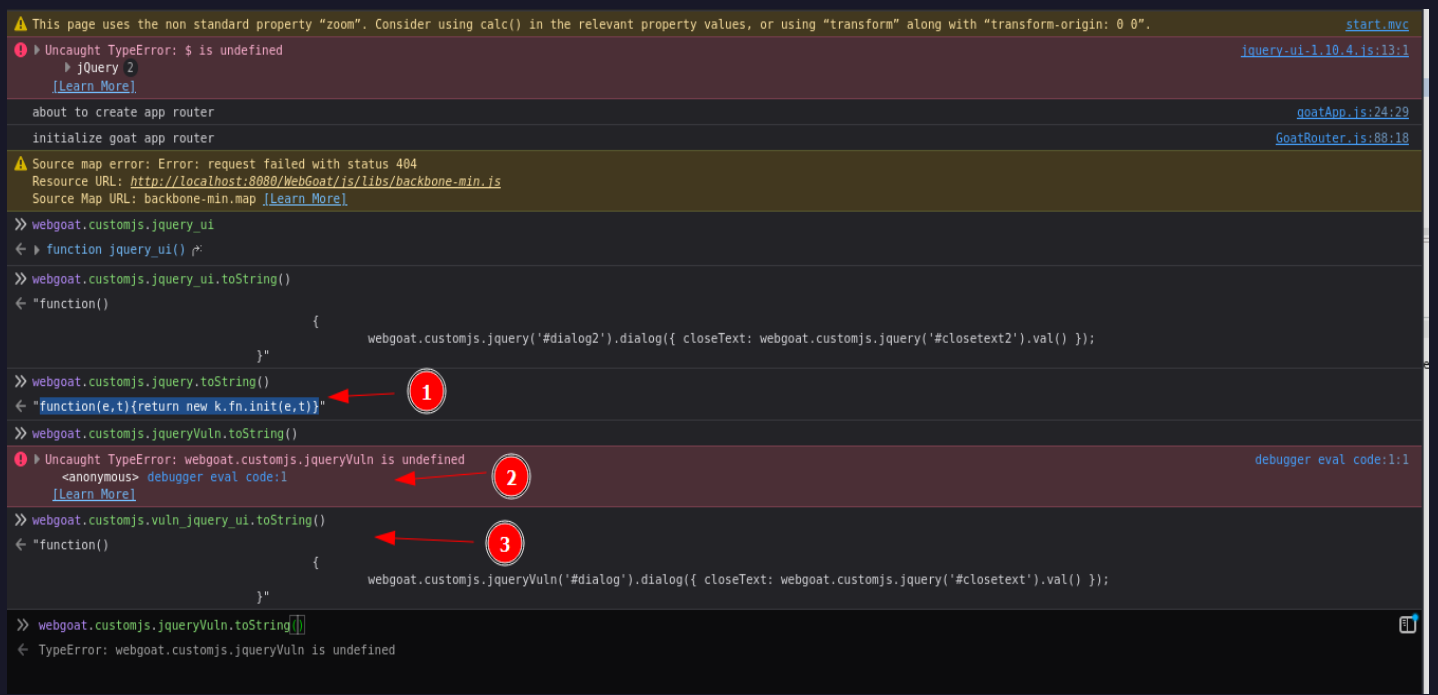
1. Al dar click al botón *Go!*, salen los siguientes errores por consola



1. La función a la que llama no está definida o no es una función así que decido investigar a ver qué ha pasado. Reviso los dos enlaces
2. Me lleva al código fuente del directorio donde se debería alojar la función y descubro que se llama a una función que se llama *webgoat.customjs.vuln_jquery_ui()* la cual no está definida.

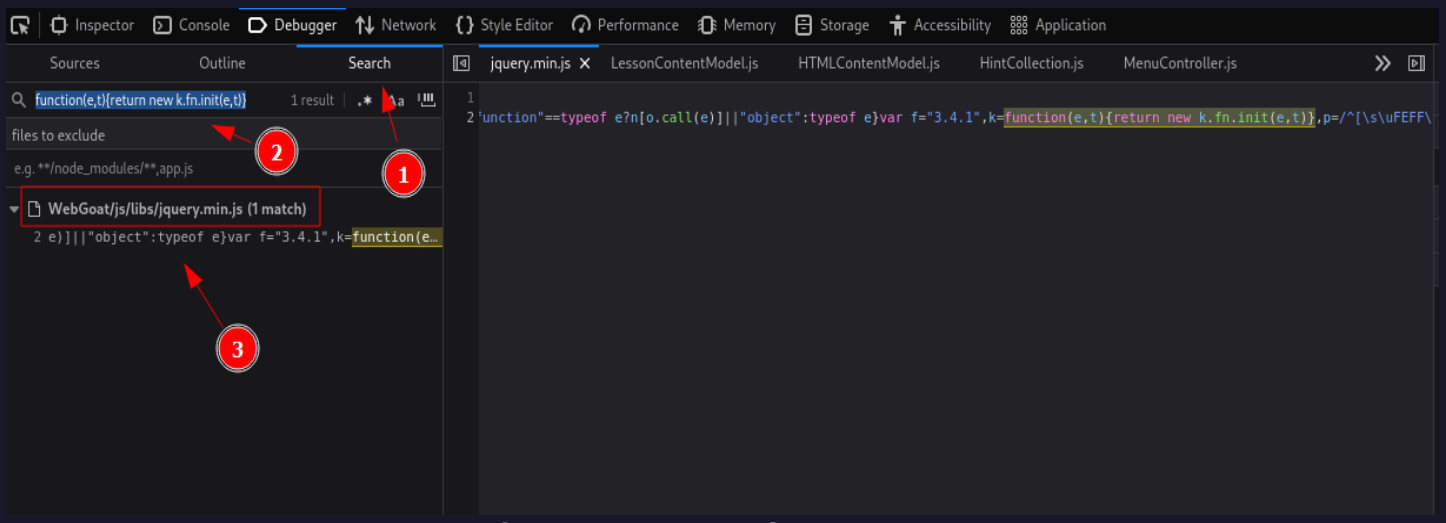


1. Intento imprimir por consola la función con la esperanza de conseguir alguna definición. Sin éxito. Me vuelve a decir que no es una función.
2. Acto seguido intentó imprimir por consola la función del segundo apartado, que si me saca el alert con el diálogo de la versión no vulnerable.
3. Diálogo que sale al llamar a la función *webgoat.customjs.jquery_iu()*



1. Sigo haciendo más pruebas para comprobar a qué llama cada función. La intentó pasar a la cadena, para ver si me da alguna pista de que contiene. Bingo!
2. La función que no está definida da error

3. El diálogo llama a la función *jqueryVuln()* que realmente no está definida en la web y por eso no saca nada por pantalla. Realmente no llama a nada.



1. Dentro del Debugger de Firefox en el apartado Search:
2. Intento encontrar alguna carpeta que tenga alguna coincidencia con la función que sí está definida, y me encuentra un match
3. Está dentro de la la carpeta *WebGoat/js/libs/jquery.min.js,* se trata de un archivo muy extenso que llama a la librería *jQuery v3.4.1*

Conclusión

El departamento de IT tiene que investigar qué ha pasado con la función, y por qué no se llama correctamente. Comprobar si ha habido alguna actualización de la librería y han variado los nombres o la ruta de las funciones. Al parecer en el ejercicio están usando las versiones de las librerías *jquery-ui:1.10.4*, *jquery-ui:1.12.0 No vulnerable*, mientras que en el archivo la versión es *jQuery v3.4.1.* Puede que tenga alguna relevancia para solventar el problema.

“Puede que sea una frikada sin sentido, pero me he entretenido investigando y buscando fallas”