Resultados de PortSwigger Web Security Academy

Reporte Número 1

Vicente Vieytes

30/09/22

Contenidos

1. SQL Injection **2**
1.1. Introducción a SQLI 2
1.1.1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data 2
1.1.2. SQL injection vulnerability allowing login bypass 3
1.2. SQLI UNION Attacks 4
1.2.1. Determining the number of columns returned by the query 5
1.2.2. Finding a column containing text 6
1.2.3. Retrieving data from other tables 9
1.2.4. Retrieving multiple values within a single column 11

Introducción

Este documento es un reporte de mis avances en las soluciones de los laboratorios de *PortSwigger Web Security Academy*.

Este reporte es constituir un solucionario para las actividades y una guía paso a paso de explotación y vulnerado de las aplicaciones web asociadas a estas actividades además de un análisis de las vulnerabilidades encontradas.

Este reporte incluye las soluciones a los primeros laboratorios sobre SQL Inyection, en específico los ataques que utilizan la clausula `UNION` del lenguaje SQL.

1 SQL Injection

1.1. Introducción a SQLI

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

La página web asociada a este laboratorio simula una tienda online de distintos productos. La página contiene una vulnerabilidad SQLI en el filtro de categoría de los productos. Cuando se selecciona la categoría de producto 'Gifts', la página nos redirecciona hacia `/filter?category=Gifts`. Los productos que aparecen luego del filtrado se pueden ver en la figura 1.1. Nos indican que cuando se envía la request a este url, el query acarreado en el servidor es algo de la forma:

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

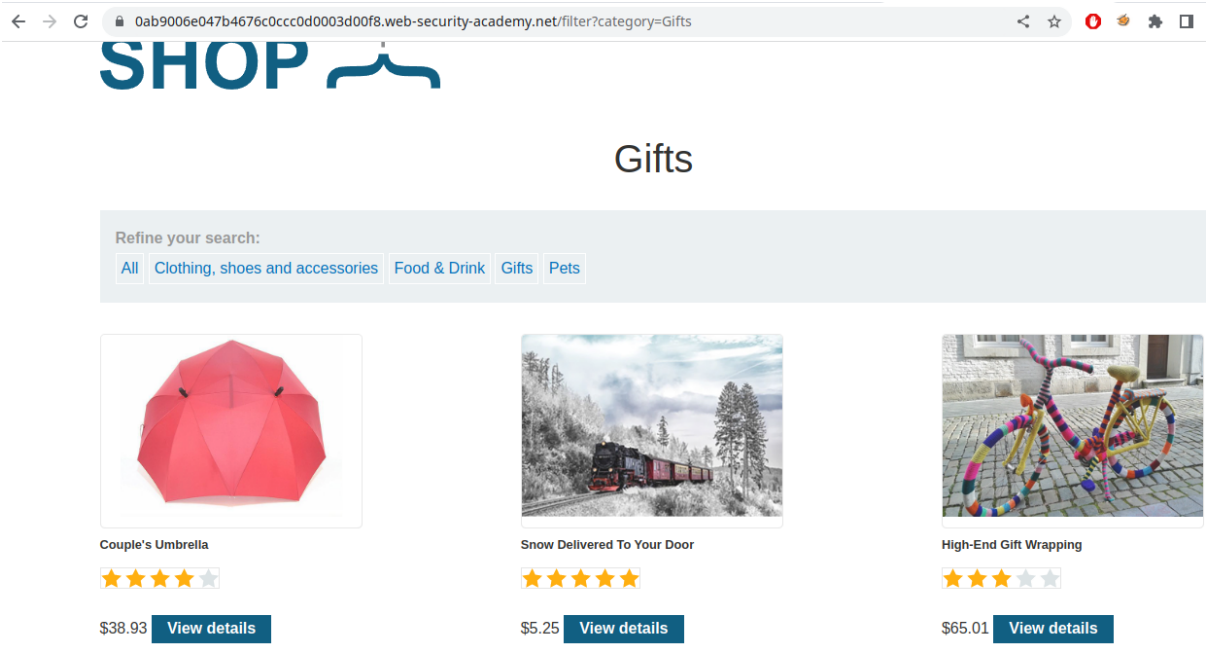

Figura 1.1: Resultados de filtrar por la categoría 'Gifts'.

El objetivo del laboratorio es realizar una inyección SQL que cause que la aplicación nos muestre detalles de todos los productos de cualquier categoría, incluso productos no publicados.

Si modificamos el valor del parametro `category` de `Gifts` a `'+OR+1=1+-` el query termina teniendo la siguiente forma:

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

Lo que llevaría a que el servidor nos devuleva todos los productos ya que siempre es verdadero que 1=1, todo lo que queda a la derecha de los dos guiones altos es considerado un comentario y no es interpretado.

Realizando esta inyección, la página nos muestra todos los productos y conseguimos superar el laboratorio, ver figura 1.2.

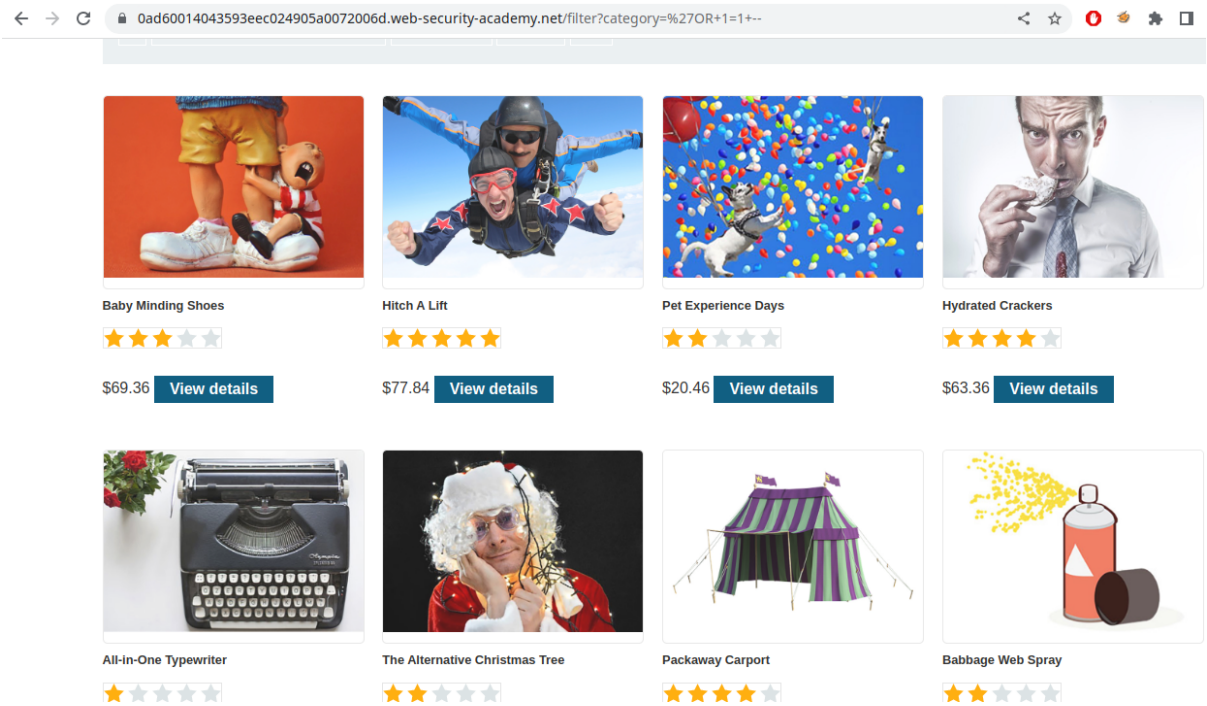

Figura 1.2: Luego del ataque vemos todos los productos de la base de datos.

1.1.2. SQL injection vulnerability allowing login bypass

Este laboratorio contiene una página de login que es vulnerable a SQLI de manera tal que se puede ingresar como administrador sin tener las credenciales correctas. Muchas veces para autenticar un usuario las aplicaciones web buscan en la base de datos algún usuario con el que coincidan las credenciales con un query de la forma:

```
1 SELECT * FROM users WHERE username = 'vicente' AND password = '1234'
```

Si en lugar de ingresar en el campo de usuario `vicente` ingresaramos `vicente'--` y no ingresamos la contraseña, la query quedaría de la siguiente forma:

```
1 SELECT * FROM users WHERE username = 'vicente'--' AND password = ''
```

Y todo lo que queda del otro lado de los dos guiones es considerado un comentario, luego la base de datos retorna el usuario.

Ingresando `administrator'--` en el campo de usuario y caracteres aleatorios en la contraseña se consiguió iniciar como administrador y bypassear la autenticación.

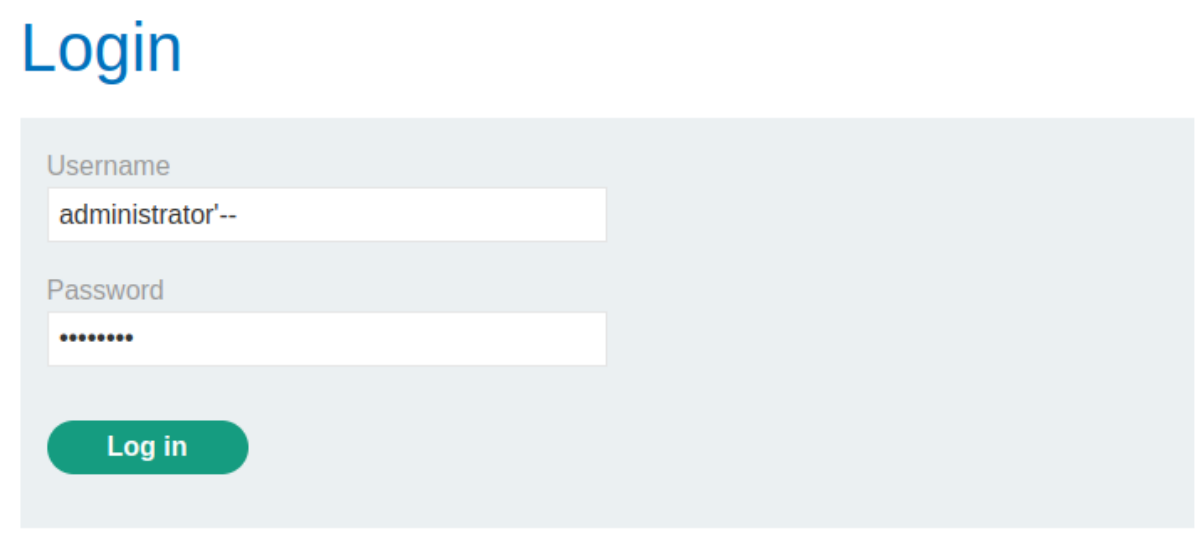

Figura 1.3: Pantalla de login.

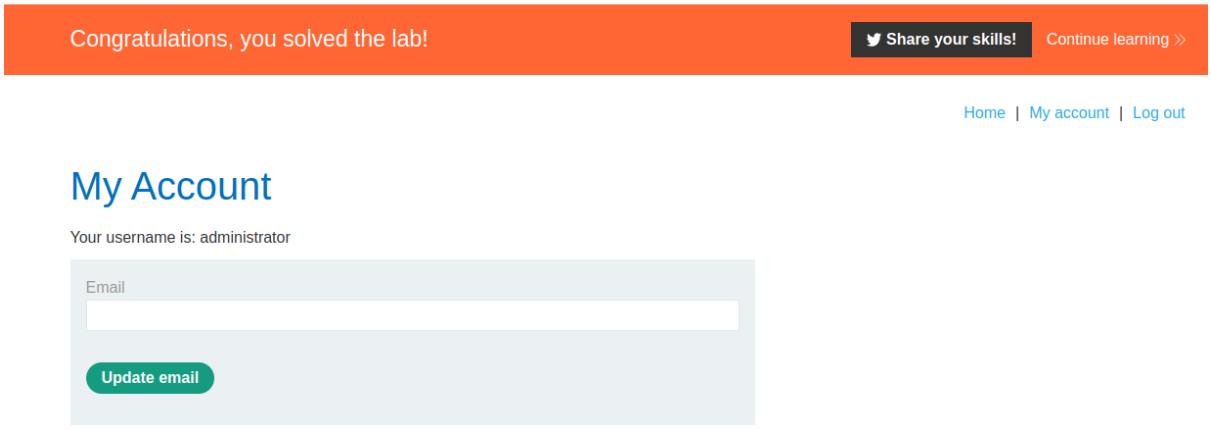

Figura 1.4: Acceso obtenido luego del ataque.

1.2. SQLI UNION Attacks

El operador `UNION` de SQL se usa para combinar el resultado de dos o más statements de `SELECT`, para que esto funcione cada uno de los tatements de `SELECT` debe devolver la misma cantidad de columnas y las columans deben tener tipos de datos similares.

Un ataque SQLI `UNION` utiliza esto para extraer datos de una tabla distinta a la que se accede en el entry point de la inyección. Para poder realizarlo hace también falta determinar la cantidad de columnas que devuelve el `SELECT` de nuestra inyección.

1.2.1. Determining the number of columns returned by the query

El objetivo de este laboratorio es determinar esta cantidad de columnas. Para esto hay dos métodos comunes:

- Inyectar clausulas `ORDER BY` con distintos valores de manera que la página devuelva un error cuando nos pasemos de la cantidad de columnas del `SELECT`.
- Inyectar statements del tipo `UNION SELECT NULL NULL-`, si la cantidad de columnas `NULL` no coincide con la cantidad de columnas necesaria entonces la página nos devolverá un error, si la cantidad de columnas coincide entonces la base de datos devuelve una tabla con una fila extra con valores `NULL` en todos los campos.

Se realizó este tipo de ataque para determinar la cantidad de columnas devueltas por la base de datos cuando se accede a la página web del laboratorio, en la figura 1.5 se puede ver el listado de items inicial, en la figura 1.6 se puede ver el error devuelto cuando hacemos `UNION` con una sola columna `NULL`, y en la figura 1.7 se puede ver que con tres columnas `NULL` no se levanta ningún error. Esto significa que la base de datos devuelve tres columnas de la tabla.

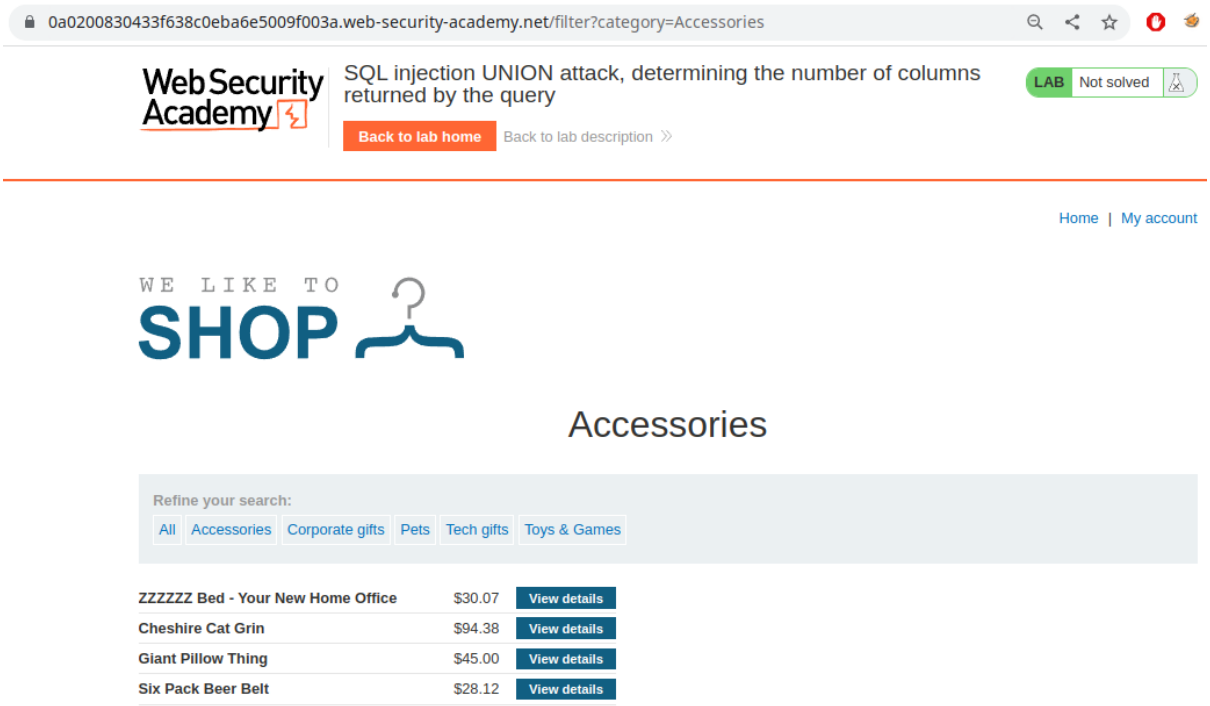

Figura 1.5: Acceso inicial a la página del laboratorio

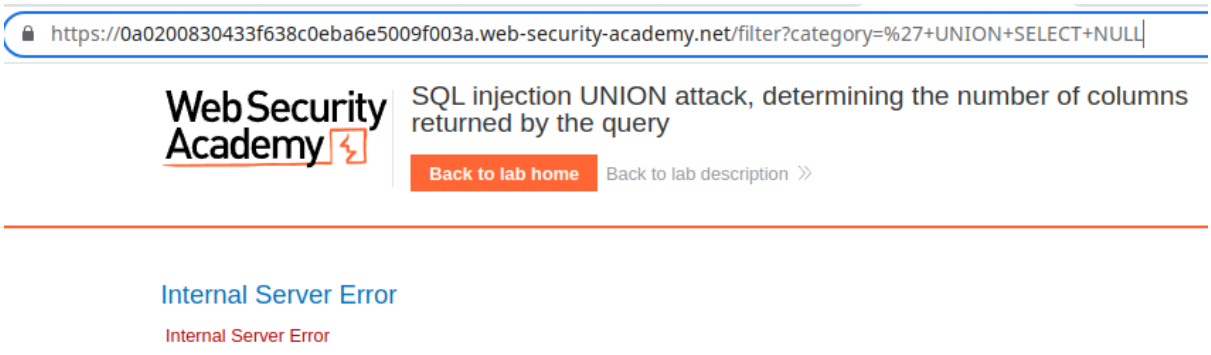

Figura 1.6: Error obtenido luego de hacer UNION con una sola columna NULL

Figura 1.7: El servidor no devuelve error con tres columnas NULL

1.2.2. Finding a column containing text

En general en un ataque `UNION` lo que se busca es extraer datos de otras tablas, los datos interesantes en general están en texto en forma de string, para que el `UNION` funcione correctamente los tipos de datos de los dos `SELECT` tienen que ser compatibles en todas las columnas por lo que es importante determinar qué columnas de las devueltas contienen strings.

Este es el objetivo de este laboratorio y para lograrlo se hizo algo similar al ataque del laboratorio anterior. Inyectando statements de la forma `UNION SELECT NULL 'a' NULL NULL` y cambiando de posición el string entre las columnas, si no ocurre un error significa que la columna es compatible con texto.

Se enviaron requests con este tipo de payloads sabiendo que la tabla devolvía 3 columnas, ingresando un string como primer columna en el `UNION` el servidor devuelve un error, pero en la segunda posición no. Esto nos indica que la segunda columnas devuelta es compatible con strings.

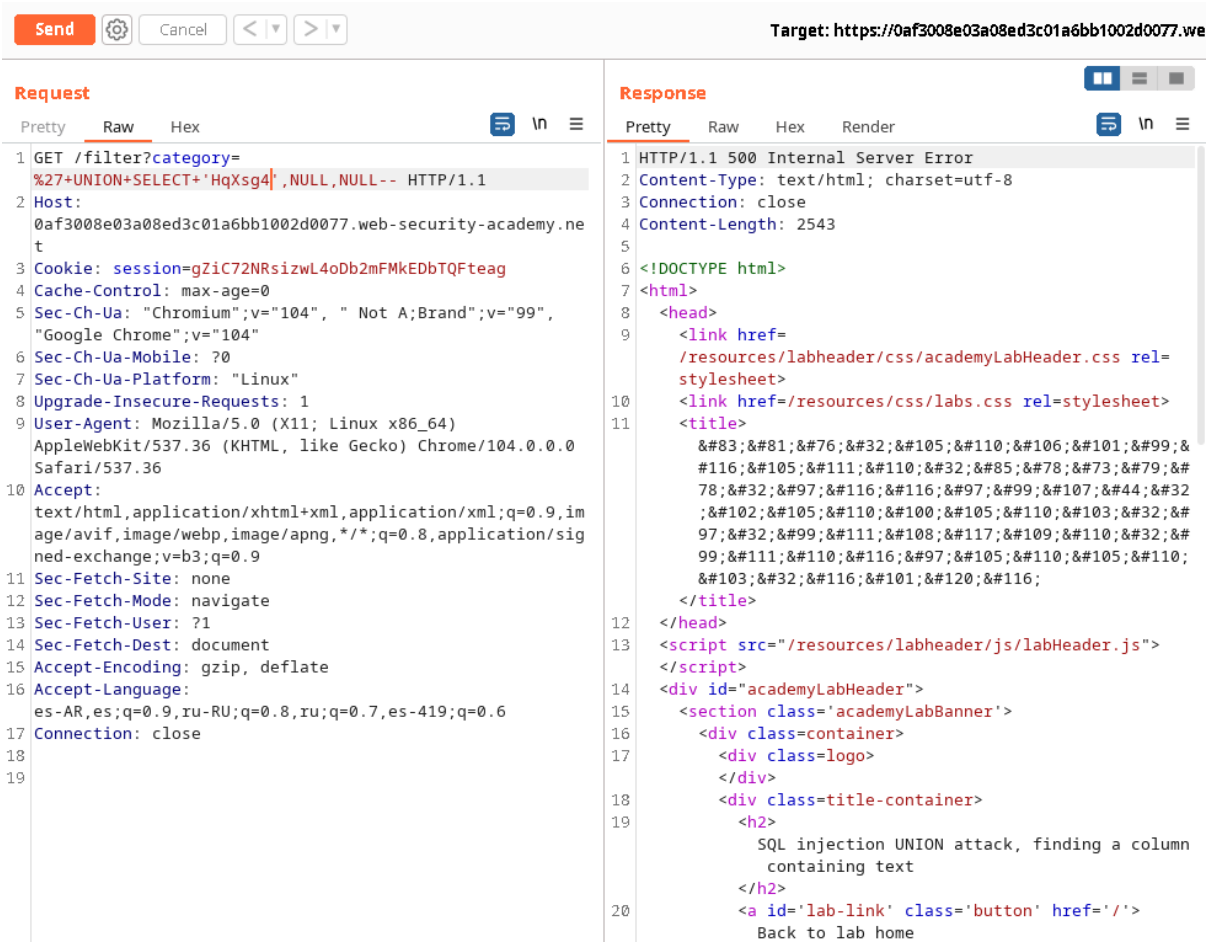

Figura 1.8: Request enviada con el string en la primer posición devuelve error.

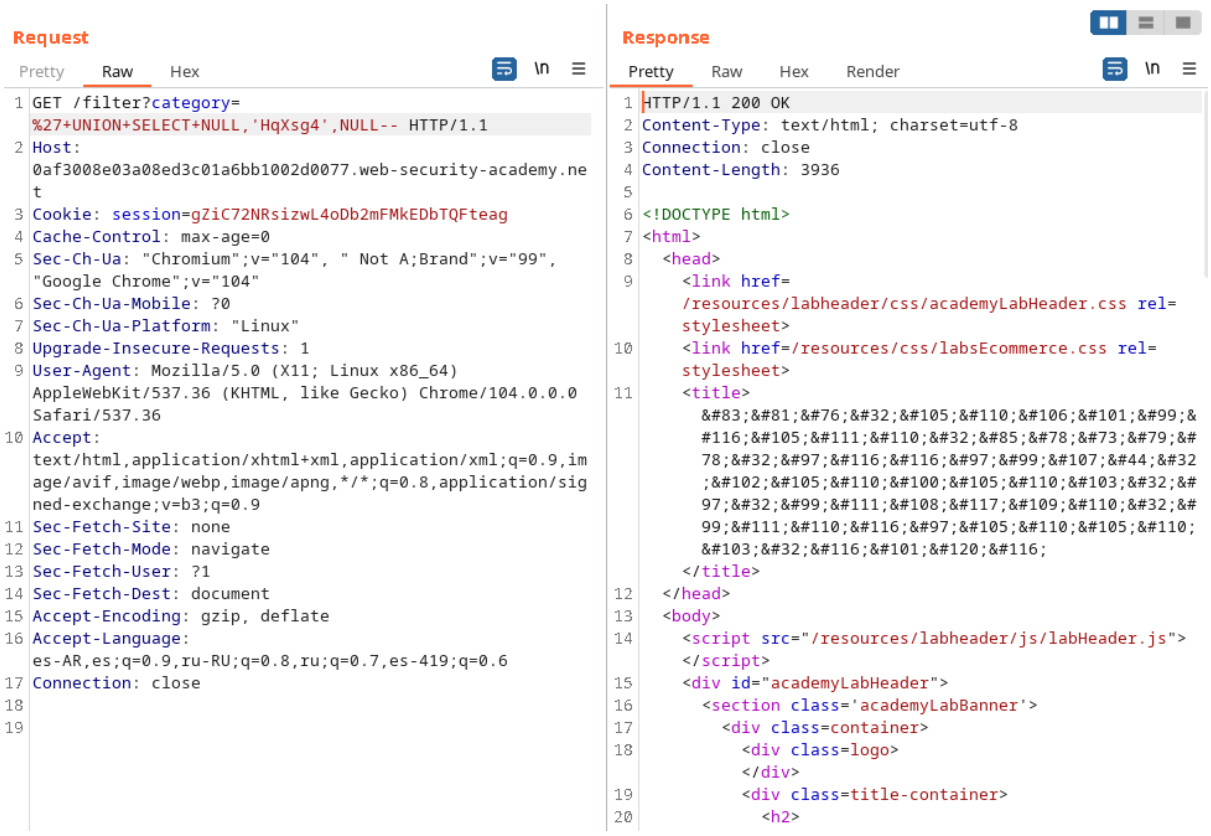

Figura 1.9: Request enviada con el string en la segunda posición no devuelve error.

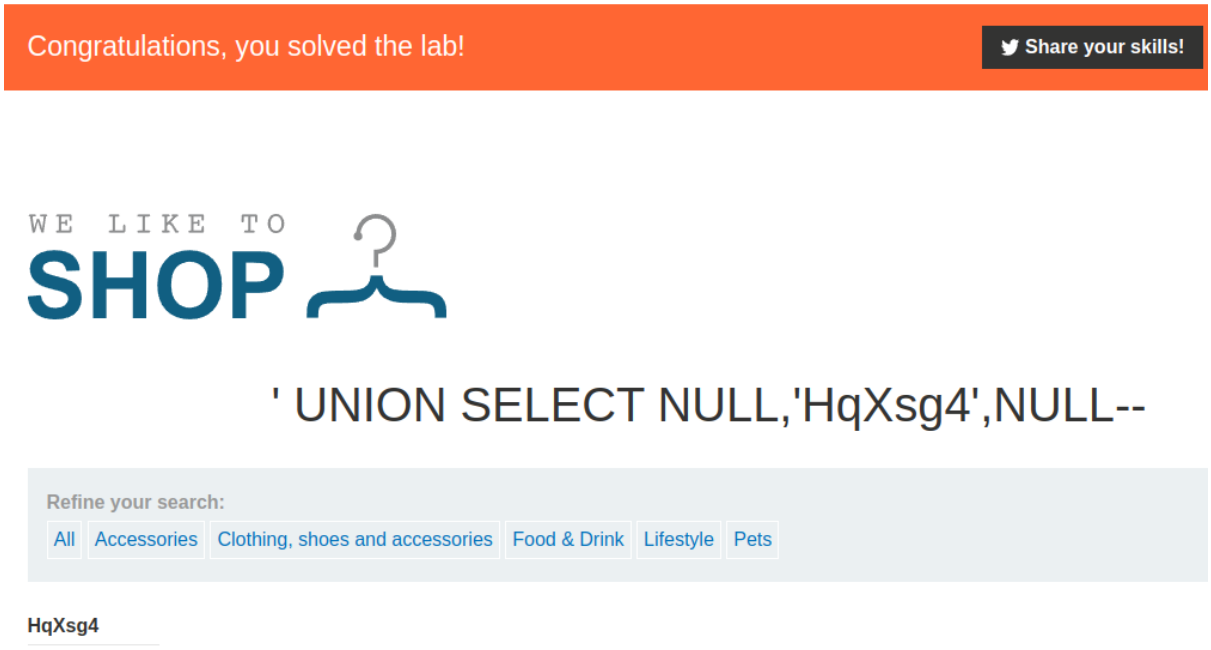

Figura 1.10: HTML renderizado de la respuesta al ataque, el string aparece reflejado en el HTML.

1.2.3. Retrieving data from other tables

Habiendo determinado el número de columnas devueltas por el query original y habiendo encontrado qué columnas pueden contener datos en forma de string estamos listos para intentar extraer data interesante de otras tablas.

Utilizando las mismas técnicas de los últimos dos laboratorios se determinó que el query original devuelve dos columnas y que las dos admiten strings, ver figura 1.11.

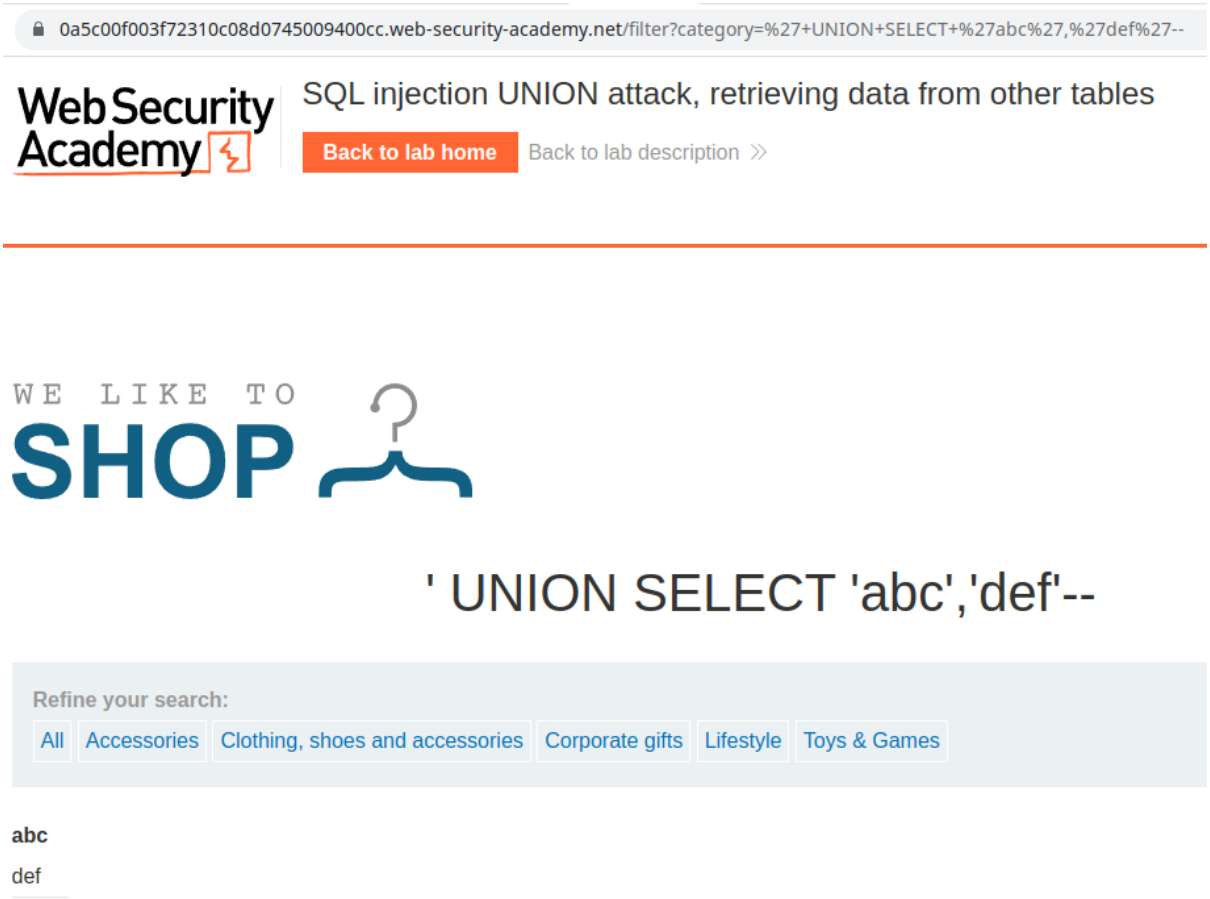

Figura 1.11: Resultado de inyectar el payload `'+UNION+SELECT+'abc','def'--`.

Tenemos información de que la base de datos contiene una tabla llamada 'users' que contiene columnas llamadas 'username' y 'password'. Se inyectó el payload `'+UNION+SELECT+username,+password+FROM+users--` el cual agregó todo el contenido de la tabla users. Ver figura 1.12

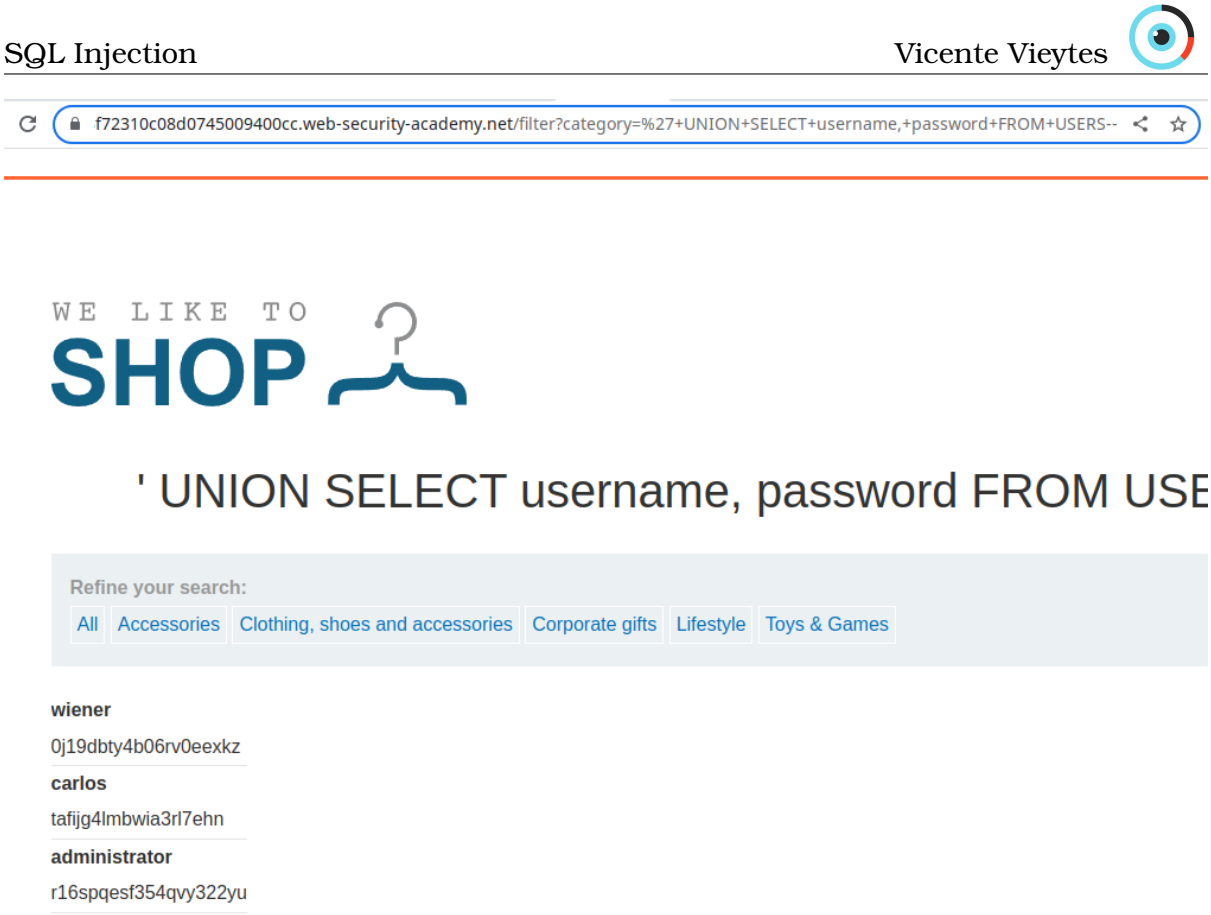

Figura 1.12: Resultado de inyectar el payload `'+UNION+SELECT+username,+password+FROM+users--`.

1.2.4. Retrieving multiple values within a single column

En el caso anterior la base de datos de la que se quería extraer datos tenía dos columnas y se unió al resultado del otro query original que también tenía dos columnas. Si en cambio el query original devolviera una única columna se deberían concatenar los valores de los strings juntos para poder extraer todo en una misma columna. En oracle por ejemplo se utiliza el operador `||` pero esto varía dependiendo de la sintaxis de la base de datos que se utilice en el servidor.

Al igual que antes sabemos que la base de datos tiene una tabla 'users' con los campos 'username' y 'password' y queremos extraer las credenciales.

Se realizó una inyección para determinar la cantidad de columnas devueltas por el query original y para determinar qué columnas admiten strings. Se encontró que estas son dos columnas y que solo la segunda admite strings, ver figura 1.13, por lo que necesitamos incluir tanto 'username' como 'password' en la segunda columna.

Se utilizó el payload `'+UNION+SELECT+NULL,username||' '||password+FROM+users--` para incluir cada usuario y contraseña separados por el caracter ' '. El resultado se puede ver en la figura 1.14.

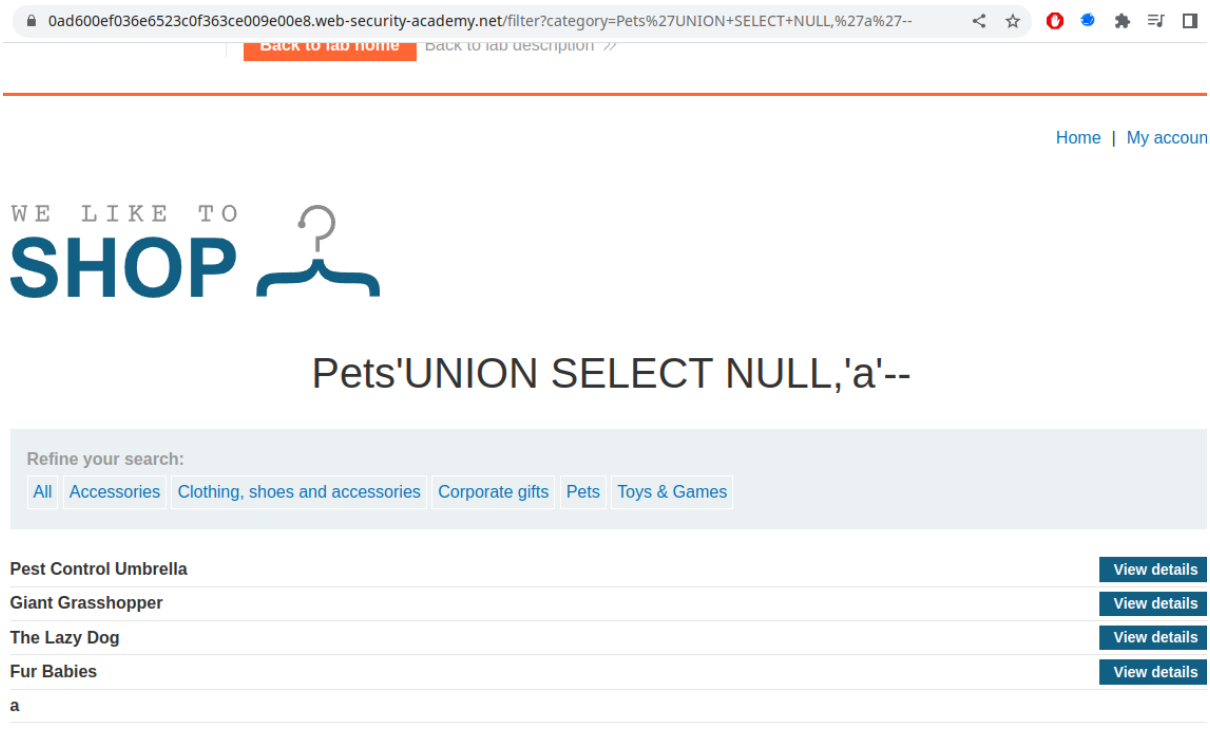

Figura 1.13: Resultado de inyectar el payload `'+UNION+SELECT+NULL,'a'--`. La inyección con el string en la primer posición devuelve error.

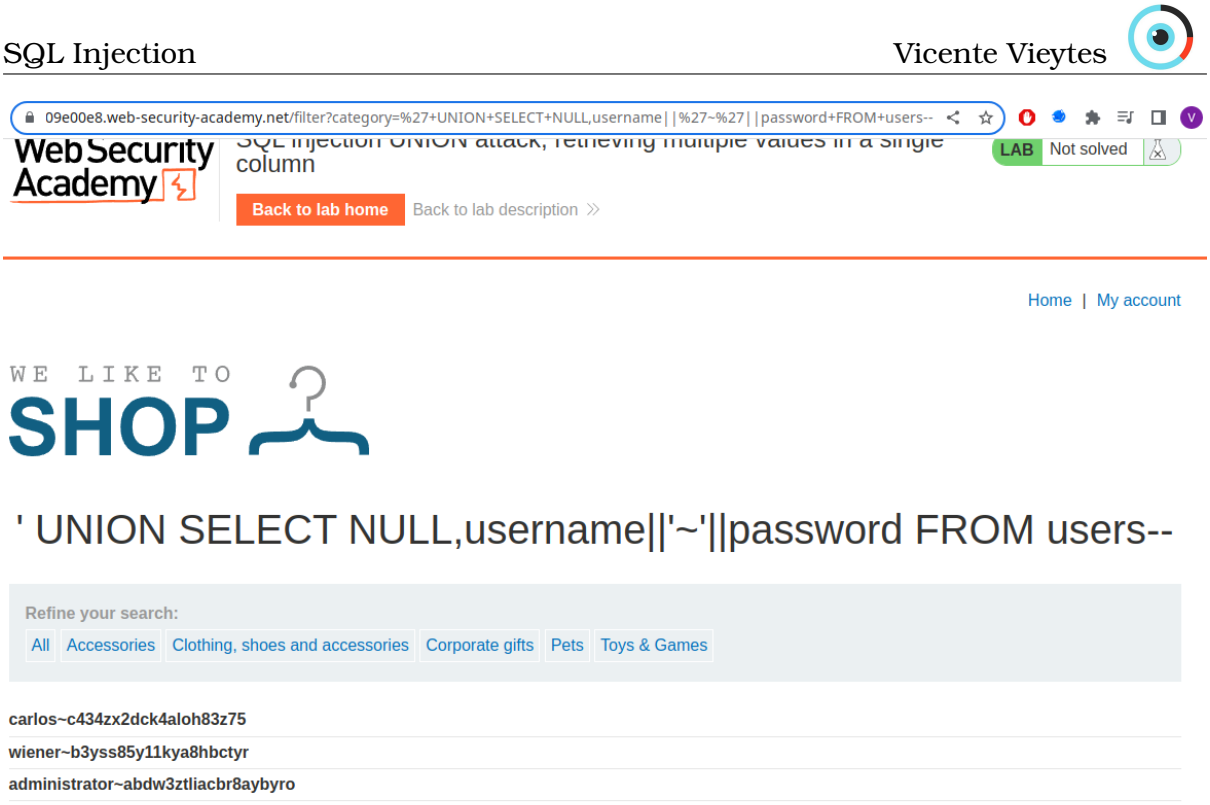

Figura 1.14: Resultado de inyectar el payload `'+UNION+SELECT+NULL,username||' '||password+FROM+users--`