Om Khadka, U51801771
Sources: N/A
Collabs: N/A

Challenge 5

I exploited the weaknesses in the salted SHA256 password implementation, where the salt was too small and the password range could be easily brute-forced.

ID'ing the problem

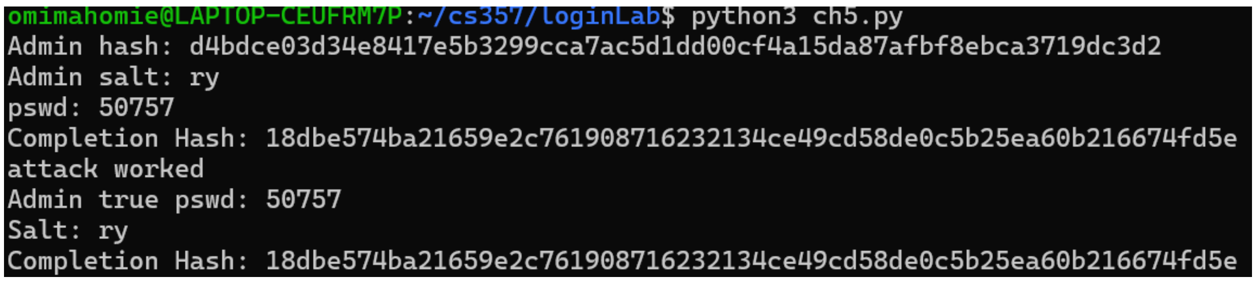
There are quite some problems with this website:

- SQL injectable in the /search endpoint
- Only 2 lowercase letters as salt (676 possibilities)
- 5-digit number pswds only (100,000 possibilities)
- Salts are stored **alongside the** hashes in the database

Attack

Everything was run under a script. The script does the following:

1. Extracted both the **admin_password_hash** and **admin_password_salt** from /search endpoint by exploiting the /search endpoint's SQL injectability
    a. " UNION SELECT password_hash || ':' || password_salt FROM members WHERE username='admin'--
2. Then, runs a brute-force attack, checking each combination of the output of the following
    a. SHA256(**admin_password_salt**, 00000-99999) → **output**
    b. Check if **output** = **admin_password_hash**
        i. If so, then that iteration's number is the password
        ii. Otherwise, keep on iterating with other numbers.
3. Login as admin with the resulting brute-force results.
    username : admin
    name : om khadka
    password : **result from 2.**



```
omimahomie@LAPTOP-CEUFRM7P:~/cs357/loginLab$ python3 ch5.py
Admin hash: d4bdce03d34e8417e5b3299cca7ac5d1dd00cf4a15da87afbf8ebca3719dc3d2
Admin salt: ry
pswd: 50757
Completion Hash: 18dbe574ba21659e2c761908716232134ce49cd58de0c5b25ea60b216674fd5e
attack worked
Admin true pswd: 50757
Salt: ry
Completion Hash: 18dbe574ba21659e2c761908716232134ce49cd58de0c5b25ea60b216674fd5e
```

Hash: 18dbe574ba21659e2c761908716232134ce49cd58de0c5b25ea60b216674fd5e