William Hupp
CTF Web II Assignment

# Web II Write-Up

# Profile II

50 points

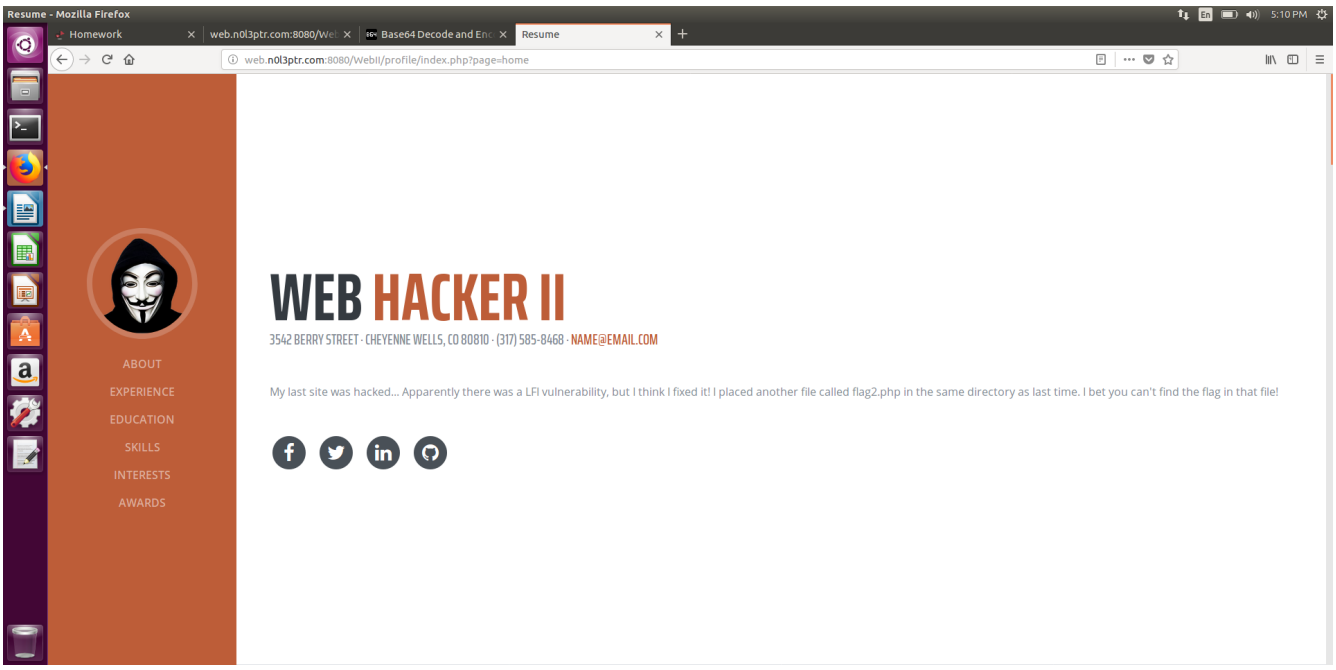
## Problem:

The infamous Web Hacker is back. He/she updated their personal website to patch the LFI bug from the Web I challenge. Now they have hidden a flag in a file called flag2.php in the same directory as before (the web application root). Can you find the flag?

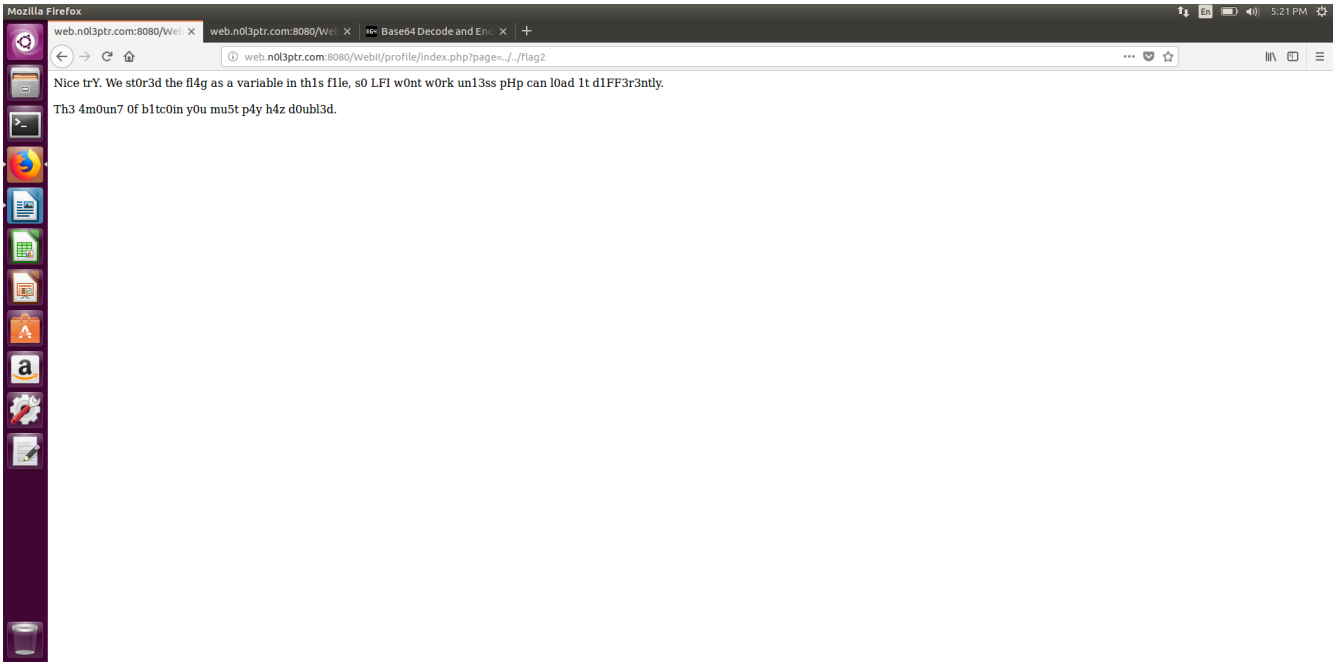
http://web.n0l3ptr.com:8080/WebII/profile/index.php?page=home

## Solution:

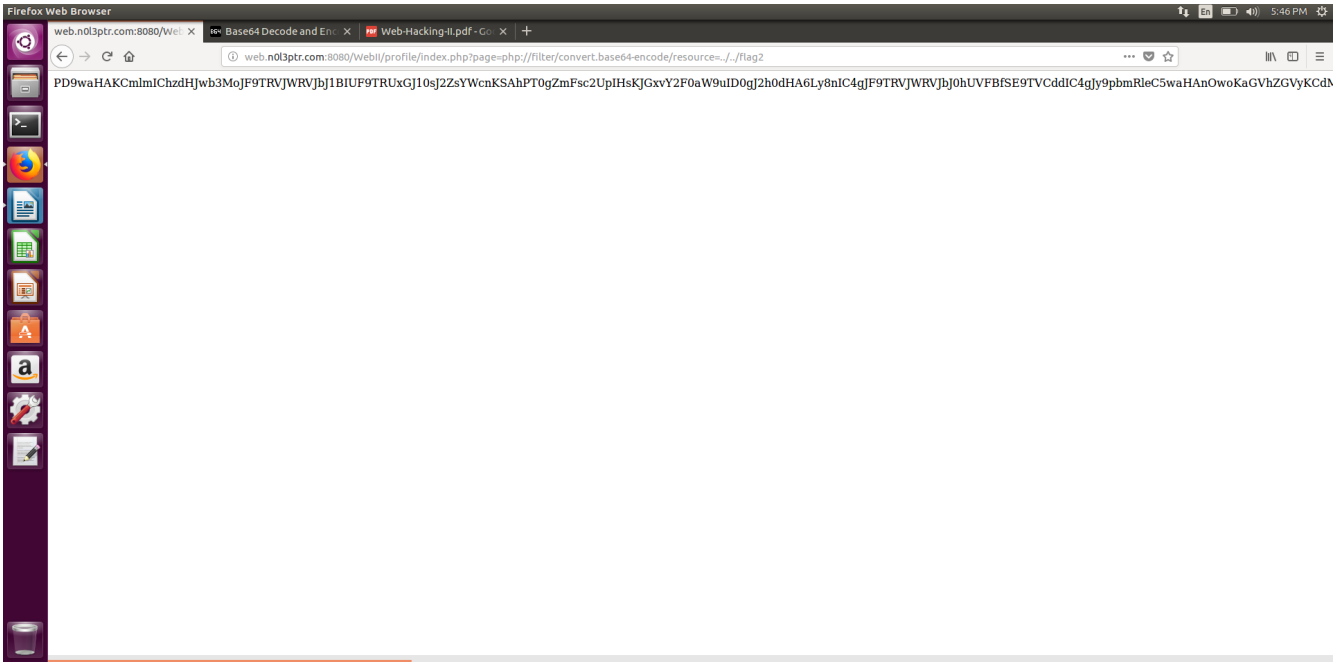
The web page is as follows.



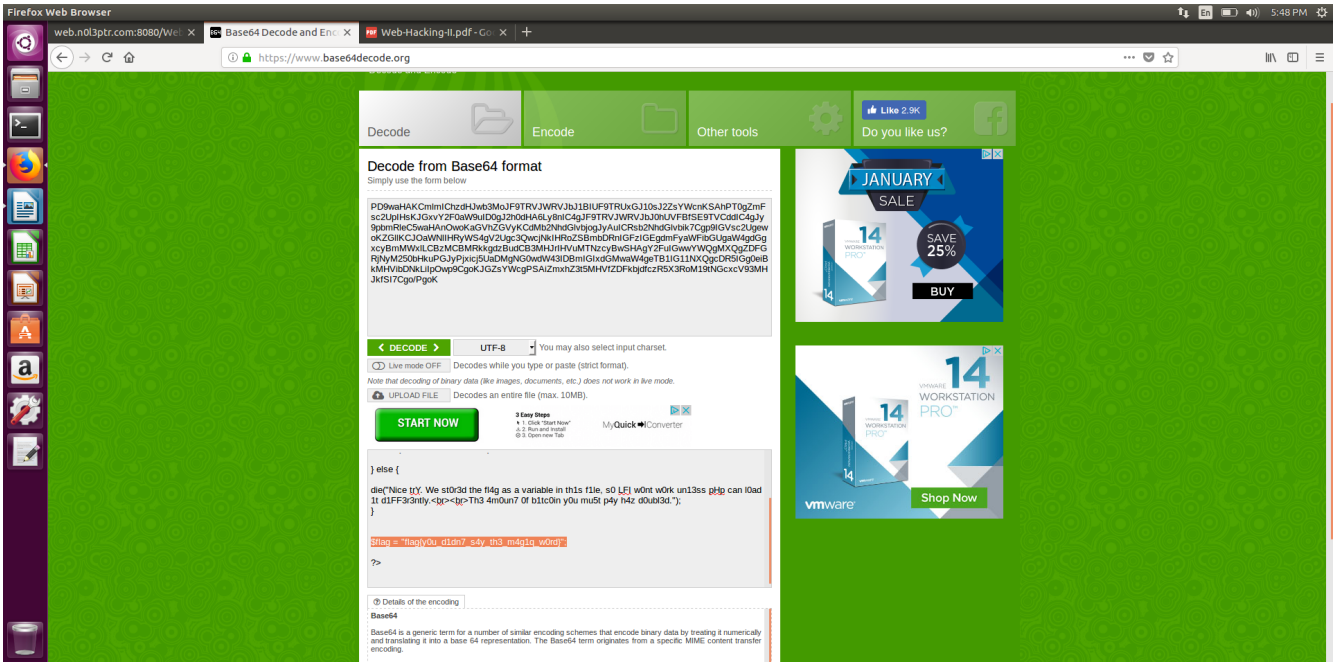

It contains URL: http://web.n0l3ptr.com:8080/WebII/profile/index.php?page=home
Since the flag is located in the same directory as before, I tried my previous solution from Web I. I utilized the LFI bug and entered http://web.n0l3ptr.com:8080/WebII/profile/index.php?page=../../flag2 which is the same solution as the previous LFI problem but with with ‘flag’ replaced with ‘flag2’. This is because the previous file was called flag.php and the new file is called ‘flag2.php’. This revealed the following:




The important message is: “Nice try. We stored the flag as a variable in this file, so LFI won’t work unless PHP can load it differently”. The message implies we need to make the PHP load the flag2.php file differently to get the flag. According to the Web II notes under Advanced LFI, PHP-speific wrappers (php://) help us get around LFIs that include(flag2.php). We can assume that include(flag2.php) is being used in the source code for the website because a simple LFI exploitation will not work.  Since the flag is stored in a PHP variable, we can use PHP wrapping to view the PHP source code and find the flag. According to the notes, *?page=php://filter/convert.base64-encode/resource=/../../../../../etc/hosts* is the template for PHP wrapping. By appending everything following ‘page=’ into our URL we can wrap the PHP source code in base64. This will allow us to see the PHP flag2 variable once the base64 is decoded. However, since we don’t need to go to the root we can change ‘r*esource=/../../../../../etc/hosts*’ to ‘r*esource=../../flag2*’ because we only have to go up two directories to get to the web root which is where flag2.php is stored. We also aren’t looking for hosts. We are looking for the flag2 file. The final URL becomes: http://web.n0l3ptr.com:8080/WebII/profile/index.php?page=php://filter/convert.base64-encode/resource=../../flag2 which displays:




Which is the base64 encoded PHP code of the web page. Simply by base64 decoding the message we can see the flag. This solution was found without any error.




Flag = flag{y0u_d1dn7_s4y_th3_m4g1q_w0rd}

# Super Heroes

50 points

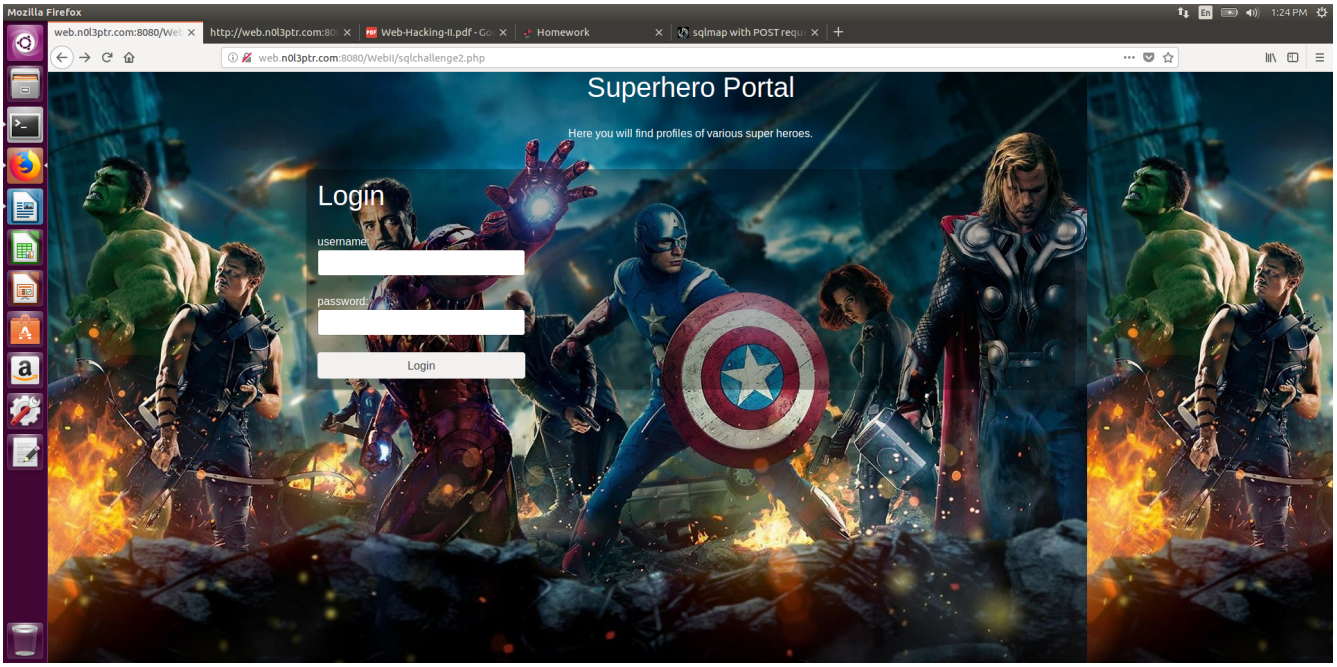
# Problem:

Superheroes need to maintain a social media presence too. We found a site (that Tony Stark paid for) where superheroes maintain their personal profiles. Unfortunately, super heroes don’t know much about web security. Can you use your super hacker skills to find the flag?

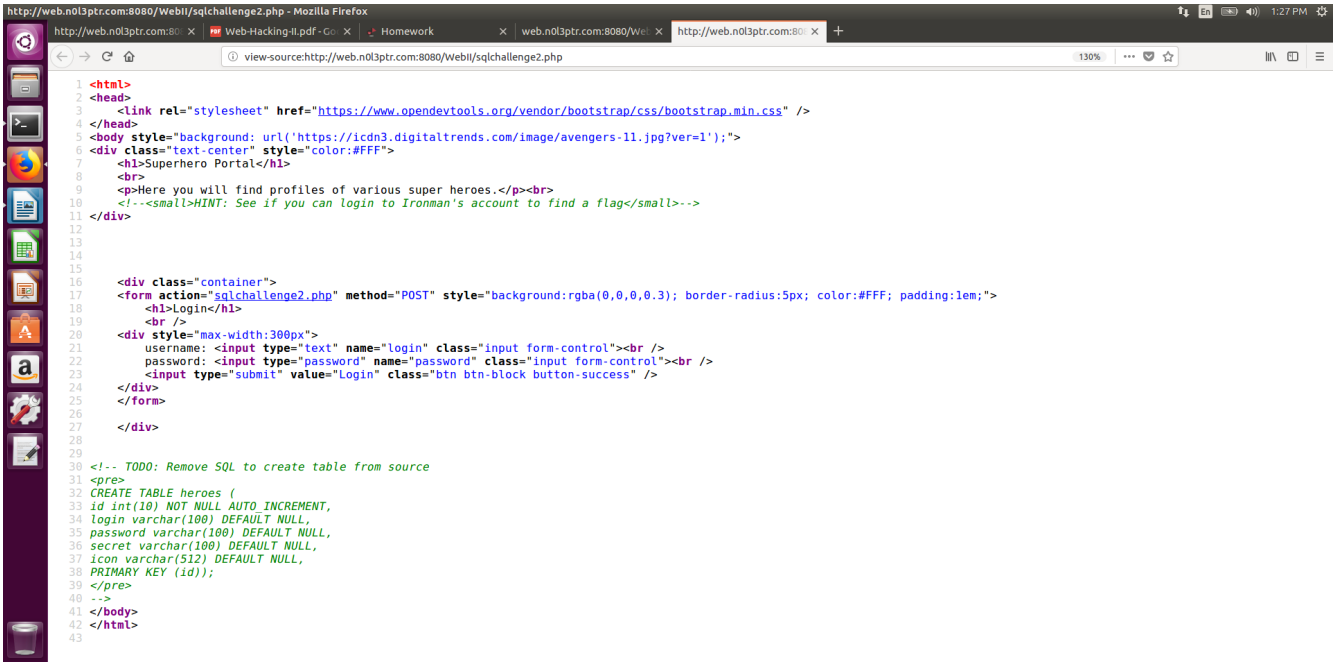
http://web.n0l3ptr.com:8080/WebII/sqlchallenge2.php

# Solution:

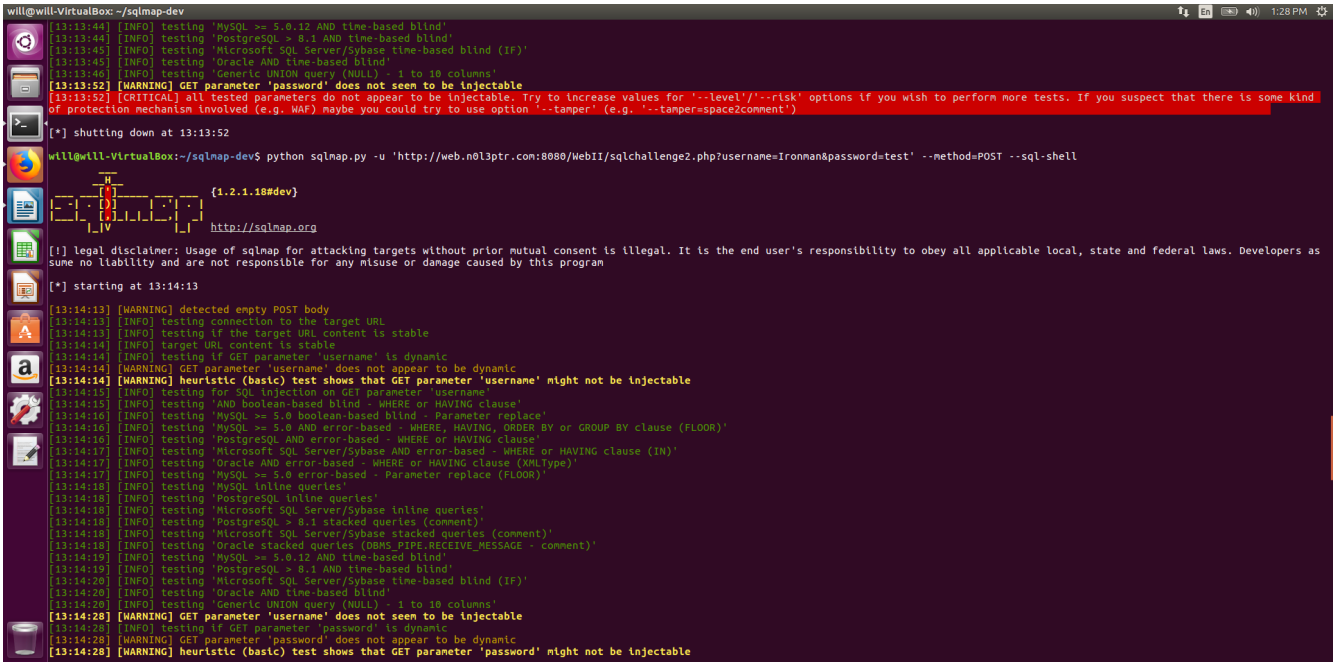
The web page is as follows:



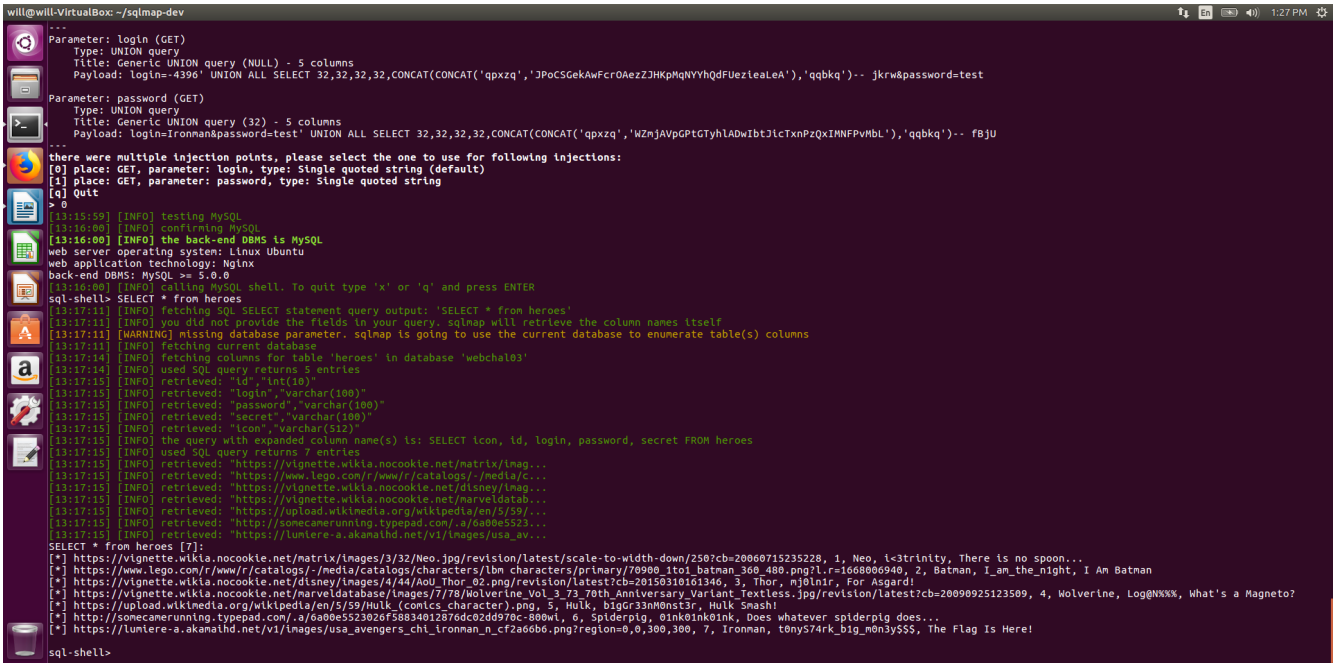

This seems like a SQL injection similar to the SQLi question from WebI. To perform a SQL injection I need to view the table layout and the structure of the SQL statement. To view this I viewed the web page source.




Upon inspection I see a hint to login to Ironman's account to find the flag. I attempted to perform a standard SQL injection by appending the URL to http://web.n0l3ptr.com:8080/WebII/sqlchallenge2.php?login=Ironman&password=test'OR '5'='5' This yielded no results. I was trying to force the SQL to always evaluate to true and therefore log me into Ironman. I knewthe 'login' and 'password' data fields from inspecting the PHP of the web page. I decided it was time to try SQLmap. After installing SQLmap I ran the command *python sqlmap.py -u 'http://web.n0l3ptr.com:8080/WebII/sqlchallenge2.php?login=Ironman&password=test' --method=POST –sql-shell.* SQLmap is a python program which takes the -u parameter followed by the website that is to be SQL injected. I appended *?login=Ironman&password=test* to the web page URL. This is because we are attempting to login as Ironman and the web page is expecting the login from method POST. Post is received from the URL. The test password could be any value. According to the slides, we should declare the method using –method=. The method is post because upon inspection of the source code we see the method is set to post. I used the sql-shell parameter because Jacob informed us in the notes that it was a good parameter to use when SQL injecting. The parameter invokes the built in SQL interpreter so one can interact with the injected websites SQL. After running the command I got the following results.

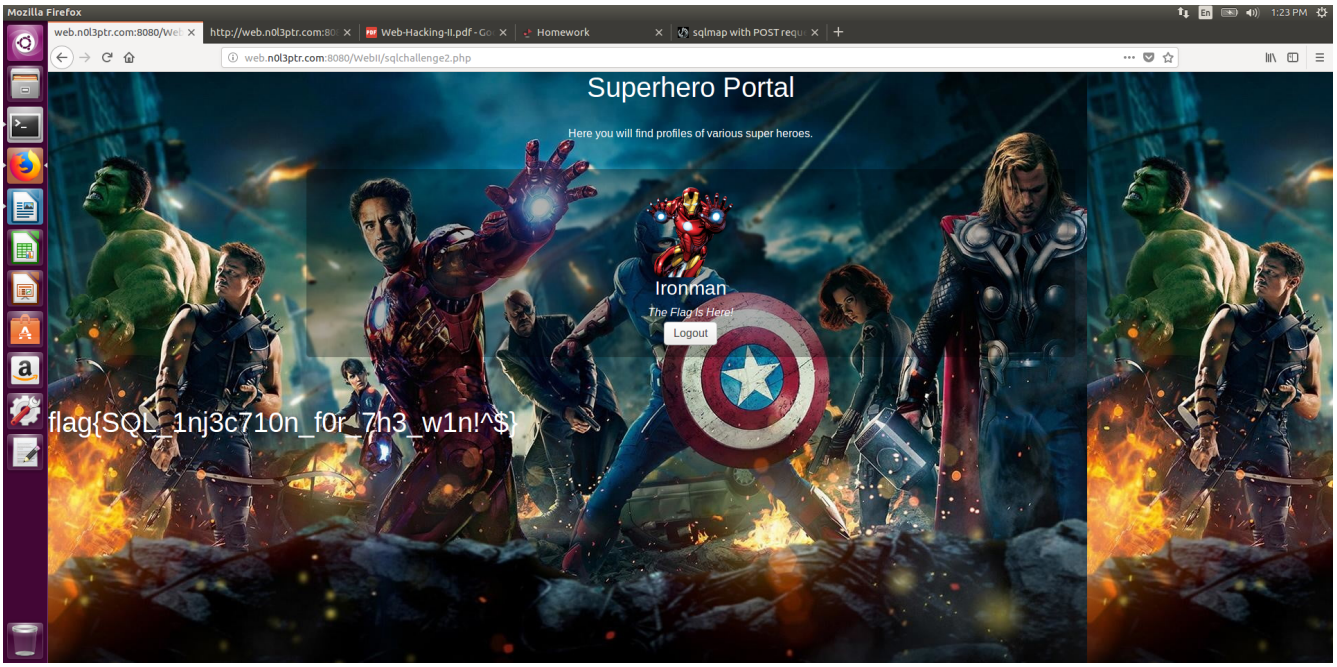



SQLmap informs us that the parameters login and password are in fact of method GET. After failing to inject the website with null values SQLmap tries a random integer union. After a lot of SQLmap testing the password and login parameters are exploited. I then told SQLmap to inject using the GET, parameter: login, type: Single quoted string. This injected us into the database and launched the sql-shell.




Now that I'm in the SQL shell I can run SQL commands on the database. The web page's source code reveals that the name of the table that the SQL queries are occurring on is 'heroes'. To get the passwords I typed 'SELECT * from heroes' as instructed in the notes. This revealed the the id, login, password, secret, and icon in that order of all seven super hero accounts. I see the Ironman entry is: [*]

*https://lumiere-a.akamaihd.net/v1/images/usa_avengers_chi_ironman_n_cf2a66b6.png?region=0,0,300,300, 7, Ironman, t0nyS74rk_b1g_m0n3y$$$, The Flag Is Here!*
Since the password follows login and the login is Ironman the password must be t0nyS74rk_b1g_m0n3y$$$. Upon entering the login credentials on the website we get here and the flag is revealed.




Flag = flag{SQL_1nj3c710n_f0r_7h3_w1n^$}