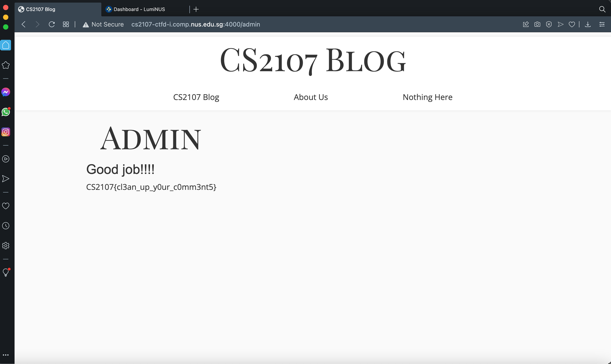
E1 - Nothing to See Here, Really?

I uncommented the commented-out portions of the HTML code. Below is the output

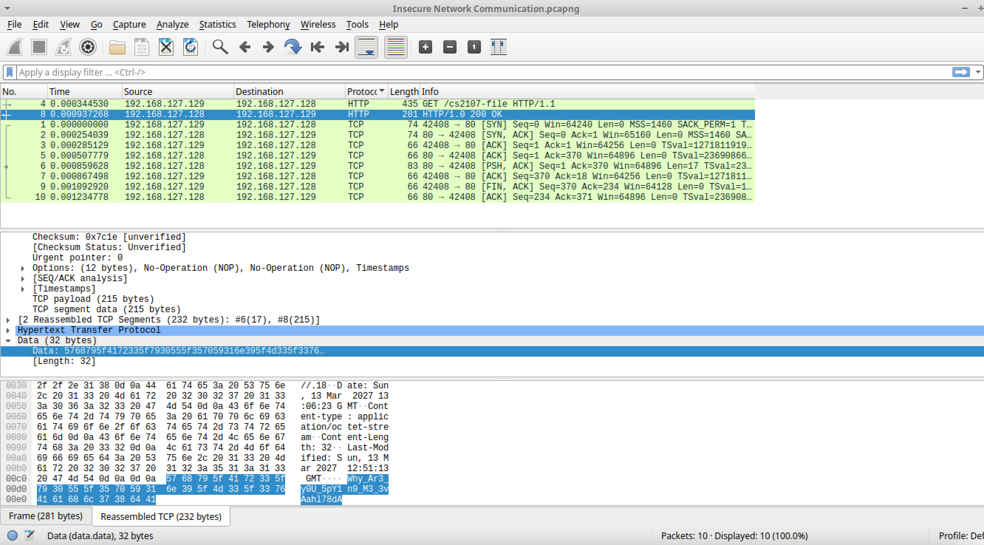


E2 - Linux Access Control

I read up chmod and saw that the correct permission should be **743** for -rwxr—wx

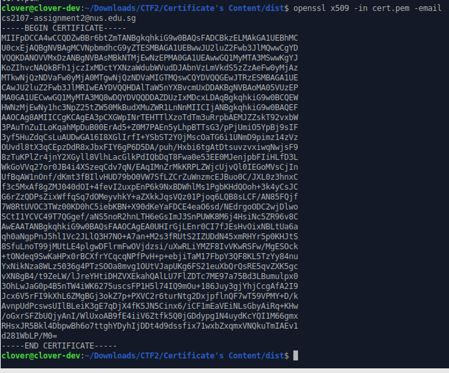
E3 - Insecure Network Communication

I downloaded Wireshark and saw that it intercepted the following data.



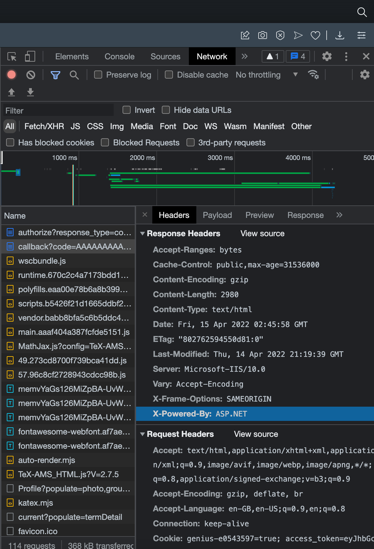
E4 - Certificate’s Consent

I used the command ***openssl x509 -in cert.pm -email*** to obtain the email.



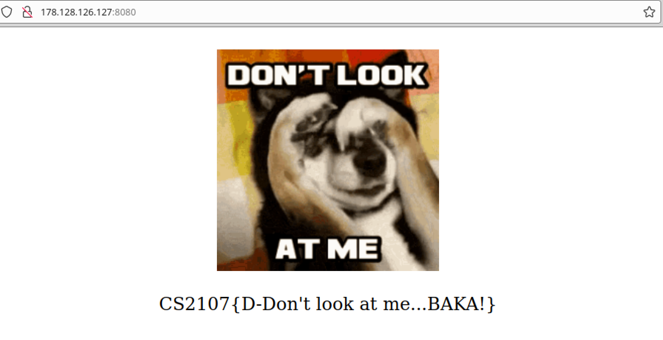
E5 - What Powers Me

I checked out the response headers for LumiNUS and found that it is ASP.Net



E6 - Host Reconnaissance

First I used the command ***nmap 178.128.126.127*** to scan for open ports, and found 4 open ports. Scanning them individually, I found out that the flag is located in port 8080, so I used the command ***curl 178.128.126.127:8080*** to find the flag, and typed it in a url to confirm.

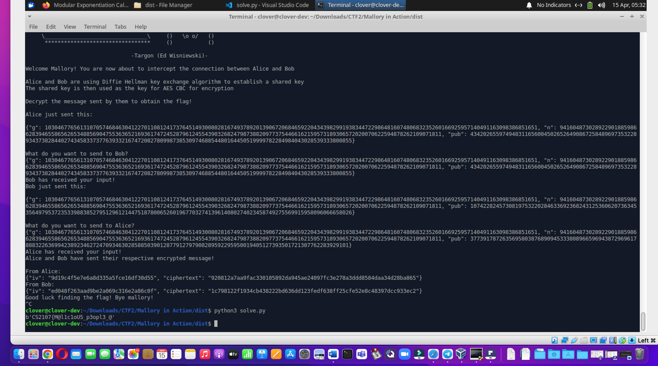


M1 - Mallory in Action

I used the idea that I can spoof either Alice or Bob’s private key to generate a spoofed public key, which in turn means I can trick either Alice or Bob into auto-generating a shared key that I can calculate by taking (A or B’s public key) ^ (spoofed private key) mod (n) to get the shared key. I never have to determine what Alice or Bob’s private keys actually are.

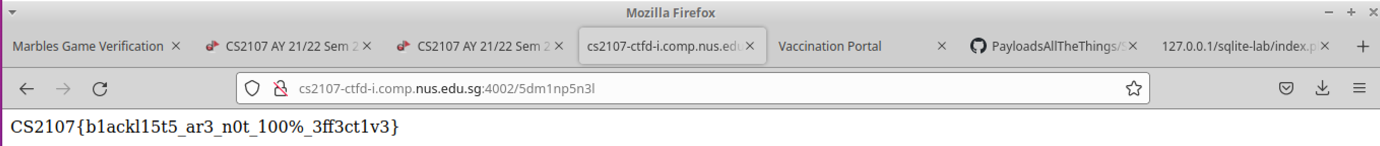
Below is a screenshot of my methodology, when I spoofed Bob’s private key as 2 to obtain a ciphertext for Alice that I then decrypted.

I re-used the steps above the obtain Bob’s part of the flag.



M2 - The Prequel: I Dislike Some Keywords

I typed ***admin’ --*** into the username field, allowing me to bypass login.



M3 - Please Join the SQLi Games

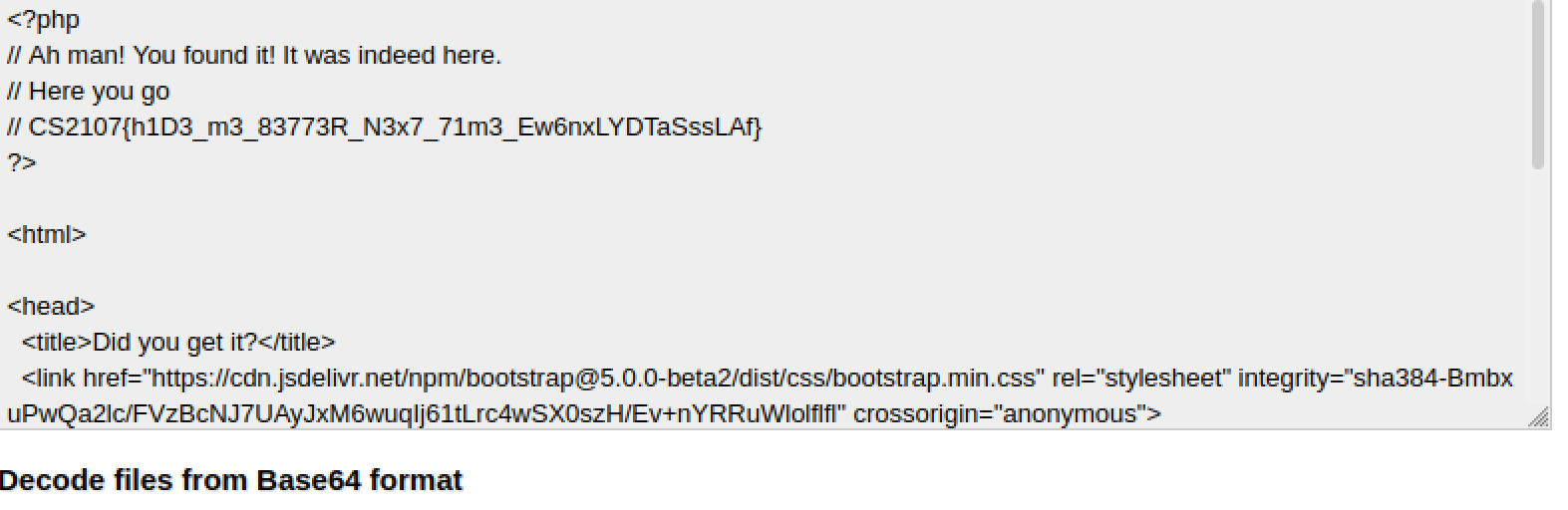
I typed ***admin’--*** into the username fields for all 3 challenges, and used an online SHA256 tool to convert the flags to hashes.



M4 - File Inclusion

I modified the link to [**http://cs2107-ctfd-i.comp.nus.edu.sg:4004/?f=php://filter/convert.base64-encode/resource=secret.php**](http://cs2107-ctfd-i.comp.nus.edu.sg:4004/?f=php://filter/convert.base64-encode/resource=secret.php)

Then I used an online base64 decoder tool to decode the base64-encoded message.



M5 - Ret2Win

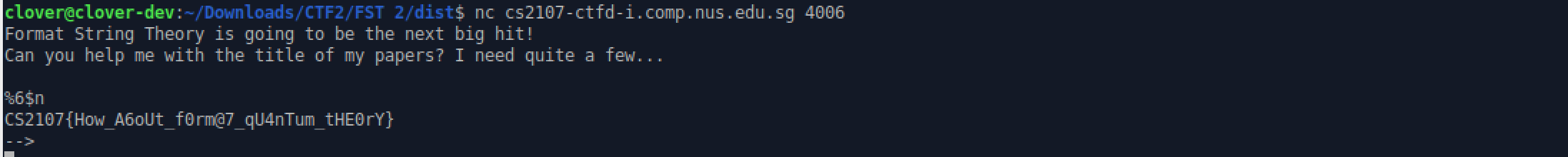
First I used arbitrarily long strings of ‘A’ chars to the program as input. Using gdb, I managed to figure out that the buffer size is 40 before reaching the return address. I then overwrote the initial return address of the main() function with the address of the win() function

The command I used is ***python3 -c ‘print(“A”\*40 + “\x46\x11\x40\x00\x00\x00”)’ | nc cs2107-ctfd-i.comp.nus.edu.sg 4005***



M6 - Format String Theory

Using gdb, I figured out that the address of the permission variable is the 6th argument on the stack, so I used the command ***%6$n*** to write change the permission variable, thus opening fst\_secrets.txt



H2 - The Sequel

First, to see the number of SELECT arguments I need to have, I typed in ***‘ORDER BY 1 --***. When it returned the vaccination table to me, I incremented 1 to 2, and repeated the process until **‘ORDER BY 5 –** gives an error, then I know the number of arguments is 4.

From then on, I used UNION in all my attacks. First, I used ***‘ UNION SELECT 1, 2, 3, sqlite\_version()*** **--** to determine that the database is written SQLite. Using this information, I know that SQLite has a table called sqlite\_master which stores all the table names and the sql code used to write the tables. I used the query ***‘ UNION SELECT 1,2, tbl\_name, sql FROM sqlite\_master --*** to find all the important information.

From the sql code, I now know that there is a table called users, with attributes uname and peedoubleyou. Using the query ***‘UNION SELECT 1, 2, uname, peedoubleyou FROM users --***, I obtain the flag.

