Practical Cyber Security Fundamentals
Assignment 2: Web

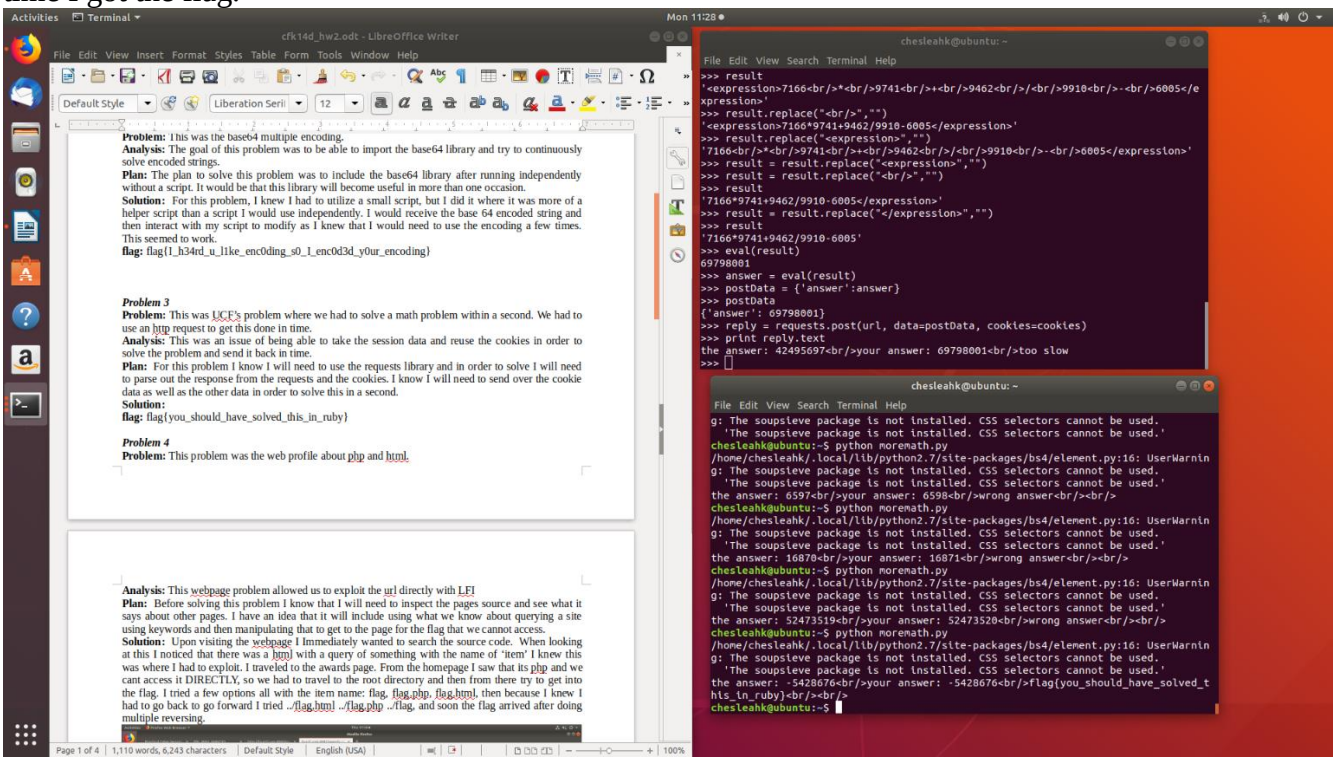
Overview
Assignment: Web Exploitation
Student: Chesleah Kribs
Date: January 25, 2019

Problem 1
Problem: This was an ascii/hex encoding problem
Analysis: The main goal of this problem was to be able to decipher ascii encoding using the url lib library.
Plan: The plan to solve this problem was straightforward as I knew I had to use the urllib library in python and use the unquote function.
Solution: In order to solve this problem, I created a small script that imported the urllib library and used the ascii given as a flag and used the function unquote to find the flag.
flag: flag{h1d1ng_1n_pl41n_s1t3}

Problem 2
Problem: This was the base64 multiple encoding.
Analysis: The goal of this problem was to be able to import the base64 library and try to continuously solve encoded strings.
Plan: The plan to solve this problem was to include the base64 library after running independently without a script. It would be that this library will become useful in more than one occasion.
Solution: For this problem, I knew I had to utilize a small script, but I did it where it was more of a helper script than a script I would use independently. I would receive the base 64 encoded string and then interact with my script to modify as I knew that I would need to use the encoding a few times. This seemed to work.
flag: flag{I_h34rd_u_l1ke_enc0ding_s0_I_enc0d3d_y0ur_encoding}

Problem 3
Problem: This was UCF's problem where we had to solve a math problem within a second. We had to use an http request to get this done in time.
Analysis: This was an issue of being able to take the session data and reuse the cookies in order to solve the problem and send it back in time.
Plan: For this problem I know I will need to use the requests library and in order to solve I will need to parse out the response from the requests and the cookies. I know I will need to send over the cookie data as well as the other data in order to solve this in a second.
Solution: For my solving of this problem I knew I had to import BeautifulSoup and requests. I grabbed the url and then was able to grab the url using get and then set the cookies to the response.cookies. This was a major hangup in my ability to solve as I would get further and further into solving and my math would be correct but it would be going off of a different 'get' as I would do two of them (major mistake). I would then be able to parse the text out and then replace everything that would

not allow me to evaluate the math portion. Once I parsed it and replaced everything, I used eval(result) to get me the result and then sent this back in a post response along with the cookies and after the third time I got the flag.



flag: flag{you_should_have_solved_this_in_ruby}

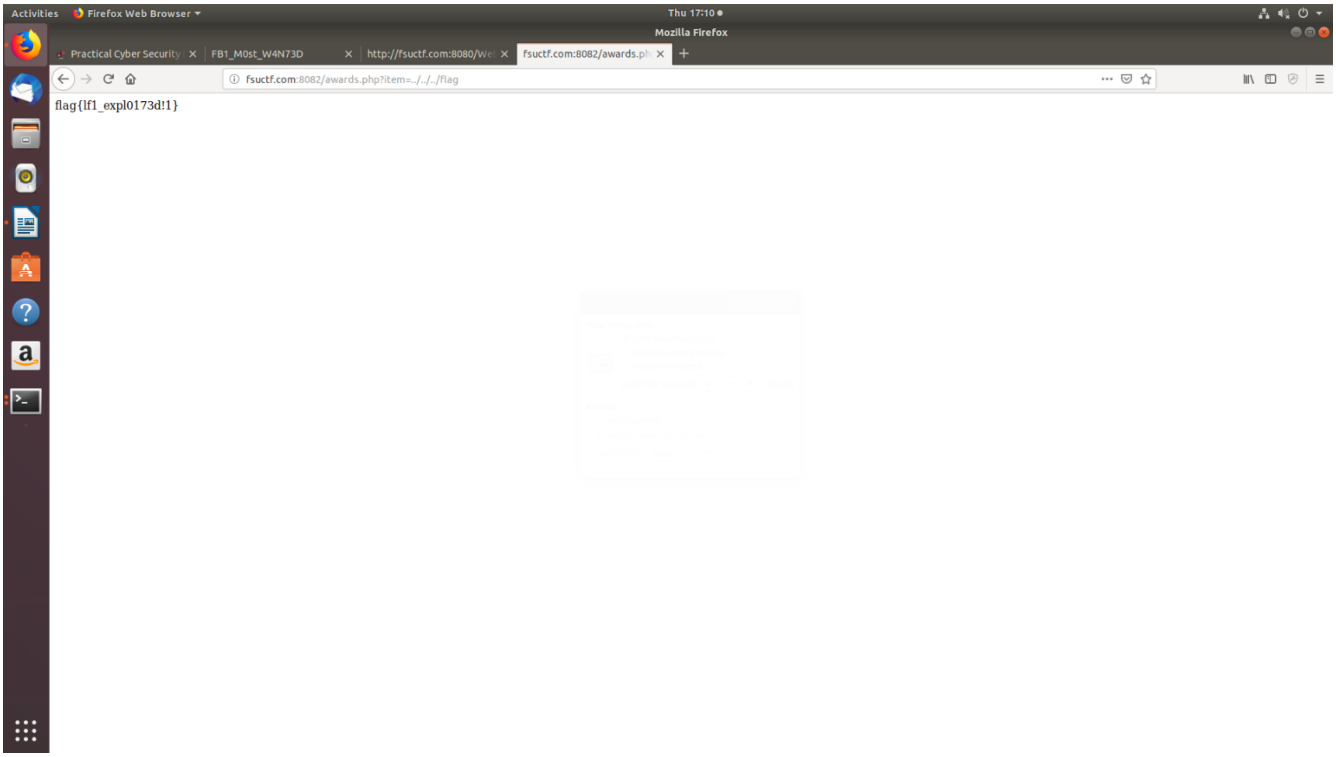
Problem 4

Problem: This problem was the web profile about php and html.

Analysis: This webpage problem allowed us to exploit the url directly with LFI

Plan: Before solving this problem I know that I will need to inspect the pages source and see what it says about other pages. I have an idea that it will include using what we know about querying a site using keywords and then manipulating that to get to the page for the flag that we cannot access.

Solution: Upon visiting the webpage I Immediately wanted to search the source code. When looking at this I noticed that there was a html with a query of something with the name of 'item' I knew this was where I had to exploit. I traveled to the awards page. From the homepage I saw that its php and we cant access it DIRECTLY, so we had to travel to the root directory and then from there try to get into the flag. I tried a few options all with the item name: flag, flag.php, flag.html, then because I knew I had to go back to go forward I tried ../flag.html ../flag.php ../flag, and soon the flag arrived after doing multiple reversing.



flag: flag{lf1_expl0173d!1}

Problem 5

Problem: This was the SQL injection problem.

Analysis: This problem used a vulnerability where we could exploit with a SQL injection.

Plan: For this problem, I had to use the notes to see what an array key would be in the source code. I knew that I would have to use the items described to query the page. I would also have to use items that are always true and try to solve the users and passwords.

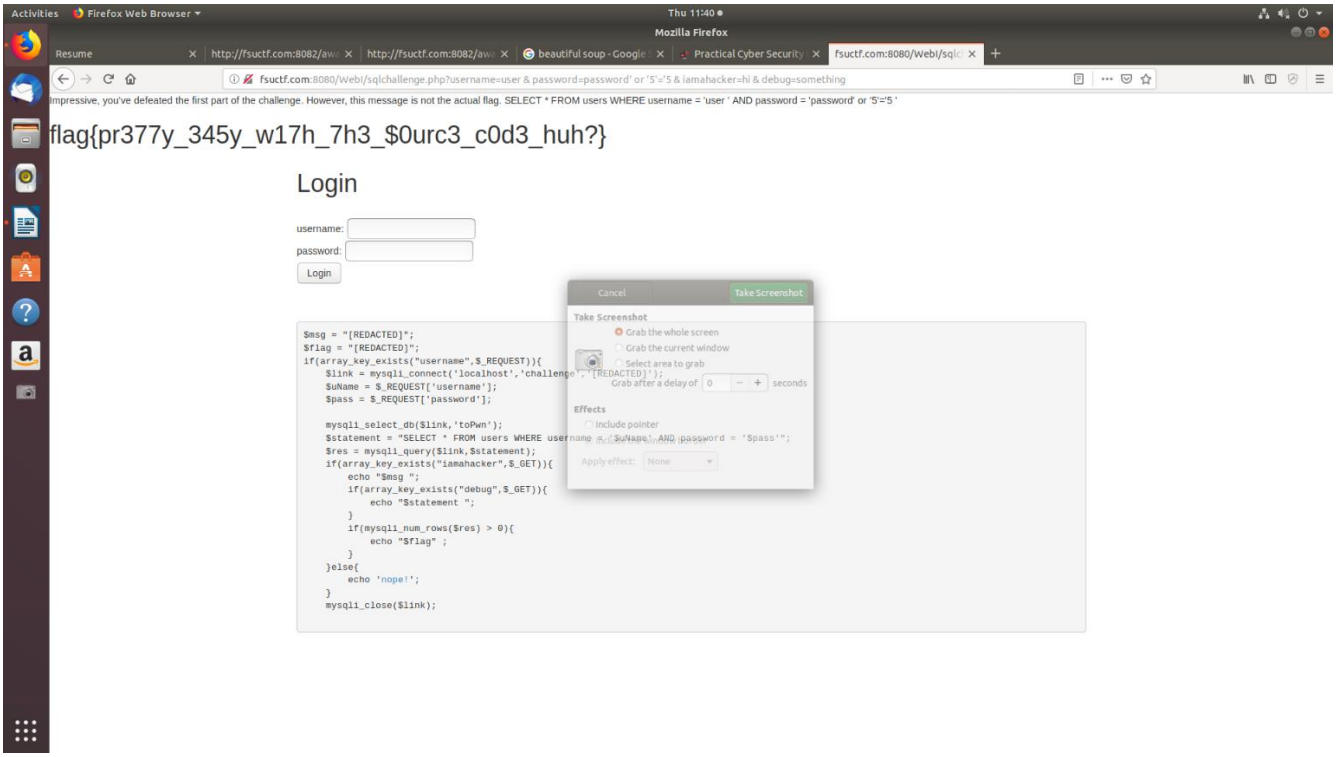
Solution: For the first part of this it was all about trial and error. In order to first enter the loop, I knew username had to be an array key used, so I input a random username. This allowed me to enter the loop. Next I knew I had to have the item called ‘password’ so I entered that with another random password. I then saw in order to get the message I needed a key called ‘iamahacker’ and to get the statement I had to enter debug, so I entered iamahacker=hi & debug=something, which gave me a message.

Impressive, you've defeated the first part of the challenge. However, this message is not the actual flag.
SELECT * FROM users WHERE username = 'user ' AND password = 'password '

Then I knew that if there was a comparison being made I could override this where one row will always be true. So I entered:

http://fsuctf.com:8080/WebI/sqlchallenge.php?username=user%20&%20password=password%27%20or%20%275%27=%275%20&%20iamahacker=hi%20&%20debug=something

And I was able to get the flag.



flag: flag{pr377y_345y_w17h_7h3_$0urc3_c0d3_huh?}

Problem 6

Problem: This was a problem that helped us work with obfuscated code, it was about the hackers.

Analysis: This problem helped us to understand about a vulnerability that allowed for Cross Site Scripting (XSS). We were able to find obfuscated code by decoding a string which consisted of a function that we could call in the console of F12.

Plan: My plan to solve this problem involved going through the source code and using the f12 key. I knew at some point the webpage could not be shown and from there I would need to request what gets sent back to the browser from http. Once I would received this information I can go from there and see how to accept what comes back and then solve it.

Solution: To solve this problem, I first went to the webpage: http://fsuctf.com:8080/WebI/tophackers.php#. I right clicked to see the page source and I could see that after Jonathan James was another image that was hidden in plain sight through the code. If I were to click on that image it would take me to another webpage. Which says it cannot be displayed because it contains errors. So I tried to make a small script that requests that url and sees what it sends back to me! I first request the url, and then print out what it sends back. It sends back a bas64 encoded string, which I then decode. I then, yes, do by hand and copy this string over to the f12 console.log. I attempted to do this in the url over the browser but it returns that it is too long. When I do this I am able to define the printFlag function, and then therefore call it to find the flag.

flag:flag{y0u_d1d_n07_d0_7h47_By_h4nD_D1d_y0U?}

```
import requests

url = "http://fsuctf.com:8080/img/trinity.jpg"

response = requests.get(url)

from bs4 import BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

import base64
decoded = base64.b64decode(response.text)


print decoded
~
~
~
~
~
~
~
~
~
-- INSERT --                                    8,51          All
```

```
apache-ant-1.10.5  encode2.py         newusers.sh        ssl.log
bro_logs           examples.desktop   newusers.txt       Templates
cat                files.log          packet_filter.log  users
coinsolve.py       gamesolve.py       peld               users.sh
conn.log           hashsolve.py       Pictures           Videos
Desktop            httphelp.py        pubkeys            weird.log
dns.log            http.log           Public             x509.log
chesleahk@ubuntu:~$ cat moremath.py
import requests
from bs4 import BeautifulSoup

url = 'http://ctf.hackucf.org:4000/calc/calc.php'
response = requests.get(url)

# the name field is called 'answer'

soup = BeautifulSoup(response.text, 'html.parser')
print soup.prettify()


practice = soup

print practice
chesleahk@ubuntu:~$
```

```
<html>
<head><title>414 Request-URI Too Large</title></head>
<body bgcolor="white">
<center><h1>414 Request-URI Too Large</h1></center>
<hr><center>nginx/1.4.6 (Ubuntu)</center>
</body>
</html>
```

```
» printFlag()
ReferenceError: printFlag is not defined [Learn More]
»
```