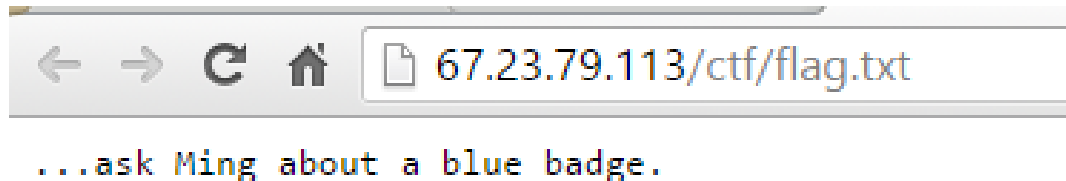


Comp 116 Fall 2015 CTF Writeup

by Arthur Berman, Hannah Clark, Nitesh Gupta, and Dylan Phelan

Social Engineering

We were slightly discouraged that we were explicitly instructed **not** to give Ming a hug. This hurt. But after we swallowed our pride, we realized that the description for Challenge 4 was likely a social engineering related hint. So like a social engineer, we did the easiest thing first: we asked nicely. From there we were pointed in the direction of a page titled flag.txt, where we found the following:

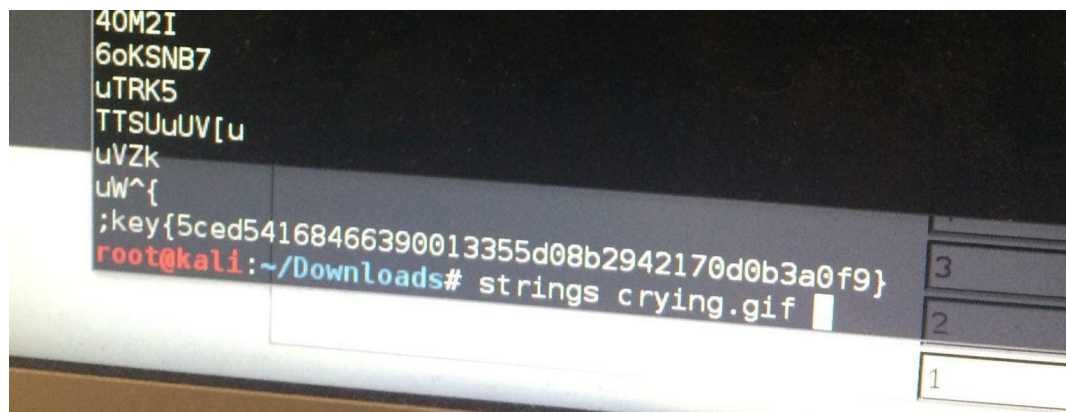


One of us walked over, asked about that badge, and were pulled into a separate room where Ming instructed us to copy down a long sequences of strictly numbers. Upon entering this sequence as a key, we were granted points for Challenge 4.

Images

We noticed a number of images available directly on the web pages. To see if anything interesting was hidden in the images, we downloaded every image we could find, ran strings on each, and looked for things resembling flags. We had success with the image crying.gif.

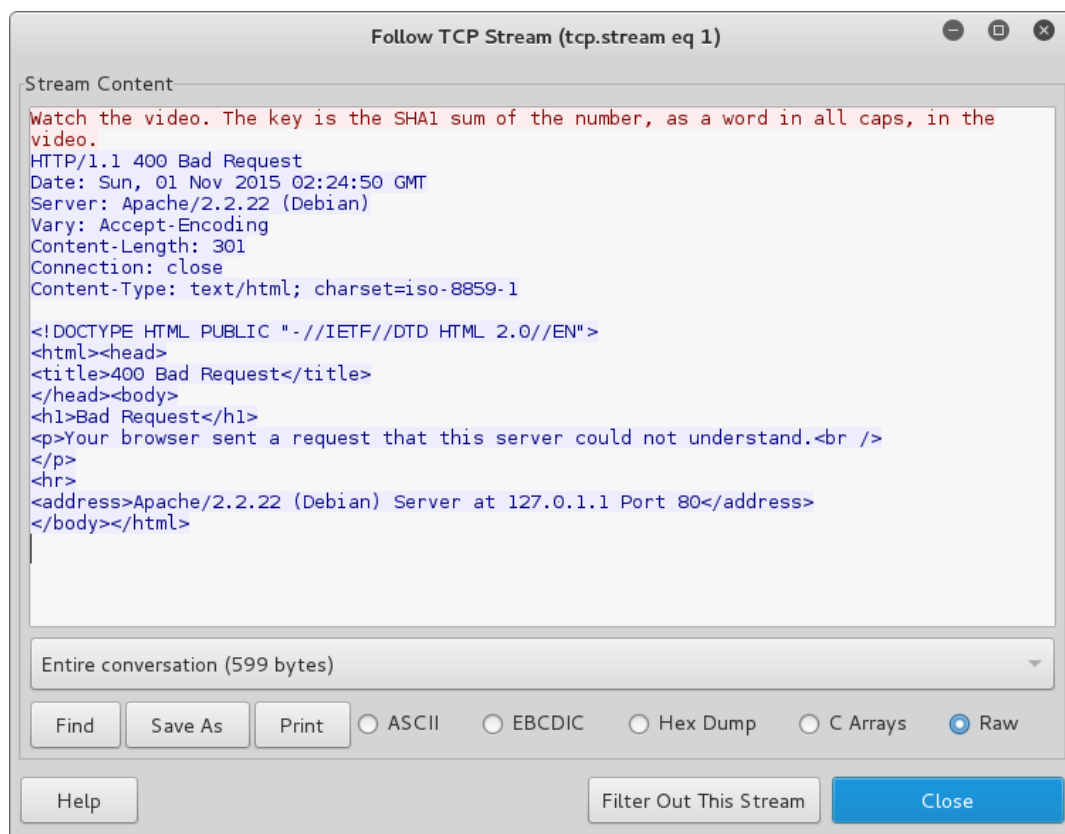
```
strings crying.gif | grep key
```



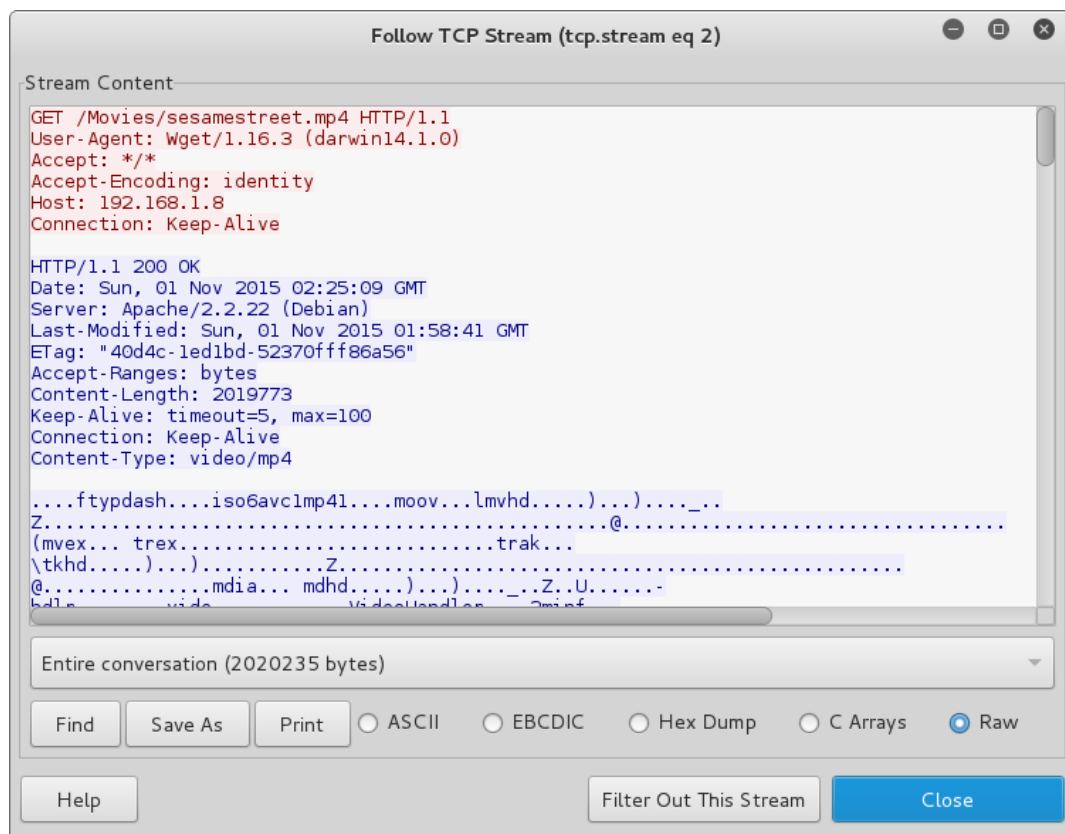
Extraction from Pcap File

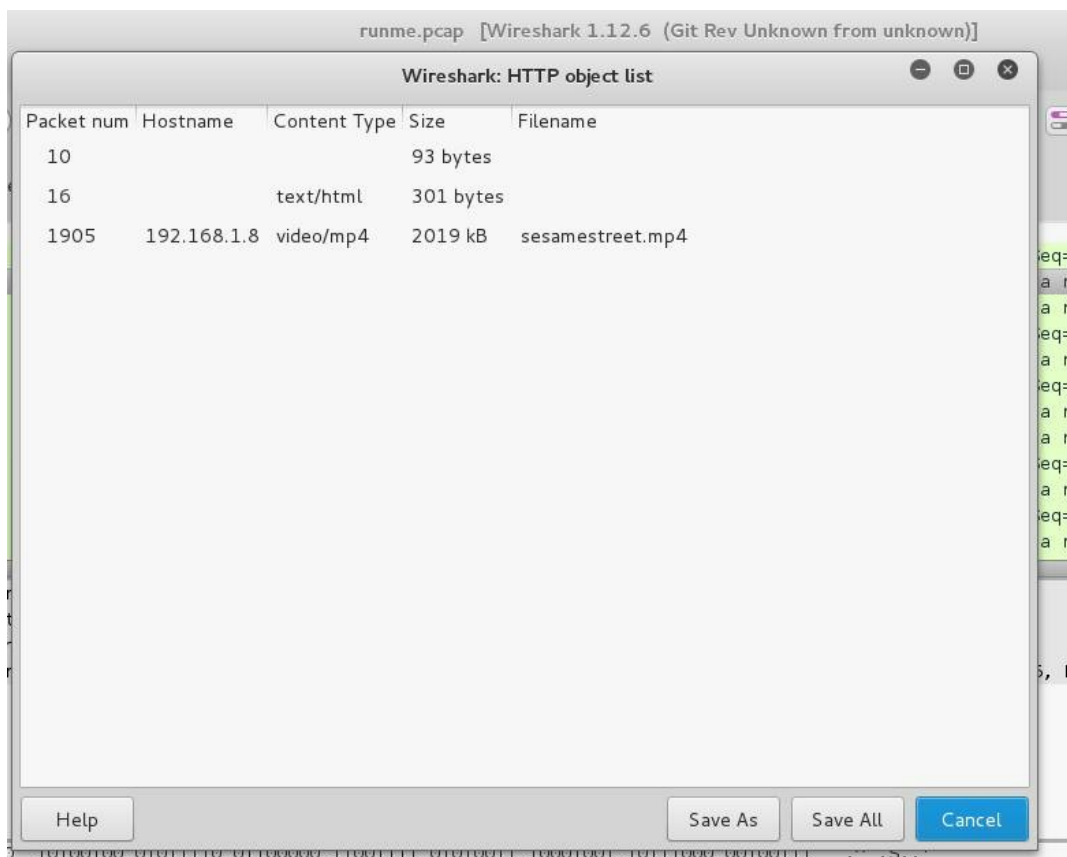
Upon downloading the runme.exe file from the 67.23.79.113/ctf/ page, we noticed that it was actually a pcap file by running file. We ran strings runme.exe and discovered

a hint about how to generate a flag. The same text, found by the following the TCP conversation of the packet in Wireshark, is shown in the image below.



We then found the TCP conversation where a video was transmitted in Wireshark, shown below, and used Wireshark's built-in Export Objects -> HTTP feature to find the video (sesamestreet.mp4) and save it locally, so we could watch it.





The number from the video was quite obviously 7, so we then generated the flag using sha1-online.com

Home Page | [SHA1 in JAVA](#) | [Secure password generator](#) | [Linux](#)

SHA1 and other hash functions online generator

SEVEN hash

sha-1

Result for

sha1: cabd534c35ee6a39365f4ed3bce4eafdcc3d4b8d

[SHA-1](#) [MD5](#) on Wikipedia

We also looked at any metadata for the video, but did not see anything. Beyond this, we continued looking through the packets for better hidden flags or clues. Nothing was immediately apparent. Although the word key itself did appear in the raw data for the video when viewed as ascii, it could not be clearly grouped with other characters to form a flag. When further investigating the dump it Wireshark, it became apparent that a small portion of the pcap file was not being displayed. However, we were not able to find a way to display this.

FTP

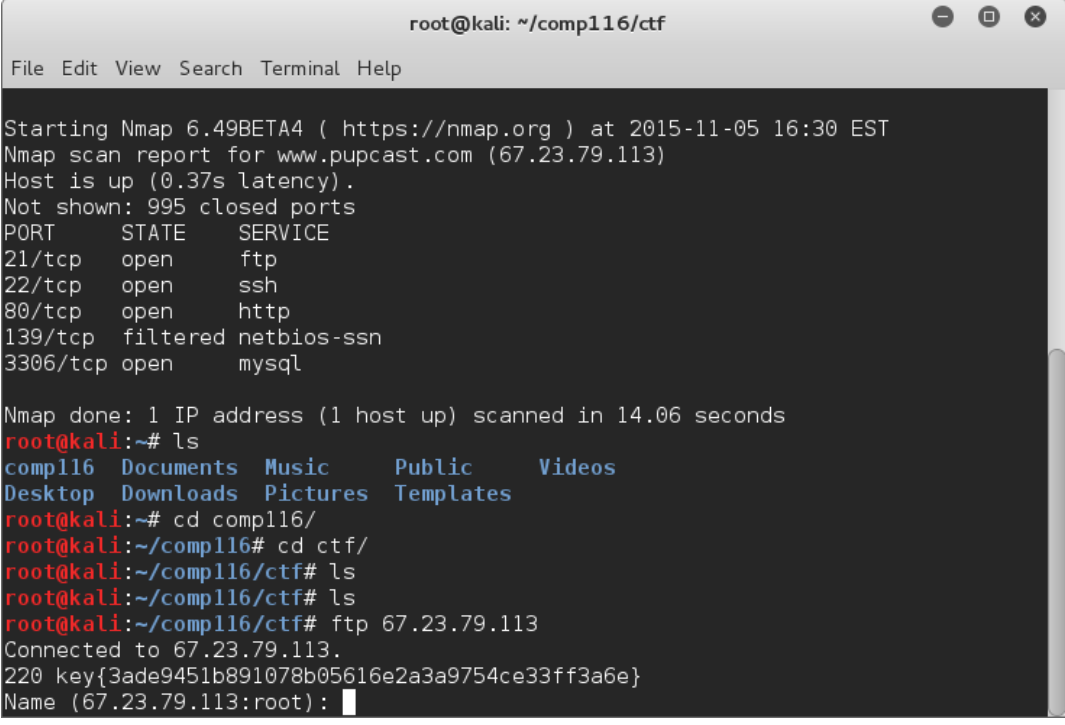
First, we conducted an NMAP scan for open ports.

```
nmap 67.23.79.113
```

Among many ports, we found that FTP was open. So we attempted to login to it.

```
ftp 67.23.79.113
```

Upon logging in, we got access to the first flag.

A screenshot of a terminal window titled 'root@kali: ~/comp116/ctf'. The terminal shows the output of an Nmap scan for 67.23.79.113, listing open ports 21/tcp (ftp), 22/tcp (ssh), 80/tcp (http), 139/tcp (netbios-ssn), and 3306/tcp (mysql). It then shows the user navigating through directories (comp116, ctf) and finally connecting to the FTP server at 67.23.79.113, logging in as root. The terminal text is as follows:

```
root@kali: ~/comp116/ctf
File Edit View Search Terminal Help

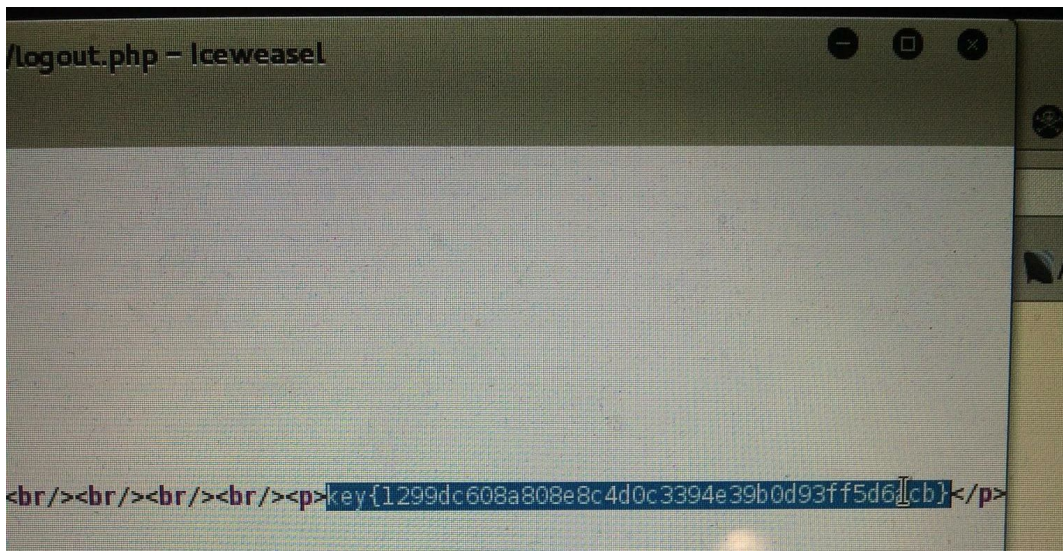
Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2015-11-05 16:30 EST
Nmap scan report for www.pupcast.com (67.23.79.113)
Host is up (0.37s latency).
Not shown: 995 closed ports
PORT      STATE      SERVICE
21/tcp    open      ftp
22/tcp    open      ssh
80/tcp    open      http
139/tcp   filtered  netbios-ssn
3306/tcp  open      mysql

Nmap done: 1 IP address (1 host up) scanned in 14.06 seconds
root@kali:~# ls
comp116  Documents  Music      Public     Videos
Desktop  Downloads  Pictures   Templates
root@kali:~# cd comp116/
root@kali:~/comp116# cd ctf/
root@kali:~/comp116/ctf# ls
root@kali:~/comp116/ctf# ls
root@kali:~/comp116/ctf# ftp 67.23.79.113
Connected to 67.23.79.113.
220 key{3ade9451b891078b05616e2a3a9754ce33ff3a6e}
Name (67.23.79.113:root):
```

After successfully connecting to the server through FTP, we logged in as an anonymous user and then searched for exploits. We attempted some of them in this [thread](http://www.webhostingtalk.com/showthread.php?t=37065&s=) (<http://www.webhostingtalk.com/showthread.php?t=37065&s=>) but nothing ended up working. We were also going to run metasploit to see if we could access to different FTP user but could not get that to work in time.

logout.php

Upon navigating to <http://67.23.79.113/ctf/logout.php>, we noticed that we were immediately redirected back to login.php. By catching the redirect with burpsuite, we were able to view the source on the logout page, which included a key. Upon reflection, we suspect that accessing this page meant that an earlier attempt to sql-inject authentication credentials succeeded without us realizing.

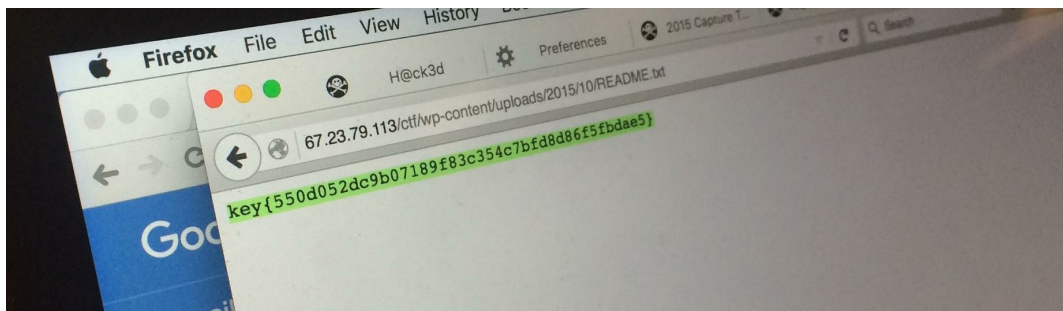


WordPress

After Arthur mentioned that the whole thing was a WordPress blog, we became very excited because they are notorious for vulnerabilities. We ran [wpscan \(http://wpscan.org/\)](http://wpscan.org/) to find what vulnerabilities could be exploited.

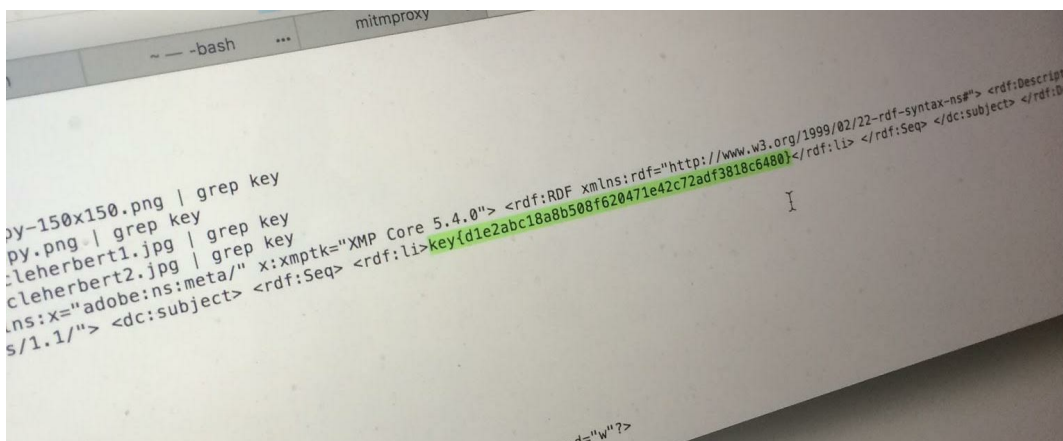
```
ruby wpscan.rb --url http://67.23.79.113/ctf
```

We found three vulnerabilities. First, the uploads directory was exposed. This allowed us to find one flag in the readme.txt.



By running strings on all the images in the uploads directory, we were able to find another flag from the metadata of an image.

```
strings uncleherbert2.png | grep key
```



The next thing we tested was xml-rpc.php. Nitesh knew of this attack vector after experiencing it first hand a few months ago when some botnet was attacking a WordPress site he had set up. Xml-rpc.php allows remote login for WordPress, a feature

that few people actually end up using.

After going to the login page on wordpress, we found that admin and bobo were both suitable usernames. We guessed bobo could be a login by checking out the board page. When entering a wrong username, WordPress will specify that the *username* is wrong. When entering a correct username, it will say that only the password was wrong.

We ran a huge brute force attack on XML-RPC.php with a large wordlist, but we did not have any luck with that.

```
wget
```

```
https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/10_1_passwords.txt
```

```
php ./wpbruteforce.php http://67.23.79.113/ctf/xmlrpc.php
usernames.txt passwords.txt
```

SQL Injection

We found that on the board.php page, we could inject SQL through the ID parameter. We knew that there must be a tool to exploit SQL injections to get more information about a database. After a quick Google search, we found [sqlmap \(http://sqlmap.org/\)](http://sqlmap.org/).

The first command told us that there was indeed a SQL injection possible. It gave us a few bits of sample code that we confirmed in the browser.

```
python sqlmap.py -u "http://67.23.79.113/ctf/board.php?id=1"
```

After that, we tried to get access to the wordpress database. We knew that the default database for wordpress is "wordpress."

```
python sqlmap.py -u "http://67.23.79.113/ctf/board.php?id=1" -z
"ign,flu,bat" --tables -D wordpress
```

But that database was not found.

However, simply passing the "tables" argument allowed us to get a full list of the tables in the sql database!

```
python sqlmap.py -u "http://67.23.79.113/ctf/board.php?id=1" -z
"ign,flu,bat" --tables
```

From there, we got access to all of the hashes in the database.

After that, we started to try to crack the password hashes. We had the password hash for root. We were so close to victory! We felt untouchable and on top of the world, just like the NSA before Snowden's leaks.

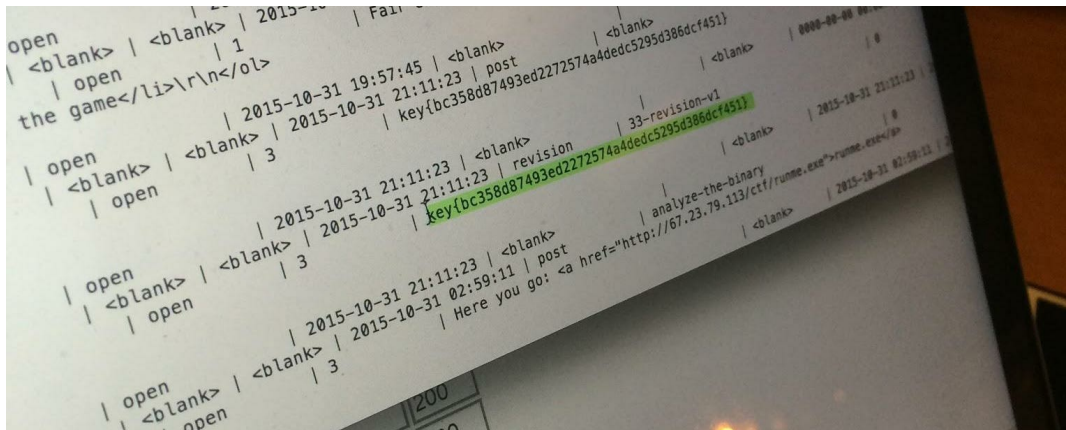
The user "defcon" had a password of NULL, so we tried to get into mysql using this. However we got an error that said there were too many connections being attempted. It gave us a SQL command that we could have used to fix this error, but we never ended up successfully injecting the command, perhaps because of a lack of user permissions.

After figuring out that the database table for Wordpress was called "board," we decided to use sqlmap to dump the entire table.

```
python sqlmap.py -u "http://67.23.79.113/ctf/board.php?id=1" -z
```

```
"ign,flu,bat" --tables -D board
```

From there, a quick grep of the wp_posts table allowed us to find a draft of a post that contained another flag.



Looking in the wp_users table, we found the usernames and hashed passwords of every user. We used John The Ripper to find that the password of Bobo was "superman." We logged into WordPress, but could only find that same draft post that contained a key we had already stolen.

The last thing we tried to do was test for SQL injection on admin.php. We ran out of time at this point. We used mitmproxy and burp to capture the request. After doing so, we tried to run sqlmap on the POST request to see what kind of data we could get. Unfortunately we could not get the POST request into the right format for sqlmap to run it.

We later found out there was a database table ctf_flags that contained all of the flags that we could have dumped but missed due to our powers of observation being weakened from tiredness, hunger, and some frustration from some of our unsuccessful attempts at finding flags. That we missed such an opportunity has taught us to look more carefully at what we have, improve communication about what we can do with what we have, and to remember plenty of caffeine and snacks for the next time.