HGAME 2025 Week 2 Writeup

Aqua Cat #000247

Crypto (3/3)

Ancient Recall

`Fortune_wheel` 函数本质是对 `FATE` 数组做一个线性变换，可以求逆得到初始的 `Value` 数组。初始的 `card` 和 `Value` 的元素存在对应关系，从而可以求得 `YOUR_initial_FATE` 。

```
value =
[2532951952066291774890498369114195917240794704918210520571067085311474675019,
2532951952066291774890327666074100357898023013105443178881294700381509795270,
2532951952066291774890554459287276604903130315859258544173068376967072335730,
2532951952066291774890865328241532885391510162611534514014409174284299139015,
2532951952066291774890830662608134156017946376309989934175833913921142609334]

for i in range(250):
    assert sum(value) % 2 == 0
    s = sum(value) // 2
    value = [s - value[(i + 1) % 5] - value[(i + 3) % 5] for i in range(5)]

Major_Arcana = ["The Fool", "The Magician", "The High Priestess","The
Empress", "The Emperor", "The Hierophant","The Lovers", "The Chariot",
"Strength","The Hermit", "Wheel of Fortune", "Justice","The Hanged Man",
"Death", "Temperance","The Devil", "The Tower", "The Star","The Moon", "The
Sun", "Judgement","The World"]
wands = ["Ace of Wands", "Two of Wands", "Three of Wands", "Four of Wands",
"Five of Wands", "Six of Wands", "Seven of Wands", "Eight of Wands", "Nine
of Wands", "Ten of Wands", "Page of Wands", "Knight of Wands", "Queen of
Wands", "King of Wands"]
cups = ["Ace of Cups", "Two of Cups", "Three of Cups", "Four of Cups",
"Five of Cups", "Six of Cups", "Seven of Cups", "Eight of Cups", "Nine of
Cups", "Ten of Cups", "Page of Cups", "Knight of Cups", "Queen of Cups",
"King of Cups"]
swords = ["Ace of Swords", "Two of Swords", "Three of Swords", "Four of
Swords", "Five of Swords", "Six of Swords", "Seven of Swords", "Eight of
Swords", "Nine of Swords", "Ten of Swords", "Page of Swords", "Knight of
Swords", "Queen of Swords", "King of Swords"]
pentacles = ["Ace of Pentacles", "Two of Pentacles", "Three of Pentacles",
"Four of Pentacles", "Five of Pentacles", "Six of Pentacles", "Seven of
Pentacles", "Eight of Pentacles", "Nine of Pentacles", "Ten of Pentacles",
"Page of Pentacles", "Knight of Pentacles", "Queen of Pentacles", "King of
Pentacles"]
Minor_Arcana = wands + cups + swords + pentacles
tarot = Major_Arcana + Minor_Arcana
```

```
YOUR_initial_FATE = []
for v in value:
    for card in tarot:
        if card in Major_Arcana:
            if v == tarot.index(card) ^ (-1):
                YOUR_initial_FATE.append("re-" + card)
            elif v == tarot.index(card):
                YOUR_initial_FATE.append(card)
        else:
            if v == tarot.index(card):
                YOUR_initial_FATE.append(card)

FLAG=("hgame{"+"&".join(YOUR_initial_FATE)+"}").replace(" ","_")
print(FLAG)
```

Intergalactic Bound

THCurve 是 Twisted Hessian Curve 的缩写，其方程为 $ax^3+y^3+1=dxy$。可以用已知的两组 (x,y) 解出参数 a,d 的值。

在 Explicit Formula Database 里可以找到 Twisted Hessian Curve 转 Weierstrass 的公式，之后用 sage 自带的 log 函数就能得到 x 的值。

```
p = 55099055368053948610276786301
G = (19663446762962927633037926740, 35074412430915656071777015320)
Q = (26805137673536635825884330180, 26376833112609309475951186883)
ciphertext =
b"k\xe8\xbe\x94\x9e\xfc\xe2\x9e\x97\xe5\xf3\x04'\x8f\xb2\x01T\x06\x88\x04\xeb3J
Pk$\x00:\xf5"

# a*x^3+y^3+1=d*x*y
F = GF(p)
a, d = matrix(F, [[G[0]^3, -G[0]*G[1]], [Q[0]^3, -
Q[0]*Q[1]]]).solve_right(vector(F, [-(G[1]^3+1),-(Q[1]^3+1)]))
assert a * G[0]^3 + G[1]^3 + 1 == d * G[0] * G[1]
assert a * Q[0]^3 + Q[1]^3 + 1 == d * Q[0] * Q[1]

a0 = 1
a1 = - 3 * (d/3) / (a - (d/3) * (d/3) * (d/3))
a3 = - 9 / ((a - (d/3) * (d/3) * (d/3)) * (a - (d/3) * (d/3) * (d/3)))
a2 = - 9 * (d/3) * (d/3) / ((a - (d/3) * (d/3) * (d/3)) * (a - (d/3) *
(d/3) * (d/3)))
a4 = - 27 * (d/3) / ((a - (d/3) * (d/3) * (d/3)) * (a - (d/3) * (d/3) *
(d/3)) * (a - (d/3) * (d/3) * (d/3)))
a6 = - 27 / ((a - (d/3) * (d/3) * (d/3)) * (a - (d/3) * (d/3) * (d/3)) * (a
- (d/3) * (d/3) * (d/3)) * (a - (d/3) * (d/3) * (d/3)))
E = EllipticCurve(F, [a1, a2, a3, a4, a6])

x, y = F(G[0]), F(G[1])
u = (-3 / (a - d * d * d/27)) * x / (d * x/3 - (-y) + 1)
v = (-9 / ((a - d * d * d/27) * (a - d * d * d/27))) * (-y) / (d * x/3 - (-
y) + 1)
```

```
G = E(u, v)

x, y = F(Q[0]), F(Q[1])
u = (-3 / (a - d * d * d/27)) * x / (d * x/3 - (-y) + 1)
v = (-9 / ((a - d * d * d/27) * (a - d * d * d/27))) * (-y) / (d * x/3 - (-
y) + 1)
Q = E(u, v)

x = Q.log(G)

from Crypto.Cipher import AES
import hashlib
key = hashlib.sha256(str(x).encode()).digest()
cipher = AES.new(key, AES.MODE_ECB)
print(cipher.decrypt(ciphertext))
```

SPiCa

论文：A Polynomial-Time Algorithm for Solving the Hidden Subset Sum Problem

给定的数组 h（长度 m）的每个元素都是数组 x（长度 n）的随机子集之和。

直接建格子，用 LLL 就可以得到这 m 个子集间的 $m-n$ 组线性关系。根据 flag 格式可以得到另外 n 组方程，求解得到 flag。

```
from Crypto.Util.number import long_to_bytes, bytes_to_long

n, m, p = 70, 247,
247277048012919122688351297363409775675698657843668825666817599178436476580602 3
a = eval(open("data.txt", "r").readlines()[2])

M = matrix(ZZ, m+1, m+1)
for i in range(m):
    M[i, i] = 1
    M[i, m] = a[i] << 24
M[m, m] = p << 24
M = M.LLL(delta=1-1e-6)
M = M[:-1,:-1]
M = M[:m-n]

lhs = []
rhs = []
for row in M:
    lhs.append(row)
    rhs.append(0)

prefix = list(map(int, bin(bytes_to_long(b"hgame{"))[2:]))
for j in range(len(prefix)):
    lhs.append([0] * j + [1] + [0] * (m - j - 1))
    rhs.append(prefix[j])
```

```
for j in range(len(prefix), m):
    if (j - len(prefix)) % 8 == 0:
        lhs.append([0] * j + [1] + [0] * (m - j - 1))
        rhs.append(0)

sol = matrix(lhs).solve_right(vector(rhs))
sol = "".join(map(str, sol))
print(long_to_bytes(int(sol, 2)))
```

Misc (3/5)

Invest in hints

搜索得到至少需要开 5 个提示才能集齐 flag 的所有字符，方案不唯一。

```
hints = """Hint 51:
000011001010011110100000000100100011101000000000000000000001101111000100
Hint 52:
011010001110110000000000101000100001001101100000000001001000111001100000
Hint 53:
101001000000010110001100010011010000100011010111010101100010000000000000
Hint 54:
000010100000010010000100110000100000001000010010110011100001011100000111
Hint 55:
011100101001001000000000000000000011010110011000001111000101100000001000
Hint 56:
011101000010010000100101111011110111010010001000100110010000100111000000
Hint 57:
100001010100000000011000001100101001010110100000110110010001000100011000
Hint 58:
000001111101000001001000001100100100000110000110000101000001101110100000
Hint 59:
010011010010010000000001001001110100000000000000101100010001000001010101
Hint 60:
100100101001100110111000100110011001001000001110010010101001000100001111
Hint 61:
010010001000110000001000000000011010001110001000000101100001000100010100
Hint 62:
001010000100001110001011100000100010000000001000111100010001101001001101
Hint 63:
010000101110100000000101000010100010110001001000100000000000000001000000
Hint 64:
011101101100110000000010000011000000010000000000000111000000010000010001
Hint 65:
011000000000110001100000000100010000000000011001100000110010001011010000
Hint 66:
011100110010001010011000010110000110100000011000101000000110100000001000
Hint 67:
001110110000110000001001001010001001001010000100011001110010001000001000
Hint 68:
```

```python
01000110010101011100110101110010001111100011010000000101010100000010010
Hint 69:
11111010111000110100010000000010001101111010011010001100000011000001001
Hint 70:
00000010110101100100100011001011011001100000100010011111000011000001101
Hint 71:
00001100001110101000010111001100011100100010011100001010000000001000010
Hint 72:
01100000000011001001011100000101000110111000101100010101111000001010100
Hint 73:
00001000001010010000001101010110110000110111011011100101011110010110000
Hint 74:
01010010100000000111011110001000010110100001000111001101010100000010000
Hint 75:
11010000011000010100001010000111011010100001111010100100100000111110110""".spli

a = [int(hint.split()[-1], 2) for hint in hints]

def dfs(depth, known, selected):
    if depth == len(a) or len(selected) == 5:
        if known == (1 << 71) - 1:
            print(selected)
        return
    dfs(depth + 1, known, selected)
    if known | a[depth] != known:
        dfs(depth + 1, known | a[depth], selected + [depth])

# dfs(0, ((1 << 6) - 1) | (1 << 70), [])
dfs(0, 0, [])
```

我选择了开提示 $[6, 16, 17, 18, 24]$（下标从 0 开始），然后手动拼 flag。

```plain
***me***g***k***9**Ci*Lr*****K*Wy*A*q**i*9**hN*****8r********E*L*m****}
06
***m****g***k**o99**i7***g****a*y**z**t***1*h**C**u******AD****Ld*Ha4**
16
*g**e******M*f*o*9*******g*SC***y2Azq***6**DhN*C*xu**R2m*A*5*E**dm***2*
17
h**m*****5Y******9A***L*0g**C*aWy2*zq***6********x***R*mC***LEw*d*Ha42}
18
*ga*e{Aug*****3**9**i*L*0gQS****y*A*q3*i69****l*b****R*m****LE*****a*2}
24
hgame{Aug5YMkf3o99ACi7Lr0gQSCKaWy2Azq3ti691DhNlCbxu8rR2mCAD5LEwLdmHa42}
```

Level 729 易画行

`transfer.ts` 在 sepolia 链上交易了 NFT，搜索给定地址 得到 NFT 所有者地址，查看 其创建 NFT 的交易 的字符串。

得到的字符串 `ipfs://QmUusCYT8GTNgbDk5WAHZsHmHSxqcxuHov94inyFcpPqM6` 是一个 IPFS 地址，在网页访问，得到 flag。

Computer cleaner plus

尝试 `ps -a` 查看进程，但是即使 sudo 也会得到 permission denied。这是怎么回事呢?

`cd /bin` 后 `ls -lt` 查看，发现 `ps` 的文件大小和修改时间都不对。 `cat ps` 得到恶意文件名称 `B4ck_D0_oR.elf` 。

Lost Disks

TODO

串行调试模式

TODO

Reverse (5/5)

Signin

程序有反调试，如果断点对应的寄存器值不为 0 则报错。

程序生成了 256 个伪随机数，然后用这些数和程序自身的内容生成了 4 个 32 位整数作为密钥。可以直接 clone 这部分的逻辑生成相同的密钥。

然后程序用密钥加密了 flag，算法是某种魔改的 Feistel 网络，每步都是可逆的。

```python
data = open("signin.exe", "rb").read()
data = data[0x278C:][:0x10000]

a = []
for i in range(256):
    x = i
    for j in range(8):
        if x & 1:
            x = (x >> 1) ^ 0xEDB88320
        else:
            x = x >> 1
    a.append(x)

b = []
for i in range(4):
    x = 0xFFFFFFFF
    for c in data[0x4000 * i: 0x4000 * (i + 1)]:
        x = a[(c ^ x) & 0xFF] ^ (x >> 8)
    b.append(0xFFFFFFFF ^ x)
```

```python
c = [0, 0, 0, 0]

flag = bytes.fromhex("""23 EA 50 30 00 4C 51 47 EE 9C 76 2B D5 E6 94 17 ED
2B E4 B3 CB 36 D5 61 C0 C2 A0 7C FE 67 D7 5E AF E0 79 C5""")
flag = [int.from_bytes(flag[i: i + 4], "little") for i in range(0, 36, 4)]
for r in range(1, 12):
    for i in reversed(range(9)):
        lst = flag[(i - 1) % 9]
        nxt = flag[(i + 1) % 9]
        x = ((lst ^ b[i & 3]) + nxt) ^ (((lst << 4) ^ (nxt >> 3)) + ((nxt << 2) ^ (lst >> 5)))
        flag[i] = (flag[i] - x) % (2 ** 32)

print(b"".join([int.to_bytes(x, 4, "little") for x in flag]))
```

Mysterious signals

下发文件的 serve 是服务端，app-release 是客户端，两者只需逆向其一。

我选择了逆向 client，它发送的 JSON 请求数据带有用户名（`"username": "admin"`）和文件名（`"filename": "hello"`），HTTP 头中有签名（`sign: <signature>`）。字段的名称都用异或混淆过。

签名是用户名和文件名拼接后加密的结果，加密方式是 S 盒替换加 XXTEA。

发送请求后得到一个神秘的十六进制字符串，猜测它是签名得到的密文。编写解密算法得到原文，即为 flag。

```python
def decrypt(ciphertext):
    key = []
    for i in range(4):
        for j in range(4):
            for k in range(16):
                key.append((i * j * k + (b"e7c10e42b7a68e14"[k] ^ (0x11223344 >> (8 * j)))) & 0xFF)
    key = bytes(key)
    key = [int.from_bytes(key[i:i+4], "little") for i in range(0, len(key), 4)]

    ciphertext = bytes.fromhex(ciphertext)
    ciphertext = [int.from_bytes(ciphertext[i:i+4], "little") for i in range(0, len(ciphertext), 4)]
    plaintext = ciphertext.copy()
    for i in range(0, len(plaintext), 2):
        x, y = ciphertext[i], ciphertext[i + 1]
        k = (-1640531527 * 32) % (2 ** 32)
        for j in reversed(range(32)):
            k += 1640531527
            k %= 2 ** 32
            y -= (x ^ ((key[2 * j + 1] + k) % (2 ** 32)) ^ (x >> 5) ^ (x << 4)) % (2 ** 32)
            y %= 2 ** 32
            x -= (y ^ ((key[2 * j + 0] + k) % (2 ** 32)) ^ (y >> 3) ^ (y <<
2)) % (2 ** 32)
            x %= 2 ** 32
        plaintext[i], plaintext[i + 1] = x, y

    message = b"".join([int.to_bytes(x, 4, "little") for x in plaintext])

    sbox = bytes.fromhex("""63 7C 77 7B F2 6B 6F C5 30 01 67 2B FE D7 AB 76
CA 82 C9 7D FA 59 47 F0 AD D4 A2 AF 9C A4 72 C0 B7 FD 93 26 36 3F F7 CC 34
A5 E5 F1 71 D8 31 15 04 C7 23 C3 18 96 05 9A 07 12 80 E2 EB 27 B2 75 09 83
2C 1A 1B 6E 5A A0 52 3B D6 B3 29 E3 2F 84 53 D1 00 ED 20 FC B1 5B 6A CB BE
39 4A 4C 58 CF D0 EF AA FB 43 4D 33 85 45 F9 02 7F 50 3C 9F A8 51 A3 40 8F
92 9D 38 F5 BC B6 DA 21 10 FF F3 D2 CD 0C 13 EC 5F 97 44 17 C4 A7 7E 3D 64
5D 19 73 60 81 4F DC 22 2A 90 88 46 EE B8 14 DE 5E 0B DB E0 32 3A 0A 49 06
24 5C C2 D3 AC 62 91 95 E4 79 E7 C8 37 6D 8D D5 4E A9 6C 56 F4 EA 65 7A AE
08 BA 78 25 2E 1C A6 B4 C6 E8 DD 74 1F 4B BD 8B 8A 70 3E B5 66 48 03 F6 0E
61 35 57 B9 86 C1 1D 9E E1 F8 98 11 69 D9 8E 94 9B 1E 87 E9 CE 55 28 DF 8C
A1 89 0D BF E6 42 68 41 99 2D 0F B0 54 BB 16""")
    inv_sbox = [sbox.index(i) for i in range(256)]
    message = bytes(inv_sbox[c] for c in message)

    return message

print(decrypt("4b181fd6f8b852a9e23a4a7776e5f6905b71341af8f194a5db07d2902d265540
```

Fast and frustrating

由于 AOT，代码中的字符串常量无法直接通过 xref 得到了。所以我使用 `strings` 命令，发现两个可疑的 base64 字符串。

其中第一个字符串长 1348 字节，base64 解码后开头是 `1f 8b 08 00`，搜索得到它是 gzip 压缩头。于是解码，得到一串 json，有矩阵 `mat_a` 和向量 `vec_b`，猜测是解线性方程。方程的解在 ASCII 范围内，转字符串得 `CompressedEmbeddedResources`。

```
data =
"H4sIABh9j2cC/21Wy47bMAz8lWDPESBS7/7KYrHYFj32VvRS9N+rGVKynQSIZUtiOCI5JPX37dfX78

import base64
data = base64.b64decode(data)

import gzip
data = gzip.decompress(data).decode()

import json
data = json.loads(data)

import numpy as np
x = np.linalg.inv(data["mat_a"]) @ data["vec_b"]
print(bytes(list(map(round, x))))
```

第二个字符串在 base64 解码后长 48 字节，结合反汇编的 `Cryptography.SymmetricAlgorithm.DecryptCbc` 判定为 AES 在 CBC 模式下的密文。

根据反汇编代码，AES 密钥的生成方式是 `Cryptography.HKDF.DeriveKey`。调用 HKDF 时，使用的 hash 函数是 SHA256，ikm 未知，outputLength 未知，salt 是 NULL，info 未知。

经过一些猜测与尝试，ikm 是第一部分解出的 `CompressedEmbeddedResources`，info 是 `HGAME2025`（它在 `strings` 输出结果中，紧随着之前的 base64 字符串），生成的密钥长度是 32（而不是常见的 16）。

这时，如果认为 IV 是密文的前 16 个字节，已经能解出 flag 的后半部分了。尽管是缺少前半部分，理论上能根据标题猜出 flag 的，但是我没有成功 :(

继续猜测，IV 也是用相同的 HKDF 生成的后 16 字节，就能解出完整的 flag 了。

```python
from Crypto.Protocol.KDF import HKDF
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
import base64

key_iv = HKDF(b"CompressedEmbeddedResources", 48, None, SHA256, 1,
b"HGAME2025")
key, iv = key_iv[:32], key_iv[-16:]
ct =
base64.b64decode("GFxmVucV6MVUXiWCMAnWpyvzXoLdHc5CmFeim+JjUBszB8HFX8Ku8NMc201AG
validate=True)
print(AES.new(key, AES.MODE_CBC, iv).decrypt(ct))
```

Middlemen

代码的核心都在 `libmiddlemen.so` 里，其中有三个重要的函数：

函数 `middlemen(JNIEnv *env, jobject this, jstring FLAG)`：

- 将 UUID 格式的 FLAG 转为 16 字节
- `syscall` 调用了 `getpid`，参数是 FLAG（4 个 DWORD，分别存在寄存器 `x1, x2, x3, x4` 中）和 `0x221221`，如果结果大于 `0x1000000000000` 则通过

函数 `_INIT_0()`（在初始化时被调用）：

- `prctl(PR_SET_NO_NEW_PRIVS, ...)` 和 `prctl(PR_SET_SECCOMP, ...)` 通过 SECCOMP 禁止了一些 `syscall` 的调用
- `sigaction(SIGSYS, ...)` 设置了 `syscall` 函数被禁止时的处理函数 `handler`

函数 `handler(int sig, siginfo_t *info, void *ucontext)`：

- 初始化密钥 `"Sevenlikeseccmop"`，将密钥和 8 字节的字符串（`ucontext->uc_mcontext->regs[3]`，也就是寄存器 `x3, x4` 的值）循环异或

- 用 AES 加密了 16 字节的字符串（`ucontext->uc_mcontext->regs[1]`，也就是寄存器 `x1, x2, x3, x4` 的值）
- 将 AES 密文和结果对比，如果相同则返回 `0x1145141919810`，否则返回正常的 `getpid` 值

处理 SECCOMP，发现只有特定的 flag 会导致 pid 调用被禁止：

```
plain
$ seccomp-tools disasm rules.bpf
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x26 0xc00000b7  if (A != ARCH_AARCH64) goto 0040
 0002: 0x20 0x00 0x00 0x00000020  A = args[2]
 0003: 0x02 0x00 0x00 0x00000000  mem[0] = A
 0004: 0x20 0x00 0x00 0x00000028  A = args[3]
 0005: 0x02 0x00 0x00 0x00000001  mem[1] = A
 0006: 0x64 0x00 0x00 0x00000004  A <<= 4
 0007: 0x04 0x00 0x00 0x65766573  A += 0x65766573
 0008: 0x02 0x00 0x00 0x00000002  mem[2] = A
 0009: 0x60 0x00 0x00 0x00000001  A = mem[1]
 0010: 0x07 0x00 0x00 0x00000000  X = A
 0011: 0x00 0x00 0x00 0x22122122  A = 571613474
 0012: 0x0c 0x00 0x00 0x00000000  A += X
 0013: 0x07 0x00 0x00 0x00000000  X = A
 0014: 0x60 0x00 0x00 0x00000002  A = mem[2]
 0015: 0xac 0x00 0x00 0x00000000  A ^= X
 0016: 0x07 0x00 0x00 0x00000000  X = A
 0017: 0x60 0x00 0x00 0x00000000  A = mem[0]
 0018: 0x0c 0x00 0x00 0x00000000  A += X
 0019: 0x15 0x00 0x14 0x93cd6340  if (A != 2479711040) goto 0040
 0020: 0x02 0x00 0x00 0x00000000  mem[0] = A
 0021: 0x74 0x00 0x00 0x00000005  A >>= 5
 0022: 0x04 0x00 0x00 0x6e6e6e6e  A += 0x6e6e6e6e
 0023: 0x02 0x00 0x00 0x00000002  mem[2] = A
 0024: 0x60 0x00 0x00 0x00000000  A = mem[0]
 0025: 0x07 0x00 0x00 0x00000000  X = A
 0026: 0x00 0x00 0x00 0x22122122  A = 571613474
 0027: 0x0c 0x00 0x00 0x00000000  A += X
 0028: 0x07 0x00 0x00 0x00000000  X = A
 0029: 0x60 0x00 0x00 0x00000002  A = mem[2]
 0030: 0xac 0x00 0x00 0x00000000  A ^= X
 0031: 0x07 0x00 0x00 0x00000000  X = A
 0032: 0x60 0x00 0x00 0x00000001  A = mem[1]
 0033: 0x0c 0x00 0x00 0x00000000  A += X
 0034: 0x15 0x00 0x05 0xb5f40d3f  if (A != 3052670271) goto 0040
 0035: 0x20 0x00 0x00 0x00000000  A = sys_number
 0036: 0x15 0x00 0x03 0x000000ac  if (A != aarch64.getpid) goto 0040
 0037: 0x20 0x00 0x00 0x00000030  A = args[4]
 0038: 0x15 0x00 0x01 0x00221221  if (A != 0x221221) goto 0040
 0039: 0x06 0x00 0x00 0x00030000  return TRAP
```

```
    0040: 0x06 0x00 0x00 0x7fff0000  return ALLOW
    0041: 0x06 0x00 0x00 0x00050000  return ERRNO(0)
```

用 z3 求解，只有形如 `getpid(?, ?, 0x4d19d88c, 0xef20af55, 0x221221)` 的 `syscall` 会被 `handler` 处理，由此得到 flag 的后 8 字节：

```python
from z3 import *

s = Solver()
u = BitVec("u", 32)
v = BitVec("v", 32)

A = u + (((v << 4) + 0x65766573) ^ (v + 571613474))
B = v + ((LShR(A, 5) + 0x6e6e6e6e) ^ (A + 571613474))
s.add(A == 2479711040)
s.add(B == 3052670271)

assert s.check(), "No solution"
m = s.model()
u, v = int(str(m[u])), int(str(m[v]))
print(u.to_bytes(4, "little").hex(), v.to_bytes(4, "little").hex())
```

然后解 AES 得到完整的 flag：

```python
from Crypto.Cipher import AES
from pwn import xor

key = xor(b"Sevenlikeseccmop", bytes.fromhex("8cd8194d 55af20ef")) # eccmop?
ct = bytes.fromhex("B7 62 40 6A EB 70 B9 ED 81 71 DB 9D AC 82 FF 94")
pt = AES.new(key, AES.MODE_ECB).decrypt(ct)
print("hgame{%s-%s-%s-%s-%s}" % (pt[0:4].hex(), pt[4:6].hex(), pt[6:8].hex(), pt[8:10].hex(), pt[10:16].hex()))
```

Nop'd

`./launcher` 调用了 `fork` 函数，子进程 `execv` 执行了 `./game`，而父进程在 `return 0` 后，（因为 `atexit` 函数）追踪子进程并且改变了子进程中 `nop` 指令的行为。

`./launcher` 同时改变了 `./game` 的 `syscall` 函数的行为；改写后，`syscall` 调用的参数分别存在寄存器 `r9, r8, ...` 内。`rbx` 决定了函数的类型：

- `0`：真 · `nop`
- `1`：`read`
- `2`：`puts`
- `3`：`chacha20` 的 $\frac{1}{4}$ 轮
- `4`：`chacha20` 的矩阵移位函数
- `5`：`chacha20` 初始化 `iv` （`expand 32-byte k` 是该算法的标志性常数）
- `6`：`chacha20` 最后一步的加法
- `7`：`memcmp`

- `8`：`return 0x61C88646;`
- `9`：`rip += 128;`

运行 `strace ./launcher ./game` 可以得到 `./launcher` 对函数的调用，由此直接得到 `ChaCha20` 生成密钥流：

```
stream = b""
for line in """ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30c0, 0x465687f32a5b694a) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30c8, 0x16d67365c67407f3) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30d0, 0x6db3760398d9fe45) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30d8, 0xa4b0bc4cf796e050) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30e0, 0xa70838d893dcf2ea) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30e8, 0xd16e84873b6bde23) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30f0, 0x8ee3f562ac34d04) = 0
ptrace(PTRACE_POKEDATA, 4662, 0x59fe020f30f8, 0xca48bc6be676d8a3) = 0""".splitlines():
    c = line.split(", ")[-1]
    c = c[:c.index(")")]
    stream += int(c, 16).to_bytes(8, "little")
```

或者用 `pycryptodome` 里的函数：

```
from Crypto.Cipher import ChaCha20
stream = ChaCha20.new(key = b"It's all written in the Book of ", nonce = b"What's your ").encrypt(bytes(64))
```

最后逆向 `./game` 在 `0x2247` 处的异或密钥流和对 flag 的判定。

```
ct = bytes.fromhex("""64 6A 50 17 81 7D 6F 1A 87 B1 A4 00 09 03 F8 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""")
ct = bytes([0x46]) + ct
b = bytes([ct[i - 1] ^ ct[i] for i in range(1, len(ct))])
print(bytes([i ^ j for i, j in zip(b, stream)]))
```

Pwn (0/3)

Signin2Heap

TODO

Where is the vulnerability

TODO

Hit list

TODO

Web (0/4)

Level 21096 HoneyPot / Level 21096 HoneyPot_Revenge

TODO

Level 60 SignInJava

TODO

Level 111 不存在的车厢

TODO

Level 257 日落的紫罗兰

TODO