DASCTF二进制专项-Reverse

DASCTF二进制专项-Reverse

被队里大佬的做题速度震惊到了😢

careful

对`gethostname`函数交叉引用会发现其被Hook

```
char *v0; // edx
DWORD flOldProtect; // [esp+4h] [ebp-4h] BYREF

VirtualProtect(lpAddress, 5u, 0x40u, &flOldProtect);
v0 = (char *)lpAddress;
dword_504374 = *(_DWORD *)lpAddress;
byte_504378 = *((_BYTE *)lpAddress + 4);
*(_DWORD *)lpAddress = *(_DWORD *)sub_5010C0;
*v0 = -23;
*(_DWORD *)(v0 + 1) = (char *)sub_5010C0 - (char *)gethostbyname - 5;
return VirtualProtect(v0, 5u, flOldProtect, &flOldProtect);
}
```

下面是实际调用的函数，打个断点就能获取到flag了

```
VirtualProtect(lpAddress, 5u, 0x40u, &flOldProtect);
v1 = lpAddress;
*(_DWORD *)lpAddress = dword_504374;
v1[4] = byte_504378;
VirtualProtect(v1, 5u, flOldProtect, &flOldProtect);
name[0] = *a1 ^ 0x3D;
name[1] = a1[1] ^ 0x10;
name[2] = a1[2] ^ 0x1F;
name[3] = a1[3] ^ 0x17;
name[4] = a1[4] ^ 0x30;
name[5] = a1[5] ^ 0x2C;
name[6] = a1[6] ^ 0xB;
name[7] = a1[7];
name[8] = a1[8] ^ 0x35;
name[9] = a1[9] ^ 0x60;
name[10] = a1[10] ^ 0x16;
name[11] = a1[11] ^ 0x2C;
name[12] = a1[12] ^ 0x51;
name[13] = a1[13] ^ 0x43;
name[14] = a1[14] ^ 8;
name[15] = a1[15] ^ 0x45;
name[16] = a1[16] ^ 0x57;
name[17] = a1[17];
name[18] = a1[18];
name[19] = a1[19];
name[20] = a1[20];
name[21] = a1[21];
return gethostbyname(name);
```

babyRe

考点

多线程、RC4、反调试、花指令

解题

获取资源文件，异或解密，之后创建新线程，最后再创建一个线程来进行比较。通过命令行参数传入flag进而加密比对

```
v8 = OpenProcess(0x1FFFFFu, 0, dwProcessId);
if ( a1 != 2 )
  exit(0);
for ( j = 0; ; ++j )
{
  v19 = j;
  if ( j >= strlen(*(const char **)(a2 + 8)) )
    break;
  Parameter[j] = *(_BYTE *)(*(_QWORD *)(a2 + 8) + j);
}
hResInfo = FindResourceW(0i64, (LPCWSTR)0x65, L"cod");
LODWORD(Size) = SizeofResource(0i64, hResInfo);
hResData = LoadResource(0i64, hResInfo);
Src = LockResource(hResData);
v15 = malloc((unsigned int)Size);
memcpy(v15, Src, (unsigned int)Size);
sub_7FF732AC1087((__int64)v15);
v16 = off_7FF732ACFC30(v8, 0i64, 874i64, 4096i64, 64);
off_7FF732ACFC28(v8, v16, v15, 874i64, 0i64);
v17 = off_7FF732ACFC18(v8, 0i64, 0i64, v16, Parameter, 0, 0i64);
Sleep(500u);
for ( k = 0; k < 44; ++k )
  ;
CreateThread(0i64, 0i64, StartAddress, Parameter, 0, 0i64);
Sleep(0x190u);
sub_7FF732AC1253(v5, &unk_7FF732ACBEE0);
return 0i64;
```

根据交叉引用发现其根据调试状态来修改异或的key，下图为patch之后的

```
{
  sub_7FF732AC12A8(&unk_7FF732AD40F4)
  if ( !IsDebuggerPresent() )
    byte_7FF732ACF000[3] = 36;
  return 0i64;
}
```

解密之后代码中的花指令，nop掉之后反编译即可

```
F0022 rep stosd
F0024 jz      short near ptr loc_284101F0028+1
F0024
F0026 jnz     short near ptr loc_284101F0028+1
F0026
F0028
F0028 loc_284101F0028:                        ; CODE X
F0028                                         ; debug0
F0028 jmp     near ptr 28420648D75h
```

```
01F0115 mov     dword ptr [rbp+134h], 0
01F011F call    $+5
01F011F
01F0124 add     byte ptr [rsp], 5
01F0128 retn
01F0128
```

加密过程

```
48  v8 = 0;
49  sub_284101F01EE();
50  v7 = v8;
51  for ( k = 0; ; ++k )
52  {
53    result = v11;
54    if ( k >= v11 )
55      break;
56    v7 = (v8 + v7) % 0x100;
57    v8 = ((unsigned __int8)v5[v7] + v8) % 0x100;
58    v9 = (unsigned __int8)v5[v7];
59    v5[v7] = v5[v8];
60    v5[v8] = v9;
61    v13 = (unsigned __int8)v5[((unsigned __int8)v5[v7] + v8 + (unsigned __int8)v5[v8]) % 0x100];
62    a1[k] ^= v13;
63    v14 = k;
64    a1[k] += k % 0xD;
65  }
66  return result;
67 }
```

构造数据提取密钥流，然后恢复密文即可

```
1         char raw[] = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
2         unsigned char data[84] = {
3         0xD2, 0x0E, 0x02, 0xD2, 0x5F, 0xF2, 0x48, 0x59, 0xE9, 0xBF, 0xFC, 0x82
4         0x49, 0x37, 0xD6, 0x88, 0x30, 0xED, 0xA4, 0x02, 0x74, 0xB6, 0x97, 0x07
5         0xEA, 0x5E, 0xF1, 0x42, 0xAD, 0xBC, 0x71, 0x32, 0xA4, 0x81, 0xAD, 0x4C
6         0xAA, 0x94, 0x64, 0x3B, 0xAE, 0xDF, 0xDE, 0x38, 0xE9, 0x2C, 0xC2, 0x5C
7         0x1B, 0x98, 0x12, 0xCB, 0xE8, 0x29, 0xC1, 0xFD, 0xD9, 0x84, 0x05, 0xE4
8         0x44, 0x81, 0x07, 0x57
9         };
10        for (int i = 0; i < strlen(raw); ++i) {
11                printf("%c", (code[i] - (i % 0xD)) ^ (data[i] - (i % 0xD)) ^ r
12        }
```

`Tips`:因为涉及多线程，所以调试的时候会出现几个线程来回跳的情况，所以可以通过ida的`Thread`窗口对线程挂起来调试

unsym

这题`go_parser`可以恢复符号，虽然其最多支持到go1.18，但是go1.20有一部分结构和go1.18的`firstmoduledata`结构是相似的

步骤如下：

首先go编译的exe没有办法直接通过`gopclntab`段来定位特征码，可以通过搜索字符串`go:buildid`搭配交叉引用来定位，需要将0xFFFFFFF1改为0xFFFFFFF0(或者直接改脚本中的特征码)

```
.rdata:00000000004EB6F0 00 00 00 00 00 00 00 00 00 00+align 20h
.rdata:00000000004EB700 F0 FF FF FF                   runtime_symtab dd 0FFFFFFF0h
.rdata:00000000004EB700
.rdata:00000000004EB704 00 00                         dw 0
.rdata:00000000004EB706 01                            db    1
```

之后直接运行脚本即可恢复符号

这一段是对key进行RSA加密(65537以及exp函数可以大致猜出，当时被卡住了😭)，通过`yafu`可以分`n`

```
  v6 = v44;
}
v45 = 65537LL;
v76[0] = 0;
v77 = &v45;
v78 = 1LL;
v79 = 1LL;
v73 = 0;
v74 = 0LL;
v75 = v4;
v70 = 0;
v71 = 0LL;
v72 = v4;
math_big__Int_SetString(16LL, v8, (__int64)&v45, v8);
math_big__Int_exp(v67, 0LL, v14, (__int64)v76);
v15 = v73;
v16 = 16LL;
v17 = math_big_nat_itoa(v73, 16LL);
v51 = runtime_slicebytetostring();
runtime_makeslice();
```

写脚本解出key：`E@sy_RSA_enc7ypt`

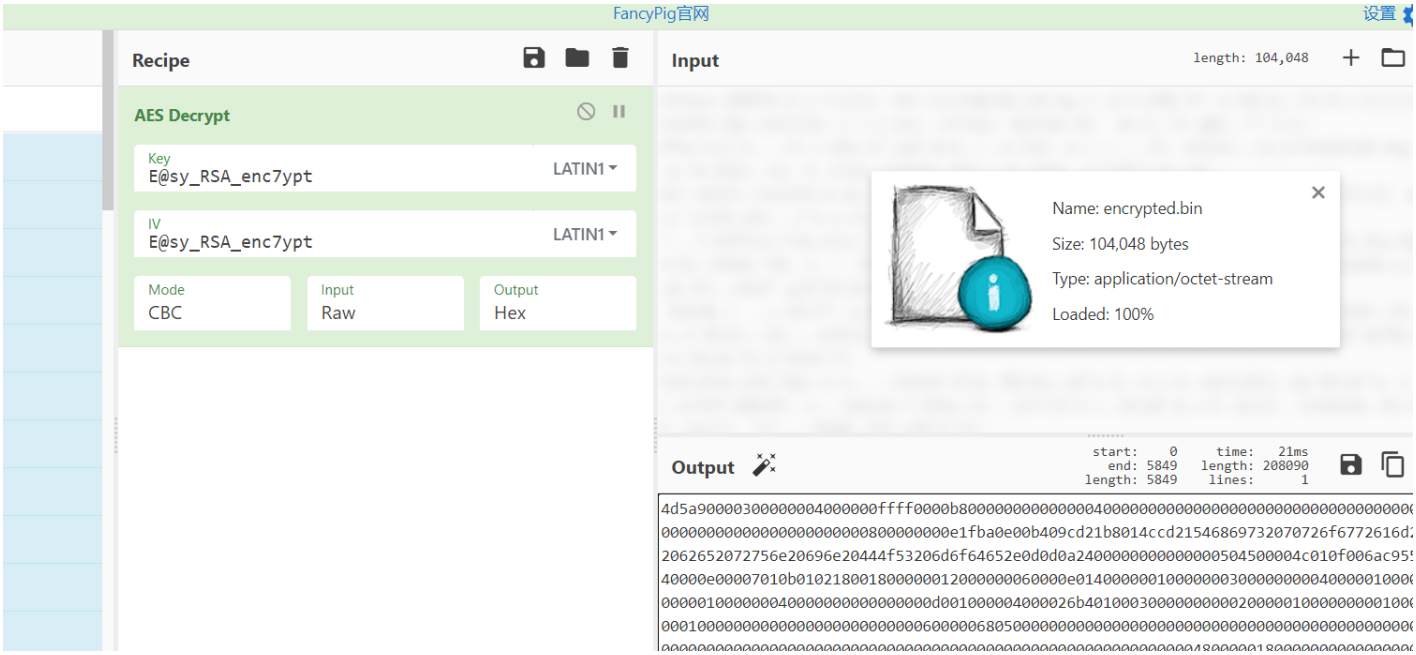
```python
import libnum
from Crypto.Util.number import long_to_bytes, bytes_to_long

n = 0x1D884D54D21694CCD120F145C8344B729B301E782C69A8F3073325B9C5
c = 0xFAD53CE897D2C26F8CAD910417FBDD1F0F9A18F6C1748FACA10299DC8
```

```python
q = 211549048872157489492804106164784 23
p = 376363184577451672341408081301567 39
e = 65537
d = libnum.invmod(e, (p - 1) * (q - 1))
m = pow(c, d, n)
string = long_to_bytes(m)
print(string)
```

之后将key作为AES_KEY和AES_IV对message文件进行AES-CBC加密

```
193  if ( !v30 )
194    runtime_panicdivide();
195  v39 = v30 - 7 % v30;
196  runtime_makeslicecopy(v37);
197  for ( j = 7LL; j < v39 + 7; ++j )
198    *(_BYTE *)(v31 + j) = v39;
199  v57 = v31;
200  v33 = (*(__int64 (**)(void))(v58 + 24))();
201  if ( v33 > v41 )
202LABEL_23:
203    runtime_panicSliceAcap();
204  v34 = v55;
205  v40 = crypto_cipher_NewCBCEncrypter(v33);
206  v52 = v34;
207  v56 = runtime_makeslice();
208  ((void (*)(void))v40[4])();
209  if ( os_WriteFile() )
210  {
211    runtime_gopanic(v38);
212    goto LABEL_23;
213  }
```

解密文件，解密得到exe文件，运行即可得到flag


FancyPig官网
设置
Recipe
AES Decrypt
Key
E@sy_RSA_enc7ypt
LATIN1
IV
E@sy_RSA_enc7ypt
LATIN1
Mode
CBC
Input
Raw
Output
Hex
Input
length: 104,048
Name: encrypted.bin
Size: 104,048 bytes
Type: application/octet-stream
Loaded: 100%
Output
start: 0
end: 5849
length: 5849
time: 21ms
length: 208090
lines: 1
4d5a900003000000040000000ffff0000b8000000000000004000000000000000000000000000000000000000
0000000000000000000000000800000000e1fba0e00b409cd21b8014ccd21546869732070726f6772616d2
2062652072756e20696e20444f53206d6f64652e0d0d0a2400000000000000504500004c010f006ac955
40000e00007010b0102180018000000120000000600000e0140000001000000030000000040000010000
0000010000000400000000000000d0010000040000026b401000300000000020000010000000001000
0001000000000000000000000000060000068050000000000000000000000000000000000000000000

1