meow

首先看到main:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[32]; // [rsp+20h] [rbp-70h] BYREF
  char flag[80]; // [rsp+40h] [rbp-50h] BYREF

  sub_401E70(argc, argv, envp);
  sub_401604();
  printf("Gimme flag: ");
  scanf("%39s", flag);
  if ( strlen(flag) == 39 && (proc_hollowing((__int64)v4), pipe(flag), (unsigned int)check_flag(flag)) )
    puts("Correct ;-)");
  else
    puts("Wrong :-O");
  return 0;
}
```

接下來一個function一個function看:

先看第三個function，這邊看起來就是在做flag比對，但他的值明顯是被加密過的，不能直接拿來用，因此先去

```
__int64 __fastcall check_flag(__int64 a1)
{
  char v2[40]; // [rsp+0h] [rbp-30h]
  int i; // [rsp+28h] [rbp-8h]
  unsigned int v4; // [rsp+2Ch] [rbp-4h]

  v2[0] = 36;
  v2[1] = 29;
  v2[2] = 27;
  v2[3] = 49;
  v2[4] = 33;
  v2[5] = 11;
  v2[6] = 79;
  v2[7] = 15;
  v2[8] = -24;
  v2[9] = 80;
  v2[10] = 55;
  v2[11] = 91;
  v2[12] = 8;
  v2[13] = 64;
  v2[14] = 74;
  v2[15] = 8;
  v2[16] = 29;
  v2[17] = 17;
  v2[18] = 74;
  v2[19] = -72;
  v2[20] = 17;
  v2[21] = 103;
  v2[22] = 63;
  v2[23] = 103;
  v2[24] = 56;
  v2[25] = 20;
  v2[26] = 63;
  v2[27] = 25;
  v2[28] = 11;
  v2[29] = 84;
  v2[30] = -76;
  v2[31] = 9;
  v2[32] = 99;
  v2[33] = 18;
  v2[34] = 104;
  v2[35] = 42;
  v2[36] = 69;
  v2[37] = 83;
  v2[38] = 14;
  v4 = 1;
  for ( i = 0; i <= 38; ++i )
  {
    if ( *(_BYTE *)(a1 + i) != v2[i] )
      v4 = 0;
  }
  return v4;
}
```

回去看第一個function:

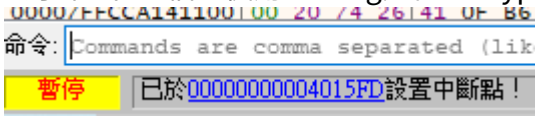
```
__int64 __fastcall proc_hollowing(__int64 a1)
{
  __int64 v1; // rax
  SIZE_T nSize; // [rsp+60h] [rbp-20h]
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+80h] [rbp+0h] BYREF
  struct _STARTUPINFOA StartupInfo; // [rsp+A0h] [rbp+20h] BYREF
  CHAR Filename[96]; // [rsp+110h] [rbp+90h] BYREF
  LPCONTEXT lpContext; // [rsp+170h] [rbp+F0h]
  DWORD64 v8; // [rsp+178h] [rbp+F8h]
  LPVOID v9; // [rsp+180h] [rbp+100h]
  BOOL v10; // [rsp+18Ch] [rbp+10Ch]
  __int64 v11; // [rsp+190h] [rbp+110h]
  LPVOID lpBaseAddress; // [rsp+198h] [rbp+118h]
  int v13; // [rsp+1A4h] [rbp+124h]
  struct_v14 *v14; // [rsp+1A8h] [rbp+128h]
  __int64 v15; // [rsp+1B0h] [rbp+130h]
  unsigned __int64 i; // [rsp+1B8h] [rbp+138h]

  memset(&StartupInfo, 0, sizeof(StartupInfo));
  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
  GetModuleFileNameA(0i64, Filename, 0x60u);
  CreateProcessA(0i64, Filename, 0i64, 0i64, 0, 4u, 0i64, 0i64, &StartupInfo, &ProcessInformation);
  v15 = sub_401693(ProcessInformation.hProcess);
  decrypt_exe();
  v14 = (struct_v14 *)malloc(0x58ui64);
  *(_QWORD *)&v14->gap0[24] = (char *)exe + exe[15];
  *(_QWORD *)&v14->gap2C[4] = (char *)&exe[66] + exe[15];
  v14->unsigned_int28 = *(unsigned __int16 *)(*(_QWORD *)&v14->gap0[24] + 6i64);
  v14->unsigned_int50 = *(_DWORD *)(*(_QWORD *)&v14->gap0[24] + 80i64);
  v13 = unk_40C980(ProcessInformation.hProcess, *(_QWORD *)(v15 + 16));
  lpBaseAddress = VirtualAllocEx(
                    ProcessInformation.hProcess,
                    *(LPVOID *)(v15 + 16),
                    v14->unsigned_int50,
                    0x3000u,
                    0x40u);
  v11 = (__int64)lpBaseAddress - *(_QWORD *)(*(_QWORD *)&v14->gap0[24] + 48i64);
  v10 = WriteProcessMemory(
          ProcessInformation.hProcess,
          lpBaseAddress,
          exe,
          *(unsigned int *)(*(_QWORD *)&v14->gap0[24] + 84i64),
          0i64);
  for ( i = 0i64; i < v14->unsigned_int28; ++i )
  {
    v1 = *(_QWORD *)&v14->gap2C[4] + 40 * i;
    nSize = *(_QWORD *)(v1 + 16);
    v9 = (char *)lpBaseAddress + (unsigned int)HIDWORD(*(_QWORD *)(v1 + 8));
    v10 = WriteProcessMemory(ProcessInformation.hProcess, v9, (char *)exe + HIDWORD(nSize), (unsigned int)nSize, 0i64);
  }
  v8 = (DWORD64)lpBaseAddress + *(unsigned int *)(*(_QWORD *)&v14->gap0[24] + 40i64);
  lpContext = (LPCONTEXT)malloc(0x4D0ui64);
  memset(lpContext, 0, sizeof(struct _CONTEXT));
  lpContext->ContextFlags = 1048578;
  GetThreadContext(ProcessInformation.hThread, lpContext);
  lpContext->Rcx = v8;
  SetThreadContext(ProcessInformation.hThread, lpContext);
  ResumeThread(ProcessInformation.hThread);
  return a1;
}
```

這邊主要是在做process hollowing，也就是把一個程式寫到記憶體中，讓他執行。
而那個程式的來源就在這個程式的0x404040的位置。
但他有一個function用來做XOR，也就是本來0x404040位置的資料是被加密過的。

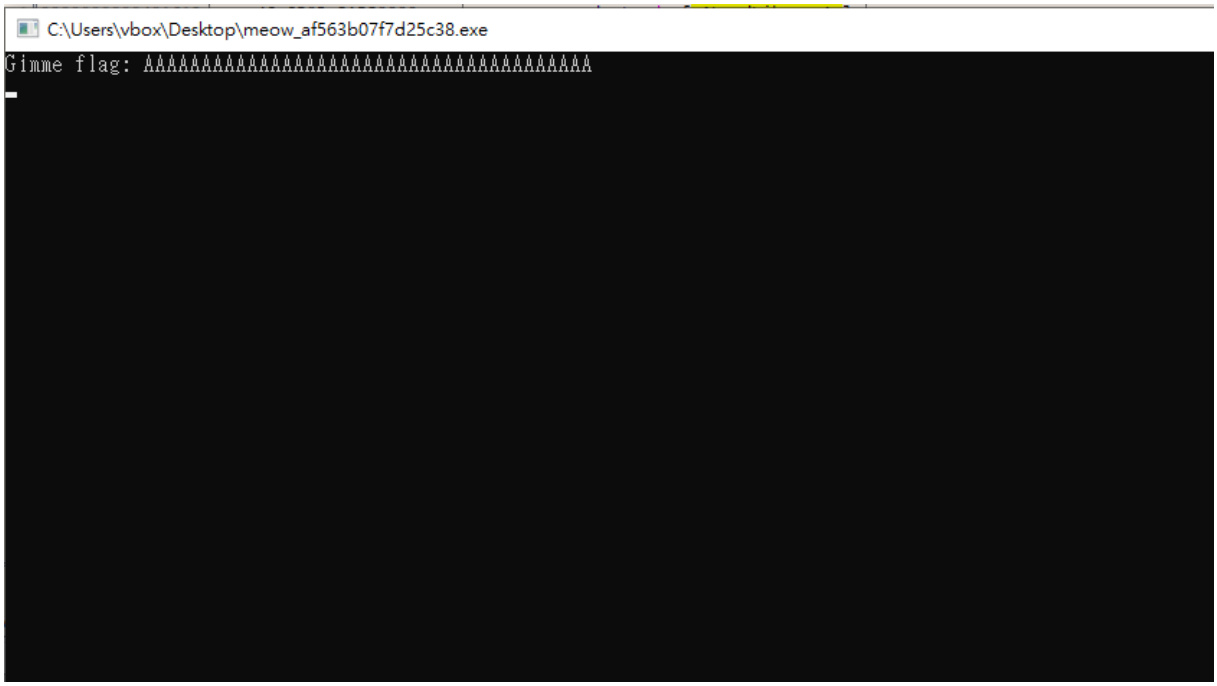
```
__int64 decrypt_exe()
{
  __int64 result; // rax
  unsigned int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i > 0x3FFF )
      break;
    *((_BYTE *)exe + (int)i) += (char)i % 7;
    *((_BYTE *)exe + (int)i) ^= byte_404020[i & 7];
  }
  return result;
}
```

dump

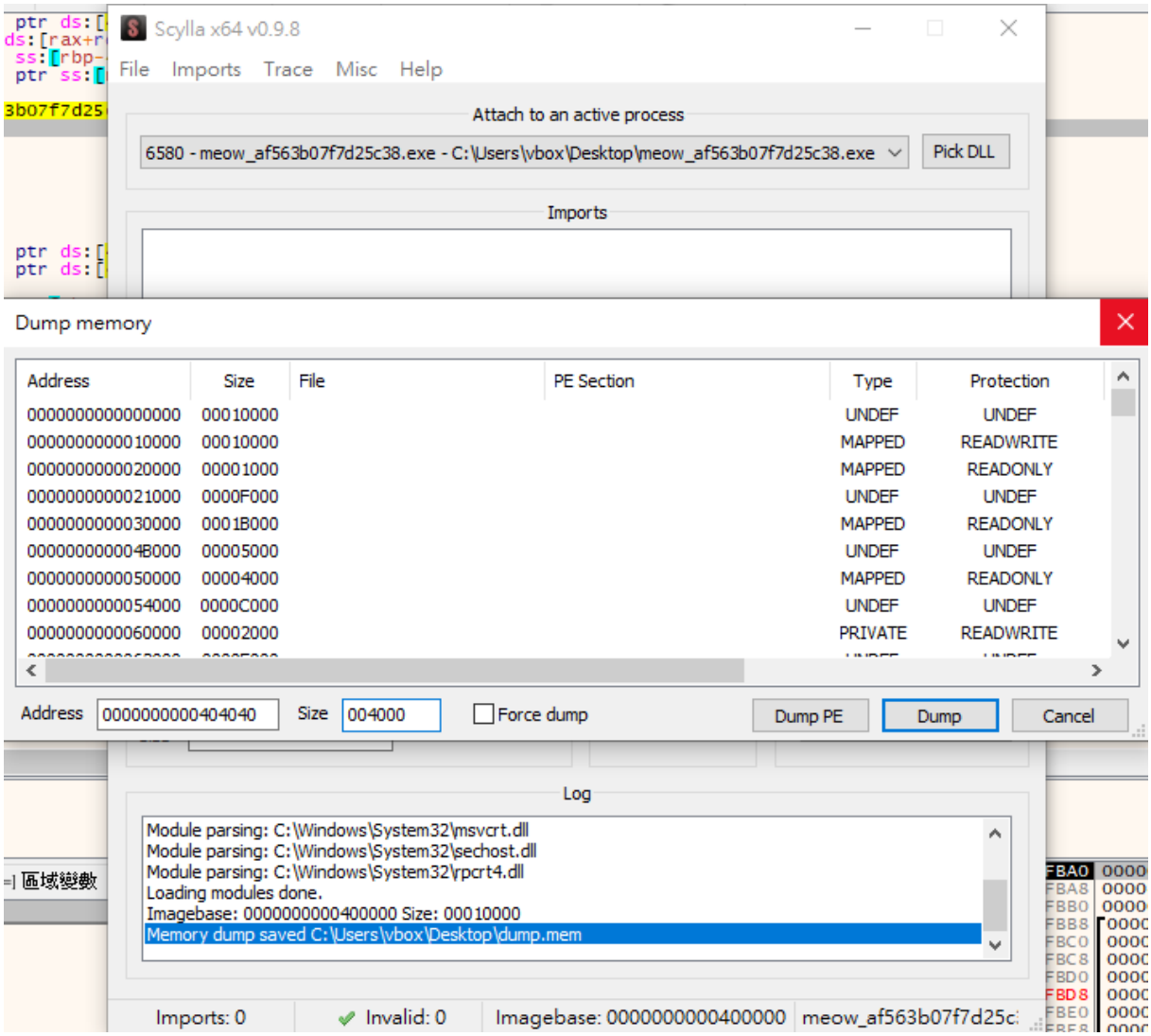
爲了取出exe檔，使用x64dbg先在decrypt完的地方設中斷點，再dump memory。



設好中斷點後，繼續執行，直到程式要求輸入flag爲止。
這邊需要輸入一個長度爲39的字串，因爲程式會先檢查flag長度，如果長度錯誤則後面都無法被執行到了。



抵達中斷點後，按下Scylla的dump memory功能，將memory dump出來。
起始位置在0x404040，長度則可以在decrypt_exe的function中看到，因此設成0x4000。



disassemble

dump出來後是一個新的PE執行檔，使用ida打開，看到main:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   sub_401820(argc, argv, envp);
4   sub_40161D();
5   return 0;
6 }
```

點進function查看:

```
__int64 sub_40161D()
{
  DWORD v0; // eax
  __int64 result; // rax
  DWORD NumberOfBytesWritten; // [rsp+48h] [rbp-28h] BYREF
  DWORD NumberOfBytesRead; // [rsp+4Ch] [rbp-24h] BYREF
  void *lpBuffer; // [rsp+50h] [rbp-20h]
  BOOL v5; // [rsp+5Ch] [rbp-14h]
  HANDLE hNamedPipe; // [rsp+60h] [rbp-10h]
  LPCSTR lpName; // [rsp+68h] [rbp-8h]

  lpName = "\\\\.\\pipe\\m30w";
  hNamedPipe = CreateNamedPipeA("\\\\.\\pipe\\m30w", 3u, 4u, 0xFFu, 0x100u, 0x100u, 0, 0i64);
  v5 = ConnectNamedPipe(hNamedPipe, 0i64);
  NumberOfBytesRead = 0;
  lpBuffer = malloc(0x100ui64);
  while ( !NumberOfBytesRead )
  {
    memset(lpBuffer, 0, 0x100ui64);
    v5 = ReadFile(hNamedPipe, lpBuffer, 0x100u, &NumberOfBytesRead, 0i64);
  }
  sub_401550((__int64)lpBuffer);
  for ( NumberOfBytesWritten = 0; ; v5 = WriteFile(hNamedPipe, lpBuffer, v0, &NumberOfBytesWritten, 0i64) )
  {
    result = NumberOfBytesWritten;
    if ( NumberOfBytesWritten )
      break;
    v0 = strlen((const char *)lpBuffer);
  }
  return result;
}
```

可以看的出來這邊是在跟原本的程式使用NamedPipe溝通，回去對應到原本程式的第二個function:

```
__int64 __fastcall pipe(void *a1)
{
  __int64 result; // rax
  DWORD v2; // eax
  DWORD NumberOfBytesRead; // [rsp+40h] [rbp-20h] BYREF
  DWORD NumberOfBytesWritten; // [rsp+44h] [rbp-1Ch] BYREF
  DWORD Mode; // [rsp+48h] [rbp-18h] BYREF
  HANDLE hNamedPipe; // [rsp+50h] [rbp-10h]
  LPCSTR lpFileName; // [rsp+58h] [rbp-8h]

  lpFileName = "\\\\.\\pipe\\m30w";
  Sleep(0x32u);
  while ( 1 )
  {
    hNamedPipe = CreateFileA(lpFileName, 0xC0000000, 0, 0i64, 3u, 0, 0i64);
    if ( hNamedPipe != (HANDLE)-1i64 )
      break;
    if ( GetLastError() != 231 )
      return 0xFFFFFFFFi64;
    if ( !WaitNamedPipeA(lpFileName, 0x4E20u) )
      return 0xFFFFFFFFi64;
  }
  Mode = 2;
  if ( !SetNamedPipeHandleState(hNamedPipe, &Mode, 0i64, 0i64) )
    return 0xFFFFFFFFi64;
  for ( NumberOfBytesWritten = 0; !NumberOfBytesWritten; WriteFile(hNamedPipe, a1, v2, &NumberOfBytesWritten, 0i64) )
    v2 = strlen((const char *)a1);
  for ( NumberOfBytesRead = 0; ; ReadFile(hNamedPipe, a1, 0x100u, &NumberOfBytesRead, 0i64) )
  {
    result = NumberOfBytesRead;
    if ( NumberOfBytesRead )
      break;
  }
  return result;
}
```

這兩個程式使用NamedPipe傳資料，傳的就是flag，而新的程式那邊又看到一個看起來是用來加密flag的地方

```
__int64 __fastcall sub_401550(__int64 a1)
{
  int v1; // r8d
  __int64 result; // rax
  int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; i <= 38; ++i )
  {
    *(_BYTE *)(a1 + i) ^= unk_403010[i % 0xBui64];
    v1 = *(unsigned __int8 *)(a1 + i);
    result = (unsigned int)(v1 + 2 * (i % 3));
    *(_BYTE *)(a1 + i) = v1 + 2 * (i % 3);
  }
  return result;
}
```

到這邊答案已經基本出來，只需要把資料還原回去即可。

summary

總結一下這個程式的行爲:
1. 檢查flag長度正不正確 2. process hollowing 3. 跟hollow出來的程式溝通 4. 檢查flag是否正確(從hollow的程

solve

```
data = [0x24, 0x1d, 0x1b, 0x31, 0x21, 0x0b, 0x4f, 0x0f, 0xe8, 0x50,
        0x37, 0x5b, 0x08, 0x40, 0x4a, 0x08, 0x1d, 0x11, 0x4a, 0xb8,
        0x11, 0x67, 0x3f, 0x67, 0x38, 0x14, 0x3f, 0x19, 0x0b, 0x54,
        0xb4, 0x09, 0x63, 0x12, 0x68, 0x2a, 0x45, 0x53, 0x0e, 0x01, 0x00]

key = [0x62, 0x57, 0x56, 0x76, 0x64, 0x77,
       0x3D, 0x3D, 0x87, 0x63, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0]

for i in range(39):
    data[i] = data[i] - 2 * (i % 3)
    if data[i] < 0:
        data[i] += 0xff
    data[i] ^= key[i % 0xb]
```

```
flag = [chr(i) for i in data]
print("".join(flag))
```

result

```
$ python3 sol.py
FLAG{pr0c355_h0ll0w1ng_4nd_n4m3d_p1p35}
$
```