题目名称	pipe_snake
题目类型	REVERSE
题目描述	win10+
下载链接	
解题思路	审计代码，看懂贪吃蛇代码，找到进程间 pipe 通信的数据流向。

首先通过对比父进程 exe 路径，判断当前进程是不是自己的子进程，进入两个分支。

```
17  pid = GetCurrentProcessId();
18  h = CreateToolhelp32Snapshot(2u, 0);
19  memset(&pe, 0, sizeof(pe));
20  pe.dwSize = 296;
21  if ( Process32First(h, &pe) )
22  {
23    do
24    {
25      if ( pe.th32ProcessID == pid )
26        ppid = pe.th32ParentProcessID;
27    }
28    while ( Process32Next(h, &pe) );
29  }
30  CloseHandle(h);
31  v0 = OpenProcess(0x410u, 0, ppid);
32  K32GetModuleFileNameExA(v0, 0, parentExe, 0x104u);
33  GetModuleFileNameA(0, currentExe, 0x104u);
34  v10 = parentExe;
35  v11 = currentExe;
36  while ( 1 )                                   // strcmp(currentExe, parentExe) == 0
37  {
38    v13 = *v11;
```

```
1  int __cdecl main(int argc, const char **argv
2  {
3    if ( isSubProcess() )
4      child_main();
5    else
6      parent_main();
7    return 0;
8  }
```

先看作为父进程（例如双击运行的情况）。

先新建了个线程，然后启动了 30 个自身 exe 的子进程，然后等待他们执行完毕。

```
21  hThread = CreateThread(0, 0, pipe_start, 0, 0, &threadId);
22  GetModuleFileNameA(0, szPath, 0x104u);
23  for ( i = 0; i < 30; ++i )
24  {
25    memset(&dst[68 * i], 0, 0x44u);
26    *&dst[68 * i] = 68;
27    v0 = &pi[i];
28    v0->hProcess = 0;
29    v0->hThread = 0;
30    v0->dwProcessId = 0;
31    v0->dwThreadId = 0;
32  }
33  for ( j = 0; j < 30; ++j )
34  {
35    CreateProcessA(szPath, 0, 0, 0, 0, 0, 0, 0, &dst[68 * j], &pi[j]);
36    hProcesses[j] = pi[j].hProcess;
37  }
38  WaitForMultipleObjects(0x1Eu, hProcesses, 1, 0xFFFFFFFF);
39  for ( k = 0; k < 30; ++k )
40  {
41    CloseHandle(pi[k].hProcess);
42    CloseHandle(pi[k].hThread);
43  }
```

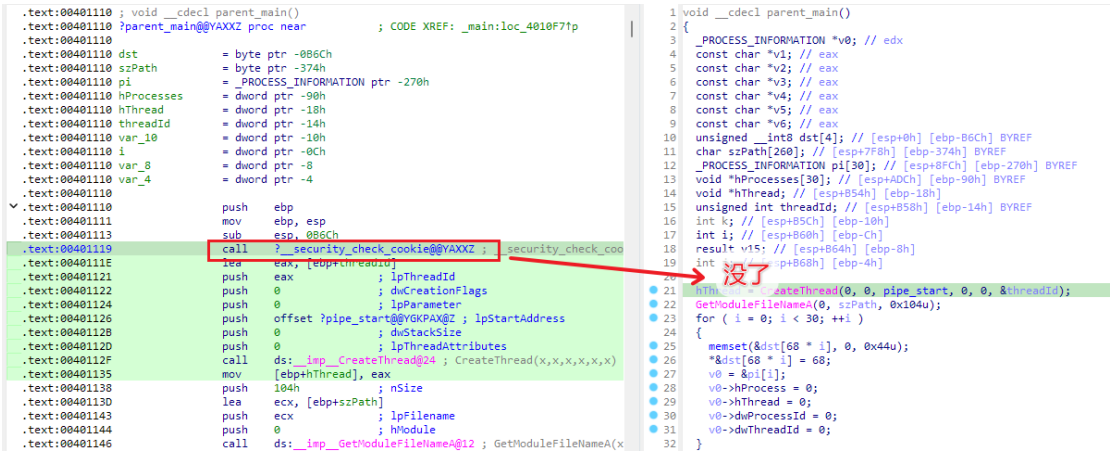
然后调用一个函数 4043D0（图里的 snake_start），将它的返回值与几个值对比，打印结果。

```
44  v15 = snake_start();
45  if ( v15 > STARVE )
46  {
47    if ( v15 == NOTNOW )
48    {
49      v3 = static_encrypt(STR_NOTNOW, 16);
50      printf(v3);
51      static_encrypt(STR_NOTNOW, 16);
52    }
53    else if ( v15 == TOOMUCH )
54    {
55      v4 = static_encrypt(STR_TOOMUCH, 15);
56      printf(v4);
57      static_encrypt(STR_TOOMUCH, 15);
58    }
```

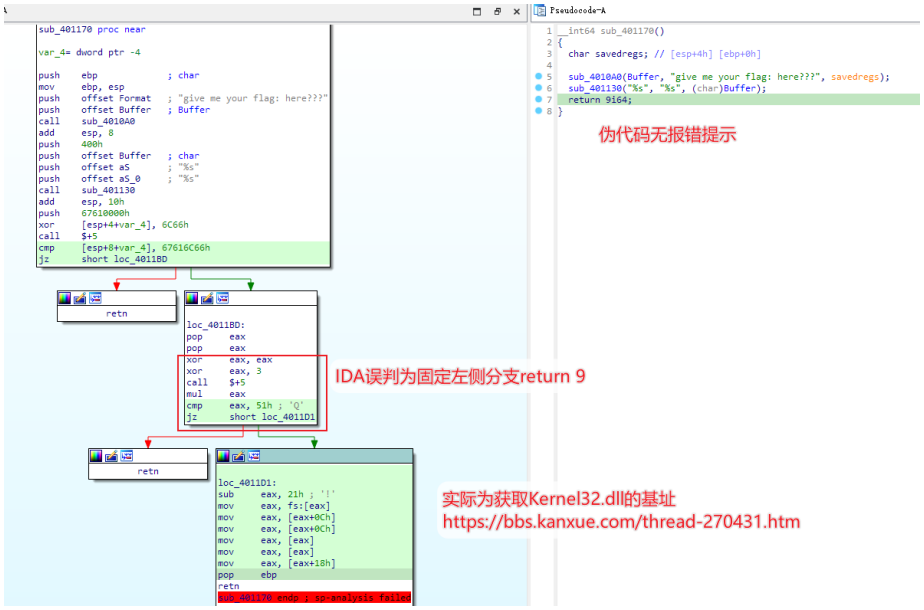
所有引用的静态字符串，都使用 401520 函数进行异或加密，最后一个字节为密钥。

```
1 char *__cdecl static_encrypt(char *str, int len)
2 {
3   unsigned int i; // [esp+0h] [ebp-4h]
4
5   for ( i = 0; i < len - 1; ++i )
6     str[i] ^= str[len - 2];
7   return str;
8 }
```

除此外，尝试通过调试等方法寻找 scanf 时，你可能会发现 IDA 伪代码有问题。由于 IDA 的一些特性，在 F5 伪代码中可能看不到函数开头**假的** GS 函数__security_check_cookie。



此前在 2022 年的 HITCTF 的题目 debug_maze 中也使用了针对 IDA 伪代码功能的隐藏机制，看雪上有选手 wp 带了题目，感兴趣可以了解一下。



这个隐藏函数就是输入函数，判断了长度 36 及格式 flag{}，将中间的 30 个 hex 转换成 15 个字节，覆盖掉输入的 flag。

```
13  v0 = static_encrypt(STR_Welcome, 28);
14  printf(v0);
15  static_encrypt(STR_Welcome, 28);
16  v1 = static_encrypt(STR_pS, 4);
17  scanf_s(v1, flag, 100);
18  if ( strlen(flag) != 36 || (v2 = static_encrypt(STR_FLAG_PREFIX, 7), strncmp(flag, v2, 5u), v3) || flag[35] != '}' )
19  {                                           // 长度36，开头结尾格式flag{}
20    v4 = static_encrypt(STR_YOULOST, 19);
21    printf(v4);
22    static_encrypt(STR_YOULOST, 19);
23    static_encrypt(STR_FLAG_PREFIX, 7);
24    exit(1);
25  }
26  for ( i = 0; i < 15; ++i )                  // {}中间的部分30个hex转成15个字节
27  {
28    if ( !ishex(flag[2 * i + 5]) || !ishex(flag[2 * i + 6]) )
29    {
30      v5 = static_encrypt(STR_YOULOST, 19);
31      printf(v5);
32      static_encrypt(STR_YOULOST, 19);
33      exit(1);
34    }
35    if ( flag[2 * i + 5] <= '9' )
36      v7 = flag[2 * i + 5] - '0';
37    else
38      v7 = flag[2 * i + 5] - 'W';
39    flag[i] = 16 * v7;
40    if ( flag[2 * i + 6] <= '9' )
41      v6 = flag[2 * i + 6] - '0';
42    else
43      v6 = flag[2 * i + 6] - 'W';
44    flag[i] |= v6;
45    flag[i] = flag[i];                        // 转换出来存到flag里，从下标0开始，会覆盖掉flag头
46  }
```

回头看开头创建的线程，其中有些 Pipe 操作，名称为\\.\pipe\anappleaday。搜索相关函数，可以看出来整个线程就是微软文档中的“重叠 IO Pipe 服务器”（https://learn.microsoft.com/zh-cn/windows/win32/ipc/named-pipe-server-using-overlapped-i-o）。

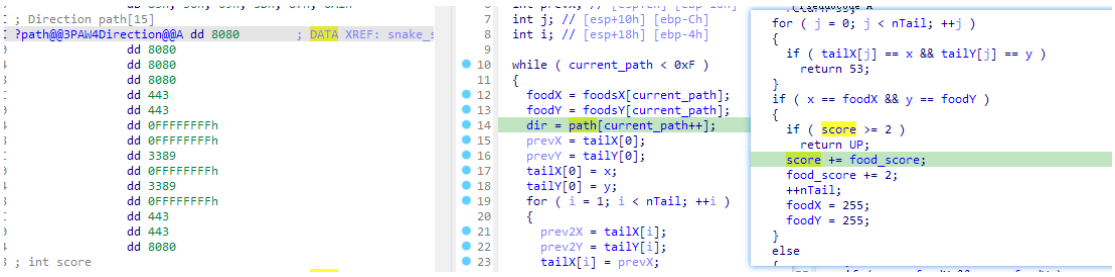
以此即可快速找到数据读写的函数 401980。数据收发都用 SM4 加密（有对应的 sbox 常量）。首字节接收到 0xff 会发送 flag+id（就是每个进程的递增编号），接收到非 15 会把接收数据第 2、3 个字节写到 数组[第 1 字节作下标] 里，这个就是贪吃蛇的食物投喂位置 X、Y 坐标。

```
 8  pipe = &Pipe[i];
 9  sm4_decrypt(pipe->chRequest, output, 16);
10  if ( output[0] == 255 )
11  {
12    *reply = *flag;
13    *&reply[4] = *&flag[4];
14    *&reply[8] = *&flag[8];
15    *&reply[12] = *&flag[12];
16    reply[14] = flag[14];
17    reply[15] = i;
18    sm4_encrypt(reply, output, 16);
19    memcpy_s(pipe->chReply, 0x1000u, output, 0x10u);
20  }
21  else
22  {
23    if ( output[0] != 15 )
24    {
25      foodsX[output[0]] = output[1];
26      foodsY[output[0]] = output[2];
27    }
28    chReply = pipe->chReply;
29    *pipe->chReply = -1;
30    *(chReply + 1) = -1;
31    *(chReply + 2) = -1;
32    *(chReply + 3) = -1;
33  }
34  pipe->cbToWrite = 16;
```

贪吃蛇那边就是普通的贪吃蛇代码，但是它自己前进，你的 flag 控制的是食物的位置。初始 5 分；空走一步-1；吃食物+3 每吃一次额外+2（即+3/+5/+7）；必须饿到正好 1 才能吃；8 分获胜；必须正好投喂到下一步的位置，不能放太远。

所以得分顺序只能是 5-4-3-2-1-(4)-3-2-1-(6)-5-4-3-2-1-(8)，在括号处投喂。只需要找到对应位置 X、Y 坐标，-1/255 表示不投喂，然后对应 pipe 通信中的两个字节即可。

下图左边是贪吃蛇路径，四个常数对应上下左右方向。右边是食物加分的地方。



接下来看一下子进程（即通过代码创建的 30 个进程）。

进去之后，开头第一个函数里面有一个特殊的“MZ”文件头判断，外面一堆内存操作，还 VirtualAlloc 申请了 0x40（PAGE_EXECUTE_READWRITE 可执行）权限的内存。

有恶意代码分析经验的选手，应该容易看出这个是在模拟操作系统对 PE 文件的加载，直接 IDA 导出一下 MZ 开头的数据就能获取到子进程的实际 PE 文件。

此处出题时本来想跨进程来注入（Process Hollowing），但是容易报毒，就在自身进程了。

```
?lpszPipename@@3PBDB dd offset aPipeAnappleada
                                        ; DATA XREF: pipe_start(void *)+F0↑r
                                        ; "\\\\.\\pipe\\anappleaday"
; unsigned __int8 payload[93184]
?payload@@3PAEA db 'MZ'                 ; DATA XREF: child_main(void)+5↑o
                db 90h,0,3,0,0,0,4,0,0,0,0FFh,0FFh,0,0,0B8h,0,0,0,0,0,0,0,'@',0,
                db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                db 0, 0Eh, 1Fh, 0BAh, 0Eh, 0, 0B4h, 9, 0CDh, 21h, 0B8h
                db 1
                db 4Ch, 0CDh
                db  21h ; !
aThisProgramCan db 'This program cannot be run in DOS mode.',0Dh,0Dh,0Ah
                db '$',0
                db    0
                db    0
```

这样提取后，对子进程分析，就是从一个数组中拿出自己进程 ID 对应的字节，与 flag 进行对比，如果找到就发给父进程作为对应步数 i 的食物坐标 id%5、id/5。

```
27  send_pipe_and_recv(hNamedPipe, (int)data, 16u, (int)recv_data);
28  v10 = recv_data[15];                          // 父进程发送的进程ID
29  for ( i = 0; i < 15u; ++i )                   // 对每一位flag进行判断
30  {
31    if ( recv_data[i] == byte_414D18[v10] )     // 如果有数组里的对应上了
32    {
33      data[0] = i;
34      if ( v10 == 25 )
35      {
36        *(_WORD *)&data[1] = -1;
37      }
38      else
39      {
40        data[1] = v10 % 5;                      // 发送坐标为，id%5，id/5
41        data[2] = v10 / 5;
42      }
43      send_pipe_and_recv(hNamedPipe, (int)data, 0x10u, 0);
44    }
45  }
```

因此获取贪吃蛇的路径和分数对应的坐标，转换成对应的子进程中的字节，就可以获得 flag{b3b3b3b340b3b3b3cfb3b3b3b3b3fd}